~\Documents\UIUC Courses\ECE420\ECE420_FinalProject\iCardScanner.py

```python
# Import libraries
import cv2
import numpy as np
import imageio.v3 as iio
import matplotlib.pyplot as plt
import sys
import os
import pytesseract

#%%

# Function that returns the contour of the largest rectangular object in the frame
def get_bounding_quadrangle(img):
    kernel = np.ones((3,3), np.uint8)
    img_dilate = cv2.dilate(img, kernel, iterations=1)
    # cv2.imshow("Dilate", img_dilate)

    # Perform Canny Edge detection
    img_edgedetect = cv2.Canny(img_dilate, threshold1=100, threshold2=200)
    cv2.imshow('Edges', img_edgedetect)

    # Get Contours
    img_contours, hierarchy = cv2.findContours(img_edgedetect, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

    img_with_contours = np.ones_like(img, dtype=np.uint8) * 255

    # Find the contour with the maximum area
    max_area = 0
    max_idx = -1

    for idx, contour in enumerate(img_contours):
        contour_convex_hull = cv2.convexHull(contour)
        convex_hull_area = cv2.contourArea(contour_convex_hull)
        # print(contour_convex_hull)

        if(max_area < convex_hull_area):
            max_idx = idx
            max_area = convex_hull_area

    if (max_idx >= 0):
        cv2.drawContours(img_with_contours, [img_contours[max_idx]], 0, (0,255,0), 2)
        # cv2.imshow('Contours no hull', img_with_contours)
        cv2.drawContours(img_with_contours, [cv2.convexHull(img_contours[max_idx])], 0, (0,255,0), 2)
        cv2.imshow('Contours', img_with_contours)
    else:
        print("No max contour found")
        return -1

    # Apply polygon approximation
    for err in np.linspace(0.005, 0.09, 80):
        approx_contour = cv2.approxPolyDP(cv2.convexHull(img_contours[max_idx]), err * cv2.arcLength(img_contours[max_idx], True), True)
        # print(f"err: {err}, numpt: {approx_contour.shape}")
        if(approx_contour.shape[0] == 4):
            break

    return approx_contour

# Uses tesseract OCR to return all the characters in an image, thresholding applied as pre processing for tesseract
def extract_text_from_image(image):
    # Threhold the image
    blur = cv2.GaussianBlur(image,(3,3),0)
    ret3,img_uin = cv2.threshold(blur,0,255,cv2.THRESH_BINARY+cv2.THRESH_OTSU)

    # Convert the image to RGB (Tesseract requires RGB images)
    rgb_image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

    # Use Tesseract to extract text, signle line mode (psm7)
    text = pytesseract.image_to_string(rgb_image, config='--psm 7')

    cv2.imshow("thresh_crop", img_uin)

    return text


#%%

IMG_DIR = 'imgs/'

# Video for testing: Change the video to test and the accurate UIN
video = cv2.VideoCapture(f'{IMG_DIR}icard0_still.mp4')
TRUE_UIN = "659750250"

if not video.isOpened():
    print('Cannot open video')
    sys.exit()

# Set up output video file writer (records annotated frames to an mp4 file, for presentation)
# output_video_path = f'{IMG_DIR}icard0_outputvid_1.mp4'
# frame_width = int(video.get(cv2.CAP_PROP_FRAME_WIDTH))
# frame_height = int(video.get(cv2.CAP_PROP_FRAME_HEIGHT))
# fps = 2 #video.get(cv2.CAP_PROP_FPS)
# fourcc = cv2.VideoWriter_fourcc(*'mp4v')  # MP4 file
# out = cv2.VideoWriter(output_video_path, fourcc, fps, (frame_width, frame_height))

# TESTING: variables to store accuracy measurement in each frame
user_input_accurate_frames = [] # We manually indicate 1 for an accurately detected card and 0 if not
accurate_uin = [] # Records if text detected in the frame matches the true UIN (1) or not (0)
false_uin = [] # Records if the text detected contains digits of the expected length of a UIN, but is incorrect (1)


# for filename in jpg_names:
#     img = cv2.imread(f"{IMG_DIR}{filename}", cv2.IMREAD_GRAYSCALE)

# Loop through each frame of the bideo
while True:
    read_success, img_color = video.read()
    if not read_success:
        # We have reached the last frame of the video
        print('End of video')
        cv2.waitKey(0)
        cv2.destroyAllWindows()

        # For the output annotated video
        #out.release()

        # TESTING: accuracy metrics
        # user_input_accurate_frames = user_input_accurate_frames[1:]
        # accurate_uin = accurate_uin[1:]
        # print(f"Card Detection Accuracy = {sum(user_input_accurate_frames) / len(user_input_accurate_frames)}")
        # print(f"UIN Detection Accuracy = {sum(accurate_uin) / sum(user_input_accurate_frames)}")
        # print(f"False UIN Detection = {sum(false_uin) / len(user_input_accurate_frames)}")
        # print(f"Num frames = {len(user_input_accurate_frames)}")

        sys.exit()

    # Switch to grayscale for processing
    img = cv2.cvtColor(img_color, cv2.COLOR_BGR2GRAY)


    img_poly_contour = img_color
    approx_contour = get_bounding_quadrangle(img)

    if(type(approx_contour) == np.ndarray and approx_contour.shape[0] == 4):
        cv2.drawContours(img_poly_contour, [approx_contour], -1, (0,255,0), 2)

        # Get top left corner based on side lengths
        vec1 = approx_contour[0,0] - approx_contour[1,0]
        vec2 = approx_contour[0,0] - approx_contour[-1,0]

        # Get CW/CCW by cross product magnitude
        if(vec1[0]*vec2[1] - vec1[1]*vec2[0] > 0):
            approx_contour = approx_contour[::-1]
```

```
        vec1 = approx_contour[0,0] - approx_contour[1,0]
        vec2 = approx_contour[0,0] - approx_contour[-1,0]
        if(np.linalg.norm(vec1) > np.linalg.norm(vec2)):
            approx_contour = np.roll(approx_contour, 1, axis=0)

        # Get perspective transformation
        W = 480
        H = int(W // 1.6)
        M_perspective = cv2.getPerspectiveTransform(np.float32(approx_contour), np.float32([[0, 0], [0, H], [W, H], [W, 0]]))

        hh, ww = img.shape[:2]
        perspective_img = cv2.warpPerspective(img, M_perspective, (ww,hh))[0:H,0:W]



        # Tesseract text recognition
        # Crop the card to the UIN text
        uin_x_start = (int)(W*2/11)
        uin_width = (int)(W/4)
        uin_y_start = H - (int)(H/10)
        uin_height = (int)(H/10)
        img_uin = perspective_img[uin_y_start : uin_y_start + uin_height, uin_x_start : uin_x_start + uin_width]
        uin_text = extract_text_from_image(img_uin)
        uin_text = uin_text.replace("\n","")
        cv2.imshow("NetID Cropped", img_uin)

        # If the text identified is not the uin, try flipping the image
        if(not uin_text.isdigit() or len(uin_text) != 9):
            perspective_img = perspective_img[::-1, ::-1]
            img_uin = perspective_img[uin_y_start : uin_y_start + uin_height, uin_x_start : uin_x_start + uin_width]
            uin_text = extract_text_from_image(img_uin)
            uin_text = uin_text.replace("\n","")
            cv2.imshow("NetID Cropped flipped", img_uin)
            cv2.imshow('Perspectived', perspective_img)
            print("rotated")

            if(not uin_text.isdigit() or len(uin_text) != 9):
                uin_text = "not found"


        # If the text still does not resemble a uin, print no text detected
        if(not uin_text.isdigit() or len(uin_text) != 9):
            print("No UIN Detected")

        print(f"UIN: {uin_text}, length: {len(uin_text)}")

        cv2.putText(img_poly_contour, f"UIN: {uin_text}", (10, 30), cv2.FONT_HERSHEY_SIMPLEX, 0.75, (0, 255, 0), 2)

        cv2.imshow('Perspectived', perspective_img)
        cv2.imshow("Annotated", img_poly_contour)

        #out.write(img_poly_contour)

        # TESTING: user input for card detection accuracy
        # user_input = input("Enter 1 for correct identification of a card and 0 if not: ")
        # user_in = False
        # while(user_in == False):
        #     if user_input.isdigit():
        #         user_input_accurate_frames.append(int(user_input))
        #         user_in = True
        #     else:
        #         user_input = input("Invalid input, please enter a digit between 0-1: ")

        # TESTING record if UIN is correct
        if(uin_text == TRUE_UIN):
            print("Correct UIN")
            accurate_uin.append(1)
            false_uin.append(0)
        else:
            accurate_uin.append(0)
            if(len(uin_text) == len(TRUE_UIN) and uin_text.isdigit()):
                false_uin.append(1)
            else:
                false_uin.append(0)


    else:
        print("No card detected")


    # cv2.waitKey(0)
    cv2.waitKey(1)

#%%
cv2.waitKey(0)

cv2.destroyAllWindows()
```