INTRODUCTION

Correlation matching is used to solve the correspondence problem in stereo vision in this lab assignment. In this problem, we have two images for same scene from left and right perspectives. We will search for best right sub image which is most similar to fixed left sub image in some predetermined search area.

Results of my code with different size of k values and search areas and related discussion can be find in the following parts.

FOLLOWED PROCEDURE / ALGORITHM STEPS

- Convert input images to grey scale image
- Make necessary padding as we have an additional search area for sub images
- Take a window (2k+1 x 2k+1) in left image, then starting from same center pixel point search for minimum similarity valued window in the right image

NOTE: We can observe that right image is the right shifted along x-axis version of the left image. Therefore, we should look for the left side of the starting point in an area with shape (w1 x w2). We also know that w2 should be very small as x-axis is the main shifted part.

- After fixing left window in the left image, start taking windows from right image in the search area and calculate similarity value with the following formula:

$$SSD = \sum_{k=-W}^{k=W} \sum_{l=-W}^{l=W} [f(i+k, j+l) - g(i+k, j+l)]^2$$

Figure 1. SSD Formula

- For each window in the right image, record the distances and the SSD value.

- After recording all windows in the search area, find the minimum SSD value with the following code:

```
ind = find(dist(:,3) == min(dist(:,3)));
```

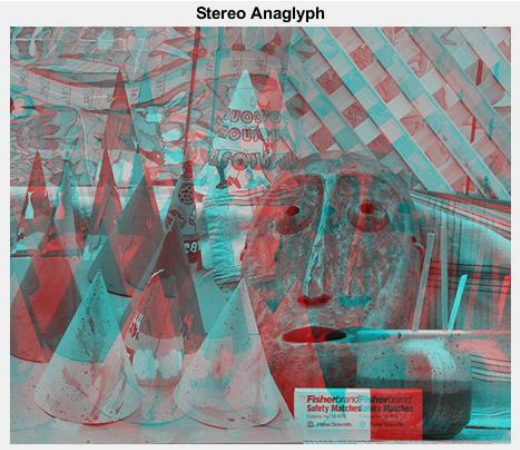
- Then, put this indices' distance value in a disparity matrix

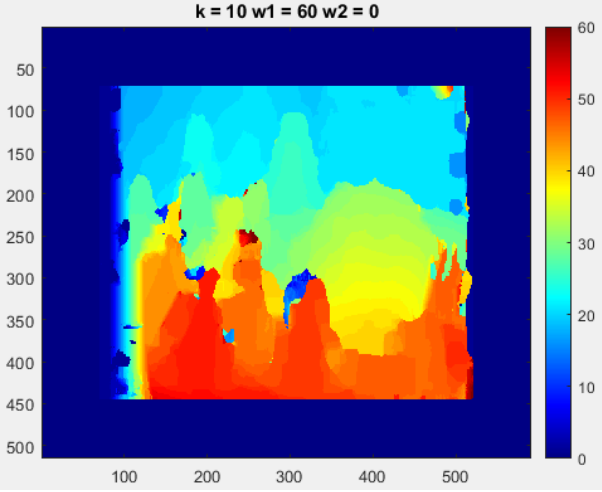
NOTE: Since the important distance parameter is the one along columns (x-axis), I used only w1 distance in my disparity matrix.

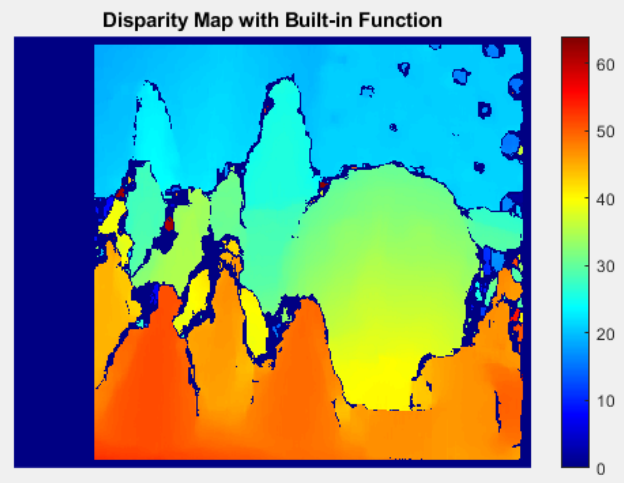
- After going through from each window in the left image, show the disparity matrix.

IN-LAB & POST-LAB RESULTS

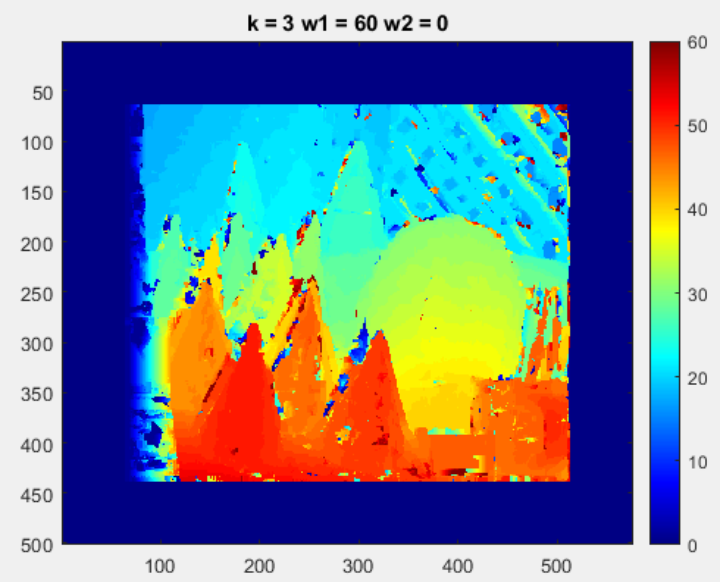
I started with png images, and the resulted images with different k and search areas can be seen below:



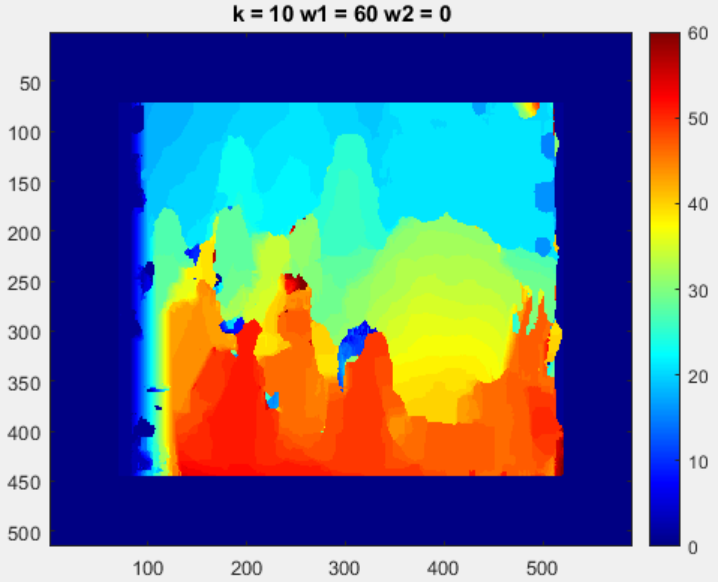




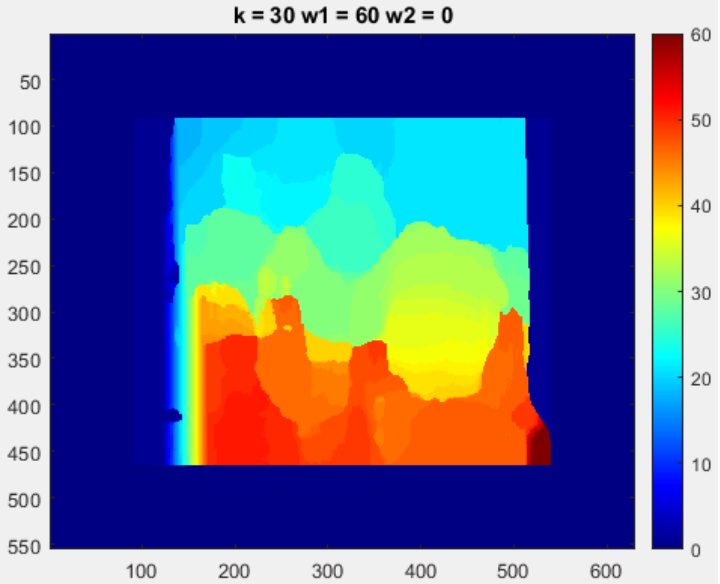
Change in window size (k):



k = 3

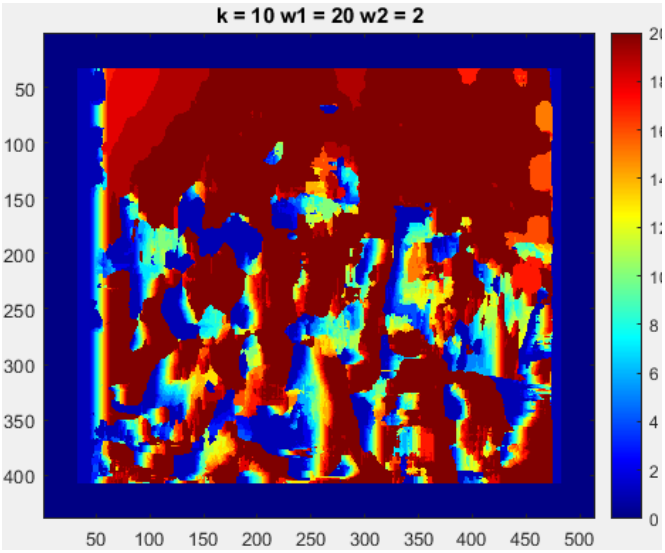


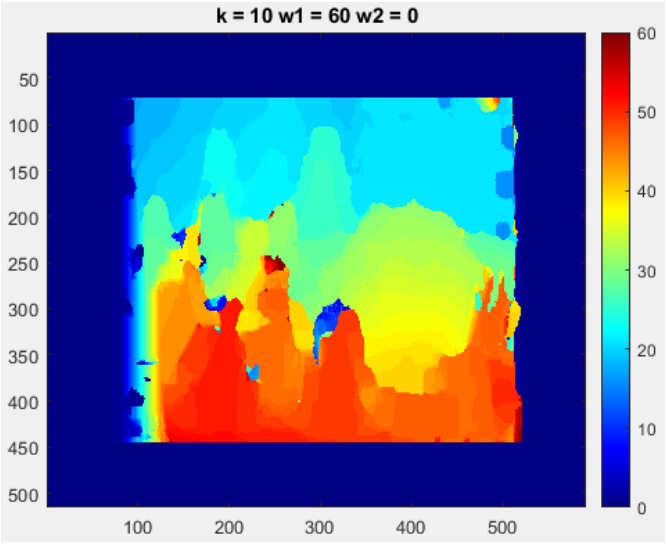
k = 10

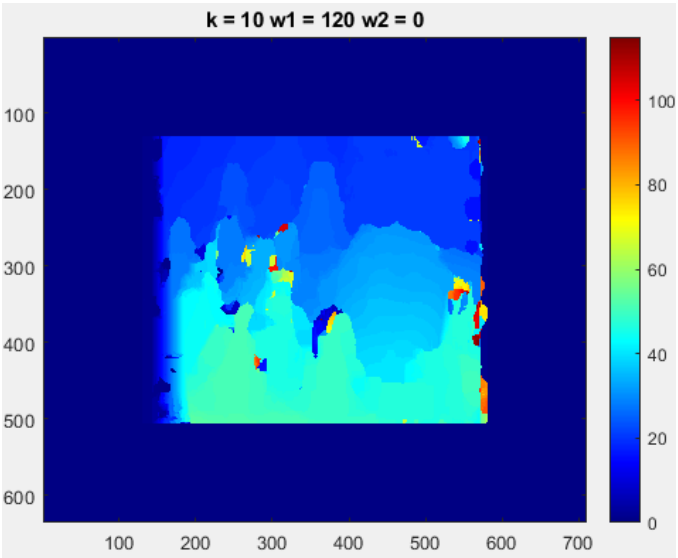


k = 30

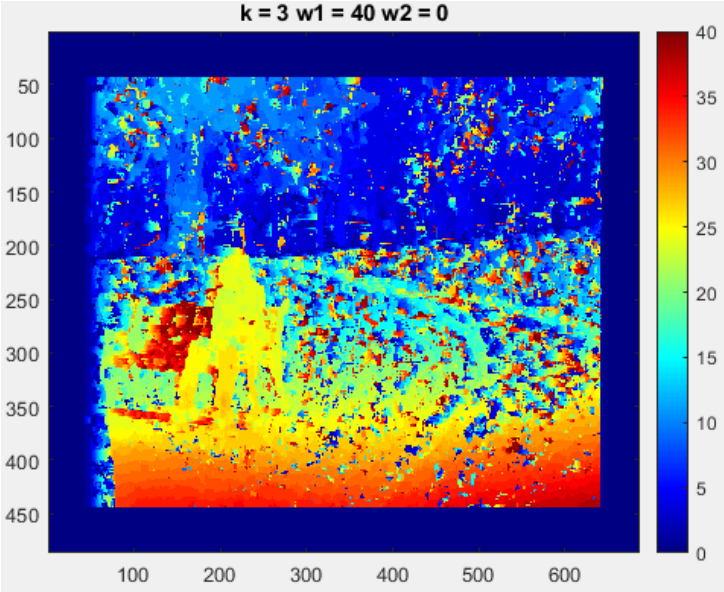
Change in Search Area (w1 & w2) with same k = 10:

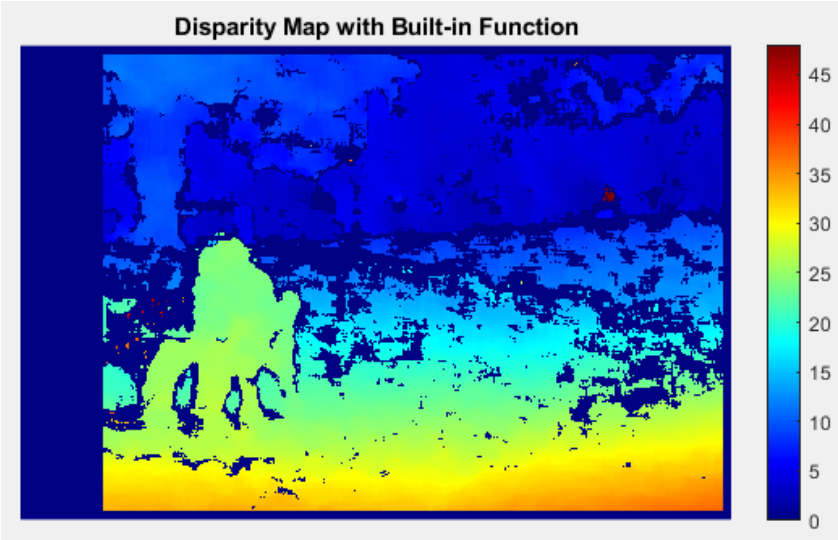




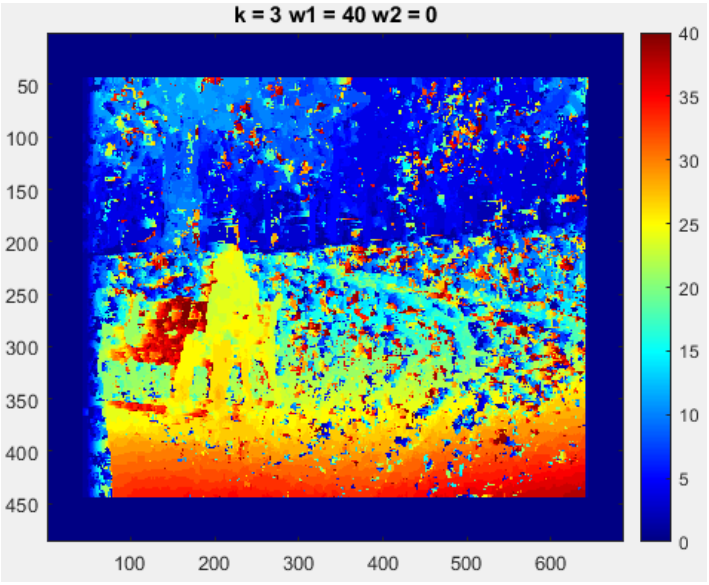


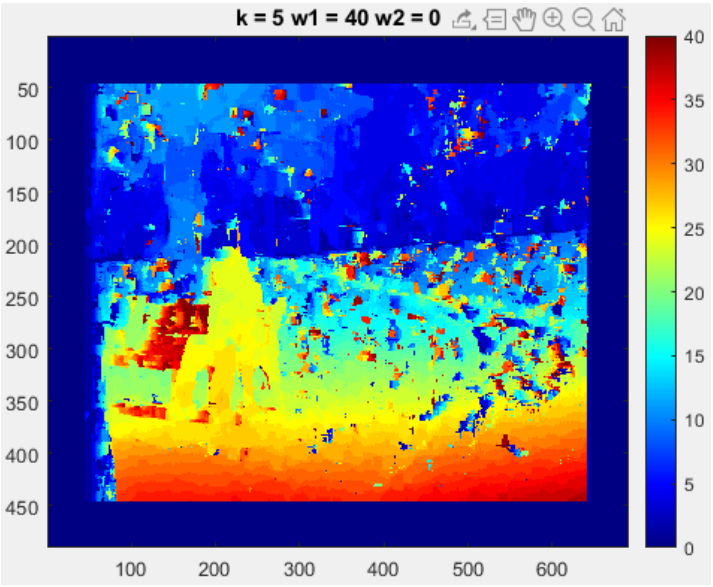
Now, try .tif images:

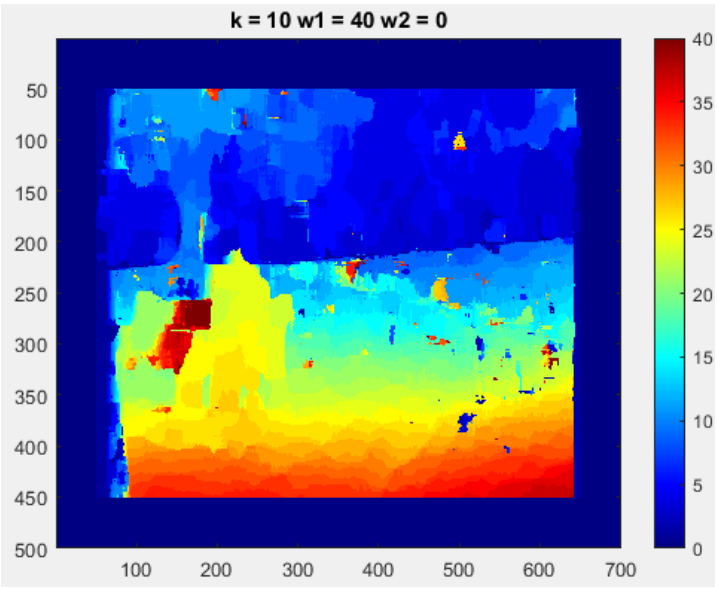




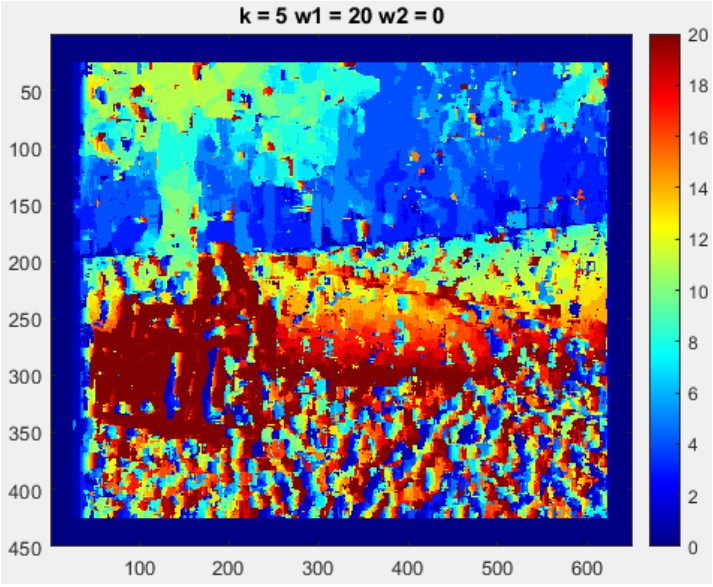
Change in window size (k):

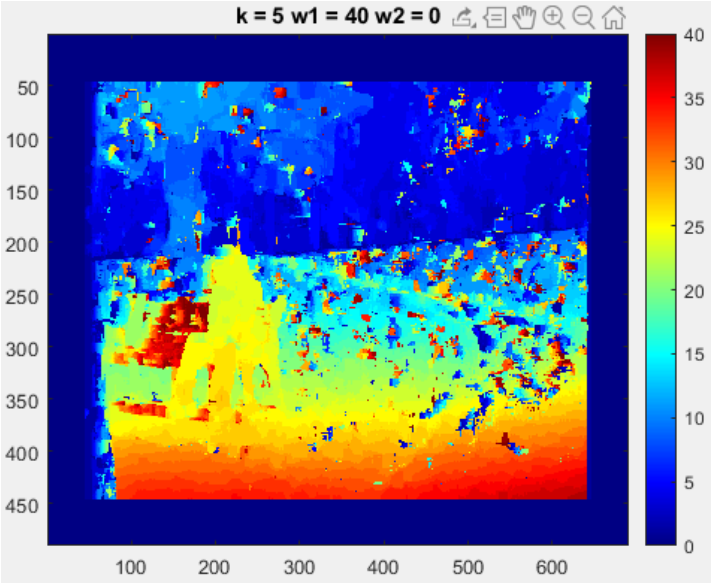


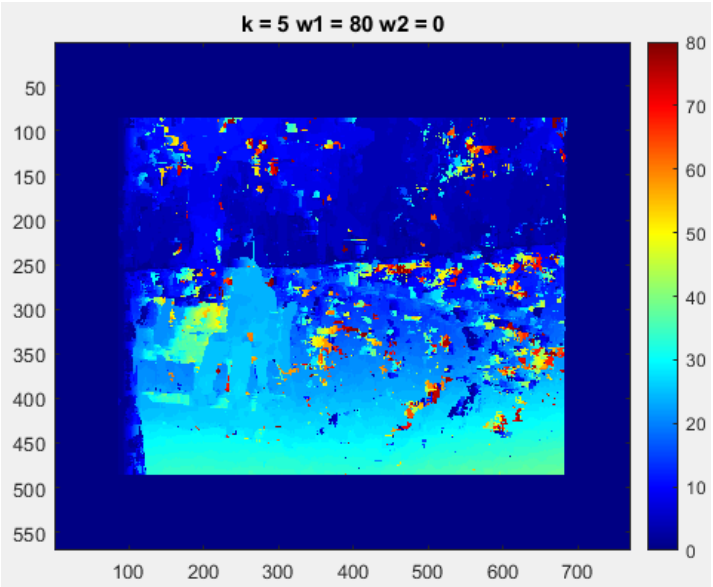




Change in Search Area (w1 & w2) with same k = 5:







DISCUSSION

We can observe that it is possible to solve correspondence problem by using correlation matching. Our results are getting closer to built-in disparity function by using optimal k and search areas.

We can observe that when we increase the k value (window size), the output getting smoother. However, if we increase too much we are starting to loosing correct output result. Therefore, we can say that there is an optimal valued window size for each input images and we need to find it. For instance, k = 10 is a good choice for the image with face. On the other hand, k = 5 is a good choice for the image with person sitting on the bank.

We can also observe that when we change the search area (the amount of left shift in the right image), the output is affected from this change. When we choose too low or too high search sizes, output is more prune to error and also computation time is increasing as we are looking more windows in the right image to find best similar window. Therefore, we can say that there is an optimal valued search area size for each input images and we need to find it. For instance, w1 = 60 is a good choice for the image with face. On the other hand, w1 = 40 is a good choice for the image with person sitting on the bank.

CODES

```
left = imread("S01L.png");
right = imread("S01R.png");
left = rgb2gray(left);
right = rgb2gray(right);

% % To see with following images uncomment below lines and comment upper 4
% left = imread("S00L.tif");
% right = imread("S00R.tif");
% left = rgb2gray(left);
% right = rgb2gray(right);

k = 10;
w1 = 60;
w2 = 0;
[row, col, ch] = size(left);
paddedleft = padarray(left, [k+w1+w2 k+w1+w2], "both");
paddedright = padarray(right, [k+w1+w2 k+w2+w1], "both");

data_left = double(paddedleft);
data_right = double(paddedright);

[padded_row,padded_col,ch] = size(paddedleft);
disparl = zeros(padded_row,padded_col);

for i = k+w1+w2+1:row+k+w1+w2
    for j = k+w1+w2+1:col+k+w1+w2
        left_window = data_left((i-k):(i+k), (j-k):(j+k));
        dist = [];
        for m = 1:w1
            for m2 = 0:w2
            right_window = data_right((i-k-m2):(i+k-m2), (j-k-m):(j+k-m));
            diff = left_window - right_window;
            square = diff.*diff;
            result = sum(sum(square));
            dist = [dist; [m m2 result]];
            end
        end
        ind = find(dist(:,3) == min(dist(:,3)));
        disparl(i, j) = dist(ind(1), 1);
    end
end
```

```
42      % disparl = uint8(disparl);
43 -    figure();
44 -    imshow(stereoAnaglyph(left,right));
45 -    title("Stereo Anaglyph");
46 -    figure; imagesc(disparl); colormap jet; colorbar
47 -    title("k = "+k + " wl = " + wl + " w2 = " + w2);
48
49
50      % % Disparity Map with built-in function for S01 images
51 -    disparityRange = [0 64];
52 -    disparityMap = disparityBM(left, right,'DisparityRange',disparityRange);
53 -    figure();
54 -    imshow(disparityMap,disparityRange);
55 -    title('Disparity Map with Built-in Function');
56 -    colormap(gca,jet)
57 -    colorbar
58
59
60      % % Disparity Map with built-in function for S00 images
61      % disparityRange = [0 48];
62      % disparityMap = disparityBM(left, right,'DisparityRange',disparityRange);
63      % figure();
64      % imshow(disparityMap,disparityRange);
65      % title('Disparity Map with Built-in Function');
66      % colormap(gca,jet)
67      % colorbar
```