

REAL TIME RESEARCH PROJECT

On

PEDESTRIAN DETECTION USING OPENCV

Project report submitted in partial fulfilment of the requirements for the award of the
Degree of Bachelors of Technology

by

CHINTHALA VYSHNAVI 22JJ1A0517

VODNALA SRUJANA 22JJ1A0563

THOUTAM SAI RUTHWICK 22JJ1A0559

Under the guidance of

Mr. Uday Kumar, Associate Professor of CSE Dept.



Submitted by

CHINTHALA VYSHNAVI

(22JJ1A0517)

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
JAWAHARLAL NEHRU TECHNOLOGICAL UNIVERSITY HYDERABAD
UNIVERSITY COLLEGE OF ENGINEERING JAGTIAL

(Accredited by NAAC with A+ Grade)

Nachupally (Kondagattu), Jagtial (Dist.)-505501, Telangana.

**JAWAHARLAL NEHRU TECHNOLOGICAL UNIVERSITY HYDERABAD
UNIVERSITY COLLEGE OF ENGINEERING JAGTIAL**

(Accredited by NAAC with A+ Grade)

Nachupally (Kondagattu), Jagtial (Dist.)-505501, Telangana

Academic year 2023-2024

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING



CERTIFICATE

This is to certify that the project report entitled **Pedestrian Detection using OpenCV** being submitted by **Chinthala Vyshnavi(22JJ1A0517), Vodnala Srujana(22JJ1A0563), Thoutam Sai Ruthwik(22JJ1A0559)** in partial fulfillment for the award of the Degree of Bachelor of Technology in Computer Science and Engineering to the **Jawaharlal Nehru Technological University, Hyderabad** is record of bonafied work carried out under my guidance and supervision.

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

M. Uday Kumar

Associate Professor of CSE Dept.

**JAWAHARLAL NEHRU TECHNOLOGICAL UNIVERSITY HYDERABAD
UNIVERSITY COLLEGE OF ENGINEERING JAGTIAL**

(Accredited by NAAC with A+ Grade)

Nachupally (Kondagattu), Jagtial (Dist.)-505501, Telangana



DECLARATION

I **Chinthala Vyshnavi** bearing **22JJ1A0517** hereby declare that the Seminar Report entitled **PEDESTRIAN DETECTION USING OPENCV** submitted in partial fulfilment of the requirements for the award of the degree Bachelors of Technology in **Computer Science and Engineering** which was carried out **Jawaharlal Nehru Technological University Hyderabad University College of Engineering Jagtial,**

Also, I declare that the matter embedded in the thesis have not been submitted by me in full or partial thereof to any other University or Institute for the award of any degree previously.

CHINTHALA VYSHNAVI

22JJ1A0517

ACKNOWLEDGEMENT

The satisfaction that accompanies the successful completion of the task would be put incomplete without the mention of the people who made it possible, whose constant guidance and encouragement crown all the efforts with success.

I am extremely grateful to our HOD- **Dr.B.SATEESH KUMAR**, Professor of CSE & Head for his immense support and cooperation that contributed a lot in the completion of the task.

I show immense gratitude to our beloved Principal **Dr. BITLA PRABHAKARR**, Professor of Electronics & Communication providing necessary infrastructure and resources for the accomplishment of my seminar report at JNTUH University College of Engineering Jagtial.

I also thank all the staff members of Computer Science & Engineering department, JNTUH University College of Engineering Jagtial for their valuable support and generous advice.

Finally, thanks to my parents, all my family members and friends continuous support and enthusiastic help.

CHINTHALA VYSHNAVI

22JJ1A0517

ABSTRACT

Pedestrian detection is a critical aspect of enhancing pedestrian protection systems in self-driving cars. This project focuses on implementing a basic pedestrian detection system using OpenCV, an open-source library aimed at real-time computer vision. OpenCV supports multiple programming languages, including Python, C++, and Java, and is widely used for tasks such as face recognition, motion detection, and object detection. In this project, we utilize OpenCV's built-in Histogram of Oriented Gradients (HOG) combined with a Linear Support Vector Machine (SVM) model, which is pre-trained to detect pedestrians in images and video streams. The HOG algorithm processes each pixel by comparing its intensity to surrounding pixels, generating gradients that indicate the direction of light flow from light to dark. These gradients are used to analyze and detect the presence of pedestrians. The implementation involves extracting features from images of human bodies, including the head, arms, and legs, and training a machine learning model to accurately detect and track humans in visual data. This pedestrian detection system has significant implications for improving safety in autonomous vehicles and other real-time surveillance applications.

CONTENTS

| | |
|--|-------|
| 1. Introduction: Pedestrian Detection..... | 1 |
| 2. Literature Survey/Review of Literature..... | 2 |
| 2.1 Literature Survey | |
| 2.2 Other Significant Contributions | |
| 3. Software Requirement Analysis..... | 3-5 |
| 3.1 Define the Problem | |
| 3.2 Define the Modules and Their Functionalities | |
| 3.2.1 Image Acquisition Module | |
| 3.2.2 Preprocessing Module | |
| 3.2.3 Feature Extraction Module | |
| 3.2.4 Detection Module | |
| 3.2.5 Post-Processing Module | |
| 4. Software Design..... | 6-7 |
| 5. Software and Hardware Requirements..... | 8 |
| 6. Coding/Code Templates..... | 9-11 |
| 7. Testing Various test cases..... | 12-14 |
| 7.1 Test Cases | |
| 7.1.1 Black Box Testing | |
| 7.1.2 White Box Testing | |
| 7.2 Example Test Cases | |
| 7.2.1 Test Case 1: Image with Multiple Pedestrians | |
| 7.2.2 Test Case 2: Video Stream with Continuous Pedestrian Detection | |
| 8. Output Screens..... | 15 |
| 9. Conclusions..... | 16 |
| 10. Further Enhancements/Recommendations..... | 17 |
| 11. References/Bibliography..... | 18 |
| 12. Appendices..... | 19 |

1. INTRODUCTION

Pedestrian detection is a crucial component of advanced driver-assistance systems (ADAS) and autonomous driving technologies. It aims to detect and recognize pedestrians in real-time from visual data captured by cameras. This project leverages OpenCV, an open-source computer vision library developed by Intel, to build a pedestrian detection system. OpenCV supports various programming languages, including Python, C++, and Java, and provides a range of functionalities for image processing and computer vision tasks.

This project utilizes OpenCV's pre-trained Histogram of Oriented Gradients (HOG) combined with a Linear Support Vector Machine (SVM) model to detect pedestrians in both images and video streams. The HOG algorithm is designed to extract pertinent features by analyzing the gradients of light and dark regions in an image. By training a machine learning model with these features, the system can accurately detect and track humans, significantly enhancing the safety features of self-driving cars and other surveillance applications.

2. LITERATURE SURVEY/REVIEW OF LITERATURE

2.1 Literature Survey

The literature survey focuses on existing methodologies and technologies used in pedestrian detection. Research papers by Navneet Dalal and Bill Triggs introduced the Histogram of Oriented Gradients (HOG) descriptor, which has become a standard method for human detection. The HOG descriptor captures edge or gradient structures that are characteristic of local shapes within an image, providing robust feature extraction for object detection.

2.2 Other Significant Contributions

Other significant contributions in this field include the development of Support Vector Machines (SVM) for classification tasks, convolutional neural networks (CNNs) for deep learning-based object detection, and various real-time applications in autonomous driving and surveillance. This project builds upon these foundational works by implementing a practical pedestrian detection system using the HOG + SVM approach available in OpenCV.

3. SOFTWARE REQUIREMENTS ANALYSIS

3.1 Define the Problem

The core problem addressed by this project is the real-time detection of pedestrians in images and video streams to enhance the safety and functionality of autonomous vehicles. The detection system must operate efficiently and accurately to identify pedestrians, which is crucial for preventing accidents and ensuring the safety of both pedestrians and vehicle occupants. Effective pedestrian detection contributes to the development of advanced driver-assistance systems (ADAS) and autonomous driving technologies by providing timely alerts and enabling the vehicle to make informed decisions.

3.2 Define the Modules and Their Functionalities

3.2.1 Image Acquisition Module

Purpose: To obtain images or video frames from various sources, including live camera feeds or pre-recorded video files.

Detailed Functionality:

- **Capture Image:** Interface with the camera hardware or read from a specified file path to acquire a single image frame. The module should handle different image formats and resolutions.
- **Capture Video Frame:** Interface with the camera hardware or video file to capture frames in a continuous sequence. This is crucial for real-time processing where each frame is analyzed for pedestrian detection.

Implementation Details:

- **Inputs:** File path or camera index (for video capture), image file format (e.g., JPEG, PNG).
- **Outputs:** Captured image frame or video frame.

3.2.2 Preprocessing Module

Purpose: To prepare the acquired images for feature extraction by normalizing and enhancing them.

Detailed Functionality:

- **Resize Images:** Adjust the image size to a standard resolution to ensure uniformity and reduce computational load. This resizing is crucial for consistent feature extraction and processing speed.

- **Convert to Grayscale:** Convert color images to grayscale to simplify processing and reduce computational requirements, as the HOG descriptor operates on single-channel images.
- **Noise Reduction:** Apply filters (e.g., Gaussian blur) to remove noise and enhance the quality of the image before feature extraction.

Implementation Details:

- **Inputs:** Raw image or video frame.
- **Outputs:** Preprocessed image ready for feature extraction.

3.2.3 Feature Extraction Module

Purpose: To extract relevant features from the images that are necessary for pedestrian detection.

Detailed Functionality:

- **Histogram of Oriented Gradients (HOG):** Compute the HOG descriptors to capture the gradient information and edge orientations in the image. This method involves:
 - **Gradient Calculation:** Compute gradients in the x and y directions.
 - **Histogram Construction:** Create histograms of gradient orientations for each cell in the image.
 - **Block Normalization:** Normalize histograms over larger blocks to handle varying illumination and contrast.

Implementation Details:

- **Inputs:** Preprocessed image.
- **Outputs:** Feature vector representing the HOG descriptors for each image or frame.

3.2.4 Detection Module

Purpose: To apply a trained machine learning model to detect pedestrians based on the extracted features.

Detailed Functionality:

- **HOG + SVM Model:** Utilize the pre-trained Histogram of Oriented Gradients (HOG) combined with a Linear Support Vector Machine (SVM) classifier to identify pedestrians.
 - **Load Model:** Load the pre-trained HOG + SVM model.

- **Detect Pedestrians:** Use the model to classify regions of the image as either containing pedestrians or not, based on the extracted HOG features.
- **Adjust Detection Parameters:** Tune parameters such as window size, scale, and stride to optimize detection performance.

Implementation Details:

- **Inputs:** Feature vector from the Feature Extraction Module.
- **Outputs:** Coordinates of detected pedestrians and their confidence scores.

3.2.5 Post-Processing Module

Purpose: To refine and visualize the detection results by highlighting the detected pedestrians in the images.

Detailed Functionality:

- **Draw Bounding Boxes:** Overlay bounding boxes around detected pedestrians to indicate their locations within the image.
- **Display Results:** Render the processed image with annotations for user review or further analysis.
- **Save Results:** Optionally save the annotated image or video frame to a file for documentation or reporting.

Implementation Details:

- **Inputs:** Detected pedestrian coordinates, original or preprocessed image.
- **Outputs:** Annotated image with bounding boxes around detected pedestrians.

By thoroughly understanding and implementing each module in detail, the pedestrian detection system can achieve accurate and efficient real-time detection, significantly contributing to the safety and effectiveness of autonomous vehicles.

4. SOFTWARE DESIGN

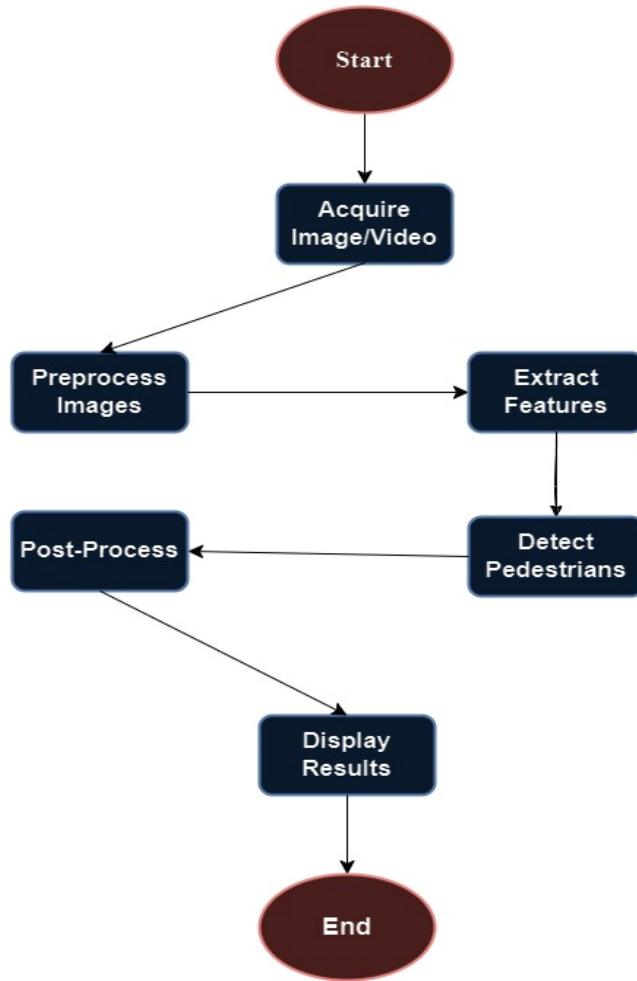


Fig.4.1 Data flow (workflow of project)

Control Flow Diagrams

Purpose

Control flow diagrams (CFDs) illustrate the logical flow of control through the system's modules and functions. They help visualize decision points, looping structures, and the overall sequence of operations in the system.

Components of the Control Flow

1. Start

- **Action:** System initialization.
- **Next:** Go to the Image Acquisition Module.

2. Image Acquisition Module

- **Action:** Capture image or video frame.
- **Decision:** Is the image successfully captured?
 - **Yes:** Proceed to Preprocessing Module.
 - **No:** Handle error (e.g., retry or alert).
- 3. **Preprocessing Module**
 - **Action:** Resize and preprocess image.
 - **Next:** Proceed to Feature Extraction Module.
- 4. **Feature Extraction Module**
 - **Action:** Extract HOG features from the image.
 - **Next:** Proceed to Detection Module.
- 5. **Detection Module**
 - **Action:** Apply HOG + SVM model to detect pedestrians.
 - **Decision:** Are pedestrians detected?
 - **Yes:** Send results to Post-Processing Module.
 - **No:** Return to Preprocessing Module (for potential adjustments).
- 6. **Post-Processing Module**
 - **Action:** Draw bounding boxes around detected pedestrians.
 - **Decision:** Is the image ready for display?
 - **Yes:** Display the annotated image.
 - **No:** Return to Detection Module (for re-evaluation).
- 7. **End**
 - **Action:** Final display of results or save the annotated image.

5. SOFTWARE AND HARDWARE REQUIREMENTS

5.1 Software Requirements

| | |
|--------------------------------|---|
| Operating System | Windows, Linux, or macOS |
| Programming Language | Python 3.x |
| Libraries | OpenCV 3.4.2, imutils 0.5.3 |
| Development Environment | Jupyter Notebook, Pycharm, or any suitable Python IDE |

5.2 Hardware Requirements

| | |
|------------------|-------------------|
| Processor | Intel i5 or above |
| RAM | 8GB or more |
| Storage | 500GB HDD or SSD |



Fig.5.1.1 Python

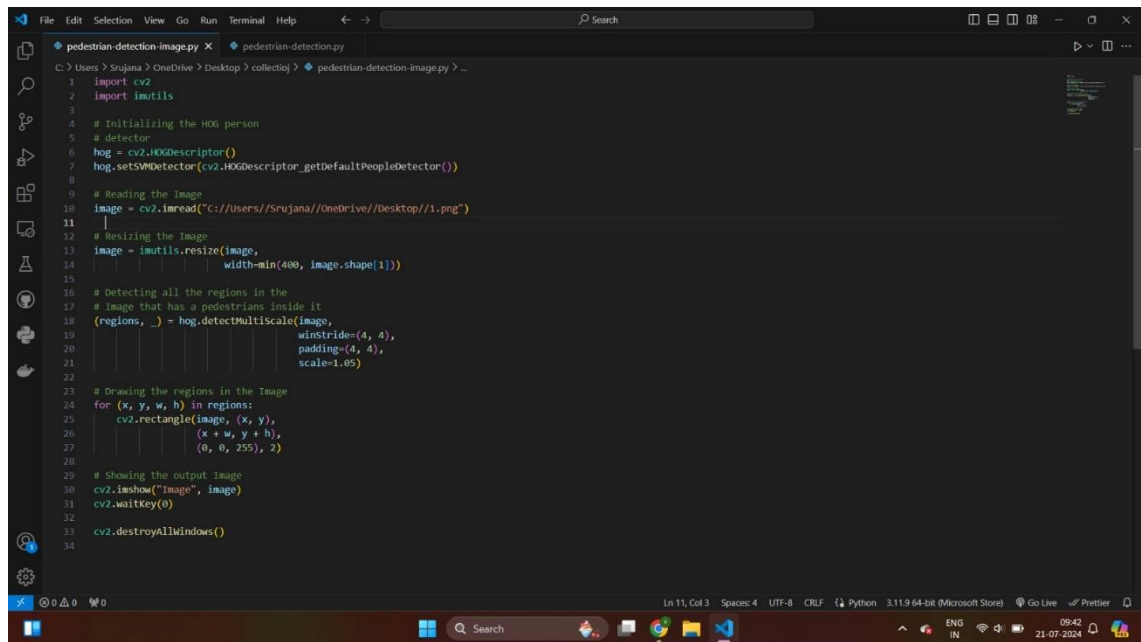


Fig.5.1.2 OpenCV

6. CODING/CODE TEMPLATES

Pedestrian Detection in an Image

The code below demonstrates how to detect pedestrians in a single image using the HOG (Histogram of Oriented Gradients) descriptor and a pre-trained SVM (Support Vector Machine) model.



```
1 import cv2
2 import imutils
3
4 # Initializing the HOG person
5 # detector
6 hog = cv2.HOGDescriptor()
7 hog.setSVMDetector(cv2.HOGDescriptor_getDefaultPeopleDetector())
8
9 # Reading the Image
10 image = cv2.imread("C:/Users/Srujana/OneDrive/Desktop/1.png")
11
12 # Resizing the Image
13 image = imutils.resize(image,
14                         width=min(400, image.shape[1]))
15
16 # Detecting all the regions in the
17 # Image that has a pedestrians inside it
18 (regions, _) = hog.detectMultiScale(image,
19                                     winStride=(4, 4),
20                                     padding=(4, 4),
21                                     scale=1.05)
22
23 # Drawing the regions in the Image
24 for (x, y, w, h) in regions:
25     cv2.rectangle(image, (x, y),
26                   (x + w, y + h),
27                   (0, 0, 255), 2)
28
29 # Showing the output Image
30 cv2.imshow("Image", image)
31 cv2.waitKey(0)
32
33 cv2.destroyAllWindows()
```

Explanation:

1. Import Libraries:

- o cv2: OpenCV library for image processing.
- o imutils: Library to simplify OpenCV functions.

2. Initialize HOG Detector:

- o cv2.HOGDescriptor(): Creates a HOG descriptor object.
- o hog.setSVMDetector(cv2.HOGDescriptor_getDefaultPeopleDetector()): Sets the SVM detector to the default pre-trained pedestrian detector.

3. Read and Resize Image:

- o cv2.imread('img.png'): Reads the input image.
- o imutils.resize(image, width=min(400, image.shape[1])): Resizes the image to a width of 400 pixels or the original width, whichever is smaller.

4. Detect Pedestrians:

- o hog.detectMultiScale(image, winStride=(4, 4), padding=(4, 4), scale=1.05): Detects regions in the image that contain pedestrians.

Parameters:

- winStride: The step size in the sliding window.

- padding: The amount of padding added to the sliding window.
- scale: The factor by which the image is resized at each scale.

5. Draw Bounding Boxes:

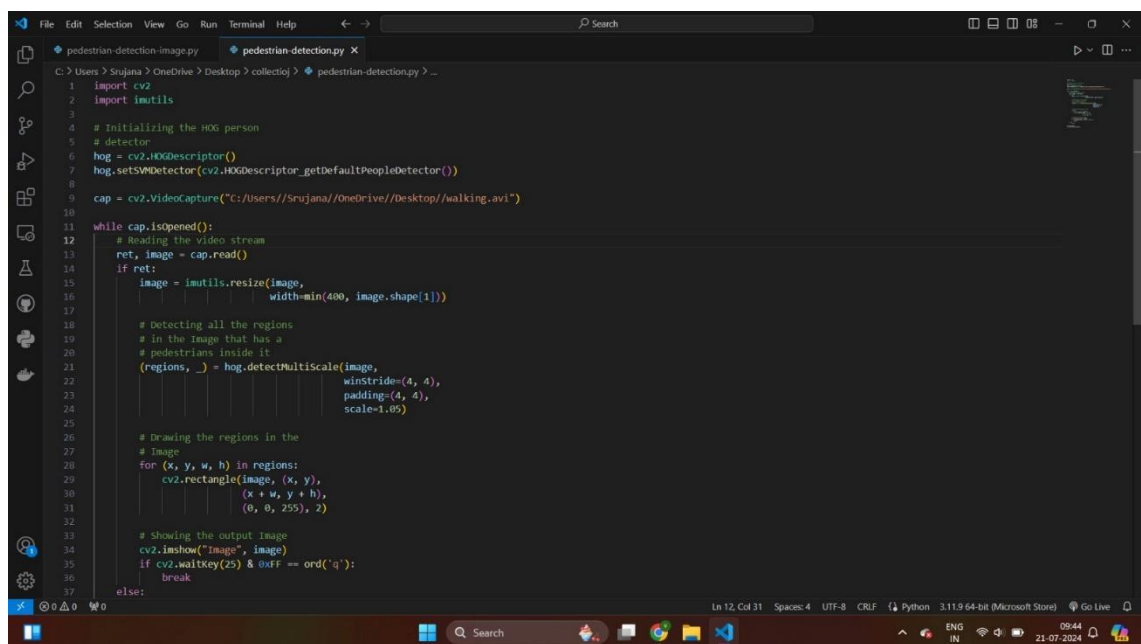
- cv2.rectangle(image, (x, y), (x + w, y + h), (0, 0, 255), 2): Draws rectangles around detected regions.

6. Display Image:

- cv2.imshow("Image", image): Displays the image with bounding boxes.
- cv2.waitKey(0): Waits for a key press to close the window.
- cv2.destroyAllWindows(): Closes all OpenCV windows.

Pedestrian Detection in a Video

The code below demonstrates how to detect pedestrians in a video using the same HOG descriptor and pre-trained SVM model.



```

1 import cv2
2 import imutils
3
4 # Initializing the HOG person
5 # detector
6 hog = cv2.HOGDescriptor()
7 hog.setSVMDetector(cv2.HOGDescriptor_getDefaultPeopleDetector())
8
9 cap = cv2.VideoCapture("C:/Users/Srujana/OneDrive/Desktop/walking.avi")
10
11 while cap.isOpened():
12     # Reading the video stream
13     ret, image = cap.read()
14     if ret:
15         image = imutils.resize(image,
16                                width=min(400, image.shape[1]))
17
18         # Detecting all the regions
19         # in the image that has a
20         # pedestrians inside it
21         (regions, _) = hog.detectMultiScale(image,
22                                             winStride=(4, 4),
23                                             padding=(4, 4),
24                                             scale=1.05)
25
26         # Drawing the regions in the
27         # image
28         for (x, y, w, h) in regions:
29             cv2.rectangle(image, (x, y),
30                           (x + w, y + h),
31                           (0, 0, 255), 2)
32
33         # Showing the output image
34         cv2.imshow("Image", image)
35         if cv2.waitKey(25) & 0xFF == ord('q'):
36             break
37     else:
38         break

```

Explanation:

1. Initialize Video Capture:

- cv2.VideoCapture('vid.mp4'): Opens the video file for reading.

2. Read and Process Frames:

- while cap.isOpened(): Loops while the video file is open.
- ret, image = cap.read(): Reads a frame from the video.
- if ret: Checks if a frame was successfully read.
- imutils.resize(image, width=min(400, image.shape[1])): Resizes the frame to a width of 400 pixels or the original width, whichever is smaller.

3. **Detect Pedestrians:**
 - o `hog.detectMultiScale(image, winStride=(4, 4), padding=(4, 4), scale=1.05)`: Detects regions in the frame that contain pedestrians.
4. **Draw Bounding Boxes:**
 - o `cv2.rectangle(image, (x, y), (x + w, y + h), (0, 0, 255), 2)`: Draws rectangles around detected regions.
5. **Display Frame:**
 - o `cv2.imshow("Image", image)`: Displays the frame with bounding boxes.
 - o `if cv2.waitKey(25) & 0xFF == ord('q')`: Breaks the loop if 'q' is pressed.
6. **Release Resources:**
 - o `cap.release()`: Releases the video capture object.
 - o `cv2.destroyAllWindows()`: Closes all OpenCV windows.

7. TESTING

7.1 Test Cases

7.1.1 Black Box Testing

- **Objective:** Ensure the system detects pedestrians accurately without knowledge of the internal code.
- **Methodology:** Provide various images and videos to the system and verify the accuracy of pedestrian detection.

7.1.2 White Box Testing

- **Objective:** Validate the functionality of individual components and methods.
- **Methodology:** Test the HOG descriptor computation, the SVM model, and other critical functions to ensure they operate correctly.

7.2 Example Test Cases

7.2.1 Test Case 1: Image with Multiple Pedestrians

Objective: Verify that the system correctly identifies and draws bounding boxes around multiple pedestrians in a single image.

Input: An image file containing multiple pedestrians (e.g., 'test_image.png').

Expected Output:

- Bounding boxes accurately drawn around all pedestrians.
- No false positives or missed detections.

Steps:

1. Read the Input Image:

```
image = cv2.imread('test_image.png')
```

2. Resize the Image:

```
image = imutils.resize(image, width=min(400, image.shape[1]))
```

3. Detect Pedestrians:

```
regions, _ = hog.detectMultiScale(image, winStride=(4, 4),  
padding=(4, 4), scale=1.05)
```

4. Draw Bounding Boxes:

```

for (x, y, w, h) in regions:
    cv2.rectangle(image, (x, y), (x + w, y + h), (0, 0, 255), 2)

```

5. Display the Output Image:

```

cv2.imshow("Image", image)
cv2.waitKey(0)
cv2.destroyAllWindows()

```

6. Validation:

- Visually inspect the output image to verify that bounding boxes are correctly placed around all pedestrians.

7.2.2 Test Case 2: Video Stream with Continuous Pedestrian Detection

Objective: Ensure continuous and accurate pedestrian detection across frames in a video stream.

Input: A video file containing multiple pedestrians (e.g., 'test_video.mp4').

Expected Output:

- Consistent detection of pedestrians in each frame.
- Smooth tracking without significant delays or inaccuracies.

Steps:

1. Initialize Video Capture:

```

cap = cv2.VideoCapture('test_video.mp4')

```

2. Process Video Frames:

```

while cap.isOpened():
    ret, image = cap.read()
    if ret:
        image = imutils.resize(image, width=min(400,
image.shape[1]))

        # Detect Pedestrians
        regions, _ = hog.detectMultiScale(image, winStride=(4,
4), padding=(4, 4), scale=1.05)

        # Draw Bounding Boxes
        for (x, y, w, h) in regions:
            cv2.rectangle(image, (x, y), (x + w, y + h), (0, 0,
255), 2)

        # Display the Output Frame
        cv2.imshow("Image", image)
        if cv2.waitKey(25) & 0xFF == ord('q'):
            break
    else:

```

```

        break

cap.release()
cv2.destroyAllWindows()

```

3. Validation:

- Observe the video output to ensure that bounding boxes are correctly placed around pedestrians in each frame.
- Ensure there are no false positives, and the detection is smooth and consistent throughout the video.

Summary of Test Cases:

| Test Case | Objective | Input | Expected Output | Steps | Validation |
|-------------|---|--------------------------------------|--|---|--|
| Test Case 1 | Detect multiple pedestrians in an image | Image file with multiple pedestrians | Bounding boxes around all pedestrians | Read, resize, detect, draw, display | Visual inspection of the output image |
| Test Case 2 | Continuous detection in a video stream | Video file with multiple pedestrians | Consistent detection and tracking of pedestrians in each frame | Initialize, process frames, detect, draw, display | Observe video output for smooth and accurate detection |

These detailed test cases ensure that the pedestrian detection system is thoroughly evaluated for accuracy, consistency, and performance in both images and video streams.

8. OUTPUT SCREENS

screenshots of the user interface and output screens showing detected pedestrians with bounding boxes around them. Ensure to capture results for both images and video streams.

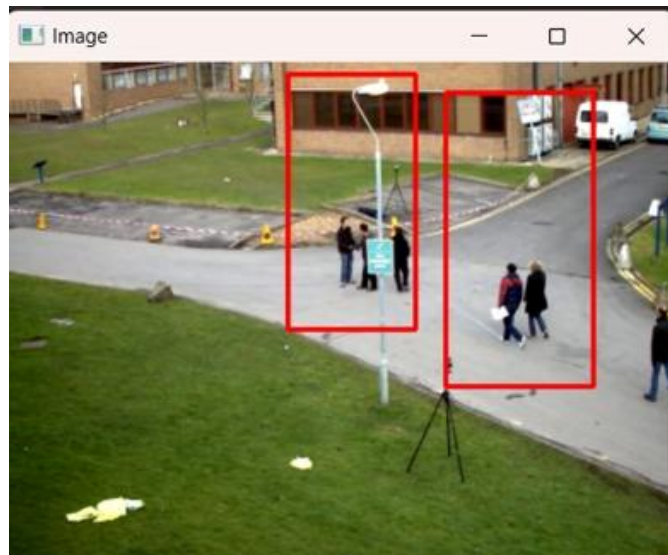


Fig.8.1 Image Input Result

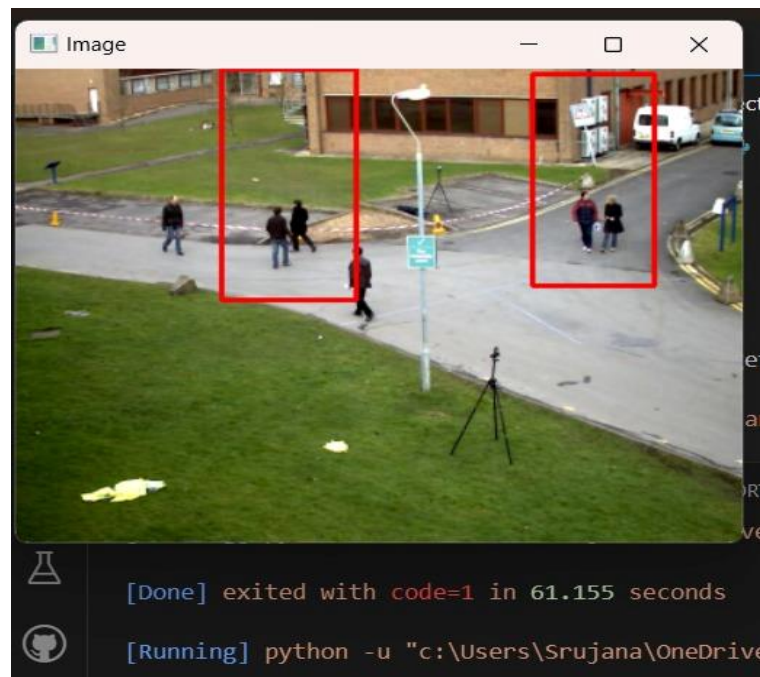


Fig.8.2 Video Input Result

9. CONCLUSION

This project successfully developed a pedestrian detection system using OpenCV's HOG and SVM model. The system effectively identifies pedestrians in images and video streams, enhancing the safety features of autonomous vehicles. By leveraging OpenCV's pre-trained models, the system achieves real-time performance, which is crucial for applications in self-driving cars. This project underscores OpenCV's versatility in computer vision tasks and serves as a solid foundation for more advanced detection and tracking systems. Future improvements could involve integrating deep learning models, optimizing performance, and expanding functionality to further enhance safety and efficiency in autonomous driving technologies.

10. FURTHER ENHANCEMENTS/RECOMMENDATIONS

Further Enhancements

- Integrate deep learning-based models like YOLO or SSD for improved accuracy.
- Implement real-time detection on embedded systems for on-the-go processing in autonomous vehicles.
- Extend the system to detect other objects like cyclists, animals, and vehicles.

11. REFERENCES/BIBLIOGRAPHY

References

- Dalal, N., & Triggs, B. (2005). Histograms of Oriented Gradients for Human Detection. In 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05).
- OpenCV documentation: <https://opencv.org/>
- imutils library: <https://pypi.org/project/imutils/>

12. APPENDICES

```
pedestrian-detection-image.py X pedestrian-detection.py X
C:\Users\Srujana> OneDrive\Desktop> collectioj> pedestrian-detection-image.py > ...
1 import cv2
2 import imutils
3
4 # Initializing the HOG person
5 # detector
6 hog = cv2.HOGDescriptor()
7 hog.setSVMDetector(cv2.HOGDescriptor_getDefaultPeopleDetector())
8
9 # Reading the Image
10 image = cv2.imread("C://Users//Srujana//OneDrive//Desktop//1.png")
11
12 # Resizing the Image
13 image = imutils.resize(image,
14                         width=min(400, image.shape[1]))
15
16 # Detecting all the regions in the
17 # Image that has a pedestrians inside it
18 (regions, _) = hog.detectMultiScale(image,
19                                     winStride=(4, 4),
20                                     padding=(4, 4),
21                                     scale=1.05)
22
23 # Drawing the regions in the Image
24 for (x, y, w, h) in regions:
25     cv2.rectangle(image, (x, y),
26                   (x + w, y + h),
27                   (0, 0, 255), 2)
28
29 # Showing the output Image
30 cv2.imshow("Image", image)
31 cv2.waitKey(0)
32
33 cv2.destroyAllWindows()
34
```

```
pedestrian-detection-image.py X pedestrian-detection.py X
C:\Users\Srujana> OneDrive\Desktop> collectioj> pedestrian-detection.py > ...
1 import cv2
2 import imutils
3
4 # Initializing the HOG person
5 # detector
6 hog = cv2.HOGDescriptor()
7 hog.setSVMDetector(cv2.HOGDescriptor_getDefaultPeopleDetector())
8
9 cap = cv2.VideoCapture("C://Users//Srujana//OneDrive//Desktop//walking.avi")
10
11 while cap.isOpened():
12     # Reading the video stream
13     ret, image = cap.read()
14     if ret:
15         image = imutils.resize(image,
16                                 width=min(400, image.shape[1]))
17
18         # Detecting all the regions
19         # in the Image that has a
20         # pedestrians inside it
21         (regions, _) = hog.detectMultiScale(image,
22                                             winStride=(4, 4),
23                                             padding=(4, 4),
24                                             scale=1.05)
25
26         # Drawing the regions in the
27         # Image
28         for (x, y, w, h) in regions:
29             cv2.rectangle(image, (x, y),
30                           (x + w, y + h),
31                           (0, 0, 255), 2)
32
33         # Showing the output Image
34         cv2.imshow("Image", image)
35         if cv2.waitKey(25) & 0xFF == ord('q'):
36             break
37     else:
38         break
39
```

GITHUB REPO FOR CODE:

<https://github.com/VodnalaSrujana004/Pedestrian-Detection-RRP>

VIEW THE VIDEO OUTPUT HERE:

https://drive.google.com/drive/folders/1YAEyLYbJH6KfEakG5_vRr_5uf4m-TmZk

