16-720A — Spring 2021 — Homework 4

Jen-Hung Ho
jenhungh@andrew.cmu.edu

April 11, 2021

Part 1

Q1.1

We know that the projected points of P in each plane are $\mathbf{x}_1 = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$ and $\mathbf{x}_2 = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$.

Also, from the Epipolar Geometry, we know that

$$\mathbf{x_2^T F x_1} = 0 \tag{1}$$

$$\begin{bmatrix} 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} F_{11} & F_{12} & F_{13} \\ F_{21} & F_{22} & F_{23} \\ F_{31} & F_{32} & F_{33} \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} = 0 \tag{2}$$

$$\begin{bmatrix} F_{31} & F_{32} & F_{33} \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} = 0 \tag{3}$$

$$F_{33} = 0 \tag{4}$$

We can thus prove that the F_{33} element of the Fundamental Matrix is zero.

Q1.2

We know that the second camera differs from the first by a pure translation that is parallel to the x-axis. So, we can assume the translation vector $\mathbf{t}$ and the rotation matrix $\mathbf{R}$ and compute the Essential matrix $\mathbf{E}$.

$$\mathbf{t} = \begin{bmatrix} a \\ 0 \\ 0 \end{bmatrix}, \hat{\mathbf{t}} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & -a \\ 0 & a & 0 \end{bmatrix} \tag{5}$$

$$\mathbf{R} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{6}$$

$$\mathbf{E} = \hat{\mathbf{t}}\mathbf{R} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & -a \\ 0 & a & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & -a \\ 0 & a & 0 \end{bmatrix} \tag{7}$$

Compute the epipolar lines in the two cameras:

$$\boldsymbol{x1^T}\mathbf{E}\boldsymbol{x2} = \begin{bmatrix} x_1 & y_1 & 1 \end{bmatrix} \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & -a \\ 0 & a & 0 \end{bmatrix} \begin{bmatrix} x_2 \\ y_2 \\ 1 \end{bmatrix} \tag{8}$$

$$= \begin{bmatrix} x_1 & y_1 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ -a \\ ay_2 \end{bmatrix} = 0 \tag{9}$$

$$\boldsymbol{x2^T}\mathbf{E}\boldsymbol{x1} = \begin{bmatrix} x_2 & y_2 & 1 \end{bmatrix} \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & -a \\ 0 & a & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \\ 1 \end{bmatrix} \tag{10}$$

$$= \begin{bmatrix} x_2 & y_2 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ -a \\ ay_1 \end{bmatrix} = 0 \tag{11}$$

From equation (9) and (11), we know that the equation of the first epipolar line is (-a) y_1 + (ay_2) = 0, and the equation of the first epipolar line is (-a) y_2 + (ay_1) = 0. Both of the lines are parallel to the x-axis.

Q1.3

Assume the point in the world frame $\mathbf{P}$, and the points in the camera frame at time i and j are $\mathbf{p}_i$ and $\mathbf{p}_j$. Then, we know that

$$\mathbf{p_i} = \mathbf{R_i}\mathbf{P} + \mathbf{t_i} \tag{12}$$

$$\mathbf{p_j} = \mathbf{R_j}\mathbf{P} + \mathbf{t_j} \tag{13}$$

From equation 12, we can derive $\mathbf{P}$

$$\mathbf{P} = \mathbf{R_i^{-1}}(\mathbf{p_i} - \mathbf{t_i}) \tag{14}$$

Combine equation 13 and 14, we can get

$$\mathbf{p_j} = \mathbf{R_j}\mathbf{P} + \mathbf{t_j} = \mathbf{R_j}[\mathbf{R_i^{-1}}(\mathbf{p_i} - \mathbf{t_i})] + \mathbf{t_j} \tag{15}$$

$$\mathbf{p_j} = (\mathbf{R_j}\mathbf{R_i^{-1}})\mathbf{p_i} + (-\mathbf{R_j}\mathbf{R_i^{-1}}\mathbf{t_i} + \mathbf{t_j}) = \mathbf{R_{rel}}\mathbf{p_i} + \mathbf{t_{rel}} \tag{16}$$

$$\mathbf{R_{rel}} = \mathbf{R_j}\mathbf{R_i^{-1}} \tag{17}$$

$$\mathbf{t_{rel}} = -\mathbf{R_j}\mathbf{R_i^{-1}}\mathbf{t_i} + \mathbf{t_j} \tag{18}$$

Equation 17 and 18 show the effective rotation and translation between frame at time i and frame at time j.
Based on the definition, the essential matrix $\mathbf{E}$ and fundamental matrix $\mathbf{F}$ are

$$\mathbf{E} = \mathbf{t_{rel}} \times \mathbf{R_{rel}} = \mathbf{\hat{t}_{rel}}\mathbf{R_{rel}} \tag{19}$$

$$\mathbf{F} = \mathbf{K^{-T}}\mathbf{E}\mathbf{K^{-1}} = \mathbf{K^{-T}}(\mathbf{\hat{t}_{rel}}\mathbf{R_{rel}})\mathbf{K^{-1}} \tag{20}$$

Q1.4

Assume that the object is flat, meaning that all the points of the object are of equal distance to the mirror. This means that the transformation between the object and its reflection is a pure translation. Therefore, the effective rotation matrix $\mathbf{R}$ and translation vector $\mathbf{t}$ are:

$$\mathbf{R} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \mathbf{I} \tag{21}$$

$$\mathbf{t} = \begin{bmatrix} x \\ y \\ z \end{bmatrix} \tag{22}$$

From equation 19, we can compute the Essential Matrix $\mathbf{E}$:

$$\mathbf{E} = \hat{\mathbf{t}}\mathbf{R} = \mathbf{t} \times \mathbf{R} = \begin{bmatrix} 0 & z & -y \\ -z & 0 & x \\ y & -x & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 & z & -y \\ -z & 0 & x \\ y & -x & 0 \end{bmatrix} \tag{23}$$

From equation 23, we know that the Essential Matrix is a "skew-symmetric matrix." Based on the relationship between the Essential Matrix $\mathbf{E}$ and the Fundamental Matrix $\mathbf{F}$ (equation 20), we can also prove that the Fundamental Matrix $\mathbf{F}$ is also a skew-symmetric matrix.

Part 2

Q2.1

Fundamental Matrix $\mathbf{F}$:

$$\mathbf{F} = \begin{bmatrix} 9.78833285e-10 & -1.32135929e-07 & 1.12585666e-03 \\ -5.73843315e-08 & 2.96800276e-09 & -1.17611996e-05 \\ -1.08269003e-03 & 3.04846703e-05 & -4.47032655e-03 \end{bmatrix} \tag{24}$$

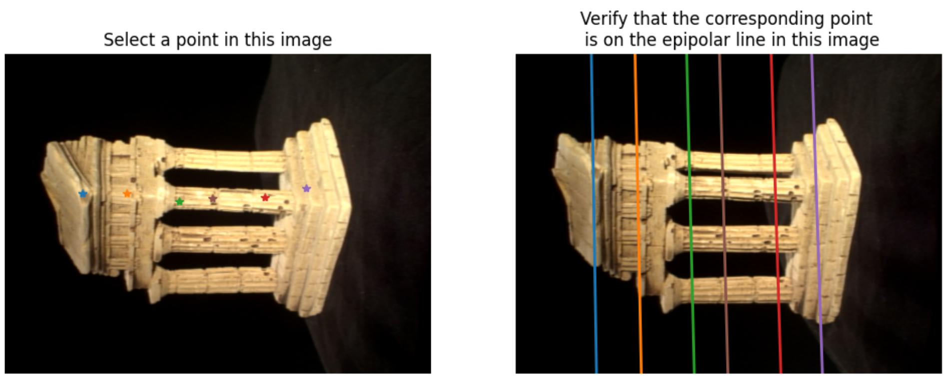

Figure 1: Visualization of the Epipolar Lines

Q3.1

Essential Matrix $\mathbf{E}$:

$$\mathbf{E} = \begin{bmatrix} 2.26268684e-03 & -3.06552495e-01 & 1.66260633e+00 \\ -1.33130407e-01 & 6.91061098e-03 & -4.33003420e-02 \\ -1.66721070e+00 & -1.33210350e-02 & -6.72186431e-04 \end{bmatrix} \tag{25}$$

Q3.2.1

Assume the 3D points $\mathbf{W}_i = \begin{bmatrix} X_i \\ Y_i \\ Z_i \end{bmatrix}$, we can use $\mathbf{C}_1$ and $\mathbf{C}_2$ to project it back to the 2D images.

$$\mathbf{p_1} = \begin{bmatrix} x_1 \\ y_1 \\ 1 \end{bmatrix} = \mathbf{C_1}\mathbf{\tilde{W}_i} = \begin{bmatrix} C_{11}^{(1)} & C_{12}^{(1)} & C_{13}^{(1)} & C_{14}^{(1)} \\ C_{21}^{(1)} & C_{22}^{(1)} & C_{23}^{(1)} & C_{24}^{(1)} \\ C_{31}^{(1)} & C_{32}^{(1)} & C_{33}^{(1)} & C_{34}^{(1)} \end{bmatrix} \begin{bmatrix} X_i \\ Y_i \\ Z_i \\ 1 \end{bmatrix} = \begin{bmatrix} -C_1^{(1)}- \\ -C_2^{(1)}- \\ -C_3^{(1)}- \end{bmatrix} \begin{bmatrix} X_i \\ Y_i \\ Z_i \\ 1 \end{bmatrix} \tag{26}$$

$$\mathbf{p_2} = \begin{bmatrix} x_2 \\ y_2 \\ 1 \end{bmatrix} = \mathbf{C_2}\mathbf{\tilde{W}_i} = \begin{bmatrix} C_{11}^{(2)} & C_{12}^{(2)} & C_{13}^{(2)} & C_{14}^{(2)} \\ C_{21}^{(2)} & C_{22}^{(2)} & C_{23}^{(2)} & C_{24}^{(2)} \\ C_{31}^{(2)} & C_{32}^{(2)} & C_{33}^{(2)} & C_{34}^{(2)} \end{bmatrix} \begin{bmatrix} X_i \\ Y_i \\ Z_i \\ 1 \end{bmatrix} = \begin{bmatrix} -C_1^{(2)}- \\ -C_2^{(2)}- \\ -C_3^{(2)}- \end{bmatrix} \begin{bmatrix} X_i \\ Y_i \\ Z_i \\ 1 \end{bmatrix} \tag{27}$$

From equation 26 and 27, we can derive that

$$\begin{bmatrix} x_1 C_3^{(1)} - C_1^{(1)} \\ y_1 C_3^{(1)} - C_2^{(1)} \end{bmatrix} \begin{bmatrix} X_i \\ Y_i \\ Z_i \\ 1 \end{bmatrix} = 0 \tag{28}$$

$$\begin{bmatrix} x_2 C_3^{(2)} - C_1^{(2)} \\ y_2 C_3^{(2)} - C_2^{(2)} \end{bmatrix} \begin{bmatrix} X_i \\ Y_i \\ Z_i \\ 1 \end{bmatrix} = 0 \tag{29}$$

Combine equation 28 and 29 to get

$$\begin{bmatrix} x_1 C_3^{(1)} - C_1^{(1)} \\ y_1 C_3^{(1)} - C_2^{(1)} \\ x_2 C_3^{(2)} - C_1^{(2)} \\ y_2 C_3^{(2)} - C_2^{(2)} \end{bmatrix} \begin{bmatrix} X_i \\ Y_i \\ Z_i \\ 1 \end{bmatrix} = \mathbf{A_i}\mathbf{\tilde{W}_i} = 0 \tag{30}$$

$$\mathbf{A_i} = \begin{bmatrix} x_1 C_3^{(1)} - C_1^{(1)} \\ y_1 C_3^{(1)} - C_2^{(1)} \\ x_2 C_3^{(2)} - C_1^{(2)} \\ y_2 C_3^{(2)} - C_2^{(2)} \end{bmatrix} \tag{31}$$

Q3.2.2

The scalar reprojection error = 352.2302235117389

Q3.3

Projective Camera Matrix $\mathbf{M}_2$:

$$\mathbf{M2} = \begin{bmatrix} 0.99942701 & 0.03331428 & 0.0059843 & -0.02601138 \\ -0.03372743 & 0.96531375 & 0.25890503 & -1. \\ 0.00284851 & -0.25895852 & 0.96588424 & 0.07981688 \end{bmatrix} \tag{32}$$

Q4.1

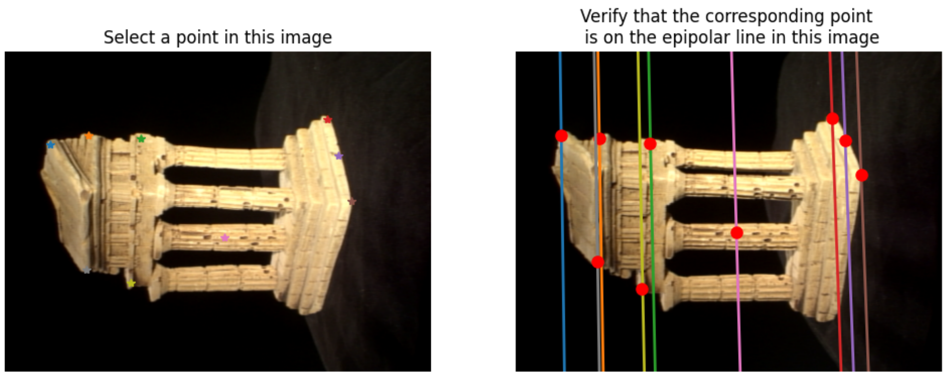

Figure 2: Visualization of the Epipolar Lines and the Corresponding Points

Q4.2

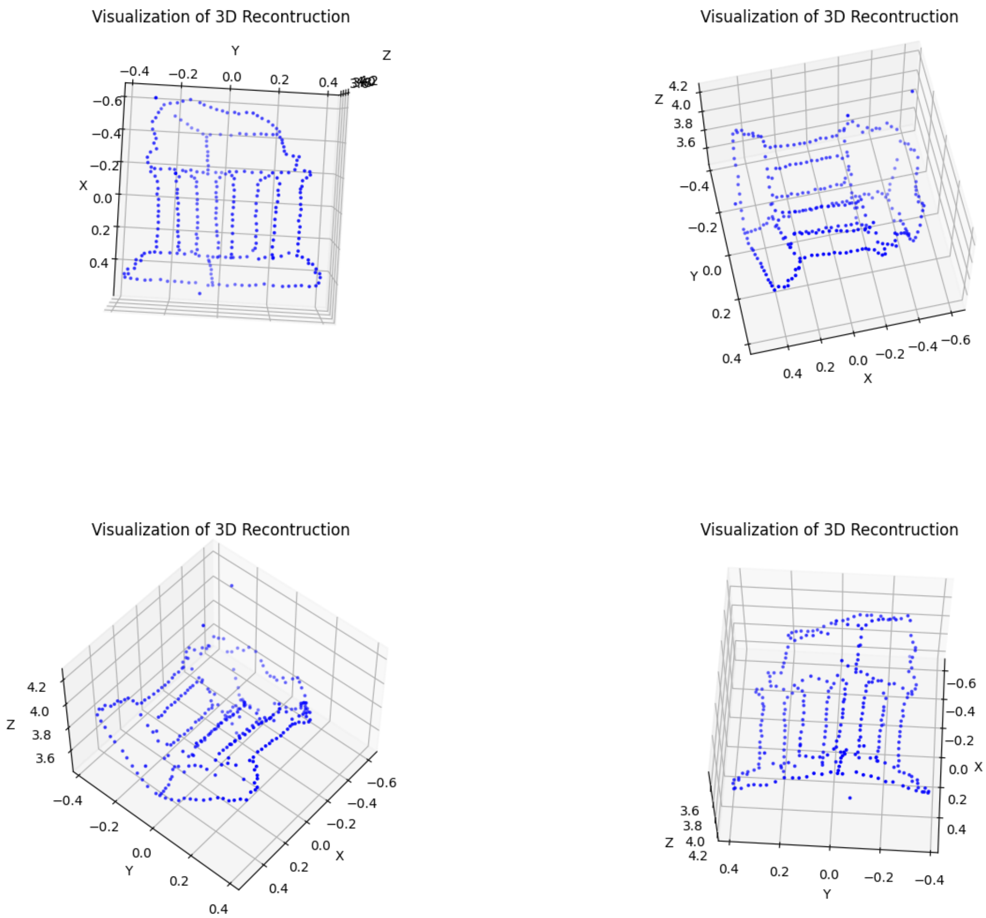

Figure 3: Visualization of the 3D Reconstruction

Q5.1

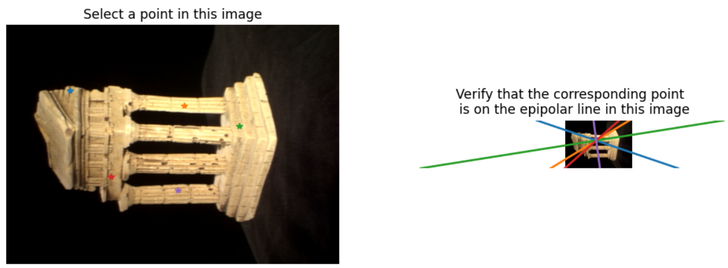

Figure 4: Visualization of the Epipolar Lines and the Corresponding Points using Eight Point Algorithm

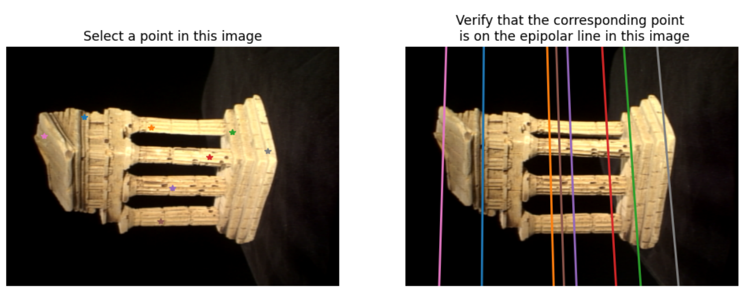

Figure 5: Visualization of the Epipolar Lines and the Corresponding Points using RANSAC

The Fundamental Matrix $\mathbf{F}$ obtained from the RANSAC algorithm:

$$\mathbf{F} = \begin{bmatrix} -8.77851454e-09 & 7.87966154e-08 & 1.06704280e-03 \\ -2.85381105e-07 & -2.94550521e-09 & 5.16841948e-05 \\ -1.01621163e-03 & -2.45719616e-05 & -6.16797059e-03 \end{bmatrix} \tag{33}$$

The error metrics I used is the euclidean distance between the epipolar line and the real point (pts2) on frame2.

$$error = \mathbf{X_2^T F X_1} \tag{34}$$

where $\mathbf{FX}_1$ solves the coefficients of the epipolar lines.

If the euclidean disatance between the epipolar line and the real point (pts2) on frame2 is smaller than the tolerance, then the points are the inliers.

Q5.3

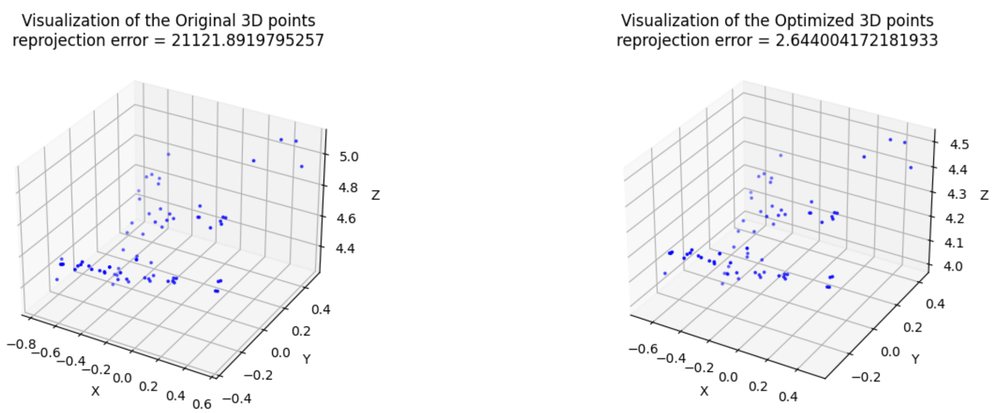

Figure 6: Visualization of the Original 3D Points and the Optimized 3D Points

Reprojection Error with initial $\mathbf{M}_2$ and $\mathbf{w}$ = 21121.8919795257
Reprojection Error with optimized matrices = 2.644004172181933

Bundle Adjustment minimize the reprojection error.

Code Appendix

0.1 submission.py

```python
"""
Homework4.
Replace 'pass' by your implementation.
"""

import numpy as np
import scipy.ndimage
import util
import scipy

'''
Q2.1: Eight Point Algorithm
    Input:  pts1, Nx2 Matrix
            pts2, Nx2 Matrix
            M, a scalar parameter computed as max (imwidth, imheight)
    Output: F, the fundamental matrix
'''
def eightpoint(pts1, pts2, M):
    # Scale the data
    pts1 = pts1 / M
    pts2 = pts2 / M

    # Form matrix A
    x1, y1 = pts1[:, 0].reshape(-1, 1), pts1[:, 1].reshape(-1, 1)
    x2, y2 = pts2[:, 0].reshape(-1, 1), pts2[:, 1].reshape(-1, 1)
    one = np.ones((x1.shape[0], 1))
    A = np.concatenate((x2*x1, x2*y1, x2, y2*x1, y2*y1, y2, x1, y1, one),
                                                                    axis = 1)

    # Singular Value Decomposition
    u, s, vh = np.linalg.svd(A)
    F = vh.T[:, -1].reshape(3, 3)

    # Refine and Singularize F
    F = util.refineF(F, pts1, pts2)

    # Unscale the Fundamental Matrix
    T = np.array([[1/M, 0, 0], [0, 1/M, 0], [0, 0, 1]])
    F = T.T @ F @ T

    return F

'''
Q3.1: Compute the essential matrix E.
```

```
    Input:  F, fundamental matrix
            K1, internal camera calibration matrix of camera 1
            K2, internal camera calibration matrix of camera 2
    Output: E, the essential matrix
'''
def essentialMatrix(F, K1, K2):
    # Form the Essential Matrix
    E = K2.T @ F @ K1
    return E

'''
Q3.2: Triangulate a set of 2D coordinates in the image to a set of 3D points.
    Input:  C1, the 3x4 camera matrix
            pts1, the Nx2 matrix with the 2D image coordinates per row
            C2, the 3x4 camera matrix
            pts2, the Nx2 matrix with the 2D image coordinates per row
    Output: P, the Nx3 matrix with the corresponding 3D points per row
            err, the reprojection error.
'''
def triangulate(C1, pts1, C2, pts2):
    # Initialize P and error
    N = pts1.shape[0]
    P = np.zeros((N, 3))
    err = 0

    # Extract 2D values
    x1, y1 = pts1[:, 0], pts1[:, 1]
    x2, y2 = pts2[:, 0], pts2[:, 1]

    # Triangulation
    for i in range(N):
        # Form matrix A
        A = np.asarray([x1[i] * C1[2, :] - C1[0, :],
                        y1[i] * C1[2, :] - C1[1, :],
                        x2[i] * C2[2, :] - C2[0, :],
                        y2[i] * C2[2, :] - C2[1, :]])

        # Singular Value Decomposition
        u, s, vh = np.linalg.svd(A)
        p = vh.T[:, -1]
        # Normalization
        p = p / p[-1]
        # Update P
        P[i, :] = p[0:3]

        # Reprojection
        p1 = C1 @ p
```

```python
        p2 = C2 @ p
        # Normailization
        p1 = p1 / p1[-1]
        p2 = p2 / p2[-1]
        # Compute and Update reprojection error
        error = np.sum((p1[0:2] - pts1[i, :])**2) +
                                                 np.sum((p2[0:2] - pts2[i, :])**2)
        err = err + error

    return P, err

'''
Q4.1: 3D visualization of the temple images.
    Input:  im1, the first image
            im2, the second image
            F, the fundamental matrix
            x1, x-coordinates of a pixel on im1
            y1, y-coordinates of a pixel on im1
    Output: x2, x-coordinates of the pixel on im2
            y2, y-coordinates of the pixel on im2
'''
def epipolarCorrespondence(im1, im2, F, x1, y1):
    # Apply Gaussian Filter to both images
    img1_filter = scipy.ndimage.gaussian_filter(im1, sigma = 1)
    img2_filter = scipy.ndimage.gaussian_filter(im2, sigma = 1)

    # Find the epipolar line on image2
    p1 = np.array([x1, y1, 1]).reshape(-1, 1)
    epi_line = F @ p1
    # Get the coefficients : ax + by + c = 0
    a, b, c = epi_line

    # Find the possible matches along the epi_line
    search = 40
    poss_y = np.arange(y1 - search, y1 + search)
    poss_x = (- c - b * poss_y) / a
    poss_x = poss_x.astype(int)
    # Check the validity (depend on window size)
    H, W, D = im2.shape
    window = 10
    half_w = window//2
    valid = (poss_x >= half_w) & (poss_x < W-half_w) & (poss_y >= half_w) &
                                                                  (poss_y < H-half_w)
    poss_x, poss_y = poss_x[valid], poss_y[valid]

    # Correspondence Matching
    error = np.inf
```

```python
    for i in range(poss_x.shape[0]):
        # Possible corresponding points on image2
        p2_x, p2_y = poss_x[i], poss_y[i]

        # Compute the window similarity
        window1 = img1_filter[y1-half_w:y1+half_w+1, x1-half_w:x1+half_w+1, :]
        window2 = img2_filter[p2_y-half_w:p2_y+half_w+1,
                                                    p2_x-half_w:p2_x+half_w+1, :]
        dis = np.sum((window1 - window2) ** 2)

        # Find the closest correspondences
        if dis < error:
            error = dis
            x2, y2 = p2_x, p2_y

    return x2, y2

'''
Q5.1: Extra Credit RANSAC method.
    Input:  pts1, Nx2 Matrix
            pts2, Nx2 Matrix
            M, a scaler parameter
    Output: F, the fundamental matrix
            inliers, Nx1 bool vector set to true for inliers
'''
def ransacF(pts1, pts2, M, nIters=1000, tol=0.42):
    # Initialize max inliers
    N = pts1.shape[0]
    max_inliers = 0

    # RANSAC Algorithm
    for iter in range(nIters):
        # Print out iteration index
        print(f"iteration = {iter+1}")

        # Initialize inliers
        current_inliers = np.zeros(N, dtype = np.bool)

        # Randomly select 8 points to compute F
        # np.random.seed()
        # sample = np.random.choice(N, size = 8, replace = False)
        sample = np.random.choice(N, size = 8)
        pts1_sample = pts1[sample, :]
        pts2_sample = pts2[sample, :]

        # Compute the Fundamental Matrix using Eight Point Algorithm
        current_F = eightpoint(pts1_sample, pts2_sample, M)

```

```
        # Compute the epipolar line
        p1_homo = np.concatenate((pts1, np.ones((N, 1))), axis = 1)
        p2_pred = (current_F @ p1_homo.T).T

        # Compute the euclidean distance between epipolar line and pts2
        p2_homo = np.concatenate((pts2, np.ones((N, 1))), axis = 1)
        factor = np.sqrt(np.sum(p2_pred[:, 0:2] ** 2, axis = 1))
        dis = abs(np.sum(p2_pred * p2_homo, axis = 1)) / factor

        # Update current inliers
        current_inliers[sample] = True
        current_inliers[dis < tol] = True
        # Compute the number of inliers
        num_inliers = np.sum(current_inliers)

        # Update F and inliers if needed
        if (num_inliers > max_inliers):
            max_inliers = num_inliers
            F = current_F
            inliers = current_inliers

        # Print max_inliers to check
        print(f"max_inliers = {max_inliers}")

    return F, inliers

'''
Q5.2:Extra Credit  Rodrigues formula.
    Input:  r, a 3x1 vector
    Output: R, a rotation matrix
'''
def rodrigues(r):
    # Compute the rotation angle theta
    theta = np.sqrt(np.sum(r ** 2))

    # Deal with corner case : no rotation
    if theta == 0:
        k = r
    else:
        k = r / theta

    # Compute the cross-product matrix K
    k1, k2, k3 = k[:, 0]
    K = np.array([[0, -k3, k2],
                  [k3, 0, -k1],
                  [-k2, k1, 0]])

```

```
    # Apply Rodrigues Rotation Formula
    # R = I + sin(theta) * K + (1-cos(theta)) * K^2
    R = np.eye(3) + np.sin(theta) * K + (1 - np.cos(theta)) * (K @ K)

    return R

'''
Q5.2:Extra Credit  Inverse Rodrigues formula.
    Input:  R, a rotation matrix
    Output: r, a 3x1 vector
'''
def invRodrigues(R):
    '''
    Reference:
    https://www2.cs.duke.edu/courses/compsci527/fall13/notes/rodrigues.pdf
    '''
    # Define A, rho, s, and c
    A = (R - R.T) /2
    rho = np.array([A[2, 1], A[0, 2], A[1, 0]]).reshape(-1, 1)
    s = np.sqrt(np.sum(rho ** 2))
    c = (np.sum(np.diag(R)) - 1) / 2

    # s = 0 and c = 1
    if s == 0 and c == 1:
        r = np.zeros((3, 1))

    # s = 0 and c = -1
    elif s ==0 and c == -1:
        # Compute v
        R_plus_I = R + np.eye(3)
        for col in range(3):
            if np.sum(R_plus_I[:, col]) != 0:
                v = R_plus_I[:, col]
                break
        # Compute u and r
        u = v / np.sqrt(np.sum(v ** 2))
        r = u * np.pi

        # Distinguish r or -r
        r1, r2, r3 = r[:, 0]
        if np.sqrt(np.sum(r ** 2)) == np.pi
                                       and ((r1 == 0 and r2 == 0 and r3 < 0)
                                       or (r1 == 0 and r2 < 0) or (r1 < 0)):
            r = -r
        else:
            r = r
```

```
    # remaining cases
    else:
        u = rho / s
        theta = np.arctan2(s, c)
        r = u * theta

    return r

'''
Q5.3: Extra Credit Rodrigues residual.
    Input:  K1, the intrinsics of camera 1
            M1, the extrinsics of camera 1
            p1, the 2D coordinates of points in image 1
            K2, the intrinsics of camera 2
            p2, the 2D coordinates of points in image 2
            x, the flattened concatenationg of P, r2, and t2.
    Output: residuals, 4N x 1 vector, the difference between original and
                       estimated projections
'''
def rodriguesResidual(K1, M1, p1, K2, p2, x):
    # Extract w, r, and t
    N = p1.shape[0]
    w = x[0: -6].reshape(N, 3)
    w_homo = np.concatenate((w, np.ones((N, 1))), axis = 1)
    r2 = x[-6: -3].reshape(3, 1)
    t2 = x[-3:].reshape(3, 1)

    # Compute C1 and C2
    C1 = K1 @ M1
    M2 = np.concatenate((rodrigues(r2), t2), axis = 1)
    C2 = K2 @ M2

    # Compute estimated projections
    p1_est = C1 @ w_homo.T
    p2_est = C2 @ w_homo.T
    p1_hat = p1_est.T / p1_est.T[:, -1].reshape(-1, 1)
    p2_hat = p2_est.T / p2_est.T[:, -1].reshape(-1, 1)

    # Compute Residuals
    residuals = np.concatenate([(p1-p1_hat[:, :2]).reshape([-1]),
                                (p2-p2_hat[:, :2]).reshape([-1])])

    return residuals

'''
Q5.3 Extra Credit  Bundle adjustment.
    Input:  K1, the intrinsics of camera 1
```

```
            M1, the extrinsics of camera 1
            p1, the 2D coordinates of points in image 1
            K2,  the intrinsics of camera 2
            M2_init, the initial extrinsics of camera 1
            p2, the 2D coordinates of points in image 2
            P_init, the initial 3D coordinates of points
    Output: M2, the optimized extrinsics of camera 1
            w, the optimized 3D coordinates of points
'''
def bundleAdjustment(K1, M1, p1, K2, M2_init, p2, P_init):
    # Set up the x for rodriguesResidual
    r2_init = invRodrigues(M2_init[:, 0:3]).reshape(-1, 1)
    t2_init = M2_init[:, 3].reshape(-1, 1)
    w_init = P_init.reshape(-1, 1)
    x_init = np.concatenate((w_init, r2_init, t2_init), axis = 0).reshape([-1])

    # Define the residual function
    def residual_func(x):
        return rodriguesResidual(K1, M1, p1, K2, p2, x)

    # Apply Least Square Optimizer to solve for x
    x = scipy.optimize.leastsq(residual_func, x_init)
    x = x[0]

    # Extract w, r2, and t2
    N = p1.shape[0]
    w = x[0: -6].reshape(N, 3)
    r2 = x[-6: -3].reshape(3, 1)
    t2 = x[-3:].reshape(3, 1)

    # Build M2 using r2 and t2
    R2 = rodrigues(r2)
    M2 = np.concatenate((R2, t2), axis = 1)

    return M2, w
```

0.2 findM2.py

```
'''
Q3.3:
    1. Load point correspondences
    2. Obtain the correct M2
    3. Save the correct M2, C2, and P to q3_3.npz
'''

# Import necessary package
import numpy as np
import matplotlib.pyplot as plt
import helper
import submission
import os

# Load the image and M
data_dir = '../data/'
img1 = plt.imread(data_dir + 'im1.png')
M = np.max(img1.shape)

# Load the correspondences
corresp = np.load(data_dir + 'some_corresp.npz')
pts1 = corresp['pts1']
pts2 = corresp['pts2']

# Compute the Fundamental Matrix
F = submission.eightpoint(pts1, pts2, M)

# Load the Intrinsic Matrices
intrinsics = np.load(data_dir + 'intrinsics.npz')
K1 = intrinsics['K1']
K2 = intrinsics['K2']

# Compute the Essential Matrix
E = submission.essentialMatrix(F, K1, K2)

# Compute M1, C1, and M2s
M1 = np.concatenate((np.eye(3), np.zeros((3, 1))), axis = 1)
C1 = K1 @ M1
M2s = helper.camera2(E)

# Find the correct M2
for i in range(4):
    # Check each M2
    M2 = M2s[:, :, i]
    C2 = K2 @ M2
    P, err = submission.triangulate(C1, pts1, C2, pts2)
```

```

    # Check the validity (z is positive)
    if np.min(P[:, 2]) > 0:
        # Print the reprojection error
        print(f"reprojection error = {err}")
        break

# Save M2, C2, and P
results_dir = '../results/'
if not os.path.exists(results_dir):
    os.makedirs(results_dir)
np.savez(results_dir+ 'q3_3.npz', M2 = M2, C2 = C2, P = P)
```

0.3 visualize.py

```
'''
Q4.2:
    1. Integrating everything together.
    2. Loads necessary files from ../data/ and visualizes 3D reconstruction
    using scatter
'''

# Import necessary package
import numpy as np
import matplotlib.pyplot as plt
import helper
import submission
import os

# Load the image and M
data_dir = '../data/'
img1 = plt.imread(data_dir + 'im1.png')
img2 = plt.imread(data_dir + 'im2.png')
M = np.max(img1.shape)

# Load the correspondences
corresp = np.load(data_dir + 'some_corresp.npz')
pts1 = corresp['pts1']
pts2 = corresp['pts2']

# Compute the Fundamental Matrix
F = submission.eightpoint(pts1, pts2, M)

# Load the Intrinsic Matrices
intrinsics = np.load(data_dir + 'intrinsics.npz')
K1 = intrinsics['K1']
K2 = intrinsics['K2']

# Compute the Essential Matrix
E = submission.essentialMatrix(F, K1, K2)

# Compute M1, C1, and M2s
M1 = np.concatenate((np.eye(3), np.zeros((3, 1))), axis = 1)
C1 = K1 @ M1
M2s = helper.camera2(E)

# Load the Temple Coordinates on image1
templeCoords = np.load(data_dir + 'templeCoords.npz')
x1s = templeCoords['x1']
y1s = templeCoords['y1']
p1 = np.concatenate((x1s, y1s), axis = 1)
```

```

# Find the epipolar correspondence
p2 = np.zeros_like(p1)
for i in range(p2.shape[0]):
    x1, y1 = p1[i, 0], p1[i, 1]
    p2[i, 0], p2[i, 1] = submission.epipolarCorrespondence(img1, img2, F, x1, y1)

# Find the correct M2
for i in range(4):
    # Check each M2
    M2 = M2s[:, :, i]
    C2 = K2 @ M2
    P, err = submission.triangulate(C1, p1, C2, p2)

    # Check the validity (z is positive)
    if np.min(P[:, 2]) > 0:
        # Print the reprojection error
        print(f"reprojection error = {err}")
        break

# Plot the 3D reconstruction
fig = plt.figure()
ax = fig.add_subplot(1, 1, 1, projection = '3d')
ax.scatter(P[:, 0], P[:, 1], P[:, 2], c = 'b', s = 3)
ax.set_title('Visualization of 3D Recontruction')
plt.setp(ax, xlabel = 'X', ylabel = 'Y', zlabel = 'Z')
plt.show()

# Save F, M1, M2, C1, and C2
results_dir = '../results/'
if not os.path.exists(results_dir):
    os.makedirs(results_dir)
np.savez(results_dir+ 'q4_2.npz', F = F, M1 = M1, M2 = M2, C1 = C1, C2 = C2)
```

0.4 util.py

```
import numpy as np
import scipy.optimize

def _singularize(F):
    U, S, V = np.linalg.svd(F)
    S[-1] = 0
    F = U.dot(np.diag(S).dot(V))
    return F

def _objective_F(f, pts1, pts2):
    F = _singularize(f.reshape([3, 3]))
    num_points = pts1.shape[0]
    hpts1 = np.concatenate([pts1, np.ones([num_points, 1])], axis=1)
    hpts2 = np.concatenate([pts2, np.ones([num_points, 1])], axis=1)
    Fp1 = F.dot(hpts1.T)
    FTp2 = F.T.dot(hpts2.T)

    r = 0
    for fp1, fp2, hp2 in zip(Fp1.T, FTp2.T, hpts2):
        r += (hp2.dot(fp1))**2 * (1/(fp1[0]**2 + fp1[1]**2)
                                  + 1/(fp2[0]**2 + fp2[1]**2))
    return r

def refineF(F, pts1, pts2):
    f = scipy.optimize.fmin_powell(
        lambda x: _objective_F(x, pts1, pts2), F.reshape([-1]),
        maxiter=100000,
        maxfun=10000
    )
    return _singularize(f.reshape([3, 3]))
```

0.5 helper.py

```
"""
Homework4.
Helper functions.

Written by Dinesh Reddy, 2020.
"""
import numpy as np
import matplotlib.pyplot as plt
import scipy.optimize
import submission as sub
from mpl_toolkits.mplot3d import Axes3D
import os


connections_3d = [[0,1], [1,3], [2,3], [2,0], [4,5], [6,7], [8,9], [9,11],
                  [10,11], [10,8], [0,4], [4,8], [1,5], [5,9], [2,6], [6,10],
                  [3,7], [7,11]]
color_links = [(255,0,0),(255,0,0),(255,0,0),(255,0,0),(0,0,255),(255,0,255),
               (0,255,0),(0,255,0),(0,255,0),(0,255,0),(0,0,255),(0,0,255),
               (0,0,255),(0,0,255),(255,0,255),(255,0,255),(255,0,255),
               (255,0,255)]
colors = ['blue','blue','blue','blue','red','magenta','green','green','green',
          'green','red','red','red','red','magenta','magenta','magenta',
          'magenta']


def visualize_keypoints(image, pts, Threshold=None):
    '''
    plot 2d keypoint
    :param image: image
    :param car_points: np.array points * 3
    '''
    import cv2
    image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
    for i in range(12):
        cx, cy = pts[i][0:2]
        if pts[i][2]>Threshold:
            cv2.circle(image,(int(cx),int(cy)),5,(0,255,255),5)

    for i in range(len(connections_3d)):
        idx0, idx1 = connections_3d[i]
        if pts[idx0][2]>Threshold and pts[idx1][2]>Threshold:
            x0, y0 = pts[idx0][0:2]
            x1, y1 = pts[idx1][0:2]
            cv2.line(image, (int(x0), int(y0)), (int(x1), int(y1)),
                     color_links[i], 2)
```

```
    while True:
        cv2.imshow("sample", image)
        if cv2.waitKey(0) == 27:
            break
    cv2.destroyAllWindows()
    return (image)

def plot_3d_keypoint(pts_3d):
    '''
    plot 3d keypoint
    :param car_points: np.array points * 3
    '''
    fig = plt.figure()
    num_points = pts_3d.shape[0]
    ax = fig.add_subplot(111, projection='3d')
    for j in range(len(connections_3d)):
        index0, index1 = connections_3d[j]
        xline = [pts_3d[index0,0], pts_3d[index1,0]]
        yline = [pts_3d[index0,1], pts_3d[index1,1]]
        zline = [pts_3d[index0,2], pts_3d[index1,2]]
        ax.plot(xline, yline, zline, color=colors[j])
    np.set_printoptions(threshold=1e6, suppress=True)
    ax.set_xlabel('X Label')
    ax.set_ylabel('Y Label')
    ax.set_zlabel('Z Label')
    plt.show()


def _epipoles(E):
    U, S, V = np.linalg.svd(E)
    e1 = V[-1, :]
    U, S, V = np.linalg.svd(E.T)
    e2 = V[-1, :]
    return e1, e2

def displayEpipolarF(I1, I2, F):
    e1, e2 = _epipoles(F)

    sy, sx, _ = I2.shape

    f, [ax1, ax2] = plt.subplots(1, 2, figsize=(12, 9))
    ax1.imshow(I1)
    ax1.set_title('Select a point in this image')
    ax1.set_axis_off()
    ax2.imshow(I2)
    ax2.set_title('Verify that the corresponding point \n is on the epipolar
                  line in this image')
    ax2.set_axis_off()
```

```python

    while True:
        plt.sca(ax1)
        x, y = plt.ginput(1, timeout=3600, mouse_stop=2)[0]

        xc = x
        yc = y
        v = np.array([xc, yc, 1])
        l = F.dot(v)
        s = np.sqrt(l[0]**2+l[1]**2)

        if s==0:
            print('Zero line vector in displayEpipolar')

        l = l/s

        if l[0] != 0:
            ye = sy-1
            ys = 0
            xe = -(l[1] * ye + l[2])/l[0]
            xs = -(l[1] * ys + l[2])/l[0]
        else:
            xe = sx-1
            xs = 0
            ye = -(l[0] * xe + l[2])/l[1]
            ys = -(l[0] * xs + l[2])/l[1]

        # plt.plot(x,y, '*', 'MarkerSize', 6, 'LineWidth', 2);
        ax1.plot(x, y, '*', MarkerSize=6, linewidth=2)
        ax2.plot([xs, xe], [ys, ye], linewidth=2)
        plt.draw()




def camera2(E):
    U,S,V = np.linalg.svd(E)
    m = S[:2].mean()
    E = U.dot(np.array([[m,0,0], [0,m,0], [0,0,0]])).dot(V)
    U,S,V = np.linalg.svd(E)
    W = np.array([[0,-1,0], [1,0,0], [0,0,1]])

    if np.linalg.det(U.dot(W).dot(V))<0:
        W = -W

    M2s = np.zeros([3,4,4])
    M2s[:,:,0] = np.concatenate([U.dot(W).dot(V), U[:,2].reshape([-1, 1])
                                 /abs(U[:,2]).max()], axis=1)
```

```
    M2s[:,:,1] = np.concatenate([U.dot(W).dot(V), -U[:,2].reshape([-1, 1])
                             /abs(U[:,2]).max()], axis=1)
    M2s[:,:,2] = np.concatenate([U.dot(W.T).dot(V), U[:,2].reshape([-1, 1])
                             /abs(U[:,2]).max()], axis=1)
    M2s[:,:,3] = np.concatenate([U.dot(W.T).dot(V), -U[:,2].reshape([-1, 1])
                             /abs(U[:,2]).max()], axis=1)
    return M2s

def epipolarMatchGUI(I1, I2, F):
    e1, e2 = _epipoles(F)

    sy, sx, _ = I2.shape

    f, [ax1, ax2] = plt.subplots(1, 2, figsize=(12, 9))
    ax1.imshow(I1)
    ax1.set_title('Select a point in this image')
    ax1.set_axis_off()
    ax2.imshow(I2)
    ax2.set_title('Verify that the corresponding point \n is on the epipolar
               line in this image')
    ax2.set_axis_off()

    # Create p1s and p2s to store the corresponding points
    p1s = []
    p2s = []

    while True:
        plt.sca(ax1)
        # Get the input point on image1
        p1 = plt.ginput(1, mouse_stop=2)
        # Check p1
        if not p1:
            break
        # Extract x1 and y1
        x1, y1 = p1[0]

        xc = int(x1)
        yc = int(y1)
        v = np.array([xc, yc, 1])
        l = F.dot(v)
        s = np.sqrt(l[0]**2+l[1]**2)

        if s==0:
            print('Zero line vector in displayEpipolar')

        l = l/s;

        if l[0] != 0:
```

```
            ye = sy-1
            ys = 0
            xe = -(l[1] * ye + l[2])/l[0]
            xs = -(l[1] * ys + l[2])/l[0]
        else:
            xe = sx-1
            xs = 0
            ye = -(l[0] * xe + l[2])/l[1]
            ys = -(l[0] * xs + l[2])/l[1]

        # plt.plot(x,y, '*', 'MarkerSize', 6, 'LineWidth', 2);
        ax1.plot(x1, y1, '*', MarkerSize=6, linewidth=2)
        ax2.plot([xs, xe], [ys, ye], linewidth=2)

        # draw points
        x2, y2 = sub.epipolarCorrespondence(I1, I2, F, xc, yc)
        ax2.plot(x2, y2, 'ro', MarkerSize=8, linewidth=2)
        plt.draw()

        # Append p1 and p2 to p1s and p2s
        p1 = [x1, y1]
        p2 = [x2, y2]
        p1s.append(p1)
        p2s.append(p2)

    # Save F, p1s, and p2s
    results_dir = '../results/'
    if not os.path.exists(results_dir):
        os.makedirs(results_dir)
    np.savez(results_dir+ 'q4_1.npz', F = F, pts1 = p1s, pts2 = p2s)
```