Computer Vision

Assignment 7

Keio University

STEFAN Vlad 82123434
12/06/2022

1 Code

The code starts by computing the epipolar lines with the method described in the previous report.

```
import cv2 as cv
import numpy as np

def drawlines(img1, img2, lines, pts1, pts2):
    """
    Draw computed epilines
    :param img1: image on which we draw the epilines for the points in img2
    :param img2: image containing the points from which epilines are drawn in img1
    :param lines: corresponding epilines
    :param pts1: points in img1
    :param pts2: points in img2
    """
    r, c = img1.shape
    img1 = cv.cvtColor(img1, cv.COLOR_GRAY2BGR)
    img2 = cv.cvtColor(img2, cv.COLOR_GRAY2BGR)
    np.random.seed(0)
    for r, pt1, pt2 in zip(lines, pts1, pts2):
        color = tuple(np.random.randint(0, 255, 3).tolist())
        x0, y0 = map(int, [0, -r[2] / r[1]])
        x1, y1 = map(int, [c, -(r[2] + r[0] * c) / r[1]])
        img1 = cv.line(img1, (x0, y0), (x1, y1), color, 1)
        img1 = cv.circle(img1, tuple(pt1), 5, color, -1)
        img2 = cv.circle(img2, tuple(pt2), 5, color, -1)
    return img1, img2

# Read the images
img1 = cv.imread('left.jpg', 0)  # queryimage # left image
img2 = cv.imread('right.jpg', 0)  # trainimage # right image

# 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)
```

Afterwards, the homography matrices that project both images on the same plane are computed with `stereoRectifyUncalibrated`. These matrices can then be used to rectify both images.

```
# 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.png", img1_rectified)
cv.imwrite("rectified_right.png", img2_rectified)
```

To draw the feature points and epipolar lines on the rectified images, two functions are created. The first one, `warp_pts`, applies a homography on an array of positions. The second one, `draw_line_H`, uses the first function to rectify the epipolar lines and the position of feature points.

```
def warp_pts(pts, M):
    """
    Apply a homography to an array of points
    :param pts: array of 2D points
    :param M: Homography matrix
    :return: Array of points after the homography
    """
    second = np.copy(pts)
    for l in range(len(pts)):
        p = pts[l]
        px = (M[0][0] * p[0] + M[0][1] * p[1] + M[0][2]) / (M[2][0] * p[0] + M[2][1] * p[1] +
     M[2][2])
        py = (M[1][0] * p[0] + M[1][1] * p[1] + M[1][2]) / (M[2][0] * p[0] + M[2][1] * p[1] +
     M[2][2])
        second[l] = np.array([int(px), int(py)])
    return second


def draw_line_H(img1, lines, pts, H):
    """
        Draw recified epilines
        :param img1: image on which we draw the epilines for the points in img2
        :param lines: corresponding epilines
        :param pts: feature points in img1
        :param H: homography matrix of img1
        :return: Image with feature points and epipolar lines drawn
        """
    r, c = img1.shape
```

```
    dst1 = cv.cvtColor(img1, cv.COLOR_GRAY2BGR)
    np.random.seed(0)
    for r, pt in zip(lines, pts):
        color = tuple(np.random.randint(0, 255, 3).tolist())
        x0, y0 = map(int, [0, -r[2] / r[1]])
        x1, y1 = map(int, [c, -(r[2] + r[0] * c) / r[1]])
        pts_warp = np.array([[x0, y0], [x1, y1], pt])
        pts_warp = warp_pts(pts_warp, H)
        dst1 = cv.line(dst1, tuple(pts_warp[0]), tuple(pts_warp[1]), color, 1)
        dst1 = cv.circle(dst1, tuple(pts_warp[2]), 5, color, -1)
    return dst1
```

With these functions, the epipolar lines and feature points are drawn on each image.

```
# Draw the warped feature points and epipolar lines on each image
img1_rectified_lines = draw_line_H(img1_rectified, lines1, pts1, H1)
img2_rectified_lines = draw_line_H(img2_rectified, lines2, pts2, H2)

cv.imwrite("rectified_left_lines.png", img1_rectified_lines)
cv.imwrite("rectified_right_lines.png", img2_rectified_lines)
```

To finish everything, a disparity map is generated with the function `StereoBM_create`.

```
# Create a disparity map
stereo = cv.StereoBM_create(numDisparities=16, blockSize=5)
disparity = stereo.compute(img1_rectified, img2_rectified)
cv.imwrite("disparity.png", disparity)
```

The previously described code has been adapted from [1][1].

2 Results

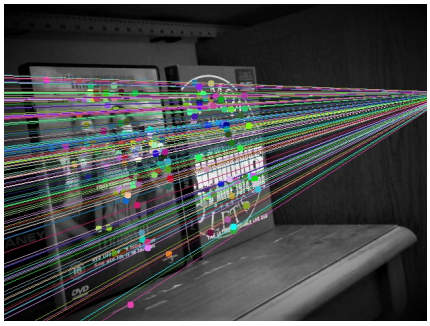
Figures 1a and 1b show the same scene photographed from different angles.

(a) Left picture

(b) Right picture

With the method described in the previous report, epipolar lines and their corresponding points are traced, resulting in figures 2a and 2b.

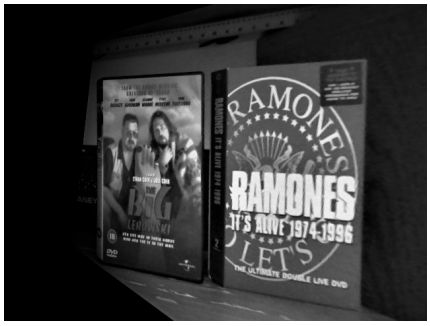
[1]The code and test pictures are available at this link



(a) Left picture with epipolar lines

(b) Right picture with epipolar lines

After using a homography to rectify both pictures, images 3a and 3b are obtained.



(a) Rectified left picture

(b) Rectified right picture

After projecting the feature points and epipolar lines, it is easy to see that the epipolar lines have become horizontal in figures 4a and 4b.

(a) Rectified left picture with epipolar lines

(b) Rectified right picture with epipolar lines

The rectified images are used to obtain the depth map of the scene, which is shown in figure 5. As expected, the result is very noisy, but it could probably be improved with global optimization or segmentation techniques.

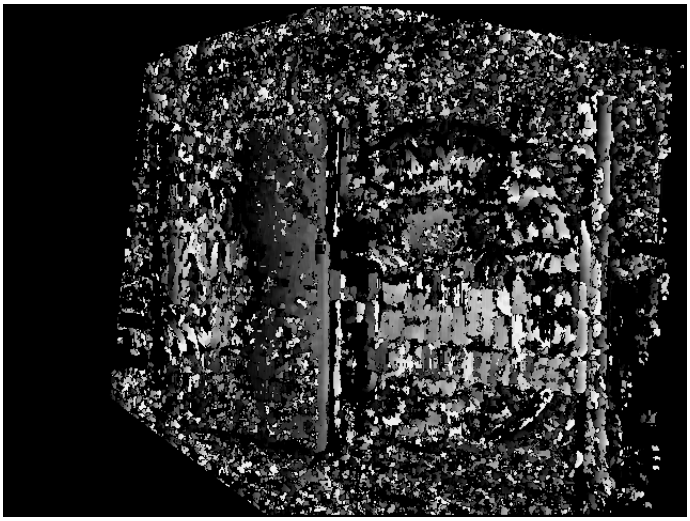


Figure 5: Depth map

References

[1] Andreas Jakl. *Understand and Apply Stereo Rectification for Depth Maps (Part 2)*. URL: `https://www.andreasjakl.com/understand-and-apply-stereo-rectification-for-depth-maps-part-2/` (visited on 06/13/2022).