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) » 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) for View the Question .

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 :

```
import base64
import io

image1_str = b'iVBORw0KGgoAAAANSUhEUgAAAgAAAAEoCAAAAAALBBbgAAAAB3RJTUUH5AYEAykTg53xrwAAIABJREFUeJw8vEmvZdmVHvZ9a
image2_str = b'iVBORw0KGgoAAAANSUhEUgAAAlYAAAIFCAAAAAA11OyaAAAAB3RJTUUH5AYEAykUHflkDAAAIABJREFUeJzsvXmw7dlV37fW2

image_bytes1 = base64.b64decode(image1_str)
image_bytes2 = base64.b64decode(image2_str)

image_file1 = io.BytesIO(image_bytes1)
image_file2 = io.BytesIO(image_bytes2)
import imageio
import numpy as np

def find_rt(points):
    """
    Computes the transformation (involving only rotation & translation) matrix
    given 2 pairs of corresponding points.

    Args:
        points (np.array): 2 x 2 x 2 np.array that holds the point correspondences.
                           Each slice holds points in one image. So, points[:, :, 0] has points
                           in IMG1, while points[:, :, 1] has points in IMG2

    Returns:
        np.array: Transformation matrix
    """

    A = np.zeros((4, 4), dtype=np.float)
    A[np.arange(4), [2, 3, 2, 3]] = 1
    A[0::2, 0:2] = points[0]
    A[1::2, 0:2] = np.flip(points[0], 1)
    A[0::2, 1] *= -1
    b = points[1].reshape(-1, order='C')
    x = np.linalg.solve(A, b)
    T = np.array([[x[0], -x[1], x[2]],
                  [x[1], x[0], x[3]],
                  [0, 0, 1]])
    return T

def image_size(image):
    if image.ndim == 2:
        return image.shape
    else:
        return image.shape[:-1]

def bilinear_interpolation(source_image, source_point):
    """
    Computes the intensity at `source_point` by bilinearly interpolating
    intensities in the immediate 2 X 2 neighborhood of the `source_point`.

    Args:
        source_image (np.array): The source image
        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)

```
    Returns:
        uint8: Pixel intensity at source_point
    """
    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, j + 1
    bl = i + 1, j
    br = i + 1, j + 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_i) * (1 - del_j) * source_image[tl] + \
                      (1 - del_i) * del_j * source_image[tr] + \
                      del_i * (1 - del_j) * source_image[bl] + \
                      del_i * del_j * source_image[br]
    # --->

    return np.uint8(pixel_intensity)

def transform(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, uint): 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 if target_size else (source_rows, source_cols)
    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
            i_s, j_s = i_s / v, j_s / v
            # --->

            # 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:
                # Assign 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(source_image, (i_s, j_s))
                # --->

    return target_image

# Create a variable called `correspondences` and fill
# it point correspondences given in the table above.
# Make sure that you fill it in the format expected
# by `find_rt` (read its documentation)
# <---
correspondences = np.array([
    [[29, 124],
     [157, 372]],
    [[93, 248],
     [328, 399]]
])
# --->
T = find_rt(correspondences)

img1 = imageio.imread(image_file1, format='PNG')
img2 = imageio.imread(image_file2, format='PNG')


# Align `img2` with `img1` using the `T` above and the `transform` function.
# Make sure that the aligned image is the same size as `img1`.
# <---
img2_aligned = transform(img2, T, img1.shape)
# --->

# Subtract the aligned img2 and img1 to notice any changes.
# Name the difference as `diff_img`.
# <---
diff_img = np.abs(img2_aligned.astype(np.float) - img1.astype(np.float)).astype(np.uint8)
# --->
def bilinear_interpolation_c(source_image, source_point):
    """
    Computes the intensity at `source_point` by bilinearly interpolating
```

```
    intensities in the immediate 2 X 2 neighborhood of the `source_point`.

    Args:
        source_image (np.array): The source image
        source_point (float, float): The source point

    Returns:
        uint8: Pixel intensity at source_point
    """
    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, j + 1
    bl = i + 1, j
    br = i + 1, j + 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_i) * (1 - del_j) * source_image[tl] + \
                      (1 - del_i) * del_j * source_image[tr] + \
                      del_i * (1 - del_j) * source_image[bl] + \
                      del_i * del_j * source_image[br]
    # --->

    return np.uint8(pixel_intensity)

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, uint): 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 if target_size else (source_rows, source_cols)
    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
            i_s, j_s = i_s / v, j_s / v
            # --->

            # 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:
                # Assign 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


# Align `img2` with `img1` using the `T` above and the `transform` function.
# Make sure that the aligned image is the same size as `img1`.
# <---
img2_aligned_c = transform_c(img2, T, img1.shape)
# --->

# Subtract the aligned img2 and img1 to notice any changes.
# Name the difference as `diff_img`.
# <---
diff_img_c = np.abs(img2_aligned_c.astype(np.float) - img1.astype(np.float)).astype(np.uint8)
# --->

if np.mean((diff_img_c - diff_img)**2) < 5:
    print('ok',end='')
else:
    print('not ok', end='')
```