1 Introduction

1.1 Matrices in imaging

Matrices are frequently encountered in imaging problems viewed from the discrete domain. A matrix could represent a digital image, a collection of measurements, or an operator. For example, many imaging systems can be modeled as a linear mapping of an input vector $\mathbf{x}$ to an output vector $\mathbf{y}$ through a matrix–vector multiplication with a matrix $\mathbf{A}$ as in Eq. (1). If $\mathbf{A}$ is an $n \times m$ matrix, it maps a vector of m dimensions (or $\mathbf{x}\ \varepsilon\ R^m$) into an n-dimensional vector:

$$\mathbf{y} = \mathbf{A}\mathbf{x}. \tag{1}$$

Imaging problems can be classified into two general types: direct and inverse. In the direct problem, $\mathbf{A}$ and $\mathbf{x}$ are known, and $\mathbf{y}$ must be calculated. In the usually harder inverse problems, $\mathbf{A}$ and $\mathbf{y}$ are known and $\mathbf{x}$ must be estimated, or $\mathbf{y}$ and $\mathbf{x}$ are known, and $\mathbf{A}$ must be estimated. Typically, the estimates are found using least squares. The singular value decomposition (SVD) helps in all these imaging problems.

1.2 Singular value decomposition

The SVD states that any matrix $\mathbf{A}$, with m rows and n columns, can be decomposed into the product of three matrices: $\mathbf{U}$, $\mathbf{D}$, and $\mathbf{V}^{\mathbf{T}}$.[1] Five mathematicians were primarily responsible in developing the SVD in the 1800s:[2]

$$\mathbf{A} = \mathbf{U}\mathbf{D}\mathbf{V}^{\mathbf{T}} = \sum_{\mathbf{i}=1}^{\mathbf{r}} \sigma_{\mathbf{i}}\mathbf{u}_{\mathbf{i}}\mathbf{v}_{\mathbf{i}}^{\mathbf{T}}. \tag{2}$$

Matrices $\mathbf{U}$ and $\mathbf{V}$ are orthogonal (their inverse is equal to their transpose) and are of size $m \times m$ and $n \times n$, respectively. Matrix $\mathbf{D}$ is of size $m \times n$ and has r nonzero values only along the main diagonal as shown in Fig. 1. These are called singular values and represented by σ_i. Their number r is also the rank of the matrix. Their number gives the number of linearly independent columns or rows. They are arranged in decreasing order along the diagonal with $\sigma_{i1} > \ldots > \sigma_{ir} > 0$. This formulation is known as the full SVD. The reduced SVD truncates the

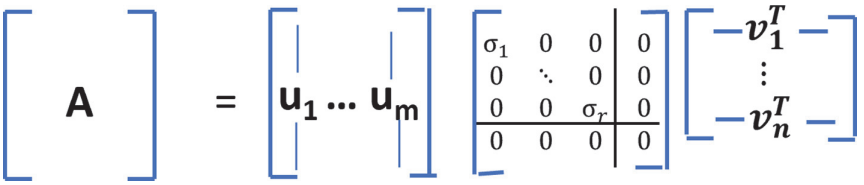

Figure 1 SVD of matrix A.

$$\mathbf{A} \approx \sigma_1 \mathbf{u}_1 \mathbf{v}_1^T + \sigma_2 \mathbf{u}_2 \mathbf{v}_2^{\mathrm{T}}. \tag{14}$$

The Eckart–Young theorem states that this is the best rank-2 approximation to matrix **A,** and in general, using k components to approximate **A** will produce the best rank-k approximation.[1] The error in this approximation can be seen to come from the sum of the smallest eigenvalues λ_n of $\mathbf{AA}^{\mathbf{T}}$. If $\mathbf{A}_k$ refers to the rank-k approximation of **A,** then the reconstruction error is

$$\|\mathbf{A} - \mathbf{A}_k\|^2 = \left(\sum_{n=k+1}^{r} \sigma_n u_n v_n^T \right)^2 = \sum_{n=k+1}^{r} \lambda_n. \tag{15}$$

1.3 Applications to imaging problems

From the above discussion in approximating a matrix, an application of the SVD is in compression. If matrix **A** represents an image, it may be compressed by calculating the SVD of **A** and retaining the most dominant components. For example, if the image is $m \times n$ pixels and each pixel is represented with b bits, then the whole image takes bmn bits. In comparison, keeping only the first set of eigenvectors $\mathbf{u}_1$ and $\mathbf{v}_1$ would require $b(m+n)$ bits (assuming a real number in a vector is represented with b bits). If the image was square ($m = n$), then it would take $n/2$ sets of eigenvectors to use the same space as the original image. Figure 4 shows a sample image on the left and a compressed version on the right using the SVD. The figure on the right was reconstructed using 10 pairs of vectors out of 96 possible values (image is 96 × 150 pixels). The approximation is reasonably good for the purposes of recognizing the scene as a classroom. In some cases, these basis vectors may serve to compress other images. Figure 5 shows a plot of the singular values showing that a few have the largest values.

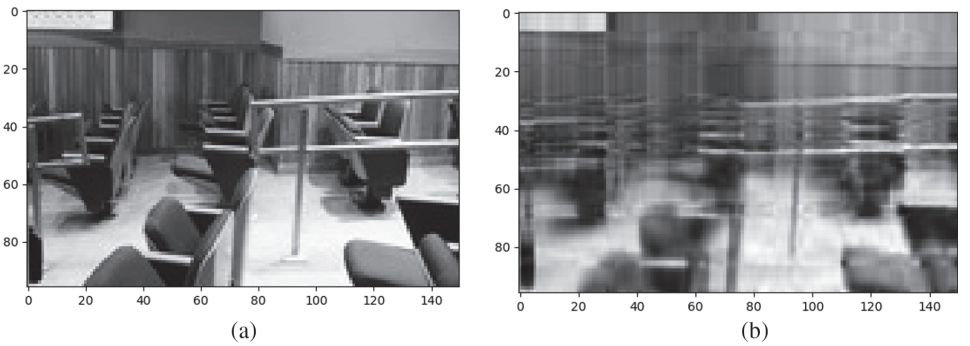

Figure 4 (b) The SVD reconstruction of the image on the (a) using 10 pairs of vectors. (a) Classroom image courtesy of Ref. 4.

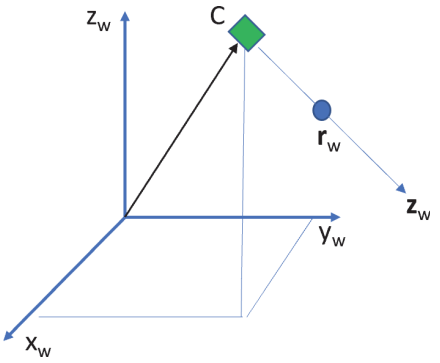

Figure 7 Camera C and 3D point **r** with respect to a world coordinate frame.

Equation (20) expresses the transformation assuming the camera is at the origin. In general, the camera may be displaced from the origin and have a different orientation from a general reference frame known as the world coordinate frame as shown in Fig. 7. A rotation and a translation are needed to first transform from world coordinates to camera coordinates. Homogeneous coordinates allow this transformation to be expressed as a matrix multiplication as shown in Eq. (21). The rotation and translation form the camera's extrinsic parameters. The final matrix **H** is 3 × 4 and has 11 degrees of freedom:

$$w\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} af & \mathrm{s} & c_x \\ 0 & f & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_c \\ r_{21} & r_{22} & r_{23} & t_c \\ r_{31} & r_{32} & r_{33} & t_c \end{bmatrix} \begin{bmatrix} x_w \\ y_w \\ z_w \\ 1 \end{bmatrix}, \tag{21}$$

$$\mathbf{p} = \mathbf{K}[\mathbf{R}|\mathbf{t}]\mathbf{r} = \mathbf{H}\mathbf{r},$$
$$\mathbf{t} = -\mathbf{R}\mathbf{c}.$$

2.2 Direct linear transform method

Frequently, the parameters of the camera matrix are either not known or not known with enough accuracy for the application. Camera calibration is the task of estimating these parameters. The direct linear transform (DLT) method consists of identifying the corresponding points in 3D and 2D, with known coordinates and uses them to set up a linear system of equations for the unknown camera matrix parameters. Figure 8 shows a typical checkerboard calibration pattern. The dimensions of the squares in 3D are known. The corner points of the squares are easy to find in an image and serve as the corresponding 2D points.

The camera matrix has 11 unknown parameters. Minimum six corresponding points are needed to estimate the parameters, but usually there may be more points to account for noise in the measurements. The camera matrix is estimated following Eq. (18).

3 Multiple View Geometry

3.1 Image to image projections

In the last section, 3D points were mapped to 2D points. In this section, 2D points are mapped to 2D points by a projective transformation. Figure 9 shows several possible transformations including translation and rotation. The most general transformation is a projective transformation or a homography, which is a 3×3 matrix applied to homogeneous coordinates of 2D points.

The mapping arises in several contexts such as in registering images or creating panoramas. Figure 10 shows a common situation where two cameras take a picture of the same plane from a different point of view. There could also be a single camera taking a picture at a different time and position. The cameras image the 3D point r in the object plane to points $\mathbf{p}_A$ and $\mathbf{p}_B$. A homography $\mathbf{H}$ relates these two points:

$$\mathbf{p_B} = \mathbf{H}\mathbf{p}_A. \tag{25}$$

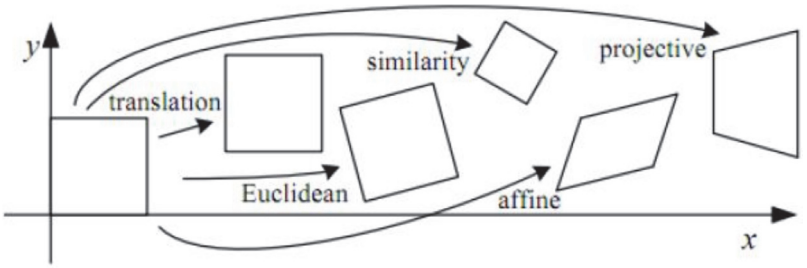

Figure 9 Various types of projective transformations.

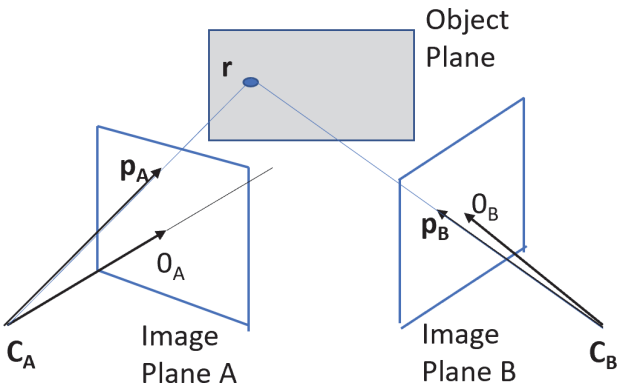

Figure 10 Two cameras take a picture of the same scene, which lies on a plane.

```
from PIL import Image
import numpy as np
import matplotlib.pyplot as plt
from scipy.cluster.vq import *

im = np.array(Image.open('airplane_segmentation.jpeg'))
im1 = rgb2gray(im)

w = 50 # larger size may consume too much memory

rim = Image.fromarray(im1)
rim=rim.resize(size=(w, w))
rim = np.array(rim,'f')

A = getSimilarityMatrix(rim,w)

segmentedImg, V = cluster(A,k=3,ndim=3)
segmentedImg = segmentedImg.reshape(w,w)
```

5 Simulation of Partially Coherent Systems

5.1 Optical system simulation

Computer simulation of an imaging system is very useful in many applications, such as the design of an imaging system. It is cheaper to simulate a system to see how it would behave than to actually build it to try it out. The simple pinhole camera model described previously is not accurate enough in the cases where the wave nature of light needs to be considered. This occurs when the feature sizes become comparable to the wavelength of the light used. This section will describe how to model the light propagation, in the scalar case, covering the topic of partial coherence. One-dimensional (1D) description will be used for clarity.

In several cases, light transmission can be modeled as a linear operation on the field. A superposition integral describes the transmission. A point in the image plane is formed by the contribution of the responses of multiple points in the object plane. This is in contrast with the pin-hole model where there is a one-to-one correspondence. The function h is called the impulse response of the system:[10]

$$u_o(x) = \int u(x)h(x,x')d\mathbf{x}'. \tag{33}$$

The electric field is too fast to observe at optical frequencies (frequency $\sim 10^{14}$ Hz). The averaged intensity is observable and given by averaging the squared the magnitude of the field ($<|\mathbf{u}|^2>$).

```
G2e = np.concatenate((G2,np.zeros((2,1))),axis=1)
G2e = G2e.T
onet = np.array([0,0,1])
onet = np.reshape(onet,(3,1))
G2e = np.concatenate((G2e,onet),axis=1)
Uc = np.dot(U2c.T,G2e)
Uc = np.dot(U1c.T,Uc)
```

$$\mathbf{A} = \mathbf{UDV}^*$$

7 Appendix: Code Listings

7.1 Camera calibration

The function *generate3DPoints()* generates 3D corner points of the checker board configuration assuming the squares have unit widths and returns them in a matrix **X**. The function *makeReferenceP()* creates a projection matrix using an intrinsic camera matrix, camera rotations, and translations. These are made up for demonstration purposes. The reference projection matrix is used to project the 3D points to 2D. The function *estimateP()* then takes the pairs of corresponding 3D and 2D points and uses the SVD to estimate the projection matrix. The RQ decomposition is then used to estimate the intrinsic camera matrix. The RQ decomposition is not directly supported in Numpy but can be obtained from the QR decomposition.

```
import numpy as np

def generate3DPoints():
    # geberate 3D points
    X = np.zeros((4,12))
    X[3,:] = 1

    for i in range(1,8):
        b = np.binary_repr(i,width=3)
        bA = np.array(list(b))
        X[:3,i] =bA

    X[:3,8:12] = 2*X[:3,1:5]
    return X
```