

X


<https://swayam.gov.in>

https://swayam.gov.in/nc_details/NPTEL

reviewer5@nptel.iitm.ac.in

[NPTEL \(https://swayam.gov.in/explorer?ncCode=NPTEL\)](https://swayam.gov.in/explorer?ncCode=NPTEL) » [Image Signal Processing \(course\)](#)
[Announcements \(announcements\)](#)
[About the Course \(preview\)](#)
[Ask a Question \(forum\)](#)
[Progress \(student/home\)](#)
[Mentor \(student/mentor\)](#)

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/1tSscLDV9PlmBwfEPjoi81e9YOUyqz4-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 is 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.

Private Test cases used for evaluation

Test Case 1

Input	Expected Output	Actual Output	Status
pisa image	ok	ok	Passed

The due date for submitting this assignment has passed.

1 out of 1 tests passed.

You scored 100.0/100.

Assignment submitted on 2020-10-06, 09:41 IST

Your last recorded submission was :

```

1 import base64
2 import io
3
4 pisa_rotate_str = b'ivB0Rw0KGgoAAAANSUHEUGAAAM8AAAHICAAAAABqr8fFAAAB3RJTUUH5AYFDgou1Tg+6QAAIABJREFUeJxcvGmsZt11
5 image_bytes = base64.b64decode(pisa_rotate_str)
6 image_file = io.BytesIO(image_bytes)
7
8
9
10 import numpy as np
11 import imageio
12
13 def image_size(image):
14     if image.ndim == 2:
15         return image.shape
16     else:
17         return image.shape[:-1]
18
19
20 def bilinear_interpolation(source_image, source_point):
21     """
22     Computes the intensity at `source_point` by bilinearly interpolating
23     intensities in the immediate 2 X 2 neighborhood of the `source_point`.
24
25     Args:
26         source_image (np.array): The source image
27         source_point (float, float): The source point
28
29     Returns:
30         uint8: Pixel intensity at source_point
31
32         """
33     i_s, j_s = source_point
34     # Floor `i_s` to get `i`
35     i = int(np.floor(i_s))
36
37     # Similarly, compute `j`
38     # <---
39     j = int(np.floor(j_s))
40     # --->
41
42     # The co-ordinates of the top-left (`tl`) corner are simply (i, j)
43     tl = i, j
44
45     # Write down the co-ordinates of the remaining three corners
46     # (top-right, bottom-left, bottom-right) below.
47     # Use the variable names `tr`, `bl`, `br` respectively.
48
49     # <---
50     tr = i, j + 1

```

Course outline

How does an NPTEL online course work?

Week 1

- Course Introduction (unit? unit=17&lesson=29)
- Applications of Image processing (unit? unit=17&lesson=30)
- Applications of Image processing (continued) (unit? unit=17&lesson=31)
- Basics of Images (unit? unit=17&lesson=32)
- Shot Noise (unit? unit=17&lesson=33)
- Geometric Transformations (unit?unit=17&lesson=34)
- Geometric Transformations (continued) (unit? unit=17&lesson=35)
- Bilinear Interpolation (unit? unit=17&lesson=36)
- Programming assignment: Translate an image (/noc20_ee83/progassignment?name=113)
- Programming assignment: Rotate an image (/noc20_ee83/progassignment?name=115)
- Programming assignment: Scale an image (/noc20_ee83/progassignment?name=117)
- Image Signal Processing : Week 1 Feedback Form (unit?unit=17&lesson=146)
- Week 1 Assignments solutions (unit? unit=17&lesson=162)
- Lecture materials (unit? unit=17&lesson=165)

Week 2

Week 3

Week 4

Week 5

Week 6

Week 7

```

51 bl = i + 1, j
52 br = i + 1, j + 1
53 # --->
54
55 # Next, we compute the distance of `source_point` from top-left corner along
56 # vertical and horizontal directions separately.
57 del_i, del_j = i_s - i, j_s - j
58
59 # Create a variable called `pixel_intensity` and assign the
60 # pixel value obtained by bilinearly interpolating pixel values
61 # at tl, tr, bl, br.
62 # Use `del_i`, `del_j` computed in the previous step to obtain
63 # the weights for interpolation.
64 # <---
65 pixel_intensity = (1 - del_i) * (1 - del_j) * source_image[tl] + \
66                 (1 - del_i) * del_j * source_image[tr] + \
67                 del_i * (1 - del_j) * source_image[bl] + \
68                 del_i * del_j * source_image[br]
69 # --->
70 return np.uint8(pixel_intensity)
71
72 def transform(source_image, transformation, target_size=None):
73     """
74     Transforms `source_image` as dictated by `transformation`.
75
76     Note that this function does T-S mapping. So, `transformation` is actually from Target to Source.
77
78     Args:
79
80         source_image (np.array): The source image
81         transformation (np.array): 3 x 3 transformation matrix
82         target_size (uint, uint): Size of the target_image
83
84     Returns:
85     """
86     np.array: Transformed image
87     source_rows, source_cols = image_size(source_image)
88
89     # When no `target_size` is supplied, `target_image` will be the same size as `source_image`
90     target_rows, target_cols = target_size if target_size else (source_rows, source_cols)
91     target_image = np.zeros((target_rows, target_cols), dtype=np.uint8)
92
93     # We iterate over each pixel in `target_image` and assign the appropriate intensity
94     for i_t in range(target_rows):
95         for j_t in range(target_cols):
96
97             # Map each target point `(i_t`, `j_t`) through `transformation`
98             # to obtain its corresponding source_point `(i_s`, `j_s`)
99             # <---
100            i_s, j_s, v = np.array([i_t, j_t, 1]) @ transformation.T
101            i_s, j_s = i_s / v, j_s / v
102            # --->
103
104            # We ignore all target points whose source points lie outside the
105            # source image. All these intensities remain 0.
106            if 0 <= i_s < source_rows - 1 and 0 <= j_s < source_cols - 1:
107                # Assign the intensity value of target_image at `(i_t, j_t)` using the
108                # bilinear interpolation function above.
109                # <---
110                target_image[i_t, j_t] = bilinear_interpolation(source_image, (i_s, j_s))
111                # --->
112
113     return target_image
114
115 def rotate(source_image, theta):
116     """
117     Rotates the `source_image` by `theta` in counter clockwise direction.
118
119     Args:
120
121         source_image (np.array): The source image
122         theta (float): Angle of rotation (in degrees)
123
124     Returns:
125     """
126     np.array: Rotated image
127
128     theta *= np.pi / 180
129     num_rows, num_cols = image_size(source_image)
130
131     # Create a variable called `rotation` which holds a 3 x 3
132     # numpy array that corresponds to rotation by `theta`
133     # Note that this matrix should be T-S,
134     # so it would be the inverse of what you might think of for
135     # an S-T rotation matrix.
136     # <---
137     rotation = np.array([[np.cos(theta), np.sin(theta), 0],
138                         [-np.sin(theta), np.cos(theta), 0],
139                         [0, 0, 1]])
140     # --->
141
142     # We pre and post multiply by some translation matrices
143     # because we want the rotation to be about the center, not
144     # the top-left of the image which is the origin in our co-ordinate system
145     transformation = np.array([[1, 0, num_rows / 2],
146                               [0, 1, num_cols / 2],
147                               [0, 0, 1]]) @ \
148                       rotation @ \
149                       np.array([[1, 0, -num_rows / 2],
150                               [0, 1, -num_cols / 2],
151                               [0, 0, 1]])
152     return transform(source_image, transformation)
153
154 pisa = imageio.imread(image_file, format='PNG')
155 # Call the `rotate` function above with the right parameters.
156 # <---
157 pisa_rotated = rotate(pisa, 5)
158 # --->
159
160 def bilinear_interpolation_c(source_image, source_point):
161     """
162     Computes the intensity at `source_point` by bilinearly interpolating
163     intensities in the immediate 2 X 2 neighborhood of the `source_point`.
164
165     Args:
166         source_image (np.array): The source image
167         source_point (float, float): The source point
168
169     Returns:
170     """
171     uint8: Pixel intensity at source_point

```

```

172 i_s, j_s = source_point
173
174 # Floor `i_s` to get `i`
175 i = int(np.floor(i_s))
176
177 # Similarly, compute `j`
178 # <---
179 j = int(np.floor(j_s))
180 # --->
181
182 # The co-ordinates of the top-left (`tl`) corner are simply (i, j)
183 tl = i, j
184
185 # Write down the co-ordinates of the remaining three corners
186 # (top-right, bottom-left, bottom-right) below.
187 # Use the variable names `tr`, `bl`, `br` respectively.
188
189 # <---
190 tr = i, j + 1
191 bl = i + 1, j
192 br = i + 1, j + 1
193 # --->
194
195 # Next, we compute the distance of `source_point` from top-left corner along
196 # vertical and horizontal directions separately.
197 del_i, del_j = i_s - i, j_s - j
198
199 # Create a variable called `pixel intensity` and assign the
200 # pixel value obtained by bilinearly interpolating pixel values
201 # at `tl`, `tr`, `bl`, `br`.
202 # Use `del_i`, `del_j` computed in the previous step to obtain
203 # the weights for interpolation.
204 # <---
205 pixel_intensity = (1 - del_i) * (1 - del_j) * source_image[tl] + \
206                 (1 - del_i) * del_j * source_image[tr] + \
207                 del_i * (1 - del_j) * source_image[bl] + \
208                 del_i * del_j * source_image[br]
209 # --->
210 return np.uint8(pixel_intensity)
211
212 def transform_c(source_image, transformation, target_size=None):
213     """
214     Transforms `source_image` as dictated by `transformation`.
215
216     Note that this function does T-S mapping. So, `transformation` is actually from Target to Source.
217
218     Args:
219
220         source_image (np.array): The source image
221         transformation (np.array): 3 x 3 transformation matrix
222         target_size (uint, uint): Size of the target_image
223
224     Returns:
225     """
226     source_rows, source_cols = image_size(source_image)
227
228     # When no `target_size` is supplied, `target_image` will be the same size as `source_image`
229     target_rows, target_cols = target_size if target_size else (source_rows, source_cols)
230     target_image = np.zeros((target_rows, target_cols), dtype=np.uint8)
231
232     # We iterate over each pixel in `target_image` and assign the appropriate intensity
233     for i_t in range(target_rows):
234         for j_t in range(target_cols):
235
236             # Map each target point (`i_t`, `j_t`) through `transformation`
237             # to obtain its corresponding source_point (`i_s`, `j_s`)
238             # <---
239             i_s, j_s, v = np.array([i_t, j_t, 1]) @ transformation.T
240             i_s, j_s = i_s / v, j_s / v
241             # --->
242
243             # We ignore all target points whose source points lie outside the
244             # source image. All these intensities remain 0.
245             if 0 <= i_s < source_rows - 1 and 0 <= j_s < source_cols - 1:
246                 # Assign the intensity value of target image at `(i_t, j_t)` using the
247                 # bilinear interpolation function above.
248                 # <---
249                 target_image[i_t, j_t] = bilinear_interpolation_c(source_image, (i_s, j_s))
250                 # --->
251
252     return target_image
253
254 def rotate_c(source_image, theta):
255     """
256     Rotates the `source_image` by `theta` in counter clockwise direction.
257
258     Args:
259
260         source_image (np.array): The source image
261         theta (float): Angle of rotation (in degrees)
262
263     Returns:
264     """
265     theta *= np.pi / 180
266     num_rows, num_cols = image_size(source_image)
267
268     # Create a variable called `rotation` which holds a 3 x 3
269     # numpy array that corresponds to rotation by `theta`
270     # Note that this matrix should be T-S,
271     # so it would be the inverse of what you might think of for
272     # an S-T rotation matrix.
273     # <---
274     rotation = np.array([[np.cos(theta), np.sin(theta), 0],
275                          [-np.sin(theta), np.cos(theta), 0],
276                          [0, 0, 1]])
277     # --->
278
279     # We pre and post multiply by some translation matrices
280     # because we want the rotation to be about the center, not
281     # the top-left of the image which is the origin in our co-ordinate system
282     transformation = np.array([[1, 0, num_rows / 2],
283                               [0, 1, num_cols / 2],
284                               [0, 0, 1]]) @ \
285                       rotation @ \
286                       np.array([[1, 0, -num_rows / 2],
287                               [0, 1, -num_cols / 2],
288                               [0, 0, 1]])

```

```
293     return transform_c(source_image, transformation)
294
295 pisa_rotated_c = rotate_c(pisa, 5)
296
297 if np.mean((pisa_rotated_c - pisa_rotated)**2) < 5:
298     print('ok', end='')
299 else:
300     print('not ok', end='')
301
```

