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 $\bar{\mathbf{X}}_i \leftrightarrow \mathbf{m}_i$

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 $\nabla f(\mathbf{x}) = J(\mathbf{x})^T\mathbf{r}(\mathbf{x})$

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

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 $\Delta\mathbf{x}^{LM} = (H^T H + \lambda I)^{-1} H^T\mathbf{r}$

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

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

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)

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

- When the parameter is represented by a rotation matrix :

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

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

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

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

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

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

$\wedge : \mathbb{R}^n \to so(n)$ 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

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)

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

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

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

initialize $\boldsymbol{p}$ and λ
for $k = 1$ **to** $maxIters$
  $\mathbf{f} = \text{eval\_objective}(\boldsymbol{p})$ // see Eq. 21
  $\mathbf{J}_g = \text{compute\_jacobian\_g}(\boldsymbol{p})$ // see Eqs. 26–34
  $\mathbf{J}_f = \text{compute\_jacobian\_f}(\boldsymbol{p})$ // see Eqs. 24–25
  $\mathbf{J} = \mathbf{J}_f \cdot \mathbf{J}_g$
  $\boldsymbol{p} = \boldsymbol{p} + \text{inv}\left(\mathbf{J}^T\mathbf{J} + \lambda\,\text{diag}\left(\mathbf{J}^T\mathbf{J}\right)\right) \cdot (\mathbf{b} - \mathbf{f})$
end for

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

(in camera centered coordinates) $\begin{pmatrix} x^c \\ y^c \\ w^c \end{pmatrix}$

(ithen 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 $g : \mathbb{R}^6 \rightarrow \mathbb{R}^9$ to the homography as

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

r11
r12
tx
r21
= r22
ty
-r31
-r32
-tz

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 $f : \mathbb{R}^9 \rightarrow \mathbb{R}^8$ maps them to the projected 2D point coordinates of our 4 reference points as

$$f(\mathbf{h}) = \begin{pmatrix} f_1(\mathbf{h}) \\ f_2(\mathbf{h}) \\ \vdots \\ f_7(\mathbf{h}) \\ f_8(\mathbf{h}) \end{pmatrix} = \begin{pmatrix} x_1^n \\ y_1^n \\ x_2^n \\ y_2^n \\ x_3^n \\ y_3^n \\ x_4^n \\ y_4^n \end{pmatrix} = \begin{pmatrix} \frac{h_1 x_1 + h_2 y_1 + h_3}{h_7 x_1 + h_8 y_1 + h_9} \\ \frac{h_4 x_1 + h_5 y_1 + h_6}{h_7 x_1 + h_8 y_1 + h_9} \\ \vdots \\ \frac{h_1 x_4 + h_2 y_4 + h_3}{h_7 x_4 + h_8 y_4 + h_9} \\ \frac{h_4 x_4 + h_5 y_4 + h_6}{h_7 x_4 + h_8 y_4 + h_9} \end{pmatrix} \quad (20)$$

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

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

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 $\mathbf{J}_g$

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

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

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

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.

Levenberg-Marquardt notes

from Szeliski 2010 Chap 6.

Transformation	Matrix	# DoF	Preserves	Icon
translation	$\begin{bmatrix} \boldsymbol{I} \mid \boldsymbol{t} \end{bmatrix}_{2\times3}$	2	orientation	
rigid (Euclidean)	$\begin{bmatrix} \boldsymbol{R} \mid \boldsymbol{t} \end{bmatrix}_{2\times3}$	3	lengths	
similarity	$\begin{bmatrix} s\boldsymbol{R} \mid \boldsymbol{t} \end{bmatrix}_{2\times3}$	4	angles	
affine	$\begin{bmatrix} \boldsymbol{A} \end{bmatrix}_{2\times3}$	6	parallelism	
projective	$\begin{bmatrix} \tilde{\boldsymbol{H}} \end{bmatrix}_{3\times3}$	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.

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

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

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

$$\boldsymbol{b} = \sum_i \boldsymbol{J}^T(\boldsymbol{x}_i)\boldsymbol{r}_i \qquad (6.17)$$

[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 $\boldsymbol{f}(\boldsymbol{x}_i;\boldsymbol{p}+\Delta\boldsymbol{p})$.

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

Levenberg-Marquardt notes

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

$$(\boldsymbol{A} + \lambda \text{diag}(\boldsymbol{A}))\Delta \boldsymbol{p} = \boldsymbol{b}, \tag{6.18}$$

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

Levenberg-Marquardt notes

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

$$(\boldsymbol{A} + \lambda \text{diag}(\boldsymbol{A}))\Delta \boldsymbol{p} = \boldsymbol{b}, \tag{6.18}$$

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.,

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

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

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

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.