International Journal of Computer Vision 56(3), 221–255, 2004
© 2004 Kluwer Academic Publishers. Manufactured in The Netherlands.

Lucas-Kanade 20 Years On: A Unifying Framework

SIMON BAKER AND IAIN MATTHEWS
The Robotics Institute, Carnegie Mellon University, 5000 Forbes Avenue, Pittsburgh, PA 15213, USA
simonb@cs.cmu.edu
iainm@cs.cmu.edu

Received July 10, 2002; Revised February 6, 2003; Accepted February 7, 2003

추상. Lucas-Kanade 알고리즘이 1981 년에 제안 된 이래로 이미지 정렬은 컴퓨터 비전에서 가장 널리 사용되는 기술 중 하나가되었습니다. 응용 프로그램은 광학 흐름 및 추적에서 계층화 된 모션, 모자이크 구성 및 얼굴 코딩에 이르기까지 다양합니다. 수많은 알고리즘이 제안되었고 다양한 확장이 원래 공식에 만들어졌습니다. 우리는 일관된 프레임 워크에서 대부분의 알고리즘과 그 확장을 설명하는 이미지 정렬에 대한 개요를 제시합니다. 최근에 제안한 효율적인 알고리즘 인 역 합성 알고리즘에 중점을 둡니다. Lucas-Kanade에 대한 확장이 효율성을 크게 떨어 뜨리지 않고 역 합성 알고리즘과 함께 사용할 수 있는지 여부를 검토합니다. 본 백서의 제 1 부에서는 일련의 논문에서 대략적인 양, 워프 업데이트 규칙 및 그래디언트 디센트 근사를 다룹니다. 향후 논문에서는 오류 기능의 선택, 선형 외형 변화 허용 방법 및 매개 변수에 priors를 부과하는 방법에 대해 다룰 것입니다.

Keywords: 이미지 정렬, Lucas-Kanade, 통합 프레임 워크, 첨가물 대 구성 알고리즘, 전진 대 역 알고리즘, 역 구성 알고리즘, 효율성, 가장 가파른 하강, Gauss-Newton, Newton, Levenberg-Marquardt

1. Introduction

이미지 정렬은 템플리트와 이미지의 차이를 최소화하기 위해 템플리트를 이동하고 가능한 경우 변형하는 것으로 구성됩니다.
Lucas-Kanade 옵티컬 플로우 알고리즘 (Lucas and Kanade, 1981)에서 이미지 정렬을 처음 사용 한 이래로 이미지 정렬은 컴퓨터 비전에서 가장 널리 사용되는 기술 중 하나가되었습니다. 옵티컬 플로우 외에도 다른 일부 애플리케이션으로는 추적 (Black and Jepson, 1998; Hager and Belhumeur, 1998), 파라 메트릭 및 계층화 된 모션 추정 (Bergen et al., 1992), 모자이크 구조 (Shum and Szeliski, 2000) 이미지 등록 (Christensen and Johnson, 2001), 얼굴 코딩 (Baker and Matthews, 2001, Cootes et al., 1998).

이미지 정렬에 대한 일반적인 접근법은 그래디언트 디센트입니다.

차이 분해 (Gleicher, 1997) 및 선형 회귀 (Cootes et al., 1998)와 같은 다양한 다른 수치 알고리즘도 제안되었지만, 그래디언트 디센트가 표준이었다.
그래디언트 디센트는 다양한 방법으로 수행 될 수 있습니다. 다양한 접근법 사이의 한 가지 차이점은 매개 변수에 대한 추가 증분을 추정하는지 (추가 접근법 (Lucas and Kanade, 1981)) 또는 증분 워프를 추정하는지 여부입니다. 증분 워프는 현재 워프 추정과 구성됩니다 (구성 접근법 (Shum and Szeliski, 2000)). 또 다른 차이점은 알고리즘이 Gauss-Newton, Newton, 가장 가파른 하강 또는 Levenberg-Marquardt 근사를 각각의 그래디언트 강하 단계에서 수행하는지 여부입니다

우리는 이미지 정렬을위한 통합 프레임 워크를 제안하여 다양한 알고리즘과 확장을 일관된 방식으로 설명합니다.

프레임 워크 전반에 걸쳐 우리는 최근에 제안한 효율적인 알고리즘 인 역 합성 알고리즘에 중점을 둡니다 (Baker and Matthews, 2001). Lucas-Kanade에 대한 확장이 효율성을 크게 떨어 뜨리지 않고 역 합성 알고리즘에 적용 할 수 있는지, 추가 확장이 필요한지 검토합니다. 가능한 한 우리는 다양한 알고리즘과 그 확장을 설명하기 위해 경험적 결과를 제공합니다.

본 백서에서는 일련의 논문 중 1 부에서 Lucas-Kanade 알고리즘을 검토하여 섹션 2부터 설명하겠습니다. 우리는 다양한 이미지 정렬 알고리즘과 사용되는 워프 업데이트 규칙에 의해 근사되는 양을 분석하기 위해 3 절을 진행합니다.
우리는 알고리즘을 가산 또는 합성으로 분류하고 순방향 또는 역순으로 분류합니다. 우리는 다양한 대안의 첫 번째 등가성을 증명하고, 결과 알고리즘의 효율성을 도출하고, 각 대안을 적용 할 수있는 일련의 경사를 설명하고, 마지막으로 경험적으로 알고리즘을 비교합니다. 섹션 4에서는 각 반복에서 사용할 수있는 다양한 그래디언트 디센트 근사(계산)를 설명합니다. Gauss-Newton, Newton, 대각선 Hessian, LevenbergMarquardt 및 급강하 (Press 외., 1992).
속도와 경험적인 측면에서 이러한 대안을 비교합니다. 우리는 5 절에서 논의를 마친다. 이 연재물 (웹 사이트 http://www.ri.cmu.edu/projects/projects/projects 515.html에서 제공 될 예정)의 향후 논문에서 오류 기능의 선택, 선형 모양 허용 방법 변형 및 매개 변수에 전임자를 추가하는 방법에 대해 설명합니다.

2. Background: Lucas-Kanade

기존의 이미지 정렬 알고리즘은 Lucas-Kanade 알고리즘을 사용합니다 (Lucas and Kanade, 1981). Lucas-Kanade의 목표는 템플릿 이미지 T (x)를 입력 이미지 I (x)에 정렬하는 것입니다. 여기서 x = (x, y)의 T승은 픽셀 좌표를 포함하는 열 벡터입니다. Lucas-Kanade 알고리즘이 광학 흐름을 계산하거나 시간 t = 1에서 시간 t = 2까지 이미지 패치를 추적하는 데 사용되는 경우 템플릿 T (x)는 추출 된 하위 영역 (5x5 창, 어쩌면 t = 1에서 I (x)는 t = 2에서의 이미지입니다.

.LetW (x; p)는 매개 변수화 된 허용 된 워프 세트를 나타냅니다. 여기서 p = (p1, ... pn)의 T승은 매개 변수의 벡터입니다. 워프 W (x; p)는 템플릿 T의 좌표 프레임에서 픽셀 x를 이미지 I의 좌표 프레임에서 서브 픽셀 위치 W (x; p)로 가져오게 됩니다.
예를 들어 옵티컬 플로우를 계산할 때 워프 W (x; p)는 다음과 같이 번역 될 수 있습니다.

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

여기서 파라미터 p = (p1, p2)의 벡터는 광학 흐름이다. 3D로 움직이는 더 큰 이미지 패치를 추적하는 경우, 대신 아핀 워프 세트를 고려할 수 있습니다.

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

예를 들어, Bergen et al.에서 수행 된 것처럼, 6 개의 파라미터 p = (p1, p2, p3, p4, p5, p6) (1992). (아핀 왜곡을 매개 변수화하는 다른 방법이있다. 이 프레임 워크에서 나중에 가장 좋은 방법을 조사 할 것이다.) 일반적으로 매개 변수 n의 수는 임의로 클 수 있으며 W (x; p)는 임의적 일 수있다. - 이리 복잡한. 복잡한 워프의 한 예는 Active Appearance Models (Cootes et al., 1998, Baker and Matthews,2001)와 Active Blobs (Sclaroff and Isidoro, 1998)에 사용 된 piecewise affine warps의 집합입니다.

2.1. *Goal of the Lucas-Kanade Algorithm*

Lucas-Kanade 알고리즘의 목표는 템플릿 T와 템플릿 I의 좌표 프레임에 뒤틀린 이미지 두 이미지 사이의 제곱 오차의 합을 최소화하는 것입니다.

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

I (W (x; p))를 계산하기 위해 되돌아가는 워핑은 서브 픽셀 위치 W (x; p)에서 이미지 I를 폴링 (polating) 할 것을 요구한다.식의 최소화는 식 (3)은 p에 대하여 형성되고, 그 합은 템플릿 이미지 T (x) 내의 모든 픽셀 x에 대해 수행된다.Eq. (1)은 픽셀 값 I (x)가 일반적으로 x에서 비선형이기 때문에 W (x; p)가 p에서 선형이라 할지라도 비선형 최적화 작업이다.
실제로, 픽셀 값 I (x)는 픽셀 좌표 x와 본질적으로 관련이 없다. Eq. (3)에서, Lucas-Kanade 알고리즘은 p의 현재 추정치가 알려져 있다고 가정하고 파라미터 델타p에 대한 증가분을 반복적으로 계산한다. 즉, 다음 식은 (대략) 최소화된다 :

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

델타p에 관해서는, 다음 매개 변수가 최신입니다 :

$$\mathbf{p} \leftarrow \mathbf{p}+\Delta\mathbf{p}. \qquad (5)$$

이 두 단계는 매개 변수 p의 추정치가 수렴 될 때까지 반복됩니다. 일반적으로 수렴 테스트는 벡터 델타p의 표준이 임계 값 미만인지 여부입니다; ϵ; i.e.$\|\Delta\mathbf{p}\|\leq\epsilon$.

2.2. *Derivation of the Lucas-Kanade Algorithm*

루카스 - 카나데 알고리즘 (Gauss-Newton 그래디언트 디센트 비선형 최적화 알고리즘)은 다음과 같이 유도됩니다. Eq.에서의 비선형 예측 (4)는 I (W (x; p + ρ p))에 대해 1 차 테일러 확장을 수행함으로써 선형화되어 다음을 얻습니다 :

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

이 식에서 $\nabla I = (\frac{\partial I}{\partial x}, \frac{\partial I}{\partial y})$
는 W (x; p)에서 평가 한 이미지의 그래디언트입니다. 즉 델타I는 I의 좌표 프레임에서 계산 된 후 현재의 워프 W (x; p)를 사용하여 T의 좌표 프레임으로 다시 왜곡된다. 용어 ∂W / ∂p는 휘어진 코비안이다. 만약 W (x; p) = (Wx (x; p), Wy (x; p))T승 이라면 :

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

우리는 열 벡터에 대한 편도 함수가 행 벡터로 배치된다는 표기 규칙을 따른다.
이 규칙은 식 (6)의 표현식에서와 같이 행렬 곱셈이 행렬 규칙을 따른다는 이점이있다.
예를 들어, Eq. (2) 야코비 행렬 :

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

The Lucas-Kanade Algorithm

Iterate:

(1) Warp I with $\mathbf{W}(\mathbf{x};\mathbf{p})$ to compute $I(\mathbf{W}(\mathbf{x};\mathbf{p}))$
(2) Compute the error image $T(\mathbf{x})-I(\mathbf{W}(\mathbf{x};\mathbf{p}))$
(3) Warp the gradient ∇I with $\mathbf{W}(\mathbf{x};\mathbf{p})$
(4) Evaluate the Jacobian $\frac{\partial\mathbf{W}}{\partial\mathbf{p}}$ at $(\mathbf{x};\mathbf{p})$
(5) Compute the steepest descent images $\nabla I\frac{\partial\mathbf{W}}{\partial\mathbf{p}}$
(6) Compute the Hessian matrix using Equation (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 $\|\Delta\mathbf{p}\|\leq\epsilon$

Lucas - Kanade알고리즘(Lucas and Kanade,1981)은 Eqs 를 반복적으로 적용하는 것으로 구성된다 .(p)의 추정치가 수렴할때까지 식(10)과 (5)를 적용한다. 일반적으로 수렴 테스트는 벡터 델타 p의 일부표준이 사용자가 지정한 임계값 미만인지 여부입니다. 그라데이션 델타 I 는 W(x; p)에서 평가되어야 하고 Jacobian $\frac{\partial\mathbf{W}}{\partial\mathbf{p}}$ 는 p 에서 평가되어야 하기 때문에 모든 9단계가 알고리즘이 계속 반복되어야 합니다 .

Eq. (6)은 최소 자승 문제이며 다음과 같이 도출 할 수 있는 해를가집니다. 식의 편미분은 다음과 같다. (6)은 델타p에 대하여 :

$$2\sum_{\mathbf{x}}\left[\nabla I\frac{\partial\mathbf{W}}{\partial\mathbf{p}}\right]^{\mathrm{T}}\left[I(\mathbf{W}(\mathbf{x};\mathbf{p}))+\nabla I\frac{\partial\mathbf{W}}{\partial\mathbf{p}}\Delta\mathbf{p}-T(\mathbf{x})\right] \qquad (9)$$

(델타 I $\frac{\partial\mathbf{W}}{\partial\mathbf{p}}$ 는 가장 가파른 강하 영상으로 참조한다.)

(제 4.3 절 참조)이 식을 0으로 놓고 풀면 식 (6)의 최소 식에 대한 닫힌 형태의 해를 구할 수있다.

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

여기서 H는 n × n (가우스 - 뉴턴 근사치) 헤시안 행렬이다.

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

나중에 명확 해지는 이유에 대해서는 가장 가파른 하강 매개 변수 업데이트로
$\sum^{\mathbf{x}}[\nabla I\frac{\partial\mathbf{W}}{\partial\mathbf{p}}]^{\mathrm{T}}[T(\mathbf{x})-I(\mathbf{W}(\mathrm{x};\,\mathrm{p}))]$
를참조합니다. 방정식 (10)은 파라미터 업데이트 델타 p가 헤시안 행렬의 역수를 곱한 가장 가파른 하강 파라미터 갱신이라는 사실을 나타낸다. Lucas - Kanade 알고리즘 (Lucas and Kanade, 1981)은 Eqs를 반복적으로 적용하는 것으로 구성된다. (10) 및 (5)를 만족한다. 그림 참조. 1과 2는 요약입니다. 그라디언트 델타 I는 W (x; p)에서 평가되어야하고 Jacobian $\frac{\partial\mathbf{W}}{\partial\mathbf{p}}$ 는 p

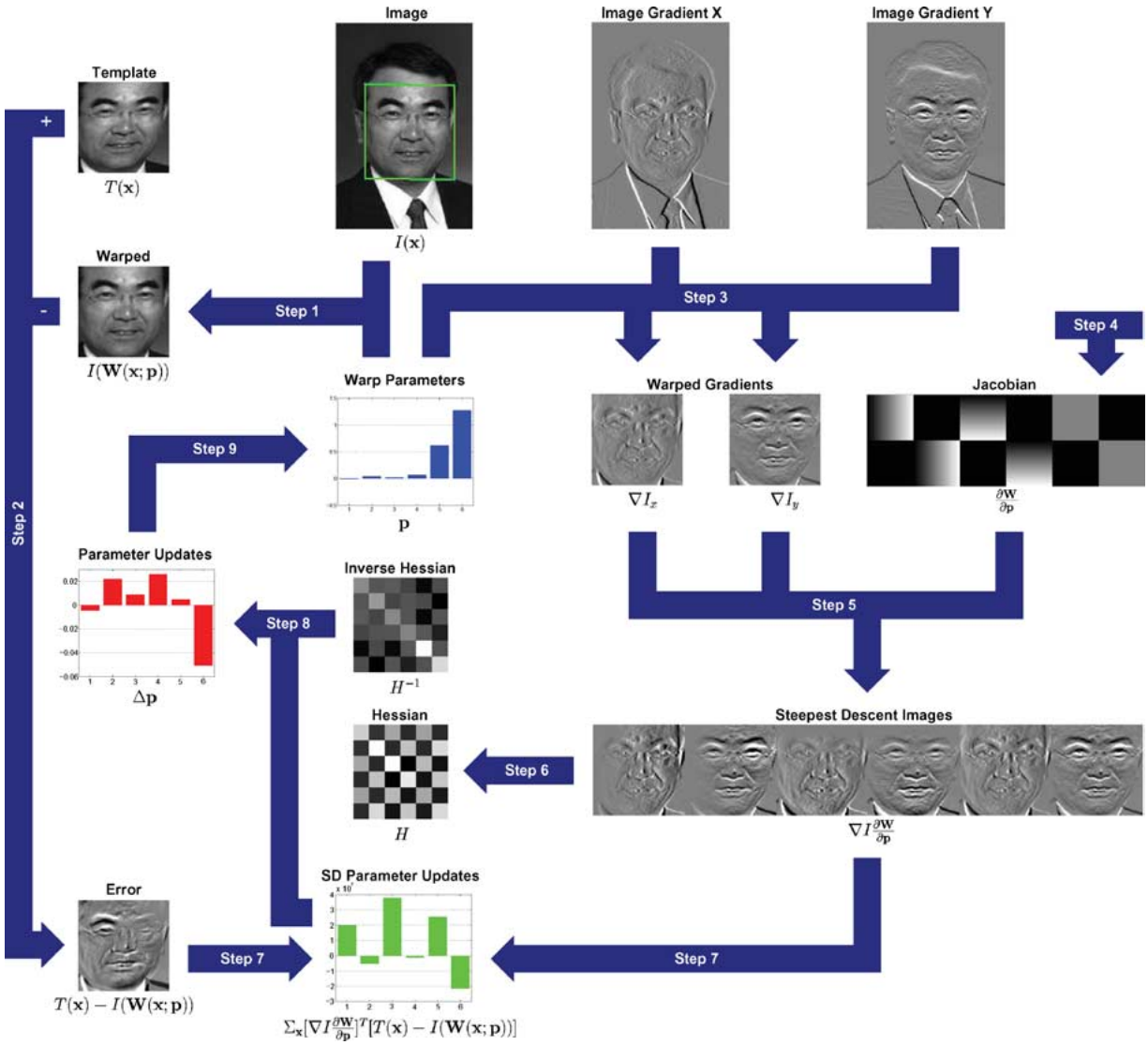


Figure-2 Lucas-Kanade 알고리즘 개요 (Lucas and Kanade, 1981). 이미지 I는 단계 1에서의 워프의 현재 추정치와 뒤틀리고 결과는 단계 2에서 템플릿으로부터 감산되어 에러 이미지를 산출한다. I의 기울기는 3 단계에서 왜곡되고, Jacobian은 4 단계에서 계산되고, 2 단계는 5 단계에서 결합되어 가장 가파른 강하 영상을 제공합니다. 6 단계에서 헤 시안은 가장 가파른 하강 영상으로부터 계산됩니다.Step7에서 가장 가파른 강하 파라미터 업데이트는 에러 이미지를 가장 가파른 하강 영상과 곱해서 계산됩니다. 단계 8에서는 헤 시안이 반전되고 가장 가파른 하강 매개 변수 업데이트 단계가 곱해지고 최종 매개 변수 업데이트 p가 단계 9에서 매개 변수 p에 추가됩니다.

그들은 둘 다 일반적으로 p에 의존한다. 식 (1)에서의 변환과 식 (2)에서의 아핀 워프와 같은 간단한 워프에 대해, 야코비 행렬은 때때로 일정 할 수 있다. 예를 들어 식 (8)을 참조하십시오. 그러나 일반적으로 알고리즘의 모든 9 단계는 모든 반복에서 반복되어야합니다. 왜냐하면 매개 변수 p의 추정값이 계속해서 반복되기 때문입니다.

2.3. *Requirements on the Set of Warps*

워프W(x; p)에 대한 유일한 요구 사항은 그것들이 워프 파라미터 p에 대해 미분 가능하다는 것이다. 이 조건은 야 코비 행렬$\frac{\partial \mathbf{W}}{\partial \mathbf{p}}$을 계산하는 데 필요합니다. 보통 워프는 x와 관련하여 (조각 별) 차별화가 가능하지만 이 조건조차 엄격하게 요구되는 것은 아닙니다.

Table.1 Lucas-Kanade 알고리즘의 한 번의 반복 계산 비용. n이 워프 파라미터의 수이고 N이 템플릿 T의 픽셀 수이면 각 반복의 비용은 O(n²N + n³)입니다. 가장 비용이 많이 드는 단계는 Hessian의 계산인 Step 6으로, 시간 O(n² N)

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

2.4. Computational Cost of the Lucas-Kanade Algorithm

워프 파라미터의 수는 n이고 T의 픽셀 수는 N이라고 가정한다. Lucas-Kanade 알고리즘의 1 단계는 일반적으로 O (n N) 시간이 걸린다. T의 각 픽셀 x에 대해 W (x; p)를 계산 한 다음 그 위치에서 샘플 I을 계산합니다. computingW (x; p)의 계산 비용은 W에 달려 있지만 대부분의 warp의 경우 비용은 픽셀 당 O (n)입니다. 단계 2는 O (N) 시간이 걸린다. 3 단계는 1 단계와 동일한 시간, 일반적으로 O (n N)을 사용합니다. 4 단계에서 자 코비안을 계산하는 것은 W에 달려 있지만 대부분의 워프에서 픽셀 당 비용은 O (n)입니다. 따라서 4 단계의 총 비용은 O (n N)입니다. 단계 5는 시간 O (n N)을 취하고, 단계 6은 시간 O (n² N)을 취하고, 단계 7은 시간 O (n N)을 취한다. 단계 8은 단계 7에서 계산 된 가장 가파른 하강 매개 변수에 결과를 곱하기 위해 헤 시안 행렬과 시간 O (n²)를 반전하는 데 시간 O (n³)을 사용합니다. 단계 9는 업데이트로 매개 변수를 증가시키는 데 시간 O (n) . 그러므로 각 반복의 총 계산 비용은 O (n² N + n³)이며, 가장 비싼 단계는 6 단계입니다. 이러한 계산 비용의 요약은 표 1을 참조하십시오.

3. The Quantity Approximated and the Warp Update Rule

각 반복에서 Lucas-Kanade는 p에 대해 ∑x [I (W (x; p + ∆ρ)) - T (x)]²를 대략 minize한 다음 9 단계에서 매개 변수의 추정을 갱신한다. p ← p + ∆p. 아마 놀랍게도 이 두 단계를 반복하는 것이 Eq.의 식을 최소화하는 유일한 방법은 아닙니다. 섹션 3에서는 Lucas-Kanade 알고리즘과 동등한 3 가지 대체 접근법에 대해 설명합니다. 우리는 경험적으로 그것들이 동등하다는 것을 보여줍니다.

3.1. Compositional Image Alignment

Lucas-Kanade 알고리즘의 첫 번째 대안은 합성 알고리즘입니다.

3.1.1. Goal of the Compositional Algorithm.

Shum and Szeliski (2000)에 의해 가장 많이 사용 된 구성 알고리즘은 대략 다음을 최소화합니다.

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

각각의 반복에서 Δp에 대해 다음과 같이 워핑의 추정치를 업데이트한다 :

$$\mathbf{W}(\mathbf{x};\mathbf{p}) \leftarrow \mathbf{W}(\mathbf{x};\mathbf{p}) \circ \mathbf{W}(\mathbf{x};\Delta\mathbf{p}), \qquad (13)$$

즉, 구성 접근법은 파라미터 ρ에 대한 부가적인 업데이트가 아닌 증분 워핑 W (x; Δp)를 반복적으로 해결한다. 이 문맥에서 우리는 식 (3)에서 Lucas-Kanade 알고리즘을 참조한다. (4)와 (5)를 부가 적 접근법으로 사용하여 식 (12) 및 (13)을 만족한다. 조성 및 추가적 접근법이 섹션 3.1.5에서 [ΔP에서 일차 동등한 것으로 판명된다. 표현식 :

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

2 개의 워프의 구성입니다. 예를 들어, W (x; p)가 식 (2)의 아핀 워프라면 :

$$\mathbf{W}(\mathbf{x};\mathbf{p}) \circ \mathbf{W}(\mathbf{x};\Delta\mathbf{p}) = \begin{pmatrix} (1+p_1)\cdot((1+\Delta p_1)\cdot x + \Delta p_3 \cdot y + \Delta p_5) \\ \quad + p_3 \cdot (\Delta p_2 \cdot x + (1+\Delta p_4)\cdot y + \Delta p_6) \\ \quad + p_5 \\ p_2 \cdot ((1+\Delta p_1)\cdot x + \Delta p_3 \cdot y + \Delta p_5) \\ \quad + (1+p_4)\cdot(\Delta p_2 \cdot x + (1+\Delta p_4)\cdot y \\ \quad + \Delta p_6) + p_6 \end{pmatrix}, \qquad (15)$$

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

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

W (x; p)와 W (x; ▵p)의 매개 변수의 간단한 쌍 선형 조합.

3.1.2. Derivation of the Compositional Algorithm.

합성 접근법에서 Lucas-Kanade 알고리즘의 등가물을 유도하기 위해 다음을 얻기 위해 1 차 테일러 확장을 식 (12)에 적용합니다.

$$\sum_{\mathbf{x}} \left[I(\mathbf{W}(\mathbf{W}(\mathbf{x};\mathbf{0});\mathbf{p})) + \nabla I(\mathbf{W}) \frac{\partial \mathbf{W}}{\partial \mathbf{p}} \Delta\mathbf{p} - T(\mathbf{x}) \right]^2. \quad (17)$$

이 식에서 I (W) (x)는 뒤틀린 이미지 I (W (x; p))를 나타냅니다. 추가 확장이 가능합니다 :

$$\nabla I(\mathbf{W}) \equiv \frac{\partial I(\mathbf{W})}{\partial \mathbf{x}} = \frac{\partial I}{\partial \mathbf{x}} \frac{\partial \mathbf{W}}{\partial \mathbf{x}} = \nabla I \frac{\partial \mathbf{W}}{\partial \mathbf{x}} \quad (18)$$

그러나 워핑 된 이미지 I (W)는 Lucas Kanade 알고리즘의 1 단계에서 계산되므로 (그림 1 참조) 그래디언트를 쉽게 대신 사용할 수 있으므로이 작업을 수행하지 않아도됩니다.

계속 진행하기 위해 우리는 하나의 가정을합니다. 우리는 W (x; 0)가 항등 왜곡이라고 가정한다. 즉 W (x; 0) = x이다.아이덴티티 워프가 고려되고있는 워프 집합에 있다면, W (x; 0) = x가되도록 워프를 다시 매개 변수화 할 수 있습니다.그러므로 우리는 동일한 워프가 이 세트에서 사용되고 있다고 생각할 수 있습니다.
W (x; 0) = x 라고 가정하면, 식 (17)은 다음과 같이 단순화됩니다.

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

식 (19)과 식 (6)의 2 가지 차이점이 Lucas-Kanade 유추법의 등가 방정식이다. 첫 번째 차이점은 W (x; p)에서 계산 된 I (x)의 기울기가 I (W)의 기울기로 대체된다는 것입니다. 이러한 기울기는 다릅니다.두 번째 차이점은 간결한 표기법으로 숨겨져 있습니다. Jacobian $\frac{\partial \mathbf{W}}{\partial \mathbf{p}}$는 식 (6)에서 (x; p)로 평가되지만, 식 (19)에서 (x; 0)에서 평가된다. 왜냐하면 그것이 테일러 확장이 수행 된 곳이기 때문입니다 그러므로 자 코비안은 상수이고 미리 계산 될 수 있습니다. 일반적으로 분석적으로 더 간단합니다 (Shum and Szeliski, 2000).

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. Shum and Szeliski (2000)에서 사용 된 합성 알고리즘은 Lucas-Kanade 알고리즘과 유사하다. 유일한 차이점은 다음과 같다 : (1) I (W (x; p))의 기울기는 W (x; p)에서 평가 된 기울기보다는 3 단계에서 사용된다. (2) Jacobian can (x; 0)에서 평가되고 상수이기 때문에 사전 계산되어야하고, (3) 단계 9에서 증분 워핑 (x; ▵p)을 현재 추정치로 구성하여 워프를 갱신해야한다.

따라서 구성 알고리즘은 거의 사실상 가산 알고리즘 (Lucas Kanade 알고리즘)과 동일합니다. 단 3 가지 차이점은 다음과 같습니다.(1) 단계 3에서 I의 기울기 (W (x; p))를 사용해야하며,(2)(2) 야 코비 행렬은 단계 4에서 각 반복에서 재 계산되기보다는 (x; 0)에서 평가되고, (3) 단계 9에서 워프가
W (x; p) ← W x, p) ◦ W (x ; ▵ p)
로 업데이트되기 때문에 사전 계산 될 수 있습니다. 합성 알고리즘은 그림 3에 요약되어있습니다.

3.1.3.Requirements on the Set of Warps.

루카스 - 카나 데 (Lucas-Kanade) 알고리즘 에서처럼 파라미터 p의 현재 추정치에 단순히 첨자 업데이트 ▵p를 가산하는 대신, 워프 W (x; ▵p)에 대한 증분식 업데이트는 워프 W (x; p). 이 연산은 전형적으로 아핀 워프에 대해 식 (16)에서와 같이 구성된 워프의 파라미터를 계산하기 위해 2 개의 행렬을 곱하는 것을 포함한다. 더 복잡한 워프의 경우 두 가지 워프의 구성이 더 복잡 할 수 있습니다 (Baker and Matthews, 2001). 그러므로 우리는 워프 세트에 두 가지 요구 사항을 가지고있다 :
(1) 워프 세트는 식별 워프를 포함해야하고 (2) 워프 세트는 합성 하에서 닫혀야만한다. 워프 세트는 반쪽 그룹을 형성해야합니다.
이 요구 사항은 그다지 복잡하지 않습니다. 동역학 및 3D 회전 (Shum and Szeliski, 2000)을 포함하여 컴퓨터 비전에 사용되는 대부분의 휘어짐은 자연적으로 반 그룹을 형성합니다.

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

Pre-computation								
Step 4	Total							
$O(n\,N)$	$O(n\,N)$							
Per-iteration								
Step 1	Step 2	Step 3	Step 5	Step 6	Step 7	Step 8	Step 9	Total
$O(nN)$	$O(N)$	$O(N)$	$O(nN)$	$O(n^2N)$	$O(nN)$	$O(n^3)$	$O(n^2)$	$O(n^2N + n^3)$

3.1.4. Computational Cost of the Compositional Algorithm.

합성 알고리즘의 계산 비용은 Lucas-Kanade 알고리즘의 계산 비용과 거의 동일합니다. 자세한 내용은 표 2를 참조하고 Baker와 Matthhews (2002)를 참조하십시오. 변경되는 유일한 단계는 3,4 단계 및 9 단계입니다. 가장 중요한 점은 9 단계에서 2 개의 워프를 작성하는 비용은 W 대부분의 워프에 대해 비용은 O (n²) 이 하 이 며, 식 (16) 의 아핀워프를 포함한다 .

3.1.5. Equivalence of the Additive and Compositional Algorithms.

우리는 이제 첨부 접근법과 합성 접근법이 ∆p의 1 차 근사법에 대해 각 반복에서 동일한 단계를 취한다는 의미에서 동등하다는 것을 보여줍니다. 즉, 워프에 대한 업데이트는 거의 동일하다. 식 (6)의 첨부 공식에서 우리는 다음을 최소화한다.

$$\sum_{\mathbf{x}} \left[I(\mathbf{W}(\mathbf{x};\mathbf{p})) + \nabla I \frac{\partial \mathbf{W}}{\partial \mathbf{p}} \Delta\mathbf{p} - T(\mathbf{x}) \right]^2 \qquad (20)$$

$$\mathbf{W}(\mathbf{x};\mathbf{p}) \leftarrow \mathbf{W}(\mathbf{x};\mathbf{p} + \Delta\mathbf{p}) \approx \mathbf{W}(\mathbf{x};\mathbf{p}) + \frac{\partial \mathbf{W}}{\partial \mathbf{p}} \Delta\mathbf{p} \qquad (21)$$

테일러 확장이 이루어진 후에 식 (19)의 조성식에서 우리는 다음을 최소화한다.

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

식 (18)을 사용하면 다음과 같이 단순화된다.

$$\sum_{\mathbf{x}} \left[I(\mathbf{W}(\mathbf{x};\mathbf{p})) + \nabla I \frac{\partial \mathbf{W}}{\partial \mathbf{x}} \frac{\partial \mathbf{W}}{\partial \mathbf{p}} \Delta\mathbf{p} - T(\mathbf{x}) \right]^2. \quad (23)$$

구성 접근법에서 워프에 대한 업데이트는 W (x; p) ← W (x; p) ◦ W (x; ∆p)입니다. 이 표현을 단순화하기 위해 다음 사항에 유의하십시오.

$$\mathbf{W}(\mathbf{x};\Delta\mathbf{p}) \;\approx\; \mathbf{W}(\mathbf{x};\mathbf{0}) + \frac{\partial \mathbf{W}}{\partial \mathbf{p}} \Delta\mathbf{p} \;=\; \mathbf{x} + \frac{\partial \mathbf{W}}{\partial \mathbf{p}} \Delta\mathbf{p} \qquad (24)$$

는 W(x; ∆p)의 1 차 테일러 확장이며 다음과 같습니다.

$$\mathbf{W}(\mathbf{x};\mathbf{p}) \circ \mathbf{W}(\mathbf{x};\Delta\mathbf{p}) \;=\; \mathbf{W}(\mathbf{W}(\mathbf{x};\Delta\mathbf{p});\mathbf{p}). \qquad (25)$$

이 마지막 두 방정식을 결합하고 테일러 확장을 다시 적용하면 다음과 같이 구성 공식에 업데이트가 제공됩니다.

$$\mathbf{W}(\mathbf{x};\mathbf{p}) \;\leftarrow\; \mathbf{W}(\mathbf{x};\mathbf{p}) + \frac{\partial \mathbf{W}}{\partial \mathbf{x}} \frac{\partial \mathbf{W}}{\partial \mathbf{p}} \Delta\mathbf{p} \qquad (26)$$

식 (20)과 (21)의 첨가제와 식 (23)과 (26)의 조성식의 차이는 $\frac{\partial \mathbf{W}}{\partial \mathbf{p}}$ 가 $\frac{\partial \mathbf{W}}{\partial \mathbf{x}} \frac{\partial \mathbf{W}}{\partial \mathbf{p}}$ 로 대체된다는 것이다. 따라서 식 (20)과 식 (22)는 일반적으로 p에 대한 다른 평가를 유도한다.이 식의 두 번째 식에서 $\frac{\partial \mathbf{W}}{\partial \mathbf{p}}$ 는 첫 번째 식의 (x; p)가 아닌 (x; 0) .)

그러나 첨가제 제형의 벡터 와 구성 공식의 $\frac{\partial \mathbf{W}}{\partial \mathbf{x}} \frac{\partial \mathbf{W}}{\partial \mathbf{p}}$가 모두 동일한 선형 공간에 걸쳐있는 경우, Eqs.(21)과 (26)은 ∆p에서 첫 번째 순서와 동일 할 것이며 두 공식은 ∆p에서 첫 번째 순서와 같습니다. 즉, 식 (20)의 $\frac{\partial \mathbf{W}}{\partial \mathbf{p}}$ ∆p의 최적 값은 식 (22)의 $\frac{\partial \mathbf{W}}{\partial \mathbf{x}} \frac{\partial \mathbf{W}}{\partial \mathbf{p}}$ ∆p의 최적 값과 대략 동일 할 것입니다. 식 (21)로부터 우리는 다음 식 중 첫 번째를 볼 수 있습니다.

$$\frac{\partial \mathbf{W}}{\partial \mathbf{p}} \;=\; \frac{\partial \mathbf{W}(\mathbf{x};\mathbf{p} + \Delta\mathbf{p})}{\partial \Delta\mathbf{p}} \qquad (27)$$

식 (26)으로부터 우리는 두 번째 표현식을 볼 수있다.

$$\frac{\partial \mathbf{W}}{\partial \mathbf{x}} \frac{\partial \mathbf{W}}{\partial \mathbf{p}} = \frac{\partial \mathbf{W}(\mathbf{x};\mathbf{p}) \circ \mathbf{W}(\mathbf{x};\mathbf{p}+\Delta\mathbf{p})}{\partial \Delta \mathbf{p}}. \quad (28)$$

따라서 첨가제의 벡터 $\frac{\partial \mathbf{W}}{\partial \mathbf{p}}$와 조성식의 $\frac{\partial \mathbf{W}}{\partial \mathbf{p}} \frac{\partial \mathbf{W}}{\partial \mathbf{x}}$는 동일한 선형 공간, 매니 폴드 W (x; p)의 탄젠트 공간을 확장하며, Δp에 대해 ($\in$ > 0 인 경우) ΔP ' :(번역이 거지같이되넹..맞는지모르겟음)

$$\mathbf{W}(\mathbf{x};\mathbf{p}+\Delta\mathbf{p}) = \mathbf{W}(\mathbf{x};\mathbf{p}) \circ \mathbf{W}(\mathbf{x};\mathbf{p}+\Delta\mathbf{p}'). \quad (29)$$

이 조건은 Δp와 Δp' 사이의 함수가 양방향으로 정의됨을 의미합니다. 따라서 식 (27)과 식 (28)의 식은 동일한 선형 공간에 걸쳐있다.휨이 뒤집을 수 있다면 Eq. Δp'는 다음과 같이 선택 될 수 있기 때문에 항상 유지된다.

$$\mathbf{W}(\mathbf{x};\mathbf{p}+\Delta\mathbf{p}') = \mathbf{W}(\mathbf{x};\mathbf{p})^{-1} \circ \mathbf{W}(\mathbf{x};\mathbf{p}+\Delta\mathbf{p}). \quad (30)$$

요약하면, 워프가 반전될 수 있다면 두 공식은 동등합니다. 위 3.1.3 절에서 우리는 구성 알고리즘이 적용될 수 있도록 일련의 워프가 반 그룹을 형성해야한다고 설명했습니다. 이것이 사실이지만, 합성 알고리즘이 Lucas-Kanade 알고리즘과 동등한 것으로 입증되기 위해서는 일련의 워프가 그룹을 형성해야합니다. 즉, 모든 워프는 반전 가능해야한다.

3.2. Inverse Compositional Image Alignment

많은 저자들이 지적했듯이 루카스 - 카나데 (Lucas-Kanade) 알고리즘의 모든 반복에서 헤시안을 반복(재계산) 할 때 계산 비용이 많이 든다. Belhumeur, 1998; Dellaert and Collins, 1999; Shum and Szeliski, 2000). 헤시안이 일정하다면 사전 계산되어 다시 사용될 수 있습니다.
알고리즘의 각 반복 (그림 1 참조)은 이미지 왜곡 (1 단계), 이미지 차이 (2 단계), 이미지 "점 유형"수집 (7 단계), 헤시안의 결과 (8 단계), 매개 변수의 업데이트 (9 단계). 이러한 모든 작업은 프레임 속도 (근접)로 수행 될 수 있습니다 (Dellaert and Collins, 1999).

불행하게도 헤센은 두 공식 모두에서 p의 함수이다. 여러 가지 근사해를 사용할 수 있지만 (예 : 몇 번의 반복마다 헤 시안을 업데이트하고 헤 시안을 이미지에 대해 거의 일정하다고 가정하여 헤 시안을 근사하는 것 (Shum and Szeliski, 2000)) 이러한 근사값은 우아하지 않으며 그들이 얼마나 정확한 근사치인지 말할 수없는 경우가 종종 있습니다.문제가 같은 방식으로 재구성 될 수 있지만 일정한 헤시안 (Hessian)이 있다면 훨씬 더 좋을 것입니다.

3.2.1. Goal of the Inverse Compositional Algorithm.

효율성의 핵심은 Hager and Belhumeur (1998) 에서처럼 템플릿과 이미지의 역할을 전환하거나 반전시키기 위해 변수를 변경하는 것과 같이 이미지와 템플릿의 역할을 전환하는 것입니다. 이러한 변수의 변화는 Hager와 Belhumeur의 첨가물 (1998)이나 작곡 기법 (Baker and Matthews, 2001)에서 수행 될 수있다.
(역 합성 알고리즘의 제한된 버전이 Dellaert and Collins (1999)의 동질성에 대해 제안되었으며 Gleicher (1997)에서 inverse com? 위치 알고리즘과 동일한 것이 사용되었을 수도있다.
La Cascia et al.의 "difference decompoitionation"알고리즘 (2000)은 덧셈 접근법을 사용한다.) 우리는 먼저 역 구성 접근법을 보다 간단하기 때문에 설명한다.이전 알고리즘과 새로운 알고리즘을 구별하기 위해 원래의 알고리즘을 전달이라고 합니다. 첨부물(?) (즉, 루카스 - 카나 데 (Lucas-Kanade)) 및 포워드 컴포지셔널 (Forward compostitional) 알고리즘을 포함한다.역전 후의 해당 알고리즘을 역 첨가 및 합성 알고리즘으로 부릅니다.

순방향 및 역 구성 알고리즘 간의 동등성 증명은 3.2.5 절에 있습니다. 결과적으로 역 합성 알고리즘은 다음을 최소화합니다.

$$\sum_{\mathbf{x}} [\, T(\mathbf{W}(\mathbf{x};\Delta\mathbf{p})) - I(\mathbf{W}(\mathbf{x};\mathbf{p}))\,]^2 \quad (31)$$

Δp와 관련하여 (I와 T의 역할이 뒤 바뀌 었음을 주목하십시오)워프를 업데이트합니다 :

$$\mathbf{W}(\mathbf{x};\mathbf{p}) \leftarrow \mathbf{W}(\mathbf{x};\mathbf{p}) \circ \mathbf{W}(\mathbf{x};\Delta\mathbf{p})^{-1}. \quad (32)$$

식 (13)의 전방 합성 알고리즘에서의 업데이트와의 유일한 차이점은 증분 워핑 (x; Δp)이 현재 추정치와 합성되기 전에 반전된다는 것이다.
예를 들어, 식 (2)에서의 아핀 워프의 역수의 파라미터는 다음과 같다 :

$$\frac{1}{(1+p_1)\cdot(1+p_4)-p_2\cdot p_3} \times \begin{pmatrix} -p_1-p_1\cdot p_4+p_2\cdot p_3 \\ -p_2 \\ -p_3 \\ -p_4-p_1\cdot p_4+p_2\cdot p_3 \\ -p_5-p_4\cdot p_5+p_3\cdot p_6 \\ -p_6-p_1\cdot p_6+p_2\cdot p_5 \end{pmatrix}. \quad (33)$$

(1+ p1) · (1+ p4)− p2 · p3 = 0이면, 아핀 왜곡은 퇴보하고 역환이 불가능하다. 모든 픽셀은 I에서 직선으로 매핑됩니다. 우리는 이러한 모든 아핀 경사를 고려에서 제외합니다. 그런 다음 모든 affine warps의 집합은 식 (16)의 매개 변수에 대해 (1 + p1) · (1 + p4) − p2 · p3 를 계산하여 알 수 있듯이 여전히 합성 상태로 닫힙니다. 상당한 단순화 이후,이 값은 [(1 + p1) · (1 + Δp4) − p2 · p3] · [(1 + Δp1) · (1 + Δp4) − Δp2 · Δp3]가되며, 구성된 두 개의 워프 중 하나가 퇴화 된 경우에만 0이 될 수 있습니다.

3.2.2. Derivation of the Inverse Compositional Algorithm.

식 (31)에 대한 1 차 테일러 확장을 수행하면

$$\sum_{\mathbf{x}}\left[T(\mathbf{W}(\mathbf{x};\mathbf{0}))+\nabla T\frac{\partial \mathbf{W}}{\partial \mathbf{p}}\Delta\mathbf{p}-I(\mathbf{W}(\mathbf{x};\mathbf{p}))\right]^2. \quad (34)$$

W (x; 0)가 정체성 워프라는 일반성을 잃지 않고 다시 가정하면,이 최소 제곱 문제에 대한 해답은 다음과 같다.

$$\Delta\mathbf{p}=H^{-1}\sum_{\mathbf{x}}\left[\nabla T\frac{\partial \mathbf{W}}{\partial \mathbf{p}}\right]^{\mathrm{T}}[I(\mathbf{W}(\mathbf{x};\mathbf{p}))-T(\mathbf{x})] \quad (35)$$

여기서 H는 T로 대체 된 헤 시안 행렬입니다.

$$H=\sum_{\mathbf{x}}\left[\nabla T\frac{\partial \mathbf{W}}{\partial \mathbf{p}}\right]^{\mathrm{T}}\left[\nabla T\frac{\partial \mathbf{W}}{\partial \mathbf{p}}\right] \quad (36)$$

Jacobian $\frac{\partial \mathbf{W}}{\partial \mathbf{p}}$는 (x; 0)에서 평가됩니다.
p에 의존하는 헤 시안에는 아무 것도 없으므로 반복을 통해 일정하며 미리 계산 될 수 있습니다.

The Inverse Compositional Algorithm

Pre-compute:

(3) Evaluate the gradient ∇T of the template $T(\mathbf{x})$
(4) Evaluate the Jacobian $\frac{\partial \mathbf{W}}{\partial \mathbf{p}}$ at $(\mathbf{x};\mathbf{0})$
(5) Compute the steepest descent images $\nabla T\frac{\partial \mathbf{W}}{\partial \mathbf{p}}$
(6) Compute the Hessian matrix using Equation (36)

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 (35)
(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 $\|\Delta\mathbf{p}\|\le\epsilon$

Figure.4 역 구성 알고리즘 (Baker and Matthews, 2001)은 Hager and Belhumeur (1998)의 접근법과 유사하게 I와 T의 역할을 반전시킴으로써 전진 조성 알고리즘에서 유도된다. 계산적으로 요구되는 모든 단계는 사전 계산 단계에서 한 번 수행됩니다. 주요 알고리듬은 단순히 이미지 워핑 (1 단계), 이미지 차이 (2 단계), 이미지 점 제품 (7 단계), 헤시안의 역수 곱하기 (8 단계) 및 워프 업데이트 (9 단계)로 구성됩니다.이 모든 단계는 효율적인 O (n N + n³)입니다.

따라서 그림 3의 전방 합성 알고리즘의 3-6 단계는 반복 당 1 회가 아닌 사전 계산으로 한 번만 수행하면됩니다. 단 하나의 차이점은 순방향 및 역합성 알고리즘 (그림 3 및 4 참조)은 다음과 같습니다. (1) I 및 T의 역할을 전환 한 후에 오류 이미지가 작성됩니다. (2) 3, 5 및 3 단계 6은 I의 기울기보다는 T의 기울기를 사용하고, 미리 계산 될 수있다. (3) 식 (35)는 식 (10) 대신에 Δp를 계산하는데 사용되며, 마지막으로 증분 워프는 단계 9에서 현재 추정치와 합성되기 전에 반전됩니다.

3.2.3.Requirements on the Set of Warps.

순방향 합성 알고리즘의 세미 그룹 요구 사항 외에, 역 합성 알고리즘은 또한 증분 워핑 W (x; Δp)가 현재 추정치와 합성되기 전에 반전되는 것을 요구한다. 따라서 역 구성 알고리즘은 그룹을 형성하는 일련의 경사에만 적용 할 수 있습니다. 다행히도 동질 및 3D 회전 (Shum and Szeliski, 2000)을 포함하여 컴퓨터 비전에 사용되는 대부분의 날실은 양식 그룹을 수행합니다.
주목할만한 예외 중 하나는 Active Appearance Models에 사용되는 piece-wise affine warps입니다

Table 3 역합성 알고리즘의 계산 비용. 단계 3 ~ 6에서 가장 가파른 하강 영상과 헤 시안을 계산하는 사전 계산 비용은 O (n² N)입니다. 그 후에, 각 반복의 비용은 O (nN + n²)로 Lucas-Kanade 및 합성 알고리즘보다 상당히 절약됩니다.

Pre-computation					
Step 3	Step 4	Step 5	Step 6	Total	
$O(N)$	$O(n\,N)$	$O(n\,N)$	$O(n^2\,N)$	$O(n^2\,N)$	
Per iteration					
Step 1	Step 2	Step 7	Step 8	Step 9	Total
$O(n\,N)$	$O(N)$	$O(n\,N)$	$O(n^3)$	$O(n^2)$	$O(n\,N + n^3)$

(Cootes et al., 1998), Active Blobs (Sclaroff and Isidoro, 1998), FLE (Flexible Appearance Models) (Baker and Matthews, 2001)를 포함한다. Baker and Matthews (2001)에서 우리는 역합성 알고리즘을 piecewise affine warps로 확장하는 방법을 보여 주었다. 유사한 확장법이 다른 그룹이 아닌 워프 세트에도 적용될 수 있습니다.

3.2.4. Computational Cost of the Inverse Compositional Algorithm.

역 합성 알고리즘은 다음 중 하나보다 훨씬 더 계산상 효율적입니다. 루카스 - 카나데 (Lucas-Kanade) 알고리즘 또는 합성적 알고리즘 일 수있다.
요약은 표 3을, 자세한 내용은 (Baker and Matthews, 2002)를 참조하십시오. 가장 시간이 많이 소요되는 단계 인 단계 6의 헤시안 계산은 사전 계산으로 한 번 수행 할 수 있습니다. (이 사전 계산은 헤시안 및 가장 가파른 하강 영상을 저장해야 함을 의미합니다. 일반적인 문제는 n = 10, N ≈ 100 × 100으로 급하강 영상이 프로세서 캐시에 쉽게 들어 맞습니다하강 영상은 순차적으로 처리되므로 매우 큰 문제 일지라도 메모리 문제로 인해 계산이 거의 느려지지 않습니다.) 유일한 추가 비용은 W(x; Δp)를 반전하고 W (x; p)로 구성하는 것입니다. 이 두 단계는 전형적으로 시간 O (n²)를 필요로하는데, Eqs.(15) 및 (33)을 만족한다. 잠재적으로이 두 단계는 Baker와 Matthews (2001) 에서처럼 상당히 복잡할 수 있지만 추가 계산은 거의 무시할 수 있습니다.

3.2.5.Equivalence of the Forwards and Inverse Compositional Algorithms.

이제 역합성 알고리즘이 포워드와 동일하다는 것을 보여줍니다. 합성 알고리즘은 3.1 절에서 소개했다. 전방 합성 알고리즘은 3.1.5 절의 Lucas-Kanade 알고리즘과 동등한 것으로 나타 났으므로 역합성 알고리즘이 이에 해당합니다. 등가성 증명은 3.1.5 절의 증명과 매우 다른 형태를 취합니다. 첫 번째 단계는 (12)절과 (31)절은 적분에 대한 이산 근사치입니다. 방정식 (12)는 다음의 이산 버전입니다.

$$\int_T [\, I(\mathbf{W}(\mathbf{W}(\mathbf{x};\Delta\mathbf{p});\mathbf{p})) - T(\mathbf{x})\,]^2\ \mathrm{d}\mathbf{x} \qquad (37)$$

통합은 온도 판 (T)상에서 수행된다. 등가 적으로 x = W (y; Δp)⁻¹로 설정하고 변수를 변경하면 식 (37)은 다음과 같이 된다.

$$\int_{\mathbf{W}(T)} [I(\mathbf{W}(\mathbf{y};\mathbf{p})) - T(\mathbf{W}(\mathbf{y};\Delta\mathbf{p})^{-1})]^2 \left|\frac{\partial \mathbf{W}^{-1}}{\partial \mathbf{y}}\right| \mathrm{d}\mathbf{y} \qquad (38)$$

여기서 통합은 워프 W (x; Δp) 하에서 T의 시점에 대해 수행되며, 여기서 W (T) = {W (x; Δp) | x ∈ T}이다.
W (x; 0)는 항등 워프이므로 다음과 같습니다.

$$\left|\frac{\partial \mathbf{W}^{-1}}{\partial \mathbf{y}}\right| = 1 + O(\Delta\mathbf{p}). \qquad (39)$$

통합 도메인 W (T)는 T = {W (x; 0) | x ∈ T}를 0 차 근사로 변환한다. 우리가 Δp에서 고차항을 무시하기 때문에, 식 (38)는 다음과 같이 단순화한다.

$$\int_T [T(\mathbf{W}(\mathbf{x};\Delta\mathbf{p})^{-1}) - I(\mathbf{W}(\mathbf{x};\mathbf{p}))]^2\,\mathrm{d}\mathbf{x}. \qquad (40)$$

이 단순화를 만들 때 T (W (y; Δp)⁻¹) - I (W (y; p)) 또는 이와 동등하게T (y) − I(W(y; p)), 는 O(Δp). (이 가정은 Hager and Belhumeur (1998)에서 매개 변수의 현재 추정치가 대략 정확하다는 가정과 동일하다.) 따라서 야 코비 행렬의 첫 번째 항과 통합 영역은 무시 될 수있다. 식 (40)은 항 W (x; Δρ)가 반전된다는 점을 제외하고 식 (31)의 연속 버전이다. 식 (1)을 사용하여 역 합성 알고리즘에 의해 계산 된 ρ의 추정치. 따라서 식 (31)은 식 (33)을 사용하여 구성 알고리즘에 의해 계산 된 in_extension_warp의 역수인 W (x; Δρ)의 추정치를 제공한다. (12). 역 합성 알고리즘은 단계 9에서 W (x; Δp)와 합성하기 전에 W (x; ρ)를 반전하므로 두 알고리즘은 Δp에서 첫 번째 단계까지 동일한 단계를 취한다.

3.3. Inverse Additive Image Alignment

A natural question which arises at this point is whether the same trick of changing variables to convert Eq. (37) into Eq. (38) 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 Eq. (39) 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 (Hager and Belhumeur, 1998). Although the derivation in Hager and Belhumeur (1998) 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 $\sum_{\mathbf{x}} [\, I(\mathbf{W}(\mathbf{x}; \mathbf{p} + \Delta\mathbf{p})) - T(\mathbf{x}) \,]^2$ with respect to $\Delta\mathbf{p}$ and then update the parameters $\mathbf{p} \leftarrow \mathbf{p} + \Delta\mathbf{p}$. 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:

$$\sum_{\mathbf{x}} \left[I(\mathbf{W}(\mathbf{x}; \mathbf{p})) + \nabla I \frac{\partial \mathbf{W}}{\partial \mathbf{p}} \Delta\mathbf{p} - T(\mathbf{x}) \right]^2. \quad (41)$$

The template and the image are then switched by deriving the relationship between ∇I and ∇T. In Hager and Belhumeur (1998) it is assumed that the current estimates of the parameters are approximately correct: i.e.

$$I(\mathbf{W}(\mathbf{x}; \mathbf{p})) \approx T(\mathbf{x}) \quad (42)$$

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:

$$\nabla I \frac{\partial \mathbf{W}}{\partial \mathbf{x}} \approx \nabla T. \quad (43)$$

Inverting $\frac{\partial \mathbf{W}}{\partial \mathbf{x}}$ and substituting Eq. (43) into Eq. (41) gives:

$$\sum_{\mathbf{x}} \left[I(\mathbf{W}(\mathbf{x}; \mathbf{p})) + \nabla T \left(\frac{\partial \mathbf{W}}{\partial \mathbf{x}} \right)^{-1} \frac{\partial \mathbf{W}}{\partial \mathbf{p}} \Delta\mathbf{p} - T(\mathbf{x}) \right]^2. \quad (44)$$

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:

$$\sum_{\mathbf{x}} \left[T(\mathbf{x}) + \nabla T \left(\frac{\partial \mathbf{W}}{\partial \mathbf{x}} \right)^{-1} \frac{\partial \mathbf{W}}{\partial \mathbf{p}} \Delta\mathbf{p} - I(\mathbf{W}(\mathbf{x}; \mathbf{p})) \right]^2. \quad (45)$$

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 Eq. (45) in terms of the Hessian, just like in Section 2.2. In general, however, the Hessian depends on $\mathbf{p}$ through $(\frac{\partial \mathbf{W}}{\partial \mathbf{x}})^{-1}$ 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:

$$\left(\frac{\partial \mathbf{W}}{\partial \mathbf{x}} \right)^{-1} \frac{\partial \mathbf{W}}{\partial \mathbf{p}} = \Gamma(\mathbf{x}) \Sigma(\mathbf{p}) \quad (46)$$

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 Eq. (2):

$$\begin{aligned} \left(\frac{\partial \mathbf{W}}{\partial \mathbf{x}} \right)^{-1} &= \begin{pmatrix} 1 + p_1 & p_3 \\ p_2 & 1 + p_4 \end{pmatrix}^{-1} \\ &= \frac{1}{(1 + p_1) \cdot (1 + p_4) - p_2 \cdot p_3} \\ &\quad \times \begin{pmatrix} 1 + p_4 & -p_3 \\ -p_2 & 1 + p_1 \end{pmatrix} \end{aligned} \quad (47)$$

and so:

$$\left(\frac{\partial \mathbf{W}}{\partial \mathbf{x}}\right)^{-1}\frac{\partial \mathbf{W}}{\partial \mathbf{p}} = \frac{1}{(1+p_1)\cdot(1+p_4)-p_2\cdot p_3} \times \begin{pmatrix} 1+p_4 & -p_3 \\ -p_2 & 1+p_1 \end{pmatrix}\begin{pmatrix} x & 0 & y & 0 & 1 & 0 \\ 0 & x & 0 & y & 0 & 1 \end{pmatrix}. \tag{48}$$

Since diagonal matrices commute with any other matrix, and since the 2×6 matrix $\frac{\partial \mathbf{W}}{\partial \mathbf{p}}$ can be thought of 3 blocks of 2×2 matrices, the expression in Eq. (48) can be re-written as:

$$\frac{1}{\det}\begin{pmatrix} x & 0 & y & 0 & 1 & 0 \\ 0 & x & 0 & y & 0 & 1 \end{pmatrix} \times \begin{pmatrix} 1+p_4 & -p_3 & 0 & 0 & 0 & 0 \\ -p_2 & 1+p_1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1+p_4 & -p_3 & 0 & 0 \\ 0 & 0 & -p_2 & 1+p_3 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1+p_4 & -p_3 \\ 0 & 0 & 0 & 0 & -p_2 & 1+p_3 \end{pmatrix} \tag{49}$$

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 Eq. (46). Equation (45) can then be re-written as:

$$\sum_{\mathbf{x}} \left[T(\mathbf{x}) + \nabla T\, \Gamma(\mathbf{x})\Sigma(\mathbf{p})\Delta\mathbf{p} - I(\mathbf{W}(\mathbf{x};\mathbf{p}))\right]^2. \tag{50}$$

Equation (50) has the closed form solution:

$$\Delta\mathbf{p} = H^{-1}\sum_{\mathbf{x}} \left[\nabla T\, \Gamma(\mathbf{x})\Sigma(\mathbf{p})\right]^{\mathrm{T}} \left[I(\mathbf{W}(\mathbf{x};\mathbf{p})) - T(\mathbf{x})\right] \tag{51}$$

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

$$H = \sum_{\mathbf{x}} [\nabla T\, \Gamma(\mathbf{x})\Sigma(\mathbf{p})]^{\mathrm{T}} [\nabla T \Gamma(\mathbf{x})\Sigma(\mathbf{p})]. \tag{52}$$

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

$$H = \Sigma(\mathbf{p})^{\mathrm{T}}\left(\sum_{\mathbf{x}} \left[\nabla T\, \Gamma(\mathbf{x})\right]^{\mathrm{T}} \left[\nabla T\, \Gamma(\mathbf{x})\right]\right)\Sigma(\mathbf{p}) \tag{53}$$

Denoting:

$$H_* = \sum_{\mathbf{x}} \left[\nabla T\, \Gamma(\mathbf{x})\right]^{\mathrm{T}} \left[\nabla T\, \Gamma(\mathbf{x})\right] \tag{54}$$

and assuming that $\Sigma(\mathbf{p})$ is invertible (Hager and Belhumeur actually consider a slightly more general case. The interested reader is referred to Hager and Belhumeur (1998) for more details.), we have:

$$H^{-1} = \Sigma(\mathbf{p})^{-1} H_*^{-1} \Sigma(\mathbf{p})^{-\mathrm{T}}. \tag{55}$$

Inserting this expression into Eq. (51) and simplifying yields:

$$\Delta\mathbf{p} = \Sigma(\mathbf{p})^{-1} H_*^{-1} \times \sum_{\mathbf{x}} \left[\nabla T\, \Gamma(\mathbf{x})\right]^{\mathrm{T}} \left[I(\mathbf{W}(\mathbf{x};\mathbf{p})) - T(\mathbf{x})\right]. \tag{56}$$

Equation (56) can be split into two steps:

$$\Delta\mathbf{p}_* = H_*^{-1} \sum_{\mathbf{x}} \left[\nabla T\, \Gamma(\mathbf{x})\right]^{\mathrm{T}} \left[I(\mathbf{W}(\mathbf{x};\mathbf{p})) - T(\mathbf{x})\right] \tag{57}$$

and:

$$\Delta\mathbf{p} = \Sigma(\mathbf{p})^{-1}\Delta\mathbf{p}_* \tag{58}$$

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 Eq. (57) and then updating the parameters $\mathbf{p} \leftarrow \mathbf{p} - \Sigma(\mathbf{p})^{-1}\Delta\mathbf{p}_*$. For the affine warp of Eq. (2):

$$\Sigma(\mathbf{p})^{-1} = \begin{pmatrix} 1+p_1 & p_3 & 0 & 0 & 0 & 0 \\ p_2 & 1+p_4 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1+p_1 & p_3 & 0 & 0 \\ 0 & 0 & p_2 & 1+p_4 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1+p_1 & p_3 \\ 0 & 0 & 0 & 0 & p_2 & 1+p_4 \end{pmatrix}. \tag{59}$$

The Hager-Belhumeur inverse additive algorithm is summarized in Fig. 5. The main differences from the

The Hager-Belhumeur Inverse Additive Algorithm

Pre-compute:

(3) Evaluate the gradient ∇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 (54)

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 (57)
(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 (Hager and Belhumeur, 1998) is very similar to the inverse compositional algorithm in Fig. 4. The two main differences are that Γ 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 Eq. (46) holds.

inverse compositional algorithm in Fig. 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) Eq. (57) 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.

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 Eq. (46) 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 Hager and Belhumeur (1998) 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 Eqs. (47)–(49). 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 (Baker and Matthews 2001).

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 $k = n$.

Precomputation					
Step 3	Step 4	Step 5	Step 6	Total	
$O(N)$	$O(k\,N)$	$O(k\,N)$	$O(k^2\,N)$	$O(k^2\,N)$	
Periteration					
Step 1	Step 2	Step 7	Step 8	Step 9	Total
$O(n\,N)$	$O(N)$	$O(k\,N)$	$O(k^3)$	$O(k^2)$	$O(n\,N + k\,N + k^3)$

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. When $k = n$ the algorithms take almost exactly the same time. See Table 4 for a summary and (Baker and Matthews, 2002) for the details.

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 Hager and Belhumeur (1998) 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 Eq. (2). The Hager-Belhumeur algorithm cannot be applied to homographies and other more interesting sets of warps.

Comparing Eqs. (8) and (49) 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:

$$\mathbf{p} \leftarrow \mathbf{p} - \begin{pmatrix} 1+p_1 & p_3 & 0 & 0 & 0 & 0 \\ p_2 & 1+p_4 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1+p_1 & p_3 & 0 & 0 \\ 0 & 0 & p_2 & 1+p_4 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1+p_1 & p_3 \\ 0 & 0 & 0 & 0 & p_2 & 1+p_4 \end{pmatrix} \Delta\mathbf{p}_* \tag{60}$$

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:

$$\frac{1}{(1+\Delta p_1)\cdot(1+\Delta p_4)-\Delta p_2\cdot\Delta p_3} \times \begin{pmatrix} -\Delta p_1 - \Delta p_1\cdot\Delta p_4 + \Delta p_2\cdot\Delta p_3 \\ -\Delta p_2 \\ -\Delta p_3 \\ -\Delta p_4 - \Delta p_1\cdot\Delta p_4 + \Delta p_2\cdot\Delta p_3 \\ -\Delta p_5 - \Delta p_1\cdot\Delta p_5 + \Delta p_3\cdot\Delta p_6 \\ -\Delta p_6 - \Delta p_1\cdot\Delta p_6 + \Delta p_2\cdot\Delta p_5 \end{pmatrix} = \begin{pmatrix} -\Delta p_1 \\ -\Delta p_2 \\ -\Delta p_3 \\ -\Delta p_4 \\ -\Delta p_5 \\ -\Delta p_6 \end{pmatrix} \tag{61}$$

to first order in $\Delta\mathbf{p}$. Substituting this expression into Eq. (16) 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 Eq. (60). 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 warps, such as homographies and 3D rotations (Shum and Szeliski, 2000), for which the Hager-Belhumeur algorithm cannot be applied, even though the inverse compositional algorithm can be (Baker and Matthews, 2001).

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.

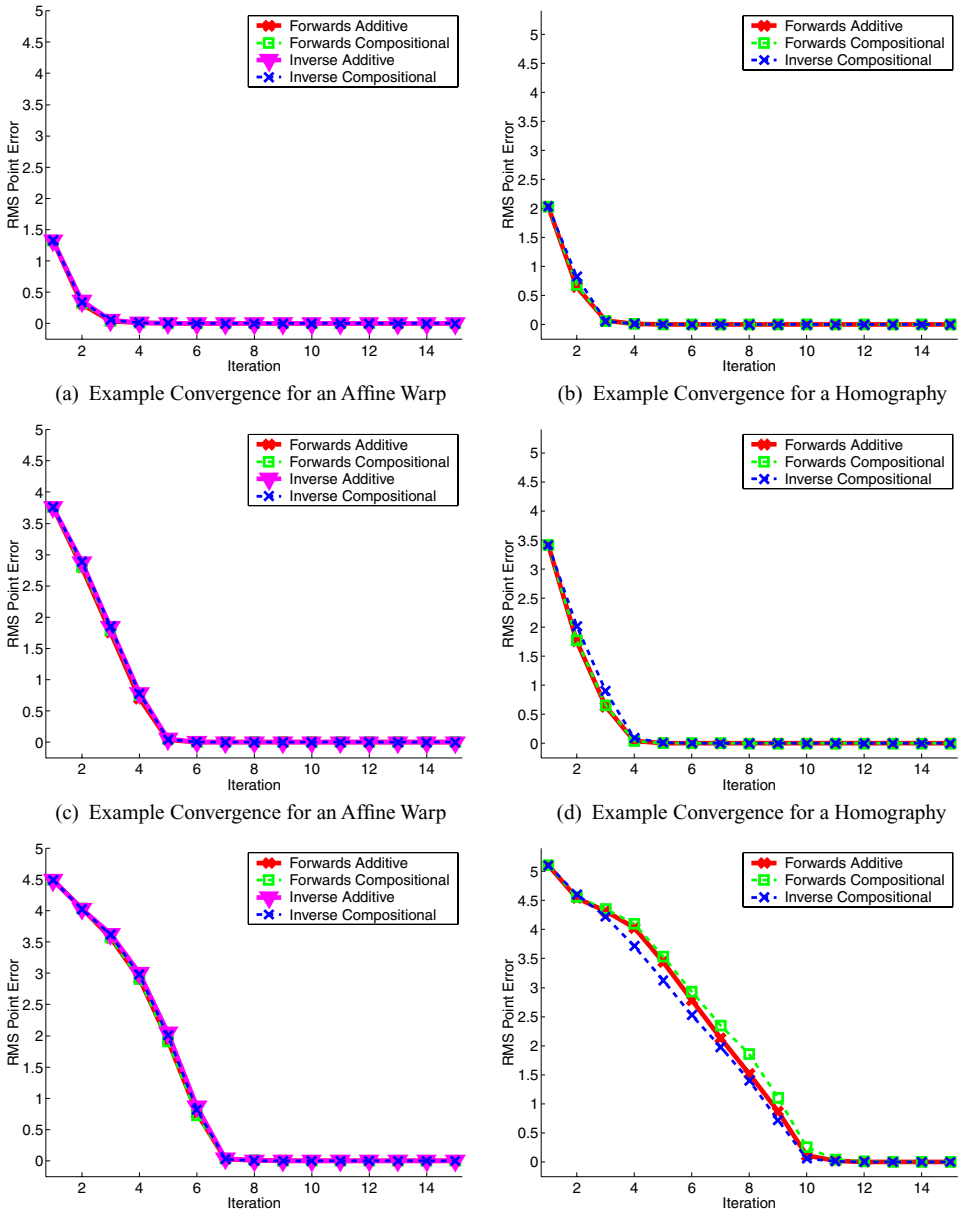
3.4.1. Example Convergence Rates. We experimented with the image $I(\mathbf{x})$ in Fig. 2. We manually selected a 100×100 pixel template $T(\mathbf{x})$ in the center of the face. We then randomly generated affine warps $\mathbf{W}(\mathbf{x};\mathbf{p})$ in the following manner. We selected 3 canonical points in the template. We used the bottom left corner $(0, 0)$, the bottom right corner $(99, 0)$, and the center top pixel $(49, 99)$ as the canonical points. We then randomly perturbed these points with additive white Gaussian noise of a certain variance and fit for the affine warp parameters $\mathbf{p}$ that these 3 perturbed points define. We then warped I with this affine warp $\mathbf{W}(\mathbf{x};\mathbf{p})$ and run the 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 prefer this error measure to normalizing the units so the errors in the 6 parameters are comparable.)

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 Fig. 6 we include examples of the algorithms converging. The RMS error in the canonical point locations is plot against the iteration number. In Fig. 6(a), (c), and (e) we plot results for affine warps. In Fig. 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 Fig. 6(b), (d), and (f). See Appendix A for the derivation of the inverse compositional algorithm for the homography.

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



(a) Example Convergence for an Affine Warp

(b) Example Convergence for a Homography

(c) Example Convergence for an Affine Warp

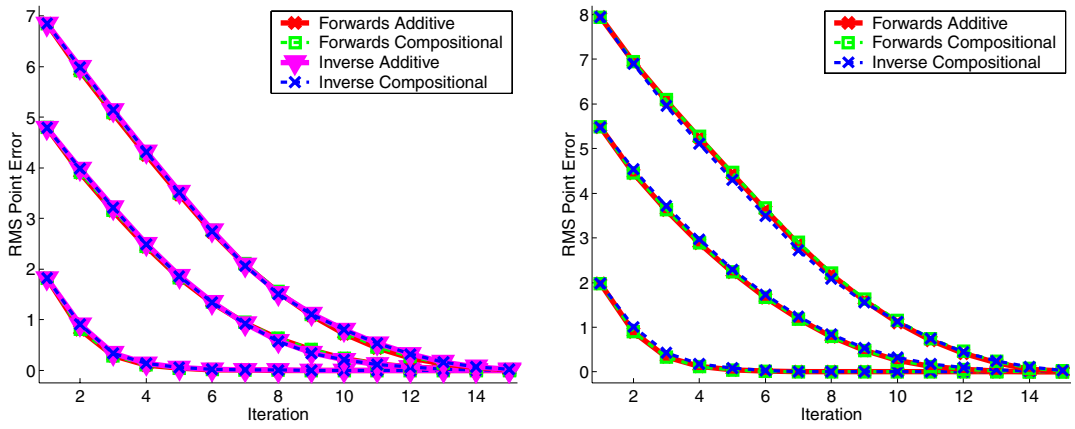
(d) Example Convergence for a Homography

(e) Example Convergence for an Affine Warp

(f) Example Convergence for a Homography

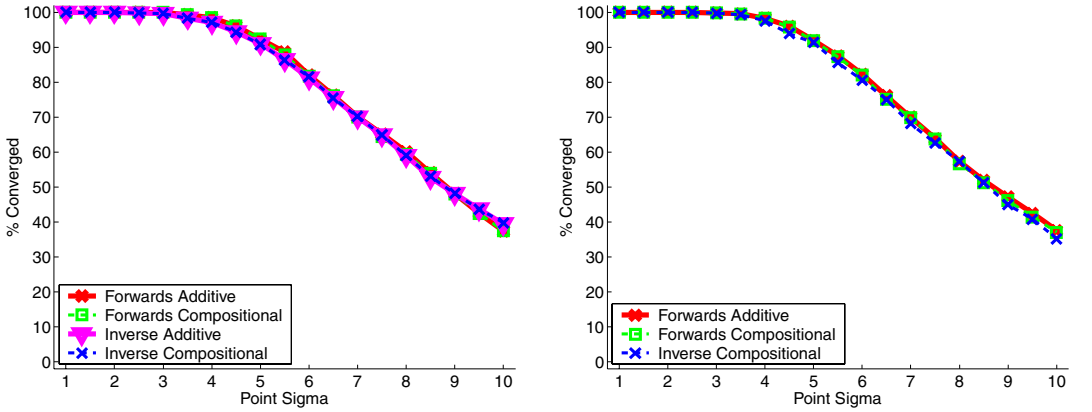
Figure 6. Examples of the four algorithms converging. The RMS error in the location of 3 canonical points in the template (4 for the homographies) is plot against the algorithm iteration number. In all examples the algorithms converge at the same rate validating their equivalence. Only 3 plots are shown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

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 Fig. 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. (Faster convergence can



(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.



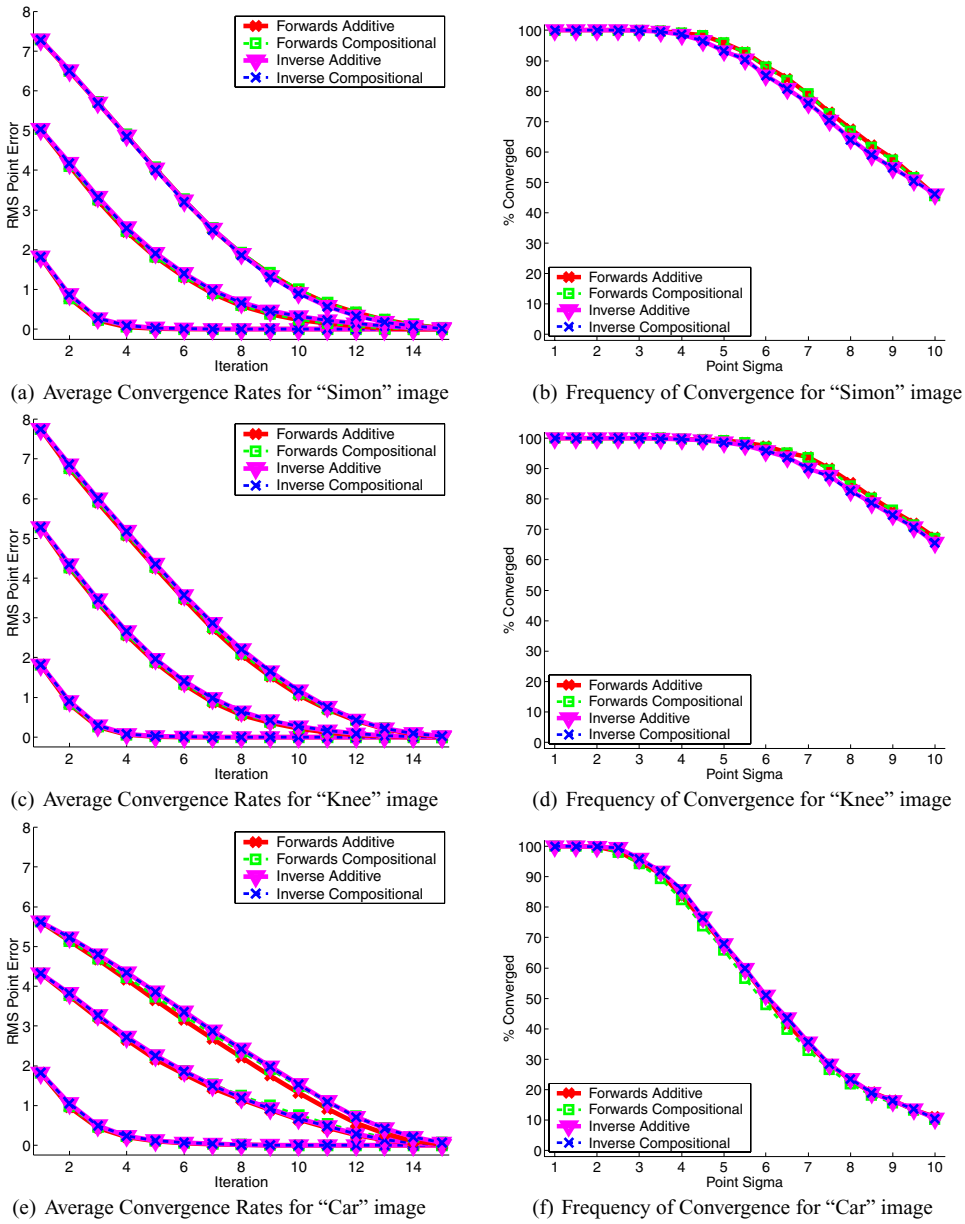
(a) Frequency of Convergence for Affine Warps (b) Frequency of Convergence for Homographies

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.

be obtained by processing hierarchically on a Gaussian pyramid (Bergen et al., 1992.)

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 Fig. 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 multi-scale 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



(a) Average Convergence Rates for "Simon" image
(b) Frequency of Convergence for "Simon" image
(c) Average Convergence Rates for "Knee" image
(d) Frequency of Convergence for "Knee" image
(e) Average Convergence Rates for "Car" image
(f) Frequency of Convergence for "Car" image

Figure 9. The average rates of convergence and average frequencies of convergence for the affine warp for three different images: "Simon" (an image of the face of the first author), "Knee" (an image of an x-ray of a knee), and "Car" (an image of car.) These results demonstrate the same qualitative behavior as those in Figs. 7 and 8. Most importantly, the four algorithms all converge equally fast and diverge equally often. Their equivalence is therefore further validated. However, there is considerable variation in performance with the image. The performance of all four algorithms is far worse on "Car" than on the other images.

effect of using a multi-scale pyramid will be studied in a future paper.

3.4.4. Variability with the Input Image. To investigate the variability of the algorithms with the input image, we re-ran the experiments in Sections 3.4.2 and 3.4.3 for three more images: "Simon" (an image of the face of the first author), "Knee" (an image of an x-ray of a knee), and "Car" (an image of car.) The results are contained in Fig. 9. For lack of space we only

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 × 100 pixel template on a 933 MHz Pentium-IV.

	Step 3	Step 4	Step 5	Step 6	Total
Pre-computation:					
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

	Step 1	Step 2	Step 3	Step 4	Step 5	Step 6	Step 7	Step 8	Step 9	Total
Per iteration:										
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

include the results for the affine warp. The results for the homography are similar and can be reconstructed from the implementation of our algorithms that we are making available online. See Section 6. Qualitatively the results for these three new images are the same as before. Most importantly, the four algorithms all converge equally fast and diverge equally often. Their equivalence is therefore further validated. There is considerable variation in performance across the images, however. In particular, the performance of all four algorithms for the "Car" image is far worse than for the other three images, both in terms of average rate of convergence and in terms of average frequency of divergence.

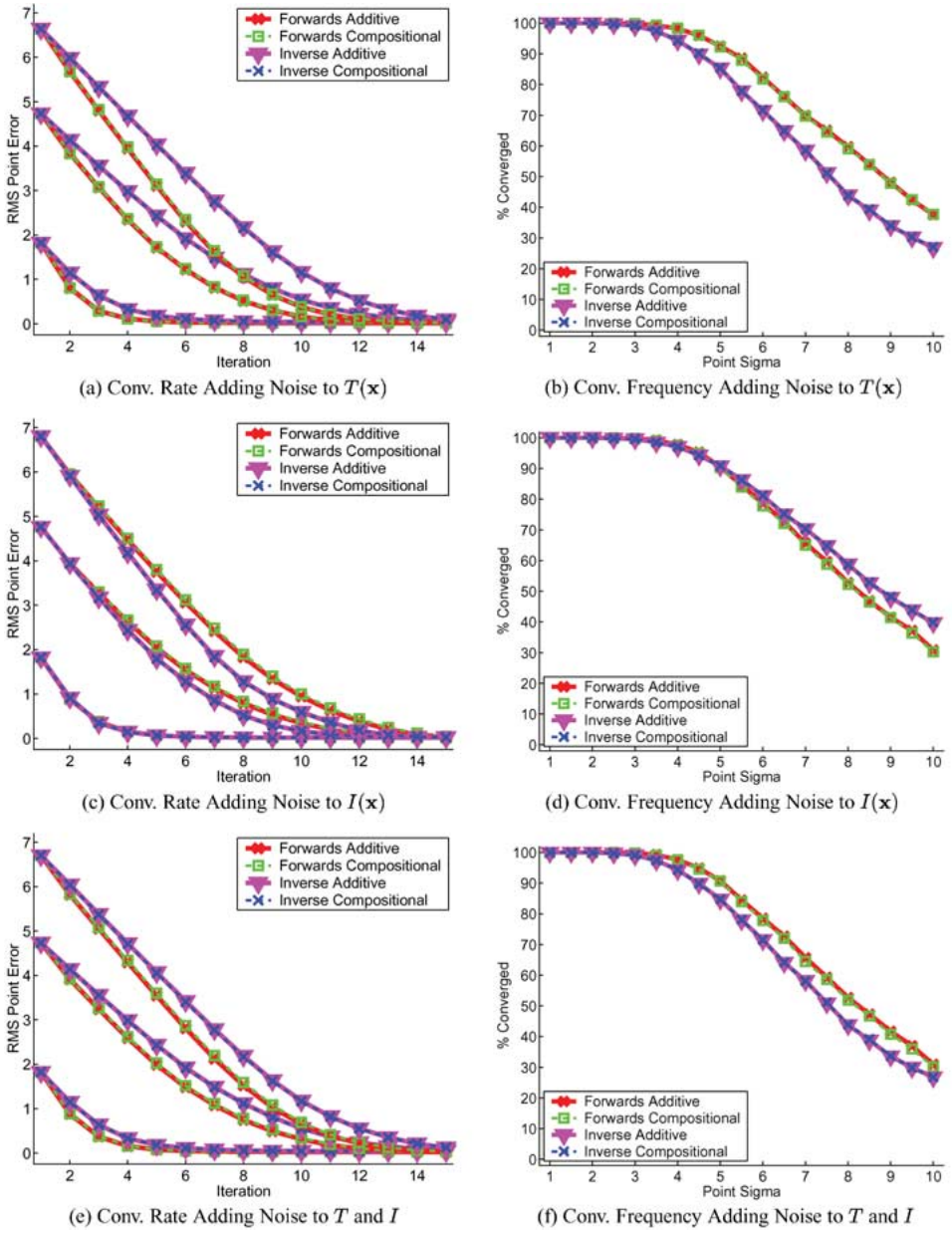
3.4.5. The Effect of Additive Intensity Noise. The results that we have presented so far have not included any noise on the images themselves (just on the canonical point locations which is not really noise but just a way of generating random warps.) The image $I(\mathbf{x})$ is warped with $\mathbf{W}(\mathbf{x};\mathbf{p})$ to generate the input image, a process which introduces a very small amount of re-sampling noise, but that re-sampling 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 Fig. 10.

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 $T(\mathbf{y}) - I(\mathbf{W}(\mathbf{y};\mathbf{p}))$ 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, the forwards algorithms perform marginally better than the inverse algorithms because the inverse algorithms are only first-order approximations to the forwards algorithms.

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 (Baker and Matthews, 2001), 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 image $I(\mathbf{x})$. In such cases, the inverse algorithms will be more robust to noise.

3.4.6. 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 × 100 pixel grey-scale template on a 933 MHz 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



(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 10. 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.

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*

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	Complexity	can be applied to
Forwards Additive	Lucas-Kanade (1981)	$O(n^2N + n^3)$	Any warp
Forwards Compositional	Shum-Szeliski (2000)	$O(n^2N + n^3)$	Any semi-group
Inverse Additive	Hager-Belhumeur (1998)	$O(nN + n^3)$	Simple linear 2D +
Inverse Compositional	Baker-Matthews (2001)	$O(nN + kN + k^3)$	Any group

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 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 has the slight advantage that the Jacobian is constant, and is in general simpler so is less likely to be computed erroneously (Shum and Szeliski, 2000). 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 Eq. (31) and updates the parameters using Eq. (32).

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 Eq. (31):

$$\sum_{\mathbf{x}}[F(\mathbf{x};\Delta\mathbf{p})]^2 \equiv \sum_{\mathbf{x}}[T(\mathbf{W}(\mathbf{x};\Delta\mathbf{p})) - I(\mathbf{W}(\mathbf{x};\mathbf{p}))]^2 \tag{62}$$

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 $F(\mathbf{x};\Delta\mathbf{p})$:

$$\sum_{\mathbf{x}}\left[F(\mathbf{x};\mathbf{0}) + \frac{\partial F}{\partial \mathbf{p}}\Delta\mathbf{p}\right]^2 \equiv \sum_{\mathbf{x}}\left[T(\mathbf{W}(\mathbf{x};\mathbf{0})) + \nabla T\frac{\partial \mathbf{W}}{\partial \mathbf{p}}\Delta\mathbf{p} - I(\mathbf{W}(\mathbf{x};\mathbf{p}))\right]^2. \tag{63}$$

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 Eq. (63) is quadratic in $\Delta\mathbf{p}$ and has the closed form solution:

$$\begin{aligned}\Delta\mathbf{p} &= -H^{-1}\sum_{\mathbf{x}}\left[\frac{\partial F}{\partial \mathbf{p}}\right]^{\mathrm{T}} F(\mathbf{x};\mathbf{0}) \\ &\equiv H^{-1}\sum_{\mathbf{x}}\left[\nabla T\frac{\partial \mathbf{W}}{\partial \mathbf{p}}\right]^{\mathrm{T}}[I(\mathbf{W}(\mathbf{x};\mathbf{p})) - T(\mathbf{x})]\end{aligned} \tag{64}$$

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

$$\begin{aligned}H &= \sum_{\mathbf{x}}\left[\frac{\partial F}{\partial \mathbf{p}}\right]^{\mathrm{T}}\left[\frac{\partial F}{\partial \mathbf{p}}\right] \\ &\equiv \sum_{\mathbf{x}}\left[\nabla T\frac{\partial \mathbf{W}}{\partial \mathbf{p}}\right]^{\mathrm{T}}\left[\nabla T\frac{\partial \mathbf{W}}{\partial \mathbf{p}}\right].\end{aligned} \tag{65}$$

One simple way to see that Eqs. (64) and (65) are the closed-form solution to Eq. (63) is to differentiate the expression in Eq. (63) with respect to $\Delta\mathbf{p}$ and set the result to zero:

$$2\sum_{\mathbf{x}}\left[\frac{\partial F}{\partial \mathbf{p}}\right]^{\mathrm{T}}\left[F(\mathbf{x};\mathbf{0}) + \frac{\partial F}{\partial \mathbf{p}}\Delta\mathbf{p}\right] = \mathbf{0}. \tag{66}$$

Rearranging this expression gives the closed form result in Eqs. (64) and (65).

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 (Gill et al., 1986; Press et al., 1992) 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:

$$\begin{aligned}\sum_{\mathbf{x}} G(\mathbf{x};\Delta\mathbf{p}) &= \sum_{\mathbf{x}} G(\mathbf{x};\mathbf{0}) + \sum_{\mathbf{x}}\frac{\partial G}{\partial \mathbf{p}}\Delta\mathbf{p} \\ &\quad + \frac{1}{2}\sum_{\mathbf{x}}\Delta\mathbf{p}^{\mathrm{T}}\frac{\partial^2 G}{\partial \mathbf{p}^2}\Delta\mathbf{p}\end{aligned} \tag{67}$$

where:

$$\sum_{\mathbf{x}}\frac{\partial G}{\partial \mathbf{p}} = \sum_{\mathbf{x}}\left(\frac{\partial G}{\partial p_1}\ \frac{\partial G}{\partial p_2}\cdots\frac{\partial G}{\partial p_n}\right) \tag{68}$$

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

$$\sum_{\mathbf{x}}\frac{\partial^2 G}{\partial \mathbf{p}^2} = \sum_{\mathbf{x}}\begin{pmatrix}\frac{\partial^2 G}{\partial p_1\partial p_1} & \frac{\partial^2 G}{\partial p_1\partial p_2} & \cdots & \frac{\partial^2 G}{\partial p_1\partial p_n} \\ \frac{\partial^2 G}{\partial p_2\partial p_1} & \frac{\partial^2 G}{\partial p_2\partial p_2} & \cdots & \frac{\partial^2 G}{\partial p_2\partial p_n} \\ \vdots & \vdots & & \vdots \\ \frac{\partial^2 G}{\partial p_n\partial p_1} & \frac{\partial^2 G}{\partial p_n\partial p_2} & \cdots & \frac{\partial^2 G}{\partial p_n\partial p_n}\end{pmatrix} \tag{69}$$

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 (Gill et al., 1986) Section 4.7.2. Expanding Eq. (63) gives the quadratic expression:

$$\begin{aligned}&\frac{1}{2}\sum_{\mathbf{x}}\left[F(\mathbf{x};\mathbf{0}) + \frac{\partial F}{\partial \mathbf{p}}\Delta\mathbf{p}\right]^2 \\ &\quad = \frac{1}{2}\sum_{\mathbf{x}}\left[F^2(\mathbf{x};\mathbf{0}) + 2F(\mathbf{x};\mathbf{0})\frac{\partial F}{\partial \mathbf{p}}\Delta\mathbf{p}\right. \\ &\qquad \left. + \Delta\mathbf{p}^{\mathrm{T}}\frac{\partial F}{\partial \mathbf{p}}^{\mathrm{T}}\frac{\partial F}{\partial \mathbf{p}}\Delta\mathbf{p}\right].\end{aligned} \tag{70}$$

Comparing Eqs. (67) and (70) 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:

$$\frac{\partial^2 G}{\partial \mathbf{p}^2} = \frac{\partial F}{\partial \mathbf{p}}^{\mathrm{T}} \frac{\partial F}{\partial \mathbf{p}}. \tag{71}$$

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:

$$\frac{\partial^2 G}{\partial \mathbf{p}^2} = \frac{\partial F}{\partial \mathbf{p}}^{\mathrm{T}} \frac{\partial F}{\partial \mathbf{p}} + F\frac{\partial^2 F}{\partial \mathbf{p}^2}. \tag{72}$$

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:

$$\frac{\partial G}{\partial \mathbf{p}} = \left[\nabla T \frac{\partial \mathbf{W}}{\partial \mathbf{p}}\right][T(\mathbf{W}(\mathbf{x};\Delta\mathbf{p})) - I(\mathbf{W}(\mathbf{x};\mathbf{p}))] \tag{73}$$

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:

$$\begin{aligned}\frac{\partial^2 G}{\partial \mathbf{p}^2} = &\left[\frac{\partial \mathbf{W}}{\partial \mathbf{p}}\right]^{\mathrm{T}}\left[\frac{\partial^2 T}{\partial \mathbf{x}^2}\right]\left[\frac{\partial \mathbf{W}}{\partial \mathbf{p}}\right] \\ &\times [T(\mathbf{W}(\mathbf{x};\Delta\mathbf{p})) - I(\mathbf{W}(\mathbf{x};\mathbf{p}))] \\ &+ \nabla T\left[\frac{\partial^2 \mathbf{W}}{\partial \mathbf{p}^2}\right][T(\mathbf{W}(\mathbf{x};\Delta\mathbf{p})) - I(\mathbf{W}(\mathbf{x};\mathbf{p}))] \\ &+ \left[\nabla T \frac{\partial \mathbf{W}}{\partial \mathbf{p}}\right]^{\mathrm{T}}\left[\nabla T \frac{\partial \mathbf{W}}{\partial \mathbf{p}}\right]\end{aligned} \tag{74}$$

where:

$$\frac{\partial T}{\partial \mathbf{x}} = \begin{pmatrix} \frac{\partial^2 T}{\partial x^2} & \frac{\partial^2 T}{\partial x \partial y} \\ \frac{\partial^2 T}{\partial x \partial y} & \frac{\partial^2 T}{\partial y^2} \end{pmatrix} \tag{75}$$

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

$$\begin{aligned}\nabla T\left[\frac{\partial^2 \mathbf{W}}{\partial \mathbf{p}^2}\right] = &\frac{\partial T}{\partial x}\begin{pmatrix} \frac{\partial^2 W_x}{\partial p_1 \partial p_1} & \cdots & \frac{\partial^2 W_x}{\partial p_1 \partial p_n} \\ \vdots & & \vdots \\ \frac{\partial^2 W_x}{\partial p_n \partial p_1} & \cdots & \frac{\partial^2 W_x}{\partial p_n \partial p_n} \end{pmatrix} \\ &+ \frac{\partial T}{\partial y}\begin{pmatrix} \frac{\partial^2 W_y}{\partial p_1 \partial p_1} & \cdots & \frac{\partial^2 W_y}{\partial p_1 \partial p_n} \\ \vdots & & \vdots \\ \frac{\partial^2 W_y}{\partial p_n \partial p_1} & \cdots & \frac{\partial^2 W_y}{\partial p_n \partial p_n} \end{pmatrix}.\end{aligned} \tag{76}$$

Equations (73) and (74) 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:

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

and the Hessian simplifies to:

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

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 Eq. (2) these values are:

$$\frac{\partial \mathbf{W}}{\partial \mathbf{p}} = \begin{pmatrix} x & 0 & y & 0 & 1 & 0 \\ 0 & x & 0 & y & 0 & 1 \end{pmatrix} \quad \text{and} \quad \frac{\partial^2 \mathbf{W}}{\partial \mathbf{p}^2} = \mathbf{0}. \tag{79}$$

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 Eq. (67), 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 (Gill et al., 1986; Press

The Newton Inverse Compositional Algorithm

Pre-compute:

(3) Evaluate the gradient ∇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 (78)
(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 (81)
(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. Compared to the Gauss-Newton inverse compositional algorithm in Fig. 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 $\mathrm{O}(n^2 N)$.

et al., 1992):

$$\left[\sum_{\mathbf{x}} \frac{\partial G}{\partial \mathbf{p}} \right]^{\mathrm{T}} + \sum_{\mathbf{x}} \frac{\partial^2 G}{\partial \mathbf{p}^2} \Delta \mathbf{p} = 0. \tag{80}$$

The minimum is then attained at:

$$\Delta \mathbf{p} = - \left[\sum_{\mathbf{x}} \frac{\partial^2 G}{\partial \mathbf{p}^2} \right]^{-1} \left[\sum_{\mathbf{x}} \frac{\partial G}{\partial \mathbf{p}} \right]^{\mathrm{T}} \tag{81}$$

where the gradient $\frac{\partial G}{\partial \mathbf{p}}$ and the Hessian $\frac{\partial^2 G}{\partial \mathbf{p}^2}$ are given in Eqs. (77) and (78). Ignoring the sign change and the transpose, Eqs. (77), (78), and (81) are almost identical to Eqs. (35) and (36) 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 Eq. (78); 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 Fig. 11.

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. See Table 7 for a summary and (Baker and Matthews, 2002) for the details.

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 Fig. 4, and asymptotically the same as the original Lucas-Kanade algorithm described in Fig. 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
$\mathrm{O}(n N)$	$\mathrm{O}(N)$	$\mathrm{O}(n^2 N)$	$\mathrm{O}(n N)$	$\mathrm{O}(n^3)$	$\mathrm{O}(n^2)$	$\mathrm{O}(n^2 N + n^3)$

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 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:

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

with the first order term:

$$\left[\nabla T \frac{\partial \mathbf{W}}{\partial \mathbf{p}}\right]^{\mathrm{T}}\left[\nabla T \frac{\partial \mathbf{W}}{\partial \mathbf{p}}\right]. \tag{83}$$

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:

$$\left[\sum_{\mathbf{x}} \frac{\partial^2 G}{\partial \mathbf{p}^2}\right]^{-1} = c\,\mathrm{I} \tag{84}$$

where I is the $n \times n$ identity matrix, we obtain the *steepest-descent algorithm*. Substituting this expression into Eq. (81) yields:

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

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 (Gill et al., 1986). One simple way to choose c is to substitute Eq. (85) back into Eq. (67) to yield:

$$\sum_{\mathbf{x}} G(\mathbf{x};\Delta\mathbf{p}) = \sum_{\mathbf{x}} G(\mathbf{x};\mathbf{0}) - c\left[\sum_{\mathbf{x}} \frac{\partial G}{\partial \mathbf{p}}\right] \times \left[\sum_{\mathbf{x}} \frac{\partial G}{\partial \mathbf{p}}\right]^{\mathrm{T}} + \frac{1}{2}c^2\left[\sum_{\mathbf{x}} \frac{\partial G}{\partial \mathbf{p}}\right] \times \left[\sum_{\mathbf{x}} \frac{\partial^2 G}{\partial \mathbf{p}^2}\right]\left[\sum_{\mathbf{x}} \frac{\partial G}{\partial \mathbf{p}}\right]^{\mathrm{T}}. \tag{86}$$

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

$$c = \frac{\left[\sum_{\mathbf{x}} \frac{\partial G}{\partial \mathbf{p}}\right]\left[\sum_{\mathbf{x}} \frac{\partial G}{\partial \mathbf{p}}\right]^{\mathrm{T}}}{\left[\sum_{\mathbf{x}} \frac{\partial G}{\partial \mathbf{p}}\right]\left[\sum_{\mathbf{x}} \frac{\partial^2 G}{\partial \mathbf{p}^2}\right]\left[\sum_{\mathbf{x}} \frac{\partial G}{\partial \mathbf{p}}\right]^{\mathrm{T}}} \tag{87}$$

where, as before:

$$\frac{\partial G}{\partial \mathbf{p}} = \left[\nabla T \frac{\partial \mathbf{W}}{\partial \mathbf{p}}\right][T(\mathbf{x}) - I(\mathbf{W}(\mathbf{x};\mathbf{p}))]. \tag{88}$$

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 Eq. (87) 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 Fig. 12.

4.3.2. Computational Cost of the Gauss-Newton Steepest Descent Algorithm. The computational cost of the Gauss-Newton steepest descent algorithm is almost exactly the same as the original inverse compositional algorithm. See Table 8 for a summary and Baker and Matthews (2002) for the details.

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, Eq. (87) is used to compute the value of c. This takes time $O(n^2)$ rather than time $O(n^3)$. The cost per iteration is therefore $O(n\,N+n^2)$ rather than $O(n\,N+n^3)$. The pre-computation time is still $O(n^2\,N)$.

Pre-computation

Step 3	Step 4	Step 5	Step 6	Total
$O(N)$	$O(n\,N)$	$O(n\,N)$	$O(n^2\,N)$	$O(n^2\,N)$

Per iteration

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

The GN Steepest Descent Inverse Compositional Algorithm

Pre-compute:

(3) Evaluate the gradient ∇T of the template $T(\mathbf{x})$
(4) Evaluate the Jacobian $\frac{\partial \mathbf{W}}{\partial \mathbf{p}}$ at $(\mathbf{x};\mathbf{0})$
(5) Compute the steepest descent images $\nabla T \frac{\partial \mathbf{W}}{\partial \mathbf{p}}$
(6) Compute $\sum_{\mathbf{x}} \frac{\partial^2 G}{\partial \mathbf{p}^2} = \sum_{\mathbf{x}} [\nabla T \frac{\partial \mathbf{W}}{\partial \mathbf{p}}]^{\mathrm{T}} [\nabla T \frac{\partial \mathbf{W}}{\partial \mathbf{p}}]$

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 (87) and $\Delta\mathbf{p}$ 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 12. 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 $O(n\,N + n^2)$ per iteration.

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:

$$\mathrm{Diag}\left[\frac{\partial^2 G}{\partial \mathbf{p}^2}\right] = \sum_{\mathbf{x}} \begin{pmatrix} \frac{\partial^2 G}{\partial p_1 \partial p_1} & 0 & \cdots & 0 \\ 0 & \frac{\partial^2 G}{\partial p_2 \partial p_2} & \cdots & 0 \\ \vdots & \vdots & & \vdots \\ 0 & 0 & \cdots & \frac{\partial^2 G}{\partial p_n \partial p_n} \end{pmatrix}. \quad (89)$$

The Diagonal Hessian Inverse Compositional Algorithm

Pre-compute:

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

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 $\mathrm{Diag}[\sum_{\mathbf{x}} \frac{\partial^2 G}{\partial \mathbf{p}^2}]$
(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 (91)
(9) Update the warp $\mathbf{W}(\mathbf{x};\mathbf{p}) \leftarrow \mathbf{W}(\mathbf{x};\mathbf{p}) \circ \mathbf{W}(\mathbf{x};\Delta\mathbf{p})^{-1}$

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

Figure 13. The Newton inverse compositional algorithm with the diagonal approximation to the Hessian is almost exactly the same as the full Newton version in Fig. 11. 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.

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 $O(n\,N)$ rather than $O(n^2\,N)$. If we denote the vector:

$$\mathrm{Diag}\left[\sum_{\mathbf{x}} \frac{\partial^2 G}{\partial \mathbf{p}^2}\right]^{-1} = \left(\frac{1}{\sum_{\mathbf{x}} \frac{\partial^2 G}{\partial p_1 \partial p_1}}, \frac{1}{\sum_{\mathbf{x}} \frac{\partial^2 G}{\partial p_2 \partial p_2}}, \ldots, \frac{1}{\sum_{\mathbf{x}} \frac{\partial^2 G}{\partial p_n \partial p_n}}\right)^{\mathrm{T}} \quad (90)$$

the update to the warp parameters in Eq. (81) becomes:

$$\Delta\mathbf{p} = -\mathrm{Diag}\left[\sum_{\mathbf{x}} \frac{\partial^2 G}{\partial \mathbf{p}^2}\right]^{-1} \cdot \left[\sum_{\mathbf{x}} \frac{\partial G}{\partial \mathbf{p}}\right]^{\mathrm{T}} \quad (91)$$

where the product $\cdot$ between these two vectors denotes element-wise multiplication and where the gradient and Hessian are given by Eqs. (77) and (78). The diagonal approximation to the Hessian can then be used directly in the Newton algorithm. See Fig. 13 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 (Szeliski and Golland, 1998) and super-resolution (Baker and Kanade, 2000).

Table 9. The computational cost of the Newton inverse compositional algorithm with the diagonal approximation to the Hessian. See Fig. 13 for the algorithm. Because only n elements in the Hessian ever need to be computed, Steps 4, 5, and 6 take time $O(nN)$ rather than $O(n^2N)$. Step 8 is also slightly quicker. Overall, the pre-computation only takes time $O(nN)$ and the cost per iteration is only $O(nN + n^2)$.

Pre-computation						
Step 3	Step 4	Step 5	Total			
$O(N)$	$O(nN)$	$O(nN)$	$O(nN)$			
Per iteration						
Step 1	Step 2	Step 6	Step 7	Step 8	Step 9	Total
$O(nN)$	$O(N)$	$O(nN)$	$O(nN)$	$O(n)$	$O(n^2)$	$O(nN + n^2)$

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. The cost is comparable with the Gauss-Newton inverse compositional algorithm. Because there are only n elements on the leading diagonal, Steps 4, 5, and 6 only take time $O(nN)$ rather than $O(n^2N)$. See Table 9 for a summary and (Baker and Matthews, 2002) for the details.

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 work better close to the local minima where the quadratic approximation is good (Gill et al., 1986; Press et al., 1992). 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:

$$\mathrm{H}_{\mathrm{LM}} = \sum_{\mathbf{x}} \left[\nabla T \frac{\partial \mathbf{W}}{\partial \mathbf{p}} \right]^{\mathrm{T}} \left[\nabla T \frac{\partial \mathbf{W}}{\partial \mathbf{p}} \right] + \delta \sum_{\mathbf{x}} \begin{pmatrix} \left(\nabla T \frac{\partial \mathbf{W}}{\partial p_1} \right)^2 0 & \cdots & 0 \\ 0 & \left(\nabla T \frac{\partial \mathbf{W}}{\partial p_2} \right)^2 & \cdots & 0 \\ \vdots & \vdots & & \vdots \\ 0 & 0 & \cdots & \left(\nabla T \frac{\partial \mathbf{W}}{\partial p_n} \right)^2 \end{pmatrix} \tag{92}$$

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

$$\Delta \mathbf{p} = -{\mathrm{H}_{\mathrm{LM}}}^{-1} \sum_{\mathbf{x}} \left[\nabla T \frac{\partial \mathbf{W}}{\partial \mathbf{p}} \right] [T(\mathbf{x}) - I(\mathbf{W}(\mathbf{x}; \mathbf{p}))] . \tag{93}$$

For very small $\delta \ll 1$, the Levenberg-Marquardt Hessian H_{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 δ, 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 δ increased, $\delta \rightarrow \delta \times 10$, say. The Levenberg-Marquardt inverse compositional algorithm is summarized in Fig. 14.

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. $I(\mathbf{W}(\mathbf{x};\mathbf{p})) - T(\mathbf{x})$, e $= \sum_{\mathbf{x}}[I(\mathbf{W}(\mathbf{x};\mathbf{p})) - T(\mathbf{x})]^2$
(3) Evaluate the gradient ∇T of the template $T(\mathbf{x})$
(4) Evaluate the Jacobian $\frac{\partial \mathbf{W}}{\partial \mathbf{p}}$ at $(\mathbf{x};\mathbf{0})$
(5) Compute the steepest descent images $\nabla T \frac{\partial \mathbf{W}}{\partial \mathbf{p}}$
(6) Compute the GN Hessian matrix $\sum_{\mathbf{x}}[\nabla T \frac{\partial \mathbf{W}}{\partial \mathbf{p}}]^{\mathrm{T}}[\nabla T \frac{\partial \mathbf{W}}{\partial \mathbf{p}}]$

Iterate:

(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 H_{LM} using Equation (92) and $\Delta\mathbf{p}$ using Equation (93)
(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. $I(\mathbf{W}(\mathbf{x};\mathbf{p})) - T(\mathbf{x})$, e$^* = \sum_{\mathbf{x}}[I(\mathbf{W}(\mathbf{x};\mathbf{p})) - T(\mathbf{x})]^2$
(10) If e $<$ e* then $\delta \rightarrow \delta \times 10$; undo Steps 9, 1, & 2; else $\delta \rightarrow \delta/10$; e $=$ e*

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

Figure 14. 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 δ is reduced and the next iteration started. If the error increased, the value of δ 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 $O(nN + n^3)$ per iteration. The pre-computation cost is $O(n^2N)$.

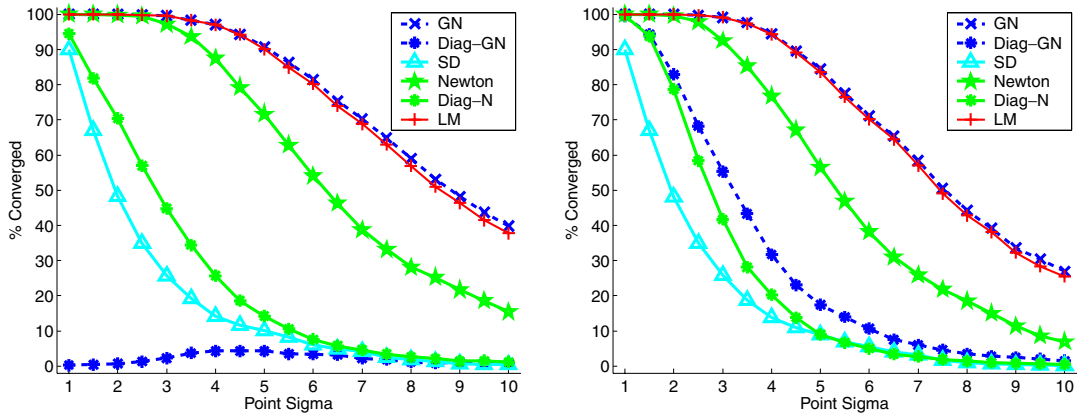
4.5.1. Computational Cost of the Levenberg-Marquardt Inverse Compositional Algorithm. Compared with the Gauss-Newton inverse compositional algorithm, the Levenberg-Marquardt algorithm requires that a number of steps be re-ordered, a couple of steps to be extended, and two new steps added. The re-ordering doesn't affect the computation cost of the algorithm; it only marginally increases the pre-computation time, although not asymptotically. The new Steps 0 and 10 both take very little time, constant if implemented with pointers to the new and old data. Overall the Levenberg-Marquardt algorithm is just as efficient as the Gauss-Newton inverse compositional algorithm. See Table 10 for a summary and Baker and Matthews (2002) for the details.

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.

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 Fig. 4. However, overall the new algorithm is just as efficient as the original algorithm taking time $O(nN + n^3)$ per iteration and $O(n^2N)$ as a pre-computation cost.

Pre-computation							
Step 0	Step 1	Step 2	Step 3	Step 4	Step 5	Step 6	Total
$O(1)$	$O(nN)$	$O(N)$	$O(N)$	$O(nN)$	$O(nN)$	$O(n^2N)$	$O(n^2N)$
Per iteration							
Step 7	Step 8	Step 9	Step 1	Step 2	Step 10	Total	
$O(nN)$	$O(n^3)$	$O(n^2)$	$O(nN)$	$O(N)$	$O(1)$	$O(nN + n^3)$	



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

Figure 15. 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.

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 Fig. 15(a) we plot the average frequency of convergence (computed over 5000 samples) with no intensity noise. In Fig. 15(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.

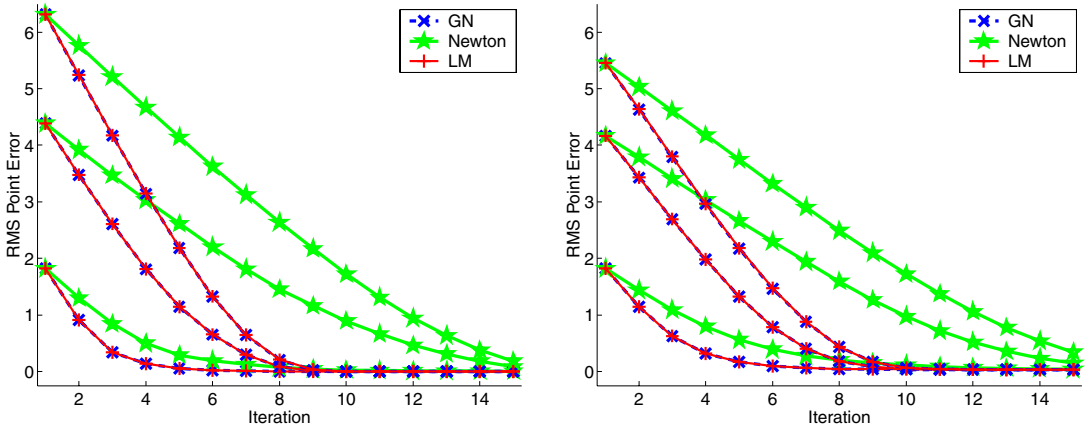
The first thing to notice in Fig. 15 is that the best performing algorithms are Gauss-Newton and Levenberg-Marquardt. Perhaps surprisingly, Levenberg-Marquardt doesn't perform any better than Gauss-Newton, even for larger displacements of the canonical point locations.

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, 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 Fig. 15 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.

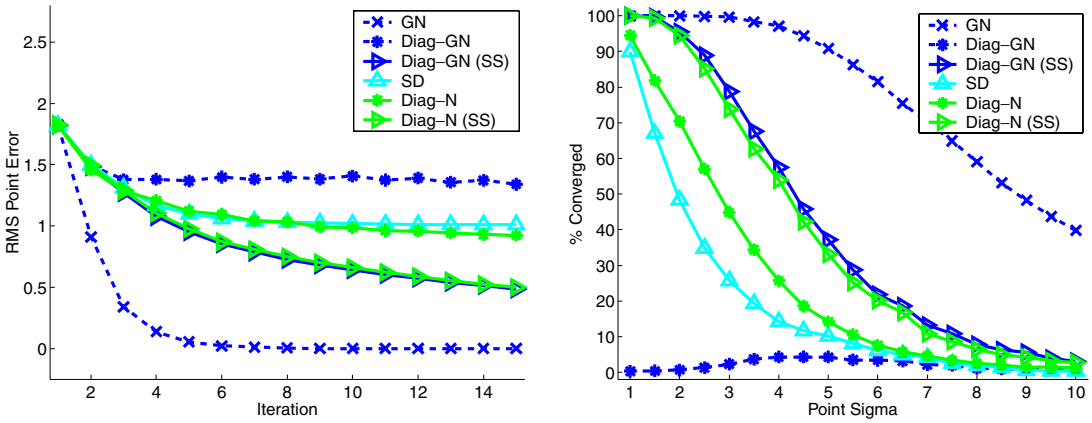
4.6.2. Average Convergence Rates. Since the steepest descent and diagonal Hessian algorithms converge so rarely, we only plot the average rate of convergence for the Gauss-Newton, Newton, and Levenberg-Marquardt algorithms. The other three algorithms converge very slowly, and sometimes oscillate. See Section 4.6.3 for more discussion. In Fig. 16(a) we plot the average rate of convergence with no intensity noise and in Fig. 16(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



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

Figure 16. 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.



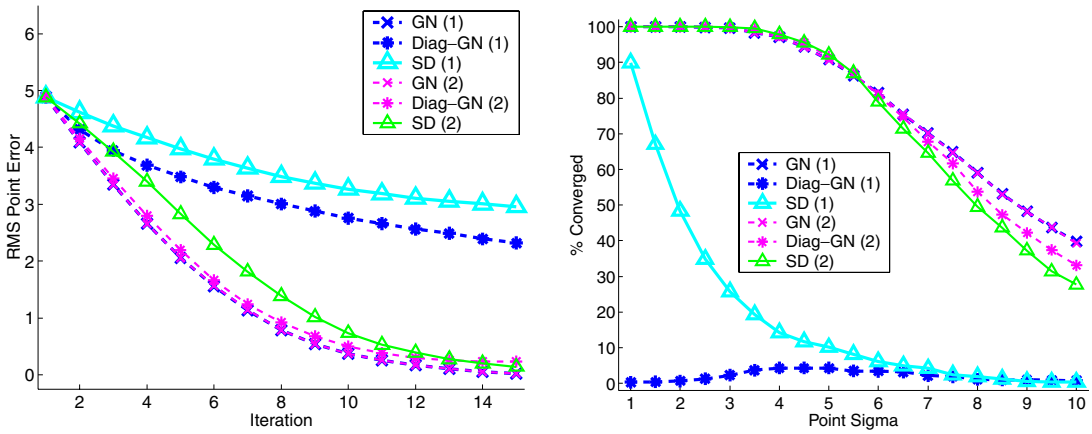
(a) Average Convergence Rate (b) Average Frequency of Convergence

Figure 17. 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 Eqs. (87) and (88). With the step size estimation step, the performance of the diagonal Hessian algorithms improves dramatically.

very poorly. One particular surprise in Fig. 15 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 Eqs. (87) and (88). 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 Gill et al., 1986 for some possibilities.) The performance of the diagonal Hessian algorithms improves dramatically when this step is added. See Fig. 17 for a comparison of the performance of the diagonal Hessian algorithms with and without the step size estimation step.



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

Figure 18. 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 Eqs. (1) and (2) a parameterization based on the location of three canonical points. Gauss-Newton is relatively unaffected by the re-parameterization whereas the performance of the other algorithms is dramatically improved.

The diagonal approximation to the Hessian is used in various vision algorithms such as stereo (Szeliski and Golland, 1998) and super-resolution (Baker and Kanade, 2000). Our results indicate that these algorithms may be under-performing 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 Eq. (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 re-parameterization 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 Fig. 18.

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 (Szeliski and Golland, 1998) and super-resolution (Baker and Kanade, 2000). More on the question of how to best parameterize a set of warps is left to a future paper in this 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 100×100 pixel grey-scale template on a 933 MHz Pentium-IV. As can be seen, all of the algorithms are roughly equally fast 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)

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 × 100 pixel grey-scale template on a 933 MHz Pentium-IV. Steps 0 and 10 for Levenberg-Marquardt are negligible and so are omitted for lack of space.

	Step 1	Step 2	Step 3	Step 4	Step 5	Step 6	Step 7	Step 8	Step 9	Total
Pre-computation										
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
	Step 1	Step 2	Step 3	Step 4	Step 5	Step 6	Step 7	Step 8	Step 9	Total
Per iteration										
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

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 efficient except Newton. When combined with the forwards compositional algorithm, only the steepest descent and the diagonal Hessian algorithms are efficient. Only Gauss-Newton and Levenberg-Marquardt converge well empirically.

Algorithm	Complexity w/ inverse comp?	Complexity w/ forwards comp?	Convergence rate	Convergence frequency
Gauss-Newton	$O(nN + n^3)$	$O(n^2N + n^3)$	Fast	High
Newton	$O(n^2N + n^3)$	$O(n^2N + n^3)$	Medium	Medium
Steepest Descent	$O(nN + n^2)$	$O(nN + n^2)$	Slow	Low
Gauss-Newton Diagonal Hessian	$O(nN + n^2)$	$O(nN + n^2)$	Slow	Low
Newton Diagonal Hessian	$O(nN + n^2)$	$O(nN + n^2)$	Slow	Low
Levenberg-Marquardt	$O(nN + n^3)$	$O(n^2N + n^3)$	Fast	High

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. 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 Section 4 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 Shum and Szeliski (2000) 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 entire template. See Eq. (11). In Shum and Szeliski (2000) it is suggested that this computation can be speeded up by splitting the template into patches and assuming that the term summed in Eq. (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 Dellaert and Collins (1999) 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 Eq. (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 assuming that linear relationship is constant and: (1) using linear regression to find the coefficients (Cootes et al., 1998) and (2) numerically estimating the coefficients using a "difference decomposition" (Gleicher, 1997; Sclaroff and Isidoro, 1998; La Cascia et al., 2000).

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 two choices are orthogonal. For example, one could derive a *forwards compositional steepest descent* algorithm.

The results of the first half are summarized in Table 6. All four algorithms empirically perform equivalently. The only differences between the algorithms are their computational complexity and the set of warps they can be applied to. The results of the second half are summarized in Table 12. The algorithms differ in both their computational complexity and their empirical performance.

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 techniques to try to avoid local minima. In future papers in this 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.

6. Matlab Code, Test Images, and Scripts

Matlab implementations of all of the algorithms described in this paper are available on the World Wide Web at: http://www.ri.cmu.edu/projects/project_515.html. We have also included all of the test images and the scripts used to generate the experimental results in this paper.

Appendix A: Inverse Compositional Derivations for the Homography

A.1. Gauss-Newton Inverse Compositional Algorithm

To apply the Gauss-Newton inverse compositional algorithm (see Fig. 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:

$$\mathbf{W}(\mathbf{x};\mathbf{p}) = \frac{1}{1+p_7x+p_8y}\begin{pmatrix}(1+p_1)x+p_3y+p_5\\ p_2x+(1+p_4)y+p_6\end{pmatrix}. \tag{94}$$

There are other ways to parameterize homographies, however this way is perhaps the most common. The Jacobian of the homography in Eq. (94) is:

$$\frac{\partial\mathbf{W}}{\partial\mathbf{p}} = \frac{1}{1+p_7x+p_8y} \times \begin{pmatrix} x & 0 & y & 0 & 1 & 0 & \frac{-x[(1+p_1)x+p_3y+p_5]}{1+p_7x+p_8y} & \frac{-y[(1+p_1)x+p_3y+p_5]}{1+p_7x+p_8y} \\ 0 & x & 0 & y & 0 & 1 & \frac{-x[p_2x+(1+p_4)y+p_6]}{1+p_7x+p_8y} & \frac{-y[p_2x+(1+p_4)y+p_6]}{1+p_7x+p_8y} \end{pmatrix}. \tag{95}$$

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

$$\frac{1}{1+p_7\Delta p_5+p_8\Delta p_6} \times \begin{pmatrix} p_1+\Delta p_1+p_1\Delta p_1+p_3\Delta p_2+p_5\Delta p_7 \\ \quad -p_7\Delta p_5-p_8\Delta p_6 \\ p_2+\Delta p_2+p_2\Delta p_1+p_4\Delta p_2+p_6\Delta p_7 \\ p_3+\Delta p_3+p_1\Delta p_3+p_3\Delta p_4+p_5\Delta p_8 \\ p_4+\Delta p_4+p_2\Delta p_3+p_4\Delta p_4+p_6\Delta p_8 \\ \quad -p_7\Delta p_5-p_8\Delta p_6 \\ p_5+\Delta p_5+p_1\Delta p_5+p_3\Delta p_6 \\ p_6+\Delta p_6+p_2\Delta p_5+p_4\Delta p_6 \\ p_7+\Delta p_7+p_7\Delta p_1+p_8\Delta p_2 \\ p_8+\Delta p_8+p_7\Delta p_3+p_8\Delta p_4 \end{pmatrix}. \tag{96}$$

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

$$\frac{1}{\det\cdot[(1+p_1)(1+p_4)-p_2p_3]} \times \begin{pmatrix} 1+p_4-p_6p_8-\det \\ \quad \cdot[(1+p_1)(1+p_4)-p_2p_3] \\ -p_2+p_6p_7 \\ -p_3+p_5p_8 \\ 1+p_1-p_5p_7-\det \\ \quad \cdot[(1+p_1)(1+p_4)-p_2p_3] \\ -p_5-p_4p_5+p_3p_6 \\ -p_6-p_1p_6+p_2p_5 \\ -p_7-p_4p_7+p_2p_8 \\ -p_8-p_1p_8+p_3p_7 \end{pmatrix} \tag{97}$$

where det is the determinant:

$$\det = \begin{vmatrix} 1+p_1 & p_3 & p_5 \\ p_2 & 1+p_4 & p_6 \\ p_7 & p_8 & 1 \end{vmatrix}. \tag{98}$$

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 Newton Hessian of the homography. The only part of the Newton Hessian that has not be derived before is $\frac{\partial^2\mathbf{W}}{\partial\mathbf{p}^2}$. This quantity 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

$$\frac{\partial^2 W_x}{\partial \mathbf{p}^2} = \frac{1}{(1+p_7x+p_8y)^2} \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & -x^2 & -xy \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -xy & -y^2 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -x & -y \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -x^2 & 0 & -xy & 0 & -x & 0 & \frac{2x^2[(1+p_1)x+p_3y+p_5]}{1+p_7x+p_8y} & \frac{2xy[(1+p_1)x+p_3y+p_5]}{1+p_7x+p_8y} \\ -xy & 0 & -y^2 & 0 & -y & 0 & \frac{2xy[(1+p_1)x+p_3y+p_5]}{1+p_7x+p_8y} & \frac{2y^2[(1+p_1)x+p_3y+p_5]}{1+p_7x+p_8y} \end{pmatrix} \tag{99}$$

and the y component is

$$\frac{\partial^2 W_y}{\partial \mathbf{p}^2} = \frac{1}{(1+p_7x+p_8y)^2} \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -x^2 & -xy \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -xy & -y^2 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -x & -y \\ 0 & -x^2 & 0 & -xy & 0 & -x & \frac{2x^2[p_2x+(1+p_4)y+p_6]}{1+p_7x+p_8y} & \frac{2xy[p_2x+(1+p_4)y+p_6]}{1+p_7x+p_8y} \\ 0 & -xy & 0 & -y^2 & 0 & -y & \frac{2xy[p_2x+(1+p_4)y+p_6]}{1+p_7x+p_8y} & \frac{2y^2[p_2x+(1+p_4)y+p_6]}{1+p_7x+p_8y} \end{pmatrix}. \tag{100}$$

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 IJCV reviewers and the CVPR reviewers of Baker and Matthews (2001) for their feedback. The research described in this paper was conducted under U.S. Office of Naval Research contract N00014-00-1-0915.

References

Baker, S. and Kanade, T. 2000. Limits on super-resolution and how to break them. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 372–379.

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

Baker, S. and Matthews, I. 2002. Lucas-kande 20 years on: A unifying framework: Part 1. Technical Report CMU-RI-TR-02-16, Carnegie Mellon University Robotics Institute.

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

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

La Cascia, M., Sclaroff, S., and Athitsos, V. 2000. 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.

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

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

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

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

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

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

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

Press, W.H., Flannery, B.P., Teukolsky, S.A., and Vetterling, W.T. 1992. *Numerical Recipes in C: The Art of Scientific Computing*, 2nd edn., Cambridge University Press.

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

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

Szeliski, R. and Golland, P. 1998. Stereo matching with transparency and matting. In *Proceedings of the 6th IEEE International Conference on Computer Vision*, pp. 517–524.