12 DECEMBER 2022

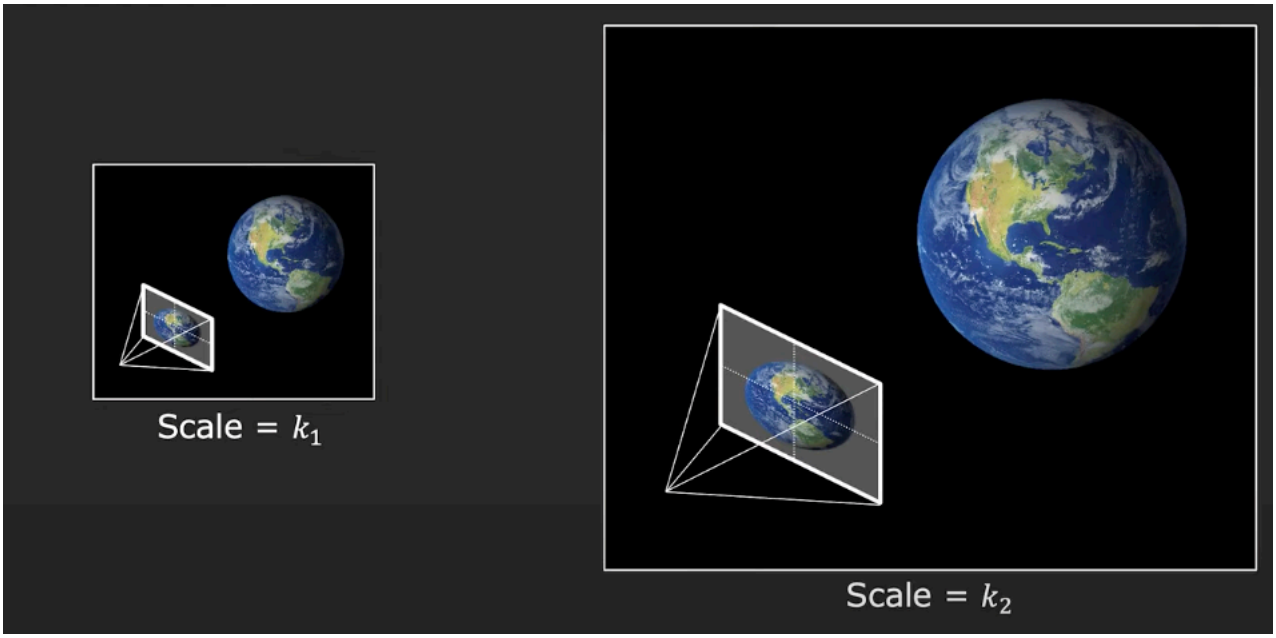
EE 417 - COMPUTER VISION

LAB #7 REPORT

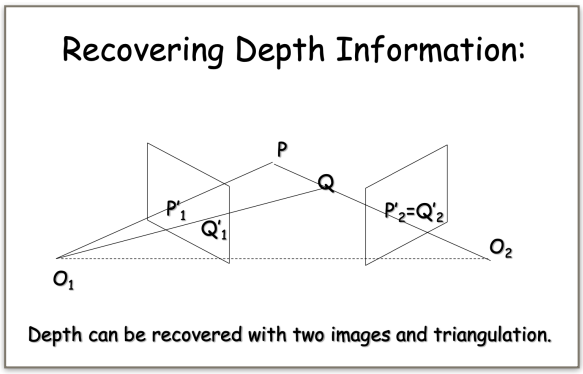
GOKTUG KORKULU
27026

INTRODUCTION

We know that the reflection of 3-dimensional objects on the 2-dimensional image plane causes them to lose the effect of depth. For example, two objects at different distances and different sizes may cause the same image on a image plane. For example;



Even if the sizes and distances of the objects are quite different from each other, we get the same results on the picture surface as seen above example. Depth perception is the most important feature we lose when projecting a 3D object onto a 2D surface.

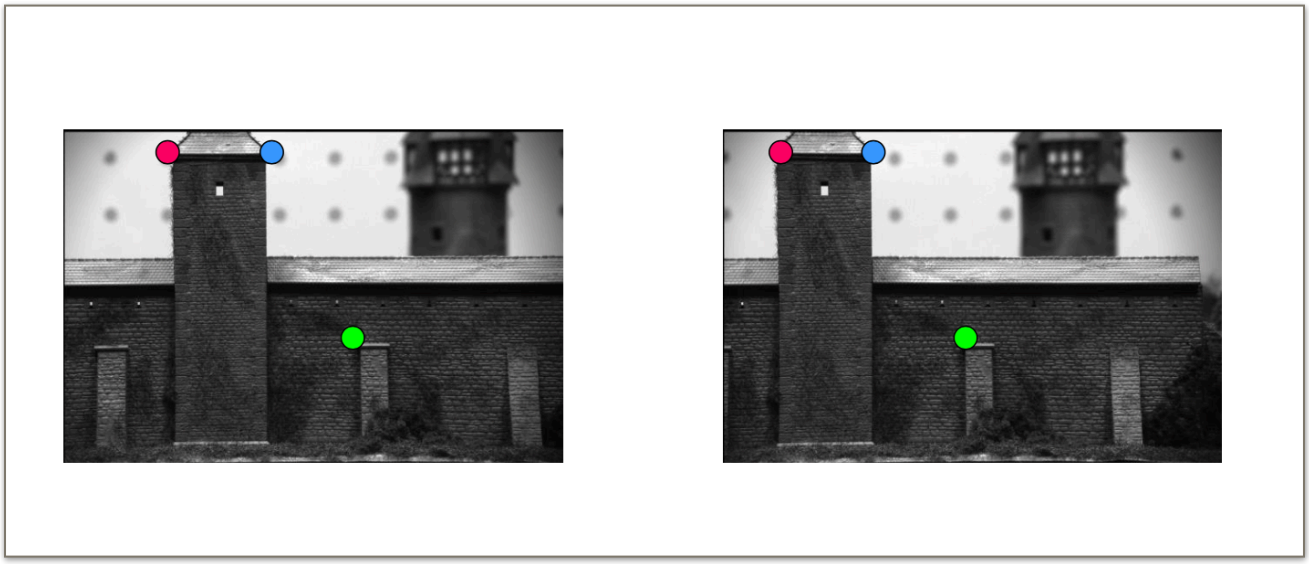


The topic we will be working on this week is “how can we recover this depth information?” and it will be a topic that we are looking for an answer to the question. According to the theory, we can recover the 3D structure and distance/depth of objects through 2 or more photographs of the same object taken from different angles. In the slides of the lesson, this idea is associated with the visual on the right.

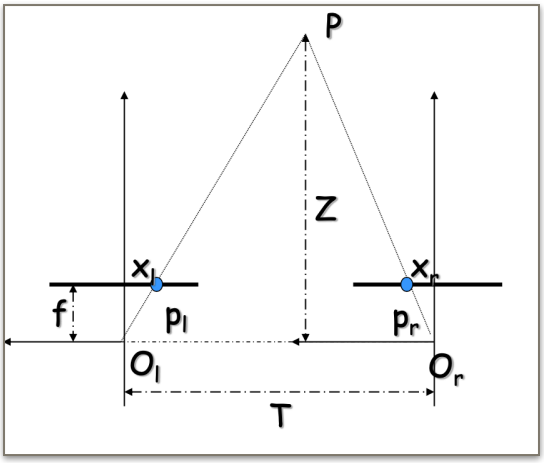
We call this event “Stereo Vision Notion”. This concept works on 2 problems.

1) Correspondence problem
2) Reconstruction problem

This week we will work on Problem 1. The “correspondence problem” is trying to find which pixel corresponds to which pixel in the other picture by analyzing photographs taken from two different angles of the same object or scene.



As seen above, we find the corresponding pixels on the photos taken from 2 different angles of the same scene.



Now, in order to approach the problem more scientifically, we need to define a few parameters, in the figure we see on the right, Ol and Or camera centers, the distance between these two camera centers T: “baseline” , f: “focal length”, Pl and Pr: PL: “projection” points corresponding to the picture surface of the two cameras of the same point, P: any point in 3D space, Z: the depth value of the P object we want to reach as a result.
Our last and most important parameter is d: “disparity” ; A measure of the distance between points in 3D space that correspond to two retinal surfaces.

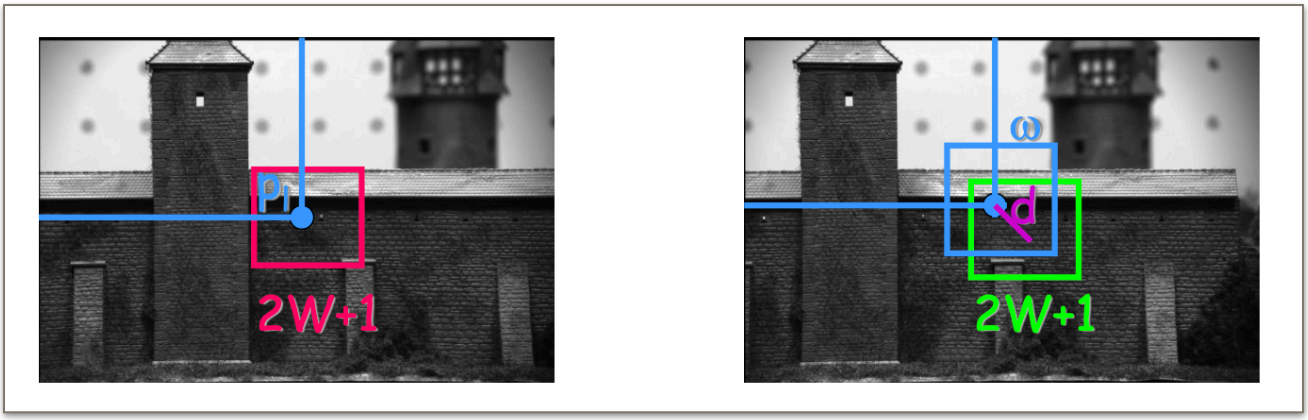
According to the calculations in the lecture slide I attached above, the disparities are inversely proportional to the depth of the point in 3D space. So in other words; As the distance of any point in a scene to the cameras increases, the distances of those points on the picture surface decrease.

We should not forget that we have to make a few assumptions so that the concept we mentioned can be adapted. These are simply
1) Most scene points are visible in both images and
2) Corresponding image regions re similar.

In order for these assumptions to hold, the distance of the fixation point from the cameras are much larger than the stereo baseline, i.e., Z>>T.

Lastly, this Correspondence Problem we mentioned consists of two main algorithms. One of them, and we will use in this lab, is the "correlation-based" algorithm. The other is the "feature-based" algorithm. The correlation-based algorithm we will use produces a dense set of correspondences.

Normally, the "correlation-based" algorithm is implemented by comparing two 2D windows with fixed sizes. For example, with parameters W: “search window size”, 2W+1: “correlation window size”; The working principle of the “correlation based” algorithm can be visualized with the figure below.



In general, the algorithm applied for each pixel p[i,j] in the left picture;

- For each displacement $d=[d_1,d_2]$ in $R(p_l)$ do:

$$C(d) = \sum_{l=-W}^{k=W} \sum_{k=-W}^{k=W} \Psi(I_l(i+k,j+l),I_r(i+k-d_1,j+l-d_2))$$

- The disparity of p_l is the vector d that maximizes $C(d)$ over $R(p_l)$

In general terms, the advantages and disadvantages of the "correlation method" are as follows:

- Creates dense maps, good for surface reconstructions
- Require textured images
- Sensitive to illumination variations
- İnadequate for very different viewpoints.

However, we will take a different approach, since the processing time is too long in the 2-dimensional comparison window and the efficiency is very low. Images were given to us as “rectified” in order to increase efficiency and shorten the processing time. This means that the "epipolar lines", which we will talk about in more detail next week, are parallel and horizontal in both images, so that the search for pixel matching can only be done in a horizontal line. Thus, time complexity is considerably reduced.

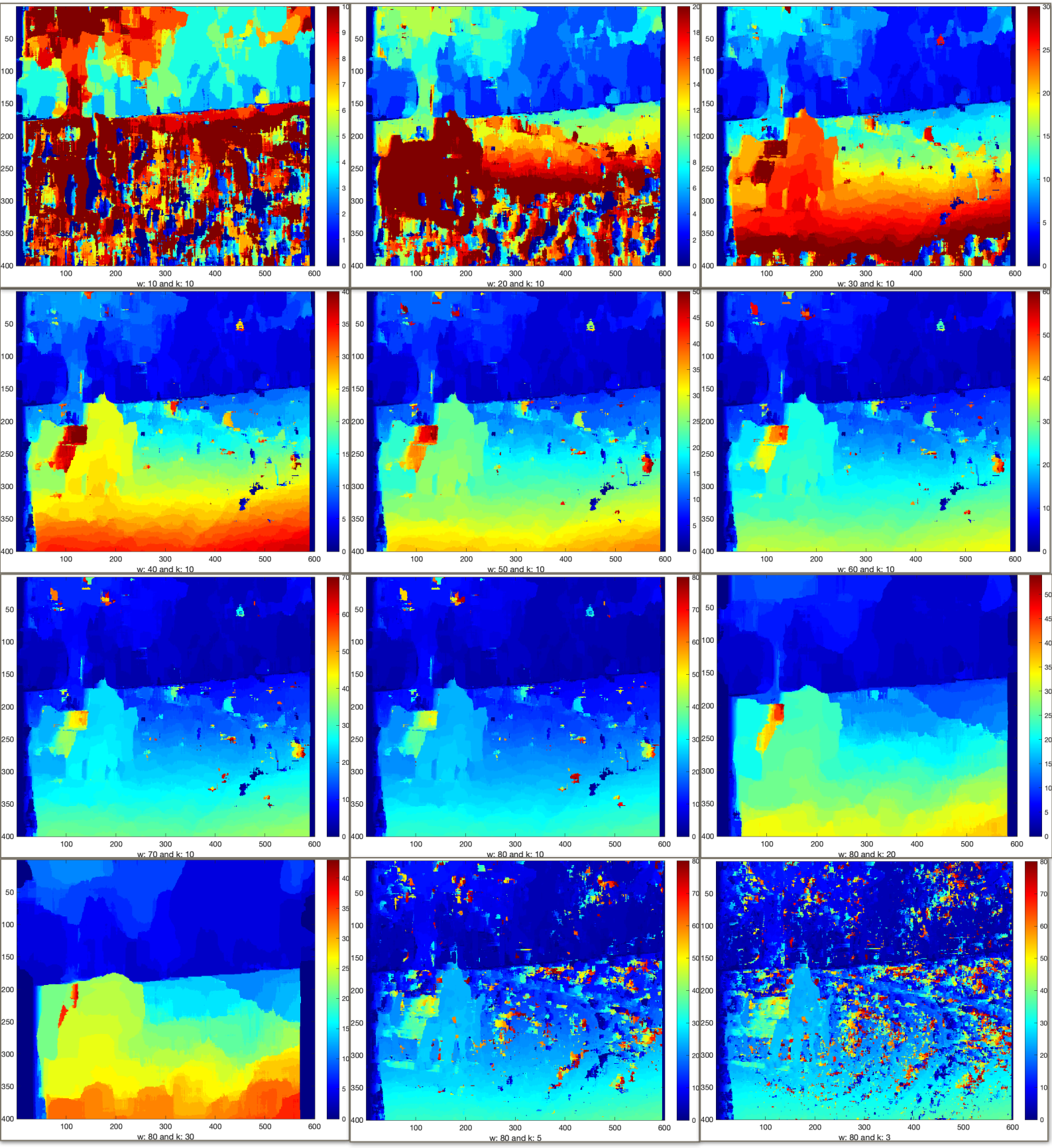
METHOD AND RESULTS

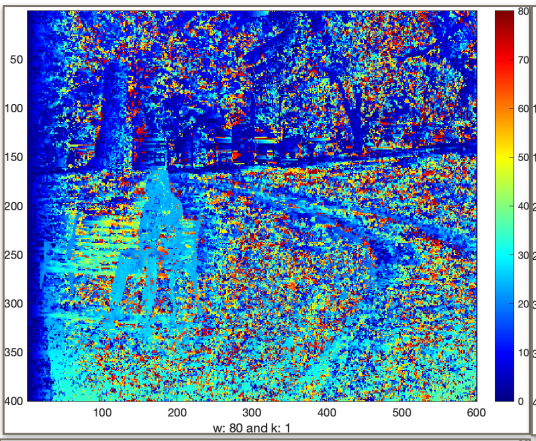
```
clear all, close all, clc;

leftImage =  imread("left.png");
rightImage = imread("right.png");
%leftImage =  imread("S00L.tif");
%rightImage = imread("S00R.tif");

[row, col, ch] = size(rightImage);

if (ch==3)
    %convert left and right images to grayscaled
    leftImage = rgb2gray(leftImage);
    rightImage = rgb2gray(rightImage);
end
%double the pixel values
leftImage = double(leftImage);
rightImage = double(rightImage);

%window size declarations. will change later on.
w=10;
k=100;

%pad the images by the size w+k
leftImage = padarray(leftImage, [w+k w+k], 'both');
rightImage = padarray(rightImage, [w+k w+k], 'both');
figure, subplot(1,3,1), imshow(uint8(leftImage)), subplot(1,3,2), imshow(uint8(rightImage));

%show the combination of images I1 and I2 into a red-cyan anaglyph
subplot(1,3,3), imshow(stereoAnaglyph(uint8(leftImage),uint8(rightImage)));

%create empty disparity matrix.
disparity = zeros(row,col);

tic
for y = w+k+1:row+w+k
    for x = w+k+1:col+w+k

        depo = [];
        leftFrame = leftImage(y-k:y+k, x-k:x+k);
        for i = 0:1:w
            rightFrame = rightImage(y-k:y+k, x-k-i:x+k-i);

            SSD =  sum(sum( (-1) * (rightFrame - leftFrame).^2  ));

            depo = [depo ; i SSD];

        end

        ind = find(depo(:,2) == max(depo(:,2)));
        d = depo(ind(1),1);

        disparity(y-w-k,x-w-k) = d;


    end
end
toc
figure, imagesc(uint8(disparity)); colormap jet; colorbar
xlabel("w: " + w + " and k: " + k)
```

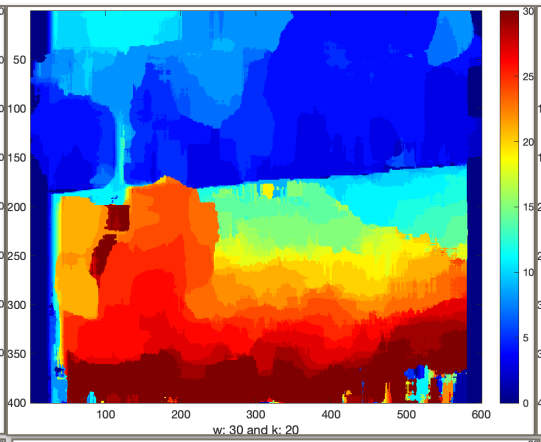
As usual, we start our method by reading the pictures given in the document. We will read the images left.png / right.png and S00L.tif / SOOR.tif as we are asked to analyze two different image sequences. First I start by reading the letf.png and right.png images. I take the dimensions of these pictures with the code in the 8th line and then convert them to gray scale if they are RGB. Then I convert the pixel values of both to double. The processes we have done so far were already routine processes that we do every week.
Next, I assign my w and k values. These values will then be changed to achieve the desired result.
I add padding to the pictures I take as input in the 24th and 27th line spacing. The size of these paddings will be the sum of the w and k values on both sides. In order to better understand what we are doing visually, I print these images as figures.
In the rest of the code, I print two images with the "stereoAnaglyph()" function, just for visual reasons.
In the continuation of the code, I create a matrix named "disparity" in the size of the first input pictures I received in which I will keep the disparity values for each pixel. This matrix is initially rendered as completely black.
Next, I enter the loop of calculating the disparities using the two-image comparison method, which is the most crucial part of the method.
Inside this loop, I find each pixel location of the image on the right and the corresponding pixel location on the left and keep the distances between the two in a matrix. I already created this matrix, it's called "disparity". To dig deeper, I calculate the difference for each value inside a pixel comparison window inside the innermost for loop and throw it into a matrix. Then I go out of the loop and choose the lowest among these values and save it as my “disparity”.
Finally, I present this matrix, in which I keep the disparity values, relative to the colors representing the distances.

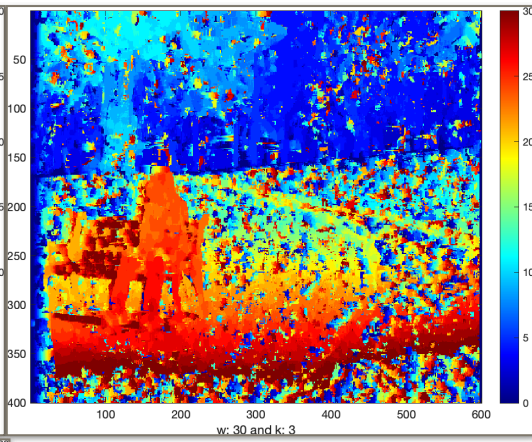
OBTAINED RESULTS BY CHANGING THE w AND k VALUES

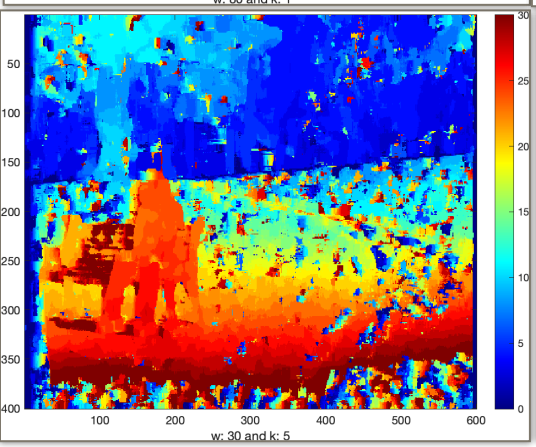
SOOL.png and SOOR.png

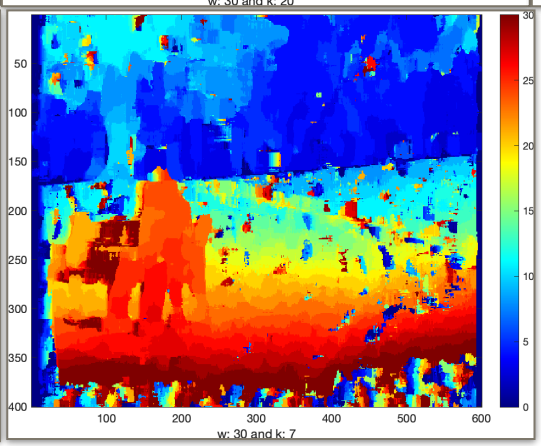



w: 80 and k: 1

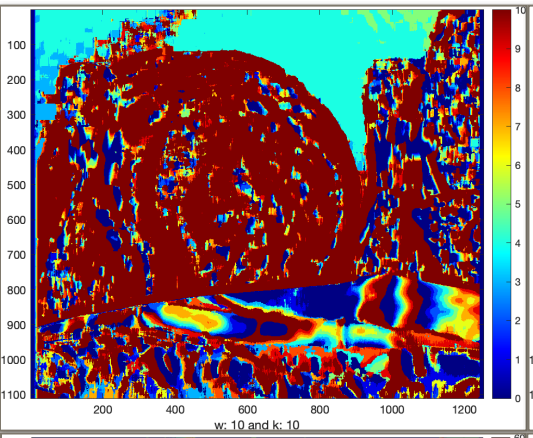

w: 30 and k: 20

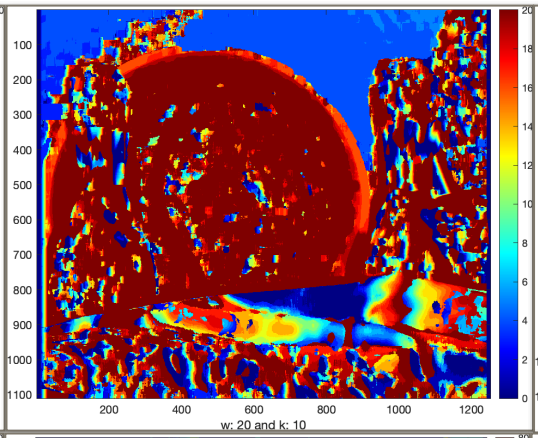

w: 30 and k: 3

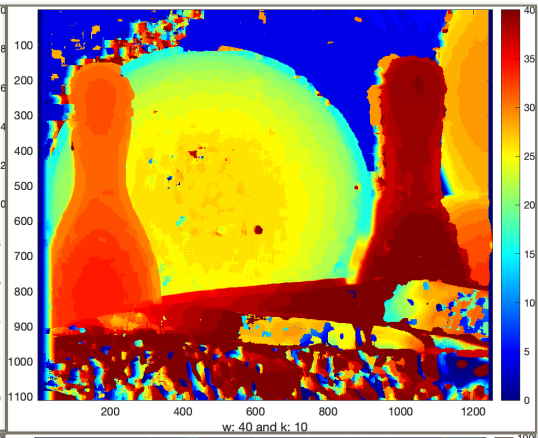

w: 30 and k: 5

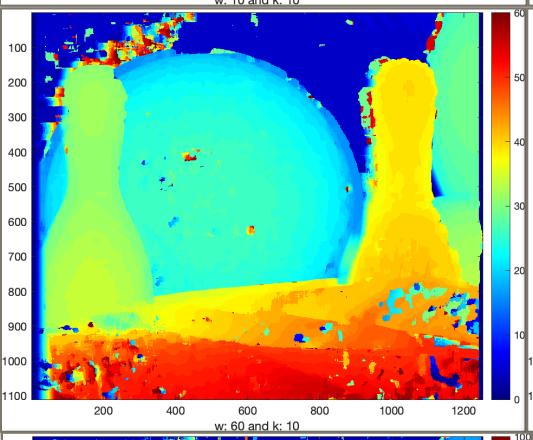

w: 30 and k: 7

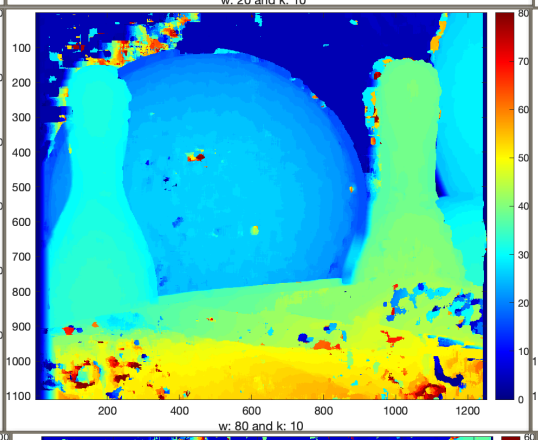
left.png and right.png

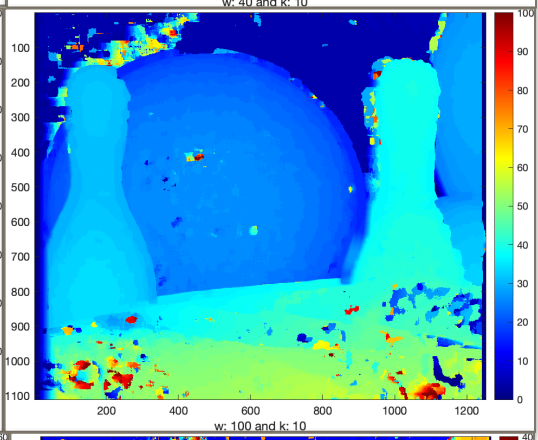


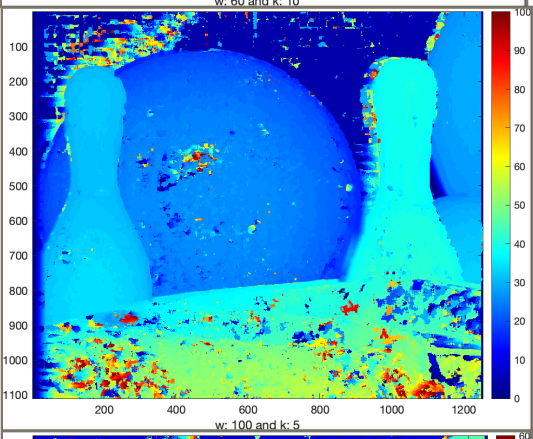


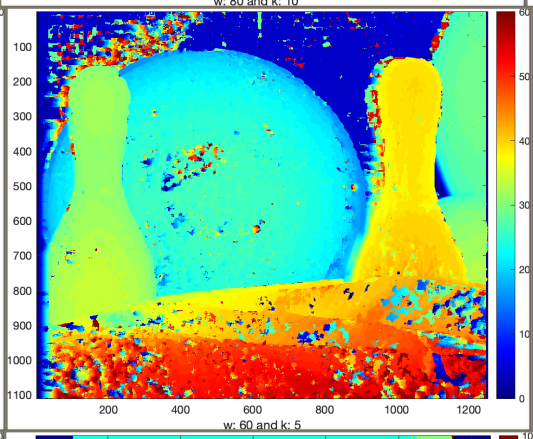


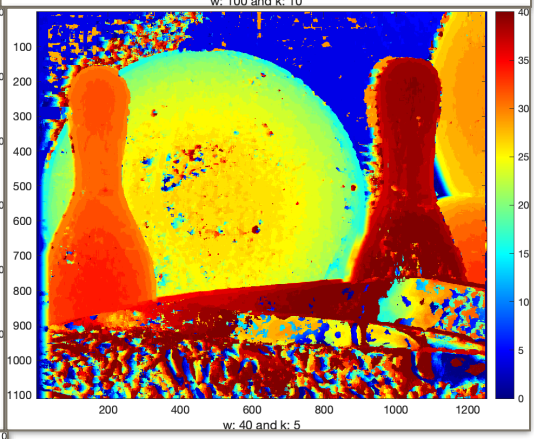


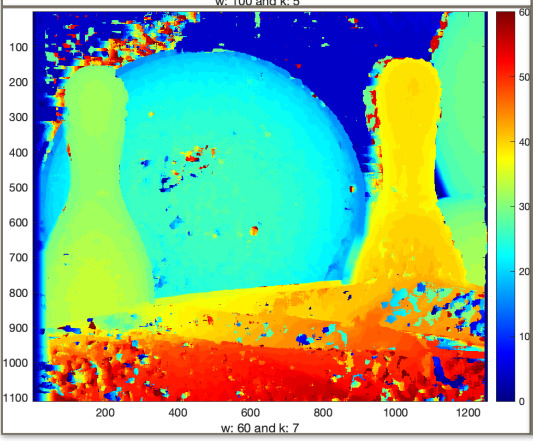


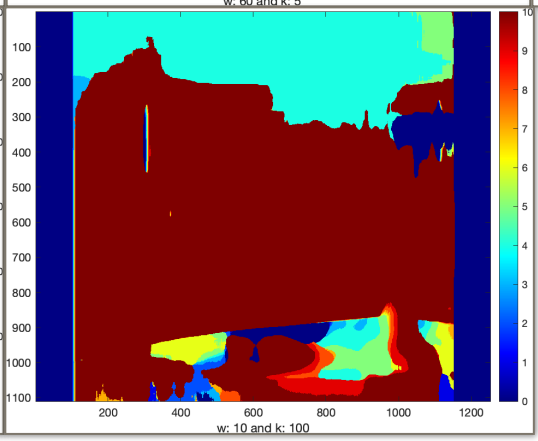












DISCUSSIONS AND CONCLUSION

In this week's report, I first talked about the deepening methods that we will apply in the beginning part, accompanied by quite detailed explanations. Later, I introduced the code I wrote for this purpose in detail, and I tried to explain the thought behind each line and the effect it had as much as I could do.
After explaining the code in detail, I tested the algorithm I wrote using two photos each of two different scenes given to us. I will now discuss the results I got.

Discussion on S00L.tif and SOOR.tif

First of all, I didn't have much idea about which w and k values to start with, so I gave 10 values for both and tested the algorithm. The result I got was a complete disaster. Let alone regaining the depth perception, I also lost the clarity of the objects.

I then increased my w values up to 20, 30, 40, 50, 60, 70, and 80, respectively, and analyzed the results. As you can see, as the values increase, the depth perceptions become very clear. In the extreme case of w=80 and k=10, the depth differences started to get smaller, so when we looked at the result, we could no longer see the depth differences in such different colors.

I thought the problem might be related to the k values and I played around with the k values. First I went up to k = 20, 30, then I went down to 5, 3, 1. The result I got when increasing K was to lose details of objects while restoring depth differences. So, on the one hand, I was regaining the depth information, which was my target, but on the other hand, my objects were becoming obscure.

The way I lowered the K value had, predictably, had a tradeoff, causing me to lose depth values even more, even though it brought out more details of my objects. But my even bigger problem was that

the algorithm was starting to become very sensitive to noise in the picture. For example, as a result of w=80, k=1, although the objects were very clear, I could not make much comment about their depth and I was getting a very ugly and distorted result.

So I decided to go with a smaller w value. Now that I discovered that when I increased the k value, my depth information decreased a bit, I could choose an image with relatively many colors (i.e. with more depth contrast) and lower the k value on it. I chose w=30 for this. While w=30, k=10 is relatively acceptable, I was looking for a better result.

What I didn't like was that the objects were not clear enough. So I tried k= 7, 5, 3 values respectively by decreasing the k value. W=30, k=7 values were the most preferable for me from the detail gain and noise sensitization tradeoff. As a result, I prefer w=30 and k=7 in these images.

Discussion on left.png and right.png

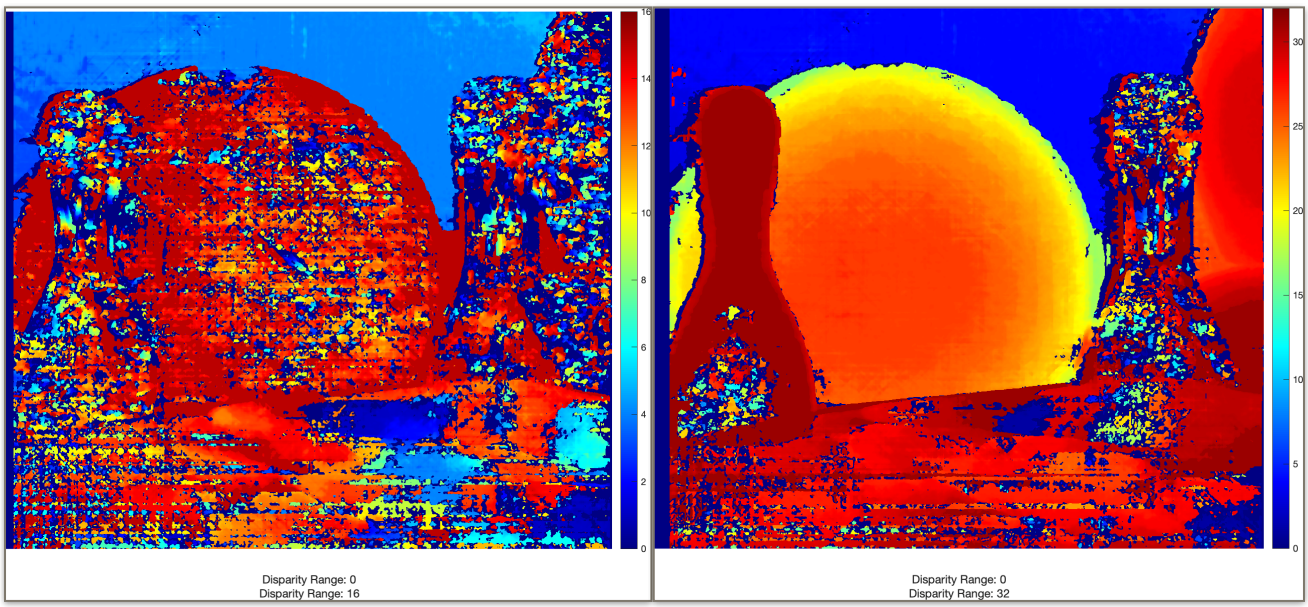
Very similar to the choices I made above and the approach methods I applied, I chose the result that best reveals the depth perception in these photographs. While giving the values and changing them according to the results I get, it is the same as the tactics I applied for the other images above. My preferred k and w values are w=60 and k=7 for this sequence of images.

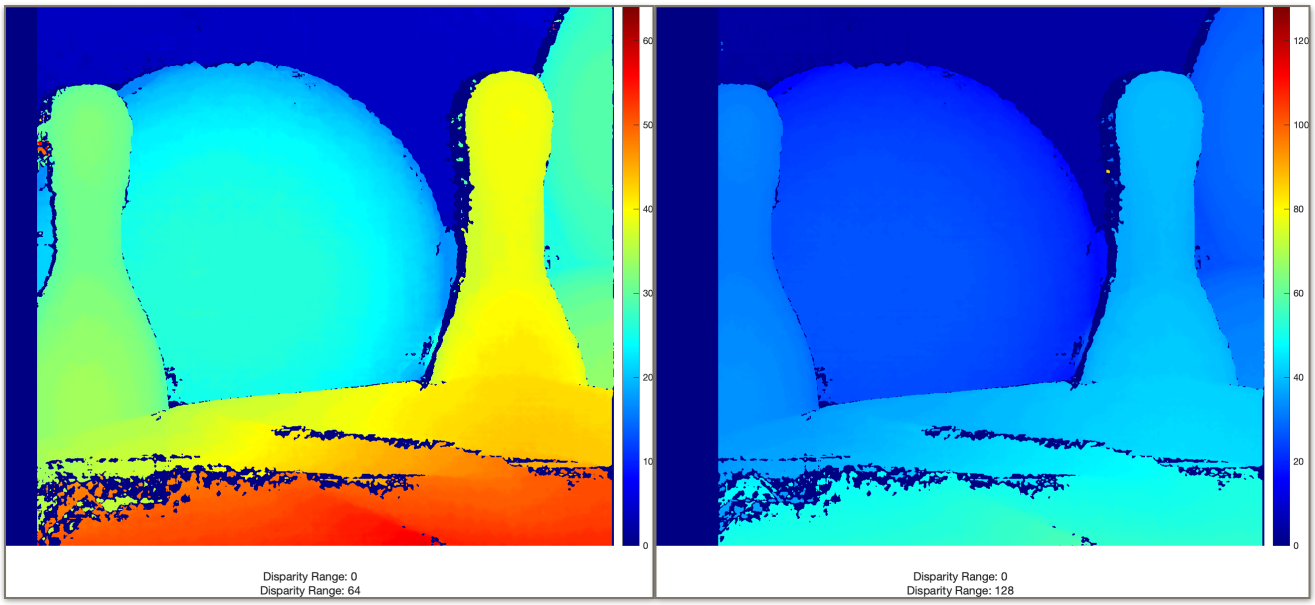
The discussion on built-in MATLAB function ‘disparity’.

```
%%
clc; clear all;
leftImage =  imread("left.png");
rightImage = imread("right.png");
[row, col, ch] = size(rightImage);
if (ch==3)
    %convert left and right images to grayscaled
    leftImage = rgb2gray(leftImage);
    rightImage = rgb2gray(rightImage);
end


a = stereoAnaglyph(leftImage,rightImage);

disparityRange = [0 16];

disparityMap = disparity(leftImage, rightImage, 'BlockSize', 7, 'DisparityRange', disparityRange);
figure;
imshow(disparityMap, disparityRange);
colormap jet;
colorbar
xlabel("Disparity Range: " + disparityRange);
```

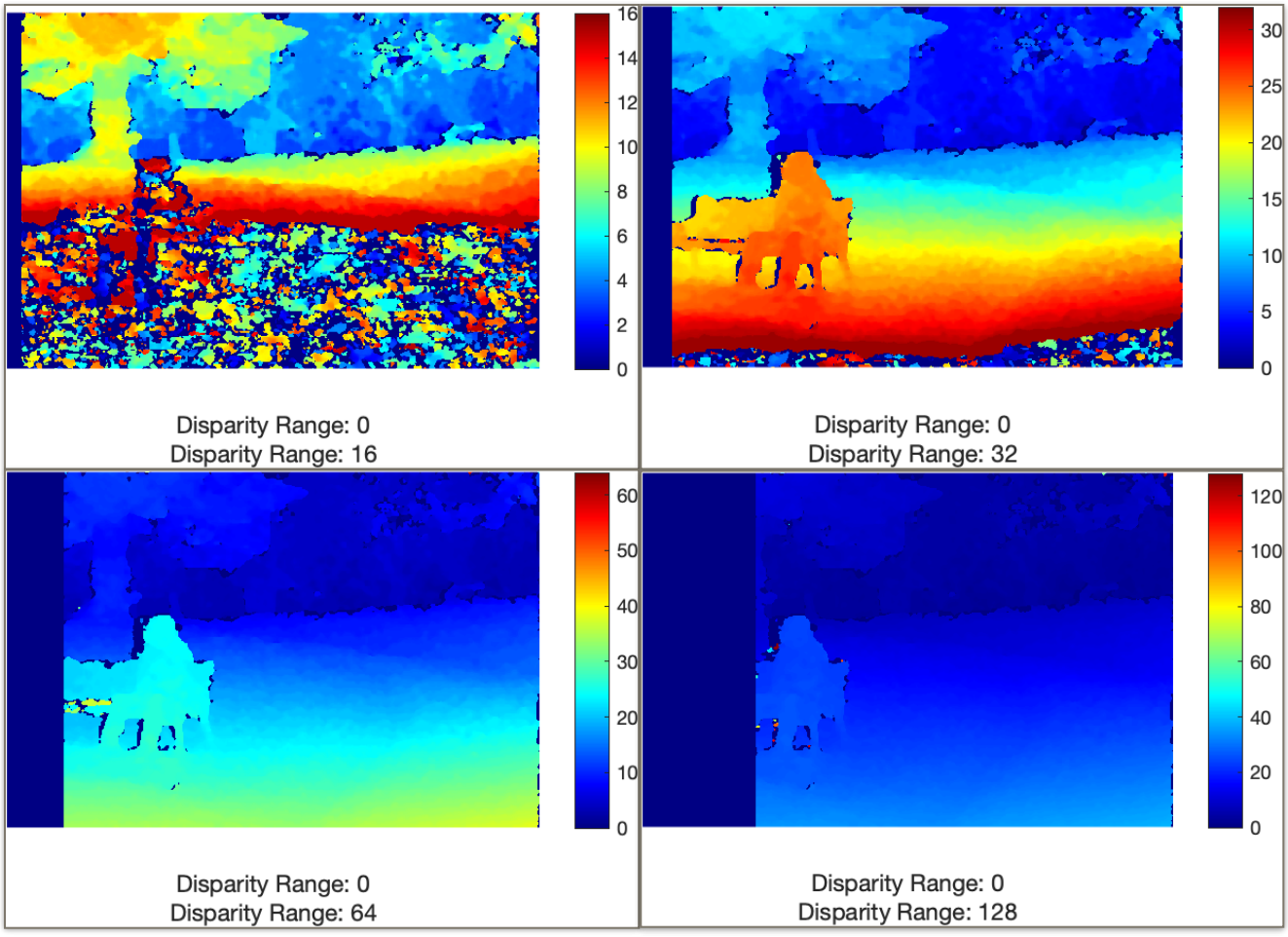
I wrote all the code above to use the 'disparity' function that is already built-in in MATLAB. In the beginning I read my pictures as usual and turned them to gray scale. Then I set the parameters inside the function and ran the code and looked at the different values. Here they are.

left.png and right.png





S00L.tif and SOOR.tif



Here, I choose the one with the disparity range [0 64] from the first four images, and the one with [0 32] from the other four results.

If I have to compare the algorithm I wrote with the built-in function, I cannot ignore the fact that the built-in one is hundreds of times faster. However, it is obvious that it is not as successful as the algorithm I wrote in revealing the details of the objects. On the other hand, I think that while he successfully conveyed the depth perception to us in general, he did not do much detailed work there either. For example, it could not deliver the depth differences in the area behind the man sitting on the bench or on the rough surface of the bowling ball towards us as successfully as the results in my algorithm. So, which algorithm I choose may vary depending on whether I pay more attention to processing time or details.