CS4186

ASSIGNMENT 2

Disparity Map Calculation

Created By ~

NAME: Rishabh NAUTIYAL

SID: 5533169

Introduction

The objective of the assignment is to generate Disparity Maps from a set of stereoscopic images of the same scene. The correctness of the generated disparity map is calculated using PSNR value, which is a calculation between the created disparity map and the ground truth. I will be using the library called OpenCV for the task and will write the code in Python.

Note: Even though the images were rectified to begin with, while experimenting with the code I found out that the PSNR value of the results is much better (almost double from 5-10) if I rectify the images once more. My assumption is that the original rectification might have left behind some small errors and movements in vertical direction which are smoothed out after rectifying again, thus giving a much better result.

Task Process

The whole code is written in a modular style, with each function carrying out a particular task. The whole code is run through the main function. The important Functions are:

Main Function:

```
if __name__=="__main__":

    folders=["Art", "Dolls","Reindeer"]

    for p in folders:

        path1="StereoMatchingTestings/"+p+"/view1.png"
        path2="StereoMatchingTestings/"+p+"/view5.png"

        im1, im2 = readImages(path1, path2)

        kp1, kp2, d1, d2=sift(im1, im2, "Results/"+p)

        pt1, pt2 = matchKeypoints(im1, im2, kp1, kp2, d1, d2, "Results/"+p)

        pt1, pt2, fm = calcFundamentalMatrix(pt1, pt2)

        im1_rect, im2_rect = rectify_images(pt1, pt2, im1, im2, fm, "Results/"+p)

        calcDisparityMap(im1_rect, im2_rect, "Results/pred/"+p)
```

The main function is the controller function which reads the images and then calls all the other functions to orchestrate the disparity map generation. The directories are present in the folders array and the code loops over these directories to produce the disparity Map for each set of images in these libraries.

Sift

```
def sift(im1, im2, path):
    sift = cv.SIFT_create()
    kp1, d1 = sift.detectAndCompute(im1, None)
    kp2, d2 = sift.detectAndCompute(im2, None)
    image_keypoints = cv.drawKeypoints(im1, kp1, None, flags=cv.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)
    path = path+"/_keypoints.png"
    cv.imwrite(path, image_keypoints)

    return kp1, kp2, d1, d2
```

This function is an interface function for the OpenCV SIFT code. The function accepts two images and an outputPath and calculates the key points and descriptors (using SIFT) for the two images.

Since the images are almost identical, to see the result I am only storing one image(view1) with key points marked.

Keypoint Matching

This function is the main function, matching the corresponding key points in the two views.

It accepts two images along with the descriptors and keypoints calculated from the sift function and uses BF Matcher (Brute Force Matcher) with KNN (which gives K best matches) to match the corresponding points on the two images.

```
def matchKeypoints(im1, im2, kp1, kp2, d1, d2, p):

    # BFMatcher with default params
    bf = cv.BFMatcher()
    matches = bf.knnMatch(d1,d2, k=2)
    # Apply ratio test
    good = []
    pt1 = []
    pt2 = []
    for m,n in matches:
        if m.distance < 0.9*n.distance:
            good.append([m])
            pt1.append(kp1[m.queryIdx].pt)
            pt2.append(kp2[m.trainIdx].pt)

    # cv2.drawMatchesKnn expects list of lists as matches.
    result_img = cv.drawMatchesKnn(im1,kp1,im2,kp2,good, None, flags=2)
    path = p+"/_matched_keypoints.png"
    cv.imwrite(path, result_img)

    return pt1, pt2
```

After Calculating the Matches, I have used a ratio test to just keep the matches that meet my set ratio. Here the ratio is very high and even if the best match distance is < 0.9*the distance of the second best match then we consider it a good match. The ratio was set by, manual hyperparameter tuning.

Fundamental Matrix Calculation

```
def calcFundamentalMatrix(pt1, pt2):
    pt1 = np.int32(pt1)
    pt2 = np.int32(pt2)
    fm, inliers = cv.findFundamentalMat(pt1, pt2, cv.FM_RANSAC)
    #selecting only the inliers.
    pt1 = pt1[inliers.ravel() == 1]
    pt2 = pt2[inliers.ravel() == 1]

    return pt1, pt2, fm
```

As the name suggests, this function calculates the fundamental matrix from the corresponding vectors of the key points on the two images. I am using RANSAC to calculate the fundamental matrix (i.e, only keeping the inliers and not the outliers for the boost in performance)

Image Rectification

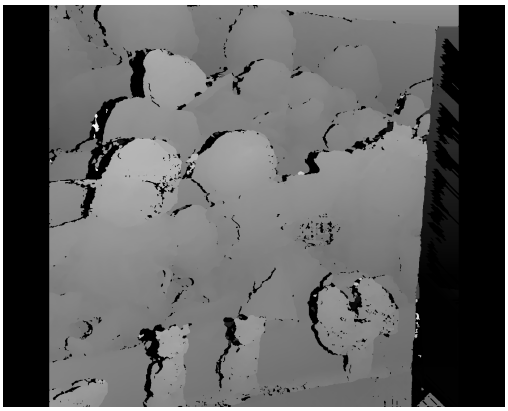
```
: def rectify(pt1, pt2, im1, im2, fm, path):
  #corresponding epiline are drawn

      h1, w1 = im1.shape
      h2, w2 = im2.shape
      _, HF1, HF2 = cv.stereoRectifyUncalibrated(np.float32(pt1), np.float32(pt2),fm, imgSize=(w1, h1))
      im1_rect = cv.warpPerspective(im1, HF1, (w1, h1))
      im2_rect = cv.warpPerspective(im2, HF2, (w2, h2))

      #saving rectified images
      path1 = path+"/_rectifiedView1.png"
      path2 = path+"/_rectifiedView2.png"
      cv.imwrite(path1, im1_rect)
      cv.imwrite(path2, im2_rect)

      return im1_rect, im2_rect
```

This method is used to rectify two images using warpPerspective from openCV. As stated earlier, even though the images are already rectified, doing image rectification again resulted in a much better result.

Disparity Map Calculation

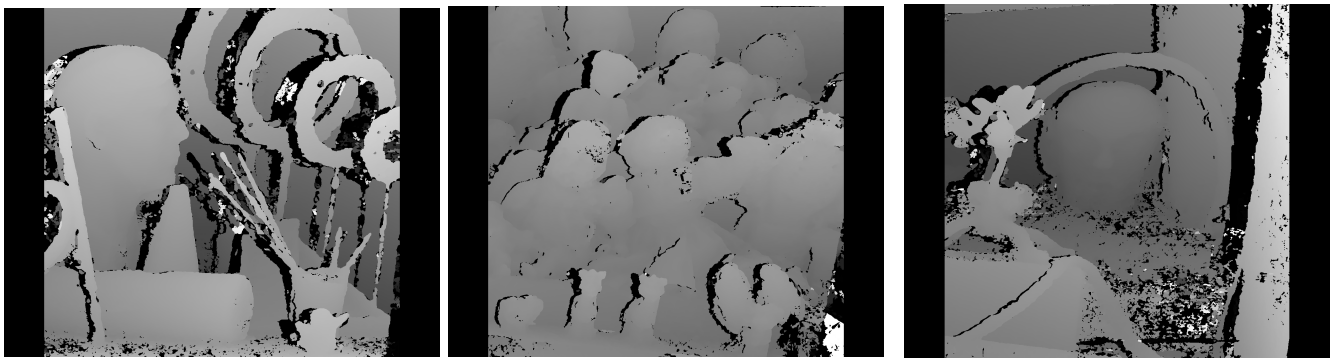
```
: def calcDisparityMap(im1, im2, path):
      stereo = cv.StereoSGBM_create(
          P1=420,
          P2=2869,
          blockSize=9,
          speckleRange=5,
          disp12MaxDiff=0,
          uniquenessRatio=3,
          minDisparity=-128,
          numDisparities=256,
          speckleWindowSize=25
      )

      computed_disparity= stereo.compute(im1, im2)
      computed_disparity_norm = cv.normalize(computed_disparity, computed_disparity, alpha=255, beta=0, norm_type=cv.NORM_MINMAX)
      computed_disparity_norm = np.uint8(computed_disparity_norm)
      path = path+"/disp1.png"
      plt.imshow(computed_disparity_norm)
      plt.colorbar()
      plt.show()
      cv.imwrite(path, computed_disparity_norm)
```

This function finally calculates the disparity map using StereoSGBM from openCV library. The parameters were set by manual tuning. The process resulted in the best PSNR value for the Dolls image and the result image can be seen on the side.



Results:

Final Images:



PSNR Results:

Without Image Rectification

```
The Peak-SNR value is %0.4f
 6.079518678451747
The Peak-SNR value is %0.4f
 5.738305033665103
The Peak-SNR value is %0.4f
 5.563439544197125
```

With Rectification

```
The Peak-SNR value is %0.4f
 10.297194092733461
The Peak-SNR value is %0.4f
 10.898422412873629
The Peak-SNR value is %0.4f
 9.232659680708553
```

Conclusion:

This was a very fun assignment where I got to learn a lot about 3D vision. I also got to learn how human eyes work and are able to capture depth information with 2 eyes. Moreover, the assignment gave me hands-on practice with OpenCV and learned how to use modern Computer Vision Libraries. Overall I really enjoyed the assignment a lot and would like to thank the professors and the TAs for providing us with such creative assignments.