Homework5

October 22, 2019

1 Homework 5: Stereo

In this homework, you will implement an two-view stereo algorithm. Given two images from a stereo rig, you can estimate the 3D depth of the scene by first calculating the disparity between pixels, and use the camera intrinsics to triangulate the depth. This homework will focus only on estimating the disparity between pixels.

You will estimate two stereo algorithms, which are both worth 50 points. In the first stereo algorithm, you should implement a greedy matching approach. In the second stereo algorithm, you should implement a stereo algorithm that incorporates a prior of spatial smoothness, which you optimize with dynamic programming.

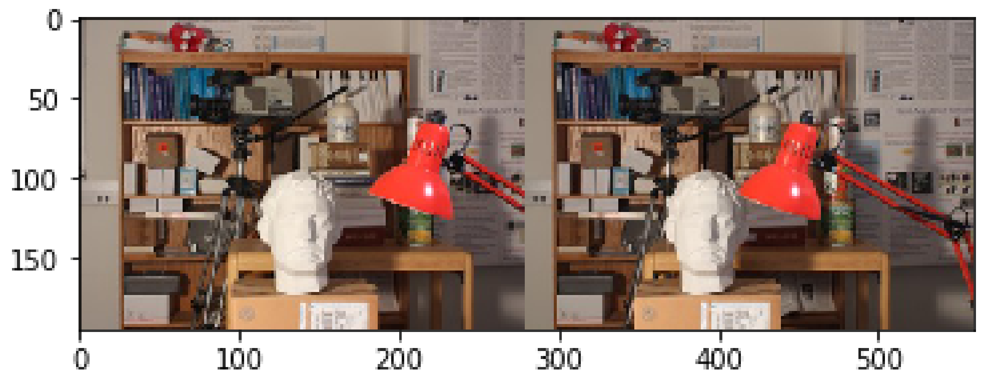
But, before we begin, let's load some helper functions.

```
In [1]: import numpy as np
        import matplotlib.pyplot as plt
        from PIL import Image
        from IPython import display
        from scipy.signal import convolve2d
        from math import *
        from tqdm import tqdm
        import time
        plt.rcParams['figure.figsize'] = [7, 7]

        def load_image(filename):
            img = np.asarray(Image.open(filename))
            img = img.astype("float32") / 255.
            return img

        def gray2rgb(image):
            return np.repeat(np.expand_dims(image, 2), 3, axis=2)

        def show_image(img):
            if len(img.shape) == 2:
                img = gray2rgb(img)
            plt.imshow(img, interpolation='nearest')
```

1.1 Visualizing Stereo Pairs

Let's visualize the images captured by the left and right camera pair. These images from the Middlebury Stereo Dataset (http://vision.middlebury.edu/stereo/data/).

In [2]:
```
left = load_image('im0.jpg')
right = load_image('im1.jpg')

height,width,_ = left.shape

pad_size = 100

left_pad = np.pad(left, ((pad_size, pad_size), (pad_size, pad_size), (0,0)), mode='cons
right_pad = np.pad(right, ((pad_size, pad_size), (pad_size, pad_size), (0,0)), mode='c

show_image(np.concatenate([left, right], axis=1))
```



1.2 Problem 1: Greedy Stereo Matching (50 points)

From the left and right image, calculate the disparity between each pixel using a greedy matching algorithm. You may assume that the stereo pairs are rectified, which means the camera pairs are only horizontally translated. In your PDF, be sure to include both the code and the visualization of the estimated disparity.

Distance Function: There are a variety of distance functions that you can use. For this problem, you can just use sum of squared differences between RGB patches.

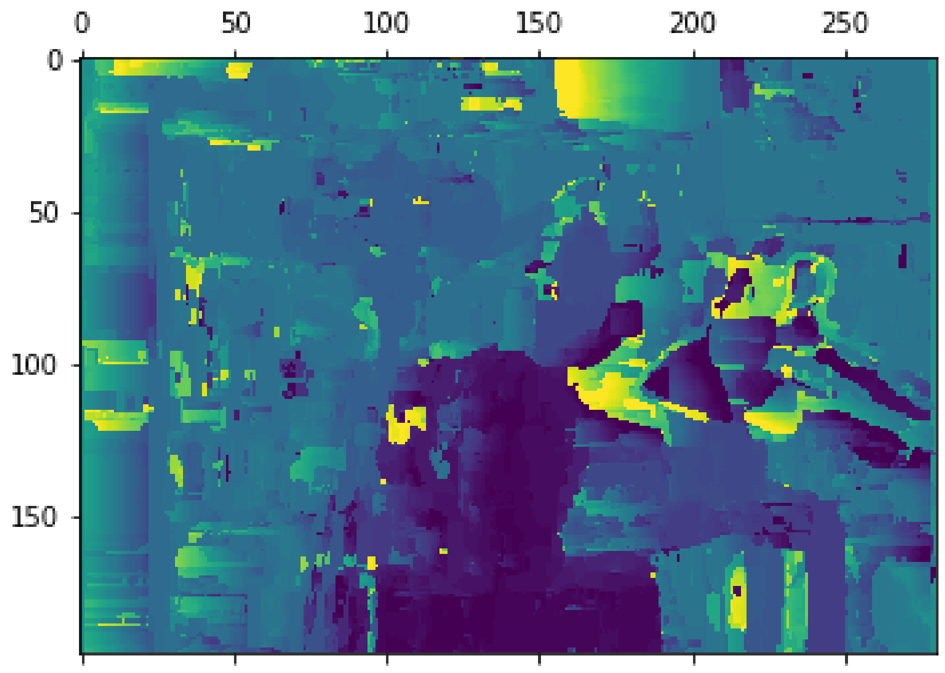
Accuracy: It is very hard to get the right result with a greedy approach. However, your approach should at least put the head and background in right place.

In [3]:
```
def compute_ssd(matrix1, matrix2):
    # compute ssd between two 2-D arrays
    return np.sum((matrix1-matrix2)**2)
```

In [4]:
```
# Some variables that might help you. But you don't have to use them.
max_dx = 30 # max number of displacements to search. make this smaller to speed up
```

```
dxs = np.linspace(-max_dx, max_dx, num=2*max_dx+1).astype("int32")
win_size = 5 # size of window you use for ssd computation

def compute_disparity():
    # TODO: calculate the disparity using a greedy approach between left and right
    disparity = np.zeros((height,width))
    half_win_size = win_size // 2
    best_offset = 0
    min_ssd = np.inf

    for y in range(height):
        for x in range(width):
            left_temp = left_pad[pad_size+y-half_win_size:pad_size+y+half_win_size+1,pa
            ssd = []
            for offset in dxs:
                right_temp = right_pad[pad_size+y-half_win_size:pad_size+y+half_win_si
                ssd_temp = compute_ssd(left_temp, right_temp)
                ssd.append(ssd_temp)

            best_offset = dxs[np.argmin(ssd)]
            disparity[y,x] = best_offset

    return disparity

disparity = compute_disparity()
plt.matshow(disparity)
```

```
Out[4]: <matplotlib.image.AxesImage at 0x1ffd62b95c0>
```



1.3 Problem 2: Dynamic Programming (50 points)

In this problem, implement a dynamic programming based stereo estimation algorithm that incorporates the spatial smoothness between adjacent disparities. For the local matching cost, you can use the same sum-of-squared-differences as above. For the pairwise cost, you can use L1 distance. In your PDF, include both code and the estimated disparity map.

Hint: In your algorithms course, you should have covered the Viterbi algorithm. Apply it to this problem. Write out the objective function as a recursive function. As you compute the recurrence, keep track of the pointers to the previous iteration. After you finish the recursive computation, just walk backwards using the stored pointers.

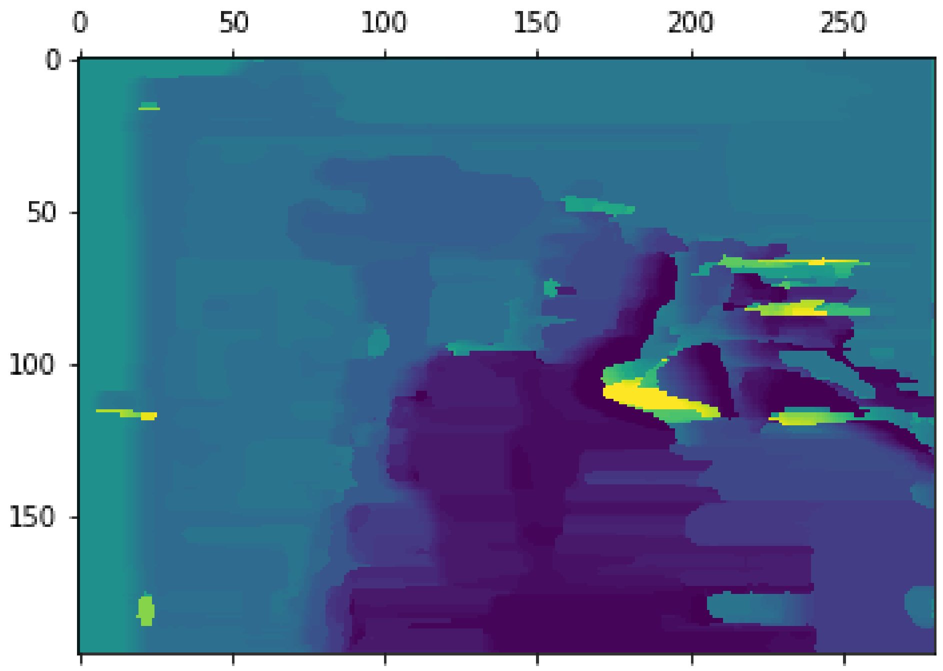
Hint Two: To help you debug, you can change the strength of the pairwise term. If you turn off the pairwise term, your implementation should recover the same solution as the greedy approach. If you make the strength of the pairwise term very high, your implementation should assign pixels to be identical disparity.

Accuracy: It is OK if you do not get perfect stereo reconstruction. Carl's solution gets it mostly right except for the lamp, which fails due to specular reflection.

```
In [5]: # Some variables that might help you. But you don't have to use them.
        max_dx = 30 # max number of displacements to search. make this smaller to speed up
        dxs = np.linspace(-max_dx, max_dx, num=2*max_dx+1).astype("int32")

        win_size = 5 # size of window you use for ssd computation
        lamb_coeff = 1 # coefficient in front of pairwise cost

        def l1_score(a,b):
            # TODO: implement a score for L1 distance
            return np.abs(a-b)

        half_win_size = win_size // 2

        def compute_ssd_by_index(y, x, prev_ssd=None):
            # compute ssd given location parameters and offset
            left_temp = left_pad[pad_size+y-half_win_size:pad_size+y+half_win_size+1,pad_size+
            ssd = []

            if x != 0:
                for offset in dxs:
                    right_temp = right_pad[pad_size+y-half_win_size:pad_size+y+half_win_size+1
                    ssd_temp = compute_ssd(left_temp, right_temp)
                    new_offset = lamb_coeff * np.abs(dxs - offset) + prev_ssd
                    ssd_temp += np.min(new_offset)
                    ssd.append(ssd_temp)

            else:
                for offset in dxs:
```

```
                right_temp = right_pad[pad_size+y-half_win_size:pad_size+y+half_win_size+1
                ssd_temp = compute_ssd(left_temp, right_temp)
#                 new_offset = lamb_coeff * np.abs(dxs - offset) + prev_ssd
#                 ssd_temp += np.min(new_offset)
                ssd.append(ssd_temp)

        return np.array(ssd)

    def compute_disparity():
        # TODO: calculate the disparity using dynamic programming between left and right
        disparity = np.zeros((height,width))
        for y in range(height):
            for x in range(width):
                if x == 0:
                    ssd = compute_ssd_by_index(y,x)
                    disparity[y,x] = dxs[np.argmin(ssd)]
                    prev_ssd = ssd
                else:
                    ssd = compute_ssd_by_index(y,x,prev_ssd)
                    disparity[y,x] = dxs[np.argmin(ssd)]
                    prev_ssd = ssd


        return disparity

    disparity = compute_disparity()
    plt.matshow(disparity)
```

```
Out[5]: <matplotlib.image.AxesImage at 0x1ffd63304a8>
```



1.4 Deliverables

Export your completed notebook as a PDF. Make sure the PDF includes both the code and the resulting output. We will be grading both your code and final output. (Runtime will not impact your score.)