A Tutorial on Parametric Image Registration

Leonardo Romero and Félix Calderón
División de Estudios de Postgrado, Facultad de Ingeniería Eléctrica
Universidad Michoacana de San Nicolás de Hidalgo
Morelia, Michoacán, México

1. Resume

This chapter introduces the reader to the area of parametric image registration, from a beginner's point of view. Given a model, an input image and a reference image, the parametric registration task is to find a set of parameters (of the model) that transform the input image into the reference image. This chapter reviews models of the general projective, affine, similarity and Euclidean transformations of images, and develop a full example for affine and projective transformation. It also describes two new methods of computing the set of image derivatives needed, besides the classical method reported in the literature. The new methods for computing derivatives are faster and more accurate than the classical method.

2. Introduction

Image registration is the process of overlaying two or more images of the same scene taken at different times, from different viewpoints or by different sensors [Zitova and Flusser, 2003]. In this chapter, only two images are considered: a reference image and an input image. The idea is to find a way to convert the input image into another image, similar to the reference image. If the model, that transforms the input image, has a small set of parameters, the task is called parametric image registration. Otherwise, the task it is called non-parametric registration [Calderon and Marroquin, 2003] (e.g. a set of parameters for each pixel of the image).
The literature is plenty of parametric registration techniques. Some of them are based on Spatiotemporal Energy [Adelson and Bergen, 1986], [Barman et al., 1986] and [Heeger, 1987], other methods are based on correlation [Kaneko et al., 2002] [Kaneko et al., 2003], others are based on the minimization of the Sum of Squared Differences (SSD) [Lai and Vemuri, 98], [Szeliski and Coughlan, 1994] (also named radial basis function in [Zitova and Flusser, 2003]), and others are based on optical Flow [Barron et al., 1994].
This tutorial describes in detail a SSD technique which can be extended easily to the M Estimators (for different M estimators see [Huber, Peter J. 2003]). The literature on parametric images registration often reports only advanced applications of this technique, but research papers do not address details of the implementation of these kinds of methods. Also surveys have been writing for experts (e.g. [Zitova and Flusser, 2003]) and this area is not fully covered in computer vision books. To our knowledge there is not a tutorial of

Source: Scene Reconstruction, Pose Estimation and Tracking, Book edited by: Rustam Stolkin,
ISBN 978-3-902613-06-6, pp.530, I-Tech, Vienna, Austria, June 2007

parametric image registration and this chapter tries to introduce beginners in computer vision into this area.
The rest of this chapter is organized as follows. Section 3 describes the registration problem as an optimization problem and Section 4 introduces the bilinear interpolation to compute accurate transformations of images. Section 5 introduces some basic transformations, from Euclidean to general projective transformations. Section 6 shows three methods to compute the set of derivatives of images needed, two new methods and the classical method reported in the literature. The first new method is a fast method based on interpolation of derivatives of the input image. The second method is the classical method based on derivatives of the transformed image. The third method is a new one and it is a more accurate and complete method than the classical method. Section 7, gives minimization details for an error function and subsection 7.1 presents the well known Levenberg-Marquard non-linear optimization method [Nocedal and Wright, 1999], commonly used in many computer vision problems. Experimental results are shown in section 8 using the three methods of computing derivatives. Results confirm the accuracy of the third method of computing derivatives. Finally, some conclusions are given in Section 9.

3. Parametric Registration Problem

Let *I(i,j)* denote a gray level image (typically an integer value from 0 to 255), for integer coordinates *<i,j>*, *I(i,j)* gives the intensity value of the pixel associated to position *<i,j>* (see Figure 1), and $I_r(i,j)$ denotes the reference image.
If the set of parameters is denoted by Θ, the parametric registration problem is to find a set Θ that minimizes an error function *E*, between the transformed input image $I_t(i,j)$ and the reference image. Considering the SSD, *E* can be expressed in the following way:

$$E(\Theta) = \sum\nolimits_{\forall <i,j> \in I_r} \left(I\left(x(\Theta,i,j), y(\Theta,i,j)\right) - I_r(i,j) \right)^2 \tag{1}$$

For instance, given a position $x=i+1$ and $y=j$, one pixel $I_r(i,j)$ is going to be compared with pixel $I(i+1,j)$. This situation is equivalent to have a transformed input image, $I_t(i,j)=I(i+1,j)$, where all pixels of the input image, have moved to the next position upwards (see Figure 1). The error *E* compares each pixel, between I_t and I_r, at the same position *<i,j>*. With the right Θ^*, image I_t and I_r should be very similar and *E* should reach a minimum value. The new image $I_t(i,j)$ can be computed by

$$I_t(i,j) = I\left(x(\Theta,i,j), y(\Theta,i,j)\right) \tag{2}$$

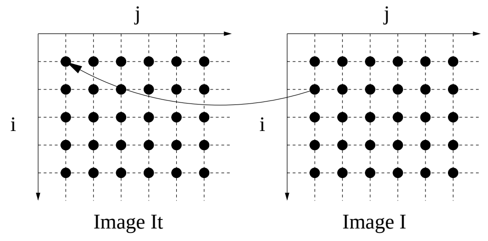


Figure 1. Computing transformed image I_t from I

If *x(Θ,i,j)* and *y(Θ,i,j)* are outside of the image *I*, a common strategy is to assign zero value which represents a black pixel. But, What happen when *x(Θ,i,j)* and *y(Θ,i,j)* have real values instead of integer values?. Remember that image *I(x,y)* have only valid values when *x* and *y* are integer values. An inaccurate method to solve this problem is to use their nearest integer values. Next section presents a much better method to solve this problem.

4. Bilinear Interpolation

If x_i and x_f are the integer and fractional part of x ($x = x_i+x_f$), and y_i and y_f the integer and fractional part of y ($y = y_i+y_f$), Figure 2 illustrates the bilinear interpolation method [Faugeras, 1993] to find $I(x_i+x_f, y_i+y_f)$, given the four nearest pixels to position $<x_i+x_f, y_i+y_f>$: $I(x_i, y_i)$, $I(x_i+1, y_i)$, $I(x_i, y_i+1)$ and $I(x_i+1, y_i+1)$ (image values at particular positions are represented by vertical bars in Figure 2). First two linear interpolations are used to compute two new values ($I_{new}(x_i, y_i+y_f)$ and $I_{new}(x_i+1, y_i+y_f)$) and then another linear interpolation is used to compute the desired value $I(x_i+x_f, y_i+y_f)$ from the new computed values:

$$\begin{aligned} I_{new}(x_i, y_i + y_f) &= (1 - y_f)I(x_i, y_i) + y_f I(x_i, y_i + 1) \\ I_{new}(x_i + 1, y_i + y_f) &= (1 - y_f)I(x_i + 1, y_i) + y_f I(x_i + 1, y_i + 1) \\ I(x_i + x_f, y_i + y_f) &= (1 - x_f)I_{new}(x_i, y_i + y_f) + x_f I_{new}(x_i + 1, y_i + y_f) \end{aligned} \tag{3}$$

Using the bilinear interpolation, a smooth transformed image is computed. Next section introduces a hierarchy of transformations that maps lines, in the input image, to lines in the transformed image [Hartley and Zisserman, 2000].

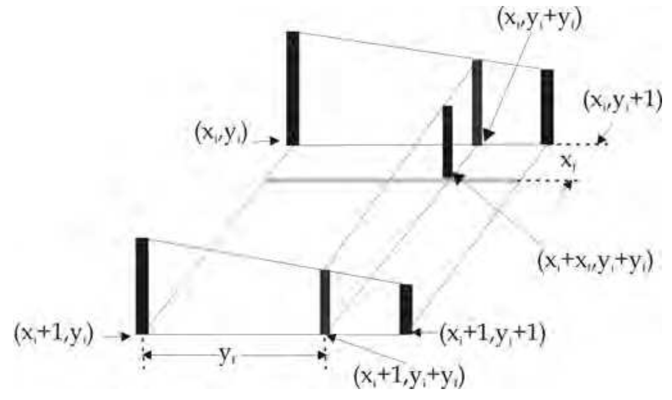


Figure 2. Using the Bilinear Interpolation

5. Basic Transformations

In this section Euclidean, Similarity, Affine and Projective transformations are reviewed briefly. In order to have a uniform frame of reference for these transformations, homogeneous coordinates are going to be used [Hartley and Zisserman, 2000]. A point $<x,y>$ in a plane is represented in homogeneous coordinates (HC) by a vector of 3 coordinates, $[x_h, y_h, w_h]^T$, and both coordinates are related by $x=x_h/w_h$ and $y=y_h/w_h$. In HC, a vector v and $\kappa\, v$ ($\kappa \in \Re$) represent the same point, an important advantage of using homogeneous coordinates is that original and transformed positions, as well as composition of transformations, are related by matrix multiplications [Hartley and Zisserman, 2000]. Figure 3, illustrates the Euclidian, Similarity, Affine and Projective transformation.

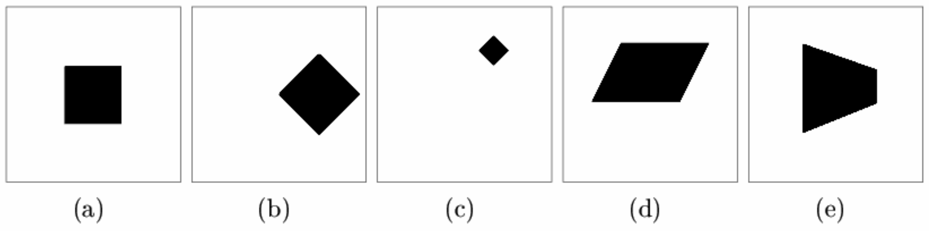


Figure 3. Linear Transformations in Homogeneous Coordinates: (a) Original, (b) Euclidean, (c) Similarity, (d) Affine and (e) Projective

5.1 Euclidean Transformations

In the case of Euclidian Transformation, angles and length of line segments are preserved, and only translations and rotations are allowed (see Figure 3(b)). This transformation have three parameters, $\Theta = \{\phi, t_i, t_j\}$, where ϕ is the rotation angle, and t_i, t_j are translation in directions *i* and *j* respectively. Using HC, $x(\Theta,i,j)=x_h/w_h$ and $y(\Theta,i,j)=y_h/w_h$, can be represented by:

$$\begin{aligned}[x_h, y_h, w_h]^T &= H_e[i,j,1]^T \\ H_e &= \begin{bmatrix} \cos\phi & -\sin\phi & t_i \\ \sin\phi & \cos\phi & t_j \\ 0 & 0 & 1 \end{bmatrix}\end{aligned} \tag{4}$$

5.2 Similarity Transformations

Besides translations and rotations, an isotropic scaling given by *s* is allowed (the same in both directions). Under this transformation, objects can be bigger or smaller, but their original shape is preserved (see Figure 3 (c)). The matrix representation, H_s, for this transformation is given by

$$\begin{aligned}[x_h, y_h, w_h]^T &= H_s[i,j,1]^T \\ H_s &= \begin{bmatrix} s\cos\phi & -s\sin\phi & st_i \\ s\ \sin\phi & s\cos\phi & st_j \\ 0 & 0 & 1 \end{bmatrix}\end{aligned} \tag{5}$$

5.3 Affine Transformations

An affine transformation is the most general transformation that preserves parallelism between lines (see Figure 3 (d)). This case is represented by H_a and has six parameters,

$$[x_h, y_h, w_h]^T = H_a[i, j, 1]^T$$
$$H_a = \begin{bmatrix} \theta_0 & \theta_1 & \theta_2 \\ \theta_3 & \theta_4 & \theta_5 \\ 0 & 0 & 1 \end{bmatrix} \tag{6}$$

Where θ_2 and θ_5 represent the translation in both directions. This transformation allows rotation, scaling, shearing, translation or combinations of these transformations.

5.4 Projective Transformations

This is the most general transformation that maps lines into lines, and it generalizes an affine transformation. The matrix H_p for this transformation has nine elements (actually only eight independent ratios among the nine elements of H_p, because in HC proportional vectors represent the same vector). An example of this transformation where parallelism is not preserved is shown in Figure 3(e). In most interesting cases, projective transformations H_p, has the form:

$$[x_h, y_h, w_h]^T = H_p[i, j, 1]^T$$
$$H_p = \begin{bmatrix} \theta_0 & \theta_1 & \theta_2 \\ \theta_3 & \theta_4 & \theta_5 \\ \theta_6 & \theta_7 & 1 \end{bmatrix} \tag{7}$$

Next section develops examples of finding the set of parameter for affine and projective transformations.

6. Finding the Set of Parameters

In order to compute a set of parameter Θ, equation (1), can be rewritten as follows

$$E(I(\Theta), I_r) = \sum\nolimits_{\forall < i,j > \in I_r} \rho(e_{ij})$$
$$e_{ij} = I(x(\Theta, i, j), y(\Theta, i, j)) - I_r(i, j) \tag{8}$$

ρ(e) is an error function which could be a quadratic function or any M-estimator (see [Huber, Peter J. 2003]). In case of a quadratic function the solution is known as Least Squares. Given a set $\Theta = \{\theta_0, \ldots, \theta_k, \ldots, \theta_K\}$ of K parameters, E will have a minimum value when $\partial E / \partial \theta_k = 0$ for all k. The *k-th* element of the gradient value $G_k = \partial E / \partial \theta_k$, can be computed as

$$G_k(\Theta) = \frac{\partial E}{\partial \theta_k} = \sum_{\forall <i,j> \in I_r} \frac{\partial \rho(e_{ij})}{\partial e_{ij}} \frac{\partial e_{ij}}{\partial \theta_k} \quad k=0,1,\ldots K \tag{9}$$

In the literature, the derivative $\partial \rho(e_{ij}) / \partial e_{ij}$ is called the influence function [Huber, Peter J. 2003] and it is represented by $\varphi(e_{ij})$. In the case of a quadratic error function $\varphi(e_{ij}) = 2e_{ij}$. If we introduced the gradient vector $G(\Theta)=[G_0(\Theta), G_1(\Theta), \ldots G_K(\Theta)]^T$ and the vector $J_{ij}(\Theta)$ as

$$J_{ij}(\Theta) = \left[\frac{\partial e_{ij}}{\partial \theta_0}, \frac{\partial e_{ij}}{\partial \theta_1}, \cdots, \frac{\partial e_{ij}}{\partial \theta_K} \right]^T \tag{10}$$

$G(\Theta)$ can be written in a single one :

$$G(\Theta) = \sum_{\forall <i,j> \in I_r} \varphi_{ij}(\Theta) J_{ij}(\Theta) \tag{11}$$

Let's develop an expression for each term of the vector $J_{ij}(\Theta)=[J_{ij0}(\Theta), J_{ij1}(\Theta), \ldots J_{ijK}(\Theta)]^T$ from equation (8) so an expression for $J_{ijk}(\Theta)$ can be derived as*:*

$$J_{ijk}(\Theta) = \frac{\partial e_{ij}}{\partial \theta_k} = \frac{\partial I(x(\Theta,i,j), y(\Theta,i,j))}{\partial \theta_k} \tag{12}$$

Using the chain rule from the differential calculus, the desired value can be computed as follows,

$$J_{ijk}(\Theta) = \frac{\partial I(x(\Theta,i,j), y(\Theta,i,j))}{\partial x(\Theta,i,j)} \frac{\partial x(\Theta,i,j)}{\partial \theta_k} + \frac{\partial I(x(\Theta,i,j), y(\Theta,i,j))}{\partial y(\Theta,i,j)} \frac{\partial y(\Theta,i,j)}{\partial \theta_k} \tag{13}$$

Using matrix notation $J_{ij}(\Theta) = M(\Theta,i,j)\nabla I(x,y)$ with

$$M(\Theta,i,j) = \begin{bmatrix} \frac{\partial x(\Theta,i,j)}{\partial \theta_0} & \frac{\partial y(\Theta,i,j)}{\partial \theta_0} \\ \vdots & \vdots \\ \frac{\partial x(\Theta,i,j)}{\partial \theta_K} & \frac{\partial y(\Theta,i,j)}{\partial \theta_K} \end{bmatrix} \tag{14}$$

$$\nabla I(x,y) = \begin{bmatrix} \dfrac{\partial I(x(\Theta,i,j), y(\Theta,i,j))}{\partial x(\Theta,i,j)} \\ \dfrac{\partial I(x(\Theta,i,j), y(\Theta,i,j))}{\partial y(\Theta,i,j)} \end{bmatrix} \tag{15}$$

M will called the Coordinate Matrix Model (CMM) which depends of the characteristics of the model and $\nabla I(x,y)$ the gradient vector respect to *x* and *y* (remember that the image only have integer coordinates values). Subsection 6.1 presents the CMM for affine and projective transformations and subsection 6.2 presents three methods of computing the gradient vector $\nabla I(x,y)$.

6.1 Coordinate Matrix Model

An affine transformation given by equation (6) can be rewritten as follows.

$$\begin{aligned} x(\Theta,i,j) &= \theta_0 i + \theta_1 j + \theta_2 \\ y(\Theta,i,j) &= \theta_3 i + \theta_4 j + \theta_5 \end{aligned} \tag{16}$$

By definition of CMM given by equation (14), the CMM, for affine transformation, has a simple form given by equation (17).

$$M(\Theta,i,j) = \begin{bmatrix} \dfrac{\partial x}{\partial \theta_0} & \dfrac{\partial y}{\partial \theta_0} \\ \dfrac{\partial x}{\partial \theta_1} & \dfrac{\partial y}{\partial \theta_1} \\ \dfrac{\partial x}{\partial \theta_2} & \dfrac{\partial y}{\partial \theta_2} \\ \dfrac{\partial x}{\partial \theta_3} & \dfrac{\partial y}{\partial \theta_3} \\ \dfrac{\partial x}{\partial \theta_4} & \dfrac{\partial y}{\partial \theta_4} \\ \dfrac{\partial x}{\partial \theta_5} & \dfrac{\partial y}{\partial \theta_5} \end{bmatrix} = \begin{bmatrix} i & 0 \\ j & 0 \\ 1 & 0 \\ 0 & i \\ 0 & j \\ 0 & 1 \end{bmatrix} \tag{17}$$

In a similar way, the projective transformation (eq. (7)), can be rewritten as

$$
\begin{aligned}
x(\Theta,i,j) &= \frac{\theta_0 i + \theta_1 j + \theta_2}{w(\Theta,i,j)} \\
y(\Theta,i,j) &= \frac{\theta_3 i + \theta_4 j + \theta_5}{w(\Theta,i,j)} \\
w(\Theta,i,j) &= \theta_6 i + \theta_7 j + 1
\end{aligned}
\tag{18}
$$

The CMM, in this case, is given by equation (19)

$$
M(\Theta,i,j) = \frac{1}{w}\begin{bmatrix} i & 0 \\ j & 0 \\ 1 & 0 \\ 0 & i \\ 0 & j \\ 0 & 1 \\ -ix & -iy \\ -jx & -jy \end{bmatrix}
\tag{19}
$$

6.2 Computing Derivatives of Images

Here we present three methods to compute the terms not previously described in eq. (14). In all the three methods, derivative of images are needed which can be approximated by the following central-difference approximations [Trucco and Verri, 1998]:

$$
\begin{aligned}
\frac{\partial I(i,j)}{\partial i} &= \frac{I(i+1,j) - I(i-1,j)}{2} \\
\frac{\partial I(i,j)}{\partial j} &= \frac{I(i,j+1) - I(i,j-1)}{2}
\end{aligned}
\tag{20}
$$

More accurate approximations consider more pixels in the neighborhood [Trucco and Verri, 1998]:

$$
\begin{aligned}
\frac{\partial I(i,j)}{\partial i} &= \frac{-I(i+2,j) + 8I(i+1,j) - 8I(i-1,j) + I(i-2,j)}{12} \\
\frac{\partial I(i,j)}{\partial j} &= \frac{-I(i,j+2) + 8I(i,j+1) - 8I(i,j-1) + I(i,j-2)}{12}
\end{aligned}
\tag{21}
$$

An even better method is called derivative of Gaussian Filters [Ma et al., 2004] [Romeny, 1994], and it computes much smaller noise responses, compared with the previous ones. The Gaussian function and its derivative are given, respectively, by

$$g(t)=\frac{1}{\sqrt{2\pi\sigma}}e^{\frac{-t^2}{2\sigma^2}}$$
$$g'(t)=\frac{dg(t)}{dt}=\frac{-t}{\sigma^3\sqrt{2\pi}}e^{\frac{-t^2}{2\sigma^2}} \tag{22}$$

The computation of image derivatives is accomplished as a pair of 1-D convolutions with filters obtained by sampling the continuous Gaussian function and its derivative,

$$\frac{\partial I(i,j)}{\partial i}=I(i,j)*g'(i)*g(j)=\sum_{k=-w/2}^{w/2}\sum_{l=-w/2}^{w/2}\left[I(i,j)g'(i-k)g(j-l)\right]$$
$$\frac{\partial I(i,j)}{\partial j}=I(i,j)*g(i)*g'(j)=\sum_{k=-w/2}^{w/2}\sum_{l=-w/2}^{w/2}\left[I(i,j)g(i-k)g'(j-l)\right] \tag{23}$$

σ (in pixels units) controls the Gaussian form, and usually *w* = *3 σ*. If the window defined by *w* is bigger, then more pixels in the neighborhood are considered. In equation (23), the operator (*) represents convolution.
Now three methods, to compute the gradient vector of the input image, ($\nabla I(x,y)$ in eq. (15)) are presented in next section. Remember that *x* and *y* in general can be real numbers.

6.2.1 Method 1: Using derivatives of the input image

Considering that $<i,j>\in N^2$ and $<x,y>\in R^2$, if $\partial I(i,j)/\partial i$ and $\partial I(i,j)/\partial j$ are computed using one of the previous methods, a simple and fast method to compute $\partial I(x,y)/\partial x$ is to use bilinear interpolation from $\partial I(i,j)/\partial i$ to get an approximate value. In a similar way, $\partial I(x,y)/\partial y$ can be estimated from $\partial I(i,j)/\partial j$.

6.2.2 Method 2: Using approximate derivatives of the transformed image

Considering the transformed image I_t and from equation (2), another approximation is given by

$$\frac{\partial I(x(\Theta,i,j),y(\Theta,i,j))}{\partial x(\Theta,i,j)}=\frac{\partial I_t(i,j)}{\partial i}$$
$$\frac{\partial I(x(\Theta,i,j),y(\Theta,i,j))}{\partial y(\Theta,i,j)}=\frac{\partial I_t(i,j)}{\partial j} \tag{24}$$

Because this approximation is reported in many papers, we named it the classical method.

6.2.3 Method 3: Using derivatives of the transformed image

Method 2 considers $I_t(i,j)$ = $I(x(\Theta,i,j),y(\Theta,i,j))$, and also considers derivatives given by equation (24). The first one is correct, but not the second one, because increments in *x* does not necessarily correspond to the same increments of *i* (and the same argument with *y* and *j*). The right derivatives can be computed using the chain rule, as follows:

$$\begin{aligned}\frac{\partial I(x,y)}{\partial x} &= \frac{\partial I_t(i,j)}{\partial i}\frac{\partial i}{\partial x} + \frac{\partial I_t(i,j)}{\partial j}\frac{\partial j}{\partial x}\\ \frac{\partial I(x,y)}{\partial y} &= \frac{\partial I_t(i,j)}{\partial i}\frac{\partial i}{\partial y} + \frac{\partial I_t(i,j)}{\partial j}\frac{\partial j}{\partial y}\end{aligned} \tag{25}$$

Using matrix notation equation (25) can be rewritten as

$$\begin{gathered}\begin{bmatrix}\frac{\partial I(x,y)}{\partial x}\\ \frac{\partial I(x,y)}{\partial y}\end{bmatrix} = \begin{bmatrix}\frac{\partial i}{\partial x} & \frac{\partial j}{\partial x}\\ \frac{\partial i}{\partial y} & \frac{\partial j}{\partial y}\end{bmatrix}\begin{bmatrix}\frac{\partial I_t(i,j)}{\partial i}\\ \frac{\partial I_t(i,j)}{\partial j}\end{bmatrix}\\ \nabla I(x,y) = N(\Theta,x,y)\nabla I_i(i,j)\end{gathered} \tag{26}$$

Lets define *N(Θ,x,y)* as the Derivative Correction Matrix (DCM) and $\nabla I_t(i,j)$ as the gradient vector image of Image I_t and it can be computed using, for instance, the derivative of Gaussian Filter previously mentioned.

Now a closed expression for the matrix *N*, in case of a projective transformation, is developed. To compute the elements of DCM matrix, explicit formulas for *i* and *j* are needed, for this reason equation (18) is rewritten as follows:

$$\begin{aligned}(x\theta_6-\theta_0)i+(x\theta_7-\theta_1)j &= (\theta_2-x)\\ (y\theta_6-\theta_3)i+(y\theta_7-\theta_4)j &= (\theta_5-y)\end{aligned} \tag{27}$$

Then the system of equation given by (27) is solved using the well known Cramer's rule from linear algebra and its solution for *i* and *j* are given as:

$$\begin{aligned}i &= \frac{(\theta_4-\theta_5\theta_7)x+(\theta_2\theta_7-\theta_1)y+(\theta_1\theta_5-\theta_2\theta_4)}{(\theta_3\theta_7-\theta_4\theta_6)x+(\theta_1\theta_6-\theta_0\theta_7)y+(\theta_0\theta_4-\theta_1\theta_3)}\\ j &= \frac{(\theta_5\theta_6-\theta_3)x+(\theta_0-\theta_2\theta_6)y+(\theta_2\theta_3-\theta_0\theta_5)}{(\theta_3\theta_7-\theta_4\theta_6)x+(\theta_1\theta_6-\theta_0\theta_7)y+(\theta_0\theta_4-\theta_1\theta_3)}\end{aligned} \tag{28}$$

From equation (28) the elements of matrix *N* are derived by definition given in equation (26) and after a little algebra the DCM matrix is given as:

$$N(\Theta,x,y) = F\begin{bmatrix}-(\theta_7y-\theta_4) & (\theta_6y-\theta_3)\\ (\theta_7x-\theta_1) & -(\theta_6x-\theta_0)\end{bmatrix} \tag{29}$$

With

$$F = \frac{\theta_0(\theta_4 - \theta_5\theta_7) - \theta_1(\theta_3 - \theta_5\theta_6) + \theta_2(\theta_3\theta_7 - \theta_4\theta_6)}{[(\theta_3\theta_7 - \theta_4\theta_6)x + (\theta_1\theta_6 - \theta_0\theta_7)y + (\theta_0\theta_4 - \theta_1\theta_3)]^2} \tag{30}$$

When the projective transformation is only an affine transformation the vector parameter can be written as $\Theta = [\theta_0, \theta_1, \theta_2, \theta_3, \theta_4, \theta_5, 0, 0]$ and the DCM, in this case, is

$$N(\Theta, x, y) = \frac{1}{\theta_0\theta_4 - \theta_1\theta_3}\begin{bmatrix} \theta_4 & -\theta_3 \\ -\theta_1 & \theta_0 \end{bmatrix} \tag{31}$$

In the translation transformation case, the parameter vector can be written as $\Theta = [1, 0, \theta_2, 0, 1, \theta_5, 0, 0]$ and its DCM is

$$N(\Theta, x, y) = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \tag{32}$$

Note, that only for translation model, this method and method 2 are the same.

7. Minimization procedure

Let $G_k(\Theta^n) = \partial E / \partial\theta_k$, ($k=0, \ldots, K$), Θ^n be an initial set of parameter values, and Θ^{n+1} an improved set of parameter values, where $\theta_k^{n+1} = \theta_k^n + \delta\theta_k^n$. The condition to reach an optimum Θ^{n+1} is to find a set of increments $\delta\theta_k^n, (k = 0, \ldots K)$, such that the $G_k(\Theta^{n+1}) = 0$ (for all k). To compute the set of increments, functions G_k can be approximate using the Taylor expansion using only first order derivatives:

$$G_k(\Theta^{n+1}) = G_k(\Theta^n) + \frac{\partial G_k(\Theta^n)}{\partial\theta_0}\delta\theta_0^n + \frac{\partial G_k(\Theta^n)}{\partial\theta_1}\delta\theta_1^n + \ldots + \frac{\partial G_k(\Theta^n)}{\partial\theta_K}\delta\theta_K^n$$

$$k=0, 1, ..K \tag{33}$$

Next we can compute the set of increments, doing $G_k(\Theta^{n+1}) = 0$ and solving the system of K equations of the form:

$$\frac{\partial G_k(\Theta^n)}{\partial\theta_0}\delta\theta_0^n + \frac{\partial G_k(\Theta^n)}{\partial\theta_1}\delta\theta_1^n + \ldots + \frac{\partial G_k(\Theta^n)}{\partial\theta_K}\delta\theta_K^n = -G_k(\Theta^n)$$

$$k=0, 1, ..K \tag{34}$$

In matrix form we have,

$$H\Delta\Theta = G \tag{35}$$

Where

$$H\left(\Theta^n\right)=\begin{bmatrix} \frac{\partial G_0\left(\Theta^n\right)}{\partial\theta_0} & \frac{\partial G_0\left(\Theta^n\right)}{\partial\theta_1} & \cdots & \frac{\partial G_0\left(\Theta^n\right)}{\partial\theta_K} \\ \frac{\partial G_1\left(\Theta^n\right)}{\partial\theta_0} & \frac{\partial G_1\left(\Theta^n\right)}{\partial\theta_1} & \cdots & \frac{\partial G_1\left(\Theta^n\right)}{\partial\theta_K} \\ \cdots & \cdots & \cdots & \cdots \\ \frac{\partial G_K\left(\Theta^n\right)}{\partial\theta_0} & \frac{\partial G_K\left(\Theta^n\right)}{\partial\theta_1} & \cdots & \frac{\partial G_K\left(\Theta^n\right)}{\partial\theta_K} \end{bmatrix}$$

$$\Delta\Theta=\left[\delta\theta_0^n,\delta\theta_1^n,\cdots,\delta\theta_K^n\right]^T$$

$$G=\left[-G_0\left(\Theta^n\right),-G_1\left(\Theta^n\right),\cdots,-G_K\left(\Theta^n\right)\right]^T \tag{36}$$

Once this system of equation is solved, a new set of parameter Θ^{n+1} can be computed, and using the same procedure another set of parameter Θ^{n+2} is estimated, and so on. The iterative process ends when all the increments are very small.

In the literature matrix H is known like the Hessian matrix. If full derivatives for the elements $H_{rc}\left(\Theta^n\right)=\partial G_r\left(\Theta^n\right)/\partial\theta_c$ of the Hessian matrix (where r and c represents the row and column of H) are computed from equation (9), then results the Newton's method [Nocedal and Wright, 1999]. But, if we discard second order derivatives, the method is called Gauss-Newton (GN) [Nocedal and Wright, 1999], and it is commonly used due to its simple form and because it warranties to have a semi positive defined matrix H. The Matrix H for the GN method is presented in equation (37)

$$H_{rc}(\Theta^n)=\frac{\partial^2 E(\Theta^n)}{\partial\theta_r\partial\theta_c}=\sum_{\forall<i,j>\in I_r}\psi\left(e_{ij}\right)\frac{\partial e_{ij}}{\partial\theta_r}\frac{\partial e_{ij}}{\partial\theta_c} \tag{37}$$

Where $\psi(e_{ij})=\partial^2\rho(e_{ij})/\partial^2 e_{ij}$ and for a quadratic error function $\psi(e_{ij})=2$. In matrix form, the equation (37) can be rewritten as

$$H(\Theta^n)=\sum_{\forall<i,j>\in I_r}\psi\left(e_{ij}\right)\ \left[J_{ij}(\Theta^n)\right]\ \left[J_{ij}(\Theta^n)\right]^T \tag{38}$$

Unfortunately, the Newton or Gauss-Newton not always reaches a minimum value for the error E, because only first order derivatives in the Taylor Expansion are used. More robust and better methods, like the one presented in subsection 0, expand the Newton or Gauss-Newton to avoid the case when $E(\Theta^{n+1}) > E(\Theta^n)$.

7.1 The Levenberg-Marquard Method

The Levenberg-Marquard method (LM) [Nocedal and Wright, 1999] is a non-linear iterative technique specifically designated for minimizing functions which has the form of Sum of Square functions, like E. At each iteration, the increment of parameters $\Delta\Theta$, is computed solving the following linear matrix equation:

$$(H+\Lambda)\Delta\Theta = G \tag{39}$$

Where G and H are defined by equation (11) and (38) respectively for GN, Λ is a diagonal matrix $\Lambda=diag(\lambda, \lambda, ..., \lambda)$, and λ is a variable parameter at each iteration.
The process starts with the input image, I, and the reference image, I_r, and initial values for parameters Θ^0. The
Algorithm 1 describes the LM method.

1. Pick a small value for λ (say $\lambda=0.001$), an initial value for Θ^0, an error function (for instance $\rho(e) = e^2$) and set $n=0$.
2. For a given Θ^n, compute the transformed image I_t^n (eq. (2)) applying bilinear interpolation to improve the quality of the image using equation (3).
3. Compute the total error, $E(\Theta^n)$ using equation (1).
4. Compute a new set of parameter using the following steps
   a. Compute $M(\Theta^n,i,j)$ and $N(\Theta^n,x,y)$ using, the equation (19) and (29) for projective transformation or the equation (17) and (31) for affine transformation, respectively.
   b. Compute the Gradient vector image $\nabla I_t^n(i,j)$ applying a derivative of Gaussian Filter (eq. (23))
   c. Compute the matrix $J_{ij}(\Theta^n) = M(\Theta^n,i,j)N(\Theta^n,x,y)\nabla I_t^n(i,j)$
   d. Compute the Gradient vector $G(\Theta^n)$ and Hessian matrix $H(\Theta^n)$ by equation (11) and (38) respectively.
   e. Solve the linear system of equations given by (39) for $\Delta\Theta$, and then calculate $E(\Theta^n + \Delta\Theta)$
5. If $E(\Theta^n + \Delta\Theta) >= E(\Theta^n)$, increase λ by a factor of 10, and go to step 4. If λ grows very large, it means that there is no way to improve the solution Θ^n and the algorithm ends with this solution $\Theta^* = \Theta^n$.
6. If $E(\Theta^n + \Delta\Theta) < E(\Theta^n)$, decrease λ by a factor of 10. Set $\Theta^{n+1} = \Theta^n + \Delta\Theta$, $n=n+1$ and go to the step 2.

Algorithm 1. The Levenberg-Marquard Method

Note when $\lambda = 0$, the LM method is a Gauss-Newton method, and when λ tends to infinity, $\Delta\Theta$ turns to so called steepest descent direction and the size of increments in $\Delta\Theta$ tends to zero.

8. Experimental results

To test the methods previously described, a computer program was built under the Redhat 9 Linux operating System, using the C language. All the experiments were running in a PC Pentium 4, 2.26 Ghz. and we use standard routines from the Free Gnu Scientific library

(GSL) to solve the linear system of equations. The error function was defined as a quadratic function ($\rho(e) = e^2$ with $\varphi(e) = 2e$ and $\psi(e) = 2$).
The PGM image format was selected because its simplicity to load and save gray level images with 256 gray levels, from 0 to 255. The PGM format is as follows,

```
P5 {nl}
# CREATOR: The GIMP's PNM Filter Version 1.0 {nl}
640 480 {nl}
255
<I(0,0)><I(0,1)>...<I(0,639)><I(1,0)><I(1,1)> ...
```

Where P5 means gray level images (P6 is reserved for color images), # starts a comment, 640 and 480 is the width and height of the image respectively, and {nl} is the new line character. <I(i,j)> is a single byte (an unsigned char in C) and they are ordered from left to right of the first row of pixels, then the second row of pixels, and so on.
Figure 4 shows two binary input images (Figure 4 (a) and Figure 4 (b)) and the associated reference image (Figure 4 (c)). Sequences of images for the three methods, to compute derivatives, are shown in Figures 5, 6 and 7; the number, below each Figure, indicates the number of iteration for Algorithm 1. In these cases the input image was the image shown in Figure 4 (a) and derivative of Gaussian Filter with $\sigma = 6$ was used. The numerical results for Algorithm 1, are presented in Table 1. The results of this Table show us, that methods 1 and 3 find the right transformation (a very low error), method 2 have the highest error and method 1 is the fastest.

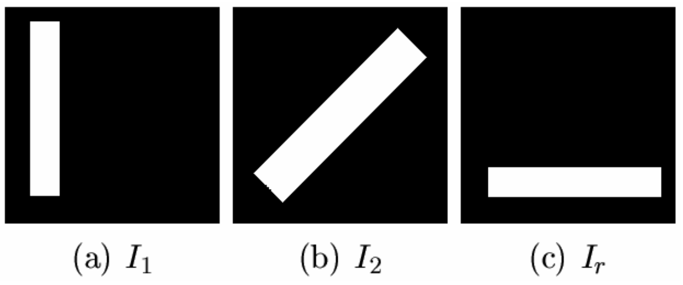


Figure 4. Input images (I_1 and I_2) and the reference image I_r. Images are of dimension *300 X 300*

Method	Time (seconds)	Iteration	Error
1	11	88	3.78 E-9
2	18	41	1902.6
3	18	49	3.22 E-9

Table 1. Comparing the three derivative methods for the test case of Figure 4(a) and (c)

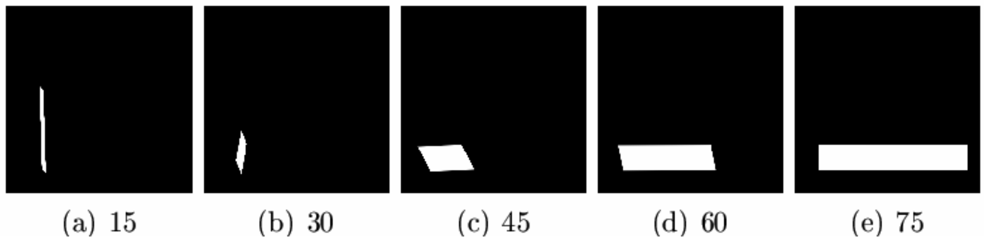


Figure 5. Comparing method 1, sequence a-b-c-d-e. The number of iteration of each image is shown

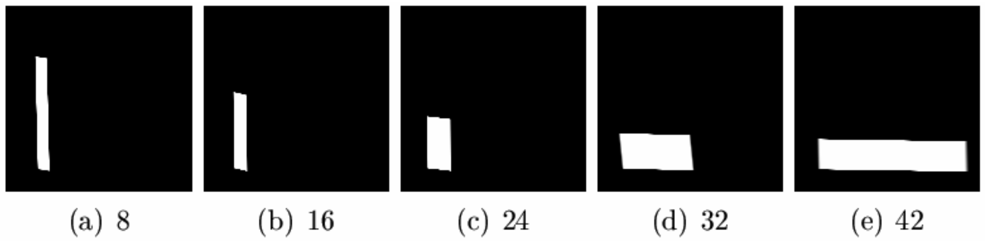


Figure 6. Comparing method 2, sequence a-b-c-d-e. The number of iteration of each image is shown

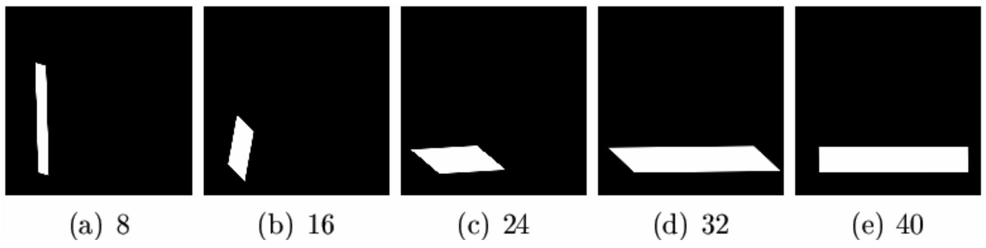


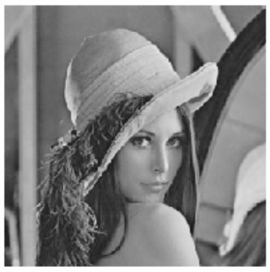
Figure 7. Comparing method 3, sequence a-b-c-d-e. The number of iteration of each image is shown

Numerical results for images of Figure 4 (b) and (c) are shown in Table 2. Again Methods 1 and 3 find the right transformation (a low error), but Method 2 have a higher error. In this case Method 1 is also the fastest.

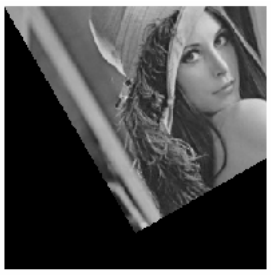
Method	Time (seconds)	Iteration	Error
1	12	70	1.59
2	26	59	51.51
3	27	44	1.59

Table 2. Comparing the three derivative methods for the test case of Figure 4(b) and (c)

Figure 8(a) and 8(b) shown an input image and the associated reference image respectively, with dimensions *256 X 256.* In this case, the case, the Algorithm 1, was applied two times; at first with derivatives of a Gaussian function with $\sigma = 10$ and then with $\sigma = 1$. In the first case with $\sigma = 10$, derivatives include information of a big window around the desired pixel value and so the derivatives are good enough to guide the search near to the right set of parameters. In the second, with $\sigma = 1$, derivatives are more accurate, given the previous set of parameters, and the final error, *E*, gets smaller values than in the first case. Final results for the three methods are shown in Figure 9. In this case only method 3 was able to find the right transformation.

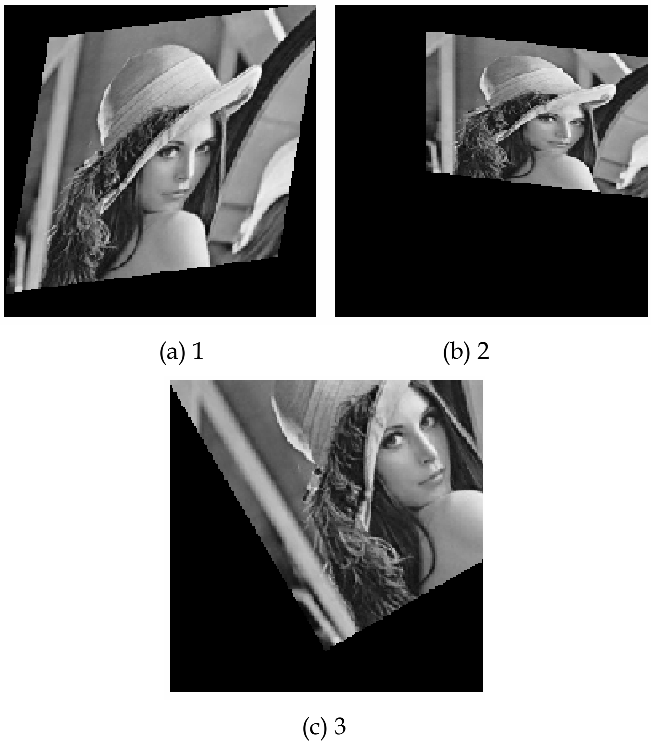


(a)



(b)

Figure 8. Another case of test. a) Input image and b) Image Reference


(a) 1 (b) 2

(c) 3

Figure 9. Final results of the three methods for image of Figure 8

9. Conclusions

A tutorial, with all the details related to the parametric image registration task, has been presented. Also two new methods (Method 1 and Method 3) are presented besides the classical method to compute derivatives of images.

Method 1 computes the image derivatives faster than the other two methods, but it does not give accurate estimations. Methods 2 and 3 take more time because they compute derivatives of the transformed image (at each iteration) while method 1 computes derivatives of the input image only once.

Method 3 is an improved version of method 2 because it takes into account the exact derivatives needed. However the classical method 2, reported in the literature is the same as the method 3 under translations. Experiments confirm the poor estimation computed by method 2 when rotations, scaling, or an affine transformation are involved.

In general, derivatives of images with big σ values are recommended when the right transformation is far away from identity, in other words, when big translations, rotations, scaling, etc., are involved. In contrast, small values of σ are recommended to get more accurate results. In fact, we mixed both strategies, big values at first and then small values.

10. References

[Adelson and Bergen, 1986] Adelson, E. and Bergen, J. R. (1986). The extraction of spatiotemporal energy in human and machine vision. *In IEEE, editor, IEEE Workshop on Visual Motion*, pp. 151-156, Charleston USA.

[Barman et al., 1986] Barman, H., Haglund, L., Knutsson, H., and Beauchemin, S. S. (1986). Estimation of velocity, acceleration and disparity in time sequences. *In Princeton, editor, IEEE Workshop on Visual Motion*, pp. 151-156, USA.

[Barron et al., 1994] Barron, J. L., Fleet, D. J., and Beauchemin, S. S. (1994). Performance of optical flow techniques. *International Journal of Computer vision*, 12(1): pp.43-47.

[Calderon and Marroquin, 2003] Calderon, F. and Marroquin, J. L. (2003). A new algorithm for computing optical flow and his application to image registration. *Computacion y Sistemas*, 6 (3) pp.213-226.

[Faugeras, 1993] Faugeras, O. (Nov 19 1993). *Three-Dimensional Computer Vision.* The MIT Press.

[Hartley and Zisserman, 2000] Hartley, R. and Zisserman, A. (2000). *Multiple View Geometry in Computer Vision.* Cambridge, University Press, second edition.

[Heeger, 1987] Heeger, D. (1987). Model for the extraction of image flow. *J. Opt. Soc Am*, 4: pp. 1455--1471.

[Huber, Peter J. 2003] Huber, Peter J. (Dec 23, 2003). *Robust Statistics*. Wiley Series in Probability and Statistics

[Kaneko et al., 2002] Kaneko, S., Murase, I., and Igarashi, S. (2002). Robust image registration by increment sign correlation. *Pattern Recognition*, 35: pp. 2223-2234.

[Kaneko et al., 2003] Kaneko, S., Satoh, Y., and Igarashi, S. (2003). Using selective correlation coefficient for robust image registration. *Pattern Recognition*, 29: pp. 1165-1173.

[Lai and Vemuri, 98] Lai, S. H. and Vemuri, B. C. (98). Reliable and efficient computation of optical flow. *International Journal of Computer vision*, 29(2) pp.87--105.

[Ma et al., 2004] Ma, Y., Soatto, S., Kosecka, J., and Sastry, S.S. (2004). *An Invitation to 3-D Vision From Images to Geometric Models.* Springer.

[Nocedal and Wright, 1999] Nocedal, J. and Wright, S. J. (1999). *Numerical Optimization.* Springer.

[Szeliski and Coughlan, 1994] Szeliski R. and Coughlan, J. (1994). Spline-based image registration. *Technical Report, Harvard University, Departmet of Physics*, Cambridge, Ma 02138.

[Romeny, 1994] Romeny, Bar ter Haar, editor (1994). *Geometry-Driven Diffusion in Computer Vision*. Kluwer Academic Publishers.

[Trucco and Verri, 1998] Trucco, E. and Verri, A. (1998). *Introductory techniques for 3-D Computer Vision.* Prentice Hall.

[Zitova and Flusser, 2003] Zitova, B. and Flusser, J. (2003). Image registration methods: A survey. *Image and vision Computing*. 21 pp. 977-1000