Computer Vision **(Due: 02/05/2020)**

Image stitching and Disparity (Quiz_C2) [2 marks]

Instructor: Tamal Maharaj *Name:* Jimut Bahan Pal

#TASK: Coding Quiz 2

Part 1:

Take two images (with approx. 50% overlap)

Use the code to stitch the images:

https://towardsdatascience.com/image-stitching-using-opencv-817779c86a83
Write a short report containing the original images, the stitched image and a short description of the OpenCV methods used.

Part 2:

Take two images just by moving the camera sidewise (image-plane parallel to the scene). Use the code to produce the disparity map:

https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_calib3d/py_depthmap/py_depthmap.html
https://github.com/opencv/opencv/blob/master/samples/python/stereo_match.py
https://sourceforge.net/projects/meshlab/

Write a short report containing the original images and the disparity map and a short description of the OpenCV methods used and why. Recommended: Provide 3D constructed view of the scene.

#SOLN:

Part 1: **Panorama stitching**
Panorama stitching is done with the help of 2 images with some overlap. These type of stitching can be done to create a continuous image. We firstly compute the SIFT descriptor before blurring [1], (using $cv2.xfeatures2d.SIFT\_create()$ and $sift.detectAndCompute(img1, None)$) for each of the images to be stitched. We compute the distances between every descriptor in one image to every descriptor to another image. We then perform warp using OpenCV functions to align for stitching and at last we stitch them together. We use the $cv2.BFMatcher()$ for matching the features which are more similar. We then set parameter $k = 2$ for k-NN Matcher to give out 2 best matches for each of the descriptor. We needed to do this because there will be some trivial matches which could result in errors. We then compute the homography using $cv2.findHomography()$. We then select the top 'n' matches for each descriptor for an image. Then we run the RANSAC algorithm to estimate the homography.

Some of our experiment in a wide range of setting is shown in Figure 1, 2 and 3. We do this for images related to a 360° video captured by me during daytime from our rooftop as shown in Figure 1. To test this even further we do this for a night image as shown in Figure 2, which still gives state of the art result. We do this for an unknown setting as shown in Figure 3, which also gives state of the art result.

Part 2: **Disparity Map with 3D reconstruction**

When two images of a same environment is takes with just a minor linear shift, we can compute the disparity between two of the images as shown in Figure 4c. We can even use this feature to convert the disparity map to some sort of polygon rendering scheme and visualise the actual environment in 3D as shown in Figure 4d. This

is visually more interesting and amazing thing to work on since we are using only two images and creating a whole 3D model reconstructed from it.

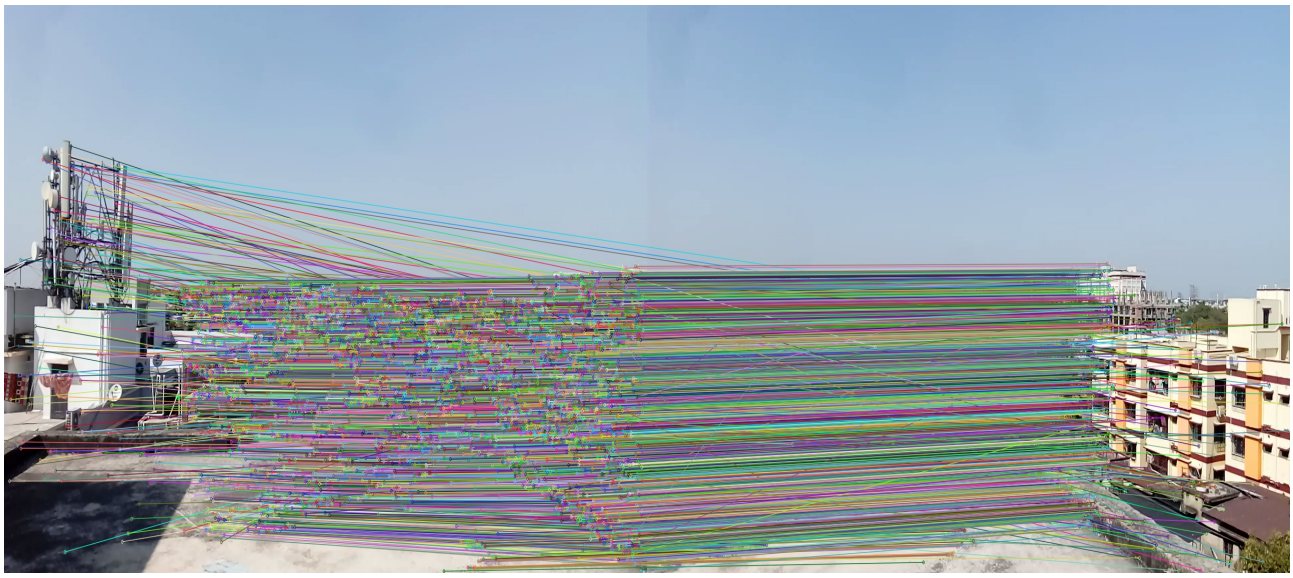
For computing the stereo we have used $cv2.StereoBM_create(numDisparities = 16, blockSize = 13)$ method with a block size of 13 and number of disparity 16. We have used the $stereo.compute(imgL, imgR)$ function by passing the left and right images, to compute the actual disparity. We have normalized and visualised the same as shown in Figure 4, where we have further computed the polygon and visualised the 3D model using meshlab.

References

[1] Jimut Bahan Pal. A deeper look into hybrid images, 2020. https://arxiv.org/abs/2001.11302 available on web, last accessed on May 2, 2020.

(a) First image captured from rooftop.

(b) Second with some overlap with the first image.



(c) SIFT feature match between two images.

(d) Panorama stitching for the two images.

Figure 1: Images showing the steps performed in image stitching in a day setting.

(a) First image captured from veranda in night setting.

(b) Second image with some overlap with the first image.

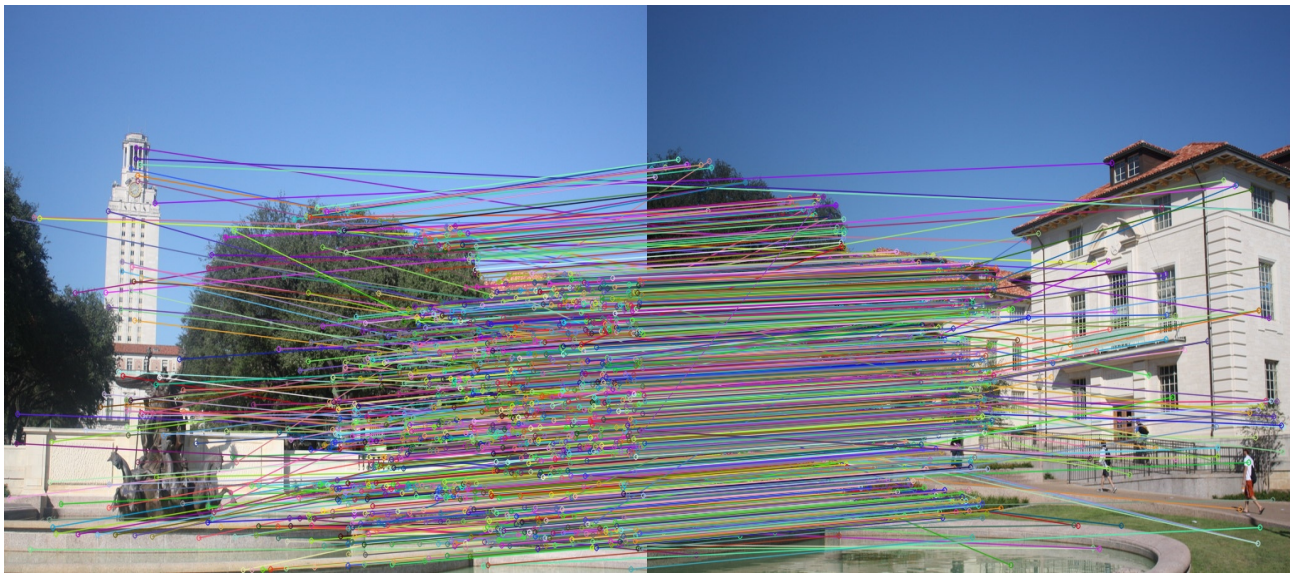
(c) SIFT feature match between two images.

(d) Panorama stitching for the two images.

Figure 2: Images showing the steps performed in image stitching in a night setting.

(a) First image in an unknown environment.

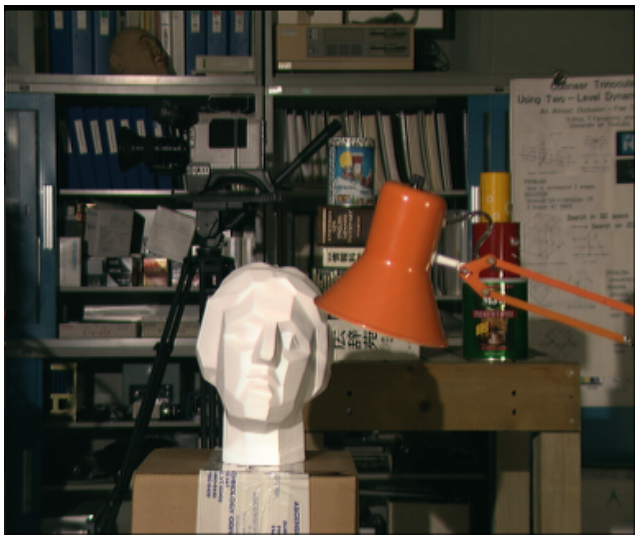
(b) Second image in an unknown environment.



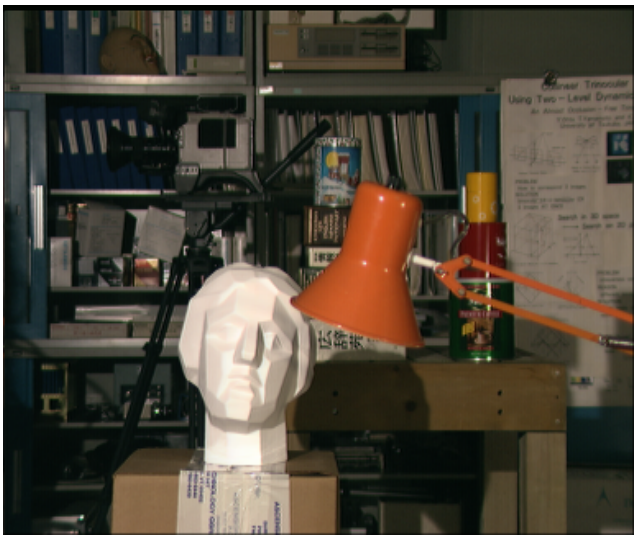
(c) SIFT feature match between two images.

(d) Panorama stitching for the two images.

Figure 3: Images showing the steps performed in image stitching in an unknown environment.

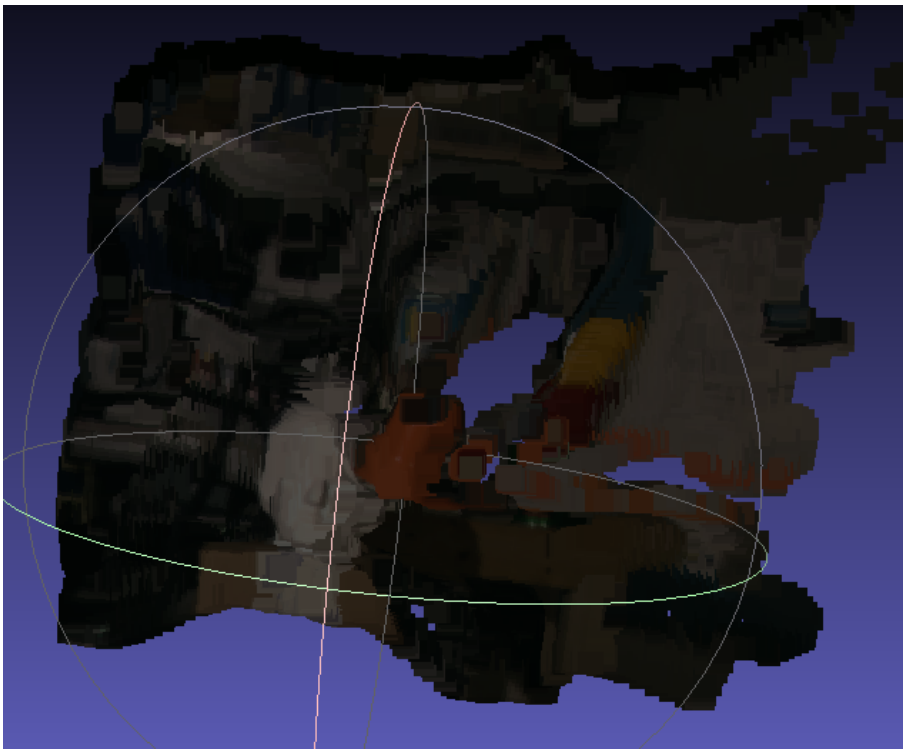


(a) Image captured from left.



(b) Image captured from right.

(c) Disparity map between two images.



(d) Disparity map in 3D using meshlab.

Figure 4: Images showing the creation of disparity map and 3D reconstruction of a personal environment.

image_stitch2

May 2, 2020

[5]:
```
import cv2
import numpy as np
import sys

class Image_Stitching():
    def __init__(self) :
        self.ratio = 0.85
        self.min_match = 10
        self.sift = cv2.xfeatures2d.SIFT_create()
        self.smoothing_window_size = 800

    def registration(self,img1,img2):
        kp1, des1 = self.sift.detectAndCompute(img1, None)
        kp2, des2 = self.sift.detectAndCompute(img2, None)
        matcher = cv2.BFMatcher()
        raw_matches = matcher.knnMatch(des1, des2, k=2)
        good_points = []
        good_matches = []
        for m1, m2 in raw_matches:
            if m1.distance < self.ratio * m2.distance:
                good_points.append((m1.trainIdx, m1.queryIdx))
                good_matches.append([m1])
        img3 = cv2.drawMatchesKnn(img1, kp1, img2, kp2, good_matches, None,␣
↪flags=2)
        cv2.imwrite('matching.jpg', img3)
        if len(good_points) > self.min_match:
            image1_kp = np.float32(
                [kp1[i].pt for (_, i) in good_points])
            image2_kp = np.float32(
                [kp2[i].pt for (i, _) in good_points])
            H, status = cv2.findHomography(image2_kp, image1_kp, cv2.RANSAC,5.0)
        return H

    def create_mask(self,img1,img2,version):
        height_img1 = img1.shape[0]
        width_img1 = img1.shape[1]
        width_img2 = img2.shape[1]
```

```
        height_panorama = height_img1
        width_panorama = width_img1 + width_img2
        offset = int(self.smoothing_window_size / 2)
        barrier = img1.shape[1] - int(self.smoothing_window_size / 2)
        mask = np.zeros((height_panorama, width_panorama))
        if version== 'left_image':
            mask[:, barrier - offset:barrier + offset ] = np.tile(np.
↪linspace(1, 0, 2 * offset ).T, (height_panorama, 1))
            mask[:, :barrier - offset] = 1
        else:
            mask[:, barrier - offset :barrier + offset ] = np.tile(np.
↪linspace(0, 1, 2 * offset ).T, (height_panorama, 1))
            mask[:, barrier + offset:] = 1
        return cv2.merge([mask, mask, mask])

    def blending(self,img1,img2):
        H = self.registration(img1,img2)
        height_img1 = img1.shape[0]
        width_img1 = img1.shape[1]
        width_img2 = img2.shape[1]
        height_panorama = height_img1
        width_panorama = width_img1 +width_img2

        panorama1 = np.zeros((height_panorama, width_panorama, 3))
        mask1 = self.create_mask(img1,img2,version='left_image')
        panorama1[0:img1.shape[0], 0:img1.shape[1], :] = img1
        panorama1 *= mask1
        mask2 = self.create_mask(img1,img2,version='right_image')
        panorama2 = cv2.warpPerspective(img2, H, (width_panorama,␣
↪height_panorama))*mask2
        result=panorama1+panorama2

        rows, cols = np.where(result[:, :, 0] != 0)
        min_row, max_row = min(rows), max(rows) + 1
        min_col, max_col = min(cols), max(cols) + 1
        final_result = result[min_row:max_row, min_col:max_col, :]
        return final_result
def main(argv1,argv2):
    img1 = cv2.imread(argv1)
    img2 = cv2.imread(argv2)
    final=Image_Stitching().blending(img1,img2)
    cv2.imwrite('panorama.jpg', final)
```

[7]:
```
if __name__ == '__main__':
    try:
        #main(sys.argv[1],sys.argv[2])
        #main("uttower_left.JPG","uttower_right.JPG")
```

```
        main("night1.jpg","night2.jpg")
    except IndexError:
        print ("Please input two source images: ")
        print ("For example: python Image_Stitching.py '/Users/jimutbp/Desktop/
↪picture/p1.jpg' '/Users/jimutbp/Desktop/picture/p2.jpg'")
```

[ ]:

disparity

May 2, 2020

[1]:
```
import numpy as np
import cv2
from matplotlib import pyplot as plt

imgL_ = cv2.imread('imL.png',0)
imgR_ = cv2.imread('imR.png',0)

stereo = cv2.StereoBM_create(numDisparities=16, blockSize=13)
```

[2]:
```
plt.imshow(imgL_,cmap='gray')
plt.show()
```

[3]:
```
shape_Res = max(imgL_.shape,imgR_.shape)
```

[4]:
```
imgL = cv2.resize(imgL_, (shape_Res[0], shape_Res[1]))
imgR = cv2.resize(imgR_, (shape_Res[0], shape_Res[1]))
```

[6]:
```
disparity = stereo.compute(imgL,imgR)

min_ = disparity.min()
max_ = disparity.max()
disparity = np.uint8(255*(disparity - min_)/(max_-min_))

plt.figure(figsize=(20,40))
plt.imshow(np.hstack((imgL,imgR,disparity)),'gray')
plt.show()
```