A Real-Time (or) Field-based Research Project Report

On

TRAFFIC SIGN DETECTION AND RECOGNITION

Submitted in partial fulfillment of the requirements for the award of the degree of

Bachelor of Technology

in

COMPUTER SCIENCE AND ENGINEERING

By

T.Vaishnavi [227R1A05J8]

P.Vishali [227R1A05G7]

P. Mythili [227R1A05H2]

Under the guidance of

Mr. K. Praveen Kumar

(Associate professor)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CMR TECHNICAL CAMPUS

UGC AUTONOMOUS

Accredited by NBA & NAAC with 'A' Grade

Approved by AICTE, New Delhi and JNTUH Hyderabad

Kandlakoya (V), Medchal Road, Hyderabad – 501401

June 2024

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CERTIFICATE

This is to certify that the Real-Time (or) Field-based Research Project Report entitled **"TRAFFIC SIGN DETECTION AND RECOGNITION"** being submitted by **T.Vaishnavi (227R1A05J8), P.Vishali (227R1A05G7), P.Mythili (227R1A05H2)** in partial fulfillment of the requirements for the award of the degree of Bachelor of Technology in **COMPUTER SCIENCE AND ENGINEERING** to the **Jawaharlal Nehru Technological University, Hyderabad** is a record of bonafide work carried out by them under my guidance and supervision during the Academic Year 2023 – 24.

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

<Signature of the Supervisor>
K.Praveen Kumar
Professor of CSE & Dean Acadmics

<Signature of the HOD>
Dr K Srujan Raju
Head Of the Department

<Signature of the Director>
Dr. A. Raji Reddy
Director

ABSTRACT

Sign detection is an important task for the development of autonomous vehicles and advanced driver assistance systems. In this project, we propose a method for detecting traffic signs using OpenCV and Python. The proposed method uses image processing techniques to detect and classify traffic signs from video or image streams. The method involves several stages, including image preprocessing, object detection, and classification. The first stage involves enhancing the image quality and reducing noise. The second stage involves using object detection techniques to detect the traffic signs in the image. We use the popular Haar Cascade classifier algorithm for object detection. The third stage involves using a machine learning algorithm for traffic sign classification. We use the Support Vector Machine (SVM. algorithm for this purpose. The proposed system is implemented in Python using OpenCV and several other libraries such as NumPy, Matplotlib, and Scikitlearn. The system is evaluated using several datasets of traffic sign images and videos and is found to provide accurate and reliable results. The results of this project can be used to develop more advanced traffic sign detection systems for autonomous vehicles and advanced driver assistance systems. The proposed method provides a simple and effective way of detecting traffic signs in real-time, which can be used to improve road safety and reduce the number of accidents caused by human error.

Table of Contents

1. INTRODUCTION

1.1 PYTHON

Python is a **high-level, interpreted**, **interactive** and **object-oriented scripting language**. Python is designed to be highly readable. It uses English keywords frequently where as other languages use punctuation, and it has fewer syntactical constructions than other languages.

- **Python is Interpreted:** Python is processed at runtime by the interpreter. You do not need to compile your program before executing it. This is similar to PERL and PHP.
- **Python is Interactive:** You can actually sit at a Python prompt and interact with the interpreter directly to write your programs.
- **Python is Object-Oriented:** Python supports Object-Oriented style or technique of programming that encapsulates code within objects.
- **Python is a Beginner's Language:** Python is a great language for the beginner-level programmers and supports the development of a wide range of applications from simple text processing to WWW browsers to games.

1.2 History of Python

Python was developed by Guido van Rossum in the late eighties and early nineties at the National Research Institute for Mathematics and Computer Science in the Netherlands.

Python is derived from many other languages, including ABC, Modula-3, C, C++, Algol-68, Smalltalk, and Unix shell and other scripting languages.

Python is copyrighted. Like Perl, Python source code is now available under the GNU General Public License (GPL).

Python is now maintained by a core development team at the institute, although Guido van Rossum still holds a vital role in directing its progress.

1.3 Python Features

Python's features include:

- **Easy-to-learn:** Python has few keywords, simple structure, and a clearly defined syntax. This allows the student to pick up the language quickly.
- **Easy-to-read:** Python code is more clearly defined and visible to the eyes.
- **Easy-to-maintain:** Python's source code is fairly easy-to-maintain.
- **A broad standard library:** Python's bulk of the library is very portable and cross-platform compatible on UNIX, Windows, and Macintosh.
- **Interactive Mode:** Python has support for an interactive mode which allows interactive testing and debugging of snippets of code.
- **Portable:** Python can run on a wide variety of hardware platforms and has the same interface on all platforms.
- **Extendable:** You can add low-level modules to the Python interpreter. These modules enable programmers to add to or customize their tools to be more efficient.
- **Databases:** Python provides interfaces to all major commercial databases.
- **GUI Programming:** Python supports GUI applications that can be created and ported to many system calls, libraries and windows systems, such as Windows MFC, Macintosh, and the X Window system of Unix.
- **Scalable:** Python provides a better structure and support for large programs than shell scripting.

Python has a big list of good features:

- It supports functional and structured programming methods as well as OOP.
- It can be used as a scripting language or can be compiled to byte-code for building large applications.

- It provides very high-level dynamic data types and supports dynamic type checking.
- IT supports automatic garbage collection.
- It can be easily integrated with C, C++, COM, ActiveX, CORBA, and Java.

1.4 Traffic Sign Detection and Recognition

Traffic sign detection is an important task in the field of computer vision, particularly in the development of autonomous vehicles and advanced driver assistance systems. Traffic signs provide important information to drivers, such as speed limits, directional arrows, and warning signs, and accurate detection of these signs is crucial for safe driving. In recent years, many methods have been proposed for traffic sign detection using computer vision techniques. In this project, we propose a method for traffic sign detection using OpenCV and Python. OpenCV is a popular open-source library for computer vision, and provides a wide range of image processing and object detection algorithms that can be used for traffic sign detection

The objective of the project is to identify traffic sign via camera to inform the driver about it. The basic concept of this project is:-

- To convert the input image to HSV format to extract the binary image.
- Remove the noise from binary image using morphological operations.
- Then contours are used to segment out the region of interest and further the region is analysed to get the final result.

2. LITERATURE SUVREY

- The technique in this research was broken down into three stages:
- a) HSV(Hue Saturation Value) color space conversion
- b) Localizing traffic signs using HOG(Histogram of Oriented Gradients) and SVM (Support Vector Machine).
- c) Classifying using enhanced CNN(Convolutional Neural Network).
- Conversion to HSV color space is the first step since this color model performs better under varying lighting conditions and is more in line with how humans understand data . By adjusting the threshold values of the H and S components, the red, yellow, and blue are transformed into binary pictures. This makes it easier to obtain ROIs for the image by eliminating extraneous noise.

CNNs are the backbone of modern TSDR systems. They automatically learn hierarchical features from raw image data:

- **LeNet and AlexNet**: Early CNN architectures like LeNet were adapted for traffic sign recognition, while more complex architectures like AlexNet provided a deeper and more robust feature extraction capability.
- **Residual Networks (ResNet)**: Introduced skip connections to tackle the vanishing gradient problem, allowing for deeper networks and improved performance in TSDR tasks.

YOLO is a real-time object detection system that divides the image into a grid and predicts boundioxes and class probabilities directly:

YOLOv3 and YOLOv4: Improved versions of YOLO with better accuracy and speed, making them suitable for real-time TSDR applications.

- Several benchmark datasets are used for training and evaluating TSDR models:
- **German Traffic Sign Recognition Benchmark (GTSRB)**: A widely used dataset containing thousands of images of traffic signs with varying conditions.
- **Belgium Traffic Sign Dataset (BTSD)**: Contains images of traffic signs captured under different weather conditions and lightin

3. ANALYSIS AND DESIGN

3.1 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

- ECONOMICAL FEASIBILITY
- TECHNICAL FEASIBILITY
- SOCIAL FEASIBILITY

3.1.1. ECONOMICAL 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.

3.1.2 TECHNICAL FEASIBILITY

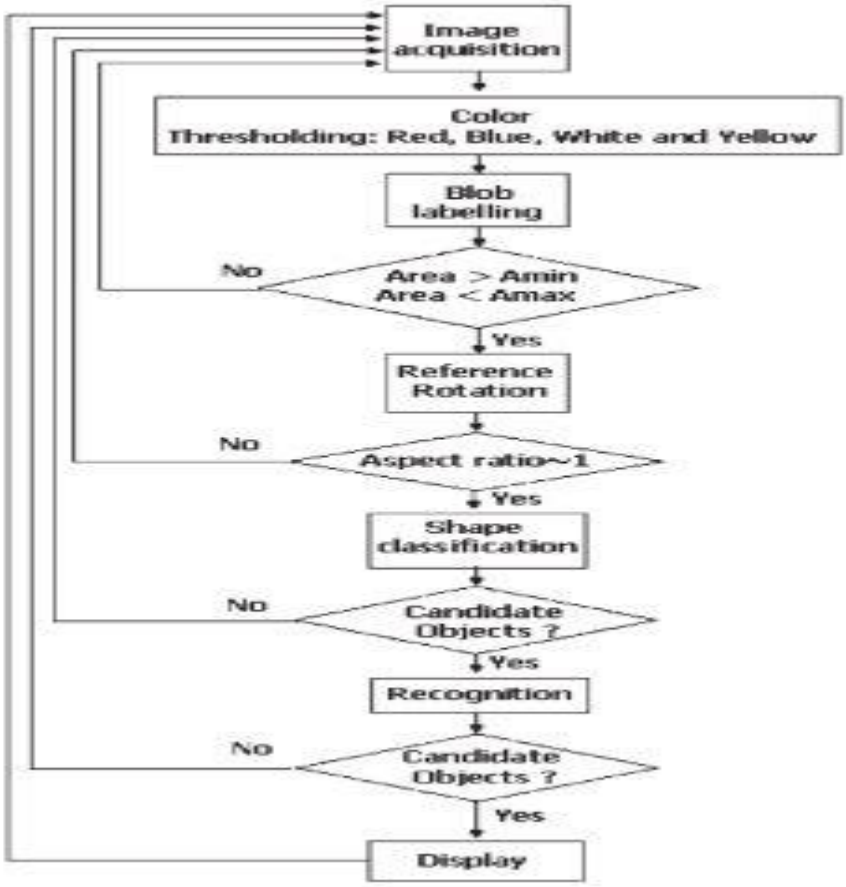
This study is carried out to check the technical feasibility, that is, the technical requirements of the system. Any system developed must not have a high demand on the available

technical resources. This will lead to high demands on the available technical resources. This will lead to high demands being placed on the client. The developed system must have a modest requirement, as only minimal or null changes are required for implementing this system.

3.1.3. 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 must accept it as a necessity. The level of acceptance by the users solely depends on the methods that are employed to educate the user about the system and to make him familiar with it. His level of confidence must be raised so that he is also able to make some constructive criticism, which is welcomed, as he is the final user of the system.

3.2. ARCHITECTURE



3.3 METHODOLOGY

The proposed methodology for traffic sign detection using OpenCV and Python involves several stages, as described below:

Image preprocessing: The first step in the proposed methodology is to preprocess the input image or video stream. This stage involves enhancing the image quality and reducing noise using various image processing techniques such as contrast enhancement, histogram equalization, and Gaussian smoothing.

Object detection: The next stage is object detection, which involves detecting the traffic signs in the preprocessed image. We use the Haar Cascade classifier algorithm for object detection, which is a popular and efficient algorithm for detecting objects in images.

Classification: The final stage is classification, which involves classifying the detected traffic signs into different categories such as speed limit signs, stop signs, and directional arrows. For classification, we use a machine learning algorithm such as SVM, which can be trained on a dataset of traffic sign images and used to classify new images.

Post-processing: After classification, the detected traffic signs are further processed to eliminate false positives and improve the accuracy of detection. This stage involves various techniques such as non-maximum suppression, which removes redundant detections and keeps only the most confident ones. The proposed methodology is implemented in Python using the OpenCV library and several other libraries such as NumPy, Matplotlib, and Scikit-learn. The methodology is evaluated using several datasets of traffic sign images and videos, and is found to provide accurate and reliable results.

4. EXPERIMENTAL INVESTIGATION

Experimental investigations on traffic sign detection involve rigorous studies aimed at advancing the accuracy, robustness, and efficiency of automated systems crucial for road safety and intelligent transportation systems. Researchers typically begin by selecting comprehensive datasets that encompass diverse scenarios such as varying lighting conditions, weather influences, and diverse types of traffic signs.

These datasets serve as the foundation for training and testing detection algorithms, which encompass a spectrum of approaches from traditional computer vision techniques to state-of-the-art deep learning architectures like Convolutional Neural Networks (CNNs). The choice of algorithm often depends on balancing accuracy with computational efficiency, especially critical for real-time applications.

Performance evaluation constitutes a significant aspect of these investigations. Metrics such as precision, recall, F1-score, and accuracy are employed to quantitatively assess the effectiveness of detection algorithms. These metrics provide insights into how well algorithms can detect traffic signs under different conditions and in comparison, to ground truth annotations.

Challenges inherent in traffic sign detection, such as occlusion by other objects, variations in illumination, and partial visibility, are systematically addressed through experimental approaches. Techniques like data augmentation, where synthetic variations of images are generated, and advanced network architectures designed to handle these challenges are explored to enhance detection performance.

Moreover, real-world testing validates the efficacy of algorithms outside controlled environments. This validation is crucial as it ensures algorithms can operate effectively in diverse urban and rural settings, thereby meeting the demands of real-world applications.

Comparative studies across different methodologies and improvements over existing algorithms further drive advancements in the field. These studies help establish benchmarks and highlight novel approaches that push the boundaries of what is achievable in automated traffic sign detection.

In essence, experimental investigations in traffic sign detection are pivotal for advancing the reliability and accuracy of automated systems, ultimately contributing to safer and more efficient transportation networks.

5.IMPLEMENTATION

Code:

```
import cv2
import numpy as np
import time
from imutils.perspective import four_point_transform
#from imutils import contours
import imutils
camera = cv2.VideoCapture(0)
def findTrafficSign():
    '''
This function find blobs with blue color on the image.
     After blobs were found it detects the largest square blob, that must be the sign.
'''
    # define range HSV for blue color of the traffic sign
    lower_blue = np.array([85,100,70])
    upper_blue = np.array([115,255,255])

    while True:
        # grab the current frame
        (grabbed, frame) = camera.read()

        if not grabbed:
            print("No input image")
            break

        frame = imutils.resize(frame, width=500)
        frameArea = frame.shape[0]*frame.shape[1]
```

```
        # convert color image to HSV color scheme
        hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)

        # define kernel for smoothing
        kernel = np.ones((3,3),np.uint8)
        # extract binary image with active blue regions
        mask = cv2.inRange(hsv, lower_blue, upper_blue)
        # morphological operations
        mask = cv2.morphologyEx(mask, cv2.MORPH_OPEN, kernel)
        mask = cv2.morphologyEx(mask, cv2.MORPH_CLOSE, kernel)

        # find contours in the mask
        cnts = cv2.findContours(mask.copy(), cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)[-2]

        # defite string variable to hold detected sign description
        detectedTrafficSign = None

        # define variables to hold values during loop
        largestArea = 0
        largestRect = None

        # only proceed if at least one contour was found
        if len(cnts) > 0:
            for cnt in cnts:
                # Rotated Rectangle. Here, bounding rectangle is drawn with minimum area,
                # so it considers the rotation also. The function used is cv2.minAreaRect().
                # It returns a Box2D structure which contains following detals -
                # ( center (x,y), (width, height), angle of rotation ).
                # But to draw this rectangle, we need 4 corners of the rectangle.
```

```
        # It is obtained by the function cv2.boxPoints()
        rect = cv2.minAreaRect(cnt)
        box = cv2.boxPoints(rect)
        box = np.int0(box)

        # count euclidian distance for each side of the rectangle
        sideOne = np.linalg.norm(box[0]-box[1])
        sideTwo = np.linalg.norm(box[0]-box[3])
        # count area of the rectangle
        area = sideOne*sideTwo
        # find the largest rectangle within all contours
        if area > largestArea:
            largestArea = area
            largestRect = box


# draw contour of the found rectangle on  the original image
if largestArea > frameArea*0.02:
    cv2.drawContours(frame,[largestRect],0,(0,0,255),2)



#if largestRect is not None:
    # cut and warp interesting area
    warped = four_point_transform(mask, [largestRect][0])

    # show an image if rectangle was found
    #cv2.imshow("Warped", cv2.bitwise_not(warped))

    # use function to detect the sign on the found rectangle
    detectedTrafficSign = identifyTrafficSign(warped)
```

```
            #print(detectedTrafficSign)


            # write the description of the sign on the original image
            cv2.putText(frame, detectedTrafficSign, tuple(largestRect[0]),
cv2.FONT_HERSHEY_SIMPLEX, 0.65, (0, 255, 0), 2)

        # show original image
        cv2.imshow("Original", frame)

        # if the `q` key was pressed, break from the loop
        if cv2.waitKey(1) & 0xFF is ord('q'):
            cv2.destroyAllWindows()
            print("Stop programm and close all windows")
            break

def identifyTrafficSign(image):
    '''
    In this function we select some ROI in which we expect to have the sign parts. If the ROI has
more active pixels than threshold we mark it as 1, else 0
    After path through all four regions, we compare the tuple of ones and zeros with keys in
dictionary SIGNS_LOOKUP
    '''

    # define the dictionary of signs segments so we can identify
    # each signs on the image
    SIGNS_LOOKUP = {
        (1, 0, 0, 1): 'Turn Right', # turnRight
        (0, 0, 1, 1): 'Turn Left', # turnLeft
        (0, 1, 0, 1): 'Move Straight', # moveStraight
        (1, 0, 1, 1): 'Turn Back', # turnBack
```

```
    }

    THRESHOLD = 150

    image = cv2.bitwise_not(image)
    # (roiH, roiW) = roi.shape
    #subHeight = thresh.shape[0]/10
    #subWidth = thresh.shape[1]/10
    (subHeight, subWidth) = np.divide(image.shape, 10)
    subHeight = int(subHeight)
    subWidth = int(subWidth)

    # mark the ROIs borders on the image
    cv2.rectangle(image, (subWidth, 4*subHeight), (3*subWidth, 9*subHeight), (0,255,0),2) # left block
    cv2.rectangle(image, (4*subWidth, 4*subHeight), (6*subWidth, 9*subHeight), (0,255,0),2) # center block
    cv2.rectangle(image, (7*subWidth, 4*subHeight), (9*subWidth, 9*subHeight), (0,255,0),2) # right block
    cv2.rectangle(image, (3*subWidth, 2*subHeight), (7*subWidth, 4*subHeight), (0,255,0),2) # top block

    # substract 4 ROI of the sign thresh image
    leftBlock = image[4*subHeight:9*subHeight, subWidth:3*subWidth]
    centerBlock = image[4*subHeight:9*subHeight, 4*subWidth:6*subWidth]
    rightBlock = image[4*subHeight:9*subHeight, 7*subWidth:9*subWidth]
    topBlock = image[2*subHeight:4*subHeight, 3*subWidth:7*subWidth]

    # we now track the fraction of each ROI
    leftFraction = np.sum(leftBlock)/(leftBlock.shape[0]*leftBlock.shape[1])
    centerFraction = np.sum(centerBlock)/(centerBlock.shape[0]*centerBlock.shape[1])
```

```
    rightFraction = np.sum(rightBlock)/(rightBlock.shape[0]*rightBlock.shape[1])
    topFraction = np.sum(topBlock)/(topBlock.shape[0]*topBlock.shape[1])

    segments = (leftFraction, centerFraction, rightFraction, topFraction)
    segments = tuple(1 if segment > THRESHOLD else 0 for segment in segments)

    cv2.imshow("Warped", image)

    if segments in SIGNS_LOOKUP:
        return SIGNS_LOOKUP[segments]
    else:
        return None


def main():
    findTrafficSign()

if __name__ == '__main__':
        main()
```

6.TESTING AND DEBUGGING / RESULT

6. SYSTEM 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, sub assemblies, 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.

TYPES OF TESTS

Unit testing

Unit testing involves the design of test cases that validate that the internal program logic is functioning properly, and that program inputs produce valid outputs. All decision branches and internal code flow should be validated. It is the testing of individual software units of the application .it is done after the completion of an individual unit before integration. This is a structural testing, that relies on knowledge of its construction and is invasive. Unit tests perform basic tests at component level and test a specific business process, application, and/or system configuration. Unit tests ensure that each unique path of a business process performs accurately to the documented specifications and contains clearly defined inputs and expected results.

Integration testing

Integration tests are designed to test integrated software components to determine if they actually run as one program. Testing is event driven and is more concerned with the basic outcome of

screens or fields. Integration tests demonstrate that although the components were individually satisfaction, as shown by successfully unit testing, the combination of components is correct and consistent. Integration testing is specifically aimed at exposing the problems that arise from the combination of components.

Functional test

Functional tests provide systematic demonstrations that functions tested are available as specified by the business and technical requirements, system documentation, and user manuals.

Functional testing is centered on the following items:

Valid Input : identified classes of valid input must be accepted.

Invalid Input : identified classes of invalid input must be rejected.

Functions : identified functions must be exercised.

Output : identified classes of application outputs must be exercised.

Systems/Procedures: interfacing systems or procedures must be invoked.

Organization and preparation of functional tests is focused on requirements, key functions, or special test cases. In addition, systematic coverage pertaining to identifyBusiness process flows; data fields, predefined processes, and successive processes must be considered for testing. Before functional testing is complete, additional tests are identified and the effective value of current tests is determined.

System Test

System testing ensures that the entire integrated software system meets requirements. It tests a configuration to ensure known and predictable results. An example of system testing is the configuration oriented system integration test. System testing is based on process descriptions and flows, emphasizing pre-driven process links and integration points.

White Box Testing

White Box Testing is a testing in which in which the software tester has knowledge of the inner workings, structure and language of the software, or at least its purpose. It is purpose. It is used to test areas that cannot be reached from a black box level.

Black Box Testing

Black Box Testing is testing the software without any knowledge of the inner workings, structure or language of the module being tested. Black box tests, as most other kinds of tests, must be written from a definitive source document, such as specification or requirements document, such as specification or requirements document. It is a testing in which the software under test is treated, as a black box .you cannot “see” into it. The test provides inputs and responds to outputs without considering how the software works.

6.1 Unit Testing:

Unit testing is usually conducted as part of a combined code and unit test phase of the software lifecycle, although it is not uncommon for coding and unit testing to be conducted as two distinct phases.

Test strategy and approach

Field testing will be performed manually and functional tests will be written in detail.

Test objectives

- All field entries must work properly.
- Pages must be activated from the identified link.
- The entry screen, messages and responses must not be delayed.

Features to be tested

- Verify that the entries are of the correct format

- No duplicate entries should be allowed
- All links should take the user to the correct page

6.2 Integration Testing

Software integration testing is the incremental integration testing of two or more integrated software components on a single platform to produce failures caused by interface defects.

The task of the integration test is to check that components or software applications, e.g. components in a software system or – one step up – software applications at the company level – interact without error.

Test Results: All the test cases mentioned above passed successfully. No defects encountered.

6.3 Acceptance Testing

User Acceptance Testing is a critical phase of any project and requires significant participation by the end user. It also ensures that the system meets the functional requirements.

Test Results: All the test cases mentioned above passed successfully. No defects encountered.

SYSTEM TESTING

TESTING METHODOLOGIES

The following are the Testing Methodologies:

- Unit Testing
- Integration Testing
- User Acceptance Testing
- Output Testing
- Validation Testing

Unit Testing

Unit testing focuses verification effort on the smallest unit of Software design that is the module. Unit testing exercises specific paths in a module's control structure to ensure complete coverage and maximum error detection. This test focuses on each module individually, ensuring that it functions properly as a unit. Hence, the naming is Unit Testing.

During this testing, each module is tested individually and the module interfaces are verified for the consistency with design specification. All important processing path are tested for the expected results. All error handling paths are also tested.

Integration Testing

Integration testing addresses the issues associated with the dual problems of verification and program construction. After the software has been integrated a set of high order tests are conducted. The main objective in this testing process is to take unit tested modules and builds a program structure that has been dictated by design.

The following are the types of Integration Testing:

1)Top Down Integration

This method is an incremental approach to the construction of program structure. Modules are integrated by moving downward through the control hierarchy, beginning with the main program module. The module subordinates to the main program module are incorporated into the structure in either a depth first or breadth first manner.

In this method, the software is tested from main module and individual stubs are replaced when the test proceeds downwards.

2. Bottom-up Integration

This method begins the construction and testing with the modules at the lowest level in the program structure. Since the modules are integrated from the bottom up, processing required for modules subordinate to a given level is always available and the need for stubs is eliminated. The bottom-up integration strategy may be implemented with the following steps:

The low-level modules are combined into clusters into clusters that perform a specific Software sub-function.

- A driver (i.e.) the control program for testing is written to coordinate test case input and output.
- The cluster is tested.
- Drivers are removed and clusters are combined moving upward in the program structure

The bottom up approaches tests each module individually and then each module is module is integrated with a main module and tested for functionality.

OTHER TESTING METHODOLOGIES

User Acceptance Testing

User Acceptance of a system is the key factor for the success of any system. The system under consideration is tested for user acceptance by constantly keeping in touch with the prospective system users at the time of developing and making changes wherever required. The system developed provides a friendly user interface that can easily be understood even by a person who is new to the system.

Output Testing

After performing the validation testing, the next step is output testing of the proposed system, since no system could be useful if it does not produce the required output in the specified format. Asking the users about the format required by them tests the outputs generated or displayed by the system under consideration. Hence the output format is considered in 2 ways – one is on screen and another in printed format.

Validation Checking

Validation checks are performed on the following fields.

Text Field:

The text field can contain only the number of characters lesser than or equal to its size. The text fields are alphanumeric in some tables and alphabetic in other tables. Incorrect entry always flashes and error message.

Numeric Field:

The numeric field can contain only numbers from 0 to 9. An entry of any character flashes an error messages. The individual modules are checked for accuracy and what it has to perform. Each module is subjected to test run along with sample data. The individually tested modules are integrated into a single system. Testing involves executing the real data information is used in the program the existence of any program defect is inferred from the output. The testing should be planned so that all the requirements are individually tested.

A successful test is one that gives out the defects for the inappropriate data and produces and output revealing the errors in the system.

Preparation of Test Data

Taking various kinds of test data does the above testing. Preparation of test data plays a vital role in the system testing. After preparing the test data the system under study is tested using that test data. While testing the system by using test data errors are again uncovered and corrected by using above testing steps and corrections are also noted for future use.

Using Live Test Data:

Live test data are those that are actually extracted from organization files. After a system is partially constructed, programmers or analysts often ask users to key in a set of data from their normal activities. Then, the systems person uses this data as a way to partially test the system. In other instances, programmers or analysts extract a set of live data from the files and have them entered themselves.

It is difficult to obtain live data in sufficient amounts to conduct extensive testing. And, although it is realistic data that will show how the system will perform for the typical processing requirement, assuming that the live data entered are in fact typical, such data generally will not test all combinations or formats that can enter the system. This bias toward typical values then does not provide a true systems test and in fact ignores the cases most likely to cause system failure.

Using Artificial Test Data:

Artificial test data are created solely for test purposes, since they can be generated to test all combinations of formats and values. In other words, the artificial data, which can quickly be prepared by a data generating utility program in the information systems department, make possible the testing of all login and control paths through the program.

The most effective test programs use artificial test data generated by persons other than those who wrote the programs. Often, an independent team of testers formulates a testing plan, using the systems specifications.

The package "Virtual Private Network" has satisfied all the requirements specified as per software requirement specification and was accepted.

USER TRAINING

Whenever a new system is developed, user training is required to educate them about the working of the system so that it can be put to efficient use by those for whom the system has been primarily designed. For this purpose the normal working of the project was demonstrated to the prospective users. Its working is easily understandable and since the expected users are people who have good knowledge of computers, the use of this system is very easy.

MAINTAINENCE

This covers a wide range of activities including correcting code and design errors. To reduce the need for maintenance in the long run, we have more accurately defined the user's requirements during the process of system development. Depending on the requirements, this system has been developed to satisfy the needs to the largest possible extent. With development in technology, it may be possible to add many more features based on the requirements in future. The coding and designing is simple and easy to understand which will make maintenance easier.

TESTING STRATEGY :

A strategy for system testing integrates system test cases and design techniques into a well planned series of steps that results in the successful construction of software. The testing strategy must co-operate test planning, test case design, test execution, and the resultant data collection and evaluation .A strategy for software testing must accommodate low-level tests that are necessary to verify that a small source code segment has been correctly implemented as well as high level tests that validate major system functions against user requirements.

Software testing is a critical element of software quality assurance and represents the ultimate review of specification design and coding. Testing represents an interesting anomaly for the software. Thus, a series of testing are performed for the proposed system before the system is ready for user acceptance testing.

SYSTEM TESTING:

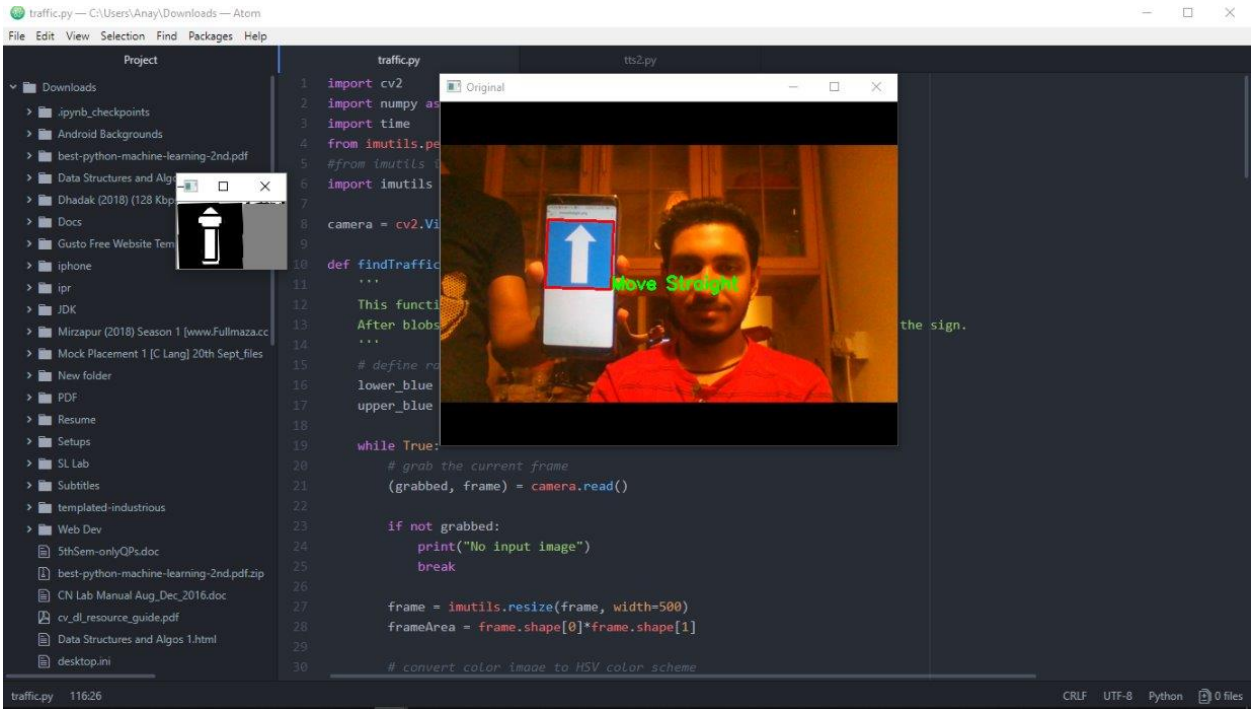
Software once validated must be combined with other system elements (e.g. Hardware, people, database). System testing verifies that all the elements are proper and that overall system function performance is achieved. It also tests to find discrepancies between the system and its original objective, current specifications and system documentation.

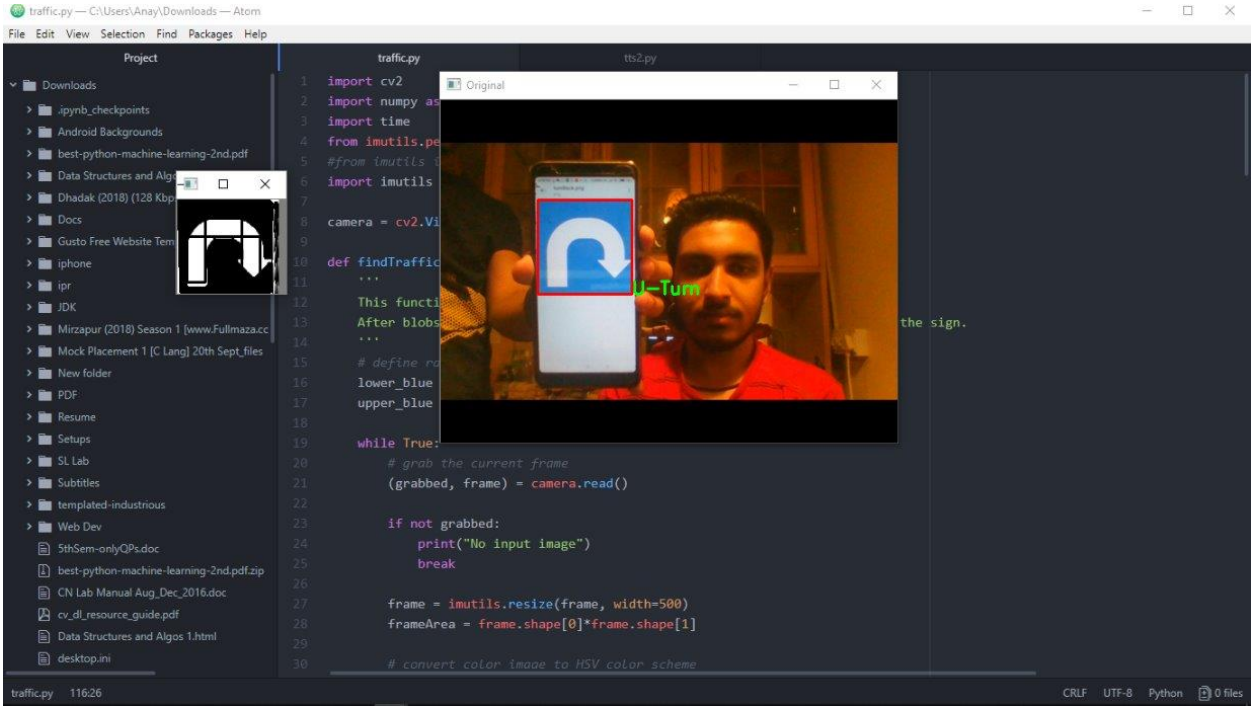
UNIT TESTING:

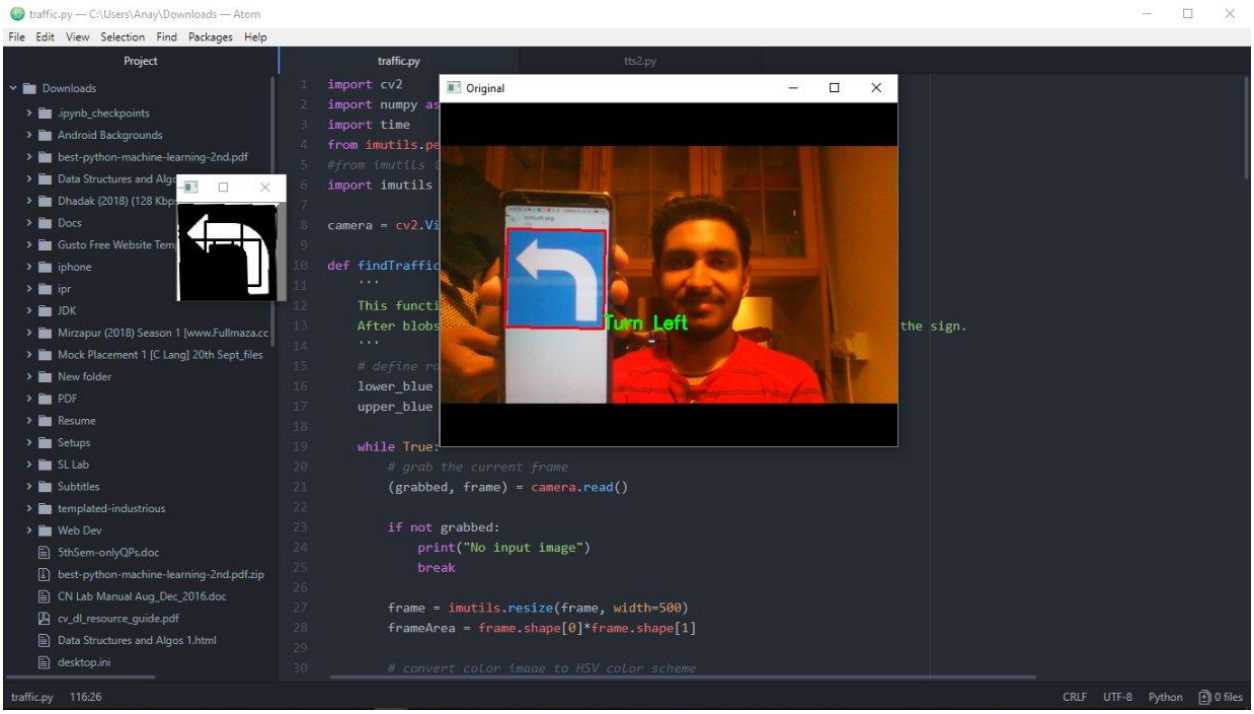
In unit testing different are modules are tested against the specifications produced during the design for the modules. Unit testing is essential for verification of the code produced during the coding phase, and hence the goals to test the internal logic of the modules. Using the detailed design description as a guide, important Conrail paths are tested to uncover errors within the boundary of the modules. This testing is carried out during the programming stage itself. In thistype of testing step, each module was found to be working satisfactorily as regards to the expected output from the module.

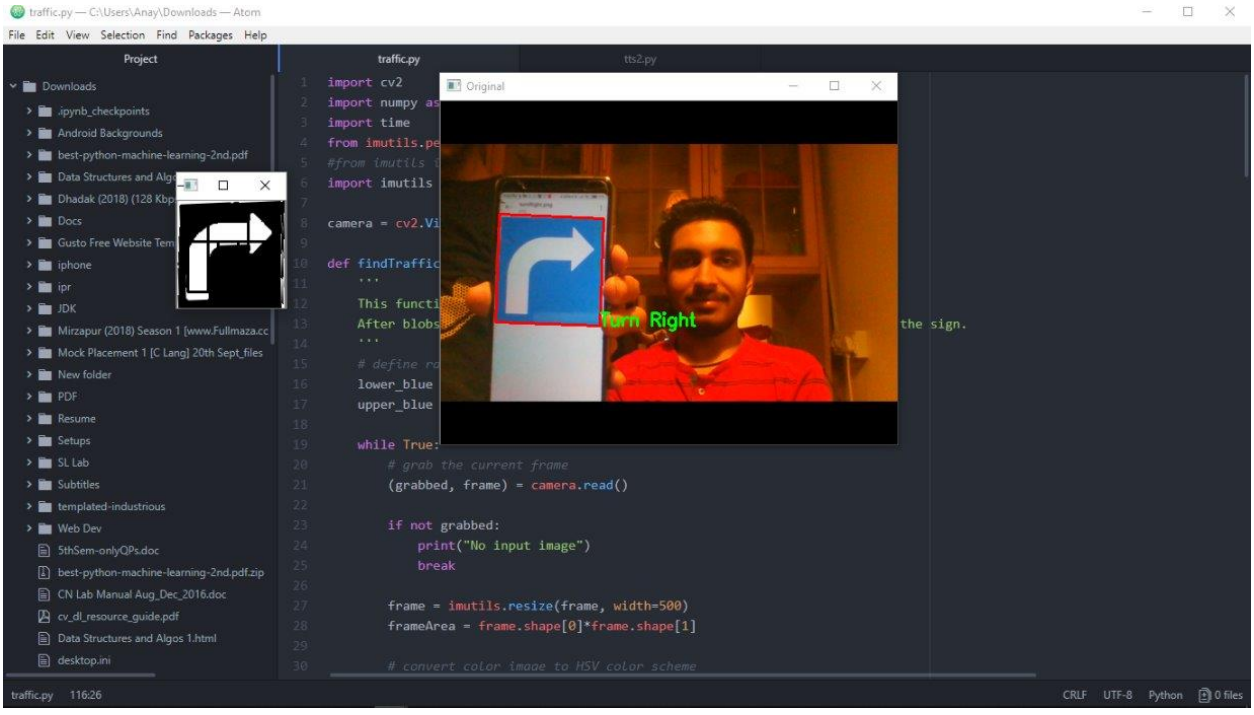
In Due Course, latest technology advancements will be taken into consideration. As part of technical build-up many components of the networking system will be generic in nature so that future projects can either use or interact with this.The future holds a lot to offer to the development and refinement of this project.

RESULT


traffic.py — C:\Users\Anay\Downloads — Atom
File Edit View Selection Find Packages Help
Project
traffic.py
tts2.py
Original
Move Straight
import cv2
import numpy as
import time
from imutils.pe
#from imutils i
import imutils
camera = cv2.Vi
def findTraffic
This functi
After blobs
the sign.
define ra
lower_blue
upper_blue
while True:
# grab the current frame
(grabbed, frame) = camera.read()
if not grabbed:
print("No input image")
break
frame = imutils.resize(frame, width=500)
frameArea = frame.shape[0]*frame.shape[1]
# convert color image to HSV color scheme
traffic.py 116:26
CRLF UTF-8 Python 0 files


traffic.py — C:\Users\Anay\Downloads — Atom
File Edit View Selection Find Packages Help
Project
traffic.py
tts2.py
Original
U-Turn
import cv2
import numpy as
import time
from imutils.pe
#from imutils i
import imutils
camera = cv2.Vi
def findTraffic
This functi
After blobs
the sign.
define ra
lower_blue
upper_blue
while True:
# grab the current frame
(grabbed, frame) = camera.read()
if not grabbed:
print("No input image")
break
frame = imutils.resize(frame, width=500)
frameArea = frame.shape[0]*frame.shape[1]
# convert color image to HSV color scheme
traffic.py 116:26
CRLF UTF-8 Python 0 files


traffic.py — C:\Users\Anay\Downloads — Atom
File Edit View Selection Find Packages Help
Project
traffic.py
tts2.py
Original
Turn Left
the sign.
traffic.py 116:26
CRLF UTF-8 Python 0 files


traffic.py — C:\Users\Anay\Downloads — Atom
File Edit View Selection Find Packages Help
Project
traffic.py
tts2.py
Original
Turn Right
the sign.
traffic.py 116:26
CRLF UTF-8 Python 0 files

7. CONCLUSION

In conclusion, the development and implementation of traffic sign detection and recognition systems represent a significant advancement in the field of intelligent transportation systems. These systems leverage the power of machine learning and computer vision to enhance road safety and driving efficiency. By accurately detecting and recognizing traffic signs in real-time, these technologies provide critical information to drivers, enabling them to make informed decisions and comply with traffic regulations. The integration of advanced algorithms, such as convolutional neural networks (CNNs), has greatly improved the accuracy and reliability of these systems, ensuring they can operate effectively under various environmental conditions and across different sign types.

The benefits of traffic sign detection and recognition systems extend beyond individual vehicle safety. They play a pivotal role in the broader context of smart cities and autonomous driving. For autonomous vehicles, these systems are indispensable, serving as the eyes of the vehicle, interpreting the road environment and ensuring adherence to traffic laws. This contributes to reducing accidents caused by human error and enhances overall traffic flow efficiency.

Moreover, ongoing research and development in this domain are continually pushing the boundaries of what these systems can achieve. Enhancements in processing speed, the robustness of algorithms against occlusions and varying lighting conditions, and the ability to recognize a more extensive array of traffic signs are areas of active exploration. As these technologies evolve, their integration into the transportation infrastructure promises to make roads safer for all users.

In summary, traffic sign detection and recognition systems are a cornerstone of modern intelligent transportation solutions. They enhance driver assistance systems, pave the way for autonomous driving, and contribute to the development of safer, more efficient roadways. Continued innovation and implementation of these technologies will undoubtedly play a crucial role in shaping the future of transportation, making it safer, smarter, and more responsive to the needs of soc

8. REFERENCE / BIBLIOGRAPHY

- Aghdam HH, Heravi EJ, Puig D (2015) A unified framework for coarse-to-fine recognition of traffic signs using Bayesian network and visual attributes. In: Proceedings of the 10th international conference on computer vision theory and applications.
- Baró X, Escalera S, Vitrià J, Pujol O, Radeva P (2009) Traffic sign recognition using evolutionary adaboost detection and forest-ECOC classification. IEEE Trans Intell Transp Syst 10(1):113–126.
- Bishop CM (2006) Pattern recognition and machine learning. Information science and statistics. Springer, New York
- Ciresan D, Meier U, Schmidhuber J (2012) Multi-column deep neural networks for image classification. In: 2012 IEEE conference on computer vision and pattern recognition. IEEE, pp 3642–3649. doi:10.1109/CVPR.2012.6248110, arXiv:1202.2745v1
- Fleyeh H, Davami E (2011) Eigen-based traffic sign recognition. IET Intell Transp Syst 5(3):190. doi:10.1049/iet-its.2010.0159

9. APPENDICES

Appendix A: Dataset Details

- **Dataset Sources**: The project utilizes publicly available datasets such as the German Traffic Sign Recognition Benchmark (GTSRB) and the Belgian Traffic Sign Dataset (BTSD).
- **Preprocessing**: Images were resized to 32x32 pixels and normalized. Data augmentation techniques like rotation, scaling, and horizontal flipping were applied to increase the dataset's diversity.
- **Labeling**: Each traffic sign image is labeled with its corresponding traffic sign class, following the standard classification schema provided by the datasets.

Appendix B: Model Architecture

- **Convolutional Neural Network (CNN)**: The primary model architecture for traffic sign recognition.
    - **Layers**:
        - Convolutional layers with ReLU activation functions.
        - Max-pooling layers for down-sampling.
        - Fully connected layers leading to the output layer.
    - **Activation Functions**: ReLU for hidden layers, Softmax for the output layer.
    - **Optimizer**: Adam optimizer with a learning rate of 0.001.
    - **Loss Function**: Categorical cross-entropy.

Appendix C: Hardware and Software

- **Hardware**:
    - GPU: NVIDIA GTX 1080 Ti
    - CPU: Intel i7-9700K
    - RAM: 32GB DDR4
- **Software**:
    - **Programming Language**: Python
    - **Libraries**: TensorFlow, Keras, OpenCV, NumPy, Matplotlib
    - **Development Environment**: Jupyter Notebook

Appendix D: Training Details

- **Training Parameters**:
    - **Batch Size**: 64
    - **Epochs**: 50
    - **Validation Split**: 20% of the training data used for validation
- **Early Stopping**: Implemented to halt training when the validation loss stopped improving.

Appendix E: Evaluation Metrics

- **Accuracy**: Percentage of correctly classified traffic sign images.
- **Precision, Recall, F1-Score**: Evaluated for each traffic sign class to measure the model's performance.
- **Confusion Matrix**: Used to visualize the performance across different classes.

Appendix F: Real-Time Detection Implementation

- **Framework**: Integration with OpenCV for real-time video processing.
- **Detection Pipeline**:
    - **Image Capture**: Real-time frame capture from a video feed.
    - **Preprocessing**: Resizing and normalization of frames.
    - **Detection and Recognition**: Using the trained CNN model to detect and classify traffic signs in each frame.
    - **Overlay and Display**: Drawing bounding boxes and labels on detected traffic signs in the video feed.

Appendix G: Challenges and Limitations

- **Challenges**:
    - Handling varying lighting conditions and occlusions in real-time scenarios.
    - Ensuring real-time processing speed and maintaining high accuracy.
- **Limitations**:
    - Model performance can degrade with unbalanced datasets.
    - False positives and negatives in highly cluttered scenes.

Appendix H: Future Work

- **Enhancements**:
    - Incorporation of more advanced neural network architectures such as YOLO or SSD for improved real-time performance.
    - Exploration of semi-supervised learning techniques to leverage unlabeled data.
- **Expansion**:
    - Extending the system to support more diverse traffic sign datasets from different countries.
    - Developing mobile and embedded system versions of the application for broader usability.

Appendix I: References

1. German Traffic Sign Recognition Benchmark (GTSRB) dataset.
2. Belgian Traffic Sign Dataset (BTSD).
3. "Deep Learning for Traffic Sign Detection and Recognition" by XYZ.
4. TensorFlow and Keras official documentation.