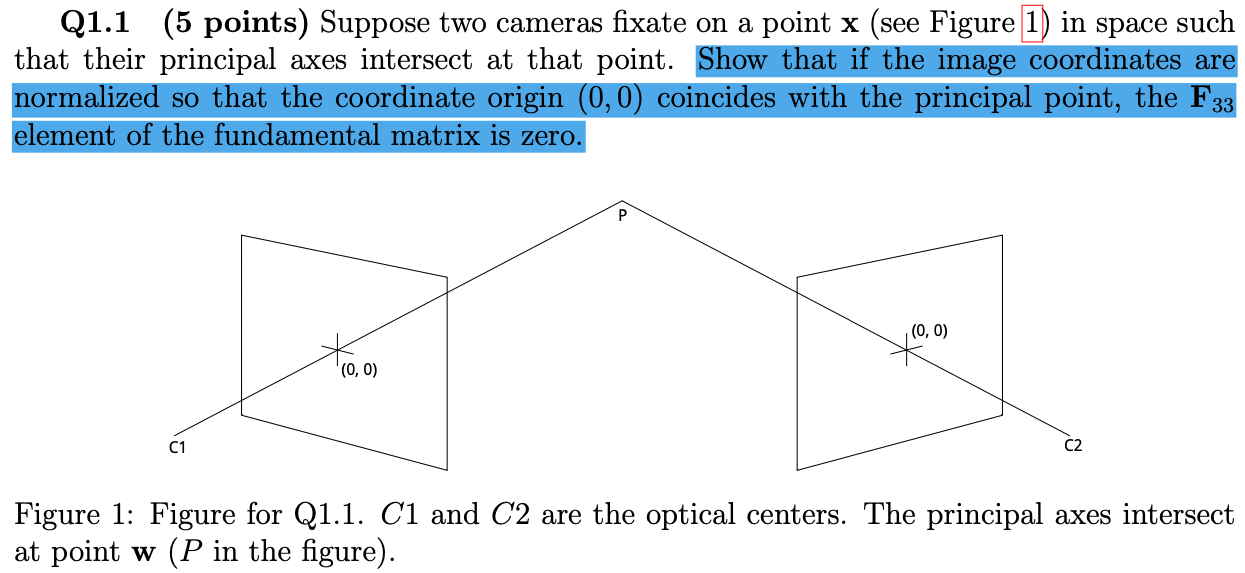
**16-720 Computer Vision: Homework 4 (Fall 2022)**

**3D Reconstruction**

Haejoon Lee

Epipolar constraint:



텍스트, 시계, 클립아트이(가) 표시된 사진

자동 생성된 설명



텍스트이(가) 표시된 사진

자동 생성된 설명

Pure translation -> R = I

Epipolar constraints:

텍스트, 시계이(가) 표시된 사진

자동 생성된 설명



Thus, if we fix single y, then the epipolar line is parallel to the x-axis.

테이블이(가) 표시된 사진

자동 생성된 설명











텍스트이(가) 표시된 사진

자동 생성된 설명

From the matrix D, we can assume the x-axis of the world coordinate is perpendicular to the mirror.



텍스트이(가) 표시된 사진

자동 생성된 설명

Thus, it is equivalent to having another camera whose matrix is K’.

Let’s assume that the camera coordinate’s axis is parallel to word coordinates.

Then, the virtual camera is only at the translated position as below:

Epipolar constraints:



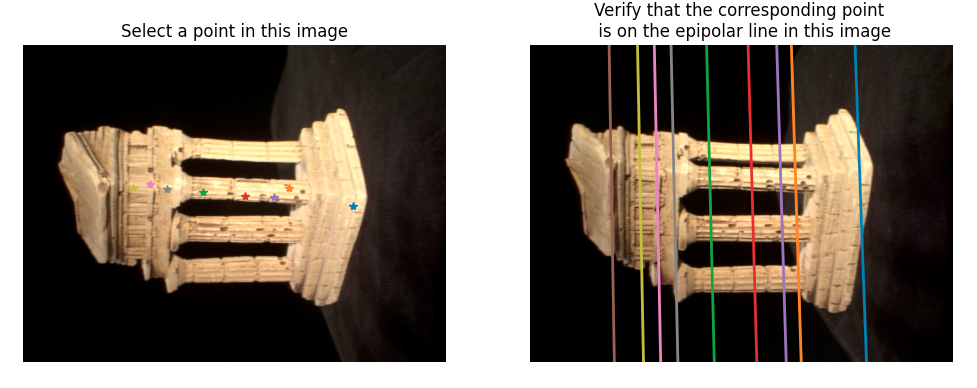
If  is skew-symmetric, then the fundamental matrix will be so.

텍스트, 시계이(가) 표시된 사진

자동 생성된 설명

Thus, the fundamental matrix is skew-symmetric.

텍스트이(가) 표시된 사진

자동 생성된 설명텍스트이(가) 표시된 사진

자동 생성된 설명

def eightpoint(pts1, pts2, M):

# Replace pass by your implementation

N = pts1.shape[0]

# Normalization

pts1, pts2 = pts1/float(M), pts2/float(M)

xcoords1, ycoords1 = pts1[:, 0], pts1[:, 1]

xcoords2, ycoords2 = pts2[:, 0], pts2[:, 1]

# A Matix

cul0 = xcoords2 \* xcoords1

cul1 = xcoords2 \* ycoords1

cul2 = xcoords2

cul3 = ycoords2 \* xcoords1

cul4 = ycoords2 \* ycoords1

cul5 = ycoords2

cul6 = xcoords1

cul7 = ycoords1

cul8 = np.ones((N,), dtype=np.float32)

A = np.stack((cul0, cul1, cul2, cul3, cul4, cul5, cul6, cul7, cul8), axis=1)

# solve a raw f

\_, \_, Vt = np.linalg.svd(A)

F\_vec = Vt[-1, :] #(9,)

F\_raw = F\_vec.reshape(3, 3)

#Refine F

F\_norm = \_singularize(F\_raw)

F\_norm = refineF(F\_norm, pts1, pts2) # ?

# Unscale

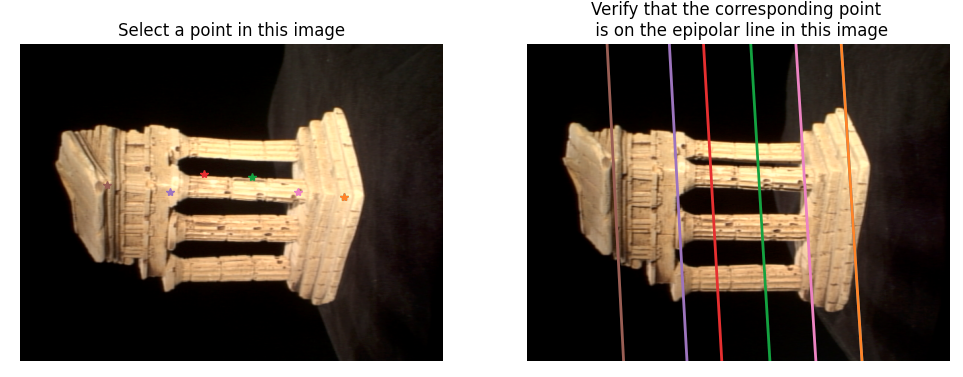
T = np.zeros((3, 3), dtype=np.float32)

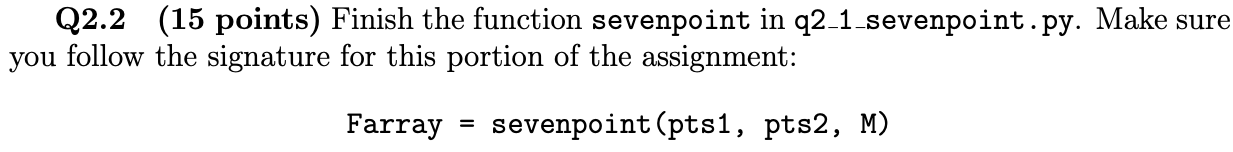
T[0, 0] = T[1, 1] = 1.0 / M

T[2, 2] = 1.0

F\_final = T.transpose() @ F\_norm @ T

return F\_final

텍스트이(가) 표시된 사진

자동 생성된 설명

def sevenpoint(pts1, pts2, M):

Farray = []

# ----- TODO -----

# YOUR CODE HERE

N = pts1.shape[0]

# Normalization

pts1, pts2 = pts1/float(M), pts2/float(M)

xcoords1, ycoords1 = pts1[:, 0], pts1[:, 1]

xcoords2, ycoords2 = pts2[:, 0], pts2[:, 1]

# A Matix

cul0 = xcoords2 \* xcoords1

cul1 = xcoords2 \* ycoords1

cul2 = xcoords2

cul3 = ycoords2 \* xcoords1

cul4 = ycoords2 \* ycoords1

cul5 = ycoords2

cul6 = xcoords1

cul7 = ycoords1

cul8 = np.ones((N,), dtype=np.float32)

A = np.stack((cul0, cul1, cul2, cul3, cul4, cul5, cul6, cul7, cul8), axis=1)

# Get F1 and F2

\_, \_, Vt = np.linalg.svd(A)

F1\_vec, F2\_vec = Vt[-1, :], Vt[-2, :] #(9,)

F1, F2 = F1\_vec.reshape(3, 3), F2\_vec.reshape(3, 3)

#Find the coefficients for F1 and F2 spanning the null space

a, b = F1-F2, F2

funct = lambda x: np.linalg.det(x\*a + b)

c0 = funct(0)

c1 = (2.0/3)\*(funct(1)-funct(-1)) - (1.0/12)\*(funct(2)-funct(-2))

c3 = (1.0/12)\*(funct(2) - funct(-2)) - (1.0/6)\*(funct(1)-funct(-1))

c2 = funct(1) - c0 - c1 - c3

#Solve the polynomial

roots = poly.polyroots([c0, c1, c2, c3])

# Unscale F

T = np.zeros((3, 3), dtype=np.float32)

T[0, 0] = T[1, 1] = 1.0 / M

T[2, 2] = 1.0

for root in roots:

F\_norm = root\*a + b

F\_norm = \_singularize(F\_norm)

# F\_norm = refineF(F\_norm, pts1, pts2)

F\_final = T.transpose() @ F\_norm @ T

Farray.append(F\_final)

return Farray

텍스트이(가) 표시된 사진

자동 생성된 설명

def essentialMatrix(F, K1, K2):

# Replace pass by your implementation

E = K2.transpose() @ F @ K1

return E

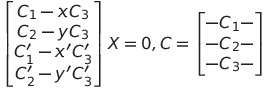
테이블이(가) 표시된 사진

자동 생성된 설명

Assume we have two points x and x’







텍스트이(가) 표시된 사진

자동 생성된 설명

def triangulate(C1, pts1, C2, pts2):

# Replace pass by your implementation

coord\_word\_list = []

err\_reproject = 0

for i in range(pts1.shape[0]):

x1, y1 = pts1[i, :]

x2, y2 = pts2[i, :]

# A matrix

A0 = C1[0, :] - x1\*C1[2, :]

A1 = C1[1, :] - y1\*C1[2, :]

A2 = C2[0, :] - x2\*C2[2, :]

A3 = C2[1, :] - y2\*C2[2, :]

A = np.stack((A0, A1, A2, A3), axis=0)

#Get world coordinates through SVD

\_, \_, Vt = np.linalg.svd(A)

coord\_word = Vt[-1, :] #(4,)

coord\_word = coord\_word[0:3] / coord\_word[3] #(3,)

coord\_word\_list.append(coord\_word)

#Reprojection

coord\_word\_homo = np.zeros((4, 1), dtype=np.float32)

coord\_word\_homo[0:3, 0] = coord\_word

coord\_word\_homo[3, 0] = 1

coord\_cam1\_rep = C1 @ coord\_word\_homo

coord\_cam2\_rep = C2 @ coord\_word\_homo

#Calculate reprojection error

x1\_cam1\_rep, y1\_cam1\_rep = coord\_cam1\_rep[0:2, 0] / coord\_cam1\_rep[2, 0]

x2\_cam2\_rep, y2\_cam2\_rep = coord\_cam2\_rep[0:2, 0] / coord\_cam2\_rep[2, 0]

err\_reproject += (x1\_cam1\_rep-x1)\*\*2 + (y1\_cam1\_rep-y1)\*\*2 + (x2\_cam2\_rep-x2)\*\*2 + (y2\_cam2\_rep-y2)\*\*2

P = np.stack(coord\_word\_list, axis=0)

return P, err\_reproject

def findM2(F, pts1, pts2, intrinsics, filename = 'q3\_3.npz'):

'''

Q2.2: Function to find the camera2's projective matrix given correspondences

Input: F, the pre-computed fundamental matrix

pts1, the Nx2 matrix with the 2D image coordinates per row

pts2, the Nx2 matrix with the 2D image coordinates per row

intrinsics, the intrinsics of the cameras, load from the .npz file

filename, the filename to store results

Output: [M2, C2, P] the computed M2 (3x4) camera projective matrix, C2 (3x4) K2 \* M2, and the 3D points P (Nx3)

\*\*\*

Hints:

(1) Loop through the 'M2s' and use triangulate to calculate the 3D points and projection error. Keep track

of the projection error through best\_error and retain the best one.

(2) Remember to take a look at camera2 to see how to correctly reterive the M2 matrix from 'M2s'.

'''

K1, K2 = intrinsics['K1'], intrinsics['K2']

E = essentialMatrix(F, K1, K2)

M2s = camera2(E)

# Assume camera coordinates = word coordinates

M1 = np.zeros((3, 4), dtype=np.float32)

M1[0,0] = 1

M1[1,1] = 1

M1[2,2] = 1

C1 = K1 @ M1

# Find C2, P for each M2

P\_list = []

C2\_list = []

err\_list = []

for i in range(M2s.shape[2]):

M2 = M2s[:, :, i]

C2 = K2 @ M2

C2\_list.append(C2)

P, err\_rep = triangulate(C1, pts1, C2, pts2)

P\_list.append(P)

err\_list.append(err\_rep)

#Remain best M2

min\_idx = np.argmin(np.abs(np.array(err\_list)))

# print(min\_idx)

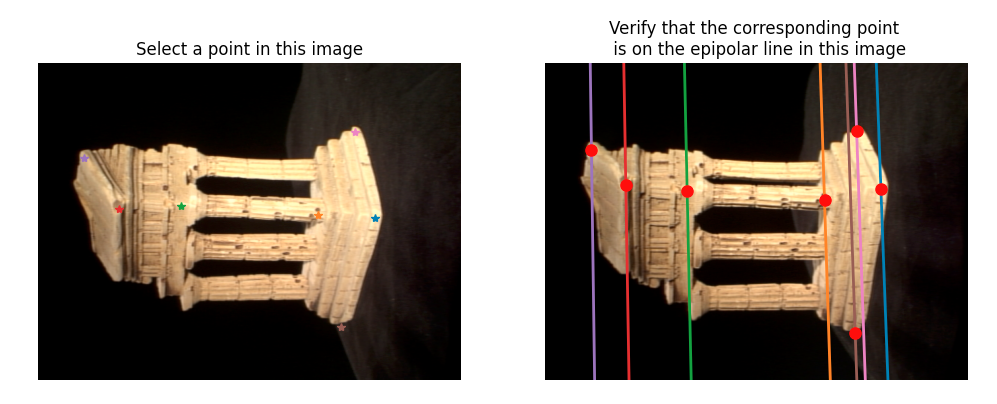
M2 = M2s[:, :, min\_idx]

P = P\_list[min\_idx]

C2 = C2\_list[min\_idx]

return M2, C2, P

테이블이(가) 표시된 사진

자동 생성된 설명

def KernelResponse(im, x, y, kxs, kys):

H, W = im.shape[0:2]

xs, ys = kxs + x, kys + y

xs, ys = np.clip(xs, 0, W-1).astype(np.int32), np.clip(ys, 0, H-1).astype(np.int32)

response = im[ys, xs, :]

return response

def epipolarCorrespondence(im1, im2, F, x1, y1):

coord\_img1\_homo = np.array([[x1], [y1], [1]], dtype=np.float32)

# Calculate epipolar line on im2

epipolarline\_img2 = F @ coord\_img1\_homo

# Set search boundary

r = 100

ll = np.array([1, 0, -(x1-r)], dtype=np.float32).reshape(-1, 1)

lt = np.array([0, 1, -(y1-r)], dtype=np.float32).reshape(-1, 1)

lr = np.array([1, 0, -(x1+r)], dtype=np.float32).reshape(-1, 1)

lb = np.array([0, 1, -(y1+r)], dtype=np.float32).reshape(-1, 1)

lines\_boundary = [ll, lt, lr, lb]

# Get intersected points between search boundary and epipolar line

points\_intersect = [np.cross(l.reshape(-1), epipolarline\_img2.reshape(-1)).reshape(-1, 1) for l in lines\_boundary]

points\_end = []

for points in points\_intersect:

if np.abs(points[2, 0]) > 0.0000001:

points\_homo = points / points[2, 0]

dist\_max = np.max(np.abs(points\_homo[0:2, :]-coord\_img1\_homo[0:2, :]))

if dist\_max < r\*(1.1):

points\_end.append(points\_homo)

search\_begin = None

search\_end = None

if len(points\_end) > 2:

point0\_end = points\_end[0]

for i in range(1, len(points\_end)):

coord\_img1\_homo = points\_end[i]

if np.min(np.abs(point0\_end[0:2, :]-coord\_img1\_homo[0:2, :])) > r\*0.1:

search\_begin = point0\_end[0:2, :]

search\_end = coord\_img1\_homo[0:2, :]

break

else:

search\_begin = points\_end[0][0:2, :]

search\_end = points\_end[1][0:2, :]

# Generate Gaussian weighting kernel

kernel\_r = 20

kernel\_xaxis = np.arange(-kernel\_r, kernel\_r+1, 1.0)

kernel\_yaxis = np.arange(-kernel\_r, kernel\_r+1, 1.0)

kernel\_xaxis, kernel\_yaxis = np.meshgrid(kernel\_xaxis, kernel\_yaxis)

kernel\_xaxis, kernel\_yaxis = kernel\_xaxis.reshape(-1), kernel\_yaxis.reshape(-1)

STD = kernel\_r / 2.0

kernel\_window = np.exp( -( (kernel\_xaxis\*\*2 + kernel\_yaxis\*\*2)/(2\*STD\*\*2) ) ) / (2\*np.pi\*STD\*\*2)

response\_img1 = KernelResponse(im1, x1, y1, kernel\_xaxis, kernel\_yaxis)

#Calculate distances by compare kernel response and get the best matching point on im2

dist\_min = np.Inf

num\_steps = int(np.sum((search\_begin-search\_end)\*\*2)\*\*0.5)

x\_step = (search\_end[0, 0] - search\_begin[0, 0]) / num\_steps

y\_step = (search\_end[1, 0] - search\_begin[1, 0]) / num\_steps

x2, y2 = search\_begin[0, 0], search\_begin[1, 0]

x2\_best, y2\_best = -1, -1

for i in range(num\_steps):

response\_img2 = KernelResponse(im2, x2, y2, kernel\_xaxis, kernel\_yaxis)

dist = np.sum((response\_img2 - response\_img1)\*\*2, axis=1)\*\*0.5 # (Nk,)

dist = np.sum(dist\*kernel\_window)

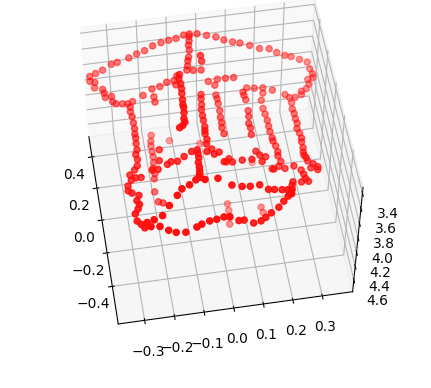
if dist < dist\_min:

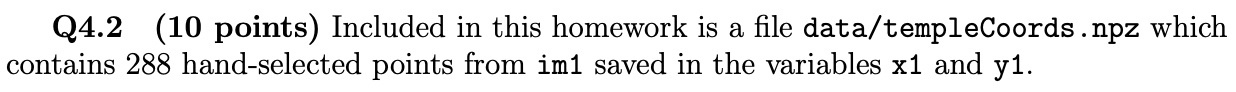
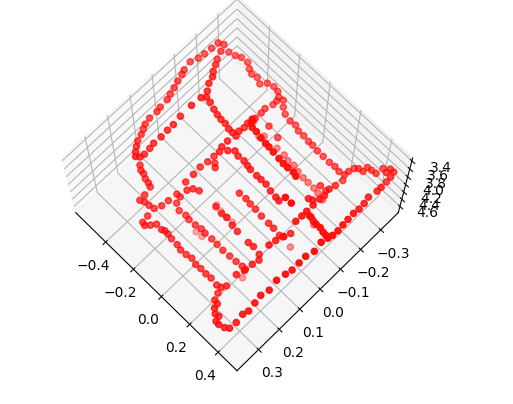
dist\_min = dist

x2\_best, y2\_best = x2, y2

x2, y2 = x2 + x\_step, y2 + y\_step

return x2\_best, y2\_best





def compute3D\_pts(temple\_pts1, intrinsics, F, im1, im2):

# ----- TODO -----

# YOUR CODE HERE

coords\_x1 = temple\_pts1['x1']

coords\_y1 = temple\_pts1['y1']

coords\_x2 = coords\_y2 = []

# Find x2, y2

for i in range(coords\_x1.shape[0]):

x1, y1 = coords\_x1[i, 0], coords\_y1[i, 0]

x2, y2 = epipolarCorrespondence(im1, im2, F, x1, y1)

coords\_x2.append(x2)

coords\_y2.append(y2)

coords\_x2, coords\_y2 = np.array(coords\_x2).reshape(-1, 1), np.array(coords\_y2).reshape(-1, 1)

pts1 = np.concatenate((coords\_x1, coords\_y1), axis=1)

pts2 = np.concatenate((coords\_x2, coords\_y2), axis=1)

K1, K2 = intrinsics['K1'], intrinsics['K2']

E = essentialMatrix(F, K1, K2)

M2s = camera2(E)

# Assume camera coordinates = word coordinates

M1 = np.zeros((3, 4), dtype=np.float32)

M1[0,0] = M1[1,1] = M1[2,2] = 1

C1 = K1 @ M1

# Find C2, P for each M2

P\_list = []

C2\_list = []

err\_list = []

for i in range(M2s.shape[2]):

M2 = M2s[:, :, i]

C2 = K2 @ M2

C2\_list.append(C2)

P, err\_rep = triangulate(C1, pts1, C2, pts2)

P\_list.append(P)

print('Reprojection error of M2\_%d: %f' % (i, err\_rep))

err\_list.append(err\_rep)

#Remain best M2

# min\_idx = np.argmin(np.abs(np.array(err\_list)))

min\_idx = 2

print(min\_idx)

M2 = M2s[:, :, min\_idx]

P = P\_list[min\_idx]

C2 = C2\_list[min\_idx]

np.savez('q4\_2.npz', F=F, M1=M1, M2=M2, C1=C1, C2=C2)

return P

텍스트, 창문이(가) 표시된 사진

자동 생성된 설명텍스트이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명

Using sevenpoint algorithm, the F was computed and then multiplied with pts2 to get epipolar line on im2. Then, the distance between the line and pts1 was computed, and inliers were selected with smaller distance than ‘tol’. Using the inliers, F was computed again. Above process went interatively. By increasing ‘nIters’, we can go more iteration and get more accurate F for the noisy points with increased computation time. By increasing tol, we can include more candidates for inliers.

def dist\_to\_epipolarline(pts1\_homo\_t, pts2\_homo\_t, F):

epipolar\_lines = (pts2\_homo\_t @ F)

# Calculate the distance between pts1 and epipolar line of pt2 on im1

dist = np.abs(np.sum(epipolar\_lines \* pts1\_homo\_t, axis=1)) / (epipolar\_lines[:, 0]\*\*2 + epipolar\_lines[:, 1]\*\*2)\*\*0.5

return np.abs(dist)

def ransacF(pts1, pts2, M, nIters=1000, tol=2.0):

# Replace pass by your implementation

N = pts1.shape[0]

pts1\_homo\_t = np.concatenate((pts1, np.ones((N, 1))), axis=1)

pts2\_homo\_t = np.concatenate((pts2, np.ones((N, 1))), axis=1)

max\_inlier\_num = 0

best\_inlier\_idx = None

for i in range(nIters):

idx\_selected = np.random.choice(N, 7, replace=False)

pts1\_selected = pts1[idx\_selected, :]

pts2\_selected = pts2[idx\_selected, :]

#Try sevenpoint algorithm

Farray = sevenpoint(pts1\_selected, pts2\_selected, M)

for F in Farray:

dist\_epipolarline2\_pts1 = dist\_to\_epipolarline(pts1\_homo\_t, pts2\_homo\_t, F)

inlier\_idx = np.where(np.abs(dist\_epipolarline2\_pts1) < tol)[0]

if inlier\_idx.shape[0] > max\_inlier\_num:

max\_inlier\_num = inlier\_idx.shape[0]

best\_inlier\_idx = inlier\_idx

print('Inlier number: {}'.format(max\_inlier\_num))

if i == 0:

dist\_epipolarline2\_pts1 = dist\_to\_epipolarline(pts1\_homo\_t, pts2\_homo\_t, F)

print("Distances before ransac: {0}".format(np.abs(dist\_epipolarline2\_pts1)))

#Run sevenpoint algorithm with the best inliers

pts1\_best, pts2\_best = pts1[best\_inlier\_idx, :], pts2[best\_inlier\_idx, :]

Farray = sevenpoint(pts1\_best, pts2\_best, M)

#Find best F maximizing inliers within Farray

F\_best = None

max\_inlier\_num = 0

for F in Farray:

epipolar\_lines = (pts2\_homo\_t @ F)

dist\_epipolarline2\_pts1 = dist\_to\_epipolarline(pts1\_homo\_t, pts2\_homo\_t, F)

inlier\_idx = np.where(np.abs(dist\_epipolarline2\_pts1) < 2.0)[0]

if inlier\_idx.shape[0] > max\_inlier\_num:

max\_inlier\_num = inlier\_idx.shape[0]

F\_best = F

dist\_epipolarline2\_pts1 = dist\_to\_epipolarline(pts1\_homo\_t, pts2\_homo\_t, F)

print("Distances after ransac: {0}".format(np.abs(dist\_epipolarline2\_pts1)))

return F\_best, best\_inlier\_idx

텍스트이(가) 표시된 사진

자동 생성된 설명

'''

Q5.2: Rodrigues formula.

Input: r, a 3x1 vector

Output: R, a rotation matrix

'''

def rodrigues(r):

# Replace pass by your implementation

eps = 0.001

theta = np.sum(r\*\*2)\*\*0.5

if np.abs(theta) < eps:

return np.eye(3, dtype=np.float32)

else:

u = r / theta

u1, u2, u3 = u[0, 0], u[1, 0], u[2, 0]

# u1, u2, u3 = u[0], u[1], u[2]

u\_cross = np.array([[0, -u3, u2], [u3, 0, -u1], [-u2, u1, 0]], dtype=np.float32)

R = np.eye(3, dtype=np.float32) \* np.cos(theta) \

+ (1 - np.cos(theta)) \* (u @ u.transpose()) \

+ u\_cross \* np.sin(theta)

return R

'''

Q5.2: Inverse Rodrigues formula.

Input: R, a rotation matrix

Output: r, a 3x1 vector

'''

## additionally defined functions

def eq(a, b):

eps = 0.001

return np.abs(a - b) < eps

def gt(a, b):

eps = 0.001

return a - b > eps

def S\_half(r):

length = np.sum(r\*\*2)\*\*0.5

r1, r2, r3 = r[0, 0], r[1, 0], r[2, 0]

if (eq(length, np.pi) and eq(r1, r2) and eq(r1, 0) and gt(0, r3)) \

or (eq(r1, 0) and gt(0, r2)) \

or (gt(0, r1)):

return -r

else:

return r

def arctan2(y, x):

if gt(x, 0):

return np.arctan(y / x)

elif gt(0, x):

return np.pi + np.arctan(y / x)

elif eq(x, 0) and gt(y, 0):

return np.pi\*0.5

elif eq(x, 0) and gt(0, y):

return -np.pi\*0.5

def invRodrigues(R):

# Replace pass by your implementation

eps = 0.001

A = (R - R.transpose())\*0.5

a32, a13, a21 = A[2, 1], A[0, 2], A[1, 0]

rho = np.array([[a32], [a13], [a21]], dtype=np.float32)

s = np.sum(rho\*\*2)\*\*0.5

c = (R[0, 0]+R[1, 1]+R[2, 2] - 1) / 2.0

if eq(s, 0) and eq(c, 1):

return np.zeros((3, 1), dtype=np.float32)

elif eq(s, 0) and eq(c, -1):

V = R+np.eye(3, dtype=np.float32)

# find a nonzero column of V

mark = np.where(np.sum(V\*\*2, axis=0) > eps)[0]

v = V[:, mark[0]]

u = v / (np.sum(v\*\*2)\*\*0.5)

r = S\_half(u\*np.pi)

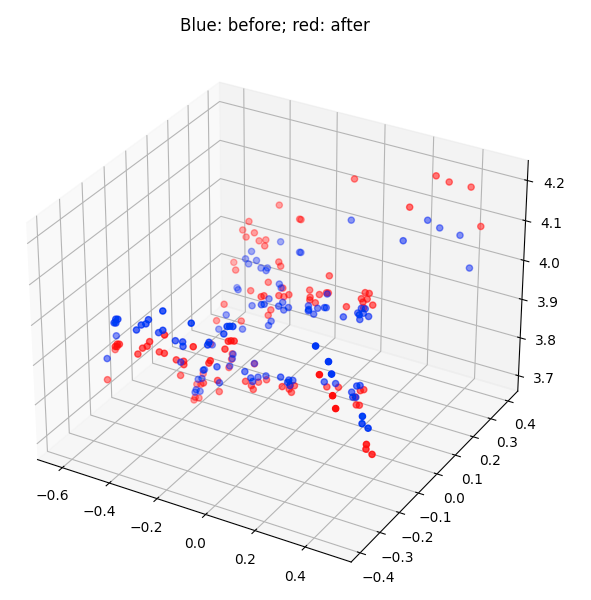
return r

elif not eq(s, 0):

u = rho / s

theta = arctan2(s, c)

return u\*theta







'''

Q5.3: 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

'''

## additionally defined functions

def flatten(P, r2, t2):

# P: (N, 3)

# r2: (3, 1)

# t2: (3, 1)

# (3+3+N\*3,)

return np.concatenate((r2.reshape(-1), t2.reshape(-1), P.reshape(-1)), axis=0)

def inflate(x):

r2 = x[0:3].reshape(-1, 1)

t2 = x[3:6].reshape(-1, 1)

P = x[6:].reshape(-1, 3)

return P, r2, t2

#Addi functions

def rodriguesResidual(K1, M1, p1, K2, p2, x):

# Replace pass by your implementation

P, r2, t2 = inflate(x)

R2 = rodrigues(r2)

M2 = np.concatenate((R2, t2), axis=1)

points\_word\_homo = np.concatenate( ( P, np.ones( (P.shape[0], 1) ) ), axis=1 ).transpose()

points\_img1\_rep\_homo = K1 @ M1 @ points\_word\_homo

points\_img1\_rep = points\_img1\_rep\_homo[0:2, :] / points\_img1\_rep\_homo[2, :]

points\_img2\_rep\_homo = K2 @ M2 @ points\_word\_homo

points\_img2\_rep = points\_img2\_rep\_homo[0:2, :] / points\_img2\_rep\_homo[2, :]

error\_img1\_rep = (p1 - points\_img1\_rep).reshape(-1)

error\_img2\_rep = (p2 - points\_img2\_rep).reshape(-1)

residuals = np.concatenate((error\_img1\_rep, error\_img2\_rep), axis=0)

return residuals

'''

Q5.3 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.

'''

def bundleAdjustment(K1, M1, p1, K2, M2\_init, p2, P\_init):

# Replace pass by your implementation

obj\_start = obj\_end = 0

# ----- TODO -----

# YOUR CODE HERE

p1 = p1.transpose()

p2 = p2.transpose()

residual = lambda x: rodriguesResidual(K1, M1, p1, K2, p2, x)

R2\_init = M2\_init[:, 0:3]

t2\_init = M2\_init[:, 3]

r2\_init = invRodrigues(R2\_init)

x\_init = flatten(P\_init, r2\_init, t2\_init)

x\_optim, \_ = scipy.optimize.leastsq(residual, x\_init)

# print('Reprojection error after BA: %f' % np.sum(residual(x\_optim)\*\*2))

P2, r2, t2 = inflate(x\_optim)

R2 = rodrigues(r2)

M2 = np.concatenate((R2, t2), axis=1)

obj\_start = rodriguesResidual(K1, M1, p1, K2, p2, x\_init)

obj\_end = rodriguesResidual(K1, M1, p1, K2, p2, x\_optim)

# print('Object start: {}'.format(obj\_start))

# print('Object end: {}'.format(obj\_end))

return M2, P2, obj\_start, obj\_end

# raise NotImplementedError()

# return M2, P, obj\_start, obj\_start

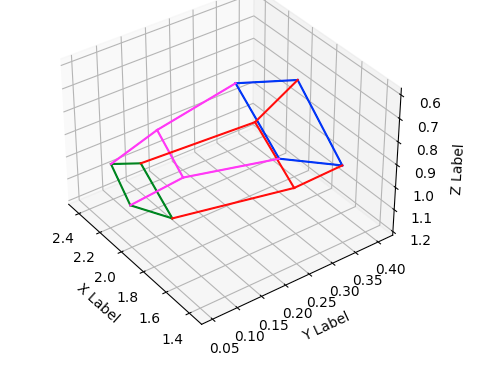
텍스트이(가) 표시된 사진

자동 생성된 설명

I modified triangulate function to generate A matrix with 3 points and compute word coordinates according to it as follow:

텍스트이(가) 표시된 사진

자동 생성된 설명



When only one of points’ confidence score is smaller than threshold, I just computed 3D coordinates using the other two points.

def triangulate\_3pts(C1, pts1, C2, pts2, C3, pts3):

# Replace pass by your implementation

coord\_word\_list = []

err\_reproject = 0

for i in range(pts1.shape[0]):

x1, y1 = pts1[i, :]

x2, y2 = pts2[i, :]

x3, y3 = pts3[i, :]

# A matrix

A0 = C1[0, :] - x1\*C1[2, :]

A1 = C1[1, :] - y1\*C1[2, :]

A2 = C2[0, :] - x2\*C2[2, :]

A3 = C2[1, :] - y2\*C2[2, :]

A4 = C3[0, :] - x3\*C3[2, :]

A5 = C3[1, :] - y3\*C3[2, :]

A = np.stack((A0, A1, A2, A3, A4, A5), axis=0)

#Get world coordinates through SVD

U, s, Vt = np.linalg.svd(A)

coord\_word = Vt[-1, :] #(4,)

coord\_word = coord\_word[0:3] / coord\_word[3] #(3,)

coord\_word\_list.append(coord\_word)

#Ceprojection

coord\_word\_homo = np.zeros((4, 1), dtype=np.float32)

coord\_word\_homo[0:3, 0] = coord\_word

coord\_word\_homo[3, 0] = 1

coord\_cam1\_rep = C1 @ coord\_word\_homo

coord\_cam2\_rep = C2 @ coord\_word\_homo

coord\_cam3\_rep = C3 @ coord\_word\_homo

#Calculate reprojection error

x1\_cam1\_rep, y1\_cam1\_rep = coord\_cam1\_rep[0:2, 0] / coord\_cam1\_rep[2, 0]

x2\_cam2\_rep, y2\_cam2\_rep = coord\_cam2\_rep[0:2, 0] / coord\_cam2\_rep[2, 0]

x3\_cam3\_rep, y3\_cam3\_rep = coord\_cam3\_rep[0:2, 0] / coord\_cam3\_rep[2, 0]

err\_reproject += (x1\_cam1\_rep-x1)\*\*2 + (y1\_cam1\_rep-y1)\*\*2 + (x2\_cam2\_rep-x2)\*\*2 + (y2\_cam2\_rep-y2)\*\*2 + (x3\_cam3\_rep-x3)\*\*2 + (y3\_cam3\_rep-y3)\*\*2

P = np.stack(coord\_word\_list, axis=0)

print(P.shape)

return P, err\_reproject

def MultiviewReconstruction(C1, pts1, C2, pts2, C3, pts3, Thres = 300):

p1s, p2s, p3s = pts1[:,:2], pts2[:,:2], pts3[:,:2]

confidence1s, confidence2s, confidence3s = pts1[:,2], pts2[:,2], pts3[:,2]

P, err = triangulate\_3pts(C1, p1s, C2, p2s ,C3, p3s)

P\_12, \_ = triangulate(C1, p1s, C2, p2s)

P\_23, \_ = triangulate(C2, p2s, C3, p3s)

P\_31, \_ = triangulate(C3, p3s, C1, p1s)

# If only a point's confidence score is smaller than threshold, don't consider it and get P from triangulation using the other two points

N\_pts = len(pts1)

for i in range(N\_pts):

if confidence1s[i] < Thres and confidence2s[i] > Thres and confidence3s[i] > Thres:

P[i] = P\_23[i]

if confidence2s[i] < Thres and confidence3s[i] > Thres and confidence1s[i] > Thres:

P[i] = P\_31[i]

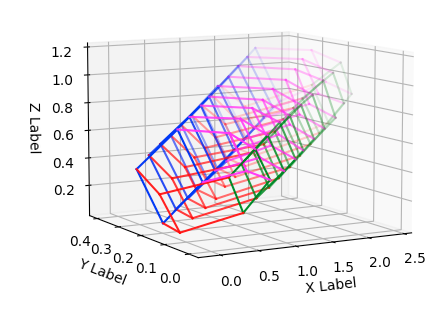
if confidence3s[i] < Thres and confidence1s[i] > Thres and confidence2s[i] > Thres:

P[i] = P\_12[i]

return P, err

텍스트이(가) 표시된 사진

자동 생성된 설명



def plot\_3d\_keypoint\_video(pts\_3d\_video):

fig = plt.figure()

ax = fig.add\_subplot(111, projection='3d')

for i in range(10):

pts\_word = pts\_3d\_video[i]

for j in range(len(connections\_3d)):

index0, index1 = connections\_3d[j]

xline = [pts\_word[index0,0], pts\_word[index1,0]]

yline = [pts\_word[index0,1], pts\_word[index1,1]]

zline = [pts\_word[index0,2], pts\_word[index1,2]]

ax.plot(xline, yline, zline, color = colors[j], alpha = 0.1 \* i)

np.set\_printoptions(threshold = 1e6, suppress = True)

ax.set\_xlabel('X Label')

ax.set\_ylabel('Y Label')

ax.set\_zlabel('Z Label')

plt.show()