Levenberg-Marquardt notes

Danping Zou @Shanghai Jiao Tong University

`http://drone.sjtu.edu.cn/dpzou/teaching/course/lecture07-08-nonlinear_least_square_ransac.pdf`
EE382-Visual localization & Perception, “Lecture 08- Nonlinear least square & RANSAC”

$\min_{R,\mathbf{t}} \sum_{i=1}^{N} r_i(R,\mathbf{t})^2 \quad (N \geq 3)$ where $r_i(\cdot)$ is the re-projection error of the corresponding points $\mathbf{X}_i \leftrightarrow \mathbf{m}_i$

which can be written as $f(\mathbf{x}) = \frac{1}{2}\mathbf{r}(\mathbf{x})^T\mathbf{r}(\mathbf{x})$

Its Jacobian H (a.k.a. J by others) is $\frac{\partial f}{\partial x_j} = \sum_{k=1}^{m} r_k(\mathbf{x})\frac{\partial r_k(\mathbf{x})}{\partial x_j}$

the gradient vector is $\boxed{\nabla f(\mathbf{x}) = J(\mathbf{x})^T\mathbf{r}(\mathbf{x})}$

Its Hessian is $\nabla^2 f(\mathbf{x}) = \frac{\partial}{\partial \mathbf{x}}\nabla f(\mathbf{x}) \boxed{= J(\mathbf{x})^T J(\mathbf{x}) + \sum_{k=1}^{m} r_k(\mathbf{x})\nabla^2 r_k(\mathbf{x})}$

when r(x) —> 0: $\nabla^2 f(\mathbf{x}) \approx J(\mathbf{x})^T J(\mathbf{x})$

Levenberg-Marquardt is the Gauss-Newton with a damping term $\boxed{\Delta\mathbf{x}^{LM} = (H^T H + \lambda I)^{-1} H^T \mathbf{r}}$

For large values of λ: $\Delta\mathbf{x}^{LM} \approx H^T\mathbf{r}/\lambda = -\nabla f/\lambda$ a short step in the steepest descent direction

For **small** values of λ: the step size tis good for the final iterations (similar to Gauss-Newton steps)

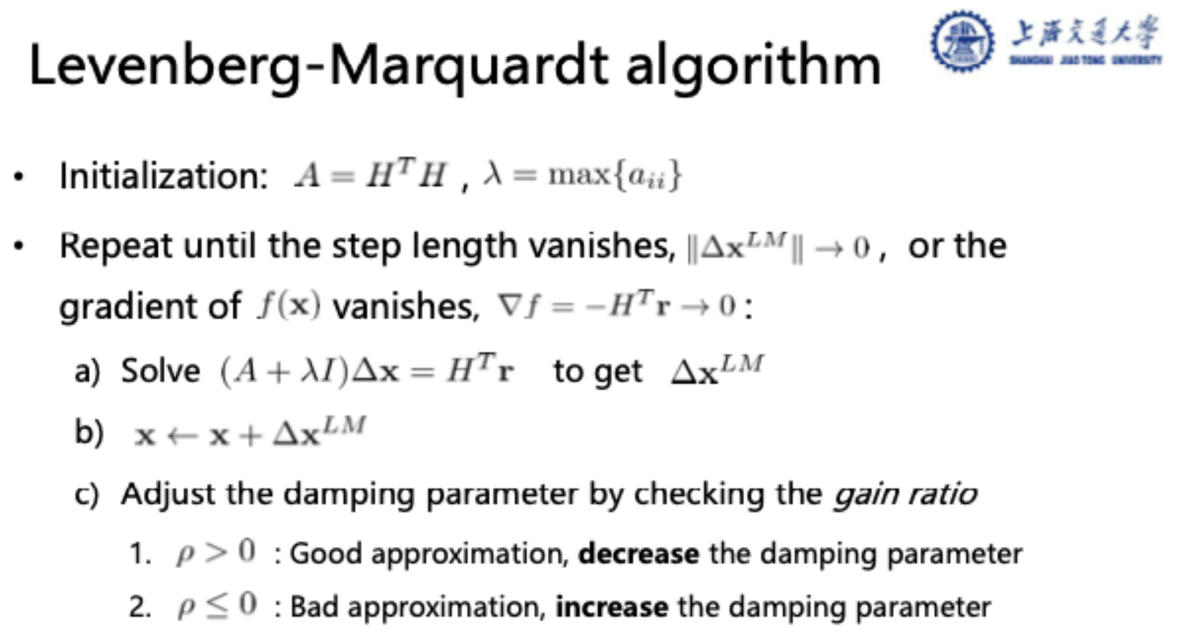
the authors use an initial damping term: $\lambda = \max\{a_{ii}\}$ where $A = H^T H$

the updating of the damping term is controlled by the gain ratio: $\rho = \frac{f(\mathbf{x}+\Delta\mathbf{x}^{LM}) - f(\mathbf{x})}{l(\Delta\mathbf{x}^{LM})}$

Levenberg-Marquardt notes

Danping Zou @Shanghai Jiao Tong University

http://drone.sjtu.edu.cn/dpzou/teaching/course/lecture07-08-nonlinear_least_square_ransac.pdf
EE382-Visual localization & Perception, “Lecture 08- Nonlinear least square & RANSAC”



Levenberg-Marquardt algorithm

- Initialization: $A = H^T H$, $\lambda = \max\{a_{ii}\}$
- Repeat until the step length vanishes, $\|\Delta\mathbf{x}^{LM}\| \to 0$, or the gradient of $f(\mathbf{x})$ vanishes, $\nabla f = -H^T\mathbf{r} \to 0$:
  - a) Solve $(A + \lambda I)\Delta\mathbf{x} = H^T\mathbf{r}$ to get $\Delta\mathbf{x}^{LM}$
  - b) $\mathbf{x} \leftarrow \mathbf{x} + \Delta\mathbf{x}^{LM}$
  - c) Adjust the damping parameter by checking the *gain ratio*
    1. $\rho > 0$: Good approximation, **decrease** the damping parameter
    2. $\rho \leq 0$: Bad approximation, **increase** the damping parameter

Levenberg-Marquardt notes

Danping Zou @Shanghai Jiao Tong University

http://drone.sjtu.edu.cn/dpzou/teaching/course/lecture07-08-nonlinear_least_square_ransac.pdf
EE382-Visual localization & Perception, “Lecture 08- Nonlinear least square & RANSAC”

Parameter perturbations: operator $\boxplus$ represents adding a perturbation to parameters

$$\boxplus : \mathcal{X} \times \mathbb{R}^n \rightarrow \mathcal{X}, \quad (\mathcal{X} - \text{parameter space})$$

if the parameters are vectors: $\mathbf{x} \boxplus \Delta\mathbf{x} = \mathbf{x} + \Delta\mathbf{x}$

if the parameters is a Lie group like rotation: $\mathbf{x} \boxplus \Delta\mathbf{x} = \mathbf{x} \otimes \exp(\Delta\mathbf{x}^{\wedge})$ (Right multiplication)

$$= \exp(\Delta\mathbf{x}^{\wedge}) \otimes \mathbf{x} \quad \text{(Left multiplication)}$$

- When the parameter is represented by a rotation matrix :

$$\mathbf{x} \boxplus \Delta\mathbf{x} \leftrightarrow R \cdot \delta R$$

- Matrix exponential : $A \in \mathbb{R}^{n \times n}$

$$\exp(A) = I + A + \frac{1}{2!}A^2 + \frac{1}{3!}A^3 + \dots$$

- The incremental rotation is computed from the perturbation vector by :

$$\delta R = \exp([\Delta\mathbf{x}]_\times) = I + [\Delta\mathbf{x}]_\times + [\Delta\mathbf{x}]_\times[\Delta\mathbf{x}]_\times + \dots$$
$$\approx I + [\Delta\mathbf{x}]_\times$$

$$\boxed{\mathbf{x} \boxplus \Delta\mathbf{x} \approx R(I + [\Delta\mathbf{x}]_\times)}$$

$\wedge : \mathbb{R}^n \rightarrow so(n)$ is an one-to-one mapping from the perturbation vector space to *lie algebra*.

EE 267 Virtual Reality
Course Notes: 6-DOF Pose Tracking with the VRduino

https://stanford.edu/class/ee267/notes/

Gordon Wetzstein
gordon.wetzstein@stanford.edu

projection from 3D points (x_i, y_i, z_i) *to normalized 2D locations* (x_i^n, y_i^n)

With the homography matrix in hand, our next goal is to estimate the actual translation vector $t_x,\ t_y,\ t_z$. Let's start by repeating how the rotation and translation, i.e. the pose, are related to the scaled homography (see Eq. 2)

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & -1 \end{pmatrix} \begin{pmatrix} r_{11} & r_{12} & t_x \\ r_{21} & r_{22} & t_y \\ r_{31} & r_{32} & t_z \end{pmatrix} = s \begin{pmatrix} h_1 & h_2 & h_3 \\ h_4 & h_5 & h_6 \\ h_7 & h_8 & 1 \end{pmatrix} \tag{11}$$

To estimate the scale factor s we use the insight that any valid rotation matrix has normalized rows and columns, i.e. their length or ℓ_2-norm equals 1. During our homography estimation, we did not actually enforce any normalization on the matrix columns, so let's just impose this constraint now by setting s to the inverse of the average length of the two rotation matrix columns:

$$s = \frac{2}{\sqrt{h_1^2 + h_4^2 + h_7^2} + \sqrt{h_2^2 + h_5^2 + h_8^2}} \tag{12}$$

Multiplying this scale factor with the estimated homography results in the first two columns to be approximately normalized.

Estimating translation from the homography matrix Using the scale factor s and the estimated homography matrix, we can compute the translational component of the pose as

$$t_x = sh_3, \qquad t_y = sh_6, \qquad t_z = -s \tag{13}$$

EE 267 Virtual Reality **https://stanford.edu/class/ee267/notes/ee267_notes_tracking.pdf**

Course Notes: 6-DOF Pose Tracking with the VRduino

Gordon Wetzstein
gordon.wetzstein@stanford.edu

Estimating rotation from the homography matrix We can also compute the full 3×3 rotation matrix from the first two columns of the homography matrix. This is done by orthogonalizing the first two columns of the rotation matrix that we can now easily compute and then by computing the third row using the cross-product of the others.

Specifically, we compute the first column $\mathbf{r}_1$ as

$$\mathbf{r}_1 = \begin{pmatrix} r_{11} \\ r_{21} \\ r_{31} \end{pmatrix} = \begin{pmatrix} \frac{h_1}{\sqrt{h_1^2+h_4^2+h_7^2}} \\ \frac{h_4}{\sqrt{h_1^2+h_4^2+h_7^2}} \\ -\frac{h_7}{\sqrt{h_1^2+h_4^2+h_7^2}} \end{pmatrix} \tag{14}$$

Similarly, we extract the second column of the rotation matrix $\mathbf{r}_2$ from the homography, but we have to make sure that it is orthogonal to the first column. We can enforce that as follows

$$\widetilde{\mathbf{r}}_2 = \begin{pmatrix} r_{12} \\ r_{22} \\ r_{32} \end{pmatrix} = \begin{pmatrix} h_2 \\ h_5 \\ -h_8 \end{pmatrix} - \begin{pmatrix} r_{11}\left(r_{11}h_2 + r_{21}h_5 - r_{31}h_8\right) \\ r_{21}\left(r_{11}h_2 + r_{21}h_5 - r_{31}h_8\right) \\ r_{31}\left(r_{11}h_2 + r_{21}h_5 - r_{31}h_8\right) \end{pmatrix}, \qquad \mathbf{r}_2 = \frac{\widetilde{\mathbf{r}}_2}{\|\widetilde{\mathbf{r}}_2\|_2} \tag{15}$$

Now, $\mathbf{r}_2$ should be normalized and orthogonal to $\mathbf{r}_1$, i.e. $\mathbf{r}_1 \cdot \mathbf{r}_2 = 0$.

Finally, we can recover the missing third column of the rotation matrix using the cross product of the other two

$$\mathbf{r}_3 = \mathbf{r}_1 \times \mathbf{r}_2 = \begin{pmatrix} r_{21}r_{32} - r_{31}r_{22} \\ r_{31}r_{12} - r_{11}r_{32} \\ r_{11}r_{22} - r_{21}r_{12} \end{pmatrix} \tag{16}$$

This gives us the full 3×3 rotation matrix $\mathbf{R} = [\mathbf{r}_1\, \mathbf{r}_2\, \mathbf{r}_3]$.

https://stanford.edu/class/ee267/notes/ee267_notes_tracking.pdf

see the next several slides for the objective and Jacobian details

6.2 Pseudo Code

To help you with the implementation, Algorithm 1 outlines pseudo code for pose tracking with Levenberg-Marquardt.

Algorithm 5 Levenberg-Marquardt for Pose Tracking

```
1: initialize $p$ and $\lambda$          $\boxed{p = (\theta_x, \theta_y, \theta_z, t_x, t_y, t_z)}$   $\boxed{\lambda = \max\{J^\wedge T^* J\}}$
2: for $k = 1$ to $maxIters$
3:     $\mathbf{f} \;\; = \text{eval\_objective}(p)$            // see Eq. 21
4:     $\mathbf{J}_g = \text{compute\_jacobian\_g}(p)$        // see Eqs. 26–34   $\boxed{(2N) \times 9}$
5:     $\mathbf{J}_f = \text{compute\_jacobian\_f}(p)$        // see Eqs. 24–25   $\boxed{9 \times 6}$
6:     $\mathbf{J} \;\; = \mathbf{J}_f \cdot \mathbf{J}_g$                                $\boxed{(2N) \times 6}$
7:     $p \;\; = p + \text{inv}\left(\mathbf{J}^T\mathbf{J} + \lambda\,\text{diag}\left(\mathbf{J}^T\mathbf{J}\right)\right) \cdot (\mathbf{b} - \mathbf{f})$   $\boxed{\text{missing factor } J^\wedge T \text{ before (b-f)}}$
8: end for
```

NOTE: the damping term is used like a factor in the perturbation of a symmetric matrix. see: https://nhigham.com/2021/02/16/diagonally-perturbing-a-symmetric-matrix-to-make-it-positive-definite/

$$\underbrace{\begin{pmatrix} x_1^n \\ y_1^n \\ \vdots \\ x_M^n \\ y_M^n \end{pmatrix}}_{\boxed{\mathbf{b}}} = \underbrace{\begin{pmatrix} x_1 & y_1 & 1 & 0 & 0 & 0 & -x_1\,x_1^n & -y_1\,x_1^n \\ 0 & 0 & 0 & x_1 & y_1 & 1 & -x_1\,y_1^n & -y_1\,y_1^n \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ x_M & y_M & 1 & 0 & 0 & 0 & -x_M\,x_M^n & -y_M\,x_M^n \\ 0 & 0 & 0 & x_M & y_M & 1 & -x_M\,y_M^n & -y_M\,y_M^n \end{pmatrix}}_{\boxed{\mathbf{A}}} \underbrace{\begin{pmatrix} h_1 \\ h_2 \\ h_3 \\ h_4 \\ h_5 \\ h_6 \\ h_7 \\ h_8 \end{pmatrix}}_{\mathbf{h}} \qquad (10)$$

(in camera centered coordinates)

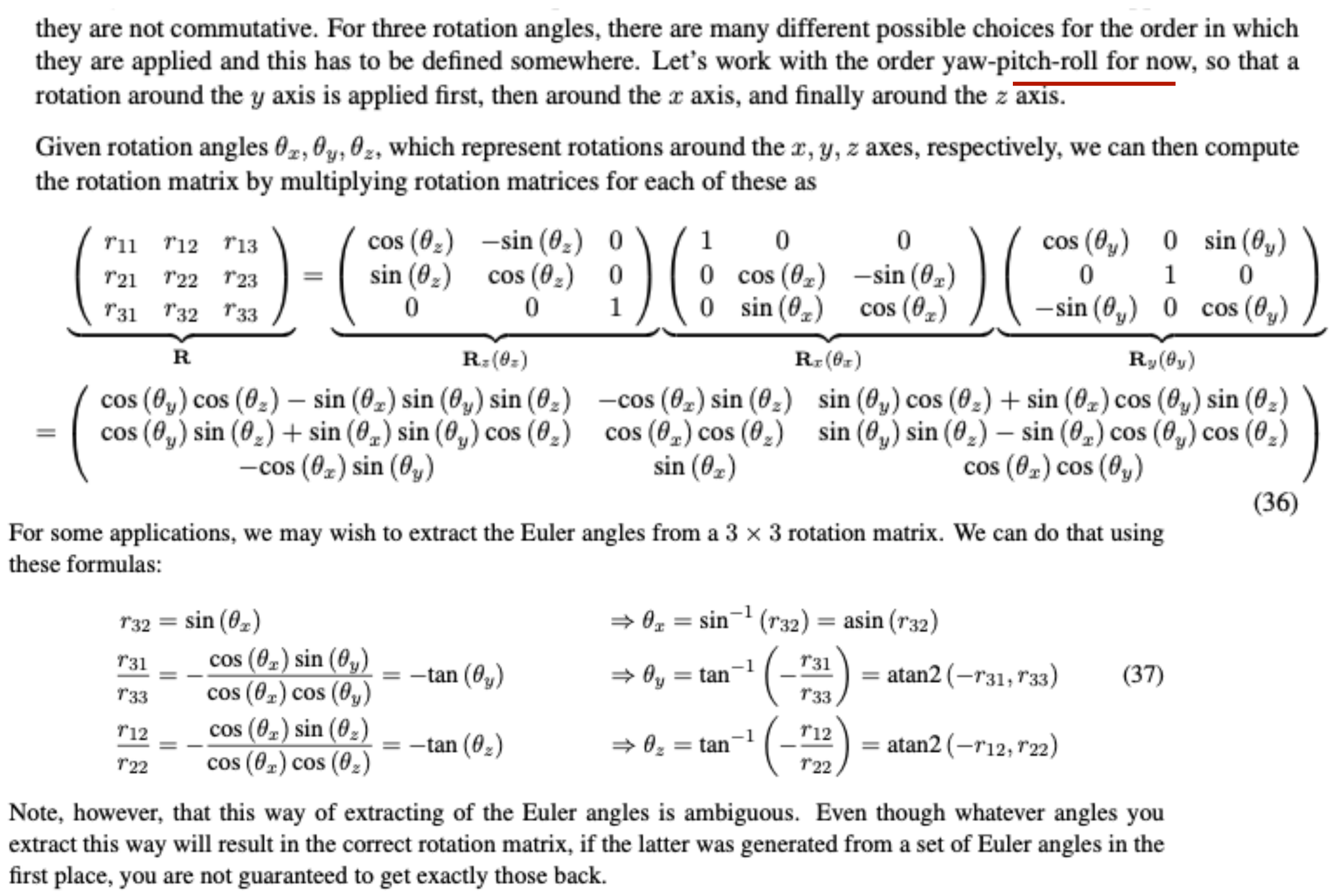
$$\begin{pmatrix} x^c \\ y^c \\ w^c \end{pmatrix}$$

(then normalized))

$$x^n = \frac{x^c}{w^c}$$

https://stanford.edu/class/ee267/notes/ee267_notes_tracking.pdf

rotation matrix details. note that only the first 2 columns are used in the optimization though.



they are not commutative. For three rotation angles, there are many different possible choices for the order in which they are applied and this has to be defined somewhere. Let's work with the order yaw-pitch-roll for now, so that a rotation around the y axis is applied first, then around the x axis, and finally around the z axis.

Given rotation angles $\theta_x, \theta_y, \theta_z$, which represent rotations around the x, y, z axes, respectively, we can then compute the rotation matrix by multiplying rotation matrices for each of these as

$$\underbrace{\begin{pmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{pmatrix}}_{\mathbf{R}} = \underbrace{\begin{pmatrix} \cos(\theta_z) & -\sin(\theta_z) & 0 \\ \sin(\theta_z) & \cos(\theta_z) & 0 \\ 0 & 0 & 1 \end{pmatrix}}_{\mathbf{R}_z(\theta_z)} \underbrace{\begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos(\theta_x) & -\sin(\theta_x) \\ 0 & \sin(\theta_x) & \cos(\theta_x) \end{pmatrix}}_{\mathbf{R}_x(\theta_x)} \underbrace{\begin{pmatrix} \cos(\theta_y) & 0 & \sin(\theta_y) \\ 0 & 1 & 0 \\ -\sin(\theta_y) & 0 & \cos(\theta_y) \end{pmatrix}}_{\mathbf{R}_y(\theta_y)}$$

$$= \begin{pmatrix} \cos(\theta_y)\cos(\theta_z) - \sin(\theta_x)\sin(\theta_y)\sin(\theta_z) & -\cos(\theta_x)\sin(\theta_z) & \sin(\theta_y)\cos(\theta_z) + \sin(\theta_x)\cos(\theta_y)\sin(\theta_z) \\ \cos(\theta_y)\sin(\theta_z) + \sin(\theta_x)\sin(\theta_y)\cos(\theta_z) & \cos(\theta_x)\cos(\theta_z) & \sin(\theta_y)\sin(\theta_z) - \sin(\theta_x)\cos(\theta_y)\cos(\theta_z) \\ -\cos(\theta_x)\sin(\theta_y) & \sin(\theta_x) & \cos(\theta_x)\cos(\theta_y) \end{pmatrix} \tag{36}$$

For some applications, we may wish to extract the Euler angles from a 3×3 rotation matrix. We can do that using these formulas:

$$\begin{aligned} r_{32} &= \sin(\theta_x) & &\Rightarrow \theta_x = \sin^{-1}(r_{32}) = \operatorname{asin}(r_{32}) \\ \frac{r_{31}}{r_{33}} &= -\frac{\cos(\theta_x)\sin(\theta_y)}{\cos(\theta_x)\cos(\theta_y)} = -\tan(\theta_y) & &\Rightarrow \theta_y = \tan^{-1}\left(-\frac{r_{31}}{r_{33}}\right) = \operatorname{atan2}(-r_{31}, r_{33}) \\ \frac{r_{12}}{r_{22}} &= -\frac{\cos(\theta_x)\sin(\theta_z)}{\cos(\theta_x)\cos(\theta_z)} = -\tan(\theta_z) & &\Rightarrow \theta_z = \tan^{-1}\left(-\frac{r_{12}}{r_{22}}\right) = \operatorname{atan2}(-r_{12}, r_{22}) \end{aligned} \tag{37}$$

Note, however, that this way of extracting of the Euler angles is ambiguous. Even though whatever angles you extract this way will result in the correct rotation matrix, if the latter was generated from a set of Euler angles in the first place, you are not guaranteed to get exactly those back.

https://stanford.edu/class/ee267/notes/ee267_notes_tracking.pdf

In this section, we derive a nonlinear optimization approach to pose tracking using the Levenberg-Marquardt (LM) algorithm. The derivation of the LM algorithm can be found in the lecture slides and you can find more details in standard optimization textbooks or on wikipedia[5]. For this approach, we need to specify which representation for the rotations we use. In the following derivations, we use Euler angles in the yaw-pitch-roll order, but you can derive a similar algorithm using quaternions. Using Equations 2 and 36, we can relate the set of pose parameters $\boldsymbol{p} = (\theta_x, \theta_y, \theta_z, t_x, t_y, t_z)$ via the function $g : \mathbb{R}^6 \to \mathbb{R}^9$ to the homography as

$$g(\boldsymbol{p}) = \begin{pmatrix} g_1(\boldsymbol{p}) \\ g_2(\boldsymbol{p}) \\ g_3(\boldsymbol{p}) \\ g_4(\boldsymbol{p}) \\ g_5(\boldsymbol{p}) \\ g_6(\boldsymbol{p}) \\ g_7(\boldsymbol{p}) \\ g_8(\boldsymbol{p}) \\ g_9(\boldsymbol{p}) \end{pmatrix} = \begin{pmatrix} h_1 \\ h_2 \\ h_3 \\ h_4 \\ h_5 \\ h_6 \\ h_7 \\ h_8 \\ h_9 \end{pmatrix} = \begin{pmatrix} \cos(\theta_y)\cos(\theta_z) - \sin(\theta_x)\sin(\theta_y)\sin(\theta_z) \\ -\cos(\theta_x)\sin(\theta_z) \\ t_x \\ \cos(\theta_y)\sin(\theta_z) + \sin(\theta_x)\sin(\theta_y)\cos(\theta_z) \\ \cos(\theta_x)\cos(\theta_z) \\ t_y \\ \cos(\theta_x)\sin(\theta_y) \\ -\sin(\theta_x) \\ -t_z \end{pmatrix} \tag{19}$$

$$= \boxed{\begin{matrix} r11 \\ r12 \\ tx \\ r21 \\ r22 \\ ty \\ -r31 \\ -r32 \\ -tz \end{matrix}}$$

Note that we are using all 9 elements of the homography matrix for this nonlinear approach. The elements of the homography matrix are only used as an intermediate variables. The function $f : \mathbb{R}^9 \to \mathbb{R}^8$ maps them to the projected 2D point coordinates of our 4 reference points as

$$f(\mathbf{h}) = \begin{pmatrix} f_1(\mathbf{h}) \\ f_2(\mathbf{h}) \\ \vdots \\ f_7(\mathbf{h}) \\ f_8(\mathbf{h}) \end{pmatrix} = \begin{pmatrix} x_1^n \\ y_1^n \\ x_2^n \\ y_2^n \\ x_3^n \\ y_3^n \\ x_4^n \\ y_4^n \end{pmatrix} = \begin{pmatrix} \frac{h_1x_1+h_2y_1+h_3}{h_7x_1+h_8y_1+h_9} \\ \frac{h_4x_1+h_5y_1+h_6}{h_7x_1+h_8y_1+h_9} \\ \vdots \\ \frac{h_1x_4+h_2y_4+h_3}{h_7x_4+h_8y_4+h_9} \\ \frac{h_4x_4+h_5y_4+h_6}{h_7x_4+h_8y_4+h_9} \end{pmatrix} \tag{20}$$

https://stanford.edu/class/ee267/notes/ee267_notes_tracking.pdf

Equations 19 and 20 model the same image formation that we have been using throughout the document. The objective function that we are trying to minimize is

$$\begin{aligned} &\underset{\{\boldsymbol{p}\}}{\text{minimize}}\ \|\mathbf{b} - f\left(g\left(\boldsymbol{p}\right)\right)\|_2^2 \\ &= \left(x_1^n - f_1\left(g\left(\boldsymbol{p}\right)\right)\right)^2 + \left(y_1^n - f_2\left(g\left(\boldsymbol{p}\right)\right)\right)^2 + \ldots + \left(x_4^n - f_7\left(g\left(\boldsymbol{p}\right)\right)\right)^2 + \left(y_4^n - f_8\left(g\left(\boldsymbol{p}\right)\right)\right)^2 \end{aligned} \tag{21}$$

The Levenberg-Marquardt algorithm is an iterative method that starts from some initial guess $\boldsymbol{p}^{(0)}$ and then updates it as

$$\boldsymbol{p}^{(k)} = \boldsymbol{p}^{(k-1)} + \left(\mathbf{J}^T\mathbf{J} + \lambda \text{diag}\left(\mathbf{J}^T\mathbf{J}\right)\right)^{-1}\left(\mathbf{b} - f\left(g\left(\boldsymbol{p}\right)\right)\right) \tag{22}$$

For these updates, we just need $\boldsymbol{p}^{(0)}$ as well as a user-defined parameter λ. The Jacobian matrix $\mathbf{J} \in \mathbb{R}^{8\times 6}$ includes the partial derivatives of the image formation model.

https://stanford.edu/class/ee267/notes/ee267_notes_tracking.pdf

chain rule to assemble $\mathbf{J}$ from the individual Jacobian matrices as

$$\frac{\partial}{\partial \boldsymbol{p}} f\left(g\left(\boldsymbol{p}\right)\right) = \mathbf{J} = \mathbf{J}_f \cdot \mathbf{J}_g = \begin{pmatrix} \frac{\partial f_1}{\partial h_1} & \cdots & \frac{\partial f_1}{\partial h_9} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_8}{\partial h_1} & \cdots & \frac{\partial f_8}{\partial h_9} \end{pmatrix} \begin{pmatrix} \frac{\partial g_1}{\partial p_1} & \cdots & \frac{\partial g_1}{\partial p_6} \\ \vdots & \ddots & \vdots \\ \frac{\partial g_9}{\partial p_1} & \cdots & \frac{\partial g_9}{\partial p_6} \end{pmatrix} \tag{23}$$

where $\mathbf{J}_f \in \mathbb{R}^{8\times 9}$ is the Jacobian matrix of the function $f\left(\mathbf{h}\right)$ and $\mathbf{J}_g \in \mathbb{R}^{9\times 6}$ is the Jacobian matrix of the function $g\left(\boldsymbol{p}\right)$.

You should derive the partial derivatives yourself, but for completeness we list them in the following. If you want to derive these yourself or you just want to verify the equations below, take a really close look at Equation 19 when deriving the entries of $\mathbf{J}_g$ and Equation 20 when deriving the entries of $\mathbf{J}_f$.

The formulas to compute the entries of the first row of the Jacobian matrix $\mathbf{J}_f$ are

$$\begin{aligned} &\frac{\partial f_1}{\partial h_1} = \frac{x_1}{h_7 x_1 + h_8 y_1 + h_9}, && \frac{\partial f_1}{\partial h_2} = \frac{y_1}{h_7 x_1 + h_8 y_1 + h_9}, && \frac{\partial f_1}{\partial h_3} = \frac{1}{h_7 x_1 + h_8 y_1 + h_9}, \\ &\frac{\partial f_1}{\partial h_4} = 0, && \frac{\partial f_1}{\partial h_5} = 0, && \frac{\partial f_1}{\partial h_6} = 0, \\ &\frac{\partial f_1}{\partial h_7} = -\left(\frac{h_1 x_1 + h_2 y_1 + h_3}{\left(h_7 x_1 + h_8 y_1 + h_9\right)^2}\right) x_1, && \frac{\partial f_1}{\partial h_8} = -\left(\frac{h_1 x_1 + h_2 y_1 + h_3}{\left(h_7 x_1 + h_8 y_1 + h_9\right)^2}\right) y_1, && \frac{\partial f_1}{\partial h_9} = -\left(\frac{h_1 x_1 + h_2 y_1 + h_3}{\left(h_7 x_1 + h_8 y_1 + h_9\right)^2}\right) \end{aligned} \tag{24}$$

https://stanford.edu/class/ee267/notes/ee267_notes_tracking.pdf

The entries of the second row are

$$\begin{aligned}
&\frac{\partial f_2}{\partial h_1} = 0, && \frac{\partial f_2}{\partial h_2} = 0, && \frac{\partial f_2}{\partial h_3} = 0, \\
&\frac{\partial f_2}{\partial h_4} = \frac{x_1}{h_7x_1+h_8y_1+h_9}, && \frac{\partial f_2}{\partial h_5} = \frac{y_1}{h_7x_1+h_8y_1+h_9}, && \frac{\partial f_2}{\partial h_6} = \frac{1}{h_7x_1+h_8y_1+h_9}, \\
&\frac{\partial f_2}{\partial h_7} = -\left(\frac{h_4x_1+h_5y_1+h_6}{(h_7x_1+h_8y_1+h_9)^2}\right)x_1, && \frac{\partial f_2}{\partial h_8} = -\left(\frac{h_4x_1+h_5y_1+h_6}{(h_7x_1+h_8y_1+h_9)^2}\right)y_1, && \frac{\partial f_2}{\partial h_9} = -\left(\frac{h_4x_1+h_5y_1+h_6}{(h_7x_1+h_8y_1+h_9)^2}\right)
\end{aligned} \tag{25}$$

The remaining 6 rows of J_f can be computed using the same pattern, only the coordinates x_i, y_i have to be adjusted.

https://stanford.edu/class/ee267/notes/ee267_notes_tracking.pdf

Similarly, here are formulas to compute the entries of the first row of the Jacobian matrix $\mathbf{J}_g$

$$\begin{aligned}
\tfrac{\partial g_1}{\partial p_1} &= -\cos(\theta_x)\sin(\theta_y)\sin(\theta_z)\,,\\
\tfrac{\partial g_1}{\partial p_2} &= -\sin(\theta_y)\cos(\theta_z) - \sin(\theta_x)\cos(\theta_y)\sin(\theta_z)\,,\\
\tfrac{\partial g_1}{\partial p_3} &= -\cos(\theta_y)\sin(\theta_z) - \sin(\theta_x)\sin(\theta_y)\cos(\theta_z)\,,\\
\tfrac{\partial g_1}{\partial p_4} &= 0, \quad \tfrac{\partial g_1}{\partial p_5} = 0, \quad \tfrac{\partial g_1}{\partial p_6} = 0
\end{aligned} \tag{26}$$

the entries of the second row

$$\frac{\partial g_2}{\partial p_1} = \sin(\theta_x)\sin(\theta_z)\,, \quad \frac{\partial g_2}{\partial p_2} = 0, \quad \frac{\partial g_2}{\partial p_3} = -\cos(\theta_x)\cos(\theta_z)\,, \quad \frac{\partial g_2}{\partial p_4} = 0, \quad \frac{\partial g_2}{\partial p_5} = 0, \quad \frac{\partial g_2}{\partial p_6} = 0 \tag{27}$$

the third row

$$\frac{\partial g_3}{\partial p_1} = 0, \quad \frac{\partial g_3}{\partial p_2} = 0, \quad \frac{\partial g_3}{\partial p_3} = 0, \quad \frac{\partial g_3}{\partial p_4} = 1, \quad \frac{\partial g_3}{\partial p_5} = 0, \quad \frac{\partial g_3}{\partial p_6} = 0 \tag{28}$$

the fourth row

$$\begin{aligned}
\tfrac{\partial g_4}{\partial p_1} &= \cos(\theta_x)\sin(\theta_y)\cos(\theta_z)\,,\\
\tfrac{\partial g_4}{\partial p_2} &= -\sin(\theta_y)\sin(\theta_z) + \sin(\theta_x)\cos(\theta_y)\cos(\theta_z)\,,\\
\tfrac{\partial g_4}{\partial p_3} &= \cos(\theta_y)\cos(\theta_z) - \sin(\theta_x)\sin(\theta_y)\sin(\theta_z)\,,\\
\tfrac{\partial g_4}{\partial p_4} &= 0, \quad \tfrac{\partial g_4}{\partial p_5} = 0, \quad \tfrac{\partial g_4}{\partial p_6} = 0
\end{aligned} \tag{29}$$

https://stanford.edu/class/ee267/notes/ee267_notes_tracking.pdf

the fifth row

$$\frac{\partial g_5}{\partial p_1} = -\sin(\theta_x)\cos(\theta_z), \quad \frac{\partial g_5}{\partial p_2} = 0, \quad \frac{\partial g_5}{\partial p_3} = -\cos(\theta_x)\sin(\theta_z), \quad \frac{\partial g_5}{\partial p_4} = 0, \quad \frac{\partial g_5}{\partial p_5} = 0, \quad \frac{\partial g_5}{\partial p_6} = 0 \tag{30}$$

the sixth row

$$\frac{\partial g_6}{\partial p_1} = 0, \quad \frac{\partial g_6}{\partial p_2} = 0, \quad \frac{\partial g_6}{\partial p_3} = 0, \quad \frac{\partial g_6}{\partial p_4} = 0, \quad \frac{\partial g_6}{\partial p_5} = 1, \quad \frac{\partial g_6}{\partial p_6} = 0 \tag{31}$$

the seventh row

$$\frac{\partial g_7}{\partial p_1} = -\sin(\theta_x)\sin(\theta_y), \quad \frac{\partial g_7}{\partial p_2} = \cos(\theta_x)\cos(\theta_y), \quad \frac{\partial g_7}{\partial p_3} = 0, \quad \frac{\partial g_7}{\partial p_4} = 0, \quad \frac{\partial g_7}{\partial p_5} = 0, \quad \frac{\partial g_7}{\partial p_6} = 0 \tag{32}$$

the eigth row

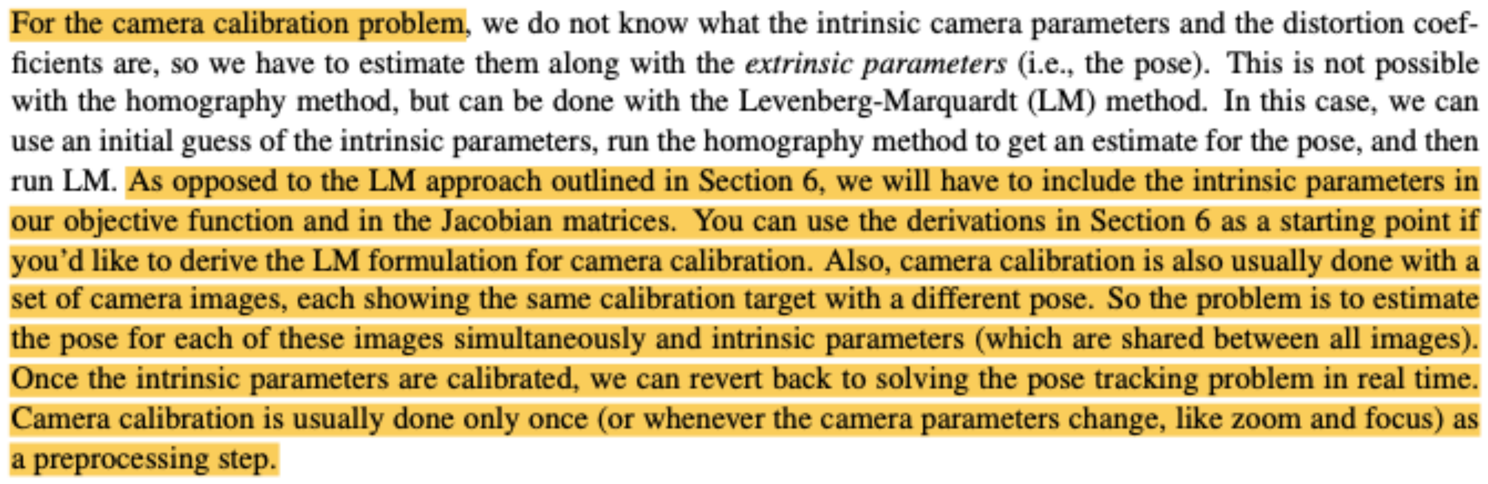
$$\frac{\partial g_8}{\partial p_1} = -\cos(\theta_x), \quad \frac{\partial g_8}{\partial p_2} = 0, \quad \frac{\partial g_8}{\partial p_3} = 0, \quad \frac{\partial g_8}{\partial p_4} = 0, \quad \frac{\partial g_8}{\partial p_5} = 0, \quad \frac{\partial g_8}{\partial p_6} = 0 \tag{33}$$

and the ninth row

$$\frac{\partial g_9}{\partial p_1} = 0, \quad \frac{\partial g_9}{\partial p_2} = 0, \quad \frac{\partial g_9}{\partial p_3} = 0, \quad \frac{\partial g_9}{\partial p_4} = 0, \quad \frac{\partial g_9}{\partial p_5} = 0, \quad \frac{\partial g_9}{\partial p_6} = -1 \tag{34}$$

With these partial derivatives, we are now ready to implement the entire nonlinear algorithm for pose estimation.

https://stanford.edu/class/ee267/notes/ee267_notes_tracking.pdf



For the camera calibration problem, we do not know what the intrinsic camera parameters and the distortion coefficients are, so we have to estimate them along with the *extrinsic parameters* (i.e., the pose). This is not possible with the homography method, but can be done with the Levenberg-Marquardt (LM) method. In this case, we can use an initial guess of the intrinsic parameters, run the homography method to get an estimate for the pose, and then run LM. As opposed to the LM approach outlined in Section 6, we will have to include the intrinsic parameters in our objective function and in the Jacobian matrices. You can use the derivations in Section 6 as a starting point if you'd like to derive the LM formulation for camera calibration. Also, camera calibration is also usually done with a set of camera images, each showing the same calibration target with a different pose. So the problem is to estimate the pose for each of these images simultaneously and intrinsic parameters (which are shared between all images). Once the intrinsic parameters are calibrated, we can revert back to solving the pose tracking problem in real time. Camera calibration is usually done only once (or whenever the camera parameters change, like zoom and focus) as a preprocessing step.

Levenberg-Marquardt notes

```
from Szeliski 2010 Chap 6.
```

Transformation	Matrix	# DoF	Preserves	Icon
translation	$\left[\, \boldsymbol{I} \mid \boldsymbol{t} \,\right]_{2\times 3}$	2	orientation	□
rigid (Euclidean)	$\left[\, \boldsymbol{R} \mid \boldsymbol{t} \,\right]_{2\times 3}$	3	lengths	◇
similarity	$\left[\, s\boldsymbol{R} \mid \boldsymbol{t} \,\right]_{2\times 3}$	4	angles	◇
affine	$\left[\, \boldsymbol{A} \,\right]_{2\times 3}$	6	parallelism	▱
projective	$\left[\, \tilde{\boldsymbol{H}} \,\right]_{3\times 3}$	8	straight lines	⏢

Table 2.1 Hierarchy of 2D coordinate transformations. Each transformation also preserves the properties listed in the rows below it, i.e., similarity preserves not only angles but also parallelism and straight lines. The 2×3 matrices are extended with a third $[\mathbf{0}^T \; 1]$ row to form a full 3×3 matrix for homogeneous coordinate transformations.

Transform	Matrix	Parameters $\boldsymbol{p}$	Jacobian $\boldsymbol{J}$
translation	$\begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \end{bmatrix}$	(t_x, t_y)	$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$
Euclidean	$\begin{bmatrix} c_\theta & -s_\theta & t_x \\ s_\theta & c_\theta & t_y \end{bmatrix}$	(t_x, t_y, θ)	$\begin{bmatrix} 1 & 0 & -s_\theta x - c_\theta y \\ 0 & 1 & c_\theta x - s_\theta y \end{bmatrix}$
similarity	$\begin{bmatrix} 1+a & -b & t_x \\ b & 1+a & t_y \end{bmatrix}$	(t_x, t_y, a, b)	$\begin{bmatrix} 1 & 0 & x & -y \\ 0 & 1 & y & x \end{bmatrix}$
affine	$\begin{bmatrix} 1+a_{00} & a_{01} & t_x \\ a_{10} & 1+a_{11} & t_y \end{bmatrix}$	$(t_x, t_y, a_{00}, a_{01}, a_{10}, a_{11})$	$\begin{bmatrix} 1 & 0 & x & y & 0 & 0 \\ 0 & 1 & 0 & 0 & x & y \end{bmatrix}$
projective	$\begin{bmatrix} 1+h_{00} & h_{01} & h_{02} \\ h_{10} & 1+h_{11} & h_{12} \\ h_{20} & h_{21} & 1 \end{bmatrix}$	$(h_{00}, h_{01}, \ldots, h_{21})$	(see Section 6.1.3)

Table 6.1 Jacobians of the 2D coordinate transformations $\boldsymbol{x}' = \boldsymbol{f}(\boldsymbol{x}; \boldsymbol{p})$ shown in Table 2.1, where we have re-parameterized the motions so that they are identity for $\boldsymbol{p} = 0$.

Definition 1.12 (Jacobian). The *Jacobian* of $f : \mathbb{R}^n \to \mathbb{R}^m$ is the matrix $Df(\vec{x}) \in \mathbb{R}^{m\times n}$ with entries

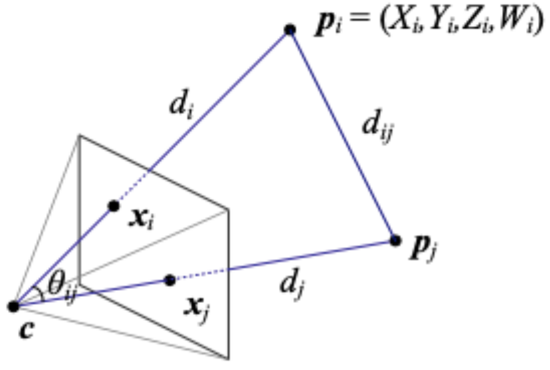
$$(Df)_{ij} \equiv \frac{\partial f_i}{\partial x_j}.$$

Example 1.21 (Jacobian computation). Suppose $f(x, y) = (3x, -xy^2, x + y)$. Then,

$$Df(x, y) = \begin{pmatrix} 3 & 0 \\ -y^2 & -2xy \\ 1 & 1 \end{pmatrix}.$$

Levenberg-Marquardt notes

`from Szeliski 2010 Chap 6.`



the camera matrix form of perspective projection

$$x_i = \frac{p_{00}X_i + p_{01}Y_i + p_{02}Z_i + p_{03}}{p_{20}X_i + p_{21}Y_i + p_{22}Z_i + p_{23}}$$

$$y_i = \frac{p_{10}X_i + p_{11}Y_i + p_{12}Z_i + p_{13}}{p_{20}X_i + p_{21}Y_i + p_{22}Z_i + p_{23}}$$

the projection equations

$$\boldsymbol{x}_i = \boldsymbol{f}(\boldsymbol{p}_i; \boldsymbol{R}, \boldsymbol{t}, \boldsymbol{K})$$

where (x_i, y_i) are the measured 2D feature locations and (X_i, Y_i, Z_i) are the known 3D feature locations

direction $\hat{\boldsymbol{x}}_i = \boldsymbol{K}^{-1}\boldsymbol{x}_i / \|\boldsymbol{K}^{-1}\boldsymbol{x}_i\|$

measured location $\hat{\boldsymbol{x}}_i'$

current *predicted* location

$$\tilde{\boldsymbol{x}}_i' = \boldsymbol{f}(\boldsymbol{x}_i; \boldsymbol{p}).$$

where $\boldsymbol{r}_i = \tilde{\boldsymbol{x}}_i - \hat{\boldsymbol{x}}_i$ is the current residual vector (2D error in predicted position) the projection equations $\boldsymbol{x}_i = \boldsymbol{f}(\boldsymbol{p}_i; \boldsymbol{R}, \boldsymbol{t}, \boldsymbol{K})$

where **p_i is**

Levenberg-Marquardt notes

`from Szeliski 2010 Chap 6.`

this transformation, given in Table 6.1, depends on the current value of θ. Notice how in Table 6.1, we have re-parameterized the motion matrices so that they are always the identity at the origin $\boldsymbol{p} = 0$, which makes it easier to initialize the motion parameters.

To minimize the non-linear least squares problem, we iteratively find an update $\Delta\boldsymbol{p}$ to the current parameter estimate $\boldsymbol{p}$ by minimizing

$$
\begin{aligned}
E_{\text{NLS}}(\Delta\boldsymbol{p}) &= \sum_i \|\boldsymbol{f}(\boldsymbol{x}_i; \boldsymbol{p} + \Delta\boldsymbol{p}) - \boldsymbol{x}'_i\|^2 && (6.13)\\
&\approx \sum_i \|\boldsymbol{J}(\boldsymbol{x}_i; \boldsymbol{p})\Delta\boldsymbol{p} - \boldsymbol{r}_i\|^2 && (6.14)\\
&= \overset{\boxed{\text{1X6}}}{\Delta\boldsymbol{p}^T} \left[\overset{\boxed{\text{6X6}}}{\sum_i \boldsymbol{J}^T\boldsymbol{J}} \right] \overset{\boxed{\text{6X1}}}{\Delta\boldsymbol{p}} - 2\overset{\boxed{\text{1X6}}}{\Delta\boldsymbol{p}^T} \left[\overset{\boxed{\text{6X1}}}{\sum_i \boldsymbol{J}^T\boldsymbol{r}_i} \right] + \overset{\boxed{\text{1X1}}}{\sum_i \|\boldsymbol{r}_i\|^2} && (6.15)\\
&= \Delta\boldsymbol{p}^T\boldsymbol{A}\Delta\boldsymbol{p} - 2\Delta\boldsymbol{p}^T\boldsymbol{b} + c, && (6.16)
\end{aligned}
$$

where the "Hessian"[5] $\boldsymbol{A}$ is the same as Equation (6.9) and the right hand side vector

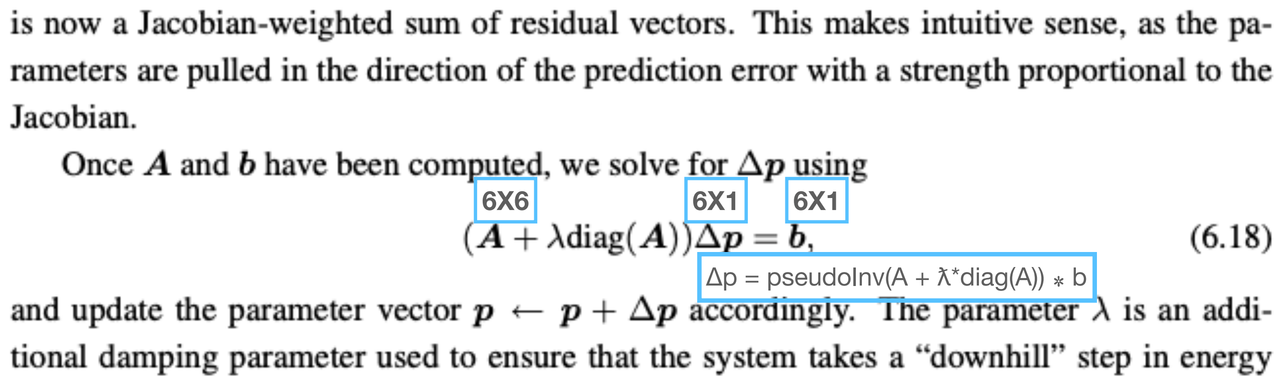
$$\boldsymbol{b} = \sum_i \underset{\boxed{\text{6X1}}}{\boldsymbol{J}^T(\boldsymbol{x}_i)\boldsymbol{r}_i} \qquad (6.17)$$

[5] The "Hessian" $\boldsymbol{A}$ is not the true Hessian (second derivative) of the non-linear least squares problem (6.13). Instead, it is the approximate Hessian, which neglects second (and higher) order derivatives of $\boldsymbol{f}(\boldsymbol{x}_i; \boldsymbol{p} + \Delta\boldsymbol{p})$.

$$\mathbf{c} = \sum_i \|\boldsymbol{r}_i\|^2 \qquad\qquad \boldsymbol{A} = \sum_i \overset{\boxed{\text{6X6}}}{\boldsymbol{J}^T(\boldsymbol{x}_i)\boldsymbol{J}(\boldsymbol{x}_i)} \qquad (6.9)$$

<u>Levenberg-Marquardt notes</u>

`from Szeliski 2010 Chap 6.`



is now a Jacobian-weighted sum of residual vectors. This makes intuitive sense, as the parameters are pulled in the direction of the prediction error with a strength proportional to the Jacobian.

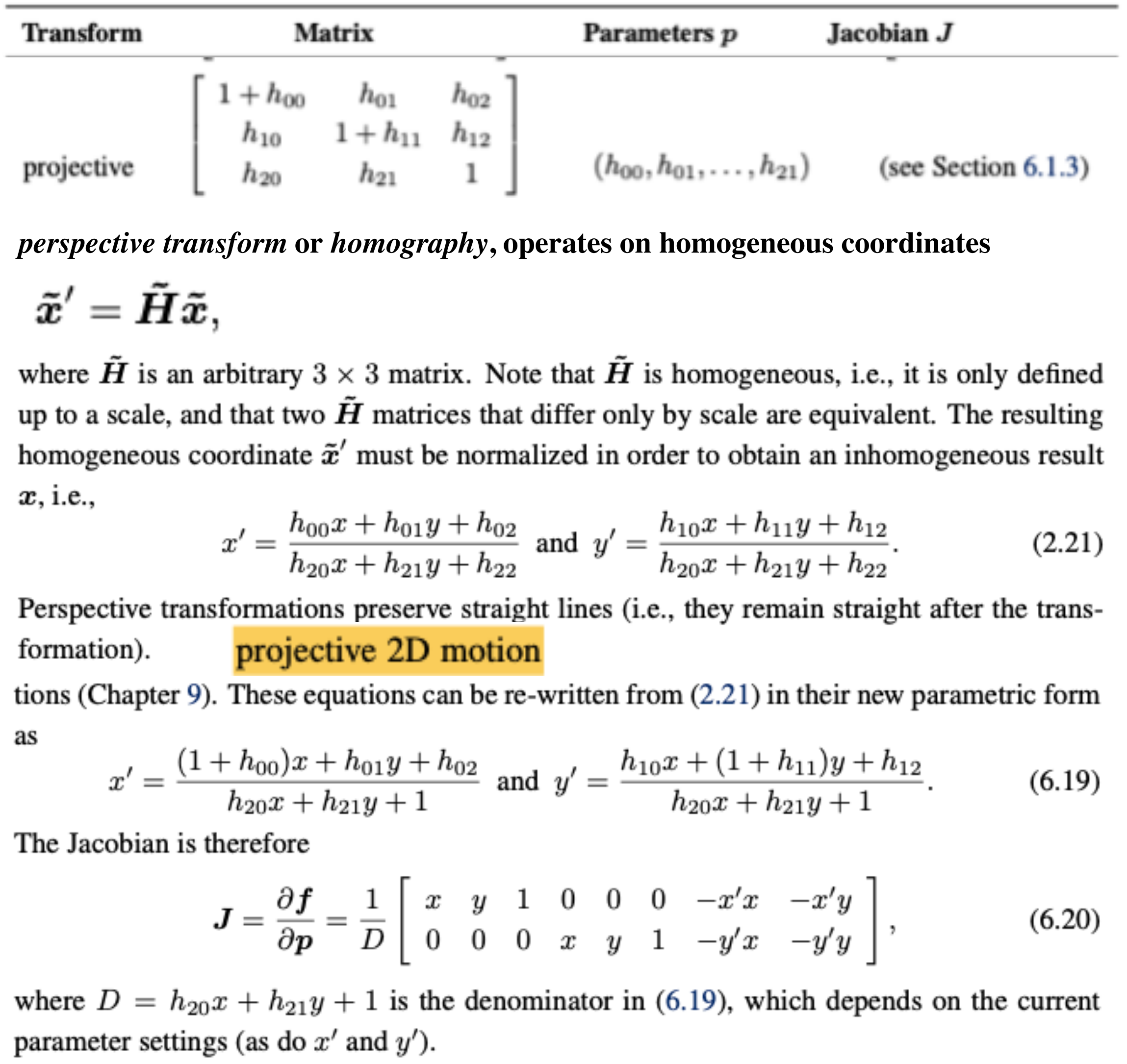
Once $\boldsymbol{A}$ and $\boldsymbol{b}$ have been computed, we solve for $\Delta\boldsymbol{p}$ using

$$(\overset{\text{6X6}}{\boldsymbol{A}} + \lambda \operatorname{diag}(\boldsymbol{A}))\overset{\text{6X1}}{\Delta\boldsymbol{p}} = \overset{\text{6X1}}{\boldsymbol{b}}, \tag{6.18}$$

$\boxed{\Delta p = \text{pseudoInv}(A + \lambda\text{*diag}(A)) * b}$

and update the parameter vector $\boldsymbol{p} \leftarrow \boldsymbol{p} + \Delta\boldsymbol{p}$ accordingly. The parameter λ is an additional damping parameter used to ensure that the system takes a "downhill" step in energy

from Szeliski Chap 6 on iterative non-linear least squares:



Transform	Matrix	Parameters p	Jacobian J
projective	$\begin{bmatrix} 1+h_{00} & h_{01} & h_{02} \\ h_{10} & 1+h_{11} & h_{12} \\ h_{20} & h_{21} & 1 \end{bmatrix}$	$(h_{00}, h_{01}, \ldots, h_{21})$	(see Section 6.1.3)

***perspective transform* or *homography*, operates on homogeneous coordinates**

$$\tilde{\boldsymbol{x}}' = \tilde{\boldsymbol{H}}\tilde{\boldsymbol{x}},$$

where $\tilde{\boldsymbol{H}}$ is an arbitrary 3×3 matrix. Note that $\tilde{\boldsymbol{H}}$ is homogeneous, i.e., it is only defined up to a scale, and that two $\tilde{\boldsymbol{H}}$ matrices that differ only by scale are equivalent. The resulting homogeneous coordinate $\tilde{\boldsymbol{x}}'$ must be normalized in order to obtain an inhomogeneous result $\boldsymbol{x}$, i.e.,

$$x' = \frac{h_{00}x + h_{01}y + h_{02}}{h_{20}x + h_{21}y + h_{22}} \quad \text{and} \quad y' = \frac{h_{10}x + h_{11}y + h_{12}}{h_{20}x + h_{21}y + h_{22}}. \tag{2.21}$$

Perspective transformations preserve straight lines (i.e., they remain straight after the transformation). projective 2D motion

tions (Chapter 9). These equations can be re-written from (2.21) in their new parametric form as

$$x' = \frac{(1+h_{00})x + h_{01}y + h_{02}}{h_{20}x + h_{21}y + 1} \quad \text{and} \quad y' = \frac{h_{10}x + (1+h_{11})y + h_{12}}{h_{20}x + h_{21}y + 1}. \tag{6.19}$$

The Jacobian is therefore

$$\boldsymbol{J} = \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{p}} = \frac{1}{D}\begin{bmatrix} x & y & 1 & 0 & 0 & 0 & -x'x & -x'y \\ 0 & 0 & 0 & x & y & 1 & -y'x & -y'y \end{bmatrix}, \tag{6.20}$$

where $D = h_{20}x + h_{21}y + 1$ is the denominator in (6.19), which depends on the current parameter settings (as do x' and y').

from Szeliski Chap 6 on iterative non-linear least squares:

An initial guess for the eight unknowns $\{h_{00}, h_{01}, \ldots, h_{21}\}$

The most principled way to do the estimation, however, is to directly minimize the squared residual equations (6.13) using the Gauss–Newton approximation, i.e., performing a first-order Taylor series expansion in $\boldsymbol{p}$, as shown in (6.14), which yields the set of equations

$$\begin{bmatrix} \hat{x}' - \tilde{x}' \\ \hat{y}' - \tilde{y}' \end{bmatrix} = \frac{1}{D} \begin{bmatrix} x & y & 1 & 0 & 0 & 0 & -\tilde{x}'x & -\tilde{x}'y \\ 0 & 0 & 0 & x & y & 1 & -\tilde{y}'x & -\tilde{y}'y \end{bmatrix} \begin{bmatrix} \Delta h_{00} \\ \vdots \\ \Delta h_{21} \end{bmatrix}. \tag{6.23}$$

While these look similar to (6.22), they differ in two important respects. First, the left hand side consists of unweighted *prediction errors* rather than point displacements and the solution vector is a *perturbation* to the parameter vector $\boldsymbol{p}$. Second, the quantities inside $\boldsymbol{J}$ involve *predicted* feature locations $(\tilde{x}', \tilde{y}')$ instead of *sensed* feature locations $(\hat{x}', \hat{y}')$. Both of these differences are subtle and yet they lead to an algorithm that, when combined with proper checking for downhill steps (as in the Levenberg–Marquardt algorithm), will converge to a local minimum. Note that iterating Equations (6.22) is not guaranteed to converge, since it is not minimizing a well-defined energy function.

measured location $\hat{\boldsymbol{x}}'_i$

current *predicted* location

$$\tilde{\boldsymbol{x}}'_i = \boldsymbol{f}(\boldsymbol{x}_i; \boldsymbol{p}).$$

notes for partial derivatives of the intrinsic and extrinsic camera parameters

EFFICIENT OPTIMIZATION FOR ROBUST BUNDLE ADJUSTMENT

handed in
MASTER'S THESIS

Master of Science Zhongnan Qu

m: number of feature points P_i
n: number of images

$\{\boldsymbol{x}_i\}_{i=1}^{m} \subset \mathbb{R}^3$: 3D coordinates of feature points. 3X1

$\{(\boldsymbol{\phi}_j, \boldsymbol{t}_j)\}_{j=1}^{n} \subset \mathrm{SE}(3)$: camera pose parameters, rotation angle vector and translation vector.

3D rotation vector $\boldsymbol{\phi}$ to define a rotation transformation. Its magnitude, $\|\boldsymbol{\phi}\|$, defines the rotation angle; and its unit orientation vector, $\boldsymbol{\phi}/\|\boldsymbol{\phi}\|$, defines the rotation axis.

$$\boldsymbol{R}_j = (\mathbf{I} - \boldsymbol{n}_{\phi_j}\boldsymbol{n}_{\phi_j}^T)\cos(\|\boldsymbol{\phi}_j\|) + \widehat{\boldsymbol{n}}_{\phi_j}\sin(\|\boldsymbol{\phi}_j\|) + \boldsymbol{n}_{\phi_j}\boldsymbol{n}_{\phi_j}^T .$$

$$\boldsymbol{y}_j = \begin{pmatrix} y_{j,1} \\ y_{j,2} \\ y_{j,3} \end{pmatrix} = \begin{pmatrix} f_j \\ k_{1,j} \\ k_{2,j} \end{pmatrix}$$

$\underline{\mathrm{s} = 9\mathrm{n} + 3\mathrm{m}}$: the total number of parameters

- $(i, j) \in \mathcal{S} \subset \{1, ..., \mathrm{m}\} \times \{1, ..., \mathrm{n}\}$ if and only if the i-th feature point P_i is observed in the j-th frame. $\mathcal{S}$ is defined as the observation field of the dataset.
- $\mathrm{l} = \|\mathcal{S}\|$: the total number of observations. **l <= m*n**
- $\{\mathbf{b}_{ij}\}_{(i,j)\in\mathcal{S}} \subset \mathbb{R}^2$: 2D coordinates of the observations.

3X1 9X1

$$\boldsymbol{c}'_{ij} = \Pi(\boldsymbol{x}_i, \boldsymbol{\phi}_j, \boldsymbol{t}_j, \boldsymbol{y}_j)$$

a single re-projection error for one feature point Pi in j-th image frame is the difference between its observed position and its re-projected position:

$$\boldsymbol{F}_{ij} = \boldsymbol{c}'_{ij} - \mathbf{b}_{ij} = \begin{pmatrix} c'_{ij,\mathrm{x}} \\ c'_{ij,\mathrm{y}} \end{pmatrix} - \begin{pmatrix} \mathrm{b}_{ij,\mathrm{x}} \\ \mathrm{b}_{ij,\mathrm{y}} \end{pmatrix}$$ 2X1 or 3X1

up to [2*m*n×1]

$$\boldsymbol{F} = \begin{pmatrix} \boldsymbol{F}_{11} & \boldsymbol{F}_{12} & \dots & \boldsymbol{F}_{1\mathrm{n}} & \boldsymbol{F}_{21} & \dots & \boldsymbol{F}_{ij} & \dots & \boldsymbol{F}_{\mathrm{mn}} \end{pmatrix}^T_{(i,j)\in\mathcal{S}} . \quad (2.25)$$

step $\boldsymbol{p}^k$ related to current point $\boldsymbol{u}^k$. $\boldsymbol{p}^k$ has a similar structure as $\boldsymbol{u}^k$,

$$\boldsymbol{p}^k = \{\{\boldsymbol{\delta\phi}_j^k\}_{j=1}^{\mathrm{n}}, \{\boldsymbol{\delta t}_j^k\}_{j=1}^{\mathrm{n}}, \{\boldsymbol{\delta y}_j^k\}_{j=1}^{\mathrm{n}}, \{\boldsymbol{\delta x}_i^k\}_{i=1}^{\mathrm{m}}\} . \quad (2.34)$$

If the real objective function is reduced with this step, the increment step and the trust region radius are accepted, i.e. $\boldsymbol{u}^{k+1} = \boldsymbol{u}^k + \boldsymbol{p}^k$. If the candidate solution does not produce a sufficient decrease, which means the trust region is too large, it is shrunken and resolved [Hau07b].

$$\boldsymbol{u} = \{\{\boldsymbol{\phi}_j\}_{j=1}^{\mathrm{n}}, \{\boldsymbol{t}_j\}_{j=1}^{\mathrm{n}}, \{\boldsymbol{y}_j\}_{j=1}^{\mathrm{n}}, \{\boldsymbol{x}_i\}_{i=1}^{\mathrm{m}}\} , \quad (2.23)$$

more specifically,

$$\boldsymbol{u} = \begin{pmatrix} \boldsymbol{\phi}_1 \\ \boldsymbol{t}_1 \\ \boldsymbol{y}_1 \\ \vdots \\ \boldsymbol{\phi}_j \\ \boldsymbol{t}_j \\ \boldsymbol{y}_j \\ \vdots \\ \boldsymbol{\phi}_\mathrm{n} \\ \boldsymbol{t}_\mathrm{n} \\ \boldsymbol{y}_\mathrm{n} \\ \boldsymbol{x}_1 \\ \vdots \\ \boldsymbol{x}_i \\ \vdots \\ \boldsymbol{x}_\mathrm{m} \end{pmatrix} . \quad (2.24)$$

J is up to m ∗ n, (9 ∗ n + 3 ∗ m)

u is [(9 ∗ n + 3 ∗ m)×1]

F_I_J is [2×1] (or [3X1] in Engels)

F is size up to [2*m*n×1]

All single reprojection error vectors (dimension of 2×1), $\{\boldsymbol{F}_{ij}\}_{(i,j)\in\mathcal{S}}$, also form a new reprojection error vector $\boldsymbol{F}$ with dimension of $2\mathrm{l}\times1$, since each single reprojection error vector contains the errors in x and y.

Efficient Optimization for Robust Bundle Adjustment

handed in
MASTER'S THESIS

Master of Science Zhongnan Qu

$$\frac{\partial \boldsymbol{F}_{ij}}{\partial \boldsymbol{\phi}_j} = \frac{d\boldsymbol{F}_{ij}}{d\boldsymbol{c}'_{ij}}\frac{\partial \boldsymbol{c}'_{ij}}{\partial \boldsymbol{c}_{ij}}\frac{d\boldsymbol{c}_{ij}}{d\boldsymbol{X}_{ij}}\frac{\partial \boldsymbol{X}_{ij}}{\partial \boldsymbol{\phi}_j} = \boxed{\frac{\partial \boldsymbol{c}'_{ij}}{\partial \boldsymbol{c}_{ij}}}\boxed{\frac{d\boldsymbol{c}_{ij}}{d\boldsymbol{X}_{ij}}}\boxed{\frac{\partial \boldsymbol{X}_{ij}}{\partial \boldsymbol{\phi}_j}}\,. \qquad \boxed{\text{2X3}} \tag{3.34}$$

$$\frac{\partial \boldsymbol{F}_{ij}}{\partial \boldsymbol{t}_j} = \frac{d\boldsymbol{F}_{ij}}{d\boldsymbol{c}'_{ij}}\frac{\partial \boldsymbol{c}'_{ij}}{\partial \boldsymbol{c}_{ij}}\frac{d\boldsymbol{c}_{ij}}{d\boldsymbol{X}_{ij}}\frac{\partial \boldsymbol{X}_{ij}}{\partial \boldsymbol{t}_j} = \boxed{\frac{\partial \boldsymbol{c}'_{ij}}{\partial \boldsymbol{c}_{ij}}}\boxed{\frac{d\boldsymbol{c}_{ij}}{d\boldsymbol{X}_{ij}}}\,. \qquad \boxed{\text{2X3}} \tag{3.35}$$

$$\frac{\partial \boldsymbol{F}_{ij}}{\partial \boldsymbol{y}_j} = \frac{\partial \boldsymbol{F}_{ij}}{\partial \boldsymbol{c}'_{ij}}\frac{\partial \boldsymbol{c}'_{ij}}{\partial \boldsymbol{y}_j} = \boxed{\frac{\partial \boldsymbol{c}'_{ij}}{\partial \boldsymbol{y}_j}}\,. \qquad \boxed{\text{2X3}} \tag{3.36}$$

$$\frac{\partial \boldsymbol{F}_{ij}}{\partial \boldsymbol{x}_i} = \frac{d\boldsymbol{F}_{ij}}{d\boldsymbol{c}'_{ij}}\frac{\partial \boldsymbol{c}'_{ij}}{\partial \boldsymbol{c}_{ij}}\frac{d\boldsymbol{c}_{ij}}{d\boldsymbol{X}_{ij}}\frac{\partial \boldsymbol{X}_{ij}}{\partial \boldsymbol{x}_i} = \boxed{\frac{\partial \boldsymbol{c}'_{ij}}{\partial \boldsymbol{c}_{ij}}}\boxed{\frac{d\boldsymbol{c}_{ij}}{d\boldsymbol{X}_{ij}}}\boldsymbol{R}_j\,. \qquad \boxed{\text{2X3}} \tag{3.37}$$

$\boxed{\text{2X3}}$

$$\frac{d\boldsymbol{c}_{ij}}{d\boldsymbol{X}_{ij}} = \begin{pmatrix} -\frac{1}{X_{ij,z}} & 0 & \frac{X_{ij,x}}{X_{ij,z}^2} \\ 0 & -\frac{1}{X_{ij,z}} & \frac{X_{ij,y}}{X_{ij,z}^2} \end{pmatrix} \tag{3.16}$$

$$\frac{\partial \boldsymbol{c}'_{ij}}{\partial \boldsymbol{c}_{ij}} = \begin{pmatrix} \frac{\partial c'_{ij,x}}{\partial c_{ij,x}} & \frac{\partial c'_{ij,x}}{\partial c_{ij,y}} \\ \frac{\partial c'_{ij,y}}{\partial c_{ij,x}} & \frac{\partial c'_{ij,y}}{\partial c_{ij,y}} \end{pmatrix} \tag{3.12}$$

$\boxed{\text{2X2}}$

$$\frac{\partial c'_{ij,x}}{\partial c_{ij,x}} = f_j(1 + k_{1,j}(3{c_{ij,x}}^2 + {c_{ij,y}}^2) + k_{2,j}(5{c_{ij,x}}^4 + {c_{ij,y}}^4 + 6{c_{ij,x}}^2{c_{ij,y}}^2)) \tag{3.13}$$

$$\frac{\partial c'_{ij,x}}{\partial c_{ij,y}} = \frac{\partial c'_{ij,y}}{\partial c_{ij,x}} = 2f_j c_{ij,x}c_{ij,y}(k_{1,j} + 2k_{2,j}({c_{ij,x}}^2 + {c_{ij,y}}^2)) \tag{3.14}$$

$$\frac{\partial c'_{ij,y}}{\partial c_{ij,y}} = f_j(1 + k_{1,j}(3{c_{ij,y}}^2 + {c_{ij,x}}^2) + k_{2,j}(5{c_{ij,y}}^4 + {c_{ij,x}}^4 + 6{c_{ij,x}}^2{c_{ij,y}}^2)) \tag{3.15}$$

$$\boldsymbol{X}_{ij} = \begin{pmatrix} X_{ij,x} \\ X_{ij,y} \\ X_{ij,z} \end{pmatrix} = \boldsymbol{R}_j\boldsymbol{x}_i + \boldsymbol{t}_j \tag{3.5}$$

uses + t_j

$$\boldsymbol{c}_{ij} = \begin{pmatrix} c_{ij,x} \\ c_{ij,y} \end{pmatrix} = \begin{pmatrix} -\frac{X_{ij,x}}{X_{ij,z}} \\ -\frac{X_{ij,y}}{X_{ij,z}} \end{pmatrix} \tag{3.6}$$

$\boxed{\text{2X3}}$

$$\frac{\partial \boldsymbol{c}'_{ij}}{\partial \boldsymbol{y}_j} = \begin{pmatrix} dis \times c_{ij,x} & f_j \times \|\boldsymbol{c}_{ij}\|^2 \times c_{ij,x} & f_j \times \|\boldsymbol{c}_{ij}\|^4 \times c_{ij,x} \\ dis \times c_{ij,y} & f_j \times \|\boldsymbol{c}_{ij}\|^2 \times c_{ij,y} & f_j \times \|\boldsymbol{c}_{ij}\|^4 \times c_{ij,y} \end{pmatrix} \tag{3.10}$$

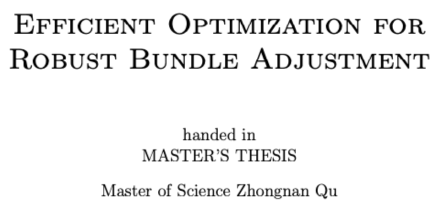
$$dis = 1 + k_{1,j} \times \|\boldsymbol{c}_{ij}\|^2 + k_{2,j} \times \|\boldsymbol{c}_{ij}\|^4\,. \tag{3.11}$$

$\boxed{\text{3X3}}$

$$\frac{\partial \boldsymbol{X}_{ij}}{\partial \boldsymbol{\phi}_j} = \begin{pmatrix} \frac{\partial X_{ij,x}}{\partial \phi_{j,x}} & \frac{\partial X_{ij,x}}{\partial \phi_{j,y}} & \frac{\partial X_{ij,x}}{\partial \phi_{j,z}} \\ \frac{\partial X_{ij,y}}{\partial \phi_{j,x}} & \frac{\partial X_{ij,y}}{\partial \phi_{j,y}} & \frac{\partial X_{ij,y}}{\partial \phi_{j,z}} \\ \frac{\partial X_{ij,z}}{\partial \phi_{j,x}} & \frac{\partial X_{ij,z}}{\partial \phi_{j,y}} & \frac{\partial X_{ij,z}}{\partial \phi_{j,z}} \end{pmatrix} \tag{3.28}$$

see **(3.28)** thru **(3.33)**

notes for partial derivatives of the intrinsic camera parameters

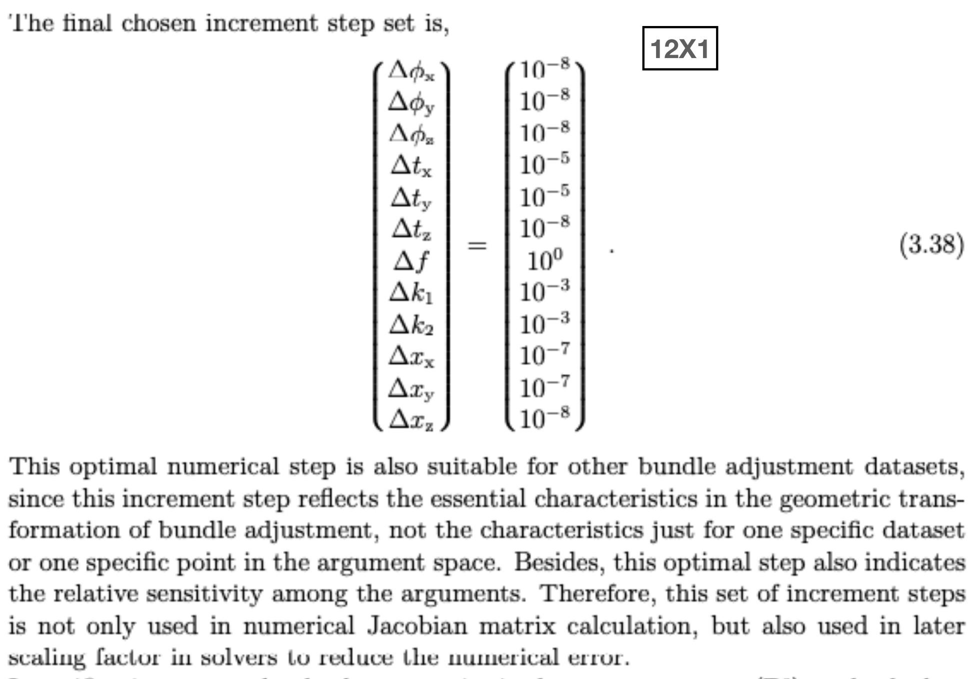


Efficient Optimization for
Robust Bundle Adjustment

handed in
MASTER'S THESIS

Master of Science Zhongnan Qu

m: number of feature points P_i
n: number of images



The final chosen increment step set is,

12X1

$$\begin{pmatrix} \Delta\phi_x \\ \Delta\phi_y \\ \Delta\phi_z \\ \Delta t_x \\ \Delta t_y \\ \Delta t_z \\ \Delta f \\ \Delta k_1 \\ \Delta k_2 \\ \Delta x_x \\ \Delta x_y \\ \Delta x_z \end{pmatrix} = \begin{pmatrix} 10^{-8} \\ 10^{-8} \\ 10^{-8} \\ 10^{-5} \\ 10^{-5} \\ 10^{-8} \\ 10^{0} \\ 10^{-3} \\ 10^{-3} \\ 10^{-7} \\ 10^{-7} \\ 10^{-8} \end{pmatrix} . \tag{3.38}$$

This optimal numerical step is also suitable for other bundle adjustment datasets, since this increment step reflects the essential characteristics in the geometric transformation of bundle adjustment, not the characteristics just for one specific dataset or one specific point in the argument space. Besides, this optimal step also indicates the relative sensitivity among the arguments. Therefore, this set of increment steps is not only used in numerical Jacobian matrix calculation, but also used in later scaling factor in solvers to reduce the numerical error.

A Brief Description of the Levenberg-Marquardt Algorithm Implemened by `levmar`

Manolis I. A. Lourakis
Institute of Computer Science
Foundation for Research and Technology - Hellas (FORTH)
Vassilika Vouton, P.O. Box 1385, GR 711 10
Heraklion, Crete, GREECE

February 11, 2005

and μ is referred to as the *damping term*. If the updated parameter vector $\mathbf{p} + \delta_\mathbf{p}$ with $\delta_\mathbf{p}$ computed from Eq. (3) leads to a reduction in the error ϵ, the update is accepted and the process repeats with a decreased damping term. Otherwise, the damping term is increased, the augmented normal equations are solved again and the process iterates until a value of $\delta_\mathbf{p}$ that decreases error is found. The process of repeatedly solving Eq. (3) for different values of the damping term until an acceptable update to the parameter vector is found corresponds to one iteration of the LM algorithm.

In LM, the damping term is adjusted at each iteration to assure a reduction in the error ϵ. If the damping is set to a large value, matrix N in Eq. (3) is nearly diagonal and the LM update step $\delta_\mathbf{p}$ is near the steepest descent direction. Moreover, the magnitude of $\delta_\mathbf{p}$ is reduced in this case. Damping also handles situations where the Jacobian is rank deficient and $\mathbf{J}^T\mathbf{J}$ is therefore singular [3]. In this way, LM can defensively navigate a region of the parameter space in which the model is highly nonlinear. If the damping is small, the LM step approximates the exact quadratic step appropriate for a fully linear problem. LM is adaptive because it controls its own damping: it raises the damping if a step fails to reduce ϵ; otherwise it reduces the damping. In this way LM is capable to alternate between a slow descent approach when being far from the minimum and a fast convergence when being at the minimum's neighborhood [3]. The LM algorithm terminates when at least one of the following conditions is met:

- The magnitude of the gradient of $\epsilon^T\epsilon$, i.e. $\mathbf{J}^T\epsilon$ in the right hand side of Eq. (2), drops below a threshold ε_1
- The relative change in the magnitude of $\delta_\mathbf{p}$ drops below a threshold ε_2
- The error $\epsilon^T\epsilon$ drops below a threshold ε_3
- A maximum number of iterations k_{max} is completed

there. Indicative values for the user-defined parameters are $\tau = 10^{-3}$, $\varepsilon_1 = \varepsilon_2 = \varepsilon_3 = 10^{-15}$, $k_{max} = 100$. `levmar` is a free C/C++ implementation of this LM algorithm that can be found at `http://www.ics.forth.gr/~lourakis/levmar`.