Lucas-Kanade 20 Years On: A Unifying Framework: Part 2

Simon Baker, Ralph Gross, Takahiro Ishikawa, and Iain Matthews

CMU-RI-TR-03-01

Abstract

Since the Lucas-Kanade algorithm was proposed in 1981, image alignment has become one of the most widely used techniques in computer vision. Applications range from optical flow, tracking and layered motion, to mosaic construction, medical image registration, and face coding. Numerous algorithms have been proposed and a wide variety of extensions have been made to the original formulation. We present an overview of image alignment, describing most of the algorithms and their extensions in a consistent framework. We concentrate on the *inverse compositional* algorithm, an efficient algorithm that we recently proposed. We examine which of the extensions to the Lucas-Kanade algorithm can be used with the inverse compositional algorithm without any significant loss of efficiency, and which require extra computation. In this paper, Part 2 in a series of papers, we cover the choice of the error function. We first consider weighted L2 norms. Afterwards we consider robust error functions.

Keywords: Image alignment, Lucas-Kanade, a unifying framework, the inverse compositional algorithm, weighted L2 norms, robust error functions.

1 Introduction

Image alignment consists of moving, and possibly deforming, a template to minimize the difference between the template and an input image. Since the first use of image alignment in the Lucas-Kanade algorithm [12], image alignment has become one of the most widely used techniques in computer vision. Other applications include tracking [4, 9], parametric motion estimation [3], mosaic construction [15], medical image registration [5], and face coding [1, 6].

The usual approach to image alignment is gradient descent [2]. A variety of other numerical algorithms have also been proposed, but gradient descent is the defacto standard. We propose a unifying framework for image alignment, describing the various algorithms and their extensions in a consistent manner. Throughout the framework we concentrate on the *inverse compositional* algorithm, an efficient algorithm that we recently proposed [1, 2]. We examine which of the extensions to the Lucas-Kanade algorithm can be applied to the inverse compositional algorithm without any significant loss of efficiency, and which extensions require additional computation. Wherever possible we provide empirical results to illustrate the various algorithms and their extensions.

In this paper, Part 2 in a series of papers, we cover the choice of the error function. The Lucas-Kanade algorithm [12] uses the Euclidean L2 norm (or sum of squared difference, SSD) to measure the degree of fit between the template and the input image. This is not the only choice. The most straightforward extension is use to a weighted L2 norm. We first show how the inverse compositional algorithm can be extended to use an arbitrary weighted L2 norm. We describe three applications of weighted L2 norms: (1) weighting the pixels with confidence values, (2) pixel selection for efficiency, and (3) allowing linear appearance variation without any lost of efficiency.

Another natural extension is to use a robust error function. In the second part of this paper we investigate how the inverse compositional algorithm can be extended to use a robust error function. We first derive the iteratively reweighted least squares (IRLS) algorithm and show that it results in a substantial loss of efficiency. Since the iteratively reweighted least squares algorithm is so slow, we describe two efficient approximations to it: (1) the H-Algorithm described in [8] and used in [9] and (2) an algorithm we recently proposed [11] that uses the spatial coherence of the outliers.

The remainder of this paper is organized as follows. We begin in Section 2 with a review of the Lucas-Kanade and inverse compositional algorithms. In Section 3 we extend the inverse compositional algorithm to use an arbitrary weighted L2 norm and describe its applications. We proceed in Section 4 to study robust error functions, to investigate several iteratively reweighted

least squares algorithms, both efficient and inefficient. We conclude in Section 5 with a summary and discussion. In future papers in this multi-part series we will cover various algorithms to allow appearance variation, and various algorithms to add priors on the warp and appearance parameters.

2 Background: Image Alignment Algorithms

2.1 Lucas-Kanade

The original image alignment algorithm was the Lucas-Kanade algorithm [12]. The goal of Lucas-Kanade is to align a template image $T(\mathbf{x})$ to an input image $I(\mathbf{x})$, where $\mathbf{x} = (x, y)^{\mathrm{T}}$ is a column vector containing the pixel coordinates. If the Lucas-Kanade algorithm is being used to track an image patch from time $t = 1$ to time $t = 2$, the template $T(\mathbf{x})$ is an extracted sub-region (a 64×64 window, maybe) of the image at $t = 1$ and $I(\mathbf{x})$ is the image at $t = 2$.

Let $\mathbf{W}(\mathbf{x};\mathbf{p})$ denote the parameterized set of allowed warps, where $\mathbf{p} = (p_1, \ldots p_n)^{\mathrm{T}}$ is a vector of parameters. The warp $\mathbf{W}(\mathbf{x};\mathbf{p})$ takes the pixel $\mathbf{x}$ in the coordinate frame of the template T and maps it to the sub-pixel location $\mathbf{W}(\mathbf{x};\mathbf{p})$ in the coordinate frame of the image I. If we are tracking a large image patch moving in 3D we may consider the set of affine warps:

$$\mathbf{W}(\mathbf{x};\mathbf{p}) = \begin{pmatrix} (1+p_1) \cdot x & + & p_3 \cdot y & + & p_5 \\ p_2 \cdot x & + & (1+p_4) \cdot y & + & p_6 \end{pmatrix} = \begin{pmatrix} 1+p_1 & p_3 & p_5 \\ p_2 & 1+p_4 & p_6 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} \quad (1)$$

where there are 6 parameters $\mathbf{p} = (p_1, p_2, p_3, p_4, p_5, p_6)^{\mathrm{T}}$ as, for example, was done in [3]. In general, the number of parameters n may be arbitrarily large and $\mathbf{W}(\mathbf{x};\mathbf{p})$ can be arbitrarily complex. One example of a complex warp is the set of piecewise affine warps used in [6, 14, 1].

2.1.1 Goal of the Lucas-Kanade Algorithm

The goal of the Lucas-Kanade algorithm is to minimize the sum of squared error between two images, the template T and the image I warped back onto the coordinate frame of the template:

$$\sum_{\mathbf{x}} [\, I(\mathbf{W}(\mathbf{x};\mathbf{p})) - T(\mathbf{x}) \,]^2 . \quad (2)$$

Warping I back to compute $I(\mathbf{W}(\mathbf{x};\mathbf{p}))$ requires interpolating the image I at the sub-pixel locations $\mathbf{W}(\mathbf{x};\mathbf{p})$. The minimization in Equation (2) is performed with respect to $\mathbf{p}$ and the sum is

performed over all of the pixels $\mathbf{x}$ in the template image $T(\mathbf{x})$. Minimizing the expression in Equation (2) is a non-linear optimization even if $\mathbf{W}(\mathbf{x};\mathbf{p})$ is linear in $\mathbf{p}$ because the pixel values $I(\mathbf{x})$ are, in general, non-linear in $\mathbf{x}$. In fact, the pixel values $I(\mathbf{x})$ are essentially un-related to the pixel coordinates $\mathbf{x}$. To optimize the expression in Equation (2), the Lucas-Kanade algorithm assumes that a current estimate of $\mathbf{p}$ is known and then iteratively solves for increments to the parameters $\Delta\mathbf{p}$; i.e. the following expression is (approximately) minimized:

$$\sum_{\mathbf{x}} [\, I(\mathbf{W}(\mathbf{x};\mathbf{p}+\Delta\mathbf{p})) - T(\mathbf{x}) \,]^2 \tag{3}$$

with respect to $\Delta\mathbf{p}$, and then the parameters are updated:

$$\mathbf{p} \;\leftarrow\; \mathbf{p} + \Delta\mathbf{p}. \tag{4}$$

These two steps are iterated until the estimates of the parameters $\mathbf{p}$ converge. Typically the test for convergence is whether some norm of the vector $\Delta\mathbf{p}$ is below a threshold ϵ; i.e. $||\Delta\mathbf{p}|| \leq \epsilon$.

2.1.2 Derivation of the Lucas-Kanade Algorithm

The Lucas-Kanade algorithm (which is a Gauss-Newton gradient descent non-linear optimization algorithm) is then derived as follows. The non-linear expression in Equation (3) is linearized by performing a first order Taylor expansion of $I(\mathbf{W}(\mathbf{x};\mathbf{p}+\Delta\mathbf{p}))$ to give:

$$\sum_{\mathbf{x}} \left[I(\mathbf{W}(\mathbf{x};\mathbf{p})) + \boldsymbol{\nabla} I \frac{\partial \mathbf{W}}{\partial \mathbf{p}} \Delta\mathbf{p} - T(\mathbf{x}) \right]^2. \tag{5}$$

In this expression, $\boldsymbol{\nabla} I = (\frac{\partial I}{\partial x}, \frac{\partial I}{\partial y})$ is the *gradient* of image I evaluated at $\mathbf{W}(\mathbf{x};\mathbf{p})$; i.e. $\boldsymbol{\nabla} I$ is computed in the coordinate frame of I and then warped back onto the coordinate frame of T using the current estimate of the warp $\mathbf{W}(\mathbf{x};\mathbf{p})$. (We follow the notational convention that the partial derivatives with respect to a column vector are laid out as a row vector. This convention has the advantage that the chain rule results in a matrix multiplication, as in Equation (5).) The term $\frac{\partial \mathbf{W}}{\partial \mathbf{p}}$ is the *Jacobian* of the warp. If $\mathbf{W}(\mathbf{x};\mathbf{p}) = (W_x(\mathbf{x};\mathbf{p}), W_y(\mathbf{x};\mathbf{p}))^{\mathrm{T}}$ then:

$$\frac{\partial \mathbf{W}}{\partial \mathbf{p}} = \begin{pmatrix} \frac{\partial W_x}{\partial p_1} & \frac{\partial W_x}{\partial p_2} & \cdots & \frac{\partial W_x}{\partial p_n} \\ \frac{\partial W_y}{\partial p_1} & \frac{\partial W_y}{\partial p_2} & \cdots & \frac{\partial W_y}{\partial p_n} \end{pmatrix}. \tag{6}$$

For example, the affine warp in Equation (1) has the Jacobian:

$$\frac{\partial \mathbf{W}}{\partial \mathbf{p}} = \begin{pmatrix} x & 0 & y & 0 & 1 & 0 \\ 0 & x & 0 & y & 0 & 1 \end{pmatrix}. \tag{7}$$

Equation (5) is a least squares problem and has a closed from solution which can be derived as follows. The partial derivative of the expression in Equation (5) with respect to $\Delta\mathbf{p}$ is:

$$\sum_{\mathbf{x}} \left[\nabla I \frac{\partial \mathbf{W}}{\partial \mathbf{p}} \right]^{\mathrm{T}} \left[I(\mathbf{W}(\mathbf{x};\mathbf{p})) + \nabla I \frac{\partial \mathbf{W}}{\partial \mathbf{p}} \Delta\mathbf{p} - T(\mathbf{x}) \right] \tag{8}$$

where we refer to $\nabla I \frac{\partial \mathbf{W}}{\partial \mathbf{p}}$ as the *steepest descent* images. Setting this expression to equal zero and solving gives the closed form solution of Equation (5) as:

$$\Delta\mathbf{p} = H^{-1} \sum_{\mathbf{x}} \left[\nabla I \frac{\partial \mathbf{W}}{\partial \mathbf{p}} \right]^{\mathrm{T}} [T(\mathbf{x}) - I(\mathbf{W}(\mathbf{x};\mathbf{p}))] \tag{9}$$

where H is the $n \times n$ (Gauss-Newton approximation to the) *Hessian* matrix:

$$H = \sum_{\mathbf{x}} \left[\nabla I \frac{\partial \mathbf{W}}{\partial \mathbf{p}} \right]^{\mathrm{T}} \left[\nabla I \frac{\partial \mathbf{W}}{\partial \mathbf{p}} \right]. \tag{10}$$

The Lucas-Kanade algorithm [12] consists of iteratively applying Equations (9) and (4). Because the gradient ∇I must be evaluated at $\mathbf{W}(\mathbf{x};\mathbf{p})$ and the Jacobian $\frac{\partial \mathbf{W}}{\partial \mathbf{p}}$ at $\mathbf{p}$, they both in depend on $\mathbf{p}$. In general, therefore, the Hessian must be recomputed in every iteration because the parameters $\mathbf{p}$ vary from iteration to iteration. The Lucas-Kanade algorithm is summarized in Figure 1.

2.1.3 Computational Cost of the Lucas-Kanade Algorithm

Assume that the number of warp parameters is n and the number of pixels in T is N. Step 1 of the Lucas-Kanade algorithm usually takes time $\mathrm{O}(n\,N)$. For each pixel $\mathbf{x}$ in T we compute $\mathbf{W}(\mathbf{x};\mathbf{p})$ and then sample I at that location. The computational cost of computing $\mathbf{W}(\mathbf{x};\mathbf{p})$ depends on $\mathbf{W}$ but for most warps the cost is $\mathrm{O}(n)$ per pixel. Step 2 takes time $\mathrm{O}(N)$. Step 3 takes the same time as Step 1, usually $\mathrm{O}(n\,N)$. Computing the Jacobian in Step 4 also depends on $\mathbf{W}$ but for most warps the cost is $\mathrm{O}(n)$ per pixel. The total cost of Step 4 is therefore $\mathrm{O}(n\,N)$. Step 5 takes time $\mathrm{O}(n\,N)$, Step 6 takes time $\mathrm{O}(n^2\,N)$, and Step 7 takes time $\mathrm{O}(n\,N)$. Step 8 takes time

The Lucas-Kanade Algorithm

Iterate:

(1) Warp I with $\mathbf{W}(\mathbf{x};\mathbf{p})$ to compute $I(\mathbf{W}(\mathbf{x};\mathbf{p}))$
(2) Compute the error image $T(\mathbf{x}) - I(\mathbf{W}(\mathbf{x};\mathbf{p}))$
(3) Warp the gradient ∇I with $\mathbf{W}(\mathbf{x};\mathbf{p})$
(4) Evaluate the Jacobian $\frac{\partial \mathbf{W}}{\partial \mathbf{p}}$ at $(\mathbf{x};\mathbf{p})$
(5) Compute the steepest descent images $\nabla I \frac{\partial \mathbf{W}}{\partial \mathbf{p}}$
(6) Compute the Hessian matrix using Equation (10)
(7) Compute $\sum_{\mathbf{x}} [\nabla I \frac{\partial \mathbf{W}}{\partial \mathbf{p}}]^{\mathrm{T}} [T(\mathbf{x}) - I(\mathbf{W}(\mathbf{x};\mathbf{p}))]$
(8) Compute $\Delta \mathbf{p}$ using Equation (9)
(9) Update the parameters $\mathbf{p} \leftarrow \mathbf{p} + \Delta \mathbf{p}$

until $||\Delta \mathbf{p}|| \leq \epsilon$

Figure 1: The Lucas-Kanade algorithm [12] consists of iteratively applying Equations (9) & (4) until the estimates of the parameters $\mathbf{p}$ converge. Typically the test for convergence is whether some norm of the vector $\Delta \mathbf{p}$ is below a user specifi ed threshold ϵ. Because the gradient ∇I must be evaluated at $\mathbf{W}(\mathbf{x};\mathbf{p})$ and the Jacobian $\frac{\partial \mathbf{W}}{\partial \mathbf{p}}$ must be evaluated at $\mathbf{p}$, all 9 steps must be repeated in every iteration of the algorithm.

Table 1: The computational cost of one iteration of the Lucas-Kanade algorithm. If n is the number of warp parameters and N is the number of pixels in the template T, the cost of each iteration is $\mathrm{O}(n^2 N + n^3)$. The most expensive step by far is Step 6, the computation of the Hessian, which alone takes time $\mathrm{O}(n^2 N)$.

Step 1	Step 2	Step 3	Step 4	Step 5	Step 6	Step 7	Step 8	Step 9	Total
$\mathrm{O}(nN)$	$\mathrm{O}(N)$	$\mathrm{O}(nN)$	$\mathrm{O}(nN)$	$\mathrm{O}(nN)$	$\mathrm{O}(n^2N)$	$\mathrm{O}(nN)$	$\mathrm{O}(n^3)$	$\mathrm{O}(n)$	$\mathrm{O}(n^2N + n^3)$

$\mathrm{O}(n^3)$ to invert the Hessian matrix and time $\mathrm{O}(n^2)$ to multiply the result by the steepest descent parameter updated computed in Step 7. Step 9 just takes time $\mathrm{O}(n)$ to increment the parameters by the updates. The total computational cost of each iteration is therefore $\mathrm{O}(n^2 N + n^3)$, the most expensive step being Step 6. See Table 1 for a summary of these computational costs.

2.2 The Inverse Compositional Algorithm

2.2.1 Goal of the Inverse Compositional Algorithm

As a number of authors have pointed out, there is a huge computational cost in re-evaluating the Hessian in every iteration of the Lucas-Kanade algorithm [9, 7, 15]. If the Hessian were constant it could be precomputed and then re-used. In [2] we proposed the inverse compositional algorithm as a way of reformulating image alignment so that the Hessian is constant and can be precomputed. Although the goal of the inverse compositional algorithm is the same as the Lucas-

Kanade algorithm; i.e. to minimize:

$$\sum_{\mathbf{x}} [\, I(\mathbf{W}(\mathbf{x};\mathbf{p})) - T(\mathbf{x}) \,]^2 \tag{11}$$

the inverse compositional algorithm iteratively minimizes:

$$\sum_{\mathbf{x}} [\, T(\mathbf{W}(\mathbf{x};\Delta\mathbf{p})) - I(\mathbf{W}(\mathbf{x};\mathbf{p})) \,]^2 \tag{12}$$

with respect to $\Delta\mathbf{p}$ (note that the roles of I and T are reversed) and then updates the warp:

$$\mathbf{W}(\mathbf{x};\mathbf{p}) \;\leftarrow\; \mathbf{W}(\mathbf{x};\mathbf{p}) \circ \mathbf{W}(\mathbf{x};\Delta\mathbf{p})^{-1}. \tag{13}$$

The expression:

$$\mathbf{W}(\mathbf{x};\mathbf{p}) \circ \mathbf{W}(\mathbf{x};\Delta\mathbf{p}) \;\equiv\; \mathbf{W}(\mathbf{W}(\mathbf{x};\Delta\mathbf{p});\mathbf{p}) \tag{14}$$

is the composition of 2 warps. For example, if $\mathbf{W}(\mathbf{x};\mathbf{p})$ is the affine warp of Equation (1) then: $\mathbf{W}(\mathbf{x};\mathbf{p}) \circ \mathbf{W}(\mathbf{x};\Delta\mathbf{p}) =$

$$\begin{pmatrix} (1+p_1)\cdot((1+\Delta p_1)\cdot x + \Delta p_3 \cdot y + \Delta p_5) + p_3 \cdot (\Delta p_2 \cdot x + (1+\Delta p_4)\cdot y + \Delta p_6) + p_5 \\ p_2 \cdot ((1+\Delta p_1)\cdot x + \Delta p_3 \cdot y + \Delta p_5) + (1+p_4)\cdot (\Delta p_2 \cdot x + (1+\Delta p_4)\cdot y + \Delta p_6) + p_6 \end{pmatrix}, \tag{15}$$

i.e. the parameters of $\mathbf{W}(\mathbf{x};\mathbf{p}) \circ \mathbf{W}(\mathbf{x};\Delta\mathbf{p})$ are:

$$\begin{pmatrix} p_1 + \Delta p_1 + p_1 \cdot \Delta p_1 + p_3 \cdot \Delta p_2 \\ p_2 + \Delta p_2 + p_2 \cdot \Delta p_1 + p_4 \cdot \Delta p_2 \\ p_3 + \Delta p_3 + p_1 \cdot \Delta p_3 + p_3 \cdot \Delta p_4 \\ p_4 + \Delta p_4 + p_2 \cdot \Delta p_3 + p_4 \cdot \Delta p_4 \\ p_5 + \Delta p_5 + p_1 \cdot \Delta p_5 + p_3 \cdot \Delta p_6 \\ p_6 + \Delta p_6 + p_2 \cdot \Delta p_5 + p_4 \cdot \Delta p_6 \end{pmatrix}, \tag{16}$$

a simple bilinear combination of the parameters of $\mathbf{W}(\mathbf{x};\mathbf{p})$ and $\mathbf{W}(\mathbf{x};\Delta\mathbf{p})$. The expression $\mathbf{W}(\mathbf{x};\Delta\mathbf{p})^{-1}$ is the inverse of $\mathbf{W}(\mathbf{x};\Delta\mathbf{p})$. The parameters of the *inverse* of the affine warp are:

$$\frac{1}{(1+p_1)\cdot(1+p_4) - p_2 \cdot p_3} \begin{pmatrix} -p_1 - p_1 \cdot p_4 + p_2 \cdot p_3 \\ -p_2 \\ -p_3 \\ -p_4 - p_1 \cdot p_4 + p_2 \cdot p_3 \\ -p_5 - p_4 \cdot p_5 + p_3 \cdot p_6 \\ -p_6 - p_1 \cdot p_6 + p_2 \cdot p_5 \end{pmatrix}. \tag{17}$$

If $(1+p_1)\cdot(1+p_4)-p_2\cdot p_3=0$, the affine warp is degenerate and not invertible. All pixels are mapped onto a straight line in I. We exclude all such affine warps from consideration.

The Lucas-Kanade algorithm iteratively applies Equations (3) and (4). The inverse compositional algorithm iteratively applies Equations (12) and (13). Perhaps somewhat surprisingly, these two algorithms can be shown to be equivalent to first order in $\Delta\mathbf{p}$. They both take the same steps as they minimize the expression in Equation (2). See [2] for the proof of equivalence.

2.2.2 Derivation of the Inverse Compositional Algorithm

Performing a first order Taylor expansion of Equation (12) gives:

$$\sum_{\mathbf{x}}\left[T(\mathbf{W}(\mathbf{x};\mathbf{0}))+\nabla T\frac{\partial\mathbf{W}}{\partial\mathbf{p}}\Delta\mathbf{p}-I(\mathbf{W}(\mathbf{x};\mathbf{p}))\right]^2. \tag{18}$$

Assuming without loss of generality that $\mathbf{W}(\mathbf{x};\mathbf{0})$ is the identity warp, the solution to this least-squares problem is:

$$\Delta\mathbf{p}\;=\;H^{-1}\sum_{\mathbf{x}}\left[\nabla T\frac{\partial\mathbf{W}}{\partial\mathbf{p}}\right]^{\mathrm{T}}\left[\,I(\mathbf{W}(\mathbf{x};\mathbf{p}))-T(\mathbf{x})\,\right] \tag{19}$$

where H is the Hessian matrix with I replaced by T:

$$H\;=\;\sum_{\mathbf{x}}\left[\nabla T\frac{\partial\mathbf{W}}{\partial\mathbf{p}}\right]^{\mathrm{T}}\left[\nabla T\frac{\partial\mathbf{W}}{\partial\mathbf{p}}\right] \tag{20}$$

and the Jacobian $\frac{\partial\mathbf{W}}{\partial\mathbf{p}}$ is evaluated at $(\mathbf{x};\mathbf{0})$. Since there is nothing in the Hessian that depends on $\mathbf{p}$, it can be pre-computed. The inverse composition algorithm is summarized in Figures 2 and 3.

2.2.3 Computational Cost of the Inverse Compositional Algorithm

The inverse compositional algorithm is far more computationally efficient than the Lucas-Kanade algorithm. See Table 2 for a summary. The most time consuming steps, Steps 3–6, can be performed once as a pre-computation. The pre-computation takes time $\mathrm{O}(n^2\,N+n^3)$. The only additional cost is inverting $\mathbf{W}(\mathbf{x};\Delta\mathbf{p})$ and composing it with $\mathbf{W}(\mathbf{x};\mathbf{p})$. These two steps typically require $\mathrm{O}(n^2)$ operations, as for the affine warp in Equations (15) and (17). Potentially these 2 steps could be fairly involved, as in [1], but the computational overhead is almost always completely negligible. Overall the cost of the inverse compositional algorithm is $\mathrm{O}(n\,N+n^2)$ per iteration rather than $\mathrm{O}(n^2\,N+n^3)$ for the Lucas-Kanade algorithm, a substantial saving.

The Inverse Compositional Algorithm

Pre-compute:

(3) Evaluate the gradient ∇T of the template $T(\mathbf{x})$
(4) Evaluate the Jacobian $\frac{\partial \mathbf{W}}{\partial \mathbf{p}}$ at $(\mathbf{x}; \mathbf{0})$
(5) Compute the steepest descent images $\nabla T \frac{\partial \mathbf{W}}{\partial \mathbf{p}}$
(6) Compute the inverse Hessian matrix using Equation (20)

Iterate:

(1) Warp I with $\mathbf{W}(\mathbf{x}; \mathbf{p})$ to compute $I(\mathbf{W}(\mathbf{x}; \mathbf{p}))$
(2) Compute the error image $I(\mathbf{W}(\mathbf{x}; \mathbf{p})) - T(\mathbf{x})$
(7) Compute $\sum_{\mathbf{x}} [\nabla T \frac{\partial \mathbf{W}}{\partial \mathbf{p}}]^{\mathrm{T}} [I(\mathbf{W}(\mathbf{x}; \mathbf{p})) - T(\mathbf{x})]$
(8) Compute $\Delta \mathbf{p}$ using Equation (19)
(9) Update the warp $\mathbf{W}(\mathbf{x}; \mathbf{p}) \leftarrow \mathbf{W}(\mathbf{x}; \mathbf{p}) \circ \mathbf{W}(\mathbf{x}; \Delta \mathbf{p})^{-1}$

until $\|\Delta \mathbf{p}\| \leq \epsilon$

Figure 2: The inverse compositional algorithm [1, 2]. All of the computationally demanding steps are performed once in a pre-computation step. The main algorithm simply consists of image warping (Step 1), image differencing (Step 2), image dot products (Step 7), multiplication with the inverse of the Hessian (Step 8), and the update to the warp (Step 9). All of these steps are effi cient $\mathrm{O}(n\,N + n^3)$.

Table 2: The computational cost of the inverse compositional algorithm. The one time pre-computation cost of computing the steepest descent images and the Hessian in Steps 3-6 is $\mathrm{O}(n^2\,N + n^3)$. After that, the cost of each iteration is $\mathrm{O}(n\,N + n^2)$ a substantial saving over the Lucas-Kanade iteration cost of $\mathrm{O}(n^2\,N + n^3)$.

Pre-Computation	Step 3	Step 4	Step 5	Step 6	Total
	$\mathrm{O}(N)$	$\mathrm{O}(n\,N)$	$\mathrm{O}(n\,N)$	$\mathrm{O}(n^2\,N + n^3)$	$\mathrm{O}(n^2\,N + n^3)$

Per Iteration	Step 1	Step 2	Step 7	Step 8	Step 9	Total
	$\mathrm{O}(n\,N)$	$\mathrm{O}(N)$	$\mathrm{O}(n\,N)$	$\mathrm{O}(n^2)$	$\mathrm{O}(n^2)$	$\mathrm{O}(n\,N + n^2)$

3 Weighted L2 Norms

All of the algorithms in [2] (there are 9 of them) aim to minimize the expression in Equation (2). This is not the only choice. Equation (2) uses the Euclidean L2 norm or "sum of square differences" (SSD) to measure the "error" between the template $T(\mathbf{x})$ and the warped input image $I(\mathbf{W}(\mathbf{x}; \mathbf{p}))$. Other functions could be used to measure the error. Perhaps the most natural generalization of the Euclidean L2 norm is to use the weighted L2 norm:

$$\sum_{\mathbf{x}} \sum_{\mathbf{y}} Q(\mathbf{x}, \mathbf{y}) \cdot [\, I(\mathbf{W}(\mathbf{x}; \mathbf{p})) - T(\mathbf{x}) \,] \cdot [\, I(\mathbf{W}(\mathbf{y}; \mathbf{p})) - T(\mathbf{y}) \,] \tag{21}$$

where $Q(\mathbf{x}, \mathbf{y})$ is an arbitrary symmetric, positive definite quadratic form. The Euclidean L2 norm is the special case where $Q(\mathbf{x}, \mathbf{y}) = \delta(\mathbf{x}, \mathbf{y})$; i.e. $Q(\mathbf{x}, \mathbf{y}) = 1$ if $\mathbf{x} = \mathbf{y}$ and $Q(\mathbf{x}, \mathbf{y}) = 0$ otherwise.

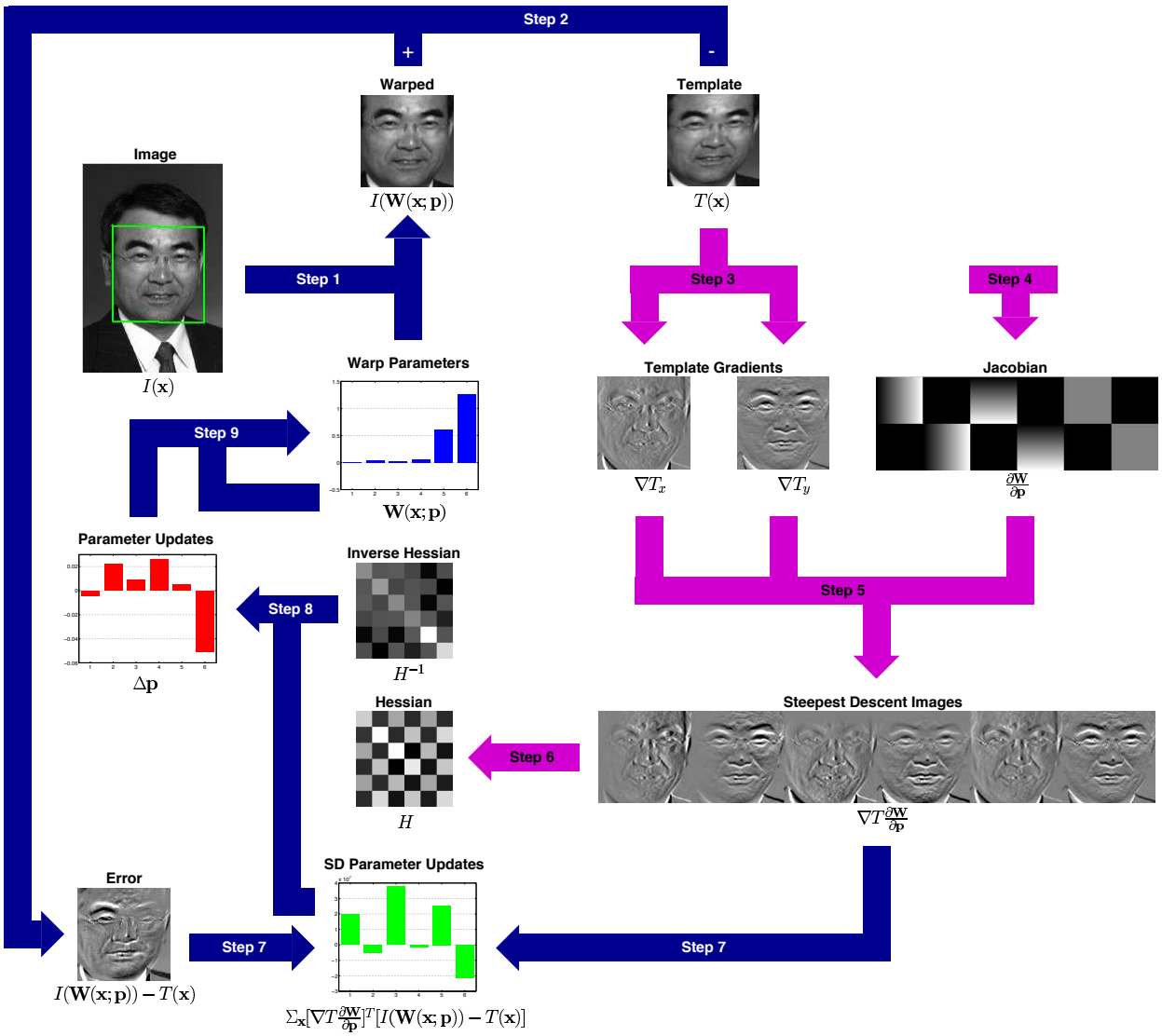


Figure 3: A schematic overview of the inverse compositional algorithm. Steps 3-6 (light-color arrows) are performed once as a pre-computation. The main algorithm simply consists of iterating: image warping (Step 1), image differencing (Step 2), image dot products (Step 7), multiplication with the inverse of the Hessian (Step 8), and the update to the warp (Step 9). All of these steps can be performed effi ciently.

3.1 Inverse Compositional Algorithm with a Weighted L2 Norm

3.1.1 Goal of the Algorithm

The inverse compositional algorithm with a weighted L2 norm minimizes the expression in Equation (21) by iteratively approximately minimizing:

$$\sum_{\mathbf{x}} \sum_{\mathbf{y}} Q(\mathbf{x},\mathbf{y}) \cdot [T(\mathbf{W}(\mathbf{x};\Delta\mathbf{p})) - I(\mathbf{W}(\mathbf{x};\mathbf{p}))] \cdot [T(\mathbf{W}(\mathbf{y};\Delta\mathbf{p})) - I(\mathbf{W}(\mathbf{y};\mathbf{p}))] \tag{22}$$

with respect to $\Delta\mathbf{p}$ and updating the warp:

$$\mathbf{W}(\mathbf{x};\mathbf{p}) \leftarrow \mathbf{W}(\mathbf{x};\mathbf{p}) \circ \mathbf{W}(\mathbf{x};\Delta\mathbf{p})^{-1}. \tag{23}$$

The proof of the first order equivalence between iterating these two steps and the *forwards additive* (Lucas-Kanade) minimization of the expression in Equation (21) is essentially the same as the proofs in Sections 3.1.5 and 3.2.5 of [2]. The complete proofs are contained in Appendix A.1.

3.1.2 Derivation of the Algorithm

Performing a first order Taylor expansion on Equation (22) gives:

$$\sum_{\mathbf{x}}\sum_{\mathbf{y}} Q(\mathbf{x},\mathbf{y}) \cdot \left[T(\mathbf{x}) + \boldsymbol{\nabla} T \frac{\partial \mathbf{W}}{\partial \mathbf{p}} \Delta\mathbf{p} - I(\mathbf{W}(\mathbf{x};\mathbf{p})) \right] \cdot \left[T(\mathbf{y}) + \boldsymbol{\nabla} T \frac{\partial \mathbf{W}}{\partial \mathbf{p}} \Delta\mathbf{p} - I(\mathbf{W}(\mathbf{y};\mathbf{p})) \right] \tag{24}$$

where as usual we have assumed that $\mathbf{W}(\mathbf{x};\mathbf{0})$ is the identity warp. Taking the partial derivatives of this expression with respect to $\Delta\mathbf{p}$ gives:

$$\begin{aligned} &\sum_{\mathbf{x}}\sum_{\mathbf{y}} Q(\mathbf{x},\mathbf{y}) \cdot \left[T(\mathbf{y}) + \boldsymbol{\nabla} T \frac{\partial \mathbf{W}}{\partial \mathbf{p}} \Delta\mathbf{p} - I(\mathbf{W}(\mathbf{y};\mathbf{p})) \right] \cdot \left[\boldsymbol{\nabla} T \frac{\partial \mathbf{W}}{\partial \mathbf{p}} \right]^{\mathrm{T}} + \\ &\sum_{\mathbf{x}}\sum_{\mathbf{y}} Q(\mathbf{x},\mathbf{y}) \cdot \left[T(\mathbf{x}) + \boldsymbol{\nabla} T \frac{\partial \mathbf{W}}{\partial \mathbf{p}} \Delta\mathbf{p} - I(\mathbf{W}(\mathbf{x};\mathbf{p})) \right] \cdot \left[\boldsymbol{\nabla} T \frac{\partial \mathbf{W}}{\partial \mathbf{p}} \right]^{\mathrm{T}}. \end{aligned} \tag{25}$$

Since $Q(\mathbf{x},\mathbf{y})$ is symmetric, this expression simplifies to:

$$2 \cdot \sum_{\mathbf{x}}\sum_{\mathbf{y}} Q(\mathbf{x},\mathbf{y}) \cdot \left[T(\mathbf{y}) + \boldsymbol{\nabla} T \frac{\partial \mathbf{W}}{\partial \mathbf{p}}(\mathbf{y};\mathbf{0}) \Delta\mathbf{p} - I(\mathbf{W}(\mathbf{y};\mathbf{p})) \right] \cdot \left[\boldsymbol{\nabla} T \frac{\partial \mathbf{W}}{\partial \mathbf{p}}(\mathbf{x};\mathbf{0}) \right]^{\mathrm{T}} \tag{26}$$

where we have made explicit the fact that the steepest descent images $\boldsymbol{\nabla} T \frac{\partial \mathbf{W}}{\partial \mathbf{p}}(\mathbf{x};\mathbf{0})$ are evaluated at $\mathbf{x}$ or $\mathbf{y}$ (appropriately) and $\mathbf{p} = \mathbf{0}$. The solution of Equation (26) is:

$$\Delta\mathbf{p} = H_Q^{-1} \sum_{\mathbf{y}} \left(\sum_{\mathbf{x}} Q(\mathbf{x},\mathbf{y}) \left[\boldsymbol{\nabla} T \frac{\partial \mathbf{W}}{\partial \mathbf{p}}(\mathbf{x};\mathbf{0}) \right]^{\mathrm{T}} \right) [I(\mathbf{W}(\mathbf{y};\mathbf{p})) - T(\mathbf{y})] \tag{27}$$

where H_Q is the weighted Hessian matrix:

$$H_Q = \sum_{\mathbf{x}}\sum_{\mathbf{y}} Q(\mathbf{x},\mathbf{y}) \left[\boldsymbol{\nabla} T \frac{\partial \mathbf{W}}{\partial \mathbf{p}}(\mathbf{x};\mathbf{0}) \right]^{\mathrm{T}} \left[\boldsymbol{\nabla} T \frac{\partial \mathbf{W}}{\partial \mathbf{p}}(\mathbf{y};\mathbf{0}) \right]. \tag{28}$$

The Inverse Compositional Algorithm with a Weighted L2 Norm

Pre-compute:

(3) Evaluate the gradient ∇T of the template $T(\mathbf{x})$
(4) Evaluate the Jacobian $\frac{\partial \mathbf{W}}{\partial \mathbf{p}}$ at $(\mathbf{x}; \mathbf{0})$
(5) Compute the weighted steepest descent images $\mathbf{SD}_Q(\mathbf{y})$ using Equation (29)
(6) Compute the inverse weighted Hessian matrix H_Q using Equation (28)

Iterate:

(1) Warp I with $\mathbf{W}(\mathbf{x}; \mathbf{p})$ to compute $I(\mathbf{W}(\mathbf{x}; \mathbf{p}))$
(2) Compute the error image $I(\mathbf{W}(\mathbf{x}; \mathbf{p})) - T(\mathbf{x})$
(7) Compute $\sum_{\mathbf{y}} \mathbf{SD}_Q(\mathbf{y})[I(\mathbf{W}(\mathbf{y}; \mathbf{p})) - T(\mathbf{x})]$
(8) Compute $\Delta \mathbf{p}$ using Equation (30)
(9) Update the warp $\mathbf{W}(\mathbf{x}; \mathbf{p}) \leftarrow \mathbf{W}(\mathbf{x}; \mathbf{p}) \circ \mathbf{W}(\mathbf{x}; \Delta \mathbf{p})^{-1}$

until $\|\Delta \mathbf{p}\| \leq \epsilon$

Figure 4: The inverse compositional algorithm with a weighted L2 norm. The computation in each iteration is exactly the same as without the weighted L2 norm. The only difference is that the weighted Hessian H_Q and the weighted steepest descent images $\mathbf{SD}_Q(\mathbf{y})$ must be used in place of their unweighted equivalents. The pre-computation is signifi cantly more costly. In particular, Step 5 takes time $\mathrm{O}(n\,N^2)$ rather than $\mathrm{O}(n\,N)$ and Step 6 takes time $\mathrm{O}(n^2\,N^2)$ rather than $\mathrm{O}(n^2\,N)$. If the quadratic form $Q(\mathbf{x}, \mathbf{y})$ is diagonal, the computational cost of these steps is reduced and is essentially the same as for the Euclidean L2 norm.

As a notational convenience, denote:

$$\mathbf{SD}_Q(\mathbf{y}) \;=\; \sum_{\mathbf{x}} Q(\mathbf{x}, \mathbf{y}) \left[\nabla T \frac{\partial \mathbf{W}}{\partial \mathbf{p}}(\mathbf{x}; \mathbf{0}) \right]^{\mathrm{T}}, \tag{29}$$

the *weighted steepest descent images*. Equation (27) then simplifies to:

$$\Delta \mathbf{p} \;=\; H_Q^{-1} \sum_{\mathbf{y}} \mathbf{SD}_Q(\mathbf{y}) \left[I(\mathbf{W}(\mathbf{y}; \mathbf{p})) - T(\mathbf{y}) \right]. \tag{30}$$

The inverse compositional algorithm with a weighted L2 norm therefore consists of iteratively applying Equations (30) and (23). The algorithm is summarized in Figure 4.

3.1.3 Computational Cost

The only difference between the unweighted and weighted inverse compositional algorithms is that the weighted steepest descent images and the weighted Hessian are used in place of the unweighted versions. The online computation per iteration is exactly the same. The weighted algorithm is equally efficient and runs in time $\mathrm{O}(n\,N + n^2)$. See Table 3 for a summary. The pre-computation of the weighted steepest descent images in Step 5 and the weighted Hessian in Step 6 is substantially

Table 3: The online computational cost of the inverse compositional algorithm with a weighted L2 norm is exactly the same as for the original algorithm. The only difference is that the weighted steepest descent images and the weighted Hessian are used in place of the unweighted versions. The pre-computation cost is substantially more however. The pre-computation of the weighted steepest descent images in Step 5 takes time $\mathrm{O}(n\,N^2)$ rather than $\mathrm{O}(n\,N)$ and the pre-computation of the weighted Hessian in Step 6 takes time $\mathrm{O}(n^2\,N^2 + n^3)$ rather than $\mathrm{O}(n^2\,N + n^3)$. The total pre-computation cost is $\mathrm{O}(n^2\,N^2 + n^3)$.

Pre-Computation	Step 3	Step 4	Step 5	Step 6	Total
	$\mathrm{O}(N)$	$\mathrm{O}(n\,N)$	$\mathrm{O}(n\,N^2)$	$\mathrm{O}(n^2\,N^2 + n^3)$	$\mathrm{O}(n^2\,N^2 + n^3)$

Per Iteration	Step 1	Step 2	Step 7	Step 8	Step 9	Total
	$\mathrm{O}(n\,N)$	$\mathrm{O}(N)$	$\mathrm{O}(n\,N)$	$\mathrm{O}(n^2)$	$\mathrm{O}(n^2)$	$\mathrm{O}(n\,N + n^2)$

more however. Step 5 takes time $\mathrm{O}(n\,N^2)$ rather than $\mathrm{O}(n\,N)$ and Step 6 takes time $\mathrm{O}(n^2\,N^2 + n^3)$ rather than $\mathrm{O}(n^2\,N + n^3)$. The total pre-computation is $\mathrm{O}(n^2\,N^2 + n^3)$ rather than $\mathrm{O}(n^2\,N + n^3)$.

3.1.4 Special Case: Diagonal Quadratic Form

A special case of the inverse compositional algorithm with weighted L2 norm occurs when the quadratic form $Q(\mathbf{x}, \mathbf{y})$ is diagonal:

$$Q(\mathbf{x}, \mathbf{y}) \;=\; Q(\mathbf{x}) \cdot \delta(\mathbf{x}, \mathbf{y}) \tag{31}$$

where we abuse the terminology slightly and denote the diagonal of $Q(\mathbf{x}, \mathbf{y})$ by $Q(\mathbf{x})$. Here, $\delta(\mathbf{x}, \mathbf{y})$ is the Kronecker delta function; $\delta(\mathbf{x}, \mathbf{y}) = 1$ if $\mathbf{x} = \mathbf{y}$ and $Q(\mathbf{x}, \mathbf{y}) = 0$ otherwise. The goal is then to minimize:

$$\sum_{\mathbf{x}} Q(\mathbf{x}) \cdot [\, I(\mathbf{W}(\mathbf{x}; \mathbf{p})) - T(\mathbf{x}) \,]^2 . \tag{32}$$

This expression is just like the original Euclidean L2 norm (SSD) formulation except that each pixel $\mathbf{x}$ in the template is weighted by $Q(\mathbf{x})$. With this special case, the weighted steepest descent images simplify to:

$$\mathbf{SD}_Q(\mathbf{x}) \;=\; Q(\mathbf{x}) \left[\nabla T \frac{\partial \mathbf{W}}{\partial \mathbf{p}}(\mathbf{x}; \mathbf{0}) \right]^{\mathrm{T}}, \tag{33}$$

and the weighted Hessian simplifies to:

$$H_Q \;=\; \sum_{\mathbf{x}} Q(\mathbf{x}) \left[\nabla T \frac{\partial \mathbf{W}}{\partial \mathbf{p}}(\mathbf{x}; \mathbf{0}) \right]^{\mathrm{T}} \left[\nabla T \frac{\partial \mathbf{W}}{\partial \mathbf{p}}(\mathbf{x}; \mathbf{0}) \right]. \tag{34}$$

Step 5 of the inverse compositional algorithm with a diagonal weighted L2 norm then takes time $\mathrm{O}(n\,N)$ and Step 6 times time $\mathrm{O}(n^2\,N)$ just like the original inverse compositional algorithm.

3.2 Application 1: Weighting the Pixels with Confidence Values

Some pixels in the template may be more reliable than others. If we can determine which pixels are the most reliable, we could weight them more strongly with a large value of $Q(\mathbf{x})$. A greater penalty will then be applied in Equation (32) if those pixels are incorrectly aligned. As a result, the overall fitting should be more accurate and greater robustness achieved. The questions of how to best measure confidence and weight the pixels are deferred to Sections 4.1 and 5.2. Until then, we make somewhat ad-hoc, but sensible, choices for $Q(\mathbf{x})$ to show that the inverse compositional algorithm with a weighted L2 norm can achieve superior performance to the original algorithm. We consider two scenarios: (1) spatially varying noise and (2) uniform (spatially invariant) noise.

3.2.1 Spatially Varying Noise

In the first scenario we assume that the noise is spatially varying. We assume that the spatially varying noise is white Gaussian noise with variance $\sigma^2(\mathbf{x})$ at pixel $\mathbf{x}$. Naturally it makes sense to give extra weight to pixels with lower noise variance. In particular, we set:

$$Q(\mathbf{x}) \;=\; \frac{1}{\sigma^2(\mathbf{x})}. \tag{35}$$

(This choice is actually optimal in the Maximum Likelihood sense, assuming that the pixels $\mathbf{x}$ are independent. See Sections 4.1 and 5.2 for more discussion.)

To validate this choice and show that the inverse compositional algorithm with a weighted L2 norm can achieve superior performance to the unweighted inverse compositional algorithm, we conducted experiments similar to those in [2]. In particular, we experimented with the image $I(\mathbf{x})$ in Figure 3. We manually selected a 100×100 pixel template $T(\mathbf{x})$ in the center of the face. We then randomly generated affine warps $\mathbf{W}(\mathbf{x};\mathbf{p})$ in the following manner. We selected 3 canonical points in the template. We used the bottom left corner $(0,0)$, the bottom right corner $(99,0)$, and the center top pixel $(49,99)$ as the canonical points. We then randomly perturbed these points with additive white Gaussian noise of a certain variance and fit for the affine warp parameters $\mathbf{p}$ that these 3 perturbed points define. We then warped I with this affine warp $\mathbf{W}(\mathbf{x};\mathbf{p})$ and run the two inverse compositional algorithms (with and without weighting) starting from the identity warp.

Since the 6 parameters in the affine warp have different units, we use the following error measure rather than the errors in the parameters. Given the current estimate of the warp, we compute

the destinations of the 3 canonical points and compare them with the correct locations. We compute the RMS error over the 3 points of the distance between their current and correct locations. (We prefer this error measure to normalizing the units so the errors in the 6 parameters are comparable.)

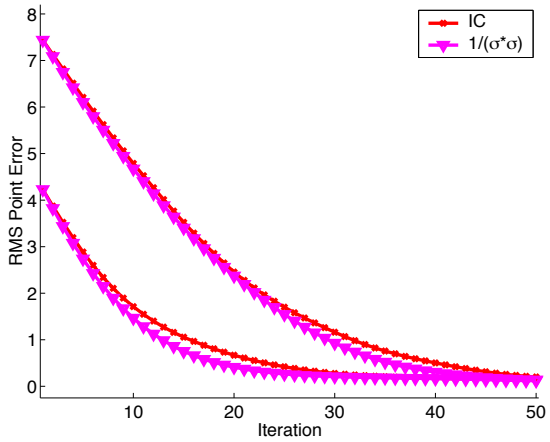
As in [2], we compute the "average rate of convergence" and the "average frequency of convergence" over a large number of randomly generated inputs (5000 to be precise). Each input consists of a different randomly generated affine warp (and template) to estimate. In addition, we add spatially varying additive white Gaussian noise to both the input image I and the template T. The average rates of convergence are plot in Figures 5(a), (c), and (e), and the average frequencies of convergence are plot in Figures 5(b), (d), and (f). In Figures 5(a) and (b) the standard deviation of the noise varies linearly across the template from $\sigma = 8.0$ to $\sigma = 24.0$, in Figures 5(c) and (d) from $\sigma = 4.0$ to $\sigma = 32.0$, and in Figures 5(e) and (f) from $\sigma = 8.0$ to $\sigma = 40.0$. The variation across the image is arranged so that the variation across the region where the template was extracted from is the same as the variation across the template itself. In Figures 5(a)–(d) the noise varies horizontally and in Figures 5(e) and (f) the noise varies vertically. In all cases, we compare the inverse compositional algorithm with a Euclidean L2 norm against the inverse compositional algorithm with a weighted L2 norm with weighting given by Equation (35).

The main thing to note in Figure 5 is that in all cases the weighted L2 norm performs better than the Euclidean L2 norm, the rate of convergence is faster and the frequency of convergence is higher. (The rate of convergence is relatively low because the amount of noise that is added is quite large.) The other thing to note in Figure 5 is that the difference between the two algorithms increases as the amount of noise increases. Although in these synthetic experiments the weighted algorithm has perfect knowledge of the distribution of the added noise, the results do clearly demonstrate that the algorithm with the weighted L2 norm can outperform the unweighted algorithm.

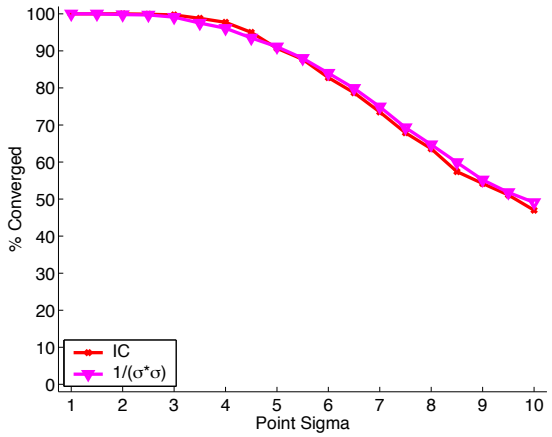
3.2.2 Uniform Noise

In the second scenario we assume that the noise is uniform (spatially invariant.) If the noise is uniform the choice in Equation (35) above is to use a uniform weighting function; i.e. the original unweighted inverse compositional algorithm. Although this choice is "optimal" (in the Maximum Likelihood sense; see Sections 4.1 and 5.2), here we experiment with the choice:

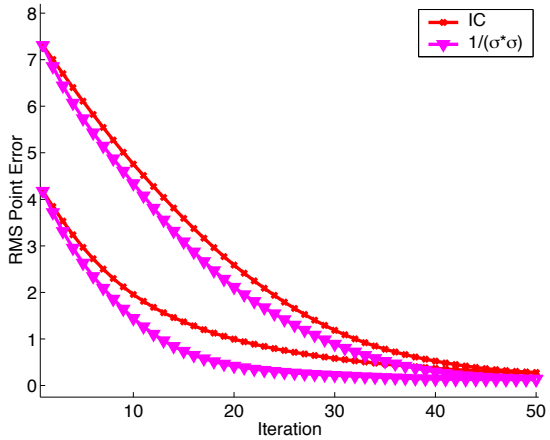
$$Q(\mathbf{x}) = |\nabla T|. \tag{36}$$



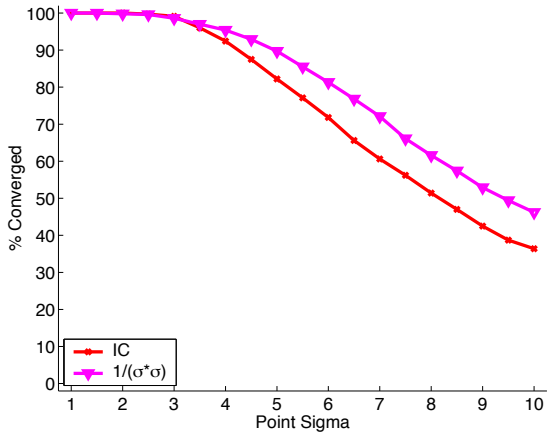
(a) Convergence Rate, σ varying: $8.0 \rightarrow 24.0$



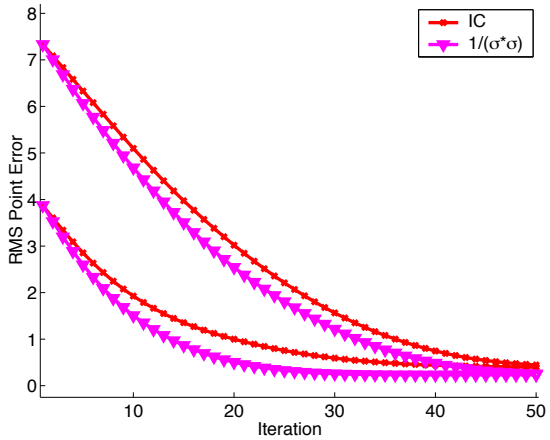
(b) Convergence Frequency, σ varying: $8.0 \rightarrow 24.0$



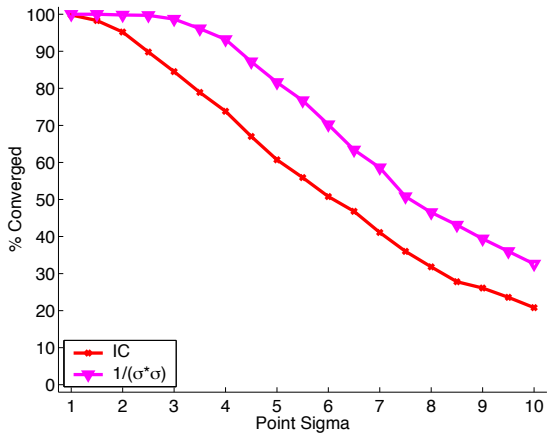
(c) Convergence Rate, σ varying: $4.0 \rightarrow 32.0$



(d) Convergence Frequency, σ varying: $4.0 \rightarrow 32.0$



(e) Convergence Rate, σ varying: $8.0 \rightarrow 40.0$



(f) Convergence Frequency, σ varying: $8.0 \rightarrow 40.0$

Figure 5: An empirical comparison between the inverse compositional algorithm and the inverse compositional algorithm with a weighted L2 norm using the weighting function in Equation (35). In all cases, the rate of convergence of the algorithm with the weighted L2 norm is faster than that of the original algorithm. The frequency of convergence is also always higher. Although we have assumed that the variance of the spatially varying noise is known to the algorithm with the weighted L2 norm, these results do clearly demonstrate that using a weighted L2 norm can, in certain circumstances, improve the performance.

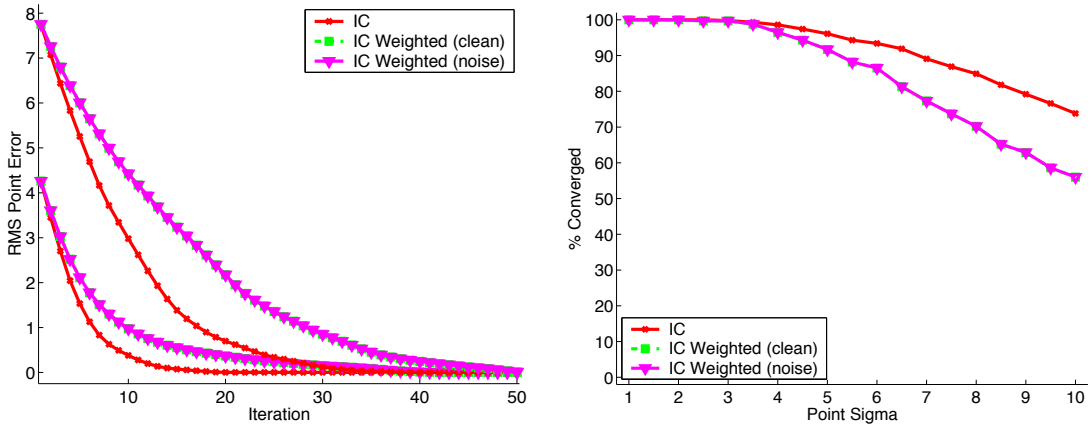
The intuitive reason for this choice is that pixels with high gradient $|\nabla T|$ should be more reliable than those with low gradient. They should therefore be given extra weight.

We repeated the experiments in Section 3.2.1 above, but now used the quadratic form in Equation (36) and added uniform, white (i.i.d.) Gaussian noise rather than spatially varying noise. Although the noise is spatially uniform, we varied the standard deviation in a variety of increments from 0 to 32 grey-levels. The results for $\sigma = 0.0, \sigma = 16.0$, and $\sigma = 32.0$ are included in Figure 6. In this figure, we compare three algorithms. The first is the original unweighted inverse compositional algorithm. The second and third algorithm are both the weighted inverse compositional algorithm, but with the weighting function in Equation (36) computed in two different ways. In the first variant "IC Weighted (clean)", ∇T is computed before any noise is added to the template T. In the second variant "IC Weighted (noisy)", ∇T is computed after the noise was added to T.

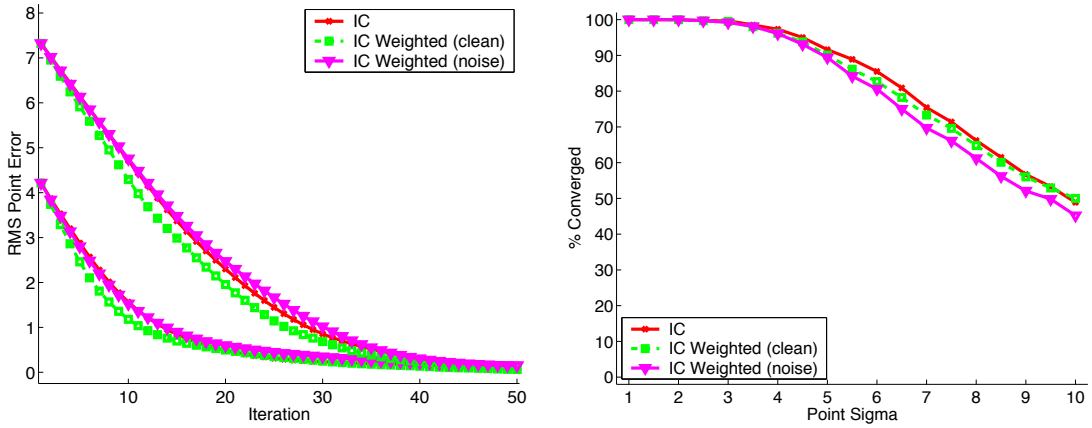
For $\sigma = 0.0$ in Figures 6(a) and (b), the unweighted inverse compositional algorithm outperforms the weighted version by a large amount. This is consistent with the optimal weighting function being given by Equation (35); i.e. the Euclidean L2 norm is optimal for uniform noise. For $\sigma = 16.0$ and $\sigma = 32.0$ in Figures 6(c)–(f), the weighted L2 norm outperforms the Euclidean L2 norm, to a greater extent for the "clean" algorithm and to a lessor extent for the "noisy" algorithm. This is surprising given that the Euclidean norm is supposed to be optimal. The reason for this counter-intuitive result is that the derivation of optimality of the Euclidean algorithm assumes that the gradient of the template ∇T is computed exactly. When there is noise, the estimate of ∇T is relatively more accurate the larger $|\nabla T|$ is. For large enough noise, this effect overcomes the theoretical optimality of the Euclidean L2 norm. Weighting with the quadratic form in Equation (36) therefore results in superior performance because extra weight is given to the pixels where the gradient of the template is estimated the most accurately.

3.3 Application 2: Pixel Selection for Efficiency

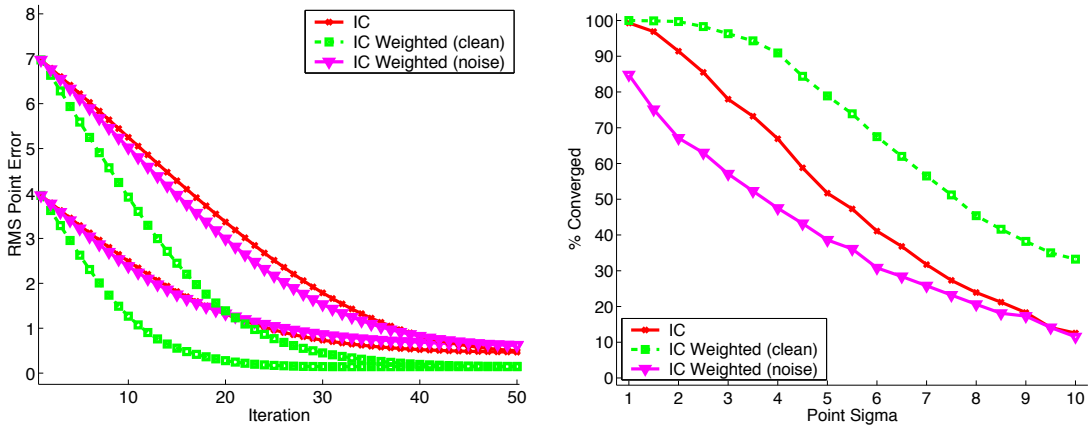
We have just shown how to give extra weight to the more reliable pixels. Conversely, less weight is given to less reliable pixels. In the extreme case we might select the most reliable pixels and just use them; i.e. give zero weight to the unreliable pixels. For example, we might just use the 1000 most reliable pixels, or the most reliable 10% of pixels. The major advantage of doing this (rather than just giving the pixels a weight depending on our confidence in them) is that the computation time can then be reduced. Steps 1, 2, and 7 in the algorithms (see Tables 2 and 3) are linear in the



(a) Convergence Rate, $\sigma = 0.0$ (b) Convergence Frequency, $\sigma = 0.0$

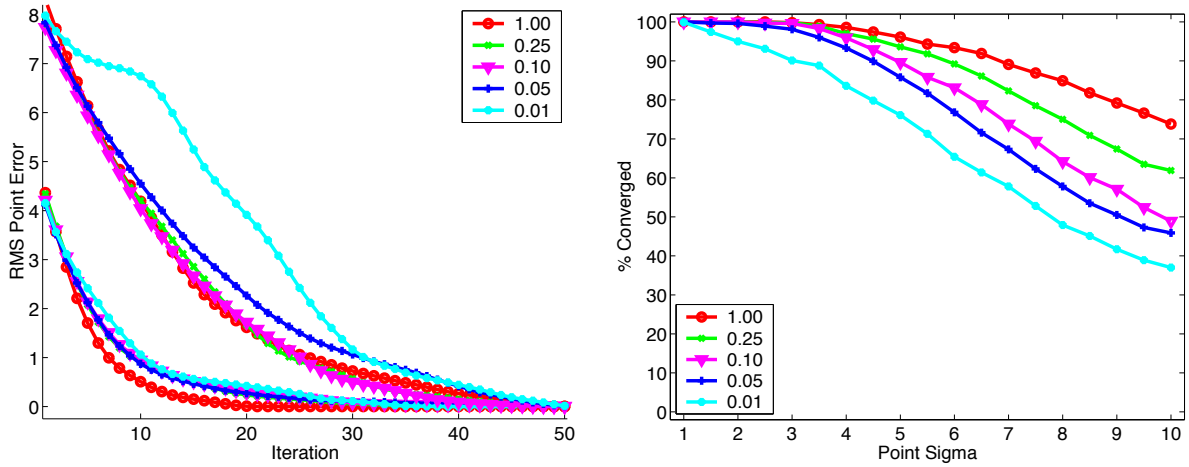


(c) Convergence Rate, $\sigma = 16.0$ (d) Convergence Frequency, $\sigma = 16.0$



(e) Convergence Rate, $\sigma = 32.0$ (f) Convergence Frequency, $\sigma = 32.0$

Figure 6: An empirical comparison between the inverse compositional algorithm and the inverse compositional algorithm with a weighted L2 norm using the weighting function in Equation (36). Two variants of the weighted algorithm are used; "clean" where the gradient of the template in the weighting function is computed before any noise is added and "noisy" where the gradient if computed after the noise is added to the template. For $\sigma = 0.0$ the results are as one would expect. The optimal Euclidean algorithm performs by far the best. When $\sigma = 16.0$ and $\sigma = 32.0$, however, the weighted algorithm out-performs the Euclidean algorithm. This surprising result occurs because the estimate of the gradient of the template used in the algorithm is relatively more reliable the greater its magnitude is. This effect is not modeled in the derivation of the optimality of the Euclidean norm when the noise is uniform.



(a) Convergence Rate (b) Convergence Frequency

Figure 7: The results of using a subset of the pixels. We plot the rate of convergence and the frequency of convergence for various different subsets of pixels: 100%, 25%, 10%, 5%, 1%. The rate of convergence is unaffected until only around 1-5% of the pixels are used. Similarly, the range of point sigmas that yield approximately 100% convergence is also relatively unaffected until only around 1-5% of the pixels are used. We conclude that we can use just 5-10% of the pixels and not signifi cantly affect the performance.

number of pixels N. If we use 10% of the pixels, these steps will all be approximately 10 times faster. Since these three steps are the most time consuming ones, the overall algorithm is speeded up dramatically. This technique was used in [7] to estimate the pan-tilt of a camera in real-time.

3.3.1 Experimental Results

An open question is how many pixels can be ignored in this way and yet not significantly affect the fitting performance. To answer this question, we ran an experiment similar to that above. We computed the average rates of convergence and average frequencies of convergence as above by randomly generating a large number of different affine warp inputs. Again, 5000 inputs were used. We added no intensity noise to the input image I or the template T in this case, however. Instead we varied the percentage of pixels used. When using n% of the pixels, we selected the best n% of pixels as measured by the magnitude of the gradient. (A more sophisticated and computationally expensive way of selecting the pixels was proposed in [7].)

The results are presented in Figure 7. The rate of convergence in Figure 7(a) shows that the speed of convergence does not decline significantly until only 1% of the pixels are used. The frequency of convergence drops steadily with the percentage of pixels used, however the range of point sigma over which the frequency of convergence is close to 100% stays steady until only 1% of the pixels are used. For the 100×100 pixel template used in our experiments, we conclude that

we can use approximately 5-10% of the pixels and not significantly affect the performance.

3.4 Application 3: Linear Appearance Variation

Another use of weighted L2 norms is for modeling linear appearance variation. The goal of the original Lucas-Kanade algorithm was to minimize the expression in Equation (2):

$$\sum_{\mathbf{x}} [\, I(\mathbf{W}(\mathbf{x};\mathbf{p})) - T(\mathbf{x}) \,]^2 . \tag{37}$$

Performing this minimization assumes that the template $T(\mathbf{x})$ appears in the input image $I(\mathbf{x})$, albeit warped by $\mathbf{W}(\mathbf{x};\mathbf{p})$. In various scenarios we may instead want to assume that:

$$T(\mathbf{x}) \; + \; \sum_{i=1}^{m} \lambda_i A_i(\mathbf{x}) \tag{38}$$

appears in the input image (warped appropriately) where $A_i, i = 1, \ldots, m$, is a set of known appearance variation images and $\lambda_i, i = 1, \ldots, m$, are a set of unknown appearance parameters. For example, if we want to allow an arbitrary change in gain and bias between the template and the input image we might set A_1 to be T and A_2 to be the "all one" image. Given the appropriate values of λ_1 and λ_2, the expression in Equation (38) can then model any possible gain and bias. More generally, the appearance images A_i can be used to model arbitrary linear illumination variation [9] or general appearance variation [4, 1]. If the expression in Equation (38) should appear (appropriately warped) in the input image $I(\mathbf{x})$, instead of Equation (37) we should minimize:

$$\sum_{\mathbf{x}} \left[I(\mathbf{W}(\mathbf{x};\mathbf{p})) - T(\mathbf{x}) - \sum_{i=1}^{m} \lambda_i A_i(\mathbf{x}) \right]^2 \tag{39}$$

simultaneously with respect to the warp and appearance parameters, $\mathbf{p}$ and $\boldsymbol{\lambda} = (\lambda_1, \ldots, \lambda_m)^{\mathrm{T}}$.

3.4.1 Derivation of the Algorithm

If we treat the images as vectors we can rewrite Equation (39) as:

$$\sum_{\mathbf{x}} \left[I(\mathbf{W}(\mathbf{x};\mathbf{p})) - T(\mathbf{x}) - \sum_{i=1}^{m} \lambda_i A_i(\mathbf{x}) \right]^2 = \left\| I(\mathbf{W}(\mathbf{x};\mathbf{p})) - T(\mathbf{x}) - \sum_{i=1}^{m} \lambda_i A_i(\mathbf{x}) \right\|^2 \tag{40}$$

where $\|\cdot\|$ is the unweighted (Euclidean) L2 norm. This expression must be minimized simultaneously with respect to $\mathbf{p}$ and $\boldsymbol{\lambda}$. If we denote the linear subspace spanned by a collection of vectors A_i by $\mathrm{span}(A_i)$ and its orthogonal complement by $\mathrm{span}(A_i)^\perp$ Equation (40) can be rewritten as:

$$\left\| I(\mathbf{W}(\mathbf{x};\mathbf{p})) - T(\mathbf{x}) - \sum_{i=1}^{m} \lambda_i A_i(\mathbf{x}) \right\|^2_{\mathrm{span}(A_i)} + \left\| I(\mathbf{W}(\mathbf{x};\mathbf{p})) - T(\mathbf{x}) - \sum_{i=1}^{m} \lambda_i A_i(\mathbf{x}) \right\|^2_{\mathrm{span}(A_i)^\perp} \tag{41}$$

where $\|\cdot\|_L$ denotes the Euclidean L2 norm of a vector projected into the linear subspace L. The second of these two terms immediately simplifies. Since the norm in the second term only considers the component of the vector in the orthogonal complement of $\mathrm{span}(A_i)$, any component in $\mathrm{span}(A_i)$ can be dropped. We therefore wish to minimize:

$$\left\| I(\mathbf{W}(\mathbf{x};\mathbf{p})) - T(\mathbf{x}) - \sum_{i=1}^{m} \lambda_i A_i(\mathbf{x}) \right\|^2_{\mathrm{span}(A_i)} + \left\| I(\mathbf{W}(\mathbf{x};\mathbf{p})) - T(\mathbf{x}) \right\|^2_{\mathrm{span}(A_i)^\perp} . \tag{42}$$

The second of these two terms does not depend upon $\boldsymbol{\lambda}$. For any $\mathbf{p}$, the minimum value of the first term is always 0. Therefore, the simultaneous minimum over both $\mathbf{p}$ and $\boldsymbol{\lambda}$ can be found sequentially by first minimizing the second term with respect to $\mathbf{p}$ alone, and then treating the optimal value of $\mathbf{p}$ as a constant to minimize the first term with respect to $\boldsymbol{\lambda}$. Assuming that the appearance variation vectors A_i are orthonormal (if they are not they can easily be orthonormalized using Gramm-Schmidt) the minimization of the first term has the closed-form solution:

$$\lambda_i \;=\; \sum_{\mathbf{x}} A_i(\mathbf{x}) \cdot [I(\mathbf{W}(\mathbf{x};\mathbf{p})) - T(\mathbf{x})] . \tag{43}$$

The only difference between minimizing the second term in Equation (42) and the original goal of the Lucas-Kanade algorithm (see Equation (37)) is that we need to work in the linear subspace $\mathrm{span}(A_i)^\perp$. Working in this subspace can be achieved by using a weighted L2 norm with:

$$Q(\mathbf{x},\mathbf{y}) \;=\; \delta(\mathbf{x},\mathbf{y}) - \sum_{i=1}^{m} [A_i(\mathbf{x}) \cdot A_i(\mathbf{y})] \tag{44}$$

(assuming the vectors A_i are orthonormal.) We can therefore use the inverse compositional algorithm with this weighted L2 norm to minimize the second term in Equation (42). The weighted steepest descent images (see Equation (29)) are:

$$\mathbf{SD}_Q(\mathbf{x}) \;=\; \sum_{\mathbf{y}} \left[\delta(\mathbf{x},\mathbf{y}) - \sum_{i=1}^{m} (A_i(\mathbf{x}) \cdot A_i(\mathbf{y})) \right] \left[\boldsymbol{\nabla} T \frac{\partial \mathbf{W}}{\partial \mathbf{p}}(\mathbf{y};\mathbf{0}) \right]^{\mathrm{T}} \tag{45}$$

and so can be computed:

$$\mathbf{SD}_Q(\mathbf{x}) \;=\; \boldsymbol{\nabla} T \frac{\partial \mathbf{W}}{\partial \mathbf{p}}(\mathbf{x};\mathbf{0}) \;-\; \sum_{i=1}^{m} \left[\sum_{\mathbf{y}} A_i(\mathbf{y}) \cdot \boldsymbol{\nabla} T \frac{\partial \mathbf{W}}{\partial \mathbf{p}}(\mathbf{y};\mathbf{0}) \right] A_i(\mathbf{x}) \tag{46}$$

i.e. the unweighted steepest descent images $\boldsymbol{\nabla} T \frac{\partial \mathbf{W}}{\partial \mathbf{p}}(\mathbf{x};\mathbf{0})$ are projected into $\text{span}(A_i)^\perp$ by removing the component in the direction of A_i, for $i = 1, \ldots, m$ in turn. The weighted Hessian matrix:

$$H_Q \;=\; \sum_{\mathbf{x}} \sum_{\mathbf{y}} Q(\mathbf{x},\mathbf{y}) \left[\boldsymbol{\nabla} T \frac{\partial \mathbf{W}}{\partial \mathbf{p}}(\mathbf{x};\mathbf{0}) \right]^{\mathrm{T}} \left[\boldsymbol{\nabla} T \frac{\partial \mathbf{W}}{\partial \mathbf{p}}(\mathbf{y};\mathbf{0}) \right] \tag{47}$$

can then also be computed as:

$$H_Q \;=\; \sum_{\mathbf{x}} \mathbf{SD}_Q(\mathbf{x})\, \mathbf{SD}_Q^{\mathrm{T}}(\mathbf{x}) \tag{48}$$

rather than using Equation (28) because the inner product of two vectors projected into a linear subspace is the same as if just one of the two is projected into the linear subspace.

In summary, minimizing the expression in Equation (39) with respect to $\mathbf{p}$ and $\boldsymbol{\lambda}$ can be performed by first minimizing the second term in Equation (42) with respect to $\mathbf{p}$ using the inverse compositional algorithm with the quadratic form in Equation (44). The only changes needed to the algorithm are: (1) to use the weighted steepest descent images in Equation (46) and (2) to use the weighted Hessian in Equation (48). Once the inverse compositional algorithm has converged, the optimal value of $\boldsymbol{\lambda}$ can be computed using Equation (43) where $\mathbf{p}$ are the optimal warp parameters.

3.4.2 Discussion

Although the algorithm described above has been used before by several authors [9, 1], it is not the only choice for modeling linear appearance variation. A variety of other algorithms are possible. There are also various caveats that need to be considered when using this algorithm, most notably the choice of the step size and the theoretical incorrectness of the algorithm when combined with a robust error function. In Part 3 of this series of papers we will present a more complete treatment of appearance variation. The reader is advised to read that paper before using the algorithm above, and in particular to look at the empirical comparison with the other algorithms. Since the question of appearance variation is so involved, we do not present any experimental results here.

4 Robust Error Functions

Another generalization of the expression in Equation (2) is to use a *robust error function* instead of the Euclidean L2 norm. The goal is then to minimize:

$$\sum_{\mathbf{x}} \rho\left([I(\mathbf{W}(\mathbf{x};\mathbf{p})) - T(\mathbf{x})]; \boldsymbol{\sigma} \right) \tag{49}$$

with respect to the warp parameters $\mathbf{p}$ where $\rho(t;\boldsymbol{\sigma})$ is a *robust error function* [10] and $\boldsymbol{\sigma} = (\sigma_1, \sigma_2, \ldots, \sigma_s)^{\mathbf{T}}$ is a vector of *scale parameters*. For now we treat the scale parameters as known constants. In Section 4.5.2 we briefly describe how our algorithms can be extended to estimate the scale parameters from sample estimates of the error or noise $I(\mathbf{W}(\mathbf{x};\mathbf{p})) - T(\mathbf{x})$. Until Section 4.5.2, we drop the scale parameters from $\rho(t;\boldsymbol{\sigma})$ and denote the robust function $\rho(t)$.

4.1 Choosing the Robust Function

The only formal requirements on $\rho(t)$ are: (1) that $\rho(t) > 0$ for all t, (2) that $\rho(t)$ is monotonically increasing for $t \geq 0$, (3) that $\rho(t)$ is monotonically decreasing for $t \leq 0$, and (4) that $\rho(t)$ is piecewise differentiable. When $\rho(t) = t^2$ the expression in Equation (49) reduces to the Euclidean L2 norm of Equation (2). For $\rho(t)$ to be a "robust" function, it should increase asymptotically "less fast" than t^2 for large $|t|$. Outliers (pixels $\mathbf{x}$ with large $|I(\mathbf{W}(\mathbf{x};\mathbf{p})) - T(\mathbf{x})|$) will therefore be penalized less in Equation (49) than in Equation (2).

A variety of choices of $\rho(t)$ have been used in the literature. Black and Jepson [4] and Sawhney and Ayer [13] both used the *Geman-McLure* function:

$$\rho(t) = \frac{t^2}{\sigma_1^2 + t^2}. \tag{50}$$

On the other hand, Hager and Belhumeur [9] used the *Huber* function:

$$\rho(t) = \begin{cases} \frac{1}{2}t^2 & \text{if } |t| \leq \sigma_1 \\ \sigma_1|t| - \frac{1}{2}\sigma_1^2 & \text{otherwise.} \end{cases} \tag{51}$$

Since there are a wide variety of choices for $\rho(t)$ a natural question is how to choose the best one. The choice of $\rho(t)$ is outside the scope of this paper, however if we assume that the pixels $\mathbf{x}$ are independent, the "theoretically correct" answer is that the robust function should be:

$$\rho(t) \propto -\log \mathbf{P}[I(\mathbf{W}(\mathbf{x};\mathbf{p})) - T(\mathbf{x}) = t] \tag{52}$$

the negative log probability that the error (or noise) between the template and the warped input images is t. Of course this does not really answer the question of how to choose $\rho(t)$, but just rephrases it as a question of what probability distribution best models the noise in real images. In our synthetic experiments we know the distribution and so can set $\rho(t)$ appropriately. The question of how the algorithms degrade if the wrong $\rho(t)$ is used is outside the scope of this paper.

4.2 Newton vs. Gauss-Newton Formulations

So far we have formulated the robust image alignment problem in the same way that most authors have done in the vision literature; see for example [13, 4, 9]. In this formulation the robust function $\rho(t)$ is given the argument $I(\mathbf{W}(\mathbf{x};\mathbf{p})) - T(\mathbf{x})$. Since there is no "squared" term, the derivation of any algorithm requires a second order Taylor expansion to obtain a quadratic form in $\Delta\mathbf{p}$. Such a derivation would therefore be like the derivation of the Newton algorithm in Section 4.2 of [2]. So that we only need to apply a first order Taylor expansion and thereby obtain a Gauss-Newton derivation, we replace the robust function $\rho(t)$ with:

$$\varrho(t) \;=\; \rho(\sqrt{t}). \tag{53}$$

Substituting this expression into Equation (49) means that we should aim to minimize:

$$\sum_{\mathbf{x}} \varrho\left(\left[I(\mathbf{W}(\mathbf{x};\mathbf{p})) - T(\mathbf{x}) \right]^2 \right). \tag{54}$$

The change of variables in Equation (53) can be performed for any robust function $\rho(t)$ with one condition, $\rho(y)$ must be symmetric:

$$\rho(t) \;=\; -\rho(-t). \tag{55}$$

This condition is fairly weak. Almost all robust functions that have ever been used are symmetric, including the Geman-McLure and Huber functions mentioned above. With the change of variables the Geman-McLure function becomes:

$$\varrho(t) \;=\; \frac{t}{\sigma_1^2 + t} \tag{56}$$

and the Huber function becomes:

$$\varrho(t) \;=\; \begin{cases} \frac{1}{2}t & \text{if } 0 \le t \le \sigma_1^2 \\ \sigma_1\sqrt{t} - \frac{1}{2}\sigma_1^2 & \text{if } t > \sigma_1^2. \end{cases} \tag{57}$$

4.3 Inverse Compositional Iteratively Reweighted Least Squares

4.3.1 Goal of the Algorithm

The inverse compositional algorithm with a robust function $\varrho(t)$ minimizes the expression in Equation (54) by iteratively approximately minimizing:

$$\sum_{\mathbf{x}} \varrho\left([T(\mathbf{W}(\mathbf{x};\Delta\mathbf{p})) - I(\mathbf{W}(\mathbf{x};\mathbf{p}))]^2 \right) \tag{58}$$

with respect to $\Delta\mathbf{p}$ and updating the warp:

$$\mathbf{W}(\mathbf{x};\mathbf{p}) \leftarrow \mathbf{W}(\mathbf{x};\mathbf{p}) \circ \mathbf{W}(\mathbf{x};\Delta\mathbf{p})^{-1}. \tag{59}$$

The proof of the first order equivalence between iterating these two steps and the *forwards additive* (Lucas-Kanade) minimization of the expression in Equation (54) is essentially the same as the proofs in Sections 3.1.5 and 3.2.5 of [2]. The complete proofs are contained in Appendix A.2.

4.3.2 Derivation of the Algorithm

Performing a first order Taylor expansion on $T(\mathbf{W}(\mathbf{x};\Delta\mathbf{p}))$ in Equation (58) gives:

$$\sum_{\mathbf{x}} \varrho\left(\left[T(\mathbf{x}) + \nabla T \frac{\partial \mathbf{W}}{\partial \mathbf{p}} \Delta\mathbf{p} - I(\mathbf{W}(\mathbf{x};\mathbf{p})) \right]^2 \right) \tag{60}$$

where again we have assumed that $\mathbf{W}(\mathbf{x};\mathbf{0})$ is the identity warp. Expanding gives:

$$\sum_{\mathbf{x}} \varrho\left(E(\mathbf{x})^2 + E(\mathbf{x}) \nabla T \frac{\partial \mathbf{W}}{\partial \mathbf{p}} \Delta\mathbf{p} + \Delta\mathbf{p}^{\mathrm{T}} \left[\nabla T \frac{\partial \mathbf{W}}{\partial \mathbf{p}} \right]^{\mathrm{T}} \left[\nabla T \frac{\partial \mathbf{W}}{\partial \mathbf{p}} \right] \Delta\mathbf{p} \right) \tag{61}$$

where $E(\mathbf{x}) = T(\mathbf{x}) - I(\mathbf{W}(\mathbf{x};\mathbf{p}))$. Performing a first order Taylor expansion gives:

$$\sum_{\mathbf{x}} \left(\varrho\left(E(\mathbf{x})^2 \right) + \varrho'\left(E(\mathbf{x})^2 \right) \left[E(\mathbf{x}) \nabla T \frac{\partial \mathbf{W}}{\partial \mathbf{p}} \Delta\mathbf{p} + \Delta\mathbf{p}^{\mathrm{T}} \left[\nabla T \frac{\partial \mathbf{W}}{\partial \mathbf{p}} \right]^{\mathrm{T}} \left[\nabla T \frac{\partial \mathbf{W}}{\partial \mathbf{p}} \right] \Delta\mathbf{p} \right] \right). \tag{62}$$

The minimum of this quadratic form is attained at:

$$\begin{aligned} \Delta\mathbf{p} &= -H_{\varrho}^{-1} \sum_{\mathbf{x}} \varrho'\left(E(\mathbf{x})^2 \right) \nabla T \frac{\partial \mathbf{W}}{\partial \mathbf{p}} E(\mathbf{x}) \\ &= H_{\varrho}^{-1} \sum_{\mathbf{x}} \varrho'\left([I(\mathbf{W}(\mathbf{x};\mathbf{p})) - T(\mathbf{x})]^2 \right) \nabla T \frac{\partial \mathbf{W}}{\partial \mathbf{p}} [I(\mathbf{W}(\mathbf{x};\mathbf{p})) - T(\mathbf{x})] \end{aligned} \tag{63}$$

Inverse Compositional Iteratively Reweighted Least Squares

Pre-compute:

(3) Evaluate the gradient ∇T of the template $T(\mathbf{x})$
(4) Evaluate the Jacobian $\frac{\partial \mathbf{W}}{\partial \mathbf{p}}$ at $(\mathbf{x}; \mathbf{0})$
(5) Compute the steepest descent images $\nabla T \frac{\partial \mathbf{W}}{\partial \mathbf{p}}$

Iterate:

(1) Warp I with $\mathbf{W}(\mathbf{x}; \mathbf{p})$ to compute $I(\mathbf{W}(\mathbf{x}; \mathbf{p}))$
(2) Compute the error image $I(\mathbf{W}(\mathbf{x}; \mathbf{p})) - T(\mathbf{x})$
(6) Compute the Hessian matrix H_ϱ using Equation (64)
(7) Compute $\sum_{\mathbf{x}} \varrho' \left([I(\mathbf{W}(\mathbf{x}; \mathbf{p})) - T(\mathbf{x})]^2 \right) [\nabla T \frac{\partial \mathbf{W}}{\partial \mathbf{p}}]^{\mathrm{T}} [I(\mathbf{W}(\mathbf{x}; \mathbf{p})) - T(\mathbf{x})]$
(8) Compute $\Delta \mathbf{p}$ using Equation (63)
(9) Update the warp $\mathbf{W}(\mathbf{x}; \mathbf{p}) \leftarrow \mathbf{W}(\mathbf{x}; \mathbf{p}) \circ \mathbf{W}(\mathbf{x}; \Delta \mathbf{p})^{-1}$

until $\|\Delta \mathbf{p}\| \leq \epsilon$

Figure 8: The inverse compositional iteratively reweighted least squares algorithm consists of iteratively applying Equations (63) & (59). Because the Hessian H_ϱ depends on the warp parameters $\mathbf{p}$ it must be re-computed in every iteration. Since this step is the slowest one in the algorithm (see Table 4), the naive implementation of this algorithm is almost as slow as the original Lucas-Kanade algorithm.

where:

$$H_\varrho \;=\; \sum_{\mathbf{x}} \varrho' \left([I(\mathbf{W}(\mathbf{x}; \mathbf{p})) - T(\mathbf{x})]^2 \right) \left[\nabla T \frac{\partial \mathbf{W}}{\partial \mathbf{p}} \right]^{\mathrm{T}} \left[\nabla T \frac{\partial \mathbf{W}}{\partial \mathbf{p}} \right] \tag{64}$$

is the Hessian matrix. Comparing Equations (63) and (64) with those in Section 3.1.4 we see that each step of the inverse compositional algorithm with robust error function $\varrho(t)$ is the same as a step of the inverse compositional algorithm with diagonal weighted L2 norm with quadratic form:

$$Q(\mathbf{x}) \;=\; \varrho' \left([I(\mathbf{W}(\mathbf{x}; \mathbf{p})) - T(\mathbf{x})]^2 \right). \tag{65}$$

This diagonal quadratic form (or weighting function) depends on $\mathbf{p}$ through $I(\mathbf{W}(\mathbf{x}; \mathbf{p}))$ and so must be updated or reweighted from iteration to iteration. This algorithm is therefore referred to as the *inverse compositional iteratively reweighted least squares algorithm* and can be use to perform image alignment with an arbitrary (symmetric) robust function $\varrho(t)$. A summary of the inverse compositional iteratively reweighted least squares algorithm is included in Figure 8.

4.3.3 Computational Cost

Since the cost of evaluating ϱ' is constant (does not depend on n or N), the computational cost of the inverse compositional iteratively reweighted least squares algorithm is as summarized in Table 4. The cost of each iteration is asymptotically as slow as the Lucas-Kanade algorithm.

Table 4: The computational cost of the inverse compositional iteratively reweighted least squares algorithm. The one time pre-computation cost of computing the steepest descent images in Steps 3-5 is $O(n\,N)$. The cost of each iteration is $O(n^2\,N + n^3)$ which is asymptotically as slow as the Lucas-Kanade algorithm.

Pre-Computation	Step 3	Step 4	Step 5	Total
	$O(N)$	$O(n\,N)$	$O(n\,N)$	$O(n\,N)$

Per Iteration	Step 1	Step 2	Step 6	Step 7	Step 8	Step 9	Total
	$O(n\,N)$	$O(N)$	$O(n^2\,N)$	$O(n\,N)$	$O(n^3)$	$O(n^2)$	$O(n^2\,N + n^3)$

4.4 Efficient Approximations

Although the iteratively reweighted least squares algorithm is inefficient, there are several approximations to it that are efficient, yet perform almost as well. We now describe two examples.

4.4.1 The *H*-Algorithm

Dutter and Huber describe several different robust least squares algorithms in [8]. One of these algorithms, the *W*-Algorithm, is equivalent to our inverse compositional iteratively reweighted least squares algorithm. The only differences are: (1) we use the inverse compositional formulation and so the updates to the vectors of parameters are different, and (2) the *W*-Algorithm includes scale estimation steps. Section 4.5.2 describes how scale estimation can be added to our algorithm.

Dutter and Huber also described the *H*-Algorithm, a variant of the *W*-Algorithm. The only significant difference between these two algorithms is that the *H*-Algorithm effectively uses the unweighted Hessian of Equation (20) rather than the weighted Hessian of Equation (64). As in the inverse compositional algorithm, the unweighted Hessian can be precomputed. Step 6, the one slow step in the iteratively reweighted least squares algorithm, can be moved to pre-computation. The *H*-Algorithm is asymptotically as efficient as the original inverse compositional algorithm.

The *H*-Algorithm effectively assumes that the unweighted Hessian is a good approximation to the weighted Hessian. This approximation is similar to the various approximations made to the Hessian in Section 4 of [2]. So long as the weighting function $Q(\mathbf{x}) = \varrho'([I(\mathbf{W}(\mathbf{x};\mathbf{p})) - T(\mathbf{x})]^2)$ is not too extreme, the performance of the *H*-Algorithm is not much worse than that of the *W*-Algorithm. It is important, however, to ensure that the weighting function does not affect the step size. If $Q(\mathbf{x})$ is on average $\ll 1$, the algorithm will take too small steps and so will converge slowly. If $Q(\mathbf{x})$ is on average $\gg 1$, the algorithm will take too large steps and may well diverge. (This is not an issue with the weighted Hessian because the $Q(\mathbf{x})$ appears in the definition of the

Hessian too.) To obtain the best performance, the weighting function $Q(\mathbf{x})$ should be normalized when using the *H*-Algorithm. Any norm could be used to compute the "average" value of $Q(\mathbf{x})$. If the L1 norm is used, the weighting function is normalized by replacing $Q(\mathbf{x})$ with:

$$\frac{N}{\sum_{\mathbf{x}} Q(\mathbf{x})} Q(\mathbf{x}) \tag{66}$$

where N is the number of pixels in the template. This normalization must be performed in every iteration of the iteratively reweighted least squares algorithm (in Step 7.) See Figure 8. The computational cost of this normalization is $\mathrm{O}(n\,N)$ and so the asymptotic complexity is not affected. The *H*-Algorithm was used by Hager and Belhumeur in [9] to obtain real-time robust tracking.

4.4.2 Spatial Coherence of Outliers

One way to interpret robust image alignment algorithms is that they give *outliers* less weight by making $\varrho'([I(\mathbf{W}(\mathbf{x};\mathbf{p})) - T(\mathbf{x})]^2)$ smaller for large $|I(\mathbf{W}(\mathbf{x};\mathbf{p})) - T(\mathbf{x})|$). If we assume that the outliers appear in spatially coherent groups, we can derive a second efficient approximation. Assuming the outliers are spatially coherent is often a reasonable assumption. For example, when part of the template is occluded, the outliers will form a spatially coherent group around the occluder. Similarly, specularities and shadow pixels are also usually highly spatially localized.

To take advantage of the spatial coherence of the outliers, the template is subdivided into a set of sub-templates or *blocks*. Suppose there are K blocks $B_1, B_2, \ldots, B_K$ with N_i pixels in the i^{th} block. Although this subdivision of the template can be performed in an arbitrary manner, typically a rectangular template would be split into a collection of rectangular blocks arranged on a regular grid. See [11] for an example. Equation (64) can then be rewritten as:

$$H_\varrho = \sum_{i=1}^{K} \sum_{\mathbf{x} \in B_i} \varrho' \left([I(\mathbf{W}(\mathbf{x};\mathbf{p})) - T(\mathbf{x})]^2 \right) \left[\nabla T \frac{\partial \mathbf{W}}{\partial \mathbf{p}} \right]^{\mathrm{T}} \left[\nabla T \frac{\partial \mathbf{W}}{\partial \mathbf{p}} \right]. \tag{67}$$

Based on the spatial coherence of the outliers, assume that $\varrho'([I(\mathbf{W}(\mathbf{x};\mathbf{p})) - T(\mathbf{x})]^2)$ is constant on each block; i.e. assume $\varrho'([I(\mathbf{W}(\mathbf{x};\mathbf{p})) - T(\mathbf{x})]^2) = \varrho'_i$, say, for all $\mathbf{x} \in B_i$. In practice this assumption never holds completely and so ϱ'_i must be estimated from $\varrho'([I(\mathbf{W}(\mathbf{x};\mathbf{p})) - T(\mathbf{x})]^2)$. One possibility is to set ϱ'_i to be the mean value:

$$\varrho'_i = \frac{1}{N_i} \sum_{\mathbf{x} \in B_i} \varrho'([I(\mathbf{W}(\mathbf{x};\mathbf{p})) - T(\mathbf{x})]^2). \tag{68}$$

Another way might to be argue that if there is ever an outlier in the block, then the entire block should be classified as an outlier. The value of ϱ'_i might then to be set to be the minimum value across the block; i.e. set the weight of the block to be the weight of the worst pixel in the block:

$$\varrho'_i \;=\; \min_{\mathbf{x}\in B_i} \varrho'([I(\mathbf{W}(\mathbf{x};\mathbf{p})) - T(\mathbf{x})]^2). \tag{69}$$

Assuming $\varrho'([I(\mathbf{W}(\mathbf{x};\mathbf{p})) - T(\mathbf{x})]^2) = \varrho'_i$ for all $\mathbf{x} \in B_i$, Equation (67) can then be rearranged to:

$$H_\varrho \;=\; \sum_{i=1}^{K} \varrho'_i \sum_{\mathbf{x}\in B_i} \left[\nabla T \frac{\partial \mathbf{W}}{\partial \mathbf{p}}\right]^{\mathrm{T}} \left[\nabla T \frac{\partial \mathbf{W}}{\partial \mathbf{p}}\right]. \tag{70}$$

The internal part of this expression does not depend on the robust function ϱ' and so is constant across iterations. Denote:

$$H_i \;=\; \sum_{\mathbf{x}\in B_i} \left[\nabla T \frac{\partial \mathbf{W}}{\partial \mathbf{p}}\right]^{\mathrm{T}} \left[\nabla T \frac{\partial \mathbf{W}}{\partial \mathbf{p}}\right]. \tag{71}$$

The Hessian H_i is the Hessian for the sub-template B_i and can be precomputed. Equation (70) then simplifies to:

$$H_\varrho \;=\; \sum_{i=1}^{K} \varrho'_i \cdot H_i. \tag{72}$$

Although this Hessian does vary from iteration to iteration, the cost of computing it is minimal: $\mathrm{O}(K \cdot n^2)$. Typically $K \ll N$ (where N is the number of pixels in the template) and so $\mathrm{O}(K \cdot n^2)$ is substantially smaller than $\mathrm{O}(N \cdot n^2)$, the cost of computing the Hessian in Step 6 of the original inverse compositional iteratively reweighted least squares algorithm. See Figure 8 and Table4.

The spatial coherent approximation to the inverse compositional iteratively reweighted least squares algorithm then just consists of using Equation (72) to estimate the Hessian in Step 6 rather than Equation (64). Using Equation (72), of course, requires that the Hessians H_i are precomputed for each block $i = 1, \ldots, K$. The total cost of this pre-computation is $\mathrm{O}(N \cdot n^2)$. Note that when $K = 1$ and there is just one block the size of the original template, this efficiency approximation reduces to the *H*-algorithm of the previous section. When $K = N$ and there is one block for each pixel, this algorithm reduces to the original inverse compositional iteratively reweighted least squares algorithm. By varying the value of K, we can trade-off efficiency for performance.

Note that the spatial coherence of the outliers has been used before in robust image alignment algorithms [11]. Also, Shum and Szeliski [15] have suggested using the spatial coherence of the

Hessian itself (rather than the robust weighting function ϱ') to obtain efficiency improvements. In this approach, the Hessian is estimated from a single sample of $[\nabla T \frac{\partial \mathbf{W}}{\partial \mathbf{p}}]^{\mathrm{T}}[\nabla T \frac{\partial \mathbf{W}}{\partial \mathbf{p}}]$ per block B_i.

4.5 Extensions to the Algorithms

There are a number of extensions that can be applied to any of the algorithms above, the inverse compositional iteratively reweighted least squares algorithm and its two efficient approximations.

4.5.1 Spatially Varying Robust Error Functions

The robust error function in Equation (49) treats every pixel identically; the quantity to be minimized depends on the error $I(\mathbf{W}(\mathbf{x};\mathbf{p}))-T(\mathbf{x})$, but not directly on the pixel $\mathbf{x}$. One straightforward extension is to use a different robust function at each pixel; i.e. minimize:

$$\sum_{\mathbf{x}} \varrho_{\mathbf{x}} \left([I(\mathbf{W}(\mathbf{x};\mathbf{p})) - T(\mathbf{x})]; \boldsymbol{\sigma} \right) \tag{73}$$

where each $\varrho_{\mathbf{x}}(t;\boldsymbol{\sigma})$ is a robust function. Making this change does not affect the algorithms significantly, $\varrho'_{\mathbf{x}}$ is simply used instead of ϱ' in Equations (63) and (64).

There are a variety of applications of this generalization. One is to give a spatially varying weight, essentially producing a "weighted robust error function." For example, we might set:

$$\varrho_{\mathbf{x}}(t;\boldsymbol{\sigma}) \; = \; Q(\mathbf{x}) \, \varrho(t;\boldsymbol{\sigma}) \tag{74}$$

where $Q(\mathbf{x})$ is a spatially varying weighting function that plays the same role as it did in Section 3 and $\varrho(t;\boldsymbol{\sigma})$ is the original spatially invariant robust function. Another closely related, but slightly different, idea is to down-weight the error by $Q(\mathbf{x})$ by setting:

$$\varrho_{\mathbf{x}}(t;\boldsymbol{\sigma}) \; = \; \varrho\left(\frac{t}{Q(\mathbf{x})};\boldsymbol{\sigma}\right). \tag{75}$$

As in Section 3 one possible choice for $Q(\mathbf{x})$ is $|\nabla T(\mathbf{x})|$. Using this choice in Equation (75) down-weights the error $I(\mathbf{W}(\mathbf{x};\mathbf{p}))-T(\mathbf{x})$ so that pixels with high gradient $|\nabla T(\mathbf{x})|$ are penalized less. (A small alignment error produces a large error $I(\mathbf{W}(\mathbf{x};\mathbf{p})) - T(\mathbf{x})$ if the gradient is large.)

4.5.2 Scale Estimation

Until now we have assumed that the scale parameters $\boldsymbol{\sigma}$ are known constants. It is also possible to assume that the scale parameters are unknown and must be estimated along with the warp parameters $\mathbf{p}$. One way to do this is to assume a parametric form for the distribution $-\log \mathbf{P}[I(\mathbf{W}(\mathbf{x};\mathbf{p})) - T(\mathbf{x}) = x]$ as a function of the scale parameters $\boldsymbol{\sigma}$. (In Section 4.1 we pointed out that the best choice for the robust function is $\rho(t) \propto -\log \mathbf{P}[I(\mathbf{W}(\mathbf{x};\mathbf{p})) - T(\mathbf{x}) = t]$.) If we knew the warp parameters $\mathbf{p}$ we could compute $I(\mathbf{W}(\mathbf{x};\mathbf{p})) - T(\mathbf{x})$ for each pixel $\mathbf{x}$. We could then estimate the scale parameters $\boldsymbol{\sigma}$ by fitting the distribution $-\log \mathbf{P}[I(\mathbf{W}(\mathbf{x};\mathbf{p})) - T(\mathbf{x}) = x]$ to the samples $I(\mathbf{W}(\mathbf{x};\mathbf{p})) - T(\mathbf{x})$. Of course the warp parameters are not known and so a simultaneous optimization must be performed over both sets of parameters. This can be achieved by iteratively updating the warp and scale parameters in turn. After each iteration of the iteratively weighted least squares algorithm, the current estimates of the scale parameters are updated by performing one iteration of the algorithm to fit them [8, 13]. This step depends on the robust function but is in general a non-linear optimization, which might be performed using Gauss-Newton, say. For certain distributions, however, there is a simple closed-form solution for the scale parameters.

4.6 Experimental Results

4.6.1 Generation of the Inputs

Evaluating robust fitting algorithms is hard because there is no obvious noise model to use. (The template and the starting affine parameters can be generated exactly as in Section 3.2.1 and [2]. This is the easy part. The hard part is to generate the input image $I(\mathbf{x})$.) We start with the same input image as in Section 3.2.1. We assume that the main cause of noise (i.e. outliers) is occlusion and generate the input image in the following manner. The evaluation is governed by one parameter, the percentage of occlusion. Given this parameter, we randomly generate a rectangle in $I(\mathbf{x})$ entirely within the template region that occludes the template by the appropriate percentage. We allow a small relative error of 5% in the occlusion region to allow for the discrete nature of the pixels. We then synthetically "occlude" the randomly generated rectangle by replacing that part of the image with another image of the appropriate size. We use a variety of occluding images, from the constant "all-black" image, to images of other faces and natural scenery. By varying the occluder image we can evaluate how sensitive the algorithms are to outlier detection. See Experi-

ment 2 in Section 4.6.4. By varying the percentage of occlusion we can evaluate how sensitive the algorithms are to the amount of occlusion. See Experiment 1 in Section 4.6.3.

4.6.2 The Robust Error Function and Scale Estimation

Another thing that makes evaluating robust fitting algorithms hard is choosing the robust error function. We use the very simple robust error function:

$$\varrho(t;\sigma_1) = \begin{cases} t & \text{if } 0 \le t \le \sigma_1 \\ \sigma_1 & \text{if } t > \sigma_1. \end{cases} \tag{76}$$

Intuitively this function classifies pixels as outliers if the magnitude of the error is larger than the parameter (threshold) σ_1. Inliers are weighted equally and outliers are given zero weight. In the spatial coherent approximation to the inverse compositional iteratively re-weighted least squares algorithms, we combine the multiple estimates of ϱ_i' using Equation (68).

We estimate the scale parameter σ_1 in the following way. We assume that we know the number of outliers and the number of inliers. We then estimate σ_1 by sorting the error values (divided by the magnitude of the gradient of the template) and setting σ_1 so that the correct number of pixels are classified as outliers. By varying the estimate of the number of outliers, we can evaluate how sensitive the algorithms are to inaccurate estimates of σ_1. See Experiment 3 in Section 4.6.5.

Each of the choices we made above for the input generation, the robust error function, and for scale estimation are just one of many possibilities. We like these choices, but others may disagree. We see no obvious reason, however, why the relative performance of the algorithms we are comparing would be significantly affected if different choices were made. If the reader wishes to try other options, they can download our algorithms, test images, and scripts. See Section 5.4.

4.6.3 Experiment 1: Varying the Percentage of Occlusion

In our first experiment we compare four algorithms: (1) the original inverse compositional algorithm, (2) the iteratively re-weighted least squares algorithm, (3) the H-algorithm, and (4) the spatial coherent approximation to the iteratively re-weighted least squares algorithm with block size 5×5 pixels. As above, we compute the average speed of convergence and the average frequency of convergence. We use the "all-black" image as the occluder. The results for three different percentages of occlusion are shown in Figure 9. The results for 10% occlusion are shown in Figures 9(a) and (b), for 30% occlusion in Figures 9(c) and (d), and for 50% occlusion in Figures 9(e) and (f).

The first thing to note from Figure 9 is that as the percentage of occlusion increases from 10% to 30% and 50%, the performance of the original inverse compositional algorithm drops off rapidly. By 50% occlusion, the algorithm almost never converges. On the other hand, the 3 robust algorithm all converge fairly well. In comparison, the inverse compositional iteratively reweighted least squares and the spatial coherence approximation perform significantly better than the H-algorithm both in terms of the rate and frequency of convergence, especially with more occlusion.

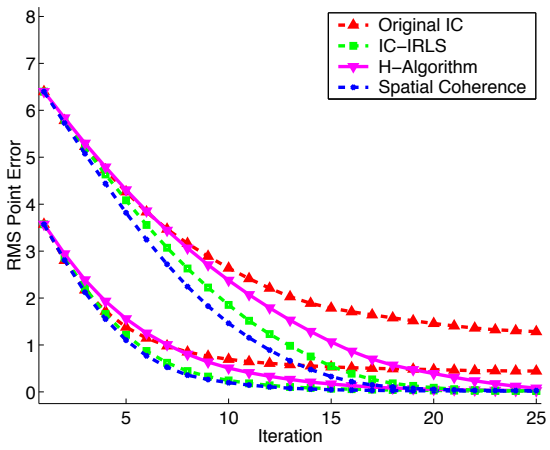
4.6.4 Experiment 2: Varying the Occluder Image

In our second experiment we investigate how the performance depends on the occluder image. We compute the average speed of convergence and the average frequency of convergence for the same four algorithms. We use three different occluder images, an image of scenery, an image of another face approximately aligned, and a constant intensity image with constant intensity set to be the average of the template. The results for 30% occlusion are shown in Figure 10.

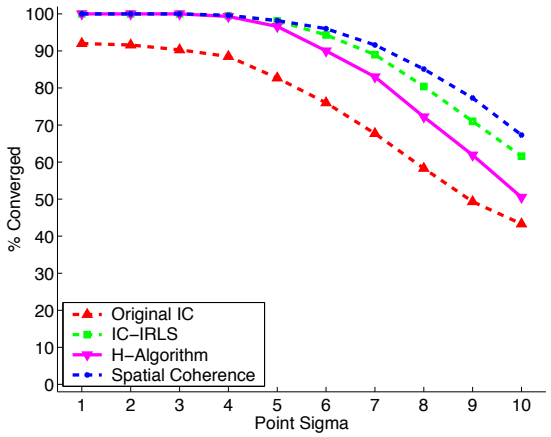
The results for the "scenery" occluder in Figures 10(a) and (b), and for the "Face" occluder in Figures 10(c) and (d) are qualitatively very similar to those for the "all-black" occluder in Figures 9(c) and (d). The results for the "average grey-level" occluder in Figures 10(e) and (f) are quite different, however. In particular, the original inverse compositional algorithm performs better than the iteratively re-weighted least squares algorithm, although the spatial coherence approximation to it performs as well as the original inverse compositional algorithm. The H-algorithm performs far worse. In this case, the "average grey-level" occluder is not much of an outlier and so doesn't affect the original inverse compositional algorithm. The iteratively re-weighted least squares algorithm throws away the most likely outliers (usually pixels with large gradients) and so performs slightly worse than the original inverse compositional algorithm which does not.

4.6.5 Experiment 3: Varying the Estimated Percentage of Outliers

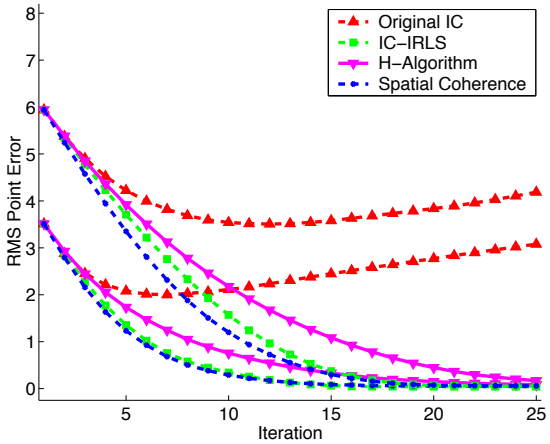
In our third experiment we investigate how the performance of the robust algorithms varies depending on how well the scale estimation is performed. We vary the estimate of the number of occluders from zero to far more than actuality. The results for the "all-black" occluder and for 30% occlusion are shown in Figure 11. The results in Figures 11(a) and (b) are for an estimated 10% of occluders, the results in Figures 11(c) and (d) for (the correct) 30% of occluders, and the results in Figures 11(e) and (f) for an estimated 50% of occluders. If the estimated number of oc-



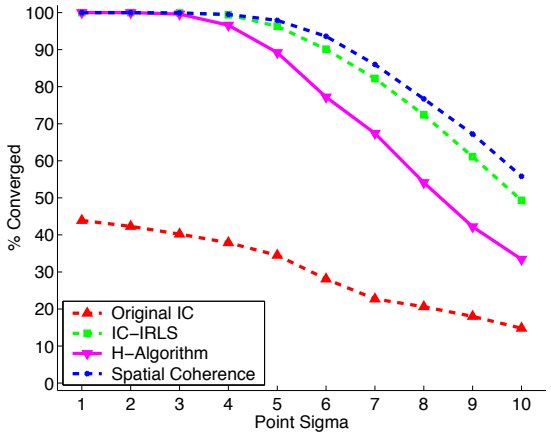
(a) Convergence Rate, 10% Occlusion



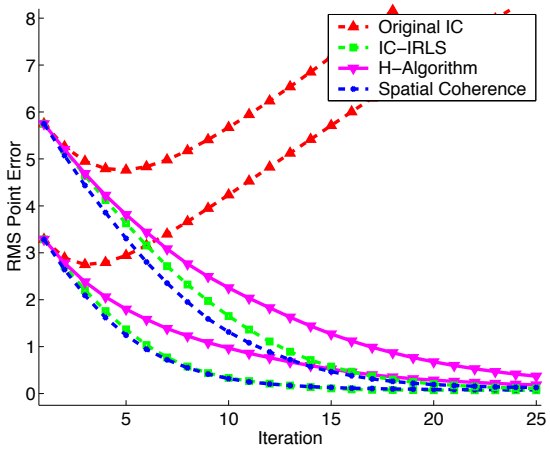
(b) Convergence Frequency, 10% Occlusion



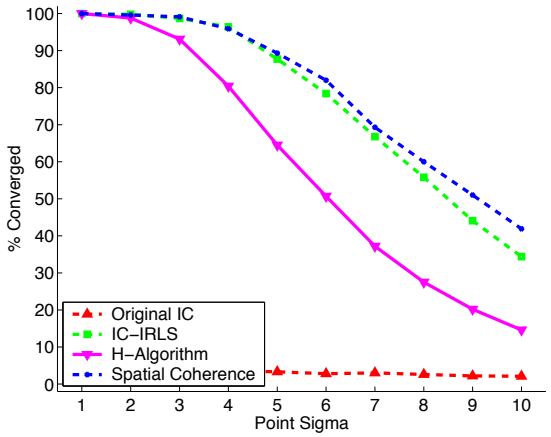
(c) Convergence Rate, 30% Occlusion



(d) Convergence Frequency, 30% Occlusion

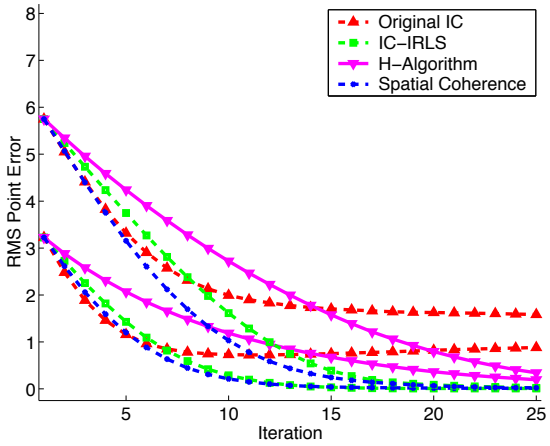


(e) Convergence Rate, 50% Occlusion

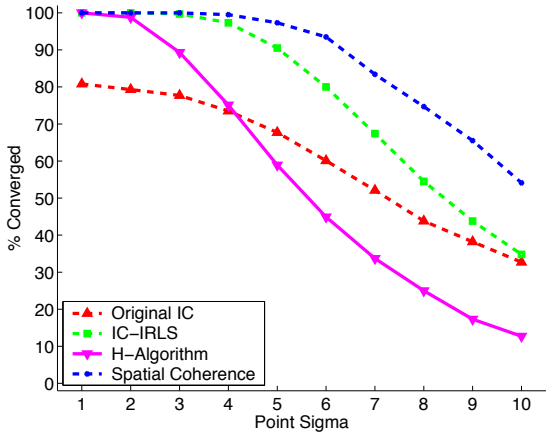


(f) Convergence Frequency, 50% Occlusion

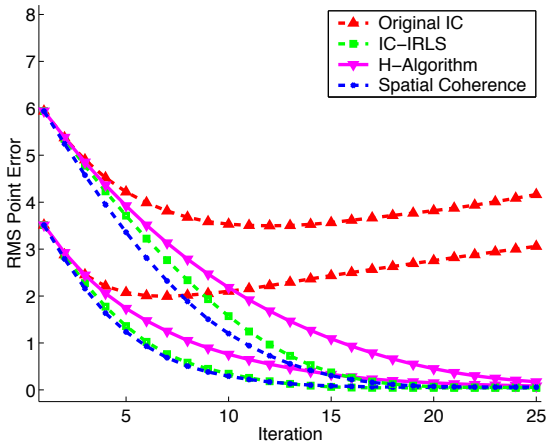
Figure 9: A comparison of the original inverse compositional algorithm, the iteratively re-weighted least squares algorithm, the H-algorithm, and the spatial coherent approximation to the iteratively re-weighted least squares algorithm using the "all-black" occluder. The original inverse compositional algorithm performs very poorly, converging very infrequently. The iteratively re-weighted least squares algorithm and the spatial approximation to it both perform very well, and the H-algorithm not quite so well.



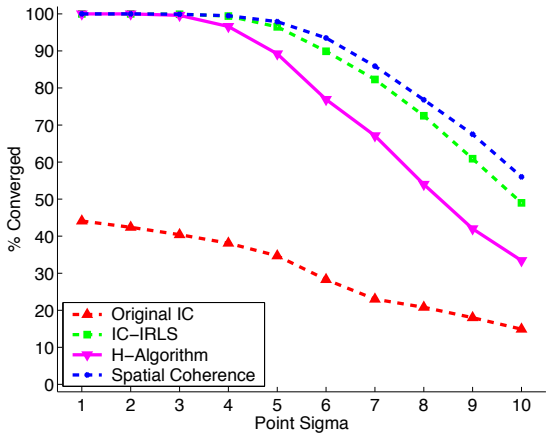
(a) Convergence Rate, "Scenery" Occluer



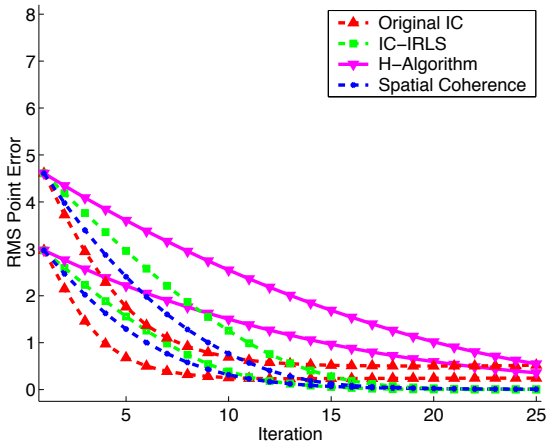
(b) Divergence Rate, "Scenery" Occluer



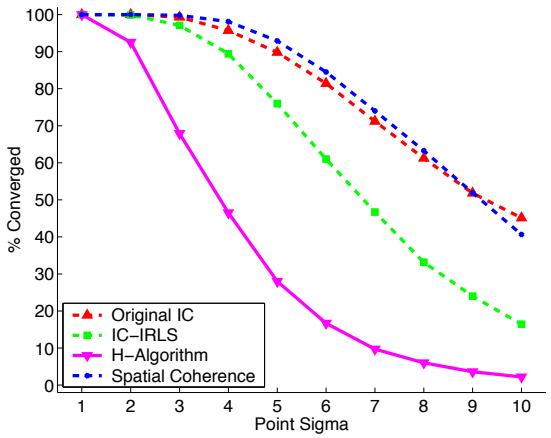
(c) Convergence Rate, "Face" Occluer



(d) Divergence Rate, "Face" Occluer

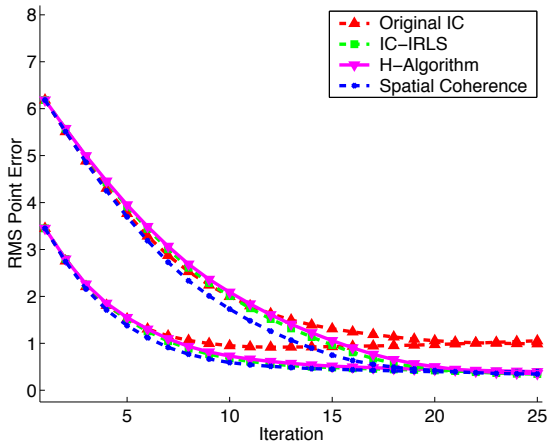


(e) Convergence Rate, "Average GL" Occluer

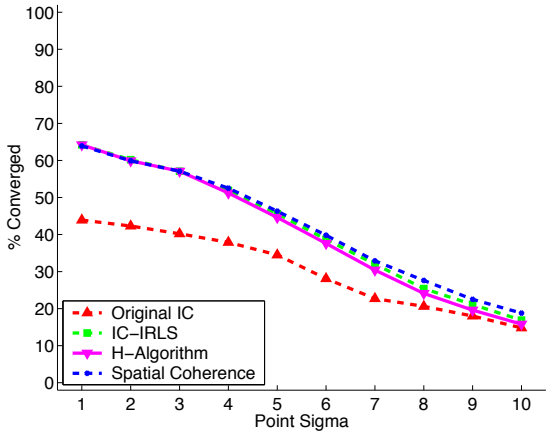


(f) Divergence Rate, "Average GL" Occluer

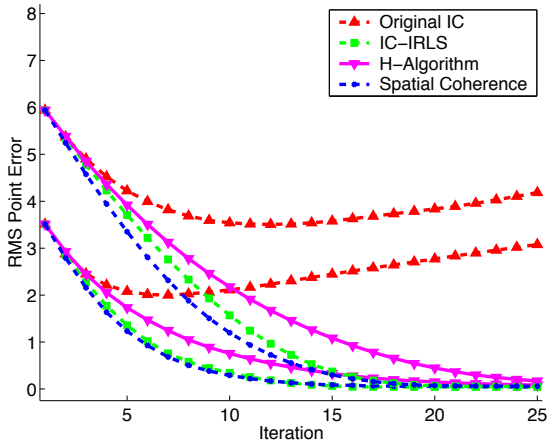
Figure 10: A comparison of the four algorithms for three different occluders for 30% occlusion. The results with a "scenery" occluder and a "face" occluder are qualitatively the same as the corresponding "all-black" occluder results in Figures 9(c) and (d). The results for the "average grey-level" occluder are different because this occluder does not lead to strong outliers. In this case, the robust algorithms perform slightly worse than the original inverse compositional algorithm. The H-algorithm performs signifi cantly worse.



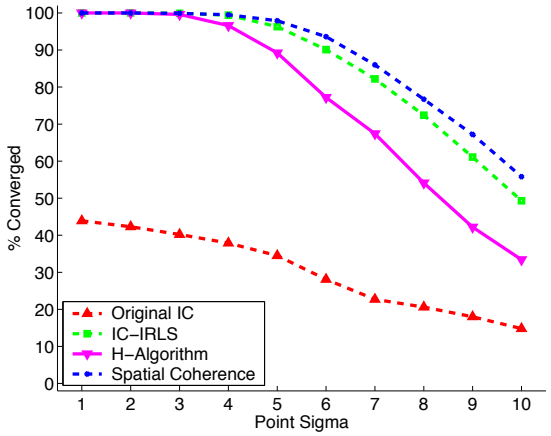
(a) Convergence Rate, Est. 10% Occlusion



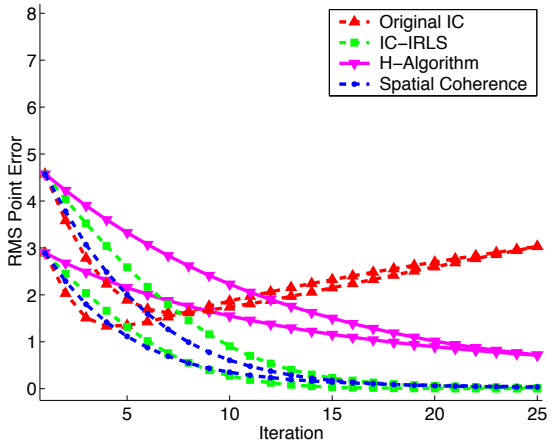
(b) Convergence Frequency, Est. 10% Occlusion



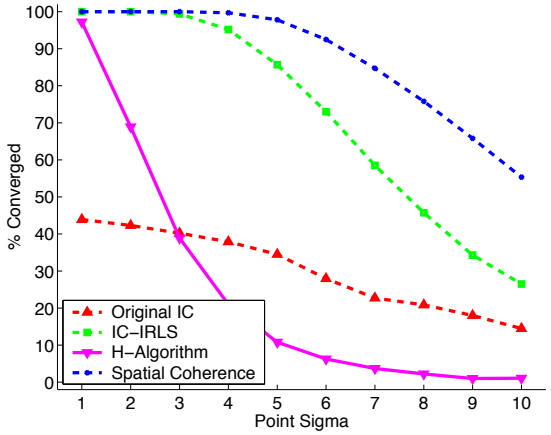
(c) Convergence Rate, Est. 30% Occlusion



(d) Convergence Frequency, Est. 30% Occlusion

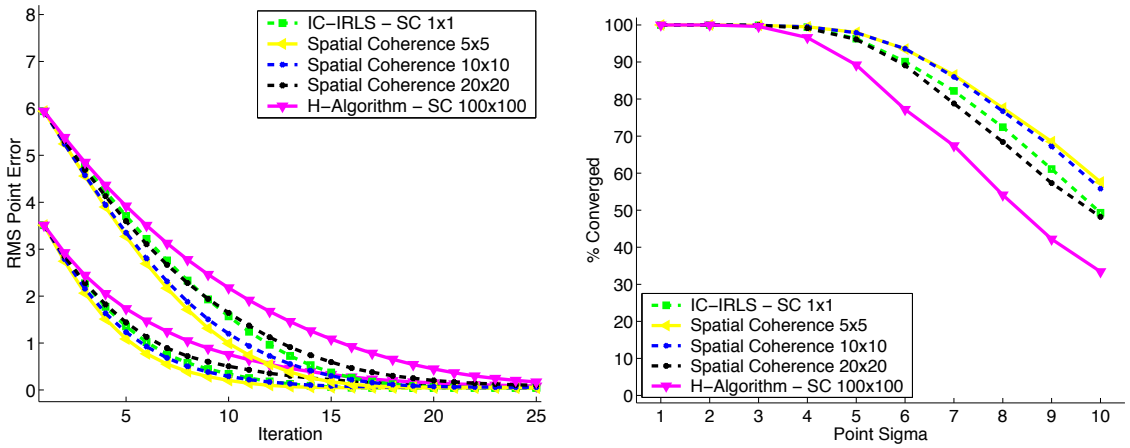


(e) Convergence Rate, Est. 50% Occlusion



(f) Convergence Frequency, Est. 50% Occlusion

Figure 11: Varying the percentage of estimated occlusion from the correct value of 30%. If the estimated percentage of occlusion is too low, as in (a) and (b), the robust algorithms perform poorly, and almost as poorly as the original inverse compositional algorithm. If the estimated percentage of occlusion is too high, as in (e) and (f), the iteratively re-weighted least squares algorithm and the spatial coherence approximation to it perform reasonably well, although not quite as well as with the correct amount of occlusion. The H-algorithm performs much worse than the other two robust algorithms.



(a) Convergence Rate (b) Convergence Frequency

Figure 12: Varying the block size in the spatial coherence algorithm. When the block size is 1×1 pixels, the spatial coherence algorithm reduces to the iteratively re-weighted least squares algorithm and when the block size is 100×100 pixels (the entire template), the algorithm reduces to the H-algorithm.

cluders is too low, as in Figures 11(a) and (b), the robust algorithms all perform poorly, and almost as poorly as the original inverse compositional algorithm. If the estimated number of occluders is too high, as in Figures 11(e) and (f), the iteratively re-weighted least squares algorithm and the spatial coherence approximation to it perform fairly well (although not as well as when the correct number of outliers is used). The H-algorithm performs much worse, however.

4.6.6 Experiment 4: Varying the Block Size in the Spatial Coherence Algorithm

In our final experiment we vary the block size in the spatial coherence approximation to the iteratively re-weighted least squares algorithm. The results are presented in Figure 12 for the "all-black" occluder and 30% occlusion. The results confirm that as the block size varies from 1×1 pixels (where the algorithm reduces to the iteratively re-weighted least squares algorithm) to 100×100 pixels (where the algorithm reduces to the H-algorithm) the results get generally worse. Somewhat surprisingly, however, the performance gets slightly better at first when the block size increases to 5×5 or 10×10 pixels. This is a result of the slight smoothing in the computation of the Hessian.

5 Conclusion

5.1 Summary

In Table 5 we present a summary of the algorithms described in this paper. In Section 3 we introduced the inverse compositional algorithm with a weighted L2 norm. This algorithm is just

Table 5: A summary of the algorithms described in this paper. In Section 3 we described the inverse compositional algorithm with a weighted L2 norm. In Section 4 we described the inverse compositional iteratively reweighted least squares algorithm for a robust error function and two efficient approximations.

Error Function	Algorithm	Efficient?	Correct?	Performance
Weighted L2	IC With Weighted L2 Norm	Yes	Yes	Good
Robust	IC Iteratively Reweighted Least Squares	No	Yes	Good
Robust	IC IRLS H-Algorithm	Yes	Approximate	Poor
Robust	IC IRLS with Spatial Coherence	Yes	Approximate	Medium

as efficient as the original inverse compositional algorithm, and performs as well. We described three applications of this algorithm: (1) weighting the pixels with confidence values for increased robustness and speed of convergence, (2) pixel selection for computational efficiency, and (3) efficiently allowing linear appearance variation. In Section 4 we described the inverse compositional iteratively reweighted least squares algorithm for a robust error function. This algorithm, although it performs well, is substantially less efficient than the original inverse compositional algorithm. We also described two efficient approximations to this algorithm: (1) the H-Algorithm [8] and (2) an approximation that takes advantage of spatial coherence. Both of these algorithms are efficient (the H-Algorithm is slightly more efficient), however the spatial coherence approximation performs far better in practice (and approximately as well as the IC IRLS algorithm.)

5.2 Discussion

In Part 1 we discussed which is the best algorithm to use. The discussion centered on two topics: (1) the nature of the noise and (2) whether or not an efficient algorithm is required. The same is true here. The algorithm to use depends on the noise and the computational requirements.

As discussed in Section 4.1 the "theoretically best" error functions to use (in the Maximum Likelihood sense, and assuming that the pixels are independent) is:

$$\rho(t) \propto -\log \mathbf{P}[I(\mathbf{W}(\mathbf{x};\mathbf{p})) - T(\mathbf{x}) = t]. \tag{77}$$

If the noise is Gaussian, this means that a weighted L2 norm should be used (and $Q(\mathbf{x},\mathbf{y})$ should be set to be the inverse of the covariance of the noise.) This was empirically verified in Section 3.2.1. If the noise is not Gaussian, the appropriate robust error function should be used instead. To obtain the best performance, the inverse compositional iteratively reweighted least squares algorithm should be used. Note, however, that this algorithm is relatively inefficient.

If we care about obtaining high efficiency and are using an unweighted or weighted L2 norm, the pixel selection algorithm of Section 3.3 can be used. If we care about obtaining high efficiency and are using a robust error function, one of the two approximations to the inverse compositional iteratively reweighted least squares algorithm should be used. Of the two, the spatial coherence approximation is the more general and the block size can be used to control the trade-off between computational efficiency and convergence performance. It is therefore the better choice.

5.3 Future Work

Besides the choices we have described in this paper and in Part 1 [2], there are a variety of others that can be made by an image alignment algorithm. These include, whether to allow appearance variation and whether to add priors on the parameters. In future papers in this series we will extend our framework to cover these choices and, in particular, investigate whether the efficient inverse compositional algorithm is compatible with these extensions of the Lucas-Kanade algorithm.

5.4 Matlab Code, Test Images, and Scripts

Matlab implementations of all of the algorithms described in this paper will be made available on the World Wide Web at: http://www.cs.cmu.edu/~iainm/lk20. We will also include all of the test images and the scripts used to generate the experimental results in this paper.

Acknowledgments

The research described in this report was partially supported by Denso Corporation, Japan, and was conducted at Carnegie Mellon University Robotics Institute while Takahiro Ishikawa was a Visiting Industrial Scholar at CMU from Denso Corporation. This research was also supported, in part, by the U.S. Department of Defense through award number N41756-03-C4024.

References

[1] S. Baker and I. Matthews. Equivalence and efficiency of image alignment algorithms. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, volume 1, pages 1090–1097, 2001.

[2] S. Baker and I. Matthews. Lucas-Kanade 20 years on: A unifying framework: Part 1: The quantity approximated, the warp update rule, and the gradient descent approximation. *Accepted to Appear in the International Journal of Computer Vision*, 2003.

[3] J.R. Bergen, P. Anandan, K.J. Hanna, and R. Hingorani. Hierarchical model-based motion estimation. In *Proceedings of the European Conference on Computer Vision*, pages 237–252, 1992.

[4] M. Black and A. Jepson. Eigen-tracking: Robust matching and tracking of articulated objects using a view-based representation. *International Journal of Computer Vision*, 36(2):101–130, 1998.

[5] G.E. Christensen and H.J. Johnson. Image consistent registration. *IEEE Transactions on Medical Imaging*, 20(7):568–582, 2001.

[6] T.F. Cootes, G.J. Edwards, and C.J. Taylor. Active appearance models. In *Proceedings of the European Conference on Computer Vision*, volume 2, pages 484–498, 1998.

[7] F. Dellaert and R. Collins. Fast image-based tracking by selective pixel integration. In *Proceedings of the ICCV Workshop on Frame-Rate Vision*, pages 1–22, 1999.

[8] R. Dutter and P.J. Huber. Numerical methods for the nonlinear robust regression problem. *Journal of Statistical and Computational Simulation*, 13:79–113, 1981.

[9] G.D. Hager and P.N. Belhumeur. Efficient region tracking with parametric models of geometry and illumination. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(10):1025–1039, 1998.

[10] P.J. Huber. *Robust Statistics*. John Wiley & Sons, 1981.

[11] T. Ishikawa, I. Matthews, and S. Baker. Efficient image alignment with outlier rejection. Technical Report CMU-RI-TR-02-27, Carnegie Mellon University Robotics Institute, 2002.

[12] B. Lucas and T. Kanade. An iterative image registration technique with an application to stereo vision. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 674–679, 1981.

[13] H.S. Sawhney and S. Ayer. Compact representations of videos through dominant and multiple motion estimation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 18(8):814–830, 1996.

[14] S. Sclaroff and J. Isidoro. Active blobs. In *Proceedings of the 6th IEEE International Conference on Computer Vision*, pages 1146–1153, 1998.

[15] H.-Y. Shum and R. Szeliski. Construction of panoramic image mosaics with global and local alignment. *International Journal of Computer Vision*, 16(1):63–84, 2000.

A Proofs of Equivalence

A.1 Weighted L2 Norms

The goal of image alignment with a weighted L2 norm is to minimize:

$$\sum_{\mathbf{x}} \sum_{\mathbf{y}} Q(\mathbf{x}, \mathbf{y}) \cdot [\, I(\mathbf{W}(\mathbf{x}; \mathbf{p})) - T(\mathbf{x}) \,] \cdot [\, I(\mathbf{W}(\mathbf{y}; \mathbf{p})) - T(\mathbf{y}) \,] \tag{78}$$

with respect to the warp parameters $\mathbf{p}$. The *forwards additive* algorithm minimizes this expression by iteratively minimizing:

$$\sum_{\mathbf{x}} \sum_{\mathbf{y}} Q(\mathbf{x}, \mathbf{y}) \cdot [\, I(\mathbf{W}(\mathbf{x}; \mathbf{p} + \Delta\mathbf{p})) - T(\mathbf{x}) \,] \cdot [\, I(\mathbf{W}(\mathbf{y}; \mathbf{p} + \Delta\mathbf{p})) - T(\mathbf{y}) \,] \tag{79}$$

with respect to $\Delta\mathbf{p}$ and then updating the parameters $\mathbf{p} \leftarrow \mathbf{p} + \Delta\mathbf{p}$. The *forwards compositional* algorithm minimizes the same expression by iteratively minimizing:

$$\sum_{\mathbf{x}} \sum_{\mathbf{y}} Q(\mathbf{x}, \mathbf{y}) \cdot [\, I(\mathbf{W}(\mathbf{W}(\mathbf{x}; \Delta\mathbf{p}); \mathbf{p})) - T(\mathbf{x}) \,] \cdot [\, I(\mathbf{W}(\mathbf{W}(\mathbf{y}; \Delta\mathbf{p}); \mathbf{p})) - T(\mathbf{y}) \,] \tag{80}$$

with respect to $\Delta\mathbf{p}$ and then updating the warp $\mathbf{W}(\mathbf{x}; \mathbf{p}) \leftarrow \mathbf{W}(\mathbf{x}; \mathbf{p}) \circ \mathbf{W}(\mathbf{x}; \Delta\mathbf{p})$. Finally, the *inverse compositional* algorithm minimizes the same expression by iteratively minimizing:

$$\sum_{\mathbf{x}} \sum_{\mathbf{y}} Q(\mathbf{x}, \mathbf{y}) \cdot [\, T(\mathbf{W}(\mathbf{x}; \Delta\mathbf{p})) - I(\mathbf{W}(\mathbf{x}; \mathbf{p})) \,] \cdot [\, T(\mathbf{W}(\mathbf{y}; \Delta\mathbf{p})) - I(\mathbf{W}(\mathbf{y}; \mathbf{p})) \,] \tag{81}$$

with respect to $\Delta\mathbf{p}$ and updating the warp: $\mathbf{W}(\mathbf{x}; \mathbf{p}) \leftarrow \mathbf{W}(\mathbf{x}; \mathbf{p}) \circ \mathbf{W}(\mathbf{x}; \Delta\mathbf{p})^{-1}$. To show the equivalence of the forwards additive and inverse compositional formulations, we first show the equivalence of the forwards additive and forwards compositional formulations. Afterwards we show the equivalence of the forwards and inverse compositional formulations. The desired result follows by transitivity of equivalence.

A.1.1 Equivalence of Forwards Additive and Compositional Algorithms

We now show that the forwards additive and forwards compositional approaches are equivalent in the sense that, to a first order approximation in $\Delta\mathbf{p}$, they take the same steps in each iteration; i.e. the updates to the warps are approximately the same. In the forwards additive formulation we

iteratively minimize Equation (79) which simplifies to:

$$\sum_{\mathbf{x}}\sum_{\mathbf{y}} Q(\mathbf{x},\mathbf{y}) \cdot \left[I(\mathbf{W}(\mathbf{x};\mathbf{p})) + \boldsymbol{\nabla} I \frac{\partial \mathbf{W}}{\partial \mathbf{p}} \Delta\mathbf{p} - T(\mathbf{x}) \right] \cdot \left[I(\mathbf{W}(\mathbf{y};\mathbf{p})) + \boldsymbol{\nabla} I \frac{\partial \mathbf{W}}{\partial \mathbf{p}} \Delta\mathbf{p} - T(\mathbf{y}) \right]. \tag{82}$$

We then update $\mathbf{p} \leftarrow \mathbf{p} + \Delta\mathbf{p}$. The corresponding update to the warp is:

$$\mathbf{W}(\mathbf{x};\mathbf{p}) \;\leftarrow\; \mathbf{W}(\mathbf{x};\mathbf{p}+\Delta\mathbf{p}) \;\approx\; \mathbf{W}(\mathbf{x};\mathbf{p}) + \frac{\partial \mathbf{W}}{\partial \mathbf{p}} \Delta\mathbf{p} \tag{83}$$

after a Taylor expansion is made. The forwards compositional formulation in Equation (80) simplifies to:

$$\sum_{\mathbf{x}}\sum_{\mathbf{y}} Q(\mathbf{x},\mathbf{y}) \cdot \left[I(\mathbf{W}(\mathbf{x};\mathbf{p})) + \boldsymbol{\nabla} I \frac{\partial \mathbf{W}}{\partial \mathbf{x}} \frac{\partial \mathbf{W}}{\partial \mathbf{p}} \Delta\mathbf{p} - T(\mathbf{x}) \right] \cdot \left[I(\mathbf{W}(\mathbf{y};\mathbf{p})) + \boldsymbol{\nabla} I \frac{\partial \mathbf{W}}{\partial \mathbf{x}} \frac{\partial \mathbf{W}}{\partial \mathbf{p}} \Delta\mathbf{p} - T(\mathbf{y}) \right]. \tag{84}$$

In the forwards compositional approach, the warp update is $\mathbf{W}(\mathbf{x};\mathbf{p}) \leftarrow \mathbf{W}(\mathbf{x};\mathbf{p}) \circ \mathbf{W}(\mathbf{x};\Delta\mathbf{p})$. To simplify this expression, note that:

$$\mathbf{W}(\mathbf{x};\Delta\mathbf{p}) \;\approx\; \mathbf{W}(\mathbf{x};\mathbf{0}) + \frac{\partial \mathbf{W}}{\partial \mathbf{p}} \Delta\mathbf{p} \;=\; \mathbf{x} + \frac{\partial \mathbf{W}}{\partial \mathbf{p}} \Delta\mathbf{p} \tag{85}$$

is the first order Taylor expansion of $\mathbf{W}(\mathbf{x};\Delta\mathbf{p})$ and that:

$$\mathbf{W}(\mathbf{x};\mathbf{p}) \circ \mathbf{W}(\mathbf{x};\Delta\mathbf{p}) \;=\; \mathbf{W}(\mathbf{W}(\mathbf{x};\Delta\mathbf{p});\mathbf{p}). \tag{86}$$

Combining these last two equations, and applying the Taylor expansion again, gives the update in the forwards compositional formulation as:

$$\mathbf{W}(\mathbf{x};\mathbf{p}) \;\leftarrow\; \mathbf{W}(\mathbf{x};\mathbf{p}) + \frac{\partial \mathbf{W}}{\partial \mathbf{x}} \frac{\partial \mathbf{W}}{\partial \mathbf{p}} \Delta\mathbf{p} \tag{87}$$

to first order in $\Delta\mathbf{p}$. The difference between the forwards additive formulation in Equations (82) and (83), and the forwards compositional formulation in Equations (84) and (87) is that $\frac{\partial \mathbf{W}}{\partial \mathbf{p}}$ is replaced by $\frac{\partial \mathbf{W}}{\partial \mathbf{x}} \frac{\partial \mathbf{W}}{\partial \mathbf{p}}$. Equations (82) and (84) therefore generally result in different estimates for $\Delta\mathbf{p}$. (Note that in the second of these expressions $\frac{\partial \mathbf{W}}{\partial \mathbf{p}}$ is evaluated at $(\mathbf{x};\mathbf{0})$, rather than at $(\mathbf{x};\mathbf{p})$.)

If the vectors $\frac{\partial \mathbf{W}}{\partial \mathbf{p}}$ in the forwards additive formulation and $\frac{\partial \mathbf{W}}{\partial \mathbf{x}}\frac{\partial \mathbf{W}}{\partial \mathbf{p}}$ in the forwards compositional formulation both span the same linear space, however, the final updates to the warp in Equations (83) and (87) will be the same to first order in $\Delta\mathbf{p}$ and the two formulations are equivalent; i.e. the optimal value of $\frac{\partial \mathbf{W}}{\partial \mathbf{p}}\Delta\mathbf{p}$ in Equation (82) will approximately equal the optimal value of $\frac{\partial \mathbf{W}}{\partial \mathbf{x}}\frac{\partial \mathbf{W}}{\partial \mathbf{p}}\Delta\mathbf{p}$ in Equation (84). From Equation (83) we see that the first of these expressions:

$$\frac{\partial \mathbf{W}}{\partial \mathbf{p}} \;=\; \frac{\partial \mathbf{W}(\mathbf{x};\mathbf{p}+\Delta\mathbf{p})}{\partial \Delta\mathbf{p}} \tag{88}$$

and from Equation (87) we see that the second of these expressions:

$$\frac{\partial \mathbf{W}}{\partial \mathbf{x}}\frac{\partial \mathbf{W}}{\partial \mathbf{p}} \;=\; \frac{\partial \mathbf{W}(\mathbf{x};\mathbf{p}) \circ \mathbf{W}(\mathbf{x};\mathbf{p}+\Delta\mathbf{p})}{\partial \Delta\mathbf{p}}. \tag{89}$$

The vectors $\frac{\partial \mathbf{W}}{\partial \mathbf{p}}$ in the forwards additive formulation and $\frac{\partial \mathbf{W}}{\partial \mathbf{x}}\frac{\partial \mathbf{W}}{\partial \mathbf{p}}$ in the forwards compositional formulation therefore span the same linear space, the tangent space of the manifold $\mathbf{W}(\mathbf{x};\mathbf{p})$, if (there is an $\epsilon > 0$ such that) for any $\Delta\mathbf{p}$ ($||\Delta\mathbf{p}|| \leq \epsilon$) there is a $\Delta\mathbf{p}'$ such that:

$$\mathbf{W}(\mathbf{x};\mathbf{p}+\Delta\mathbf{p}) \;=\; \mathbf{W}(\mathbf{x};\mathbf{p}) \circ \mathbf{W}(\mathbf{x};\mathbf{p}+\Delta\mathbf{p}'). \tag{90}$$

This condition means that the function between $\Delta\mathbf{p}$ and $\Delta\mathbf{p}'$ is defined in both directions. The expressions in Equations (88) and (89) therefore span the same linear space. If the warp is invertible Equation (90) always holds since $\Delta\mathbf{p}'$ can be chosen such that:

$$\mathbf{W}(\mathbf{x};\mathbf{p}+\Delta\mathbf{p}') \;=\; \mathbf{W}(\mathbf{x};\mathbf{p})^{-1} \circ \mathbf{W}(\mathbf{x};\mathbf{p}+\Delta\mathbf{p}). \tag{91}$$

If the warps are invertible then the two formulations are equivalent. In [2] we stated that the set of warps must form a semi-group for the forwards compositional algorithm to be applied. While this is true, for the forwards compositional algorithm also to be provably equivalent to the forwards additive algorithm, the set of warps must form a group; i.e. every warp must be invertible.

A.1.2 Equivalence of Forwards and Inverse Compositional Algorithms

We now show that the forwards and inverse and compositional approaches are equivalent. The proof of equivalence here takes a very different form to the proof in Appendix A.1.1. The first step

is to note that the summations in Equations (80) and (81) are discrete approximations to integrals. Equation (80) is the discrete version of:

$$\int_T \int_T Q(\mathbf{x}, \mathbf{y}) \cdot [\, I(\mathbf{W}(\mathbf{W}(\mathbf{x}; \Delta\mathbf{p}); \mathbf{p})) - T(\mathbf{x}) \,] \cdot [\, I(\mathbf{W}(\mathbf{W}(\mathbf{x}; \Delta\mathbf{p}); \mathbf{p})) - T(\mathbf{y}) \,] \, \mathrm{d}\mathbf{x} \, \mathrm{d}\mathbf{y} \quad (92)$$

where the integrations are both performed over the template T. Setting $\mathbf{x}' = \mathbf{W}(\mathbf{x}; \Delta\mathbf{p})$ (or equivalently $\mathbf{x} = \mathbf{W}(\mathbf{x}'; \Delta\mathbf{p})^{-1}$) and $\mathbf{y}' = \mathbf{W}(\mathbf{y}; \Delta\mathbf{p})$ (or equivalently $\mathbf{y} = \mathbf{W}(\mathbf{y}'; \Delta\mathbf{p})^{-1}$), and changing variables, Equation (92) becomes:

$$\int_{\mathbf{W}(T)} \int_{\mathbf{W}(T)} \Big[\, I(\mathbf{W}(\mathbf{x}'; \mathbf{p})) - T(\mathbf{W}(\mathbf{x}'; \Delta\mathbf{p})^{-1}) \,\Big] \cdot \Big[\, I(\mathbf{W}(\mathbf{y}'; \mathbf{p})) - T(\mathbf{W}(\mathbf{y}'; \Delta\mathbf{p})^{-1}) \Big] \cdot \left| \frac{\partial \mathbf{W}^{-1}}{\partial \mathbf{x}'} \right| \cdot \left| \frac{\partial \mathbf{W}^{-1}}{\partial \mathbf{y}'} \right| \mathrm{d}\mathbf{x}' \, \mathrm{d}\mathbf{y}' \quad (93)$$

where the integration is now performed over the image of T under the warp $\mathbf{W}(\mathbf{x}; \Delta\mathbf{p})$ which we denote: $\mathbf{W}(T) = \{\mathbf{W}(\mathbf{x}; \Delta\mathbf{p}) \,|\, \mathbf{x} \in T\}$. Since $\mathbf{W}(\mathbf{x}; \mathbf{0})$ is the identity warp, it follows that:

$$\left| \frac{\partial \mathbf{W}^{-1}}{\partial \mathbf{x}'} \right| = 1 + O(\Delta\mathbf{p}) \quad \text{and} \quad \left| \frac{\partial \mathbf{W}^{-1}}{\partial \mathbf{y}'} \right| = 1 + O(\Delta\mathbf{p}). \quad (94)$$

The integration domain $\mathbf{W}(T)$ is equal to $T = \{\mathbf{W}(\mathbf{x}; \mathbf{0}) \,|\, \mathbf{x} \in T\}$ to a zeroth order approximation also. Since we are ignoring higher order terms in $\Delta\mathbf{p}$, Equation (93) simplifies to:

$$\int_T \int_T \Big[\, T(\mathbf{W}(\mathbf{x}'; \Delta\mathbf{p})^{-1}) - I(\mathbf{W}(\mathbf{x}'; \mathbf{p})) \,\Big] \cdot \Big[\, T(\mathbf{W}(\mathbf{y}'; \Delta\mathbf{p})^{-1}) - I(\mathbf{W}(\mathbf{y}'; \mathbf{p})) \Big] \, \mathrm{d}\mathbf{x}' \, \mathrm{d}\mathbf{y}' \quad (95)$$

where we have assumed that $T(\mathbf{W}(\mathbf{x}'; \Delta\mathbf{p})^{-1}) - I(\mathbf{W}(\mathbf{x}'; \mathbf{p}))$ and $T(\mathbf{W}(\mathbf{y}'; \Delta\mathbf{p})^{-1}) - I(\mathbf{W}(\mathbf{y}'; \mathbf{p}))$ are both $O(\Delta\mathbf{p})$. (This assumption is equivalent to the assumption made in [9] that the current estimate of the parameters is approximately correct.) The first order terms in the Jacobian and the area of integration can therefore be ignored. Equation (95) is the continuous version of Equation (81) except that the terms $\mathbf{W}(\mathbf{x}; \Delta\mathbf{p})$ and $\mathbf{W}(\mathbf{y}; \Delta\mathbf{p})$ are inverted. The estimate of $\Delta\mathbf{p}$ that is computed by the inverse compositional algorithm using Equation (81) therefore gives an estimate of $\mathbf{W}(\mathbf{x}; \Delta\mathbf{p})$ that is the inverse of the incremental warp computed by the forwards compositional algorithm using Equation (80). Since the inverse compositional algorithm inverts $\mathbf{W}(\mathbf{x}; \Delta\mathbf{p})$ before composing it with $\mathbf{W}(\mathbf{x}; \mathbf{p})$, the two algorithms take the same steps to first order in $\Delta\mathbf{p}$.

A.2 Robust Error Functions

The goal of image alignment with a robust error functions is to minimize:

$$\sum_{\mathbf{x}} \varrho\left(\left[I(\mathbf{W}(\mathbf{x};\mathbf{p})) - T(\mathbf{x}) \right]^2 \right) \tag{96}$$

with respect to the warp parameters $\mathbf{p}$. The *forwards additive* algorithm minimizes this expression by iteratively minimizing:

$$\sum_{\mathbf{x}} \varrho\left(\left[I(\mathbf{W}(\mathbf{x};\mathbf{p}) + \Delta\mathbf{p}) - T(\mathbf{x}) \right]^2 \right) \tag{97}$$

with respect to $\Delta\mathbf{p}$ and then updating the parameters $\mathbf{p} \leftarrow \mathbf{p} + \Delta\mathbf{p}$. The *forwards compositional* algorithm minimizes the same expression by iteratively minimizing:

$$\sum_{\mathbf{x}} \varrho\left(\left[I(\mathbf{W}(\mathbf{W}(\mathbf{x};\Delta\mathbf{p});\mathbf{p})) - T(\mathbf{x}) \right]^2 \right) \tag{98}$$

with respect to $\Delta\mathbf{p}$ and then updating the warp $\mathbf{W}(\mathbf{x};\mathbf{p}) \leftarrow \mathbf{W}(\mathbf{x};\mathbf{p}) \circ \mathbf{W}(\mathbf{x};\Delta\mathbf{p})$. Finally, the *inverse compositional* algorithm minimizes the same expression by iteratively minimizing:

$$\sum_{\mathbf{x}} \varrho\left(\left[T(\mathbf{W}(\mathbf{x};\Delta\mathbf{p})) - I(\mathbf{W}(\mathbf{x};\mathbf{p})) \right]^2 \right). \tag{99}$$

with respect to $\Delta\mathbf{p}$ and updating the warp: $\mathbf{W}(\mathbf{x};\mathbf{p}) \leftarrow \mathbf{W}(\mathbf{x};\mathbf{p}) \circ \mathbf{W}(\mathbf{x};\Delta\mathbf{p})^{-1}$. To show the equivalence of the forwards additive and inverse compositional formulations, we first show the equivalence of the forwards additive and forwards compositional formulations. Afterwards we show the equivalence of the forwards and inverse compositional formulations. The desired result follows by transitivity of equivalence.

A.2.1 Equivalence of Forwards Additive and Compositional Algorithms

The proof of equivalence of the forwards additive and compositional algorithms with a robust error function is almost exactly the same as the proof of equivalence of the corresponding algorithms with a weighted L2 norm. See Appendix A.1.1. The only difference in the argument is that the two equations used to estimate the updates to the parameters, Equations (82) and (84), must be replaced with their equivalents for the robust error function. The equivalent of Equation (82) is:

$$\sum_{\mathbf{x}} \varrho\left(\left[I(\mathbf{W}(\mathbf{x};\mathbf{p})) + \nabla I \frac{\partial \mathbf{W}}{\partial \mathbf{p}} \Delta\mathbf{p} - T(\mathbf{x}) \right]^2 \right) \tag{100}$$

and the equivalent of Equation (84) is:

$$\sum_{\mathbf{x}} \varrho \left(\left[I(\mathbf{W}(\mathbf{x};\mathbf{p})) + \boldsymbol{\nabla} I \frac{\partial \mathbf{W}}{\partial \mathbf{x}} \frac{\partial \mathbf{W}}{\partial \mathbf{p}} \Delta\mathbf{p} - T(\mathbf{x}) \right]^2 \right). \tag{101}$$

Otherwise the argument proceeds exactly as in Appendix A.1.1, but using Equation (100) in place of Equation (82) and Equation (101) in place of Equation (84).

A.2.2 Equivalence of Forwards and Inverse Compositional Algorithms

The proof of equivalence of the forwards and inverse compositional algorithms with a robust error function is almost exactly the same as the proof of equivalence of the corresponding algorithms with a weighted L2 norm. See Appendix A.1.2. The first step is to write down the integral version of Equation (98) which is:

$$\int_T \varrho \left([\, I(\mathbf{W}(\mathbf{W}(\mathbf{x};\Delta\mathbf{p});\mathbf{p})) - T(\mathbf{x}) \,]^2 \right) \mathrm{d}\mathbf{x}. \tag{102}$$

Setting $\mathbf{x}' = \mathbf{W}(\mathbf{x};\Delta\mathbf{p})$, or equivalently $\mathbf{x} = \mathbf{W}(\mathbf{x}';\Delta\mathbf{p})^{-1}$, and changing variables, Equation (102) becomes:

$$\int_{\mathbf{W}(T)} \varrho \left(\left[I(\mathbf{W}(\mathbf{x}';\mathbf{p})) - T(\mathbf{W}(\mathbf{x}';\Delta\mathbf{p})^{-1}) \right]^2 \right) \left| \frac{\partial \mathbf{W}^{-1}}{\partial \mathbf{x}} \right| \mathrm{d}\mathbf{x}' \tag{103}$$

where the integration is now performed over the image of T under the warp $\mathbf{W}(\mathbf{x};\Delta\mathbf{p})$ which we denote: $\mathbf{W}(T) = \{\mathbf{W}(\mathbf{x};\Delta\mathbf{p}) \,|\, \mathbf{x} \in T\}$. Since $\mathbf{W}(\mathbf{x};\mathbf{0})$ is the identity warp, it follows that:

$$\left| \frac{\partial \mathbf{W}^{-1}}{\partial \mathbf{x}'} \right| = 1 + O(\Delta\mathbf{p}). \tag{104}$$

The integration domain $\mathbf{W}(T)$ is equal to $T = \{\mathbf{W}(\mathbf{x};\mathbf{0}) \,|\, \mathbf{x} \in T\}$ to a zeroth order approximation also. Since we are ignoring higher order terms in $\Delta\mathbf{p}$, Equation (103) simplifies to:

$$\int_T \varrho \left(\left[T(\mathbf{W}(\mathbf{x}';\Delta\mathbf{p})^{-1}) - I(\mathbf{W}(\mathbf{x}';\mathbf{p})) \right]^2 \right) \mathrm{d}\mathbf{x}'. \tag{105}$$

In making this simplification we have assumed that $T(\mathbf{W}(\mathbf{x}';\Delta\mathbf{p})^{-1}) - I(\mathbf{W}(\mathbf{x}';\mathbf{p}))$ is $O(\Delta\mathbf{p})$. (This assumption is equivalent to the assumption made in [9] that the current estimate of the parameters is approximately correct.) The first order terms in the Jacobian and the area of integration

can therefore be ignored. Equation (105) is the continuous version of Equation (99) except that the term $\mathbf{W}(\mathbf{x}; \Delta\mathbf{p})$ is inverted. The estimate of $\Delta\mathbf{p}$ that is computed by the inverse compositional algorithm using Equation (99) therefore gives an estimate of $\mathbf{W}(\mathbf{x}; \Delta\mathbf{p})$ that is the inverse of the incremental warp computed by the forwards compositional algorithm using Equation (98). Since the inverse compositional algorithm inverts $\mathbf{W}(\mathbf{x}; \Delta\mathbf{p})$ before composing it with $\mathbf{W}(\mathbf{x}; \mathbf{p})$, the two algorithms take the same steps to first order in $\Delta\mathbf{p}$.