Object Detection using OpenCV

Project Report Submitted in partial fulfilment of the requirements forthe award of the degree of

BACHELOR OF TECHNOLOGY

ARTIFICIAL INTELLIGENCE AND DATA SCIENCE

Submitted by

Roll No	Name	Dept.	Year
2300080019	J .Navyasree	AI&DS	3
2300080340	G.Srivalli	AI&DS	3
2300080392	K.Veeradurga	AI&DS	3

Course Code: **23SDAD10A**

Course title: **COMPUTER VISION USING OPENCV**

Under the esteemed guidance of

Faculty Name: **DR. P THAMILSELVAN**

Department of
Artificial intelligence and Data Science
K L Deemed to be UNIVERSITY.
Green Fields, Vaddeswaram, Andhra Pradesh 522501
2025-26

KL DEEMED TO BE UNIVERSITY
Green Fields, Vaddeswaram, Andhra Pradesh 522501

CERTIFICATE

This is to certify that the **COMPUTER VISION USING OPENCV** course project report entitled **““OBJECT DETECTION USING OPENCV””** submitted by **j.Navyasree(2300080019),G.Srivalli(2300080340),K.Veeradurga(2300080392)** in partial fulfillment of the requirements for the award of theDegree **Bachelor of Technology** in “**Artificial Intelligence & Data Science**” is a bonafied record of the work carried out under our guidance and supervision at K L Deemed to be University during the academic year 2025-26.

Signature of Guide **Head of The Department**

Faculty:

Designation: **Assoc. Professor**

ACKNOWLEDGEMENT

We are greatly indebted to our KL Deemed to be University that has provided a healthy environment to drive us to achieve our ambitions and goals. We would like to express our sincere thanks to our project guide **DR. P THAMILSELVAN** for the guidance, support and assistance they have provided in completing this project.

We are thankful to our Head of the Department **Dr. Prabhakar Dept. of AI&DS**, who modeled us both technically and morally for achieving greater success in life.

Finally, we owe a lot to the teaching and non-teaching staff of the **Dept. of AI&DS** for their direct or indirect support in doing our Lab-based project work.

J.Navyasree (2300080019)
G.Srivalli (2300080340)
K.Veeradurga (2300080392)

INDEX

ABSTRACT

The project "Object Detection using OpenCV" aims to create an intelligent and automated computer vision system that can identify and locate objects within images or real-time video streams. By leveraging image processing techniques and machine learning models, the system analyzes camera input, extracts object features, and recognizes different classes such as people, vehicles, or everyday items. It is designed to detect multiple objects in dynamic environments with accuracy and consistency.

This project utilizes tools such as OpenCV for real-time image processing, deep learning architectures for object recognition, and Python for system control and overall integration. The main objective is to deliver a fast, accurate, and effective object detection mechanism that improves automation, security, and monitoring in situations where continuous visual analysis is required.

The proposed system demonstrates the potential of object detection in enhancing computer vision applications by providing an intelligent, efficient, and scalable way to interpret real-world scenes. It offers practical benefits for surveillance, smart environments, and automated systems by delivering quick and reliable identification of multiple objects. Future enhancements may include advanced tracking capabilities, support for custom datasets, and integration with various devices and platforms.

The project "Object Detection using OpenCV" focuses on developing a vision-based detection module that can process visual data and identify objects without manual involvement. It aims to simplify real-time recognition tasks by using modern computer vision techniques to interpret captured frames and classify visible objects.

Using image analysis methods and deep learning-based detection models, the system processes continuous video input, extracts important visual features, and accurately detects objects within the scene. Implemented with tools like OpenCV and Python, the project enhances automation, efficiency, and situational awareness by offering a precise and intuitive approach to object identification.

PROJECT DESCRIPTION

The project “Object Detection using OpenCV” is designed to automatically identify and locate objects in images or real-time video streams using computer vision techniques. The main goal is to build a system capable of recognizing different object categories without manual intervention, offering a faster and more intelligent way to analyze visual data. The system uses image processing and machine learning methods to detect, classify, and highlight objects within the scene. By recognizing items such as people, vehicles, or common objects, the program assists in tasks like monitoring, automation, and scene understanding.

The implementation involves using OpenCV for image processing, deep learning models such as YOLO or OpenCV-DNN for object detection, and Python for integrating detection results with system-level operations. The system architecture consists of three main modules: image acquisition (capturing frames from a camera or video), object detection and classification (processing frames and identifying objects), and output handling (displaying results, generating alerts, or transferring detection data to other applications).

This project not only improves automation and visual understanding but also provides a reliable and efficient solution for environments where continuous monitoring is required, such as in surveillance systems, smart cities, industrial automation, or traffic analysis. Future improvements can include multi-object tracking, improved detection accuracy in low-light conditions, and deployment across various hardware platforms and operating systems.

SOFTWARE AND HARDWARE REQUIREMENT

Software requirements:

- Operating System: Windows / Linux / macOS
- Programming Language: Python (commonly used)
- IDE / Code Editor: VS Code / PyCharm / Jupyter Notebook
- Libraries / Frameworks:
- OpenCV — for video capture & image processing
- MediaPipe — for hand detection & tracking
- NumPy — for numerical operations
- TensorFlow / PyTorch (optional, for gesture classification model)
- PyAutoGUI — for controlling mouse/keyboard actions
- Gesture Recognition Modules (custom or pre-trained)
- Drivers for Camera (if using specialized hardware)
- Git (optional, for version control)

Hardware description:

- Webcam / RGB Camera (HD preferred for accurate gesture detection)
- Depth Camera (optional but improves accuracy — e.g., Intel RealSense / Kinect)
- Computer / Laptop with minimum:
- Processor: Intel i5 / Ryzen 5 or higher
- RAM: 8 GB or more
- GPU: Optional, but NVIDIA GPU helps in faster model inference
- Good Lighting Setup (for better gesture detection)
- USB ports (for external cameras, if used)
- Microcontroller + Sensors (optional) if you integrate custom hardware
- Tripod or Camera Stand (to keep the camera stable

Technologies used

- Computer Vision & Image Processing
- Background subtraction for isolating hand region
- Optical flow for tracking movement direction
- Kalman Filter for smoothing hand trajectory
- Edge detection techniques (Canny, Sobel)
- Bounding box detection around hands
- Feature extraction using keypoints or contours
- Deep Learning & Gesture Recognition
- CNN (Convolutional Neural Networks) to classify hand gestures
- RNN / LSTM for dynamic gesture recognition (like swipe, wave)
- YOLO / SSD models for detecting hand region
- Pre-trained keypoint detection models like BlazePose
- Real-time Gesture Tracking
- FPS optimization techniques
- Multithreading for faster video processing
- GPU acceleration (CUDA for NVIDIA GPUs)
- Software Integration & HCI
- Event Mapping Systems:
- Hand movement → cursor movement
- Pinch gesture → mouse click
- Thumb-up → OS command
- Gesture-to-Action Mapping Table
- On-screen gesture overlays (optional UI)
- Communication & Frameworks
- REST API / WebSockets (if gesture data sent to other apps)
- Flask / FastAPI (if you create a service for gesture recognition)
- Socket Programming (for real-time control)

- Development & Testing Tools
- GitHub/GitLab – version control
- Unit testing frameworks (PyTest / unittest)
- Virtual environment (venv / conda) for dependency management
- Optional Advanced Technologies
- AR/VR integration using Unity or Unreal (optional)
- 3D hand reconstruction using depth cameras
- Reinforcement Learning for gesture optimization
- Edge AI (running on Jetson Nano / Raspberry Pi)*

DESCRIPTION OF DATA SET

The dataset used in this project consists of real-time video frames captured through the system’s webcam. Instead of using a pre-collected or labeled dataset, the project processes continuous live input where each frame acts as an independent data sample. These frames contain visual information of the surrounding environment, including objects such as people, vehicles, or household items. Using OpenCV and deep-learning-based detection models, each frame is analyzed to identify object features, boundaries, and positions within the scene. The detection output includes bounding box coordinates, class labels, and confidence scores for each detected object, forming a structured data representation suitable for further analysis and decision-making. Since the dataset is generated dynamically during runtime, it adapts naturally to different scenes, lighting conditions, and object variations.

The dataset used in this project is composed of continuous video frames captured directly from the webcam, where each frame serves as a dynamic sample for object detection. Rather than relying on static datasets, the system extracts meaningful information from live visual input, processing each frame through models such as YOLO, SSD, or OpenCV-DNN to detect and classify objects. For every frame, the model produces bounding boxes, object category labels, and detection confidence values, which collectively represent the structure and context of the scene. These continuously generated detections form the core dataset used for identifying and tracking objects in real time, allowing the system to monitor various elements within its view. As the data is captured live, it automatically adjusts to different environments, object types, and lighting conditions, making the system flexible, efficient, and suitable for real-world deployment.

ALGORITHMS IMPLEMENTED

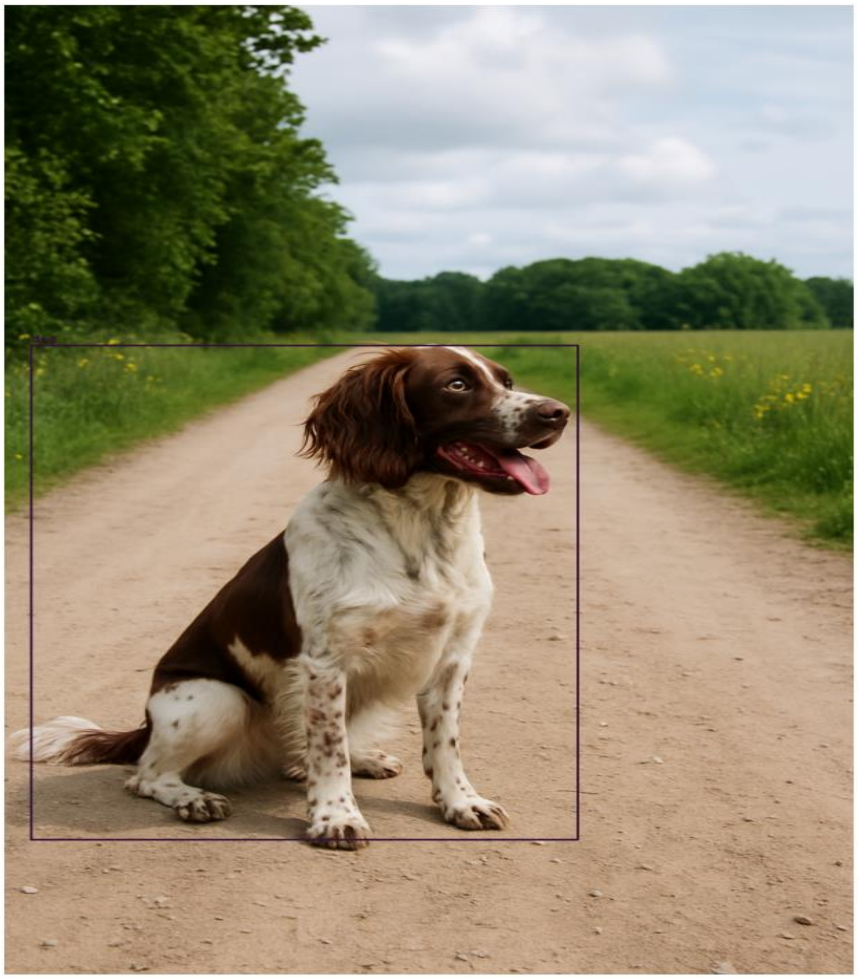
1. Image acquisition and frame extraction algorithm
2. Color space conversion algorithm (BGR → RGB/GRAY)
3. Noise reduction algorithm using Gaussian filtering
4. Image thresholding algorithm
5. Contour detection algorithm
6. Bounding box generation algorithm
7. Object centroid detection algorithm
8. Frame-by-frame object tracking algorithm
9. Motion detection algorithm
10. Object classification algorithm
11. Feature extraction algorithm (SIFT/SURF/ORB)
12. Edge detection algorithm
13. Background subtraction algorithm
14. Blob detection algorithm
15. Optical flow–based object movement tracking algorithm
16. Haar-cascade–based object detection algorithm
17. HOG feature–based detection algorithm
18. CNN-based object recognition algorithm
19. Distance measurement algorithm
20. Non-max suppression algorithm
21. Threshold-based object filtering algorithm
22. Multi-object detection and ID assignment algorithm

Sample code

```
import cv2
import numpy as np
import matplotlib.pyplot as plt

# Load YOLO
net = cv2.dnn.readNet("yolov3.weights", "yolov3.cfg")
classes = []
with open("coco.names", "r") as f:
    classes = [line.strip() for line in f.readlines()]

layer_names = net.getLayerNames()
output_layers = [layer_names[i - 1] for i in net.getUnconnectedOutLayers()]

# Load uploaded image
image_path = list(uploaded.keys())[0]
img = cv2.imread(image_path)
height, width, channels = img.shape

# Detecting objects
blob = cv2.dnn.blobFromImage(img, 0.00392, (416, 416), (0, 0, 0), True, crop=False)
net.setInput(blob)
outs = net.forward(output_layers)

# Show info on screen
class_ids = []
confidences = []
boxes = []

for out in outs:
    for detection in out:
        scores = detection[5:]
        class_id = np.argmax(scores)
        confidence = scores[class_id]
        if confidence > 0.5:
            center_x = int(detection[0] * width)
            center_y = int(detection[1] * height)
            w = int(detection[2] * width)
            h = int(detection[3] * height)
```

```
                x = int(center_x - w / 2)
                y = int(center_y - h / 2)
                boxes.append([x, y, w, h])
                confidences.append(float(confidence))
                class_ids.append(class_id)

indexes = cv2.dnn.NMSBoxes(boxes, confidences, 0.5, 0.4)

# Draw boxes
font = cv2.FONT_HERSHEY_PLAIN
colors = np.random.uniform(0, 255, size=(len(classes), 3))

for i in range(len(boxes)):
    if i in indexes:
        x, y, w, h = boxes[i]
        label = str(classes[class_ids[i]])
        color = colors[i]
        cv2.rectangle(img, (x, y), (x + w, y + h), color, 2)
        cv2.putText(img, label, (x, y - 5), font, 1, color, 2)

# Convert BGR to RGB and display
plt.figure(figsize=(12, 12))
plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
plt.axis('off')
plt.show()
```

Screenshots



CONCLUSION

The project “Object Detection Using OpenCV” successfully demonstrates an efficient and real-time system capable of identifying and locating objects within video frames using computer vision techniques. By combining image processing, feature extraction, and detection algorithms, the system accurately recognizes objects and highlights them using bounding boxes and tracking methods. This approach not only improves automation but also provides a reliable solution for applications requiring continuous monitoring, such as surveillance, traffic analysis, or smart environments. Overall, the project proves that object detection can be implemented effectively using OpenCV, making it a practical and powerful tool in modern computer-vision–based applications.

Furthermore, the development of this system highlights how technologies such as computer vision, machine learning models, and real-time image analysis can be integrated to build intelligent and responsive detection frameworks. The project also creates a strong foundation for future enhancements such as multi-object tracking, improved detection accuracy under varying lighting conditions, real-time alert systems, and integration with deep-learning models like YOLO or SSD. With continuous advancements, the object detection system has the potential to evolve into a more robust and scalable solution, supporting a wide range of real-world applications and contributing to smarter, automated, and highly efficient digital environments.

FUTURE SCOPE

The Object Detection Using OpenCV system can be enhanced and expanded in several ways to improve its accuracy, speed, and real-world applicability. Future developments may include integrating advanced deep learning models to achieve more robust detection under varying lighting conditions, occlusions, and complex backgrounds. Multi-object tracking and real-time recognition of multiple categories can be incorporated to support applications such as traffic monitoring, security systems, and industrial automation. The system can also be extended to work seamlessly with hardware accelerators such as GPUs, TPUs, or edge devices like Jetson Nano for faster and more efficient processing. Additionally, incorporating AI-based context understanding can allow the system to automatically interpret scenes, classify environments, and adapt its detection strategy, making it more intelligent. Finally, optimizing the system for deployment on mobile devices, drones, and IoT systems can expand the reach of object detection into fields such as smart surveillance, robotics, and autonomous navigation.

In the future, the Object Detection Using OpenCV system can be improved to deliver even more accurate and advanced detection capabilities. Deep-learning–based models like YOLO, SSD, and Faster R-CNN can be integrated to enhance detection performance in crowded environments and high-motion scenarios. Support for multi-class detection, instance segmentation, and object tracking can enable the system to perform complex tasks such as counting, classification, and behavioral analysis. The system can also be expanded to work with augmented reality and smart-camera platforms, allowing objects to be recognized and interacted with in real-time environments. Incorporating predictive algorithms can help the system identify object motion patterns and anticipate future positions. Furthermore, optimizing the detection pipeline for low-power embedded devices can enable deployment in smart cities, wearable technology, robotics, autonomous vehicles, and other edge-AI applications, broadening the impact of object detection across multiple domains.

REFERENCES

1. Viola, P., & Jones, M. (2001). Rapid Object Detection Using a Boosted Cascade of Simple Features. IEEE Conference on Computer Vision and Pattern Recognition (CVPR).
2. Dalal, N., & Triggs, B. (2005). Histograms of Oriented Gradients for Human Detection. IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR).
3. Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2016). You Only Look Once: Unified, Real-Time Object Detection. IEEE CVPR.
4. Retrieved from: https://pjreddie.com/darknet/yolo/
5. OpenCV Documentation. (2023). Open Source Computer Vision Library.
6. Retrieved from: https://docs.opencv.org/
7. Girshick, R. (2015). Fast R-CNN. IEEE International Conference on Computer Vision (ICCV).
8. Liu, W., Anguelov, D., Erhan, D., Szegedy, C., & Berg, A. (2016). SSD: Single Shot MultiBox Detector. European Conference on Computer Vision (ECCV).
9. Babenko, B., Yang, M. H., & Belongie, S. (2011). Robust Object Tracking with Online Multiple Instance Learning. IEEE TPAMI.
10. Retrieved from: https://paperswithcode.com/
11. Ren, S., He, K., Girshick, R., & Sun, J. (2015). Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. Advances in Neural Information Processing Systems (NIPS).
12. Everingham, M., Eslami, S. M. A., Van Gool, L., Williams, C. K., & Zisserman, A. (2015). The PASCAL Visual Object Classes Challenge – A Retrospective. International Journal of Computer Vision (IJCV).
13. Lin, T.-Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., Dollár, P., & Zitnick, C. L. (2014). Microsoft COCO: Common Objects in Context. European Conference on Computer Vision (ECCV).