In [ ]:

16720 (B) 3D Reconstruction - Assignment 5

```
Instructor: Kris                         TAs: Zen (Lead), Yan, Rawal,
Wen-Hsuan, Paritosh, Qichen
```

Instructions

This section should include the visualizations and answers to specifically highlighted questions from P1 to P4. This section will need to be uploaded to gradescope as a pdf and manually graded (this is a separate submission from the coding notebooks).

1. Students are encouraged to work in groups but each student must submit their own work. Include the names of your collaborators in your write up. Code should Not be shared or copied. Please properly give credits to others by LISTING EVERY COLLABORATOR in the writeup including any code segments that you discussed, Please DO NOT use external code unless permitted. Plagiarism is prohibited and may lead to failure of this course.
2. **Start early!** This homework will take a long time to complete.
3. **Questions:** If you have any question, please look at Piazza first and the FAQ page for this homework.
4. All the theory question and manually graded questions should be included in a single writeup (this notebook exported as pdf or a standalone pdf file) and submitted to gradescope: pdf assignment.
5. **Attempt to verify your implementation as you proceed:** If you don't verify that your implementation is correct on toy examples, you will risk having a huge issue when you put everything together. We provide some simple checks in the notebook cells, but make sure you verify them on more complicated samples before moving forward.
6. **Do not import external functions/packages other than the ones already imported in the files:** The current imported functions and packages are enough for you to complete this assignment. If you need to import other functions, please remember to comment them out after submission. Our autograder will crash if you import a new function that the gradescope server does not expect.
7. Assignments that do not follow this submission rule will be **penalized up to 10\% of the total score**.

Theory Questions (25 pts)

Before implementing our own 3D reconstruction, let's take a look at some simple theory questions that may arise. The answers to the below questions should be relatively short, consisting of a few lines of math and text (maybe a diagram if it helps your understanding).

Figure1. Figure for Q1.1. $C1$ and $C2$ are the optical centers. The principal axes intersect at point $\mathbf{w}$ (P in the figure).

Q1.1

Suppose two cameras fixated on a point x (see Figure 1) in space such that their principal axes intersect at the point P. Show that if the image coordinates are normalized so that the coordinate origin $(0, 0)$ coincides with the principal point, the $\mathbf{F}_{33}$ element of the fundamental matrix is zero.

Solution

Q1.1 Solution

Given two cameras with C_1 and C_2 as optical centers.

Let x_1 and x_2 be principal points in camera 1 and camera 2

Since principle points coincide with co-ordinate origin.

$$x_1^T = [0 \quad 0 \quad 1]$$

$$x_2^T = [0 \quad 0 \quad 1]$$

Since Fundamental Matrix satisfies Longuet-Higgins equation.

$$x_1^T F x_2 = 0$$

$$\text{where } F \text{ is } \begin{bmatrix} F_{11} & F_{12} & F_{13} \\ F_{21} & F_{22} & F_{23} \\ F_{31} & F_{32} & F_{33} \end{bmatrix}$$

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

$$\Rightarrow \boxed{F_{33} = 0}$$

End

Q1.2

Consider the case of two cameras viewing an object such that the second camera differs from the first by a pure translation that is parallel to the x-axis. Show that the epipolar lines in the two cameras are also parallel to the x-axis. Backup your argument with relevant equations.

Solution

1.2) Given Second camera differs from the first camera by a pure translation parallel to x-axis.

so translation matrix $t = \begin{bmatrix} t_x \\ t_y \\ t_z \end{bmatrix}$

since translation is parallel to axis x'

$t_x = t$; $t_y = 0$; $t_z = 0$

translation matrix can be written in Skew symmetric matrix.

$$[t_x] = \begin{bmatrix} 0 & -t_z & t_y \\ +t_z & 0 & -t_x \\ -t_y & t_x & 0 \end{bmatrix} \Rightarrow \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & -t \\ 0 & t & 0 \end{bmatrix}$$

consider x and x' as points in two images. Then the corresponding epipolar lines are.

$l' = Ex$ (in $\overset{\text{line}}{\text{camera}}$ 2) $= Ex$

$l = E^T x'$ (line in camera 1)

$$l' = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & -t \\ 0 & +t & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ -t \\ y_1 t \end{bmatrix}$$

so equation of line is $yt = y_1 t$

$y t = y_1 t \Rightarrow \boxed{y = y_1}$

$\Rightarrow$ line is parallel to x axis

$$l = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & t \\ 0 & -t & 0 \end{bmatrix} \begin{bmatrix} x_2 \\ y_2 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ t \\ -y_2 t \end{bmatrix}$$

equation of line $\Rightarrow ty = y_2 t$

$\Rightarrow y = y_2 \Rightarrow$ line is parallel to x axis

So $y = y_1$ and $y = y_2$ are the epipolar lines in two cameras which are parallel to each other and also parallel to x-axis.

End

Q1.3

Suppose we have an inertial sensor which gives us the accurate extrinsics $\mathbf{R}_i$ and $\mathbf{t}_i$ (see Figure 2), the rotation matrix and translation vector of the robot at time i. What will be the effective rotation ($\mathbf{R}_{rel}$) and translation ($\mathbf{t}_{rel}$) between two frames at different time stamps? Suppose the camera intrinsics ($\mathbf{K}$) are known, express the essential matrix ($\mathbf{E}$) and the fundamental matrix ($\mathbf{F}$) in terms of $\mathbf{K}$, $\mathbf{R}_{rel}$ and $\mathbf{t}_{rel}$.

Figure 2. Figure for Q1.3. $C1$ and $C2$ are the optical centers. The rotation and the translation is obtained using inertial sensors. $\mathbf{R}_{rel}$ and $\mathbf{t}_{rel}$ are the relative rotation and translation between two frames.

Solution

1.3 Solution

Given accurate extrinsics at time i is R_i, t_i and camera intrinsics are K.

Consider a 3D point in real co-ordinate system

$$P = \begin{bmatrix} X \\ Y \\ Z \end{bmatrix}$$

At two times 1, 2 camera parameters are K, R_1, t_1 and K, R_2, t_2, the 3D in camera co-ordinates are

$$x_1 = R_1 P + t_1$$
$$x_2 = R_2 P + t_2$$
$$\text{so } P = R_1^{-1}(x_1 - t_1)$$

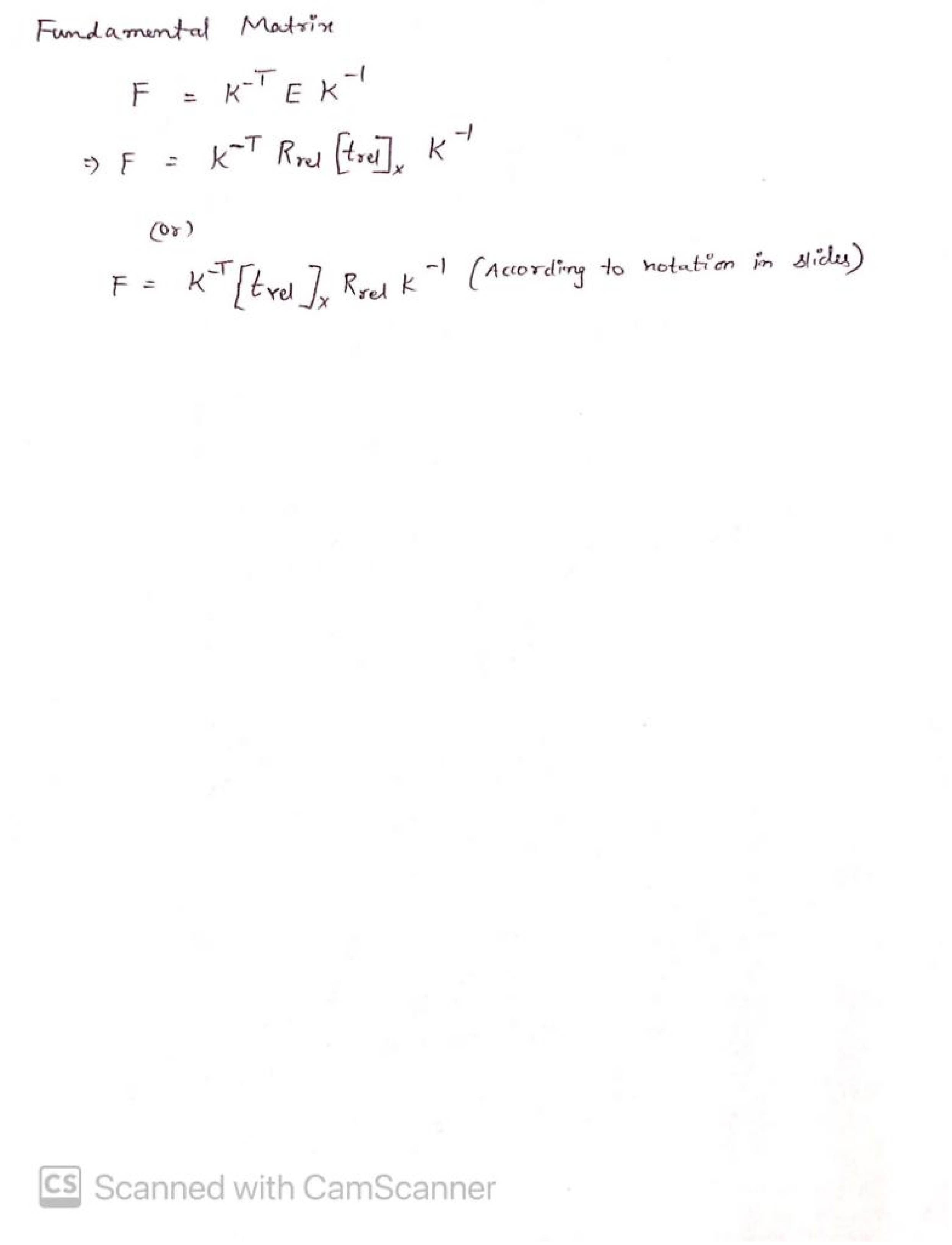
substituting

$$x_2 = R_2 R_1^{-1}(x_1 - t_1) + t_2$$
$$= R_2 R_1^{-1}(x_1) - R_2 R_1^{-1} t_1 + t_2$$
$$\Rightarrow \underset{(R_{rel})}{R'}(x_1) + \underset{(t_{rel})}{t'}$$

so $R_{rel} = R_2 R_1^{-1}$

$t_{rel} = -R_2 R_1^{-1} t_1 + t_2$

from decomposition rule $\Rightarrow E = R_{rel}[t_{rel}]_\times$ (according to notation in slides)



Fundamental Matrix

$$F = K^{-T} E K^{-1}$$

$$\Rightarrow F = K^{-T} R_{rel} [t_{rel}]_{\times} K^{-1}$$

(or)

$$F = K^{-T} [t_{rel}]_{\times} R_{rel} K^{-1}$$ (According to notation in slides)

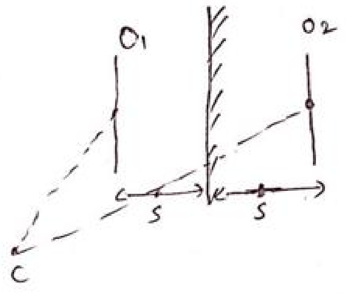
End

Q1.4

Suppose that a camera views an object and its reflection in a plane mirror. Show that this situation is equivalent to having two images of the object which are related by a skew-symmetric fundamental matrix. You may assume that the object is flat, meaning that all points on the object are of equal distance to the mirror (**Hint:** draw the relevant vectors to understand the relationship between the camera, the object, and its reflected image.)

Solution

1.4 Solution



Consider camera 'c' viewing objects O_1 and O_2 where O_2 is reflection of O_1.

Let K be intrinsic matrix and F is the Fundamental matrix.

Since the object is flat, it can be assumed that there exists only a translation b/w two objects

i.e.

$$t = \begin{bmatrix} t_x \\ t_y \\ t_z \end{bmatrix} \quad \text{and} \quad R = I$$

$$[t_x] = \begin{bmatrix} 0 & t_z & -t_y \\ -t_z & 0 & t_x \\ t_y & -t_x & 0 \end{bmatrix} \; ; \; R = I$$

$$\Rightarrow E = [t_x] R \Rightarrow \begin{bmatrix} 0 & t_z & -t_y \\ -t_z & 0 & t_x \\ t_y & -t_x & 0 \end{bmatrix}$$

$$F = K^{-T} E K^{-1} = (K^{-1})^T \begin{bmatrix} 0 & t_z & -t_y \\ -t_z & 0 & t_x \\ t_y & -t_x & 0 \end{bmatrix} K^{-1}$$

taking transponse

$$F^T = (K^{-1})^T \begin{bmatrix} 0 & t_z & -t_y \\ -t_z & 0 & t_x \\ t_y & -t_x & 0 \end{bmatrix}^T \left((K^{-1})^T\right)^T$$

$$= K^{-T} \begin{bmatrix} 0 & -t_z & t_y \\ t_z & 0 & -t_x \\ -t_y & t_x & 0 \end{bmatrix} K^{-1}$$

$$= - K^{-T} \begin{bmatrix} 0 & +t_z & -t_y \\ -t_z & 0 & t_x \\ t_y & -t_x & 0 \end{bmatrix} K^{-1}$$

$$= -F$$

$\Rightarrow \boxed{F^T = -F} \Rightarrow$ F is a skew symmetric matrix.

End

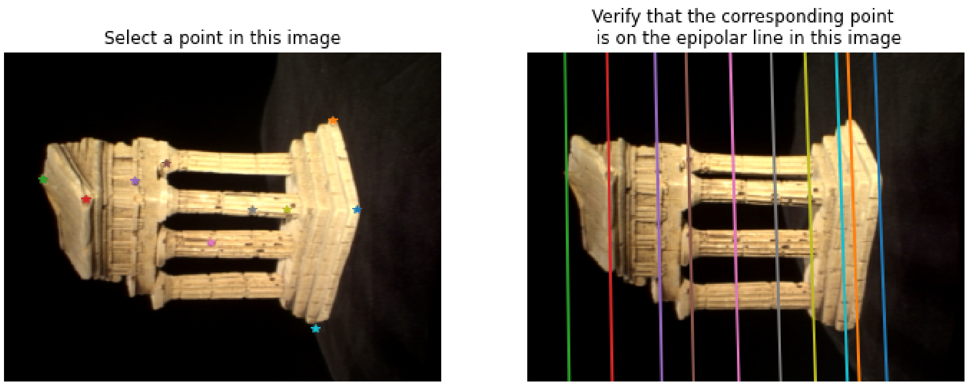
Discussed approach with Vishnu Mani Hema, Damanpreet Singh, Nitika Suresh

Coding Questions (30 pt)

Q1.1: The Eight Point Algorithm

Output: In your write-up: Write your recovered $\mathbf{F}$ and include an image of some example outputs of displayEpipolarF.

Solution


Select a point in this image
Verify that the corresponding point is on the epipolar line in this image

```
Optimization terminated successfully.
         Current function value: 0.000107
         Iterations: 8
         Function evaluations: 893
[[-0.      0.     -0.2519]
 [ 0.     -0.      0.0026]
 [ 0.2422 -0.0068  1.    ]]
Error: 0.39893072465911866
```

Code

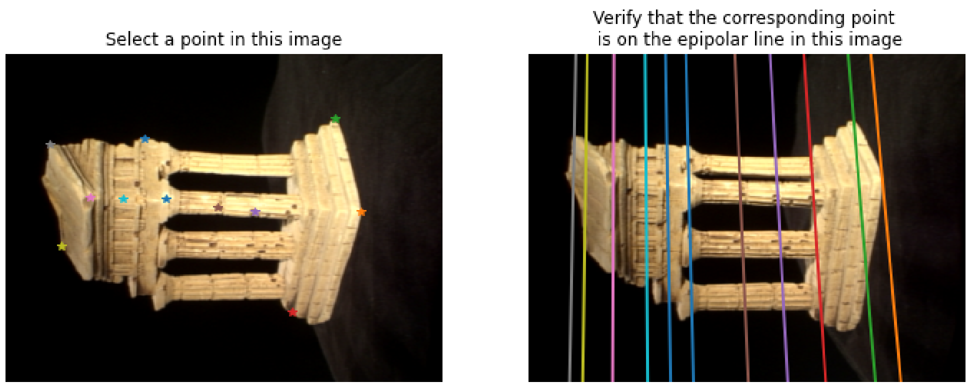
```
[4]: def eightpoint(pts1, pts2, M):
         '''
         Q1.1: Eight Point Algorithm for calculating the fundamental matrix
             Input:  pts1, Nx2 Matrix containing the corresponding points from image1
                     pts2, Nx2 Matrix containing the corresponding points from image2
                     M, a scalar parameter computed as max (imwidth, imheight)
             Output: F, the fundamental matrix of shape (3, 3)

         ***
         HINTS:
         (1) Normalize the input pts1 and pts2 using the matrix T.
         (2) Setup the eight point algorithm's equation.
         (3) Solve for the least square solution using SVD.
         (4) Use the function `_singularize` (provided) to enforce the singularity condition.
         (5) Use the function `refineF` (provided) to refine the computed fundamental matrix.
             (Remember to usethe normalized points instead of the original points)
         (6) Unscale the fundamental matrix

         '''

         F = None # output fundamental matrix
         N = pts1.shape[0] # Extrating the number of points

         # Converting the points to homogenous coordinates
         pts1_homogenous, pts2_homogenous = toHomogenous(pts1), toHomogenous(pts2)

         # Computing the 3x3 matrix used to normalize corresponding points.
         T = np.diag([1/M,1/M,1])
         spts1 = pts1_homogenous @ T
         spts2 = pts2_homogenous @ T

         singlepoint = []
         for i in range(N):
             row1 = np.array([spts1[i,0]*spts2[i,0], spts1[i,0]*spts2[i,1], spts1[i,0], spts1[i,1]*spts2[i,0], spts1[i,1]*spts2[i,1], spts1[i,1], spts2[i,0], spts2[i,1], 1])
             singlepoint.append(row1)

         A = np.array(singlepoint)
         u, s, vh = np.linalg.svd(A, full_matrices = True)
         F = vh[-1, :].reshape((3,3)).T

         F = refineF(F, spts1[:,0:2], spts2[:,0:2])

         F = F/F[2,2] #Finding the unique fundamental matrix by setting the scale to 1.

         F_un = T.T @ F @ T


         return F_un
```

End

Q1.2: The Seven Point Algorithm

Output: In your write-up: Print your recovered $\mathbf{F}$ and include an image output of `displayEpipolarF` .

Solution


Select a point in this image
Verify that the corresponding point
is on the epipolar line in this image

```
Error: 0.5866613571899385
```

```
[10]: print(F)
      displayEpipolarF(im1, im2, F)
```

```
[[ 0.     -0.     -0.1953]
 [ 0.      0.     -0.0067]
 [ 0.1869  0.0026  1.    ]]
```

Code

```
[13]: import numpy.polynomial.polynomial as roots
def sevenpoint(pts1, pts2, M):
    '''
    Q1.2: Seven Point Algorithm for calculating the fundamental matrix
        Input:  pts1, 7x2 Matrix containing the corresponding points from image1
                pts2, 7x2 Matrix containing the corresponding points from image2
                M, a scalar parameter computed as max (imwidth, imheight)
        Output: Farray, a list of estimated 3x3 fundamental matrixes.

    ***
    HINTS:
    (1) Normalize the input pts1 and pts2 scale paramter M.
    (2) Setup the seven point algorithm's equation.
    (3) Solve for the least square solution using SVD.
    (4) Pick the last two colum vector of vT.T (the two null space solution f1 and f2)
    (5) Use the singularity constraint to solve for the cubic polynomial equation of  F = a*f1 + (1-a)*f2 that leads to
        det(F) = 0. Sovling this polynomial will give you one or three real solutions of the fundamental matrix.
        Use np.polynomial.polynomial.polyroots to solve for the roots
    (6) Unscale the fundamental matrixes and return as Farray
    '''
    Farray = []
    N = pts1.shape[0]

    pts1_homogenous, pts2_homogenous = toHomogenous(pts1), toHomogenous(pts2)

    T = np.diag([1/M,1/M,1])
    spts1 = pts1_homogenous @ T
    spts2 = pts2_homogenous @ T

    singlepoint = []
    for i in range(N):
        row1 = np.array([spts1[i,0]*spts2[i,0], spts1[i,0]*spts2[i,1], spts1[i,0], spts1[i,1]*spts2[i,0], spts1[i,1]*spts2[i,1], spts1[i,1], spts2[i,0], spts2[i,1], 1])
        singlepoint.append(row1)

    A = np.array(singlepoint)
    u, s, vh = np.linalg.svd(A, full_matrices = True)

    F1 = vh[-1, :].reshape((3,3))
    F2 = vh[-2, :].reshape((3,3))

    #substituting values of 0, 1, -1 and 2 in the equation F = a*f1 + (1-a)*f2 to get the co-efficients.
```

```
    c0 = np.linalg.det(F2)
    c2 = (np.linalg.det(2*F2 - F1) + np.linalg.det(F2))/2 - np.linalg.det(F2)
    c3 = (np.linalg.det(2*F1 - F2) - 2*c2 + c0 -2*np.linalg.det(F1))/6
    c1 = np.linalg.det(F1) - c0 - c2 - c3

    rootsx = roots.polyroots((c0, c1, c2, c3))
    Farray = []
    for x in rootsx:
        if x.imag == 0:
            a = x.real
            F_ma = F1*a + F2*(1-a)
            # F_rf = refineF(F_ma, spts1[:,0:2], spts2[:,0:2])
            F_f = T.T @ F_ma @ T
            F_f = _singularize(F_f)
            F_f = F_f/F_f[2,2]
            Farray.append(F_f)

    Farray = np.stack(Farray, axis=-1)

    return Farray.T
```

End

Discussed approach with Vishnu Mani Hema, Damanpreet Singh, Nitika Suresh

Q2.2 Triangulation and find M2

Output: In your write-up: Write down the expression for the matrix $\mathbf{A}_i$

Solution

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \alpha \begin{bmatrix} — P_1^T — \\ — P_2^T — \\ — P_3^T — \end{bmatrix} \begin{bmatrix} 1 \\ X \\ 1 \end{bmatrix} \quad \left(\underset{\substack{\downarrow \\ \text{Inhomogenous co.ordinate.}}}{x} = \alpha P X \right)$$

since They are in same direction

$$x \times PX = 0$$

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \alpha \begin{bmatrix} P_1^T X \\ P_2^T X \\ P_3^T X \end{bmatrix}$$

$$x \times PX = 0$$

$$\Rightarrow \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \times \begin{bmatrix} P_1^T X \\ P_2^T X \\ P_3^T X \end{bmatrix} = \begin{bmatrix} y P_3^T X - P_2^T X \\ P_1^T X - x P_3^T X \\ \underbrace{x P_2^T X - y P_1^T X}_{\text{linear combination of first two}} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

$$\Rightarrow \begin{bmatrix} y P_3^T - P_2^T \\ P_1^T - x P_3^T \end{bmatrix} X = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \quad \left(\text{one 2D-3D point correspondence gives 2 eq'ns} \right)$$

concatenating 2D points from both images

$$\begin{bmatrix} y P_3^T - P_2^T \\ P_1^T - x P_3^T \\ y' P_3'^T - P_2'^T \\ P_1'^T - x' P_3'^T \end{bmatrix} X = 0 \quad \Rightarrow A = \begin{bmatrix} y P_3^T - P_2^T \\ P_1^T - x P_3^T \\ y' P_3'^T - P_2'^T \\ P_1'^T - x' P_3'^T \end{bmatrix}_{4 \times 4}$$

and we solve for X

```
Optimization terminated successfully.
         Current function value: 0.000107
         Iterations: 8
         Function evaluations: 893
Best Error 352.23022293434116
M2: [[ 0.9994  0.0333  0.006  -0.026 ]
 [-0.0337  0.9653  0.2589 -1.    ]
 [ 0.0028 -0.259   0.9659  0.0798]]
C2 [[ 1520.39     -27.6373   301.1047   -15.4175]
 [  -50.7615  1409.0432   633.511  -1506.1956]
 [    0.0028    -0.259      0.9659     0.0798]]
```

Code

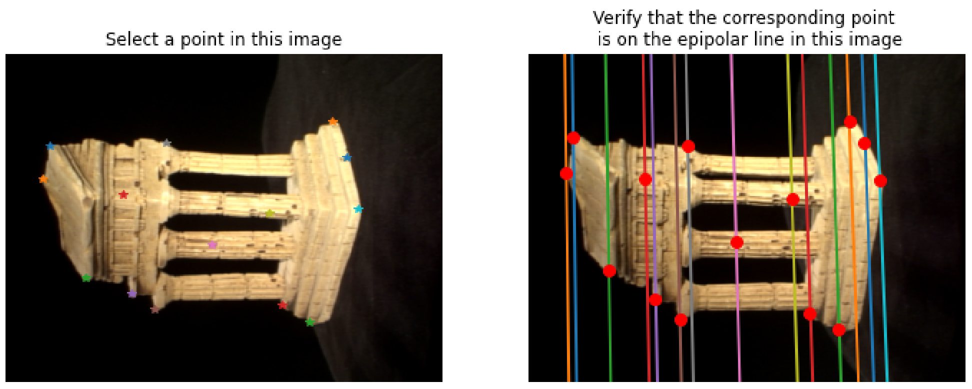
```
[5]: def triangulate(C1, pts1, C2, pts2):
         '''
         Q2.2: Triangulate a set of 2D coordinates in the image to a set of 3D points.
             Input:  C1, the 3x4 camera matrix
                     pts1, the Nx2 matrix with the 2D image coordinates per row
                     C2, the 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.

         ***
         Hints:
         (1) For every input point, form A using the corresponding points from pts1 & pts2 and C1 & C2
         (2) Solve for the least square solution using np.linalg.svd
         (3) Calculate the reprojection error using the calculated 3D points and C1 & C2 (do not forget to convert from
             homogeneous coordinates to non-homogeneous ones)
         (4) Keep track of the 3D points and projection error, and continue to next point
         (5) You do not need to follow the exact procedure above.
         '''
         # ----- TODO -----
         # YOUR CODE HERE
         # raise NotImplementedError()
         err = 0
         N = pts1.shape[0]
         A = np.zeros((4,4))
         P = np.zeros((N,3))
         for i in range(N):
             A[0,:] = C1[2,:]*pts1[i,1] - C1[1,:]
             A[1,:] = C1[0,:] - pts1[i,0]*C1[2,:]
             A[2,:] = C2[2,:]*pts2[i,1] - C2[1,:]
             A[3,:] = C2[0,:] - pts2[i,0]*C2[2,:]
             u, s, vh = np.linalg.svd(A, full_matrices = True)
             X = vh[-1, :]
             X = X / X[3]
             x1 = X @ C1.T
             x2 = X @ C2.T
             x1 = x1 / x1[2]
             x2 = x2 / x2[2]
             err = err + (np.linalg.norm(x1[0:2] - pts1[i]))**2 + (np.linalg.norm(x2[0:2] - pts2[i]))**2
             P[i,:] = X[0:3]

         return P, err
```

End

Q2.3 Epipolar Correspondence

Output: In your write-up, include a screenshot of `epipolarMatchGUI` with some detected correspondences.

Solution


Select a point in this image
Verify that the corresponding point
is on the epipolar line in this image

Code

```
[9]: def gaussian_weight(patch, sigma, window_s):
         x, y = np.meshgrid(np.linspace(-1,1,window_s+1), np.linspace(-1,1,window_s+1))
         dst = np.sqrt(x*x+y*y)
         gauss = np.exp(-( (dst)**2 / ( 2.0 * sigma**2 ) ) )
         result = np.zeros((patch.shape[0], patch.shape[1], patch.shape[-1]))
         for i in range(patch.shape[-1]):
             result[:,:,i] = gauss*patch[:,:,i]
         #result = result/np.sum(result)
         return np.mean(result, axis = -1)
```

```
def epipolarCorrespondence(im1, im2, F, x1, y1):
    '''
    Q2.3 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

    ***
    Hints:
    (1) Given input [x1, x2], use the fundamental matrix to recover the corresponding epipolar line on image2
    (2) Search along this line to check nearby pixel intensity (you can define a search window) to
        find the best matches
    (3) Use guassian weighting to weight the pixel simlairty

    '''
    # ----- TODO -----
    # YOUR CODE HERE
    X1 = np.array([x1, y1, 1])
    line = F @ X1.T

    window_s = 30
    patch1 = im1[y1-(window_s//2):y1+(window_s//2)+1, x1-(window_s//2):x1+(window_s//2)+1]
    filter1 = gaussian_weight(patch1, 5, window_s)#, sigma = 5)
    best_dist = np.finfo('float').max

    for i in range(window_s//2, im2.shape[0] - window_s//2):
        y2_i = i
        x2_i = (-line[2] - line[1]*y2_i)/line[0]
        x2_i = int(x2_i)
        patch2 = im2[y2_i-(window_s//2):y2_i+(window_s//2)+1, x2_i-(window_s//2):x2_i+(window_s//2)+1]
        filter2 = gaussian_weight(patch2, 5, window_s)#, sigma = 5)
        dist = np.linalg.norm(filter2 - filter1)
        if dist < best_dist:
            x2 = x2_i
            y2 = y2_i
            best_dist = dist

    return x2, y2
```

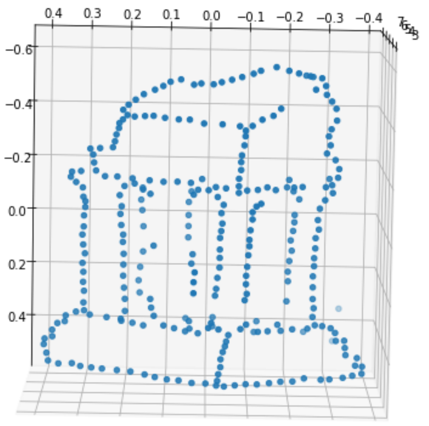
Discussed approach with Vishnu Mani Hema, Damanpreet Singh, Nitika Suresh

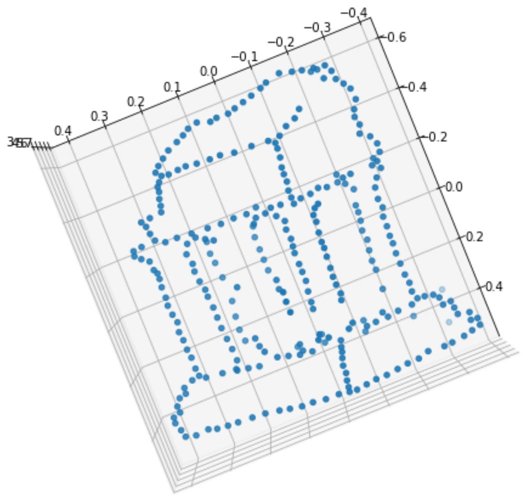
End

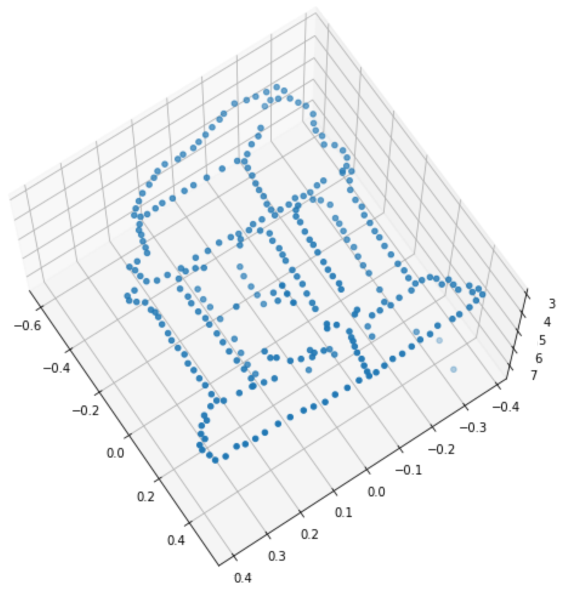
Q2.4 3D Visualization

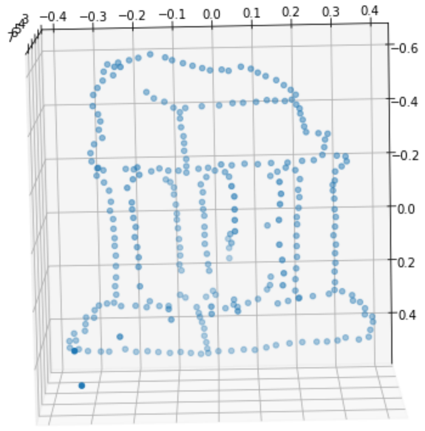
Output: In your write-up: Take a few screenshots of the 3D visualization so that the outline of the temple is clearly visible.

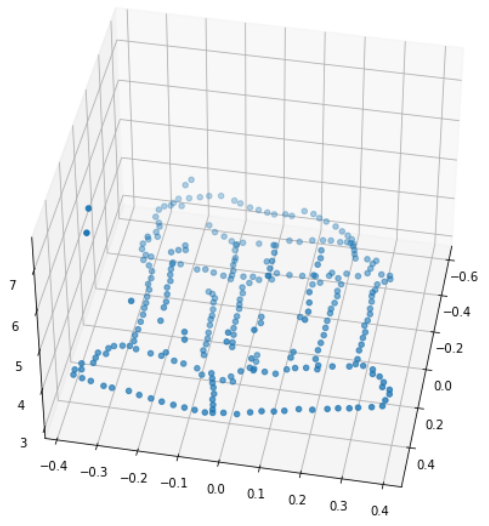
Solution


0.4
0.3
0.2
0.1
0.0
−0.1
−0.2
−0.3
−0.4
−0.6
−0.4
−0.2
0.0
0.2
0.4


−0.4
−0.3
−0.2
−0.1
0.0
0.1
0.2
0.3
0.4
−0.6
−0.4
−0.2
0.0
0.2
0.4


−0.6
−0.4
−0.2
0.0
0.2
0.4
0.4
0.3
0.2
0.1
0.0
−0.1
−0.2
−0.3
−0.4
3
4
5
6
7


−0.4
−0.3
−0.2
−0.1
0.0
0.1
0.2
0.3
0.4
−0.6
−0.4
−0.2
0.0
0.2
0.4



Error

```
Optimization terminated successfully.
         Current function value: 0.000107
         Iterations: 8
         Function evaluations: 893
Best Error 523.2012489438218
```

Code

```
[13]: def compute3D_pts(pts1, intrinsics, F, im1, im2):
          '''
          Q2.4: Finding the 3D position of given points based on epipolar correspondence and triangulation
              Input:  pts1, chosen points from im1
                      intrinsics, the intrinsics dictionary for calling epipolarCorrespondence
                      F, the fundamental matrix
                      im1, the first image
                      im2, the second image
              Output: P (Nx3) the recovered 3D points

          ***
          Hints:
          (1) Use epipolarCorrespondence to find the corresponding point for [x1 y1] (find [x2, y2])
          (2) Now you have a set of corresponding points [x1, y1] and [x2, y2], you can compute the M2
              matrix and use triangulate to find the 3D points.
          (3) Use the function findM2 to find the 3D points P (do not recalculate fundamental matrices)
          (4) As a reference, our solution's bet error is around ~2000 on the 3D points.
          '''
          x1s_temple, y1s_temple = pts1[:, 0], pts1[:, 1]
          N = x1s_temple.shape[0]
          x2s_temple = np.zeros((N,))
          y2s_temple = np.zeros((N,))
          for i in range(N):
              x2s_temple[i],y2s_temple[i] = epipolarCorrespondence(im1, im2, F, x1s_temple[i], y1s_temple[i])

          pts1 = np.stack((x1s_temple, y1s_temple), axis = 1)
          pts2 = np.stack((x2s_temple, y2s_temple), axis = 1)

          M2, C2, P = find_M2(F, pts1, pts2, intrinsics)

          # ----- TODO -----
          # YOUR CODE HERE
          # raise NotImplementedError()


          return P
```

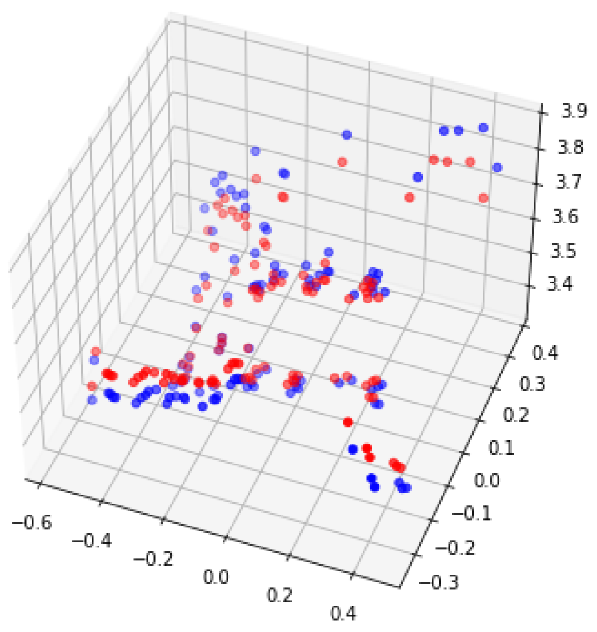
End

Q3.3 Bundle Adjustment

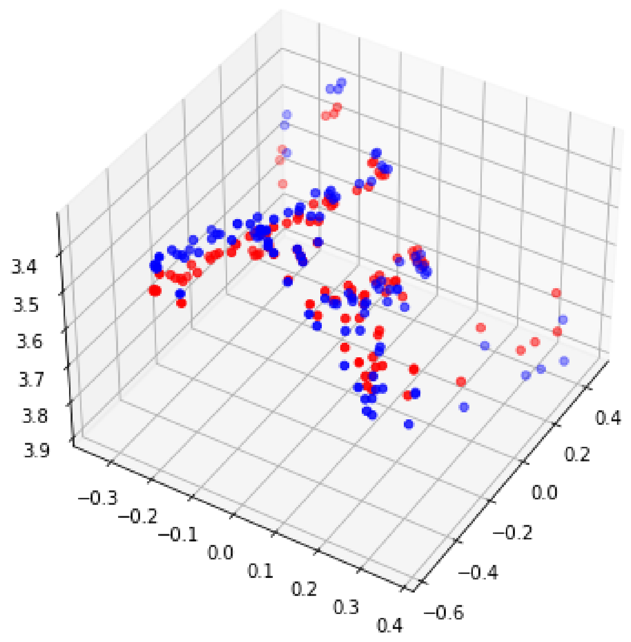
Output: In your write-up: include an image of output of the `plot_3D_dual` function by passing in the original 3D points and the optimized points. Also include the before and after reprojection error for the `rodriguesResidual` function.

Solution

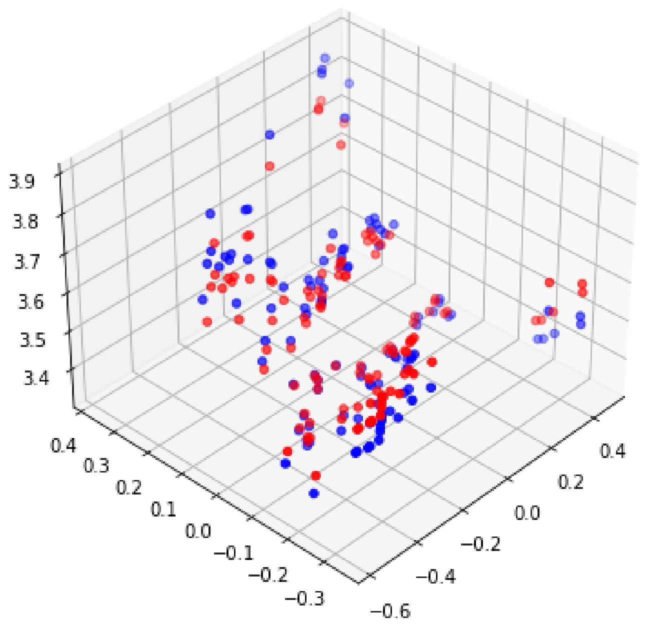
Blue: before; red: after



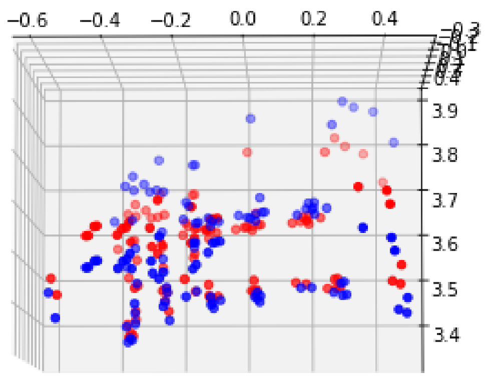
Blue: before; red: after



Blue: before; red: after



Blue: before; red: after

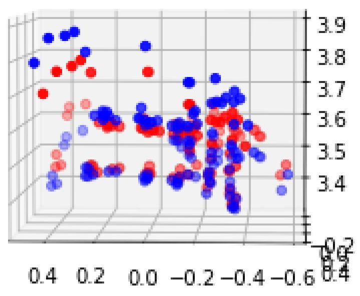


Error

```
140
Best Error 4481.788389671443
Before 4481.788389671592, After 10.884804732422543
```

Outliers(corresponding to error 10) can be seen in the following image

Blue: before; red: after



Code

```
[7]: from scipy.optimize import leastsq
def rodriguesResidual(K1, M1, p1, K2, p2, x):
    '''
    Q3.3: 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
    '''
    residuals = None
    # ----- TODO -----
    # YOUR CODE HERE
    # raise NotImplementedError()
    N = p1.shape[0]
    P = x[:-6].reshape((N,3))
    P = np.vstack((np.transpose(P), np.ones((1, N))))
    R2 = rodrigues(x[-6:-3].reshape((3,)))
    t2 = x[-3:].reshape((3,1))
    #print(t2.shape)
    M2 = np.hstack((R2, t2))

    C1 = K1 @ M1
    C2 = K2 @ M2

    p1_hat = C1 @ P
    p1_hat = p1_hat / p1_hat[2,:]
    p2_hat = C2 @ P
    p2_hat = p2_hat / p2_hat[2,:]
    p1_hat = p1_hat[0:2,:].T
    p2_hat = p2_hat[0:2,:].T
    residuals = np.concatenate([(p1-p1_hat).reshape([-1]), (p2-p2_hat).reshape([-1])])

    return residuals
```

```
def bundleAdjustment(K1, M1, p1, K2, M2_init, p2, P_init):
    '''
    Q3.3 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
                P2, the optimized 3D coordinates of points
                o1, the starting objective function value with the initial input
                o2, the ending objective function value after bundle adjustment

    ***
    Hints:
    (1) Use the scipy.optimize.minimize function to minimize the objective function, rodriguesResidual.
        You can try different (method='..') in scipy.optimize.minimize for best results.
    '''
    obj_start = obj_end = 0
    R2_init = M2_init[:, 0:3]
    t2_init = M2_init[:, 3]
    x_init = np.concatenate((P_init.flatten(), invRodrigues(R2_init).flatten(), t2_init.flatten()))
    #print(x_init.shape)

    def func(x): #K1, M1, p1, K2, p2,
        return ((rodriguesResidual(K1, M1, p1, K2, p2, x))**2).sum()

    obj_start = func(x_init) #K1, M1, p1, K2, p2,
    x_update = scipy.optimize.minimize(func,x_init,method = "CG").x#,args = (K1, M1, p1, K2, p2, x_init))
    obj_end = func(x_update)
    N = p1.shape[0]
    P = x_update[:-6].reshape((N,3))
    R2 = rodrigues(x_update[-6:-3].reshape((3,)))
    t2 = x_update[-3:].reshape((3,1))
    M2 = np.hstack((R2, t2))

    return M2, P, obj_start, obj_end
```

End

Discussed approach with Vishnu Mani Hema, Damanpreet Singh, Nitika Suresh

```
In [ ]:
```