Lucas-Kanade 20 Years On: A Unifying Framework: Part 3

Simon Baker, Ralph Gross, and Iain Matthews

CMU-RI-TR-03-35

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 variety of extensions have been made to the original formulation. We present an overview of image alignment, describing most of the algorithms 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 cannot. In this paper, Part 3 in a series of papers, we cover the extension of image alignment to allow linear appearance variation. We first consider linear appearance variation when the error function is the Euclidean L2 norm. We describe three different algorithms, the simultaneous, project out, and normalization inverse compositional algorithms, and empirically compare them. Afterwards we consider the combination of linear appearance variation with the robust error functions described in Part 2 of this series. We first derive robust versions of the simultaneous and normalization algorithms. Since both of these algorithms are very inefficient, as in Part 2 we derive efficient approximations based on spatial coherence. We end with an empirical evaluation of the robust algorithms.

Keywords: Image alignment, unifying framework, the Lucas-Kanade algorithm, the inverse compositional algorithm, linear appearance variation, robust error functions.

1 Introduction

Image alignment consists of moving, and possibly deforming, a template to minimize the difference between the template and an image. Since its first use in the Lucas-Kanade algorithm [13], image alignment has become one of the most widely used techniques in computer vision. Besides optical flow, some of its other applications include tracking [5, 11], parametric and layered motion estimation [4], mosaic construction [15], medical image registration [6], and face coding [14, 7].

The usual approach to image alignment is gradient descent. A variety of other numerical algorithms have also been proposed [10], 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 [2, 3]. 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 3 in the series, we cover image alignment with linear appearance variation. Linear appearance variation has been considered by a number of authors, most notably by Hager and Belhumeur for illumination [11], by Black and Jepson for general appearance variation [5], and by Cootes and Taylor for non-rigid face modeling [7]. As in Part 2, we distinguish two cases: (1) when the error function is the Euclidean L2 norm (the case that the error function is a general weighted L2 norm is similar), and (2) when the error function is a robust error function.

We consider the Euclidean case in Section 3. We first derive the "simultaneous" inverse compositional algorithm which, as the name implies, performs a simultaneous optimization over the warp and appearance parameters. We then derive an efficient approximation to the simultaneous inverse compositional algorithm and also describe the extremely efficient "project out" algorithm proposed by Hager and Belhumeur [11]. The project out algorithm first projects out the appearance variation and just solves for the warp parameters. Then in a second step, it solves for the

appearance parameters. We study the project out algorithm in the case that the appearance variation contains a "gain" term, show that the step size can be computed incorrectly, and propose a way of correcting the error. We also describe the "normalization" algorithm which attempts to normalize the input image so that it has the same appearance component as the template. A variant of the normalization algorithm is frequently used when the appearance variation consists of gain and bias. We end Section 3 by empirically comparing all three Euclidean algorithms and their variants.

We consider the robust case in Section 4. We first derive robust versions of the simultaneous and normalization algorithms. Since both of these algorithms are inefficient, as in Part 2 we derive efficient approximations based on spatial coherence of the outliers. It is not possible to directly generalize the project out algorithm because there is no notion of orthogonality for a robust error function. We end with an empirical evaluation of the robust appearance variation algorithms.

2 Background: Image Alignment Algorithms

2.1 The Lucas-Kanade Algorithm

The original image alignment algorithm was the Lucas-Kanade algorithm [13]. The goal of the Lucas-Kanade algorithm 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. 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 template T and maps it to the sub-pixel location $\mathbf{W}(\mathbf{x};\mathbf{p})$ in the image I.

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:

$$\sum_{\mathbf{x}} [\, T(\mathbf{x}) - I(\mathbf{W}(\mathbf{x};\mathbf{p})) \,]^2 \,. \tag{1}$$

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 (1) 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 (1) is a non-linear optimization task 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 (1), 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:

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

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

$$\mathbf{p} \;\leftarrow\; \mathbf{p} + \Delta\mathbf{p}. \tag{3}$$

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 (2) is linearized by performing a first order Taylor expansion on $I(\mathbf{W}(\mathbf{x};\mathbf{p}+\Delta\mathbf{p}))$ to give:

$$\sum_{\mathbf{x}} \left[T(\mathbf{x}) - I(\mathbf{W}(\mathbf{x};\mathbf{p})) - \boldsymbol{\nabla} I \frac{\partial \mathbf{W}}{\partial \mathbf{p}} \Delta\mathbf{p} \right]^2. \tag{4}$$

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 (4).) 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:

$$\frac{\partial \mathbf{W}}{\partial \mathbf{p}} \;=\; \begin{pmatrix} \frac{\partial W_x}{\partial p_1} & \frac{\partial W_x}{\partial p_2} & \cdots & \frac{\partial W_x}{\partial p_n} \\ \frac{\partial W_y}{\partial p_1} & \frac{\partial W_y}{\partial p_2} & \cdots & \frac{\partial W_y}{\partial p_n} \end{pmatrix}. \tag{5}$$

Equation (4) 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 (4) with respect to $\Delta\mathbf{p}$ is:

$$\sum_{\mathbf{x}} \left[\boldsymbol{\nabla} I \frac{\partial \mathbf{W}}{\partial \mathbf{p}} \right]^{\mathrm{T}} \left[T(\mathbf{x}) - I(\mathbf{W}(\mathbf{x};\mathbf{p})) - \boldsymbol{\nabla} I \frac{\partial \mathbf{W}}{\partial \mathbf{p}} \Delta\mathbf{p} \right]. \tag{6}$$

Then denote:

$$\mathbf{SD}(\mathbf{x}) \;=\; \boldsymbol{\nabla} I \frac{\partial \mathbf{W}}{\partial \mathbf{p}}, \tag{7}$$

the *steepest descent* images. Setting the expression in Equation (6) to equal zero and solving gives the closed form solution of Equation (4) as:

$$\Delta\mathbf{p} \;=\; H^{-1} \sum_{\mathbf{x}} \mathbf{SD}^{\mathrm{T}}(\mathbf{x})\, E(\mathbf{x}) \tag{8}$$

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

$$H \;=\; \sum_{\mathbf{x}} \mathbf{SD}^{\mathrm{T}}(\mathbf{x})\, \mathbf{SD}(\mathbf{x}) \tag{9}$$

and:

$$E(\mathbf{x}) \;=\; T(\mathbf{x}) - I(\mathbf{W}(\mathbf{x};\mathbf{p})) \tag{10}$$

is the *error image*. The Lucas-Kanade algorithm, summarized in Figure 1, consists of iteratively applying Equations (8) and (3). Because the gradient $\boldsymbol{\nabla} I$ 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 depend on $\mathbf{p}$. In general, therefore, both the steepest-descent images and the Hessian must be recomputed in every iteration of the algorithm. See Figure 1.

Assume that the number of warp parameters is n and the number of pixels in T is N. The total computational cost of each iteration of the Lucas-Kanade algorithm is $\mathrm{O}(n^2\, N + n^3)$. The most expensive step by far is Step 6. See Table 1 for a summary and [3] for the details.

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 $E(\mathbf{x})$ using Equation (10)
(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 $\mathbf{SD}(\mathbf{x})$ using Equation (7)
(6) Compute the Hessian matrix H using Equation (9) and invert it
(7) Compute $\sum_{\mathbf{x}} \mathbf{SD}^{\mathrm{T}}(\mathbf{x}) E(\mathbf{x})$
(8) Compute $\Delta\mathbf{p} = H^{-1} \sum_{\mathbf{x}} \mathbf{SD}^{\mathrm{T}}(\mathbf{x}) E(\mathbf{x})$
(9) Update the parameters $\mathbf{p} \leftarrow \mathbf{p} + \Delta\mathbf{p}$

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

Figure 1: The Lucas-Kanade algorithm [13] consists of iteratively applying Equations (8) & (3) 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 specified 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 computation 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 $\mathrm{O}(n^2\,N + N^3)$.

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

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 [11, 8, 15]. If the Hessian were constant it could be precomputed and then re-used. In [3] 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 (see Equation (1)) the inverse compositional algorithm iteratively minimizes:

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

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

$$\mathbf{W}(\mathbf{x};\mathbf{p}) \leftarrow \mathbf{W}(\mathbf{x};\mathbf{p}) \circ \mathbf{W}(\mathbf{x};\Delta\mathbf{p})^{-1}. \tag{12}$$

The expression:

$$\mathbf{W}(\mathbf{x};\mathbf{p}) \circ \mathbf{W}(\mathbf{x};\Delta\mathbf{p}) \equiv \mathbf{W}(\mathbf{W}(\mathbf{x};\Delta\mathbf{p});\mathbf{p}) \tag{13}$$

is the composition of 2 warps and the expression $\mathbf{W}(\mathbf{x};\Delta\mathbf{p})^{-1}$ is the inverse of $\mathbf{W}(\mathbf{x};\Delta\mathbf{p})$.

The Lucas-Kanade algorithm iteratively applies Equations (2) and (3). The inverse compositional algorithm iteratively applies Equations (11) and (12). Perhaps somewhat surprisingly, these two algorithms can be shown to be equivalent to first order in $\Delta\mathbf{p}$. They take (approximately) the same steps as they minimize the expression in Equation (1). See [3] for the proof of equivalence.

2.2.2 Derivation of the Inverse Compositional Algorithm

Performing a first order Taylor expansion on Equation (11) gives:

$$\sum_{\mathbf{x}} \left[T(\mathbf{W}(\mathbf{x};\mathbf{0})) + \nabla T \frac{\partial \mathbf{W}}{\partial \mathbf{p}} \Delta\mathbf{p} - I(\mathbf{W}(\mathbf{x};\mathbf{p})) \right]^2. \tag{14}$$

Assuming that $\mathbf{W}(\mathbf{x};\mathbf{0})$ is the identity warp, the solution to this least-squares problem is:

$$\Delta\mathbf{p} = -H_{\text{ic}}^{-1} \sum_{\mathbf{x}} \mathbf{SD}_{\text{ic}}^{\text{T}}(\mathbf{x})\, E(\mathbf{x}) \tag{15}$$

where $\mathbf{SD}_{\text{ic}}^{\text{T}}(\mathbf{x})$ are the steepest-descent images with I replaced by T:

$$\mathbf{SD}_{\text{ic}}(\mathbf{x}) = \nabla T \frac{\partial \mathbf{W}}{\partial \mathbf{p}}, \tag{16}$$

H_{ic} is the Hessian matrix computed using the new steepest-descent images:

$$H_{\text{ic}} = \sum_{\mathbf{x}} \mathbf{SD}_{\text{ic}}^{\text{T}}(\mathbf{x})\, \mathbf{SD}_{\text{ic}}(\mathbf{x}), \tag{17}$$

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 $\mathbf{SD}_{\text{ic}}(\mathbf{x})$ using Equation (16)
(6) Compute the Hessian matrix H_{ic} using Equation (17) and invert it

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 $E(\mathbf{x})$ using Equation (10)
(7) Compute $\sum_{\mathbf{x}} \mathbf{SD}^{\text{T}}_{\text{ic}}(\mathbf{x}) E(\mathbf{x})$
(8) Compute $\Delta \mathbf{p} = -H^{-1}_{\text{ic}} \sum_{\mathbf{x}} \mathbf{SD}^{\text{T}}_{\text{ic}}(\mathbf{x}) E(\mathbf{x})$
(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 [2, 3]. 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 efficient and take time $\text{O}(n\,N + n^2)$.

and the Jacobian $\frac{\partial \mathbf{W}}{\partial \mathbf{p}}$ is evaluated at $(\mathbf{x}; \mathbf{0})$. Since there is nothing in either the steepest-descent images or the Hessian that depends on $\mathbf{p}$, they can both be pre-computed. The inverse compositional algorithm is summarized in Figure 2. (See [1] for a schematic diagram of the 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 taking time $\text{O}(n^2\,N + n^3)$. 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 $\text{O}(n^2)$ operations. See [3]. Potentially these 2 steps could be fairly involved, as in [14], but the computational overhead is almost always completely negligible. Overall the cost of the inverse compositional algorithm is $\text{O}(n\,N + n^2)$ per iteration rather than $\text{O}(n^2\,N + n^3)$, a substantial saving.

3 Linear Appearance Variation with the Euclidean L2 Norm

All of the algorithms in [3] (there are 9 of them) aim to minimize the expression in Equation (1). Performing this minimization implicitly assumes that the template $T(\mathbf{x})$ appears in the input image

Table 2: The computation 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
	$\mathrm{O}(n\,N)$	$\mathrm{O}(N)$	$\mathrm{O}(n\,N)$	$\mathrm{O}(n^2)$	$\mathrm{O}(n^2)$	$\mathrm{O}(n\,N + n^2)$

$I(\mathbf{x})$, albeit warped by $\mathbf{W}(\mathbf{x};\mathbf{p})$. In various scenarios we may instead want to assume that:

$$T(\mathbf{x}) \; + \; \sum_{i=1}^{m} \lambda_i A_i(\mathbf{x}) \tag{18}$$

appears in the input image (warped appropriately) where A_i, $i = 1, \ldots, m$, is a set of known appearance variation images and λ_i, $i = 1, \ldots, m$, is 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 appropriate values of λ_1 and λ_2, the expression in Equation (18) can then model any possible gain and bias. More generally, the appearance images A_i can be used to model arbitrary linear illumination variation [11] or general appearance variation [5, 14]. If the expression in Equation (18) should appear (appropriately warped) in the input image $I(\mathbf{x})$, instead of Equation (1) we should minimize:

$$\sum_{\mathbf{x}} \left[T(\mathbf{x}) + \sum_{i=1}^{m} \lambda_i A_i(\mathbf{x}) - I(\mathbf{W}(\mathbf{x};\mathbf{p})) \right]^2 \tag{19}$$

simultaneously with respect to the warp and appearance parameters, $\mathbf{p}$ and $\boldsymbol{\lambda} = (\lambda_1, \ldots, \lambda_m)^{\mathrm{T}}$. In the remainder of this section we derive three different algorithms (and several variants of them) to minimize the expression in Equation (19), before evaluating all of the algorithms in Section 3.4.

3.1 The Simultaneous Inverse Compositional Algorithm

3.1.1 Goal of the Algorithm

Our first algorithm performs Gauss-Newton gradient descent simultaneously on the warp $\mathbf{p}$ and appearance $\boldsymbol{\lambda}$ parameters. We use the inverse compositional parameter update on the warp parameters. The appearance parameters are updated with the usual additive algorithm. Composition does not have any meaning for them. Replacing $T(\mathbf{x})$ with $T(\mathbf{x}) + \sum_{i=1}^{m} \lambda_i A_i(\mathbf{x})$ in Equation (11), the inverse compositional algorithm to minimize Equation (19) operates by iteratively minimizing:

$$\sum_{\mathbf{x}} \left[T(\mathbf{W}(\mathbf{x};\Delta\mathbf{p})) + \sum_{i=1}^{m} (\lambda_i + \Delta\lambda_i) A_i(\mathbf{W}(\mathbf{x};\Delta\mathbf{p})) - I(\mathbf{W}(\mathbf{x};\mathbf{p})) \right]^2 \tag{20}$$

simultaneously with respect to $\Delta\mathbf{p}$ and $\Delta\boldsymbol{\lambda} = (\Delta\lambda_1, \ldots, \Delta\lambda_m)^{\mathrm{T}}$, and then updating the warp $\mathbf{W}(\mathbf{x};\mathbf{p}) \leftarrow \mathbf{W}(\mathbf{x};\mathbf{p}) \circ \mathbf{W}(\mathbf{x};\Delta\mathbf{p})^{-1}$ and the appearance parameters $\boldsymbol{\lambda} \leftarrow \boldsymbol{\lambda} + \Delta\boldsymbol{\lambda}$.

3.1.2 Derivation of the Algorithm

Performing a first order Taylor expansion on $T(\mathbf{W}(\mathbf{x};\Delta\mathbf{p}))$ and $A_i(\mathbf{W}(\mathbf{x};\Delta\mathbf{p}))$ in Equation (20), and assuming as in Section 2.2.2 that $\mathbf{W}(\mathbf{x};\mathbf{0})$ is the identity warp, gives:

$$\sum_{\mathbf{x}} \left[T(\mathbf{x}) + \boldsymbol{\nabla} T \frac{\partial \mathbf{W}}{\partial \mathbf{p}} \Delta\mathbf{p} + \sum_{i=1}^{m} (\lambda_i + \Delta\lambda_i) \left(A_i(\mathbf{x}) + \boldsymbol{\nabla} A_i \frac{\partial \mathbf{W}}{\partial \mathbf{p}} \Delta\mathbf{p} \right) - I(\mathbf{W}(\mathbf{x};\mathbf{p})) \right]^2 . \tag{21}$$

Neglecting second order terms, the above expression simplifies to:

$$\sum_{\mathbf{x}} \left[T(\mathbf{x}) + \sum_{i=1}^{m} \lambda_i A_i(\mathbf{x}) - I(\mathbf{W}(\mathbf{x};\mathbf{p})) + \left(\boldsymbol{\nabla} T + \sum_{i=1}^{m} \lambda_i \boldsymbol{\nabla} A_i \right) \frac{\partial \mathbf{W}}{\partial \mathbf{p}} \Delta\mathbf{p} + \sum_{i=1}^{m} A_i(\mathbf{x}) \Delta\lambda_i \right]^2 . \tag{22}$$

To simplify the notation, denote:

$$\mathbf{q} = \begin{pmatrix} \mathbf{p} \\ \boldsymbol{\lambda} \end{pmatrix} \quad \text{and similarly} \quad \Delta\mathbf{q} = \begin{pmatrix} \Delta\mathbf{p} \\ \Delta\boldsymbol{\lambda} \end{pmatrix} ; \tag{23}$$

i.e. $\mathbf{q}$ is an $n+m$ dimensional column vector containing the warp parameters $\mathbf{p}$ concatenated with the appearance parameters $\boldsymbol{\lambda}$. Similarly, denote the $n+m$ dimensional steepest-descent images:

$$\mathbf{SD}_{\text{sim}}(\mathbf{x}) \;=\; \left(\left(\boldsymbol{\nabla} T + \sum_{i=1}^{m} \lambda_i \boldsymbol{\nabla} A_i \right) \frac{\partial \mathbf{W}}{\partial p_1}, \ldots, \left(\boldsymbol{\nabla} T + \sum_{i=1}^{m} \lambda_i \boldsymbol{\nabla} A_i \right) \frac{\partial \mathbf{W}}{\partial p_n}, A_1(\mathbf{x}), \ldots, A_m(\mathbf{x}) \right). \tag{24}$$

Finally, denote the modified error image:

$$E_{\text{sim}}(\mathbf{x}) \;=\; T(\mathbf{x}) + \sum_{i=1}^{m} \lambda_i A_i(\mathbf{x}) - I(\mathbf{W}(\mathbf{x};\mathbf{p})). \tag{25}$$

Equation (22) then simplifies to:

$$\sum_{\mathbf{x}} \left[\, E_{\text{sim}}(\mathbf{x}) - \mathbf{SD}_{\text{sim}}(\mathbf{x}) \, \Delta\mathbf{q} \, \right]^2 \tag{26}$$

the minimum of which is attained at:

$$\Delta\mathbf{q} \;=\; -H_{\text{sim}}^{-1} \sum_{\mathbf{x}} \mathbf{SD}_{\text{sim}}^{\text{T}}(\mathbf{x}) E_{\text{sim}}(\mathbf{x}) \tag{27}$$

where H_{sim}^{-1} is the Hessian with appearance variation:

$$H_{\text{sim}}^{-1} \;=\; \sum_{\mathbf{x}} \mathbf{SD}_{\text{sim}}^{\text{T}}(\mathbf{x}) \, \mathbf{SD}_{\text{sim}}(\mathbf{x}). \tag{28}$$

In summary, the simultaneous inverse compositional algorithm for appearance variation proceeds by iteratively applying Equations (24), (25), (27), and (28) to compute $\Delta\mathbf{q}$. The incremental updates to the warp $\Delta\mathbf{p}$ and appearance $\Delta\boldsymbol{\lambda}$ parameters are then extracted from $\Delta\mathbf{q}$ and used to update the warp $\mathbf{W}(\mathbf{x};\mathbf{p}) \leftarrow \mathbf{W}(\mathbf{x};\mathbf{p}) \circ \mathbf{W}(\mathbf{x};\Delta\mathbf{p})^{-1}$ and the appearance parameters $\boldsymbol{\lambda} \leftarrow \boldsymbol{\lambda} + \Delta\boldsymbol{\lambda}$. Unfortunately the steepest descent images depend on the (appearance) parameters $\Delta\boldsymbol{\lambda}$ and so must be re-computed in every iteration. The result is the algorithm summarized in Figure 3. Overall the algorithm is even slower than the original Lucas-Kanade algorithm because the computational cost of most of the steps depends on the total number of parameters $n+m$ rather than just the number of warp parameters n. See Table 3 for a summary of the computation cost.

The Simultaneous Inverse Compositional Algorithm

Pre-compute:

(3) Evaluate the gradients ∇T and ∇A_i for $i = 1, \ldots, m$

(4) Evaluate the Jacobian $\frac{\partial \mathbf{W}}{\partial \mathbf{p}}$ at $(\mathbf{x}; \mathbf{0})$

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 $E_{\text{sim}}(\mathbf{x})$ using Equation (25)

(5) Compute the steepest descent images $\mathbf{SD}_{\text{sim}}(\mathbf{x})$ using Equation (24)

(6) Compute the Hessian matrix H_{sim} using Equation (28) and invert it

(7) Compute $\sum_{\mathbf{x}} \mathbf{SD}_{\text{sim}}^{\text{T}}(\mathbf{x}) E_{\text{sim}}(\mathbf{x})$

(8) Compute $\Delta \mathbf{q} = -H_{\text{sim}}^{-1} \sum_{\mathbf{x}} \mathbf{SD}_{\text{sim}}^{\text{T}}(\mathbf{x}) E_{\text{sim}}(\mathbf{x})$

(9) Update $\mathbf{W}(\mathbf{x}; \mathbf{p}) \leftarrow \mathbf{W}(\mathbf{x}; \mathbf{p}) \circ \mathbf{W}(\mathbf{x}; \Delta \mathbf{p})^{-1}$ and $\boldsymbol{\lambda} \leftarrow \boldsymbol{\lambda} + \Delta \boldsymbol{\lambda}$

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

Figure 3: The simultaneous inverse compositional algorithm for appearance variation operates by iteratively applying Equations (24), (25), (27), and (28) to compute $\Delta \mathbf{q}$. The incremental updates to the warp $\Delta \mathbf{p}$ and appearance $\Delta \boldsymbol{\lambda}$ parameters are then extracted from $\Delta \mathbf{q}$ and used to update the parameters in Step 9. Because the steepest descent images depend on the appearance parameters (see Equation (24)), Steps (5) and (6) must be performed in every iteration. See Table 3 for a summary of the computational cost.

Table 3: The computation cost of the simultaneous inverse compositional algorithm. Overall the algorithm is even slower than the Lucas-Kanade algorithm because the computational cost of most of the steps depends on the total number of parameters $n + m$ rather than just the number of warp parameters n.

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

Per Iteration	Step 1	Step 2	Step 5	Step 6
	$O(n\,N)$	$O(m\,N)$	$O((n+m)\,N)$	$O((n+m)^2\,N + (n+m)^3)$

Step 7	Step 8	Step 9	Total
$O((n+m)\,N)$	$O((n+m)^2)$	$O(n^2+m)$	$O((n+m)^2\,N + (n+m)^3)$

3.1.3 An Efficient Approximation

The main reason the simultaneous inverse compositional algorithm is so slow is because the steepest descent images depend on the appearance parameters. See Equation (24). One possible approximation to the algorithm is to assume that the appearance parameters do not vary significantly. The steepest descent images are only computed once using the initial estimates of the appearance

parameters. Afterwards they are never updated. As a result Steps (5) and (6) can be moved to pre-computation. The one time pre-computation cost therefore becomes $\mathrm{O}((n+m)^2 N + (n+m)^3)$ and the cost per iteration becomes $\mathrm{O}((n+m) N + (n+m)^2)$. This approximation results in a huge performance increase. However, the result is still not as efficient as the inverse compositional algorithm without appearance variation because the computational cost of most of the steps still depends on the total number of parameters $n+m$ rather than just the number of warp parameters n. In Section 3.4 we empirically investigate the extent to which this efficiency approximation reduces the robustness and speed of convergence of the simultaneous inverse compositional algorithm.

3.2 The Project Out Inverse Compositional Algorithm

3.2.1 Goal of the Algorithm

Although the optimization in Equation (19) is non-linear with respect to the warp parameters $\mathbf{p}$, it is linear with respect to the appearance parameters $\boldsymbol{\lambda}$. In [11], Hager and Belhumeur proposed a way of decomposing a very similar optimization into two steps. The first step is a non-linear optimization with respect to the warp parameters, but performed in a subspace in which the appearance variation can be ignored. The second step is a closed form linear optimization with respect to the appearance parameters. We refer to this type of algorithm as a "project out" algorithm because in the first step the appearance variation is "projected out." We now derive the equivalent project out inverse compositional algorithm. (Hager and Belhumeur [11] used an algorithm that is less general than the inverse compositional algorithm [3].) We also point out one failing of the project out algorithm, namely the estimation of the step size, and propose a method of correcting it.

3.2.2 Derivation of the Algorithm

If we treat the images as vectors over the pixels $\mathbf{x}$ we can rewrite Equation (19) as:

$$\sum_{\mathbf{x}} \left[T(\mathbf{x}) + \sum_{i=1}^{m} \lambda_i A_i(\mathbf{x}) - I(\mathbf{W}(\mathbf{x};\mathbf{p})) \right]^2 = \left\| T(\mathbf{x}) + \sum_{i=1}^{m} \lambda_i A_i(\mathbf{x}) - I(\mathbf{W}(\mathbf{x};\mathbf{p})) \right\|^2 \quad (29)$$

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 (29) can be rewritten as:

$$\left\| T(\mathbf{x}) + \sum_{i=1}^{m} \lambda_i A_i(\mathbf{x}) - I(\mathbf{W}(\mathbf{x};\mathbf{p})) \right\|^2_{\mathrm{span}(A_i)} + \left\| T(\mathbf{x}) + \sum_{i=1}^{m} \lambda_i A_i(\mathbf{x}) - I(\mathbf{W}(\mathbf{x};\mathbf{p})) \right\|^2_{\mathrm{span}(A_i)^\perp} \tag{30}$$

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:

$$\left\| T(\mathbf{x}) + \sum_{i=1}^{m} \lambda_i A_i(\mathbf{x}) - I(\mathbf{W}(\mathbf{x};\mathbf{p})) \right\|^2_{\mathrm{span}(A_i)} + \left\| T(\mathbf{x}) - I(\mathbf{W}(\mathbf{x};\mathbf{p})) \right\|^2_{\mathrm{span}(A_i)^\perp} . \tag{31}$$

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 exactly 0 because the term $\sum_{i=1}^{m} \lambda_i A_i(\mathbf{x})$ can represent any vector in $\mathrm{span}(A_i)$. As a result, 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 Gram-Schmidt) the minimization of the first term has the closed-form solution:

$$\lambda_i \;=\; \sum_{\mathbf{x}} A_i(\mathbf{x}) \; [I(\mathbf{W}(\mathbf{x};\mathbf{p})) - T(\mathbf{x})] \, . \tag{32}$$

The only difference between minimizing the second term in Equation (31) and the original goal of the Lucas-Kanade algorithm (see Equation (1)) 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 [1] with:

$$Q(\mathbf{x},\mathbf{y}) \;=\; \delta(\mathbf{x},\mathbf{y}) - \sum_{i=1}^{m} [A_i(\mathbf{x}) \cdot A_i(\mathbf{y})] \tag{33}$$

(assuming again that the vectors A_i are orthonormal) and minimizing:

$$\sum_{\mathbf{x}} \sum_{\mathbf{y}} Q(\mathbf{x}, \mathbf{y}) \cdot [\, T(\mathbf{x}) - I(\mathbf{W}(\mathbf{x}; \mathbf{p}))\,] \cdot [\, T(\mathbf{y}) - I(\mathbf{W}(\mathbf{y}; \mathbf{p}))\,]\,; \tag{34}$$

i.e. minimizing the second term of Equation (31) and minimizing the expression in Equations (33) and (34) are exactly the same thing. See [1] for the details. We can therefore use the inverse compositional algorithm with this weighted L2 norm (described in Part 2 [1]) to minimize the second term in Equation (31). The weighted steepest descent images are:

$$\mathbf{SD}_{\mathrm{po}}(\mathbf{x}) \;=\; \sum_{\mathbf{y}} \left[\delta(\mathbf{x}, \mathbf{y}) - \sum_{i=1}^{m} (A_i(\mathbf{x}) \cdot A_i(\mathbf{y})) \right] \left[\boldsymbol{\nabla} T \frac{\partial \mathbf{W}}{\partial \mathbf{p}}(\mathbf{y}; \mathbf{0}) \right] \tag{35}$$

(see Equation (29) of [1]) and so can be computed:

$$\mathbf{SD}_{\mathrm{po}}(\mathbf{x}) \;=\; \boldsymbol{\nabla} T \frac{\partial \mathbf{W}}{\partial \mathbf{p}}(\mathbf{x}; \mathbf{0}) \;-\; \sum_{i=1}^{m} \left[\sum_{\mathbf{y}} A_i(\mathbf{y})\, \boldsymbol{\nabla} T \frac{\partial \mathbf{W}}{\partial \mathbf{p}}(\mathbf{y}; \mathbf{0}) \right] A_i(\mathbf{x}) \tag{36}$$

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

$$H_{\mathrm{po}} \;=\; \sum_{\mathbf{x}} \sum_{\mathbf{y}} Q(\mathbf{x}, \mathbf{y}) \left[\boldsymbol{\nabla} T \frac{\partial \mathbf{W}}{\partial \mathbf{p}}(\mathbf{x}; \mathbf{0}) \right]^{\mathrm{T}} \left[\boldsymbol{\nabla} T \frac{\partial \mathbf{W}}{\partial \mathbf{p}}(\mathbf{y}; \mathbf{0}) \right] \tag{37}$$

can then also be computed as:

$$H_{\mathrm{po}} \;=\; \sum_{\mathbf{x}} \mathbf{SD}_{\mathrm{po}}^{\mathrm{T}}(\mathbf{x})\, \mathbf{SD}_{\mathrm{po}}(\mathbf{x}) \tag{38}$$

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. Again, see [1] for more details.

In summary, minimizing the expression in Equation (19) simultaneously with respect to $\mathbf{p}$ and $\boldsymbol{\lambda}$ can be performed by first minimizing the second term in Equation (31) with respect to $\mathbf{p}$ using the inverse compositional algorithm with the quadratic form in Equation (33). The only changes

The Project Out 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 $\mathbf{SD}_{\mathrm{po}}(\mathbf{x})$ using Equation (36)
(6) Compute the Hessian matrix H_{po} using Equation (38) and invert it

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 $E(\mathbf{x})$ using Equation (10)
(7) Compute $\sum_{\mathbf{x}} \mathbf{SD}^{\mathrm{T}}_{\mathrm{po}}(\mathbf{x}) E(\mathbf{x})$
(8) Compute $\Delta \mathbf{p} = -H^{-1}_{\mathrm{po}} \sum_{\mathbf{x}} \mathbf{SD}^{\mathrm{T}}_{\mathrm{po}}(\mathbf{x}) E(\mathbf{x})$
(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 $\|\Delta \mathbf{p}\| \leq \epsilon$

Post-Computation:

(10) Compute the appearance parameters $\boldsymbol{\lambda}$ using Equation (32)

Figure 4: The project out inverse compositional algorithm is very similar to the original inverse compositional algorithm. The only changes are: (1) to compute the project out steepest descent images in Step 5, (2) to compute the project out Hessian in Step 6, and (3) to compute the appearance parameters in Step 10. The online computational cost is almost identical to the original algorithm and is summarized in Table 4.

needed to the algorithm are: (1) to use the weighted steepest descent images in Equation (36), and (2) to use the weighted Hessian in Equation (38). Once the inverse compositional algorithm has converged, the optimal value of $\boldsymbol{\lambda}$ can be computed using Equation (32) where $\mathbf{p}$ are the optimal warp parameters. The project out inverse compositional algorithm is summarized in Figure 4.

The computational cost of the project out inverse compositional algorithm is almost identical to that of the original inverse compositional algorithm. See Table 2. The only extra cost is in Step 5 and Step 10. The computation of the steepest descent images in Step 5 is substantially more involved, but this step is a pre-computation step. The computation of the appearance parameters is minimal just taking time $\mathrm{O}(m\,N)$. The computation cost is summarized in Table 4.

3.2.3 Caveat: Step Size Modeling Gain

Overall the project out algorithm performs well, as is demonstrated in Section 3.4. There is one scenario, however, when it performs particularly poorly. In particular, if the linear appearance

Table 4: The computational cost of the project out inverse compositional algorithm is almost identical to the original inverse compositional algorithm. The only additional cost is: (1) computing the steepest descent images in Step 5, and (2) the one off extra cost of computing the appearance parameters in Step 10.

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

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

Post-Computation	Step 10	Total
	$O(m\,N)$	$O(m\,N)$

variation is used to model "gain," the step size is estimated incorrectly as we now show.

As mentioned above, one common use of linear appearance variation is to model gain and bias. If $A_1 = T$ and A_2 is the "all one" image, Equation (18) becomes:

$$T(\mathbf{x}) \;+\; \lambda_1\,T(\mathbf{x}) \;+\; \lambda_2 \tag{39}$$

which has a gain of $1 + \lambda_1$ and a bias of λ_2 (relative to the original template.) Any gain and bias can therefore be represented. For now suppose that only gain is modeled; i.e. set $\lambda_2 = 0$.

If the input image $I(\mathbf{x})$ is a perfect match to the template $T(\mathbf{x})$ with no appearance correction (for some set of warp parameters $\mathbf{p}$), then the input image $g\,I(\mathbf{x})$ must also be a perfect match with the appearance correction $(g-1)\,T(\mathbf{x})$ (and the same set of warp parameters $\mathbf{p}$.) When the appearance variation consists of $A_1 = T$, the parameter update for $I(\mathbf{x})$ is:

$$\Delta\mathbf{p} \;=\; -H_{\text{po}}^{-1} \sum_{\mathbf{x}} \mathbf{SD}_{\text{po}}^{\text{T}}(\mathbf{x})\,[T(\mathbf{x}) - I(\mathbf{W}(\mathbf{x};\mathbf{p}))] \;=\; H_{\text{po}}^{-1} \sum_{\mathbf{x}} \mathbf{SD}_{\text{po}}^{\text{T}}(\mathbf{x})\,I(\mathbf{W}(\mathbf{x};\mathbf{p})). \tag{40}$$

The $T(\mathbf{x})$ term disappears because $A_1 = T$ and so the steepest descent images have no component in the T direction. The inner product $\sum_{\mathbf{x}} \mathbf{SD}_{\text{po}}(\mathbf{x})\,T(\mathbf{x})$ is therefore zero. Similarly, the parameter update for $g\,I(\mathbf{x})$ is:

$$\Delta\mathbf{p} \;=\; -H_{\text{po}}^{-1} \sum_{\mathbf{x}} \mathbf{SD}_{\text{po}}^{\text{T}}(\mathbf{x})\,[T(\mathbf{x}) - g\,I(\mathbf{W}(\mathbf{x};\mathbf{p}))] \;=\; g\,H_{\text{po}}^{-1} \sum_{\mathbf{x}} \mathbf{SD}_{\text{po}}^{\text{T}}(\mathbf{x})\,I(\mathbf{W}(\mathbf{x};\mathbf{p})). \tag{41}$$

So, although the project out algorithm with the two inputs $I(\mathbf{x})$ and $g\,I(\mathbf{x})$ should converge to the same warp parameters $\mathbf{p}$, the algorithm takes steps that are larger by a factor of g in the second case. If $g \gg 1$ the algorithm will therefore likely diverge because it takes too big steps. If $g \ll 1$ the algorithm will probably still converge, but the rate of convergence will be very slow.

There are various ways to correct for this "step-size" problem. One possibility is to use a dynamic step-size adjustment algorithm like Levenberg-Marquardt [3]. Such algorithms check that each step of the algorithm results in an improvement in the error. If the error does not improve and the algorithm is diverging, a smaller step size is tried. Another possibility is to compute the magnitude of the component of $I(\mathbf{W}(\mathbf{x};\mathbf{p}))$ in the direction of $T(\mathbf{x})$ (normalized appropriately):

$$\gamma \;=\; \frac{\sum_{\mathbf{x}} I(\mathbf{W}(\mathbf{x};\mathbf{p}))\,T(\mathbf{x})}{\sum_{\mathbf{x}}(T(\mathbf{x}))^2} \tag{42}$$

and then reduce the step size computed in the project out algorithm by that factor; i.e. instead of using Step 8 in Figure 4 use:

$$\Delta\mathbf{p} \;=\; -\frac{1}{\gamma} H_{\mathrm{po}}^{-1} \sum_{\mathbf{x}} \mathbf{SD}_{\mathrm{po}}^{\mathrm{T}}(\mathbf{x}) E(\mathbf{x}). \tag{43}$$

As can be seen, when the gain is approximately 1.0 and $I(\mathbf{W}(\mathbf{x};\mathbf{p})) \approx T(\mathbf{x})$, then $\gamma \approx 1$ and so the step-size correction does not affect the algorithm. Similarly, when the gain is g and $I(\mathbf{W}(\mathbf{x};\mathbf{p})) \approx g\,T(\mathbf{x})$, then $\gamma \approx g$ and the step-size is corrected appropriately. The additional computational cost of evaluating Equations (42) and (43) is just $\mathrm{O}(N)$ and so is negligible. In Section 3.4 we empirically evaluate the step-size correction algorithm defined by Equations (42) and (43).

3.2.4 Discussion

Suppose that $T = A_1$. The set of images spanned by $T(\mathbf{x}) + \sum_{i=1}^{m} \lambda_i A_i(\mathbf{x})$ is then the same if $T = A_i$ for any $i = 2, \ldots, m$, or even for T any other image in $\mathrm{span}(A_i)$; i.e. the appearance model is the same for a variety of different templates. Any of these templates could be used with the project out algorithm and the result should theoretically be the same. In particular, if the model

includes a bias term, the "all one" image is a very bad choice for the template. In this case, the gradient of the template and the steepest descent images would all be exactly zero, a degenerate case, and the algorithm would not be applicable. This degeneracy suggests that some images are not as good a choice for the template as others. The question of what is the best choice for the template, and whether the choice of the template affects the project out algorithm differently than it does the simultaneous algorithm are outside the scope of this paper and are left for future study.

3.3 The Normalization Inverse Compositional Algorithm

3.3.1 Goal of the Algorithm

One frequently used way of coping with gain and bias variation is to "normalize" both the template $T(\mathbf{x})$ and the input image $I(\mathbf{W}(\mathbf{x};\mathbf{p}))$. In particular, in many applications such as face recognition using eigenfaces the mean pixel intensities, or colors, are set to zero via:

$$\mu \leftarrow \frac{1}{N}\sum_{\mathbf{x}} T(\mathbf{x}) \quad \text{and then} \quad T(\mathbf{x}) \leftarrow T(\mathbf{x}) - \mu. \tag{44}$$

Next, the vector $T(\mathbf{x})$ is set to have unit norm via:

$$\sigma^2 \leftarrow \frac{1}{N}\sum_{\mathbf{x}} (T(\mathbf{x}))^2 \quad \text{and then} \quad T(\mathbf{x}) \leftarrow \frac{1}{\sigma} T(\mathbf{x}). \tag{45}$$

Similarly, these two steps are also applied to $I(\mathbf{W}(\mathbf{x};\mathbf{p}))$. The result is to remove the effect of gain and bias. Moreover, the gain and bias can easily be computed from μ and σ. We now derive a similar normalization algorithm for arbitrary linear variation.

3.3.2 Derivation of the Algorithm

Equation (44) can be re-arranged to:

$$\mu \leftarrow \sum_{\mathbf{x}} (T(\mathbf{x}) \times \frac{1}{\sqrt{N}}) \quad \text{and then} \quad T(\mathbf{x}) \leftarrow T(\mathbf{x}) - \mu \times \frac{1}{\sqrt{N}}. \tag{46}$$

Equation (44) can therefore be regarded as projecting out the (unit) vector $(\frac{1}{\sqrt{N}}, \ldots, \frac{1}{\sqrt{N}})$. Our proposal for a normalization algorithm is to perform an analogous step for each appearance vector A_i. A minor difference, however, is that instead of projecting out the entire component in the direction A_i from both $I(\mathbf{W}(\mathbf{x};\mathbf{p}))$ and $T(\mathbf{x})$, we instead project out only enough so that the component of $I(\mathbf{W}(\mathbf{x};\mathbf{p}))$ is the same as that of the template $T(\mathbf{x})$. One possible way to do this would be to apply the appropriate normalization to $I(\mathbf{W}(\mathbf{x};\mathbf{p}))$ directly after Step 1 of the algorithm (see Figure 2). In turns out, however, that after some simple algebra, this normalization can be achieved by normalizing the error image so that the component of the error image in the direction A_i is zero; i.e. this makes the component of $I(\mathbf{W}(\mathbf{x};\mathbf{p}))$ and $T(\mathbf{x})$ in the direction A_i the same. As an added benefit, an estimate of the appearance parameter λ_i can be computed in the process. In particular, the normalization step consists of:

$$\lambda_i \;=\; -\sum_{\mathbf{x}} A_i(\mathbf{x})\, E(\mathbf{x}) \;\; \text{for } i = 1, \ldots, m \;\; \text{and then} \;\; E_{\text{norm}}(\mathbf{x}) \;\leftarrow\; E(\mathbf{x}) + \sum_{i=1}^{m} \lambda_i\, A_i(\mathbf{x}) \quad (47)$$

where again we assume that the A_i are orthonormal. Inserting this step into the inverse compositional algorithm gives the normalization inverse compositional algorithm summarized in Figure 5. The only difference between the normalization algorithm and the inverse compositional algorithm is the addition of Step 2a. Hence, the computation cost of the two algorithms is similar. See Table 5 for a summary. Step 2a must be performed in every iteration (unlike the analogous Step 10 in the project out algorithm) and so the normalization algorithm can be substantially slower than the project out inverse compositional algorithm when $m \gg n$.

3.3.3 Step Size Adjustment Modeling Gain

The normalization inverse compositional algorithm is prone to the same error in the step size estimate that the project out algorithm is. See Section 3.2.3. The same correction can be applied. We can even use the normalization algorithm to compute the step size correction even more efficiently.

The Normalization 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 $\mathbf{SD}_{\text{ic}}(\mathbf{x})$ using Equation (16)
(6) Compute the Hessian matrix H_{ic} using Equation (17) and invert it

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 $E(\mathbf{x})$ using Equation (10)
(2a) Estimate $\boldsymbol{\lambda}$ and compute $E_{\text{norm}}(\mathbf{x})$ using Equation (47)
(7) Compute $\sum_{\mathbf{x}} \mathbf{SD}_{\text{ic}}^{\text{T}}(\mathbf{x}) E_{\text{norm}}(\mathbf{x})$
(8) Compute $\Delta \mathbf{p} = -H_{\text{ic}}^{-1} \sum_{\mathbf{x}} \mathbf{SD}_{\text{ic}}^{\text{T}}(\mathbf{x}) E_{\text{norm}}(\mathbf{x})$
(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 5: The normalization inverse compositional algorithm is almost exactly the same as the original inverse compositional algorithm. The only difference is the addition of Step 2a in which the input image $I(\mathbf{W}(\mathbf{x}; \mathbf{p}))$ is normalized (using the error image $E(\mathbf{x})$). The computation cost of the algorithm is therefore similar and is summarized in Table 5. If $m >> n$ the normalization algorithm can be a lot slower however.

Table 5: The computation cost of the normalization inverse compositional algorithm is almost identical to that of the original inverse compositional algorithm. The only change is the addition of Step 2a. Step 2a is analogous to Step 10 in the project out algorithm. The difference, however, is that Step 2a must be performed in every iteration and so the normalization algorithm can be a lot slower than the project out algorithm.

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 2a	Step 7	Step 8	Step 9	Total
	$\mathrm{O}(n\,N)$	$\mathrm{O}(N)$	$\mathrm{O}(m\,N)$	$\mathrm{O}(n\,N)$	$\mathrm{O}(n^2)$	$\mathrm{O}(n^2)$	$\mathrm{O}((n+m)\,N + n^2)$

If we are modeling gain and $A_1 = T$, then γ in Equation (42) can be estimated as:

$$\gamma \;=\; 1 + \lambda_1 \tag{48}$$

and then the step size corrected:

$$\Delta \mathbf{p} \;=\; -\frac{1}{\gamma} H_{\text{ic}}^{-1} \sum_{\mathbf{x}} \mathbf{SD}_{\text{ic}}^{\text{T}}(\mathbf{x}) E_{\text{norm}}(\mathbf{x}). \tag{49}$$

In Section 3.4 we empirically evaluate the step-size correction defined by Equations (48) and (49).

3.4 Experimental Results

We conducted a variety of experiments to compare the performance of the 3 linear appearance variation algorithms (simultaneous inverse compositional, project out, and normalization) and their variants. Our experimental procedure is similar to that in [3]. In particular, we started with the image $I(\mathbf{x})$ in Figure 2 of [3]. We manually selected a 100×100 pixel template $T(\mathbf{x})$ in the center of the face. We then added the appearance variation $\sum_{i=1}^{m} \lambda_i A_i(\mathbf{x})$ to $I(\mathbf{x})$. The exact choice of m, λ_i, and $A_i(\mathbf{x})$ depends on the experiment in question and is described in detail below.

We then randomly generated affine warps $\mathbf{W}(\mathbf{x};\mathbf{p})$ in the following manner. (We use the same procedure that was used in [3].) 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 $I(\mathbf{x}) + \sum_{i=1}^{m} \lambda_i A_i(\mathbf{x})$ with the affine warp $\mathbf{W}(\mathbf{x};\mathbf{p})$ and run the various algorithms starting from the identity warp. Where appropriate, the appearance parameters are initialized to 0.

Since the 6 parameters in the affine warp have different units, they must be combined in some way. We use the same error measure as in [3]. 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 the correct locations. (We prefer this error measure to normalizing the units so that the 6 parameters error are comparable.)

As in [3], 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. For the simultaneous and normalization algorithms we sometimes also plot a "rate of convergence of the appearance parameters". This measure is only meaningful for the project out algorithm after the algorithm has converged. The error measure for the appearance parameters is the Euclidean L2 norm of the appearance parameter vector $\boldsymbol{\lambda}$.

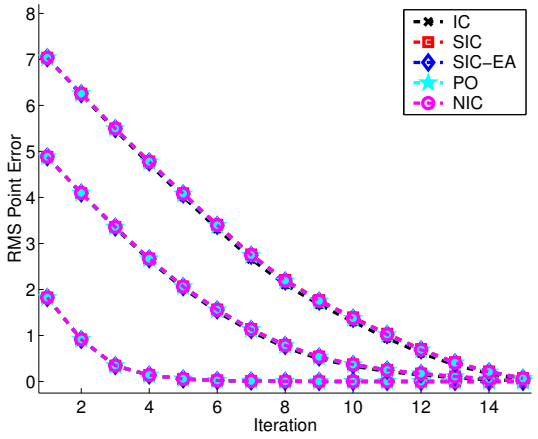
3.4.1 Experiment 1: Comparison with Inverse Compositional

The goal of Experiment 1 is to show that all of the linear appearance variation algorithms can cope with some appearance variation, and yet still perform as well as the original inverse compositional algorithm when there is no appearance variation. In other experiments we will investigate how the algorithms compare with varying degrees of appearance variation, and in the presence of additive noise. For now, we just consider a very simple case. In particular, we just use $m = 1$ appearance variation image A_1. The specific image that we used is the image of a different face approximately aligned with the face in the template. (As described in Section 5.4 we are making all of our code available so that the reader can experiment with other choices for A_1.)

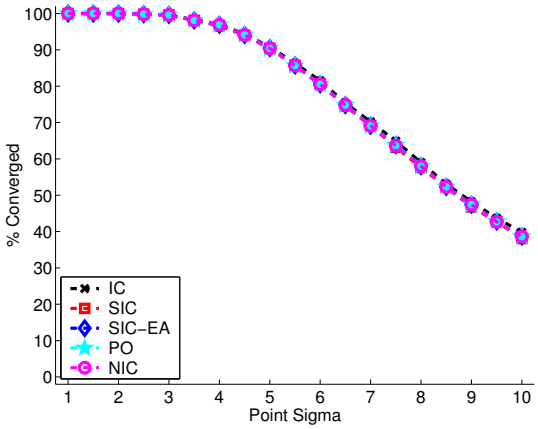
The results are shown in Figure 6. In Figures 6(a) and (b) we include plots of the convergence rate and frequency of convergence for $\lambda_1 = 0.0$, no appearance variation. In Figures 6(c) and (d) we include similar plots for $\lambda_1 = 0.35 \times \frac{||T||}{||A_1||}$, a relatively large amount of appearance variation (the appearance variation accounts for approximately one quarter of the combined input image). We include curves for 5 algorithms, (1) the original inverse compositional algorithm with no appearance variation modeling (IC), (2) the simultaneous inverse compositional algorithm (SIC), (3) the efficient variant of the simultaneous algorithm (SIC-EA), (4) the project out algorithm (PO), and (5) the normalization algorithm (NIC). The main things to note in Figure 6 are: (1) with no appearance variation ($\lambda_1 = 0.00$) all of the algorithms perform almost identically, and (2) with $\lambda_1 = 0.35 \times \frac{||T||}{||A_1||}$ the original inverse compositional algorithm performs far worse, whereas all four of the appearance variation algorithms perform very well, and similarly. These results demonstrate that all 4 of the appearance variation algorithms can cope with fairly substantial linear appearance variation, whereas the original inverse compositional algorithm cannot.

3.4.2 Experiment 2: Varying λ_1

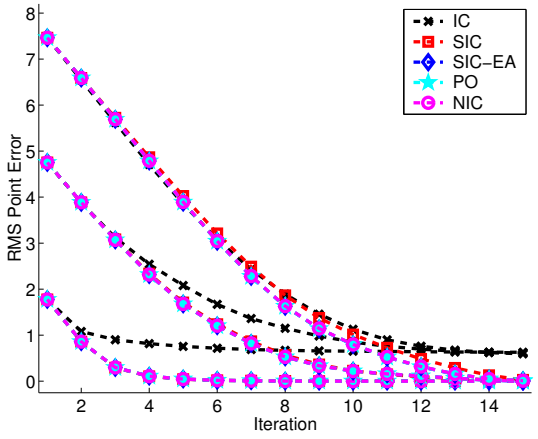
In Experiment 2 we investigate how the performance of the 4 appearance variation algorithms (SIC, SIC-EA, PO, and NIC) varies with the amount of appearance variation. We re-ran Experiment 1



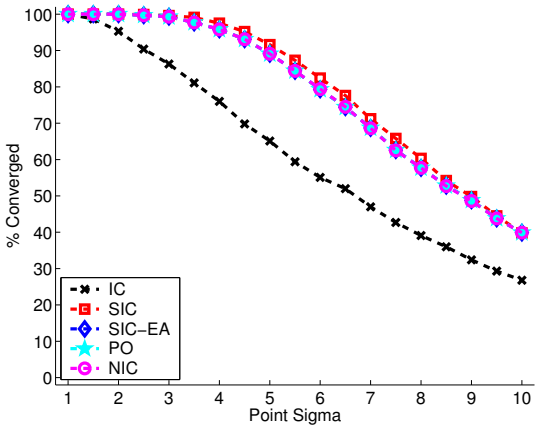
(a) Convergence Rate, $\lambda_1 = 0.00$



(b) Convergence Frequency, $\lambda_1 = 0.00$



(c) Convergence Rate, $\lambda_1 = 0.35 \times \frac{||T||}{||A_1||}$

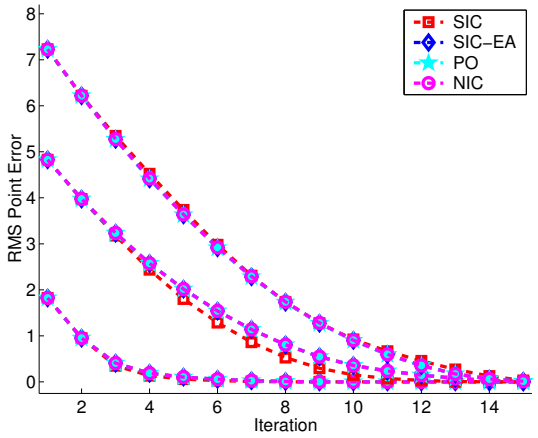


(d) Convergence Frequency, $\lambda_1 = 0.35 \times \frac{||T||}{||A_1||}$

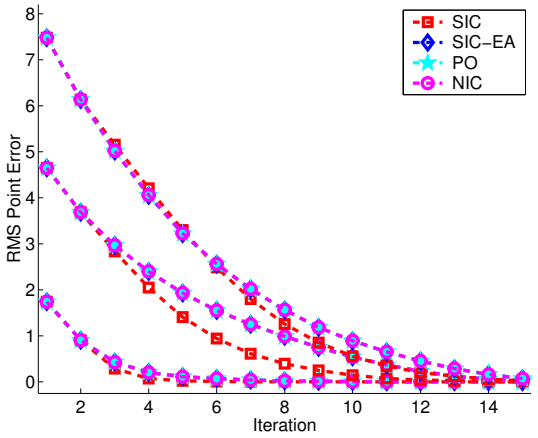
Figure 6: A comparison of the inverse compositional algorithm with no appearance variation modeling (IC) with 4 different linear appearance variation algorithms: (1) the simultaneous inverse compositional algorithm (SIC), (2) the efficient variant of the simultaneous algorithm (SIC-EA), (3) the project out algorithm (PO), and (4) the normalization algorithm (NIC). (a) and (b) contain results for $\lambda_1 = 0.0$ and demonstrate that all of the algorithms perform similarly with no appearance variation. (c) and (d) contain results for $\lambda_1 = 0.35 \times \frac{||T||}{||A_1||}$ and demonstrate that the inverse compositional algorithm breaks down with appearance variation whereas all 4 of the appearance variation algorithms are able to cope with the appearance variation.

with the same single appearance image A_1 for a variety of different values of λ_1. In Figure 7 we include results for $\lambda_1 = 1.0 \times \frac{||T||}{||A_1||}$ and $\lambda_1 = 2.0 \times \frac{||T||}{||A_1||}$.

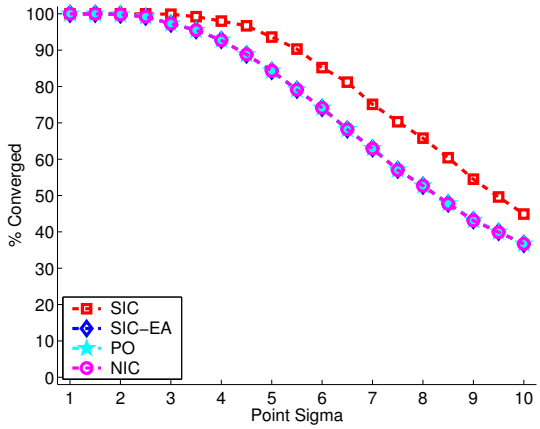
In Figures 7(a) and (b) we plot the rate of convergence, and in (c) and (d) the frequency of convergence. The results show that as λ_1 increases the 4 algorithms perform very differently. The simultaneous algorithm performs by far the best, whereas the performance of the other 3 algorithms (SIC-EC, PO, and NIC) is far worse. (Note, however, that when $\lambda = 2.0 \times \frac{||T||}{||A_1||}$ the contribution



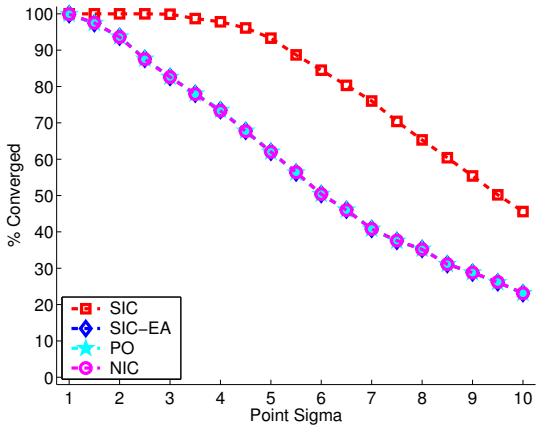
(a) Convergence Rate, $\lambda_1 = 1.0 \times \frac{\|T\|}{\|A_1\|}$



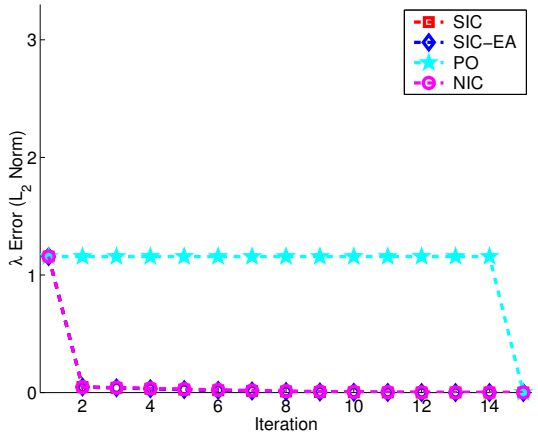
(b) Convergence Rate, $\lambda_1 = 2.0 \times \frac{\|T\|}{\|A_1\|}$



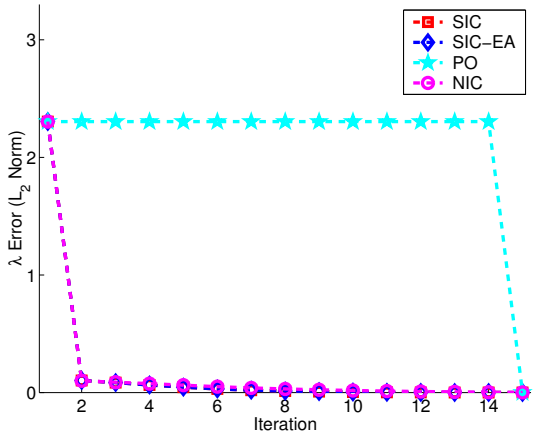
(c) Convergence Frequency, $\lambda_1 = 1.0 \times \frac{\|T\|}{\|A_1\|}$



(d) Convergence Frequency, $\lambda_1 = 2.0 \times \frac{\|T\|}{\|A_1\|}$



(e) Appearance Convergence, $\lambda_1 = 1.0 \times \frac{\|T\|}{\|A_1\|}$



(f) Appearance Convergence, $\lambda_1 = 2.0 \times \frac{\|T\|}{\|A_1\|}$

Figure 7: A comparison of the 4 appearance variation algorithms (SIC, SIC-EA, PO, NIC) with varying amounts of appearance variation. The results show that the simultaneous algorithm (SIC) performs far better than the other 3 algorithms for very large amounts of appearance variation. In (e) and (f) we plot the rate of convergence of the appearance parameter λ_1. For all 3 algorithms that estimate it every iteration, the estimate of λ_1 converges very quickly. For the project out algorithm, λ_1 is only estimated once the algorithm has converged. The estimate at that point is as accurate as with the other 3 algorithms, however.

of the appearance image A_1 to the input image I is twice the contribution of the template itself T. In this case, the template is almost impossible to see in the input image.) In conclusion, the simultaneous algorithm performs by far the best when the appearance variation is very large. Also note that the other 3 algorithms all perform almost identically. The project out and normalization algorithms are very similar. They both work in the subspace orthogonal to the appearance variation. The project out algorithm projects the steepest descent images into this subspace. The normalization algorithm projects the error image into the same subspace. It is therefore to be expected that they perform almost identically. The fact that the efficient approximation to the simultaneous algorithm also performs almost identically is harder to explain. See Section 5 for a discussion.

Why does the simultaneous algorithm perform far better than the other three algorithms? It is fairly easy to understand why the efficient approximation to the simultaneous algorithm does not perform as well. The approximation in the steepest descent images is explicit in the efficient approximation algorithm. For the normalization and project out algorithms, the poor performance is caused by the fact that the component of the error image caused by the appearance variation in the subspace orthogonal to the appearance variation is non-zero when the alignment is not correct. Implicitly it is assumed that this component is zero, which it only true when the template is perfectly aligned to the input image. This non-zero component generates an unexpected perturbation to the parameter updates which, if large enough, can cause the algorithm to diverge.

In Figures 7(e) and (f) we plot the rate of convergence of the estimate of the appearance variation parameter λ_1. The results show that the estimates of the appearance variation for all 3 algorithms that estimate it every iteration (SIC, SIC-EA, and NIC) all converge very quickly. For the project out algorithm, the appearance variation is not estimated until the algorithm has converged, but when it is estimated, it is estimated as accurately as with the other 3 algorithms.

3.4.3 Experiment 3: Varying the Number of Appearance Images

In Experiment 3 we investigate how the performance of the 4 appearance variation algorithms varies with the number of appearance images. To do this, we need to decide: (1) what the ap-

pearance images A_i are, and (2) what the appearance parameters λ_i are. We chose the appearance images by taking a large image of some scenery and randomly selecting a number of sub-images of the appropriate size. The results are then orthonormalized and used as the A_i. To avoid any dependence on the magnitude of the appearance variation (see Experiment 2), we chose the appearance parameters such that $\sum_i \lambda_i^2 \|A_i\|^2 = (0.35)^2 \times \|T\|^2$ as we vary the number of appearance images. We also set $\lambda_i^2 \|A_i\|^2$ to be the same for each i to spread out the appearance variation equally.

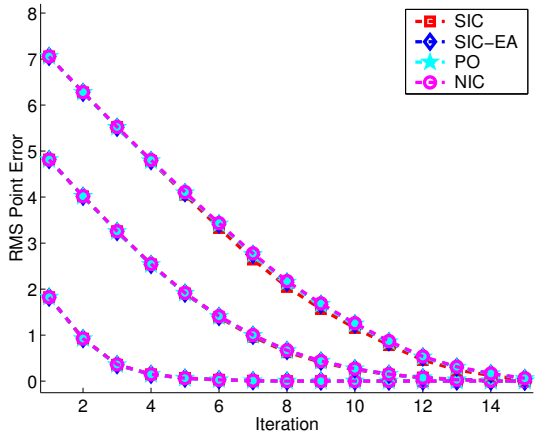
The results for 2 appearance images and 10 appearance images are shown in Figure 8. The main point to note is that the performance of all 4 appearance variation algorithms is almost identical. There is a very small drop off in performance between 2 and 10 appearance images, but there is no evidence that any of the algorithms performs significantly worse than any of the others.

3.4.4 Experiment 4: Robustness to Additive Noise

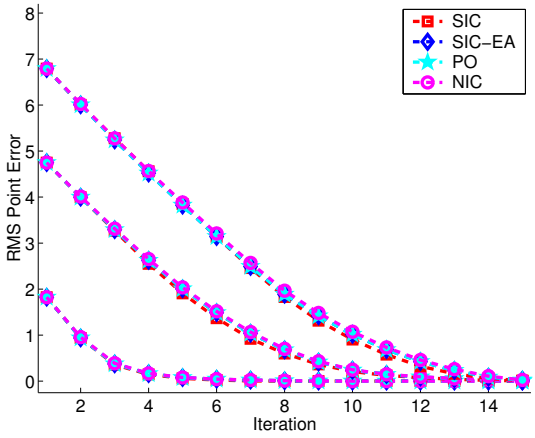
In Experiment 4 we investigate how the performance of the 4 appearance variation algorithms varies in the presence of noise. We repeated the conditions of Experiment 1, but following the procedure in [3], we added zero-mean, white Gaussian noise to the input image I before we ran the algorithms. The results for additive noise with standard deviation 4.0 grey levels and 16.0 grey levels are included in Figure 9. The results show that overall the simultaneous algorithm is slightly more robust to noise than the other algorithms (which perform almost identically), but not by much. As in [3] we also ran experiments adding noise to the template and the appearance images. As in Figure 9, the results show that all 4 of the algorithms perform similarly and none of the algorithms is significantly more robust to noise than any of the others. The results are omitted for lack of space, but can be regenerated using the code that we are making available. (See Section 5.4.)

3.4.5 Experiment 5: Modeling Gain

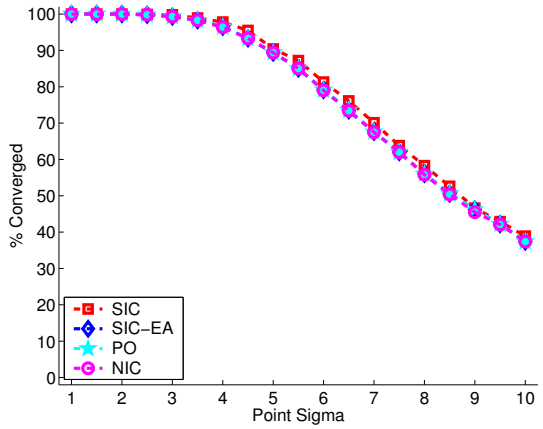
In Experiment 5 we investigate how the performance of the algorithms varies modeling gain. To model gain, we set $A_1 = T$ and run the algorithms for a variety of different values of λ_1. As well as the 4 appearance variation algorithms (SIC, SIC-EA, PO, and NIC), we also run the project out



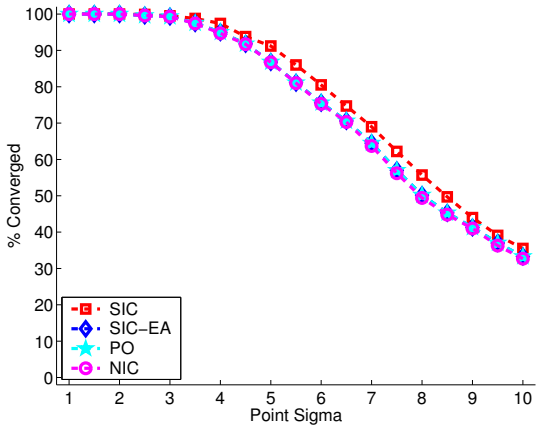
(a) Convergence Rate, 2 app. images



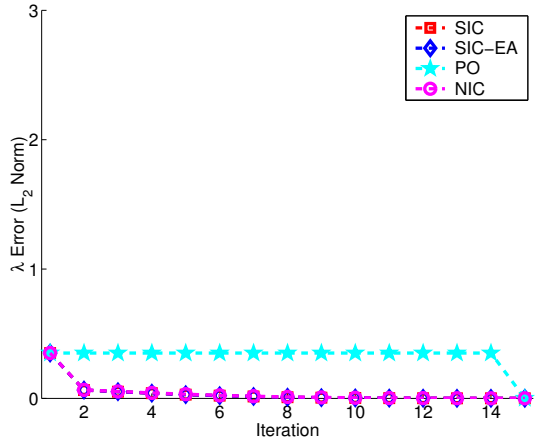
(b) Convergence Rate, 10 app. images



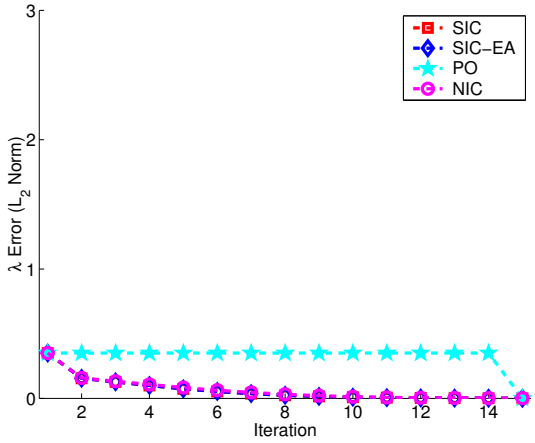
(c) Convergence Frequency, 2 app. images



(d) Convergence Frequency, 10 app. images

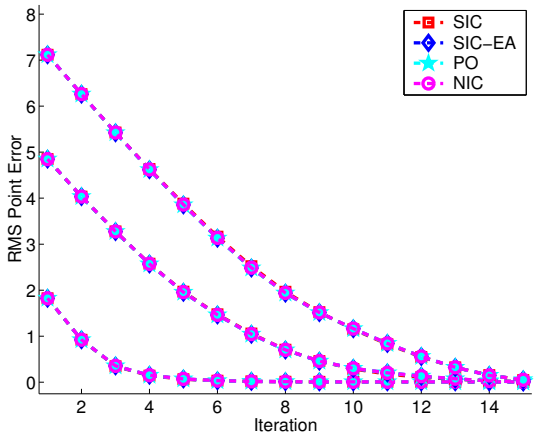


(e) Appearance Convergence, 2 app. images

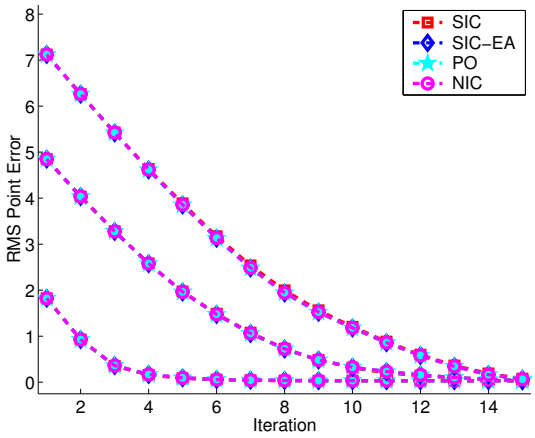


(f) Appearance Convergence, 10 app. images

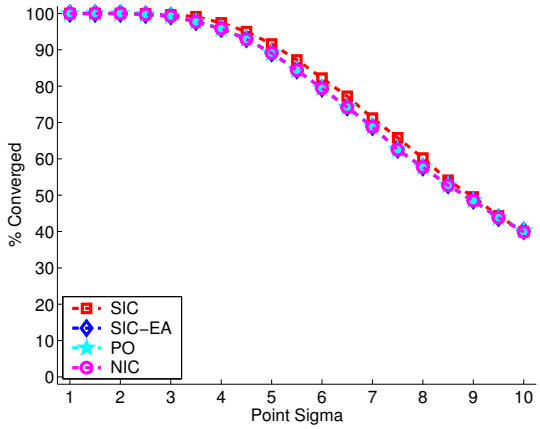
Figure 8: A comparison of the 4 appearance variation algorithms with varying numbers of appearance images. The performance with 2 appearance images in (a), (c), and (e) is very similar to the performance with 10 appearance images in (b), (d), and (e). If anything the performance is slightly worse with 10 images. There is little evidence that any of the algorithms performs much worse than any of the others.



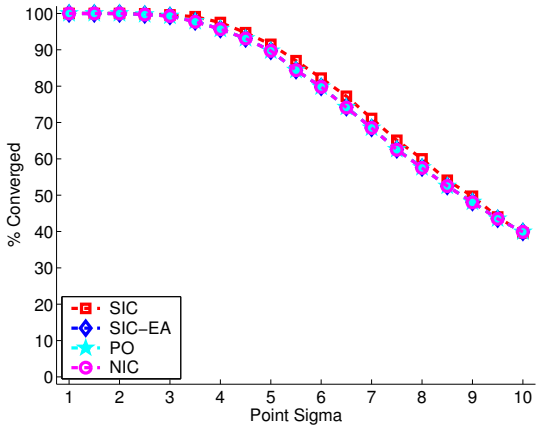
(a) Convergence Rate, Noise SD = 4.0



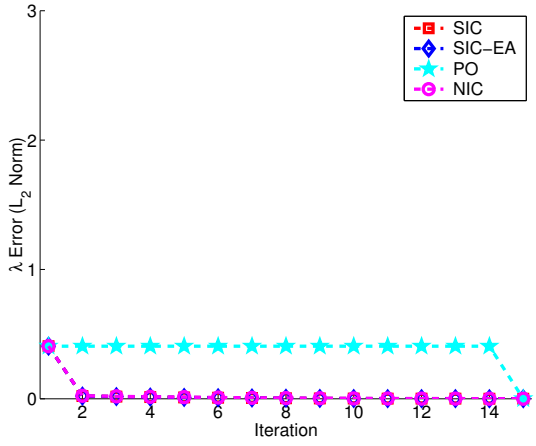
(b) Convergence Rate, Noise SD = 16.0



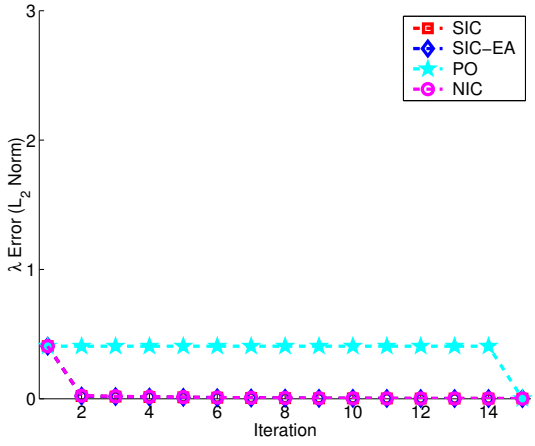
(c) Convergence Frequency, Noise SD = 4.0



(d) Convergence Frequency, Noise SD = 16.0



(e) Appearance Convergence, Noise SD = 4.0



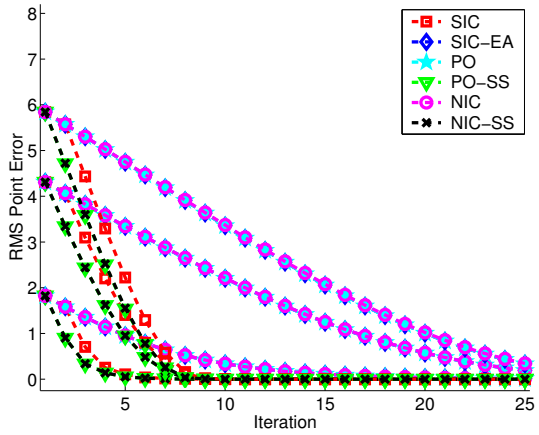
(f) Appearance Convergence, Noise SD = 16.0

Figure 9: A comparison of the 4 appearance variation algorithms with varying amounts of zero mean, white Gaussian noise added to the input image before the algorithms are run. Otherwise, the experimental conditions are identical to Experiment 1. The results for noise with standard deviation 4.0 grey levels are included in (a), (c), and (e), and for noise with standard deviation 16.0 grey levels in (b), (d), and (f). Overall the simultaneous algorithm is slightly more robust to noise than the other 3 appearance variation algorithms.

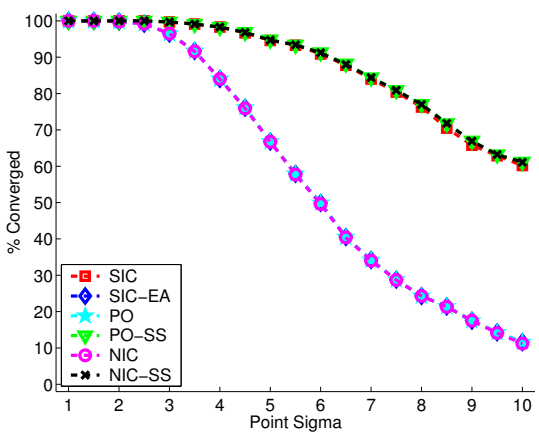
algorithm with the step size correction described in Section 3.2.3 (PO-SS), and the normalization algorithm with the step size correction described in Section 3.3.3 (NIC-SS). The results for $\lambda_1 = -0.75$ (gain $= 0.25$) are included in Figures 10(a) and (b), the results for $\lambda_1 = 1.0$ (gain $= 2.0$) in Figures 10(c) and (d), and the results for $\lambda_1 = 1.5$ (gain $= 2.5$) in Figures 10(e) and (f).

For $\lambda_1 = -0.75$ (gain$= 0.25$) the project out and normalization algorithms converge far more slowly than the simultaneous algorithm. When the perturbation to the affine warp is large enough, the algorithms converge so slowly than by 25 iterations not all of the trials converge. Hence the convergence frequency is affected. The convergence frequency of the project out and normalization algorithms is far worse than the other algorithms. For $\lambda_1 = 1.00$ (gain$= 2.0$) the project out and normalization algorithms do converge most of the time. Looking at the rate of convergence plot in Figure 10(c), however, we see that they converge to a much higher error than the other algorithms. After a point, the project out and normalization algorithms oscillate around the correct solution. (This was verified by watching the algorithms converge.) Because the algorithms take a step that is roughly twice what it should be, once they reach the approximately quadratic part of the error function close to the correct answer, they remain equally far away from the correct answer, oscillating backwards and forwards. In our implementation, we use a very loose definition of convergence (RMS point error < 1.0 pixels) and so these cases are counted as converging and so the frequency of convergence in Figure 10(d) is unaffected. For $\lambda_1 = 1.5$ (gain$= 2.5$) the project out and normalization algorithms fail to converge most of the time. They take steps that are over twice as big as they should be and so they immediately diverge.

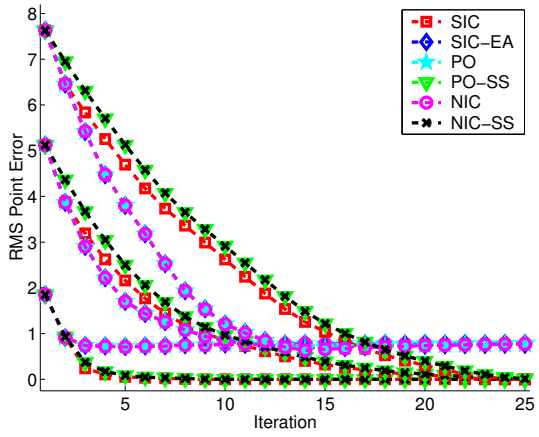
As in all previous experiments, the project out algorithm, the normalization algorithm, and the efficient approximation to the simultaneous algorithm perform almost identically. The step size corrections to the project out and normalization algorithms do appear to work correctly. Both of these modified algorithms perform very similarly to the simultaneous algorithm. Note, however, that the underlying cause of the poor performance of the unmodified project out and normalization algorithms with large gain variation is essentially the same as the cause of the poor performance of these algorithm with large appearance variation in Experiment 2. The difference is that when



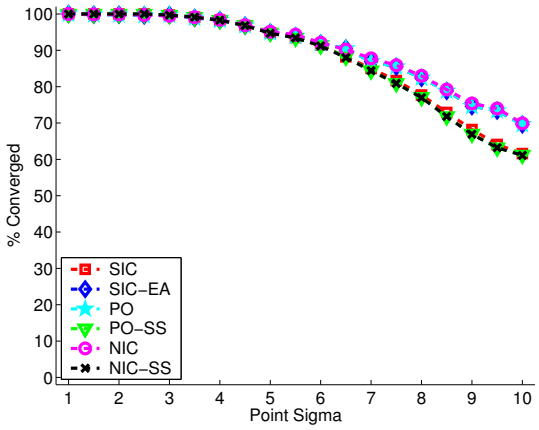
(a) Convergence Rate, $\lambda_1 = -0.75$, gain $= 0.25$



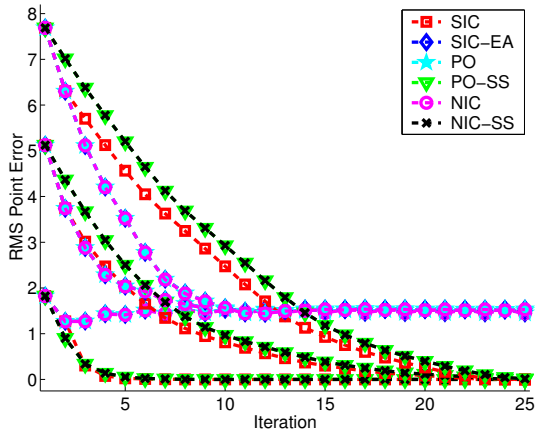
(b) Convergence Freq., $\lambda_1 = -0.75$, gain $= 0.25$



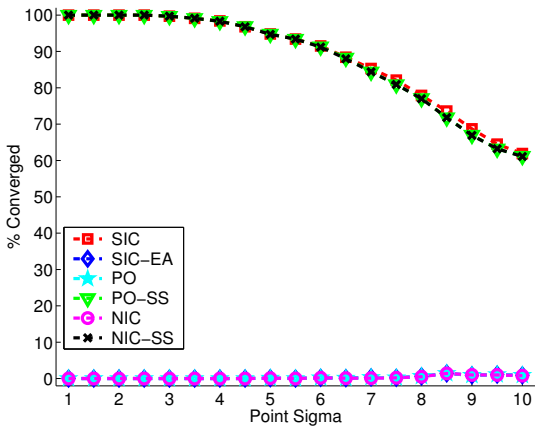
(c) Convergence Rate, $\lambda_1 = 1.0$, gain $= 2.0$



(d) Convergence Freq., $\lambda_1 = 1.0$, gain $= 2.0$



(e) Convergence Rate, $\lambda_1 = 1.5$, gain $= 2.5$



(f) Convergence Freq., $\lambda_1 = 1.5$, gain $= 2.5$

Figure 10: A comparison of the appearance variation algorithms modeling gain. In addition to the 4 algorithms studied in the previous experiments, we also consider the project out algorithm with the step size correction described in Section 3.2.3 (PO-SS), and the normalization algorithm with the step size correction described in Section 3.3.3 (NIC-SS). The performance of the project out and normalization algorithms without the step size correction is significantly worse than the simultaneous algorithm. The step size correction modification to these two algorithm does correct the problem resulting in much better performance.

the appearance variation models gain, the error in the computation of the parameter updates is only manifested as a step size error that can be corrected very simply with the corrections in Sections 3.2.3 and 3.3.3. Whether there is a similar correction that could be applied to the algorithms for arbitrary large appearance variation is an open question and is left as future work.

4 Linear Appearance Variation with a Robust Error Function

Another generalization of the expression in Equation (1) is to use a *robust error function* instead of the "sum of squares" or Euclidean L2 norm. Robust extensions to the original inverse compositional algorithm are the subject of Part 2 of this series [1]. The goal in [1] is to minimize:

$$\sum_{\mathbf{x}} \varrho \left(\left[T(\mathbf{x}) - I(\mathbf{W}(\mathbf{x};\mathbf{p})) \right]^2 ; \boldsymbol{\sigma} \right) \tag{50}$$

with respect to the warp parameters $\mathbf{p}$ where $\varrho(t;\boldsymbol{\sigma})$ is a symmetric *robust error function* [12] and $\boldsymbol{\sigma} = (\sigma_1, \sigma_2, \ldots, \sigma_s)^{\mathbf{T}}$ is a vector of *scale parameters.* (See [1] for a discussion of why we only consider symmetric error functions.) In [1] we described how the scale parameters $\boldsymbol{\sigma}$ can be estimated from the error image $T(\mathbf{x}) - I(\mathbf{W}(\mathbf{x};\mathbf{p}))$. For ease of explanation, in this paper we treat the scale parameters as known constants and so drop the scale parameters from $\varrho(\cdot;\boldsymbol{\sigma})$. Hence, we simply denote the robust function $\varrho(\cdot)$. Finally, see [1] for a discussion of how to choose $\varrho(\cdot)$.

In this section we consider the combination of linear appearance variation with a robust error function. In particular, we investigate how to minimize:

$$\sum_{\mathbf{x}} \varrho \left(\left[T(\mathbf{x}) + \sum_{i=1}^{m} \lambda_i A_i(\mathbf{x}) - I(\mathbf{W}(\mathbf{x};\mathbf{p})) \right]^2 \right) \tag{51}$$

simultaneously with respect to the warp $\mathbf{p}$ and appearance $\boldsymbol{\lambda}$ parameters. We begin in Section 4.1 with a review of the inverse compositional iteratively reweighted least squares algorithm [1] to optimize Equation (50). We proceed in Sections 4.2 and 4.3 to describe robust extensions to the simultaneous and normalization algorithms of Section 3. Both of these algorithms are very inefficient. Hence we also derive efficient approximations based on spatial coherence [1]. It is not

possible to generalize the project out algorithm because there is no notion of orthogonality for a robust error function. We explain why in more detail in Section 4.4. We end in Section 4.5 by empirically evaluating the robust appearance variation algorithms and their efficient approximations.

4.1 Background: IC Iteratively Reweighted Least Squares

4.1.1 Goal of the Algorithm

The *inverse compositional iteratively reweighted least squares algorithm* minimizes the expression in Equation (50) by iteratively approximately minimizing:

$$\sum_{\mathbf{x}} \varrho \left(\left[T(\mathbf{W}(\mathbf{x}; \Delta\mathbf{p})) - I(\mathbf{W}(\mathbf{x}; \mathbf{p})) \right]^2 \right) \tag{52}$$

with respect to $\Delta\mathbf{p}$ and then updating the warp $\mathbf{W}(\mathbf{x}; \mathbf{p}) \leftarrow \mathbf{W}(\mathbf{x}; \mathbf{p}) \circ \mathbf{W}(\mathbf{x}; \Delta\mathbf{p})^{-1}$. The proof of the first order equivalence between iterating these two steps and the *forwards additive* (Lucas-Kanade) minimization of the expression in Equation (51) is contained in [1].

4.1.2 Derivation of the Algorithm

Performing a first order Taylor expansion on $T(\mathbf{W}(\mathbf{x}; \Delta\mathbf{p}))$ in Equation (52) gives:

$$\sum_{\mathbf{x}} \varrho \left(\left[T(\mathbf{x}) + \boldsymbol{\nabla} T \frac{\partial \mathbf{W}}{\partial \mathbf{p}} \Delta\mathbf{p} - I(\mathbf{W}(\mathbf{x}; \mathbf{p})) \right]^2 \right) \tag{53}$$

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

$$\sum_{\mathbf{x}} \varrho \left(E(\mathbf{x})^2 + 2E(\mathbf{x}) \mathbf{SD}_{\text{ic}}(\mathbf{x}) \Delta\mathbf{p} + \Delta\mathbf{p}^{\text{T}} \mathbf{SD}_{\text{ic}}^{\text{T}}(\mathbf{x}) \mathbf{SD}_{\text{ic}}(\mathbf{x}) \Delta\mathbf{p} \right) \tag{54}$$

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

$$\sum_{\mathbf{x}} \left(\varrho \left(E(\mathbf{x})^2 \right) + 2\varrho' \left(E(\mathbf{x})^2 \right) \left[E(\mathbf{x}) \mathbf{SD}_{\text{ic}}(\mathbf{x}) \Delta\mathbf{p} + \Delta\mathbf{p}^{\text{T}} \mathbf{SD}_{\text{ic}}^{\text{T}}(\mathbf{x}) \mathbf{SD}_{\text{ic}}(\mathbf{x}) \Delta\mathbf{p} \right] \right). \tag{55}$$

Inverse Compositional Iteratively Reweighted Least Squares

Pre-compute:

(3) Evaluate the gradient $\boldsymbol{\nabla} T$ 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 $\mathbf{SD}_{\text{ic}}(\mathbf{x})$

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 $E(\mathbf{x}) = T(\mathbf{x}) - I(\mathbf{W}(\mathbf{x}; \mathbf{p}))$
(6) Compute the Hessian matrix H_ϱ using Equation (57)
(7) Compute $\sum_{\mathbf{x}} \varrho' \left((E(\mathbf{x}))^2 \right) \mathbf{SD}_{\text{ic}}^{\text{T}}(\mathbf{x}) E(\mathbf{x})$
(8) Compute $\Delta \mathbf{p}$ using Equation (56)
(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 11: The inverse compositional iteratively reweighted least squares algorithm consists of iteratively applying Equation (56) and updating the warp $\mathbf{W}(\mathbf{x}; \mathbf{p}) \leftarrow \mathbf{W}(\mathbf{x}; \mathbf{p}) \circ \mathbf{W}(\mathbf{x}; \Delta \mathbf{p})^{-1}$. Because the Hessian H_ϱ depends on the warp parameters $\mathbf{p}$ it must be re-computed in every iteration. The naive implementation of this algorithm is almost as slow as the original Lucas-Kanade algorithm. See Table 6 for the details.

Table 6: The computational cost of the inverse compositional iteratively reweighted least squares algorithm. The cost of each iteration is $\text{O}(n^2 N + n^3)$ which is asymptotically as slow as the Lucas-Kanade algorithm. Since the algorithm is so slow, in [1] we considered two efficient approximations to it: (1) the *H*-Algorithm [9, 11] and (2) an algorithm that takes advantage of the spatial coherence of outliers. Both of these approximations move (most of) the cost of computing the Hessian into the pre-computation.

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

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

The minimum of this quadratic form is attained at:

$$\Delta \mathbf{p} = -H_\varrho^{-1} \sum_{\mathbf{x}} \varrho' \left(E(\mathbf{x})^2 \right) \mathbf{SD}_{\text{ic}}^{\text{T}}(\mathbf{x}) E(\mathbf{x}) \tag{56}$$

where:

$$H_\varrho = \sum_{\mathbf{x}} \varrho' \left(E(\mathbf{x})^2 \right) \mathbf{SD}_{\text{ic}}^{\text{T}}(\mathbf{x})\, \mathbf{SD}_{\text{ic}}(\mathbf{x}) \tag{57}$$

is the Hessian matrix. The algorithm is summarized in Figure 11.

The computational cost of the iteratively reweighted least squares algorithm is summarized in Table 6. The algorithm is as slow as the Lucas-Kanade algorithm. Since the algorithm is so slow,

in [1] we considered two efficient approximations to it: (1) the *H*-algorithm [9, 11] and (2) an algorithm that takes advantage of the spatial coherence of outliers. Both of these approximations move (most of) the cost of computing the Hessian into the pre-computation. Empirically we found that the second of these two algorithms performs far better than the first [1].

4.1.3 Spatial Coherence Approximation

In the spatial coherence approximation the template is subdivided into a set of sub-templates or *blocks.* Usually, if the template is rectangular, the blocks are sub-rectangles, although other choices are possible. Suppose there are K blocks $B_1, B_2, \ldots, B_K$ with N_i pixels in the i^{th} block. Equation (57) can then be rewritten as:

$$H_\varrho \;=\; \sum_{i=1}^{K} \sum_{\mathbf{x} \in B_i} \varrho' \left(E(\mathbf{x})^2 \right) \mathbf{SD}_{\text{ic}}^{\text{T}}(\mathbf{x})\, \mathbf{SD}_{\text{ic}}(\mathbf{x}). \tag{58}$$

Based on the spatial coherence of the outliers [1], assume that $\varrho'(E(\mathbf{x})^2)$ is constant in each block; i.e. assume $\varrho'(E(\mathbf{x})^2) = \varrho'_i$, say, for all $\mathbf{x} \in B_i$. In practice this assumption only holds approximately and so ϱ'_i must be estimated from $\varrho'(E(\mathbf{x})^2)$, for example by setting it to be the mean value computed over the block [1]. Equation (58) can then be rearranged to:

$$H_\varrho \;=\; \sum_{i=1}^{K} \varrho'_i \sum_{\mathbf{x} \in B_i} \mathbf{SD}_{\text{ic}}^{\text{T}}(\mathbf{x})\, \mathbf{SD}_{\text{ic}}(\mathbf{x}). \tag{59}$$

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

$$H_i \;=\; \sum_{\mathbf{x} \in B_i} \mathbf{SD}_{\text{ic}}^{\text{T}}(\mathbf{x})\, \mathbf{SD}_{\text{ic}}(\mathbf{x}). \tag{60}$$

The Hessian H_i is the Hessian for the sub-template B_i and can be precomputed. Equation (59) then simplifies to:

$$H_\varrho \;=\; \sum_{i=1}^{K} \varrho'_i \cdot H_i. \tag{61}$$

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 11 and Table 6.

The spatial coherence approximation to the inverse compositional iteratively reweighted least squares algorithm then just consists of using Equation (61) to estimate the Hessian in Step 6 rather than Equation (57). Using Equation (61), 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 $\mathrm{O}(N \cdot n^2)$.

4.2 Simultaneous IC Iteratively Reweighted Least Squares

4.2.1 Goal of the Algorithm

The robust simultaneous inverse compositional algorithm operates by iteratively minimizing:

$$\sum_{\mathbf{x}} \varrho \left(\left[T(\mathbf{W}(\mathbf{x}; \Delta\mathbf{p})) + \sum_{i=1}^{m} (\lambda_i + \Delta\lambda_i) A_i(\mathbf{W}(\mathbf{x}; \Delta\mathbf{p})) - I(\mathbf{W}(\mathbf{x}; \mathbf{p})) \right]^2 \right) \tag{62}$$

simultaneously with respect to $\Delta\mathbf{p}$ and $\Delta\boldsymbol{\lambda} = (\Delta\lambda_1, \ldots, \Delta\lambda_m)^{\mathrm{T}}$, and then updating the warp $\mathbf{W}(\mathbf{x}; \mathbf{p}) \leftarrow \mathbf{W}(\mathbf{x}; \mathbf{p}) \circ \mathbf{W}(\mathbf{x}; \Delta\mathbf{p})^{-1}$ and the appearance parameters $\boldsymbol{\lambda} \leftarrow \boldsymbol{\lambda} + \Delta\boldsymbol{\lambda}$.

4.2.2 Derivation of the Algorithm

The equivalent of Equation (22) in Section 3.1 is:

$$\sum_{\mathbf{x}} \varrho \left(\left[E_{\mathrm{sim}}(\mathbf{x}) + \left(\boldsymbol{\nabla} T + \sum_{i=1}^{m} \lambda_i \boldsymbol{\nabla} A_i \right) \frac{\partial \mathbf{W}}{\partial \mathbf{p}} \Delta\mathbf{p} + \sum_{i=1}^{m} A_i(\mathbf{x}) \Delta\lambda_i \right]^2 \right) \tag{63}$$

where $E_{\mathrm{sim}}(\mathbf{x})$ is defined in Equation (25). Then, following Section 4.1.2 leads to:

$$\Delta\mathbf{p} = -H_{\mathrm{sim},\varrho}^{-1} \sum_{\mathbf{x}} \varrho' \left(E_{\mathrm{sim}}(\mathbf{x})^2 \right) \mathbf{S}\mathbf{D}_{\mathrm{sim}}^{\mathrm{T}}(\mathbf{x}) E_{\mathrm{sim}}(\mathbf{x}) \tag{64}$$

The Robust Simultaneous Inverse Compositional Algorithm

Pre-compute:

(3) Evaluate the gradients $\boldsymbol{\nabla} T$ and $\boldsymbol{\nabla} A_i$ for $i = 1, \ldots, m$
(4) Evaluate the Jacobian $\frac{\partial \mathbf{W}}{\partial \mathbf{p}}$ at $(\mathbf{x}; \mathbf{0})$

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 $E_{\text{sim}}(\mathbf{x})$ using Equation (25)
(5) Compute the steepest descent images $\mathbf{SD}_{\text{sim}}(\mathbf{x})$ using Equation (24)
(6) Compute the Hessian matrix $H_{\text{sim},\varrho}$ using Equation (65) and invert it
(7) Compute $\sum_{\mathbf{x}} \varrho' \left(E_{\text{sim}}(\mathbf{x})^2 \right) \mathbf{SD}_{\text{sim}}^{\text{T}}(\mathbf{x}) E_{\text{sim}}(\mathbf{x})$
(8) Compute $\Delta \mathbf{q} = -H_{\text{sim},\varrho}^{-1} \sum_{\mathbf{x}} \varrho' \left(E_{\text{sim}}(\mathbf{x})^2 \right) \mathbf{SD}_{\text{sim}}^{\text{T}}(\mathbf{x}) E_{\text{sim}}(\mathbf{x})$
(9) Update $\mathbf{W}(\mathbf{x}; \mathbf{p}) \leftarrow \mathbf{W}(\mathbf{x}; \mathbf{p}) \circ \mathbf{W}(\mathbf{x}; \Delta \mathbf{p})^{-1}$ and $\boldsymbol{\lambda} \leftarrow \boldsymbol{\lambda} + \Delta \boldsymbol{\lambda}$

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

Figure 12: The robust simultaneous inverse compositional algorithm is almost identical to the Euclidean version in Figure 3. The main difference is that in Steps (6), (7), and (8) the term $\varrho' \left(E_{\text{sim}}(\mathbf{x})^2 \right)$ is added into the summation $\sum_{\mathbf{x}}$. The computational cost is asymptotically the same. See Table 7 for the details.

where:

$$H_{\text{sim},\varrho} \;=\; \sum_{\mathbf{x}} \varrho' \left(E_{\text{sim}}(\mathbf{x})^2 \right) \mathbf{SD}_{\text{sim}}^{\text{T}}(\mathbf{x}) \, \mathbf{SD}_{\text{sim}}(\mathbf{x}) \tag{65}$$

and where $\mathbf{SD}_{\text{sim}}(\mathbf{x})$ is defined in Equation (24). The robust simultaneous inverse compositional algorithm is summarized in Figure 12 and its computational cost in Table 7. Overall the algorithm is asymptotically just as slow as the Euclidean version.

The efficiency approximation proposed in Section 3.1.3 can also be applied to the robust algorithm; i.e. to not update the steepest descent images $\mathbf{SD}_{\text{sim}}(\mathbf{x})$ from iteration to iteration. On its own, however, this approximation is not as effective at improving the efficiency. Although the computation of the steepest descent images in Step (5) can be moved to the pre-computation, the Hessian still depends on the current error image. See Equation (65). In order to also move (most of) Step (6) to the pre-computation, we also need to make the spatial coherence approximation. See Section 4.1.3 for the details. When combined, however, these two approximations lead to a reasonably efficient algorithm which takes time $\mathrm{O}((n+m)^2 K + (n+m) N + (n+m)^3)$ per iteration. In Section 4.5 we empirically evaluate both of these algorithms, efficient and not.

Table 7: The computation cost of the robust simultaneous inverse compositional algorithm is asymptotically exactly the same as the computational cost of the Euclidean version. See Table 3 for comparison.

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

Per Iteration	Step 1	Step 2	Step 5	Step 6
	$O(n\,N)$	$O(m\,N)$	$O((n+m)\,N)$	$O((n+m)^2\,N + (n+m)^3)$

Step 7	Step 8	Step 9	Total
$O((n+m)\,N)$	$O((n+m)^2)$	$O(n^2+m)$	$O((n+m)^2\,N + (n+m)^3)$

4.3 Normalization IC Iteratively Reweighted Least Squares

4.3.1 Goal of the Algorithm

The key step in the Euclidean normalization algorithm in Section 3.3 is to normalize the input image $I(\mathbf{W}(\mathbf{x};\mathbf{p}))$ so that its component in the direction A_i for $i = 1, \ldots, m$ is the same as that of the template $T(\mathbf{x})$. It turns out that this normalization can instead by applied to the error image. See Equation (47) for the details. Unfortunately, it is no longer possible to use Equation (47) to perform the normalization because Equation (47) relies on the fact that the appearance vectors are orthonormal. With a robust error function the appearance vectors are no longer orthonormal.

4.3.2 Derivation of the Algorithm

The goal of the normalization step in Equation (47) is to make the component of the error image in the direction A_i to be zero, whilst computing λ_i at the same time. We now need to formulate this problem using the robust error function. Suppose we have a current estimate of the error image:

$$E_{\text{norm}}(\mathbf{x}) \;=\; T(\mathbf{x}) + \sum_{i=1}^{m} \lambda_i A_i(\mathbf{x}) - I(\mathbf{W}(\mathbf{x};\mathbf{p})). \tag{66}$$

We then wish to compute updates to the appearance parameters $\Delta\boldsymbol{\lambda} = (\Delta\lambda_1, \ldots, \Delta\lambda_m)^{\mathrm{T}}$ that minimize:

$$\sum_{\mathbf{x}} \varrho' \left(E_{\mathrm{norm}}(\mathbf{x})^2 \right) \left[E_{\mathrm{norm}}(\mathbf{x}) + \sum_{i=1}^{m} \Delta\lambda_i A_i(\mathbf{x}) \right]^2. \tag{67}$$

The least squares minimum of this expression is:

$$\Delta\boldsymbol{\lambda} \;=\; -H_{\mathbf{A}}^{-1} \sum_{\mathbf{x}} \varrho' \left(E_{\mathrm{norm}}(\mathbf{x})^2 \right) \mathbf{A}^{\mathrm{T}}(\mathbf{x}) E_{\mathrm{norm}}(\mathbf{x}) \tag{68}$$

where $\mathbf{A}(\mathbf{x}) = (A_1(\mathbf{x}), \ldots, A_m(\mathbf{x}))$ and $H_{\mathbf{A}}$ is the appearance Hessian:

$$H_{\mathbf{A}} \;=\; \sum_{\mathbf{x}} \varrho' \left(E_{\mathrm{norm}}(\mathbf{x})^2 \right) \mathbf{A}(\mathbf{x})^{\mathbf{T}} \mathbf{A}(\mathbf{x}). \tag{69}$$

The robust equivalent of the normalization in Equation (47) is therefore to compute $\Delta\boldsymbol{\lambda}$ using Equations (68) and (69). The appearance parameters are then updated $\boldsymbol{\lambda} \leftarrow \boldsymbol{\lambda} + \Delta\boldsymbol{\lambda}$ and the error image is updated using Equation (66). The Hessian (and steepest descent parameter updates) also needs to include the robust error function:

$$H_{\mathrm{norm},\varrho} \;=\; \sum_{\mathbf{x}} \varrho' \left(E_{\mathrm{norm}}(\mathbf{x})^2 \right) \mathbf{SD}_{\mathrm{ic}}^{\mathrm{T}}(\mathbf{x}) \, \mathbf{SD}_{\mathrm{ic}}(\mathbf{x}). \tag{70}$$

The robust normalization algorithm is summarized in Figure 13 and the computational cost in Table 8. Overall the computation cost of the robust normalization inverse compositional algorithm is far more than the Euclidean version in Table 5. The computation of the Hessian in Step (6) must be moved to the per-iteration computation. The error image normalization in Step (2a) also becomes far more computationally demanding because of the computation of the appearance Hessian in Equation (69). It is, however, possible to apply the spatial coherence approximation (see Section 4.1.3) to both the computation of the Hessian in Step (6) and the computation of the appearance Hessian in Step (2a). Most of the computation of these two computationally demanding steps can therefore be moved into the pre-computation. When both of the Hessians are computed with the spatial coherence approximation, the result is a reasonably efficient algorithm which takes

The Robust Normalization 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 $\mathbf{SD}_{\text{ic}}(\mathbf{x})$ using Equation (16)

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 $E_{\text{norm}}(\mathbf{x})$ using Equation (66)
(2a) Compute $\Delta\boldsymbol{\lambda}$ using Equations (68) and (69) and update $\boldsymbol{\lambda}$ and $E_{\text{norm}}(\mathbf{x})$
(6) Compute the Hessian matrix $H_{\text{norm},\varrho}$ using Equation (70) and invert it
(7) Compute $\sum_{\mathbf{x}} \varrho' \left(E_{\text{norm}}(\mathbf{x})^2 \right) \mathbf{SD}_{\text{ic}}^{\text{T}}(\mathbf{x}) E_{\text{norm}}(\mathbf{x})$
(8) Compute $\Delta\mathbf{p} = -H_{\text{norm},\varrho}^{-1} \sum_{\mathbf{x}} \varrho' \left(E_{\text{norm}}(\mathbf{x})^2 \right) \mathbf{SD}_{\text{ic}}^{\text{T}}(\mathbf{x}) E_{\text{norm}}(\mathbf{x})$
(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 13: The robust normalization inverse compositional algorithm is similar to the Euclidean version in Figure 5. The only changes are the robust method to normalize the error image in Steps (2a) and (2), and the incorporation of the weighting function $\varrho' \left(E_{\text{norm}}(\mathbf{x})^2 \right)$ into Steps (6), (7), and (8).

Table 8: The computation cost of the robust normalization inverse compositional algorithm is far more than the Euclidean version in Table 5. The computation of the Hessian in Step (6) must be moved to the per-iteration computation. The error image normalization in Step (2a) also becomes far more computationally demanding, primarily because of the computation of the appearance Hessian in Equation (69).

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

Per Iteration	Step 1	Step 2	Step 2a	Step 6
	$\mathrm{O}(n\,N)$	$\mathrm{O}(N + m\,N)$	$\mathrm{O}(m^2\,N)$	$\mathrm{O}(n^2\,N + n^3)$

Step 7	Step 8	Step 9	Total
$\mathrm{O}(n\,N)$	$\mathrm{O}(n^2)$	$\mathrm{O}(n^2)$	$\mathrm{O}(n^2\,N + m^2\,N + n^3)$

time $\mathrm{O}((n^2 + m^2)\,K + (n + m)\,N + n^3)$ per iteration. In Section 4.5 we empirically evaluate both of these algorithms, the robust normalization algorithm and the efficient approximation to it.

4.4 Project Out IC Iteratively Reweighted Least Squares

The Euclidean project out algorithm uses linear algebra to turn a simultaneous optimization over $\mathbf{p}$ and $\boldsymbol{\lambda}$ into a sequential optimization, first over $\mathbf{p}$ and then over $\boldsymbol{\lambda}$. The key step in the derivation

is the transition from Equation (29) to Equation (30) using orthogonality. Unfortunately, when we move to the robust error function in Equation (51), the equivalent of this step is not possible. The error image $E_{\text{sim}}(\mathbf{x}) = T(\mathbf{x}) + \sum_{i=1}^{m} \lambda_i A_i(\mathbf{x}) - I(\mathbf{W}(\mathbf{x};\mathbf{p}))$ cannot be decomposed into the sum of its components in orthogonal subspaces. Orthogonality is not even defined for a general $\varrho(\cdot)$. There is therefore no[1] robust project out algorithm that sequentially solves for $\mathbf{p}$ and then $\boldsymbol{\lambda}$.

An approximation is to ignore the lack of orthogonality and just continue to use the Euclidean project out steepest descent images. This approach is taken in [11], where the *H*-Algorithm [9] is also used to keep the Hessian constant to yield an efficient algorithm. In the following section we empirically compare such an algorithm with the robust simultaneous and normalization algorithms.

4.5 Experimental Results

We now evaluate 5 of these robust appearance variation algorithms: (1) the robust simultaneous inverse compositional algorithm (RSIC), (2) the efficient approximation to the robust simultaneous inverse compositional algorithm using the spatial coherence approximation to the Hessian (RSIC-EA-SC), (3) the robust normalization inverse compositional algorithm (RNIC), (4) the efficient spatial coherence approximation to the normalization inverse compositional algorithm (RNIC-SC), and (5) a robust project out algorithm very similar to the Hager-Belhumeur algorithm of [11] which ignores the lack of orthogonality of the appearance images, keeping the steepest descent images constant, and approximates the Hessian with the *H*-algorithm [9] (RPO-H).

As described in [1], evaluating robust fitting algorithms is difficult because there is no obvious noise model to use. As in [1], we assume that the main cause of noise (i.e. outliers) is occlusion and generate the input image in the following manner. The evaluations are governed by one parameter,

[1]Note that it is possible to derive robust variants of the project out algorithm by: (1) projecting the steepest descent images into the subspace orthogonal to the appearance variation using the weighted L2 norm with weighting function $\varrho'(E_{\text{sim}}(\mathbf{x})^2)$, or (2) re-orthonormalizing the appearance images every iteration with respect to the same weighted L2 norm. Neither of these algorithms sequentially solve for $\mathbf{p}$ and then $\boldsymbol{\lambda}$. The appearance parameters $\boldsymbol{\lambda}$ must be computed every iteration to estimate $\varrho'(E_{\text{sim}}(\mathbf{x})^2)$. Moreover both of these variants are slower than the robust normalization algorithm and perform almost identically for the same reason the project out and normalization algorithms perform identically in Section 3.4.2. Since: (1) these algorithms are not "true" project out algorithms, (2) they are slower than the robust normalization algorithm, and (3) they perform no better than it, we do not describe them in this paper.

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. In [1] we used a variety of occluding images. For lack of space, in this paper we just use an occluder extracted from an image of natural scenery.

Another thing that makes evaluating robust fitting algorithms hard is choosing the robust error function. As in [1] we use the robust error function:

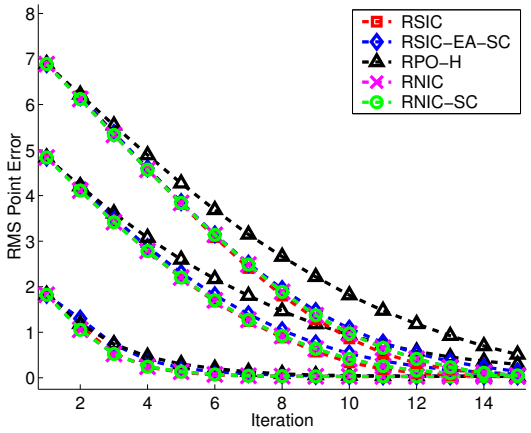
$$\varrho(t;\sigma_1) \;=\; \begin{cases} t & \text{if } 0 \leq t \leq \sigma_1 \\ \sigma_1 & \text{if } t > \sigma_1. \end{cases} \tag{71}$$

This function classifies pixels as outliers if the magnitude of the error is larger than the scale parameter σ_1. Inliers are weighted equally and outliers are given zero weight. We estimate the scale parameter σ_1 by assuming that we know the number of outliers. We then estimate σ_1 by sorting the error values and setting σ_1 so that the correct number of pixels are classified as outliers.

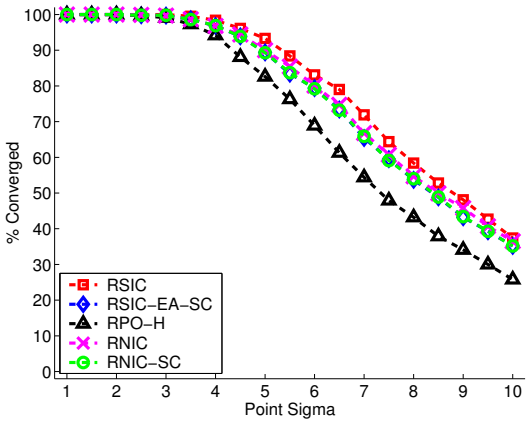
4.5.1 Experiment 6: Varying the Percentage of Occlusion

We first investigate the variation in performance of the 5 algorithms when varying the percentage of occlusion. In this experiment we use $m = 1$, set A_1 to be an image of some scenery as in Section 3.4.3, and set $\lambda_1 = 0.35 \times \frac{||T||}{||A_1||}$. The results for 10% occlusion are shown in Figures 14(a) and (b), for 30% occlusion in Figures 14(c) and (d), and for 50% occlusion in Figures 14(e) and (f). For lack of space, we only include plots of the rate of convergence and frequency of convergence.

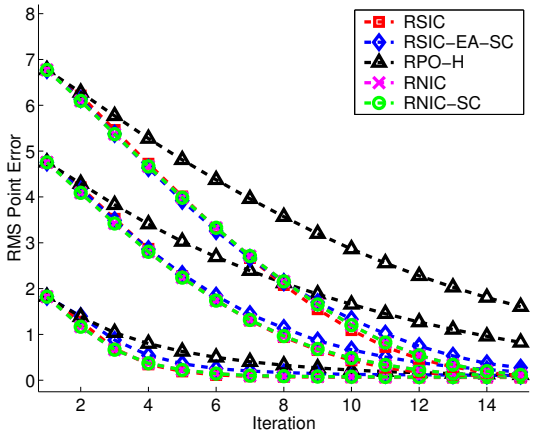
As expected the robust simultaneous inverse compositional algorithm performs the best with the robust normalization algorithm performing slightly worse, particularly for higher levels of occlusion. The two efficient versions of these algorithms (RSIC-EA-SIC and RNIC-SC) both perform similarly and slightly worse than the robust normalization algorithm. The robust project out algorithm, ignoring orthogonality and using the H-algorithm (RPO-H), performs far worse,



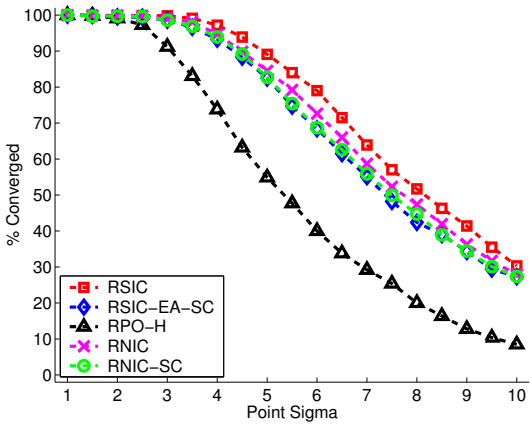
(a) Convergence Rate, 10% Occlusion



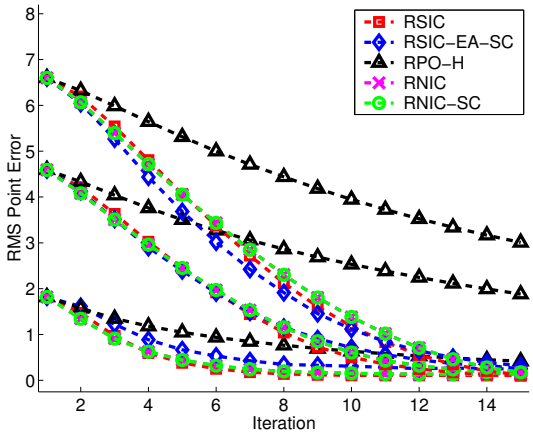
(b) Convergence Frequency, 10% Occlusion



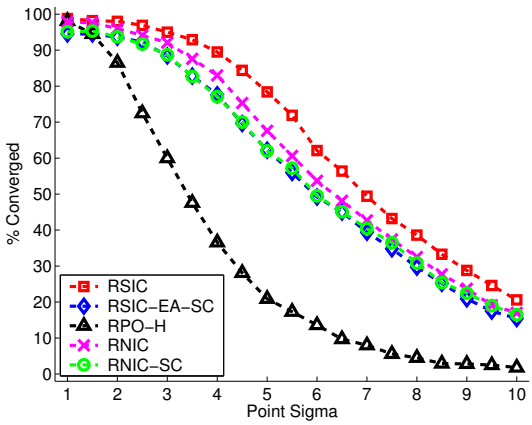
(a) Convergence Rate, 30% Occlusion



(b) Convergence Frequency, 30% Occlusion



(a) Convergence Rate, 50% Occlusion



(b) Convergence Frequency, 50% Occlusion

Figure 14: A comparison of 5 robust appearance variation algorithms: (1) the robust simultaneous algorithm (RSIC), (2) the efficient approximation to RSIC using the spatial coherence approximation to the Hessian (RSIC-EA-SC), (3) the robust normalization algorithm (RNIC), (4) the efficient spatial coherence approximation to RNIC (RNIC-SC), and (5) a robust project out algorithm which ignores the lack of orthogonality of the appearance images and approximates the Hessian with the H-algorithm (RPO-H). RSIC performs the best, and RPO-H by far the worst, highlighting the importance of using the right algorithm.

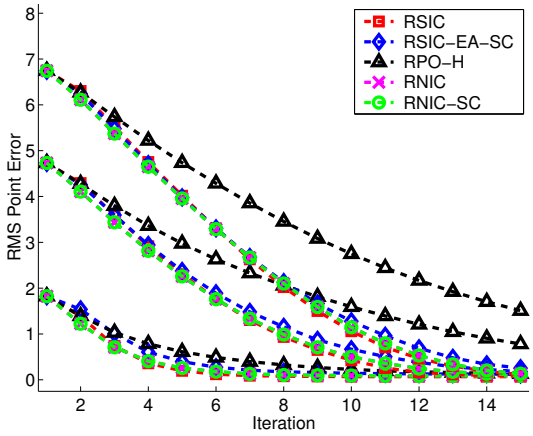
especially for larger levels of occlusion. These poor results illustrate how important it is to use the right algorithm. A minor modification to any of the algorithms can result in very poor performance.

4.5.2 Experiment 7: Varying λ_1

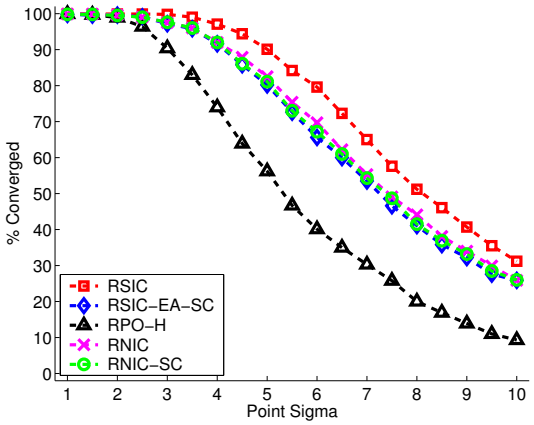
Next we investigate the variation in performance of the 5 algorithms when varying λ_1. We still set $m = 1$ and A_1 to be an image of some scenery. We set the percentage of occlusion to be 30%. The results for $\lambda_1 = 0.5 \times \frac{\|T\|}{\|A_1\|}$ are shown in Figures 15(a) and (b), for $\lambda_1 = 0.75 \times \frac{\|T\|}{\|A_1\|}$ in Figures 15(c) and (d), and for $\lambda_1 = 1.0 \times \frac{\|T\|}{\|A_1\|}$ in Figures 15(e) and (f). The results are similar to Experiment 6 with RSIC performing the best followed by RNIC. As in the experimental results in Section 3 the gap between these two algorithms increases with λ_1. Although the efficient variants of these algorithms (RSIC-EA-SC and RNIC-SC) perform similarly for $\lambda_1 = 0.5 \times \frac{\|T\|}{\|A_1\|}$ and $\lambda_1 = 0.75 \times \frac{\|T\|}{\|A_1\|}$, for $\lambda_1 = 1.0 \times \frac{\|T\|}{\|A_1\|}$ the efficient algorithms perform significantly worse. Of the two, the efficient variation of the normalization algorithm (RNIC-SC) performs slightly better. As in Experiment 6 the robust project out algorithm (RPO-H) performs quite poorly in all cases.

4.5.3 Experiment 8: Varying the Number of Appearance Images

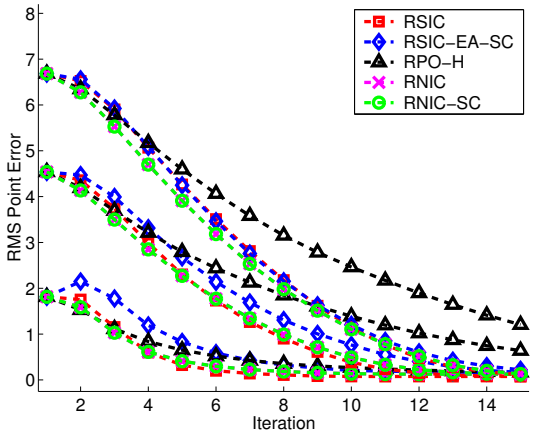
Finally we investigate the variation in performance with the number of appearance images. We fix the percentage of occlusion to be 30% and otherwise follow the procedure in Experiment 3 in Section 3.4.3. We chose the appearance images by taking a large image of some scenery and randomly selecting a number of sub-images of the appropriate size. The results are then orthonormalized and used as the A_i. To avoid any dependence on the magnitude of the appearance variation, we chose the appearance parameters such that $\sum_i \lambda_i^2 \|A_i\|^2 = (0.35)^2 \times \|T\|^2$ as we vary the number of appearance images. We also set $\lambda_i^2 \|A_i\|^2$ to be the same for each i to spread out the appearance variation equally. The results for $m = 2$ are shown in Figures 16(a) and (b), for $m = 3$ in Figures 14(c) and (d), and for $m = 5$ in Figures 14(e) and (f). The results are similar to those in Experiment 3. The performance of the algorithms does not vary significantly with the number of



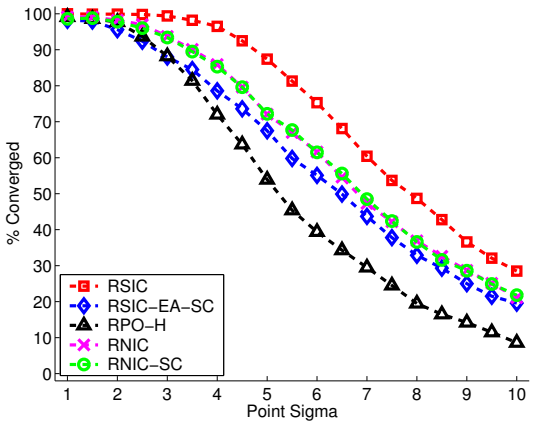
(a) Convergence Rate, $\lambda_1 = 0.5 \times \frac{||T||}{||A_1||}$



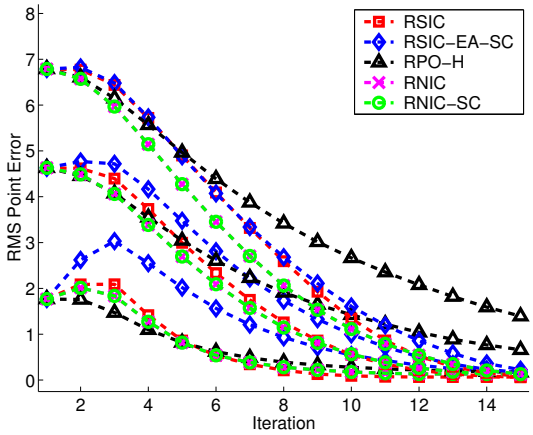
(b) Convergence Frequency, $\lambda_1 = 0.5 \times \frac{||T||}{||A_1||}$



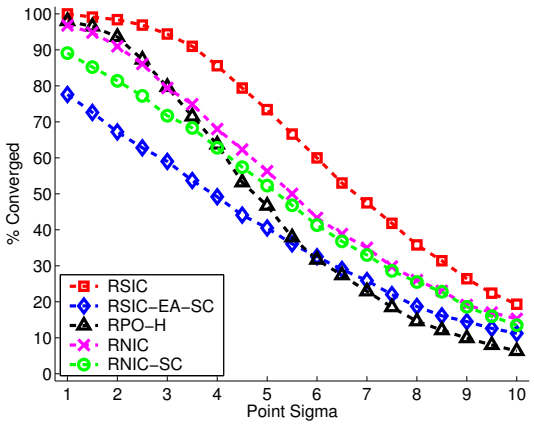
(c) Convergence Rate, $\lambda_1 = 0.75 \times \frac{||T||}{||A_1||}$



(d) Convergence Frequency,$\lambda_1 = 0.75 \times \frac{||T||}{||A_1||}$

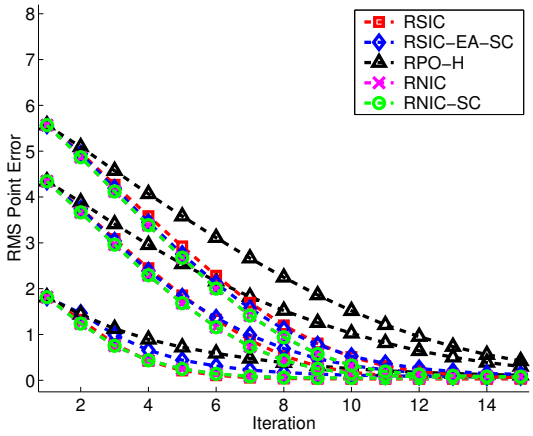


(e) Convergence Rate, $\lambda_1 = 1.0 \times \frac{||T||}{||A_1||}$

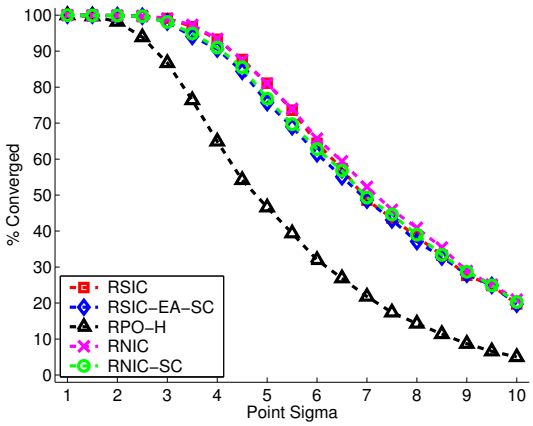


(f) Convergence Frequency,$\lambda_1 = 1.0 \times \frac{||T||}{||A_1||}$

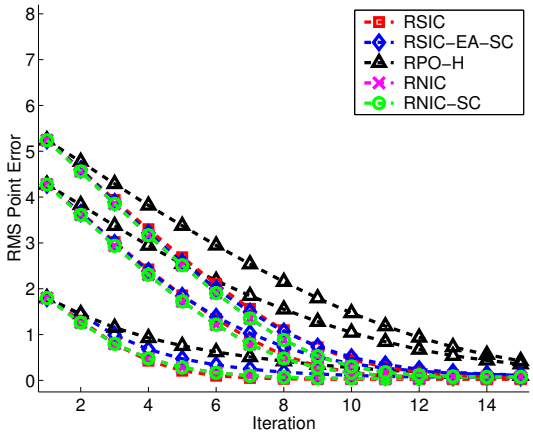
Figure 15: An evaluation of the same 5 algorithms in Figure 14 varying the appearance parameter λ_1 while keeping the percentage of occlusion fixed at 30%. All other experimental conditions remain the same. As in Figure 14, RSIC performs the best with RNIC close behind. The gap between these algorithms, however, increases with λ_1. The efficient algorithms RSIC-EA-SC and RNIC-SC before similarly in (a)–(d) but quite poorly for larger λ_1 in (e) and (f). As in Figure 14, RPO-H performs fairly poorly in all three cases.



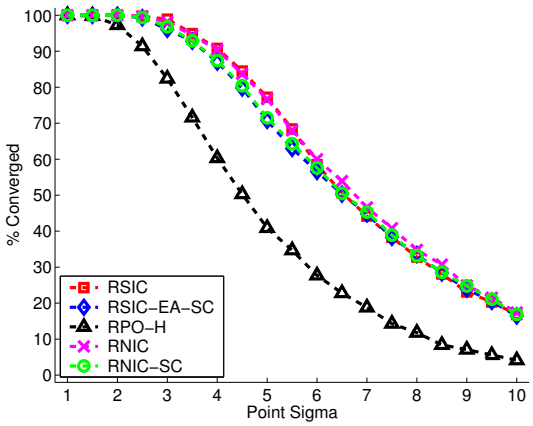
(a) Convergence Rate, $m = 2$



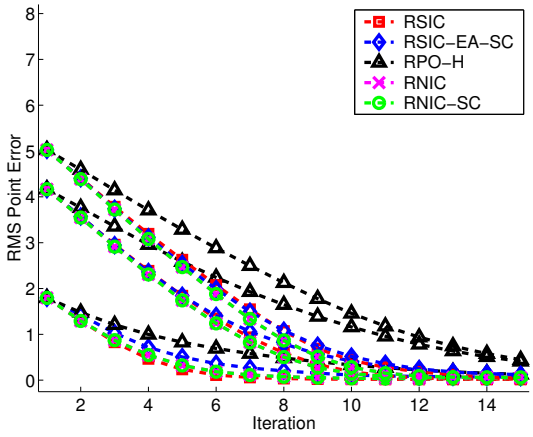
(b) Convergence Frequency, $m = 2$



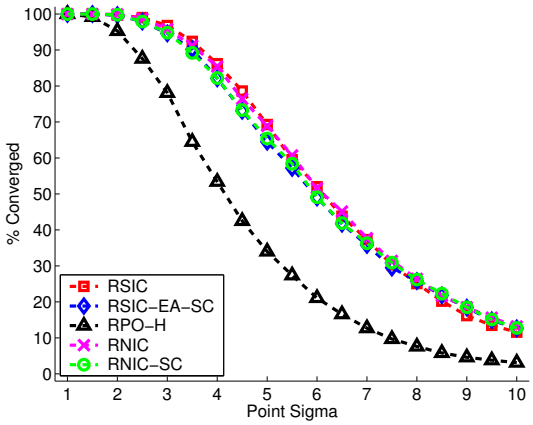
(c) Convergence Rate, $m = 3$



(d) Convergence Frequency, $m = 3$



(e) Convergence Rate, $m = 5$



(f) Convergence Frequency, $m = 5$

Figure 16: An evaluation of the same 5 algorithms in Figure 14 varying the number of appearance images m while keeping the total amount of appearance variation fixed at $\sum_i \lambda_i^2 \|A_i\|^2 = (0.35)^2 \times \|T\|^2$. The results are similar to Figure 3.4.3 in Experiment 3. The 4 algorithms RSIC, RSIC-EA-SC, RNIC, RNIC-SC all perform almost identically in all 3 cases, with RPO-H performing significantly worse.

Table 9: A summary of the 6 algorithms we described in Section 3 for image alignment with linear appearance variation using the Euclidean L2 norm. There are 3 main algorithms: (1) the simultaneous inverse compositional algorithm, (2) the project out inverse compositional algorithm, and (3) the normalization inverse compositional algorithm. Each of the 3 main algorithms has one variant.

Algorithm	Computational Cost	Performance	Gain OK
Simultaneous IC	$\mathrm{O}((n+m)^2 N + (n+m)^3)$	Good	Yes
Project Out IC	$\mathrm{O}(n N + n^2)$	Medium	No
Normalization IC	$\mathrm{O}((n+m) N + n^2)$	Medium	No
Efficient Approximation to Simultaneous IC	$\mathrm{O}((n+m) N + (n+m)^2)$	Medium	No
Project Out IC with SS Correction	$\mathrm{O}(n N + n^2)$	Medium	Yes
Normalization IC with SS Correction	$\mathrm{O}((n+m) N + n^2)$	Medium	Yes

appearance images. (Only the magnitude of the appearance variation is important.) The 4 algorithms RSIC, RSIC-EA-SC, RNIC, RNIC-SC all perform almost identically in all 3 cases. As in Experiments 6 and 7, RPO-H performs significantly worse that the other 4 algorithms.

5 Conclusion

5.1 Summary

In Section 3 we investigated the problem of image alignment with linear appearance variation using the Euclidean L2 norm. We described 3 main algorithms: (1) the simultaneous inverse compositional algorithm, (2) the project out inverse compositional algorithm, and (3) the normalization inverse compositional algorithm. We also described an efficient approximation to the simultaneous algorithm, and step-size corrections to the project out and normalization algorithms. These algorithms are summarized in Table 9. Of the 6 algorithms, the simultaneous algorithm performs the best, but unfortunately is very slow. The project out, normalization, and efficient approximation to the simultaneous algorithm are all efficient, but perform significantly worse in two main scenarios: (1) when the magnitude of the appearance variation is large (comparable in magnitude to the template itself), and (2) when the algorithm is used to model gain. The step-size correction variants of the project-out algorithm and normalization algorithm perform well in the second of these two cases, but still break down in the presence of general large magnitude appearance variation.

Table 10: A summary of the 4 algorithms we described in Section 4 for image alignment with linear appearance variation using a robust error function. There are 2 main algorithms: (1) the robust simultaneous inverse compositional algorithm, and (2) the robust normalization inverse compositional algorithm. Both of these algorithms are very slow and so for each one we derived an efficient approximation to it.

Algorithm	Computational Cost	Performance
Robust Simultaneous IC (RSIC)	$\mathrm{O}((n+m)^2\,N + (n+m)^3)$	Good
Robust Normalization IC (RNIC)	$\mathrm{O}((n^2+m^2)\,N + n^3)$	Medium
Efficient RSIC (RSIC-EA-SC)	$\mathrm{O}((n+m)^2\,K + (n+m)\,N + (n+m)^3)$	Medium
Efficient RNIC (RNIC-SC)	$\mathrm{O}((n^2+m^2)\,K + (n+m)\,N + n^3)$	Medium

Another point to note is that the project out, normalization, and efficient approximation to the simultaneous algorithm perform almost identically in all of our experiments. The project out and normalization algorithms both work in the subspace orthogonal to the appearance variation. The project out algorithm projects the steepest descent images into this subspace. The normalization algorithm projects the error image into the same subspace. It is therefore not surprising that they perform almost identically. The fact that the efficient approximation to the simultaneous algorithm also performs almost identically is harder to explain. Finding an explanation is left as future work.

In Section 4 we investigated the problem of image alignment with linear appearance variation using a robust error function. We described 2 main algorithms: (1) the robust simultaneous inverse compositional algorithm and (2) the robust normalization inverse compositional algorithm. We also described efficient approximations to both of these algorithms. These four algorithms are summarized in Table 10. Of the 4 algorithms, the robust simultaneous algorithm performs the best, but unfortunately is very slow. The robust normalization algorithm performs similarly, but slightly worse. The efficient approximations to these algorithms both perform fairly well except when the magnitude of the appearance variation is large (comparable to the template itself). We also empirically compared these 4 algorithms with a robust project out algorithm similar to the Hager-Belhumeur algorithm [11] which ignores the lack of orthogonality of the appearance images, keeping the steepest descent images constant, and approximates the Hessian with the H-algorithm [9]. This algorithm performs significantly worse than the 4 algorithms in Table 10.

5.2 Discussion

In Parts 1 and 2 of this series of papers 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 best algorithm to use depends on the noise and the computational requirements. If the image noise is approximately (i.i.d.) Gaussian then one of the Euclidean L2 norm algorithms described in Section 3 should be used. If not, an appropriate robust error function and one of the algorithms in Section 4 should be used instead.

If computational speed is not an issue, the best Euclidean algorithm to use is the simultaneous algorithm. It clearly performs the best in all of our experiments. If high efficiency is required, the project out algorithm is the fastest, and performs as well as any of the other efficient algorithms (the efficient approximation to the simultaneous algorithm and the normalization algorithm.) Care should be taken, however, when there is large appearance variation. In such cases, the project out algorithm will not perform anywhere near as well as the simultaneous algorithm. If the system is being used to model gain, the step-size correction variant of the project out algorithm should be used as it does not slow the algorithm significantly, and increases the robustness substantially.

In terms of the robust algorithms, if computational speed is not an issue the best algorithm to use is the robust simultaneous inverse compositional algorithm. It clearly performs the best in all of our experiments. If efficiency is required, the efficient approximation to the robust normalization inverse compositional algorithm using spatial coherence to compute the Hessian is probably the best choice. Care should be taken, however, when there is large appearance variation. If the magnitude of the appearance variation is more than about half the magnitude of the template, the performance of the efficient robust normalization algorithm will be adversely affected.

5.3 Future Work

In Part 1 of this series of papers we covered the forwards additive, forwards compositional, inverse additive, and inverse compositional algorithms. We also covered the Newton, Gauss-Newton,

steepest-descent, Levenberg-Marquardt, and diagonal Hessian variants of the inverse compositional algorithm. In Part 2 we covered the choice of the error function, and developed algorithms for both weighted L2 norms and robust error functions. In this, Part 3, we covered the addition of linear appearance variation, both with the Euclidean L2 norm and with a robust error function. In the upcoming, and final, Part 4, we will cover the addition of priors on the warp and appearance parameters, both with a Euclidean L2 norm and with a robust error function.

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.ri.cmu.edu/projects/project_515.html. 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 paper was conducted under U.S. Department of Defense contract N41756-03-C4024.

References

[1] S. Baker, R. Gross, I. Matthews, and T. Ishikawa. Lucas-Kanade 20 years on: A unifying framework: Part 2. Technical Report CMU-RI-TR-03-01, Carnegie Mellon University Robotics Institute, 2003.

[2] 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.

[3] 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. *International Journal of Computer Vision*, (accepted to appear in) 2004.

[4] 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.

[5] 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.

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

[7] T. F. Cootes, G. J. Edwards, and C. J. Taylor. Active appearance models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23(6):681–685, June 2001.

[8] 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.

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

[10] M. Gleicher. Projective registration with difference decomposition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 331–337, 1997.

[11] 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.

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

[13] 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.

[14] I. Matthews and S. Baker. Active appearance models revisited. Technical Report CMU-RI-TR-03-02, Carnegie Mellon University Robotics Institute, 2003.

[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.