Uma implementação do algoritmo Levenberg-Marquardt dividido para aplicações em visão computacional

An implementation of the partitioned Levenberg-Marquardt algorithm for applications in computer vision

José Alexandre de França[1]; Maria Bernadete de Morais França[2];
Marcela Hitomi Koyama[3]; Tiago Polizer da Silva[4]

Resumo

Em diversas aplicações da visão computacional, é necessário estimar-se, em um modelo, os parâmetros que melhor se ajustam a um conjunto de dados experimentais. Nesses casos, um algoritmo de minimização pode ser utilizado. Dentre estes, um dos mais conhecidos é o Levenberg-Marquardt. Apesar de diversas implementações de tal algoritmo estarem disponíveis livremente, nenhuma delas leva em consideração quando a solução do problema conduz a uma matriz jacobiana esparsa. Nesses casos, é possível reduzir significativamente a complexidade do algoritmo. Neste trabalho, apresenta-se uma implementação do algoritmo Levenberg-Marquardt para os casos em que a matriz jacobiana do problema é esparsa. Além disso, para ilustrar a aplicação do algoritmo, ele é aplicado a solução do problema de calibração monocular com gabaritos de uma única dimensão. Resultados empíricos mostram que o método converge satisfatoriamente em apenas algumas poucas iterações, mesmo na presença de ruído.
Palavras-chave: Algoritmo Levenberg-Marquardt. Calibração Monocular. Algoritmo de Newton.

Abstract

At several applications of computer vision is necessary to estimate parameters for a specific model which best fits an experimental data set. For these cases, a minimization algorithm might be used and one of the most popular is the Levenberg-Marquardt algorithm. Although several free applies from this algorithm are available, any of them has great features when the resolution of problem has a sparse Jacobian matrix . In this case, it is possible to have a great reduce in the algorithm's complexity. This work presents a Levenberg-Marquardt algorithm implemented in cases which has a sparse Jacobian matrix. To illustrate this algorithm application, the camera calibration with 1D pattern is applied to solve the problem. Empirical results show that this method is able to figure out satisfactorily with few iterations, even with noise presence.
Key words: Levenberg-Marquardt algorithm. Monocular Calibration. Newton-type algorithm.

[1] Departamento de Engenharia Elétrica, Universidade Estadual de Londrina; E-mail: josealexandre@eeol.org
[2] Departamento de Engenharia Elétrica, Universidade Estadual de Londrina; E-mail: bernadete@eeol.org
[3] Departamento de Engenharia Elétrica, Universidade Estadual de Londrina; E-mail: markoyama@yahoo.com.br
[4] Departamento de Engenharia Elétrica, Universidade Estadual de Londrina; E-mail: tiagopolizer@gmail.com.

Introdução

Em diversas aplicações da visão computacional, como o cálculo da matriz fundamental (ARMANGUÉ; SALVI, 2003), a calibração de câmeras (SALVI; ARMANGUÉ; BATLLE, 2002; ZHANG, 2000), a retificação de imagens (LOOP; ZHANG, 1999) e a estimação da transformação entre pontos (HORAUD; CSURKA; DEMIRDIJIAN, 2000), é necessário estimar os parâmetros de um determinado modelo que melhor se ajustam a um conjunto de dados experimentais. Normalmente, para uma melhor exatidão, um algoritmo de minimização não-linear deve ser utilizado. A maioria dos algoritmos utilizados são baseados no método de Newton (PRESS et al., 1992). Entre eles, salienta-se o método Levenberg-Marquardt (LEVENBERG, 1944; MARQUARDT, 1963), bastante utilizado na visão computacional sempre que deseja-se ajustar um modelo a um conjunto de dados experimentais. Como exemplo, pode-se citar os trabalhos de (KONEN; TOMBROCK; SCHOLZ, 2007), (SUN et al., 2006) e (LERASLE; RIVES; DHOME, 1999). Este método é uma derivação do método de Newton que é de convergência mais rápida.

Um dos motivos da popularidade do algoritmo Levenberg-Marquardt é a existência de diversas implementações disponíveis (LOURAKIS, 2004; MORÉ et al., 1984; PRESS et al., 1992; SHEARER; WOLFE, 1985). Contudo, apenas uma dessas (LOURAKIS; ARGYROS, 2004) toma vantagem de uma estrutura, muito comum em problemas da visão computacional, que faz com que a minimização não-linear produza uma matriz Jacobiana esparsa. Se isso for levado em consideração, para a estimação de N parâmetros, a complexidade do algoritmo reduz-se de N^3 para apenas N.

Neste trabalho, apresenta-se uma implementação do Levenberg-Marquardt, chamada Levenberg-Marquardt Dividido, que pode ser aplicada em problemas que apresentam a matriz Jacobiana esparsa. Nesse caso, o laço de iteração é dividido em vários problemas menores que possuem apenas uma fração da complexidade e são resolvidos muito mais rapidamente. Essa implementação foi codificada em linguagem do MATLAB® e pode ser executado sem modificações tanto no MATLAB® quanto no Scilab® e no Octave® .

Para exemplificar o uso do algoritmo Levenberg-Marquardt Dividido, aplicou-se a implementação proposta ao problema de calibração de câmeras com auxílio de um gabarito 1D (ZHANG, 2004). Nesse caso, o método é utilizado para, dado um conjunto de projeções do gabarito em diversas imagens, estimar-se os parâmetros intrínsecos da câmera e a localização da cada ponto do gabarito no espaço tridimensional. Resultados experimentais mostram que o método é capaz de realizar a estimação de forma satisfatória e em poucas iterações, mesmo na presença de ruído.

Notação

No decorrer do texto, matrizes e vetores são representados por letras, números ou símbolos em negrito. As constantes são expressas por letras, números ou símbolos em itálico. Além disso, adotou-se a prática notação $\mathbf{A}^{-T} = (\mathbf{A}^{-1})^T = (\mathbf{A}^{-T})^1$ para toda matriz quadrada e inversível.

Considerando o modelo de câmera *pinhole* (FAUGERAS; LUONG, 2001), as coordenadas de um ponto 3D no sistema de coordenadas do ambiente de uma câmera são apresentadas como $\mathbf{M} = [x,y,z]^T$ e a projeção correspondente no plano de imagem I, como $\mathbf{m} = [u,v]^T$. Além disso, as coordenadas homogêneas de um ponto $\mathbf{m} = [u,v,...]^T$ são representadas por $\tilde{\mathbf{m}}$, isto é, $\tilde{\mathbf{m}} = [u,v,..., 1]^T$. De forma mais geral, um ponto qualquer em coordenadas homogêneas é representado por $\tilde{\mathbf{m}} = [u,v,...,t]^T$. Um índice, se houver, indica a posição do ponto em um conjunto de pontos.

Com a notação adotada, a relação entre um ponto 3D, **M,** e sua projeção, **m**, em uma câmera *pinhole* é dada por

$$\widetilde{\mathbf{m}} \simeq \mathbf{A}\left[\mathbf{I}_3\ \mathbf{0}_3\right]\widetilde{\mathbf{M}}, \tag{1}$$

com

$$A = \begin{bmatrix} \alpha & 0 & u_0 \\ 0 & \beta & v_0 \\ 0 & 0 & 1 \end{bmatrix}, \tag{2}$$

onde "$\simeq$" indica que os dois lados da equação podem diferir por uma constante, α e β dão a relação entre pixels da imagem e distâncias no ambiente da câmera, respectivamente, nas direções horizontal e vertical. Já o ponto $\mathbf{m}_0 = [u_0, v_0]^T$ são as coordenadas do ponto central da câmera. Sendo assim, há apenas quatro parâmetros intrínsecos a serem estimados durante a calibração, ou seja, α, β, u_0 e v_0.

Algoritmo Levenberg-Marquardt

Considerando a seguinte função não-linear

$$\mathcal{F}(\mathbf{Y}) = \mathbf{X}, \tag{3}$$

onde $\mathbf{X} \in \mathbb{R}^M$ e $\mathbf{Y} \in \mathbb{R}^N$ são vetores e $M \geq N$. Freqüentemente, é necessário estimar o vetor $\widehat{\mathbf{Y}}$ que melhor se ajusta a um vetor $\widehat{\mathbf{X}}$ medido. Esse problema pode ser reformulado como, dado o vetor $\widehat{\mathbf{X}}$, encontrar $\widehat{\mathbf{Y}}$ que minimiza $\|\boldsymbol{\epsilon}\|$, sujeito a $\widehat{\mathbf{X}} = \mathcal{F}(\widehat{\mathbf{Y}}) + \boldsymbol{\epsilon}$.

Dada uma estimação inicial $\widehat{\mathbf{Y}}$, o método de Newton (PRESS et al., 1992) a refina assumindo que $\mathcal{F}(\widehat{\mathbf{Y}} + \boldsymbol{\Delta}) = \mathcal{F}(\widehat{\mathbf{Y}}) + \mathbf{J}\boldsymbol{\Delta}$, onde $\mathbf{J}$ é a matriz jacobiana, ou seja, $\mathbf{J} = \partial\mathbf{X}/\partial\mathbf{Y}$, e Δ é um que indica um "pequeno" incremento de $\widehat{\mathbf{Y}}$. Dessa forma, minimizar $\|\boldsymbol{\epsilon}\|$ equivale a minimizar

$$\|\boldsymbol{\epsilon} - \mathbf{J}\boldsymbol{\Delta}\|, \tag{4}$$

que é o mesmo que resolver

$$\mathbf{J}^T\mathbf{J}\boldsymbol{\Delta} = \mathbf{J}^T\boldsymbol{\epsilon}. \tag{5}$$

Além disso, a solução refinada, $\widehat{\mathbf{Y}}_r$, é dada por

$$\widehat{\mathbf{Y}}_r = \widehat{\mathbf{Y}} + \boldsymbol{\Delta} \tag{6}$$

e pode ser melhorada iterativamente.

Levenberg (1944) propôs uma alteração no algoritmo de Newton para acelerar sua convergência, ou seja, a equação (5) é substituída por

$$\left(\mathbf{J}^T\mathbf{J} + \mathbf{I}\lambda\right)\boldsymbol{\Delta} = \mathbf{J}^T\boldsymbol{\epsilon}. \tag{7}$$

Tipicamente, λ é feito igual a 10^{-4} inicialmente. Contudo, a cada iteração o valor de λ é alterado. Se a solução da equação (5) conduzir a uma redução do resíduo, λ é dividido por 10. Em caso contrário, este é multiplicado por 10.

O método de Levenberg possui instabilidades numéricas quando λ cresce. Em vista disso, algum tempo depois, Marquardt (MARQUARDT, 1963) propôs uma pequena alteração no algoritmo de Levenberg, ou seja, cada componente do gradiente é ponderada de acordo com sua curvatura. Assim, há uma grande tendência de convergência na direção na qual o gradiente é menor. Na prática, tal alteração é implementada substituindo a equação (7) por

$$\left(\mathbf{J}^T\mathbf{J} + \mathrm{diag}(\mathbf{J}^T\mathbf{J})\lambda\right)\boldsymbol{\Delta} = \mathbf{J}^T\boldsymbol{\epsilon}. \tag{8}$$

Assim, nos dias de hoje, o algoritmo resultante é conhecido como Levenberg-Marquardt.

A complexidade do algoritmo Levenberg-Marquardt aumenta com o número de variáveis.

Contudo, como se discute a seguir, algumas características de muitos dos problemas encontrados na visão computacional podem ser utilizadas para reduzir a complexidade do método significativamente. Isso é obtido como segue.

Considerando que $\mathbf{Y}$ na equação (3) pode ser dividido, ou seja,

$$\mathbf{Y} = [\mathbf{c}^T, \mathbf{d}^T]^T, \tag{9}$$

a matriz jacobiana, $\mathbf{J}$, da equação (4) pode ser dividida em dois blocos, ou seja, $\mathbf{J} = [\mathbf{C} \mid \mathbf{D}]$, onde $\mathbf{C} = [\partial\mathbf{X} / \partial\mathbf{c}]$ e $\mathbf{D} = [\partial\mathbf{X} / \partial\mathbf{d}]$. Logo, a equação (5) torna-se

$$\begin{bmatrix} \mathbf{C}^T\mathbf{C} & \mathbf{C}^T\mathbf{D} \\ \mathbf{D}^T\mathbf{C} & \mathbf{D}^T\mathbf{D} \end{bmatrix} \begin{bmatrix} \boldsymbol{\delta}_c \\ \boldsymbol{\delta}_d \end{bmatrix} = \begin{bmatrix} \mathbf{C}^T\boldsymbol{\epsilon} \\ \mathbf{D}^T\boldsymbol{\epsilon} \end{bmatrix}, \tag{10}$$

onde $\boldsymbol{\Delta} = [\boldsymbol{\delta}_c^T, \boldsymbol{\delta}_d^T]^T$.

Após alguma manipulação algébrica, a equação anterior pode ser reescrita como sendo

$$\begin{bmatrix} \mathbf{U} - \mathbf{G}\mathbf{V}^{-1} & \mathbf{0} \\ \mathbf{G}^T & \mathbf{V} \end{bmatrix} \begin{bmatrix} \boldsymbol{\delta}_c \\ \boldsymbol{\delta}_d \end{bmatrix} = \begin{bmatrix} \boldsymbol{\epsilon}_C - \mathbf{G}\mathbf{V}^{-1}\boldsymbol{\epsilon}_D \\ \boldsymbol{\epsilon}_D \end{bmatrix}, \tag{11}$$

onde $\mathbf{U} = \mathbf{C}^T\mathbf{C}$, $\mathbf{G} = \mathbf{C}^T\mathbf{D}$, $\mathbf{V} = \mathbf{D}^T\mathbf{D}$, $\boldsymbol{\epsilon}_C = \mathbf{C}^T\boldsymbol{\epsilon}$ e $\boldsymbol{\epsilon}_D = \mathbf{D}^T\boldsymbol{\epsilon}$.

A solução do problema anterior pode ser dividida em duas etapas. A primeira consiste em encontrar δ_C resolvendo

$$(\mathbf{U} - \mathbf{G}\mathbf{V}^{-1}\mathbf{G}^T)\boldsymbol{\delta}_c = \boldsymbol{\epsilon}_C - \mathbf{G}\mathbf{V}^{-1}\boldsymbol{\epsilon}_D \tag{12}$$

e, em seguida, δ_d pode ser encontrado resolvendo

$$\mathbf{V}\boldsymbol{\delta}_d = \boldsymbol{\epsilon}_D - \mathbf{G}^T\boldsymbol{\delta}_c. \tag{13}$$

Agora, considera-se ainda que os vetores $\mathbf{d}$, na equação (9), e $\mathbf{X}$ também possam ser divididos em n partes menores, ou seja, $\mathbf{d} = [\mathbf{d}_1^T, \mathbf{d}_2^T, \ldots, \mathbf{d}_n^T]^T$ e $\mathbf{X} = [\mathbf{X}_1^T, \mathbf{X}_1^T, \ldots, \mathbf{X}_n^T]^T$. Além disso, cabe ressaltar que, cada observação $\mathbf{X}_i$ depende apenas dos vetores $\mathbf{c}$, na equação (9), $\mathbf{d}_i$ e de nenhum outro $\mathbf{d}_k$. Neste caso, $\partial\mathbf{X}_i/\partial\mathbf{d}_k = 0$ para $i \neq k$, enquanto que nenhuma suposição pode ser feita a respeito de $\partial\mathbf{X}_i/\partial\mathbf{c}$. Isso faz com que a matriz jacobiana, $\mathbf{J} = \partial\mathbf{X}/\partial\mathbf{Y}$, tenha uma estrutura esparsa. Assim, a equação (4) torna-se

$$\left\| \begin{bmatrix} \boldsymbol{\epsilon}_1 \\ \boldsymbol{\epsilon}_2 \\ \vdots \\ \boldsymbol{\epsilon}_n \end{bmatrix} - \begin{bmatrix} \mathbf{C}_1 & \mathbf{D}_1 & & & \\ \mathbf{C}_2 & & \mathbf{D}_2 & & \\ \vdots & & & \ddots & \\ \mathbf{C}_n & & & & \mathbf{D}_n \end{bmatrix} \begin{bmatrix} \boldsymbol{\delta}_c \\ \boldsymbol{\delta}_{d_1} \\ \vdots \\ \boldsymbol{\delta}_{d_n} \end{bmatrix} \right\|, \tag{14}$$

onde $\mathbf{C_i} = [\partial\mathbf{X}_i/\partial\mathbf{c}]$ e $\mathbf{D}_i = [\partial\mathbf{X}_i/\partial\mathbf{d}_i]$ e $\boldsymbol{\epsilon} = \widehat{\mathbf{X}} - \mathcal{F}(\widehat{\mathbf{c}}, \widehat{\mathbf{d}}) = [\boldsymbol{\epsilon}_1^T, \boldsymbol{\epsilon}_2^T, \ldots, \boldsymbol{\epsilon}_n^T]^T$.

Uma importante observação a ser feita é que cada passo do algoritmo Levenberg-Marguardt Dividido tem complexidade linear em N (número de incógnitas), enquanto o não-dividido tem complexidade N^3.

Uma vez que o problema gera uma matriz Jacobiana com a estrutura da equação (14) este pode ser resolvido com o laço de iteração resumido no algoritmo 1. Tal algoritmo descreve em pormenores a solução do problema e torna possível a sua codificação em linguagem computacional. No apêndice A, apresenta-se a codificação utilizada neste trabalho.

Diversos problemas de otimização encontrados na visão computacional, por exemplo, os que envolvem as coordenadas de pontos no espaço projetivo 2D ou 3D (ARMANGUÉ; SALVI, 2003; HORAUD; CSURKA; DEMIRDIJIAN, 2000; LOOP; ZHANG, 1999; SALVI; ARMANGUÉ; BATLLE, 2002; ZHANG, 2000), podem ser formulados como um problema com a estrutura da equação (14). Na próxima seção, é dado um exemplo de aplicação que se encaixa perfeitamente nesta categoria de problemas.

Calibração com um bastão graduado

Na calibração monocular, uma câmera captura imagens de um objeto chamado gabarito de calibração. Em seguida, essas imagens são processadas e os parâmetros intrínsecos da câmera são estimados. Como exemplo de aplicação do algoritmo Levenberg-Marquardt Dividido, descreve-se o método de calibração que utiliza um gabarito de uma única dimensão, um bastão (ou mesmo um cordão) com diversas esferas (indexadas por *j*) ao longo de sua extensão. A distância entre esferas consecutivas deve ser conhecida com exatidão. Então, no processo de calibração, o gabarito deve deslocar-se no campo visual da câmera. Enquanto isso, uma seqüência de imagens (indexada por *i*) deve ser capturada. Zhang (2004) demonstrou que a calibração monocular com este tipo gabarito só é possível se duas condições forem satisfeitas: o gabarito conter três ou mais pontos e um desses pontos, $\mathbf{M}_1$ deve ser fixo, como mostra o esboço da figura (1). Dessa forma, a projeção deste ponto, $\mathbf{m}_1$, é a mesma em todas as imagens da seqüência.

No esquema da figura 1, o comprimento, *L*, do gabarito pode ser expresso em função das coordenadas dos pontos extremos $\mathbf{M}_1$ e $\mathbf{M}_{2i}$, ou seja,

$$\| \mathbf{M}_{2i} - \mathbf{M}_1 \| = L. \qquad (15)$$

Além disso, como as posições relativas das esferas são conhecidas, um ponto do gabarito localizado entre $\mathbf{M}_1$ e $\mathbf{M}_{2i}$ pode ser expresso por

$$\mathbf{M}_{ji} = \lambda_{1j}\, \mathbf{M}_1 + \lambda_{2j}\, \mathbf{M}_{2i}, \qquad (16)$$

onde λ_{1j} e λ_{2j} são conhecidos.

Entrada: Uma estimação inicial, particionada em um vetor $\widehat{\mathbf{Y}} = [\mathbf{c}^T, \mathbf{d}_1^T, \ldots, \mathbf{d}_n^T]^T$, de todos os parâmetros a serem refinados e um conjunto de observações em um vetor $\mathbf{X} = [\mathbf{X}_1^T, \ldots, \mathbf{X}_n^T]^T$.
Saída: O vetor $\mathbf{Y}_r$ que minimiza a equação (4).

1. Substituindo $\widehat{\mathbf{Y}}$ na equação (3), encontra-se $\widehat{\mathbf{X}} = [\widehat{\mathbf{X}}_1^T, \ldots, \widehat{\mathbf{X}}_n^T]^T$ e calcula-se todas as matrizes $\mathbf{C}_l = [\partial \widehat{\mathbf{X}}_l / \partial \mathbf{c}]$ e $\mathbf{D}_l = [\partial \widehat{\mathbf{X}}_l / \partial \mathbf{d}_l]$.
2. Calcula-se:

$$\mathbf{U} = \sum_{l=1}^{n} \mathbf{C}_l^T \mathbf{C}_l;$$
$$\mathbf{V} = \mathrm{diag}(\mathbf{V}_1, \ldots, \mathbf{V}_n), \text{ onde } \mathbf{V}_l = \mathbf{D}_l^T \mathbf{D}_l;$$
$$\mathbf{G} = [\mathbf{G}_1, \ldots, \mathbf{G}_n], \text{ onde } \mathbf{G}_l = \mathbf{C}_l^T \mathbf{D}_l;$$
$$\boldsymbol{\epsilon}_C = \sum_{l=1}^{n} \mathbf{C}_l^T \boldsymbol{\epsilon}_l, \text{ e;}$$
$$\boldsymbol{\epsilon}_D = [\boldsymbol{\epsilon}_{D_1}^T, \ldots, \boldsymbol{\epsilon}_{D_n}^T]^T, \text{ onde } \boldsymbol{\epsilon}_{D_l} = \mathbf{D}_l^T \boldsymbol{\epsilon}_l.$$

3. Calcula-se $\boldsymbol{\delta}_c$ resolvendo

$$\left(\mathbf{U} - \sum_{l=1}^{n} \mathbf{G}_l \mathbf{V}_l^{-1} \mathbf{G}_l^T \right) \boldsymbol{\delta}_c = \boldsymbol{\epsilon}_C - \sum_{l=1}^{n} \mathbf{G}_l \mathbf{V}_l^{-1} \boldsymbol{\epsilon}_D.$$

4. Calcula-se cada $\boldsymbol{\delta}_{d_l}$ da equação

$$\boldsymbol{\delta}_{d_l} = \mathbf{V}_l^{-1} (\boldsymbol{\epsilon}_{D_l} - \mathbf{G}_l^T \boldsymbol{\delta}_c).$$

5. Computam-se os parâmetros refinados, $\widehat{\mathbf{Y}}_r = \widehat{\mathbf{Y}} + [\boldsymbol{\delta}_c^T, \boldsymbol{\delta}_{d_1}^T, \ldots, \boldsymbol{\delta}_{d_n}^T]^T$.
6. Repetem-se todos os passos até a convergência do resíduo.

Algoritimo 1: Algoritmo Levenberg-Marguardt adaptado para problemas que levem a uma matriz Jacobiana esparsa.

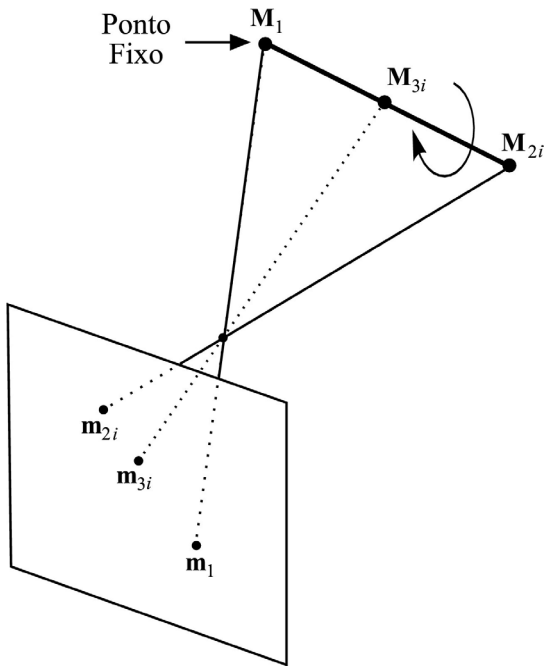

Figura 1. Esboço do gabarito 1D utilizado na calibração.

Da equação (1), considerando a profundidade (desconhecida) do ponto $\mathbf{M}_{ji}$ igual a z_{ji}, tem-se

$$\mathbf{M}_1 = z_1\mathbf{A}^{-1}\widetilde{\mathbf{m}}_1, \quad (17)$$

$$\mathbf{M}_{2i} = z_{2i}\mathbf{A}^{-1}\widetilde{\mathbf{m}}_{2i}, \quad (18)$$

$$\mathbf{M}_{ji} = z_{ji}\mathbf{A}^{-1}\widetilde{\mathbf{m}}_{ji}. \quad (19)$$

Dessa forma, substituindo as equações anteriores na equação (16), obtém-se

$$z_{ji}\widetilde{\mathbf{m}}_{ji} = z_1\lambda_{1j}\widetilde{\mathbf{m}}_{1i} + z_{2i}\lambda_{2j}\widetilde{\mathbf{m}}_{2i} \quad (20)$$

após eliminar-se $\mathbf{A}^{-1}$ de ambos os lados.

Agora, aplicando-se o produto cruzado a ambos os lados da equação anterior com $\widetilde{\mathbf{m}}_{ji}$, tem-se

$$z_1\lambda_{1j}(\widetilde{\mathbf{m}}_{1i} \times \widetilde{\mathbf{m}}_{ji}) + z_{2i}\lambda_{2j}(\widetilde{\mathbf{m}}_{2i} \times \widetilde{\mathbf{m}}_{ji}) = \mathbf{0} \quad (21)$$

que pode ser escrita como

$$z_{2i} = -z_1\frac{\lambda_{1j}(\widetilde{\mathbf{m}}_{1i} \times \widetilde{\mathbf{m}}_{ji}) \cdot (\widetilde{\mathbf{m}}_{2i} \times \widetilde{\mathbf{m}}_{ji})}{\lambda_{2j}(\widetilde{\mathbf{m}}_{2i} \times \widetilde{\mathbf{m}}_{ji}) \cdot (\widetilde{\mathbf{m}}_{2i} \times \widetilde{\mathbf{m}}_{ji})}. \quad (22)$$

Substituindo as equações (17) e (18) na equação (15) e considerando z_{2i} dado pela equação (22), obtém-se

$$z_1\|\mathbf{A}^{-1}\mathbf{h}_{ji}\| = L, \quad (23)$$

onde

$$\mathbf{h}_{ji} = \widetilde{\mathbf{m}}_1 + \frac{\lambda_{1j}(\widetilde{\mathbf{m}}_1 \times \widetilde{\mathbf{m}}_{ji}) \cdot (\widetilde{\mathbf{m}}_{2i} \times \widetilde{\mathbf{m}}_{ji})}{\lambda_{2j}(\widetilde{\mathbf{m}}_{2i} \times \widetilde{\mathbf{m}}_{ji}) \cdot (\widetilde{\mathbf{m}}_{2i} \times \widetilde{\mathbf{m}}_{ji})}\widetilde{\mathbf{m}}_{2i}. \quad (24)$$

A equação (23) é equivalente a

$$\mathbf{h}_{ji}^T\mathbf{B}\mathbf{h}_{ji} = L^2, \quad (25)$$

onde

$$\mathbf{B} = z_1^2\mathbf{A}^{-T}\mathbf{A}^{-1} = \begin{bmatrix} B_{11} & 0 & B_{13} \\ 0 & B_{22} & B_{23} \\ B_{13} & B_{23} & B_{33} \end{bmatrix}. \quad (26)$$

Levando em consideração que

$$\mathbf{h}_{ji}^T\mathbf{B}\mathbf{h}_{ji} = [a_{ji}^2, b_{ji}^2, 2a_{ji}c_{ji}, 2b_{ji}c_{ji}, c_{ji}^2]^T\mathbf{b} = \mathbf{u}_{ji}\mathbf{b}, \quad (27)$$

com $\mathbf{h}_{ji} = [a_{ji}, b_{ji}, c_{ji}]^T$ e $\mathbf{b} = [B_{11}, B_{22}, B_{13}, B_{23}, B_{33}]^T$, a equação (25) pode ser reescrita como

$$\mathbf{u}_{ji}^T\mathbf{b} = L^2. \quad (28)$$

Com n imagens, tem-se $\mathbf{U}_n = [\mathbf{u}_{j1}, \mathbf{u}_{j2}, \ldots \mathbf{u}_{jn}]^T$. Dessa forma, considerando $\mathbf{L}^2 = [L^2, \ldots, L^2]^T$, pode-se encontrar $\mathbf{b}$ resolvendo-se

$$\mathbf{U}_n \mathbf{b} = \mathbf{L}^2, \tag{29}$$

ou seja,

$$\mathbf{b} = (\mathbf{U}_n^T \mathbf{U}_n)^{-1} \mathbf{U}_n^T \mathbf{L}^2. \tag{30}$$

Dado um gabarito com mais de três pontos, tem-se um número maior de equações. Contudo, para cada imagem, apenas uma equação é linearmente independente. Assim, desde que existem 5 incógnitas (4 parâmetros de $\mathbf{A}$ e z_1), são necessários, no mínimo, 5 deslocamentos do gabarito para solucionar o problema.

Uma vez que $\mathbf{B}$ for conhecida, a matriz $z_1 \mathbf{A}^{-1}$ pode ser obtida de $\mathbf{B}$ através da decomposição de Cholesky (GOLUB; VAN LOAN, 1996). Por sua vez, com z_1 e $\mathbf{A}$ conhecidos, o ponto $\mathbf{M}_1$ pode ser obtido a partir da equação (17) e os pontos $\mathbf{M}_{2i}$ a partir das equações (22) e (18). Por último, pode-se usar a equação (16) para obter-se $\mathbf{M}_{ji}$.

Na presença de ruído, a solução do problema de calibração baseada na equação (30), geralmente, não é satisfatória. Um dos motivos para isso é que a equação (30) não tem nenhum significado físico, ou seja, trata-se de uma solução puramente algébrica. Uma solução com algum significado físico deve envolver diretamente as projeções observadas $\tilde{\mathbf{m}}_{ji}$, pois estas são as únicas informações disponíveis. Tal solução pode ser obtida como segue.

O ponto $\mathbf{M}_{2i}$ pode ser expresso em função do ponto fixo e dos ângulos θ_i e φ_i que definem a orientação do gabarito, ou seja,

$$\mathbf{M}_{2i} = \mathbf{M}_1 + L[\text{sen}\,\theta_i \cos\varphi_i, \text{sen}\,\theta_i \,\text{sen}\,\varphi_i, \cos\theta_i]^T. \tag{31}$$

Além disso, a equação (16) estende a equação (31) para os outros pontos $\mathbf{M}_{ji}$. Dessa forma, dados as constantes λ_{1j}, λ_{2j} e estimações de $\mathbf{A}$, $\mathbf{M}_1$, θ_i e φ_i, a partir das equações (1), (16) e (31), é possível obter-se uma estimação, $\hat{\mathbf{m}}_{ji}$, da projeção do ponto $\mathbf{M}_{ji}$. Com este resultado, assumindo-se que cada uma das projeções observadas, $\mathbf{m}_{ji}$, esteja corrompida por ruído aditivo, independente, mas com o mesmo desvio padrão, a "estimação ótima" da matriz $\mathbf{A}$, por meio do critério da máxima verosimilhança[1] é obtida minimizando-se

$$\sum_{i=1}^{n} \sum_{j=2}^{p} \left\| \mathbf{m}_{ji} - \hat{\mathbf{m}}_{ji}(\mathbf{A}, \mathbf{M}_1, \theta_i, \phi_i, \lambda_{1j}, \lambda_{2j}) \right\|^2, \tag{32}$$

onde $\lambda_{12} = 0$ e $\lambda_{22} = 1$, n é o número de imagens, p o número de pontos do gabarito.

Observando as equações (1), (16) e (31) fica claro que uma mudança nos elementos de $\mathbf{A}$ ou $\mathbf{M}_1$ altera os pontos $\mathbf{m}_{ji}$ de todas as imagens, enquanto uma alteração em θ_i ou φ_i reflete-se apenas nos pontos da i-ésima imagem. Dessa forma, conclui-se que o problema de minimizar a equação (32) enquadra-se perfeitamente no algoritmo Levenberg-Marguardt Dividido, com $\mathbf{c} = [\alpha, \beta, u_0, v_0, \mathbf{M}_1^T]^T$, $\mathbf{d}_i = [\theta_i, \varphi_i]^T$ e $\mathbf{X}_i = [\mathbf{m}_{2i}^T, \mathbf{m}_{3i}^T, ..., \mathbf{m}_{pi}^T]^T$.

Por exemplo, para um conjunto de 100 imagens, estimar-se os parâmetros intrínsecos de uma câmera com um gabarito 1D consiste em resolver um problema de minimização não-linear com 207 incógnitas (4 parâmetros intrínsecos, 3 coordenadas do ponto $\mathbf{M}_1$ e 100 pares $[\theta_i, \varphi_i]$). A utilização do algoritmo Levenberg-Marguardt Dividido em

[1] Do inglês: *maximum likelihood estimation.*

tal estimação, reduz drasticamente o tempo de computação devido a complexidade reduzida e rápida convergência.

Resultados empíricos

Para avaliar a função **mt_PartLevMarq**, foram utilizados dados sintéticos para minimizar a equação (32) (a estimação inicial do vetor **Y**, exigida por **mt_PartLevMarq**, foi obtida com auxílio do método linear descrito na seção anterior). Esses dados foram obtidos supondo-se uma câmera com os seguintes parâmetros: α =842, β =879, u_0 = 358 e v_0 = 207. Além disso, foi simulado um gabarito 1D de 30 cm de comprimento e cinco pontos colineares e eqüidistantes. O espaçamento entre dois pontos consecutivos foram sempre iguais.

Na geração dos dados sintéticos, os ângulos da equação (31), $\theta_i \in [-\pi / 2, \pi / 2]$ e $\varphi_i \in [-\pi / 2, \pi / 2]$, variaram aleatoriamente, mas de acordo com uma distribuição uniforme. Esses dados foram utilizados para avaliar o desempenho do algoritmo com respeito ao nível de ruído presente nos dados. Para isso, o ruído gaussiano de média zero e desvio padrão σ foi acrescentado aos pontos projetados nas imagens sintéticas. Esse ruído variou de 0,1 a 2 pixels. Para cada nível de ruído, foram realizadas 250 simulações e a mediana de cada parâmetro intrínseco foi armazenada. Essas medianas foram comparadas com os parâmetros da câmera simulada. O erro de cada um dos parâmetros estimados em função do nível de ruído é apresentado na figura 2, para a solução linear obtida da equação (30), e na figura 3, para a solução obtida com o algoritmo Levenberg-Marguardt implementado.

Analisando a figura 2, observa-se que o erro obtido com a solução linear aumenta quase que linearmente com o ruído, chegando a ordem de 15% para um erro com σ = 2. Além disso, alguns parâmetros são estimados com uma acurácia maior do que outros. Com o resultado da figura 3, por outro lado, observa-se que o algoritmo Levenberg-Marguardt Dividido conseguiu reduzir o erro drasticamente deixando-o, para erros com σ menores que 1 pixel da ordem de 0,1%. Além disso, o algoritmo consegue estimar todos os parâmetros com acurácia muito semelhante e, como mostra a figura 4, em apenas poucas iterações. De fato, o número de iterações varia muito pouco em função do nível do ruído. Isto é devido à rápida convergência do algoritmo Levenberg-Marguardt.

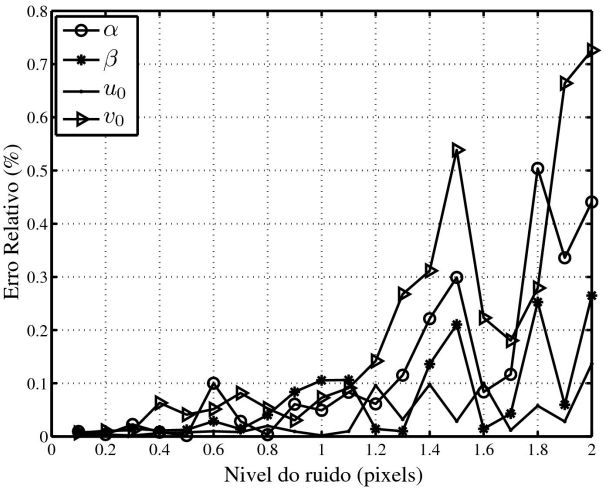

Figura 2. Erro *vs*. nível de ruído para a solução linear.

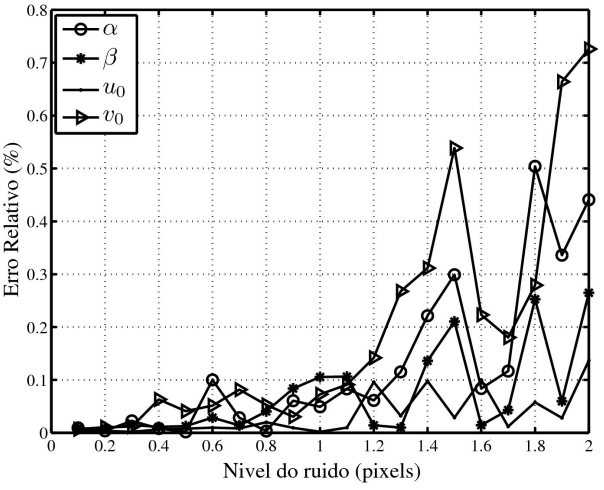

Figura 3. Erro *vs*. nível de ruído para a solução refinada.

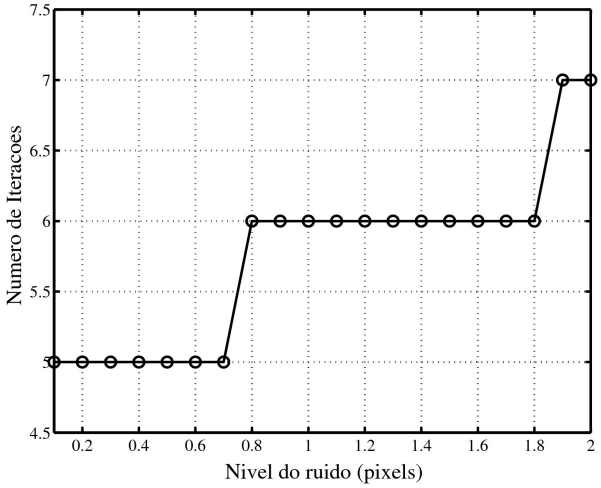

Figura 4. Número de iterações *vs.* nível de ruído.

Conclusão

Apresentou-se uma implementação do algoritmo Levenberg-Marguardt para solução de problemas que levam a uma matriz Jacobiana esparsa. Nesse caso, cada iteração do algoritmo pode ser dividida em problemas menores e de solução bem menos complexa. De fato, com relação ao número de parâmetros, a complexidade do método passa de N^3 para apenas N.

É sabido que algoritmo Levenberg-Marguardt Dividido pode ser utilizado na solução de diversos problemas da visão computacional, tais como cálculo da matriz fundamental, estimação da homografia entre pontos e calibração de câmeras. Em especial, pela primeira vez, aplicou-se esse algoritmo a tarefa de calibração monocular baseada em gabaritos de uma única dimensão. Neste caso, a calibração é realizada, dado um conjunto de projeções do gabarito em diversas imagens, estimando-se os parâmetros intrínsecos da câmera e a localização de cada ponto do gabarito no espaço tridimensional. Resultados experimentais mostram que o método é capaz de realizar a estimação de forma satisfatória e em poucas iterações, mesmo na presença de ruído. De fato, a acurácia dos parâmetros estimados é aumentada drasticamente (com relação a solução linear).

O código implementado está disponível, podendo ser executado sem modificações no MATLAB® ou Scilab® ou Octave®.

Agradecimentos

Os autores agradecem a CAPES e a PROPPG/UEL pelo financiamento das pesquisas.

Apêndice – Código fonte

O algoritmo 1 foi implementado em uma única função, chamada **mt_PartLevMarq**, com a linguagem do MATLAB®. Essa função necessita de uma estimação inicial do vetor **Y** a ser refinado; um vetor de observações **X**; a tolerância utilizada nos cálculos; uma função **Func** que, dado um vetor $\hat{\mathbf{Y}}$, calcule $\hat{\mathbf{X}}$ e; uma função **JacobFunc** que retorne dois vetores com todas as matrizes $\mathbf{C}_i$ e $\mathbf{D}_i$. (Também, é possível passar qualquer parâmetro extra que seja utilizado pelas funções **Func** e **JacobFunc**.) Já o retorno da função **mt_PartLevMarq**, este é composto pelo vetor $\hat{\mathbf{Y}}$ refinado, o número de iterações e informações sobre a convergência.

A seguir, apresenta-se o código fonte completo da função implementada.

```
function [P,exitflag,niter]=mt_PartLevMarq
X,P,Func,FuncJacob,tol,nmaxiter,varargin)
exitflag = 0;
len = length(X);

[A,B] = FuncJacob(P,X,varargin{:});
[La,Ca,N] = size(A);
[Lb,Cb,dummy] = size(B);
N = len/La;
Vd = 1:(Cb+1):(Cb*Cb);
Ud = 1:(Ca+1):(Ca*Ca);

X2 = Func(P,varargin{:});
Er = X - X2;

lambda = 1e-3;

% Loop principal
for niter=1:nmaxiter,
  [A,B] = FuncJacob(P,X2,varargin{:});

  U  = zeros(Ca,Ca);
  iV = zeros(Cb,Cb,N);
```

```
  W  = zeros(Cb,Ca,N);
  Ea = zeros(Ca,1);
  Eb = zeros(Cb,N);
  Y1 = zeros(Ca,Ca);
  Y2 = zeros(Ca,1);

  warning('off','all');
  for n=1:N,
    U = U + A(:,:,n)'*A(:,:,n);
    Vi = B(:,:,n)'*B(:,:,n);
    Vi(Vd) = (1+lambda)*Vi(Vd);
    iV(:,:,n)  = inv(Vi);
    W(:,:,n) = ( A(:,:,n)'*B(:,:,n) )';
    Ea = Ea + A(:,:,n)'*...
         Er(1+(n-1)*La:n*La,1);
    Eb(:,n) = B(:,:,n)'*...
              Er(1+(n-1)*Lb:n*Lb,1);

    tmp = W(:,:,n)'*iV(:,:,n);
    Y1 = Y1 + tmp*W(:,:,n);
    Y2 = Y2 + tmp*Eb(:,n);
  end

  U(Ud) = (1+lambda)*U(Ud);

  da = (U-Y1)\(Ea-Y2);
  warning('on','all');

  db = zeros(N*Cb,1);
  for n=1:N,
    db((1+(n-1)*Cb):n*Cb)
= iV(:,:,n)*...
(... (Eb(:,n)-W(:,:,n)*da);
  end

  Ptmp = P + [da; db];
  X2 = Func(Ptmp, vararg in {:});
  ErNew = X - X2;

  dEr = norm(Er) - norm(ErNew);
  if (dEr>0)
    P = Ptmp;
    Er = ErNew;
    lambda = lambda / 10;

    % Criterio de parada
        if (abs(dEr)<tol)
          exitflag = 1;
          break;
        end
  else
    lambda = 10*lambda;
  end
end
```

Referências

ARMANGUÉ, X.; SALVI, J. Overall view regarding fundamental matrix estimation. *Image and Vision Computing*, Guildford, v. 21, n. 2, p. 205–220, 2003

FAUGERAS, O.; LUONG, Q.-T. T*he geometry of multiple images: the laws that govern the formation of multiple images of a scene andsome of their applications*. [S.l.]: MIT Press, Cambridge, 2001.

GOLUB, G. H.; VAN LOAN, C. F. *Matrix Computations*. [S.l.]: The Johns University Press, 1996.

HORAUD, R.; CSURKA, G.; DEMIRDIJIAN, D. Stereo calibra- tion from rigid motions. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, New York, v. 22, n. 12, p. 1446-1452, 2000.

KONEN, W.; TOMBROCK, S.; SCHOLZ, M. Robust registration procedures for endoscopic imaging. *Medical Image Analysis*, London, v. 11, n. 6, p. 526–539, 2007.

LERASLE, F.; RIVES, G.; DHOME, M. Tracking of human limbs by multiocular vision. *Computer Vision and Image Understanding*, San Diego, v. 75, n. 3, p. 229–246, 1999.

LEVENBERG, K. A method for the solution of certain non-linear problems in least squares. *Quarterly of Applied Math*, Providence, v. 2, p. 164-168, 1944.

LOOP, C.; ZHANG, Z. Computing rectifying homographies for stereo vision. In: CONFERENCE ON COMPUTER VISION AND PATTERN RECOGNITION, 1., 1999, Fort Collins. *Proceedings*...Fort Collins, 1999. p. 125–131.

LOURAKIS, M.; ARGYROS, A. *The design and implementation of a generic sparse bundle adjustment software package based on the levenberg-marquardt algorithm*. 2004. Disponível em : <http://www.ics.forth.gr/_lourakis/sba/>. Acesso em: 3 de set. 2008.

LOURAKIS, M. I. A. L*evmar*: levenberg-marquardt nonlinear least squares algorithms in c/c++. 2004. Disponível em: <http://www.ics.forth.gr/_lourakis/levmar/>. Acesso em 3 de set. 2008.

MARQUARDT, D. W. An algorithm for least-squares estimation of nonlinear parameters. *Journal of the Society for Industrial and Applied Mathematics*, Philadelphia, v. 11, n. 2, p. 431–441, 1963.

MORÉ, J. J.; SORENSEN, D. C.; HILLSTROM, K. E.; GARBOW, B. S. *The MINPACK Project:* in Sources and Development of Mathematical Software. [S.l.]: Prentice-Hall, 1984.

PRESS,W. H.; TEUKOLSKY, S. A.; VETTERLING,W. T.; FLANNERY, B. P. *Numerical Recipes in C:* the Art of Scientific Computing. [S.l.]: Cambridge University Press, 1992.

SALVI, J.; ARMANGU´E, X.; BATLLE, J. A comparative review of camera calibrating methods with accuracy evaluation. *Pattern Recognition*, Ezmsford, v. 35, n. 7, p. 1617-1635, 2002.

SHEARER, J. M.; WOLFE, M. A. ALGLIB: a simple symbolmanipulation package. *Communications of the ACM*, New York, v. 28, n. 8, p. 820-825,1985.

SUN, J.; ZHANG, G.; WEI, Z.; ZHOU, F. Large 3d free surface measurement using a mobile coded light-based stereo vision system. *Sensors and Actuators A: Physical*, Lausanne, v. 132, n. 2, p. 460-471, 2006.

ZHANG, Z. A flexible new technique for camera calibration. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, New York, v. 22, n. 11, p. 1330-1334, 2000.

ZHANG, Z. Camera calibration with one-dimensional objects. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, New York, v. 26, n. 7, p.892-899, 2004.