Computer Vision
Assignment 3

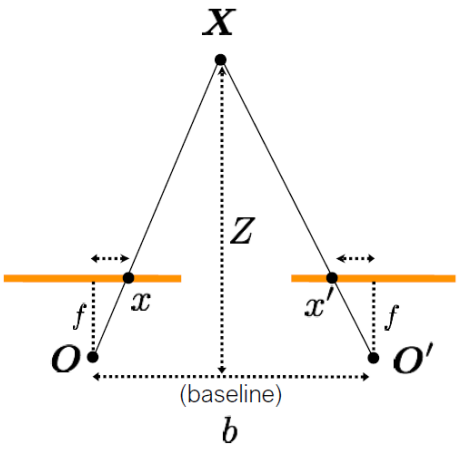
Students:

Mennatullah Ibrahim Mahmoud - 6221
Nour El-din Hazem Abdelsalam - 6261
Anas Ahmed Bekheit - 6659

Introduction

Stereo vision is the process of extracting 3D information from multiple 2D views of a scene. If we have two aligned cameras with a displacement between them in the x-axis. We can extract information about the depth of the objects from the disparity between the two images. This system is basically how the human vision works, the two cameras are our eyes and our brain estimates the depth of the objects.

$$\frac{X}{Z} = \frac{x}{f}$$



$$\frac{b - X}{Z} = \frac{x'}{f}$$

Disparity

$$d = x - x' \quad \text{(wrt to camera origin of image plane)}$$

$$= \frac{bf}{Z}$$

Using the rules of geometry, we can deduce that the depth of the object is inversely proportional to the disparity.

The first step we need to estimate the disparity therefore estimate the depth of the object is matching each object location to its corresponding location in the other image.

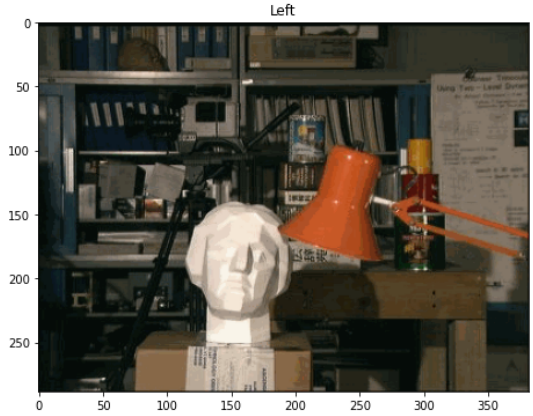
This can be done in two ways, block matching and dynamic programming.

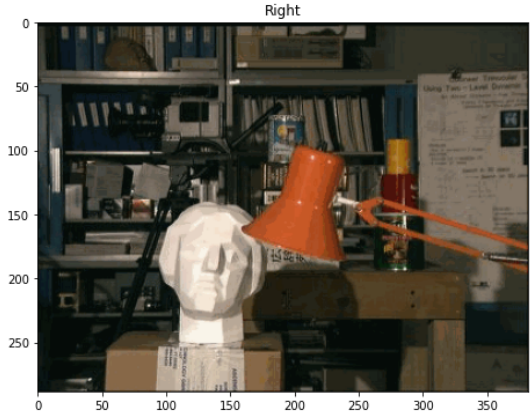
Part 1: Block Matching

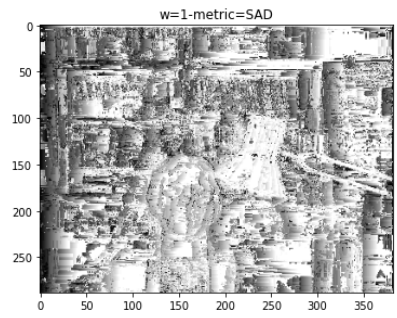
We first convert the colored images to gray scale. After this we loop over each window and calculate its cost with all the available matches that lie on the same horizontal line and are within the tolerable offset which is a hyperparameter. We have two cost functions, the sum of absolute differences and the sum of squared distances. We will refer to them as SAD and SSD in the following part. This step in the code is wrapped in a function with the name “calc_disparity”

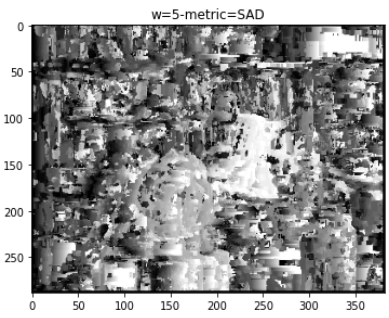
We experimented on each image with multiple windows and multiple offsets to compare the results.

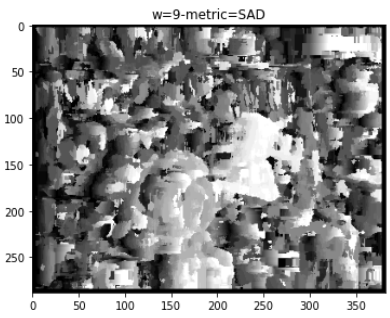
Results:

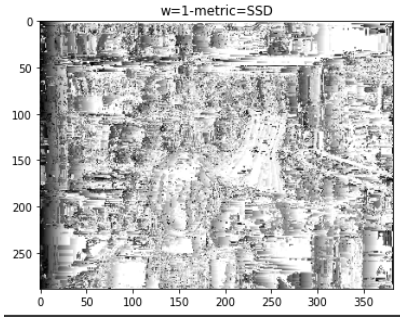

Left

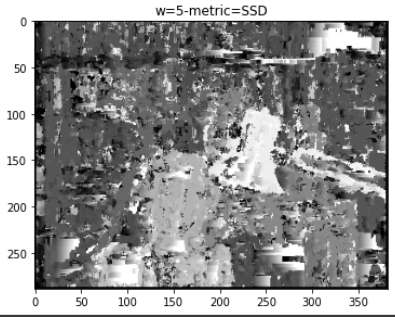

Right

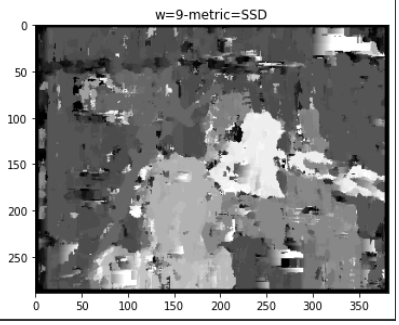

w=1-metric=SAD

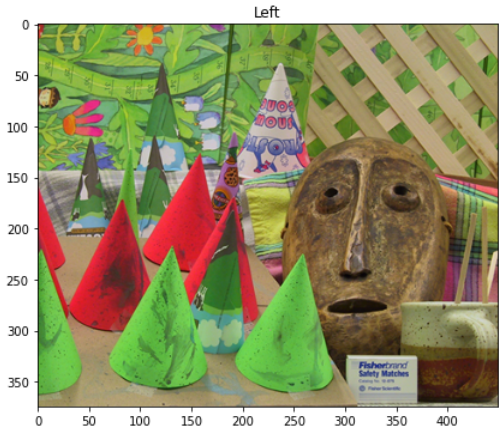

w=5-metric=SAD

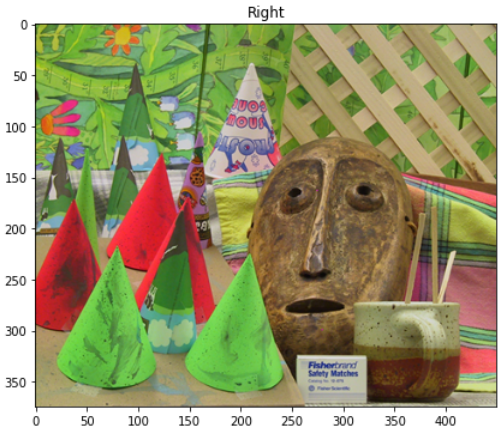

w=9-metric=SAD

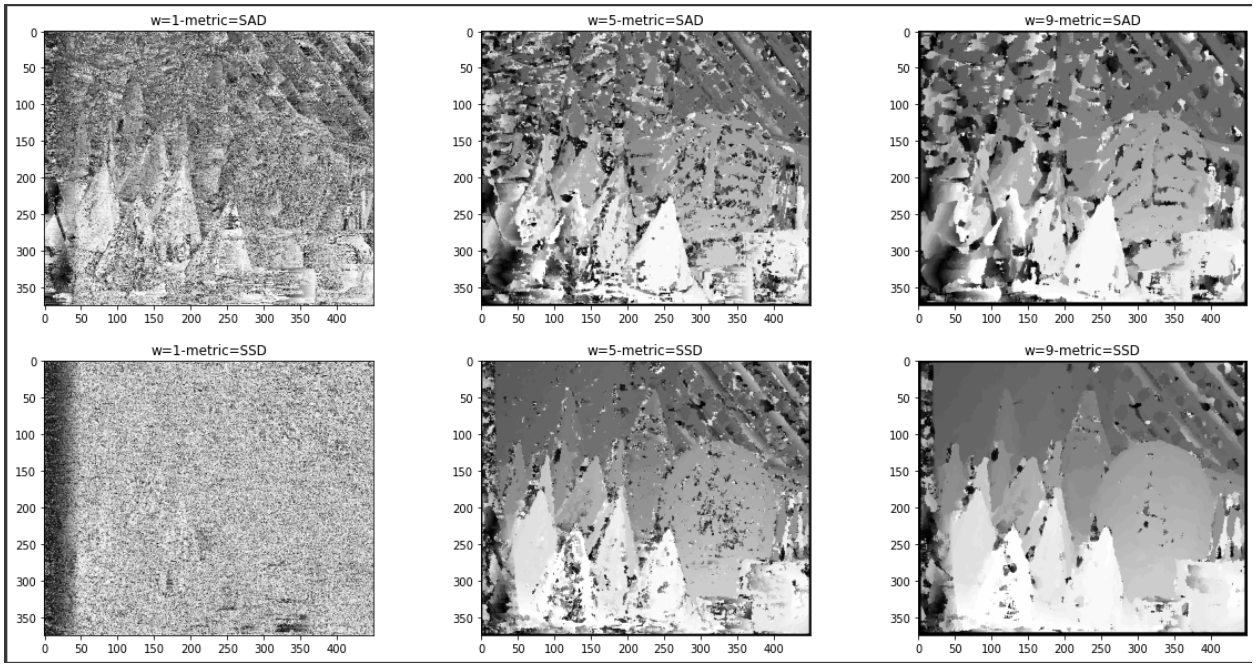

w=1-metric=SSD

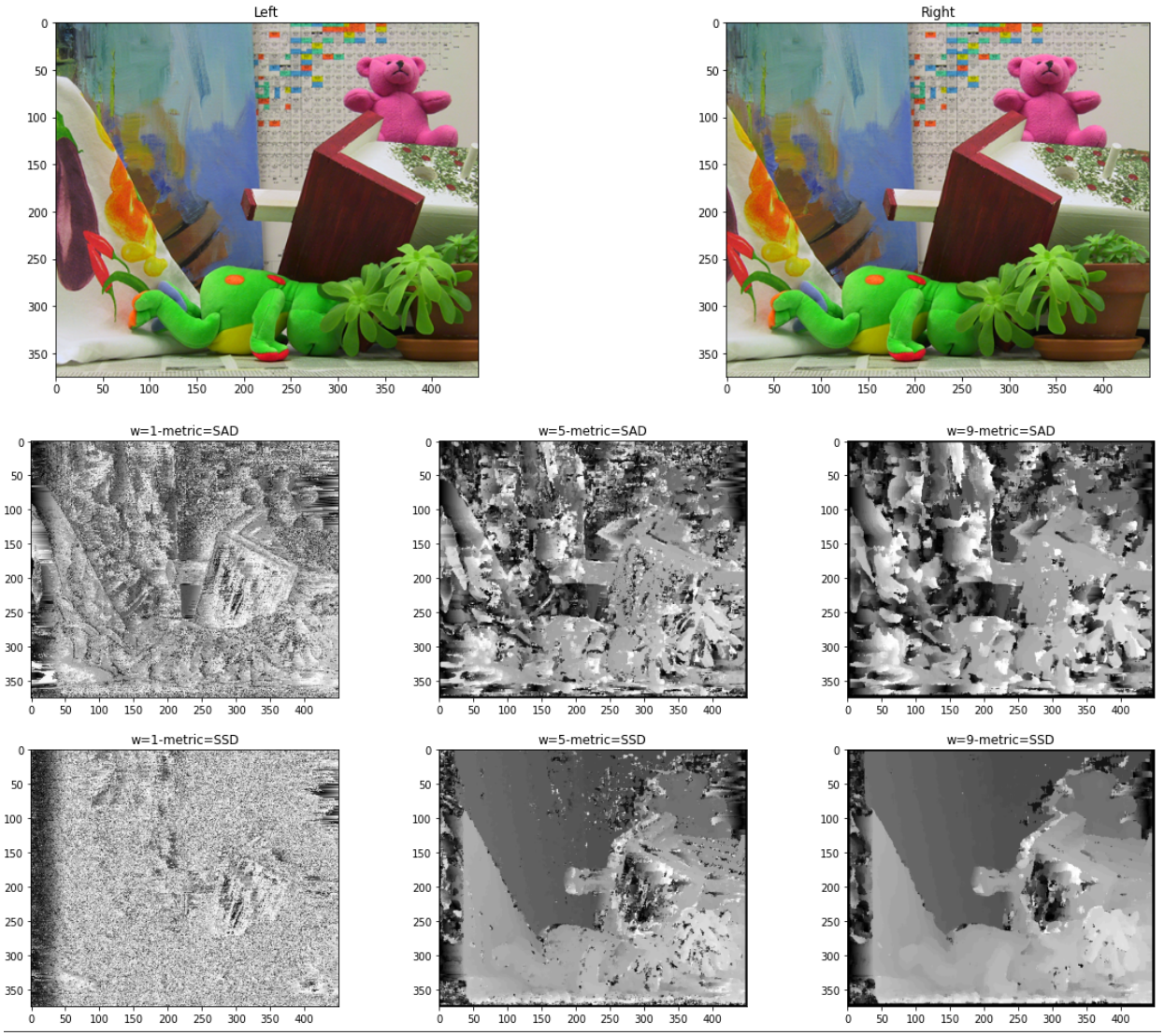

w=5-metric=SSD


w=9-metric=SSD


Left


Right


w=1-metric=SAD
w=5-metric=SAD
w=9-metric=SAD
w=1-metric=SSD
w=5-metric=SSD
w=9-metric=SSD



Conclusion:

We notice here that the SSD cost function yields in better results. This is due to the fact that it adds non-linearity as it makes the big differences even bigger compared to the smaller ones therefore it punishes outliers more strictly than the SAD which is a linear function.

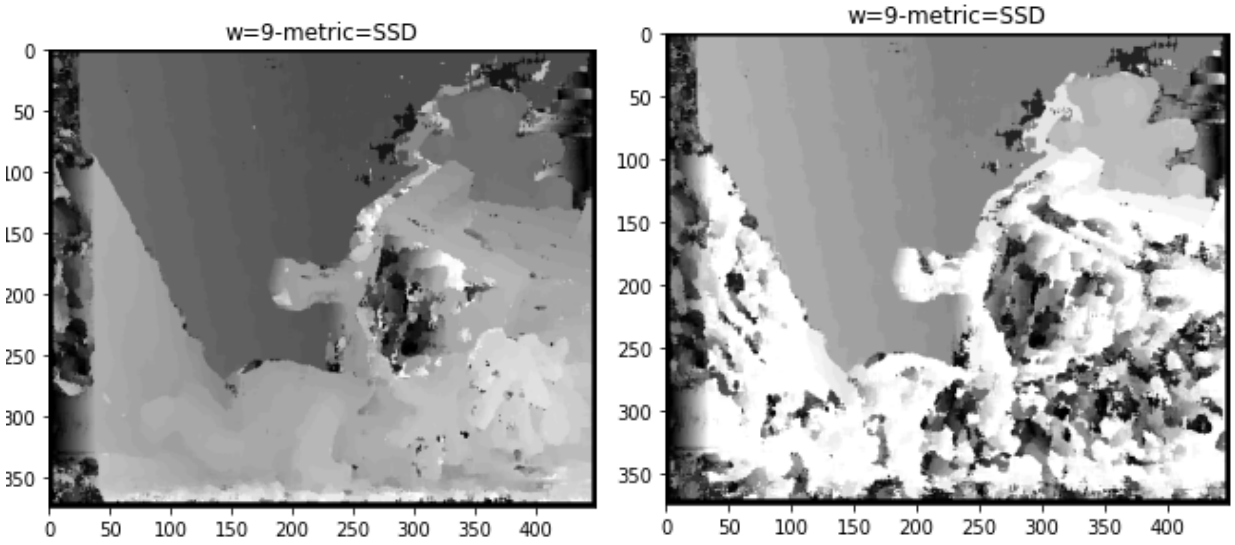
We also notice that the results get smoother when the window is larger since the patch in a larger window contains more information about the object than a pixel does. However, accompanied with this smoothness is the negative effect of blurry boundaries. It is a tradeoff

but if our application is concerned with estimating depth, larger window size will be more beneficial.

As we can see here, if we use SSD with a larger window size (20), we will have smoother results.

However, another negative side effect for the big window is the black borders that appear more prominently.

Another hyperparameter that affected the results which is the maximum offset that the match can lie within.



In these two images, the window size (9x9) and the cost function are the same, the only difference between them is the maximum offset. The image on the left has larger offset (50) and the image on the left has an offset of (30).

In our experiments, we concluded that the value of the offset depends on the nature of the image.

Part 2: Dynamic Programming

In the second part, we applied a different methodology which is dynamic programming.

Pixels in each scanline can either be matched or skipped. Each operation has its cost. The matching is determined by this operation: $dij = \frac{(Il(i) - Ir(j))^2}{\sigma^2}$ where σ is some measure of pixel noise.

The cost of skipping operation is a constant Co

Both Co and σ are hyperparameters and we set them according to the instructions in the pdf by 1 and 2 respectively.

We then built the matrix D with the same dimension as the image and we followed a bottom up approach where the base case D[0][0] = d11 and D(i, j) = min(D(i − 1, j − 1) + dij , D(i − 1, j) + Co, D(i, j − 1) + Co)

Then using the cost matrix D, we find the optimal alignment by backtracking. Starting at (i, j) = (N, N), we choose the minimum value of D from (i - 1, j - 1), (i - 1, j), (i, j - 1). Selecting (i - 1, j) corresponds to skipping a pixel in Il, so the left disparity map of i is zero. Selecting (i, j - 1) corresponds to skipping a pixel in Ir, and the right disparity map of j is zero. Selecting (i -1, j - 1) matches pixels (i, j), and therefore both disparity maps at this position are set to the absolute difference between i and j.

Results

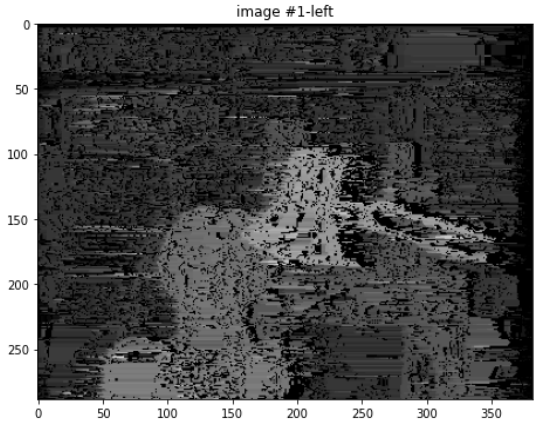


image #1-left

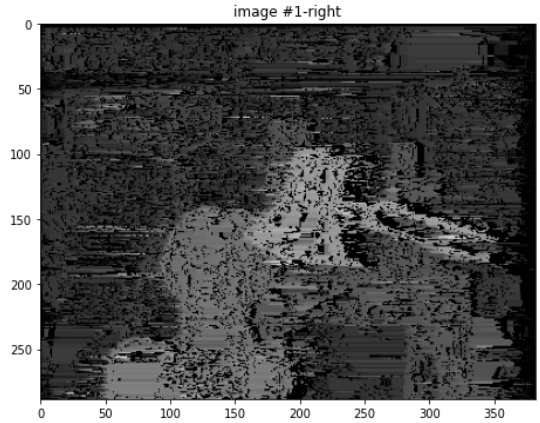


image #1-right

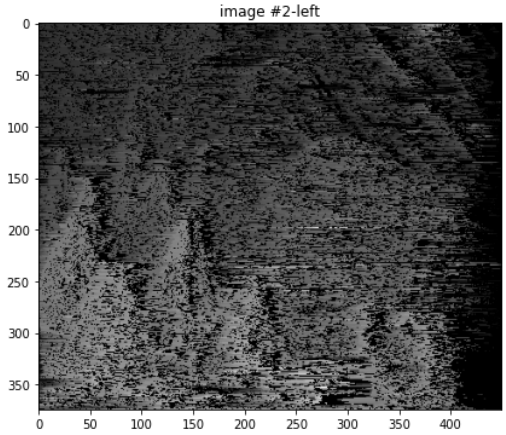


image #2-left

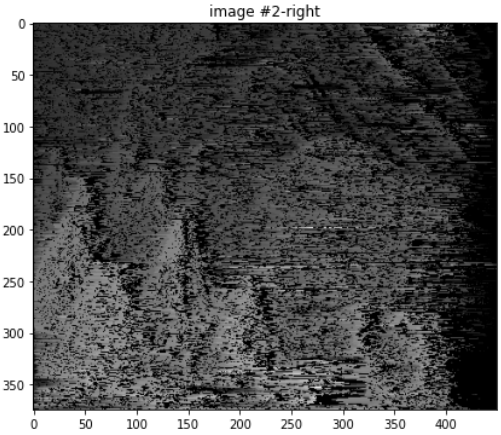


image #2-right

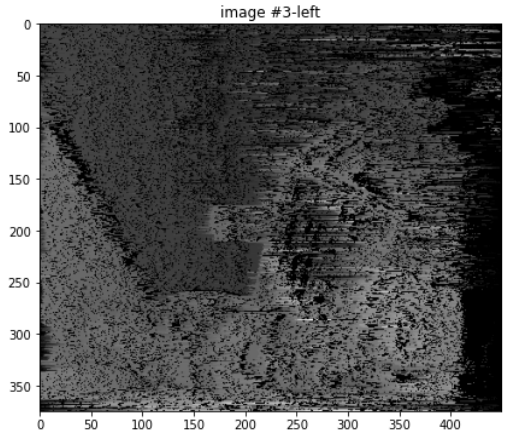


image #3-left

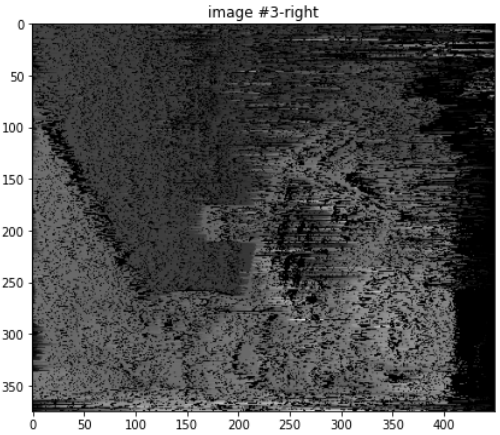


image #3-right