


<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: Occlusion Detection

Due on 2020-10-01, 23:59 IST
[Click here \(https://drive.google.com/file/d/1Tstz2O1tHM-H6_jqPSIQcqJEmkRXNPLE/view?usp=sharing\)](https://drive.google.com/file/d/1Tstz2O1tHM-H6_jqPSIQcqJEmkRXNPLE/view?usp=sharing) for View the Question

Course outline

How does an NPTEL online course work?

Week 1

Week 2

- Geometric Transformations (continued) (unit? unit=18&lesson=37)
- Projective Transformation (unit?unit=18&lesson=38)
- Homography (unit? unit=18&lesson=39)
- Homography - Special cases (unit?unit=18&lesson=40)
- Computing Homography (unit?unit=18&lesson=41)
- RANSAC (unit? unit=18&lesson=42)
- Programming assignment: Occlusion Detection (/noc20_ee83/progassignment?name=118)**

- Image Signal Processing : Week 2 Feedback Form (unit?unit=18&lesson=147)

- Week 2 Assignment solutions (unit? unit=18&lesson=163)

- Lecture materials (unit? unit=18&lesson=164)

Week 3

Week 4

Week 5

Week 6

Week 7

Week 8

Week 9

Week 10

Week 11

Week 12

Tutorials

Download Videos

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-2 (<https://drive.google.com/file/d/1tSkNrpLZ9-z2fuwyFI9s-ncdZ-VS0vRF/view?usp=sharing>) file.
- 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.

Private Test cases used for evaluation	Input	Expected Output	Actual Output	Status
Test Case 1	occlusion detection	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, 09:46 IST

Your last recorded submission was :

```

1 import base64
2 import io
3
4 image1_str = b'ivBORw0KGgoAAAANSUHEUgAAAgAAAAEoCAAAAAALBBbgAAAAAB3RJTUUH5AYEAYkTg53xrwAAIABJREFUeJw8vEmvZdmVhVZ9a
5 image2_str = b'ivBORw0KGgoAAAANSUHEUgAAALYAAAIIFCAAAAAA110yaAAAAAB3RJTUUH5AYEAYkUHfLkDAAAIABJREFUeJzsvXmw7dLV37fw2
6
7 image_bytes1 = base64.b64decode(image1_str)
8 image_bytes2 = base64.b64decode(image2_str)
9
10 image_file1 = io.BytesIO(image_bytes1)
11 image_file2 = io.BytesIO(image_bytes2)
12 import imageio
13 import numpy as np
14
15 def find_rt(points):
16     """
17     Computes the transformation (involving only rotation & translation) matrix
18     given 2 pairs of corresponding points.
19
20     Args:
21         points (np.array): 2 x 2 x 2 np.array that holds the point correspondences.
22         Each slice holds points in one image. So, points[:, :, 0] has points
23         in IMG1, while points[:, :, 1] has points in IMG2
24
25     Returns:
26         "" np.array: Transformation matrix
27
28     """
29     A = np.zeros((4, 4), dtype=np.float)
30     A[np.arange(4), [2, 3, 2, 3]] = 1
31     A[0::2, 0:2] = points[0]
32     A[1::2, 0:2] = np.flip(points[0], 1)
33     A[0::2, 1] *= -1
34     b = points[1].reshape(-1, order='C')
35     x = np.linalg.solve(A, b)
36     T = np.array([[x[0], -x[1], x[2]],
37                 [x[1], x[0], x[3]],
38                 [0, 0, 1]])
39     return T
40
41 def image_size(image):
42     if image.ndim == 2:
43         return image.shape
44     else:
45         return image.shape[:-1]
46
47 def bilinear_interpolation(source_image, source_point):
48     """
49     Computes the intensity at `source_point` by bilinearly interpolating
50     intensities in the immediate 2 X 2 neighborhood of the `source_point`.
51
52     Args:
53         source_image (np.array): The source image
54         source_point (float, float): The source point

```

Live Session

December 8 Programming
test - Session 1(10AM-11AM)December 8 programming
test - Session 2 (8PM to
9PM)

```

55
56 Returns:
57 """ uint8: Pixel intensity at source_point
58
59 i_s, j_s = source_point
60
61 # Floor `i_s` to get `i`
62 i = int(np.floor(i_s))
63
64 # Similarly, compute `j`
65 # <---
66 j = int(np.floor(j_s))
67 # --->
68
69 # The co-ordinates of the top-left (`tl`) corner are simply (i, j)
70 tl = i, j
71
72 # Write down the co-ordinates of the remaining three corners
73 # (top-right, bottom-left, bottom-right) below.
74 # Use the variable names `tr`, `bl`, `br` respectively.
75
76 # <---
77 tr = i, j + 1
78 bl = i + 1, j
79 br = i + 1, j + 1
80 # --->
81
82 # Next, we compute the distance of `source_point` from top-left corner along
83 # vertical and horizontal directions separately.
84 del_i, del_j = i_s - i, j_s - j
85
86 # Create a variable called `pixel_intensity` and assign the
87 # pixel value obtained by bilinearly interpolating pixel values
88 # at `tl`, `tr`, `bl`, `br`.
89 # Use `del_i`, `del_j` computed in the previous step to obtain
90 # the weights for interpolation.
91 # <---
92 pixel_intensity = (1 - del_i) * (1 - del_j) * source_image[tl] + \
93                 (1 - del_i) * del_j * source_image[tr] + \
94                 del_i * (1 - del_j) * source_image[bl] + \
95                 del_i * del_j * source_image[br]
96 # --->
97
98 return np.uint8(pixel_intensity)
99
100 def transform(source_image, transformation, target_size=None):
101 """
102 Transforms `source_image` as dictated by `transformation`.
103
104 Note that this function does T-S mapping. So, `transformation` is actually from Target to Source.
105
106 Args:
107     source_image (np.array): The source image
108     transformation (np.array): 3 x 3 transformation matrix
109     target_size (uint, uint): Size of the target_image
110
111 Returns:
112     np.array: Transformed image
113
114 source_rows, source_cols = image_size(source_image)
115
116 # When no `target_size` is supplied, `target_image` will be the same size as `source_image`
117 target_rows, target_cols = target_size if target_size else (source_rows, source_cols)
118 target_image = np.zeros((target_rows, target_cols), dtype=np.uint8)
119
120 # We iterate over each pixel in `target_image` and assign the appropriate intensity
121 for i_t in range(target_rows):
122     for j_t in range(target_cols):
123         # Map each target point (`i_t`, `j_t`) through `transformation`
124         # to obtain its corresponding source_point (`i_s`, `j_s`)
125         # <---
126         i_s, j_s, v = np.array([i_t, j_t, 1]) @ transformation.T
127         i_s, j_s = i_s / v, j_s / v
128         # --->
129
130         # We ignore all target points whose source points lie outside the
131         # source image. All these intensities remain 0.
132         if 0 <= i_s < source_rows - 1 and 0 <= j_s < source_cols - 1:
133             # Assign the intensity value of target image at `(i_t, j_t)` using the
134             # bilinear interpolation function above.
135             # <---
136             target_image[i_t, j_t] = bilinear_interpolation(source_image, (i_s, j_s))
137             # --->
138
139 return target_image
140
141
142 # Create a variable called `correspondences` and fill
143 # it point correspondences given in the table above.
144 # Make sure that you fill it in the format expected
145 # by `find_rt` (read its documentation)
146 # <---
147 correspondences = np.array([
148     [[29, 124],
149     [[157, 372]],
150     [[93, 248],
151     [[328, 399]]
152 ])
153 # --->
154 T = find_rt(correspondences)
155
156 img1 = imageio.imread(image_file1, format='PNG')
157 img2 = imageio.imread(image_file2, format='PNG')
158
159 # Align `img2` with `img1` using the `T` above and the `transform` function.
160 # Make sure that the aligned image is the same size as `img1`.
161 # <---
162 img2_aligned = transform(img2, T, img1.shape)
163 # --->
164
165 # Subtract the aligned img2 and img1 to notice any changes.
166 # Name the difference as `diff_img`.
167 # <---
168 diff_img = np.abs(img2_aligned.astype(np.float) - img1.astype(np.float)).astype(np.uint8)
169 # --->
170
171 def bilinear_interpolation_c(source_image, source_point):
172 """
173 Computes the intensity at `source_point` by bilinearly interpolating

```

```

176 intensities in the immediate 2 X 2 neighborhood of the `source_point`.
177
178 Args:
179     source_image (np.array): The source image
180     source_point (float, float): The source point
181
182 Returns:
183     """ uint8: Pixel intensity at source_point
184
185     i_s, j_s = source_point
186
187     # Floor `i_s` to get `i`
188     i = int(np.floor(i_s))
189
190     # Similarly, compute `j`
191     # <---
192     j = int(np.floor(j_s))
193     # --->
194
195     # The co-ordinates of the top-left (`tl`) corner are simply (i, j)
196     tl = i, j
197
198     # Write down the co-ordinates of the remaining three corners
199     # (top-right, bottom-left, bottom-right) below.
200     # Use the variable names `tr`, `bl`, `br` respectively.
201
202     # <---
203     tr = i, j + 1
204     bl = i + 1, j
205     br = i + 1, j + 1
206     # --->
207
208     # Next, we compute the distance of `source_point` from top-left corner along
209     # vertical and horizontal directions separately.
210     del_i, del_j = i_s - i, j_s - j
211
212     # Create a variable called `pixel_intensity` and assign the
213     # pixel value obtained by bilinearly interpolating pixel values
214     # at tl, tr, bl, br.
215     # Use `del_i`, `del_j` computed in the previous step to obtain
216     # the weights for interpolation.
217     # <---
218     pixel_intensity = (1 - del_i) * (1 - del_j) * source_image[tl] + \
219                     (1 - del_i) * del_j * source_image[tr] + \
220                     del_i * (1 - del_j) * source_image[bl] + \
221                     del_i * del_j * source_image[br]
222     # --->
223
224     return np.uint8(pixel_intensity)
225
226 def transform_c(source_image, transformation, target_size=None):
227     """
228     Transforms `source_image` as dictated by `transformation`.
229
230     Note that this function does T-S mapping. So, `transformation` is actually from Target to Source.
231
232     Args:
233
234     source_image (np.array): The source image
235     transformation (np.array): 3 x 3 transformation matrix
236     target_size (uint, uint): Size of the target_image
237
238     Returns:
239     """ np.array: Transformed image
240
241     source_rows, source_cols = image_size(source_image)
242
243     # When no `target_size` is supplied, `target_image` will be the same size as `source_image`
244     target_rows, target_cols = target_size if target_size else (source_rows, source_cols)
245     target_image = np.zeros((target_rows, target_cols), dtype=np.uint8)
246
247     # We iterate over each pixel in `target_image` and assign the appropriate intensity
248     for i_t in range(target_rows):
249         for j_t in range(target_cols):
250
251             # Map each target point (`i_t`, `j_t`) through `transformation`
252             # to obtain its corresponding source_point (`i_s`, `j_s`)
253             # <---
254             i_s, j_s, v = np.array([i_t, j_t, 1]) @ transformation.T
255             i_s, j_s = i_s / v, j_s / v
256             # --->
257
258             # We ignore all target points whose source points lie outside the
259             # source image. All these intensities remain 0.
260             if 0 <= i_s < source_rows - 1 and 0 <= j_s < source_cols - 1:
261                 # Assign the intensity value of target image at `(i_t, j_t)` using the
262                 # bilinear interpolation function above.
263
264                 # <---
265                 target_image[i_t, j_t] = bilinear_interpolation_c(source_image, (i_s, j_s))
266                 # --->
267
268     return target_image
269
270
271 # Align `img2` with `img1` using the `T` above and the `transform` function.
272 # Make sure that the aligned image is the same size as `img1`.
273 # <---
274 img2_aligned_c = transform_c(img2, T, img1.shape)
275 # --->
276
277 # Subtract the aligned img2 and img1 to notice any changes.
278 # Name the difference as `diff_img`.
279 # <---
280 diff_img_c = np.abs(img2_aligned_c.astype(np.float) - img1.astype(np.float)).astype(np.uint8)
281 # --->
282
283 if np.mean((diff_img_c - diff_img)**2) < 5:
284     print('ok', end='\r')
285 else:
286     print('not ok', end='')

```

