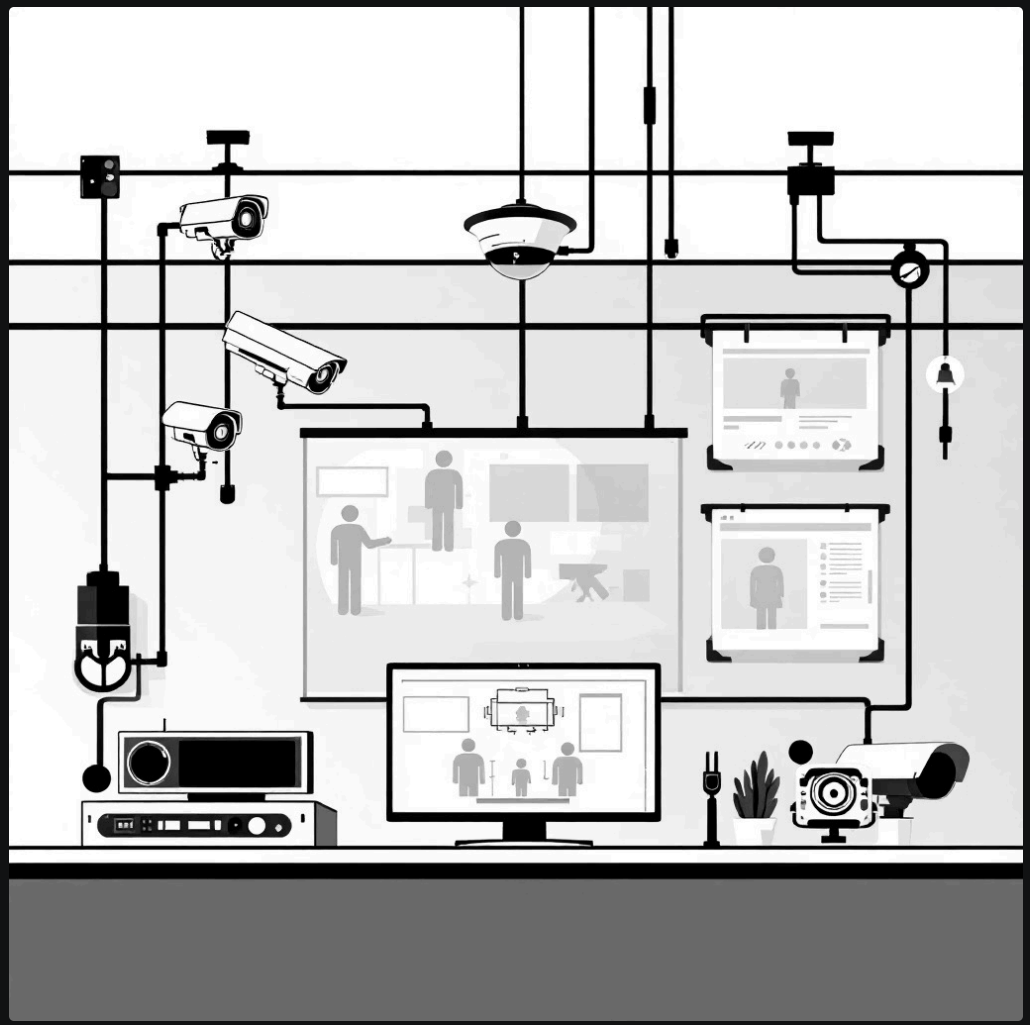
COMPUTER VISION

Moving Object Detection Using OpenCV

Real-time surveillance and monitoring powered by Python

The Challenge



Detecting Motion in Real Time

Computer vision enables automated surveillance, security, traffic monitoring, and activity recognition

Goal: Build a system that captures live video, processes frames, and highlights moving objects instantly

Project Objectives

Live Capture

Stream video from webcam in real time

Motion Detection

Identify differences between consecutive frames

Noise Reduction

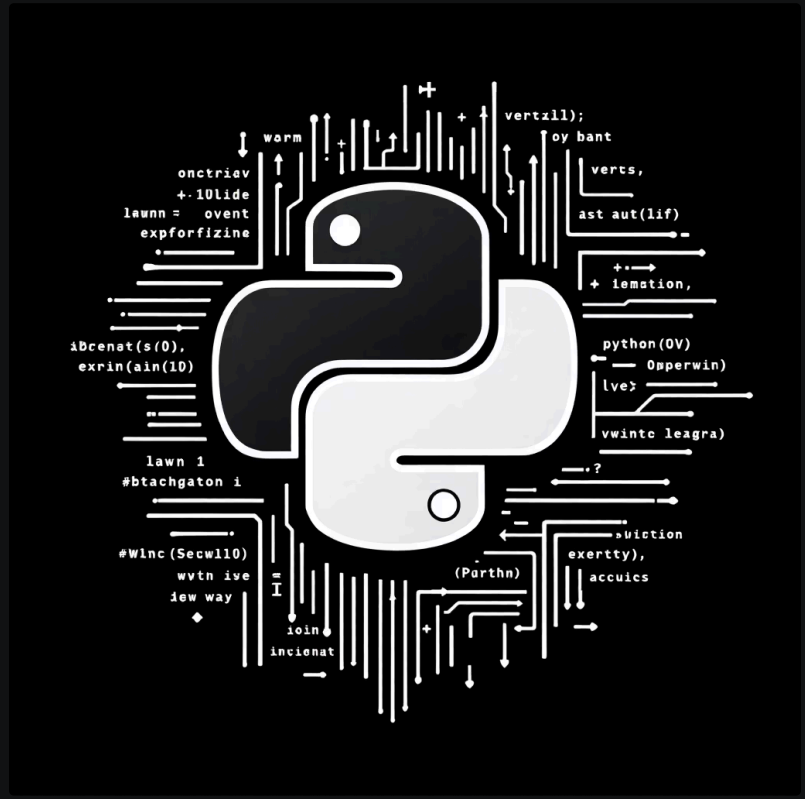
Eliminate background interference and artifacts

Visual Markers

Draw boxes around detected moving objects

Technical Stack

Technology	Purpose
Python 3.x	Core programming language
OpenCV	Image processing & computer vision
imutils	Resizing & contour handling
Webcam	Live video input stream



System Requirements

- Minimum 4GB RAM
- Any standard CPU
- Compatible: Windows, Linux, macOS

How Motion Detection Works

01

Video Capture

Capture live frames from webcam

02

Grayscale Conversion

Simplify image data for processing

03

Gaussian Blur

Reduce noise and smooth images

04

Background Reference

Store first frame as baseline

05

Frame Differencing

Compare current frame to background

06

Binary Threshold

Convert differences to black and white

07

Dilation

Fill gaps and connect regions

08

Contour Detection

Identify object boundaries

09

Filter & Display

Remove noise, draw bounding boxes

Core Algorithm Flow

Initialization

Start camera, set variables

Preprocessing

Resize, grayscale, blur frames

Difference Analysis

Compute frame differences

Object Identification

Find contours, draw boxes

Key Code Components

1

Library Import

```
import cv2
import imutils
import time
```

Load OpenCV and helper libraries

2

Camera Setu

```
cam = cv2.VideoCapture(0)
time.sleep(1)
```

Initialize webcam with stabilization delay

3

Initialize Variables

```
firstFrame = None
area = 500
```

firstFrame: Background Reference

area: Minimum contour area to ignore noise

4

Capture Frames Continuously

```
while True:
```

Infinite loop for real-time detection

5

Read Frame

```
_, img = cam.read()
text = "Normal"
```

Captures frame

Default status message

6

Resize Frame

```
img = imutils.resize(img, width=500)
```

Improves performance and consistency

7

Convert to Grayscale

```
grayimg = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
```

Convert to grayscale to simplify motion detection

8

Apply Gaussian Blur

```
gaussianimg = cv2.GaussianBlur(grayimg, (21,21), 0)
```

Removes noise and smoothens image

9

Store Background Frame

```
if firstFrame is None:
    firstFrame = gaussianimg
    continue
```

Sets first frame as background

10

Motion Detection

```
imgDiff = cv2.absdiff(firstFrame, gaussianimg)
threshimg = cv2.threshold(imgDiff, 25, 255, cv2.THRESH_BINARY)[1]
```

Calculate differences and threshold

11

Dilation

```
threshimg = cv2.dilate(threshimg, None, iterations=3)
```

Fills holes

Connects broken regions

12

Contour Analysis

```
cnts = cv2.findContours(threshimg.copy(), cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
cnts = imutils.grab_contours(cnts)
```

Detects moving objects

13

Filter Noise & Draw Bounding Box

```
for c in cnts:
    if cv2.contourArea(c) < area:
        continue
    (x,y,w,h) = cv2.boundingRect(c)
    cv2.rectangle(img,(x,y),(x+w,y+h),(0,255,0),2)
    text = "Moving Object detected"
```

Ignores small movements

Draws green box

14

Display Status

```
cv2.putText(img, text, (10,20),
            cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0,0,255), 2)
```

Displays motion status

15

Show Output

```
cv2.imshow("Camerafeed", img)
```

16

Exit Condition

```
if key == ord("q"):
    break
```

Real-World Applications

CCTV Surveillance

Automated monitoring for security systems

Traffic Monitoring

Vehicle detection and flow analysis

Smart Homes

Activity detection and automation triggers

Human Activity Recognition

Behavioral analysis and safety monitoring

Current Limitations & Future Vision

Challenges

Static Background Required

System assumes fixed camera position

Lighting Sensitivity

Changes in illumination affect accuracy

No Object Classification

Cannot identify what is moving

Enhancements

Dynamic Background

Adaptive background subtraction models

Object Tracking

Kalman filters for trajectory prediction

Deep Learning Integration

YOLO for object classification and detection

Project Outcomes

100%

Real-Time Detection

Instant motion identification and highlighting

500...

Minimum Contour Area

Effective noise filtering threshold

4+

Key Applications

Surveillance, security, traffic, smart homes

Conclusion

Successfully demonstrated a **real-time moving object detection system** using OpenCV and Python. The system efficiently processes video frames, detects motion through frame differencing, and highlights moving objects—making it ideal for surveillance and monitoring applications.