hw9

November 3, 2025

[1]:
```
import numpy as np
```

Q1: Load the 100 pairs of corresponding 2-D and 3-D points in the files 2Dpoints.txt and 3Dpoints.txt (the ith row of both files corresponds to the ith point). Use these point correspondences to solve (using Eigen-analysis) for the camera matrix P (whose rasterized vector p has a unit L2 norm). [5 pts]

[3]:
```
#Q1

# Load 2D and 3D point correspondences
pts_2D = np.loadtxt('2Dpoints.txt')   # shape (N, 2)
pts_3D = np.loadtxt('3Dpoints.txt')   # shape (N, 3)
N = pts_2D.shape[0]

# Build matrix A as shown in slides (two rows per point)
A = np.zeros((2 * N, 12))
for i in range(N):
    X, Y, Z = pts_3D[i]
    x, y = pts_2D[i]
    A[2*i]   = [X, Y, Z, 1, 0, 0, 0, 0, -x*X, -x*Y, -x*Z, -x]
    A[2*i+1] = [0, 0, 0, 0, X, Y, Z, 1, -y*X, -y*Y, -y*Z, -y]

# Instead of A A eigenanalysis, use SVD (mathematically equivalent but stable)
U, S, Vt = np.linalg.svd(A)
p = Vt[-1]                    # Last row of V  = eigenvector of smallest␣
 ↪singular value
p = p / np.linalg.norm(p)     # Normalize to unit L2 norm (per HW)

# Reshape to camera matrix
P = p.reshape(3, 4)

print("Camera matrix P:")
print(P)
```

```
Camera matrix P:
[[ 2.12332031e-03  2.03512642e-03  1.66863834e-03 -9.12324263e-01]
 [-6.18904138e-04  1.38205976e-04  2.75868874e-03 -4.09444188e-01]
 [ 4.24761311e-06 -2.12814226e-06  3.68283111e-06 -6.69695134e-04]]
```

Discussion: The computed camera matrix P correctly models the projection from 3-D world coordinates to 2-D image coordinates using homogeneous transformations.

Q2: Given the computed matrix P (from Problem 1), project the 3-D homogeneous points (Xi,Yi,Zi,1) to 2-D. Compute the sum-of-squared error (sum-of-squared distances) between the resulting 3-D-to-2-D projected points and the given 2-D points (ensure all 2-D points are inhomogeneous). [3 pts]

[4]:
```
# Append 1 to 3D points to make them homogeneous
pts_3D_h = np.hstack([pts_3D, np.ones((pts_3D.shape[0], 1))])

# Project 3D points to 2D using the computed camera matrix P
proj = (P @ pts_3D_h.T).T

# Convert back to inhomogeneous coordinates
proj_2D = proj[:, :2] / proj[:, [2]]

# Compute sum-of-squared error (SSE)
errors = np.sum((pts_2D - proj_2D)**2, axis=1)
SSE = np.sum(errors)

print("Sum of squared projection error:", SSE)
```

```
Sum of squared projection error: 18.74609121336128
```

Discussion: Reprojecting the 3-D points through P gives a small total squared error (SSE 18.75), showing good agreement between the measured and predicted 2-D points.