Amirkabir University of Technology
(Tehran Polytechnic)
Department of Computer Engineering

Computer Vision

Assignment VI

Soroush Mahdi
99131050

supervised by
Dr. Reza Safabakhsh

February 10, 2022

Contents

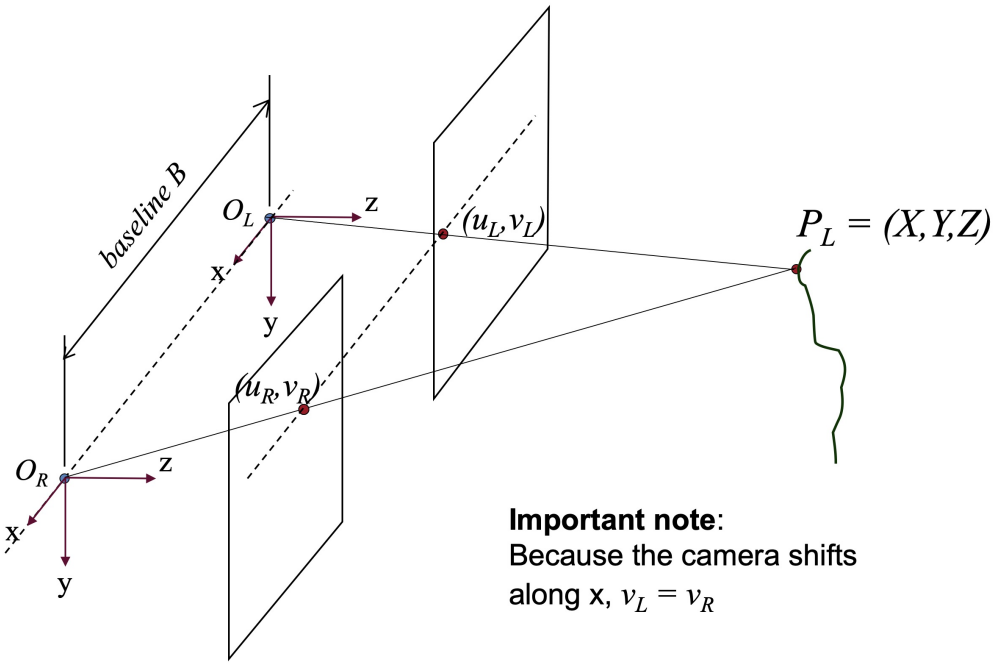
1 Left and Right images

for this homework, I have chosen these 2 images as left and right images.I selected these images from this link

Figure 1: left and right images

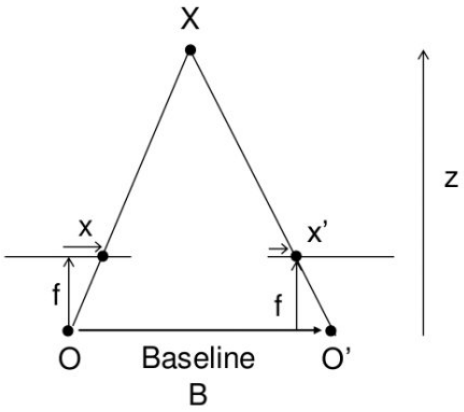
2 Disparity equation

first consider this situation:

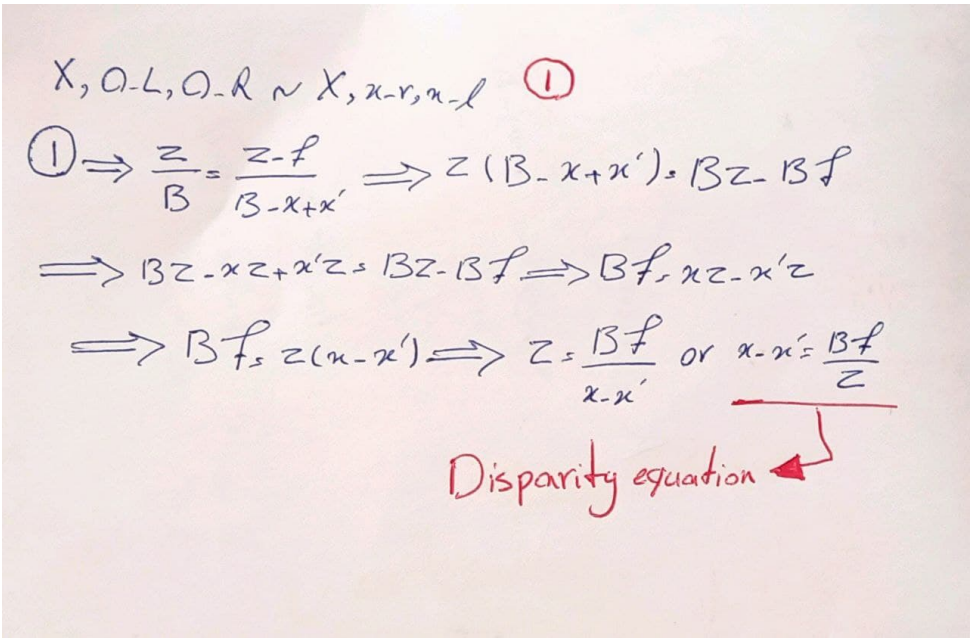


Two cameras with optical centers O-L and O-R are separated by a baseline B. The z-axis extends towards the world, away from the camera.

Now, consider just the plane spanned by the x- and z-axes, with a constant y value:



now if we name the intersection points between X O and X O-prime lines with cameras image planes, x-l and x-r respectively then we have two similar triangles: (X,O,O-prime) and (X,x-r,x-l) :



2.1 how to create disparity map?

in the above equation we have f and B so in short, the above equation says that the depth of a point in a scene is inversely proportional to the difference in distance of corresponding image points and their camera centers.in other words Far away objects (large distance from the observer) will move very little between the left and right image. Very closeby objects (small distance from the observer) will move quite a bit more. The focal length f and the baseline b between the cameras are just constant scaling factors. So with this information, we can derive the depth of all pixels in an image.

so the process for creating the disparity map involves choosing a point in the left-hand frame and then finding its match (often called the corresponding point) in the right frame. If we perform this matching process for every pixel in the left-hand image, finding its match in the right-hand frame and computing Z with respect to them, we would end up with an image where every pixel contains the distance/disparity value for that pixel in the left image.

now in the matching process, for each pixel in the left image, we should find the corresponding match in the left image. then we will find the horizontal distance between these two pixels and we will create a new photo based on these distances. in this photo, the brightness of each pixel is equal to the corresponding distance multiplied by a constant factor for better visualization.

for finding matches we will do this algorithm for each pixel x in the left photo, first, we consider a patch around x with x as its center, then we will Place the patch in the same coordinates in the right image. now we need to search for the match on the left side starting from this position and on the same line. The search area is restricted by a parameter. We will pick the candidate patch with the minimum similarity error and we can use norm 2 distance between 2 patches for the similarity error metric.

3 Creating disparity map

here is the code for creating disparity map based on the explained algorithm:

```
def disparity(imgL, imgR, patch_size , max_sv, contrast =10):
    """generate disparity map based on left and right images

    Args:
        imgL ([numpy 2D array]): [left image (gray)]
        imgR ([numpy 2D array]): [right image (gray)]
        patch_size ([int]): [defines size of patch]
        max_sv ([int]): [maximum size for searching in right image on the same line]
        contrast (int, optional): [constant value for better visualization]. Defaults
     to 10.

    Returns:
        [numpy 2D array]: [disparity map with the same size as the input photos]
    """
    H,W = imgL.shape
    dis_map = np.zeros_like(imgL)
    #doing process for each pixel
    for row in range(patch_size//2, H-patch_size//2):
        for col in range(patch_size//2, W-patch_size//2):
            #selecting patch in left image
            l_patch = imgL[row - patch_size//2:row + patch_size//2 +1 ,col -
    patch_size//2:col + patch_size//2 +1]
            min_dist = 1000
            match = 0
            for i in range(max_sv):
                if(col - patch_size//2 - i > -1):
                    #selecting patch in right image
                    r_patch = imgR[row - patch_size//2:row + patch_size//2 +1 ,col -
    patch_size//2 - i :col + patch_size//2 +1-i]
                    new_dist = dist(l_patch,r_patch)
                    if new_dist < min_dist:
                        min_dist = new_dist
                        match = i
            dis_map[row,col] = match *contrast if match*contrast < 255 else 255

    return dis_map
```

Listing 1: disparity function

before passing images to this function i blurred them using guassianblur function.the results are shown on the next page i used different ways of blurring for smoothness.

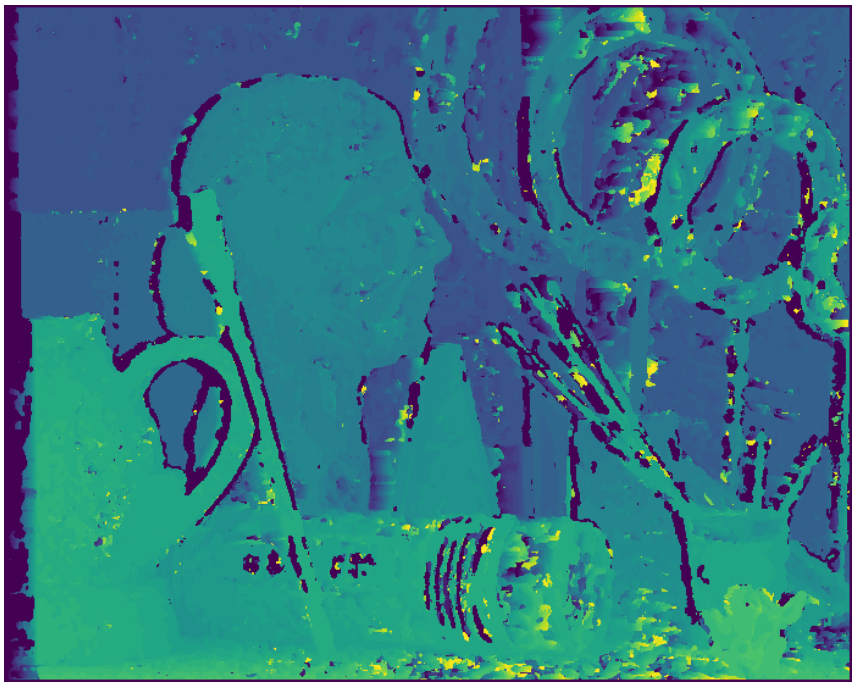


Figure 2: result of disparity function

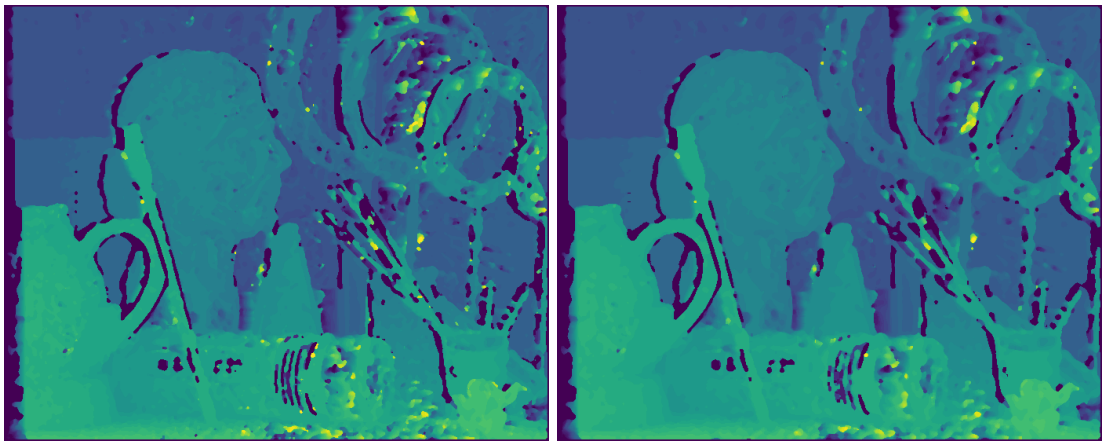


Figure 3: median blur with kernel=5(left) and median blur with kernel=7(right)

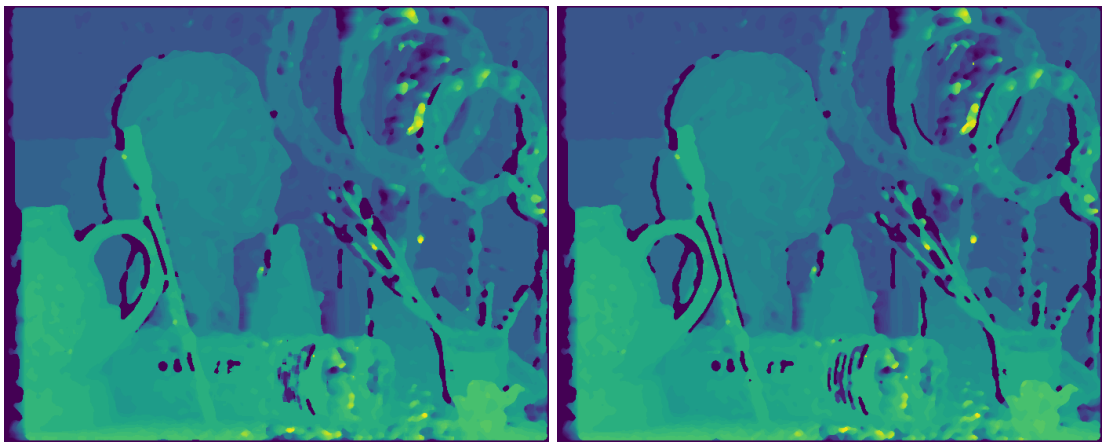


Figure 4: median blur with kernel=5(left) and iterative median blur with kernel=5(right)

4 StereoBM

we will discuss the effect of different parameters in stereoBM class in this section.

4.1 numDisparities

this parameter defines How many pixels to slide the window over. The larger it is, the larger the range of visible depths, but more computation is required.in fact this parameter defines The disparity search range. so with increasing this parameter we will have better disparity ma but we also will have more computation required.

The value for this parameter must be greater than zero and In the current implementation, this parameter must be divisible by 16.

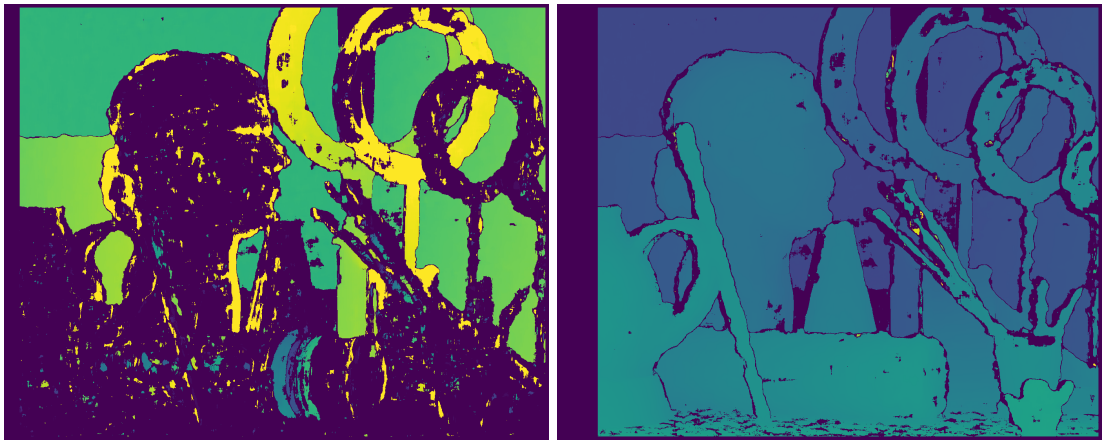


Figure 5: numDisparities = 32 (left) and numDisparities = 96 (right)

4.2 TextureThreshold

this parameter filters out areas that don't have enough texture for reliable matching and Calculate the disparity only at locations, where the texture is larger than this threshold.

Figure 6: TextureThreshold = 5 (left), TextureThreshold = 2500 (middle) and TextureThreshold = 5000 (right)

we can see that with higher values for this parameter there will be more regions in the photo that algorithm does not do calculation on them.

4.3 UniquenessRatio

If the best matching disparity is not sufficiently better than every other disparity in the search range, the pixel is filtered out. we can specify this margin using UniquenessRatio.

More specifically UniquenessRatio shows Margin in percentage by which the best (minimum) computed cost function value should "win" the second best value to consider the found match correct.

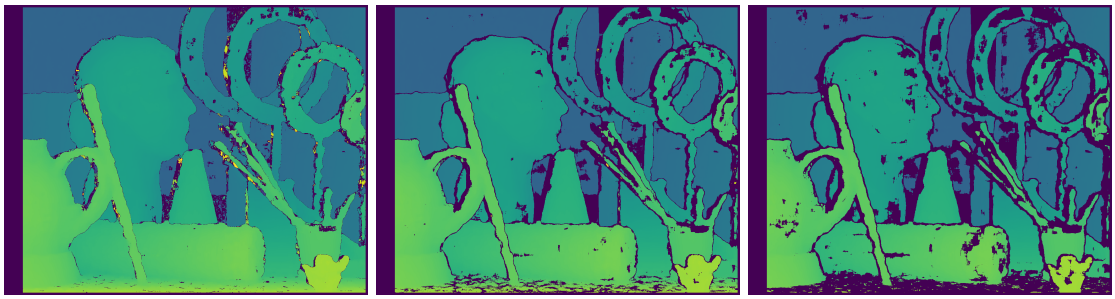


Figure 7: UniquenessRatio = 5 (left), UniquenessRatio = 20 (middle) and UniquenessRatio = 40 (right)

we can see with higher values for this parameter there will be more pixels which algorthm will filter out.

4.4 SpeckleWindowSize

Block-based matchers often produce "speckles" near the boundaries of objects, where the matching window catches the foreground on one side and the background on the other. In this scene it appears that the matcher is also finding small spurious matches in the projected texture on the table. To get rid of these artifacts we post-process the disparity image with a speckle filter controlled by the speckle-size and speckle-range parameters. speckle-size is the number of pixels below which a disparity blob is dismissed as "speckle."

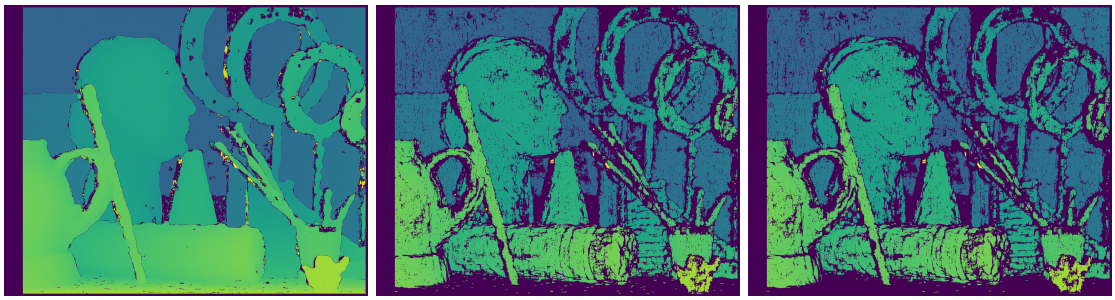


Figure 8: SpeckleWindowSize = 0 (left), SpeckleWindowSize = 10 (middle) and SpeckleWindowSize = 15 (right)

4.5 PreFilterSize

this variable defines Window size of the prefilter and will be used in The pre-filtering phase, which normalizes image brightness and enhances texture in preparation for block matching.

i tried different values for this variable, but didn't see any changes in the result.

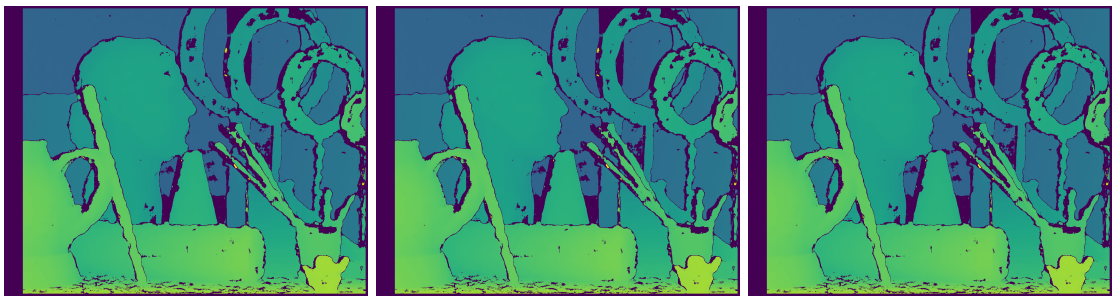


Figure 9: PreFilterSize = 5 (left), PreFilterSize = 121 (middle) and PreFilterSize = 255 (right)