Prereading for the 6th lecture

Ilia Nechaev

04.10.2024

1 Introduction

During the 6th practice we will discuss the structure from motion (SFM) methods. SFM is widely used in computer vision and photogrammetry to recover the 3D structure of the scene from a set of 2D images. Ideas of SFM are also used in SLAM (Simultaneous Localization and Mapping) to estimate the camera trajectory and the 3D structure of the scene.
The topic of SFM is very broad, and we will only scratch the surface during the lecture, however we need some mathematical background to understand the basics of SFM. In this prereading, we will cover the following topics[1]:

- Matrices and vectors
- Affine transformations
- Change of basis
- Singular Value Decomposition (SVD) and its applications in systems of linear equations

2 Matrices and vectors

In simple words vectors are just arrays of numbers. Matrices are tables of numbers or 2D arrays of numbers. Vectors and matrices are used to represent data in a compact form and to perform operations on this data. For vector and matrix operations, we use the rules of linear algebra.

1. Summation:

$$\begin{pmatrix} a_{1,1} & a_{1,2} & \dots & a_{1,n} \\ a_{2,1} & a_{2,2} & \dots & a_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m,1} & a_{m,2} & \dots & a_{m,n} \end{pmatrix} \pm \begin{pmatrix} b_{1,1} & b_{1,2} & \dots & b_{1,n} \\ b_{2,1} & b_{2,2} & \dots & b_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ b_{m,1} & b_{m,2} & \dots & b_{m,n} \end{pmatrix} = \begin{pmatrix} a_{1,1} \pm b_{1,1} & a_{1,2} \pm b_{1,2} & \dots & a_{1,n} \pm b_{1,n} \\ a_{2,1} \pm b_{2,1} & a_{2,2} \pm b_{2,2} & \dots & a_{2,n} \pm b_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m,1} \pm b_{m,n} & a_{m,2} \pm b_{m,2} & \dots & a_{m,n} \pm b_{m,n} \end{pmatrix}$$

$$(a_1, a_2 \dots a_n) \pm (b_1, b_2 \dots b_n) = (a_1 \pm b_1, a_2 \pm b_2 \dots a_n \pm b_n)$$

2. Multiplication by a scalar:

$$k \cdot \begin{pmatrix} a_{1,1} & a_{1,2} & \dots & a_{1,n} \\ a_{2,1} & a_{2,2} & \dots & a_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m,1} & a_{m,2} & \dots & a_{m,n} \end{pmatrix} = \begin{pmatrix} k \cdot a_{1,1} & k \cdot a_{1,2} & \dots & k \cdot a_{1,n} \\ k \cdot a_{2,1} & k \cdot a_{2,2} & \dots & k \cdot a_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ k \cdot a_{m,1} & k \cdot a_{m,2} & \dots & k \cdot a_{m,n} \end{pmatrix}$$

$$k \cdot (a_1, a_2 \dots a_n) = (k \cdot a_1, k \cdot a_2 \dots k \cdot a_n)$$

[1] Material discussed in this document can't be considered as complete and comprehensive. It is recommended to refer to the textbooks and other sources for more detailed information. However, knowledge of the topics discussed here is sufficient to understand the basics of SFM.

3. Dot product for vectors:

$$(a_1, a_2 \dots a_n) \cdot (b_1, b_2 \dots b_n) = a_1 \cdot b_1 + a_2 \cdot b_2 + \ldots + a_n \cdot b_n$$

4. Matrix multiplication:

$$\begin{pmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,n} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m,1} & a_{m,2} & \cdots & a_{m,n} \end{pmatrix} \cdot \begin{pmatrix} b_{1,1} & b_{1,2} & \cdots & b_{1,k} \\ b_{2,1} & b_{2,2} & \cdots & b_{2,k} \\ \vdots & \vdots & \ddots & \vdots \\ b_{n,1} & b_{n,2} & \cdots & b_{n,k} \end{pmatrix} = \begin{pmatrix} c_{1,1} & c_{1,2} & \cdots & c_{1,k} \\ c_{2,1} & c_{2,2} & \cdots & c_{2,k} \\ \vdots & \vdots & \ddots & \vdots \\ c_{m,1} & c_{m,2} & \cdots & c_{m,k} \end{pmatrix} \text{ where } c_{p,q} = \sum_{i=1}^{n} a_{p,i} \cdot b_{i,q}$$

3 Affine transformation

3.1 What is affine transformation?

An affine transformation is a type of geometric transformation that preserves straight lines and parallelism relationships between them, but can change distances and angles. Simply put, affine transformations include operations such as:

- Parallel transfer (shift): moving all points of the object to the same vector.
- Rotation: rotating the object around a certain point at a given angle.
- Scaling: increasing or decreasing the size of an object along one or more axes.

Usually when we talk about affine transformations, we would like to know how to represent them in a matrix form.

3.2 Matrix representation of affine transformations in 3D space

For 3D space we use the following vector representation:

1. Point in 3D space: $(x, y, z, 1)$
2. Vector in 3D space: $(x, y, z, 0)$

Last number in the vector is used to distinguish between points and vectors (in geometrical meaning). So here's how we can represent affine transformations in a matrix form:

1. Parallel transfer (shift) to a vector (t_x, t_y, t_z):

$$\begin{pmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

2. Rotation around the x-axis by an angle α:

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\alpha) & -\sin(\alpha) & 0 \\ 0 & \sin(\alpha) & \cos(\alpha) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

3. Rotation around the y-axis by an angle α:

$$\begin{pmatrix} \cos(\alpha) & 0 & \sin(\alpha) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin(\alpha) & 0 & \cos(\alpha) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

4. Rotation around the z-axis by an angle α:

$$\begin{pmatrix} \cos(\alpha) & -\sin(\alpha) & 0 & 0 \\ \sin(\alpha) & \cos(\alpha) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

5. Scaling along the x-axis by a factor s_x, along the y-axis by a factor s_y, and along the z-axis by a factor s_z:

$$\begin{pmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Combination of these transformations can be represented as a product of matrices.

3.3 Converting between rotation matrix and other rotation representation

In OpenCV and other libraries, rotations are often represented as rodrigues vectors or quaternions.
Rodrigues vector.
Rodrigues vector is a compact representation of rotation. It is a vector of 3 elements, the direction of which is the axis of rotation, and the length is the angle of rotation.
To convert between rotation matrix and rodrigues vector, we can use function implemented in OpenCV: cv2.Rodrigues. But be aware that this function returns a 3×3 matrix of rotation (or the left upper corner of the 4×4 matrix of affine transformation).
Quaternion.
If you aren't familiar with quaternions and their applications in representing rotations you can skip this part.
Assume we have a quaternion $q = (w, x, y, z)$ and $||q|| = 1$. To convert it to a rotation matrix R, we can use the following formula:

$$R = \begin{pmatrix} 1 - 2y^2 - 2z^2 & 2xy - 2wz & 2xz + 2wy \\ 2xy + 2wz & 1 - 2x^2 - 2z^2 & 2yz - 2wx \\ 2xz - 2wy & 2yz + 2wx & 1 - 2x^2 - 2y^2 \end{pmatrix}$$

As in previous cases, be aware that we got a 3×3 matrix of rotation (or the left upper corner of the 4×4 matrix of affine transformation).

4 Change of basis

In linear algebra, a basis is a set of linearly independent vectors that span a vector space. The basis vectors are used to represent other vectors in the space.
Example: if we have standard basis $(1, 0, 0)$, $(0, 1, 0)$, $(0, 0, 1)$ in 3D space, then any vector (x, y, z) can be represented as $x(1, 0, 0) + y(0, 1, 0) + z(0, 0, 1)$.
So generally speaking if we change basis in 3D space from standard basis to some other basis, which also consists of 3 orthogonal vectors, then we can say changing of basis is just affine transformation which consist of rotation and scaling. If we talk about not just basis, but coordinate system, then changing of coordinate system is changing of basis and parallel transfer of origin. So in general changing of basis can be represented as affine transformation, which consists of rotation, scaling and parallel transfer.

5 Singular Value Decomposition (SVD)

5.1 Eigenvalues and eigenvectors (optional)

If you are looking just for the basics of SVD, you can skip this section, however if you want to deepen your understanding of SVD, you should read this section as a start and then refer to the textbooks

and other sources.
Eigenvalues and eigenvectors are used in linear algebra to solve systems of linear equations and to represent linear transformations.
Definition: Let A be a square matrix. A scalar λ is called an eigenvalue of A if there exists a non-zero vector v such that:

$$A \cdot v = \lambda \cdot v$$

The vector v is called an eigenvector of A corresponding to the eigenvalue λ.
To find eigenvalues of a matrix A, we need to solve the following equation:

$$|A - \lambda \cdot I| = 0$$

where I is the identity matrix. And $|A|$ is determinant of matrix A.
Eigenvectors can be found by solving the following equation:

$$(A - \lambda \cdot I) \cdot v = 0$$

where v is the eigenvector corresponding to the eigenvalue λ. Important that we are looking for non-zero vector v.

5.2 SVD. Definition

Singular Value Decomposition (SVD) is a method used to decompose a matrix into three matrices. Let us have a matrix $m \times n$ A, then SVD of matrix A is:

$$A = U \cdot \Sigma \cdot V^T$$

where:

- A is the matrix we want to decompose.
- U is an orthogonal matrix.
- Σ is a diagonal matrix.
- V^T is the transpose of an orthogonal matrix.

The diagonal elements of Σ are called singular values of matrix A.

5.3 SVD. How to find

To find SVD of matrix A, we need to perform the following steps:

1. Find eigenvalues and eigenvectors of $A^T \cdot A$.
2. Find eigenvalues and eigenvectors of $A \cdot A^T$ (eigenvalues for $A \cdot A^T$ and $A^T \cdot A$ are the same).
3. Singular values of matrix A are square roots of eigenvalues of $A^T \cdot A$ or $A \cdot A^T$ ordered in descending order.
4. Columns of matrix U are eigenvectors of $A \cdot A^T$.
5. Columns of matrix V are eigenvectors of $A^T \cdot A$.

Important part in SVD is that you need to chose sign of eigenvectors correctly. To do this you can use the following rule:
Assume $\{u_i\}_{i=0}^m$ are eigenvectors of matrix $A \cdot A^T$ and $\{v_i\}_{i=0}^n$ are eigenvectors of matrix $A^T \cdot A$ and $m > n$. Then choosing of sign of eigenvectors should be done as follows:

$$\hat{u}_i = \frac{A \cdot v_i}{\sigma_i}$$

Generally speaking $\hat{u}_i = \pm u_i$, so this operation is just choosing the sign of eigenvectors. For u_i where $i > n$ you can choose any sign.
SVD exists for any matrix of any size.

5.4 SVD. Applications

There are a lot of applications of SVD in different fields. We will focus only on the application of SVD in solving overdetermined systems of linear equations.
Let us have a system of linear equations: $Ax = b$, where A is a matrix $m \times n$, x is a vector $n \times 1$, and b is a vector $m \times 1$ and $m > n$. If we have more equations than unknowns, then the system is overdetermined. In this case, we can't find the exact solution, but we can find the least squares solution: $||Ax - b||$.
To do this we need to find SVD of matrix $A = U \cdot \Sigma \cdot V^T$. Then we need to find the pseudo-inverse of matrix A: $A^+ = V \cdot \Sigma^+ \cdot U^T$. Where Σ^+ is the pseudo-inverse of matrix Σ, in other words:

$$\Sigma^+_{i,i} = \begin{cases} \frac{1}{\Sigma_{i,i}} & \text{if } \Sigma_{i,i} \neq 0 \\ 0 & \text{if } \Sigma_{i,i} = 0 \end{cases}$$

Then we can find the least squares solution as follows:

$$x = A^+ \cdot b$$

6 Practice

Here are a few tasks to check your understanding of the topics discussed in this document. You are not required to submit the solutions, but if you feel that you need to practice more, you can try to solve them. Solutions are on the last page of this document.

6.1 Task 1

Compute the following matrix operations:

$$\left[\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} + \begin{pmatrix} 5 & 6 \\ 7 & 8 \end{pmatrix}\right] \cdot \begin{pmatrix} 1 & 0 & 2 \\ 0 & 1 & 3 \end{pmatrix} - \begin{pmatrix} 2 & 2 & 2 \\ 1 & 1 & 1 \end{pmatrix}$$

6.2 Task 2

Find the matrix representation of the following affine transformation:

1. Parallel transfer to the vector $(1, 2, 3)$.
2. Rotation around the x-axis by an angle $\frac{2\pi}{3}$.
3. Scaling along the y-axis by a factor of 2.

6.3 Task 3

Find the approximate solution of the following system of linear equations using SVD:

$$\begin{cases} 2x + 3y = 5 \\ 4x + 5y = 7 \\ 6x + 7y = 9.1 \end{cases}$$

This task is better to solve using Python and numpy library. For better understanding of SVD you can try not to use np.linalg.svd(), instead use np.linalg.eig() to find eigenvalues and eigenvectors.

7 Solutions

7.1 Task 1

$$\begin{pmatrix} 4 & 6 & 34 \\ 9 & 11 & 55 \end{pmatrix}$$

7.2 Task 2

Matrices for the given affine transformations:

1. Parallel transfer to the vector $(1, 2, 3)$: $A = \begin{pmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 2 \\ 0 & 0 & 1 & 3 \\ 0 & 0 & 0 & 1 \end{pmatrix}$

2. Rotation around the x-axis by an angle $\frac{2\pi}{3}$: $B = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & -\frac{1}{2} & -\frac{\sqrt{3}}{2} & 0 \\ 0 & \frac{\sqrt{3}}{2} & -\frac{1}{2} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$

3. Scaling along the y-axis by a factor of 2: $C = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$

To find the matrix for the combination of these transformations, you need to multiply the matrices.

$$A \cdot B \cdot C = \begin{pmatrix} 1 & 0 & 0 & 1 \\ 0 & -1 & -\frac{\sqrt{3}}{2} & 2 \\ 0 & \sqrt{3} & -\frac{1}{2} & 3 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

7.3 Task 3

Firstly we need to represent the system of linear equations in matrix form:

$$\begin{pmatrix} 2 & 3 \\ 4 & 5 \\ 6 & 7 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 5 \\ 7 \\ 9.1 \end{pmatrix}$$

Then we need to find SVD of matrix A. Let's use numpy library to do this. Here we want to use numpy just as smart calculator:

```python
1  import numpy as np
2  A = np.array(
3      [
4          [2, 3],
5          [4, 5],
6          [6, 7]
7      ]
8  )
9  b = np.array([[5], [7], [9.1]])
10 AA_T = A @ A.T
11 A_T_A = A.T @ A
12 AA_T_eig_vals, U_draft = np.linalg.eig(AA_T)
13 A_T_A_eig_vals, V_draft = np.linalg.eig(A_T_A)
14 print(f"AA_T_eig_vals: {AA_T_eig_vals}\nA_T_A_eig_vals: {A_T_A_eig_vals}\n")
```

Let's explain the code:

1. In lines 1–9 we have standard imports and the definition of matrix A and vector b.

2. In lines 10–11 we prepare matrices $A^T \cdot A$ and $A \cdot A^T$. We need them to find eigenvalues and eigenvectors.

3. In lines 12–13 we find eigenvalues and eigenvectors of matrices $A^T \cdot A$ and $A \cdot A^T$.

But we need to sort the eigenvalues. That's why we firstly output eigen values to check if they are sorted in descending order. In my case I got this output:

```
AA_T_eig_vals: [ 1.38827123e+02  1.72876881e-01 -8.76424074e-15]
A_T_A_eig_vals: [  0.17287688 138.82712312]
```

As you can see eigenvalues are not sorted for matrix $A^T \cdot A$. So we need to sort them and corresponding eigenvectors. Also notice that last eigenvalue for $A \cdot A^T$ is zero, that's because matrix A is not square.

```
1 Sigma = np.vstack((np.diag(np.sqrt(-np.sort(-A_T_A_eig_vals))), (0,0)))
2 V = np.hstack((V_draft[:,1].reshape((-1,1)), V_draft[:,0].reshape((-1,1))))
3 U = np.hstack((A @ V[:,0].reshape((-1,1)) / Sigma[0,0], A @ V[:,1].reshape((-1,1)) / Sigma[1,1], U_draft
      [:,2].reshape((-1,1))))
4 V_t = V.T()
5 assert np.allclose(A, U @ Sigma @ V_t)
```

Let's explain the code:

1. In line 1 we sort eigenvalues of matrix $A \cdot A^T$ in descending order and create a matrix Σ.
2. In line 2 we sort eigenvectors of matrix $A \cdot A^T$ and create matrix V.
3. In line 3 we find matrix U using the rule for choosing the sign of eigenvectors.
4. In line 4 we find the transpose of matrix V.

Important to note that we need to add the third row to the matrix Σ, because otherwise we will get a matrix of size 2×2, which can't be multiplied by matrix U which is matrix 3×3. After that we can simply find the least squares solution:

```
Sigma_plus = np.linalg.pinv(Sigma)
x = V @ Sigma_plus @ U.T @ b
print(x)
```

And the result is:

```
[[-1.90833333]
 [ 2.93333333]]
```