CVE Assignment 6 Report

Question 1 (PS6-1)

Algorithm and Observations

The objective of this problem is to identify and highlight different shapes based on a given category list : spade terminal, ring terminal, washer, internal lock washer, external lock washer. Each of these categories have a characteristic contour signature in the hierarchy, which is used to identify each element.

Discriminatory charecteristics of each category are explained below:

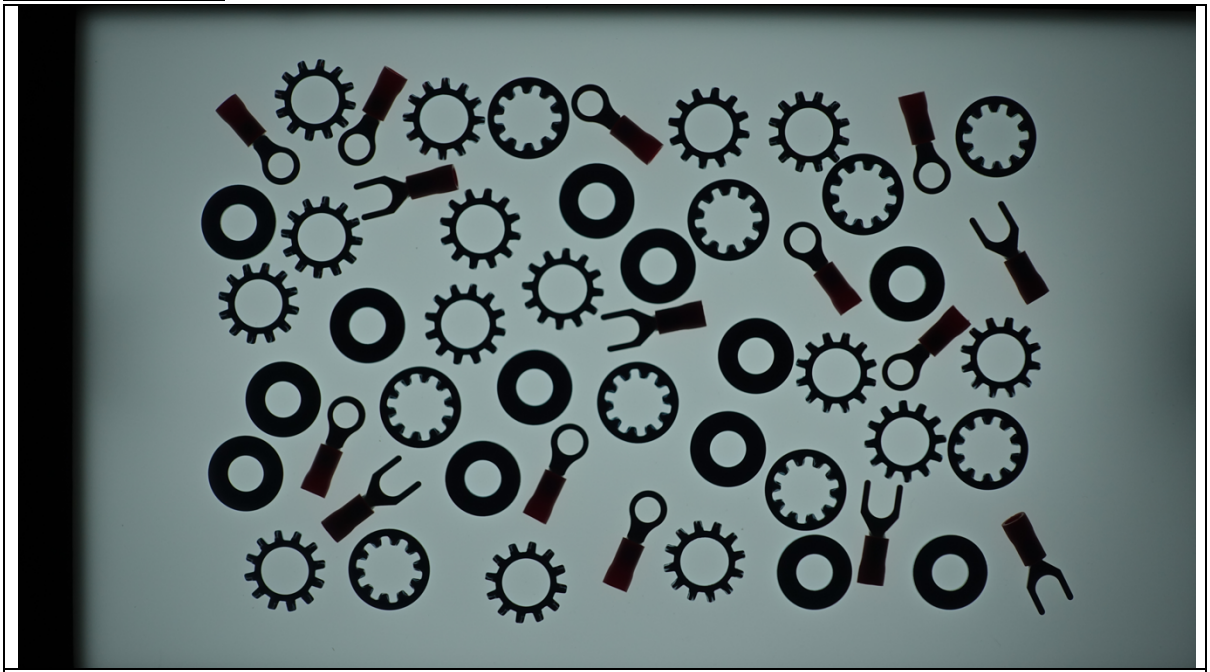
1. Spade terminal – Single contour makes up the entire object with no child contours.
2. Ring terminal – Double contour feature with one circular contour placed inside a non-circular contour. External contour is roughly rectangular in a counding box.
3. Washer – Double contour feature with both contours circular.
4. Internal lock Washer – Double contour feature with circular external contour and non-circular internal contour.
5. External lock washer – Double contour feature with circular internal contour and a non-circular external contour. External contour bounding box is roughly square in shape.

To solve this problem, two functions which measure if the shape of the contour's bounding box is square or circular were used. A conditional ladder was used to segregate the categories. The input image was thresholded at an intensity of 60 and binarized. Erode and dilate functionality was used to clean up the binarized image. The findContours() functions provided a list of all contours in the binarized image. The conditional ladder was iterated through each contour and based on the category, the contour was filled with the relevant category color.

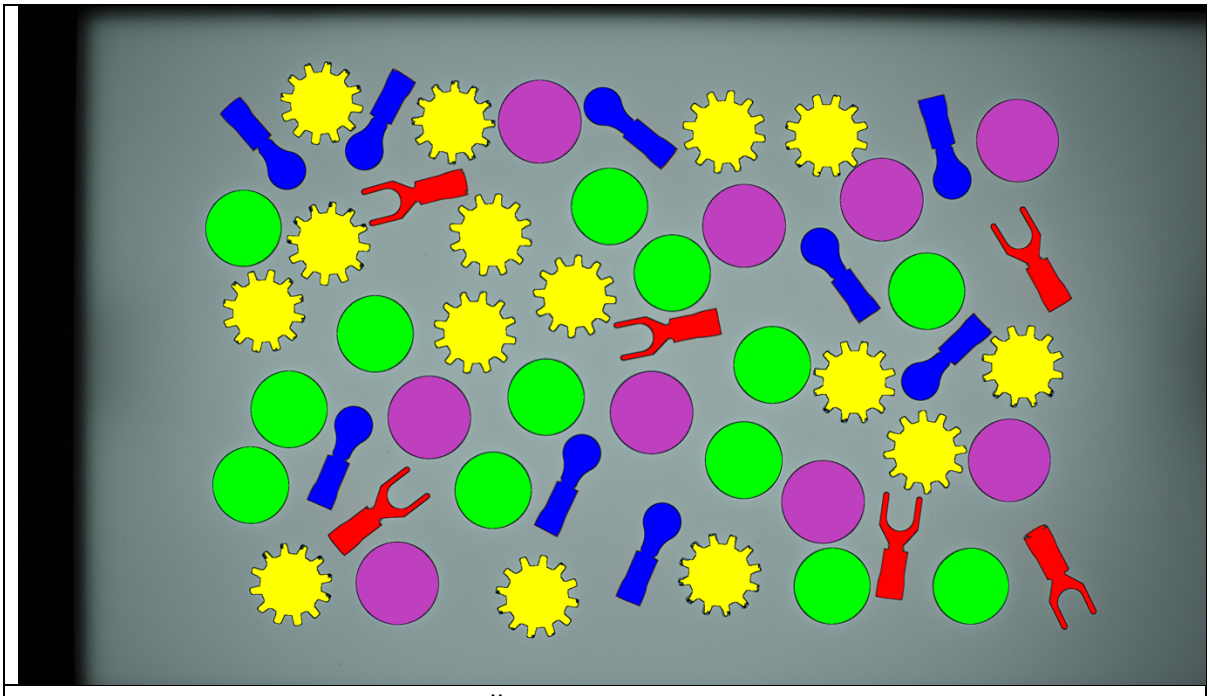
A problem encountered was the proximity of one pair of elements, which was causing a united contour. Dilate and erode functions were used in the order D-D-E-D-D-E to separate the binarized image for individual contours.

The results of the algorithm are displayed below.

Results of Wall 1



all-parts.png (Original)



all-parts-output.png

Source Code

```
# November 7th
import cv2
import numpy as np
import argparse

# check size (bounding box) is square
def isSquare(siz):
    ratio = abs(siz[0] - siz[1]) / siz[0]
    #print (siz, ratio)
    if ratio < 0.1:
        return True
    else:
        return False

# check circle from the arc length ratio
def isCircle(cnt):
    (x,y),radius = cv2.minEnclosingCircle(cnt)
    len = cv2.arcLength(cnt, True)
    ratio = abs(len - np.pi * 2.0 * radius) / (np.pi * 2.0 * radius)
    #print(ratio)
    if ratio < 0.1:
        return True
    else:
        return False

if __name__ == "__main__":
    #
    parser = argparse.ArgumentParser(description='Hough Circles')
    parser.add_argument('-i', '--input', default = 'image/all-
parts.png')

    args = parser.parse_args()
    # Read image
    img = cv2.imread(args.input)
    # Convert to gray-scale
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    # Binary
    thr,dst = cv2.threshold(gray, 60, 255, cv2.THRESH_BINARY)

    # clean up
    # for i in range(1):
    #     dst = cv2.erode(dst, None)
```

```
    # Dilate and erode combination to seperate parts very close to each
other
    dst = cv2.dilate(dst, None)
    dst = cv2.dilate(dst, None)
    dst = cv2.erode(dst, None)
    dst = cv2.dilate(dst, None)
    dst = cv2.dilate(dst, None)
    dst = cv2.erode(dst, None)

    # find contours with hierarchy
    cont, hier = cv2.findContours(dst, cv2.RETR_TREE,
cv2.CHAIN_APPROX_SIMPLE)
    # img = cv2.drawContours(img, cont, -1, (0,0,255), 2)

    # each contour
    for i in range(len(cont)):
        c = cont[i]
        h = hier[0,i]
        if h[2] == -1 and h[3] == 0: #For Spade terminal
            # no child and parent is image outer
            img = cv2.drawContours(img, cont, i, (0,0,255),-1)  #Draws
in red
        elif h[3] == 0 and hier[0,h[2]][2] == -1: # More Complex shapes
            # with child
            if isCircle(c): #  Washer and External lock washer
                if isCircle(cont[h[2]]): #Washer
                    # double circle
                    img = cv2.drawContours(img, cont, i, (0,255,0),-1)
#Draws in green
                #Add code for internal lock washer
                else:
                    img = cv2.drawContours(img, cont, i, (191,64,191),
-1)
            else:   # Ring Terminal and internal lock washer
                # 1 child and shape bounding box is not square
                if not isSquare(cv2.minAreaRect(c)[1]) and
hier[0,h[2]][0] == -1 and hier[0,h[2]][1] == -1: #Ring Terminal
                    img = cv2.drawContours(img, cont, i, (255,0, 0),-1)
#Draw in Blue
                # Add code for for external lock washer
                else:
                    img = cv2.drawContours(img, cont, i, (0,255,255), -
1)
```

```
cv2.namedWindow("Image",cv2.WINDOW_NORMAL)
cv2.imshow("Image", img)

cv2.imwrite("image/all-parts-output.png", img)
cv2.waitKey()
```

Question 2 (PS6-2)

Algorithm and Observations

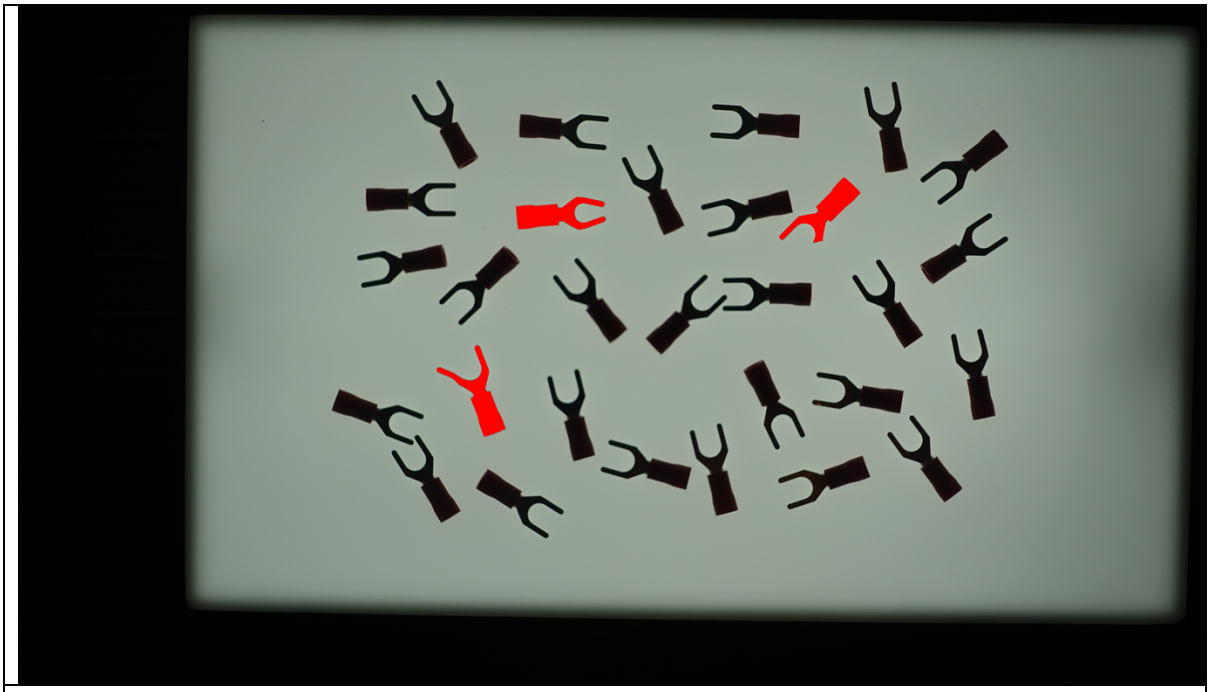
The objective of this problem is to identify defective spade terminals from a collection and highlight them. This involves matching the isolated terminals with a sample of the terminal and comparing their contours for a match. The Hu moments method is implemented using the matchShapes() function. The second distance, CONTOURS_MATCH_I2, is used for simplicity. The following algorithm was implemented to determine the defective terminals.

1. Get the relevant image from the memory and convert into grayscale.
2. Threshold the image at a value of 60 and binarise the image.
3. Apply a dilation-erosion routine following D-E-E-D to clean the image and remove any noise.
4. Feed the cleaned image for contour detection using findContours().
5. As the image includes a boundary, all spade terminals will the children of the outermost contours. Using the contour hierarchy, iterate through each of them and perform the following operations:
    a. Match each contour with all contours in the contour list. For match distance less than 1.4 (output of matchShapes() using CONTOURS_MATCH_I2), increase the match count for that object.
    b. For any contour with a match count less than a threshold, add their id into the defective contour list.
    c. The threshold for match is a function of the number of elements found inside the image. An uncertainity of 20% is applied to the calculate the threshold. For example, if an image contains 30 terminals, the match count for each teminal must be atleast 30 – 0.2x30 = 24 for it to qualify as a non-defective item. Any item with a matchcount less than this is a defect.
6. Highlight the terminals mentioned in the defective list with the color red.

As the template for a perfect terminal was not available, this method was a used. This method fails when the number of elements in the image is very low, and the algorithm is unable to identify the correct contour. A template may be used to overcome this issue. This algorithm may also fail when the dataset contains an excessive number of defective terminals, resulting in decision uncertainity. Again, this may be rectified by selecting the correct terminal contour from the images or providing a template for contour matching.

Results of Wall 1

spade-terminal.png (Original)



spade-terminal-output.png

Source Code

```
# November 7th
import cv2
import numpy as np
import argparse

if __name__ == "__main__":
    parser = argparse.ArgumentParser(description='Hough Circles')
    parser.add_argument('-i', '--input', default = 'image/spade-
terminal.png')

    args = parser.parse_args()
    # Read image
    img = cv2.imread(args.input)
    # Convert to gray-scale
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    # Binary
    thr,dst = cv2.threshold(gray, 60, 255, cv2.THRESH_BINARY)
    # D-E-E-D Routine for image cleaning
    dst = cv2.dilate(dst, None)
    dst = cv2.erode(dst, None)
    dst = cv2.erode(dst, None)
    dst = cv2.dilate(dst, None)

    # find contours with hierarchy
    cont, hier = cv2.findContours(dst, cv2.RETR_TREE,
cv2.CHAIN_APPROX_SIMPLE)
    defects_id = []
    elements = len(cont) # Number of elements in the image
    uncertaininty = int(0.2 * elements) # Uncertainininty of elemnts
being similar
    for id in range(len(cont)):
        contour = cont[id]
        h = hier[0,id]
        if h[2] == -1 and h[3] == 0:
            match = 0   # NUmber of matches for each element
            for i in range(len(cont)):
                compare = cont[i]
                if
cv2.matchShapes(contour,compare,cv2.CONTOURS_MATCH_I2,0) < 1.4:
                    match+=1
             # Elements are similar if and only if they match most of
the other elements
            if match <= elements - uncertaininty:
```

```
            defects_id.append(id)

print(defects_id)
for id in defects_id:
    img = cv2.drawContours(img, cont, id, (0,0,255), -1)

cv2.namedWindow("Image",cv2.WINDOW_NORMAL)
cv2.imshow("Image", img)

cv2.imwrite("image/spade-terminal-output.png", img)
cv2.waitKey()
```

System Specifications

Operating System: macOS Monterey Version 12.5.1
Hardware: MacBook Air 2017 (Intel Core i5)
Python: Conda environment utilizing Python 3.9.1
IDE: Visual Studio Code

Time taken
Q1: 1 hour
Q2: 1 hour