A close up of a logo

Description generated with very high confidence

MACHINE VISION

MCT 4323

Quiz 2

AHID NAIF ARIF HUSSEIN ALQAUD 1410721

**Code**

|  |
| --- |
| import cv2  import numpy as np  # custom class that I have made using TrackBar in OpenCV  from customizedTrackBar import customizedTrackBar  """  When you first initialize an instance of customizedTrackBar class which I made,  you need to pass the following parameters:  (startValue, step, endValue, defaultValue, variableName, windowName)  Examples are: following param1Bar & param2Bar  """  param1Bar = customizedTrackBar(20, 1, 255, 45, "param1", "Tuning") # Initialize a bar to tune param1 value  param2Bar = customizedTrackBar(20, 1, 50, 35, "param2", "Tuning") # Initialize a bar to tune param2 value  image = cv2.imread("eye-pupil.jpg") # load an image  gray = cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY) # convert image into gray scale  blurred = cv2.GaussianBlur(gray, (51,51), 0) # apply filter on the image with a kernel size 51x51  while True: # keep looping for the user to be able to tune the parameters  imageClone = image.copy() # get a copy of the image  maxThresh = param1Bar.getValue() # get maxThresh value of trackBar to be used in HoughCircles()  param2 = param2Bar.getValue() # get param2 value of trackBar to be used in HoughCircles()  circles = cv2.HoughCircles(blurred, cv2.HOUGH\_GRADIENT, 1, 50, *param1*=maxThresh, *param2*=param2, *minRadius*=0, *maxRadius*=0)  if circles is not None:  # convert the (x, y) coordinates and radius of the circles to integers  circles = np.round(circles[0, :]).astype("int")  for x, y, r in circles:  # draw the circle in blue color in the output image, then draw a rectangle  cv2.circle(imageClone, (x, y), r, (255,0,0), 2)  # draw a very small circle in red color representing the circle center  cv2.circle(imageClone, (x, y), 1, (0,0,255), 2)  else:  print("There was no circles detected in this image")  cv2.imshow("Detecting Circles Using Hough Transform", imageClone)  key = cv2.waitKey(1) & *0x*FF  if key == ord("q"): # break the loop if the user presses on "q" button  break |

**Screenshot of the output:**

|  |
| --- |
|  |

**Flowchart:**

**More Details:**

**For cv2.HoughCircles() function, its parameters as following:**

* **image:** 8-bit, single channel image. If working with a color image, convert to grayscale first.
* **method:** Defines the method to detect circles in images. Currently, the only implemented method is cv2.HOUGH\_GRADIENT, which corresponds to the [Yuen et al.](http://www.bmva.org/bmvc/1989/avc-89-029.pdf) paper.
* **dp:** This parameter is the inverse ratio of the accumulator resolution to the image resolution (see Yuen et al. for more details). Essentially, the larger the dp gets, the smaller the accumulator array gets.
* **minDist:** Minimum distance between the center *(x, y)* coordinates of detected circles. If the minDist is too small, multiple circles in the same neighborhood as the original may be (falsely) detected. If the minDist is too large, then some circles may not be detected at all.
* **param1:** Gradient value used to handle edge detection in the Yuen et al. method.
* **param2:** Accumulator threshold value for the cv2.HOUGH\_GRADIENT method. The smaller the threshold is, the more circles will be detected (including false circles). The larger the threshold is, the more circles will potentially be returned.
* **minRadius:** Minimum size of the radius (in pixels).
* **maxRadius:** Maximum size of the radius (in pixels).

**Code of customizedTrackBar class**

|  |
| --- |
| import cv2  *class* customizedTrackBar:  *def* \_\_init\_\_(*self*, *desiredStart*, *step*, *desiredEnd*, *defaultValue*, *description*, *windowName*):  self.description = description  self.windowName = windowName  self.defaultValue = defaultValue  self.desiredStart = desiredStart  self.step = step  self.desiredEnd = desiredEnd  self.start = 0  self.end = (self.desiredEnd - self.desiredStart)\*(1/self.step)  cv2.namedWindow(windowName, cv2.WINDOW\_FREERATIO)  self.createCutomizedTrackBar()  self.setDefaultPosition()  *def* createCutomizedTrackBar(*self*):  cv2.createTrackbar(self.description, self.windowName, self.start, *int*(self.end), self.doNothing)  *def* setDefaultPosition(*self*):  ratio = (self.defaultValue - self.desiredStart)/(self.desiredEnd - self.desiredStart)  defaultPosition = ratio \* (self.end - self.start) + self.start  cv2.setTrackbarPos(self.description, self.windowName, *int*(defaultPosition))  *def* getValue(*self*):  trackBarValue = cv2.getTrackbarPos(self.description, self.windowName)  ratio = (trackBarValue - self.start)/(self.end - self.start)  value = ratio \*(self.desiredEnd - self.desiredStart) + self.desiredStart  return *int*(value)    *def* doNothing(*self*,*\_*):  pass |