Computer Vision

Assignment 8

Keio University

CALAMVS GLADIO FORTIOR

1858

STEFAN Vlad 82123434

04/07/2022

1 Code

The code starts by loading two images that capture the same scene from slightly different perspectives. Then, image rectification is applied, and the disparity map is computed. In the end, the disparity map is used to compute an interpolated view between both captured images.

```python
def main(imgL_path, imgR_path, rectified, interpolation):
    """
    Compute the interpolated view between 2 images
    :param imgL_path: path to the first image
    :param imgR_path: path to the second image
    :param rectified: boolean used to specify if it is necessary to rectify both images
    :param interpolation: float between 1 and 2 that determines the intermediary camera
   position
    """
    # Read both images
    img1 = cv.imread(imgL_path, 0)
    img2 = cv.imread(imgR_path, 0)

    # Rectify the images if they are not
    if not rectified:
        img1, img2 = rectify(img1, img2)

    # compute the disparity map
    disparity = disparity_map(img1, img2)
    disparity_img = gray_image(disparity)
    cv.imwrite("disparity.jpg", disparity_img)

    # Use the disparity map to create an intermediary view
    intermediary = interpolate(interpolation, img1, img2, disparity)
    cv.imwrite("intermediary.jpg", intermediary)
```

If the images need to be rectified, the method described in the previous report is applied.

```python
def rectify(img1, img2):
    """
    Rectify 2 images to put them on the same plane
    :param img1: input image
    :param img2: input image
    :return: rectified images
    """
    # find the keypoints and descriptors with SIFT
    sift = cv.SIFT_create()
    kp1, des1 = sift.detectAndCompute(img1, None)
    kp2, des2 = sift.detectAndCompute(img2, None)

    # FLANN parameters
    FLANN_INDEX_KDTREE = 1
    index_params = dict(algorithm=FLANN_INDEX_KDTREE, trees=5)
    search_params = dict(checks=50)
    flann = cv.FlannBasedMatcher(index_params, search_params)
    matches = flann.knnMatch(des1, des2, k=2)
    pts1 = []
    pts2 = []

    # ratio test as per Lowe's paper
    for i, (m, n) in enumerate(matches):
        if m.distance < 0.8 * n.distance:
            pts2.append(kp2[m.trainIdx].pt)
            pts1.append(kp1[m.queryIdx].pt)

    pts1 = np.int32(pts1)
    pts2 = np.int32(pts2)

    # Compute the fundamental matrix
    F, mask = cv.findFundamentalMat(pts1, pts2, cv.FM_LMEDS)
    print(F)

    # We select only inlier points
    pts1 = pts1[mask.ravel() == 1]
    pts2 = pts2[mask.ravel() == 1]

    # Find epilines corresponding to points in right image (second image) and
    # drawing its lines on left image
    lines1 = cv.computeCorrespondEpilines(pts2.reshape(-1, 1, 2), 2, F)
    lines1 = lines1.reshape(-1, 3)
    img5, img6 = drawlines(img1, img2, lines1, pts1, pts2)

    # Find epilines corresponding to points in left image (first image) and
    # drawing its lines on right image
    lines2 = cv.computeCorrespondEpilines(pts1.reshape(-1, 1, 2), 1, F)
    lines2 = lines2.reshape(-1, 3)
    img3, img4 = drawlines(img2, img1, lines2, pts2, pts1)
    cv.imwrite("lines_left.jpg", img5)
    cv.imwrite("lines_right.jpg", img3)

    # Stereo rectification (uncalibrated variant)
    h1, w1 = img1.shape
    h2, w2 = img2.shape
    _, H1, H2 = cv.stereoRectifyUncalibrated(np.float32(pts1), np.float32(pts2), F, imgSize=(
    w1, h1))

    # Undistort (rectify) the images and save them
    img1_rectified = cv.warpPerspective(img1, H1, (w1, h1))
    img2_rectified = cv.warpPerspective(img2, H2, (w2, h2))

    cv.imwrite("rectified_left.jpg", img1_rectified)
    cv.imwrite("rectified_right.jpg", img2_rectified)
    return img1_rectified, img2_rectified
```

The disparity map is computed with the function `StereoSGBM_create`.

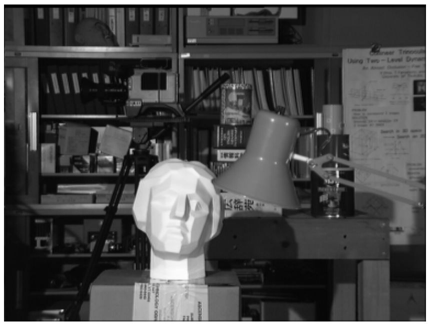
```python
def disparity_map(img1, img2):
    """
    Compute the disparity map between 2 images
    :param img1: input image
    :param img2: input image
    :return: disparity map
    """
    window_size = 5
    min_disp = -16
    num_disp = 16*2
    stereo = cv.StereoSGBM_create(minDisparity=min_disp,
                                  numDisparities=num_disp,
                                  blockSize=16,
                                  P1=8 * 3 * window_size ** 2,
                                  P2=32 * 3 * window_size ** 2,
                                  disp12MaxDiff=1,
                                  uniquenessRatio=10,
                                  speckleWindowSize=100,
                                  speckleRange=32
                                  )
    disparity = stereo.compute(img1, img2).astype(np.float32) / 16.0
    return disparity
```

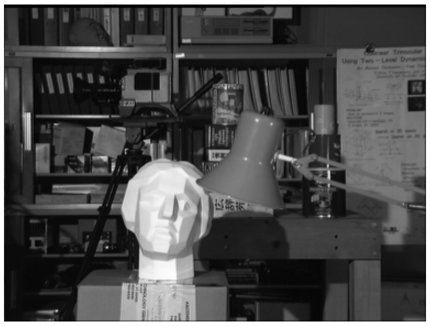
The disparity map is then used to create an intermediate view of the scene. A simple linear interpolation method is used.

```python
def interpolate(i, imgL, imgR, disparity):
    """
    Linear interpolation between 2 images
    :param i: float between 1 and 2 that determines the intermediary camera position
    :param imgL: image photographing a scene from the left
    :param imgR: image photographing a scene from the right
    :param disparity: disparity map between imgL and imgR
    :return: Interpolated view between imgL and imgR
    """
    ir = np.zeros_like(imgL)
    for y in range(imgL.shape[0]):
        for x1 in range(imgL.shape[1]):
            x2 = int(x1 - disparity[y, x1])  # correponding x position in imgR
            x_i = int((2 - i) * x1 + (i - 1) * x2)  # Intermediary x position
            if 0 <= x_i < ir.shape[1] and 0 <= x2 < imgR.shape[1]:
                ir[y, x_i] = int((2 - i) * imgL[y, x1] + (i - 1) * imgR[y, x2])
    return ir.astype(np.uint8)
```

2 Results

Figures 1a and 1b show the same scene photographed from different angles.


(a) Left picture


(b) Right picture

These pictures do not require rectification and can be used to compute the disparity map, which is displayed in figure 2.

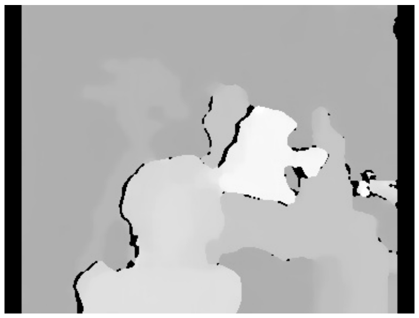


Figure 2: Disparity map

Images 1a, 1b, and disparity map 2 can be used to compute an intermediate view. The result is displayed in figure 3.

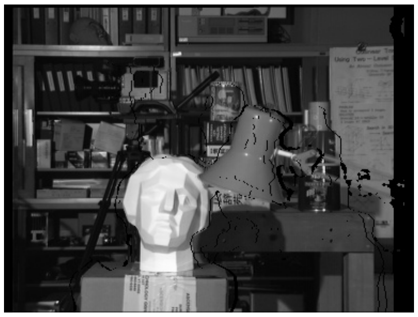
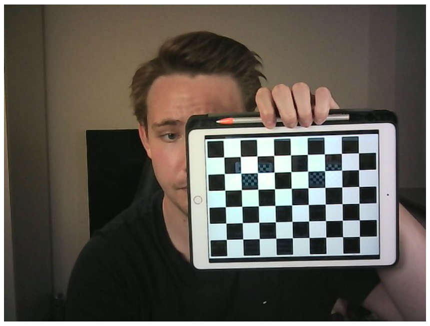
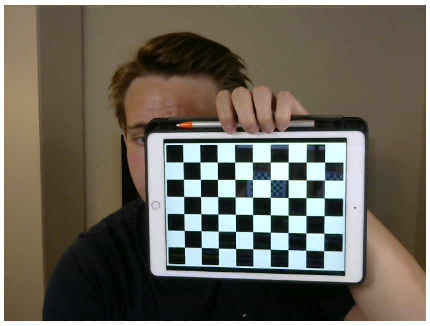


Figure 3: Interpolated view

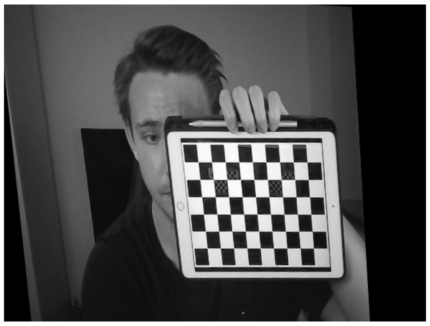
Figures 4a, 4b, 5a, 5b, 6, and 7 show the results obtained with another picture.



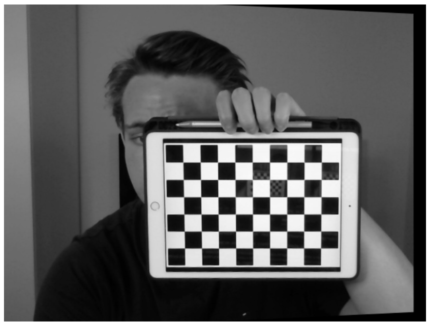
(a) Left picture



(b) Right picture



(a) Rectified left picture



(b) Rectified right picture

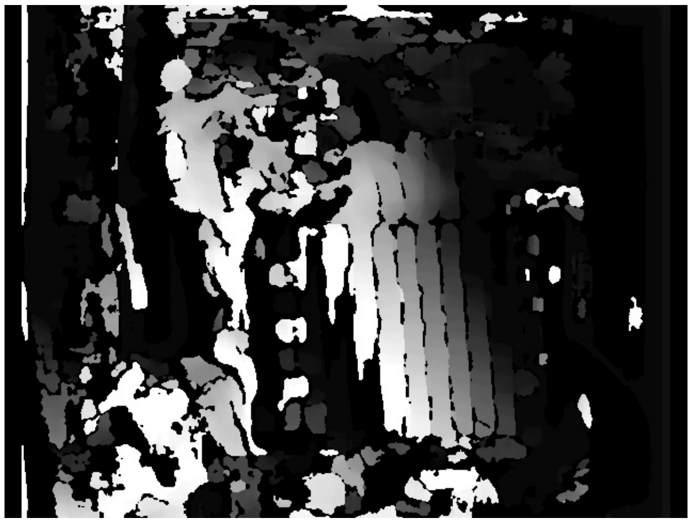


Figure 6: Disparity map

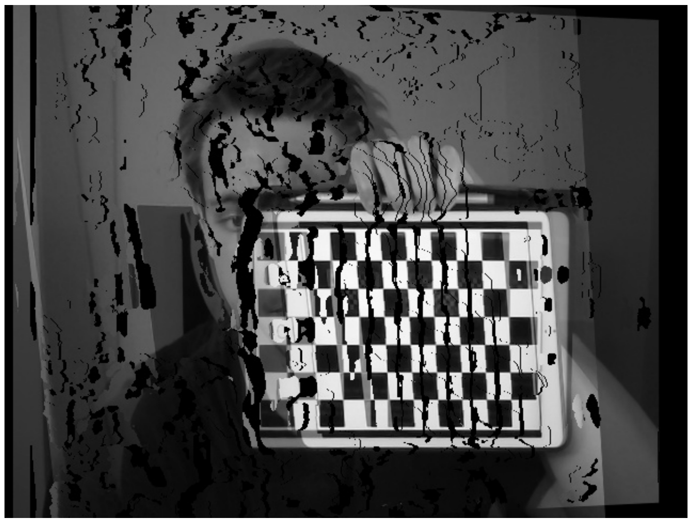


Figure 7: Interpolated view

It is clear from figures 3 and 7 that the result is not perfect. It could potentially be improved by filling some of the holes by interpolating pixel values from neighbouring pixels.
Another possibility would be to remove the noise in the original pictures and apply segmentation techniques to improve the disparity map.
The use of a more sophisticated interpolation technique could also improve the final result since currently a simple linear interpolation is applied [1].

[1]The code and test pictures are available at this link