Assignment 4
Computer Vision

Stereo Vision

Nayera Abd El Samie ID:4032
Esraa Wael El-Hawash ID:3959
Guehad Mohamed Ahmed ID:3861

Stereo Vision

In this problem set we were asked to implement and test some simple stereo algorithms discussed in class. In each case we will take two images Il and Ir (a left and a right image) and compute the horizontal disparity (ie., shift) of pixels along each scanline.
This is the so-called baseline stereocase, where the images are taken with a forward-facing camera, and the translation between cameras is along the horizontal axis.

There are 2 main ways that we implemented in this report we will mention how we implemented them and the results they showed.

1. Block Matching

To get the disparity value at each point in the left image, we will search over a range disparities, and compare the windows using two different metrics
Sum of Absolute Differences (SAD) and Sum of Squared Differences.

The essential idea in block matching is we take a particular block in the left image and in the opposing strip in the right image we search block by block for the block that matches the one chosen in the left image. This block is like a window that we decide the size of.

The algorithm:
we start by reading the left image and the right image, we convert them to Gray Scale images then we turn them to np arrays.
Next we take the shape of the left image and put in variables (width and height).

We loop over the height of the image and we divide the image to strips of height=K say the K=3
so the strip would have a height of 3 and width of (the image width)
we do this for both images and now we have a left strip from the left image and a similar but opposing right strip in the right image.

Next we call a function called ‘Match_Strips’ and we send as parameters the 2 extracted strips.

The Match_strips function takes the left strip, the right strip and the kernel size
first thing defines a number of blocks in this specific left strip and initializes an array of zeros with the same size of number of blocks.

Next a loop from 0 to number of blocks-1 takes place and the first x co-ordinate is taken as the iterator of that loop.
Using that x coordinate we identify our first block it has a size of 3x3

Then we send this block and the whole right strip to a new function called Best_Match()
this function takes a block from the left and the strip similar to it on the right and keeps searching the whole right strip for the perfectly matched block and this best match is decided upon the calculation of SSD(Sum of Squared Differences) and SAD(Sum of Absolute Differences) and minimal value after calculation of all blocks is the best match (the least different)

Best Match function takes as parameters a block and a strip.
A min variable is initialized with infinity to help us find the minimum value of ssd/sad

we loop on the right strip and we extract ‘Non-Overlapping’ blocks of kernel size and then we subtract those blocks from each other, square or take their absolute and then we sum the values and this is saved in another variable.
Finally we keep comparing all sums outputs and the minimum value’s coordinate is saved and returned from this function to the matching strips function.

In turn the matching strips function takes the right coordinate and in our previously initialized array subtracts the left x coordinate from the right x coordinate and saves the result in that array, and returns the final array after finishing all calculations for all blocks in this left strip.

Each time this function returns an array between those to strips, the loop moves one pixel down and takes 2 new strips and sends them and the whole thing repeats itself.

Lastly every array returned from matching 2 strips is stacked against the one before it and then in the end we are left with the disparity map of the whole image and that’s the output we then show.

Results:

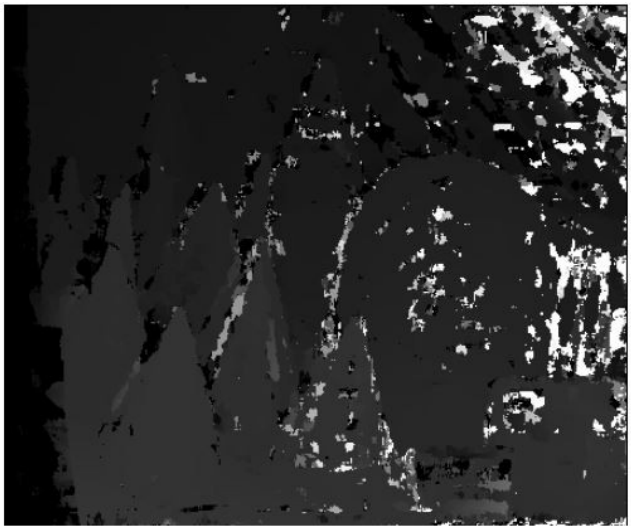
Original Image 1:

Left

Right

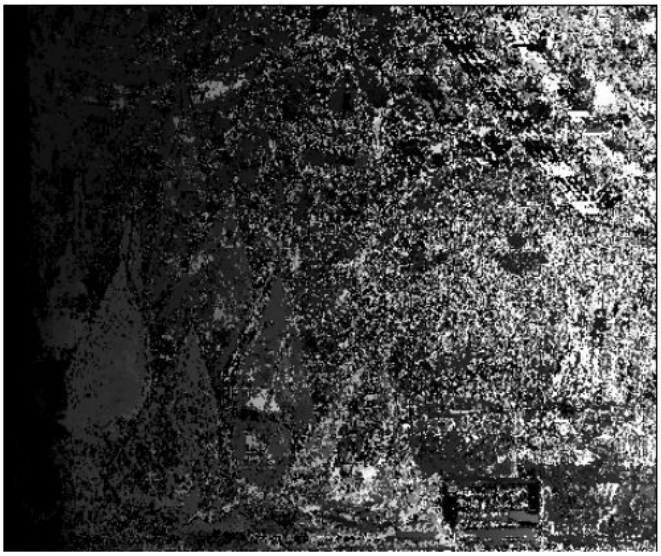
Gray Scale Left image:

SSD K=9



SSD K=5

SSD K=3



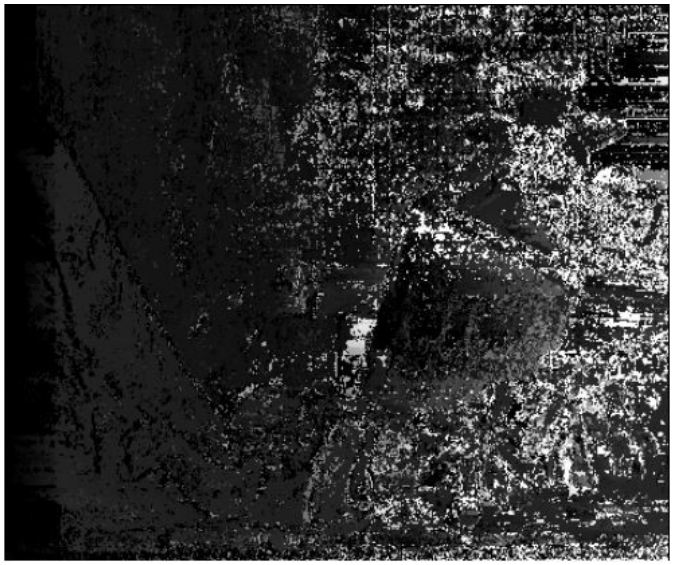
SAD K=9

SAD K=5

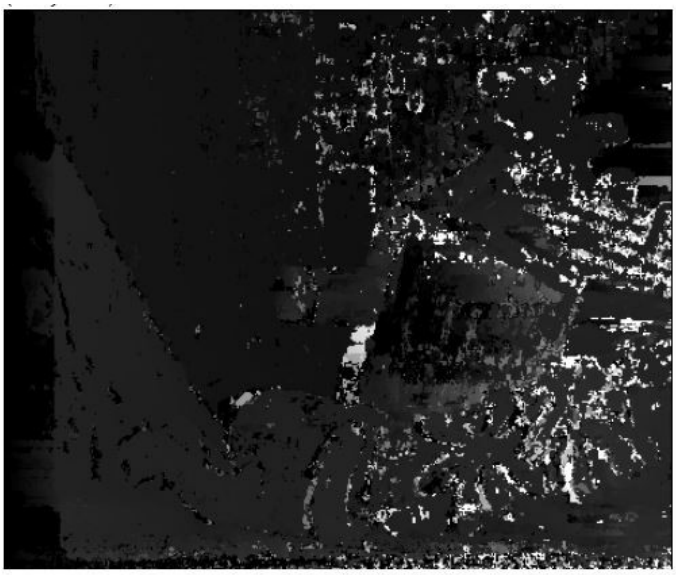
SAD K=3

Second image original

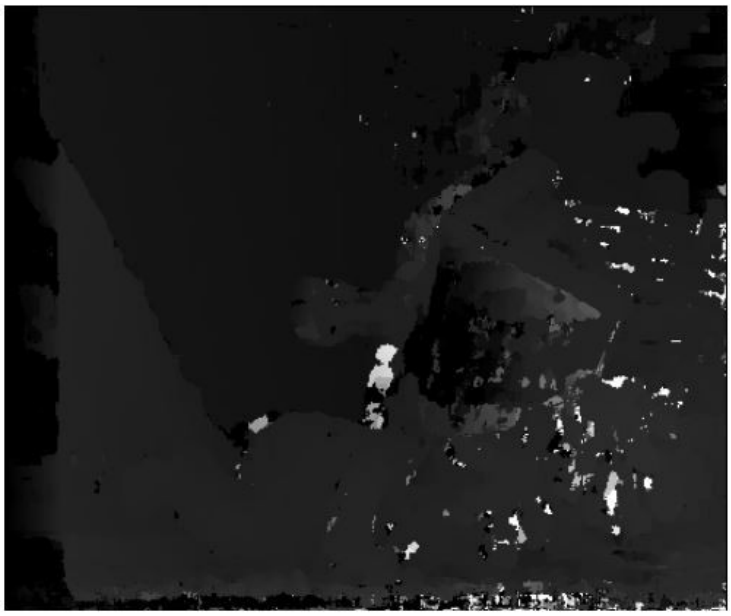
SSD k=3



SSD k=5

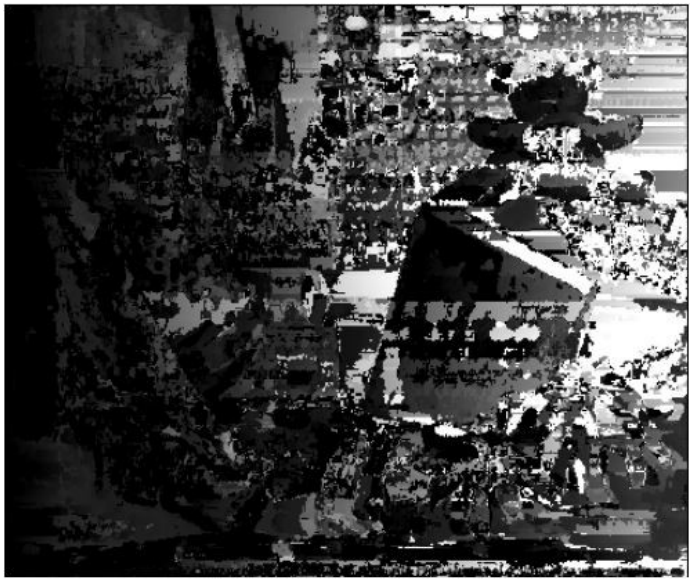


SSD K=9



SAD K=3

SAD K=5



SAD K=9

2.Dynamic Programming:

The essential idea in this technique is rather than taking blocks and comparing them we take 2 scanlines from each image and by scanline I mean a row of pixels and we work on each pixel individually, this gives out more accurate results and better outputs/ disparities.
To do that we take the row of pixels of the left image as vertical and the row of pixels from second image as horizontal and fill the cost matrix between pixels.
To fill the cost matrix we need to initialize the first column to the matrix in addition to the first row, then picking the first pixel from the left image to calculate the cost of it with each pixel in the row of the right image.

The algorithm:
We start by reading both images and converting to gray scale.

Then we implemented the di,j function mentioned in the pdf $d_{ij} = \frac{(I_l(i) \quad I_r(j))^2}{\sigma^2}$

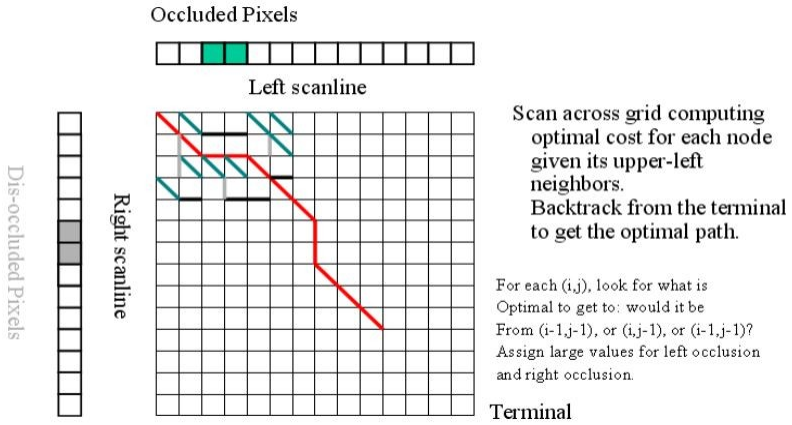
to compute the mean squared error between pixels.

Next we loop on both images and we calculate this function

$$D(i,j) = min(D(i-1,j-1) + d_{ij}, D(i-1,j) + c_0, D(i,j-1) + c_0)$$

we get each term and we compare to get the minimum and we save that
in the same step we check to see if the minimum was the first term we add 1 to a new matrix in that position and if it's the second term we add a 2 and if it's the third term then we add a 3.
next we initialize two variables with the width of image and in a while loop we backtrack starting from the last pixel to build the path and simultaneously build the disparity matrix.
We initialize a value called Match Value=0
We check if that pixel in the temporary matrix i=1 then the path moves diagonally and there's a match and the match value is added in our disparity matrix, if the pixel=2 then a pixel was skipped in the left image and a left shift is made in the path and we add 1 to the match value and add it to the disparity map and if pixel=3 a pixel was skipped in the right image a right shift is made in the path and we subtract 1 from the match value and add it to disparity map.

Once all this is over we show the Disparity map formed.



Pseudo-code:

```
for(i=1;i≤ N;i++){
   for(j=1;j≤ M;j++){
      min1 = C(i-1,j-1)+c(z_{1,i},z_{2,j});
      min2 = C(i-1,j)+Occlusion;
      min3 = C(i,j-1)+Occlusion;
      C(i,j) = cmin = min(min1,min2,min3);
      if(min1==cmin) M(i,j) = 1;
      if(min2==cmin) M(i,j) = 2;
      if(min3==cmin) M(i,j) = 3;
   }}
```

```
p=N;
q=M;
while(p!=0 && q!=0){
   switch(M(p,q)){
      case 1:
         p matches q
         p--;q--;
         break;
      case 2:
         p is unmatched
         p--;
         break;
      case 3:
         q is unmatched
         q--;
         break;
   }}
```

Results: