Yuancheng Shao, ys3420, 01886903; Xinyang Sun, xs920, 01912874

1. Collect Data

Cardbox We made a 7x7 cardbox image using Opencv in Pyhton and then print it on A4 papers. Initially, we searched that the width of an standard A4 paper is 21cm, so every length of the square is set as 3 cm for exact division. However, after printing, there is a system error that every printer will reserve a margin of at least 3mm. Thus, the modified square length in our final 7x7 cardbox is 28.5mm. The overall cardbox is shown in Figure 1.

Figure 1. Cardbox

Collect data We have three reasons for choosing a bottle as our object shown in Figure 2.: the color of the bottle which has a high contrast with the cardbox so that it is easier for recognition even in gray scale; the size and the outline of the bottle is very suitable in square cardbox for detection; the texture of the bottle does not reflect too much light so the noise can be minimized.
We used a FUJIFILM X-T200 with a VILTROX 23MM F1.4 lens to take all the FD and HG pictures most of them are shown in appendix.

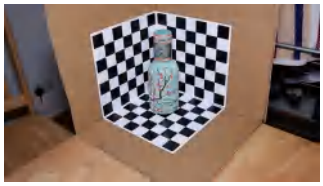


Figure 2. Object in the Cardbox

Clarification The size of the picture transformed to coordinates value which is 6000 unit pixels in x axis (from left to right) and 3376 unit pixels in y axis (from up to down), the coordinates origin is located at the top-left vertex.

2. Keypoint correspondences between images

Manual Correspondences by clicking manually are shown in Figure 3. We generate a user interface by Matlab to select several control points in related pair images by our eyes estimation and make lines between each pair.

Figure 3. Manual corresponding points by clicking

Automatic For automatic part shows in Figure 4, we applied Harris–Stephens algorithm to detect features[5]. And then we used MSAC algorithm, which is a modification of the random sample consensus (RANSAC) algorithm to exclude outliers. The main idea of MSAC is to evaluate the quality of the consensus set calculating its likelihood [1, 9].
Harris–Stephens algorithm can be divided into 5 steps:
Convert color image to grayscale image$\longrightarrow$ Spatial derivative calculation$\longrightarrow$ Structure tensor setup$\longrightarrow$ Harris response calculation$\longrightarrow$ Non-maximum suppression

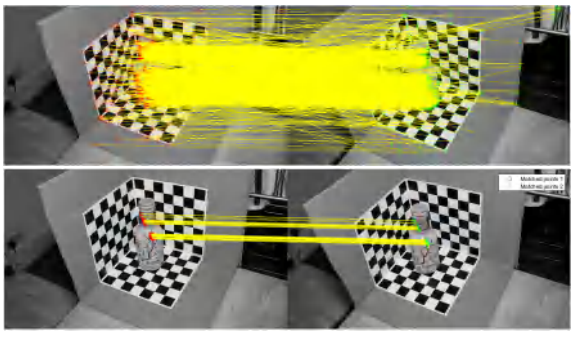


Figure 4. Above:point matches; Below:inlier point matches

Comparison From Figure 3 and Figure 4, the quality of correspondences in both images are performed well. But in actual tests, for automatic part, lots of images pairs were tested to get Figure 4, and some of image pairs which indicate poor performances are shown in appendix. In manual part, we can always connect correspondences by our eyes estimation. Thus for estimating quality in automatic method, we consider the manual correspondence as ground truth reference. After analysis, the average deviation distance between manual and automatic is 41.7, the standard deviation is 27.64. However, for quantity of correspondences, manual method is time consuming, we approximately connect 15 correspondences with 3 minutes, whereas in automatic method, plenty of correspondences can be generated within a relative much shorter time.
In another word, we obtain a higher quality with manual method and a higher quantity with automatic method.

3. Camera calibration

Camera parameters After applying *Camera Calibration Toolbox* for Matlab on 15 photos from FD, we got camera parameters and camera parameter uncertainties shown in Table 1 [4, 6].

parameter	horizontal(pixel)	vertical(pixel)
F_c	6357.185	6344.536
F_{ce}	93.071	106.957
C_c	2864.732	1924.784
C_{ce}	193.715	179.783

Table 1. Camera parameters and uncertainties

F_c:Focal length; F_{ce}:Focal length uncertainty
C_c:Principal point; C_{ce}:Principal point uncertainty

Skew and distortion With *Camera Calibration Toolbox* for Matlab, we got skew and distortion coefficients and uncertainties.

$$\alpha_c = 0.0, \alpha_{ce} = 0.0$$

$$K_c = -0.143; 0.358; 0.008; -0.004; 0.0$$

$$K_{cw} = 0.101; 0.609; 0.009; 0.006; 0.0$$

α_c:Skew coefficient; α_{ce}:Skew coefficient uncertainty
K_c:Distortion coefficients
K_{ce}:Distortion coefficients uncertainty

A part of process and undistort images will be shown in the appendix.

4. Transformation estimation

Homography matrix We use same program as Task2 automatic part to detect inliers, make sure at least 4 correspondences are recognised shown in Figure 5 and then compute a homography matrix shows in matrix H. Worst case with zoomed image is shown in appendix.

$$H = \begin{bmatrix} 0.968 & 0.251 & 345.440 \\ -0.251 & 0.968 & -592.721 \\ 0 & 0 & 1 \end{bmatrix}$$

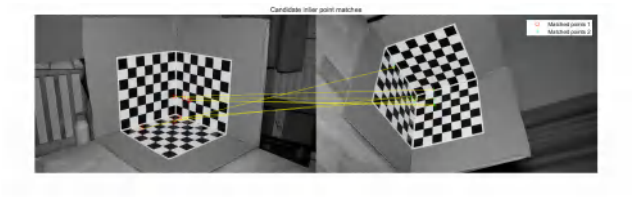


Figure 5. Correspondences for homography matrix

Fundamental Matrix The fundamental matrix between the images pair is shown below, there are eight degrees of freedom here. From formula $X_l^T F X_R = 0$, it needs at least eight equations to solve this matrix. After finding the fundamental matrix, we can use this relation to find the epipolar lines and epipoles.

$$F = \begin{bmatrix} 2.34 * 10^{-8} & 6.07 * 10^{-7} & -6.8 * 10^{-4} \\ -4.12 * 10^{-7} & 3.8 * 10^{-8} & 1.18 * 10^{-3} \\ 2.13 * 10^{-4} & 1.6 * 10^{-3} & 1 \end{bmatrix}$$

Figure 6. Keypoints and epipolar lines in the other image

The left part of Figure 6 shows the detected keypoints, the right part of figure shows their corresponding epipolar lines (in different colours). From Figure 7,epipoles are in both images which coordinates are (3078.8, 995.6) and (2707.9, 674.3) respectively. The intersections of colourful epipolar lines are the two epipoles. The two different shooting camera centers have great influences on epipole locations, When two image planes are at a great enough relative angle to each other, the epipoles are more possible to appear in the image. Some cases for epipoles outside the images are shown in appendix. From Figure 8, the red and blue

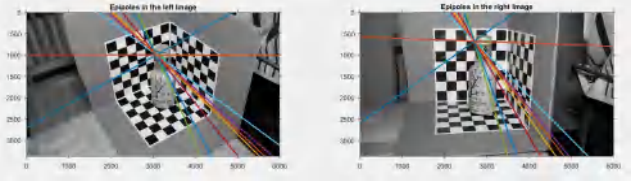


Figure 7. Epipoles in both images

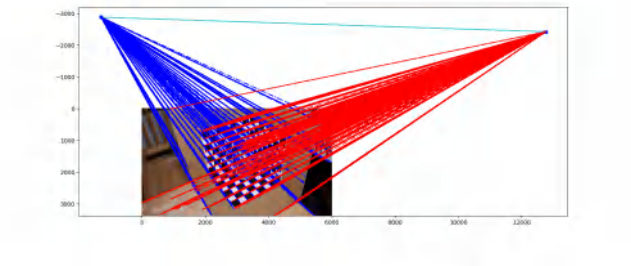


Figure 8. Vanishing points and horizon

line sets are two dominate vanishing line sets which generated from parallel lines in their respective plane. RANSAC model was used to detect these edges in picture from point clouds [2][3]. The intersections of lines are vanish points which all locates outside the image and their coordinates are (-1290, -2880) and (12755, -2423). The connection indigo blue line is the double points perspective projection corresponding horizon line [8].

5. 3D geometry

Rectification In rectification part, we derive the fundamental matrix using correspondences detected by Hough transform. Then we warp the by the fundamental matrix shown in matrix F and detect correspondences again. With new correspondences, we draw epipolar lines shown in Figure 9. Some cases and worst case are shown in appendix.

$$F = \begin{bmatrix} -2.045 & -1.515 & -2.616 \\ 3.180 & 1.324 & -1.909 \\ 1.030 & 2.180 & 1 \end{bmatrix}$$

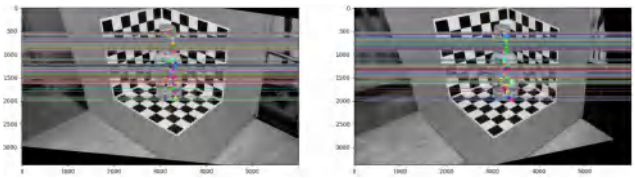


Figure 9. Stereo rectified pair

Depth map We used two algorithms to derive depth map. One is block matching (BM), the other is semi-global block matching algorithm (SGBM). These two algorithms have a little difference because they share a same theory. Both algorithms compute disparity, which is difference in location of an object in corresponding two (left and right) images, by a window/block. After some tests, we set a block size of 5 and a number of disparities of 16 and 13 separately in BM algorithm and SGBM algorithm for best result shown in Figure 10 and Figure 11 [7]. With BM algorithm, we got the depth of bottle at approximately 130mm, and with SGBM algorithm, we got the depth of bottle at approximately 80mm. BM algorithm performs better on our case.

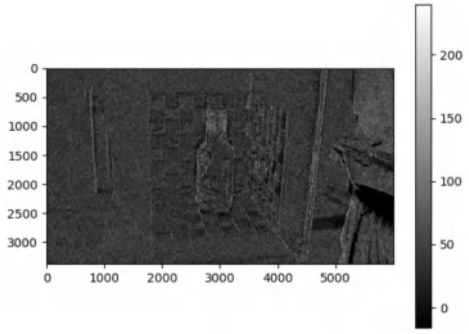


Figure 10. Depth map derived by BM

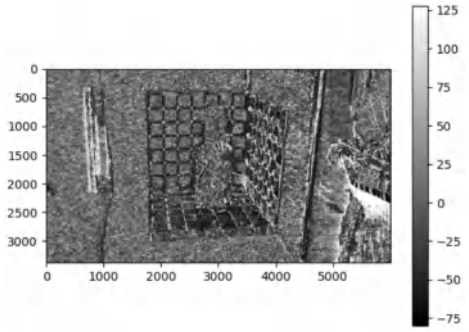


Figure 11. Depth map derived by SGBM

6. Conclusions

In this coursework, we implemented and analysed FD and HG first, then applying Matlab *Camera Calibration Toolbox* to compute camera parameters, uncertainties, skew and distortion coefficients.[4, 6]. Afterwards, the detailed transformation estimations were achieved. For the final rectification part, we used Hough transform to detect correspondences twice for warping images using fundamental matrix and computing epipolar lines. For depth map part, we applied BM and SGBM algorithms to derive disparity between a pair of images then we could get a depth map.

References

[1] A. M. Andrew. Multiple view geometry in computer vision. *Kybernetes*, 2001.

[2] J.-C. Bazin and M. Pollefeys. 3-line ransac for orthogonal vanishing point detection. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4282–4287. IEEE, 2012.

[3] K. Chaudhury, S. DiVerdi, and S. Ioffe. Auto-rectification of user photos. In *2014 IEEE International Conference on Image Processing (ICIP)*, pages 3479–3483. IEEE, 2014.

[4] C. B. Duane. Close-range camera calibration. *Photogramm. Eng*, 37(8):855–866, 1971.

[5] C. G. Harris, M. Stephens, et al. A combined corner and edge detector. In *Alvey vision conference*, volume 15, pages 10–5244. Citeseer, 1988.

[6] J. Heikkila and O. Silvén. A four-step camera calibration procedure with implicit image correction. In *Proceedings of IEEE computer society conference on computer vision and pattern recognition*, pages 1106–1112. IEEE, 1997.

[7] H. Hirschmuller. Stereo processing by semiglobal matching and mutual information. *IEEE Transactions on pattern analysis and machine intelligence*, 30(2):328–341, 2007.

[8] D. B. Thomson, M. P. Mepham, R. R. Steeves, et al. *The stereographic double projection*. Department of Surveying Engineering, University of New Brunswick, 1977.

[9] P. H. Torr and A. Zisserman. Mlesac: A new robust estimator with application to estimating image geometry. *Computer vision and image understanding*, 78(1):138–156, 2000.

7. Appendix

7.1. Images and data

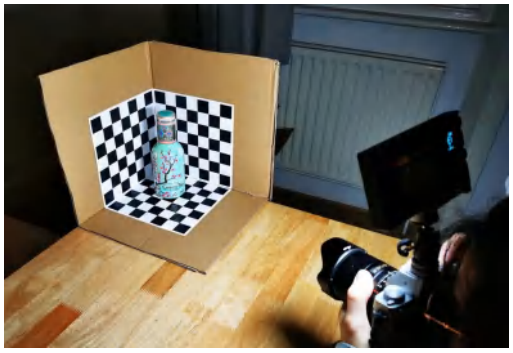
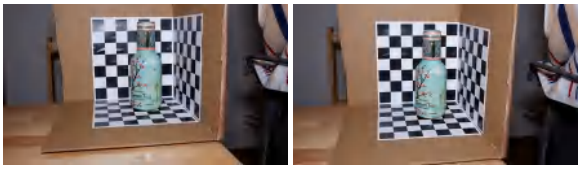


Figure 12. Process of taking photos



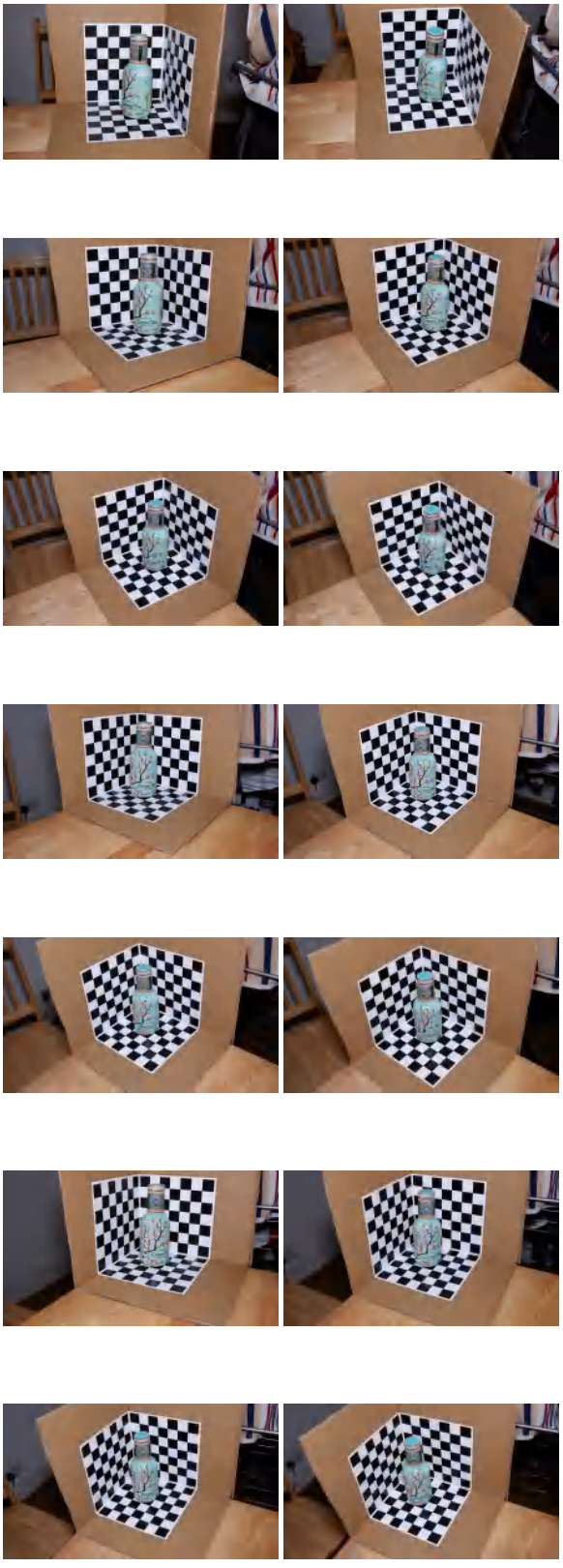
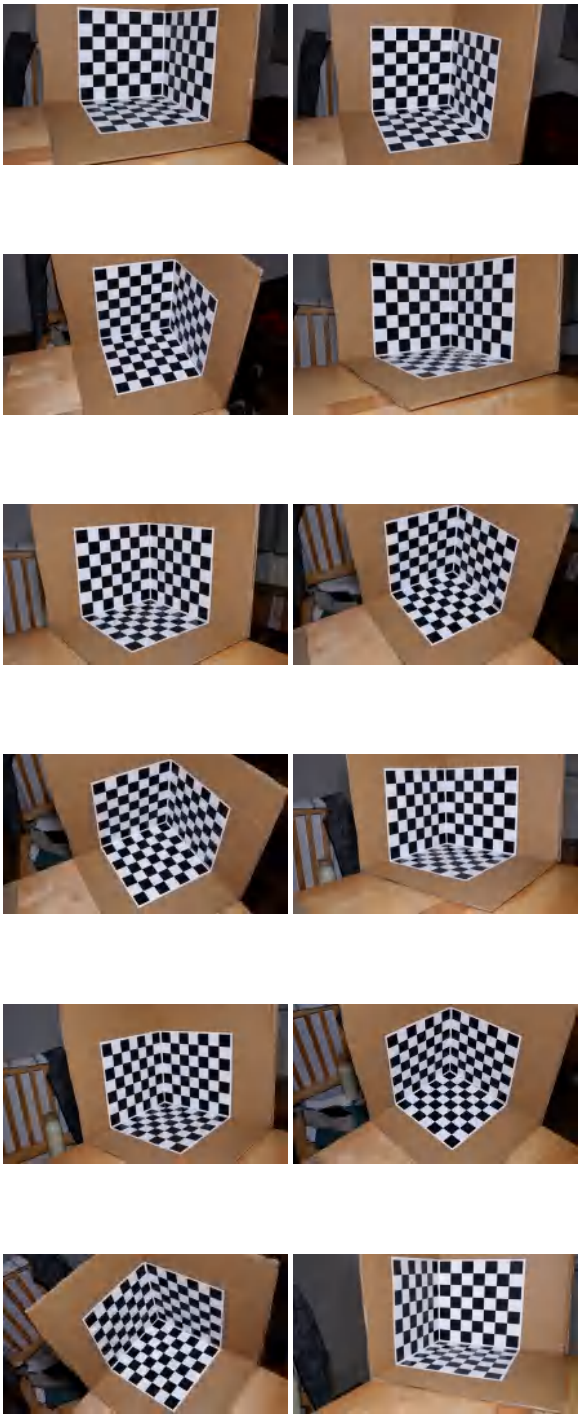


Figure 13. FD data with object



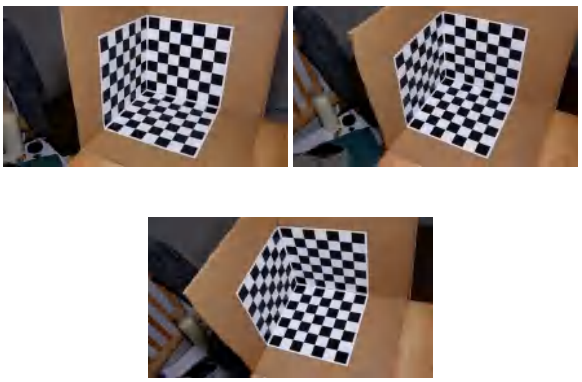
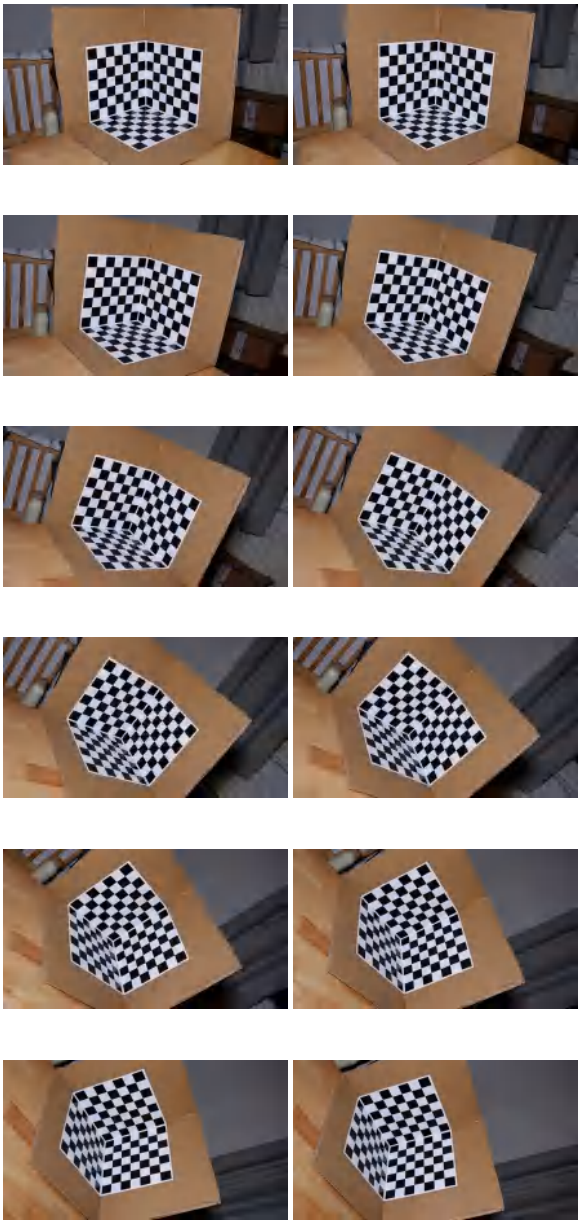


Figure 14. FD data without object



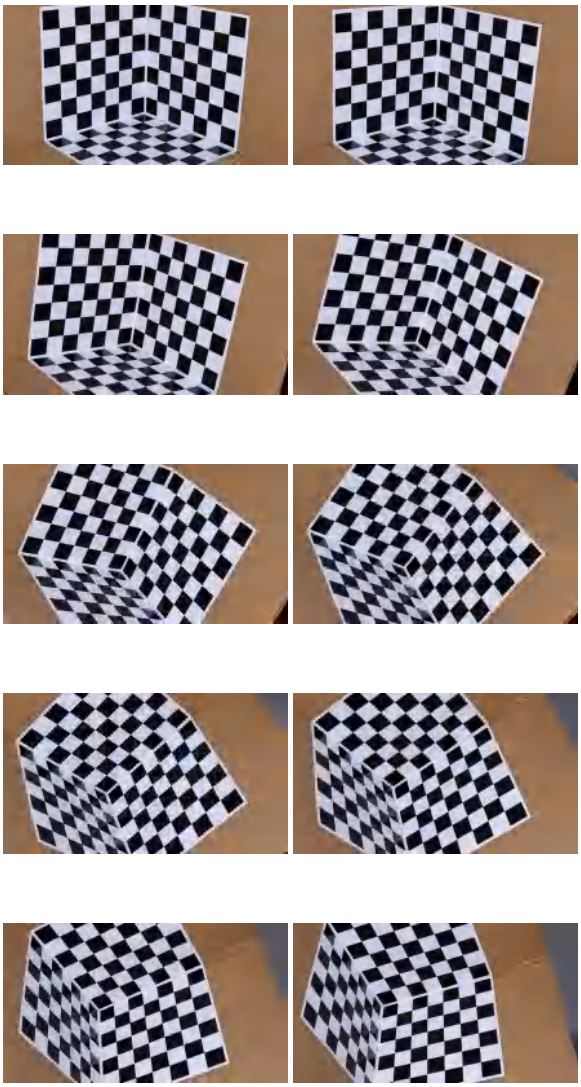
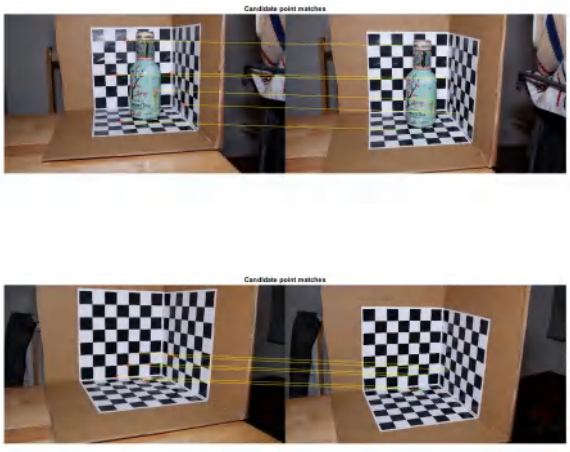


Figure 15. HG data



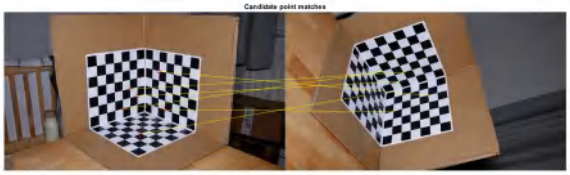


Figure 16. Manual keypoint correspondences

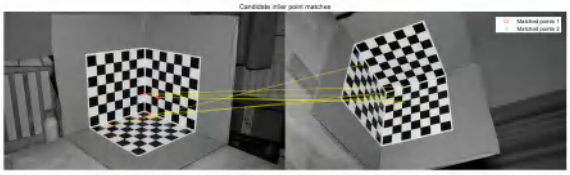


Figure 17. Automatic keypoint correspondences (a)

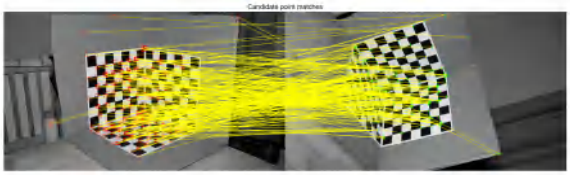


Figure 18. Automatic keypoint correspondences with outliers (a)

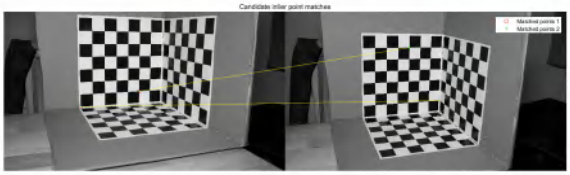


Figure 19. Automatic keypoint correspondences (b) - worst case

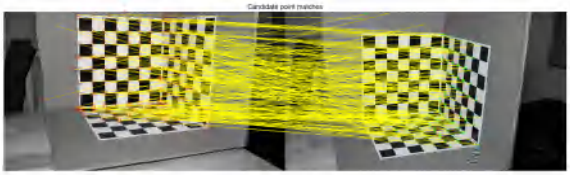


Figure 20. Automatic keypoint correspondences with outliers (b)

Manual Matched Points (x)	Manual Matched Points (y)
2692.291045	1060.88806
2693.783582	1081.783582
2684.828358	1111.634328
2695.276119	1163.873134
2720.649254	1210.141791
2723.634328	1235.514925
2723.634328	1268.350746
2768.410448	1317.604478
2916.171642	1459.395522
2911.69403	1487.753731
2905.723881	1520.589552
2908.708955	1547.455224
2950.5	1614.619403
2951.992537	1635.514925
2937.067164	1599.69403

Table 2. Data for comparison on manual methods

Automatic Matched Points (x)	Automatic Matched Points (y)
2687.4192	1091.6533
2691.1814	1086.0204
2701.6711	1129.2286
2703.0984	1149.7336
2757.9895	1227.395
2746.7266	1202.1846
2797.4495	1268.257
2768.1768	1246.7976
2916.6179	1505.5796
2915.4519	1487.9874
2907.7729	1522.9061
2908.5344	1499.6327
2951.2449	1526.8557
2952.3018	1566.3949
2938.5984	1534.7781

Table 3. Data for comparison on automatic methods

Calibration Process:
1. Read images
2. Extract grid corners

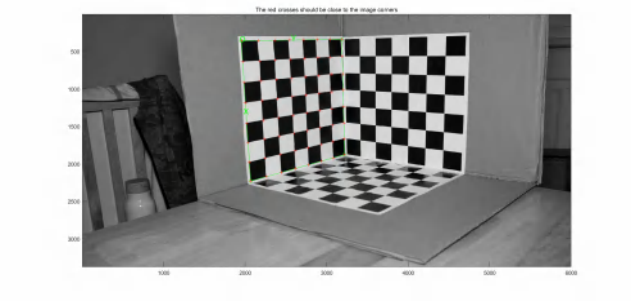


Figure 21. Extract grid corners

3. Calibration

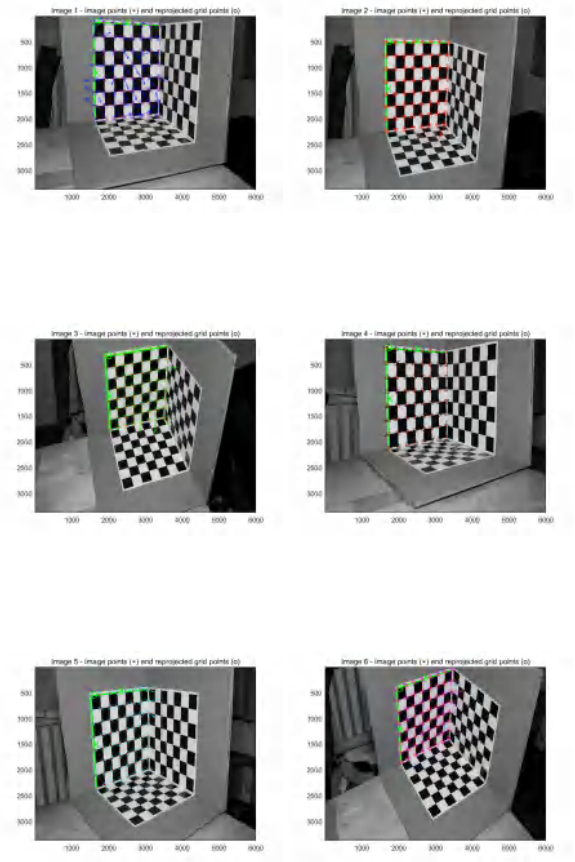


Figure 22. Reprojected pictures

4. Export calibration data

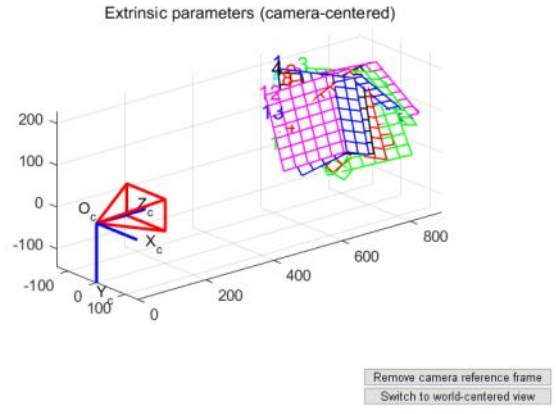


Figure 23. Extrinsic parameters

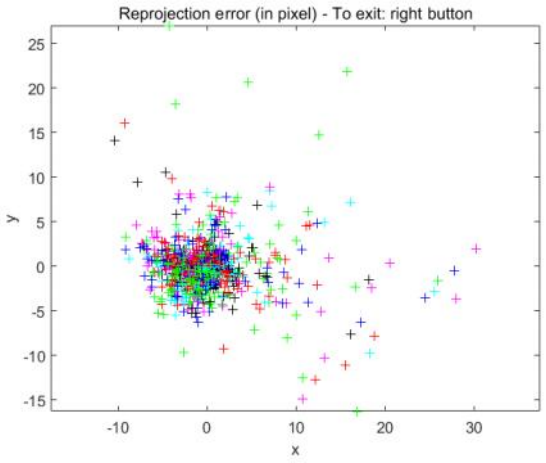


Figure 24. Reprojection error

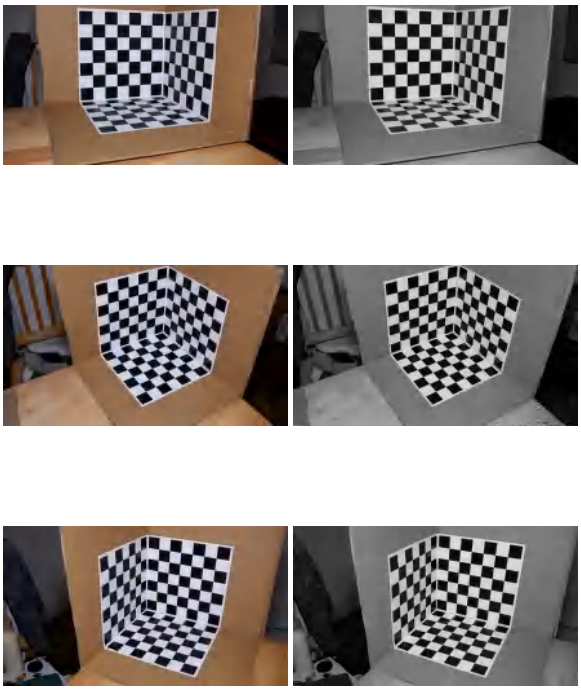


Figure 25. Undistort iamges

Homography matrix between zoomed images:

$$H = \begin{bmatrix} 0.909 & 0.417 & 1638.849 \\ -0.417 & 0.909 & -980.5 \\ 0 & 0 & 1 \end{bmatrix}$$

Figure 26. Homography matrix - worst case

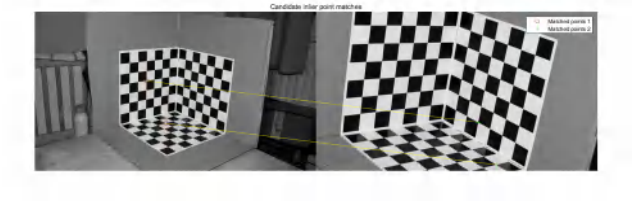


Figure 27. Correspondences for homography matrix - worst case

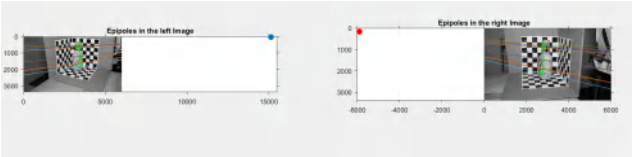


Figure 28. Epipoles outside: for another case on epipoles computing, when the image planes are at a more subtle angle to each other, the epipoles all locate outside of the images.

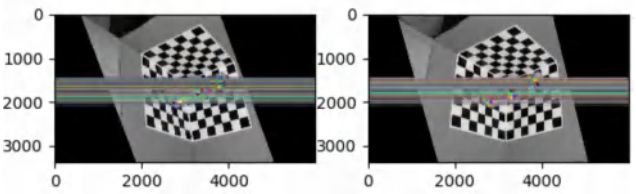


Figure 29. Rectification (a)

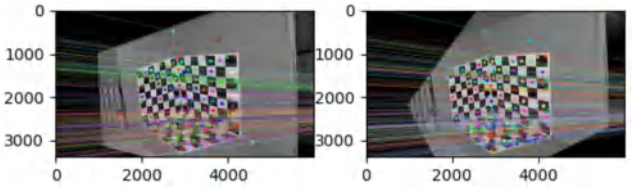


Figure 30. Rectification (b)

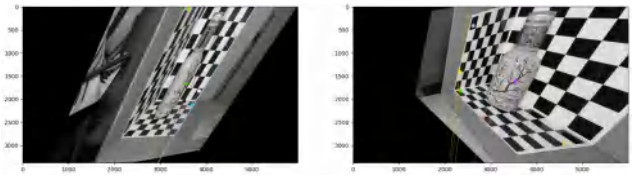


Figure 31. Rectification (c) - worst case

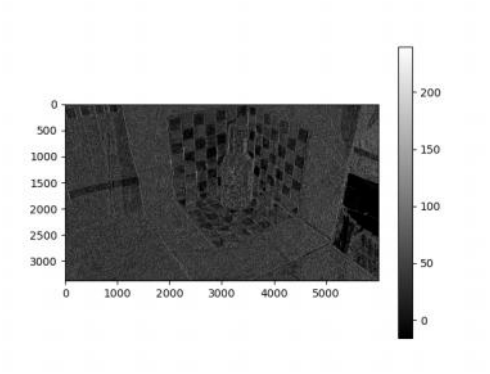


Figure 32. Depth map derived by BM (a)

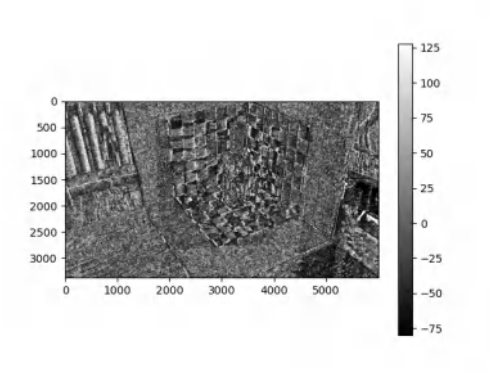


Figure 33. Depth map derived by SGBM (a)

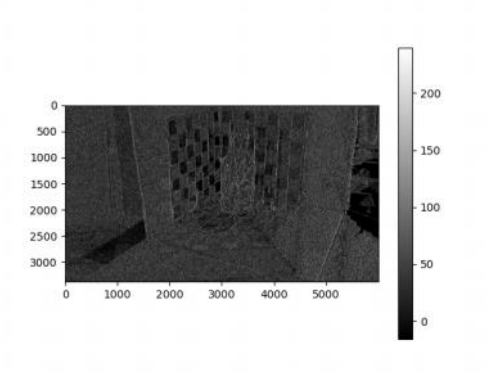


Figure 34. Depth map derived by BM (b) - worst case

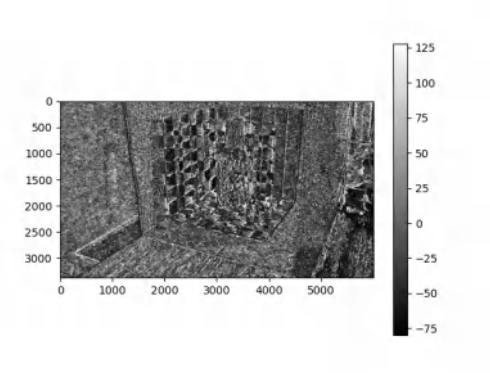


Figure 35. Depth map derived by SGBM (b) - worst case

7.2. Implementation codes

Cardbox python code:

```python
import cv2
import numpy as np
width = 700
height = 700
length = 100
image = np.zeros((width,height)
                 ,dtype = np.uint8)
print(image.shape[0],image.shape[1])

for j in range(height):
    for i in range(width):
        if((int)(i/length)
                + (int)(j/length))%2:
            image[i,j] = 255;
cv2.imwrite("cardbox1.jpg",image)
cv2.imshow("Cardbox",image)
cv2.waitKey(0)
```

Manual keypoint correspondences Matlab code:

```
I1 = imread('DSCF0178.JPG');
I2 = imread('DSCF0179.JPG');
[mp,fp] = cpselect(I1,I2,'Wait',true);
t = fitgeotrans(mp,fp,'lwm',6);
figure(); ax = axes;
showMatchedFeatures(I1,I2,mp,fp,'montage'
,'Parent',ax);
title(ax, 'Candidate point matches');
```

Automatic keypoint correspondences Matlab code:

```
I1 = rgb2gray(imread('DSCF0219.JPG'));
I2 = rgb2gray(imread('DSCF0220.JPG'));
points1 = detectHarrisFeatures(I1
,'MinQuality',0.02);
points2 = detectHarrisFeatures(I2
,'MinQuality',0.02);
[f1, vpts1] = extractFeatures(I1, points1);
[f2, vpts2] = extractFeatures(I2, points2);
indexPairs = matchFeatures(f1, f2, 'Method',
'NearestNeighborSymmetric','MatchThreshold'
,80);
matchedPoints1 = vpts1(indexPairs(:, 1));
matchedPoints2 = vpts2(indexPairs(:, 2));
[tform,inlierIdx] =
estimateGeometricTransform2D(
matchedPoints1(:),
matchedPoints2(:),'rigid');
inlierPtsDistorted =
matchedPoints1(inlierIdx,:);
inlierPtsOriginal  =
matchedPoints2(inlierIdx,:);
figure(); ax = axes;
showMatchedFeatures(I1,I2,matchedPoints1
,matchedPoints2,'montage','Parent',ax);
title(ax, 'Candidate point matches');
figure();ax = axes;
showMatchedFeatures(I1,I2,inlierPtsDistorted
,inlierPtsOriginal,'montage','Parent',ax);
title(ax, 'Candidate inlier point matches');
legend(ax, 'Matched points 1'
,'Matched points 2');
```

Calculate epipolar lines and epipoles Matlab code:

```
I1 = rgb2gray(imread('DSCF0399.JPG'));
%left image
I2 = rgb2gray(imread('DSCF0393.JPG'));
%right image

points1 = detectHarrisFeatures(I1);
points2 = detectHarrisFeatures(I2);
[features1,valid_points1]
= extractFeatures(I1,points1);
[features2,valid_points2]
= extractFeatures(I2,points2);
indexPairs =
matchFeatures(features1,features2);
matchedPoints1 =
valid_points1(indexPairs(:,1),:);
matchedPoints2 =
valid_points2(indexPairs(:,2),:);

[fLMedS,inliers] =
estimateFundamentalMatrix(
matchedPoints1,matchedPoints2,
'NumTrials',4000);
figure(); showMatchedFeatures(I1,I2,
matchedPoints1(inliers,:),
matchedPoints2(inliers,:),'montage');
%% Epipolar lines
I1 = rgb2gray(imread('DSCF0399.JPG'));
%I1 = imread('DSCF0400.JPG');
figure();
subplot(121);
imshow(I1);
title('Points in Left Image'); hold on;
plot(matchedPoints1.Location(inliers,1),
matchedPoints1.Location(inliers,2),'go')

I2 = rgb2gray(imread('DSCF0393.JPG'));
%I2 = imread('DSCF0392.JPG');
subplot(122);
imshow(I2);
title('Correspondences and
Epipolar Lines in the right Image');
hold on;
plot(matchedPoints2.Location(inliers,1),
matchedPoints2.Location(inliers,2),'go')
epiLines_2 = epipolarLine(
fLMedS,matchedPoints1.Location(
inliers,:));
points_2 = lineToBorderPoints(
epiLines_2,size(I2));
line(points_2(:,[1,3])',
```

```
points_2(:,[2,4])','linewidth',2);
truesize;
%% Epipoles
I1 = rgb2gray(imread('DSCF0399.JPG'));
%I1 = imread('DSCF0400.JPG');
figure();
subplot(121);
imshow(I1);
axis on
axis ([0 6000 0 3376]);
title('Epipoles in the left Image');
 hold on;
plot(matchedPoints1.Location(inliers,1)
,matchedPoints1.Location(inliers,2),'go')
epiLines_1 = epipolarLine(fLMedS',
matchedPoints2.Location(inliers,:));
points_1 = lineToBorderPoints(
epiLines_1,size(I1));
line(points_1(:,[1,3])',points_1(
:,[2,4])','linewidth',2);

I2 = rgb2gray(imread('DSCF0393.JPG'));
%I2 = imread('DSCF0392.JPG');
subplot(122);
imshow(I2);
axis on
axis ([0 6000 0 3376]);
title('Epipoles in the right Image');
 hold on;
plot(matchedPoints2.Location(inliers,1)
,matchedPoints2.Location(
inliers,2),'go')
epiLines_2 = epipolarLine(
fLMedS,matchedPoints1.Location(
inliers,:));
points_2 = lineToBorderPoints(
epiLines_2,size(I2));
line(points_2(:,[1,3])',points_2(
:,[2,4])','linewidth',2);
%truesize;

imageSize_1=size(I1);
[isIn_1,epipole_1] =
isEpipoleInImage(fLMedS,imageSize_1);

imageSize_2=size(I2);
[isIn_2,epipole_2] =
isEpipoleInImage(fLMedS',imageSize_2);
```

Compute vanishing points and horizon python code:

```
from skimage import feature,color
,transform, io
import numpy as np
import logging

def compute_edgelets(image, sigma=3):
    """Create edgelets."""
    gray_img = color.rgb2gray(image)
    edges = feature.canny(gray_img
    , sigma)
    lines =
    transform.probabilistic_hough_line(
    edges, line_length=3,line_gap=2)

    locations = []
    directions = []
    strengths = []

    for p0, p1 in lines:
        p0, p1 = np.array(p0)
        , np.array(p1)
        locations.append((p0+p1)/2)
        directions.append(p1 - p0)
        strengths.append(
        np.linalg.norm(p1 - p0))

 # convert to numpy arrays and normalize
    locations = np.array(locations)
    directions = np.array(directions)
    strengths = np.array(strengths)

    directions = np.array(directions) / \
        np.linalg.norm(directions
        , axis=1)[:, np.newaxis]

    return (locations, directions
    , strengths)

def edgelet_lines(edgelets):
    """Compute lines in homogenous
     system for edglets."""
    locations, directions, _ = edgelets
    normals = np.zeros_like(directions)
    normals[:, 0] = directions[:, 1]
    normals[:, 1] = -directions[:, 0]
    p = -np.sum(locations * normals
    , axis=1)
    lines = np.concatenate((normals
    , p[:, np.newaxis]), axis=1)
    return lines

def compute_votes(edgelets, model
, threshold_inlier=5):
    """Compute votes for each of
    the edgelet against a given
     vanishing point."""
    vp = model[:2] / model[2]

    locations, directions, strengths
     = edgelets

    est_directions = locations - vp
    dot_prod = np.sum(est_directions
     * directions, axis=1)
    abs_prod = np.linalg.norm(
    directions, axis=1) * \
```

```
        np.linalg.norm(est_directions
        , axis=1)
    abs_prod[abs_prod == 0] = 1e-5

    cosine_theta = dot_prod / abs_prod
    theta = np.arccos(np.abs(cosine_theta))

    theta_thresh = threshold_inlier
     * np.pi / 180
    return (theta < theta_thresh)
    * strengths

def ransac_vanishing_point(edgelets
, num_ransac_iter=2000, threshold_inlier=5):
    """Estimate vanishing point using
     Ransac."""
    locations, directions, strengths =
     edgelets
    lines = edgelet_lines(edgelets)

    num_pts = strengths.size

    arg_sort = np.argsort(-strengths)
    first_index_space = arg_sort[
    :num_pts // 5]
    second_index_space = arg_sort[
    :num_pts // 2]

    best_model = None
    best_votes = np.zeros(num_pts)

    for ransac_iter in range(
    num_ransac_iter):
        ind1 = np.random.choice(
        first_index_space)
        ind2 = np.random.choice(
        second_index_space)

        l1 = lines[ind1]
        l2 = lines[ind2]

        current_model = np.cross(l1, l2)

        if np.sum(current_model**2)
         < 1 or current_model[2] == 0:
          # reject degenerate candidates
            continue

        current_votes = compute_votes(
            edgelets, current_model
            , threshold_inlier)

        if current_votes.sum()
        > best_votes.sum():
            best_model = current_model
            best_votes = current_votes
            logging.info("Current
            best model has {} votes at
            iteration {}".format(
            current_votes.sum()
            , ransac_iter))

    return best_model

def ransac_3_line(edgelets
, focal_length, num_ransac_iter=2000
,threshold_inlier=5):
    """Estimate orthogonal vanishing
     points using 3 line
     Ransac algorithm."""
    locations, directions, strengths
     = edgelets
    lines = edgelet_lines(edgelets)

    num_pts = strengths.size

    arg_sort = np.argsort(-strengths)
    first_index_space = arg_sort[
    :num_pts // 5]
    second_index_space = arg_sort[
    :num_pts // 5]
    third_index_space = arg_sort[
    :num_pts // 2]

    best_model = (None, None)
    best_votes = 0

    for ransac_iter in range(
    num_ransac_iter):
        ind1 = np.random.choice(
        first_index_space)
        ind2 = np.random.choice(
        second_index_space)
        ind3 = np.random.choice(
        third_index_space)

        l1 = lines[ind1]
        l2 = lines[ind2]
        l3 = lines[ind3]

        vp1 = np.cross(l1, l2)
        # The vanishing line polar to v1
        h = np.dot(vp1, [1
        / focal_length**2, 1 /
         focal_length**2, 1])
        vp2 = np.cross(h, l3)

        if np.sum(vp1**2) < 1 or vp1[2] == 0:
            # reject degenerate candidates
            continue

        if np.sum(vp2**2) < 1 or vp2[2] == 0:
            # reject degenerate candidates
            continue

        vp1_votes = compute_votes(
```

```
        edgelets, vp1, threshold_inlier)
        vp2_votes = compute_votes(
        edgelets, vp2, threshold_inlier)
        current_votes = (vp1_votes
        > 0).sum() + (vp2_votes > 0).sum()

        if current_votes > best_votes:
            best_model = (vp1, vp2)
            best_votes = current_votes
            logging.info("Current
            best model has {} votes
             at iteration {}".format(
                current_votes, ransac_iter))

    return best_model

def reestimate_model(model,
 edgelets, threshold_reestimate=5):
    """Reestimate vanishing point
     using inliers and least squares."""
    locations, directions, strengths=
     edgelets

    inliers = compute_votes(edgelets
    , model, threshold_reestimate) > 0
    locations = locations[inliers]
    directions = directions[inliers]
    strengths = strengths[inliers]

    lines = edgelet_lines((locations
    , directions, strengths))

    a = lines[:, :2]
    b = -lines[:, 2]
    est_model = np.linalg.lstsq(
    a, b)[0]
    return np.concatenate((
    est_model, [1.]))

def remove_inliers(model
, edgelets, threshold_inlier=10):
    """Remove all inlier edglets
     of a given model."""
    inliers = compute_votes(edgelets
    , model, 10) > 0
    locations, directions, strengths=
     edgelets
    locations = locations[~inliers]
    directions = directions[~inliers]
    strengths = strengths[~inliers]
    edgelets = (locations, directions
    , strengths)
    return edgelets

def compute_homography_and_warp(image
, vp1, vp2, clip=True, clip_factor=3):
    """Compute homography from
    vanishing points and warp
    the image."""
    # Find Projective Transform
    vanishing_line = np.cross(vp1, vp2)
    H = np.eye(3)
    H[2] = vanishing_line / vanishing_line[2]
    H = H / H[2, 2]

    # Find directions corresponding
    # to vanishing points
    v_post1 = np.dot(H, vp1)
    v_post2 = np.dot(H, vp2)
    v_post1 = v_post1 / np.sqrt(
    v_post1[0]**2 + v_post1[1]**2)
    v_post2 = v_post2 / np.sqrt(
    v_post2[0]**2 + v_post2[1]**2)

    directions = np.array([[
    v_post1[0], -v_post1[0],
     v_post2[0], -v_post2[0]],
    [v_post1[1], -v_post1[1],
     v_post2[1], -v_post2[1]]])

    thetas = np.arctan2(directions[0]
    , directions[1])

    # Find direction closest
    #to horizontal axis
    h_ind = np.argmin(np.abs(thetas))

    # Find positve angle among
    # the rest for the vertical axis
    if h_ind // 2 == 0:
        v_ind = 2 + np.argmax(
        [thetas[2], thetas[3]])
    else:
        v_ind = np.argmax(
        [thetas[2], thetas[3]])

    A1 = np.array([[directions[
    0, v_ind], directions[0, h_ind], 0],
    [directions[1, v_ind],
    directions[1, h_ind], 0],
    [0, 0, 1]])
    # Might be a reflection.
    # If so, remove reflection.
    if np.linalg.det(A1) < 0:
        A1[:, 0] = -A1[:, 0]

    A = np.linalg.inv(A1)

    # Translate so that whole
    # of the image is covered
    inter_matrix = np.dot(A, H)

    cords = np.dot(inter_matrix
    , [[0, 0, image.shape[1],
     image.shape[1]],[0, image.shape[0]
     , 0, image.shape[0]],[1, 1, 1, 1]])
```

```
    cords = cords[:2] / cords[2]

    tx = min(0, cords[0].min())
    ty = min(0, cords[1].min())

    max_x = cords[0].max() - tx
    max_y = cords[1].max() - ty

    if clip:
        # These might be too large. Clip them.
        max_offset = max(image.shape)
         * clip_factor / 2
        tx = max(tx, -max_offset)
        ty = max(ty, -max_offset)

        max_x = min(max_x, -tx
        + max_offset)
        max_y = min(max_y, -ty
        + max_offset)

    max_x = int(max_x)
    max_y = int(max_y)

    T = np.array([[1, 0, -tx],
                  [0, 1, -ty],
                  [0, 0, 1]])

    final_homography = np.dot(T,
     inter_matrix)

    warped_img = transform.warp(
    image, np.linalg.inv(final_homography),
    output_shape=(max_y, max_x))
    return warped_img

def vis_edgelets(image,
edgelets, show=True):
    """Helper function to
    visualize edgelets."""
    import matplotlib.pyplot as plt
    plt.figure(figsize=(10, 10))
    plt.imshow(image)
    locations, directions, strengths
     = edgelets
    for i in range(locations.shape[0]):
        xax = [locations[i, 0]
        - directions[i, 0] * strengths[i] / 2,
        locations[i, 0]+ directions[
        i, 0] * strengths[i] / 2]
        yax = [locations[i, 1]
        - directions[i, 1] * strengths[i] / 2,
        locations[i, 1] + directions[i,
         1] * strengths[i] / 2]

        plt.plot(xax, yax, 'r-')

    if show:
        plt.show()


def vis_model(image,
 model_1,model_2, show=True):
    """Helper function to
    visualize computed model."""
    import matplotlib.pyplot as plt
    edgelets = compute_edgelets(image)
    locations, directions, strengths =
     edgelets
    inliers1 = compute_votes(edgelets
    , model_1, 10) > 0
    inliers2 = compute_votes(edgelets
    , model_2, 10) > 0

    edgelets1 = (locations[inliers1]
    , directions[inliers1], strengths[inliers1])
    edgelets2 = (locations[inliers2]
    , directions[inliers2], strengths[inliers2])
    locations1, directions1,
     strengths1 = edgelets1
    locations2, directions2,
     strengths2 = edgelets2
    vis_edgelets(image, edgelets1, False)
    vis_edgelets(image, edgelets2, False)
    vp1 = model_1 / model_1[2]
    vp2 = model_2 / model_2[2]
    plt.plot(vp1[0], vp1[1], 'bo')
    plt.plot(vp2[0], vp2[1], 'bo')
    plt.plot([vp1[0],vp2[0]],
    [vp1[1],vp2[1]],'c-')
    for i in range(locations1.shape[0]):
        xax = [locations1[i, 0], vp1[0]]
        yax = [locations1[i, 1], vp1[1]]
        plt.plot(xax, yax, 'b-.')
    for j in range(locations2.shape[0]):
        xax = [locations2[j, 0], vp2[0]]
        yax = [locations2[j, 1], vp2[1]]
        plt.plot(xax, yax, 'r-.')

    if show:
        plt.show()

def rectify_image(image,
 clip_factor=6, algorithm=
 'independent', reestimate=False):
    """Rectified image with
    vanishing point computed
    using ransac."""
    if type(image) is not np.ndarray:
        image = io.imread(image)

    # Compute all edgelets.
    edgelets1 = compute_edgelets(image)

    if algorithm == 'independent':
        # Find first vanishing point
        vp1 =
        ransac_vanishing_point(
```

```
        edgelets1, 2000,
         threshold_inlier=5)
        if reestimate:
            vp1 =
            reestimate_model(
            vp1, edgelets1, 5)

        # Remove inlier to
        remove dominating direction.
        edgelets2 =
        remove_inliers(
        vp1, edgelets1, 10)

        # Find second vanishing point
        vp2 =
        ransac_vanishing_point(
        edgelets2, 2000,
        threshold_inlier=5)
        if reestimate:
            vp2 =
            reestimate_model(
            vp2, edgelets2, 5)
    elif algorithm == '3-line':
        focal_length = None
        vp1, vp2 = ransac_3_line(
        edgelets1, focal_length,
         num_ransac_iter=3000,
         threshold_inlier=5)
    else:
        raise KeyError(
            "Parameter 'algorithm'
             has to be one of
             {'3-line', 'independent'}")

    # Compute the homography and warp
    warped_img =
    compute_homography_and_warp(
    image, vp1, vp2,
    clip_factor=clip_factor)

    return warped_img

if __name__ == '__main__':

    image_name =r'D:\IC-Msc\
    Computer Vision\Original Picture
    \FD\WithObject\DSCF0393.JPG'
    image = io.imread(image_name)
    print("Rectifying {}".format(
    image_name))

    edgelets1 = compute_edgelets(image)
    vis_edgelets(image, edgelets1)
    # Visualize the edgelets
    vp1 = ransac_vanishing_point(
    edgelets1, num_ransac_iter=2000
    ,threshold_inlier=5)
    vp1 = reestimate_model(
    vp1, edgelets1,
    threshold_reestimate=5)

    edgelets2 = remove_inliers(
    vp1, edgelets1, 10)
    vp2 = ransac_vanishing_point(
    edgelets2, num_ransac_iter=2000
    ,threshold_inlier=5)
    vp2 = reestimate_model(
    vp2, edgelets2,
    threshold_reestimate=5)
    print([vp1], [vp2])
    vis_model(image, vp1,vp2)
    # Visualize the vanishing point model
```

Rectified pair of images with epipolar lines python code:

```
import numpy as np
import cv2 as cv
from matplotlib import pyplot as plt

def drawlines(img1,img2,lines,pts1,pts2):
    ''' img1 - image on which we draw
     the epilines for the points in img2
        lines - corresponding epilines '''
    r,c = img1.shape
    img1 =
    cv.cvtColor(img1,cv.COLOR_GRAY2BGR)
    img2 =
    cv.cvtColor(img2,cv.COLOR_GRAY2BGR)
    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,3)
        img1 = cv.circle(img1,tuple(pt1)
        ,40,color,-1)
        img2 = cv.circle(img2,tuple(pt2)
        ,40,color,-1)
    return img1,img2

img1 = cv.imread(r'D:\IC-Msc\Computer Vision
\Original Picture\FD\WithObject\DSCF0401.JPG'
,cv.IMREAD_GRAYSCALE)
#queryimage # left image
img2 = cv.imread(r'D:\IC-Msc\Computer Vision
\Original Picture\FD\WithObject\DSCF0397.JPG'
,cv.IMREAD_GRAYSCALE)
#trainimage # right image
sift = cv.SIFT_create()
# find the keypoints and descriptors with SIFT
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 < 0.45*n.distance:
        pts2.append(kp2[m.trainIdx].pt)
        pts1.append(kp1[m.queryIdx].pt)

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

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]

h1, w1 = img1.shape
h2, w2 = img2.shape
thresh = 0.5
_, H1, H2 = cv.stereoRectifyUncalibrated
(
    np.float32(pts1), np.float32(pts2)
    , F, imgSize=(w1, h1), threshold=thresh,
)

############### Undistort (Rectify) #
imgL_undistorted = cv.warpPerspective
(img1, H1, (w1, h1))
imgR_undistorted = cv.warpPerspective
(img2, H2, (w2, h2))

sift1 = cv.SIFT_create()
# find the keypoints and descriptors
 with SIFT
kp1, des1 = sift1.detectAndCompute
(imgL_undistorted,None)
kp2, des2 = sift1.detectAndCompute
(imgR_undistorted,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 < 0.45*n.distance:
        pts2.append(kp2[m.trainIdx].pt)
        pts1.append(kp1[m.queryIdx].pt)

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

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]

lines1 = cv.computeCorrespondEpilines(
pts2.reshape(-1,1,2), 2,F)
lines1 = lines1.reshape(-1,3)
img5, img6 = drawlines(imgL_undistorted
,imgR_undistorted,lines1,pts1,pts2)

lines2 = cv.computeCorrespondEpilines(
pts1.reshape(-1,1,2), 1,F)
lines2 = lines2.reshape(-1,3)
img3, img4 = drawlines(imgR_undistorted,
imgL_undistorted,lines2,pts2,pts1)
plt.figure('rectification')
plt.subplot(121),plt.imshow(img5)
plt.subplot(122),plt.imshow(img3)
plt.show()
```

Depth map python code:

```
import cv2 as cv
from matplotlib import pyplot as plt
imgL = cv.imread(r'D:\IC-Msc
\Computer Vision\Original Picture\FD
\WithObject\DSCF0393.JPG',0)
imgR = cv.imread(r'D:\IC-Msc
\Computer Vision\Original Picture\FD
\WithObject\DSCF0404.JPG',0)
stereo = cv.StereoBM_create(
numDisparities=16, blockSize=5)
disparity = stereo.compute(imgL,imgR)
plt.imshow(disparity,'gray')
plt.colorbar()
plt.show()

win_size = 2
min_disp = -4
max_disp = 9
num_disp = max_disp - min_disp
# Needs to be divisible by 16
stereo = cv.StereoSGBM_create(
    minDisparity=min_disp,
    numDisparities=num_disp,
    blockSize=5,
```

```
    uniquenessRatio=5,
    speckleWindowSize=5,
    speckleRange=5,
    disp12MaxDiff=2,
    P1=8 * 3 * win_size ** 2,
    P2=32 * 3 * win_size ** 2,
)
disparity_SGBM = stereo.compute(imgL, imgR)
plt.imshow(disparity_SGBM, "gray")
plt.colorbar()
plt.show()
```