**19R611 AIVL INDIVIDUAL REPORT-1**

21R238

SHIBIL IRFAAN .I

**AIM:**

1. The aim of this project is to develop a computer vision-based solution for detecting pistons and estimating the centerline in images of a pump system. The system is equipped with a circular pump and multiple pistons, and the objective is to identify the pistons and determine a representative centerline for further analysis. (17-B)
2. To find the necessary coordinates of the image to draw a line connecting those two points (PSG Tech arches) and to add an given image on the original image. (17-A)

**SOFTWARES & MODULES USED:**

* Spyder(anaconda-python3.12)
* OpenCV(cv2)
* Math

**CODE:**

**17-A**

import cv2

# Global variables to store the coordinates of the two points

point1 = None

point2 = None

# Function to handle mouse events

def mouse\_callback(event, x, y, flags, param):

global point1, point2

if event == cv2.EVENT\_LBUTTONDOWN:

if point1 is None:

point1 = (x, y)

print(f"First point selected: {point1}")

elif point2 is None:

point2 = (x, y)

print(f"Second point selected: {point2}")

# Draw a line between the two points

cv2.line(image, point1, point2, (0, 0, 255), 2)

cv2.imshow('Image', image)

# Reset points for next line

point1 = None

point2 = None

# Load your image

image\_path = r"C:\Users\shibi\Desktop\AIVL\ir\a1.jpg"

image = cv2.imread(image\_path)

# Load your logo image with white background and black text

logo\_path = r"C:\Users\shibi\Desktop\AIVL\ir\a2.jpg"

logo = cv2.imread(logo\_path)

# Resize logo to fit on the image

logo\_height, logo\_width = logo.shape[:2]

resized\_logo = cv2.resize(logo, (int(logo\_width \* 0.5), int(logo\_height \* 0.5)))

# Get the height and width of the resized logo

logo\_height, logo\_width = resized\_logo.shape[:2]

# Define the region on the image where the logo will be placed (top-left corner)

roi = image[:logo\_height, :logo\_width]

# Add the logo to the image

image[:logo\_height, :logo\_width] = cv2.addWeighted(roi, 0, resized\_logo, 1, 1)

# Display the resulting image with the logo

# Create a window and set the mouse callback function

cv2.namedWindow('Image')

cv2.setMouseCallback('Image', mouse\_callback)

cv2.imshow('Image', image)

cv2.waitKey(0)

cv2.destroyAllWindows()

**CODE EXPLANATION: (17-A)**

The provided Python code leverages the OpenCV library to create an interactive image annotation tool. The primary functionalities include line drawing and logo placement. The code can be explained in the following steps:

**Image Loading:**

Load an image and a logo image using the OpenCV library.

image\_path = r"C:\Users\shibi\Desktop\AIVL\ir\a1.jpg"

logo\_path = r"C:\Users\shibi\Desktop\AIVL\ir\a2.jpg"

image = cv2.imread(image\_path)

logo = cv2.imread(logo\_path)

**Logo Resizing and Placement:**

Resize the logo to 50% of its original size and place it on the top-left corner of the image.

logo\_height, logo\_width = logo.shape[:2]

resized\_logo = cv2.resize(logo, (int(logo\_width \* 0.5), int(logo\_height \* 0.5)))

roi = image[:resized\_logo.shape[0], :resized\_logo.shape[1]]

image[:resized\_logo.shape[0], :resized\_logo.shape[1]] = cv2.addWeighted(roi, 0, resized\_logo, 1, 1)

**Line Drawing with Mouse Events:**

Implement a function to handle mouse events, allowing users to draw lines on the image by selecting two points.

def mouse\_callback(event, x, y, flags, param):

global point1, point2

if event == cv2.EVENT\_LBUTTONDOWN:

if point1 is None:

point1 = (x, y)

print(f"First point selected: {point1}")

elif point2 is None:

point2 = (x, y)

print(f"Second point selected: {point2}")

# Draw a line between the two points

cv2.line(image, point1, point2, (0, 0, 255), 2)

cv2.imshow('Image', image)

# Reset points for the next line

point1 = None

point2 = None

**Interactive Display:**

Create a window and set the mouse callback function to enable interactive drawing.

cv2.namedWindow('Image')

cv2.setMouseCallback('Image', mouse\_callback)

**Final Display:**

Display the image with the logo and enable interactive line drawing.

cv2.imshow('Image', image)

cv2.waitKey(0)

****cv2.destroyAllWindows()

**INPUT IMAGE: OUTPUT IMAGE:**

****

**CODE:  
17-B**

import cv2

import math

PUMP\_DIAMETER =240

PISTON\_DIAMETER =65

PISTON\_COUNT = 7

def plotCircles(img, circles, color):

if circles is None: return

for (x,y,r) in circles[0]:

cv2.circle(img, (int(x),int(y)), int(r), color, 2)

orig = cv2.imread(r"C:\Users\shibi\Desktop\AIVL\ir\b.jpg")

drawImg = orig

resized = cv2.resize(orig, None, fx=1.5, fy=1.5)

drawImg = resized

gray = cv2.cvtColor(resized, cv2.COLOR\_BGR2GRAY)

drawImg = cv2.cvtColor(gray, cv2.COLOR\_GRAY2BGR)

threshVal = 180

ret,thresh = cv2.threshold(gray, threshVal, 255, cv2.THRESH\_BINARY)

drawImg = cv2.cvtColor(thresh, cv2.COLOR\_GRAY2BGR)

pumpRadiusRange = ( int(PUMP\_DIAMETER/2)-2, int(PUMP\_DIAMETER/2)+2)

pumpCircles = cv2.HoughCircles(thresh, cv2.HOUGH\_GRADIENT, 1, PUMP\_DIAMETER, param2=5, minRadius=pumpRadiusRange[0], maxRadius=pumpRadiusRange[1])

plotCircles(drawImg, pumpCircles, (255,0,0))

pumpCircle = pumpCircles[0][0]

pistonArea = 3.14159 \* PISTON\_DIAMETER\*\*2 / 4

blobParams = cv2.SimpleBlobDetector\_Params()

blobParams.filterByArea = True

blobParams.minArea = 0.80 \* pistonArea

blobParams.maxArea = 1.20 \* pistonArea

blobDetector = cv2.SimpleBlobDetector\_create(blobParams)

blobs = blobDetector.detect(thresh)

drawImg = cv2.drawKeypoints(drawImg, blobs, (), (255,0,0), cv2.DRAW\_MATCHES\_FLAGS\_DRAW\_RICH\_KEYPOINTS)

pistonCenters = [(int(b.pt[0]),int(b.pt[1])) for b in blobs]

contours, h = cv2.findContours(thresh, cv2.RETR\_TREE, cv2.CHAIN\_APPROX\_SIMPLE)

maxC = max(contours, key=lambda c: cv2.contourArea(c))

boundRect = cv2.minAreaRect(maxC)

def ptDist(p1, p2):

dx=p2[0]-p1[0]; dy=p2[1]-p1[1]

return math.sqrt( dx\*dx + dy\*dy )

def ptMean(p1, p2):

return ((int(p1[0]+p2[0])/2, int(p1[1]+p2[1])/2))

def rect2centerline(rect):

p0=rect[0]; p1=rect[1]; p2=rect[2]; p3=rect[3];

width=ptDist(p0,p1); height=ptDist(p1,p2);

# centerline lies along longest median

if (height > width):

cl = ( ptMean(p0,p1), ptMean(p2,p3) )

else:

cl = ( ptMean(p1,p2), ptMean(p3,p0) )

return cl

centerline = rect2centerline(cv2.boxPoints(boundRect))

print(centerline[0],centerline[1])

cv2.line(drawImg, (int(centerline[0][0]), int(centerline[0][1])), (int(centerline[1][0]), int(centerline[1][1])), (0, 255, 0), 2)

def ptLineDist(pt, line):

x0=pt[0]; x1=line[0][0]; x2=line[1][0];

y0=pt[1]; y1=line[0][1]; y2=line[1][1];

return abs((x2-x1)\*(y1-y0)-(x1-x0)\*(y2-y1))/(math.sqrt((x2-x1)\*(x2-x1)+(y2-y1)\*(y2-y1)))

closestPiston = min( pistonCenters, key=lambda ctr: ptLineDist(ctr, centerline))

cv2.circle(drawImg, closestPiston, 5, (0,0,255), -1)

cv2.imshow("image", drawImg)

cv2.waitKey(0)

cv2.destroyAllWindows()

**CODE EXPLANATION:**

**17-B**

The provided Python code utilizes the OpenCV library to process and analyze images of the pump system. The main steps of the code can be summarized as follows:

**Image Loading:**

The code begins by loading an image of the pump system using the OpenCV library.

orig = cv2.imread(r"C:\Users\shibi\Desktop\AIVL\ir\b.jpg")

**Image Preprocessing:**

The loaded image undergoes several preprocessing steps, including resizing, conversion to grayscale, and thresholding.

resized = cv2.resize(orig, None, fx=1.5, fy=1.5)

gray = cv2.cvtColor(resized, cv2.COLOR\_BGR2GRAY)

threshVal = 180

ret, thresh = cv2.threshold(gray, threshVal, 255, cv2.THRESH\_BINARY)

**Pump Circle Detection:**

The code utilizes the HoughCircles function in OpenCV to detect circles corresponding to the pump in the thresholded image.

pumpCircles = cv2.HoughCircles(thresh, cv2.HOUGH\_GRADIENT, 1, PUMP\_DIAMETER, param2=5, minRadius=pumpRadiusRange[0], maxRadius=pumpRadiusRange[1])

**Piston Blob Detection:**

Blob detection is employed to identify regions in the thresholded image corresponding to the pistons.

blobParams = cv2.SimpleBlobDetector\_Params()

# ... (setting blob detection parameters)

blobDetector = cv2.SimpleBlobDetector\_create(blobParams)

blobs = blobDetector.detect(thresh)

**Centerline Estimation:**

The code calculates the centerline of the detected piston regions using the minAreaRect function in OpenCV.

contours, h = cv2.findContours(thresh, cv2.RETR\_TREE, cv2.CHAIN\_APPROX\_SIMPLE)

maxC = max(contours, key=lambda c: cv2.contourArea(c))

boundRect = cv2.minAreaRect(maxC)

centerline = rect2centerline(cv2.boxPoints(boundRect))

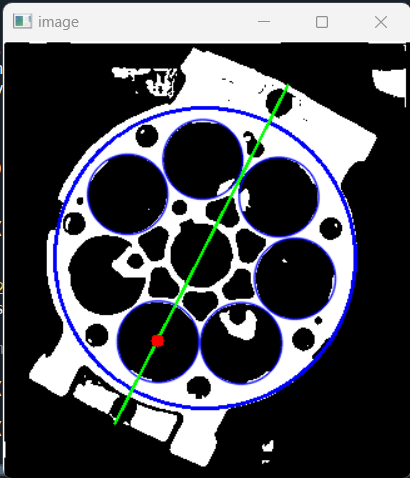
**Visualization:**

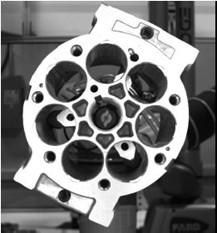
Finally, the code visualizes the results by drawing circles around the detected pump and pistons and a line representing the estimated centerline.

cv2.circle(drawImg, closestPiston, 5, (255, 255, 0), -1)

cv2.line(drawImg, (int(centerline[0][0]), int(centerline[0][1])), (int(centerline[1][0]), int(centerline[1][1])), (0, 0, 255), 2)

cv2.imshow("image", drawImg)

**INPUT IMAGE: OUTPUT IMAGE:**

****

**INBUILT FUNCTIONS USED:**

1. cv2.imread`:

- Description: Reads an image from a file.

- Usage: cv2.imread(file\_path)`

2. cv2.resize`:

- Description: Resizes an image to a specified width and height.

- Usage: cv2.resize(image, (width, height))`

3. cv2.cvtColor`:

- Description: Converts an image from one color space to another.

- Usage: cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY)` or `cv2.cvtColor(image, cv2.COLOR\_GRAY2BGR)`

4. cv2.threshold`:

- Description: Applies a fixed-level threshold to each pixel in the image.

- Usage: cv2.threshold(image, threshold\_value, max\_value, threshold\_type)`

5. cv2.HoughCircles`:

- Description: Finds circles in an image using the Hough transform.

- Usage: cv2.HoughCircles(image, method, dp, minDist, param1, param2, minRadius, maxRadius)`

6. cv2.SimpleBlobDetector`:

- Description: Detects blobs (connected components) in a binary image.

- Usage: cv2.SimpleBlobDetector\_create(params)`

7. cv2.drawKeypoints`:\*\*

- Description: Draws keypoints on an image.

- Usage: `cv2.drawKeypoints(image, keypoints, output\_image, color, flags)`

8. cv2.findContours:

- Description: Finds contours in a binary image.

- Usage: `cv2.findContours(image, mode, method)`

9. cv2.minAreaRect:

- Description: Finds the rotated rectangle with the minimum area enclosing a set of points.

- Usage: `cv2.minAreaRect(points)`

10. cv2.line:

- Description: Draws a line on an image between two specified points.

- Usage: `cv2.line(image, point1, point2, color, thickness)`

11. cv2.imshow:

- Description: Displays an image in a window.

- Usage: `cv2.imshow(window\_name, image)`

12. cv2.setMouseCallback:

- Description: Sets a callback function to handle mouse events.

- Usage: `cv2.setMouseCallback(window\_name, callback\_function)`

13. cv2.waitKey:

- Description: Waits for a keyboard event.

- Usage: `cv2.waitKey(delay)`

14. cv2.destroyAllWindows:

- Description: Destroys all OpenCV windows.

- Usage: `cv2.destroyAllWindows()`

**CONCLUSION:**

Use of the OpenCV library to perform image processing, contour detection, circle detection, blob detection, and visualization tasks. OpenCV's rich set of functions simplifies complex image analysis tasks, making it a powerful tool for computer vision applications. The modularity and versatility of OpenCV allow developers to create interactive tools, annotate images, and extract meaningful information from visual data. In conclusion, the use of OpenCV in these projects underscores its importance in the field of computer vision for both research and practical applications.