HOMEWORK 9

Mythra Varun Balakuntala SM

November 22, 2018

1 Image reconstruction using epipolar geometry

The goal of this homework is to perform 3D reconstruction of the object using two images taken from different viewpoints. This is done using principles of epipolar geometry and the concept of Fundamental matrix.

Fundamental matrix provides the relation between the corresponding image pixels of a particular object point in 3D. First we estimate the fundamental matrix using correspondences between the objects, then we refine the projection matrices in canonical form. Finally the images are rectified to make the epipoles go to infinity and to make the image planes parallel. From this we can then use the corresponding interest points and reconstruct the object in 3D. Since the reconstruction is done using uncalibrated cameras using images with projective distortions the resulting 3D reconstruction of the object will be protectively distorted from actual object. If prior knowledge of object or scene is known we can remove the projective, affine and similarity distortions using these correspondences.

1.1 Computing the Fundamental matrix

The fundamental matrix is a 3×3 matrix that captures the most fundamental relation between the pixels in the two image planes ($\mathbf{x}$ and $\mathbf{x}'$) corresponding to a given world point $\mathbf{X}$. The relationship is given by,

$$\mathbf{x'^T F x} \quad = \quad 0 \tag{1}$$

The fundamental matrix is obtained from the image correspondences using at-least 8 corresponding points. Around 40 corresponding points are needed if the points are obtained using interest point detectors like SIFT/SURF, but 8 will suffice if the correspondences are carefully selected manually. The procedure to find the Fundamental matrix is as follows,

- Obtain 8 corresponding pixels in both the images manually. These points should be carefully selected to avoid any false correspondences.
- Using one pair of correspondences gives one equation in terms of the unknown parameters of $\mathbf{F}$.
- One pair of correspondences gives one equation in terms of the unknown parameters of $\mathbf{F}$ using the Eqn. (1). We rewrite the equation $\mathbf{x'^T F x}$ into the form $\mathbf{Af = 0}$ where elements of $\mathbf{f}$ are the unknown parameters of $\mathbf{F}$, i.e. $\mathbf{f} = (\mathbf{f_{11}}, \mathbf{f_{12}}, \mathbf{f_{13}}, \mathbf{f_{21}}, \mathbf{f_{22}}, \mathbf{f_{23}}, \mathbf{f_{31}}, \mathbf{f_{32}}, \mathbf{f_{33}})$ and $\mathbf{F_{ij}} = \mathbf{f_{ij}}$.
  The resulting matrix $\mathbf{A}$ is 1 by 9 matrix of the form,

$$\mathbf{A} = (\ \mathbf{x'x} \quad \mathbf{x'y} \quad \mathbf{x'} \quad \mathbf{y'x} \quad \mathbf{y'y} \quad \mathbf{y'} \quad \mathbf{x} \quad \mathbf{y} \quad 1\ )$$

- Then we stack 8 such equations with 8 correspondences and solve the equation $\mathbf{Af = 0}$ subject to $||\mathbf{f}||_\mathbf{2} = \mathbf{1}$. This is solved using SVD, and the solution is the eigenvector corresponding to least singular value.
- Then the resulting fundamental matrix is conditioned based on the constraint that $\mathbf{F}$ is of rank 2. This is done using SVD of $\mathbf{F}$. Obtain $\mathbf{UDV^T} = \mathbf{F}$. Then set the least singular value in $\mathbf{D}$ to 0. Let the new matrix be denoted as $\mathbf{D'}$. The conditioned $\mathbf{F}$ is given by $\mathbf{F} = \mathbf{UD'V^T}$.
- Since the raw points do not result in a good estimate for $\mathbf{F}$ the points are first normalized using transformation matrices $\mathbf{T}$ and $\mathbf{T'}$ such that the pixel correspondences have a 0 mean and are at a distance of $\sqrt{2}$ from the center (0,0). If $\tilde{\mathbf{x}}$ and $\tilde{\mathbf{x}}'$ are the original points the modified points are given by,

$$\mathbf{x} = \mathbf{T}\tilde{\mathbf{x}} \quad \text{and} \quad \mathbf{x'} = \mathbf{T'}\tilde{\mathbf{x}}'$$

- Now that the $\mathbf{F}$ is computed using the normalized coordinate pairs it has to be denormalized to result in the required Fundamental matrx $\tilde{F}$ given by,

$$\tilde{\mathbf{F}} = \mathbf{T'^T F T}$$

- Next the epipoles are computed from the fundamental matrix by using the fact that the left epipole $\mathbf{e'}$ is the right null vector and right eipople $\mathbf{e'}$ is the left null vector of $\tilde{\mathbf{F}}$, i.e.

$$\tilde{\mathbf{F}}\mathbf{e} = \mathbf{0} \quad \text{and} \quad \mathbf{e'^T}\tilde{\mathbf{F}} = \mathbf{0}$$

- Finally camera projection matrix is computed by assuming they are in canonical form using the following relation,

$$\begin{aligned} \mathbf{P} &= [\,\mathbf{I} \mid \mathbf{0}\,] \\ \mathbf{P'} &= [\,[\mathbf{e'}]_\mathbf{X}\mathbf{F} \mid \mathbf{e'}\,] \end{aligned}$$

  where $[\mathbf{e'}]_\mathbf{X}$ is the skew symmetric matrix which is associated with the cross product with $\mathbf{e'}$

- Note: Ensure the world points corresponding to the image point pairs are not coplanar. This results in an unreliable $\mathbf{F}$.

1.2 3D Reconstruction

Now that we have initial estimates for the camera projection matrices and the fundamental matrix we can reconstruct the object in the scene as follows. Let the matrix $\mathbf{P'} = [\,[\mathbf{e'}]_\mathbf{X}\mathbf{F} \mid \mathbf{e'}\,] = [\,\mathbf{M} \mid \mathbf{e'}\,]$. We estimate the corresponding world point $\mathbf{X}$ for a pair of image points $\tilde{\mathbf{x}}$ and $\tilde{\mathbf{x}}'$ as follows,

- Estimate $\mathbf{X}$ using triangulation method. Let $\mathbf{P} = \begin{pmatrix} \mathbf{P_1^T} & \mathbf{P_2^T} & \mathbf{P_3^T} \end{pmatrix}^\mathbf{T}$ and $\mathbf{P'} = \begin{pmatrix} \mathbf{P_1'^T} & \mathbf{P_2'^T} & \mathbf{P_3'^T} \end{pmatrix}^\mathbf{T}$
- Then we have $\tilde{\mathbf{x}} = \mathbf{P}\mathbf{X}$ and $\tilde{\mathbf{x}}' = \mathbf{P'}\mathbf{X}$, which can be rewritten as

$$\begin{aligned} x_i(\mathbf{P_3^T X}) - (\mathbf{P_1^T X}) &= 0 \\ y_i(\mathbf{P_3^T X}) - (\mathbf{P_2^T X}) &= 0 \\ x_i'(\mathbf{P_3'^T X}) - (\mathbf{P_1'^T X}) &= 0 \\ y_i'(\mathbf{P_3'^T X}) - (\mathbf{P_2'^T X}) &= 0 \end{aligned}$$

  This gives us four equations of the form $\mathbf{AX} = \mathbf{0}$, which can be solved using linear least squares method for solving $\min_\mathbf{X} ||\mathbf{AX}||_\mathbf{2}$ subject to $||\mathbf{X}||_\mathbf{2} = \mathbf{1}$.

- Then we can refine the fundamental matrix and the estimated world coordinates using a nonlinear least squares algorithm like LM. This is done by minimizing a geometric cost function defined as

$$\sum_i d(\tilde{\mathbf{x}}_\mathbf{i}, \hat{\mathbf{x}}_\mathbf{i})^\mathbf{2} + \mathbf{d}(\tilde{\mathbf{x}}_\mathbf{i}', \hat{\mathbf{x}}_\mathbf{i}')$$

  . We see that there are $3n + 12$ optimization variables i.e. $3n$ from $\mathbf{X_i}$ and 12 from $\mathbf{P'}$. We do not refine $\mathbf{P}$ as it is fixed to be $\mathbf{P} = [\,\mathbf{I} \mid \mathbf{0}\,]$

- From the results we update $\mathbf{F}, \mathbf{e'}, \mathbf{e}, \mathbf{P'}$ and $\mathbf{X}$,

1.3 Image Rectification

To obtain an accurate 3D reconstruction we need a good quality fundamental matrix estimate, which needs a large number of image correspondence pixel pairs. To make the problem of searching for pixel correspondence easier we can transform the images such that the corresponding pixels lie in the same row. This results in the search being computationally efficient. This is done by transforming the image planes such that the epipoles are sent to infinity. The steps to achieve this are as follows,

- The second image is first translated such that the origin is at the center of the image. Let $\mathbf{T'_1}$ represent the transformation matrix for the translation.
- Now the image is rotated such that the epipole falls on the x-axis of the rotated frame. i.e.

$$\tilde{\mathbf{e}'} = \begin{pmatrix} f \\ 0 \\ 1 \end{pmatrix}$$

- Then the resulting epipole is transformed such that it goes to infinity, i.e.

$$\bar{\mathbf{e}}' = \begin{pmatrix} f \\ 0 \\ 0 \end{pmatrix}$$

  This is done by using the homography, $\mathbf{G}$ where

$$\mathbf{G} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -1/f & 0 & 1 \end{pmatrix}$$

- Finally the image center is translated back to its original position. Let this transformation be represented as $\mathbf{T'_2}$. The overall homography to perform all these transformations is given by,

$$\mathbf{H'} = \mathbf{T'_2 G R T'_1} \tag{2}$$

1.3.1 Sending e to infinity

Now we need to transform the first image to ensure the corresponding image pairs are in the same row. This is done by minimizing the distance between projection of $\tilde{\mathbf{x}}$ on right image plane and the corresponding image pixel $\tilde{\mathbf{x}'}$, i.e.

$$\min_H \sum_i d(\mathbf{H}\tilde{\mathbf{x}}_\mathbf{i}, \mathbf{H'}\tilde{\mathbf{x}}'_\mathbf{i}) \tag{3}$$

- Let the matrxix $\mathbf{P'}$ be represented as $\mathbf{P'} = [\ [\mathbf{e'}]_\mathbf{X}\mathbf{F} \mid \mathbf{e'}\ ] = [\ \mathbf{M} \mid \mathbf{e'}\ ]$. In Eqn. (3) the matrix $\mathbf{H}$ is given by $\mathbf{H} = \mathbf{H_a H_0}$ where

$$\mathbf{H_a} = \begin{pmatrix} a & b & c \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \quad \text{and} \quad \mathbf{H_0} = \mathbf{H'M}$$

- This reduces Eqn. (3) to,

$$\min_\mathbf{H} \sum_i d(\mathbf{H}\tilde{\mathbf{x}}_i, \mathbf{H'}\tilde{\mathbf{x}}'_i) \equiv \min_{a,b,c} \sum_i (a\hat{x}_i + b\hat{y}_i + c - \hat{x}'_i)$$

  where $\hat{\mathbf{x}}_\mathbf{i} = \mathbf{H_0}\tilde{\mathbf{x}}_\mathbf{i}$ and $\hat{\mathbf{x}}'_\mathbf{i} = \mathbf{H'}\tilde{\mathbf{x}}'_\mathbf{i}$

2 Results

The fundamental matrix after conditioning is found to be

$$\mathbf{F} = \begin{pmatrix} -3.89430306e-07 & -2.78955054e-07 & 8.45299029e-04 \\ 1.07058159e-06 & 4.53998624e-08 & -2.37419523e-03 \\ -3.83539350e-04 & 2.12349585e-03 & 1.00000000e+00 \end{pmatrix}$$

. The left and right epipoles are found to be,

$$\mathbf{e} = \begin{pmatrix} 2.22063026e+03 \\ -6.98380989e+01 \\ 1.00000000e+00 \end{pmatrix} \quad \text{and} \quad \mathbf{e}' = \begin{pmatrix} 8.15331391e+03 \\ 3.32406882e+03 \\ 1.00000000e+00 \end{pmatrix}$$

. The initial camera projection matrices are estimated to be,

$$\mathbf{P} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

$$\mathbf{P}' = \begin{pmatrix} -1.27491227e+00 & 7.05864631e+00 & 3.32407119e+03 & 8.15331391e+03 \\ 3.12711633e+00 & -1.73135286e+01 & -8.15331306e+03 & 3.32406882e+03 \\ 1.00232809e-02 & 1.29742513e-03 & -2.21673911e+01 & 1.00000000e+00 \end{pmatrix}$$

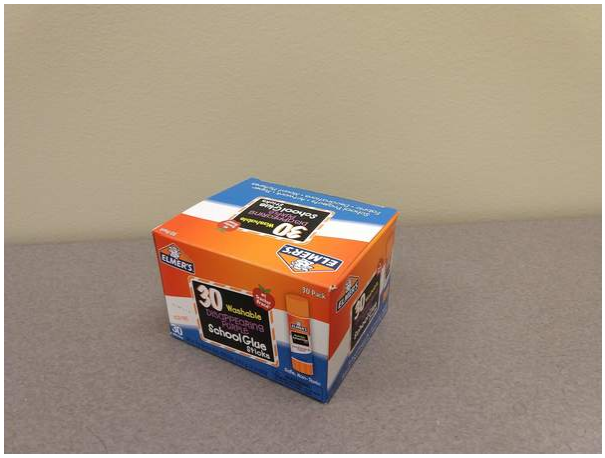

Figure 1: Original left image

Figure 2: Original right image

Figure 3: Left image selected points

Figure 4: Right image selected points

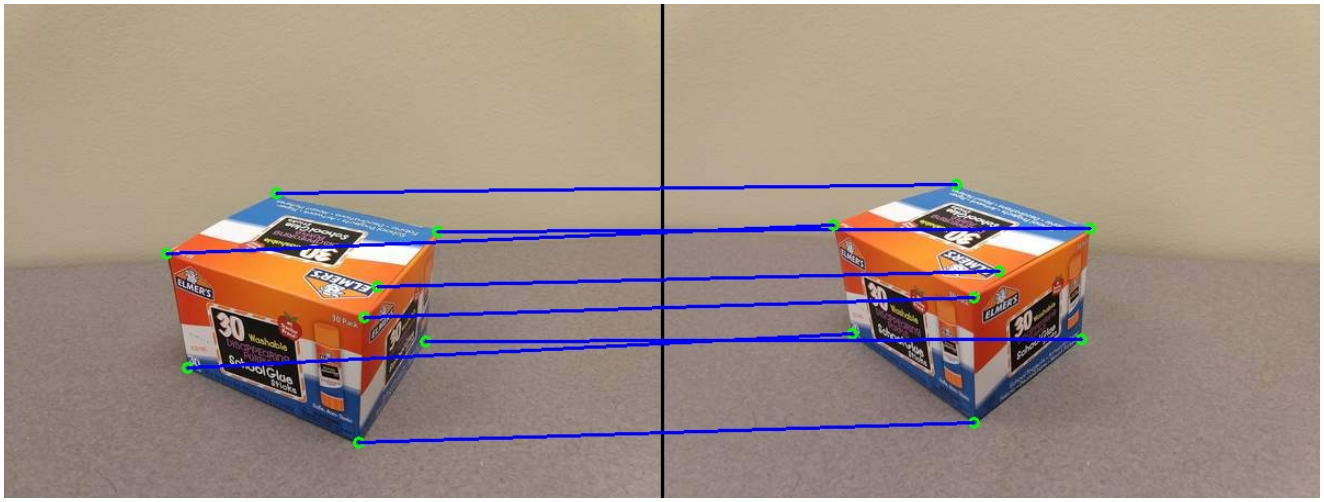

Figure 5: Manually selected correspondences

2.1 Codes

```
#!/usr/bin/env python3
################################################################################
# Script for homework 8 ECE661
# Camera callibration file
#  - Contains functions to detect corner points of chekered board pattern
#       Uses opencv canny and hough to find corners and LM to refine them.
#  - Functions to perform camera callibration using Zhang's algorithm
#       Uses corner poitns to estimate intrinsic, then extrinsic and
#       finally LM to refine parameters
# Outputs : Camera intrinsic paramters, extrinsic paramters (w.r.t specified
#   image), distortion paramters.
# Author: Mythra Balakuntala
################################################################################
import numpy as np, cv2
from scipy.optimize import least_squares as ls
from scipy import linalg as slg
# from scipy.optimize import leastsq as lsq
class ImageReconstruct():

    def __init__(self):
        self.kill = False
        self.c = [list(),list()]
        self.img = [np.array([]), np.array([])]
# Function to view image
    def viewimg(self, img, scale = 600):
        if not self.kill:
            cv2.namedWindow('resultImage',cv2.WINDOW_NORMAL)
            cv2.imshow('resultImage',img)
            cv2.resizeWindow('resultImage', scale, scale)
            key = 1
            print('Press q to quit')
            while key !=ord('q'):       # Quit on pressing q
                key = cv2.waitKey(0)
                if key == ord('s'):
                    self.kill = True
            cv2.destroyAllWindows()

    def mouse_cb(self, event, x, y, flags, img_id):
        if event ==cv2.EVENT_LBUTTONDOWN:
            self.c[img_id].append([x,y])
            cv2.circle(self.img[img_id], (x,y), 1, (255,0,0),2)
            cv2.imshow('Image',self.img[img_id])

# Function to draw draw matches
    def draw_matches(self, img1, img2, kp1, kp2):
        r1, c1 = img1.shape[0:2]
        r2, c2 = img2.shape[0:2]
        # output image
        res_img = np.zeros((max([r1, r2]), c1 + c2, 3))
        # Add first image
        res_img[0:r1, 0:c1, :] = img1
        res_img[:r2, c1:, :] = img2 # Add second image
        # Line and ponit properties
        rad = 4 # For point
        RED = (0,0,255)
```

```
        BLUE = (255,0,0)
        GREEN = (0,255,0)
        thickness = 2
        # Draw partition line
        cv2.line(res_img, (c1,0),(c1,max([r1,r2])),(0,0,0),thickness)
        # Get matches and draw lines
        for i in range(kp1.shape[0]):
            # if i in inliers[0]:
            # Draw circle for points
            cv2.circle(res_img,(kp1[i,0],kp1[i,1]),rad,GREEN,thickness) # img1 pt
            cv2.circle(res_img,(kp2[i,0]+c1,kp2[i,1]),rad,GREEN,thickness) # img2 pt
            # Draw line between the points
            cv2.line(res_img, (kp1[i,0],kp1[i,1]),(kp2[i,0]+c1,kp2[i,1]),BLUE,thickness)
            # else:
            #     # Draw circle for points
            #     cv2.circle(res_img,(kp1[i,1],kp1[i,0]),rad,RED,thickness) # img1 pt
            #     cv2.circle(res_img,(kp2[i,1]+c1,kp2[i,0]),rad,RED,thickness) # img2 pt
            #     # Draw line between the points
            #     cv2.line(res_img, (kp1[i,1],kp1[i,0]),(kp2[i,1]+c1,kp2[i,0]),RED,thickness)
        return(res_img.astype(np.uint8))

#this function will be called whenever the mouse is right-clicked
    def get_points(self, img_id):
    #right-click event value is 2
        cv2.namedWindow('Image',cv2.WINDOW_NORMAL)
        cv2.resizeWindow('Image', 1000, 1000)
        cv2.setMouseCallback('Image', self.mouse_cb, img_id)
        cv2.imshow('Image',self.img[img_id])
        key = 1
        while key !=ord('q'):       # Quit on pressing q
            key = cv2.waitKey(0)
        cv2.destroyAllWindows()
# Skew symmetric matrix from vector
    def vskew(self, e):
        return(np.array([[0,-e[2],e[1]],[e[2],0,-e[0]],[-e[1],e[0],0]]))

# function to normalize x
    def normalize(self, x):
        # Mean
        m = np.mean(x,0)
        d = x - m  # distance to mean
        mdis = np.mean(np.sum(d**2,1))
        s = np.sqrt(2)/mdis
        tmat = np.array([[s ,0, -s*m[0]],[0,s,-s*m[1]],[0,0,1]])
        e = np.ones((x.shape[0],1))
        y = np.hstack((x,e))
        return((tmat@(y.T)).T, tmat)

    def find_fundemental(self, x1, x2, t1, t2):
        n = x1.shape[0]
        # Compute the A mat
        A = np.zeros((n,9))
        for i in range(n):
            A[i,:] = [x2[i,0]*x1[i,0], x2[i,0]*x1[i,1], x2[i,0]\
            ,x2[i,1]*x1[i,0], x2[i,1]*x1[i,1], x2[i,1]\
            ,x1[i,0], x1[i,1], 1]
```

```
        u,s,vh = np.linalg.svd(A)
        f = (vh.T)[:,-1]
        fmat = f.reshape(3,3)
        # Reduce rank of F to 2
        u,s,vh = np.linalg.svd(fmat)
        s[-1] = 0   # set last singular value to 0
        fmat = u@np.diag(s)@vh
        # Denormalize
        fmat = t2.T@fmat@t1
        # Set last term to 1
        fmat = fmat/fmat[-1,-1]
        print('Rank of F is ',np.linalg.matrix_rank(fmat))
        # Find epipoles
        # Left epipose ,aka right null space
        e1 = (slg.null_space(fmat).T)[0]
        e2 = (slg.null_space(fmat.T).T)[0]
        return(fmat, e1/e1[-1], e2/e2[-1])

# Compute projection matrices from F and e'
    def get_pmats(self, f, e):
        # Compute projection matrices
        p1 = np.array([[1,0,0,0],[0,1,0,0],[0,0,0,1]])
        ex = self.vskew(e)
        p2 = np.hstack((ex@f, e[:,None]))
        return(p1,p2)

# Function to perform reconstruction
    def recon(self, p1, p2, x1, x2):
        # Compute X from x1,x2 and p1,p2 using traingulation
        n = x1.shape[0]
        # Compute the A mat
        X = np.zeros((n,4))
        for i in range(n):
            A = np.zeros((4,4))
            # From A matrix
            A[0,:] = x1[i,0]*p1[2,:] - p1[0,:]
            A[1,:] = x1[i,1]*p1[2,:] - p1[1,:]
            A[2,:] = x2[i,0]*p2[2,:] - p2[0,:]
            A[3,:] = x2[i,1]*p2[2,:] - p2[1,:]
            # Find X
            u,s,vh = np.linalg.svd(A)
            X[i,:] = (vh.T)[:,-1]
            X[i,:] = X[i,:]/X[i,-1] #Normalize
        return(X)

# Main function
    def main(self):
        self.img[0] = cv2.imread('1.jpg')
        self.img[1] = cv2.imread('2.jpg')
        # Get corresponding points manually,
        # self.get_points(0)    # array of points on left image
        # print(self.c[0])
        # cv2.imwrite('Pic_pts1.jpg',self.img[0])
        # self.get_points(1)    # array of points on right image
        # print(self.c[1])
        # cv2.imwrite('Pic_pts2.jpg',self.img[1])
```

```
        self.c[0] = [[328, 285], [395, 207], [248, 172], [148, 227], [167, 331], [323, 399], [384, 307]
        self.c[1] = [[284, 267], [390, 204], [268, 164], [156, 201], [173, 300], [284, 381], [382, 305]
        # Draw matches
        kp1 = np.array(self.c[0])
        kp2 = np.array(self.c[1])
        img = self.draw_matches(self.img[0], self.img[1], kp1, kp2)
        # self.viewimg(img)
        cv2.imwrite('Pic_spts.jpg',img)
        # Reinitialize images
        self.img[0] = cv2.imread('1.jpg')
        self.img[1] = cv2.imread('2.jpg')
        # Get normalized x and x'
        x1n,t1 = self.normalize(kp1)
        x2n,t2 = self.normalize(kp2)
        # Get fundemental matrix
        fmat, e1,e2 = self.find_fundemental(x1n,x2n,t1,t2)
        print('The fundamental matrix is ',fmat)
        print('The left epipole is ', e1)
        print('The right epipole is ', e2)
        # Compute projection matrices
        p1, p2 = self.get_pmats(fmat, e2)
        print(p2)
        # Compute X , world points
        X = self.recon(kp1,kp2,p1,p2)

if __name__ == '__main__':
    imgrec = ImageReconstruct()
    imgrec.main()
```