Ananya Srinivasan
CSC 355: Human Computer Interaction
Homework 4: Computer Vision
7 October, 2018

Computer Vision

Object Detection: The object detection program utilizes the following OpenCV functions to read, convert, transform, and extract features within a given image:

- `img = cv2.imread('./coffeecup.jpg', 0):` The imread function takes in an image file, and a flag to indicate the image color. In this case, 0 represents grayscale - thus, img is a grayscale version of the original image.

Figure 1: Original image (from AliExpress)

- `img_blur = cv2.GaussianBlur(img, (5,5), 0):` Gaussian Blur was used before thresholding to remove most noise within the image. img is the image file, the (5,5) tuple represents the kernel, and 0 represents the border type of the image.
- `ret, otsu_thresh = cv2.threshold(img_blur,0,255,cv2.THRESH_BINARY+cv2.THRESH_OTSU):` Otsu Thresholding was used to segment the image, and reduce the grayscale image to a binary level. The threshold takes an image (img_blur), threshold value (0), maximum threshold value (255) for binary thresholding, and a threshold type (cv2.THRESH_BINARY+cv2.THRESH_OTSU)

Figure 2: Image after Gaussian Blur and Otsu Thresholding
Note: This screenshot cut off some cups at the bottom

- `cv2.imshow('Otsu', otsu_thresh):` Displays the image (otsu_thresh) under the name ‘Otsu’ in the window. Refer to Figure 1 and 2.
- `kernel = np.ones((5,5), np.uint8):` The np.ones function was used for the following step to create a kernel that represents the image as an array of 1’s. This array

is a 5x5 array based on the tuple (5,5) and corresponds to the unsigned integers between 0 and 255.

- `img_erode = cv2.erode(otsu_thresh, kernel, iterations=25):` After thresholding, the erosion function is used to separate potentially overlapping objects within the image by eroding or reducing the white pixels. The function takes the image (otsu_thresh), the kernel, and the number of iterations the image will be eroded. The kernel is particularly used to convolve the image, and the ones of the kernel are changed to zeros if the ones in kernel are not under the image.

Figure 3: Erosion

- `img_dilate = cv2.dilate(img_erode, kernel, iterations=15):` After eroding away the objects, the dilation function is used to expand the objects for analysis. The function takes the image (img_erode), kernel array for the image convolution, and iterations for dilating the image (15). If there are ones in the kernel that correspond to the image, then the adjacent areas are also set to one.

Figure 4: Dilation

- `cc = cv2.connectedComponentsWithStats(img_dilate, 8, cv2.CV_32S):` After erosion and dilation, each object in the image can be extracted and detected. The function takes the image (img_dilate), the connectivity (8-way; connected horizontally, vertically, and diagonally), and image label type (cv2.CV_32S).
    - `labels = cc[1]:` Returns a list of labels, where each label corresponds to an object
    - `stats = cc[2]:` Returns a list of statistics for each connected component
    - `x = stats[label, cv2.CC_STAT_LEFT]:` Returns the x-coordinate of an object with a corresponding label
    - `width = stats[label, cv2.CC_STAT_WIDTH], height = stats[label, cv2.CC_STAT_HEIGHT]:` Returns the width and height of the object with a corresponding label
    - `area = stats[label, cv2.CC_STAT_AREA]:` Returns the area of the object given the label of that particular object

Figure 5: Connected Components

- `img_rect = cv2.rectangle(img,(x, y),(x+width, y+height),(250, 0, 0), 3):` Plots rectangles for the given image (img), top-left corner (x, y), bottom-right corner (x+width, y+height), color (RGB (250, 0, 0)), and line thickness (3)

Object Detection Results:

Number of items: 9
Object 1:
Coordinates: (0, 0)
Width: 990
Height: 965
Area: 868634

Object 2:
Coordinates: (394, 307)
Width: 79
Height: 141
Area: 10399

Object 3:
Coordinates: (569, 325)
Width: 75
Height: 132
Area: 9557

Object 4:
Coordinates: (725, 331)
Width: 83
Height: 117
Area: 9200

Object 5:
Coordinates: (117, 441)
Width: 92
Height: 110
Area: 9541

Object 6:
Coordinates: (417, 508)
Width: 81
Height: 118
Area: 9075

Object 7:
Coordinates: (614, 524)
Width: 101
Height: 153
Area: 14279

Object 8:
Coordinates: (769, 636)
Width: 87
Height: 140
Area: 11251

Object 9:
Coordinates: (498, 668)
Width: 104
Height: 142
Area: 13414

The algorithm detected 9 objects out of 10. Object 1, with top-right coordinates of 0,0 and an area of 868634, represents the collection of the cups rather than one specific cup. The algorithm, due to the number of erosion iterations, does not detect the bottom-most left cup that was cut off of the image. When the image was dilated using 15 iterations, the cup is non-existent (refer to Figure 4). The number of erosion iterations was 25, in order to ensure no cups were intersecting. The number of dilation iterations were 15 to also prevent potential overlapping of cups. As a result, the area of the cups and their bounding boxes (refer to Figure 5) were smaller than expected. The largest cup area, in fact, was Object 7's 14279.

Head Movement:

The algorithm utilizes Haar Cascading for face and eyes. The detectMultiScale() method extracts the coordinates, width, and height using the face classifier for the video frame. A similar detectMultiScale() method is also used for the eye classifier within a for-loop to extract the values for each eye in the image. The issue with this eye classifier, is that it will recognize other “pupil”-like features of the image (like a nostril - refer to Figure 6).

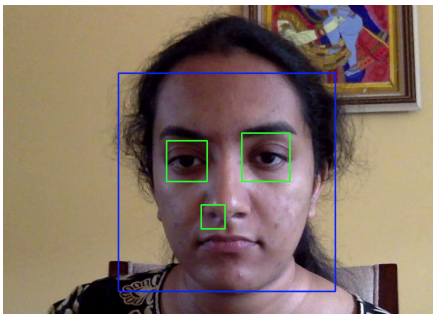

Figure 6: Rectangle plotting of eye shapes and face in a video frame

To determine whether the head moves “yes” or “no,” the x and y coordinates were compared for the frames. The `abs(history[counter+1][0] - history[counter][0]) > abs(history[counter+1][1] - history[counter][1])` statement is used to compare whether there is more horizontal (or x-axis movement) than vertical movement (or y-axis movement). The current x and y values are compared to the previous frame’s x and y values. If there is more horizontal movement, then a “NO” signal is printed out. If there is more vertical movement, then a “YES” signal is printed out.
Some improvements that could be made could be to compare eye movements in each frame in a similar manner, as the eyes can move in the same direction as the face.