Assignment 2:

Stereo Matching

Due Date: See in Moodle

The problem:

You are given a pair of images (image_left.jpg, image_right.jpg) of the same scene from two views. The two images are rectified such that the displacement between the two is only horizontal. Our goal for this assignment is to estimate the 3D structure of the image.

The exercise outline is as follows: We'll start with a naive estimation using 'Sum of Squared Differences' (SSD). Next, we'll correct the naive solution with Dynamic Programming and Semi-Global approaches.

The exercise will be implemented in python 3.9.5 or higher. We recommend working with conda. We have supplied an environment.yml file so you can create the conda environment from this file. You're free to use an IDE of your choice (some people prefer Pycharm over others), but your code needs to run from the terminal. That is, we'll run:

python main.py

This line will print all depth estimations as figures and additional data to the terminal.

It is highly recommended to use Linux, and specifically Ubuntu from an LTS version. We will test your code on an Ubuntu 16.04 machine.

Python packages in this assignment: pillow, numpy, scipy, random, matplotlib, time and opencv. No other additional libraries are allowed.

The exercise has changed from past years and it is now a "fill your code here" exercise. That means, that in addition to following the exercise as it is specified in this document, you can use the function documentation to your assistance. Though, as opposed to the first exercise, we let you be creative in some stages of the exercise. When you're asked to be creative, we will use that word explicitly: "**CREATIVE MODE**" will be announced.

We supplied a file named **solution.py**. This is the only file you can write code to. Don't use other files.

For the bonus part, add another file named bonus.py which runs the bonus sections. Only the BONUS section is an optional part. Parts A-E are mandatory.

Tip: Read the entire outline of the exercise before you implement it. You will find it helpful to devise a concise solution.

Part A: Distance Tensor Computation

During the entire exercise, we will work with a distance tensor called SSDD. This tensor will hold the Sum of Squared Difference for every pixel for every disparity value. We shall start with a simple example, which will help us grasp the SSDD definition.

Suppose we're given these two rectified images (that is, we need to match them only in the x-axis).

Left

1	2	3	4	5
6	7	8	9	10
11	12	13	14	15
16	17	18	19	20

Right

1	1	1	1	1
2	2	2	2	2
3	3	3	3	3
4	4	4	4	4

The following parameters are denoted:

- **dsp_range**: this is the disparity range we need to scan. If dsp_range=2 then there are 5 values we need to scan: -2, -1, 0, 1, 2.
- **win_size:** window size (in pixels) we need to consider around each pixel. If win_size=3 then we consider 9 pixels around each pixel (8 neighbours).
- **ssdd:** in pixels the SSD distance for each pixel and every disparity value. Therefore, ssdd's shape would be in this case (with the parameters as in the examples): 4x5x5. Why? The first 4x5 are the image dimensions and the 5 in the third dimension is due to the 5 disparity values.

We will assume python indexation. That means that the first index in each dimension is 0 and the last index is N-1 for dimension of length N.

Computation Example:

The value of ssdd(1, 2, 1): corresponds to row=1, column=2 and disparity value=-1.

1,2 means that we consider the pixel from the left image at location (row=1, col=2). The last 1 corresponds to a disparity value -1. That means we consider the same pixel with x shifted by -1. Namely, pixel in location: (1, 1) in the right image.

Left

1	2	3	4	5
6	7	8	9	10
11	12	13	14	15
16	17	18	19	20

Right

1	1	1	1	1
2	2	2	2	2
3	3	3	3	3
4	4	4	4	4

We'll take a 3x3 window around the pixel in row=1 and column=2 in the left image. We'll take a window around the pixel in row=1 and column=1 in the right image.

The sum of squared differences between the two windows is:

2-1	3-1	4-1
7-2	8-2	9-2
12-3	13-3	14-3

.^2 = 1+4+9+25+36+49+81+100+121 = 426

Assume that the image is padded with zeros when some of the window is outside the original image shape.

1. Compute ssdd(1,2,2), ssdd(1,2,3), ssdd(2,3,0), ssdd(2,3,1) in a similar way.
2. Write a function that calculates the tensor of SSD distances between the two images, according to the example above.

   Use the following API:

def ssd_distance(left_image, right_image, win_size, dsp_range):

Input:

left_image	-	Left image of shape: HxWx3, and type np.double64.
right_image	-	Right image of shape: HxWx3, and type np.double64.
win_size	-	Window size odd integer.
dsp_range	=	Half of the disparity range. The actual range is: -dsp_range, -dsp_range + 1, ..., 0, 1, ..., dsp_range.

Output:

ssdd_matrix	-	A tensor of the sum of squared differences for every pixel in a window of size: win_size X win_size for the: 2*dsp_range + 1 possible disparity values. The tensor shape should be: HxWx(2*dsp_range+1).

Notes:

- win_size and dsp_range are constant values declared at the top of the script. You may change them though we will check with the numbers given.
- Since the images have been rectified, you're asked to find correspondence only along the x-axis.
- Normalize SSDD the matrix such that its minimal cost is 0 and the maximal cost is 255.

Part B: Naive Depth Map

Our goal here is to obtain the depth image from the distances tensor, without any smoothing.

From now on we will use the term “labels” to refer to the disparity index. For example, if dsp_range is 2, then the disparity values are: -2, -1, 0, 1, 2. and the corresponding labels are:

Disparity value	-2	-1	0	1	2
label	0	1	2	3	4

3. Write a function that builds a depth map out of the SSDD tensor without any smoothing.

   Use the following API:

def naive_labeling(ssdd_tensor):

Input:

ssdd_tensor - A tensor of the sum of squared differences for every pixel in a window of size: win_size X win_size
for the: 2*dsp_range + 1 possible disparity values.
The tensor shape should be: HxWx(2*dsp_range+1).

Output:

label_no_smooth - Naive labels HxW matrix.Each entry in that matrix should be a disparity label(not value). That means, a number between 0 and 2*dsp_range.

4. Give a short explanation to the result. Outline the problems of using the SSDD tensor in a naive approach. What are the reasons for the problems you mentioned?

   Tip: You can use the forward mapping of each pixel in the left image to the corresponding pixel in the right image according to the disparity values. You can observe how much the right image and the forward mapped image are similar. Analyze the cases in which they’re dissimilar.

Part C: Depth Map Smoothing using Dynamic Programming

Our goal here is to enhance and correct the naive depth image that we obtained using Dynamic Programming.

We'll implement a scoring method which runs over horizontal slices of the distances tensor (ssdd). The dimension of every slice is 1 X nCols X nLabels. But for simplicity, we will assume that the slice dimension is nLabels X nCols. Where nLabels is the number of the different disparity values and nCols is the length of each row in the image (the width of the image).

Essentially, our score/cost method will implement the extent/cost of change in the depth of a pixel relative to its neighbourhood.

5. Implement the score method for a single slice of the ssdd tensor, using Dynamic Programming. Where the score for each value is given by:

   $$L(d,\ col) = C_{slice}(d,\ col)\ +\ M(d, col)\ -\ min\{L(:,\ col\ -\ 1)\}$$

   where:

   1. d, cols are the row and column indices in the slice.
   2. Note that we "normalize" each column with the minimal value in the previous column.
   3. M(d, col) is the cost of the optimal route until the current location. It is chosen to be the minimal value of the following items:
      a. the score of route L from the previous column for the same d value:

         L(d, col-1)

      b. the score of the optimal L from the previous column with disparity value deviating by $\pm\ 1$, in addition to a penalty p1:

         $$P1\ +\ min\{L(d\ -\ 1,\ col\ -\ 1),\ L(d\ +\ 1,\ col\ -\ 1)\}$$

      c. the score of the optimal L from the previous column with disparity value deviating by more than 2, in addition to a penalty p2:

         $$P2\ +\ min\{L(d\ +\ k1,\ col\ -\ 1)\ \forall k\colon\ |k|\ \geq\ 2\}$$

Use the API given in the next page.

Use the following API:

def dp_grade_slice(c_slice, p1, p2):

Input:

c_slice	-	A slice of the ssdd tensor.
p1	-	penalty for taking disparity value with 1 offset.
p2	-	penalty for taking disparity value more than 2 offset.

Output:

l_slice	-	Scores slice which for each column and disparity value states the score of the best route. This is a matrix of shape (2*dsp_range + 1)xW.

6. Implement a method which takes the full ssdd tensor and outputs a depth map, using Dynamic Programming.

You should invoke **dp_grade_slice** from the previous item on each and every slice of the ssdd to obtain the full L scores tensor (the full L tensor should have the same shape as ssdd).

Finally, for each pixel in L (that is for every row and column), choose the optimal disparity value = the disparity value which corresponds to the minimal L value in that dimension.

def dp_labeling(ssdd_tensor, p1, p2):

Input:

ssdd_tensor	-	A tensor of the sum of squared differences for every pixel in a window of size: win_size X win_size for the: 2*dsp_range + 1 possible disparity values. The tensor shape should be: HxWx(2*dsp_range+1).
p1	-	penalty for taking disparity value with 1 offset.
p2	-	penalty for taking disparity value more than 2 offset.

Output:

label_smooth_dp	-	A matrix of shape HxW (same number of rows and columns as in ssdd_tensor). The matrix should contain the optimal assignment according to the Dynamic Programming method.

7. What is the difference between the depth map obtained from the previous item and the map obtained naively? Include the two depth maps in your report.

Tip: You can use the forward mapping of each pixel in the left image to the corresponding pixel in the right image according to the disparity values. You can observe how much the right image and the forward mapped image are similar. Analyze the cases in which they’re dissimilar.

Part D: Depth Image Smoothing Using Semi-Global Mapping

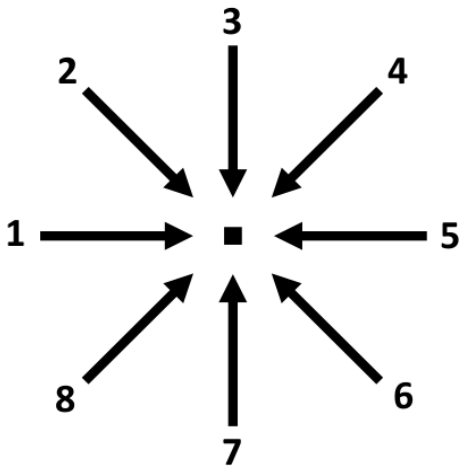
For items 8 and 9 you can be CREATIVE: That means that we will not supply a method signature for these items. you're free to choose whatever architecture you want to solve these items.

In this part we will further enhance and improve our depth map. We will do it in a Semi-Global approach. This can be thought of as a scenario where the global solution is approximated using a finite (small) set of one dimensional estimations.

We will rank (score) each disparity value in each pixel according to scores of different routes along different directions.

8. Implement a method which extracts slices from the ssdd according to a direction which it receives as an input.

We will define 8 directions:



Direction 1 is the horizontal direction we used in the previous section. Direction 5 is a horizontal direction with columns reversed.

Note that in the diagonal directions you're not asked to perform interpolations of any sort for values that fall between pixels. You should go only on the pixel values themselves.

Use symmetry to cut the running time.

9. Along diagonal directions there are slices of shorter lengths so you'll need to update **dp_grade_slice** to handle slices of shorter lengths.

10. Implement a method which, given a full ssdd tensor, computes the depth map according to the Semi-Global Mapping approach.

    a. For each direction 1, ...8 compute the L-scores matrix using dynamic programming. You should have 8 tensors: L_1,..., L_8.

    b. Average all 8 tensors to obtain a single L score matrix.

    c. Use the methods: **dp_grade_slice** and the method from item 8.

    d. Finally, for every pixel in L (for every row and column), choose the disparity value which corresponds to the minimal L value in that particular pixel.

Use the following API:

def sgm_labeling(ssdd_tensor, p1, p2):

Input:

ssdd_tensor	-	A tensor of the sum of squared differences for every pixel in a window of size: win_size X win_size for the: 2*dsp_range + 1 possible disparity values. The tensor shape should be: HxWx(2*dsp_range+1).
p1	-	penalty for taking disparity value with 1 offset.
p2	-	penalty for taking disparity value more than 2 offset.

Output:

label_smooth_sgm	-	A matrix of shape HxW (same number of rows and columns as in ssdd_tensor). The matrix should contain the optimal assignment according to the Semi-Global Mapping method.

11. What is the difference between the depth map obtained from the previous item and the map obtained naively? Include the two depth maps in your report.

Tip: You can use the forward mapping of each pixel in the left image to the corresponding pixel in the right image according to the disparity values. You can observe how much the right image and the forward mapped image are similar. Analyze the cases in which they're dissimilar.

12. To debug your result, implement a method which, given a full ssdd tensor, computes the depth map according to the Semi-Global Mapping approach for each direction. This method should return a dictionary mapping each direction (integer: 1, ..., 8) to a depth map computed with an L tensor obtained from that direction.

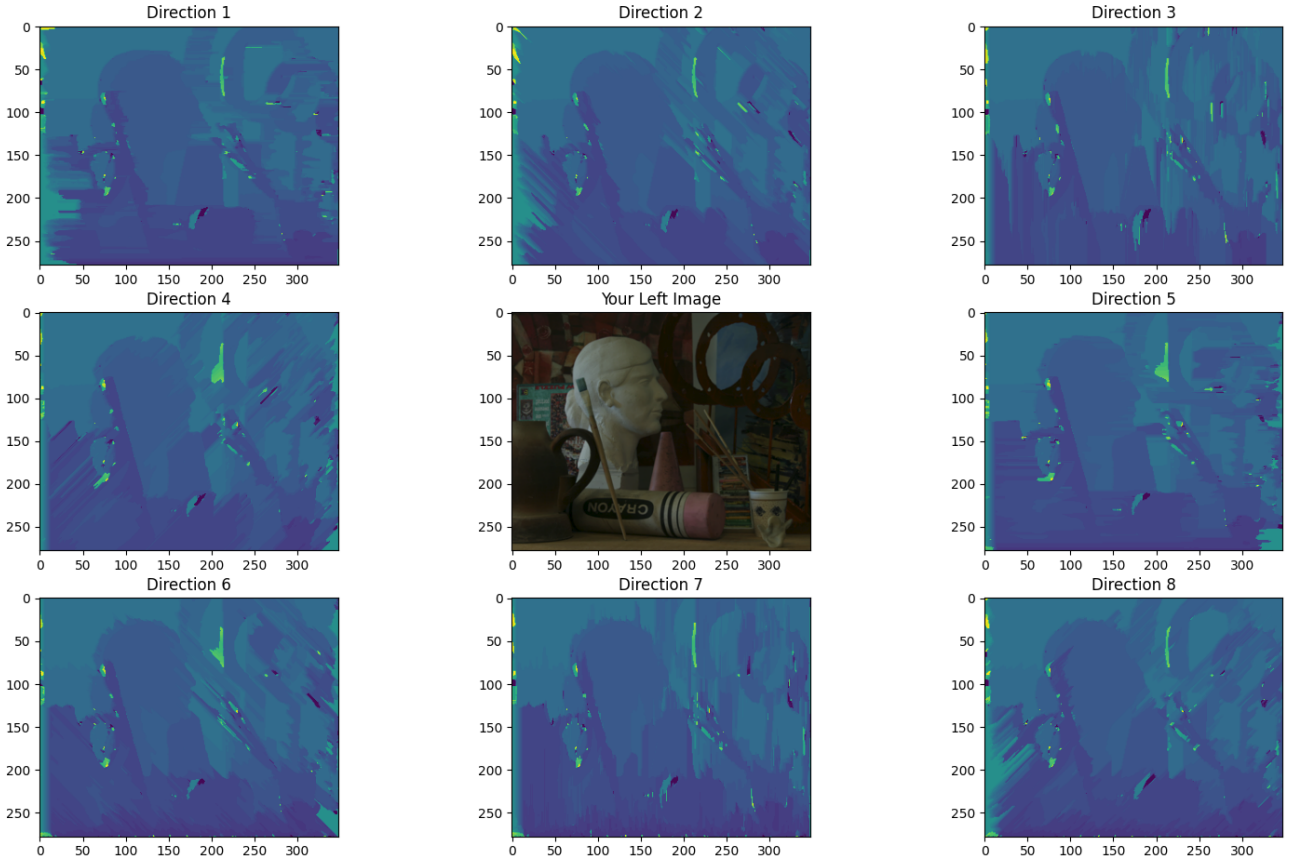
def dp_labeling_per_direction(ssdd_tensor, p1, p2):

Input:

ssdd_tensor	-	A tensor of the sum of squared differences for every pixel in a window of size: win_size X win_size for the: 2*dsp_range + 1 possible disparity values. The tensor shape should be: HxWx(2*dsp_range+1).
p1	-	penalty for taking disparity value with 1 offset.
p2	-	penalty for taking disparity value more than 2 offset.

Output:

direction_to_slice	-	Dictionary int->np.ndarray which maps each direction to the corresponding dynamic programming estimation of depth based on that direction.



Part E: Your own image

13. Use a pair of your own images. Make sure the images have been rectified. Run your code which generates the Semi-Global Mapping on the pair of images. We recommend using images from: https://vision.middlebury.edu/stereo/data/ . For the main script to work: name the left image: “my_image_left.png” and the right image: “my_image_right.png”. You may overrun the parameters: COST1, COST2, WIN_SIZE and DISPARITY_RANGE in the “Your image playground section” at the bottom of the script.

Bonus Part:

14. Suggest and implement a new metric to measure distances which replaces SSD. Show naive labeling, Dynamic Programming in a single direction and SGM using the metric you suggested. Compare your results with the SSDD results you obtained in the previous items.

15. Suggest and implement an algorithm to smooth the depth image which replaces the methods suggested in the exercise. You may use SSD as a distance measurement or the metric suggested in 14. Compare your results with the Dynamic Programming and SGM results you obtained in the previous items.

16. What are the differences between the results of the depth image from 15 to SGM?

Submission instructions:

- A document containing reference to all sections of the exercise must be submitted, showing all the results and answering all questions (no code is required in the document).
- All API-defined python functions of the assignment must be submitted, as well as any associated functions that you wrote (all functions should be in solution.py). The functions will be automatically run and tested.
- Attach your pair of images (my_image_left.png and my_image_right.png).
- Check that you are able to run the attached **main.py** without making any changes to it. This will only be possible if all the functions you need to write in the exercise appear in **solution.py**.
- **The solution with all relevant files must be submitted in the submission box in the moodle within a zip file named:**

  assignment2_ID1_<id_1>_ ID2_<id_2>

  If the zip file exceeds 50MB, you may email it to: amirjevn@mail.tau.ac.il
  The subject of the email should be stated as follows:
  Assignment 2 ID1: <your_id_number> ID2: <your_id_number>
- Late submission will result in grade reduction.

Good Luck!