Lena_ARTag

October 19, 2021

1 AR-TAG DETECTION AND SUPERIMPOSING IMAGE

IMPORTS

[1]:
```
import cv2
import numpy as np
```

FUNCTION DEFINATION

[2]:
```
# Function Name: pre_processing
# Input: raw image frame
# Output: processed image frame
# Description: Used to remove noise from image,convert to grayscale and␣
 ↪threshold the image to get AR Tag
```

[3]:
```
def pre_proceesing(raw_image):
    smooth_img=cv2.GaussianBlur(raw_image,(5,5),1)
    gray_image=cv2.cvtColor(smooth_img, cv2.COLOR_BGR2GRAY)
    p,proc_img=cv2.threshold(gray_image,0,255,cv2.THRESH_BINARY+cv2.THRESH_OTSU)
    return proc_img
```

[4]:
```
# Function Name: checkCorners
# Input: processed image frame, AR-Tag co-ordinates
# Output: Labeled co-ordinate (TL,BL,BR,BL)
# Description: Used to get the coordinates of AR Tag in the order TL TR BL BR
```

[5]:
```
def checkCorners(proc_img, coord):
    count = 0
    tl, tr, bl, br = 0, 0, 0, 0
    if coord[0] < 1900 and coord[1] < 1000:
        if proc_img[coord[1]-10][coord[0]-10] == 255:
            count = count + 1
            tl = 1
        if proc_img[coord[1]+10][coord[0]-10] == 255:
            count = count + 1
            bl = 1
        if proc_img[coord[1]-10][coord[0]+10] == 255:
            count = count + 1
```

```
            tr = 1
        if proc_img[coord[1]+10][coord[0]+10] == 255:
            count = count + 1
            br = 1

        if count == 3:
            if tl == 1 and tr == 1 and bl == 1:
                return True, 'TL'
            elif tl == 1 and tr == 1 and br == 1:
                return True, 'TR'
            elif tl == 1 and br == 1 and bl == 1:
                return True, 'BL'
            elif tr == 1 and br == 1 and bl == 1:
                return True, 'BR'
        else:
            return False, None
    else:
        return False, None
```

[6]:
```
# Function Name: tag_corner_points
# Input: processed image frame
# Output: AR-Tag corners
# Description: Used to get the corners of AR Tag
```

[7]:
```
def tag_corner_points(proc_img):
    _, contours,_ = cv2.findContours(proc_img,cv2.RETR_LIST,cv2.
 ↪CHAIN_APPROX_SIMPLE)
    tag_corners=[]
    for contour in contours:
        contour_corners = []
        if cv2.contourArea(contour) > 0:
            epsilon = 0.1 * cv2.arcLength(contour, True)
            approximation = cv2.approxPolyDP(contour, epsilon, True)
            for coor in approximation:
                result, key = checkCorners(proc_img, [coor[0][0], coor[0][1]])
                if result == True:
                    contour_corners.append([coor[0][0], coor[0][1], key])
            if len(contour_corners) == 4:
                tag_corners=contour_corners
    order_tag_corner=[[],[],[],[]]
    tl=[]
    tr=[]
    bl=[]
    br=[]
    for i in range(0,len(tag_corners)):
        if tag_corners[i][2]=="TL":
            tl.append(tag_corners[i][0])
```

```
            tl.append(tag_corners[i][1])
            order_tag_corner[0]=tl
        elif tag_corners[i][2]=="TR":
            tr.append(tag_corners[i][0])
            tr.append(tag_corners[i][1])
            order_tag_corner[1]=tr
        elif tag_corners[i][2]=="BL":
            bl.append(tag_corners[i][0])
            bl.append(tag_corners[i][1])
            order_tag_corner[2]=bl
        elif tag_corners[i][2]=="BR":
            br.append(tag_corners[i][0])
            br.append(tag_corners[i][1])
            order_tag_corner[3]=br
    #cv2.circle(proc_img, (order_tag_corner[0][0],order_tag_corner[0][1]), 10,␣
↪(0,0,255)) # Create a ring of red color on the top left corner in the frame
    #cv2.circle(proc_img, (order_tag_corner[1][0],order_tag_corner[1][1]), 10,␣
↪(0,255,0)) # Create a ring of green color on the top right corner in the frame
    #cv2.circle(proc_img, (order_tag_corner[2][0],order_tag_corner[2][1]), 10,␣
↪(255,0,0)) # Create a ring of blue color on the bottom left corner in the␣
↪frame
    #cv2.circle(proc_img, (order_tag_corner[3][0],order_tag_corner[3][1]), 10,␣
↪(0,5,25)) # Create a ring of yellow color on the bottom right corner in the␣
↪frame
    return order_tag_corner
```

[8]:
```
# Function Name: homograprhy
# Input: AR-Tag points
# Output: Homography matrix
# Description: Used to get the homographic transform form the corner points
```

[9]:
```
def homograprhy(artagpoints):
    worldpoints=np.array([[0,0],[0,199],[199,0],[199,199]])
    row=0
    A=np.empty((8,9))
    for i in range(len(artagpoints)):
        x_c=artagpoints[i][0]
        y_c=artagpoints[i][1]
        x_w=worldpoints[i][0]
        y_w=worldpoints[i][1]
        A[row] = np.array([x_c, y_c, 1, 0, 0, 0, -x_w*x_c, -x_w*y_c, -x_w])
        A[row + 1] = np.array([0, 0, 0, x_c, y_c, 1, -y_w*x_c, -y_w*y_c, -y_w])
        row = row + 2
    U, s, V = np.linalg.svd(A, full_matrices=True)
    V = (V) / ((V[8][8]))
    homographymat = V[8,:].reshape(3, 3)
    return homographymat
```

[10]:
```
# Function Name: artag_extraction
# Input: homography matrix, pre-processed image
# Output: AR-Tag
# Description: Used to get the AR-Tag in an 200*200 image
```

[11]:
```
def artag_extraction(homographymat,pre_processed_frame):
    inv_homo=-np.linalg.inv(homographymat)
    ar_tag_image = np.zeros((200, 200))
    # Copy the AR Tag tag in video to new  frame
    for m in range(0, 200):
        for n in range(0, 200):
            x1, y1, z1 = np.matmul(inv_homo, [m, n, 1])
            if (int(y1/z1) < 1080 and int(y1/z1) > 0) and (int(x1/z1) < 1920 and␣
 ↪int(x1/z1) > 0):
                    ar_tag_image[m][n] = pre_processed_frame[int(y1/z1)][int(x1/
 ↪z1)] # Creating a new image from the camera frame
    return ar_tag_image
```

[12]:
```
# Function Name: artag_id
# Input: AR-Tag image
# Output: AR-Tag ID, angle orientaion in the frame
# Description: Used to get the AR-Tag ID from the AR-Tag image
```

[13]:
```
def artag_id(ar_tag_image):
    angle=0
    id=0
    m=0
    n=0
    reshaped_artag = cv2.resize(ar_tag_image,(64,64))
    step_x=int(reshaped_artag.shape[0]/8)
    step_y=int(reshaped_artag.shape[1]/8)
    artagid=np.empty((8,8))
    for i in range(0,64,step_x):
        n=0
        for j in range(0,64,step_y):
            count_white=0
            count_black=0
            for x in range(0,step_x-1):
                for y in range(0,step_x-1):
                    if(reshaped_artag[i+x][j+y]==255):
                        count_white=count_white+1
                    else:
                        count_black=count_black+1
            if(count_white>count_black):
                artagid[m][n]=1
            else:
                artagid[m][n]=0
```

```
            n=n+1
        m=m+1

    if(artagid[2][2] == 0 and artagid[2][5] == 0 and artagid[5][2] == 0 and␣
↪artagid[5][5] == 1):
        id = artagid[3][3] +artagid[4][3]*8 +artagid[4][4]*4 + artagid[3][4]*2
        angle=0
    if(artagid[2][2] == 1 and artagid[2][5] == 0 and artagid[5][2] == 0 and␣
↪artagid[5][5] == 0):#180
        id = artagid[3][3]*4 + artagid[3][4]*2 + artagid[4][4]*1 +␣
↪artagid[4][3]*8
        angle=180
    if(artagid[2][2] == 0 and artagid[2][5] == 1 and artagid[5][2] == 0 and␣
↪artagid[5][5] == 0):#90
        id = artagid[3][3]*2 + artagid[4][3]*4 + artagid[4][4]*8 + artagid[3][4]
        angle=90
    if(artagid[2][2] ==0 and artagid[2][5] == 0 and artagid[5][2] == 1 and␣
↪artagid[5][5] == 0):#270
        id = artagid[3][3]*1 + artagid[4][3]*8 + artagid[4][4]*2 +␣
↪artagid[3][4]*4
        angle=270
    return id,angle
```

[ ]:
```
# Function Name: superposition_lena
# Input: angle,homographymat,order_tag_corner,lena,current_frame
# Output: AR-Tag covered by Lena in the frame
# Description: Used to get the Lena image on the AR-Tag
```

[23]:
```
def superposition_lena(angle,homographymat,order_tag_corner,lena,current_frame):
    if angle == 0:
        points = order_tag_corner
    elif angle == 90:
        points = [order_tag_corner[2], order_tag_corner[0], order_tag_corner[3],␣
↪order_tag_corner[1]]
    elif angle == 270:
        points = [order_tag_corner[1], order_tag_corner[3], order_tag_corner[0],␣
↪order_tag_corner[2]]
    elif angle == 180:
        points = [order_tag_corner[3], order_tag_corner[2], order_tag_corner[1],␣
↪order_tag_corner[0]]
    lena_resized = cv2.resize(lena, (200, 200))
    homographymat_lena= homograprhy(points)
    inv_homographymat_lena = np.linalg.inv(homographymat_lena)
    for j in range(0,200):
        for k in range(0,200):
            x2, y2, z2 = np.matmul(inv_homographymat_lena,[j,k,1])
```

```
            if (int(y2/z2) < 1080 and int(y2/z2) > 0) and (int(x2/z2) < 1920 and␣
 ↪int(x2/z2) > 0):
                current_frame[int(y2/z2)][int(x2/z2)] = lena_resized[j][k]
    return current_frame
```

PIPELINE

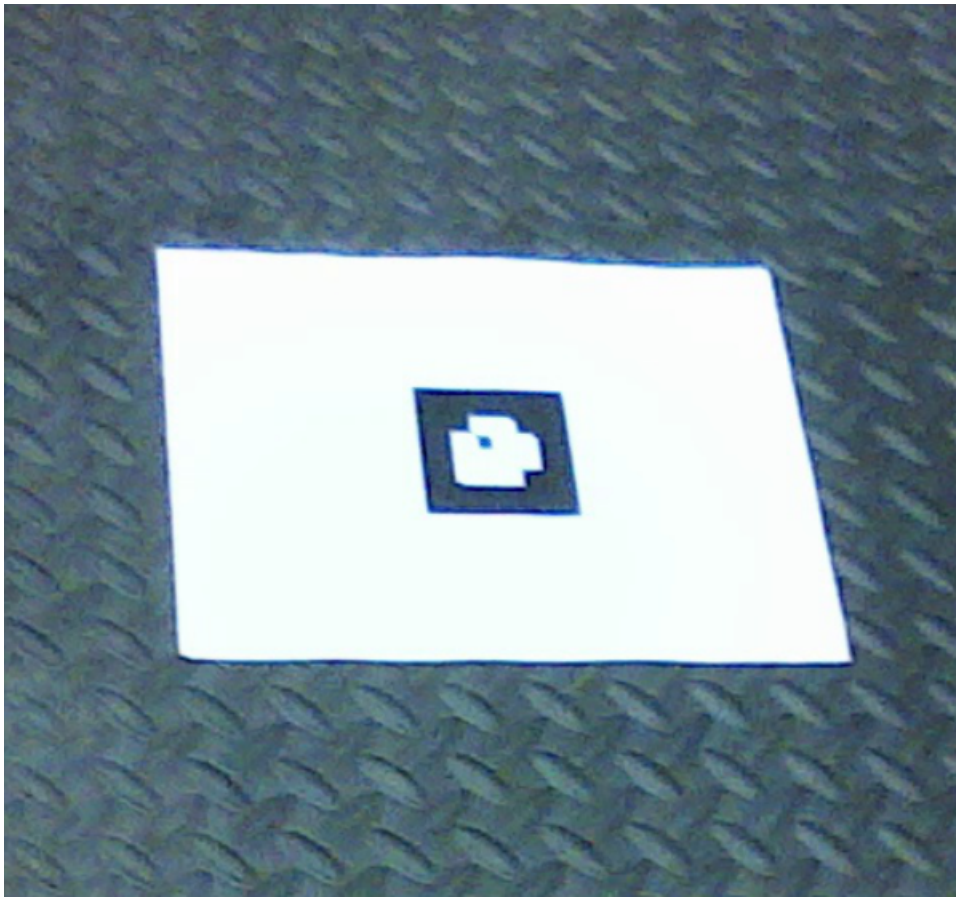
[26]:
```
lena=cv2.imread('C:/Users/AALAP RANA/Lenna.jpg')
curr_frame = cv2.imread('C:/Users/AALAP RANA/Computer vision/ar-tag/data/frames/
 ↪frame0.jpg')
pre_processed_frame=pre_proceesing(curr_frame)
order_tag_corner=tag_corner_points(pre_processed_frame)
homographymat=homographrhy(order_tag_corner)
ar_tag_image=artag_extraction(homographymat,pre_processed_frame)
id,angle=artag_id(ar_tag_image)
if(id>0):
    cv2.putText(curr_frame, "AR_Tag_ID: "+str(int(id)),␣
 ↪(order_tag_corner[0][0]-50, order_tag_corner[0][1]-20), fontFace = cv2.
 ↪FONT_ITALIC, thickness=2, fontScale = 1, color= (125,0,255))
super_lena=superposition_lena(angle,homographymat,order_tag_corner,lena,curr_frame)
```

DISPLAY

[27]:
```
cv2.imshow('super_lena',super_lena)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

[29]:
```
from IPython.display import Image
from IPython.display import display
x=Image(filename = "C:/Users/AALAP RANA/Desktop/original.png", width = 200,␣
 ↪height = 200)
print("Original Image")
p=Image(filename = "C:/Users/AALAP RANA/Desktop/call.png", width = 200, height =␣
 ↪200)
print("AR-Tag with ID")
z=Image(filename = "C:/Users/AALAP RANA/Desktop/lenna_on.png", width = 200,␣
 ↪height = 200)
print("Lena on AR-Tag")
display(x,p)
display(z)
```

```
Original Image
AR-Tag with ID
Lena on AR-Tag
```




AR_Tag_ID: 14


AR_Tag_ID: 14

[ ]: