**Assignment 6**

Develop a python code to detect any object using Haar cascade classifier.

**PYTHON CODE:-**

# import the necessary packages

from collections import deque

from imutils.video import VideoStream

import numpy as np

import argparse

import cv2

import imutils

import time

# construct the argument parse and parse the arguments

ap = argparse.ArgumentParser()

ap.add\_argument("-v", "--video",

help="path to the (optional) video file")

ap.add\_argument("-b", "--buffer", type=int, default=64,

help="max buffer size")

args = vars(ap.parse\_args())

# define the lower and upper boundaries of the "green"

# ball in the HSV color space, then initialize the

# list of tracked points

greenLower = (29, 86, 6)

greenUpper = (64, 255, 255)

pts = deque(maxlen=args["buffer"])

# if a video path was not supplied, grab the reference

# to the webcam

if not args.get("video", False):

vs = VideoStream(src=0).start()

# otherwise, grab a reference to the video file

else:

vs = cv2.VideoCapture(args["video"])

# allow the camera or video file to warm up

time.sleep(2.0)

# keep looping

while True:

# grab the current frame

frame = vs.read()

# handle the frame from VideoCapture or VideoStream

frame = frame[1] if args.get("video", False) else frame

# if we are viewing a video and we did not grab a frame,

# then we have reached the end of the video

if frame is None:

break

# resize the frame, blur it, and convert it to the HSV

# color space

frame = imutils.resize(frame, width=600)

blurred = cv2.GaussianBlur(frame, (11, 11), 0)

hsv = cv2.cvtColor(blurred, cv2.COLOR\_BGR2HSV)

# construct a mask for the color "green", then perform

# a series of dilations and erosions to remove any small

# blobs left in the mask

mask = cv2.inRange(hsv, greenLower, greenUpper)

mask = cv2.erode(mask, None, iterations=2)

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

# find contours in the mask and initialize the current

# (x, y) center of the ball

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

cv2.CHAIN\_APPROX\_SIMPLE)

cnts = imutils.grab\_contours(cnts)

center = None

# only proceed if at least one contour was found

if len(cnts) > 0:

# find the largest contour in the mask, then use

# it to compute the minimum enclosing circle and

# centroid

c = max(cnts, key=cv2.contourArea)

((x, y), radius) = cv2.minEnclosingCircle(c)

M = cv2.moments(c)

center = (int(M["m10"] / M["m00"]), int(M["m01"] / M["m00"]))

# only proceed if the radius meets a minimum size

if radius > 10:

# draw the circle and centroid on the frame,

# then update the list of tracked points

cv2.circle(frame, (int(x), int(y)), int(radius),

(0, 255, 255), 2)

cv2.circle(frame, center, 5, (0, 0, 255), -1)

# update the points queue

pts.appendleft(center)

# loop over the set of tracked points

for i in range(1, len(pts)):

# if either of the tracked points are None, ignore

# them

if pts[i - 1] is None or pts[i] is None:

continue

# otherwise, compute the thickness of the line and

# draw the connecting lines

thickness = int(np.sqrt(args["buffer"] / float(i + 1)) \* 2.5)

cv2.line(frame, pts[i - 1], pts[i], (0, 0, 255), thickness)

# show the frame to our screen

cv2.imshow("Frame", frame)

key = cv2.waitKey(1) & 0xFF

# if the 'q' key is pressed, stop the loop

if key == ord("q"):

break

# if we are not using a video file, stop the camera video stream

if not args.get("video", False):

vs.stop()

# otherwise, release the camera

else:

vs.release()

# close all windows

cv2.destroyAllWindows()

**OUTPUT:-**



