The Inverse Compositional Algorithm for Parametric Registration

Javier Sánchez

CTIM, University of Las Palmas de Gran Canaria, Spain (jsanchez@ulpgc.es)

Abstract

We present an implementation of the inverse compositional algorithm for parametric motion estimation. It computes a global motion between two images using a non-linear least square technique. Our implementation allows computing several types of planar transformations, such as translations, similarities, affinities or homographies. The algorithm is iterative so it typically yields solutions with high accuracy. The use of robust error functions, different from the L^2 norm, improves the stability of the method under the presence of noise and occlusions, and allows it to detect the predominant motion, even if there are several types of displacements. The method works with multi-channel images and makes use of a coarse–to–fine strategy for dealing with large displacements.

Source Code

The reviewed source code and documentation for this algorithm are available from the web page of this article[1]. Compilation and usage instruction are included in the `README.txt` file of the archive.

Keywords: inverse compositional algorithm; motion estimation; parametric motion; homography; rigid transformation; non-linear least square; robust error function

1 Introduction

The estimation of global parametric motion models is important in several problems, such as optical flow estimation, object tracking, video stabilization, image stitching or 3D reconstruction. The objective is to find the rigid transformation that puts in correspondence the pixels of two images.

In this article, we present an implementation of the *inverse compositional algorithm*, first proposed in [4] and thoroughly studied in [6]. This is part of a series of articles that introduced several improvements over the Lucas-Kanade method [13] for computing general parametric motion models. In the first article [6], the authors described several algorithms for image alignment and concentrated on the study of the inverse compositional algorithm.

---

[1]https://doi.org/10.5201/ipol.2016.153

Additionally, they proposed five more papers with theoretical justifications and improvements over the basic model: In part one [5], they studied the requirements on the set of warpings that are applicable to each strategy, the equivalence between the different algorithms and their performance using standard numerical methods (Gauss-Newton, Newton, diagonal Hessian, Levenberg-Marquardt and steepest descent); in part two [3] they analyzed different error functions, such as weighted L^2 norms and robust functions, its relation with noise, how to select pixels for improving the efficiency and how to deal with linear appearance variations; part three [1] deepened on the problem of linear appearance variation in the inverse compositional algorithm, with three different strategies for the L^2 norm and robust error functions; part four [2] discussed how to add priors on the parameters to the algorithms explained in the previous parts; finally, part five [7] generalized the inverse compositional algorithm to 3D volumetric data and surfaces. In our work, the main references are [6] and [3].

Our implementation is oriented to the estimation of planar transformations such as translations, Euclidean transformations, similarities, affinities, and homographies. Nevertheless, it may be easily adapted to other types of transformations with the requirement that they form a group. We have also included the use of robust error functions, as explained in [3]. This is interesting because it enables the algorithm to deal with occlusions, noise or multiple independent displacements: Typically, the algorithm can calculate the dominant motion, with high accuracy, and disregard values that do not fit in the model. We have also included a coarse–to–fine strategy for dealing with large motions and extended the algorithm to multi-channel images.

We first introduce the method of Lucas-Kanade in Section 2. In this section, we explain how to implement an incremental refinement, a coarse–to–fine strategy for estimating large displacements, and how to include robust error functions [11]. We finish this part with a complete description of the Lucas-Kanade algorithm. In Section 3, we extend this method to more general transformations. In Section 4, we introduce the inverse compositional algorithm and its variant for robust functions, which, nevertheless, is not so efficient in terms of computational complexity. These algorithms are explained in sections 4 and 4.3, which are the ones implemented.

2   The Lucas-Kanade Method

Let $\mathbf{I}_1(\mathbf{x}) := (I_1^1, I_1^2, \cdots, I_1^c)$ and $\mathbf{I}_2(\mathbf{x}) := (I_2^1, I_2^2, \cdots, I_2^c)$ be two images of multiple channels (c), with $\mathbf{x} = (x, y)$. Let $\mathbf{p} = (t_x, t_y)$ be the global displacement vector between the two images. Consider the following registration energy, depending on $\mathbf{p}$,

$$E(\mathbf{p}) = \sum_{\mathbf{x}} |\mathbf{I}_2(\mathbf{x} + \mathbf{p}) - \mathbf{I}_1(\mathbf{x})|_2^2, \tag{1}$$

with $|.|_2$ the Euclidean norm. This is a rather simple way of combining the color channels of both images, but there are other possible ways of using the color information, see [12] for instance. The Lucas-Kanade method assumes that the motion ($\mathbf{p}$) is constant in a neighborhood around $\mathbf{x}$. $\mathbf{I}_2(\mathbf{x}+\mathbf{p})$ can be linearized using first order Taylor expansions as $\mathbf{I}_2(\mathbf{x} + \mathbf{p}) \simeq \mathbf{I}_2(\mathbf{x}) + \nabla\mathbf{I}_2(\mathbf{x})\mathbf{p}$, where $\nabla\mathbf{I} := (\nabla I^1, \nabla I^2, \cdots, \nabla I^c)^T$ is the gradient of the image. The energy now reads as

$$E(\mathbf{p}) \simeq \sum_{\mathbf{x}} |\mathbf{I}_2(\mathbf{x}) + \nabla\mathbf{I}_2(\mathbf{x})\mathbf{p} - \mathbf{I}_1(\mathbf{x})|_2^2. \tag{2}$$

Minimizing the right hand term of Equation (2) and solving for $\mathbf{p}$,

$$\sum_{\mathbf{x}} \nabla\mathbf{I}_2^T(\mathbf{x}) \left(\mathbf{I}_2(\mathbf{x}) + \nabla\mathbf{I}_2(\mathbf{x})\mathbf{p} - \mathbf{I}_1(\mathbf{x})\right) = \mathbf{0}, \tag{3}$$

we obtain the solution as

$$\mathbf{p} = H^{-1} \sum_{\mathbf{x}} \nabla \mathbf{I}_2^T(\mathbf{x}) (\mathbf{I}_1(\mathbf{x}) - \mathbf{I}_2(\mathbf{x})), \tag{4}$$

with

$$H = \sum_{\mathbf{x}} \nabla \mathbf{I}_2^T(\mathbf{x}) \nabla \mathbf{I}_2(\mathbf{x}) = \left(\begin{array}{cc} \sum_{\mathbf{x}} \mathbf{I}_{2,x}^T(\mathbf{x}) \mathbf{I}_{2,x}(\mathbf{x}) & \sum_{\mathbf{x}} \mathbf{I}_{2,x}^T(\mathbf{x}) \mathbf{I}_{2,y}(\mathbf{x}) \\ \sum_{\mathbf{x}} \mathbf{I}_{2,x}^T(\mathbf{x}) \mathbf{I}_{2,y}(\mathbf{x}) & \sum_{\mathbf{x}} \mathbf{I}_{2,y}^T(\mathbf{x}) \mathbf{I}_{2,y}(\mathbf{x}) \end{array} \right). \tag{5}$$

the (Gauss-Newton approximation to the) *Hessian*[2] of the non-linear energy in Equation (1) or structure tensor matrix, and $\mathbf{I}_{2,x}^T(\mathbf{x}) \mathbf{I}_{2,x}(\mathbf{x}) = \left(\frac{\partial I_2^1}{\partial x} \right)^2 + \left(\frac{\partial I_2^2}{\partial x} \right)^2 + \cdots + \left(\frac{\partial I_2^c}{\partial x} \right)^2$.

2.1 Incremental Refinement

Due to the linearization of $\mathbf{I}_2(\mathbf{x} + \mathbf{p})$, the previous scheme provides an approximate solution near $\mathbf{p} = \mathbf{0}$. If we want to obtain a more accurate solution, we may use an iterative refinement in the following way

$$E(\Delta \mathbf{p}) = \sum_{\mathbf{x}} |\mathbf{I}_2(\mathbf{x} + \mathbf{p} + \Delta \mathbf{p}) - \mathbf{I}_1(\mathbf{x})|_2^2. \tag{6}$$

We assume that $\mathbf{p}$ is already known, so the aim is to compute an increment $\Delta \mathbf{p}$. The first order Taylor expansion is $\mathbf{I}_2(\mathbf{x} + \mathbf{p} + \Delta \mathbf{p}) \simeq \mathbf{I}_2(\mathbf{x} + \mathbf{p}) + \nabla \mathbf{I}_2(\mathbf{x} + \mathbf{p}) \Delta \mathbf{p}$. We can rewrite the energy as

$$E(\Delta \mathbf{p}) \simeq \sum_{\mathbf{x}} |\mathbf{I}_2(\mathbf{x} + \mathbf{p}) + \nabla \mathbf{I}_2(\mathbf{x} + \mathbf{p}) \Delta \mathbf{p} - \mathbf{I}_1(\mathbf{x})|_2^2, \tag{7}$$

and minimizing the right hand term with respect to $\Delta \mathbf{p}$ yields

$$\Delta \mathbf{p} = H^{-1} \sum_{\mathbf{x}} \nabla \mathbf{I}_2^T(\mathbf{x} + \mathbf{p}) (\mathbf{I}_1(\mathbf{x}) - \mathbf{I}_2(\mathbf{x} + \mathbf{p})), \tag{8}$$

with

$$H = \sum_{\mathbf{x}} \nabla \mathbf{I}_2^T(\mathbf{x} + \mathbf{p}) \nabla \mathbf{I}_2(\mathbf{x} + \mathbf{p}) = \left(\begin{array}{cc} \sum_{\mathbf{x}} \mathbf{I}_{2,x}^T(\mathbf{x} + \mathbf{p}) \mathbf{I}_{2,x}(\mathbf{x} + \mathbf{p}) & \sum_{\mathbf{x}} \mathbf{I}_{2,x}^T(\mathbf{x} + \mathbf{p}) \mathbf{I}_{2,y}(\mathbf{x} + \mathbf{p}) \\ \sum_{\mathbf{x}} \mathbf{I}_{2,x}^T(\mathbf{x} + \mathbf{p}) \mathbf{I}_{2,y}(\mathbf{x} + \mathbf{p}) & \sum_{\mathbf{x}} \mathbf{I}_{2,y}^T(\mathbf{x} + \mathbf{p}) \mathbf{I}_{2,y}(\mathbf{x} + \mathbf{p}) \end{array} \right). \tag{9}$$

Terms of the form $\mathbf{I}_2(\mathbf{x} + \mathbf{p})$ can be computed using bicubic interpolation. If the displacement is small, the initial value of the motion can be set to zero, $\mathbf{p}^0 := (0, 0)$. At each iteration, the solution for $\Delta \mathbf{p}^i$ is calculated and the parameters are updated as $\mathbf{p}^{i+1} := \mathbf{p}^i + \Delta \mathbf{p}^i$. This provides a better approximation that can be refined in subsequent iterations. The process is stopped when the magnitude of $\Delta \mathbf{p}^i$ is very small.

2.2 Coarse–to–fine approach

The incremental refinement allows us to calculate large motions using a coarse–to–fine approach. In order to estimate large displacements, we use a pyramidal structure. We follow the same strategy presented in previous IPOL articles [16, 14, 17] and reproduce here the basic ideas.

Our algorithm creates a pyramid of down-sampled images. The pyramid is created by reducing the images by a factor $\eta \in (0, 1)$. Before downsampling, the images are smoothed with a Gaussian

---

[2]The true Hessian is the matrix of second order derivatives. This is an approximate Hessian that neglects second (and higher) order derivatives of $\mathbf{I}_2(\mathbf{x} + \mathbf{p})$ [18]. In the rest of the text, we refer to it as simply the Hessian, to preserve the same name as in the original articles [6, 5, 3, 1, 2, 7] and in [18].

kernel of standard deviation depending on η. For a set of scales $s = 1, 2, \ldots, N_{scales}$, the pyramid of images is built as

$$I^s(\eta\mathbf{x}) := G_\sigma * I^{s-1}(\mathbf{x}). \tag{10}$$

After the convolution, the images are sampled using bicubic interpolation. The value of σ depends on η and is calculated as

$$\sigma(\eta) := \sigma_0\sqrt{\eta^{-2} - 1}, \text{ with } \sigma_0 := 0.6. \tag{11}$$

This allows preserving the same smoothing across the scales. The value of σ_0 is a small constant to reduce the noise of the original image and is found empirically, as explained in [14]. The motion is normally initialized as $\mathbf{p} := (0,0)$ and the system of equations is solved at the coarsest scale. This solution is then refined in the following scales. To transfer the values from a coarser scale, the transformation is updated as

$$\mathbf{p}^{s-1}(\mathbf{x}) := \frac{1}{\eta}\mathbf{p}^s(\eta\mathbf{x}). \tag{12}$$

This is valid in the case of translations. For other transformations, all the parameters are not updated in the same way, as we will see later.

2.3 Robust Error Functions

Robust functions reduce the influence of gross errors in the input data. They allow to find a model that correctly fits with the bulk of the data and remove *outliers*, which are points that do not conform with the estimated model. This is studied in the field of robust statistics [11] and it is interesting if we want to deal with problems like occlusions, noise, brightness changes or spurious motions. These functions are introduced in the energy as

$$E(\Delta\mathbf{p}) = \sum_{\mathbf{x}} \rho\left(|\mathbf{I}_2(\mathbf{x} + \mathbf{p} + \Delta\mathbf{p}) - \mathbf{I}_1(\mathbf{x})|_2^2, \lambda\right). \tag{13}$$

The purpose of function $\rho(\cdot)$ is to give less weight to large values of the argument, where the difference in image intensities is big. Typical functions are given in Table 1, see [9].

Applying first order Taylor expansions and minimizing the approximate functional that results from (13), we obtain

$$\Delta\mathbf{p} = H_\rho^{-1} \sum_{\mathbf{x}} \rho' \cdot \nabla\mathbf{I}_2^T(\mathbf{x} + \mathbf{p})\left(\mathbf{I}_1(\mathbf{x}) - \mathbf{I}_2(\mathbf{x} + \mathbf{p})\right), \tag{14}$$

with

$$H_\rho = \begin{pmatrix} \sum_{\mathbf{x}} \rho' \cdot \mathbf{I}_{2,x}^T(\mathbf{x} + \mathbf{p})\mathbf{I}_{2,x}(\mathbf{x} + \mathbf{p}) & \sum_{\mathbf{x}} \rho' \cdot \mathbf{I}_{2,x}^T(\mathbf{x} + \mathbf{p})\mathbf{I}_{2,y}(\mathbf{x} + \mathbf{p}) \\ \sum_{\mathbf{x}} \rho' \cdot \mathbf{I}_{2,x}^T(\mathbf{x} + \mathbf{p})\mathbf{I}_{2,y}(\mathbf{x} + \mathbf{p}) & \sum_{\mathbf{x}} \rho' \cdot \mathbf{I}_{2,y}^T(\mathbf{x} + \mathbf{p})\mathbf{I}_{2,y}(\mathbf{x} + \mathbf{p}) \end{pmatrix}, \tag{15}$$

and $\rho' := \rho'\left(|\mathbf{I}_2(\mathbf{x} + \mathbf{p} + \Delta\mathbf{p}) - \mathbf{I}_1(\mathbf{x})|_2^2, \lambda\right)$. This factor weighs down the influence of pixels $\mathbf{x}$ for which the error is high.

2.4 Lucas-Kanade Algorithm

The Lucas-Kanade method is outlined in Algorithm 1. The input parameters are the two images, an initial approximation, and a threshold, ϵ, that is used to stop the iterative process. The convergence criterion is $\|\Delta\mathbf{p}\| < \epsilon$, where ϵ is a small constant, e.g., $\epsilon := 10^{-3}$.

This algorithm refines an initial estimate of $\mathbf{p}$ at each scale. The method may converge very slowly to the solution, so it is necessary to stop it if a maximum number of iterations is exceeded. In
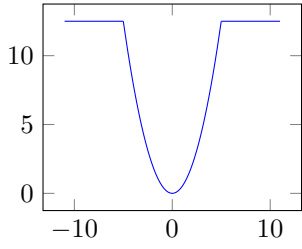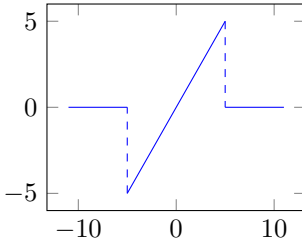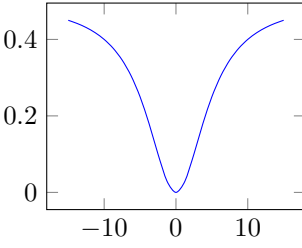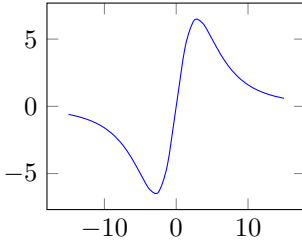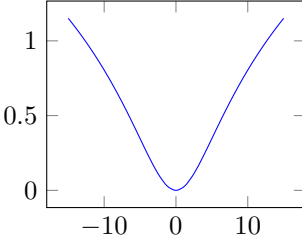
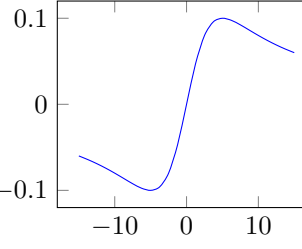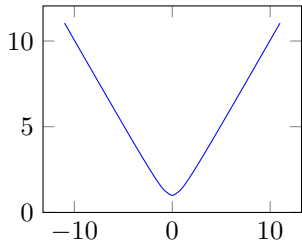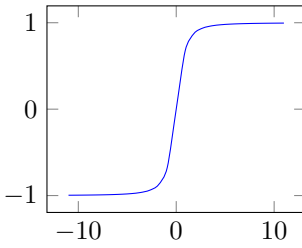| Type	$\rho(s^2, \lambda)$	$\rho'(s^2, \lambda)s$
Truncated Quadratic	$\rho(s^2, \lambda) = \begin{cases} \frac{1}{2}s^2 & \text{if } s^2 < \lambda^2 \\ \frac{1}{2}\lambda^2 & \text{otherwise} \end{cases}$	$\rho'(s^2, \lambda)s = \begin{cases} s & \text{if } s^2 < \lambda^2 \\ 0 & \text{otherwise} \end{cases}$
Geman & McClure	$\rho(s^2, \lambda) = \frac{1}{2}\frac{s^2}{\lambda^2+s^2}$	$\rho'(s^2, \lambda)s = \frac{\lambda^2 s}{(\lambda^2+s^2)^2}$
Lorentzian	$\rho(s^2, \lambda) = \frac{1}{2}\log\left(1 + \left(\frac{s}{\lambda}\right)^2\right)$	$\rho'(s^2, \lambda)s = \frac{s}{\lambda^2+s^2}$
Charbonnier	$\rho(s^2, \lambda) = 2\sqrt{s^2 + \lambda^2}$	$\rho'(s^2, \lambda)s = \frac{s}{\sqrt{s^2+\lambda^2}}$

Table 1: Examples of robust error functions. The value of λ is 5, except for the Charbonnier function $(\lambda = 1)$. This is a differentiable approximation to the L^1 norm that is typically used in variational optical flow methods, like in [17].

the presence of large displacements, this algorithm should be called at each scale using the solution obtained at previous scales.

Algorithm 1: Lucas-Kanade for translations

> **input** : $\mathbf{I}_1, \mathbf{I}_2, \mathbf{p}, \epsilon$
> **output**: $\mathbf{p}$
> **repeat**
> > Calculate $\mathbf{I}_2(\mathbf{x} + \mathbf{p})$ and $\nabla\mathbf{I}_2(\mathbf{x} + \mathbf{p})$ using bilinear or bicubic interpolation
> > Compute $\mathbf{I}_1(\mathbf{x}) - \mathbf{I}_2(\mathbf{x} + \mathbf{p})$
> > Compute the Hessian matrix using Equation (9) `// Equation (15) for the robust version`
> > Compute $\sum_{\mathbf{x}} \nabla\mathbf{I}_2^T(\mathbf{x} + \mathbf{p})\left(\mathbf{I}_1(\mathbf{x}) - \mathbf{I}_2(\mathbf{x} + \mathbf{p})\right)$ `// Equation (14) for the robust version`
> > Solve for $\Delta\mathbf{p}$ using Equation (8) `// Equation (14) for the robust version`
> > $\mathbf{p} \leftarrow \mathbf{p} + \Delta\mathbf{p}$
> **until** $\|\Delta\mathbf{p}\| < \epsilon$

Algorithm 2 implements the coarse–to–fine strategy. It creates the pyramid of images, calls the Lucas-Kanade algorithm at each scale and updates the transformation for the next finer scale. The final solution corresponds to the transformation computed at scale $s = 1$, using the original images. Note that it is not necessary a high accuracy at the coarsest scales, so we may run less iterations. This can be easily achieved, for instance, increasing the value of ϵ for $s > 1$.

Algorithm 2: Coarse–to–fine approach

> **input** : $\mathbf{I}_1, \mathbf{I}_2, N_{scales}, \eta, \epsilon$
> **output**: $\mathbf{p}$
> Create the pyramid of images $\mathbf{I}_1^s, \mathbf{I}_2^s$ using η and $s = 1, \ldots, N_{scales}$
> $\mathbf{p}_{scales}^N \leftarrow \mathbf{0}$
> **for** $s \leftarrow N_{scales}$ **to** 1 **do**
> > Lucas-Kanade for translations($\mathbf{I}_1^s, \mathbf{I}_2^s, \mathbf{p}^s, \epsilon$)
> > **if** $s > 1$ **then**
> > > $\mathbf{p}^{s-1} := \frac{1}{\eta}\mathbf{p}^s$

The higher computational costs are due to the bilinear/bicubic interpolations and the estimation of the Hessian. The complexity of the algorithm is $O(2N)$ at each iteration, with N the number of pixels. If we take into account the number of iterations, i, and the number of scales, s, the complexity is $O\left(i\frac{2N}{3}\left(4 - \frac{1}{4^s}\right)\right)$. The algorithm converges very fast, so the number of iterations is usually small (we fix its maximum to 30). The number of scales must be set according to the largest expected displacements, e.g., $s = 5$ allows to detect motions up to 32 pixels distance.

3   Parametric Motion Estimation

Instead of translations, we may compute more general motions between the two images. Typical deformations include rotations, similarities, affinities and homographies. Let $\mathbf{x}'(\mathbf{x}; \mathbf{p}, \Delta\mathbf{p})$ be the correspondence map from the left to the right image, parameterized by $\mathbf{p}$ and the incremental refinement $\Delta\mathbf{p}$. We consider the transformations in Table 2.

The corresponding functional is given by

$$E(\Delta\mathbf{p}) = \sum_{\mathbf{x}} |\mathbf{I}_2(\mathbf{x}'(\mathbf{x}; \mathbf{p}, \Delta\mathbf{p})) - \mathbf{I}_1(\mathbf{x})|_2^2. \tag{16}$$

As in Section 2, $\mathbf{I}_2(\mathbf{x}'(\mathbf{x}; \mathbf{p}, \Delta\mathbf{p}))$ can be linearized using first order Taylor expansions

$$\mathbf{I}_2(\mathbf{x}'(\mathbf{x}; \mathbf{p}, \Delta\mathbf{p})) \simeq \mathbf{I}_2(\mathbf{x}'(\mathbf{x}; \mathbf{p})) + \nabla\mathbf{I}_2(\mathbf{x}'(\mathbf{x}; \mathbf{p}))\mathbf{J}(\mathbf{x}; \mathbf{p})\Delta\mathbf{p}, \tag{17}$$

where $\mathbf{J}(\mathbf{x}; \mathbf{p}) = \frac{\partial \mathbf{x}'(\mathbf{x};\mathbf{p})}{\partial \mathbf{p}}$ is the Jacobian of the transformation. The expression $\nabla\mathbf{I}_2(\mathbf{x}'(\mathbf{x}; \mathbf{p}))\mathbf{J}(\mathbf{x}; \mathbf{p})$ is called the *steepest descent* image, see [6] for an explanation. Table 2 lists the transformations and their Jacobians using the parametrizations proposed in [18].

Transform	Parameters – p	Matrix – H(p)	Jacobian – J(x; p)
Translation	(t_x, t_y)	$\begin{pmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{pmatrix}$	$\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$
Euclidean	(t_x, t_y, θ)	$\begin{pmatrix} \cos\theta & -\sin\theta & t_x \\ \sin\theta & \cos\theta & t_y \\ 0 & 0 & 1 \end{pmatrix}$	$\begin{pmatrix} 1 & 0 & -x\sin\theta - y\cos\theta \\ 0 & 1 & x\cos\theta - y\sin\theta \end{pmatrix}$
Similarity	(t_x, t_y, a, b)	$\begin{pmatrix} 1+a & -b & t_x \\ b & 1+a & t_y \\ 0 & 0 & 1 \end{pmatrix}$	$\begin{pmatrix} 1 & 0 & x & -y \\ 0 & 1 & y & x \end{pmatrix}$
Affinity	$(t_x, t_y, a_{11}, a_{12}, a_{21}, a_{22})$	$\begin{pmatrix} 1+a_{11} & a_{12} & t_x \\ a_{21} & 1+a_{22} & t_y \\ 0 & 0 & 1 \end{pmatrix}$	$\begin{pmatrix} 1 & 0 & x & y & 0 & 0 \\ 0 & 1 & 0 & 0 & x & y \end{pmatrix}$
Homography	$(h_{11}, h_{12}, h_{13}, \cdots, h_{32})$	$\begin{pmatrix} 1+h_{11} & h_{12} & h_{13} \\ h_{21} & 1+h_{22} & h_{23} \\ h_{31} & h_{32} & 1 \end{pmatrix}$	$\frac{1}{D}\begin{pmatrix} \mathbf{x}^T & \mathbf{0}^T & -x'x & -x'y \\ \mathbf{0}^T & \mathbf{x}^T & -y'x & -y'y \end{pmatrix}$ $D = h_{31}x + h_{32}y + 1$

Table 2: Planar transformations in homogeneous coordinates and their Jacobians, with $\mathbf{x} = (x, y, 1)^T$ and $\mathbf{0} = (0, 0, 0)^T$.

The energy is now

$$E(\Delta\mathbf{p}) \simeq \sum_{\mathbf{x}} |\mathbf{I}_2(\mathbf{x}'(\mathbf{x}; \mathbf{p})) + \nabla\mathbf{I}_2(\mathbf{x}'(\mathbf{x}; \mathbf{p}))\mathbf{J}(\mathbf{x}; \mathbf{p})\Delta\mathbf{p} - \mathbf{I}_1(\mathbf{x})|_2^2. \tag{18}$$

Minimizing the right hand term of (18) yields

$$\Delta\mathbf{p} = H_J^{-1} \sum_{\mathbf{x}} \left(\nabla\mathbf{I}_2(\mathbf{x}'(\mathbf{x}; \mathbf{p}))\mathbf{J}(\mathbf{x}; \mathbf{p})\right)^T \left(\mathbf{I}_1(\mathbf{x}) - \mathbf{I}_2(\mathbf{x}'(\mathbf{x}; \mathbf{p}))\right), \tag{19}$$

with

$$H_J = \sum_{\mathbf{x}} \left(\nabla\mathbf{I}_2(\mathbf{x}'(\mathbf{x}; \mathbf{p}))\mathbf{J}(\mathbf{x}; \mathbf{p})\right)^T \nabla\mathbf{I}_2(\mathbf{x}'(\mathbf{x}; \mathbf{p}))\mathbf{J}(\mathbf{x}; \mathbf{p})$$

$$= \begin{pmatrix} \sum_{\mathbf{x}} (\mathbf{I}_{2,x}(\mathbf{x}')\mathbf{J}(\mathbf{x}; \mathbf{p}))^T \mathbf{I}_{2,x}(\mathbf{x}')\mathbf{J}(\mathbf{x}; \mathbf{p}) & \sum_{\mathbf{x}} (\mathbf{I}_{2,x}(\mathbf{x}')\mathbf{J}(\mathbf{x}; \mathbf{p}))^T \mathbf{I}_{2,y}(\mathbf{x}')\mathbf{J}(\mathbf{x}; \mathbf{p}) \\ \sum_{\mathbf{x}} (\mathbf{I}_{2,x}(\mathbf{x}')\mathbf{J}(\mathbf{x}; \mathbf{p}))^T \mathbf{I}_{2,y}(\mathbf{x}')\mathbf{J}(\mathbf{x}; \mathbf{p}) & \sum_{\mathbf{x}} (\mathbf{I}_{2,y}(\mathbf{x}')\mathbf{J}(\mathbf{x}; \mathbf{p}))^T \mathbf{I}_{2,y}(\mathbf{x}')\mathbf{J}(\mathbf{x}; \mathbf{p}) \end{pmatrix}. \tag{20}$$

In the case of a homography, the transformation is $\mathbf{x}' := \mathbf{H}(\mathbf{p})\mathbf{x}$. $\mathbf{H}$ is a 3×3 matrix in homogeneous coordinates [10], as in Table 2, and it is represented by eight parameters, $\mathbf{p} = (h_{11}, h_{12}, h_{13}, \cdots, h_{32})$, where the last component is fixed, $h_{33} = 1$. Since we are using homogeneous coordinates, we need to divide by the last element of the vector as

$$x' = \frac{(1 + h_{11})x + h_{12}y + h_{13}}{h_{31}x + h_{32}y + 1} \;\; \text{and} \;\; y' = \frac{h_{21}x + (1 + h_{22})y + h_{23}}{h_{31}x + h_{32}y + 1}. \tag{21}$$

The algorithm for this parametric model is very similar to the Lucas-Kanade algorithm. Its complexity is $O(n^2 N)$, with n the number of parameters and N the number of pixels. Taking into account the number of iterations, i, and the number of scales, s, the complexity is $O\left(i \frac{n^2 N}{3}\left(4 - \frac{1}{4^s}\right)\right)$.

4   The Inverse Compositional Algorithm

The inverse compositional algorithm is an efficient registration technique whose iterative process is based on the composition of two transformations, one of them inverse as $\mathbf{x}'(\mathbf{x}; \Delta\mathbf{p})^{-1}$.

During the incremental refinement process, the Lucas-Kanade algorithm has to compute the Hessian at every iteration. This, together with the interpolation of the second image, is the most computationally expensive step. Note that the size of this matrix depends on the number of parameters.

Various alternatives have appeared trying to improve this step, see [6] and [18]. In the inverse compositional algorithm this is solved very efficiently, since the Hessian matrix remains constant during the iterations. This is achieved by separating both unknowns, $\mathbf{p}$ and $\Delta\mathbf{p}$, in the two images as

$$E(\Delta\mathbf{p}) = \sum_{\mathbf{x}} |\mathbf{I}_2(\mathbf{x}'(\mathbf{x}; \mathbf{p})) - \mathbf{I}_1(\mathbf{x}'(\mathbf{x}; \Delta\mathbf{p}))|_2^2. \tag{22}$$

Now the first order Taylor expansion for $\mathbf{I}_1(\mathbf{x}'(\mathbf{x}; \Delta\mathbf{p})$ is

$$\mathbf{I}_1(\mathbf{x}'(\mathbf{x}; \Delta\mathbf{p})) \simeq \mathbf{I}_1(\mathbf{x}) + \nabla\mathbf{I}_1(\mathbf{x})\mathbf{J}(\mathbf{x})\Delta\mathbf{p}. \tag{23}$$

We assume that $\mathbf{x}'(\mathbf{x}; \Delta\mathbf{p})$, evaluated at $\Delta\mathbf{p} := \mathbf{0}$, is the identity, i.e., $\mathbf{x}'(\mathbf{x}; \mathbf{0}) = \mathbf{x}$. Note that the first image and its gradient do not have to be interpolated. The assumption behind this formulation is that the gradients of $\mathbf{I}_2(\mathbf{x}'(\mathbf{x}; \mathbf{p}))$ and $\mathbf{I}_1(\mathbf{x})$ are similar when $\mathbf{p}$ is close to the solution. The update rule for the parameters is now

$$\mathbf{x}'(\mathbf{x}; \mathbf{p}) = \mathbf{x}'(\mathbf{x}; \mathbf{p}) \circ (\mathbf{x}'(\mathbf{x}; \Delta\mathbf{p}))^{-1}. \tag{24}$$

In the case of the transformations in Table 2, this is equivalent to inverting and multiplying two 3×3 matrices. In practice, it is not necessary to multiply matrices, since the operations can be left as a function of the parameters. The new linearized energy is

$$E(\Delta\mathbf{p}) = \sum_{\mathbf{x}} |\mathbf{I}_2(\mathbf{x}'(\mathbf{x}; \mathbf{p})) - \nabla\mathbf{I}_1(\mathbf{x})\mathbf{J}(\mathbf{x})\Delta\mathbf{p} - \mathbf{I}_1(\mathbf{x})|_2^2. \tag{25}$$

Then, minimizing the second term of (25) yields

$$\Delta\mathbf{p} = H_I^{-1} \sum_{\mathbf{x}} (\nabla\mathbf{I}_1(\mathbf{x})\mathbf{J}(\mathbf{x}))^T (\mathbf{I}_2(\mathbf{x}'(\mathbf{x}; \mathbf{p})) - \mathbf{I}_1(\mathbf{x})), \tag{26}$$

with

$$H_I = \sum_{\mathbf{x}} \left(\nabla \mathbf{I}_1(\mathbf{x}) \mathbf{J}(\mathbf{x}) \right)^T \nabla \mathbf{I}_1(\mathbf{x}) \mathbf{J}(\mathbf{x})$$

$$= \begin{pmatrix} \sum_{\mathbf{x}} \left(\mathbf{I}_{1,x}(\mathbf{x}) \mathbf{J}(\mathbf{x}) \right)^T \mathbf{I}_{1,x}(\mathbf{x}) \mathbf{J}(\mathbf{x}) & \sum_{\mathbf{x}} \left(\mathbf{I}_{1,x}(\mathbf{x}) \mathbf{J}(\mathbf{x}) \right)^T \mathbf{I}_{1,y}(\mathbf{x}) \mathbf{J}(\mathbf{x}) \\ \sum_{\mathbf{x}} \left(\mathbf{I}_{1,x}(\mathbf{x}) \mathbf{J}(\mathbf{x}) \right)^T \mathbf{I}_{1,y}(\mathbf{x}) \mathbf{J}(\mathbf{x}) & \sum_{\mathbf{x}} \left(\mathbf{I}_{1,y}(\mathbf{x}) \mathbf{J}(\mathbf{x}) \right)^T \mathbf{I}_{1,y}(\mathbf{x}) \mathbf{J}(\mathbf{x}) \end{pmatrix} . \tag{27}$$

The clear benefit of this scheme is that $\nabla \mathbf{I}_1(\mathbf{x})$, $\mathbf{J}(\mathbf{x})$ and, thus, H_I do not depend on $\Delta \mathbf{p}$ and are constant through the iterations. These, together with the inverse of the Hessian, can be precomputed. Another benefit is that it is no longer necessary to calculate $\nabla \mathbf{I}_2(\mathbf{x}')$, which would require two bilinear/bicubic interpolations.

On the contrary, one of the drawbacks of this approach is that it is slightly more sensitive than the Lucas-Kanade method when there is noise in both images, as explained in [6]. Another shortcoming of this algorithm is that the transformation must be invertible. In the case of polynomials, it is difficult to find the inverse for degrees higher than one, however, we may compute an approximate inverse, $q(y)$, of a polynomial $p(x)$ on a bounded domain by minimizing an energy $E(q) = \int |q(p(x)) - x|^2$, which yields a system of linear equations.

Algorithm 3: Inverse Compositional Algorithm

input : $\mathbf{I}_1$, $\mathbf{I}_2$, $\mathbf{p}$, ϵ
output: $\mathbf{p}$
Compute $\nabla \mathbf{I}_1(\mathbf{x})$
Compute the Jacobian $\mathbf{J}(\mathbf{x})$
Compute $\nabla \mathbf{I}_1(\mathbf{x}) \mathbf{J}(\mathbf{x})$
Compute the Hessian using Equation (27)
repeat
    Calculate $\mathbf{I}_2(\mathbf{x}'(\mathbf{x}; \mathbf{p}))$ using bilinear or bicubic interpolation
    Compute $\mathbf{I}_2(\mathbf{x}'(\mathbf{x}; \mathbf{p})) - \mathbf{I}_1(\mathbf{x})$
    Compute $\sum_{\mathbf{x}} \left(\nabla \mathbf{I}_1(\mathbf{x}) \mathbf{J}(\mathbf{x}) \right)^T \left(\mathbf{I}_2(\mathbf{x}'(\mathbf{x}; \mathbf{p})) - \mathbf{I}_1(\mathbf{x}) \right)$
    Solve for $\Delta \mathbf{p}$ using Equation (26)
    $\mathbf{x}'(\mathbf{x}; \mathbf{p}) \leftarrow \mathbf{x}'(\mathbf{x}; \mathbf{p}) \circ (\mathbf{x}'(\mathbf{x}; \Delta \mathbf{p}))^{-1}$
until $\|\Delta \mathbf{p}\| < \epsilon$

The inverse compositional algorithm is shown in Algorithm 3. This algorithm is called from a coarse–to–fine scheme, similar to Algorithm 2. The main difference is that the transformations are updated from scale to scale according to Table 3.

The complexity of this algorithm is $O(n^2 N)$ before the iterative process. At each iteration, however, it reduces to $O(nN)$. Thus, taking into account the number of iterations, i, and scales, s, the computational cost is $O\left(n^2 N + i \frac{nN}{3} \left(4 - \frac{1}{4^s}\right)\right)$. The clear benefit of this strategy comes up when the number of iterations is large.

4.1 Robust Error Functions

In the same way as in the Lucas-Kanade method, we can use robust functions to cope with noise and occlusions, as explained in Section 2.3 (this part is explained in [3]). However, their use in the inverse compositional algorithm has the main drawback that the Hessian cannot be precomputed

Transform	Update from scale s to $s-1$
Translation	$(t_x, t_y)^{s-1} := \frac{1}{\eta}(t_x, t_y)^s$
Euclidean	$(t_x, t_y, \theta)^{s-1} := \left(\frac{1}{\eta}t_x, \frac{1}{\eta}t_y, \theta\right)^s$
Similarity	$(t_x, t_y, a, b)^{s-1} := \left(\frac{1}{\eta}t_x, \frac{1}{\eta}t_y, a, b\right)^s$
Affinity	$(t_x, t_y, a_{11}, a_{12}, a_{21}, a_{22})^{s-1} := \left(\frac{1}{\eta}t_x, \frac{1}{\eta}t_y, a_{11}, a_{12}, a_{21}, a_{22}\right)^s$
Homography	$(h_{11}, h_{12}, h_{13}, h_{21}, h_{22}, h_{23}, h_{31}, h_{32})^{s-1} := (h_{11}, h_{12}, \frac{1}{\eta}h_{13}, h_{21}, h_{22}, \frac{1}{\eta}h_{23}, \eta h_{31}, \eta h_{32})^s$

Table 3: Update rule for the transformations in the coarse–to–fine scheme.

anymore. The energy is

$$E(\Delta\mathbf{p}) = \sum_{\mathbf{x}} \rho\left(|\mathbf{I}_2(\mathbf{x}'(\mathbf{x}; \mathbf{p})) - \mathbf{I}_1(\mathbf{x}'(\mathbf{x}; \Delta\mathbf{p}))|_2^2\right), \tag{28}$$

where $\rho\left(\cdot\right)$ is one of the functions in Table 1. The solution is

$$\Delta\mathbf{p} = H_\delta^{-1} \sum_{\mathbf{x}} \rho' \cdot (\nabla\mathbf{I}_1(\mathbf{x})\mathbf{J}(\mathbf{x}))^T (\mathbf{I}_2(\mathbf{x}'(\mathbf{x}; \mathbf{p})) - \mathbf{I}_1(\mathbf{x})), \tag{29}$$

with

$$H_\delta = \sum_{\mathbf{x}} \rho' \cdot (\nabla\mathbf{I}_1(\mathbf{x})\mathbf{J}(\mathbf{x}))^T \nabla\mathbf{I}_1(\mathbf{x})\mathbf{J}(\mathbf{x})$$
$$= \begin{pmatrix} \sum_{\mathbf{x}} \rho' \cdot (\mathbf{I}_{1,x}(\mathbf{x})\mathbf{J}(\mathbf{x}))^T \mathbf{I}_{1,x}(\mathbf{x})\mathbf{J}(\mathbf{x}) & \sum_{\mathbf{x}} \rho' \cdot (\mathbf{I}_{1,x}(\mathbf{x})\mathbf{J}(\mathbf{x}))^T \mathbf{I}_{1,y}(\mathbf{x})\mathbf{J}(\mathbf{x}) \\ \sum_{\mathbf{x}} \rho' \cdot (\mathbf{I}_{1,x}(\mathbf{x})\mathbf{J}(\mathbf{x}))^T \mathbf{I}_{1,y}(\mathbf{x})\mathbf{J}(\mathbf{x}) & \sum_{\mathbf{x}} \rho' \cdot (\mathbf{I}_{1,y}(\mathbf{x})\mathbf{J}(\mathbf{x}))^T \mathbf{I}_{1,y}(\mathbf{x})\mathbf{J}(\mathbf{x}) \end{pmatrix}, \tag{30}$$

and $\rho' := \rho'\left(|\mathbf{I}_2(\mathbf{x}'(\mathbf{x}; \mathbf{p})) - \mathbf{I}_1(\mathbf{x})|_2^2\right)$. In this case, $\rho'(\cdot)$ must be re-evaluated during the iterations. Therefore, the Hessian and its inverse must also be computed inside the iterative process. Nevertheless, the gradient of the image needs not be interpolated.

The cost of the inverse compositional algorithm is $O(nN)$ per iteration, whereas the inclusion of robust error functions increases the cost to $O(n^2N)$, see [3]. This means that the order of the algorithm is still linear in the number of pixels but quadratic in the number of parameters.

4.2   Numerical Details and Parameters

The gradient of the image is calculated using central differences. We use bicubic interpolation for warping the second image, $\mathbf{I}_2(\mathbf{x}'(\mathbf{x}; \mathbf{p}))$, and for downsampling the images in the coarse–to–fine scheme. When $\mathbf{x}'(\mathbf{x}; \mathbf{p})$ falls beyond the dimensions of $\mathbf{I}_2(\mathbf{x})$, we assign a default value of 0.

This method depends on the parameters given in Table 4. These parameters are: The threshold for the robust function, λ; the parameters for the pyramidal scheme, given by the number of scales, N_{scales}, and the downsampling factor, η; and the parameters for the incremental refinement, ϵ and MAXITER.

Parameter	Description
λ	Robust function threshold (see Table 1).
N_{scales}	Number of scales in the pyramidal structure.
η	Downsampling factor to create the pyramidal structure.
ϵ	Stopping criterion threshold for the incremental refinement.
MAXITER	Maximun number of iterations in the incremental refinement.

Table 4: Parameters of the method

Most of these parameters may be fixed. In the experiments, we choose the Lorentzian function. Its threshold is initialized to a big value, $\lambda := 80$, and is reduced during the iterations as $\lambda^i := 0.9\lambda^{i-1}$, until $\lambda := 5$. This strategy is followed by several works, such as [15] and [9]. This is a continuation method that successively reduces the influence of outliers: Initially, a large value of the parameter allows to consider all the points as inliers and, as the model is being approximated, outliers are gradually eliminated.

It is possible to estimate the value of λ according to the noise of the sequence. In the case of the Lorentzian, $\lambda := \frac{\tau}{\sqrt{2}}$, with τ the maximum expected intensity difference to be considered as inlier, is the zero of the second derivative of the robust function, see [9]. From this value, the influence of the outliers starts to decrease. The value of τ can be approximated from the standard deviation of the noise, like in [15].

The number of scales, N_{scales}, is chosen so that the size of the coarsest images is bigger than 32×32 pixels. The downsampling factor is set to $\eta := 0.5$ (the images are reduced to half their size), the threshold $\epsilon := 10^{-3}$ and MAXITER$:= 30$.

4.3 Robust Inverse Compositional Algorithm

The implementation of the method is detailed in Algorithm 4.

Algorithm 4: Robust Inverse Compositional Algorithm

input : $\mathbf{I}_1$, $\mathbf{I}_2$, $\mathbf{p}$, ϵ
output: $\mathbf{p}$
Compute $\nabla \mathbf{I}_1(\mathbf{x})$
Compute the Jacobian $\mathbf{J}(\mathbf{x})$
Compute $\nabla \mathbf{I}_1(\mathbf{x})\mathbf{J}(\mathbf{x})$
repeat
  Calculate $\mathbf{I}_2(\mathbf{x}'(\mathbf{x};\mathbf{p}))$ using bilinear or bicubic interpolation
  Compute $\mathbf{I}_2(\mathbf{x}'(\mathbf{x};\mathbf{p})) - \mathbf{I}_1(\mathbf{x})$
  Compute the Hessian using Equation (30)
  Compute $\sum_{\mathbf{x}} \rho' \cdot (\nabla \mathbf{I}_1(\mathbf{x})\mathbf{J}(\mathbf{x}))^T (\mathbf{I}_2(\mathbf{x}'(\mathbf{x};\mathbf{p})) - \mathbf{I}_1(\mathbf{x}))$
  Solve for $\Delta\mathbf{p}$ using Equation (29)
  $\mathbf{x}'(\mathbf{x};\mathbf{p}) \leftarrow \mathbf{x}'(\mathbf{x};\mathbf{p}) \circ (\mathbf{x}'(\mathbf{x};\Delta\mathbf{p}))^{-1}$
until $\|\Delta\mathbf{p}\| < \epsilon$

This algorithm is called from the coarse–to–fine scheme of Algorithm 2 with the main difference that the transformations are updated from scale to scale according to Table 3.

The complexity of this algorithm is $O(nN)$ before the iterative process. At each iteration, it increases to $O(n^2N)$ and, including the number of iterations and scales, the computational cost is

$O\left(nN + i\frac{n^2N}{3}\left(4 - \frac{1}{4^s}\right)\right)$. The cost is slightly better than the Lucas-Kanade algorithm. The main benefit is that it allows to precompute some information and avoid the interpolation of the image gradient during the iterative process.

The use of robust functions makes the inverse compositional algorithm less efficient because the Hessian must be computed at every iteration. One way to speed-up the algorithm is to exploit the spatial coherence of outliers [3]: $\mathbf{I}_1$ can be divided in K blocks $B_1, B_2, \ldots, B_K$ and the Hessian can be computed as

$$H_\delta = \sum_{i=1}^{K} \sum_{\mathbf{x} \in B_i} \rho' \cdot (\nabla \mathbf{I}_1(\mathbf{x})\mathbf{J}(\mathbf{x}))^T \nabla \mathbf{I}_1(\mathbf{x})\mathbf{J}(\mathbf{x}). \tag{31}$$

If we suppose that ρ' is constant in each block, e.g., taken as the minimum or the mean value, then

$$H_\delta = \sum_{i=1}^{K} \rho'_i \cdot \sum_{\mathbf{x} \in B_i} (\nabla \mathbf{I}_1(\mathbf{x})\mathbf{J}(\mathbf{x}))^T \nabla \mathbf{I}_1(\mathbf{x})\mathbf{J}(\mathbf{x}). \tag{32}$$

The internal part can be precomputed as a set of $H_i = \sum_{\mathbf{x} \in B_i} (\nabla \mathbf{I}_1(\mathbf{x})\mathbf{J}(\mathbf{x}))^T \nabla \mathbf{I}_1(\mathbf{x})\mathbf{J}(\mathbf{x})$, $i = 1, \ldots, K$. Now the Hessian is

$$H_\delta = \sum_{i=1}^{K} \rho'_i \cdot H_i. \tag{33}$$

5 Online Demo

The accompanying online demo allows executing the algorithms explained in the previous sections (Algorithms 2, 3 and 4). It includes several synthetic sequences for which the parameters are known. The name of each sequence represents its transformation, for instance, 'zoom+rotat.+trans' stands for a zoom, rotation and translation, which can be recovered, at least, by a similarity.

Alternatively, users may upload their own sequences. The images must have the same dimensions. Optionally, the user may also upload a text file with the true parameters, which will be used for comparing with the solution obtained by the algorithm. This file is specified as [type of transformation] [parameters of the transformation]. The [type of transformation] is specified as the number of parameters of the model: translation (2), Euclidean transform (3), similarity (4), affinity (6), and homography (8). The parameters are specified in the following line of the file in the same order as in Table 2. For example, an affinity is described by the following file:

```
6
0.5 -0.5 -0.09 -0.1 -0.1 0.05
```

with $(t_x, t_y, a_{11}, a_{12}, a_{21}, a_{22}) := (0.5, -0.5, -0.09, -0.1, -0.1, 0.05)$.

Once the users have selected or uploaded the images, they may choose other parameters, such as the type of transformation, the robust error function and its parameter λ, the number of scales, the zoom factor, and the stopping criterion threshold ϵ. Additionally, they may also introduce Gaussian noise in the images by specifying the standard deviation (σ) between 0 and 100. A value of $\lambda = 0$ means that this parameter is computed automatically, as explained in Section 4.2.

After the execution, the demo shows $\mathbf{I}_1(\mathbf{x})$, $\mathbf{I}_2(\mathbf{x})$ and $\mathbf{I}_2(\mathbf{x}'(\mathbf{x}; \mathbf{p}))$. It also shows the error image $|\mathbf{I}_2(\mathbf{x}'(\mathbf{x}; \mathbf{p})) - \mathbf{I}_1(\mathbf{x})|$ and $\rho(|\mathbf{I}_2(\mathbf{x}'(\mathbf{x}; \mathbf{p})) - \mathbf{I}_1(\mathbf{x})|^2 ; \lambda)$ for $\lambda := 80$, or the value chosen by the user, and $\lambda := 5$. Then, it shows the information of the computed and ground truth transformations, as well as $RMSE$, $\tilde{d}(\mathbf{H}_{gt}\mathbf{x}_i, \mathbf{H}_e\mathbf{x}_i)$ (see next section), and the run time of the algorithm.

6 Experiments

In order to evaluate the algorithms, we used several synthetic sequences. These were generated applying a predefined parametric model to the second image, $\mathbf{I}_2(\mathbf{x})$, to obtain $\mathbf{I}_1(\mathbf{x}) := \mathbf{I}_2(\mathbf{x}'(\mathbf{x}; \mathbf{p}))$, using bicubic interpolation.

Comparisons were made using two error metrics: The root mean square error ($RMSE$) and a geometrical error, $\tilde{d}(\mathbf{H}_{gt}\mathbf{x}_i, \mathbf{H}_e\mathbf{x}_i)$. The latter results from projecting the four corners of the image with the ground truth, $\mathbf{H}_{gt}$, and the estimated transformation, $\mathbf{H}_e$, and calculating the average Euclidean distance between corresponding points. This measure is more interesting than directly comparing the transformations because it is independent of the parameter units and no normalization is necessary. These errors are given by the following expressions

$$RMSE := \sqrt{\frac{\sum |\mathbf{I}_2(\mathbf{x}'(\mathbf{x}; \mathbf{p})) - \mathbf{I}_1(\mathbf{x})|^2}{N}}, \tag{34}$$

and

$$\tilde{d}(\mathbf{H}_{gt}\mathbf{x}_i, \mathbf{H}_e\mathbf{x}_i) := \frac{\sum_{i=1}^4 d(\mathbf{H}_{gt}\mathbf{x}_i, \mathbf{H}_e\mathbf{x}_i)}{4}. \tag{35}$$

Note that the vectors are in homogeneous coordinates, so it is necessary to divide by the third component. In the experiments, we fix the parameters according to Section 4.2: $\epsilon := 10^{-3}$; the number of scales for the coarse–to–fine scheme is chosen so that the smallest images are close to but bigger than 32×32 pixels and the zoom factor is 0.5; we choose the Lorentzian error function and λ is adapted in the iterations as explained in Section 4.2.

In the first experiment, in Figure 1, we used two images of the RubberWhale sequence related by an affinity.

Figure 1: Two images of the RubberWhale sequence related by an affinity.

The ground truth and estimated affinities are

$$\mathbf{H}_{gt} = \begin{pmatrix} 0.910 & -0.100 & 0.500 \\ -0.100 & 1.050 & -0.500 \\ 0.000 & 0.000 & 1.000 \end{pmatrix}, \mathbf{H}_e = \begin{pmatrix} 0.910 & -0.100 & 0.4995 \\ -0.100 & 1.050 & -0.5005 \\ 0.000 & 0.000 & 1.000 \end{pmatrix}. \tag{36}$$

We see that the method provides very good accuracies, with $RMSE = 0.2819$ and $\tilde{d}(\mathbf{H}_{gt}\mathbf{x}_i, \mathbf{H}_e\mathbf{x}_i) = 0.0012$. In the second experiment, Figure 2, we applied a homography to the Wooden sequence.

The ground truth and estimated homographies are

$$\mathbf{H}_{gt} = \begin{pmatrix} 1.100 & 0.010 & 8.000 \\ -0.100 & 1.100 & -0.100 \\ 0.00010 & 0.00010 & 1.000 \end{pmatrix}, \mathbf{H}_e = \begin{pmatrix} 1.100 & 0.010 & 7.997 \\ -0.100 & 1.100 & -0.101 \\ 0.00010 & 0.00010 & 1.000 \end{pmatrix}, \tag{37}$$
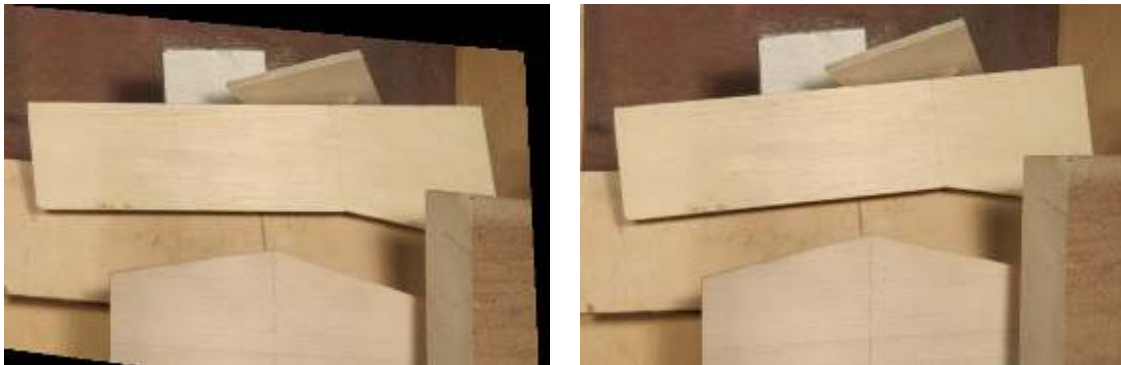
Figure 2: Two images of the Wooden sequence related by a homography.

with $RMSE = 0.279$ and $\tilde{d}(\mathbf{H}_{gt}\mathbf{x}_i, \mathbf{H}_e\mathbf{x}_i) = 0.0064$. Looking at these matrices, the most important errors seem to be related to the translation (last column of the first two rows). However, note that the influence of each parameter is different and a small variation, e.g., in the rotation, may produce larger errors. Also note that the transformation is centered at the origin of the first image.

In the third experiment in Figure 3, we used another sequence related by a homography.

Figure 3: Two images related by a homography.

The ground truth and estimated homographies are

$$\mathbf{H}_{gt} = \begin{pmatrix} 0.800 & 0.100 & -0.190 \\ -0.130 & 1.100 & -0.100 \\ 0.00010 & 0.0010 & 1.000 \end{pmatrix}, \mathbf{H}_e = \begin{pmatrix} 0.800 & 0.100 & -0.1908 \\ -0.130 & 1.100 & -0.1001 \\ 0.00010 & 0.0010 & 1.000 \end{pmatrix}. \quad (38)$$

We also obtained a very good accuracy, with $RMSE = 0.305$ and $\tilde{d}(\mathbf{H}_{gt}\mathbf{x}_i, \mathbf{H}_e\mathbf{x}_i) = 0.0025$.

6.1 Parameter Noise

Next, we analyzed the behavior of the method with respect to noise in the parameters. In the first experiment, we took the four corners of the image and added Gaussian noise of σ from 0 to 30 to each point independently. Then, we calculated the homography that relates the four points to their original positions. The second image of the Baboon sequence (Figure 4) was transformed according to this homography to get $\mathbf{I_1}$. We computed affine transformations using the error functions in Table 1 and the L^2 norm.
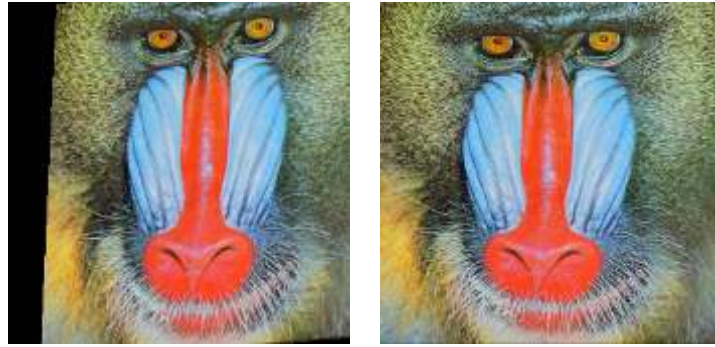
Figure 4: Baboon sequence.

We plot the average of twenty tests for each value of σ. The result is shown in Figure 5. The graphic on the left shows the results without the coarse–to–fine scheme. We observe that the error increases very fast from a small value of σ. Using four scales (graphic on the right), the solutions are much more accurate and stable. The multiscale approach also reduces the differences between the robust and L^2 functions. The error seems to increase linearly with respect to the amount of noise. As expected with Gaussian noise, the L^2 norm is the best performing strategy. The truncated quadratic and Geman & McClure robust functions present higher variations for large values of σ.
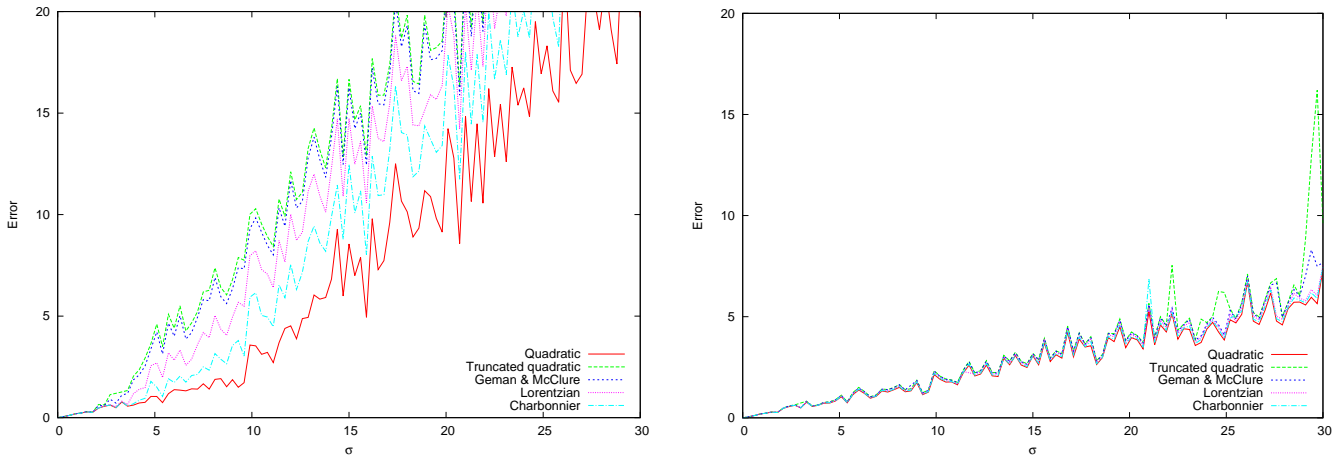

Figure 5: Parameter noise: Error evolution, $\tilde{d}(\mathbf{H}_{gt}\mathbf{x}_i, \mathbf{H}_e\mathbf{x}_i)$, with respect to Gaussian noise in the parameters. Left graphic without the coarse–to–fine approach and, right, with four scales.

In the second experiment, we added Gaussian noise to the four corners, with $\sigma := 50$. Then, we generated a set of homographies from the linear interpolation of the original, $\mathbf{x}$, and noisy points, $\widetilde{\mathbf{x}}$, as $\mathbf{x}' = (100 - \tau)/100\,\mathbf{x} + \tau/100\,\widetilde{\mathbf{x}}$. The homographies were computed from $\mathbf{x}$ to $\mathbf{x}'$. In this way, we simulated the effect of increasing noise in the parameters. The graphic on the left in Figure 6 shows the evolution of the error without the coarse–to–fine strategy. The best performing method is again the L^2 norm; the second, the Charbonnier function; and the third, the Lorentzian. The worse is the truncated quadratic. The graphic on the right shows the result using four scales, for which we observe the same benefits as in the previous experiment: higher accuracy, more stable solutions and minor differences between the error functions.

The error functions seem to present a linear evolution. Since we are using affinities for estimating the transformation between two images related by homographies, this probably means that the
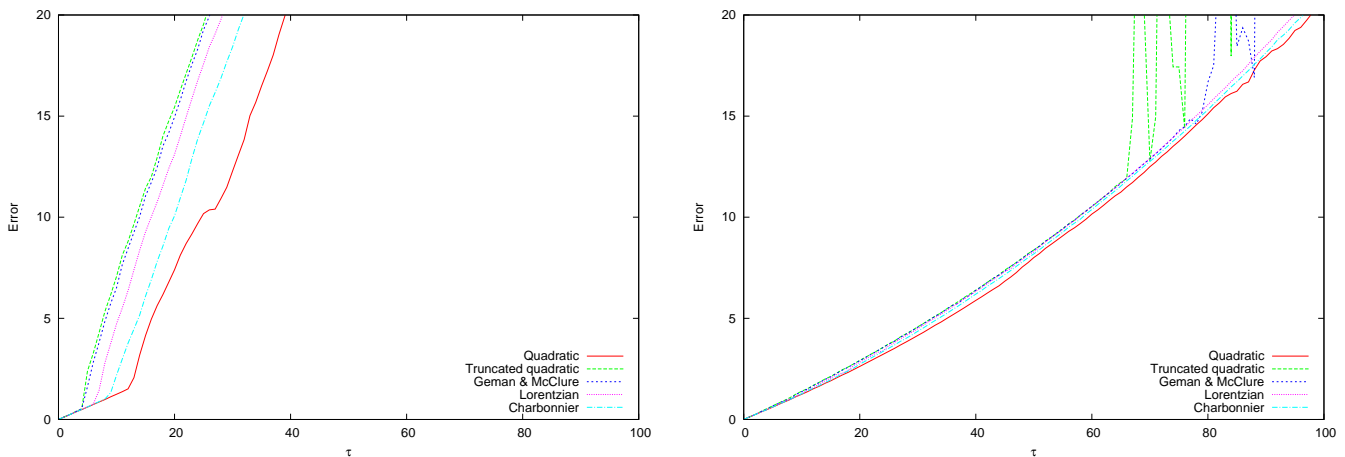
Figure 6: Linear parameter noise: Error evolution, $\tilde{d}(\mathbf{H}_{gt}\mathbf{x}_i, \mathbf{H}_e\mathbf{x}_i)$, with respect to Gaussian noise in the parameters. The transformations are calculated by linear interpolation between an initial set of points and points with Gaussian noise of $\sigma := 50$. Left graphic without the coarse–to–fine approach and, right, with four scales.

method allows to reasonably approximate a given model with another model with less parameters: Increasing the noise linearly is equivalent to go from a 'soft' homography (identity matrix) to a 'strong' homography, with a greater influence of the perspective parameters, which is farther from an affinity. The fact that we see a linear evolution of the errors means that it is proportional to the level of noise or, equivalently, to the perspective parameters. Otherwise, we should expect an exponential evolution of the error if the model fails to approximate these parameters.

In the online demo, it is easy to observe that if we choose a parametric model with less degrees of freedom, we usually obtain the transformation that allows to better align the images for that model. On the other hand, if the model has more degrees of freedom, the solutions are usually less accurate than using the correct model.

6.2 Image Noise

In the next experiment, shown in Figure 7, we studied the behavior of the algorithm with respect to noise in the images. We added Gaussian noise to the Dimetrodon sequence, with $\sigma := 20$. These images are related by a similarity.

Figure 7: Two images of the Dimetrodon sequence related by a similarity, with Gaussian noise of $\sigma := 20$. On the right, the error image given by $|\mathbf{I}_2(\mathbf{x}'(\mathbf{x}; \mathbf{p})) - \mathbf{I}_1(\mathbf{x})|$ at each pixel.

The ground truth and estimated transformations are

$$\mathbf{H}_{gt} = \begin{pmatrix} 0.8955 & -0.08985 & 47.944 \\ 0.08985 & 0.8955 & -5.9639 \\ 0.000 & 0.000 & 1.000 \end{pmatrix}, \mathbf{H}_e = \begin{pmatrix} 0.8956 & -0.08993 & 47.929 \\ 0.08993 & 0.8956 & -6.0128 \\ 0.000 & 0.000 & 1.000 \end{pmatrix}. \quad (39)$$

We obtained an $RMSE = 25.179$ and $\tilde{d}(\mathbf{H}_{gt}\mathbf{x}_i, \mathbf{H}_e\mathbf{x}_i) = 0.0489$. In the graphic depicted in Figure 8, we see the evolution of $\tilde{d}(\mathbf{H}_{gt}\mathbf{x}_i, \mathbf{H}_e\mathbf{x}_i)$ with respect to σ. In order to get a reliable evaluation, we computed the mean value of twenty estimations for each σ. This graphic shows that the method is very robust to noise in general, since the average distance to the correct points is less than one pixel for $\sigma := 100$.
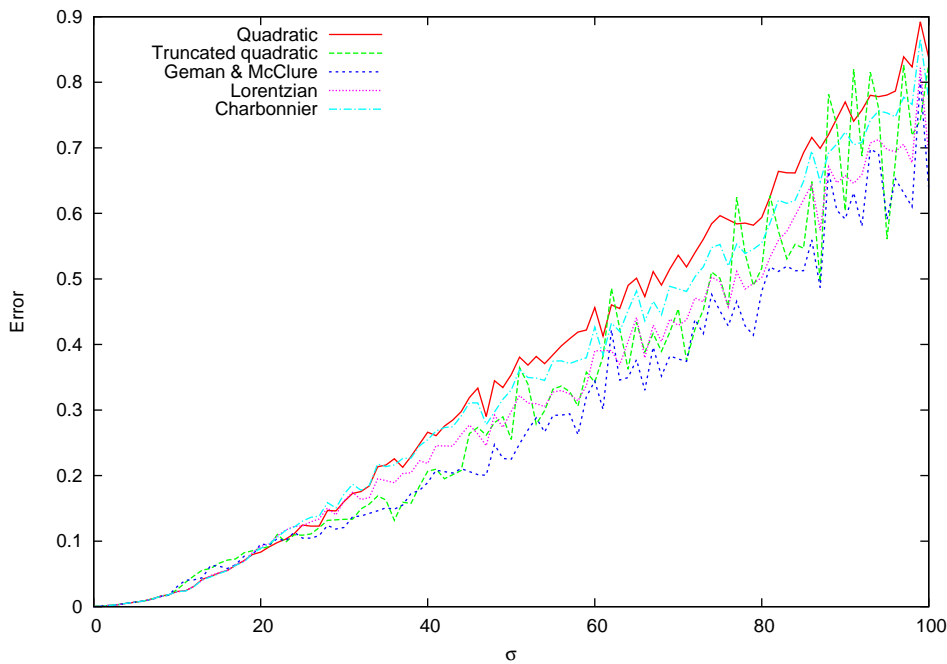

Figure 8: Error evolution, $\tilde{d}(\mathbf{H}_{gt}\mathbf{x}_i, \mathbf{H}_e\mathbf{x}_i)$, with respect to Gaussian noise of standard deviation σ. We used the Dimetrodon sequence and a similarity transform.

In this case, the quadratic function is no longer a lower bound. On the contrary, we observe that it is rather an upper bound for higher degrees of noise. The Charbonnier and Lorentzian functions consistently improve the L^2 norm. However, the truncated quadratic and Geman & McClure functions provide the best results, although they present a higher variability.

The last experiment in this section studied the behavior of the error functions with respect to λ. We added Gaussian noise of $\sigma := 50$ to the Baboon sequence (Figure 9), which is related by a homography.

The graphic in Figure 10 shows that the Charbonnier and Lorentzian functions are better than the quadratic, even for small values of λ, and are very stable. On the other hand, only for large values of λ, the truncated quadratic function outperforms the L^2 norm. For small values, the truncated quadratic probably removes too much information, producing the largest errors. The Geman & McClure function provides bad results for small λ values, similarly to the truncated quadratic, but becomes stable for larger values.
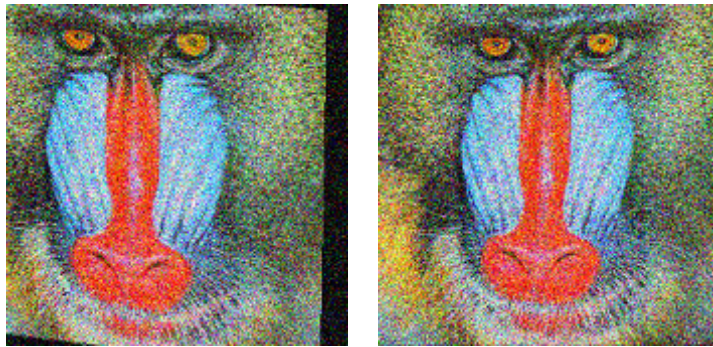
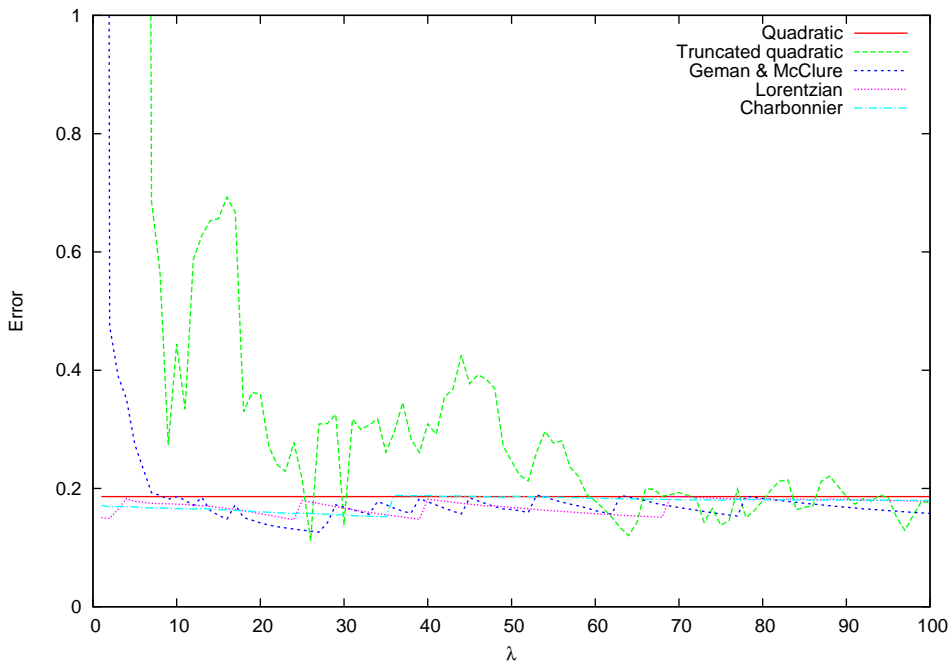Figure 9: Two images related by a homography and Gaussian noise of $\sigma := 50$.

Figure 10: Robust error parameter: Error evolution, $\tilde{d}(\mathbf{H}_{gt}\mathbf{x}_i, \mathbf{H}_e\mathbf{x}_i)$, with respect to λ.

6.3   Occlusions

Finally, we analyzed the behavior of the algorithm with respect to occlusions. In this case, we occluded half of the second image (see Figure 11) and introduced Gaussian noise of $\sigma := 5$.

The ground truth and estimated homographies are

$$\mathbf{H}_{gt} = \begin{pmatrix} 0.989 & 0.149 & -5.300 \\ -0.149 & 0.989 & 5.300 \\ 0.000 & 0.000 & 1.000 \end{pmatrix}, \mathbf{H}_e = \begin{pmatrix} 0.989 & 0.149 & -5.296 \\ -0.149 & 0.989 & 5.292 \\ 0.000 & 0.000 & 1.000 \end{pmatrix}. \tag{40}$$

We obtained $RMSE = 44.739$ and $\tilde{d}(\mathbf{H}_{gt}\mathbf{x}_i, \mathbf{H}_e\mathbf{x}_i) = 0.0151$. In the graphic depicted in Figure 12, we see the evolution of $\tilde{d}(\mathbf{H}_{gt}\mathbf{x}_i, \mathbf{H}_e\mathbf{x}_i)$ with respect to the amount of occlusion. In this case, we occluded the right part of the image with rectangles (from 1% to 100% of the image region). Again, we computed the mean value of twenty estimations for each amount of occlusion. We see that the accuracy is high even for large amounts of occlusions (up to 70%).

All the robust error functions outperform the quadratic approach, which increases linearly up

Figure 11: Two images of a sequence related by an Euclidean transform with occlusions and noise.
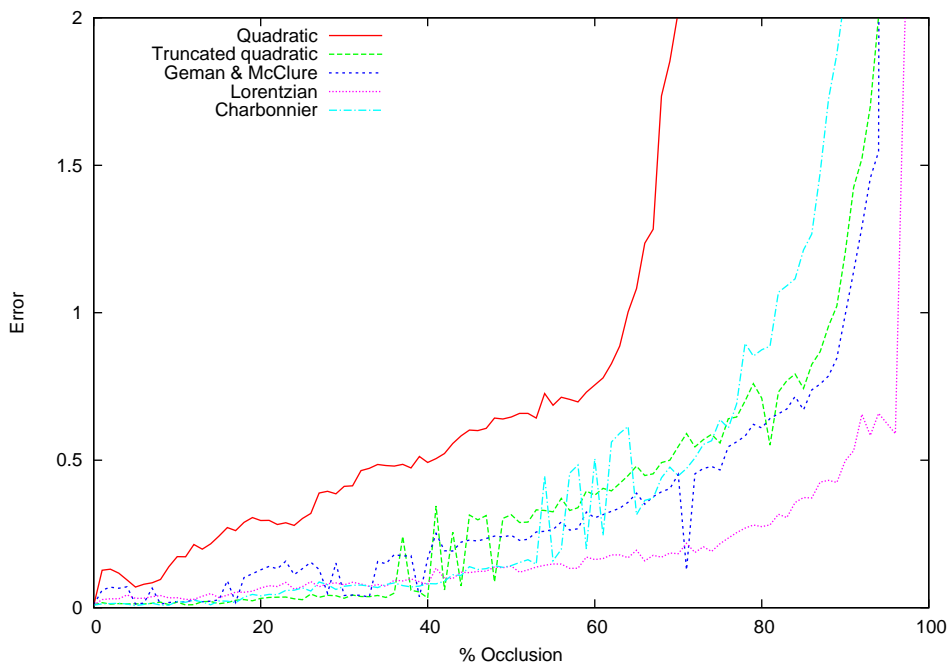

Figure 12: Occlusions: Error evolution, $\tilde{d}(\mathbf{H}_{gt}\mathbf{x}_i, \mathbf{H}_e\mathbf{x}_i)$, with respect to occlusions.

to 60% of occlusions. The best performing strategy is the truncated quadratic when the level of occlusions is less than 36%. The Charbonnier function is similar to the Lorentzian up to 50% of occlusions and then the Lorentzian is the best approach, with a good behavior even beyond 90%.

Acknowledgment

Image Credits

All images by the author (license CC-BY-SA) except:

   Middlebury benchmark database [8]

 Standard test image

 Standard test image

 Standard test image

References

[1] S. BAKER, R. GROSS, AND I. MATTHEWS, *Lucas-Kanade 20 Years On: A Unifying Framework, Part 3*, Tech. Report CMU-RI-TR-03-35, Robotics Institute, Pittsburgh, PA, November 2003.

[2] ——, *Lucas-Kanade 20 Years On: A Unifying Framework, Part 4*, Tech. Report CMU-RI-TR-04-14, Robotics Institute, Pittsburgh, PA, February 2004.

[3] S. BAKER, R. GROSS, I. MATTHEWS, AND T. ISHIKAWA, *Lucas-Kanade 20 Years On: A Unifying Framework, Part 2*, Tech. Report CMU-RI-TR-03-01, Robotics Institute, Pittsburgh, PA, February 2003.

[4] S. BAKER AND I. MATTHEWS, *Equivalence and efficiency of image alignment algorithms*, in Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2001), vol. 1, IEEE, 2001, pp. I–1090.

[5] S. BAKER AND I. MATTHEWS, *Lucas-Kanade 20 Years On: A Unifying Framework, Part 1*, Tech. Report CMU-RI-TR-02-16, Robotics Institute, Pittsburgh, PA, July 2002.

[6] S. BAKER AND I. MATTHEWS, *Lucas-Kanade 20 Years On: A Unifying Framework*, International Journal of Computer Vision, 56 (2004), pp. 221–255.

[7] S. BAKER, R. PATIL, K.M. CHEUNG, AND I. MATTHEWS, *Lucas-Kanade 20 Years On: A Unifying Framework, Part 5*, Tech. Report CMU-RI-TR-04-64, Robotics Institute, Pittsburgh, PA, November 2004.

[8] S. BAKER, D. SCHARSTEIN, J. P. LEWIS, S. ROTH, M.J. BLACK, AND R. SZELISKI, *A database and evaluation methodology for optical flow*, in International Conference on Computer Vision, 2007, pp. 1–8. http://dx.doi.org/10.1109/ICCV.2007.4408903.

[9] M.J. BLACK AND P. ANANDAN, *The robust estimation of multiple motions: Parametric and piecewise-smooth flow fields*, Computer Vision and Image Understanding, 63 (1996), pp. 75 – 104.

[10] R. I. HARTLEY AND A. ZISSERMAN, *Multiple View Geometry in Computer Vision*, Cambridge University Press, ISBN: 0521540518, second ed., 2004.

[11] P.J. Huber, *Robust Statistics*, Wiley Series in Probability and Statistics - Applied Probability and Statistics Section Series, Wiley, 2004. https://books.google.es/books?id=e62RhdqIdMkC.

[12] M. Irani and P. Anandan, *Robust multi-sensor image alignment*, in Sixth International Conference on Computer Vision, IEEE, 1998, pp. 959–966.

[13] B.D. Lucas and T. Kanade, *An iterative image registration technique with an application to stereo vision*, in Proceedings of the 7th International Joint Conference on Artificial intelligence (IJCAI), vol. 81, 1981, pp. 674–679.

[14] E. Meinhardt-Llopis, J. Sánchez, and D. Kondermann, *Horn-Schunck Optical Flow with a Multi-Scale Strategy*, Image Processing On Line, 3 (2013), pp. 151–172. http://dx.doi.org/10.5201/ipol.2013.20.

[15] J-M. Odobez and P. Bouthemy, *Robust multiresolution estimation of parametric motion models*, Journal of Visual Communication and Image Representation, 6 (1995), pp. 348–365.

[16] J. Sánchez, E. Meinhardt-Llopis, and G. Facciolo, *TV-L1 Optical Flow Estimation*, Image Processing On Line, 3 (2013), pp. 137–150. http://dx.doi.org/10.5201/ipol.2013.26.

[17] J. Sánchez, N. Monzón, and A. Salgado, *Robust Optical Flow Estimation*, Image Processing On Line, 3 (2013), pp. 252–270. http://dx.doi.org/10.5201/ipol.2013.21.

[18] R. Szeliski, *Computer vision algorithms and applications*, Springer, London; New York, 2011. http://dx.doi.org/10.1007/978-1-84882-935-0.