import cv2

import numpy as np

import sys

from tqdm import tqdm

import os

g\_cClicks = 0

aPoints = []

SECONDS = 1000

def CallBackFunc(event, x, y, flags, param):

global g\_cClicks, aPoints

if event == cv2.EVENT\_LBUTTONDOWN:

aPoints.append((x, y))

g\_cClicks += 1

print(f"Point No. {g\_cClicks} - position ({x}, {y})")

def PointMean(points):

mean\_x = sum(p[0] for p in points) / len(points)

mean\_y = sum(p[1] for p in points) / len(points)

return mean\_x, mean\_y

def transform\_bbox(bbox, map\_x, map\_y):

x\_min, y\_min, x\_max, y\_max = bbox

corners = [

(x\_min, y\_min),

(x\_min, y\_max),

(x\_max, y\_min),

(x\_max, y\_max)

]

new\_corners = []

for x, y in corners:

if 0 <= int(y) < map\_y.shape[0] and 0 <= int(x) < map\_x.shape[1]:

new\_x = map\_x[int(y), int(x)]

new\_y = map\_y[int(y), int(x)]

new\_corners.append((new\_x, new\_y))

if not new\_corners:

return None

xs, ys = zip(\*new\_corners)

new\_bbox = (int(min(xs)), int(min(ys)), int(max(xs)), int(max(ys)))

return new\_bbox

def yolo\_to\_pixel\_bbox(yolo\_bbox, img\_width, img\_height):

class\_id, x\_c, y\_c, w, h = yolo\_bbox

x\_c \*= img\_width

y\_c \*= img\_height

w \*= img\_width

h \*= img\_height

x\_min = int(x\_c - w / 2)

y\_min = int(y\_c - h / 2)

x\_max = int(x\_c + w / 2)

y\_max = int(y\_c + h / 2)

return (x\_min, y\_min, x\_max, y\_max), class\_id

def pixel\_to\_yolo\_bbox(pixel\_bbox, img\_width, img\_height, class\_id):

x\_min, y\_min, x\_max, y\_max = pixel\_bbox

w = x\_max - x\_min

h = y\_max - y\_min

x\_c = x\_min + w / 2

y\_c = y\_min + h / 2

return f"{int(class\_id)} {x\_c / img\_width:.5f} {y\_c / img\_height:.5f} {w / img\_width:.5f} {h / img\_height:.5f}"

def main():

image\_path = 'image/fe2.jpg'

yolo\_txt\_path = 'image/fe2.txt'

input\_frame = cv2.imread(image\_path)

if input\_frame is None:

print("Error: Cannot load image.")

sys.exit(-1)

nHeight, nWidth = input\_frame.shape[:2]

if nHeight > 600:

input\_frame = cv2.resize(input\_frame, (int(nWidth \* 600 / nHeight), 600))

nHeight, nWidth = input\_frame.shape[:2]

temp\_frame = input\_frame.copy()

cv2.imshow("Input Frame", temp\_frame)

cv2.setMouseCallback("Input Frame", CallBackFunc)

print("\nPlease mark 12 points on the boundary of the circle\n")

while len(aPoints) < 12:

key = cv2.waitKey(100)

temp\_frame = input\_frame.copy()

for point in aPoints:

cv2.drawMarker(temp\_frame, point, (0, 0, 255), markerType=cv2.MARKER\_CROSS, markerSize=10, thickness=2)

cv2.imshow("Input Frame", temp\_frame)

mean\_x, mean\_y = PointMean(aPoints)

A = np.array([[x, y, 1] for x, y in aPoints])

B = np.array([-((x\*\*2 + y\*\*2)) for x, y in aPoints])

C = np.linalg.lstsq(A, B, rcond=None)[0]

A, B, C = C

Cx = -A/2

Cy = -B/2

R = np.sqrt(Cx\*\*2 + Cy\*\*2 - C)

print(f"Detected Circle -> Center: ({Cx:.2f}, {Cy:.2f}), Radius: {R:.2f}")

cv2.circle(temp\_frame, (int(Cx), int(Cy)), int(R), (0, 255, 0), 4)

cv2.imshow("Input Frame", temp\_frame)

cv2.waitKey(SECONDS)

transform\_map\_x = np.zeros((nHeight, nWidth), dtype=np.float32)

transform\_map\_y = np.zeros((nHeight, nWidth), dtype=np.float32)

p\_bar = tqdm(total=nWidth \* nHeight)

for u in range(nWidth):

for v in range(nHeight):

p\_bar.update(1)

xt = float(u - Cx)

yt = float(v - Cy)

if xt != 0:

AO1 = (xt \* xt + R \* R) / (2.0 \* xt)

AB = np.sqrt(xt \* xt + R \* R)

AP = yt

PE = R - yt

a = AP / PE if PE != 0 else 1.0

b = 2.0 \* np.arcsin(AB / (2.0 \* AO1)) if AO1 != 0 else 0.0

alpha = a \* b / (a + 1.0) if a + 1.0 != 0 else 0.0

x1 = xt - AO1 + AO1 \* np.cos(alpha)

y1 = AO1 \* np.sin(alpha)

else:

x1 = xt

y1 = yt

transform\_map\_x[v, u] = x1 + Cx

transform\_map\_y[v, u] = y1 + Cy

p\_bar.close()

output\_frame = cv2.remap(input\_frame, transform\_map\_x, transform\_map\_y, cv2.INTER\_CUBIC)

# Read YOLO bboxes from txt file

if not os.path.exists(yolo\_txt\_path):

print("No YOLO bbox file found.")

cv2.imshow("Output Frame", output\_frame)

cv2.waitKey(0)

return

with open(yolo\_txt\_path, 'r') as f:

lines = f.readlines()

transformed\_yolo\_bboxes = []

for line in lines:

yolo\_values = list(map(float, line.strip().split()))

pixel\_bbox, class\_id = yolo\_to\_pixel\_bbox(yolo\_values, nWidth, nHeight)

# Draw original bbox on input frame

cv2.rectangle(input\_frame, (pixel\_bbox[0], pixel\_bbox[1]), (pixel\_bbox[2], pixel\_bbox[3]), (0, 255, 255), 2)

# Transform bbox

transformed\_bbox = transform\_bbox(pixel\_bbox, transform\_map\_x, transform\_map\_y)

if transformed\_bbox:

cv2.rectangle(output\_frame,

(transformed\_bbox[0], transformed\_bbox[1]),

(transformed\_bbox[2], transformed\_bbox[3]),

(255, 0, 0), 2)

# Convert back to YOLO format

new\_yolo = pixel\_to\_yolo\_bbox(transformed\_bbox, nWidth, nHeight, class\_id)

transformed\_yolo\_bboxes.append(new\_yolo)

# Save new YOLO bbox file

out\_txt\_path = yolo\_txt\_path.replace(".txt", "\_transformed.txt")

with open(out\_txt\_path, 'w') as f\_out:

for line in transformed\_yolo\_bboxes:

f\_out.write(line + '\n')

print(f"Saved transformed YOLO annotations to: {out\_txt\_path}")

cv2.imshow("Input Frame with BBoxes", input\_frame)

cv2.imshow("Output Frame with Transformed BBoxes", output\_frame)

cv2.waitKey(0)

cv2.destroyAllWindows()

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

main()