import cv2

import numpy as np

from scipy.spatial import distance as dist

from scipy.optimize import least\_squares

import imutils

# Define midpoint function

def midpoint(ptA, ptB):

return ((ptA[0] + ptB[0]) / 2, (ptA[1] + ptB[1]) / 2)

# Draw a small cross at the point

def draw\_small\_cross(image, center, size=1, color=(0, 0, 255), thickness=1):

x, y = center

cv2.line(image, (x - size, y), (x + size, y), color, thickness)

cv2.line(image, (x, y - size), (x, y + size), color, thickness)

# Initialize points for the reference and measurement lines/circle

ref\_points = []

measurement\_points = []

# Zoom window with crosshair

def create\_zoom\_window(image, x, y, zoom\_factor=6, window\_size=75, window\_x=1300, window\_y=-50):

#adjust zoom from variable 'zoom\_factor'

#adjust location of zoomed window according to the display monitor being used by 'window\_x' and 'window\_y'

h, w = image.shape[:2]

half\_size = window\_size // 2

left, top = max(0, x - half\_size), max(0, y - half\_size)

right, bottom = min(w, x + half\_size), min(h, y + half\_size)

roi = image[top:bottom, left:right]

if roi.size == 0:

return None

zoomed = cv2.resize(roi, None, fx=zoom\_factor, fy=zoom\_factor)

zh, zw = zoomed.shape[:2]

zoom\_window = np.zeros((window\_size \* zoom\_factor, window\_size \* zoom\_factor, 3), dtype=np.uint8)

cx, cy = zoom\_window.shape[1] // 2, zoom\_window.shape[0] // 2

zoom\_window[max(0, cy - zh // 2):cy + zh // 2, max(0, cx - zw // 2):cx + zw // 2] = zoomed

cv2.line(zoom\_window, (cx - 10, cy), (cx + 10, cy), (0, 255, 0), 1)

cv2.line(zoom\_window, (cx, cy - 10), (cx, cy + 10), (0, 255, 0), 1)

# Display the zoom window

cv2.imshow("Zoom Window", zoom\_window)

cv2.moveWindow("Zoom Window", window\_x, window\_y)

return zoom\_window

# Modified mouse callback function for reference line with zoom

def click\_event\_ref(event, x, y, flags, param):

global ref\_points

image = param.copy() #Creating a copy to avoid modifying the original

if event == cv2.EVENT\_MOUSEMOVE:

temp\_image = param.copy()

# Draw existing points and lines

for point in ref\_points:

draw\_small\_cross(image, (x, y), size=1, color=(0, 0, 255), thickness=1)

if len(ref\_points) == 2:

cv2.line(temp\_image, ref\_points[0], ref\_points[1], (255, 0, 0), 2)

# Create and show zoom window

zoom\_window = create\_zoom\_window(temp\_image, x, y)

if zoom\_window is not None:

cv2.imshow("Zoom Window", zoom\_window)

elif event == cv2.EVENT\_LBUTTONDOWN:

ref\_points.append((x, y))

draw\_small\_cross(image, (x, y), size=1, color=(0, 0, 255), thickness=1)

if len(ref\_points) == 2:

cv2.line(image, ref\_points[0], ref\_points[1], (255, 0, 0), 2)

cv2.imshow("Image", image)

# Modified mouse callback function for measurements with zoom

def click\_event\_measure(event, x, y, flags, param):

global measurement\_points

image = param.copy() #Creating a copy to avoid modifying the original

if event == cv2.EVENT\_MOUSEMOVE:

temp\_image = param.copy()

# Draw existing points

for point in measurement\_points:

draw\_small\_cross(image, (x, y), size=1, color=(0, 0, 255), thickness=1)

# Create and show zoom window

zoom\_window = create\_zoom\_window(temp\_image, x, y)

if zoom\_window is not None:

cv2.imshow("Zoom Window", zoom\_window)

elif event == cv2.EVENT\_LBUTTONDOWN:

measurement\_points.append((x, y))

draw\_small\_cross(image, (x, y), size=1, color=(0, 0, 255), thickness=1)

cv2.imshow("Image", image)

# Load image from file path

def load\_image(image\_path):

image = cv2.imread(image\_path)

if image is None:

raise FileNotFoundError(f"Image not found at {image\_path}")

return image

# Capture a line from the image as the reference line

def select\_reference\_line(image):

global ref\_points

ref\_points = [] # Reset points

print("Click on two points to select the reference line.")

clone = image.copy()

cv2.namedWindow("Image")

cv2.namedWindow("Zoom Window")

cv2.imshow("Image", clone)

cv2.setMouseCallback("Image", click\_event\_ref, param=clone)

cv2.waitKey(0)

cv2.destroyWindow("Image")

cv2.destroyWindow("Zoom Window")

if len(ref\_points) != 2:

raise ValueError("Two points must be selected to define a reference line.")

return ref\_points[0], ref\_points[1]

# Select units for measurements with shorthand notation

def select\_units():

print("Enter the unit for measurement (options: inches, centimeter, millimeter, feet, meter, yard): ")

unit\_input = input("Enter unit shorthand (in, cm, mm, ft, m, yd): ").strip().lower()

if unit\_input in ['in', 'cm', 'mm', 'ft', 'm', 'yd']:

return unit\_input

else:

raise ValueError("Invalid unit. Please select from 'in', 'cm', 'mm', 'ft', 'm', or 'yd'.")

# Calculate pixels-per-metric ratio based on known length and units

def calculate\_pixels\_per\_metric(pointA, pointB, known\_length):

pixel\_distance = dist.euclidean(pointA, pointB)

pixels\_per\_metric = pixel\_distance / known\_length

return pixels\_per\_metric

# Best-fit circle calculation using least squares method

def fit\_circle(points):

# Define the function for least squares

def calc\_R(c):

return np.sqrt((points[:, 0] - c[0])\*\*2 + (points[:, 1] - c[1])\*\*2)

def f\_2(c):

Ri = calc\_R(c)

return Ri - Ri.mean()

center\_estimate = np.mean(points, axis=0)

result = least\_squares(f\_2, center\_estimate)

center = result.x

radius = calc\_R(center).mean()

return center, radius

# Capture multiple points

def select\_measurement\_points(image):

global measurement\_points

measurement\_points = [] # Reset points

print("Click on two points for a line, three points for a circle, more than 3 points for an irregular shape. Press 'Enter' when done.")

clone = image.copy()

cv2.namedWindow("Image")

cv2.namedWindow("Zoom Window")

cv2.imshow("Image", clone)

cv2.setMouseCallback("Image", click\_event\_measure, param=clone)

cv2.waitKey(0)

cv2.destroyWindow("Image")

cv2.destroyWindow("Zoom Window")

if len(measurement\_points) < 2:

raise ValueError("At least two points are required for a measurement.")

return measurement\_points

# Calculate the area of an irregular shape using contour area

def calculate\_area\_of\_shape(shape\_points):

if len(shape\_points) > 2:

contour = np.array(shape\_points)

area = cv2.contourArea(contour)

return area

else:

return 0

# Main processing function

def process\_image(image\_path):

image = load\_image(image\_path)

# Select reference line and get known length

pointA, pointB = select\_reference\_line(image)

# Select units

unit\_name = select\_units()

known\_length = float(input(f"Enter the known length of the selected reference line in {unit\_name}: "))

pixels\_per\_metric = calculate\_pixels\_per\_metric(pointA, pointB, known\_length)

print(f"Pixels per {unit\_name}: {pixels\_per\_metric}")

while True:

# Select measurement points on target object

points = select\_measurement\_points(image)

points\_np = np.array(points)

if len(points) == 2: # Line measurement

target\_distance\_in\_pixels = dist.euclidean(points[0], points[1])

target\_distance\_in\_units = target\_distance\_in\_pixels / pixels\_per\_metric

print(f"Measured Distance: {target\_distance\_in\_units:.6f} {unit\_name}")

cv2.line(image, points[0], points[1], (0, 255, 0), 2)

cv2.putText(image, f"{target\_distance\_in\_units:.6f} {unit\_name}",

(points[0][0] - 10, points[0][1] - 10),

cv2.FONT\_HERSHEY\_SIMPLEX, 0.5, (255, 255, 255), 2)

elif len(points) == 3: # Circle measurement

center, radius = fit\_circle(points\_np)

diameter\_in\_pixels = 2 \* radius

diameter\_in\_units = diameter\_in\_pixels / pixels\_per\_metric

area\_in\_pixels = np.pi \* radius \*\* 2

area\_in\_units = area\_in\_pixels / (pixels\_per\_metric \*\* 2)

print(f"Measured Diameter of : {diameter\_in\_units:.6f} {unit\_name}")

print(f"Area of Circle: {area\_in\_units:.6f} {unit\_name}²")

center = (int(center[0]), int(center[1]))

cv2.circle(image, center, int(radius), (0, 255, 0), 2)

cv2.putText(image, f"Diameter: {diameter\_in\_units:.6f} {unit\_name}",

(center[0] - 20, center[1] - 20),

cv2.FONT\_HERSHEY\_SIMPLEX, 0.5, (255, 255, 255), 2)

cv2.putText(image, f"Area: {area\_in\_units:.6f} {unit\_name}2",

(center[0] - 20, center[1] - 40),

cv2.FONT\_HERSHEY\_SIMPLEX, 0.5, (255, 255, 255), 2)

else: # Irregular shape area measurement

area\_in\_pixels = calculate\_area\_of\_shape(points\_np)

area\_in\_units = area\_in\_pixels / (pixels\_per\_metric \*\* 2)

print(f"Area of the Irregular Shape: {area\_in\_units:.6f} {unit\_name}²")

cv2.polylines(image, [np.array(points)], isClosed=True, color=(0, 255, 255), thickness=2)

cv2.putText(image, f"Area: {area\_in\_units:.6f} {unit\_name}2", (10, 30),

cv2.FONT\_HERSHEY\_SIMPLEX, 1, (0, 255, 255), 2)

# Display result

cv2.imshow("Measurement", image)

print("Press 'Esc' to exit or 'Enter' to measure another dimension.")

key = cv2.waitKey(0) & 0xFF

if key == 27: # Escape key pressed

break

cv2.destroyAllWindows()

# Example usage

if \_\_name\_\_ == "\_\_main\_\_":

image\_path = "/Users/harmansinghjohar/Downloads/E8\_1.bmp" # Update path as needed

process\_image(image\_path)