IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 5, Issue 7, June 2025
Impact Factor: 7.67

Live Object Detection Using Python

Sakibayan Sheikh[1], Rohan Chaware[2], Suhani Kotambkar[3], Prof. Manoj Chittawar[4]
Department of Computer Science[1-4]
Rajiv Gandhi College of Engineering Research and Technology, Chandrapur, Maharashtra

Abstract: *This project presents a real-time object detection system designed for applications such as surveillance, autonomous navigation, robotics, and interactive technologies. By leveraging Python, OpenCV, and deep learning frameworks like TensorFlow and PyTorch, the system processes live video input to detect and classify objects using pre-trained models such as YOLO, SSD, and Faster R-CNN. The implementation emphasizes real-time performance, accuracy, and low false positive rates. The system's design ensures scalability and adaptability, making it suitable for deployment across various platforms requiring efficient live object detection capabilities*

Keywords: Object Detection, Real-Time Processing, OpenCV, YOLO, Deep Learning, Python

I. INTRODUCTION

Real-time object detection involves identifying and locating objects within a continuous video feed, such as one from a webcam or video file. Using Python, this can be accomplished by capturing frames from the stream and analyzing them with pre-trained object detection models. Tools like OpenCV, along with deep learning libraries such as TensorFlow and PyTorch, facilitate this process. Each frame is processed instantly, enabling the application of models like YOLO, SSD, or Faster R-CNN to detect objects and draw bounding boxes around them. This technology finds use in various domains, including robotics, augmented reality, and surveillance, and serves as a powerful and innovative solution for dynamic visual recognition tasks.

II. CONTENT DETAILS

A. Aims and Objectives.

1. Real-Time Detection: Develop a system capable of detecting and classifying objects in a live video stream.
2. Efficient Processing: Minimize latency to enable prompt decision-making in real-world scenarios.
3. Robust Performance: Maintain detection accuracy under various lighting and environmental conditions.
4. Model Integration: Utilize deep learning models like YOLO, SSD, or Faster R-CNN.
5. Video Processing: Employ OpenCV for real-time frame handling and visualization.

B. Methodology

The system follows these core steps:

1. Library Setup: Install and configure required packages like opencv-python, tensorflow, numpy.
2. Model Selection: Use a pre-trained detection model (e.g., YOLOv3 via TensorFlow).
3. Video Capture: Access camera feed using OpenCV's VideoCapture.
4. Frame Processing: Resize and normalize input for the model.
5. Detection Pipeline: Apply the detection model to each frame.
6. Visualization: Use OpenCV to draw bounding boxes and labels.
7. Live Display: Continuously display processed frames with detection overlay.

C. Module Description

This module is about real-time object detection and classification using Python and computer vision. It discusses the fundamentals of image processing, machine learning, and deep learning with hands-on implementation using libraries such as OpenCV, TensorFlow, and YOLO (You Only Look Once). The module addresses camera interfacing, data

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 5, Issue 7, June 2025
Impact Factor: 7.67

acquisition, model training, and live detection so that students can create applications capable of dynamically identifying and tracking objects from video streams or webcam feeds.

Key topics include:

- Computer Vision Fundamentals
- OpenCV and camera module configuration
- Pre-trained object detection models (YOLO, SSD, etc.)
- Real-time object detection pipeline
- Performance optimization and tuning
- Application development and deployment

At the end of this module, students will be able to design and deploy real-time object detection systems for many common everyday applications such as surveillance, autonomous navigation, and smart automation.

Algorithm Description

The detection process comprises:

1. Video Capture: Grab frames using OpenCV.
2. Preprocessing: Resize, normalize as per model input format.
3. Model Loading: Load pre-trained model (e.g., YOLOv5).
4. Object Detection: Model predicts bounding boxes, labels, confidence.
5. Post-Processing: Apply Non-Maximum Suppression (NMS).
6. Annotation: Draw bounding boxes and labels.
7. Display: Continuously refresh frames in real-time.
8. Exit: Stop on user input (q key).

E . UML & ER Diagram

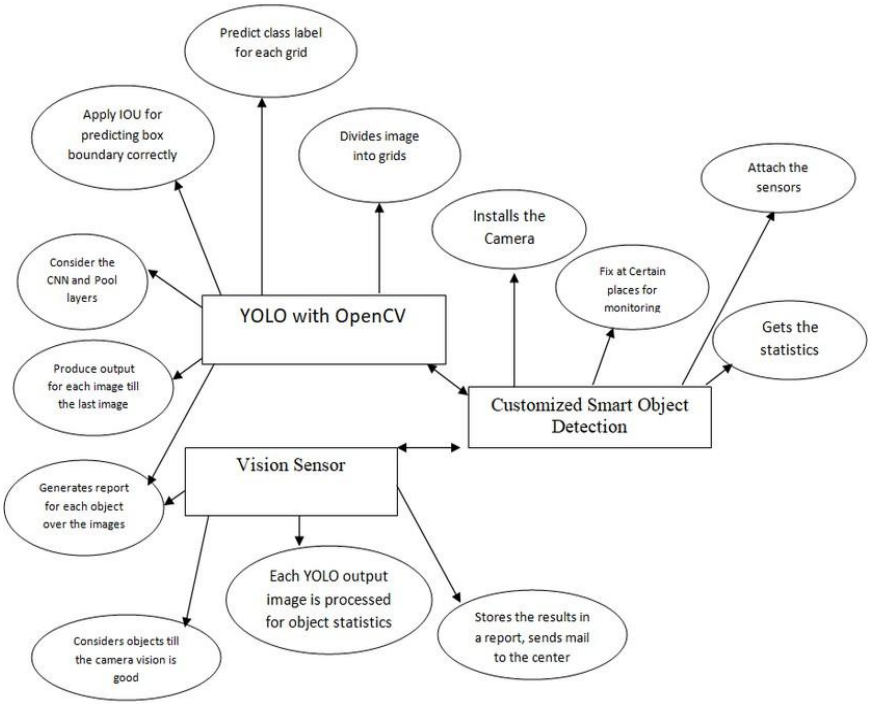

Figure.1. ER-DIAGRAM

DOI: 10.48175/568

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 5, Issue 7, June 2025
Impact Factor: 7.67

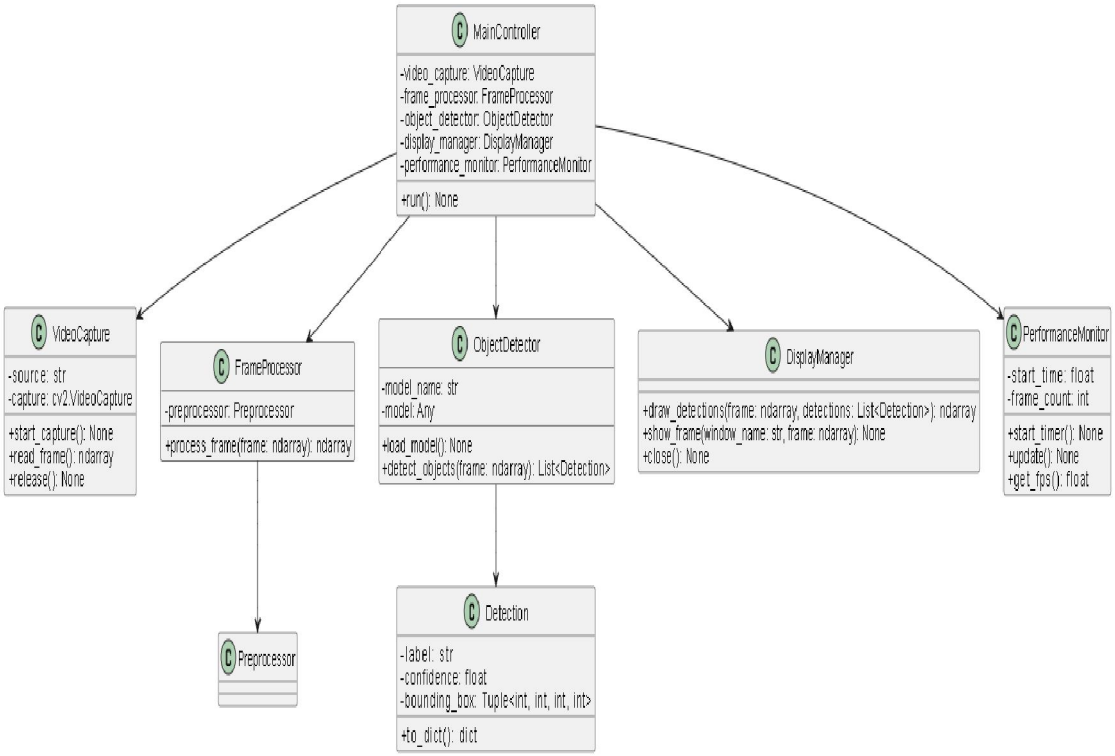

Figure . 2 .UML-Diagram

F. Output.

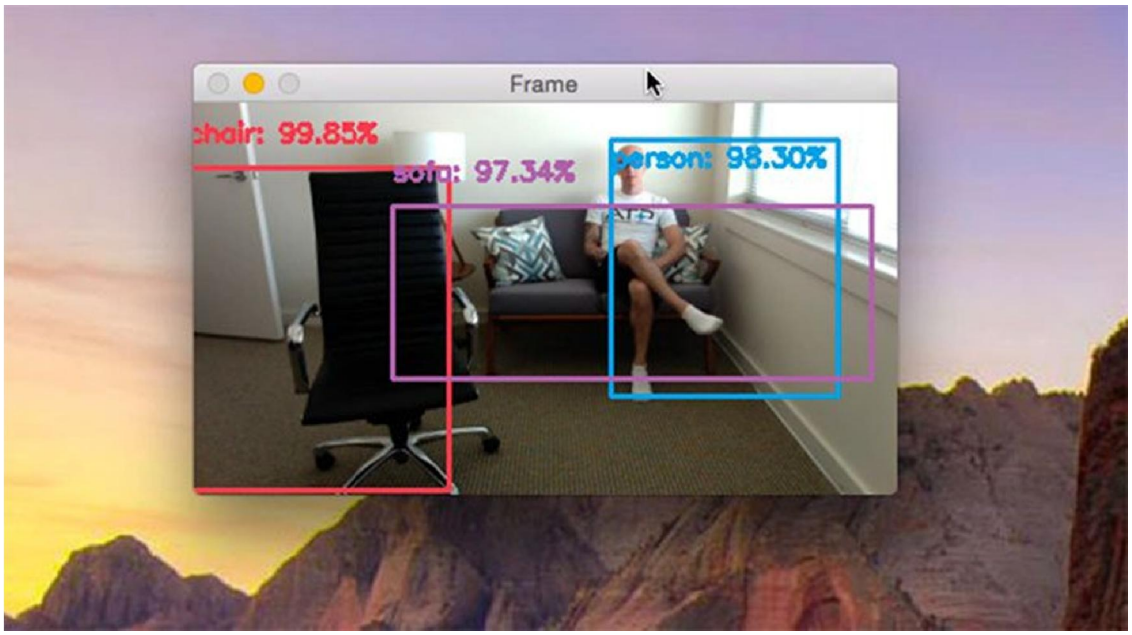

Figure.3.Expected Outcome.

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 5, Issue 7, June 2025
Impact Factor: 7.67

III. CONCLUSION

The developed live object detection system highlights the practical use of deep learning models alongside OpenCV in Python to enable accurate and efficient real-time object recognition. By integrating pre-trained models like YOLO, SSD, or Faster R-CNN, the system can detect multiple objects from live video input with low latency. Its modular structure supports adaptability, making it suitable for applications in areas such as surveillance, autonomous navigation, robotics, and intelligent automation. The use of open-source tools allows for cost-effective development and easy customization. Future improvements may involve deploying the system on edge computing platforms like Raspberry Pi or NVIDIA Jetson to enhance portability and on-site processing capabilities. Additionally, incorporating custom-trained models tailored to specific use cases and introducing event-driven actions—such as sending alerts upon identifying particular objects—can further improve the system's utility and responsiveness in practical scenarios.

IV. ACKNOWLEDGMENT

I would like to express my sincere gratitude to all those who supported me throughout the course of this project.First and foremost, I am deeply thankful to our mentor, **Prof. Manoj Chittawar**, for his invaluable guidance, encouragement, and continuous support during the course of this work. His insights and expertise were instrumental in the successful completion of this project.I also extend my heartfelt thanks to the **Department of Computer Science, RCERT College, Chandrapur, India**, for providing the necessary resources and a conducive environment for learning and development.

REFERENCES

[1] E. Matthes, Python Crash Course, 2nd ed., No Starch Press, 2019.
[2] A. Sweigart, Automate the Boring Stuff with Python, 2nd ed., No Starch Press, 2019.
[3] W. McKinney, Python for Data Analysis, 2nd ed., O'Reilly Media, 2017.
[4] M. Lutz, Learning Python, 5th ed., O'Reilly Media, 2013.
[5] Roboflow – https://roboflow.com/
[6] Ultralytics HUB – https://ultralytics.com/
[7] OpenCV – https://opencv.org