**MOVING OBJECT DETECTION USING OPENCV**

**ABSTRACT:**

Moving object detection is a fundamental task in computer vision applications, enabling various real-world applications such as surveillance, video analytics, and autonomous vehicles. It aims to develop a moving object detection system using OpenCV, an open-source computer vision library.

Moving object detection is the technique of detecting of occurrence of any movement in front of the camera when a person makes any movement. It detects and counts the movements from the web camera using OpenCV and a bounding box will be rendered to the movement detected as if any new object will introduce in the frame. Then a Box will appear surrounding the object.

Using a tracker it will count the movements of an object/person. For moving object detection, we calculate the difference between two continuous frames and if it’s higher than the set threshold, it means movement has been observed there.

**Libraries used :** cv2 , imutils , time

**OBJECTIVE:**

The main objective of object detection is to recognize and locate all moving objects within the frame.

**CONTENTS :**

**Object Detection :**

Object Detection is the process of finding and recognizing real-world object instances such as car, bike, TV, flowers, and humans out of an images or videos. An object detection technique lets you understand the details of an image or a video as it allows for the recognition, localization, and detection of multiple objects within an image.

It goes beyond simple classification by not only determining the presence of objects but also precisely locating their bounding boxes in the visual input.

Object detection algorithms employ various techniques such as deep learning, feature extraction, and machine learning to analyze the input data and recognize different object categories. The output of object detection typically includes the class labels of the detected objects along with their corresponding bounding box .

**Image:**

An image is a visual representation of an object, scene, or concept. It consists of a two-dimensional array of pixels that captures visual information. Each pixel in an image contains numerical values representing the color or intensity of the corresponding point in the image.Images can be

* **Grayscale**, consisting of shades of gray
* **Coloured**, where each pixel is represented by a combination of red, green, and blue (RGB) values.

**Resize Image :**

Resizing an image refers to the process of changing its dimensions, either by reducing or enlarging its size.

**Code:**

**1)**import cv2

**# Load the image**

img = cv2.imread("cat.jpg")

**# Define the new dimensions for the resized image**

new\_width = 80

new\_height = 60

**# Resize the image**

resize\_img = cv2.resize(img, (new\_width, new\_height))

**# Display the Original image**

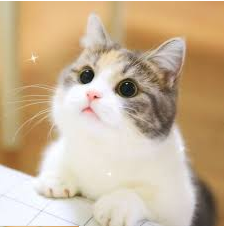
cv2.imshow("Original Image",img)

**# Display the resized image**

cv2.imshow("Resized Image",resize\_img)

cv2.waitKey(0)

cv2.destroyAllWindows()

**Original Image Resized Image **

**Explanation of Code (1) :**

**cv2.imread("cat.jpg")** - Loads the image from the specified file

**new\_width** - Specifies the desired width of the resized image in pixels.

**new\_height** - Specifies the desired height of the resized image in pixels.

**cv2.resize(img, (new\_width, new\_height))** - Resizes the image to the specified dimensions using the width and height parameters.

**cv2.imshow("Original Image",img)** - Displays the image in a window titled "Original Image".

**cv2.imshow("Resized Image",resize\_img)** - Displays the resized image in a window titled "Resized Image".

**cv2.waitKey(0)** - Waits for a key to be pressed before proceeding. The argument ‘0’ indicates that it waits indefinitely.

**cv2.destroyAllWindows()** - Closes all open windows after a key is pressed.

**Apply Gaussian Blur to an image :**

**Gaussian Blur:**

Gaussian blur is a common image filtering technique used to reduce image noise and smooth out pixel irregularities. It is based on the mathematical concept of a Gaussian distribution, which is a bell-shaped curve.

**Code:**

2)import cv2

**# Load the image**

img = cv2.imread("cat.jpg")

**# Apply Gaussian blur with a specified kernel size and standard deviation**

ksize = (5, 5)

sigma = 0

blurred\_img = cv2.GaussianBlur(img, ksize, sigma)

**# Display the original and blurred images**

cv2.imshow("Original Image", img)

cv2.imshow("Blurred Image", blurred\_img)

cv2.waitKey(0)

cv2.destroyAllWindows()

**Original Image Blurred Image**

** **

**Explanation of Code (2) :**

**cv2.imread("cat.jpg")** - Loads the image from the specified file

**ksize** - Represents the kernel size of the Gaussian blur filter. The size of the kernel used for blurring. It should be a tuple (width, height). The width and height values should be positive and odd numbers.

**sigma** - Determines the amount of blur applied.

**cv2.GaussianBlur(img, ksize, sigma)** - The 0 in cv2.GaussianBlur() represents the standard deviation of the kernel in the X and Y directions. If you want the standard deviation to be automatically calculated based on the kernel size

**cv2.imshow("Original Image",img)** - Displays the image in a window titled "Original Image".

**cv2.imshow("Blurred Image",blurred\_img)** - Displays the blurred image in a window titled "Blurred Image".

**cv2.waitKey(0)** - Waits for a key to be pressed before proceeding. The argument ‘0’ indicates that it waits indefinitely.

**cv2.destroyAllWindows()** - Closes all open windows after a key is pressed.

**Apply Thresholding to an image :**

**Thresholding:**

Thresholding is a technique used to separate regions of an image based on pixel intensity values. The basic idea behind thresholding is to convert a grayscale image into a binary image, where pixels that exceed the threshold value are assigned one intensity value (foreground) and pixels below the threshold are assigned another intensity value (background).

**Code:**

3)import cv2

**# Load the image**

img = cv2.imread("cat.jpg")

**#Convert to gray Image**

gray\_img = cv2.cvtColor(img,cv2.COLOR\_BGR2GRAY)

**# Apply thresholding with a specified threshold value**

threshold\_value = 127

max\_value = 255

\_, threshold\_img = cv2.threshold(grayImg, threshold\_value, max\_value, cv2.THRESH\_BINARY)

**# Display the original,gray and threshold images**

cv2.imshow("Original Image", img)

cv2.imshow("Gray Image",gray\_img)

cv2.imshow("Thresholded Image", threshold\_img)

cv2.waitKey(0)

cv2.destroyAllWindows()

**Original Image Gray Image Threshold Image**

**Explanation of Code (3) :**

**cv2.imread("cat.jpg")** - Loads the image from the specified file

**cv2.cvtColor(img,cv2.COLOR\_BGR2GRAY)** - Takes the input image and applies the specified color space conversion to produce the grayscale image. Thresholding is commonly applied to grayscale images

**threshold\_value** - To classify pixels as either foreground or background.

**max\_value** - Represents the maximum value assigned to pixels that exceed the threshold, typically set to 255 for binary thresholding

**cv2.imshow("Original Image",img)** - Displays the image in a window titled "Original Image".

**cv2.imshow("Gray Image",gray\_img)** - Displays the gray image in a window titled "Gray Image".

**cv2.imshow("Thresholded Image", threshold\_img)** - Displays the threshold image in a window titled "Gray Image".

**cv2.waitKey(0)** - Waits for a key to be pressed before proceeding. The argument ‘0’ indicates that it waits indefinitely.

**cv2.destroyAllWindows()** - Closes all open windows after a key is pressed.

**Drawing Rectangle to an image :**

Drawing a rectangle on an image refers to the process of adding a rectangular shape onto an image to highlight a specific region or define a bounding box around an object of interest.

**Code:**

4)import cv2

**# Load the image**

img = cv2.imread("cat.jpg")

**# Define the rectangle parameters**

x = 40

y = 20

w = 150

h = 200

**# Draw the rectangle on the image**

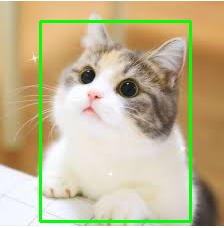
cv2.rectangle(img, (x, y), (x + w, y + h), (0, 255, 0), 2)

**# Display the image with the rectangle**

cv2\_imshow(img)

cv2.waitKey(0)

cv2.destroyAllWindows()



**Explanation of Code (4) :**

**cv2.imread("cat.jpg")** - Loads the image from the specified file

**x , y** - Coordinates of the top-left corner of the rectangle.

**Width,height** - Dimensions of the rectangle.

**(0, 255, 0)** - Corresponds to green

**‘2’** - Thickness of the rectangle's border.

**cv2.imshow(img)** - Displays the image in a window

**cv2.waitKey(0)** - Waits for a key to be pressed before proceeding. The argument ‘0’ indicates that it waits indefinitely.

**cv2.destroyAllWindows()** - Closes all open windows after a key is pressed.

**Putting Text in image:**

Putting text on an image refers to the process of adding textual content, such as labels, captions, or annotations, to an image.

**Code:**

5)import cv2

**# Load the image**

img = cv2.imread("cat.jpg")

**# Define the text string**

text = "cat"

**# Specify the font properties**

font = cv2.FONT\_HERSHEY\_SIMPLEX

font\_scale = 1.2

font\_color = (0, 255, 0)

thickness = 2

**# Determine the position of the text**

x = 100

y = 50

**# Put the text on the image**

cv2.putText(img, text, (x, y), font, font\_scale, font\_color, thickness)

# Display the image with the text

cv2\_imshow(img)

cv2.waitKey(0)

cv2.destroyAllWindows()



**Explanation of Code (5) :**

**cv2.imread("cat.jpg")** - Loads the image from the specified file

**text** - String that you want to put on the image.

**cv2.FONT\_HERSHEY\_SIMPLEX** - You can specify the font properties

**font\_scale** - Change the size of the text

**font\_color** - Color of the text in BGR format

**thickness** - Determines the thickness of the text stroke

**x , y** - Position of the starting point and baseline of the text

**cv2.imshow(img)** - Displays the image in a window

**cv2.waitKey(0)** - Waits for a key to be pressed before proceeding. The argument ‘0’ indicates that it waits indefinitely.

**cv2.destroyAllWindows()** - Closes all open windows after a key is pressed.

**Load Frames from Video:**

Videos are made of frames. A frame refers to a single still image or a complete picture that is displayed sequentially at a specific rate to create the illusion of motion.

**Code:**

6) **#Define a function to resize**

def resize(img):

return cv2.resize(img,(512,512))

**#Load Frames**

import cv2

**# Open the video file**

vs = cv2.VideoCapture('video.mp4')

while True:

**# Read the next frame**

ret, frame = vs.read()

**# Display the frame**

cv2.imshow(resize(frame))

**# Press 'q' to exit**

if cv2.waitKey(1) & 0xFF == ord('q'):

break

**# Release the video file and close all windows**

vs.release()

cv2.destroyAllWindows()







**WORKFLOW :**

Resize

Preprocess

Frames

Capture

Video

Frames

Denoise

Gray Scale

Thresholding

Loop through each frame

Initialise

Background

Model

Background

Subtraction

Visualize

Object

Tracking

Object

Filtering

Contour

Detection

**CODE IMPLEMENTATION:**

**1)**

# Import necessary Libraries

from google.colab.patches import cv2\_imshow

import imutils

import time

import cv2

**Explanation:**

Import the **cv2\_imshow** from **google.colab.patches** - Displaying images

**imutils** - Image processing

**time** - Pausing the program

**cv2** - OpenCV library.

**2)**

# Open the video file

vs = cv2.VideoCapture('video.mp4')

time.sleep(1)

**Explanation:**

**cv2.VideoCapture** - Open the video file 'video.mp4'

**time.sleep(1)** - Pause the program for 1 second

**3)**

#Initialize variables

count = 0

firstFrame = None

area = 500

**Explanation:**

**count** - Counting objects

**firstFrame** – Storing the first frame

**area** – Setting minimum contour area

**4)**

#Loop through each frame

while True:

ret, frame = vs.read()

text = "Normal"

img = imutils.resize(frame)

grayImg = cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY)

gaussianImg = cv2.GaussianBlur(grayImg, (21, 21), 0)

**Explanation:**

**vs.read()** - Reads frames from the video

**imutils.resize** – Resizing the frame

**cv2.cvtColor** – Converting frame to GrayScale

**cv2.GaussianBlur** – Applying gaussian blur

**5)**

#Check condition

if firstFrame is None:

firstFrame = gaussianImg

continue

**Explanation:**

**firstFrame is None** - Sets firstFrame to the current frame

**continue** – Continues to the next iteration of the loop

**6)**

#Calculate difference,threshold,dilation

imgDiff = cv2.absdiff(firstFrame, gaussianImg)

threshImg = cv2.threshold(imgDiff, 25, 255, cv2.THRESH\_BINARY)[1]

threshImg = cv2.dilate(threshImg, None, iterations=2)

**Explanation:**

**cv2.absdiff** - Calculate the absolute difference between firstFrame and the current frame

**cv2.threshold** – To create a binary image

**cv2.dilate** – Enhance Object Boundaries

**7)**

#Identifying Contours

cnts = cv2.findContours(threshImg.copy(), cv2.RETR\_EXTERNAL,

cv2.CHAIN\_APPROX\_SIMPLE)

cnts = imutils.grab\_contours(cnts)

**Explanation:**

**cv2.findContours** - Find contours in thresholded image

**imutils.grab\_contours** – Contours are extracted.

**8)**

#Loop iterates over each contour

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)

count += 1

text = "Moving Object detected"

**Explanation:**

**cv2.rectangle** - Draws a rectangle around the contour

**9)**

#Add text to image

cv2.putText(img, text, (10, 20),

cv2.FONT\_HERSHEY\_SIMPLEX, 0.5, (0, 0, 255), 2)

cv2\_imshow(img)

key = cv2.waitKey(1) & 0xFF

if key == ord("q"):

break

**Explanation:**

**cv2.putText** – Add text to image

**cv2.FONT\_HERSHEY\_SIMPLEX** - You can specify the font properties

**font\_scale** - Change the size of the text

**font\_color** - Color of the text in BGR format

**thickness** - Determines the thickness of the text stroke

**cv2.waitKey (1)** - Block waits for a key press for 1 millisecond

**pressed key is 'q'**- Loop is broken, and the program exits.

**10)**

vs.release()

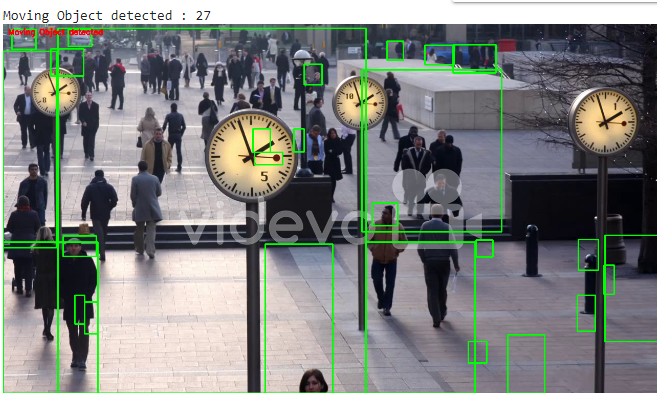
cv2.destroyAllWindows()

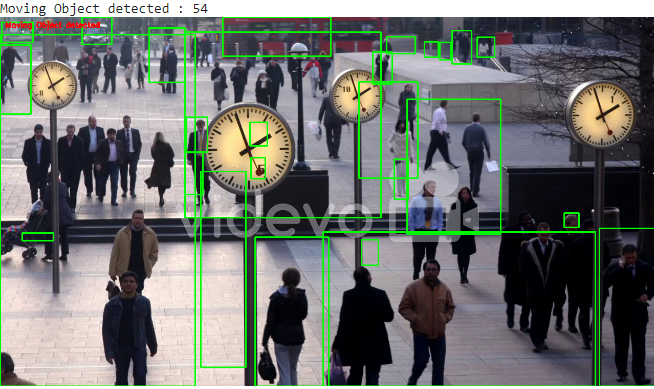
**Explanation:**

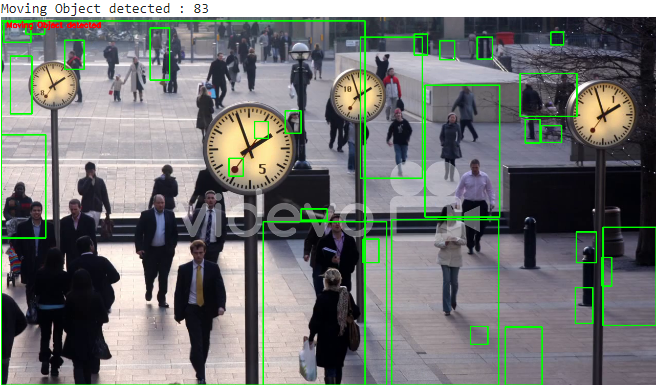
**vs.release()** – Release the video

**cv2.destroyAllWindows()**- Close all windows

**RESULTS :**

****











**CONCLUSION:**

Finally,we implemented moving object detection using OpenCV. We utilized video processing techniques to detect and track moving objects in a video stream.

This project demonstrated how OpenCV can be used for real-time moving object detection in video streams. It can be applied in various applications, such as surveillance systems, motion tracking, and video analysis. By understanding the concepts and techniques used in this project, further improvements and customizations can be made to suit specific requirements and scenarios.