Question 2-

For this question, I started with the middlebury pair images. First, I convert them to grayscale images, using im2gray function in matlab. I chose to find the disparity map using a window based (block based) approach. So, I iterate over different patches of the left image, and find a corresponding patch in the right image that has the minimum error cost. This kind of algorithms has the benefit of being simple in implementation. On the other hand, graph max flow algorithm can be used to find the optimal flow and the disparity map will be globally optimum, while it is hard to implement since the ford-falkerson algorithm to find the max flow is not implemented in matlab.

Window-based algorithm needs a defined cost between two windows. There are three options: 1- minimize the sum of absolute difference between intensities of pixels in two windows. 2- minimize the sum of squared difference (error) between the intensities of pixels in two windows. 3- maximize the normalized cross-correlation between intensities of pixels in two windows.

Cost: I chose to consider two options. I minimize the sum of squared difference (error) of two windows and also maximize the normalized cross-correlation between them. They both have pros and cons. SSD (sum of squared error) is sensitive to pixel intensities and scale, while it is relatively robust to changes in overall contrast between images. NNC normalizes the correlation by dividing by the product of the standard deviations of the two patches. The resulting values for NCC range from -1 to 1, where 1 indicates perfect positive correlation, -1 indicates perfect negative correlation, and 0 indicates no correlation. NNc is more robust to intensity scaling. Therefore, I preferred to use both, one to be minimized, and one to be maximized.

I preferred to implement the normalized cross-correlation myself. So, the numerator of cross correlation is the sum of multiplication of the difference of each block and its average. The denominator is the squared of multiplication of standard deviation of blocks. This should be maximized, so we check in a search range to see when this critera is maximized. At the same time, the SSD should be minimized.

Smoothness term: I compute the smoothness term as the absolute difference between the disparity of previous window in each direction. Smoothness term is necessary to make sure that there's not big gap between adjacent box disparities. So, if the absolute difference between adjucent disparities is minimized, we reach this goal. I do this for both x, y directions. I also got inspired by this repository for the smoothness term: https://github.com/vinceecws/Monodepth.

$$C^l_{ds} = \frac{1}{N}\sum_{i,j}\left|\partial_x d^l_{ij}\right| e^{-\left\|\partial_x I^l_{ij}\right\|} + \left|\partial_y d^l_{ij}\right| e^{-\left\|\partial_y I^l_{ij}\right\|}.$$

So, I consider the derivative of d as the difference between consequent ds, as I discussed before, and also multiplied by the exp of minux derivative of the current left block. This enforces the smoothness term to be smaller when we face an edge (whenever the gradient of block is large), and be equal to the difference between consequent ds, when we see a uniform structure in the blocks.

The final cost contains the smoothness term and the SSD, they both should be minimized. I multiply the SSD to 0.1, as the result shows it is a proper coafficient. We also consider lambda as a coaffient to change the effect of smoothness term in the final cost.

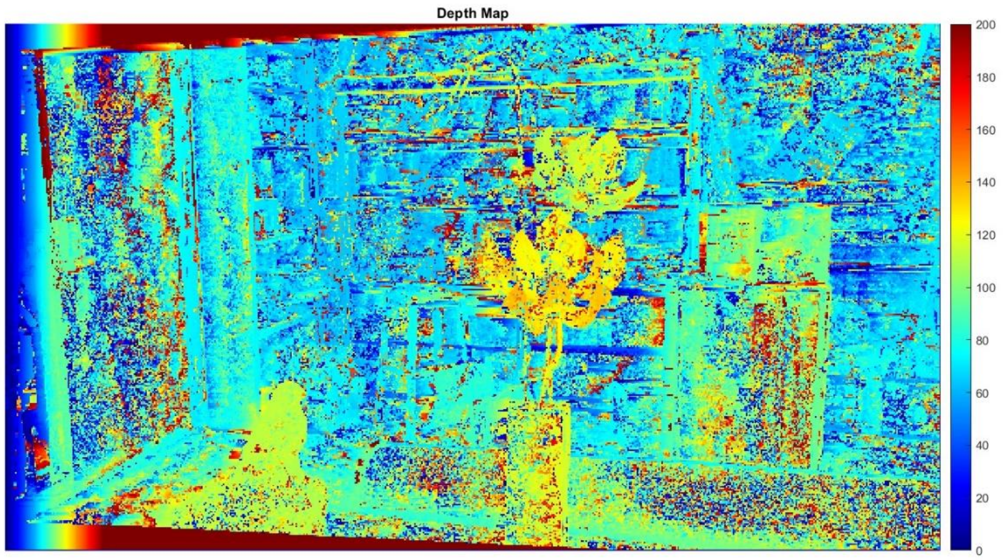
Uniqueness: To limit our results to be unique, there are two different approaches that can be considered. The first one is when we want to match blocks, if the right block is already assigned to another left block, we just skip. It means that right block is always selected uniqely. This algorithm needs to store aa disparity map for the right image as well. I tested this approach but the result was not suitable since this is a greedy algorithm and not considered the optimum solution. It just says if the right block is assigned before in the iteration order, don't assign it anymore, while the cost might be less in the latter potential assignment. I next examined another approach that came to my mind. I put a threshold for the cost to ensure uniqueness in the assignment. So, if the cost is less than a threshold, it means that fewer options are considered in assignement of blocks. If the cost is less than the threshold, only 1 block may be considered as the corresponding block in the left image. I mean, the threshold on the cost is a way to cut some inappropriate assignments and make sure that the correspondence is unique. However, finding a good threshold is tricky, but it can be set according to appearance of results or the average cost. I know this might not be the standard way of checking the uniqueness, but it came to my mind and I tested and worked pretty well.

Monotonicity: For this part, I considered if the monotonicity pattern of corresponding rows in two blocks are the same (or almost the same) or not. I iterate over different rows of the left and right blocks, and compute the difference between Consecutive elements (using diff function), and using the sign function, I compute what percantage of them have the same pattern (increasing/decreasing). I also put a threshold to say if the pattern is similar more than that threshold (80%), you can consider that as a similar pattern.

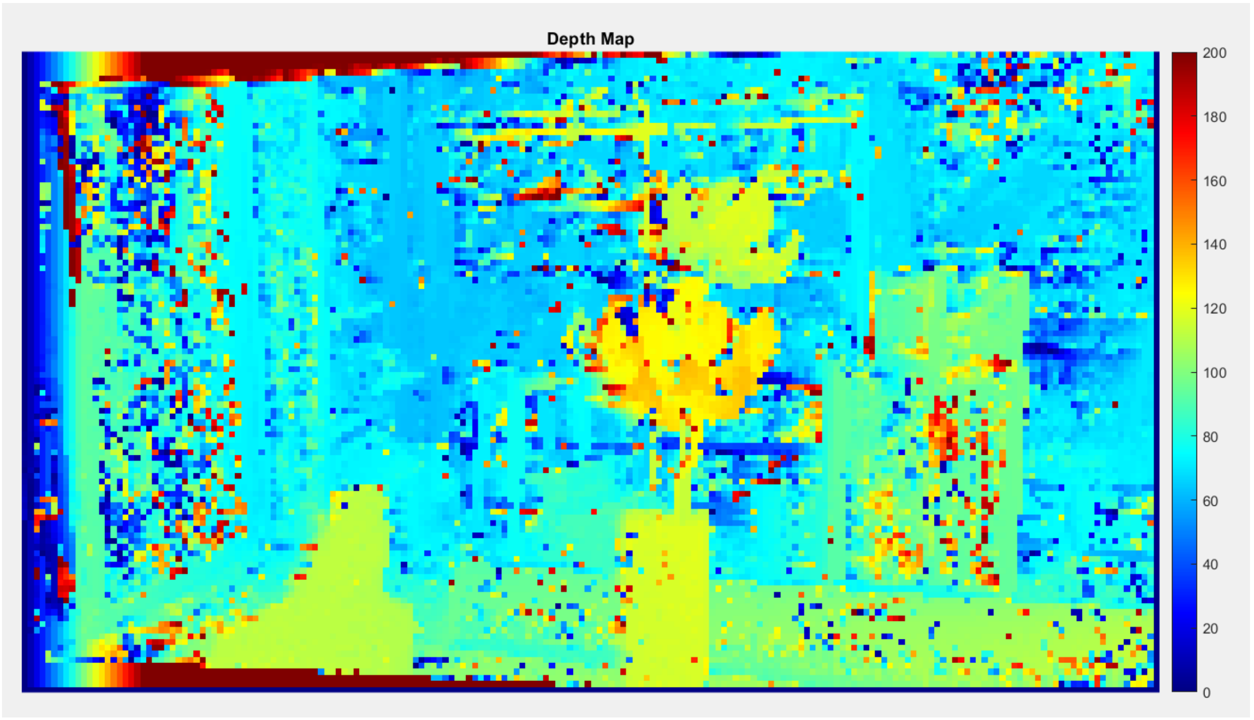
Up/down moves: My primary experiments only contains searching over a range of columns to find the disparity. Then, I chose to go over a very small number of neighbor rows to search over the best matching block. So, there's a for loop over a search range for finding best matching column, and inside the loop, we loop over small number of rows (up/down). This may help if there's an error in matching to be considered.

Experiments:

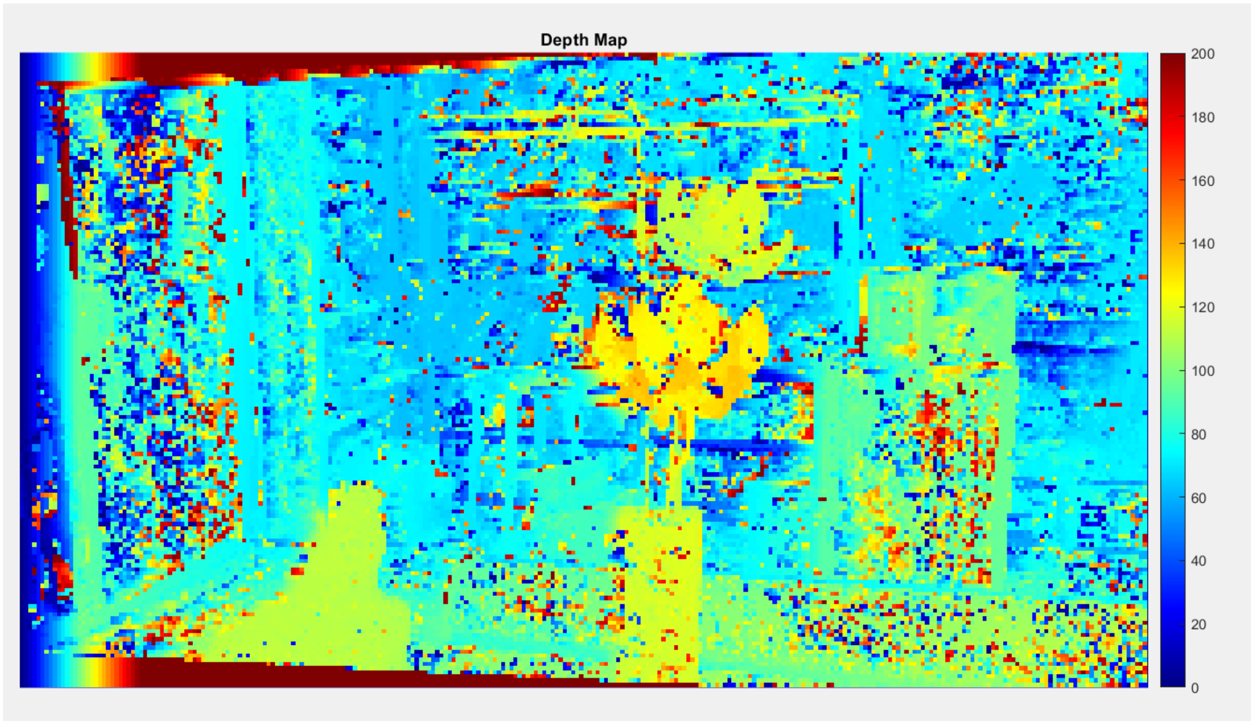
What would be the disparity map if we only consider search over rows (left/right) and consider only the SSD cost without any constraints? Window size = 3, search range = 200. Disparity map would be:



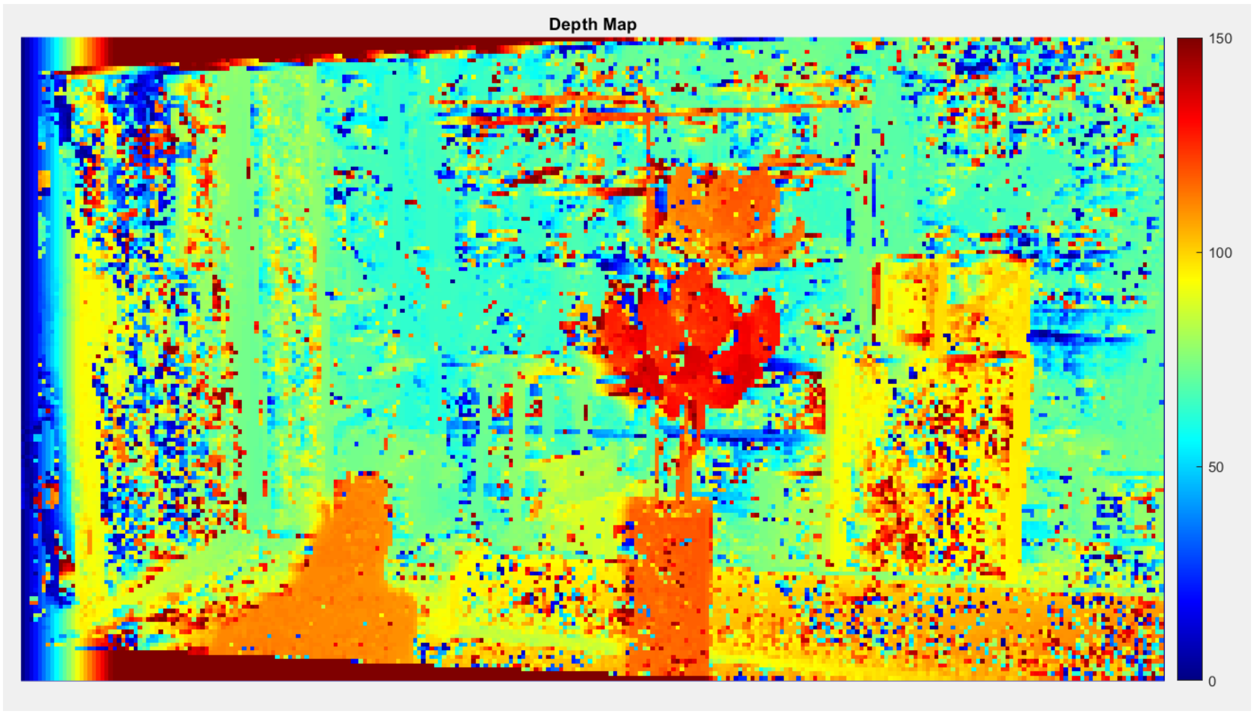
And what if the blocksize = 10.



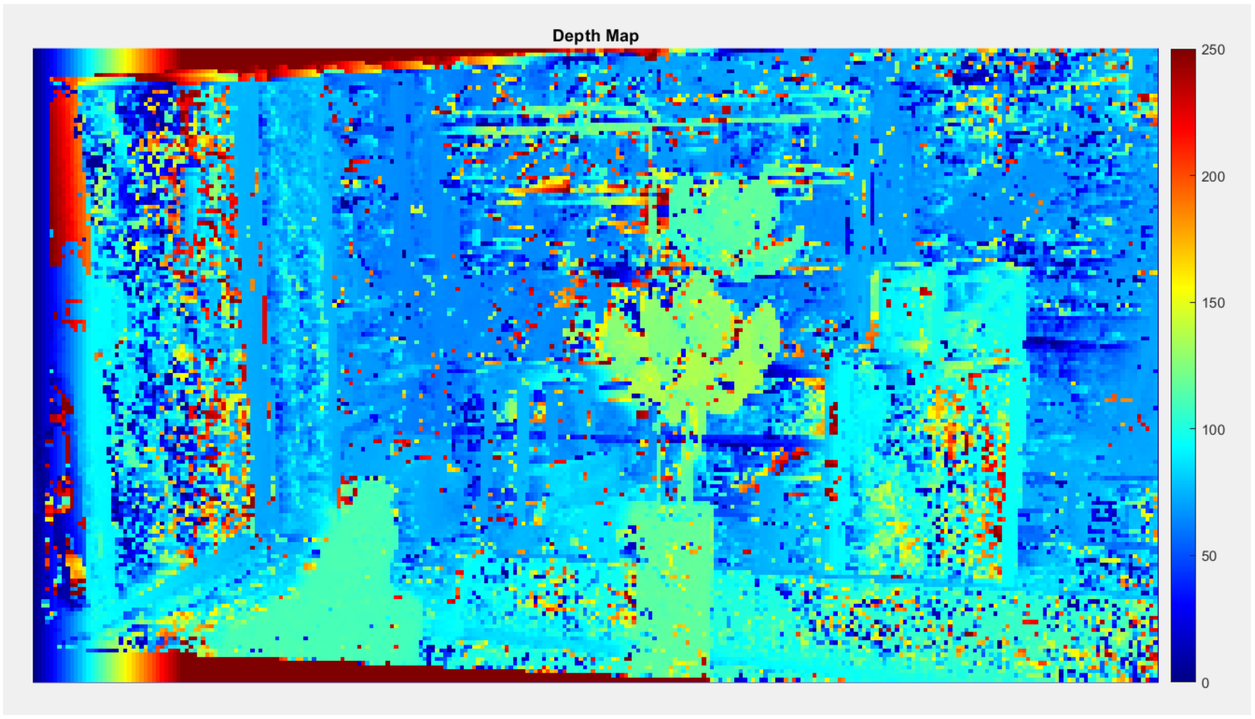
There's a trade-off between different block sizes. If the block size is too small, this gives us a noisy representation, while if it's too large, it won't give us a detailed representation. So, we examine blocksize = 7



I have the blocksize = 7, but the searchrange = 150:

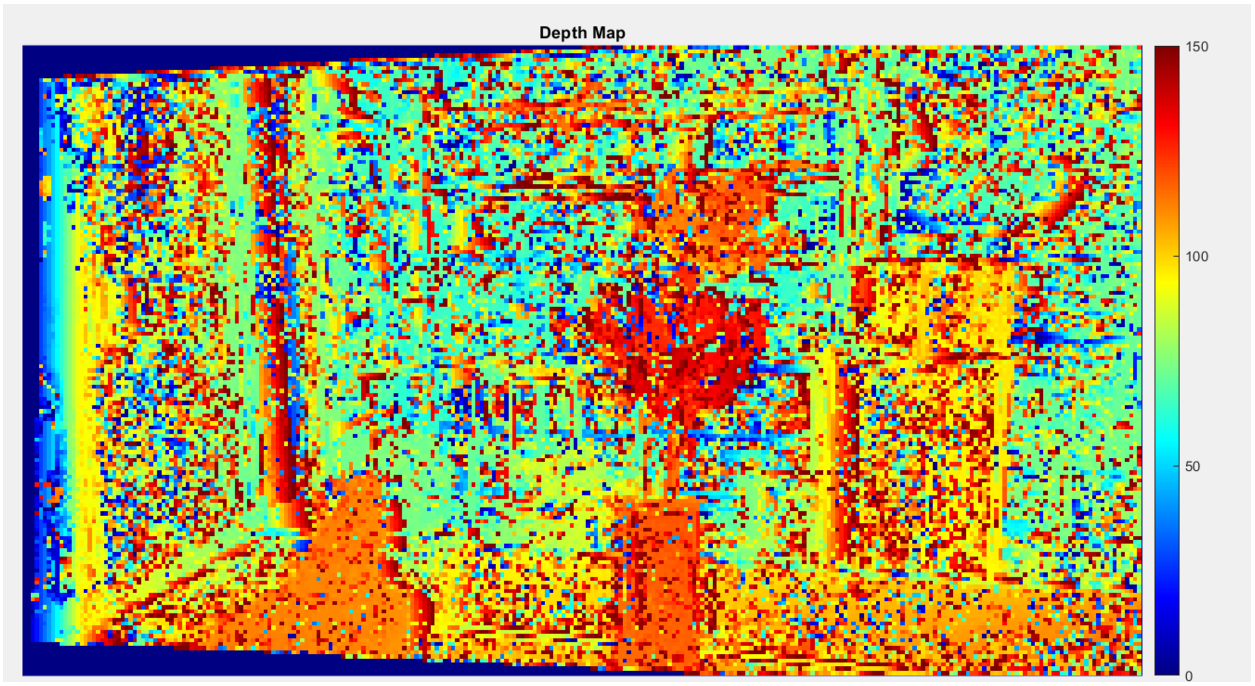


What if the searchrange = 250:

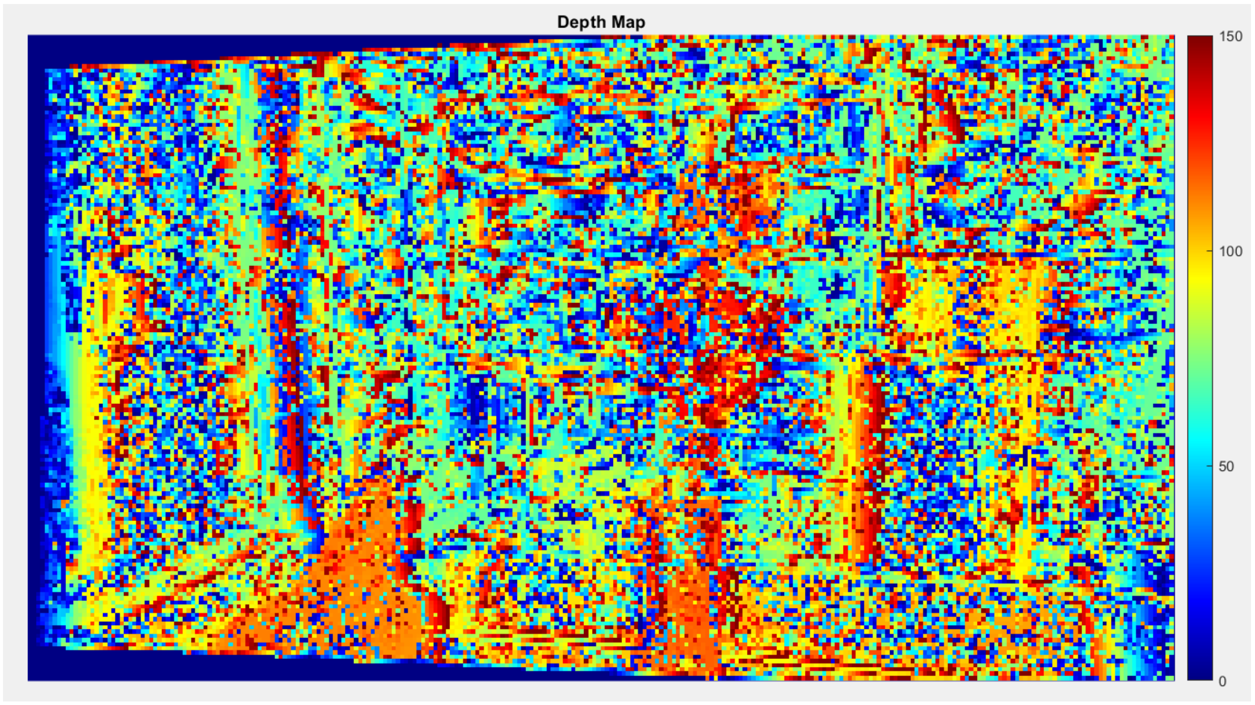


I chose searchrange=150, and the blocksize=7.

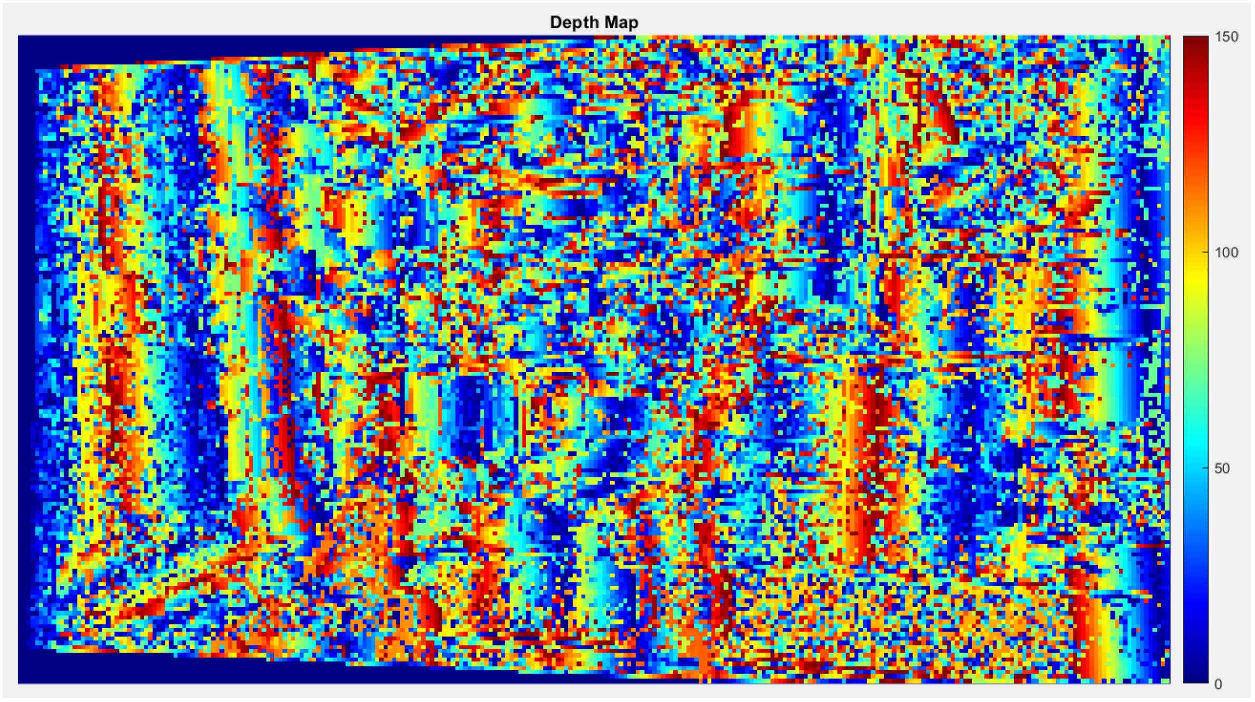
Now, I just add the **normalized cross-correlation** term that should be maximized. This is the result:



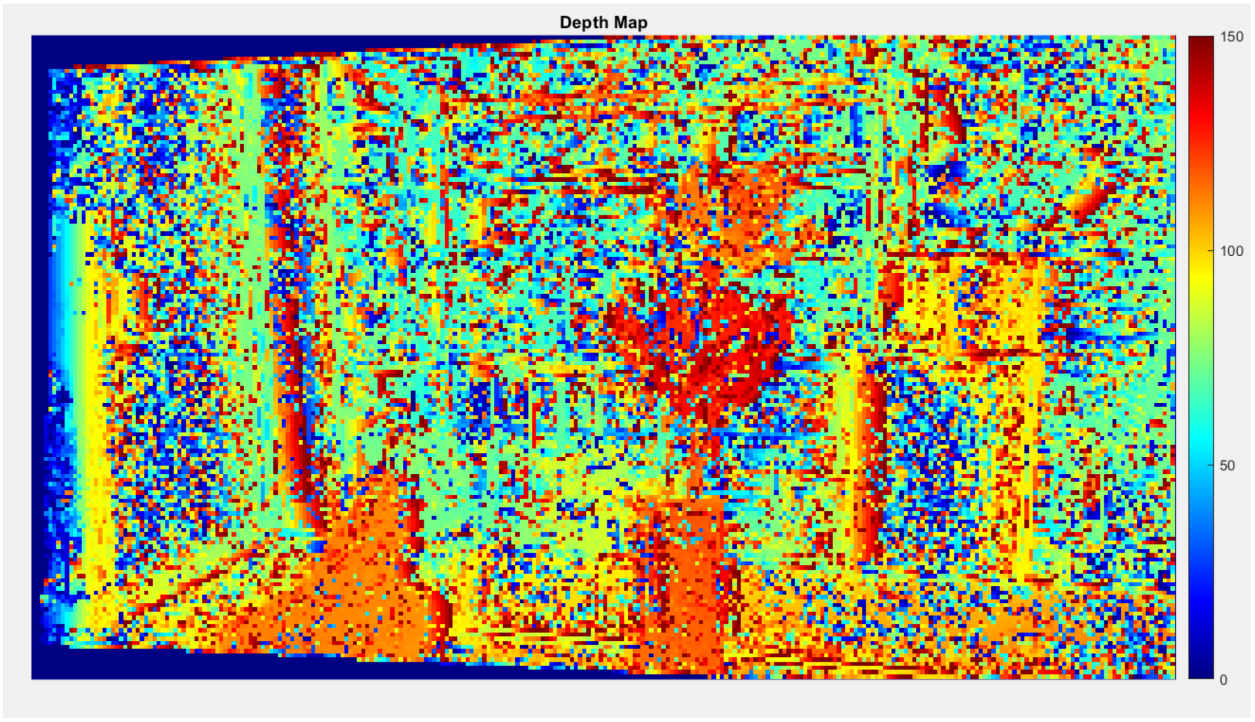
Now, I will add the **smoothness term**. Using lambda = 3, we'll get:



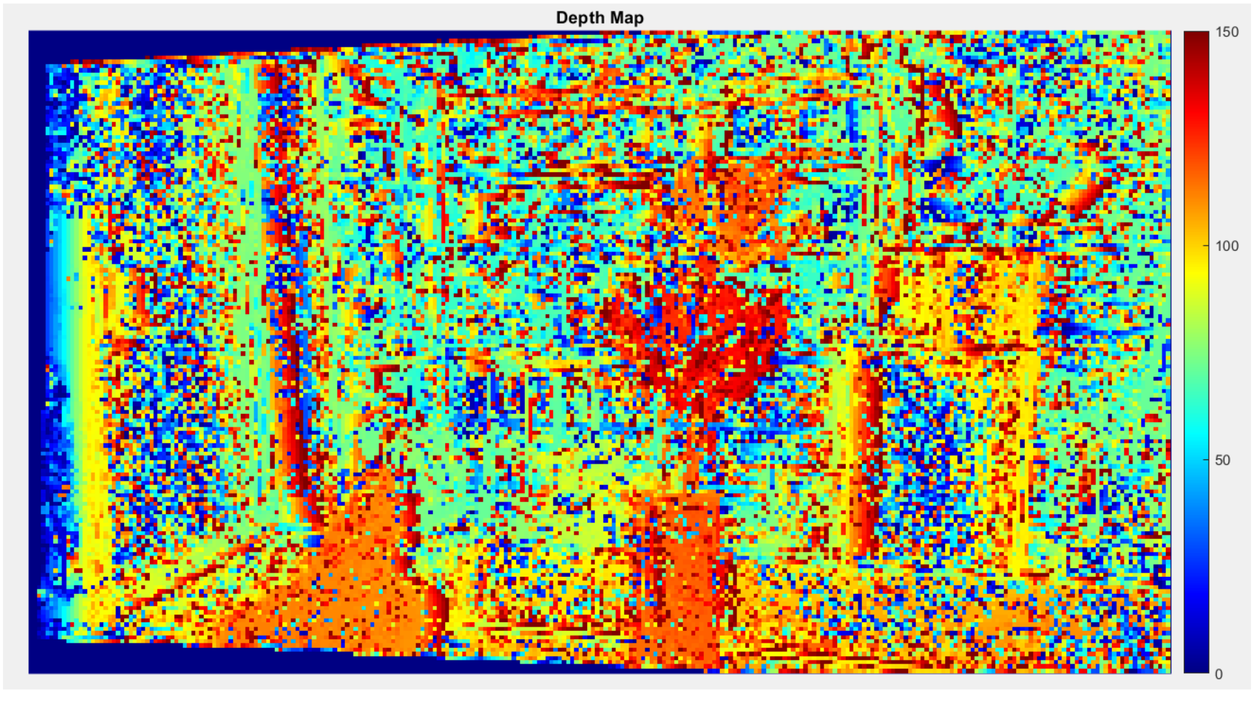
Using lambda = 10, we'll get:



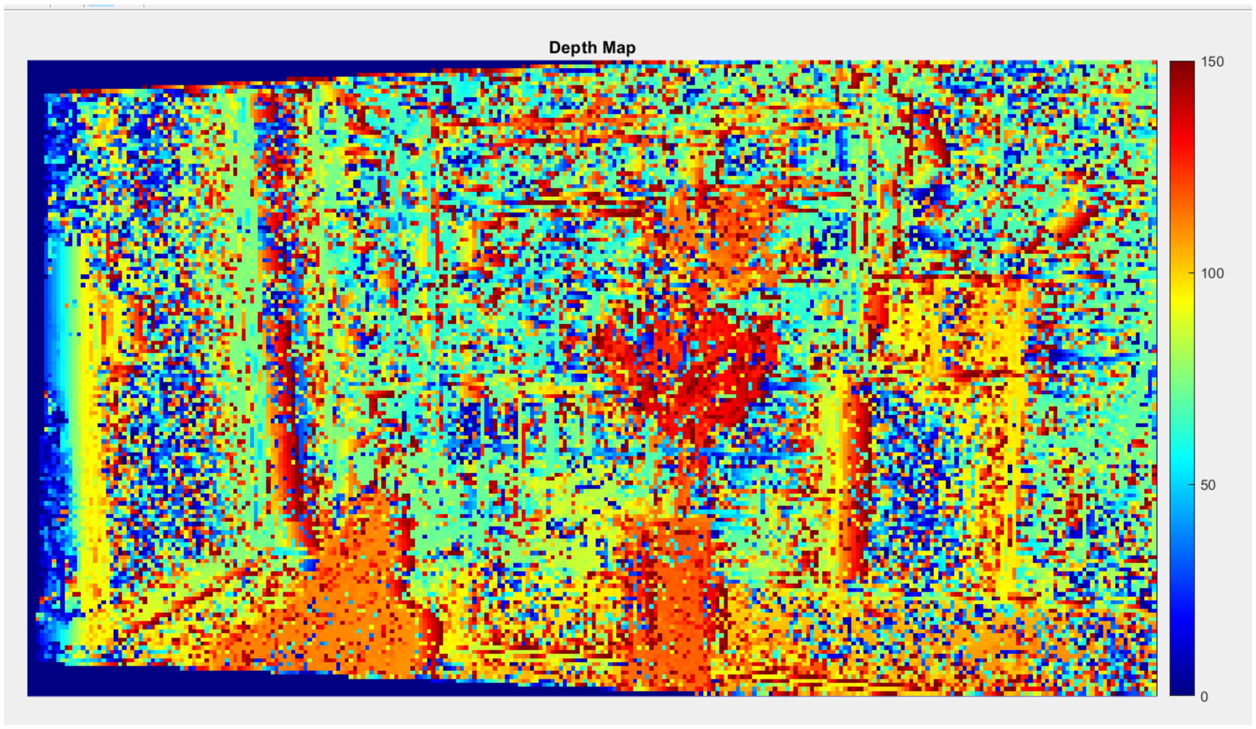
Using lambda = 0.5, we'll get:



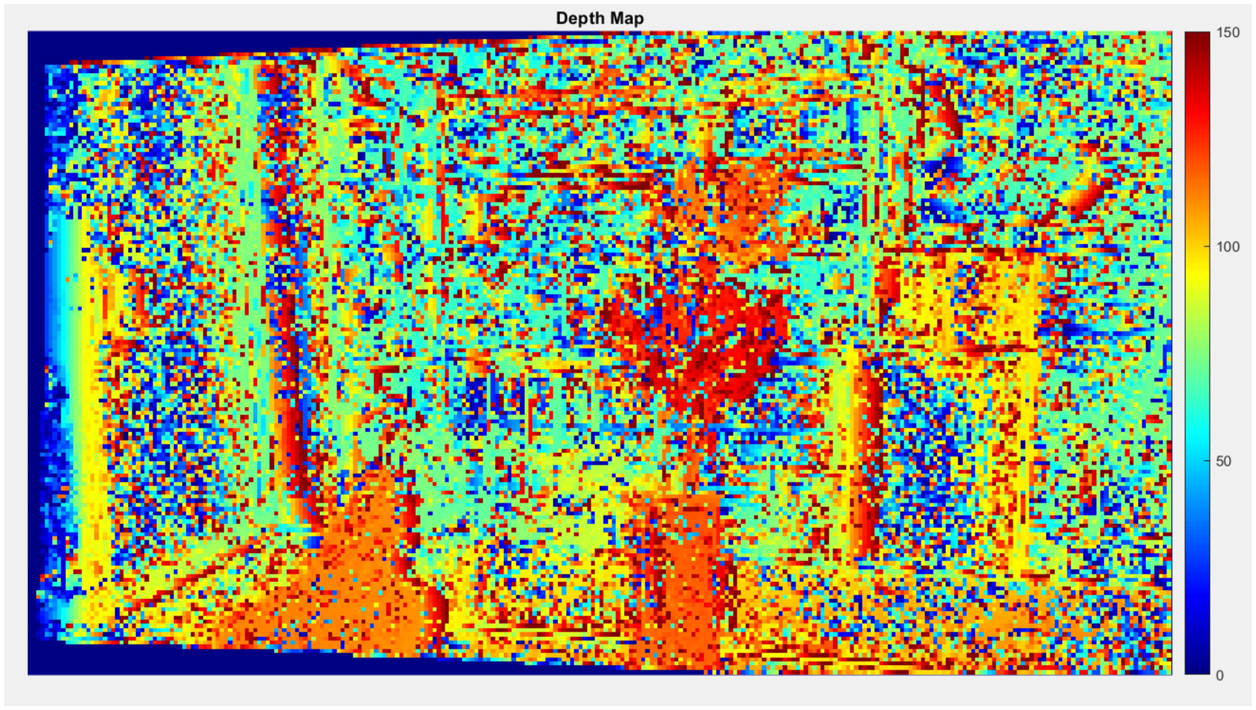
Now, what if we don't use the multiplication to exp(-grad) in different directions. This is the result:



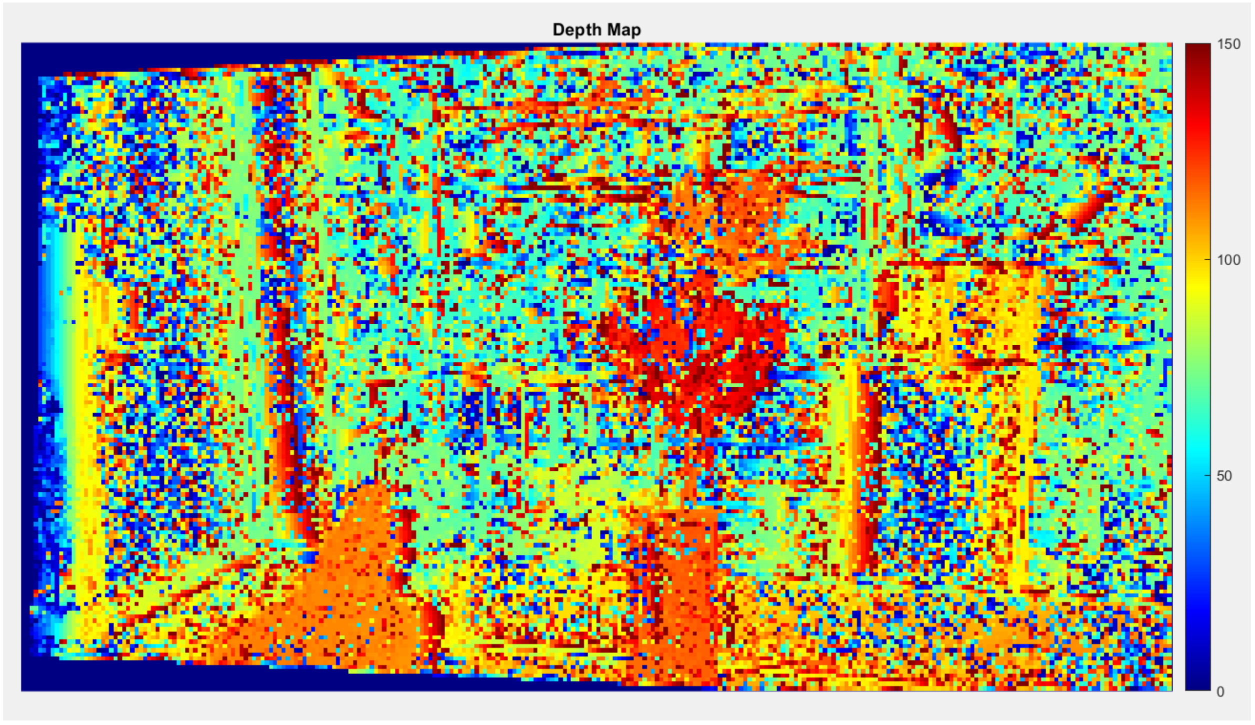
I now add the **monotonicity**, with the threshold 80%:



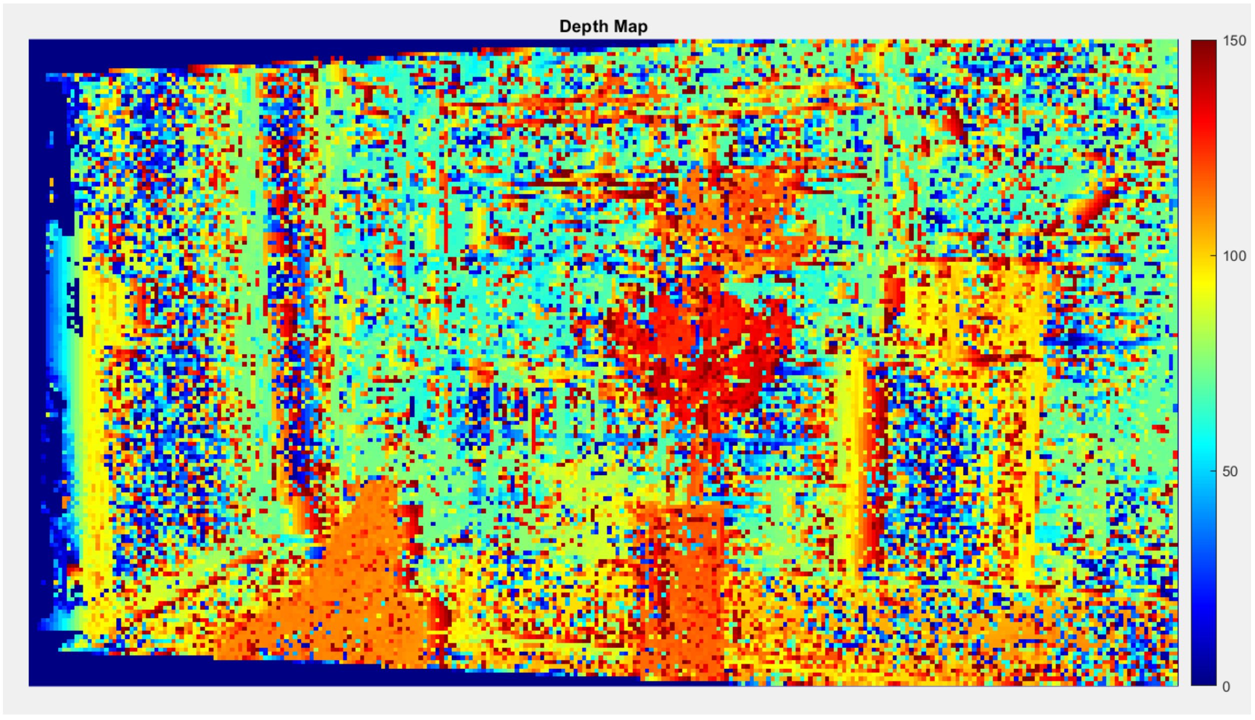
And when the threshold is 1:



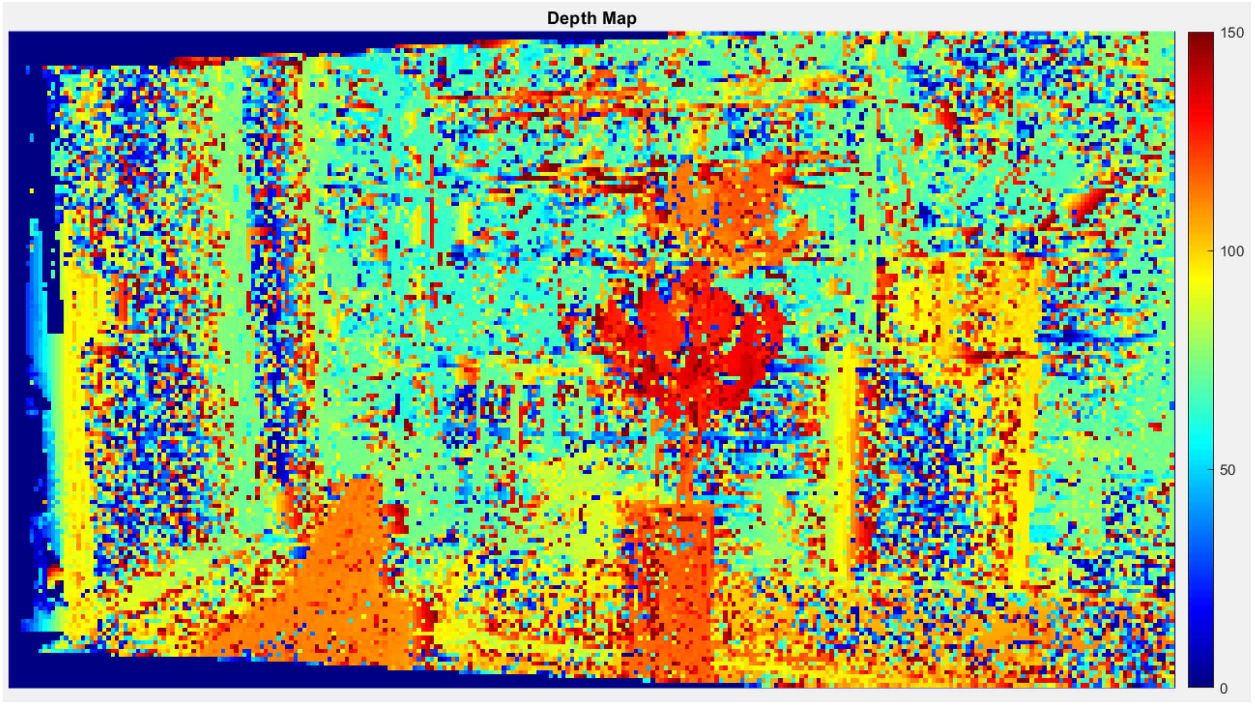
Now I add a threshold on the cost to make sure about the uniqueness. When the threshold is 1800:



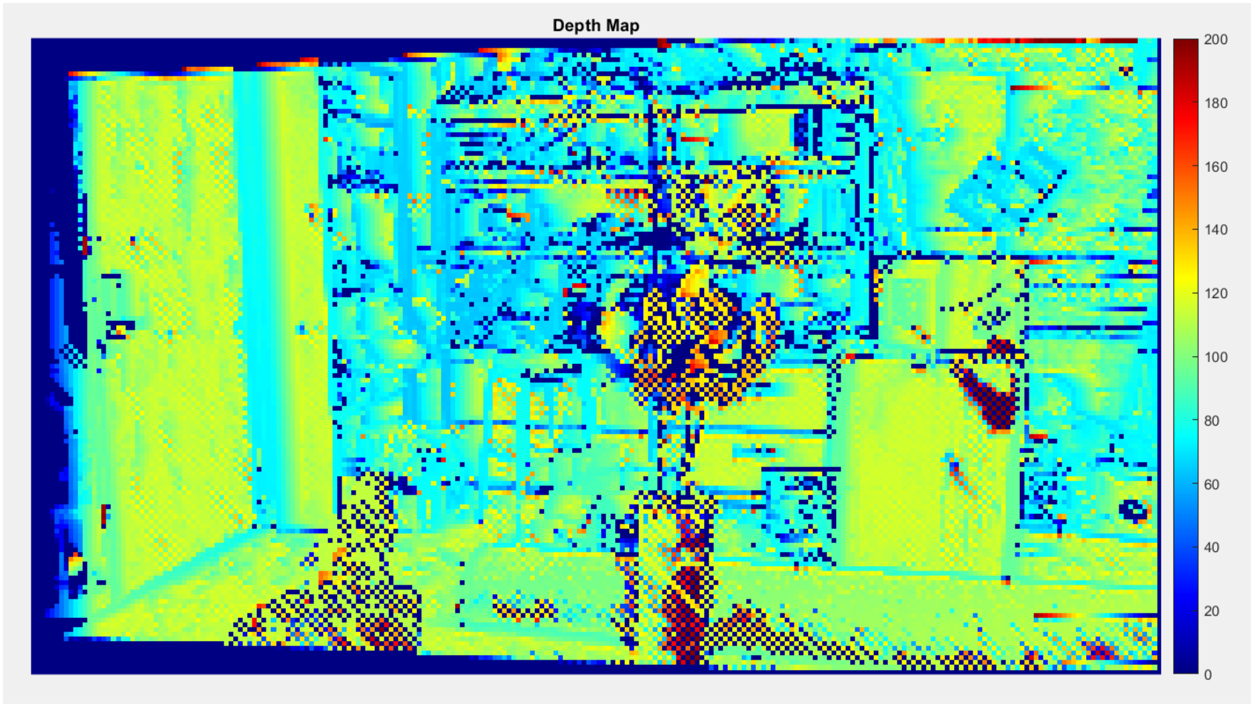
When the threshold is 1500:



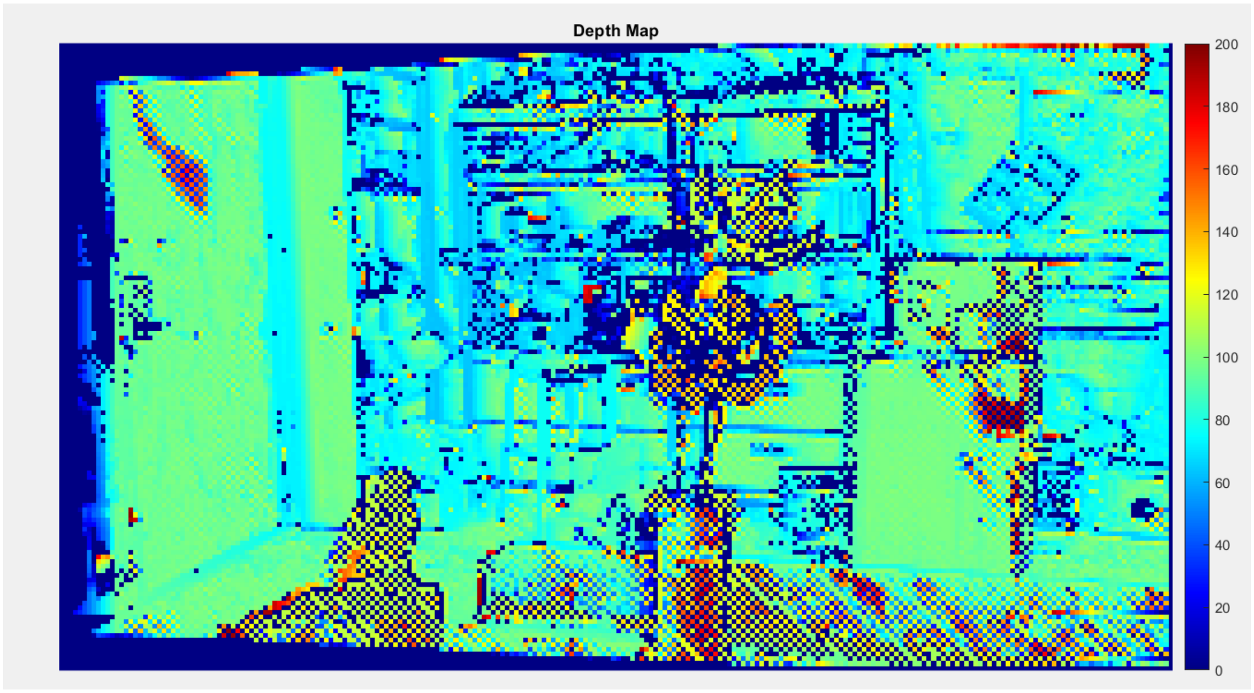
When the threshold is 1200 :



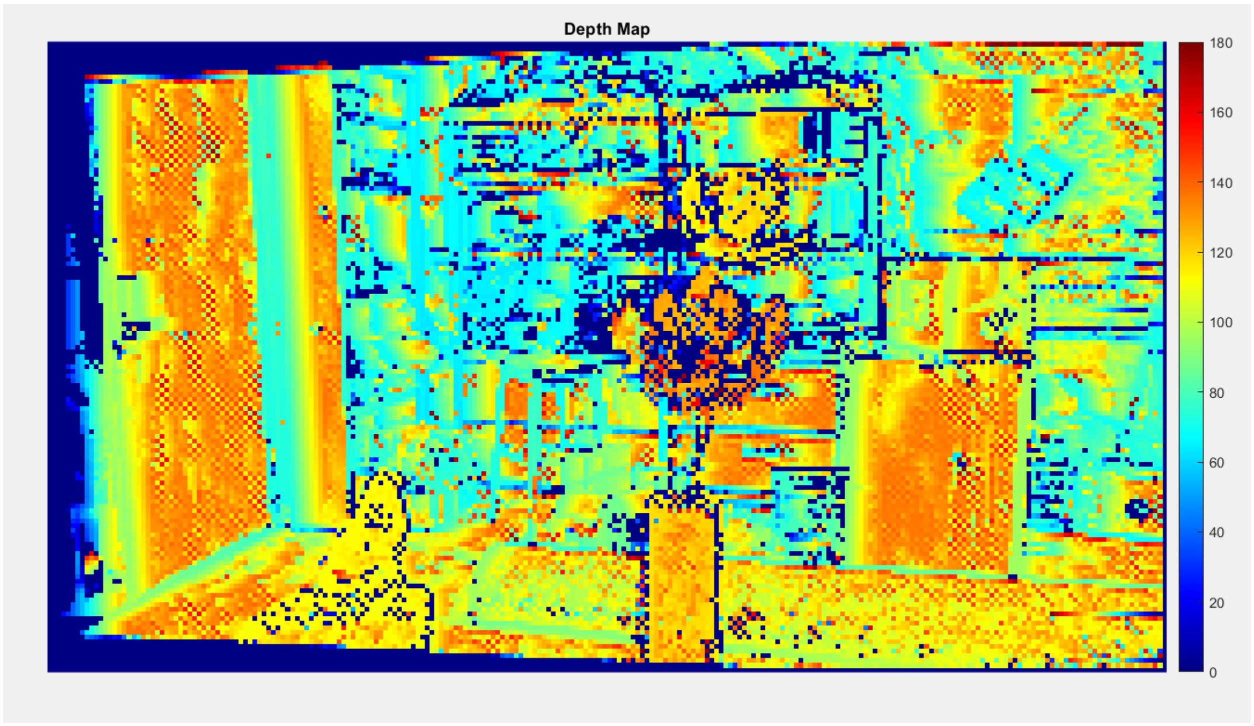
I made another experiment based on different hyper parameters. If the blocksize=8, maxrange of search = 200, lambda=3, and the cost threshold= 1400, this would be the result, which works better on background objects.



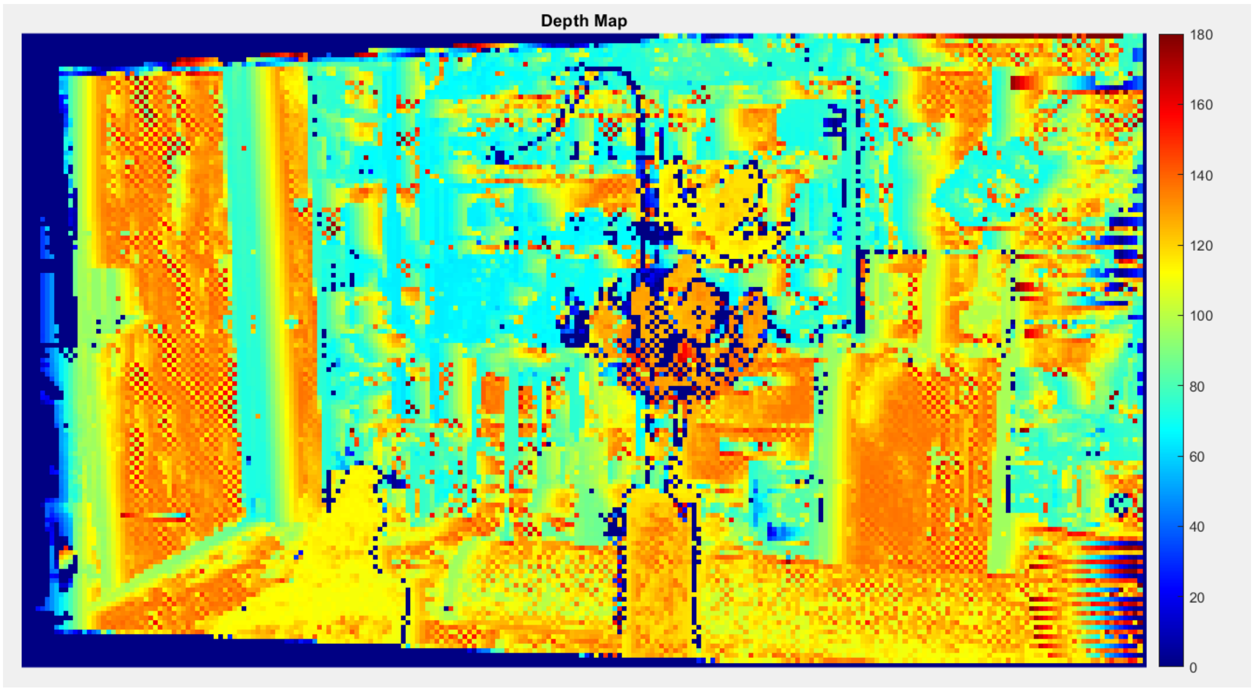
And with cost threshold = 1200 :



And with cost threshold = 1100 and lambda=2 :

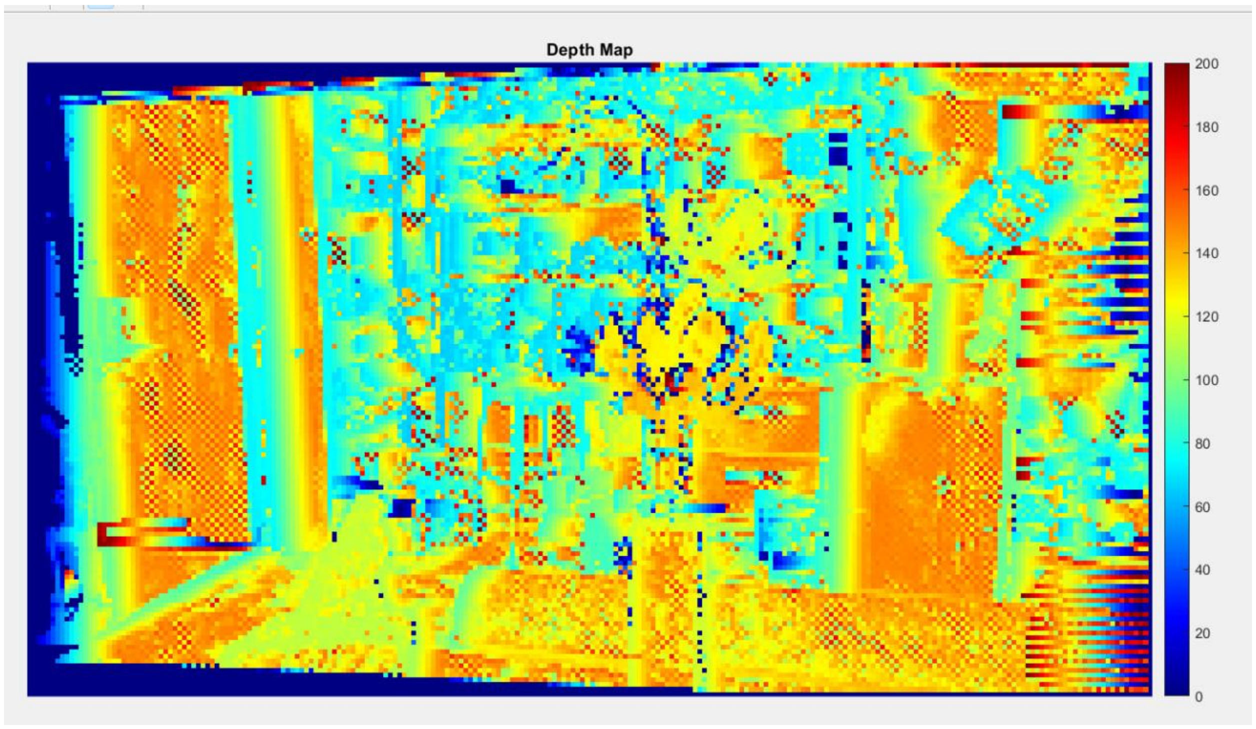


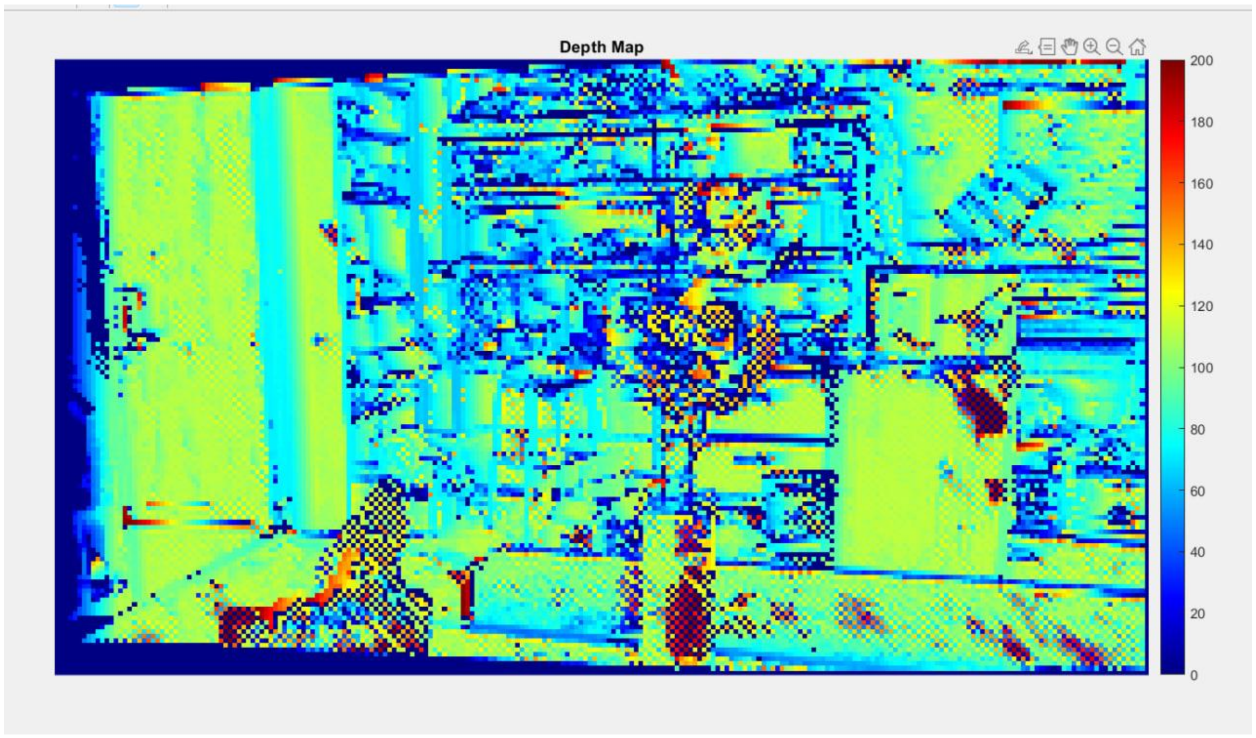
As the final step, we add the up/down move to have wider range of changes. If the amount of search for column wise search is 4, with the last parameters, we get this result:

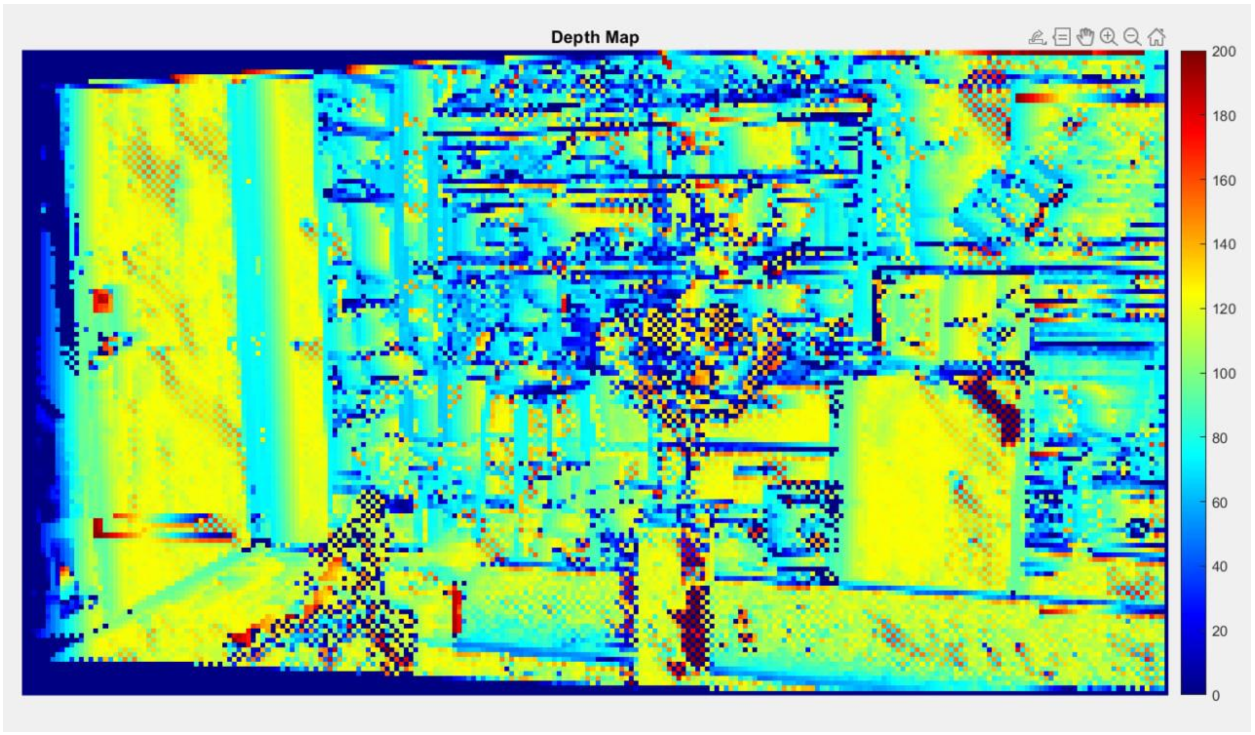


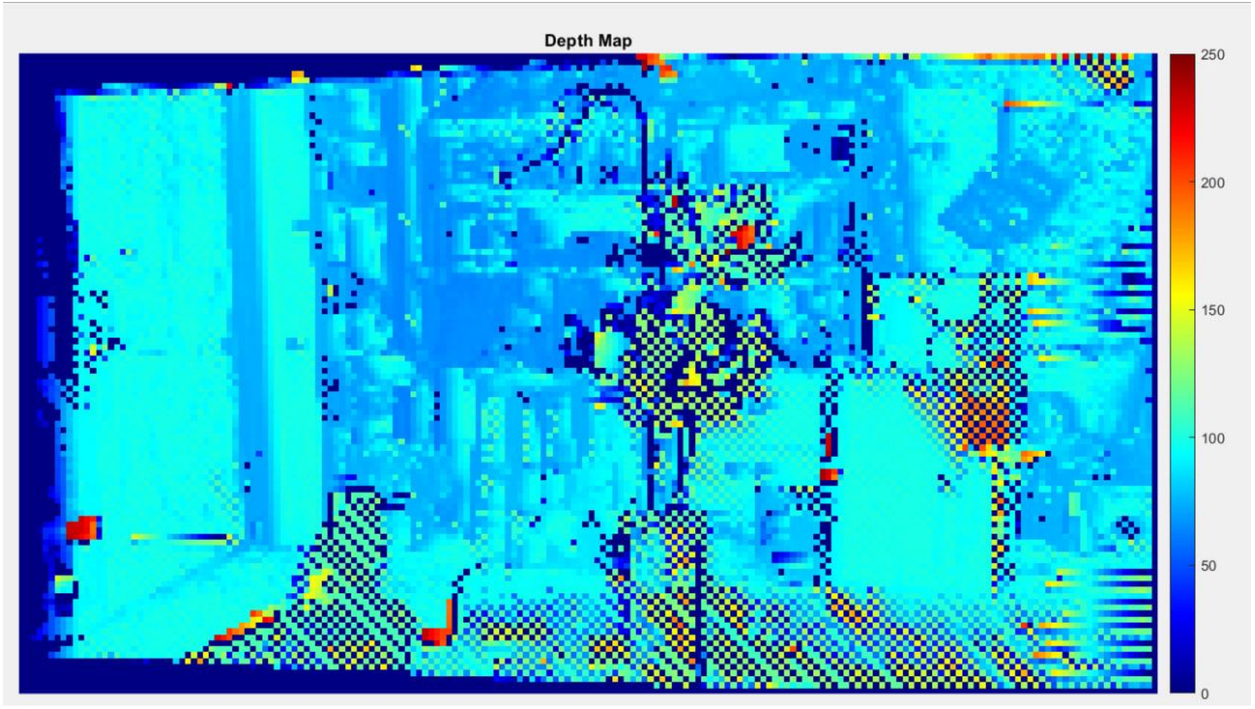
But because the up/down move makes the time complexity very high, I skipped experiment this anymore.

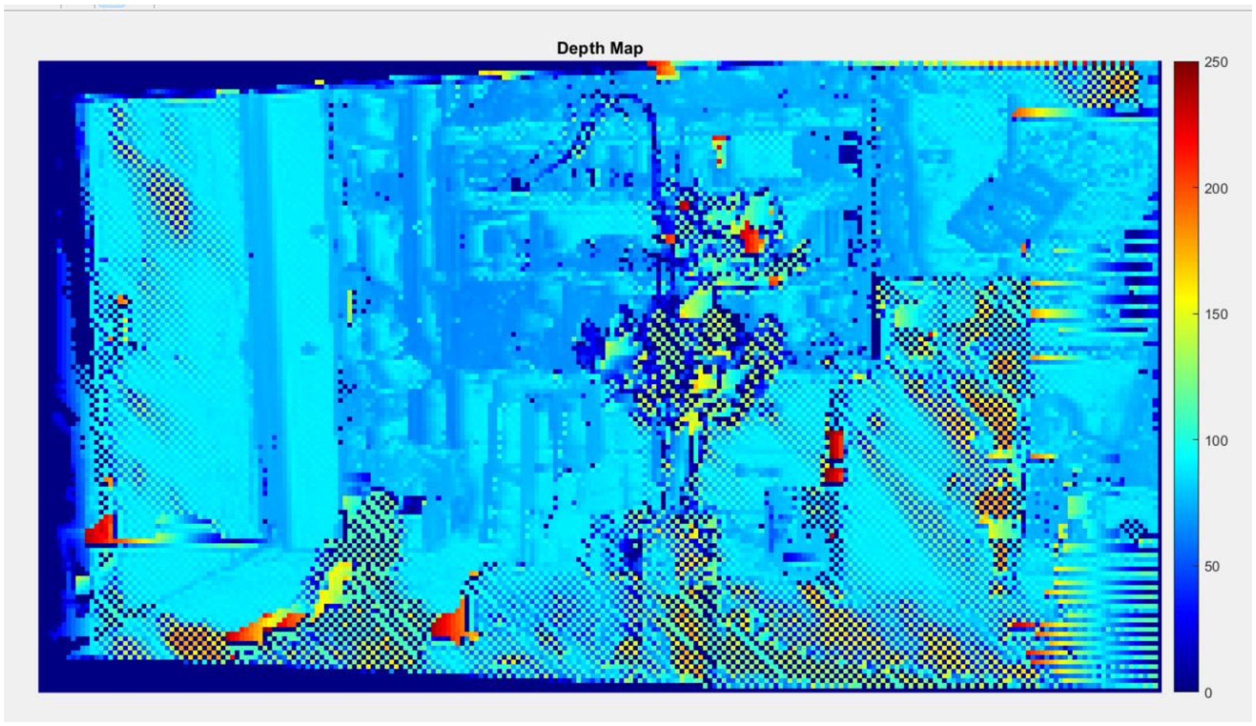
Here, I add the result of different settings, while each has different pros/cons. The searchrange, lambda, block size, cost, etc is different in these images.

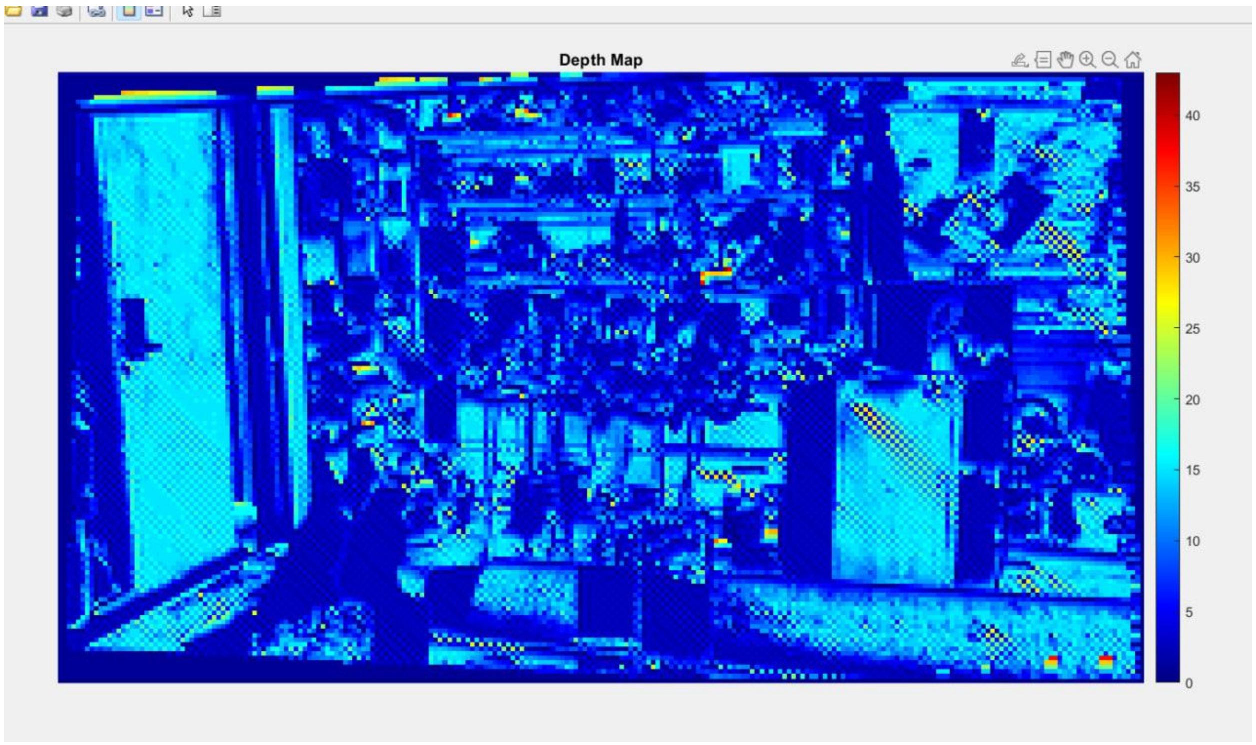

Depth Map
200
180
160
140
120
100
80
60
40
20
0


Depth Map
200
180
160
140
120
100
80
60
40
20
0

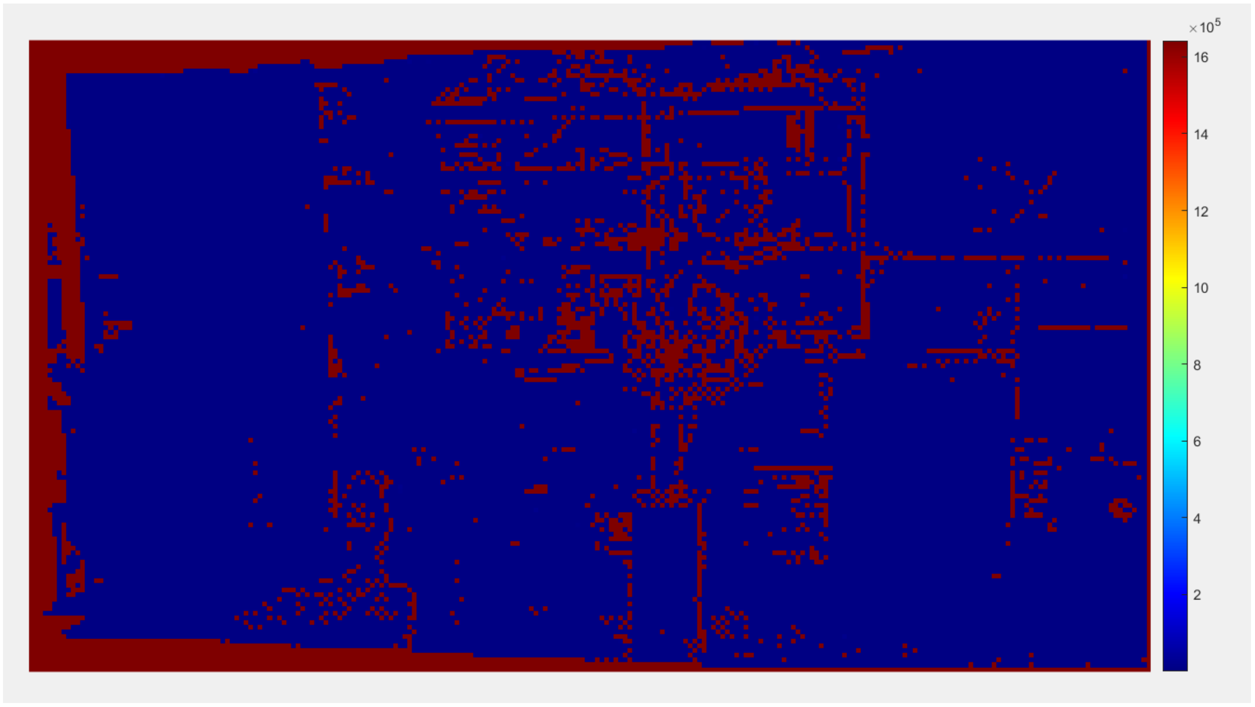

Depth Map
200
180
160
140
120
100
80
60
40
20
0


Depth Map
250
200
150
100
50
0

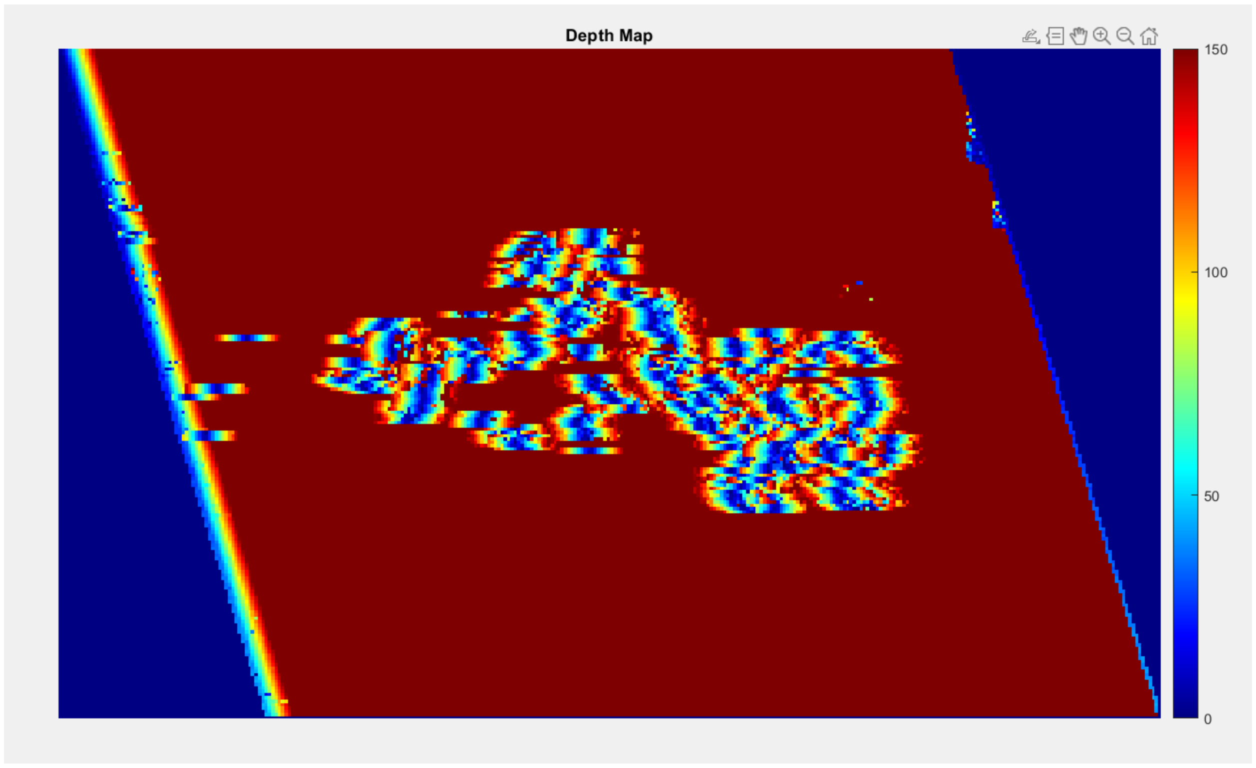

Depth Map
250
200
150
100
50
0

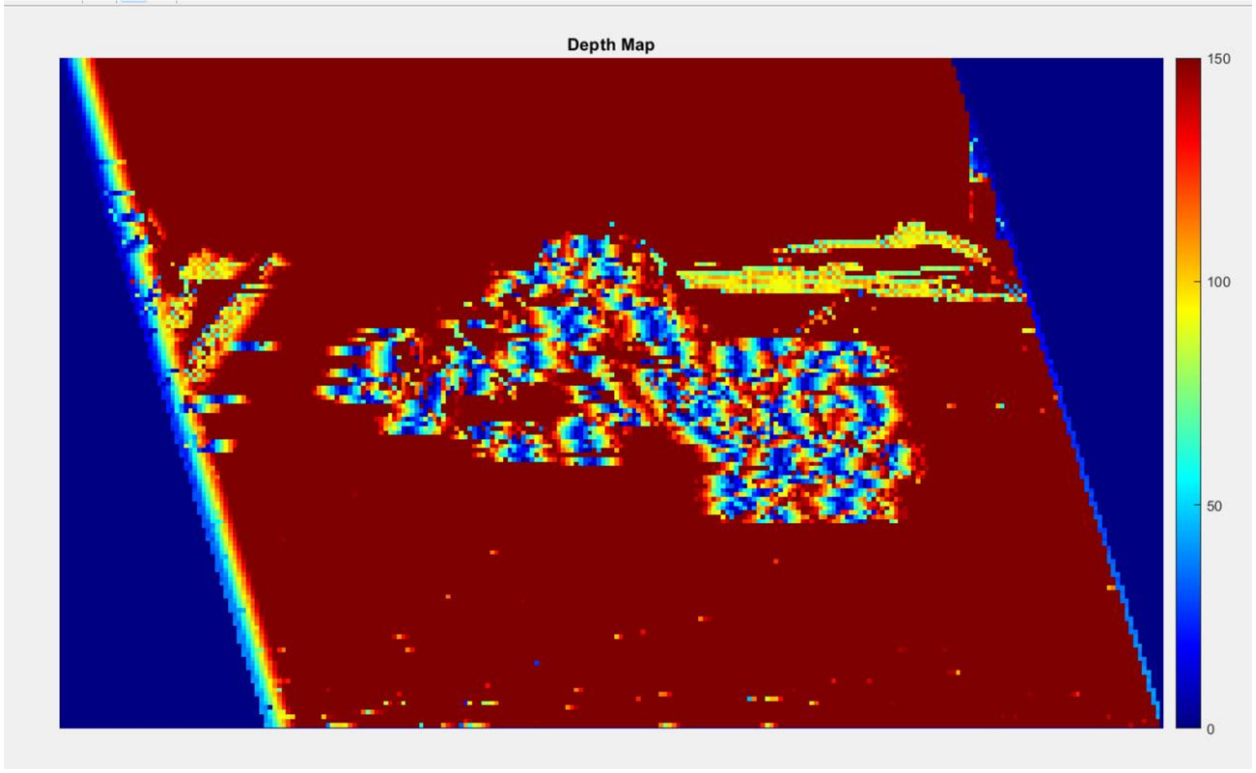

Depth Map
40
35
30
25
20
15
10
5
0

In order to get depth estimation, we pick one of the disparity maps and compute: $\frac{m_x * f * T_x}{d}$. In order to have a proper visualization, I divided the result by 100. However, the result is still not proper for visualization.



For the elephant-horse pair of images, I tried a few hyperparameters, but I didn't get good results. I just attach one. There are some reasons as I realized. The correspoinding pixels might not be in the same rows in the paired result image of Q1 (minor errors). Because the images are high resolution and the pixels are in the different rows, the time complexity is too high, so I ignored further experiments.


Depth Map
150
100
50
0


Depth Map
150
100
50
0

Discussion

Computational Efficiency: The algorithm may face challenges in terms of computational efficiency, especially with larger image sizes. The nested loops and \ search strategy over the entire search range could lead to slower processing times.

Limited Monotonicity Check: The monotonicity check in the code is applied only to the differences of rows, which may not capture more complex patterns or variations. Expanding and refining the monotonicity check could potentially improve the algorithm's robustness.

Parameter Sensitivity: The effectiveness of the algorithm may be sensitive to the chosen parameters, and finding optimal values for parameters like lambda and others might require experimentation and tuning.

-The algorithm employs a threshold for smoothness, which may affect the balance between preserving details and smoothing artifacts. Tuning this threshold could be critical to achieving an optimal trade-off between preserving fine details and suppressing noise.

-We could explore adaptive smoothness strategies that dynamically adjust the smoothness strength based on local image characteristics.

End