# Lucas-Kanade 20 Years On: A Unifying Framework: Part 1

**Simon Baker and Iain Matthews**

**CMU-RI-TR-02-16**

## 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 and tracking to layered motion, mosaic-ing, and face coding. Numerous algorithms have been proposed and a wide variety of extensions have been made to the original formulation. We present an overview of image alignment, describing most of the algorithms and their extensions in a consistent framework. We concentrate on the *inverse compositional* algorithm, an efficient algorithm that we recently proposed. We examine which of the extensions to Lucas-Kanade can be used with the inverse compositional algorithm without any significant loss of efficiency, and which cannot. In this paper, Part 1 of a 2 part series, we cover the quantity approximated, the warp update rule, and the gradient descent approximation. In a future Part 2 of this 2 paper series we will cover the choice of the norm, how to allow linear appearance variation, how to impose priors on the parameters, and various heuristics to avoid local minima.

**Keywords:** Image alignment, Lucas-Kanade, a unifying framework, additive vs. compositional algorithms, forwards vs. inverse algorithms, the inverse compositional algorithm, efficiency, steepest descent, Gauss-Newton, Newton, Levenberg-Marquardt.

# 1 Introduction

Image alignment consists of moving, and possibly deforming, a template to minimize the difference between the template and an image. Since the first use of image alignment in the Lucas-Kanade optical flow algorithm [11], image alignment has become one of the most widely used techniques in computer vision. Besides optical flow, some of its other applications include tracking [4, 10], parametric and layered motion estimation [3], mosaic-ing [14], and face coding [2, 6].

The usual approach to image alignment is gradient descent. A variety of other numerical algorithms such as *difference decomposition* [9] and *linear regression* [6] have also been proposed, but gradient descent is the defacto standard. Gradient descent can be performed in variety of different ways, however. For example, one difference between the various approaches is whether they estimate an additive increment to the parameters (the *additive* approach [11]), or whether they estimate an incremental warp that is then composed with the current estimate of the warp (the *compositional* approach [14].) Another difference is whether the algorithm performs a Gauss-Newton, a Newton, a steepest-descent, or a Levenberg-Marquardt approximation in each gradient descent step.

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]. We examine which of the extensions to Lucas-Kanade 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 1 of a 2 part series, we begin in Section 2 by reviewing the Lucas-Kanade algorithm. We proceed in Section 3 to analyze the quantity that is approximated by the various image alignment algorithms and the warp update rule that is used. We categorize algorithms as either *additive* or *compositional*, and as either *forwards* or *inverse*. We prove the first order equivalence of the various alternatives, derive the efficiency of the resulting algorithms, describe the set of warps that each alternative can be applied to, and finally empirically compare the algorithms. In Section 4 we describe the various gradient descent approximations that can be used in each iteration, *Gauss-Newton*, *Newton*, *diagonal Hessian*, *Levenberg-Marquardt*, and *steepest-descent* [12]. We compare these alternative both in terms of speed and in terms of empirical performance. We conclude in Section 5 with a discussion. In Part 2 of this 2 part framework (currently under preparation), we will cover the choice of the error norm, how to allow linear appearance variation, how to add priors on the parameters, and various heuristics to avoid local minima.

# 2 Background: Lucas-Kanade

The original image alignment algorithm was the Lucas-Kanade algorithm [11]. The goal of Lucas-Kanade is to align a template image $T(\mathbf{x})$ to an input image $I(\mathbf{x})$, where $\mathbf{x} = (x, y)^{\mathrm{T}}$ is a column vector containing the pixel coordinates. If the Lucas-Kanade algorithm is being used to compute *optical flow* or to *track* an image patch from time $t = 1$ to time $t = 2$, the template $T(\mathbf{x})$ is an extracted sub-region (a $5 \times 5$ window, maybe) of the image at $t = 1$ and $I(\mathbf{x})$ is the image at $t = 2$.

Let $\mathbf{W}(\mathbf{x};\mathbf{p})$ denote the parameterized set of allowed warps, where $\mathbf{p} = (p_1, \ldots p_n)^{\mathrm{T}}$ is a vector of parameters. The warp $\mathbf{W}(\mathbf{x};\mathbf{p})$ takes the pixel $\mathbf{x}$ in the coordinate frame of the template $T$ and maps it to the sub-pixel location $\mathbf{W}(\mathbf{x};\mathbf{p})$ in the coordinate frame of the image $I$. If we are computing optical flow, for example, the warps $\mathbf{W}(\mathbf{x};\mathbf{p})$ might be the translations:

$$\mathbf{W}(\mathbf{x};\mathbf{p}) = \begin{pmatrix} x + p_1 \\ y + p_2 \end{pmatrix} \tag{1}$$

where the vector of parameters $\mathbf{p} = (p_1, p_2)^{\mathrm{T}}$ is then the optical flow. If we are tracking a larger image patch moving in 3D we may instead consider the set of affine warps:

$$\mathbf{W}(\mathbf{x};\mathbf{p}) = \begin{pmatrix} (1+p_1) \cdot x & + & p_3 \cdot y & + & p_5 \\ p_2 \cdot x & + & (1+p_4) \cdot y & + & p_6 \end{pmatrix} = \begin{pmatrix} 1+p_1 & p_3 & p_5 \\ p_2 & 1+p_4 & p_6 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} \tag{2}$$

where there are 6 parameters $\mathbf{p} = (p_1, p_2, p_3, p_4, p_5, p_6)^{\mathrm{T}}$ as, for example, was done in [3]. (There are other ways to parameterize affine warps. In Part 2 of this framework we will investigate what is the best way.) In general, the number of parameters $n$ may be arbitrarily large and $\mathbf{W}(\mathbf{x};\mathbf{p})$ can be arbitrarily complex. One example of a complex warp is the set of piecewise affine warps used in Active Appearance Models [6], Active Blobs [13], and Flexible Appearance Models [2].

## 2.1 Goal of the Lucas-Kanade Algorithm

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

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

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 (3) 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 (3), the Lucas-Kanade algorithm assumes that a current estimate of $\mathbf{p}$ is known and then iteratively solves for increments to the parameters $\Delta\mathbf{p}$; i.e. the following expression is (approximately) minimized:

$$\sum_{\mathbf{x}} [\, I(\mathbf{W}(\mathbf{x};\mathbf{p} + \Delta\mathbf{p})) - T(\mathbf{x}) \,]^2 \tag{4}$$

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

$$\mathbf{p} \leftarrow \mathbf{p} + \Delta\mathbf{p}. \tag{5}$$

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 $\epsilon$; i.e. $||\Delta\mathbf{p}|| \leq \epsilon$.

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

$$\sum_{\mathbf{x}} \left[ I(\mathbf{W}(\mathbf{x};\mathbf{p})) + \boldsymbol{\nabla} I \frac{\partial \mathbf{W}}{\partial \mathbf{p}} \Delta\mathbf{p} - T(\mathbf{x}) \right]^2. \tag{6}$$

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 $\mathbf{W}(\mathbf{x};\mathbf{p})$. 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:

$$\frac{\partial \mathbf{W}}{\partial \mathbf{p}} = \begin{pmatrix} \frac{\partial W_x}{\partial p_1} & \frac{\partial W_x}{\partial p_2} & \cdots & \frac{\partial W_x}{\partial p_n} \\ \frac{\partial W_y}{\partial p_1} & \frac{\partial W_y}{\partial p_2} & \cdots & \frac{\partial W_y}{\partial p_n} \end{pmatrix}. \tag{7}$$

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 (6). For example, the affine warp in Equation (2) has the Jacobian:

$$\frac{\partial \mathbf{W}}{\partial \mathbf{p}} = \begin{pmatrix} x & 0 & y & 0 & 1 & 0 \\ 0 & x & 0 & y & 0 & 1 \end{pmatrix}. \tag{8}$$

Equation (6) 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 (6) with respect to $\Delta\mathbf{p}$ is:

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

where we refer to $\boldsymbol{\nabla} I \frac{\partial \mathbf{W}}{\partial \mathbf{p}}$ as the *steepest descent* images. (See Section 4.3 for why.) Setting this expression to equal zero and solving gives the closed form solution of Equation (6) as:

$$\Delta\mathbf{p} = H^{-1} \sum_{\mathbf{x}} \left[ \boldsymbol{\nabla} I \frac{\partial \mathbf{W}}{\partial \mathbf{p}} \right]^{\mathrm{T}} [T(\mathbf{x}) - I(\mathbf{W}(\mathbf{x};\mathbf{p}))] \tag{10}$$

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

$$H = \sum_{\mathbf{x}} \left[ \boldsymbol{\nabla} I \frac{\partial \mathbf{W}}{\partial \mathbf{p}} \right]^{\mathrm{T}} \left[ \boldsymbol{\nabla} I \frac{\partial \mathbf{W}}{\partial \mathbf{p}} \right]. \tag{11}$$

For reasons that will become clear later we refer to $\sum_{\mathbf{x}} [\boldsymbol{\nabla} I \frac{\partial \mathbf{W}}{\partial \mathbf{p}}]^{\mathrm{T}} [T(\mathbf{x}) - I(\mathbf{W}(\mathbf{x};\mathbf{p}))]$ as the *steepest descent parameter updates*. Equation (10) then expresses the fact that the parameter updates

**The Lucas-Kanade Algorithm**

Iterate:

(1) Warp $I$ with $\mathbf{W}(\mathbf{x};\mathbf{p})$ to compute $I(\mathbf{W}(\mathbf{x};\mathbf{p}))$
(2) Compute the error image $T(\mathbf{x}) - I(\mathbf{W}(\mathbf{x};\mathbf{p}))$
(3) Warp the gradient $\nabla I$ with $\mathbf{W}(\mathbf{x};\mathbf{p})$
(4) Evaluate the Jacobian $\frac{\partial \mathbf{W}}{\partial \mathbf{p}}$ at $(\mathbf{x};\mathbf{p})$
(5) Compute the steepest descent images $\nabla I \frac{\partial \mathbf{W}}{\partial \mathbf{p}}$
(6) Compute the Hessian matrix using Equation (11)
(7) Compute $\sum_{\mathbf{x}} [\nabla I \frac{\partial \mathbf{W}}{\partial \mathbf{p}}]^{\mathrm{T}} [T(\mathbf{x}) - I(\mathbf{W}(\mathbf{x};\mathbf{p}))]$
(8) Compute $\Delta\mathbf{p}$ using Equation (10)
(9) Update the parameters $\mathbf{p} \leftarrow \mathbf{p} + \Delta\mathbf{p}$

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

Figure 1: The Lucas-Kanade algorithm [11] consists of iteratively applying Equations (10) & (5) 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 $\epsilon$. Because the gradient $\nabla I$ 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.

$\Delta\mathbf{p}$ are the steepest descent parameter updates multiplied by the inverse of the Hessian matrix. The Lucas-Kanade algorithm [11] then consists of iteratively applying Equations (10) and (5). See Figures 1 and 2 for a summary. Because the gradient $\nabla I$ 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 in general depend on $\mathbf{p}$. For some simple warps such as the translations in Equation (1) and the affine warps in Equation (2) the Jacobian can sometimes be constant. See for example Equation (8). In general, however, all 9 steps of the algorithm must be repeated in every iteration because the estimates of the parameters $\mathbf{p}$ vary from iteration to iteration.

## 2.3 Requirements on the Set of Warps

The only requirement on the warps $\mathbf{W}(\mathbf{x};\mathbf{p})$ is that they are differentiable with respect to the warp parameters $\mathbf{p}$. This condition is required to compute the Jacobian $\frac{\partial \mathbf{W}}{\partial \mathbf{p}}$. Normally the warps are also (piecewise) differentiable with respect to $\mathbf{x}$, but even this condition is not strictly required.

## 2.4 Computational Cost of the Lucas-Kanade Algorithm

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

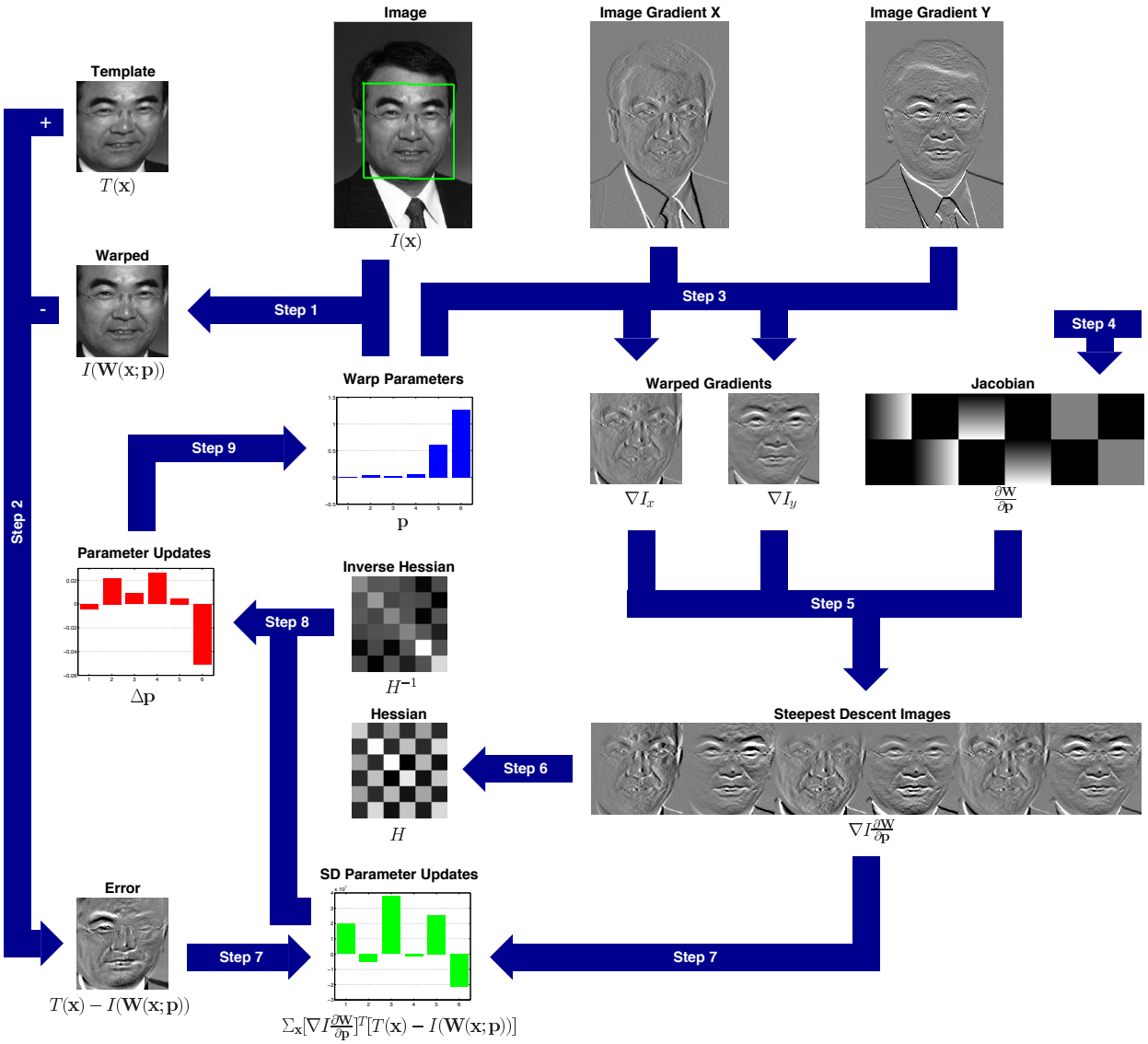



Figure 2: A schematic overview of the Lucas-Kanade algorithm [11]. The image $I$ is warped with the current estimate of the warp in Step 1 and the result subtracted from the template in Step 2 to yield the error image. The gradient of $I$ is warped in Step 3, the Jacobian is computed in Step 4, and the two combined in Step 5 to give the steepest descent images. In Step 6 the Hessian is computed from the steepest descent images. In Step 7 the steepest descent parameter updates are computed by dot producting the error image with the steepest descent images. In Step 8 the Hessian is inverted and multiplied by the steepest descent parameter updates to get the final parameter updates $\Delta \mathbf{p}$ which are then added to the parameters $\mathbf{p}$ in Step 9.

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 $\mathrm{O}(n^2\,N)$.

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

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

# 3 The Quantity Approximated and the Warp Update Rule

In each iteration the Lucas-Kanade algorithm approximately minimizes:

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

with respect to $\Delta\mathbf{p}$ and then updates the estimates of the parameters in Step 9 as:

$$\mathbf{p} \;\leftarrow\; \mathbf{p} + \Delta\mathbf{p}. \tag{13}$$

Perhaps somewhat surprisingly iterating these two steps is not the only way to minimize the expression in Equation (3). In this section we outline 3 alternative approaches that are all provably equivalent to the Lucas-Kanade algorithm. We then show empirically that they are equivalent.

## 3.1 Compositional Image Alignment

The first alternative to the Lucas-Kanade algorithm is the *compositional* algorithm.

### 3.1.1 Goal of the Compositional Algorithm

The compositional algorithm, used most notably by [14], approximately minimizes:

$$\sum_{\mathbf{x}} [\, I(\mathbf{W}(\mathbf{W}(\mathbf{x};\Delta\mathbf{p});\mathbf{p})) - T(\mathbf{x})\,]^2 \tag{14}$$

with respect to $\Delta\mathbf{p}$ in each iteration and then updates the estimate of the warp as:

$$\mathbf{W}(\mathbf{x};\mathbf{p}) \;\leftarrow\; \mathbf{W}(\mathbf{x};\mathbf{p}) \circ \mathbf{W}(\mathbf{x};\Delta\mathbf{p}), \tag{15}$$

i.e. the compositional approach iteratively solves for an incremental warp $\mathbf{W}(\mathbf{x};\Delta\mathbf{p})$ rather than an additive update to the parameters $\Delta\mathbf{p}$. In this context, we refer to the Lucas-Kanade algorithm

in Equations (12) and (13) as the *additive* approach to contrast it with the compositional approach in Equations (14) and (15). The compositional and additive approaches are proved to be equivalent to first order in $\Delta\mathbf{p}$ in Section 3.1.5. The expression:

$$\mathbf{W}(\mathbf{x};\mathbf{p}) \circ \mathbf{W}(\mathbf{x};\Delta\mathbf{p}) \equiv \mathbf{W}(\mathbf{W}(\mathbf{x};\Delta\mathbf{p});\mathbf{p}) \tag{16}$$

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

$$\begin{pmatrix} (1+p_1)\cdot((1+\Delta p_1)\cdot x + \Delta p_3 \cdot y + \Delta p_5) + p_3 \cdot (\Delta p_2 \cdot x + (1+\Delta p_4)\cdot y + \Delta p_6) + p_5 \\ p_2 \cdot ((1+\Delta p_1)\cdot x + \Delta p_3 \cdot y + \Delta p_5) + (1+p_4)\cdot (\Delta p_2 \cdot x + (1+\Delta p_4)\cdot y + \Delta p_6) + p_6 \end{pmatrix}, \tag{17}$$

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

$$\begin{pmatrix} p_1 + \Delta p_1 + p_1 \cdot \Delta p_1 + p_3 \cdot \Delta p_2 \\ p_2 + \Delta p_2 + p_2 \cdot \Delta p_1 + p_4 \cdot \Delta p_2 \\ p_3 + \Delta p_3 + p_1 \cdot \Delta p_3 + p_3 \cdot \Delta p_4 \\ p_4 + \Delta p_4 + p_2 \cdot \Delta p_3 + p_4 \cdot \Delta p_4 \\ p_5 + \Delta p_5 + p_1 \cdot \Delta p_5 + p_3 \cdot \Delta p_6 \\ p_6 + \Delta p_6 + p_2 \cdot \Delta p_5 + p_4 \cdot \Delta p_6 \end{pmatrix}, \tag{18}$$

a simple bilinear combination of the parameters of $\mathbf{W}(\mathbf{x};\mathbf{p})$ and $\mathbf{W}(\mathbf{x};\Delta\mathbf{p})$.

### 3.1.2 Derivation of the Compositional Algorithm

To derive the equivalent of the Lucas-Kanade algorithm in the compositional approach, we apply the first order Taylor expansion to Equation (14) to obtain:

$$\sum_{\mathbf{x}} \left[ I(\mathbf{W}(\mathbf{W}(\mathbf{x};\mathbf{0});\mathbf{p})) + \boldsymbol{\nabla} I(\mathbf{W}) \frac{\partial \mathbf{W}}{\partial \mathbf{p}} \Delta\mathbf{p} - T(\mathbf{x}) \right]^2. \tag{19}$$

In this expression $I(\mathbf{W})(\mathbf{x})$ denotes the warped image $I(\mathbf{W}(\mathbf{x};\mathbf{p}))$. It is possible to further expand:

$$\boldsymbol{\nabla} I(\mathbf{W}) \equiv \frac{\partial I(\mathbf{W})}{\partial \mathbf{x}} = \frac{\partial I}{\partial \mathbf{x}} \frac{\partial \mathbf{W}}{\partial \mathbf{x}} = \boldsymbol{\nabla} I \frac{\partial \mathbf{W}}{\partial \mathbf{x}} \tag{20}$$

but this turns out to be not worth doing because the warped image $I(\mathbf{W})$ is computed in Step 1 of the Lucas-Kanade algorithm anyway (see Figure 1) so its gradient can easily be used instead.

In order to proceed we make one assumption. We assume that $\mathbf{W}(\mathbf{x};\mathbf{0})$ is the identity warp; i.e. $\mathbf{W}(\mathbf{x};\mathbf{0}) = \mathbf{x}$. If the identity warp is in the set of warps being considered, we can reparameterize the warps to ensure that $\mathbf{W}(\mathbf{x};\mathbf{0}) = \mathbf{x}$. We are therefore only really assuming that the identity warp is in the set being considered. Assuming $\mathbf{W}(\mathbf{x};\mathbf{0}) = \mathbf{x}$, Equation (19) simplifies to:

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

**The Compositional Algorithm**

Pre-compute:

(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 $T(\mathbf{x}) - I(\mathbf{W}(\mathbf{x}; \mathbf{p}))$
(3) Compute the gradient $\nabla I(\mathbf{W})$ of image $I(\mathbf{W}(\mathbf{x}; \mathbf{p}))$
(5) Compute the steepest descent images $\nabla I(\mathbf{W}) \frac{\partial \mathbf{W}}{\partial \mathbf{p}}$
(6) Compute the Hessian matrix using Equation (11)
(7) Compute $\sum_{\mathbf{x}} [\nabla I(\mathbf{W}) \frac{\partial \mathbf{W}}{\partial \mathbf{p}}]^{\mathrm{T}} [T(\mathbf{x}) - I(\mathbf{W}(\mathbf{x}; \mathbf{p}))]$
(8) Compute $\Delta \mathbf{p}$ using Equation (10)
(9) Update the warp $\mathbf{W}(\mathbf{x}; \mathbf{p}) \leftarrow \mathbf{W}(\mathbf{x}; \mathbf{p}) \circ \mathbf{W}(\mathbf{x}; \Delta \mathbf{p})$

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

Figure 3: The compositional algorithm used in [14] is similar to the Lucas-Kanade algorithm. The only differences are: (1) the gradient of $I(\mathbf{W}(\mathbf{x}; \mathbf{p}))$ is used in Step 3 rather than the gradient of $I$ evaluated at $\mathbf{W}(\mathbf{x}; \mathbf{p})$, (2) the Jacobian can be pre-computed because it is evaluated at $(\mathbf{x}; \mathbf{0})$ and so is constant, and (3) the warp is updated by composing the incremental warp $\mathbf{W}(\mathbf{x}; \Delta \mathbf{p})$ with the current estimate in Step 9.

There are then 2 differences between Equation (21) and Equation (6), the equivalent equation in the Lucas-Kanade derivation. The first difference is that the gradient of $I(\mathbf{x})$ (evaluated at $\mathbf{W}(\mathbf{x}; \mathbf{p})$) is replaced with the gradient of $I(\mathbf{W})$. These gradients are different. The second difference is hidden by the concise notation. The Jacobian $\frac{\partial \mathbf{W}}{\partial \mathbf{p}}$ is evaluated at $(\mathbf{x}; \mathbf{p})$ in Equation (6), but it is evaluated at $(\mathbf{x}; \mathbf{0})$ in Equation (21) because that is where the Taylor expansion was performed. The Jacobian is therefore a constant and can be pre-computed. It is also generally simpler analytically [14].

The compositional algorithm is therefore almost exactly the same as the additive algorithm (the Lucas-Kanade algorithm). The only 3 differences are: (1) the gradient of $I(\mathbf{W}(\mathbf{x}; \mathbf{p}))$ should be used in Step 3, (2) the Jacobian can be pre-computed because it is evaluated at $(\mathbf{x}; \mathbf{0})$ rather than being re-computed in each iteration in Step 4, and (3) the warp is updated $\mathbf{W}(\mathbf{x}; \mathbf{p}) \leftarrow \mathbf{W}(\mathbf{x}; \mathbf{p}) \circ \mathbf{W}(\mathbf{x}; \Delta \mathbf{p})$ in Step 9. The compositional algorithm is summarized in Figure 3.

### 3.1.3 Requirements on the Set of Warps

Instead of simply adding the additive updates $\Delta \mathbf{p}$ to the current estimate of the parameters $\mathbf{p}$ as in the Lucas-Kanade algorithm, the incremental update to the warp $\mathbf{W}(\mathbf{x}; \Delta \mathbf{p})$ must be *composed* with the current estimate of the warp $\mathbf{W}(\mathbf{x}; \mathbf{p})$ in the compositional algorithm. This operation typically involves multiplying two matrices to compute the parameters of the composed warp, as in Equation (18) for affine warps. For more complex warps the composition of the two warps can be more involved [2]. We therefore have two requirements on the set of warps: (1) the set of warps must contain the identity warp and (2) the set of warps must be closed under composition. The set

Table 2: The computation cost of the compositional algorithm. The one time pre-computation cost of evaluating the Jacobian in Step 4 is $\mathrm{O}(n\,N)$. After that, the cost of each iteration is $\mathrm{O}(n^2\,N + n^3)$.

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

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

of warps must therefore form a *semi-group*. This requirement is not very strong. Most warps used in computer vision, including homographies and 3D rotations [14], naturally form semi-groups.

### 3.1.4 Computational Cost of the Compositional Algorithm

The computational cost of the compositional algorithm is almost exactly the same as that of the Lucas-Kanade algorithm. See Table 2 for a summary. The only steps that change are Steps 3, 4, and 9. The cost of Step 3 is slightly less at $\mathrm{O}(N)$. Step 4 can be performed as a pre-computation, but the cost is only $\mathrm{O}(n\,N)$ anyway. The cost of composing the two warps in Step 9 depends on $\mathbf{W}$ but for most warps the cost is $\mathrm{O}(n^2)$ or less; the cost for affine warps in Equation (18) is $\mathrm{O}(n^2)$. In total the cost is $\mathrm{O}(n^2\,N + n^3)$ per iteration, just like for the Lucas-Kanade algorithm, the most expensive step still being Step 6. The pre-computation cost is $\mathrm{O}(n\,N)$.

### 3.1.5 Equivalence of the Additive and Compositional Algorithms

We now show that the additive and compositional approaches are equivalent in the sense that, to a first order approximation in $\Delta\mathbf{p}$, they take the same steps in each iteration; i.e. the updates to the warps are approximately the same. In the additive formulation in Equation (6) we minimize:

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

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

$$\mathbf{W}(\mathbf{x};\mathbf{p}) \;\leftarrow\; \mathbf{W}(\mathbf{x};\mathbf{p} + \Delta\mathbf{p}) \;\approx\; \mathbf{W}(\mathbf{x};\mathbf{p}) + \frac{\partial \mathbf{W}}{\partial \mathbf{p}} \Delta\mathbf{p} \tag{23}$$

after a Taylor expansion is made. In the compositional formulation in Equation (21) we minimize:

$$\sum_{\mathbf{x}} \left[ I(\mathbf{W}(\mathbf{x};\mathbf{p}) + \boldsymbol{\nabla} I(\mathbf{W}) \frac{\partial \mathbf{W}}{\partial \mathbf{p}} \Delta\mathbf{p} - T(\mathbf{x}) \right]^2 . \tag{24}$$

which, using Equation (20), simplifies to:

$$\sum_{\mathbf{x}} \left[ I(\mathbf{W}(\mathbf{x};\mathbf{p})) + \boldsymbol{\nabla} I \frac{\partial \mathbf{W}}{\partial \mathbf{x}} \frac{\partial \mathbf{W}}{\partial \mathbf{p}} \Delta\mathbf{p} - T(\mathbf{x}) \right]^2 . \tag{25}$$

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

$$\mathbf{W}(\mathbf{x};\Delta\mathbf{p}) \;\approx\; \mathbf{W}(\mathbf{x};\mathbf{0}) + \frac{\partial \mathbf{W}}{\partial \mathbf{p}}\Delta\mathbf{p} \;=\; \mathbf{x} + \frac{\partial \mathbf{W}}{\partial \mathbf{p}}\Delta\mathbf{p} \tag{26}$$

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

$$\mathbf{W}(\mathbf{x};\mathbf{p}) \circ \mathbf{W}(\mathbf{x};\Delta\mathbf{p}) \;=\; \mathbf{W}(\mathbf{W}(\mathbf{x};\Delta\mathbf{p});\mathbf{p}). \tag{27}$$

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

$$\mathbf{W}(\mathbf{x};\mathbf{p}) \;\leftarrow\; \mathbf{W}(\mathbf{x};\mathbf{p}) + \frac{\partial \mathbf{W}}{\partial \mathbf{x}}\frac{\partial \mathbf{W}}{\partial \mathbf{p}}\Delta\mathbf{p} \tag{28}$$

to first order in $\Delta\mathbf{p}$. The difference between the additive formulation in Equations (22) and (23), and the compositional formulation in Equations (25) and (28) is that $\frac{\partial \mathbf{W}}{\partial \mathbf{p}}$ is replaced by $\frac{\partial \mathbf{W}}{\partial \mathbf{x}}\frac{\partial \mathbf{W}}{\partial \mathbf{p}}$. Equations (22) and (24) therefore generally result in different estimates for $\Delta\mathbf{p}$. (Note that in the second of these expressions $\frac{\partial \mathbf{W}}{\partial \mathbf{p}}$ is evaluated at $(\mathbf{x};\mathbf{0})$, rather than at $(\mathbf{x};\mathbf{p})$ in the first expression.)

If the vectors $\frac{\partial \mathbf{W}}{\partial \mathbf{p}}$ in the additive formulation and $\frac{\partial \mathbf{W}}{\partial \mathbf{x}}\frac{\partial \mathbf{W}}{\partial \mathbf{p}}$ in the compositional formulation both span the same linear space, however, the final updates to the warp in Equations (23) and (28) will be the same to first order in $\Delta\mathbf{p}$ and the two formulations are equivalent to first order in $\Delta\mathbf{p}$; i.e. the optimal value of $\frac{\partial \mathbf{W}}{\partial \mathbf{p}}\Delta\mathbf{p}$ in Equation (22) will approximately equal the optimal value of $\frac{\partial \mathbf{W}}{\partial \mathbf{x}}\frac{\partial \mathbf{W}}{\partial \mathbf{p}}\Delta\mathbf{p}$ in Equation (24). From Equation (23) we see that the first of these expressions:

$$\frac{\partial \mathbf{W}}{\partial \mathbf{p}} \;=\; \frac{\partial \mathbf{W}(\mathbf{x};\mathbf{p}+\Delta\mathbf{p})}{\partial \Delta\mathbf{p}} \tag{29}$$

and from Equation (28) we see that the second of these expressions:

$$\frac{\partial \mathbf{W}}{\partial \mathbf{x}}\frac{\partial \mathbf{W}}{\partial \mathbf{p}} \;=\; \frac{\partial \mathbf{W}(\mathbf{x};\mathbf{p}) \circ \mathbf{W}(\mathbf{x};\mathbf{p}+\Delta\mathbf{p})}{\partial \Delta\mathbf{p}}. \tag{30}$$

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

$$\mathbf{W}(\mathbf{x};\mathbf{p}+\Delta\mathbf{p}) \;=\; \mathbf{W}(\mathbf{x};\mathbf{p}) \circ \mathbf{W}(\mathbf{x};\mathbf{p}+\Delta\mathbf{p}'). \tag{31}$$

This condition means that the function between $\Delta\mathbf{p}$ and $\Delta\mathbf{p}'$ is defined in both directions. The expressions in Equations (29) and (30) therefore span the same linear space. If the warp is invertible Equation (32) always holds since $\Delta\mathbf{p}'$ can be chosen such that:

$$\mathbf{W}(\mathbf{x};\mathbf{p}+\Delta\mathbf{p}') \;=\; \mathbf{W}(\mathbf{x};\mathbf{p})^{-1} \circ \mathbf{W}(\mathbf{x};\mathbf{p}+\Delta\mathbf{p}). \tag{32}$$

In summary, if the warps are invertible then the two formulations are equivalent. In Section 3.1.3, above, we stated that the set of warps must form a semi-group for the compositional algorithm to be applied. While this is true, for the compositional algorithm also to be provably equivalent to the Lucas-Kanade algorithm, the set of warps must form a group; i.e. every warp must be invertible.

## 3.2 Inverse Compositional Image Alignment

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 [10, 7, 14]. If the Hessian were constant it could be precomputed and then re-used. Each iteration of the algorithm (see Figure 1) would then just consist of an image warp (Step 1), an image difference (Step 2), a collection of image "dot-products" (Step 7), multiplication of the result by the Hessian (Step 8), and the update to the parameters (Step 9). All of these operations can be performed at (close to) frame-rate [7].

Unfortunately the Hessian is a function of $\mathbf{p}$ in both formulations. Although various approximate solutions can be used (such as only updating the Hessian every few iterations and approximating the Hessian by assuming it is approximately constant across the image [14]) these approximations are inelegant and it is often hard to say how good approximations they are. It would be far better if the problem could be reformulated in an equivalent way, but with a constant Hessian.

### 3.2.1 Goal of the Inverse Compositional Algorithm

The key to efficiency is switching the role of the image and the template, as in [10], where a change of variables is made to switch or *invert* the roles of the template and the image. Such a change of variables can be performed in either the additive [10] or the compositional approach [2]. (Note that a restricted version of the inverse compositional algorithm was proposed for homographies in [7].) We first describe the inverse compositional approach because it is simpler. To distinguish the previous algorithms from the new ones, we will refer to the original algorithms as the *forwards additive* (i.e. Lucas-Kanade) and the *forwards compositional* algorithm. The corresponding algorithms after the inversion will be called the *inverse additive* and *inverse compositional* algorithms.

The proof of equivalence between the forwards compositional and inverse compositional algorithms is in Section 3.2.5. The result is that the inverse compositional algorithm minimizes:

$$\sum_{\mathbf{x}} [\, T(\mathbf{W}(\mathbf{x};\Delta\mathbf{p})) - I(\mathbf{W}(\mathbf{x};\mathbf{p}))\,]^2 \tag{33}$$

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

$$\mathbf{W}(\mathbf{x};\mathbf{p}) \leftarrow \mathbf{W}(\mathbf{x};\mathbf{p}) \circ \mathbf{W}(\mathbf{x};\Delta\mathbf{p})^{-1}. \tag{34}$$

The only difference from the update in the forwards compositional algorithm in Equation (13) is that the incremental warp $\mathbf{W}(\mathbf{x};\Delta\mathbf{p})$ is *inverted* before it is composed with the current estimate. For example, the parameters of the *inverse* of the affine warp in Equation (2) are:

$$\frac{1}{(1+p_1)\cdot(1+p_4)-p_2\cdot p_3}\begin{pmatrix} -p_1-p_1\cdot p_4+p_2\cdot p_3 \\ -p_2 \\ -p_3 \\ -p_4-p_1\cdot p_4+p_2\cdot p_3 \\ -p_5-p_4\cdot p_5+p_3\cdot p_6 \\ -p_6-p_1\cdot p_6+p_2\cdot p_5 \end{pmatrix}. \tag{35}$$

If $(1+p_1)\cdot(1+p_4)-p_2\cdot p_3=0$, the affine warp is degenerate and not invertible. All pixels are mapped onto a straight line in $I$. We exclude all such affine warps from consideration. The set of all such affine warps is then still closed under composition, as can be seen by computing $(1+p_1)\cdot(1+p_4)-p_2\cdot p_3$ for the parameters in Equation (18). After considerable simplification, this value becomes $[(1+p_1)\cdot(1+p_4)-p_2\cdot p_3]\cdot[(1+\Delta p_1)\cdot(1+\Delta p_4)-\Delta p_2\cdot\Delta p_3]$ which can only equal zero if one of the two warps being composed is degenerate.

### 3.2.2 Derivation of the Inverse Compositional Algorithm

Performing a first order Taylor expansion on Equation (33) gives:

$$\sum_{\mathbf{x}}\left[T(\mathbf{W}(\mathbf{x};\mathbf{0}))+\nabla T\frac{\partial\mathbf{W}}{\partial\mathbf{p}}\Delta\mathbf{p}-I(\mathbf{W}(\mathbf{x};\mathbf{p}))\right]^2. \tag{36}$$

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

$$\Delta\mathbf{p} \;=\; H^{-1}\sum_{\mathbf{x}}\left[\nabla T\frac{\partial\mathbf{W}}{\partial\mathbf{p}}\right]^{\mathrm{T}}\left[\,I(\mathbf{W}(\mathbf{x};\mathbf{p}))-T(\mathbf{x})\,\right] \tag{37}$$

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

$$H \;=\; \sum_{\mathbf{x}}\left[\nabla T\frac{\partial\mathbf{W}}{\partial\mathbf{p}}\right]^{\mathrm{T}}\left[\nabla T\frac{\partial\mathbf{W}}{\partial\mathbf{p}}\right] \tag{38}$$

and the Jacobian $\frac{\partial\mathbf{W}}{\partial\mathbf{p}}$ is evaluated at $(\mathbf{x};\mathbf{0})$. Since there is nothing in the Hessian that depends on $\mathbf{p}$, it is constant across iterations and can be pre-computed. Steps 3–6 of the forwards compositional algorithm in Figure 3 therefore need only be performed once as a pre-computation, rather than once per iteration. The only differences between the forwards and inverse compositional algorithms (see Figures 3 and 4) are: (1) the error image is computed after switching the roles of $I$ and $T$, (2) Steps 3, 5, and 6 use the gradient of $T$ rather than the gradient of $I$ and can be pre-computed, (3) Equation (37) is used to compute $\Delta\mathbf{p}$ rather than Equation (10), and finally (4) the incremental warp is inverted before it is composed with the current estimate in Step 9.

### 3.2.3 Requirements on the Set of Warps

Besides the semi-group requirement of the forwards compositional algorithm the inverse compositional algorithm also requires that the incremental warp $\mathbf{W}(\mathbf{x};\Delta\mathbf{p})$ be inverted before it is composed with the current estimate. The inverse compositional algorithm can therefore only be applied to sets of warps that form a *group*. Fortunately, most warps used in computer vision, including homographies and 3D rotations [14], do form groups. One notable exception are the piecewise affine warps used in Active Appearance Models (AAMs) [6], Active Blobs [13], and Flexible Appearance Models (FAMs) [2]. In [2] we showed how to extend the inverse compositional algorithm to piecewise affine warps. Similar extensions may be applicable to other non-group sets of warps.

**The Inverse Compositional Algorithm**

Pre-compute:

(3) Evaluate the gradient $\nabla T$ 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 $\nabla T \frac{\partial \mathbf{W}}{\partial \mathbf{p}}$
(6) Compute the Hessian matrix using Equation (38)

Iterate:

(1) Warp $I$ with $\mathbf{W}(\mathbf{x}; \mathbf{p})$ to compute $I(\mathbf{W}(\mathbf{x}; \mathbf{p}))$
(2) Compute the error image $I(\mathbf{W}(\mathbf{x}; \mathbf{p})) - T(\mathbf{x})$
(7) Compute $\sum_{\mathbf{x}} [\nabla T \frac{\partial \mathbf{W}}{\partial \mathbf{p}}]^{\mathrm{T}} [I(\mathbf{W}(\mathbf{x}; \mathbf{p})) - T(\mathbf{x})]$
(8) Compute $\Delta \mathbf{p}$ using Equation (37)
(9) Update the warp $\mathbf{W}(\mathbf{x}; \mathbf{p}) \leftarrow \mathbf{W}(\mathbf{x}; \mathbf{p}) \circ \mathbf{W}(\mathbf{x}; \Delta \mathbf{p})^{-1}$

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

Figure 4: The inverse compositional algorithm [2] is derived from the forwards compositional algorithm by inverting the roles of $I$ and $T$ similarly to the approach in [10]. 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 $\mathrm{O}(n\,N + n^3)$.

### 3.2.4 Computational Cost of the Inverse Compositional Algorithm

The inverse compositional algorithm is far more computationally efficient than either the Lucas-Kanade algorithm or the compositional algorithm. See Table 3 for a summary. The most time consuming step, the computation of the Hessian in Step 6 can be performed once as a pre-computation. The pre-computation takes time $\mathrm{O}(n^2\,N)$. Step 3 can also be performed as a pre-computation and can be performed slightly quicker than in the Lucas-Kanade algorithm. Step 3 takes time $\mathrm{O}(N)$ rather than time $\mathrm{O}(n\,N)$. The only additional cost is inverting $\mathbf{W}(\mathbf{x}; \Delta \mathbf{p})$ and composing it with $\mathbf{W}(\mathbf{x}; \mathbf{p})$. These two steps typically require $\mathrm{O}(n^2)$ operations, as for the affine warp in Equations (17) and (35). Potentially these 2 steps could be fairly involved, as in [2], but the computational overhead is almost always completely negligible. Overall the cost of the inverse compositional algorithm is $\mathrm{O}(n\,N + n^3)$ per iteration rather than $\mathrm{O}(n^2\,N)$, a substantial saving. The pre-computation cost is $\mathrm{O}(n^2\,N)$, however, but that only needs to be performed once.

### 3.2.5 Equivalence of the Forwards and Inverse Compositional Algorithms

We now show that the inverse compositional algorithm is equivalent to the forwards compositional algorithm introduced in Section 3.1. Since the forwards compositional algorithm was already shown to be equivalent to the Lucas-Kanade algorithm in Section 3.1.5, it follows that the inverse compositional algorithm is equivalent to it also. The proof of equivalence here takes a very different form to the proof in Section 3.1.5. The first step is to note that the summations in Equations (14)

Table 3: 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 $\mathrm{O}(n^2 N)$. After that, the cost of each iteration is $\mathrm{O}(n N + n^2)$ a substantial saving over the Lucas-Kanade and compositional algorithms.

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

| Per Iteration | Step 1 | Step 2 | Step 7 | Step 8 | Step 9 | Total |
|---|---|---|---|---|---|---|
| | $\mathrm{O}(n N)$ | $\mathrm{O}(N)$ | $\mathrm{O}(n N)$ | $\mathrm{O}(n^3)$ | $\mathrm{O}(n^2)$ | $\mathrm{O}(n N + n^3)$ |

and (33) are discrete approximations to integrals. Equation (14) is the discrete version of:

$$\int_T [\, I(\mathbf{W}(\mathbf{W}(\mathbf{x};\Delta\mathbf{p});\mathbf{p})) - T(\mathbf{x})\,]^2 \, \mathrm{d}\mathbf{x} \tag{39}$$

where the integration is performed over the template $T$. Setting $\mathbf{y} = \mathbf{W}(\mathbf{x};\Delta\mathbf{p})$, or equivalently $\mathbf{x} = \mathbf{W}(\mathbf{y};\Delta\mathbf{p})^{-1}$, and changing variables, Equation (39) becomes:

$$\int_{\mathbf{W}(T)} \Big[\, I(\mathbf{W}(\mathbf{y};\mathbf{p})) - T(\mathbf{W}(\mathbf{y};\Delta\mathbf{p})^{-1}) \,\Big]^2 \left| \frac{\partial \mathbf{W}^{-1}}{\partial \mathbf{y}} \right| \mathrm{d}\mathbf{y} \tag{40}$$

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

$$\left| \frac{\partial \mathbf{W}^{-1}}{\partial \mathbf{y}} \right| \;=\; 1 + O(\Delta\mathbf{p}). \tag{41}$$

The integration domain $\mathbf{W}(T)$ is equal to $T = \{\mathbf{W}(\mathbf{x};\mathbf{0}) \,|\, \mathbf{x} \in T\}$ to a zeroth order approximation also. Since we are ignoring higher order terms in $\Delta\mathbf{p}$, Equation (40) simplifies to:

$$\int_T \Big[\, T(\mathbf{W}(\mathbf{x};\Delta\mathbf{p})^{-1}) - I(\mathbf{W}(\mathbf{x};\mathbf{p})) \,\Big]^2 \, \mathrm{d}\mathbf{x}. \tag{42}$$

In making this simplification we have assumed that $T(\mathbf{W}(\mathbf{y};\Delta\mathbf{p})^{-1}) - I(\mathbf{W}(\mathbf{y};\mathbf{p}))$, or equivalently $T(\mathbf{y}) - I(\mathbf{W}(\mathbf{y};\mathbf{p}))$, is $O(\Delta\mathbf{p})$. (This assumption is equivalent to the assumption made in [10] that the current estimate of the parameters is approximately correct.) The first order terms in the Jacobian and the area of integration can therefore be ignored. Equation (42) is the continuous version of Equation (33) except that the term $\mathbf{W}(\mathbf{x};\Delta\mathbf{p})$ is inverted. The estimate of $\Delta\mathbf{p}$ that is computed by the inverse compositional algorithm using Equation (33) therefore gives an estimate of $\mathbf{W}(\mathbf{x};\Delta\mathbf{p})$ that is the inverse of the incremental warp computed by the compositional algorithm using Equation (14). Since the inverse compositional algorithm inverts $\mathbf{W}(\mathbf{x};\Delta\mathbf{p})$ before composing it with $\mathbf{W}(\mathbf{x};\mathbf{p})$ in Step 9, the two algorithms take the same steps to first order in $\Delta\mathbf{p}$.

## 3.3 Inverse Additive Image Alignment

A natural question which arises at this point is whether the same trick of changing variables to convert Equation (39) into Equation (40) can be applied in the additive formulation. The same step

can be applied, however the change of variables takes the form $\mathbf{y} = \mathbf{W}(\mathbf{x}; \mathbf{p} + \Delta\mathbf{p})$ rather than $\mathbf{y} = \mathbf{W}(\mathbf{x}; \Delta\mathbf{p})$. The simplification to the Jacobian in Equation (41) therefore cannot be made. The term $\frac{\partial \mathbf{W}^{-1}}{\partial \mathbf{y}}$ has to be included in an inverse additive algorithm in some form or other.

### 3.3.1 Goal of the Inverse Additive Algorithm

An image alignment algorithm that addresses this difficulty is the Hager-Belhumeur algorithm [10]. Although the derivation in [10] is slightly different from the derivation in Section 3.2.5, the Hager-Belhumeur algorithm does fit into our framework as an *inverse additive* algorithm. The initial goal of the Hager-Belhumeur algorithm is the same as the Lucas-Kanade algorithm; i.e. to minimize:

$$\sum_{\mathbf{x}} [\, I(\mathbf{W}(\mathbf{x}; \mathbf{p} + \Delta\mathbf{p})) - T(\mathbf{x}) \,]^2 \tag{43}$$

with respect to $\Delta\mathbf{p}$ and then update the parameters:

$$\mathbf{p} \leftarrow \mathbf{p} + \Delta\mathbf{p}. \tag{44}$$

Rather than changing variables like in Section 3.2.5, the roles of the template and the image are switched as follows. First the Taylor expansion is performed, just as in Section 2.1:

$$\sum_{\mathbf{x}} \left[ I(\mathbf{W}(\mathbf{x}; \mathbf{p})) + \boldsymbol{\nabla} I \frac{\partial \mathbf{W}}{\partial \mathbf{p}} \Delta\mathbf{p} - T(\mathbf{x}) \right]^2. \tag{45}$$

The template and the image are then switched by deriving the relationship between $\boldsymbol{\nabla} I$ and $\boldsymbol{\nabla} T$. In [10] it is assumed that the current estimates of the parameters are approximately correct: i.e.

$$I(\mathbf{W}(\mathbf{x}; \mathbf{p})) \approx T(\mathbf{x}) \tag{46}$$

This is equivalent to the assumption we made in Section 3.2.5 that $T(\mathbf{W}(\mathbf{y}; \Delta\mathbf{p})^{-1}) - I(\mathbf{W}(\mathbf{y}; \mathbf{p}))$ is $O(\Delta\mathbf{p})$. Taking partial derivatives with respect to $\mathbf{x}$ and using the chain rule gives:

$$\boldsymbol{\nabla} I \frac{\partial \mathbf{W}}{\partial \mathbf{x}} \approx \boldsymbol{\nabla} T. \tag{47}$$

Inverting $\frac{\partial \mathbf{W}}{\partial \mathbf{x}}$ and substituting Equation (47) into Equation (45) gives:

$$\sum_{\mathbf{x}} \left[ I(\mathbf{W}(\mathbf{x}; \mathbf{p})) + \boldsymbol{\nabla} T \left( \frac{\partial \mathbf{W}}{\partial \mathbf{x}} \right)^{-1} \frac{\partial \mathbf{W}}{\partial \mathbf{p}} \Delta\mathbf{p} - T(\mathbf{x}) \right]^2. \tag{48}$$

To completely change the role of the template and the image $I$, we replace $\Delta\mathbf{p}$ with $-\Delta\mathbf{p}$. The final goal of the Hager-Belhumeur algorithm is then to iteratively solve:

$$\sum_{\mathbf{x}} \left[ T(\mathbf{x}) + \boldsymbol{\nabla} T \left( \frac{\partial \mathbf{W}}{\partial \mathbf{x}} \right)^{-1} \frac{\partial \mathbf{W}}{\partial \mathbf{p}} \Delta\mathbf{p} - I(\mathbf{W}(\mathbf{x}; \mathbf{p})) \right]^2. \tag{49}$$

and update the parameters $\mathbf{p} \leftarrow \mathbf{p} - \Delta\mathbf{p}$.

### 3.3.2 Derivation of the Inverse Additive Algorithm

It is obviously possible to write down the solution to Equation (49) in terms of the Hessian, just like in Section 2.2. In general, however, the Hessian depends on $\mathbf{p}$ through $\left(\frac{\partial \mathbf{W}}{\partial \mathbf{x}}\right)^{-1}$ and $\frac{\partial \mathbf{W}}{\partial \mathbf{p}}$. So, in the naive approach, the Hessian will have to be re-computed in each iteration and the resulting algorithm will be just as inefficient as the original Lucas-Kanade algorithm.

To derive an efficient inverse additive algorithm, Hager and Belhumeur assumed that the warp $\mathbf{W}$ has a particular form. They assumed that the product of the two Jacobians can be written as:

$$\left(\frac{\partial \mathbf{W}}{\partial \mathbf{x}}\right)^{-1} \frac{\partial \mathbf{W}}{\partial \mathbf{p}} \;=\; \Gamma(\mathbf{x})\Sigma(\mathbf{p}) \tag{50}$$

where $\Gamma(\mathbf{x})$ is a $2 \times k$ matrix that is just a function of the template coordinates and $\Sigma(\mathbf{p})$ is a $k \times n$ matrix that is just a function of the warp parameters (and where $k$ is some positive integer.) Not all warps can be written in this form, but some can; e.g. if $\mathbf{W}$ is the affine warp of Equation (2):

$$\left(\frac{\partial \mathbf{W}}{\partial \mathbf{x}}\right)^{-1} = \begin{pmatrix} 1+p_1 & p_3 \\ p_2 & 1+p_4 \end{pmatrix}^{-1} = \frac{1}{(1+p_1)\cdot(1+p_4) - p_2 \cdot p_3} \begin{pmatrix} 1+p_4 & -p_3 \\ -p_2 & 1+p_1 \end{pmatrix} \tag{51}$$

and so:

$$\left(\frac{\partial \mathbf{W}}{\partial \mathbf{x}}\right)^{-1} \frac{\partial \mathbf{W}}{\partial \mathbf{p}} \;=\; \frac{1}{(1+p_1)\cdot(1+p_4) - p_2 \cdot p_3} \begin{pmatrix} 1+p_4 & -p_3 \\ -p_2 & 1+p_1 \end{pmatrix} \begin{pmatrix} x & 0 & y & 0 & 1 & 0 \\ 0 & x & 0 & y & 0 & 1 \end{pmatrix}. \tag{52}$$

Since diagonal matrices commute with any other matrix, and since the $2 \times 6$ matrix $\frac{\partial \mathbf{W}}{\partial \mathbf{p}}$ can be thought of 3 blocks of $2 \times 2$ matrices, the expression in Equation (52) can be re-written as:

$$\left(\frac{\partial \mathbf{W}}{\partial \mathbf{x}}\right)^{-1} \frac{\partial \mathbf{W}}{\partial \mathbf{p}} = \begin{pmatrix} x & 0 & y & 0 & 1 & 0 \\ 0 & x & 0 & y & 0 & 1 \end{pmatrix} \begin{pmatrix} \frac{1+p_4}{\det} & \frac{-p_3}{\det} & 0 & 0 & 0 & 0 \\ \frac{-p_2}{\det} & \frac{1+p_1}{\det} & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{1+p_4}{\det} & \frac{-p_3}{\det} & 0 & 0 \\ 0 & 0 & \frac{-p_2}{\det} & \frac{1+p_3}{\det} & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1+p_4}{\det} & \frac{-p_3}{\det} \\ 0 & 0 & 0 & 0 & \frac{-p_2}{\det} & \frac{1+p_3}{\det} \end{pmatrix} \tag{53}$$

where $\det = (1+p_1)\cdot(1+p_4) - p_2 \cdot p_3$. The product of the two Jacobians has therefore been written in the form of Equation (50). Equation(49) can then be re-written as:

$$\sum_{\mathbf{x}} \left[ T(\mathbf{x}) + \boldsymbol{\nabla} T \, \Gamma(\mathbf{x})\Sigma(\mathbf{p})\Delta\mathbf{p} - I(\mathbf{W}(\mathbf{x};\mathbf{p})) \right]^2. \tag{54}$$

Equation (54) has the closed form solution:

$$\Delta\mathbf{p} \;=\; H^{-1} \sum_{\mathbf{x}} \left[\boldsymbol{\nabla} T \, \Gamma(\mathbf{x})\Sigma(\mathbf{p})\right]^{\mathrm{T}} \left[I(\mathbf{W}(\mathbf{x};\mathbf{p})) - T(\mathbf{x})\right] \tag{55}$$

where $H$ is the $n \times n$ (first order approximation to the) *Hessian* matrix:

$$H \;=\; \sum_{\mathbf{x}} \left[\nabla T \, \Gamma(\mathbf{x}) \Sigma(\mathbf{p})\right]^{\mathrm{T}} \left[\nabla T \, \Gamma(\mathbf{x}) \Sigma(\mathbf{p})\right]. \tag{56}$$

Since $\Sigma(\mathbf{p})$ does not depend upon $\mathbf{x}$, the Hessian can be re-written as:

$$H \;=\; \Sigma(\mathbf{p})^{\mathrm{T}} \left( \sum_{\mathbf{x}} \left[\nabla T \, \Gamma(\mathbf{x})\right]^{\mathrm{T}} \left[\nabla T \, \Gamma(\mathbf{x})\right] \right) \Sigma(\mathbf{p}) \tag{57}$$

Denoting:

$$H_* \;=\; \sum_{\mathbf{x}} \left[\nabla T \, \Gamma(\mathbf{x})\right]^{\mathrm{T}} \left[\nabla T \, \Gamma(\mathbf{x})\right] \tag{58}$$

and assuming that $\Sigma(\mathbf{p})$ is invertible (Hager and Belhumeur actually consider a slightly more general case. The interested reader is referred to [10] for more details.), we have:

$$H^{-1} \;=\; \Sigma(\mathbf{p})^{-1} H_*^{-1} \Sigma(\mathbf{p})^{-\mathrm{T}}. \tag{59}$$

Inserting this expression into Equation (55) and simplifying yields:

$$\Delta \mathbf{p} \;=\; \Sigma(\mathbf{p})^{-1} H_*^{-1} \sum_{\mathbf{x}} \left[\nabla T \, \Gamma(\mathbf{x})\right]^{\mathrm{T}} \left[I(\mathbf{W}(\mathbf{x};\mathbf{p})) - T(\mathbf{x})\right]. \tag{60}$$

Equation (60) can be split into two steps:

$$\Delta \mathbf{p}_* \;=\; H_*^{-1} \sum_{\mathbf{x}} \left[\nabla T \, \Gamma(\mathbf{x})\right]^{\mathrm{T}} \left[I(\mathbf{W}(\mathbf{x};\mathbf{p})) - T(\mathbf{x})\right] \tag{61}$$

and:

$$\Delta \mathbf{p} \;=\; \Sigma(\mathbf{p})^{-1} \Delta \mathbf{p}_* \tag{62}$$

where nothing in the first step depends on the current estimate of the warp parameters $\mathbf{p}$. The Hager-Belhumeur algorithm consists of iterating applying Equation (61) and then updating the parameters $\mathbf{p} \leftarrow \mathbf{p} - \Sigma(\mathbf{p})^{-1} \Delta \mathbf{p}_*$. For the affine warp of Equation (2):

$$\Sigma(\mathbf{p})^{-1} \;=\; \begin{pmatrix} 1+p_1 & p_3 & 0 & 0 & 0 & 0 \\ p_2 & 1+p_4 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1+p_1 & p_3 & 0 & 0 \\ 0 & 0 & p_2 & 1+p_4 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1+p_1 & p_3 \\ 0 & 0 & 0 & 0 & p_2 & 1+p_4 \end{pmatrix}. \tag{63}$$

The Hager-Belhumeur inverse additive algorithm is summarized in Figure 5. The main differences from the inverse compositional algorithm in Figure 4 are: (1) instead of evaluating the Jacobian in Step 4 the term $\Gamma(\mathbf{x})$ is evaluated instead, (2) modified steepest descent images are computed in Step (5), (3) the modified Hessian $H_*$ is computed in Step 6, (3) Equation (61) is used to compute $\Delta \mathbf{p}_*$ in Steps 7 and 8, and finally (4) the warp is updated $\mathbf{p} \leftarrow \mathbf{p} - \Sigma(\mathbf{p})^{-1} \Delta \mathbf{p}_*$ in Step 9 rather than inverting the incremental warp and composing it with the current estimate of the warp.

**The Hager-Belhumeur Inverse Additive Algorithm**

Pre-compute:

(3) Evaluate the gradient $\nabla T$ of the template $T(\mathbf{x})$
(4) Evaluate $\Gamma(\mathbf{x})$
(5) Compute the modified steepest descent images $\nabla T\,\Gamma(\mathbf{x})$
(6) Compute the modified Hessian $H_*$ using Equation (58)

Iterate:

(1) Warp $I$ with $\mathbf{W}(\mathbf{x};\mathbf{p})$ to compute $I(\mathbf{W}(\mathbf{x};\mathbf{p}))$
(2) Compute the error image $I(\mathbf{W}(\mathbf{x};\mathbf{p})) - T(\mathbf{x})$
(7) Compute $\sum_{\mathbf{x}}[\nabla T\,\Gamma(\mathbf{x})]^{\mathrm{T}}[I(\mathbf{W}(\mathbf{x};\mathbf{p})) - T(\mathbf{x})]$
(8) Compute $\Delta\mathbf{p}_*$ using Equation (61)
(9) Compute $\Sigma(\mathbf{p})^{-1}$ and update $\mathbf{W}(\mathbf{x};\mathbf{p}) \leftarrow \mathbf{p} - \Sigma(\mathbf{p})^{-1}\Delta\mathbf{p}_*$

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

Figure 5: The Hager-Belhumeur inverse additive algorithm [10] is very similar to the inverse compositional algorithm in Figure 4. The two main differences are that $\Gamma$ is used instead of the Jacobian and the warp is updated using $\Sigma(\mathbf{p})$ rather than by inverting the incremental warp and composing it with the current estimate. Otherwise, the similarly between the two algorithms is apparent. The computational cost of the two algorithms is almost exactly identical. The problem with the Hager-Belhumeur algorithm, however, is that it can only be applied to (the very few) warps for which the assumption in Equation (50) holds.

### 3.3.3 Requirements on the Set of Warps

Besides being far more complex to derive, the other major disadvantage of the Hager-Belhumeur algorithm compared to the inverse compositional algorithm is that it can only be applied to a very small set of warps. The product of the two Jacobians has to be able to be written in the form of Equation (50) for the Hager-Belhumeur algorithm to be used. This requirement is very restrictive. It is also very hard to say whether the algorithm can be used with any particular set of warps. In [10] the authors do show that their algorithm can be used with 2D translations, 2D similarity transformations, 2D affine warps, and a small number of esoteric non-linear warps. The Hager-Belhumeur algorithm may be applicable to other sets of warps, but it is impossible to say whether it can be or not without performing analysis similar to that in Equations (51)–(53). In comparison the inverse compositional algorithm can be applied to any set of warps which form a group, a very weak requirement. Almost all warps used in computer vision form groups. Moreover, the inverse compositional algorithm can be extended to apply to many warps that don't form a group [2].

### 3.3.4 Computational Cost of the Inverse Additive Algorithm

The computational cost of the Hager-Belhumeur algorithm is similar to that of the inverse compositional algorithm. In most of the steps, the cost is a function of $k$ rather than $n$, but most of the time $k = n$ in the Hager-Belhumeur algorithm anyway. Steps 1, 2, and 3 are the same

Table 4: The computation cost of the Hager-Belhumeur algorithm. Both the pre-computation cost and the cost per iteration are almost exactly the same as the inverse compositional algorithm when $k = n$.

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

| Per Iteration | Step 1 | Step 2 | Step 7 | Step 8 | Step 9 | Total |
|---|---|---|---|---|---|---|
| | $\mathrm{O}(n\,N)$ | $\mathrm{O}(N)$ | $\mathrm{O}(k\,N)$ | $\mathrm{O}(k^3)$ | $\mathrm{O}(k^2)$ | $\mathrm{O}(n\,N + k\,N + k^3)$ |

in both algorithms. Step 4 generally takes time $\mathrm{O}(k\,N)$ and so is negligible. Step 5 takes time $\mathrm{O}(k\,N)$ rather than $\mathrm{O}(n\,N)$. The most time consuming step is Step 6 which takes time $\mathrm{O}(k^2\,N)$, but is a pre-computation steps and so only needs to be performed once. Step 7 takes time $\mathrm{O}(k\,N)$ compared to $\mathrm{O}(n\,N)$ in the inverse compositional algorithm. Similarly, Step 8 takes time $\mathrm{O}(k^3)$ compared to $\mathrm{O}(n^3)$. The update to the warp in Step 9 generally takes time $\mathrm{O}(k^2)$ compared to $\mathrm{O}(n^2)$. In total, the pre-computation cost of the Hager-Belhumeur algorithm is $\mathrm{O}(k^2\,N)$ compared to $\mathrm{O}(n^2\,N)$, and the computation per iteration is $\mathrm{O}(n\,N + k\,N + k^3)$ compared to $\mathrm{O}(n\,N + n^3)$. Overall, both algorithms take almost exactly the same time when $k = n$ (which is generally the case.) See Table 4 for a summary of the computational cost of the Hager-Belhumeur algorithm.

### 3.3.5 Equivalence of the Inverse Additive and Compositional Algorithms for Affine Warps

In Sections 3.1.5 and 3.2.5 we showed that the inverse compositional algorithm was equivalent to the Lucas-Kanade algorithm. The Hager-Belhumeur algorithm is also equivalent to the Lucas-Kanade algorithm in the same sense. See [10] for the details. The inverse compositional algorithm and the Hager-Belhumeur algorithm should therefore also take the same steps to first order in $\Delta\mathbf{p}$. The most interesting set of warps for which we can validate this equivalence is the set of 2D affine warps introduced in Equation (2). The Hager-Belhumeur algorithm cannot be applied to homographies and other more interesting sets of warps.

Comparing Equations (8) and (53) we see that for the affine warp $\Gamma(\mathbf{x}) = \frac{\partial \mathbf{W}}{\partial \mathbf{p}}$. The only difference therefore between the Hager-Belhumeur algorithm and the inverse compositional algorithm is in Step 9, the update to the parameters. The update for the Hager-Belhumeur algorithm is:

$$\mathbf{p} \leftarrow \mathbf{p} - \begin{pmatrix} 1+p_1 & p_3 & 0 & 0 & 0 & 0 \\ p_2 & 1+p_4 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1+p_1 & p_3 & 0 & 0 \\ 0 & 0 & p_2 & 1+p_4 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1+p_1 & p_3 \\ 0 & 0 & 0 & 0 & p_2 & 1+p_4 \end{pmatrix} \Delta\mathbf{p}_* \tag{64}$$

where $\Delta\mathbf{p}_*$ for the Hager-Belhumeur algorithm equals $\Delta\mathbf{p}$ for the inverse compositional algorithm since Steps 1–8 of the two algorithms are the same for affine warps. If $\mathbf{W}(\mathbf{x}; \Delta\mathbf{p})$ is the incremental

warp in the inverse compositional algorithm the parameters of $\mathbf{W}(\mathbf{x};\Delta\mathbf{p})^{-1}$ are:

$$\frac{1}{(1+\Delta p_1)\cdot(1+\Delta p_4)-\Delta p_2\cdot\Delta p_3}\begin{pmatrix} -\Delta p_1-\Delta p_1\cdot\Delta p_4+\Delta p_2\cdot\Delta p_3 \\ -\Delta p_2 \\ -\Delta p_3 \\ -\Delta p_4-\Delta p_1\cdot\Delta p_4+\Delta p_2\cdot\Delta p_3 \\ -\Delta p_5-\Delta p_1\cdot\Delta p_5+\Delta p_3\cdot\Delta p_6 \\ -\Delta p_6-\Delta p_1\cdot\Delta p_6+\Delta p_2\cdot\Delta p_5 \end{pmatrix} = \begin{pmatrix} -\Delta p_1 \\ -\Delta p_2 \\ -\Delta p_3 \\ -\Delta p_4 \\ -\Delta p_5 \\ -\Delta p_6 \end{pmatrix} \tag{65}$$

to first order in $\Delta\mathbf{p}$. Substituting this expression into Equation (18) shows that the parameters of $\mathbf{W}(\mathbf{x};\mathbf{p})\circ\mathbf{W}(\mathbf{x};\Delta\mathbf{p})^{-1}$ are the same as the parameters on the right hand side of Equation (64). The Hager-Belhumeur algorithm and the inverse compositional algorithm therefore take the same steps to first order in $\Delta\mathbf{p}$ for affine warps, where both algorithms can be applied. Of course there are many types of warp, such as homographies and 3D rotations [14], for which the Hager-Belhumeur algorithm cannot be applied, even though the inverse compositional algorithm can be [2].

## 3.4 Empirical Validation

We have proved mathematically that all four image alignment algorithms take the same steps to first order in $\Delta\mathbf{p}$, at least on sets of warps where they can all be used. The following experiments were performed to validate the equivalence of the four algorithms.

### 3.4.1 Example Convergence Rates

We experimented with the image $I(\mathbf{x})$ in Figure 2. We manually selected a $100\times100$ pixel template $T(\mathbf{x})$ in the center of the face. We then randomly generated affine warps $\mathbf{W}(\mathbf{x};\mathbf{p})$ in the following manner. We selected 3 canonical points in the template. We used the bottom left corner $(0,0)$, the bottom right corner $(99,0)$, and the center top pixel $(49,99)$ as the canonical points. We then randomly perturbed these points with additive white Gaussian noise of a certain variance and fit for the affine warp parameters $\mathbf{p}$ that these 3 perturbed points define. We then warped $I$ with this affine warp $\mathbf{W}(\mathbf{x};\mathbf{p})$ and run the four image alignment algorithms starting from the identity warp.

Since the 6 parameters in the affine warp have different units, we use the following error measure rather than the errors in the parameters. Given the current estimate of the warp, we compute the destination of the 3 canonical points and compare them with the correct locations. We compute the RMS error over the 3 points of the distance between their current and correct locations.

We used a similar experimental procedure for homographies. The only difference is that we used 4 canonical points at the corners of the template, bottom left $(0,0)$, bottom right $(99,0)$, top left $(0,99)$, and top right $(99,99)$ rather than the 3 canonical points used for affine warps.

In Figure 6 we include examples of the algorithms converging. The RMS error in the canonical point locations is plot against the iteration number. In Figures 6(a), (c), and (e) we plot results for affine warps. In Figures 6(b), (d), and (f) we plot results for homographies. As can be seen, the

algorithms converge at the same rate validating their equivalence. The inverse additive algorithm cannot be used for homographies and so only 3 curves are shown in Figures 6(b), (d), and (f).

### 3.4.2 Average Rates of Convergence

We also computed the average rate of convergence over a large number (5000) of randomly generated inputs. To avoid the results being biased by cases where one or more of the algorithms diverged, we checked that all 4 algorithms converged before including the sample in the average. We say that an algorithm has diverged in this experiment if the final RMS error in the canonical point location is larger than it was in the input. The average rates of convergence are shown in Figure 7 where we plot three curves for three different variances of the noise added to the canonical point locations. As can be seen, the 4 algorithms (3 for the homography) all converge at almost exactly the same rate, again validating the equivalence of the four algorithms. The algorithms all require between 5 and 15 iterations to converge depending on the magnitude of the initial error.

### 3.4.3 Average Frequency of Convergence

What about the case that the algorithms diverge? We ran a second similar experiment over 5000 randomly generated inputs. In this case, we counted the number of times that each algorithm converged. In this second experiment, we say that an algorithm converged if after 15 iterations the RMS error in the canonical point locations is less than 1.0 pixels. We computed the percentage of times that each algorithm converged for various different variances of the noise added to the canonical point locations. The results are shown in Figure 8. Again, the four algorithms all perform much the same, validating their equivalence. When the perturbation to the canonical point locations is less than about 4.0 pixels, all four algorithms converge almost always. Above that amount of perturbation the frequency of convergence rapidly decreases.

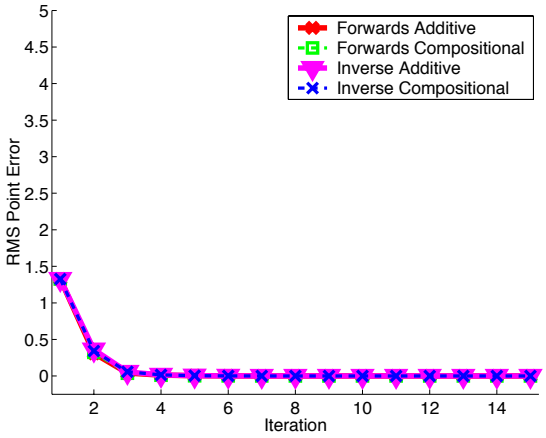
None of the algorithms used a multiscale pyramid to increase their robustness. This extension could be applied to all of the algorithms. Our goal was only to validate the equivalence of the base algorithms. The effect of using a multiscale pyramid will be studied in Part 2 of this 2 paper series.

### 3.4.4 The Effect of Additive Intensity Noise

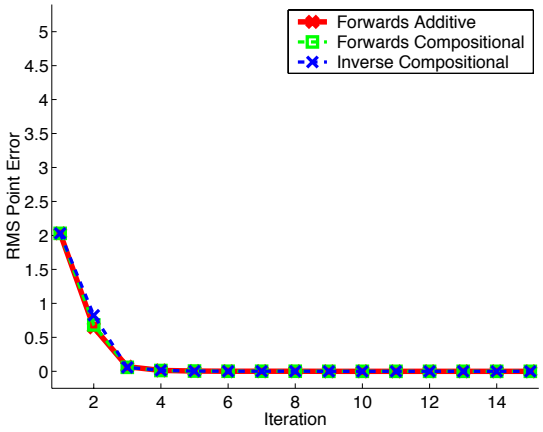
The results that we have presented so far have not included any noise on the images themselves (just on the canonical point locations which is not really noise but just a way of generating random warps.) The image $I(\mathbf{x})$ is warped with $\mathbf{W}(\mathbf{x};\mathbf{p})$ to generate the input image, a process which introduces a very small amount of resampling noise, but that resampling noise is negligible.

We repeated the above experiments, but with additive, white Gaussian noise added to the images. We considered 3 cases: (1) noise added to the image $I(\mathbf{x})$, (2) noise added to the template $T(\mathbf{x})$, and (3) noise added to both $I(\mathbf{x})$ and $T(\mathbf{x})$. The results for additive noise with standard deviation 8 grey levels is shown in Figure 9.

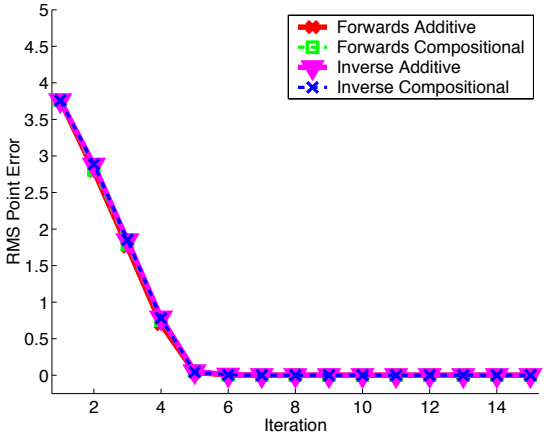
The first thing to note is that noise breaks the equivalence of the forwards and inverse algorithms. This is not too surprising. In the proof of equivalence it is assumed that $T(\mathbf{y})-I(\mathbf{W}(\mathbf{y};\mathbf{p}))$,



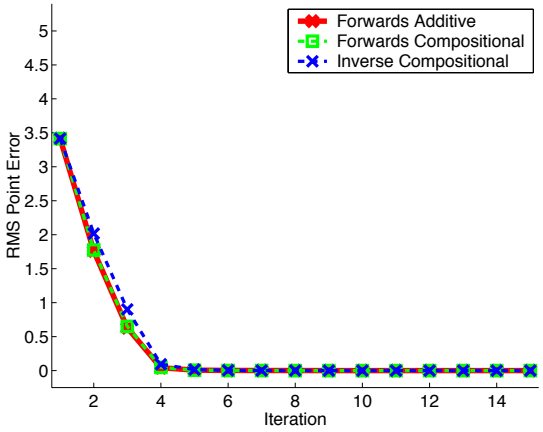

(a) Example Convergence for an Affine Warp



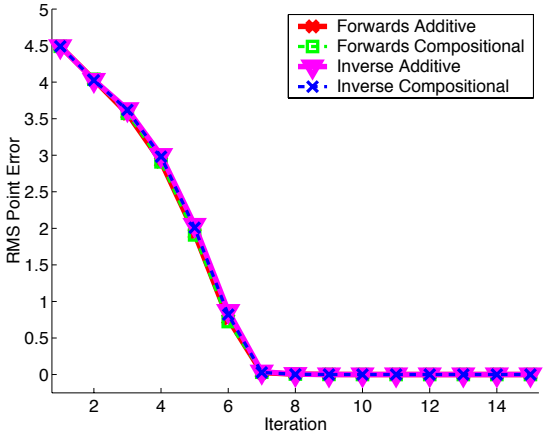

(b) Example Convergence for a Homography



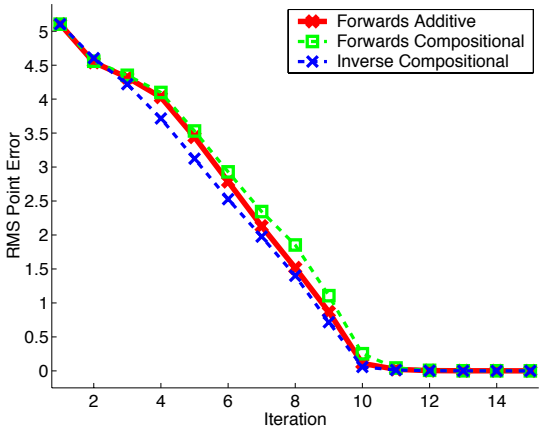

(c) Example Convergence for an Affine Warp




(d) Example Convergence for a Homography

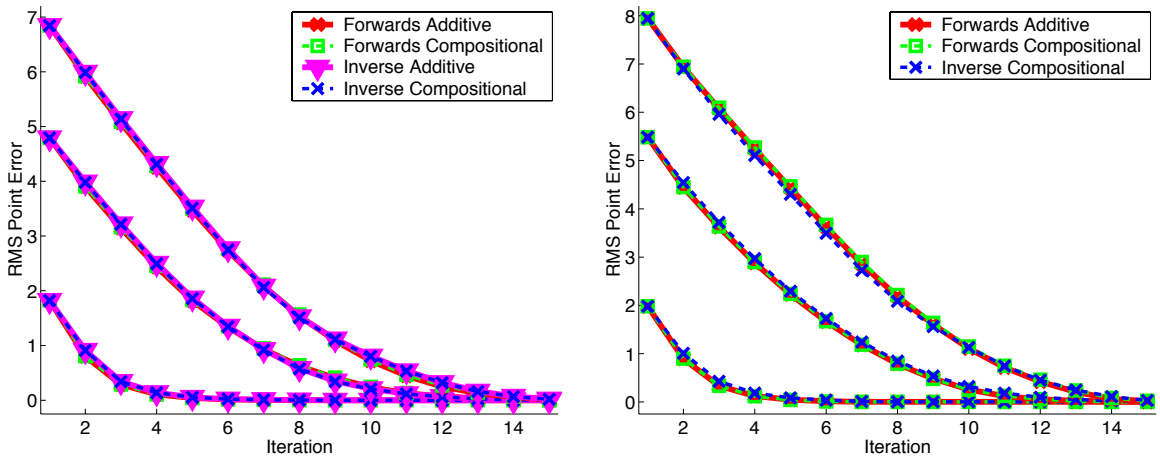



(e) Example Convergence for an Affine Warp



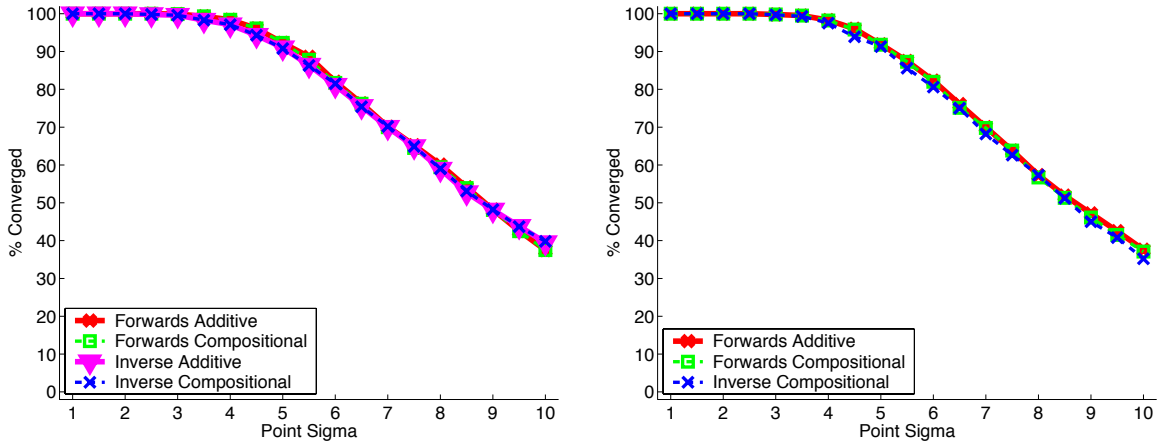

(f) Example Convergence for a Homography

Figure 6: Examples of the four algorithms converging. The RMS error in the location of 3 canonical points in the template (4 for the homographies) is plot against the algorithm iteration number. In all examples the algorithms converge at the same rate validating their equivalence. Only 3 plots are shown for the homography in (b), (d), and (f) because the inverse additive algorithm cannot be applied to homographies.




(a) Average Convergence Rates for Affine (b) Average Convergence Rates for Homography

Figure 7: The average rates of convergence computed over a large number of randomly generated warps. We plot three curves for three difference variances of the noise added to the canonical point locations to generate the input warps. The four algorithms all converge at the same rate again validating their equivalence.




(a) Frequency of Convergence for Affine Warps (b) Frequency of Convergence for Homographies

Figure 8: The average frequency of convergence computed over a large number of randomly generated warps. We compute the frequency of convergence for different variances of the noise added to the canonical point locations. The four algorithms all converge equally often validating their equivalence.

is $O(\Delta\mathbf{p})$. This is not true in the presence of noise. Since the forwards algorithms compute the gradient of $I$ and the inverse algorithms compute the gradient of $T$, it is not surprising that when noise is added to $I$ the inverse algorithms converge better (faster and more frequently), and conversely when noise is added to $T$ the forwards algorithms converge better. When equal noise is added to both images, it turns out that the forwards algorithms perform marginally better.

Overall we conclude that the forwards algorithms are ever so slightly more robust to additive noise. However, in many applications such as in face modeling [2], the template image $T(\mathbf{x})$ can be computed as an average of a large number of images and should be less noisy than the input

Table 5: Timing results for our Matlab implementation of the four algorithms in milliseconds. These results are for the 6-parameter affine warp using a $100 \times 100$ pixel template on a 933MHz Pentium-IV.

**Precomputation:**

| | Step 3 | Step 4 | Step 5 | Step 6 | Total |
|---|---|---|---|---|---|
| Forwards Additive (FA) | - | - | - | - | 0.0 |
| Forwards Compositional (FC) | - | 17.4 | - | - | 17.4 |
| Inverse Additive (IA) | 8.30 | 17.1 | 27.5 | 37.0 | 89.9 |
| Inverse Compositional (IC) | 8.31 | 17.1 | 27.5 | 37.0 | 90.0 |

**Per Iteration:**

| | Step 1 | Step 2 | Step 3 | Step 4 | Step 5 | Step 6 | Step 7 | Step 8 | Step 9 | Total |
|---|---|---|---|---|---|---|---|---|---|---|
| FA | 1.88 | 0.740 | 36.1 | 17.4 | 27.7 | 37.2 | 6.32 | 0.111 | 0.108 | 127 |
| FC | 1.88 | 0.736 | 8.17 | - | 27.6 | 37.0 | 6.03 | 0.106 | 0.253 | 81.7 |
| IA | 1.79 | 0.688 | - | - | - | - | 6.22 | 0.106 | 0.624 | 9.43 |
| IC | 1.79 | 0.687 | - | - | - | - | 6.22 | 0.106 | 0.409 | 9.21 |

image $I(\mathbf{x})$. In such cases, the inverse algorithms will be more robust to noise.

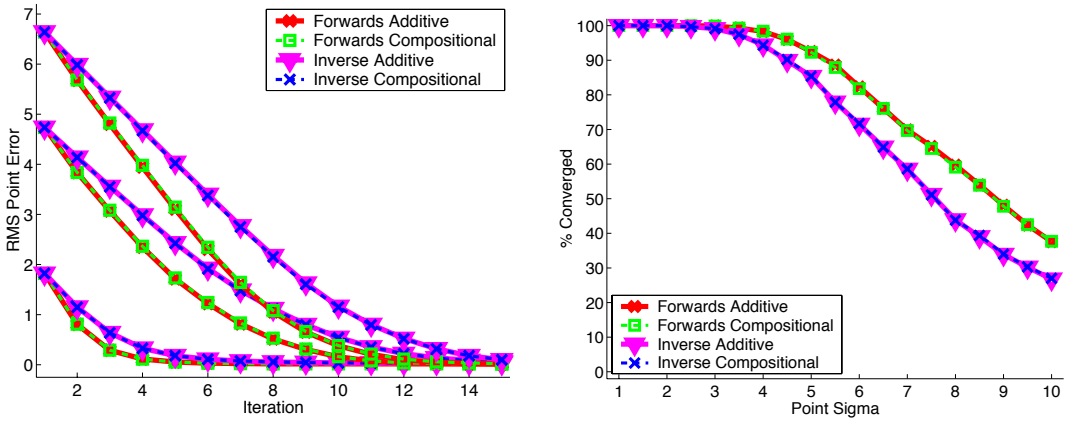
### 3.4.5 Timing Results

We implemented the four algorithms in Matlab. The Matlab implementation of image warping (Step 1, also used in Step 3 in the forwards additive algorithm) is very slow. Hence we re-implemented that step in "C." The timing results for the 6-parameter affine warp using a $100 \times 100$ pixel grey-scale template on a 933MHz Pentium-IV are included in Table 5. As can be seen, the two inverse algorithms shift much of the computation into pre-computation and so are far faster per iteration. The forwards compositional algorithm is also somewhat faster than the forwards additive algorithm since it does not need to warp the image gradients in Step 3 and the image gradients are computed on the template rather than the input image which is generally larger.

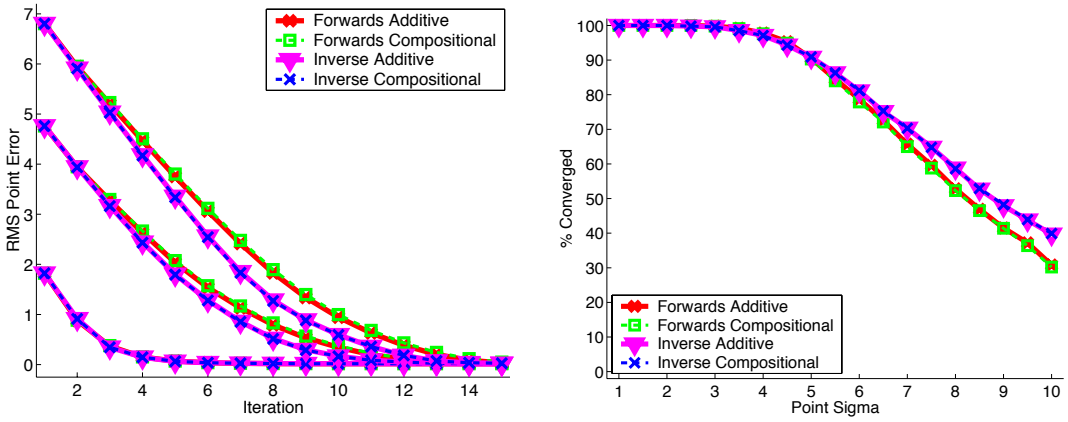
## 3.5 Summary

We have outlined three approaches to image alignment beyond the original *forwards additive* Lucas-Kanade algorithm. We refer to these approaches as the *forwards compositional*, the *inverse additive*, and the *inverse compositional* algorithms. In Sections 3.1.5, 3.2.5, and 3.3.5 we proved that all four algorithms are equivalent in the sense that they take the same steps in each iteration to first order in $\Delta\mathbf{p}$. In Section 3.4 we validated this equivalence empirically.

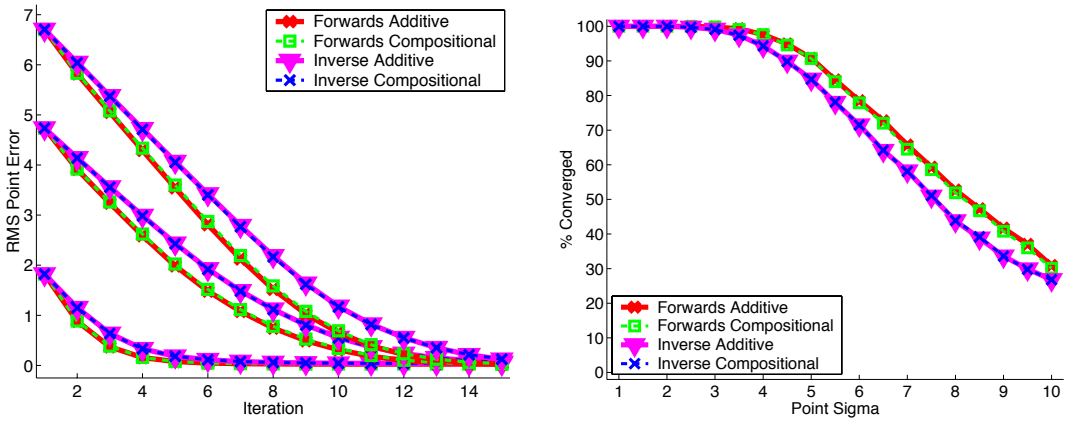
The four algorithms do differ, however, in two other respects. See Table 6 for a summary. Although the computational requirements of the two forwards algorithms are almost identical and




(a) Conv. Rate Adding Noise to $T(\mathbf{x})$ (b) Conv. Frequency Adding Noise to $T(\mathbf{x})$




(c) Conv. Rate Adding Noise to $I(\mathbf{x})$ (d) Conv. Frequency Adding Noise to $I(\mathbf{x})$




(e) Conv. Rate Adding Noise to $T$ and $I$ (f) Conv. Frequency Adding Noise to $T$ and $I$

Figure 9: The effect of intensity noise on the rate of convergence and the frequency of convergence for affine warps. The results for homographies are similar and omitted for lack of space. In all cases, additive, white Gaussian noise with standard deviation 8.0 grey levels is added to one or both of the template $T(\mathbf{x})$ and the warped input image $I(\mathbf{x})$. The results show that the forwards algorithms are more robust to noise on the template and the inverse algorithms are more robust to noise on the input image. Overall, the forwards algorithms are slightly more robust to noise added to both the template and the input image.

Table 6: A framework for gradient descent image alignment algorithms. Gradient descent image alignment algorithms can be either *additive* or *compositional*, and either *forwards* or *inverse*. The inverse algorithms are computationally efficient whereas the forwards algorithms are not. The various algorithms can be applied to different sets of warps. Most sets of warps in computer vision form groups and so the forwards additive, the forwards compositional, and the inverse compositional algorithms can be applied to most sets of warps. The inverse additive algorithm can only be applied to a very small class of warps, mostly linear 2D warps.

| Algorithm | For Example | Efficient? | Can be Applied To |
|---|---|---|---|
| Forwards Additive | Lucas-Kanade [11] | No | Any Warp |
| Forwards Compositional | Shum-Szeliski [14] | No | Any Semi-Group |
| Inverse Additive | Hager-Belhumeur [10] | Yes | Simple Linear 2D + |
| Inverse Compositional | Baker-Matthews [2] | Yes | Any Group |

the computational requirements of the two inverse algorithms are also almost identical, the two inverse algorithms are far more efficient than the two forwards algorithms. On the other hand, the forwards additive algorithm can be applied to any type of warp, the forwards compositional algorithm can only be applied to sets of warps that form *semi-groups*, and the inverse compositional algorithm can only be applied sets of warps that form *groups*. The inverse additive algorithm can be applied to very few warps, mostly simple 2D linear warps such as translations and affine warps.

A natural question which arises at this point is which of the four algorithms should be used. If efficiency is not a concern, either of the two forwards algorithms could be used, depending on which is more convenient. There is little difference between the two algorithms. The forwards compositional algorithm does has the slight advantage that the Jacobian is constant, and is in general simpler so is less likely to be computed erroneously [14]. The composition of the incremental warp with the current estimate is slightly more involved than simply adding the parameter updates, however. Overall, there is not much to distinguish the two forwards algorithms.

If obtaining high efficiency is important, the choice is between the inverse compositional algorithm and the inverse additive. The better choice here is clearly the inverse compositional algorithm. The derivation of the inverse compositional algorithm is far simpler, and it is far easier to determine if, and how, the inverse compositional algorithm can be applied to a new set of warps. Since on warps like affine warps the algorithms are almost exactly the same, there is no reason to use the inverse additive algorithm. The inverse compositional algorithm is equally efficient, conceptually more elegant, and more generally applicable than the inverse additive algorithm.

# 4 The Gradient Descent Approximation

Most non-linear optimization and parameter estimation algorithms operate by iterating 2 steps. The first step approximately minimizes the optimality criterion, usually by making some sort of linear or quadratic approximation around the current estimate of the parameters. The second step updates the estimate of the parameters. The inverse compositional algorithm, for example, approximately

minimizes the expression in Equation (33) and updates the parameters using Equation (34).

In Sections 2 and 3 above we outlined 4 equivalent quantity approximated-parameter update rule pairs. The approximation that we made in each case is known as the *Gauss-Newton* approximation. In this section we first re-derive the Gauss-Newton approximation for the inverse compositional algorithm and then consider several alternative approximations. Afterwards we apply all of the alternatives to the inverse compositional algorithm, empirically evaluating them in Section 4.6. We conclude this section with a summary of the gradient descent options in Section 4.7 and a review of several other related algorithms in Section 4.8.

## 4.1 The Gauss-Newton Algorithm

The Gauss-Newton inverse compositional algorithm attempts to minimize Equation (33):

$$\sum_{\mathbf{x}} [F(\mathbf{x};\Delta\mathbf{p})]^2 \equiv \sum_{\mathbf{x}} [\, T(\mathbf{W}(\mathbf{x};\Delta\mathbf{p})) - I(\mathbf{W}(\mathbf{x};\mathbf{p}))\,]^2 \tag{66}$$

where $F(\mathbf{x};\Delta\mathbf{p}) = T(\mathbf{W}(\mathbf{x};\Delta\mathbf{p})) - I(\mathbf{W}(\mathbf{x};\mathbf{p}))$ using a first order Taylor expansion of $F(\mathbf{x};\Delta\mathbf{p})$:

$$\sum_{\mathbf{x}} \left[ F(\mathbf{x};\mathbf{0}) + \frac{\partial F}{\partial \mathbf{p}} \Delta\mathbf{p} \right]^2 \equiv \sum_{\mathbf{x}} \left[ T(\mathbf{W}(\mathbf{x};\mathbf{0})) + \nabla T \frac{\partial \mathbf{W}}{\partial \mathbf{p}} \Delta\mathbf{p} - I(\mathbf{W}(\mathbf{x};\mathbf{p})) \right]^2. \tag{67}$$

Here $F(\mathbf{x};\mathbf{0}) = T(\mathbf{W}(\mathbf{x};\mathbf{0})) - I(\mathbf{W}(\mathbf{x};\mathbf{p})) = T(\mathbf{x}) - I(\mathbf{W}(\mathbf{x};\mathbf{p}))$, $\frac{\partial F}{\partial \mathbf{p}} = \nabla T \frac{\partial \mathbf{W}}{\partial \mathbf{p}}$, and we use the expression $\frac{\partial F}{\partial \mathbf{p}}$ to denote the partial derivative of $F$ with respect to its second vector argument $\Delta\mathbf{p}$. The expression in Equation (67) is quadratic in $\Delta\mathbf{p}$ and has the closed form solution:

$$\Delta\mathbf{p} = -H^{-1} \sum_{\mathbf{x}} \left[ \frac{\partial F}{\partial \mathbf{p}} \right]^{\mathrm{T}} F(\mathbf{x};\mathbf{0}) \equiv H^{-1} \sum_{\mathbf{x}} \left[ \nabla T \frac{\partial \mathbf{W}}{\partial \mathbf{p}} \right]^{\mathrm{T}} [\, I(\mathbf{W}(\mathbf{x};\mathbf{p})) - T(\mathbf{x})\,] \tag{68}$$

where $H$ is the first order (Gauss-Newton) approximation to the Hessian matrix:

$$H = \sum_{\mathbf{x}} \left[ \frac{\partial F}{\partial \mathbf{p}} \right]^{\mathrm{T}} \left[ \frac{\partial F}{\partial \mathbf{p}} \right] \equiv \sum_{\mathbf{x}} \left[ \nabla T \frac{\partial \mathbf{W}}{\partial \mathbf{p}} \right]^{\mathrm{T}} \left[ \nabla T \frac{\partial \mathbf{W}}{\partial \mathbf{p}} \right]. \tag{69}$$

One simple way to see that Equations (68) and (69) are the closed-form solution to Equation (67) is to differentiate the expression in Equation (67) with respect to $\Delta\mathbf{p}$ and set the result to zero:

$$2 \sum_{\mathbf{x}} \left[ \frac{\partial F}{\partial \mathbf{p}} \right]^{\mathrm{T}} \left[ F(\mathbf{x};\mathbf{0}) + \frac{\partial F}{\partial \mathbf{p}} \Delta\mathbf{p} \right] = \mathbf{0}. \tag{70}$$

Rearranging this expression gives the closed form result in Equations (68) and (69).

## 4.2 The Newton Algorithm

Rather than writing $F(\mathbf{x};\Delta\mathbf{p}) = T(\mathbf{W}(\mathbf{x};\Delta\mathbf{p})) - I(\mathbf{W}(\mathbf{x};\mathbf{p}))$ and performing a first order Taylor expansion on $F(\mathbf{x};\Delta\mathbf{p})$, in the derivation of the Newton algorithm [8, 12] we write $G(\mathbf{x};\Delta\mathbf{p}) = \frac{1}{2}[F(\mathbf{x};\Delta\mathbf{p})]^2 = \frac{1}{2}[T(\mathbf{W}(\mathbf{x};\Delta\mathbf{p})) - I(\mathbf{W}(\mathbf{x};\mathbf{p}))]^2$ and perform a second order Taylor expansion:

$$\sum_{\mathbf{x}} G(\mathbf{x};\Delta\mathbf{p}) \;=\; \sum_{\mathbf{x}} G(\mathbf{x};\mathbf{0}) + \sum_{\mathbf{x}} \frac{\partial G}{\partial \mathbf{p}}\Delta\mathbf{p} + \frac{1}{2}\sum_{\mathbf{x}} \Delta\mathbf{p}^{\mathrm{T}}\frac{\partial^2 G}{\partial \mathbf{p}^2}\Delta\mathbf{p} \tag{71}$$

where:

$$\sum_{\mathbf{x}} \frac{\partial G}{\partial \mathbf{p}} \;=\; \sum_{\mathbf{x}} \left( \frac{\partial G}{\partial p_1} \; \frac{\partial G}{\partial p_2} \; \cdots \; \frac{\partial G}{\partial p_n} \right) \tag{72}$$

is the gradient of $\sum_{\mathbf{x}} G(\mathbf{x};\Delta\mathbf{p})$ and:

$$\sum_{\mathbf{x}} \frac{\partial^2 G}{\partial \mathbf{p}^2} \;=\; \sum_{\mathbf{x}} \begin{pmatrix} \frac{\partial^2 G}{\partial p_1 \partial p_1} & \frac{\partial^2 G}{\partial p_1 \partial p_2} & \cdots & \frac{\partial^2 G}{\partial p_1 \partial p_n} \\ \frac{\partial^2 G}{\partial p_2 \partial p_1} & \frac{\partial^2 G}{\partial p_2 \partial p_2} & \cdots & \frac{\partial^2 G}{\partial p_2 \partial p_n} \\ \vdots & \vdots & & \vdots \\ \frac{\partial^2 G}{\partial p_n \partial p_1} & \frac{\partial^2 G}{\partial p_n \partial p_2} & \cdots & \frac{\partial^2 G}{\partial p_n \partial p_n} \end{pmatrix} \tag{73}$$

is the Hessian of $\sum_{\mathbf{x}} G(\mathbf{x};\Delta\mathbf{p})$.

### 4.2.1 Relationship with the Hessian in the Gauss-Newton Algorithm

Before we complete the derivation of the Newton algorithm, we briefly explain the relationship between this Hessian and the "first order approximation to the Hessian" in the Gauss-Newton approach. Also see [8] Section 4.7.2. Expanding Equation (67) gives the quadratic expression:

$$\frac{1}{2}\sum_{\mathbf{x}} \left[ F(\mathbf{x};\mathbf{0}) + \frac{\partial F}{\partial \mathbf{p}}\Delta\mathbf{p} \right]^2 \;=\; \frac{1}{2}\sum_{\mathbf{x}} \left[ F^2(\mathbf{x};\mathbf{0}) + 2F(\mathbf{x};\mathbf{0})\frac{\partial F}{\partial \mathbf{p}}\Delta\mathbf{p} + \Delta\mathbf{p}^{\mathrm{T}}\frac{\partial F}{\partial \mathbf{p}}^{\mathrm{T}}\frac{\partial F}{\partial \mathbf{p}}\Delta\mathbf{p} \right]. \tag{74}$$

Comparing Equations (71) and (74) we see that the differences between the Newton and Gauss-Newton approximations are that the gradient is $\frac{\partial G}{\partial \mathbf{p}} = F(\mathbf{x};\mathbf{0})\frac{\partial F}{\partial \mathbf{p}}$ and the Hessian is approximated:

$$\frac{\partial^2 G}{\partial \mathbf{p}^2} \;=\; \frac{\partial F}{\partial \mathbf{p}}^{\mathrm{T}}\frac{\partial F}{\partial \mathbf{p}}. \tag{75}$$

This approximation is a first order approximation in the sense that it is an approximation of the Hessian of $G$ in terms of the gradient of $F$. The Hessian of $F$ is ignored in the approximation. The full expression for the Hessian of $G$ is:

$$\frac{\partial^2 G}{\partial \mathbf{p}^2} \;=\; \frac{\partial F}{\partial \mathbf{p}}^{\mathrm{T}}\frac{\partial F}{\partial \mathbf{p}} + F\frac{\partial^2 F}{\partial \mathbf{p}^2}. \tag{76}$$

### 4.2.2 Derivation of the Gradient and the Hessian

If $G(\mathbf{x};\Delta\mathbf{p}) = \frac{1}{2}[T(\mathbf{W}(\mathbf{x};\Delta\mathbf{p})) - I(\mathbf{W}(\mathbf{x};\mathbf{p}))]^2$ then the gradient is:

$$\frac{\partial G}{\partial \mathbf{p}} = \left[\boldsymbol{\nabla} T \frac{\partial \mathbf{W}}{\partial \mathbf{p}}\right] [\, T(\mathbf{W}(\mathbf{x};\Delta\mathbf{p})) - I(\mathbf{W}(\mathbf{x};\mathbf{p}))\,] \tag{77}$$

where for convenience we use $\frac{\partial G}{\partial \mathbf{p}}$ to denote the partial derivative of $G$ with respect to its second vector argument $\Delta\mathbf{p}$. The Hessian:

$$\begin{aligned}\frac{\partial^2 G}{\partial \mathbf{p}^2} &= \left[\frac{\partial \mathbf{W}}{\partial \mathbf{p}}\right]^{\mathrm{T}} \left[\frac{\partial^2 T}{\partial \mathbf{x}^2}\right] \left[\frac{\partial \mathbf{W}}{\partial \mathbf{p}}\right] [\, T(\mathbf{W}(\mathbf{x};\Delta\mathbf{p})) - I(\mathbf{W}(\mathbf{x};\mathbf{p}))\,] \\ &+ \boldsymbol{\nabla} T \left[\frac{\partial^2 \mathbf{W}}{\partial \mathbf{p}^2}\right] [\, T(\mathbf{W}(\mathbf{x};\Delta\mathbf{p})) - I(\mathbf{W}(\mathbf{x};\mathbf{p}))\,] \\ &+ \left[\boldsymbol{\nabla} T \frac{\partial \mathbf{W}}{\partial \mathbf{p}}\right]^{\mathrm{T}} \left[\boldsymbol{\nabla} T \frac{\partial \mathbf{W}}{\partial \mathbf{p}}\right] \end{aligned} \tag{78}$$

where:

$$\frac{\partial T}{\partial \mathbf{x}} = \begin{pmatrix} \frac{\partial^2 T}{\partial x^2} & \frac{\partial^2 T}{\partial x \partial y} \\ \frac{\partial^2 T}{\partial x \partial y} & \frac{\partial^2 T}{\partial y^2} \end{pmatrix} \tag{79}$$

is the matrix of second derivatives of the template $T$ and:

$$\boldsymbol{\nabla} T \left[\frac{\partial^2 \mathbf{W}}{\partial \mathbf{p}^2}\right] = \frac{\partial T}{\partial x} \begin{pmatrix} \frac{\partial^2 W_x}{\partial p_1 \partial p_1} & \cdots & \frac{\partial^2 W_x}{\partial p_1 \partial p_n} \\ \vdots & & \vdots \\ \frac{\partial^2 W_x}{\partial p_n \partial p_1} & \cdots & \frac{\partial^2 W_x}{\partial p_n \partial p_n} \end{pmatrix} + \frac{\partial T}{\partial y} \begin{pmatrix} \frac{\partial^2 W_y}{\partial p_1 \partial p_1} & \cdots & \frac{\partial^2 W_y}{\partial p_1 \partial p_n} \\ \vdots & & \vdots \\ \frac{\partial^2 W_y}{\partial p_n \partial p_1} & \cdots & \frac{\partial^2 W_y}{\partial p_n \partial p_n} \end{pmatrix}. \tag{80}$$

Equations (77) and (78) hold for arbitrary $\Delta\mathbf{p}$. In the Newton algorithm, we just need their values at $\Delta\mathbf{p} = \mathbf{0}$. When $\Delta\mathbf{p} = \mathbf{0}$, the gradient simplifies to:

$$\frac{\partial G}{\partial \mathbf{p}} = \left[\boldsymbol{\nabla} T \frac{\partial \mathbf{W}}{\partial \mathbf{p}}\right] [\, T(\mathbf{x}) - I(\mathbf{W}(\mathbf{x};\mathbf{p}))\,] \tag{81}$$

and the Hessian simplifies to: $\frac{\partial^2 G}{\partial \mathbf{p}^2} =$

$$\left(\left[\frac{\partial \mathbf{W}}{\partial \mathbf{p}}\right]^{\mathrm{T}} \left[\frac{\partial^2 T}{\partial \mathbf{x}^2}\right] \left[\frac{\partial \mathbf{W}}{\partial \mathbf{p}}\right] + \boldsymbol{\nabla} T \left[\frac{\partial^2 \mathbf{W}}{\partial \mathbf{p}^2}\right]\right) [T(\mathbf{x}) - I(\mathbf{W}(\mathbf{x};\mathbf{p}))] + \left[\boldsymbol{\nabla} T \frac{\partial \mathbf{W}}{\partial \mathbf{p}}\right]^{\mathrm{T}} \left[\boldsymbol{\nabla} T \frac{\partial \mathbf{W}}{\partial \mathbf{p}}\right]. \tag{82}$$

These expressions depend on the Jacobian $\frac{\partial \mathbf{W}}{\partial \mathbf{p}}$ and the Hessian $\frac{\partial^2 \mathbf{W}}{\partial \mathbf{p}^2}$ of the warp. For the affine warp of Equation (2) these values are:

$$\frac{\partial \mathbf{W}}{\partial \mathbf{p}} = \begin{pmatrix} x & 0 & y & 0 & 1 & 0 \\ 0 & x & 0 & y & 0 & 1 \end{pmatrix} \quad \& \quad \frac{\partial^2 \mathbf{W}}{\partial \mathbf{p}^2} = \mathbf{0}. \tag{83}$$

See Appendix A for the derivation of the equivalent expressions for the homography.

### 4.2.3 Derivation of the Newton Algorithm

The derivation of the Newton algorithm begins with Equation (71), the second order approximation to $G(\mathbf{x};\Delta\mathbf{p}) = \frac{1}{2}[T(\mathbf{W}(\mathbf{x};\Delta\mathbf{p})) - I(\mathbf{W}(\mathbf{x};\mathbf{p}))]^2$. Differentiating this expression with respect to $\Delta\mathbf{p}$ and equating the result with zero yields [8, 12]:

$$\left[\sum_{\mathbf{x}} \frac{\partial G}{\partial \mathbf{p}}\right]^{\mathrm{T}} + \sum_{\mathbf{x}} \frac{\partial^2 G}{\partial \mathbf{p}^2}\Delta\mathbf{p} \;=\; 0. \tag{84}$$

The minimum is then attained at:

$$\Delta\mathbf{p} \;=\; -\left[\sum_{\mathbf{x}} \frac{\partial^2 G}{\partial \mathbf{p}^2}\right]^{-1} \left[\sum_{\mathbf{x}} \frac{\partial G}{\partial \mathbf{p}}\right]^{\mathrm{T}} \tag{85}$$

where the gradient $\frac{\partial G}{\partial \mathbf{p}}$ and the Hessian $\frac{\partial^2 G}{\partial \mathbf{p}^2}$ are given in Equations (81) and (82). Ignoring the sign change and the transpose, Equations (81), (82), and (85) are almost identical to Equations (37) and (38) in the description of the inverse compositional algorithm in Section 3.2. The only difference is the second order term (the first term) in the Hessian in Equation (82); if this term is dropped the equations result in exactly the same expression for $\Delta\mathbf{p}$. When the second order term is included, the Hessian is no longer independent of the parameters $\mathbf{p}$ because it depends on $\mathbf{p}$ through $I(\mathbf{W}(\mathbf{x};\mathbf{p}))$. The Hessian therefore has to be re-computed in each iteration. The Newton inverse compositional algorithm is outlined in Figure 10.

### 4.2.4 Computational Cost of the Newton Inverse Compositional Algorithm

The computational cost of the Newton inverse compositional algorithm is far more than the Gauss-Newton version because the Hessian has to be re-computed every iteration. The second order approximation to the Hessian is also more complex and requires that the second order derivatives of the template and the warp be pre-computed. Steps 1 and 2 are the same in the two algorithms. In Step 3 the second order derivatives of the template have to be computed, but these also just take time $\mathrm{O}(N)$ so there is no asymptotic increase. In Step 4 the Hessian of the warp has to be computed which typically takes time $\mathrm{O}(n^2\,N)$, an increase over the cost of just computing the Jacobian. The term $[\frac{\partial \mathbf{W}}{\partial \mathbf{p}}]^{\mathrm{T}}[\frac{\partial^2 T}{\partial \mathbf{x}^2}][\frac{\partial \mathbf{W}}{\partial \mathbf{p}}] + \boldsymbol{\nabla} T[\frac{\partial^2 \mathbf{W}}{\partial \mathbf{p}^2}]$ that has to be computed in Step 5 also takes time $\mathrm{O}(n^2\,N)$. Computing the Hessian in Step 6 takes time $\mathrm{O}(n^2\,N)$. Steps 7, 8, and 9 are all very similar to the Gauss-Newton algorithm. Overall, the pre-computation takes time $\mathrm{O}(n^2\,N)$ and the cost per iteration is $\mathrm{O}(n^2\,N + n^3)$, a substantial increase. The Newton inverse compositional algorithm is asymptotically as slow as the original Lucas-Kanade algorithm. See Table 7 for a summary.

## 4.3 Steepest Descent

The difference between the Gauss-Newton algorithm and the Newton algorithm is the approximation made to $G(\mathbf{x};\Delta\mathbf{p}) = \frac{1}{2}[T(\mathbf{W}(\mathbf{x};\Delta\mathbf{p})) - I(\mathbf{W}(\mathbf{x};\mathbf{p}))]^2$. The Gauss-Newton algorithm performs a first order Taylor expansion on $F(\mathbf{x};\Delta\mathbf{p}) = T(\mathbf{W}(\mathbf{x};\Delta\mathbf{p})) - I(\mathbf{W}(\mathbf{x};\mathbf{p}))$, the Newton

**The Newton Inverse Compositional Algorithm**

Pre-compute:

(3) Evaluate the gradient $\nabla T$ and the second derivatives $\frac{\partial^2 T}{\partial \mathbf{x}^2}$
(4) Evaluate the Jacobian $\frac{\partial \mathbf{W}}{\partial \mathbf{p}}$ and the Hessian $\frac{\partial^2 \mathbf{W}}{\partial \mathbf{p}^2}$ at $(\mathbf{x}; \mathbf{0})$
(5) Compute $\nabla T \frac{\partial \mathbf{W}}{\partial \mathbf{p}}$, $[\frac{\partial \mathbf{W}}{\partial \mathbf{p}}]^{\mathrm{T}}[\frac{\partial^2 T}{\partial \mathbf{x}^2}][\frac{\partial \mathbf{W}}{\partial \mathbf{p}}] + \nabla T[\frac{\partial^2 \mathbf{W}}{\partial \mathbf{p}^2}]$, and $[\nabla T \frac{\partial \mathbf{W}}{\partial \mathbf{p}}]^{\mathrm{T}}[\nabla T \frac{\partial \mathbf{W}}{\partial \mathbf{p}}]$

Iterate:

(1) Warp $I$ with $\mathbf{W}(\mathbf{x}; \mathbf{p})$ to compute $I(\mathbf{W}(\mathbf{x}; \mathbf{p}))$
(2) Compute the error image $I(\mathbf{W}(\mathbf{x}; \mathbf{p})) - T(\mathbf{x})$
(6) Compute the Hessian matrix $\sum_{\mathbf{x}} \frac{\partial^2 G}{\partial \mathbf{p}^2}$ using Equation (82)
(7) Compute $[\sum_{\mathbf{x}} \frac{\partial G}{\partial \mathbf{p}}]^{\mathrm{T}} = \sum_{\mathbf{x}} [\nabla T \frac{\partial \mathbf{W}}{\partial \mathbf{p}}]^{\mathrm{T}} [T(\mathbf{x}) - I(\mathbf{W}(\mathbf{x}; \mathbf{p}))]$
(8) Compute $\Delta \mathbf{p}$ using Equation (85)
(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 10: Compared to the Gauss-Newton inverse compositional algorithm in Figure 4, the Newton inverse compositional algorithm is considerably more complex. The Hessian varies from iteration to iteration and so has to be re-computed each time. The expression for the Hessian is also considerably more complex and requires the second order derivatives of both the template $T$ and the warp $\mathbf{W}(\mathbf{x}; \mathbf{p})$. The computational cost of the Newton inverse compositional algorithm is $\mathrm{O}(n^2 N + n^3)$ and the pre-computation cost is $\mathrm{O}(n^2 N)$.

Table 7: The computation cost of the Newton inverse compositional algorithm. The one time pre-computation cost in Steps 3-5 is $\mathrm{O}(n^2 N)$. After that, the cost of each iteration is $\mathrm{O}(n^2 N + n^3)$, substantially more than the cost of the Gauss-Newton inverse compositional algorithm described in Figure 4, and asymptotically the same as the original Lucas-Kanade algorithm described in Figure 1 in Section 2.

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

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

algorithm performs a second order expansion on $G(\mathbf{x}; \Delta \mathbf{p}) = \frac{1}{2}[T(\mathbf{W}(\mathbf{x}; \Delta \mathbf{p})) - I(\mathbf{W}(\mathbf{x}; \mathbf{p}))]^2$. The Gauss-Newton algorithm effectively approximates the full Newton Hessian: $\frac{\partial^2 G}{\partial \mathbf{p}^2} =$

$$\left( \left[ \frac{\partial \mathbf{W}}{\partial \mathbf{p}} \right]^{\mathrm{T}} \left[ \frac{\partial^2 T}{\partial \mathbf{x}^2} \right] \left[ \frac{\partial \mathbf{W}}{\partial \mathbf{p}} \right] + \nabla T \left[ \frac{\partial^2 \mathbf{W}}{\partial \mathbf{p}^2} \right] \right) [\, T(\mathbf{x}) - I(\mathbf{W}(\mathbf{x}; \mathbf{p}))\, ] + \left[ \nabla T \frac{\partial \mathbf{W}}{\partial \mathbf{p}} \right]^{\mathrm{T}} \left[ \nabla T \frac{\partial \mathbf{W}}{\partial \mathbf{p}} \right] \tag{86}$$

with the first order term:

$$\left[ \nabla T \frac{\partial \mathbf{W}}{\partial \mathbf{p}} \right]^{\mathrm{T}} \left[ \nabla T \frac{\partial \mathbf{W}}{\partial \mathbf{p}} \right]. \tag{87}$$

The advantage of this approximation is that the Hessian is then constant and can be pre-computed, unlike in the Newton algorithm where it has to be re-computed every iteration.

A natural question that follows is then, are there other approximations to the Hessian that result in other efficient gradient descent algorithms? The simplest possibility is to approximate the Hessian as proportional to the identity matrix. If we approximate:

$$\left[\sum_{\mathbf{x}} \frac{\partial^2 G}{\partial \mathbf{p}^2}\right]^{-1} = c\,\mathrm{I} \tag{88}$$

where $\mathrm{I}$ is the $n \times n$ identity matrix, we obtain the *steepest-descent algorithm*. Substituting this expression into Equation (85) yields:

$$\Delta\mathbf{p} = -c \left[\sum_{\mathbf{x}} \frac{\partial G}{\partial \mathbf{p}}\right]^{\mathrm{T}}; \tag{89}$$

i.e. the parameters are updated with a multiple of the gradient of the error $G(\mathbf{x}; \Delta\mathbf{p})$.

### 4.3.1 Derivation of the Steepest Descent Algorithm

One remaining question is how to choose $c$? Should it be a constant, or should it vary from iteration to iteration? In general, choosing $c$ is quite involved [8]. One simple way to choose $c$ is to substitute Equation (89) back into Equation (71) to yield:

$$\sum_{\mathbf{x}} G(\mathbf{x}; \Delta\mathbf{p}) = \sum_{\mathbf{x}} G(\mathbf{x}; \mathbf{0}) - c \left[\sum_{\mathbf{x}} \frac{\partial G}{\partial \mathbf{p}}\right] \left[\sum_{\mathbf{x}} \frac{\partial G}{\partial \mathbf{p}}\right]^{\mathrm{T}} + \frac{1}{2} c^2 \left[\sum_{\mathbf{x}} \frac{\partial G}{\partial \mathbf{p}}\right] \left[\sum_{\mathbf{x}} \frac{\partial^2 G}{\partial \mathbf{p}^2}\right] \left[\sum_{\mathbf{x}} \frac{\partial G}{\partial \mathbf{p}}\right]^{\mathrm{T}}. \tag{90}$$

Differentiating this expression with respect to $c$ to obtain the minimum value yields:

$$c = \frac{\left[\sum_{\mathbf{x}} \frac{\partial G}{\partial \mathbf{p}}\right] \left[\sum_{\mathbf{x}} \frac{\partial G}{\partial \mathbf{p}}\right]^{\mathrm{T}}}{\left[\sum_{\mathbf{x}} \frac{\partial G}{\partial \mathbf{p}}\right] \left[\sum_{\mathbf{x}} \frac{\partial^2 G}{\partial \mathbf{p}^2}\right] \left[\sum_{\mathbf{x}} \frac{\partial G}{\partial \mathbf{p}}\right]^{\mathrm{T}}} \tag{91}$$

where, as before:

$$\frac{\partial G}{\partial \mathbf{p}} = \left[\nabla T \frac{\partial \mathbf{W}}{\partial \mathbf{p}}\right] [\, T(\mathbf{x}) - I(\mathbf{W}(\mathbf{x}; \mathbf{p}))\,]. \tag{92}$$

This is why we called $\nabla T \frac{\partial \mathbf{W}}{\partial \mathbf{p}}$ the steepest descent images and $\sum_{\mathbf{x}} [\nabla T \frac{\partial \mathbf{W}}{\partial \mathbf{p}}]^{\mathrm{T}} [T(\mathbf{x}) - I(\mathbf{W}(\mathbf{x}; \mathbf{p}))]$ the steepest descent parameter updates in the Lucas-Kanade algorithm in Section 2.

This approach to determining $c$ has the obvious problem that it requires the Hessian $\sum_{\mathbf{x}} \frac{\partial^2 G}{\partial \mathbf{p}^2}$. Using the full Newton approximation to the Hessian therefore results in a slow algorithm. One solution is to use the Gauss-Newton approximation to the Hessian in Equation (91) to compute $c$. Since the Hessian is just being used to estimate a single number, it is not vital that its estimate is perfect. Another solution is to use an adaptive algorithm to compute $c$. For example, an approach like the one used in the Levenberg-Marquardt Algorithm described in Section 4.5 could be used. In this paper, we use the first solution because it compares more naturally with the other algorithms. The Gauss-Newton steepest descent inverse compositional algorithm is summarized in Figure 11.

**The GN Steepest Descent Inverse Compositional Algorithm**

Pre-compute:

(3) Evaluate the gradient $\nabla T$ 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 $\nabla T \frac{\partial \mathbf{W}}{\partial \mathbf{p}}$
(6) Compute $\sum_{\mathbf{x}} \frac{\partial^2 G}{\partial \mathbf{p}^2} = \sum_{\mathbf{x}} [\nabla T \frac{\partial \mathbf{W}}{\partial \mathbf{p}}]^{\mathrm{T}} [\nabla T \frac{\partial \mathbf{W}}{\partial \mathbf{p}}]$

Iterate:

(1) Warp $I$ with $\mathbf{W}(\mathbf{x}; \mathbf{p})$ to compute $I(\mathbf{W}(\mathbf{x}; \mathbf{p}))$
(2) Compute the error image $I(\mathbf{W}(\mathbf{x}; \mathbf{p})) - T(\mathbf{x})$
(7) Compute $\sum_{\mathbf{x}} [\frac{\partial G}{\partial \mathbf{p}}]^{\mathrm{T}} = \sum_{\mathbf{x}} [\nabla T \frac{\partial \mathbf{W}}{\partial \mathbf{p}}]^{\mathrm{T}} [I(\mathbf{W}(\mathbf{x}; \mathbf{p})) - T(\mathbf{x})]$
(8) Compute $c$ using Equation (91) and $\Delta \mathbf{p}$ using Equation (89)
(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 Gauss-Newton steepest descent inverse compositional algorithm estimates the parameter updates $\Delta \mathbf{p}$ as constant multiples of the gradient $\sum_{\mathbf{x}} [\frac{\partial G}{\partial \mathbf{p}}]^{\mathrm{T}}$ rather than multiplying the gradient with the inverse Hessian. The computation of the constant multiplication factor $c$ requires the Hessian however, estimated here with the Gauss-Newton approximation. The steepest descent inverse compositional algorithm is slightly faster than the original inverse compositional algorithm and takes time $\mathrm{O}(n\,N + n^2)$ per iteration.

Table 8: The computation cost of the GN steepest descent inverse compositional algorithm. The only change from the original inverse compositional algorithm is in Step 8. Instead of inverting the Hessian, Equation (91) is used to compute the value of $c$. This takes time $\mathrm{O}(n^2)$ rather than time $\mathrm{O}(n^3)$. The cost per iteration is therefore $\mathrm{O}(n\,N + n^2)$ rather than $\mathrm{O}(n\,N + n^3)$. The pre-computation time is still $\mathrm{O}(n^2\,N)$.

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

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

### 4.3.2 Computational Cost of the Gauss-Newton Steepest Descent Algorithm

The only step that is modified from the original inverse compositional algorithm is Step 8. Instead of inverting the Hessian and multiplying the gradient by the result, the value of $c$ is estimated in Equation (91) and then multiplied by the gradient in Equation (89). This takes time $\mathrm{O}(n^2)$ rather than time $\mathrm{O}(n^3)$. The GN steepest descent algorithm therefore takes time $\mathrm{O}(n\,N + n^2)$ per iteration. The pre-computation cost is exactly the same at $\mathrm{O}(n^2\,N)$. See Table 8 for a summary.

## 4.4 The Diagonal Approximation to the Hessian

The steepest descent algorithm can be regarded as approximating the Hessian with the identity matrix. The next simplest approximation is to make a diagonal approximation to the Hessian:

$$\text{Diag}\left[\frac{\partial^2 G}{\partial \mathbf{p}^2}\right] = \sum_{\mathbf{x}} \begin{pmatrix} \frac{\partial^2 G}{\partial p_1 \partial p_1} & 0 & \cdots & 0 \\ 0 & \frac{\partial^2 G}{\partial p_2 \partial p_2} & \cdots & 0 \\ \vdots & \vdots & & \vdots \\ 0 & 0 & \cdots & \frac{\partial^2 G}{\partial p_n \partial p_n} \end{pmatrix}. \tag{93}$$

If only the elements on the leading diagonal are ever computed, the computational cost of all steps, both the pre-computation cost and in each iteration, becomes at most $\text{O}(n\,N)$ rather than $\text{O}(n^2\,N)$. If we denote the vector:

$$\text{Diag}\left[\sum_{\mathbf{x}} \frac{\partial^2 G}{\partial \mathbf{p}^2}\right]^{-1} = \left(\frac{1}{\sum_{\mathbf{x}} \frac{\partial^2 G}{\partial p_1 \partial p_1}}, \frac{1}{\sum_{\mathbf{x}} \frac{\partial^2 G}{\partial p_2 \partial p_2}}, \ldots, \frac{1}{\sum_{\mathbf{x}} \frac{\partial^2 G}{\partial p_n \partial p_n}}\right)^{\text{T}} \tag{94}$$

the update to the warp parameters in Equation (85) becomes:

$$\Delta \mathbf{p} = -\text{Diag}\left[\sum_{\mathbf{x}} \frac{\partial^2 G}{\partial \mathbf{p}^2}\right]^{-1} \cdot \left[\sum_{\mathbf{x}} \frac{\partial G}{\partial \mathbf{p}}\right]^{\text{T}} \tag{95}$$

where the product $\cdot$ between these two vectors denotes element-wise multiplication and where the gradient and Hessian are given by Equations (81) and (82). The diagonal approximation to the Hessian can then be used directly in the Newton algorithm. See Figure 12 for a summary. This approximation is commonly used in optimization problems with a large number of parameters. Examples of this diagonal approximation in vision include stereo [15] and super-resolution [1].

### 4.4.1 Computational Cost of the Diagonal Approximation to the Hessian

The diagonal approximation to the Hessian makes the Newton inverse compositional algorithm far more efficient. Because there are only $n$ elements on the leading diagonal, compared to $n^2$ elements in the entire Hessian matrix, Steps 4, 5, and 6 only take time $\text{O}(n\,N)$ rather than $\text{O}(n^2\,N)$. Step 8 is also slightly quicker because the Hessian does not have to be inverted. It takes time $\text{O}(n)$ rather than $\text{O}(n^3)$. Overall, the pre-computation takes time $\text{O}(n\,N)$ and the cost per iteration only $\text{O}(n\,N + n^2)$. This cost is comparable with the Gauss-Newton inverse compositional algorithm. The cost of the Newton algorithm with the diagonal approximation is summarized in Table 9.

## 4.5 The Levenberg-Marquardt Algorithm

Of the various approximations, generally the steepest descent and diagonal approximations work better further away from the local minima, and the Newton and Gauss-Newton approximations

**The Diagonal Hessian Inverse Compositional Algorithm**

Pre-compute:

(3) Evaluate the gradient $\nabla T$ and the second derivatives $\frac{\partial^2 T}{\partial \mathbf{x}^2}$
(4) Evaluate $\frac{\partial \mathbf{W}}{\partial \mathbf{p}}$ and $\text{Diag}[\frac{\partial^2 \mathbf{W}}{\partial \mathbf{p}^2}]$ at $(\mathbf{x}; \mathbf{0})$
(5) Compute $\nabla T \frac{\partial \mathbf{W}}{\partial \mathbf{p}}$ and $\text{Diag}[(\frac{\partial \mathbf{W}}{\partial \mathbf{p}})^{\text{T}}(\frac{\partial^2 T}{\partial \mathbf{x}^2})(\frac{\partial \mathbf{W}}{\partial \mathbf{p}}) + \nabla T(\frac{\partial^2 \mathbf{W}}{\partial \mathbf{p}^2})]$

Iterate:

(1) Warp $I$ with $\mathbf{W}(\mathbf{x}; \mathbf{p})$ to compute $I(\mathbf{W}(\mathbf{x}; \mathbf{p}))$
(2) Compute the error image $I(\mathbf{W}(\mathbf{x}; \mathbf{p})) - T(\mathbf{x})$
(6) Compute the Diagonal Hessian matrix $\text{Diag}[\sum_{\mathbf{x}} \frac{\partial^2 G}{\partial \mathbf{p}^2}]$
(7) Compute $[\sum_{\mathbf{x}} \frac{\partial G}{\partial \mathbf{p}}]^{\text{T}} = \sum_{\mathbf{x}} [\nabla T \frac{\partial \mathbf{W}}{\partial \mathbf{p}}]^{\text{T}} [T(\mathbf{x}) - I(\mathbf{W}(\mathbf{x}; \mathbf{p}))]$
(8) Compute $\Delta \mathbf{p}$ using Equation (95)
(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 12: The Newton inverse compositional algorithm with the diagonal approximation to the Hessian is almost exactly the same as the full Newton version in Figure 10. The only difference is that only the elements on the leading diagonals of the Hessian are computed in Steps 4, 5, and 6, and used to compute the update to the parameters in Step 8. The result is a far more efficient algorithm. See Table 9 for a summary.

Table 9: The computational cost of the Newton inverse compositional algorithm with the diagonal approximation to the Hessian. See Figure 12 for the algorithm. Because only $n$ elements in the Hessian ever need to be computed, Steps 4, 5, and 6 take time $\text{O}(n\,N)$ rather than $\text{O}(n^2\,N)$. Step 8 is also slightly quicker. Overall, the pre-computation only takes time $\text{O}(n\,N)$ and the cost per iteration is only $\text{O}(n\,N + n^2)$.

| 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 |
|---|---|---|---|---|---|---|---|
| | $\text{O}(n\,N)$ | $\text{O}(N)$ | $\text{O}(n\,N)$ | $\text{O}(n\,N)$ | $\text{O}(n)$ | $\text{O}(n^2)$ | $\text{O}(n\,N + n^2)$ |

work better close to the local minima where the quadratic approximation is good [8, 12]. Another aspect that none of the algorithms that we have discussed so far take into account is whether the error gets better or worse after each iteration. If the error gets worse, a gradient descent algorithm might want to try a smaller step in the same direction rather than making the error worse.

One algorithm that tries to combine the diagonal approximation with the full Hessian to get the best of both worlds is the well known Levenberg-Marquardt algorithm. Either the Gauss-Newton Hessian or the Newton Hessian can be used. We will use the Gauss-Newton approximation to get

an efficient algorithm. The Gauss-Newton Levenberg-Marquardt algorithm uses the Hessian:

$$\mathrm{H}_{\mathrm{LM}} = \sum_{\mathbf{x}} \left[ \boldsymbol{\nabla} T \frac{\partial \mathbf{W}}{\partial \mathbf{p}} \right]^{\mathrm{T}} \left[ \boldsymbol{\nabla} T \frac{\partial \mathbf{W}}{\partial \mathbf{p}} \right] + \delta \sum_{\mathbf{x}} \begin{pmatrix} \left( \boldsymbol{\nabla} T \frac{\partial \mathbf{W}}{\partial p_1} \right)^2 & 0 & \cdots & 0 \\ 0 & \left( \boldsymbol{\nabla} T \frac{\partial \mathbf{W}}{\partial p_2} \right)^2 & \cdots & 0 \\ \vdots & \vdots & & \vdots \\ 0 & 0 & \cdots & \left( \boldsymbol{\nabla} T \frac{\partial \mathbf{W}}{\partial p_n} \right)^2 \end{pmatrix} \tag{96}$$

where $\boldsymbol{\nabla} T \frac{\partial \mathbf{W}}{\partial p_i} = \frac{\partial T}{\partial x} \frac{\partial W_x}{\partial p_i} + \frac{\partial T}{\partial y} \frac{\partial W_y}{\partial p_i}$, $\delta > 0$, and the warp update parameters are estimated:

$$\Delta \mathbf{p} = -{\mathrm{H}_{\mathrm{LM}}}^{-1} \sum_{\mathbf{x}} \left[ \boldsymbol{\nabla} T \frac{\partial \mathbf{W}}{\partial \mathbf{p}} \right] [\, T(\mathbf{x}) - I(\mathbf{W}(\mathbf{x}; \mathbf{p})) \,]. \tag{97}$$

For very small $\delta \ll 1$, the Levenberg-Marquardt Hessian $\mathrm{H}_{\mathrm{LM}}$ is approximately the Gauss-Newton Hessian. For large $\delta \gg 1$, the Hessian is approximately the Gauss-Newton diagonal approximation to the Hessian, but with a reduced step size of $\frac{1}{\delta}$. Levenberg-Marquardt starts with a small initial value of $\delta$, say $\delta = 0.01$. After each iteration, the parameters are provisionally updated and the new error evaluated. If the error has decreased, the value of delta is reduced, $\delta \rightarrow \delta/10$, say. If the error has increased, the provisional update to the parameters is reversed and $\delta$ increased, $\delta \rightarrow \delta \times 10$, say. The Levenberg-Marquardt inverse compositional algorithm is summarized in Figure 13.

### 4.5.1 Computational Cost of the Levenberg-Marquardt Inverse Compositional Algorithm

Compared with the Gauss-Newton inverse compositional algorithm, the Levenberg-Marquardt algorithm requires that a number of steps be re-ordered, a couple of steps to be extended, and two new steps added. The re-ordering doesn't affect the computation cost of the algorithm; it only marginally increases the pre-computation time, although not asymptotically. The extra computation of $\mathrm{e}$ in Step 2 doesn't affect the asymptotic time taken for that step. The computation of $\mathrm{H}_{\mathrm{LM}}$ in Step 8 only takes time $\mathrm{O}(n^2)$ compared to $\mathrm{O}(n^3)$ to subsequently invert the Hessian and $\mathrm{O}(n^2)$ to multiply it by the steepest descent parameter updates. The new Steps 0 and 10 both take very little time, constant if implemented with pointers to the new and old data structures. Overall the Levenberg-Marquardt algorithm is just as efficient as the Gauss-Newton inverse compositional algorithm taking time $\mathrm{O}(n\,N + n^3)$ per iteration and $\mathrm{O}(n^2\,N)$ as a pre-computation cost.

## 4.6 Empirical Validation

We have described six variants of the inverse compositional image alignment algorithm: Gauss-Newton, Newton, Gauss-Newton steepest descent, diagonal Hessian (Gauss-Newton and Newton), and Levenberg-Marquardt. We now empirically compare these algorithms. We only present results for the affine warp. The results for the homography are similar and omitted for lack of space.

**The Levenberg-Marquardt Inverse Compositional Algorithm**

Pre-compute:

(0) Initialize $\delta = 0.01$
(1) Warp $I$ with $\mathbf{W}(\mathbf{x};\mathbf{p})$ to compute $I(\mathbf{W}(\mathbf{x};\mathbf{p}))$
(2) Comp. $I(\mathbf{W}(\mathbf{x};\mathbf{p})) - T(\mathbf{x})$, $\mathrm{e} = \sum_{\mathbf{x}} [I(\mathbf{W}(\mathbf{x};\mathbf{p})) - T(\mathbf{x})]^2$
(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 $\boldsymbol{\nabla} T \frac{\partial \mathbf{W}}{\partial \mathbf{p}}$
(6) Compute the GN Hessian matrix $\sum_{\mathbf{x}} [\boldsymbol{\nabla} T \frac{\partial \mathbf{W}}{\partial \mathbf{p}}]^{\mathrm{T}} [\boldsymbol{\nabla} T \frac{\partial \mathbf{W}}{\partial \mathbf{p}}]$

Iterate:

(7) Compute $\sum_{\mathbf{x}} [\boldsymbol{\nabla} T \frac{\partial \mathbf{W}}{\partial \mathbf{p}}]^{\mathrm{T}} [I(\mathbf{W}(\mathbf{x};\mathbf{p})) - T(\mathbf{x})]$
(8) Compute $\mathrm{H}_{\mathrm{LM}}$ using Equation (96) and $\Delta \mathbf{p}$ using Equation (97)
(9) Update the warp $\mathbf{W}(\mathbf{x};\mathbf{p}) \leftarrow \mathbf{W}(\mathbf{x};\mathbf{p}) \circ \mathbf{W}(\mathbf{x};\Delta\mathbf{p})^{-1}$
(1) Warp $I$ with $\mathbf{W}(\mathbf{x};\mathbf{p})$ to compute $I(\mathbf{W}(\mathbf{x};\mathbf{p}))$
(2) Comp. $I(\mathbf{W}(\mathbf{x};\mathbf{p})) - T(\mathbf{x})$, $\mathrm{e}^* = \sum_{\mathbf{x}} [I(\mathbf{W}(\mathbf{x};\mathbf{p})) - T(\mathbf{x})]^2$
(10) If $\mathrm{e} < \mathrm{e}^*$ then $\delta \rightarrow \delta \times 10$; undo Steps 9, 1, & 2; else $\delta \rightarrow \delta / 10$; $\mathrm{e} = \mathrm{e}^*$

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

Figure 13: The Gauss-Newton Levenberg-Marquardt inverse compositional algorithm. Besides using the Levenberg-Marquardt Hessian in Step 8 and reordering Steps 1 and 2 slightly, the algorithm checks whether the error e has decreased at the end of each iteration in Step 10. If the error decreased, the value of $\delta$ is reduced and the next iteration started. If the error increased, the value of $\delta$ is increased and the current update reversed by "undoing" steps 9, 1, & 2. The computational cost of the Levenberg-Marquardt algorithm is detailed in Table 10. The algorithm is just as efficient as the original Gauss-Newton inverse compositional algorithm and operates in time $\mathrm{O}(n\,N + n^3)$ per iteration. The pre-computation cost is $\mathrm{O}(n^2\,N)$.

### 4.6.1 Average Frequency of Convergence

We begin with the average frequency of convergence rather than the average rate of convergence to eliminate some of the algorithms that perform very poorly. In Figure 14(a) we plot the average frequency of convergence (computed over 5000 samples) with no intensity noise. In Figure 14(b) we plot the average frequency of convergence (again computed over 5000 samples) with pixel intensity noise standard deviation 8.0 grey levels added to both the template and the input image. As in Section 3.4, we say that an algorithm failed to converge if after 15 iterations the RMS error in the canonical point locations is greater than 1.0 pixels.

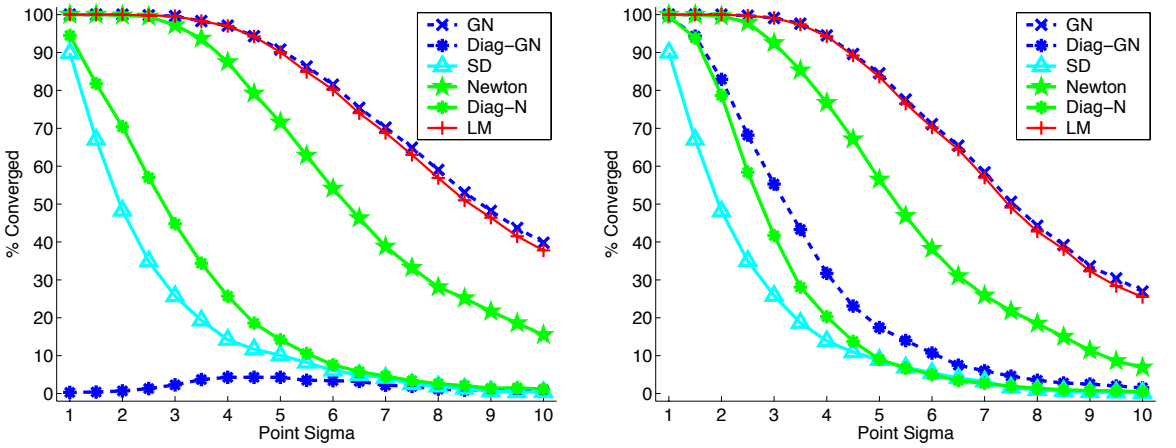
The first thing to notice in Figure 14 is that the best performing algorithms are Gauss-Newton and Levenberg-Marquardt. Perhaps surprisingly, Levenberg-Marquardt doesn't perform any better than Gauss-Newton, even for larger displacements of the canonical point locations.

The second thing to notice is that the Newton algorithm performs significantly worse than Gauss-Newton. In general, for many optimization problems we expect the Newton algorithm to perform better because it uses a more sophisticated estimate of the Hessian. That expectation,

Table 10: The computation cost of the Levenberg-Marquardt inverse compositional algorithm. A number of steps have been re-ordered, a couple of steps have been extended, and two new steps have been introduced compared to the original algorithm in Figure 4. However, overall the new algorithm is just as efficient as the original algorithm taking time $O(n\,N + n^3)$ per iteration and $O(n^2\,N)$ as a pre-computation cost.

| Pre-Computation | Step 0 | Step 1 | Step 2 | Step 3 | Step 4 | Step 5 | Step 6 | Total |
|---|---|---|---|---|---|---|---|---|
| | $O(1)$ | $O(n\,N)$ | $O(N)$ | $O(N)$ | $O(n\,N)$ | $O(n\,N)$ | $O(n^2\,N)$ | $O(n^2\,N)$ |

| Per Iteration | Step 7 | Step 8 | Step 9 | Step 1 | Step 2 | Step 10 | Total |
|---|---|---|---|---|---|---|---|
| | $O(n\,N)$ | $O(n^3)$ | $O(n^2)$ | $O(n\,N)$ | $O(N)$ | $O(1)$ | $O(n\,N + n^3)$ |




(a) Frequency of Convergence with No Noise (b) Frequency of Convergence with Noise SD 8.0

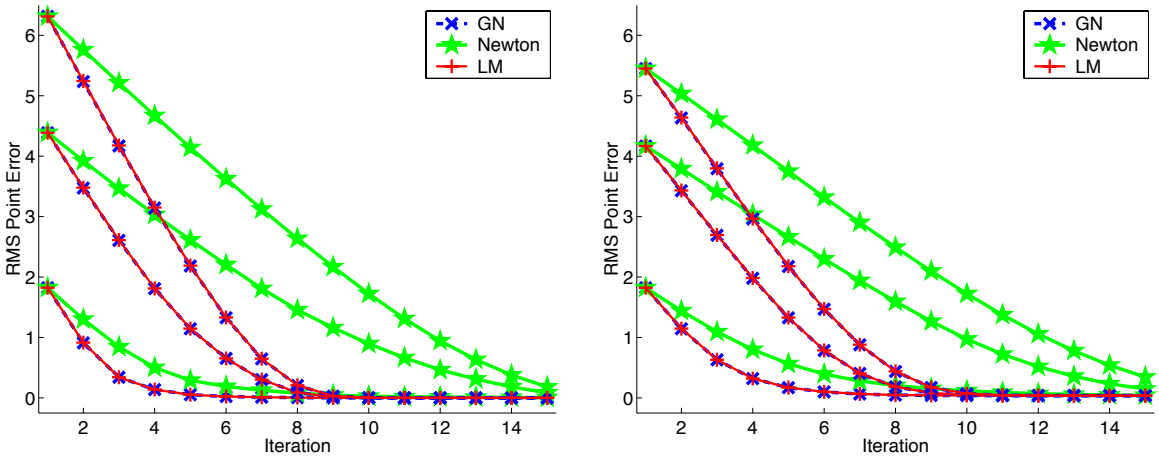
Figure 14: The average frequency of convergence of the six variants of the inverse compositional algorithm: Gauss-Newton, Newton, steepest descent, diagonal Hessian approximation (Gauss-Newton and Newton), and Levenberg-Marquardt. We find that Gauss-Newton and Levenberg-Marquardt perform the best, with Newton significantly worse. The three other algorithms all perform very poorly indeed.

however, relies on the assumption that the estimate of the Hessian is noiseless. In our case, the Gauss-Newton Hessian depends on the gradient of the template $T(\mathbf{x})$ and so is noisy. The Newton Hessian also depends on the second derivatives of the template. It appears that the increased noise in estimating the second derivatives of the template outweighs the increased sophistication in the algorithm. Overall, we conclude that the full Newton Hessian should not be used.

The final thing to notice in Figure 14 is that the steepest descent and diagonal Hessian approximations perform very poorly. We conclude that it is important to use a full Hessian approximation. More will be said on the performance of these three algorithms in Sections 4.6.3 and 4.6.4 below.

### 4.6.2 Average Convergence Rates

Since the steepest descent and diagonal Hessian algorithms converge so rarely, we only plot the average rate of convergence for the Gauss-Newton, Newton, and Levenberg-Marquardt algorithms. The other three algorithms converge very slowly, and sometimes oscillate. See Section 4.6.3 for




(a) Convergence Rate with No Noise (b) Convergence Rate with Noise SD 8.0

Figure 15: The average rates of convergence of the Gauss-Newton, Newton, and Levenberg-Marquardt algorithms. The other three algorithms converge so slowly that the results are omitted. Gauss-Newton and Levenberg-Marquardt converge similarly, and the fastest. Newton converges significantly slower.

more discussion. In Figure 15(a) we plot the average rate of convergence with no intensity noise and in Figure 15(b) the average rate of convergence with noise standard deviation 8.0 grey levels added to both the template and the input image. The results are consistent with the results in Section 4.6.1. The Gauss-Newton and Levenberg-Marquardt algorithms converge the quickest. The Newton algorithm converges significantly slower on average, even with no noise.

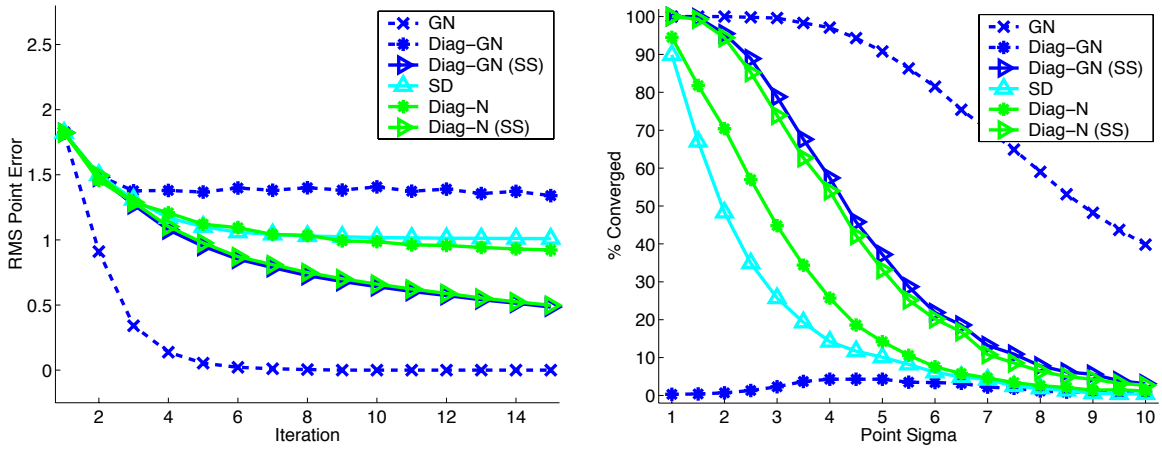
### 4.6.3 Performance of Steepest Descent and the Diagonal Hessian Approximations

The steepest descent and diagonal Hessian approximations perform very poorly. One particular surprise in Figure 14 is that the steepest descent algorithm outperforms the Gauss-Newton diagonal approximation to the Hessian. How is this possible? It seems that the diagonal approximation to the Hessian should always be better than approximating the Hessian with the identity.

The reason that the steepest descent algorithm performs better than the diagonal Hessian approximation algorithms is that it uses the Gauss-Newton Hessian to determine the step size. See Equations (91) and (92). Although it does better at estimating the direction of descent, the diagonal approximation to the Hessian often does a poor job of estimating the step size. As a result, it can "oscillate" (or even diverge) if its estimate of the step size is too big (which is the case here.)

It is possible to add a similar step size estimation step to the diagonal Hessian approximation algorithms, again using the Gauss-Newton Hessian. (Other step-size estimation algorithms could also be used. See [8] for some possibilities.) The performance of the diagonal Hessian algorithms improves dramatically when this step is added. See Figure 16 for a comparison of the performance of the diagonal Hessian algorithms with and without the step size estimation step.

The diagonal approximation to the Hessian is used in various vision algorithms such as stereo [15] and super-resolution [1]. Our results indicate that these algorithms may be under-performing




(a) Average Convergence Rate (b) Average Frequency of Convergence

Figure 16: The performance of the diagonal approximation to the Hessian algorithms, both with and without the step size estimation step used in the steepest descent algorithm. See Equations (91) and (92). With the step size estimation step, the performance of the diagonal Hessian algorithms improves dramatically.

because of the optimization algorithm used, especially if the step-size is not chosen correctly.

### 4.6.4 Importance of Parameterization

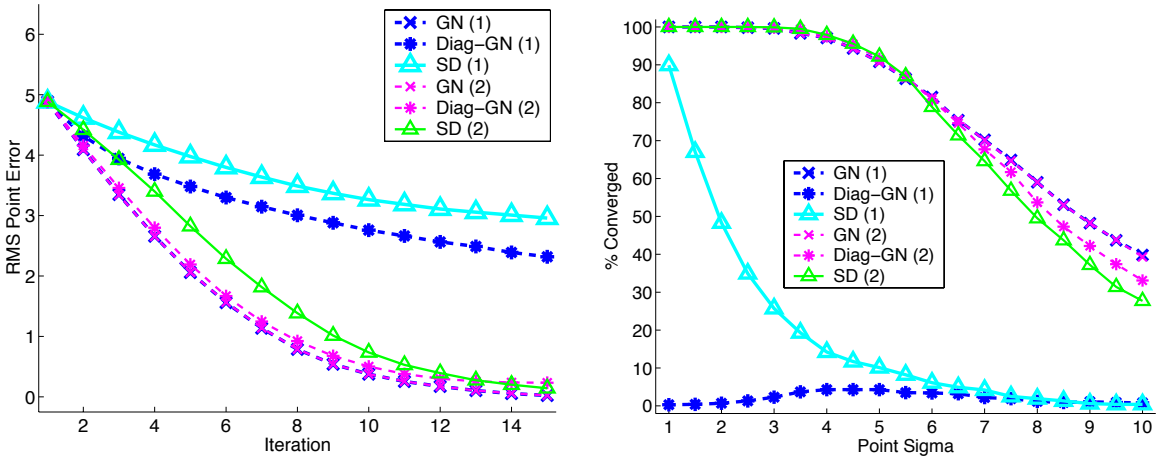
Even after correcting for the step size the performance of steepest descent and the diagonal Hessian algorithms is still surprisingly poor. One reason the performance is so poor is the parameterization. There are, in general, multiple ways to parameterize a set of warps. For affine warps the parameterization in Equation (1) is not the only way. Another way to parameterize affine warps is by the destination location of three canonical points, similar to those used in our error measure.

Both steepest descent and the diagonal Hessian algorithms are very dependent on the parameterization. Even a linear reparameterization like the one just described can dramatically change the performance. Gauss-Newton is much less dependent on the parameterization. To illustrate this point, we quickly reimplemented the algorithms using a parameterization based on the destination of three canonical points. With this new parameterization, steepest descent and the diagonal Hessian algorithms perform far better, although still a little worse than Gauss-Newton. See Figure 17.

This dramatic dependence on the parameterization is another reason for not using steepest descent or the diagonal approximations to the Hessian. It also again brings into question the optimization algorithms used in stereo [15] and super-resolution [1]. More on the question of how to best parameterize a set of warps is left to Part 2 of this 2 paper series.

### 4.6.5 Timing Results

The timing results in milliseconds for our Matlab implementation of the six algorithms are included in Table 11. These results are for the 6-parameter affine warp using a $100 \times 100$ pixel grey-scale template on a 933MHz Pentium-IV. As can be seen, all of the algorithms are roughly equally fast




(a) Average Convergence Rate (b) Average Frequency of Convergence

Figure 17: The performance of Gauss-Newton, steepest descent, and the Gauss-Newton diagonal approximation to the Hessian for two parameterizations of affine warps: (1) the parameterization in Equation (1) and (2) a parameterization based on the location of three canonical points. Gauss-Newton is relatively unaffected by the reparameterization whereas the performance of the other algorithms is dramatically improved.

except the Newton algorithm which is much slower. The diagonal Newton algorithm is also a little slower because it has to compute the Hessian matrix each iteration. The diagonal Gauss-Newton and steepest descent algorithms are a little faster to pre-compute, although not per iteration.

## 4.7 Summary

In this section we investigated the choice of the gradient descent approximation. Although image alignment algorithms have traditionally used the Gauss-Newton first order approximation to the Hessian, this is not the only possible choice. We have exhibited five alternatives: (1) Newton, (2) steepest descent, (3) diagonal approximation to the Gauss-Newton Hessian, (4) diagonal approximation to the Newton Hessian, and (5) Levenberg-Marquardt. Table 12 contains a summary of the six gradient descent approximations we considered.

We found that steepest descent and the diagonal approximations to the Hessian all perform very poorly, both in terms of the convergence rate and in terms of the frequency of convergence. These three algorithms are also very sensitive to the estimation of the step size and the parameterization of the warps. The Newton algorithm performs significantly worse than the Gauss-Newton algorithm (although better than the other three algorithms.) The most likely reason is the noise introduced in computing the second derivatives of the template. Levenberg-Marquardt can be implemented just as efficiently as Gauss-Newton, but performs no better than it.

We considered the six gradient descent approximations combined with the inverse compositional algorithm. Except for the Newton algorithm all of the alternatives are equally as efficient as the Gauss-Newton algorithm when combined with the inverse compositional algorithm. Any of the alternatives could also be used with the forwards additive or forwards compositional algorithms.

Table 11: Timing results for our Matlab implementation of the six algorithms in milliseconds. These results are for the 6-parameter affine warp using a $100 \times 100$ pixel grey-scale template on a 933MHz Pentium-IV. Steps 0 and 10 for Levenberg-Marquardt are negligible and so are omitted for lack of space.
**Precomputation:**

| | Step 1 | Step 2 | Step 3 | Step 4 | Step 5 | Step 6 | Total |
|---|---|---|---|---|---|---|---|
| Gauss-Newton (GN) | - | - | 8.31 | 17.1 | 27.5 | 37.0 | 90.0 |
| Newton (N) | - | - | 24.5 | 17.1 | 209 | - | 250 |
| Steepest-Descent (SD) | - | - | 8.36 | 17.0 | 27.5 | 36.6 | 89.5 |
| Gauss-Newton Diagonal (Diag-GN) | - | - | 8.31 | 17.1 | 27.5 | 4.48 | 57.4 |
| Newton Diagonal (Diag-N) | - | - | 24.4 | 17.1 | 78.4 | - | 120 |
| Levenberg-Marquardt (LM) | 1.83 | 0.709 | 8.17 | 17.1 | 27.6 | 40.8 | 96.2 |

**Per Iteration:**

| | Step 1 | Step 2 | Step 3 | Step 4 | Step 5 | Step 6 | Step 7 | Step 8 | Step 9 | Total |
|---|---|---|---|---|---|---|---|---|---|---|
| GN | 1.79 | 0.687 | - | - | - | - | 6.22 | 0.106 | 0.409 | 9.21 |
| N | 1.76 | 0.713 | - | - | - | 39.5 | 5.99 | 0.106 | 0.414 | 48.5 |
| SD | 1.76 | 0.688 | - | - | - | - | 6.22 | 0.140 | 0.409 | 9.22 |
| Diag-GN | 1.78 | 0.777 | - | - | - | - | 6.16 | 0.107 | 0.414 | 9.23 |
| Diag-N | 1.88 | 0.692 | - | - | - | 8.20 | 6.53 | 0.107 | 0.417 | 17.8 |
| LM | 1.83 | 0.709 | - | - | - | - | 6.06 | 0.319 | 0.348 | 9.26 |

In this case the Gauss-Newton and Levenberg-Marquardt algorithms are less efficient. Steepest descent and the diagonal Hessian algorithms are still efficient however. The details of these derivations are omitted due to lack of space. The essence is that only the diagonal of the Hessian ever needs to be computed for these algorithms and so they are always efficient.

## 4.8 Other Algorithms

Although we have considered six different gradient descent approximations, these are not the only choices. Numerous other algorithms can be used. Some of the alternatives are as follows.

### 4.8.1 Other Approximations to the Hessian

The focus of this section has been approximating the Hessian: the Gauss-Newton approximation, the steepest descent (identity) approximation, and the diagonal approximations. Other ways of approximating the Hessian have also been considered. In [14] an algorithm is proposed to estimate the Gauss-Newton Hessian for the forwards compositional algorithm, but in an efficient manner. One reason that computing the Hessian matrix is so time consuming is that it is a sum over the

Table 12: The six gradient descent approximations that we considered: Gauss-Newton, Newton, steepest descent, Diagonal Hessian (Gauss-Newton & Newton), and Levenberg-Marquardt. When combined with the inverse compositional algorithm the six alternatives are all equally efficient except Newton. When combined with a forwards algorithm, only steepest descent and the diagonal Hessian algorithms are efficient. Only Gauss-Newton and Levenberg-Marquardt converge well empirically.

| Algorithm | Efficient As Inverse? | Efficient As Forwards? | Convergence Rate | Frequency of Convergence |
|---|---|---|---|---|
| Gauss-Newton | Yes | No | Fast | High |
| Newton | No | No | Medium | Medium |
| Steepest Descent | Yes | Yes | Slow | Low |
| Gauss-Newton Diagonal Hessian | Yes | Yes | Slow | Low |
| Newton Diagonal Hessian | Yes | Yes | Slow | Low |
| Levenberg-Marquardt | Yes | No | Fast | High |

entire template. See Equation (11). In [14] it is suggested that this computation can be speeded up by splitting the template into patches and assuming that the term summed in Equation (11) is constant over each patch. The computation of the Hessian can then be performed as a sum over the (center pixels of the) patches rather than over the entire template. A related approach is [7] in which the Hessian (and steepest descent images) are only computed over a subset of the template.

#### 4.8.2 Non Gradient Descent Algorithms

The essence of the gradient descent approximation is to find a linear relationship between the increments to the parameters and the error image. See Equation (10). In the inverse compositional algorithm, we analytically derived an algorithm in which this linear relationship has constant coefficient. Since the coefficients are constant they can be pre-computed. Other approaches consist of: (1) assuming that linear relationship is constant and using linear regression to find the coefficients [6] and (2) numerically estimating the coefficients using a "difference decomposition" [9, 13, 5].

## 5 Discussion

We have described a unifying framework for image alignment consisting of two halves. In the first half we classified algorithms by the quantity approximated and the warp update rule. Algorithms can be classified as either *additive* or *compositional* and as either *forwards* or *inverse*. In the second half we considered the gradient descent approximation. We considered the *Gauss-Newton* approximation, the *Newton* approximation, the *steepest descent* approximation, two *diagonal Hessian* approximations, and the *Levenberg-Marquardt* approximation. These three choices are orthogonal. For example, one could think of the *forwards compositional steepest descent* algorithm.

Overall the choice of which algorithm to use depends on two main things: (1) whether there is likely to me more noise in the template or in the input image and (2) whether the algorithm

needs to be efficient or not. If there is more noise in the template a forwards algorithm should be used. If there is more noise in the input image an inverse algorithm should be used. If an efficient algorithm is required, the best options are the inverse compositional Gauss-Newton and the inverse compositional Levenberg-Marquardt algorithms. The diagonal Hessian and steepest descent forwards algorithms are another option, but given their poor convergence properties it is probably better to use the inverse compositional algorithm even if the template is noisy.

Besides the choices we have described in this paper, there are several other ones that can be made by an image alignment algorithm. These include the choice of the error norm, whether to allow illumination or more general appearance variation, whether to add priors on the parameters, and whether to use various heuristics to avoid local minima. In Part 2 of this 2 paper series we will extend our framework to cover these choices and, in particular, investigate whether the inverse compositional algorithm is compatible with these extensions of the Lucas-Kanade algorithm.

# Acknowledgments

We would like to thank Bob Collins, Matthew Deans, Frank Dellaert, Daniel Huber, Takeo Kanade, Jianbo Shi, Sundar Vedula, and Jing Xiao for discussions on image alignment, and Sami Romdhani for pointing out a couple of algebraic errors in a preliminary draft of this paper. We would also like to thank the anonymous CVPR reviewers of [2] for their feedback. The research described in this paper was conducted under U.S. Office of Naval Research contract N00014-00-1-0915.

# References

[1] S. Baker and T. Kanade. Limits on super-resolution and how to break them. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2000.

[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*, 2001.

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

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

[5] M. La Cascia, S. Sclaroff, and V. Athitsos. Fast, reliable head tracking under varying illumination: An approach based on registration of texture-mapped 3D models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(6):322–336, 2000.

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

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

[8] P.E. Gill, W. Murray, and M.H. Wright. *Practical Optimization*. Academic Press, 1986.

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

[10] 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.

[11] 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*, 1981.

[12] W.H. Press, B.P. Flannery, S.A. Teukolsky, and W.T. Vetterling. *Numerical Recipes in C: The Art of Scientific Computing*. Cambridge University Press, second edition, 1992.

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

[14] 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.

[15] R. Szeliski and P. Golland. Stereo matching with transparency and matting. In *Proceedings of the 6th IEEE International Conference on Computer Vision*, 1998.

# A Inverse Compositional Derivations for the Homography

## A.1 Gauss-Newton Inverse Compositional Algorithm

To apply the Gauss-Newton inverse compositional algorithm (see Figure 4 in Section 3.2) to a new set of warps, we need to do four things: (1) specify the set of warps $\mathbf{W}(\mathbf{x};\mathbf{p})$, (2) derive the Jacobian, (3) derive the expression for the composition of a warp and an incremental warp $\mathbf{W}(\mathbf{x};\mathbf{p}) \circ \mathbf{W}(\mathbf{x};\Delta\mathbf{p})$, and (4) derive the expression for the inverse of a warp $\mathbf{W}(\mathbf{x};\mathbf{p})^{-1}$. We now perform these four steps for homographies. Homographies have 8 parameters $\mathbf{p} = (p_1, p_2, p_3, p_4, p_5, p_6, p_7, p_8)^{\mathrm{T}}$ and can be parameterized:

$$\mathbf{W}(\mathbf{x};\mathbf{p}) = \frac{1}{1+p_7x+p_8y} \begin{pmatrix} (1+p_1)x & + & p_3y & + & p_5 \\ p_2x & + & (1+p_4)y & + & p_6 \end{pmatrix}. \tag{98}$$

There are other ways to parameterize homographies, however this way is perhaps the most common. The Jacobian of the homography in Equation (98) is:

$$\frac{\partial \mathbf{W}}{\partial \mathbf{p}} = \frac{1}{1+p_7x+p_8y} \begin{pmatrix} x & 0 & y & 0 & 1 & 0 & \frac{-x[(1+p_1)x+p_3y+p_5]}{1+p_7x+p_8y} & \frac{-y[(1+p_1)x+p_3y+p_5]}{1+p_7x+p_8y} \\ 0 & x & 0 & y & 0 & 1 & \frac{-x[p_2x+(1+p_4)y+p_6]}{1+p_7x+p_8y} & \frac{-y[p_2x+(1+p_4)y+p_6]}{1+p_7x+p_8y} \end{pmatrix}. \quad (99)$$

The parameters of $\mathbf{W}(\mathbf{x};\mathbf{p}) \circ \mathbf{W}(\mathbf{x};\Delta\mathbf{p})$ are:

$$\frac{1}{1+p_7\Delta p_5+p_8\Delta p_6} \begin{pmatrix} p_1+\Delta p_1+p_1\Delta p_1+p_3\Delta p_2+p_5\Delta p_7-p_7\Delta p_5-p_8\Delta p_6 \\ p_2+\Delta p_2+p_2\Delta p_1+p_4\Delta p_2+p_6\Delta p_7 \\ p_3+\Delta p_3+p_1\Delta p_3+p_3\Delta p_4+p_5\Delta p_8 \\ p_4+\Delta p_4+p_2\Delta p_3+p_4\Delta p_4+p_6\Delta p_8-p_7\Delta p_5-p_8\Delta p_6 \\ p_5+\Delta p_5+p_1\Delta p_5+p_3\Delta p_6 \\ p_6+\Delta p_6+p_2\Delta p_5+p_4\Delta p_6 \\ p_7+\Delta p_7+p_7\Delta p_1+p_8\Delta p_2 \\ p_8+\Delta p_8+p_7\Delta p_3+p_8\Delta p_4 \end{pmatrix}. \quad (100)$$

Finally, the parameters of $\mathbf{W}(\mathbf{x};\mathbf{p})^{-1}$ are:

$$\frac{1}{\det \cdot [(1+p_1)(1+p_4)-p_2p_3]} \begin{pmatrix} 1+p_4-p_6p_8-\det \cdot [(1+p_1)(1+p_4)-p_2p_3] \\ -p_2+p_6p_7 \\ -p_3+p_5p_8 \\ 1+p_1-p_5p_7-\det \cdot [(1+p_1)(1+p_4)-p_2p_3] \\ -p_5-p_4p_5+p_3p_6 \\ -p_6-p_1p_6+p_2p_5 \\ -p_7-p_4p_7+p_2p_8 \\ -p_8-p_1p_8+p_3p_7 \end{pmatrix} \quad (101)$$

where $\det$ is the determinant:

$$\det = \begin{vmatrix} 1+p_1 & p_3 & p_5 \\ p_2 & 1+p_4 & p_6 \\ p_7 & p_8 & 1 \end{vmatrix}. \quad (102)$$

## A.2 Newton Inverse Compositional Algorithm

To be able to apply the Newton inverse compositional algorithm (see Section 4.2) to the homography we also need the Hessian of the homography. The Hessian has two components one for each

of the components of the warp $\mathbf{W}(\mathbf{x};\mathbf{p}) = (W_x(\mathbf{x};\mathbf{p}), W_y(\mathbf{x};\mathbf{p}))^{\mathrm{T}}$. The $x$ component is $\frac{\partial^2 W_x}{\partial \mathbf{p}^2} =$

$$\frac{1}{(1+p_7x+p_8y)^2}\begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & -x^2 & -xy \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -xy & -y^2 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -x & -y \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -x^2 & 0 & -xy & 0 & -x & 0 & \frac{2x^2[(1+p_1)x+p_3y+p_5]}{1+p_7x+p_8y} & \frac{2xy[(1+p_1)x+p_3y+p_5]}{1+p_7x+p_8y} \\ -xy & 0 & -y^2 & 0 & -y & 0 & \frac{2xy[(1+p_1)x+p_3y+p_5]}{1+p_7x+p_8y} & \frac{2y^2[(1+p_1)x+p_3y+p_5]}{1+p_7x+p_8y} \end{pmatrix} \tag{103}$$

and the $y$ component is $\frac{\partial^2 W_y}{\partial \mathbf{p}^2} =$

$$\frac{1}{(1+p_7x+p_8y)^2}\begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -x^2 & -xy \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -xy & -y^2 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -x & -y \\ 0 & -x^2 & 0 & -xy & 0 & -x & \frac{2x^2[p_2x+(1+p_4)y+p_6]}{1+p_7x+p_8y} & \frac{2xy[p_2x+(1+p_4)y+p_6]}{1+p_7x+p_8y} \\ 0 & -xy & 0 & -y^2 & 0 & -y & \frac{2xy[p_2x+(1+p_4)y+p_6]}{1+p_7x+p_8y} & \frac{2y^2[p_2x+(1+p_4)y+p_6]}{1+p_7x+p_8y} \end{pmatrix}. \tag{104}$$