
Programming assignment: Occlusion Detection

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

Click here (https://drive.google.com/file/d/1Tstz201iHM-H6_jqPSLcqcjEmkRXNPIe/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/1SkNrpLZ9-z2fuwyF19s-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.

Assignment submitted on 2020-10-06, 09:46 IST

Your last recorded submission was:

```python
import numpy as np

# Computes the transformation (involving only rotation & translation) matrix given 2 pairs of corresponding points.

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

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'.

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

    # ... continue the code snippet
```

The due date for submitting this assignment has passed.

1 out of 1 tests passed.

You scored 100.0/100.
def transform(source_image, transformation, target_size=None):

    type(source_image)  # np.ndarray
    type(transformation)  # np.ndarray
    shape(source_image)  # (height, width)
    shape(transformation)  # (3, 3)

    # Vertical and horizontal directions separately.
    source_rows = shape(source_image)[0]
    source_cols = shape(source_image)[1]
    target_rows = shape(target_image)[0]
    target_cols = shape(target_image)[1]

    # Find the point correspondences.
    correspondences = find_rt(source_points, target_points)
    # Ensure the format expected by `find_rt`.
    correspondences = np.array(correspondences, dtype=np.float32)

    # Transforms `source_image` as dictated by `transformation`.
    # `transformation` is actually from Target to Source.
    source_point = np.array([i_s, j_s], dtype=np.int32)
    np.uint8

    # 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.
    tr = i + 1, j + 1
    br = i + 1, j - 1
    bl = i - 1, j + 1

    # Next, 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 `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[br] +
        del_i * del_j * source_image[bl]

    # ...-
    return np.uint8(pixel_intensity)

def bilateral_interpolation_c(source_image, source_point):
    # Computes the intensity at `source_point` by bilinearly interpolating
    # the weights for interpolation.
    del_i = source_point[0] - int(source_point[0])
    del_j = source_point[1] - int(source_point[1])

    tr = int(source_point[0] + del_i), int(source_point[1] + del_j)
    bl = int(source_point[0] + del_i), int(source_point[1] - del_j)
    br = int(source_point[0] - del_i), int(source_point[1] + del_j)
    tl = int(source_point[0] - del_i), int(source_point[1] - del_j)

    i, j = source_point[0], source_point[1]
    hi, hj = del_i * source_image[tr] + del_j * source_image[tl] +
        (1 - del_i) * source_image[bl] + (1 - del_j) * source_image[br]

    # ...-
    return np.uint8(hi + hj)
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:
- target_size (uint, uint): Size of the target_image

Example:
```python
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.

```python
def transform_c(source_image, transformation, target_size=None):
    # 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_intensity = [(1 - del_j) * (1 - del_i) * source_image[tl] +
                      (1 - del_j) * del_i * source_image[tr] +
                      del_i * (1 - del_j) * source_image[bl] +
                      del_i * del_j * source_image[br]]
    # 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.
    tr = i + 1, j + 1
    bl = i, j + 1
    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
    # 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_intensity = [(1 - del_j) * (1 - del_i) * source_image[tl] +
                      (1 - del_j) * del_i * source_image[tr] +
                      del_i * (1 - del_j) * source_image[bl] +
                      del_i * del_j * source_image[br]]
    # the weights for interpolation.
    # pixel_intensity = (1 - del_j) * (1 - del_i) * source_image[tl] +
    # del_i * (1 - del_j) * source_image[tr] +
    # del_i * del_j * source_image[br]
    # Note that this function does T-S mapping. So, 'transformation' is actually from Target to Source.
```