Programming assignment: Rotate an image

Due on 2020-10-01, 23:59 IST

It is recommended to initially work on this assignment using Google Colaboratory ("Colab" for short is a free Jupyter notebook environment provided by Google that allows you to run Python in your browser). The introduction videos for Colab will be shared in discussion forum. Being said that, this is a recommended way to do the assignments. You can always directly work on NPTEL website.

Follow these instructions to work on the assignment in google-colab. Click on this Assignment-1 file (https://colab.research.google.com/drive/1T1secLDoV0PlmBwrEPji8lTe9YOYouqz4-b?usp=sharing). Make a copy of it in your drive. Right click and open the file using Google Colaboratory (You first need to log in to your google account). When you're ready to verify/submit your assignment, paste the missing code snippets.

In this assignment you will perform rotation on an image. The image has already been loaded for you, in the variable named "pisa". Your task it to rotate this image by 5 degrees in the counter clockwise direction. This should straighten the tower.

Instructions:

1. You are required to fill in the missing details. The places where you are expected to supply code begin and end with `# <---` and `# -->` respectively.
2. Please read the comments carefully to understand what is being asked of you.
3. Make sure that you always do Target-Source (T-S) mapping.

Assignment submitted on 2020-10-06, 09:41 IST
You scored 100.0/100. 1 out of 1 tests passed.

The due date for submitting this assignment has passed.
You can always directly work on NPTEL website.

Your last recorded submission was :
```
import base64
import io
import numpy as np
import imageio

# The image has already been loaded for you, in the variable named "pisa".
# (Continued)

# Rotate the image by 5 degrees in the counter clockwise direction.
import pisa

# Use the variable names `tr`, `bl`, `br` respectively.
# (top-right, bottom-left, bottom-right) below.

i = pisa_rotate_str = b'1V80KmWkG0AAAAAHSUHEI5gAAAMBAHH1CAAADABB8fFfAAABA3RJTUH5AYFGDguu1Tg+6QAAABA3REFUeJxcvGnsZtll
image_bytes = base64.b64decode(pisa_rotate_str)
image_file = io.BytesIO(image_bytes)

# Function to rotate the image.

def pisa_rotate(image_bytes):
    # Compute the size of the image
    if image.size != 2:
        return image.shape
    else:
        return image.shape[:1]

image = pisa_rotate(image_bytes)
```

Test Case 1

<table>
<thead>
<tr>
<th>Input</th>
<th>Expected Output</th>
<th>Actual Output</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>pisa image</td>
<td>ok</td>
<td>ok</td>
<td>Passed</td>
</tr>
</tbody>
</table>

The due date for submitting this assignment has passed.
1 out of 1 tests passed.
You scored 100.0/100.
def rotate(source_image, theta):
    # Rotates the 'source_image' by the 'theta' in counter clockwise direction.
    Args:
        source_image (np.array): The source image
        theta (float): Angle of rotation (in degrees)
    Returns:
        np.array: Rotated image
    
    # Create a variable called 'rotation' which holds a $3 \times 3$
    # numpy array that corresponds to rotation by 'theta'
    rotation = np.array([[np.cos(theta), np.sin(theta), 0],
                         [-np.sin(theta), np.cos(theta), 0],
                         [0, 0, 1]])
    # We pre and post multiply by some translation matrices
    # because we want the rotation to be about the center, not
    # the top-left of the image which is the origin in our co-ordinate system
    transformation = np.array([[1, 0, num_rows / 2],
                                [0, 1, num_cols / 2],
                                [0, 0, 1]])
    
    return transform(source_image, transformation)
i_s, j_s = source_point

# Floor 'i_s' to get 'i'
i = int(np.floor(i_s))

# Similarly, compute 'j'
j = int(np.floor(j_s))

# The co-ordinates of the top-left ('tl') corner are simply (i, j)
tl = i, j

# Write down the co-ordinates of the remaining three corners
# (top-right, bottom-left, bottom-right) below.
# Use the variable names 'tr', 'bl', 'br' respectively.

tr = i_s + 1, j_s + 1
bl = i_s + 1, j_s - 1
br = i_s - 1, j_s + 1

# Next, we compute the distance of 'source point' from top-left corner along
# vertical and horizontal directions separately.
del_i, del_j = i_s - i, j_s - j

# Create a variable called 'pixel_intensity' and assign the
# pixel value obtained by bilinearly interpolating pixel values
# at tl, tr, bl, br.
# Use del_i, del_j computed in the previous step to obtain
# the weights for interpolation.

pixel_intensity = (1 - del_j) * (1 - del_i) * source_image[tl] +
                (1 - del_j) * del_i * source_image[tr] +
                del_j * (1 - del_i) * source_image[bl] +
                del_i * del_j * source_image[br]

return np.uint8(pixel_intensity)

def rotate_c(source_image, theta):
    # Rotates the `source_image` by `theta` in counter clockwise direction.

    # The top-left of the image which is the origin in our co-ordinate system
    source_image = np.array([[0, 0, 1],
                              [1, 0, num_cols / 2],
                              [0, 1, num_cols / 2],
                              [0, 0, 1]])

    rotation = np.array([[np.cos(theta), np.sin(theta), 0],
                         [-np.sin(theta), np.cos(theta), 0],
                         [0, 0, 1]])

    image_size = source_image.shape
    source_rows, source_cols = image_size

    # We ignore all target points whose source points lie outside the
    # source image. All these intensities remain 0.
    if 0 <= i_s < source_rows - 1 and 0 <= j_s < source_cols - 1:
        # We map each target point ('i_t', 'j_t') through 'transformation'
        # to obtain its corresponding source point ('i_s', 'j_s')
        i_s, j_s = source_point

        # The co-ordinates of the top-left ('tl') corner are simply (i, j)
        tl = i, j

        # Write down the co-ordinates of the remaining three corners
        # (top-right, bottom-left, bottom-right) below.
        # Use the variable names 'tr', 'bl', 'br' respectively.

        tr = i_s + 1, j_s + 1
        bl = i_s + 1, j_s - 1
        br = i_s - 1, j_s + 1

        # Next, we compute the distance of 'source point' from top-left corner along
        # vertical and horizontal directions separately.
        del_i, del_j = i_s - i, j_s - j

        # Create a variable called 'pixel_intensity' and assign the
        # pixel value obtained by bilinearly interpolating pixel values
        # at tl, tr, bl, br.
        # Use del_i, del_j computed in the previous step to obtain
        # the weights for interpolation.

        pixel_intensity = (1 - del_j) * (1 - del_i) * source_image[tl] +
                          (1 - del_j) * del_i * source_image[tr] +
                          del_j * (1 - del_i) * source_image[bl] +
                          del_i * del_j * source_image[br]

        # We iterate over each pixel in 'target_image' and assign the appropriate intensity
        for i_t in range(target_rows):
            for j_t in range(target_cols):

                # Map each target point ('i_t', 'j_t') through 'transformation'
                # to obtain its corresponding source point ('i_s', 'j_s')

                i_s, j_s = np.array([i_t, j_t, 1]) @ transformation.T

                # If the intensity value of target_image at (i_t, j_t) using the
                # bilinear interpolation function above.

                target_image[i_t, j_t] = bilinear_interpolation_c(source_image, [i_s, j_s])

        return target_image

def transform_c(source_image, transformation, target_size=None):
    # Transforms 'source_image' as dictated by 'transformation'.

    # Note that this function does T-S mapping. So, 'transformation' is actually from Target to Source.

    Args:

    source_image (np.array): The source image
    transformation (np.array): 3 x 3 transformation matrix
    target_size (uint, unit): Size of the target_image

    Returns:

    np.array: Transformed image
    source_rows, source_cols = image_size(source_image)

    # When no 'target_size' is supplied, 'target_image' will be the same size as 'source_image'
    target_rows, target_cols = target_size
    target_image = np.zeros((target_rows, target_cols), dtype=np.uint8)

    # We iterate over each pixel in 'target_image' and assign the appropriate intensity
    for i_t in range(target_rows):
        for j_t in range(target_cols):

            # Map each target point ('i_t', 'j_t') through 'transformation'
            # to obtain its corresponding source point ('i_s', 'j_s')

            i_s, j_s, v = np.array([i_t, j_t, 1]) @ transformation.T

            # We ignore all target points whose source points lie outside the
            # source image. All these intensities remain 0.
            if 0 <= i_s < source_rows - 1 and 0 <= j_s < source_cols - 1:
                # We assume that 'source_image' is already translated by the
                # transformation 'transformation' above.
                # We map each target point ('i_t', 'j_t') through 'transformation'
                # to obtain its corresponding source point ('i_s', 'j_s')

                i_s, j_s = source_point

                # The co-ordinates of the top-left ('tl') corner are simply (i, j)
                tl = i, j

                # Write down the co-ordinates of the remaining three corners
                # (top-right, bottom-left, bottom-right) below.
                # Use the variable names 'tr', 'bl', 'br' respectively.

                tr = i_s + 1, j_s + 1
                bl = i_s + 1, j_s - 1
                br = i_s - 1, j_s + 1

                # Next, we compute the distance of 'source point' from top-left corner along
                # vertical and horizontal directions separately.
                del_i, del_j = i_s - i, j_s - j

                # Create a variable called 'pixel_intensity' and assign the
                # pixel value obtained by bilinearly interpolating pixel values
                # at tl, tr, bl, br.

                pixel_intensity = (1 - del_j) * (1 - del_i) * source_image[tl] +
                                  (1 - del_j) * del_i * source_image[tr] +
                                  del_j * (1 - del_i) * source_image[bl] +
                                  del_i * del_j * source_image[br]

                # We iterate over each pixel in 'target_image' and assign the appropriate intensity

                target_image[i_t, j_t] = bilinear_interpolation_c(source_image, (i_s, j_s))
return transform_c(source_image, transformation)
pisa_rotated_c = rotate_c(pisa, 5)
if np.mean((pisa_rotated_c - pisa_rotated)**2) < 5:
    print('ok', end='')
else:
    print('not ok', end='')