Project Stereo

Author: Yongshun Zhang

March 28, 2018

1 Camera Basics

1.1 the Intrinsics and the Extrinsics of a Camera and the Camera Matrix

A **Camera Matrix** is used to denote a projective mapping from World Coordinates to Pixel Coordinates, which can be represented as follows. [1]

$$Camera\ Matirx\ \ \mathbf{P} = K \times \begin{bmatrix} R & T \end{bmatrix} \tag{1}$$

In equation (1), the matrix **K** is called intrisic parameters, and the matrix [R T] is the extrinsic parameters.

The intrinsic matrix **K** contains 5 intrinsic, and has the following form.

$$\mathbf{K} = \begin{bmatrix} \alpha_x & \gamma & \mu_0 & 0 \\ 0 & \alpha_y & \upsilon_0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \tag{2}$$

These parameters encompass focal length, image sensor format and principal point(image centre). Before introducing the intrinsic matrix ,we need to know how the pixel skew is define, which is shown in Figure 1 [2].

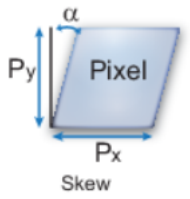


Figure 1: skew

The meanings and metrics of the parameters $\alpha_x,\ \alpha_y,\ \gamma,\ \mu_0,\ \upsilon_0$ are described in tabulation 1 [1] [2].

$(\mu_0,\ \upsilon_0)$ // Principal point, in pixels
$(\alpha_x,\ \alpha_y)$ //focal length, in pixels $\alpha_x\ \leftarrow F/p_x, \alpha_y\ \leftarrow F/p_y$ F // Focal length in world units, typically expressed in millimeters $(p_x,\ p_y)$ // Size of the pixel in world units
s $\leftarrow\ alpha_y * tan(\alpha)$ // Skew coefficient

Table 1: parameters of intrinsic matrix

Extrinsic parameters consisting of matrix **R**,**T**, are the extrinsic parameters which denote the coordinate system transformations from 3D world coordinates to 3D camera coordinates. Extrinsic parameters define the position of camera center and the camera's heading in real world. **T** is the position of the origin of the world coordinate system expressed in coordinates of camera-centered coordinate system, **R** is a rotation matrix used to perform a rotation in Euclidean space, which properties are $\mathbf{R}^T = \mathbf{R}^{-1}$ and det **R**=1 [3].

Often, we use $[\mu\ \upsilon\ 1]^T$ to represent a 2D point position in pixel coordinates, and $[x_w\ y_w\ z_w\ 1]^T$ is used to present a 3D point position in world coordinates, which are expressed in augmented notation of Homogeneous coordinates. Referring to the pinhole camera model, the camera matrix denotes a projective mapping from world coordinates to pixel coordinates.

$$z^c \times \begin{bmatrix} \mu \\ \upsilon \\ 1 \end{bmatrix} = K \times \begin{bmatrix} R & T \end{bmatrix} \times \begin{bmatrix} x_w \\ y_w \\ z_w \\ 1 \end{bmatrix} \tag{3}$$

z_c represents the coordinate of z axis in used to make the constant of proportionality of two matrices [4].

Problems There are some little problems during learning Camera Matrix. The first is the position, **C**, of the camera expressed in world coordinate, I didn't understand the equation $\mathbf{C} = -\mathbf{R}_{-1}\mathbf{T} = -\mathbf{R}_T\mathbf{T}$ originally, and I find the answer in the meaning of the Homogeneous coordinates. When we transfer the coordinates from 3D world to 2D pixel, the coordinate expressed in Homogeneous coordinates can realize rotation, zoom and translation, so translation T, which represents the origin of world coordinates expressed in pixel coordinates, used to make the 2D pixel coordinate taking the origin of pixel coordinates(image centre) as its origin of coordinates. And the second problem is that the five intrinsic parameters in matrix **K** are expressed indistinct in wikipedia, and I found the distinct expression in MathWorks website [2], which is shown in tabulation 1.

Written by Yongshun, 22:00, 21th Mar 2018

1.2 Camera Imaging

Problems When I start to deal with the transform question, I find that I'm still in a preliminary understanding of camera resectioning and camera matrix, so I read the detailed introduction of **Camera Matrix** [5] and the **Pinhole Camera model** [6] from wikipedia firstly, and have a progressive understanding of Camera Matrix including its relation with pinhole camera model and the derivation of its equation. But the next problem comes, I find that I'm confused of the axes' directions and the relationship between principal point's coordinate and optical centre's coordinate.

So I try to search the relationship between principal point and optical centre by Google, such as "the relationship between principal point and optical centre" and also try to search in Chinese, finally I found the relationship between in wikipedia "Pinhole Camera Model" [6], it tells "A point R at the intersection of the optical axis and the image plane. This point is referred to as the principal point or image center.". As for the axes' directions, I review the web pages about the camera matrix, camera resectioning I have read, and also read some documents and blogs [7] [8] [9], and finally solve the problem temporarily.

The answer of transform is as follows.

First, transform the 3D point **X** from World coordinates to Camera coordinates, using the extrinsic parameters.$[X_c,\ Y_c,\ Z_c]_T$ represents the point in Camera coordinates.

$$\begin{bmatrix} X_c \\ Y_c \\ Z_c \end{bmatrix} = \begin{bmatrix} R & T \end{bmatrix} \times \begin{bmatrix} X_1 \\ X_2 \\ X_3 \\ 1 \end{bmatrix} \tag{4}$$

Second, transform the 3D point in Camera coordinates to 2D point in Image plane.

$$Z_c \times \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & 0 \\ 0 & f_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} X_c \\ Y_c \\ Z_c \end{bmatrix} \tag{5}$$

Finally, transform the 2D point from image plane to pixel coordinates.

$$u = x + c_x, \quad v = y + c_y \tag{6}$$

All equations above can be replaced with the following one.

$$Z_c \times \begin{bmatrix} \mu \\ \upsilon \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} R & t \end{bmatrix} \times \begin{bmatrix} X_1 \\ X_2 \\ X_3 \\ 1 \end{bmatrix} \tag{7}$$

1.3 Relationship Between 2D Image Point and 3D Camera Coordinate

A 2D image point $(\mu,\ \upsilon)$ correspond to a line in 3D camera coordinate. The derivation is as follows.

First, transform the image point $(\mu,\ \upsilon)$ from pixel coordinate to image plane.

$$x = (\mu - \mu_0), \quad y = (\upsilon - \upsilon_0) \tag{8}$$

Second, according to the pinhole camera model, transform the point from image plane to camera coordinate.

$$X_c = x \times \frac{Z_c}{f}, \quad Y_c = y \times \frac{Z_c}{f} \tag{9}$$

According to equation (9), we set $\frac{x}{f} = k,\ \frac{y}{f} = t$, k and t are constants, so the 3D point in camera coordinate can be describe as $Z_c \times (k,\ t,\ 1)$, which represents a line in 3D coordinate.

Written by Yongshun, 00:43, 22th Mar 2018

1.4 Distortion

When I read the Camera Calibration and 3D Construction in opencv website [10], I find that I once read this document which has been translated in Chinese [9], and I read the document again carefully and review the introduction of distortion in Mathworks [2].

As for distortions in camera calibration, it can be classified into two types: radial distortion and tangential distortion, real lenses usually have some distortion, mostly radial distortion and slight tangential distortion.

Figure 2 show two common types of Radial distortion: barrel distortion(typically $k_1 > 0$) and pincushion distortion(typically $k_1 < 0$).

Figure 2: Radial Distortion

Tangential distortion occurs when the lens and the image plane are not parallel, shown in Figure 3.

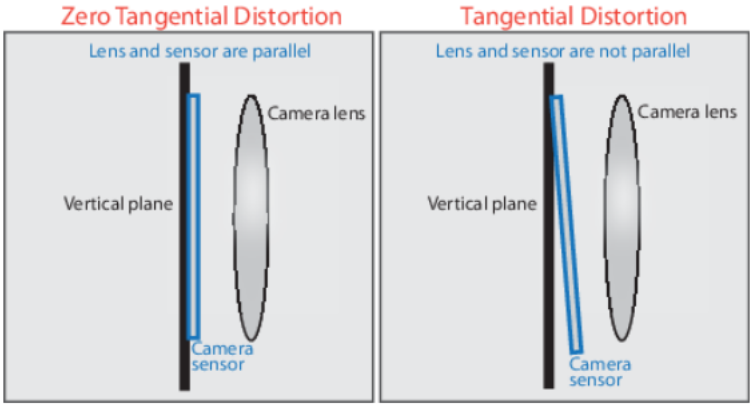


Figure 3: Tangential Distortion

When the distortions are considered, the camera resectioning model will be extend as:

$$\begin{bmatrix} X_c \\ Y_c \\ Z_c \end{bmatrix} = \begin{bmatrix} R & T \end{bmatrix} \times \begin{bmatrix} X_w \\ Y_w \\ Z_w \end{bmatrix} \tag{10}$$

$$x = \frac{X_c}{Z_c}, \quad y = \frac{Y_c}{Z_c} \tag{11}$$

$$\begin{aligned} x_d &= x \times \frac{1 + k_1 r^2 + k_2 r^4 + k_3 r^6}{1 + k_4 r^2 + k_5 r^4 + k_6 r^6} + 2p_1 xy + p_2(r^2 + 2x^2) \\ y_d &= y \times \frac{1 + k_1 r^2 + k_2 r^4 + k_3 r^6}{1 + k_4 r^2 + k_5 r^4 + k_6 r^6} + 2p_1(r^2 + 2y^2) + 2p_2 xy \\ r^2 &= x^2 + y^2 \end{aligned} \tag{12}$$

$$u = f_x \times x_d + c_x, \quad v = f_y \times y_d + c_y \tag{13}$$

Find the Origin Point Considering the distortion model with 2 distortion coefficients, the transform equation with distortions is as follows. Point (x,y) represents the ideal point without distortion.

$$\begin{aligned} \check{x} &= x + x(k_1(x^2+y^2) + k_2(x^2+y^2)^2) \\ \check{y} &= y + y(k_1(x^2+y^2) + k_2(x^2+y^2)^2) \end{aligned} \tag{14}$$

And we have

$$\begin{aligned} \check{x} &= \frac{u-u_0}{f} \\ \check{y} &= \frac{u-u_0}{f} \end{aligned} \tag{15}$$

Then we make a square of each equation in (14), and add them together, we got

$$\check{x}^2 + \check{y}^2 = (x^2+y^2) + 2k_1(x^2+y^2)^2 + (2k_2+k_1^2)(x^2+y^2)^3 + 2k_1k_2(x^2+y^2)^4 + k_2^2(x^2+y^2)^5 \tag{16}$$

This is a liner equation of (x^2+y^2), we can solve it by gradient descent. After we get the value of (x^2+y^2), we get the origin point (x, y) by bring (x^2+y^2) into equation (14).

1.5 Calibration

Base on what I know about Camera Calibration, I reckon that Camera Calibration aims to get an exact approximation of the intrinsic parameters including the distortion coefficients which only depends on the camera itself, and extrinsic parameters which represent the camera's direction and location in 3D space.

Written by Yongshun, 22:26, 23th Mar 2018

1.6 Calibration with Opencv

Before I start to use the opencv functions to calibrate the camera, I read the APIs in camera calibration introduction on OpenCV website [10], and sample code of camera calibration using OpenCV on github [11]. In the process of programming, Numpy Reference [12] is used as a reference of some numpy functions, and at first, I don't know how to judge the parameters calculated, so I read the APIs on OpenCV website [10], "Evaluating the Accuracy of Single Camera Calibration" on Mathworks [13], the introduction of re-projection error and homography in Wikipeida [14] [15], and finally adopt the re-projection errors to judge whether the parameters perfect or not.

Source codes can be find by link

https://github.com/zhangyongshun/Project$_S$*tereo/blob/master/python*$_c$*ode/.idea/opencvCameraCalibration.py*

Int the program, use OpenCV function drawChessboardCorners() to draw the corners detected in the images as Figure 4.

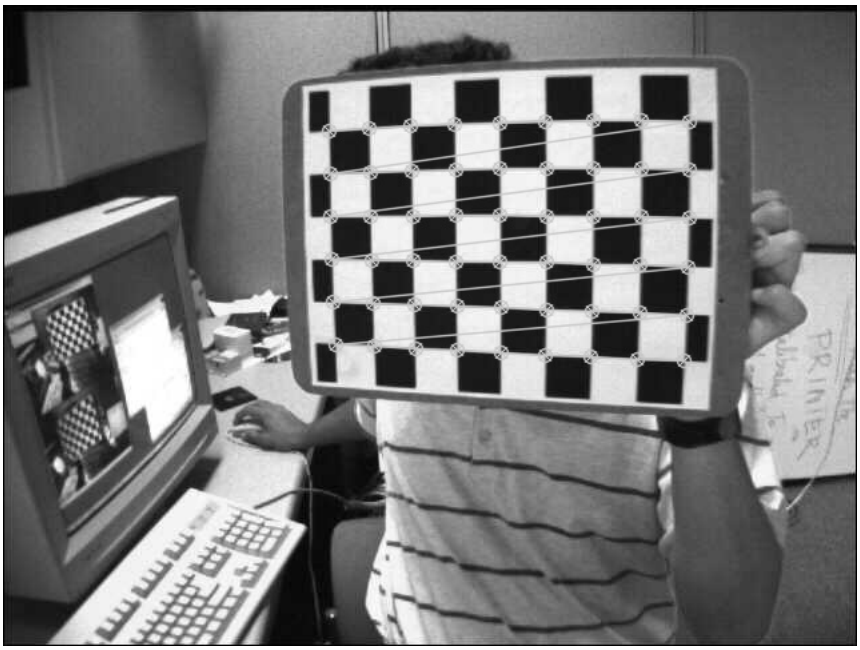


Figure 4: Corners

The camera matrix and the re-projection error output is

```
camera matrix is:
 [[532.79536562   0.          342.45825163]
 [  0.          532.91928338 233.90060514]
 [  0.            0.           1.        ]]
total error is: 0.026558812568814334
```

Figure 5: Camera Matrix and error

As a result, re-projection error is between 0.02 and 0.03.

1.7 Undistort the Image

According to the "Camera Calibration and 3D Reconstruction" on OpenCV web site [10], I found the undistort() function to undistort the image with the calibration results. In the document of the function undistort(), it mentions the function getOptimalNewCameraMatrix() can be used to compute the appropriate 'newCameraMatrix' depend on requirements, so in the process of undistorting the image, I first use the function getOptimalNewCameraMatrix() to get a proper 'newCameraMatrix' according to subset of source image, and then use the function undistort() with 'newCameraMatrix' to undistort the image. Source code could be found in github by link

https://github.com/zhangyongshun/Project_Stereo/blob/master/python_code/.idea/opencvUndistort.py

The comparison between distorted image and undistorted image as showing in Figure 6.

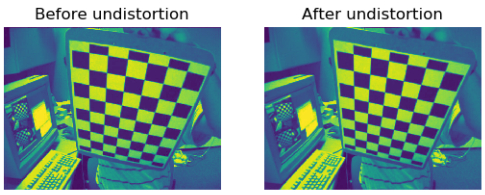


Figure 6: Comparison Between Distortion and Undistortion

1.8 Implement Zhang's Method

Problems There are some intractable problems I have met, when I try to implement the method of Z.Zhang.

The first is solving the equation $\mathbf{Vb} = \mathbf{0}$ about b, where V is a 2n*6 matrix and b is 6*1 vector, corresponding to equation(9) in Z.Zhang's paper [16]. The solution of the equation is the intrinsic parameters. What confuses me at first is "the solution to equation(9) is well known as the eigenvector of V^TV associated with the smallest eigenvalue (equivalently, the right singular vector of V associated with the smallest singular value)", because the course of 'Linear Algebra' doesn't teach the detailed knowledge of singular value and eigenvalue. So I review the basic knowledge of singular value and eigenvalue [17] [18], then I read the Singular-value decomposition on wikipedia [19]. When I am familiar with singular value and eigenvalue, and I find that I still can figure out the equation(9) until I read the Homogeneous differential equation on wikipedia [20] and a blog which tells how to use svd to solve Homogeneous equation [21].

The second problem is the Levenberg-Marquardt Algorithm(LM) [22] [23]. The most differecult of LM, used in method of Z.Zhang, is solving the jacobian matirx. According to Z.Zhang's paper, there are two minimization problem in total, which are recommended to use LM Algorithm.
One is estimation of the homography matrix H between the model plane and its image which is mentioned in Z,Zhang'paper appendix A, by minimizing the following functional

$$\sum_i (\mathbf{m}_i - \hat{\mathbf{m}}_i)^T \mathbf{\Lambda}_{\mathbf{m}_i}^{-1} (\mathbf{m}_i - \hat{\mathbf{m}}_i) \ ,$$

where $$\hat{\mathbf{m}}_i = \frac{1}{\bar{\mathbf{h}}_3^T \mathtt{M}_i} \begin{bmatrix} \bar{\mathbf{h}}_1^T \mathtt{M}_i \\ \bar{\mathbf{h}}_2^T \mathtt{M}_i \end{bmatrix} \quad \text{with } \bar{\mathbf{h}}_i \text{, the } i^{\text{th}} \text{ row of } \mathbf{H}.$$

and other is to estimate the same cost function as the following

$$\sum_{i=1}^{n} \sum_{j=1}^{m} \| \mathbf{m}_{ij} - \breve{\mathbf{m}}(\mathbf{A}, k_1, k_2, \mathbf{R}_i, \mathbf{t}_i, \mathtt{M}_j) \|^2 \ ,$$

Both of cost functions have parameters need to be estimated by LM Algorithm, and the former's cost function has 9 parameters, the latter has 23 parameters($\mathbf{A}$ and R are 3×3 matrices, $\mathbf{t}$ is 3×1 vector, k_1, k_2 are two scales), so difficulties come over when we calculate the jacobian matrix. For the former one, 9 parameters is under toleration and is also what to realize the LM Algorithm in the code, but for the latter, it is to difficult to calculate the jacobian

matrix with 23 parameters, which I don't use LM Algorithm in the code although Z.Zhang's paper recommends. For the latter problem, I learn the Powell Algorithm which doesn't need to calculate the jacobian matrix, and the powell algorithm [24] is what I'm learning now and isn't been realized yet.

Third problem is a program language problem. I use python to realize Z.Zhang's method, and when I finish the code and run it, I find that the outcome of code isn't satisfying as expected, so I look out the code in order to find what the error is. But nothing is got firstly, so I debug the code step by step, and find a problem: there are some code in my program like this

```
pattern_points.append(single_pattern_points)
image_points.append(corners)
```

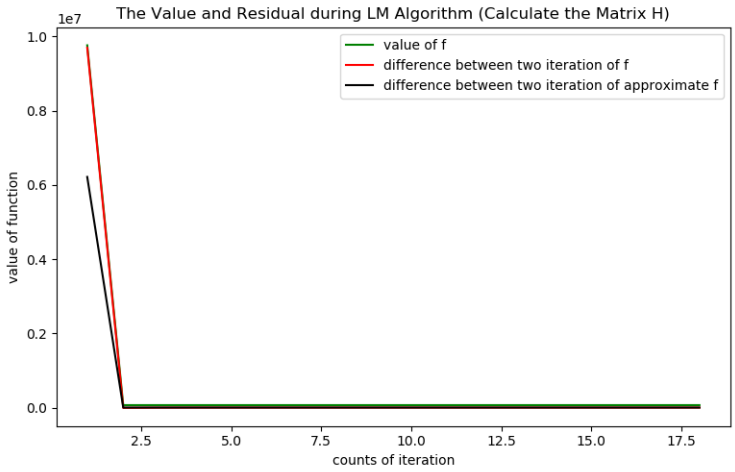
because the assignment between objects in python is passed by reference. When I assign an object's value to another by the name of object, it's only the references, so I change the object, for example 'single_pattern_points' in following code, the corresponding value in 'pattern_points' will also be change, and I solve this problem by copying the objects when assigning between objects. And this problem shows I'm unable to master python skillfully, there are so many knowledge need to be learn.

Although I solve the assignment problem between objects in python, but the outcome of code is still unsatisfying. I remind that Z.Zhang mentions that much better results can be obtained by performing a simple data normalization in appendix A, Besides, when using LM Algorithm to estimate the homography matrix H between the model plane and its image, mentioned in appendix A, the initial guess of H is got by

$$\begin{bmatrix} \widetilde{\mathbf{M}}^T & \mathbf{0}^T & -u\widetilde{\mathbf{M}}^T \\ \mathbf{0}^T & \widetilde{\mathbf{M}}^T & -v\widetilde{\mathbf{M}}^T \end{bmatrix} \mathbf{x} = \mathbf{0} \, .$$

$x = [h1^T, h2^T, h3^T]^T$. In this method to get initial guess of H, the elements in H is too small that it's easily to satisfy the threshold in LM Algorithm. So I read the paper "In defence of the 8-point algorithm" [25] and get a normalizing transformations method and then realize it. Better outcome is got.

However, although the code has been finished, the intrinsic and extrinsic parameters it gets is unpractical, which confused me for several days, and has not been figured out yet. All the code is followed by Z.Zhang's method, and LM algorithm is working well.



The picture shows the iteration of LM Algorithm, we can see that the function f value is converged to minimum value, and the difference of function f and approximate of function f between two iteration reduces fast in the beginning of iterations and slows down when it's close to minimum of f.

Although LM algorithm works well, but the outcome of code isn't well , so in next week, I will figure out the errors in the code and finish the report. Then begin to learn "Binocular Basics".

Below is the implementation of Z.Zhang's method.

https://github.com/zhangyongshun/Project_Stereo/blob/master/python_code/.idea/Z.ZhangCameraCalibration.py

References

[1] **Camera resectioning** Wikipedia https://en.wikipedia.org/wiki/Camera_resectioning

[2] **What Is Camera Calibration?** MathWorks https://cn.mathworks.com/help/vision/ug/camera-calibration.html?s_tid=gn_loc_drop

[3] **Rotation matrix** Wikipedia https://en.wikipedia.org/wiki/Rotation_matrix

[4] **Scale factor** Wikipedia https://en.wikipedia.org/wiki/Scale_factor

[5] **Camera Matrix** Wikipedia https://en.wikipedia.org/wiki/Camera_matrix

[6] **Pinhole Camera Model** Wikipedia https://en.wikipedia.org/wiki/Pinhole_camera_model

[7] **Fundamentals of Computer Vision** Magill University http://www.cim.mcgill.ca/%7Elanger/558/4-cameramodel.pdf

[8] 关于 OpenCV 那些事 -相机标定 CSDN https://blog.csdn.net/aptx704610875/article/details/48914043

[9] Cv 相机校正和 3D 重建 Opencv http://wiki.opencv.org.cn/index.php/Cv%E7%85%A7%E7%9B%B8%E6%9C%BA%E5%AE%9A%E6%A0%87%E5%92%8

[10] **Camera Calibration and 3D Reconstruction** Opencv https://docs.opencv.org/2.4/modules/calib3d/doc/camera_calibration_and_3d_reconstruction.html

[11] **SampleCodeOfCalibration** Opencv https://docs.opencv.org/2.4/modules/calib3d/doc/camera_calibration_and_3d_reconstruction.html

[12] **Numpy Functions Reference** Numpy https://docs.scipy.org/doc/numpy-dev/reference/index.html

[13] **Evaluating the Accuracy of Single Camera Calibration** MathWorks https://cn.mathworks.com/help/vision/examples/evaluating-the-accuracy-of-single-camera-calibration.html

[14] **Reprojection error** Wikipeida https://en.wikipedia.org/wiki/Reprojection_error

[15] **Homography (computer vision)** Wikipedia https://en.wikipedia.org/wiki/Homography_(computer_vision)

[16] Z. Zhang, A flexible new technique for camera calibration," IEEE Trans- actions on pattern analysis and machine intelligence, vol. 22, no. 11, pp. 1330-1334, 2000.

[17] 郑宝东, 王忠英线性代数与空间解析几何 [M]. 第 4 版. 高等教育出版社, 2013 .04

[18] David C. Lay. 线性代数及其应用 [M]. 刘深泉, 洪毅等译. 第 3 版. 机械工业出版社, 2005 .08

[19] **Singular-value decomposition** wikipedia https://en.wikipedia.org/wiki/Singular-value_decomposition

[20] **Homogeneous differential equation** wikipedia https://en.wikipedia.org/wiki/System_of_linear_equationsHomogeneous

[21] **矩阵分解 (特征值/奇异值分解 +SVD+ 解齐次/非齐次线性方程组)** CSDN https://blog.csdn.net/MyArrow/article/details/53780972

[22] J. More, "The levenberg-marquardt algorithm, implementation and theory," In G. A. Watson, editor, Numerical Analysis, Lecture Notes in Mathematics 630. Springer-Verlag, 1977.

[23] Manolis I. A. Lourakis, "A Brief Description of the Levenberg-Marquardt Algorithm Implemened by levmar," Institute of Computer Science Foundation for Research and Technology - Hellas (FORTH) Vassilika Vouton, P.O. Box 1385, GR 711 10 Heraklion, Crete, GREECE, 2005

[24] **Powell** wikipedia https://en.wikipedia.org/wiki/Powell

[25] R. Hartley. "In defence of the 8-point algorithm," In Proceedings of the 5th International Conference on Computer Vision, pages 1064–1070, Boston, MA, June 1995. IEEE Computer Society Press.