# 1 Problem Statement

Let I and J be two 2D grayscaled images. The two quantities I(x) = I(x, y) and J(x) = J(x, y) are then the
grayscale value of the two images are the location x = [x y]^ T , where x and y are the two pixel coordinates of a generic image point x.
I와 J를 2 개의 2D 회색 음영 이미지라고합시다. 2 개의 양 I (x) = I (x, y) 및 J (x) = J (x, y)는 두 이미지의 그레이 스케일 값이 위치 x = [x y]^T이고, x와 y는 일반적인 이미지의 두 픽셀 좌표 x 이다.

The image I will sometimes be referenced as the first image, and the image J as the second image.
이미지 I은 때로 첫 번째 이미지로 참조되고 이미지 J는 두 번째 이미지로 참조됩니다.

For practical issues, the images I and J are discret function (or arrays), and the upper left corner pixel coordinate vector is [0 0]^T.
실용적인 문제의 경우 이미지 I 및 J는 이산 함수 (또는 배열)이고 왼쪽 위 모서리 픽셀 좌표 벡터는 [0 0] T입니다.

Let n_x and n_y be the width and height of the two images.
n_x와 n_y를 두 이미지의 너비와 높이라고합시다.

Then the lower right pixel coordinate vector is [n_x − 1 n_y − 1]^T.
그런 다음 오른쪽 아래 픽셀 좌표 벡터는 [n_x - 1 n_y - 1]^T 입니다.

Consider an image point u = [u_x u_y]^T on the first image I.
첫 번째 이미지 I에서 이미지 포인트 u = [u_x u_y]^T를 생각해 보자.

The goal of feature tracking is to find the location v = u+d = [u_x +d_x u_y +d_y]^T on the second image J such as I(u) and J(v) are “similar”.
특징 추적의 목적은 I (u)와 J (v)와 같은 두 번째 이미지 J상의 위치 v = u + d = [u_x + d_x u_y + d_y]^T 가 ""“similar”"하다는 것을 찾는 것입니다.

The vector d = [d_x d_y]^T is the image velocity at x, also known as the optical flow at x.
벡터 d = [d_x d_y]^T는 x의 이미지 속도이며, x의 옵티컬 플로우로 알려져있다.

Because of the aperture problem, it is essential to define the notion of similarity in a 2D neighborhood sense.
구경(aperture) 문제 때문에 2D 유사도에서 유사성 개념을 정의하는 것이 필수적입니다.

Let ω_x and ω_y two integers. We define the image velocity d as being the vector that minimizes the residual function aaa defined as follows:
ω_x와 ω_y를 두 개의 정수로합시다. 우리는 이미지 속도 d를 다음과 같이 정의 된 잔류 함수 aaa 를 최소화하는 벡터로 정의한다.

$$\epsilon(\mathbf{d}) = \epsilon(d_x, d_y) = \sum_{x=u_x-\omega_x}^{u_x+\omega_x} \sum_{y=u_y-\omega_y}^{u_y+\omega_y} (I(x,y) - J(x+d_x, y+d_y))^2 .$$

(1)

Observe that following that defintion, the similarity function is measured on a image neighborhood of size (2ω_x + 1)×(2ω_y + 1).
이 정의에 따라 유사도 함수는 크기 (2ω_x + 1) × (2ω_y + 1)의 이미지 근처에서 측정됩니다.

This neighborhood will be also called integration window. Typical values for ωx and ωy are 2,3,4,5,6,7pixels.
이웃은 통합 창이라고도합니다. ωx 및 ωy의 일반적인 값은 2,3,4,5,6,7 픽셀입니다.

## 2 Description of the tracking algorithm

The two key components to any feature tracker are accuracy and robustness.
모든 기능 추적기의 두 가지 주요 구성 요소는 정확성과 견고성입니다.

The accuracy component relates to the local sub-pixel accuracy attached to tracking. Intuitively, a small integration window would be preferable in order not to “smooth out” the details contained in the images (i.e small values of ω_x and ω_y).
정확도 구성 요소는 추적에 첨부 된 로컬 하위 픽셀 정확도와 관련됩니다. 직관적으로, 작은 통합 창은 이미지에 포함 된 세부 사항을 "smooth out"만들지 않는 것이 바람직합니다 (i.e는 작은 값 ω_x 및 ω_y).

That is especially required at occluding areas in the images where two patchs potentially move with very different velocities.
이것은 특히 두 개의 패치가 잠재적으로 매우 다른 속도로 움직이는 이미지의 영역을 가려 낼 때 필요합니다.

The robustness component relates to sensitivity of tracking with respect to changes of lighting, size of image motion,... In particular, in oder to handle large motions, it is intuively preferable to pick a large integration window.

견고성 구성 요소는 조명의 변화, 이미지 모션의 크기와 관련하여 추적의 민감도와 관련이 있습니다. 특히 대형 모션을 처리하기 위해서는 대형 통합 창을 선택하는 것이 좋습니다.

Indeed, considering only equation 1, it is preferable to have dx ≤ ωx and dy ≤ ωy (unless some prior matching information is available).
견고성 구성 요소는 조명의 변화, 이미지 모션의 크기와 관련하여 추적의 민감도와 관련이 있습니다. 특히 대형 모션을 처리하기 위해서는 대형 통합 창을 선택하는 것이 좋습니다.

There is therefore a natural tradeoff between local accuracy and robustness when choosing the integration window size.

따라서 통합 창 크기를 선택할 때 로컬 정확도와 견고성 사이에는 자연스러운 트레이드 오프가 있습니다.

In provide to provide a solution to that problem, we propose a pyramidal implementation of the classical Lucas-Kanade algorithm.
그 문제에 대한 해결책을 제공하기 위해 고전적인 Lucas-Kanade 알고리즘의 피라미드 구현을 제안합니다.

An iterative implementation of the Lucas-Kanade optical flow computation provides sufficient local tracking accuracy.
Lucas-Kanade 옵티컬 플로우 계산을 반복적으로 구현하면 충분한 로컬 추적 정확도를 얻을 수 있습니다.

## 2.1 Image pyramid representation

Let us define the pyramid representsation of a generic image I of size n_x × n_y.
크기 n_x × n_y의 일반 이미지 I의 피라미드 표현을 정의하자.

Let I^0 = I be the “zero^th” level image.
I ^ 0 = "zero ^ th"레벨 이미지라고합시다.

This image is essentially the highest resolution image (the raw image).
이 이미지는 기본적으로 가장 높은 해상도의 이미지 (원시 이미지)입니다.

The image width and height at that level are defined as n^ 0_ x = n_x and n^0_y = n_y.
해당 레벨의 이미지 너비와 높이는 n ^ 0_ x = n_x 및 n ^ 0_y = n_y로 정의됩니다.

The pyramid representation is then built in a recursive fashion: compute I 1 from I 0 , then compute I 2 from I 1, and so on… Let L = 1, 2, . . . be a generic pyramidal level, and let IL−1 be the image at level L − 1. Denote n L−1 x and n L−1 y the width and height of I^L−1.
피라미드 표현은 재귀 적 방식으로 구축됩니다 : I^0에서 I^1을 계산 한 다음 I^1에서 I^2를 계산하는 등 ... L = 1, 2는 일반 피라미드 레벨이되고 IL-1은 level L − 1의 이미지가됩니다. . n^ L-1_x와 n^L-1 _y를 I^L-1의 폭과 높이로 표시하십시오.

Denote n^L−1_x and n^L−1_y the width and height of I^L−1. The image I^L−1 is then defined as follows:
n ^ L-1_x와 n ^ L-1_y는 I ^ L-1의 너비와 높이를 나타냅니다. 이미지 I ^ L-1은 다음과 같이 정의된다.

$$I^L(x,y) = \frac{1}{4}I^{L-1}(2x,2y) + \frac{1}{8}\left(I^{L-1}(2x-1,2y) + I^{L-1}(2x+1,2y) + I^{L-1}(2x,2y-1) + I^{L-1}(2x,2y+1)\right) + \frac{1}{16}\left(I^{L-1}(2x-1,2y-1) + I^{L-1}(2x+1,2y+1) + I^{L-1}(2x-1,2y+1) + I^{L-1}(2x+1,2y+1)\right). \quad (2)$$

For simplicity in the notation, let us define dummy image values one pixel around the image I^L−1(for 0 ≤ x ≤ n^ L−1_x −1 and 0 ≤ y ≤ n^L−1_y − 1):
표기법을 간단히하기 위해 이미지 I ^ L-1 (0 ≤ x ≤ n ^ L-1_x -1 및 0 ≤ y ≤ n ^ L-1_y - 1)을 중심으로 1 픽셀의 더미 이미지 값을 정의 해 보겠습니다. :

For simplicity in the notation, let us dene dummy image values one pixel around the image I^L1 (for 0<= x <= n^L-1_x-1 and 0 <= y<= n^L-1_y - 1):
표기법을 간단히하기 위해, 이미지 I ^ L1 주위에 하나의 픽셀을 더미 이미지 값으로 사용하자 :

$$
\begin{aligned}
I^{L-1}(-1,y) &\doteq I^{L-1}(0,y),\\
I^{L-1}(x,-1) &\doteq I^{L-1}(x,0),\\
I^{L-1}(n_x^{L-1},y) &\doteq I^{L-1}(n_x^{L-1}-1,y),\\
I^{L-1}(x,n_y^{L-1}) &\doteq I^{L-1}(x,n_y^{L-1}-1),\\
I^{L-1}(n_x^{L-1},n_y^{L-1}) &\doteq I^{L-1}(n_x^{L-1}-1,n_y^{L-1}-1).
\end{aligned}
$$

Then, equation 2 is only dened for values of x and y such that 0 < 2x < n^L-1_x -1 and 0 < 2y < n^L-1_y- 1. Therefore, the width n^L_x and height n^L_y of I^L are the largest integers that satisfy the two conditions:
그러면 식 2는 0 <2x <n ^ L-1_x -1 및 0 <2y <n ^ L-1_y-1이되도록 x와 y의 값에 대해서만 정의됩니다. 따라서 너비 n ^ L_x와 높이 n ^ I ^ L의 L_y는 두 조건을 만족하는 가장 큰 정수입니다.

$$
n_x^L \le \frac{n_x^{L-1}+1}{2}, \tag{3}
$$

$$
n_y^L \le \frac{n_y^{L-1}+1}{2}. \tag{4}
$$

Equations (2), (3) and (4) are used to construct recursively the pyramidal representations of the two images I and J:{I^L}L=0,.....,Lm and {J^L}L=0…….Lm . The value Lm is the height of the pyramid (picked heuristically). Practical values of Lm are 2,3,4. For typical image sizes, it makes no sense to go above a level 4. For example, for an image I of size 640x480, the images I1, I2, I3 and I4 are of respective sizes 320x240, 160x120, 80x60 and 40x30. going beyond level 4 does not make much sense in most cases. The central motivation behind pyramidal representation is to be able to handle large pixel motions (larger than the integration window sizes Wx and Wy). Therefore the pyramid height (Lm) should also be picked appropriately according to the maximum expected optical flow in the next image. Thesection describing the tracking operation in detail we let us understand that concept better. Final observation: equation 2 suggests that the lowpass lter [1/4 1/2 1/4][1/4 1/2 1/4]^T is used for image anti-aliasing before image subsampling. In practice however (in the C code) a larger anti-aliasing lter kernel is used for pyramid construction [1/16 1/4 3/8 1/4 1/16] [1/16 1/4 3/8 1/4 1/16]^T . The formalism remains the same.

식 (2), (3) 및 (4)는 두 이미지 I 및 J의 피라미드 표현을 재귀 적으로 구성하는 데 사용된다. L = 0, ....., Lm 및 {J ^ L} L = 0 ..... .Lm. 값 Lm은 피라미드의 높이 (경험적으로 선택)입니다. Lm의 실용적인 가치는 2,3,4입니다. 일반적인 이미지 크기의 경우 레벨 4 이상으로 올라가는 것은 의미가 없습니다. 예를 들어 크기 640x480의 이미지 I의 경우 이미지 I1, I2, I3 및 I4는 각각 320x240, 160x120, 80x60 및 40x30 크기입니다.레벨 4를 초과 할 대부분의 경우 큰 의미가 없습니다. 피라미드 표현 뒤에 중심적인 동기는 큰 픽셀 모션 (통합 창 크기 Wx 및 Wy보다 큰)을 처리 할 수있는 것이다. 따라서 피라미드의 높이 (Lm)도 다음 이미지의 최대 예상 광학 흐름에 따라 적절하게 선택해야합니다. 추적 작업을 자세히 설명하는 부분에서는 개념을 더 잘 이해할 수 있도록했습니다. 최종 관찰 : 방정식 2는 저역 통과 [1 / 4 / 1 / 4] [1 / 4 / 1 / 4] ^ T가 이미지 서브 샘플링 전에 이미지 앤티 앨리어싱에 사용된다는 것을 나타냅니다.

## 2.2 Pyramidal Feature Tracking

Recall the goal of feature tracking: for a given point u in image I, nd its corresponding location v = u + d in image J, or alternatively nd its pixel displacement vector d (see equation 1).
For L = 0;.....;Lm, dene u^L = [u^L_x u^L_y ], the corresponding coordinates of the point u on the pyramidal images I^L. Following our denition of the pyramid representation equations (2), (3) and (4), the vectors uL are computed as follows:

피쳐 추적의 목표를 떠올려보십시오 : 이미지 I의 주어진 점 u에 대해 이미지 J의 해당 위치 v = u + d를 찾거나 픽셀 변위 벡터 d (식 1 참조)를 택하십시오.L = 0, ....., Lm, dene u ^ L = [u ^ L_x u ^ L_y], 피라미드 이미지 I ^ L상의 점 u의 상응하는 좌표. 피라미드 표현 방정식 (2), (3) 및 (4)의 정의에 따라, 벡터 u^L은 다음과 같이 계산됩니다.

$$\mathbf{u}^L = \frac{\mathbf{u}}{2^L}. \qquad (5)$$

The division operation in equation 5 is applied to both coordinates independently (so will be the multiplication operations appearing in subsequent equations). Observe that in particular, u^0 = u. The overall pyramidal tracking algorithm proceeds as follows: rst, the optical folllow is comptuted at the deepest pyramid level Lm. Then, the result of the that computation is propagated to the upper level Lm - 1 in a form of an initial guess for the pixel displacement (at level Lm-1). Given that initial guess, the rened optical flow is computed at level Lm - 1, and the result is propagated to level Lm - 2 and so on up to the level 0 (the original image).
Let us now describe the recursive operation between two generics levels L+1 and L in more mathematical details. Assume that an initial guess for optical flow at level L, g^L = [g^L_x g^L_
y ]^T is available from the computations done from level Lm to level L+1. Then, in order to compute the optical flow at level L, it is necessary to nd the residual pixel displacement vector dL = [d^L_x d^L_y]^T that minimizes the new image matching error function E^L:

$$\epsilon^L(\mathbf{d}^L) = \epsilon^L(d_x^L, d_y^L) = \sum_{x=u_x^L-\omega_x}^{u_x^L+\omega_x} \sum_{y=u_y^L-\omega_y}^{u_y^L+\omega_y} \left(I^L(x,y) - J^L(x+g_x^L+d_x^L, y+g_y^L+d_y^L)\right)^2. \quad (6)$$

방정식 5의 나누기 연산은 두 좌표에 독립적으로 적용됩니다 (따라서 후속 방정식에 나타나는 곱셈 연산이됩니다). 특히 u ^ 0 = u를 관찰하십시오. 전반적인 피라미드 추적 알고리즘은 다음과 같이 진행됩니다. 첫째, 광학 folllow는 가장 깊은 피라미드 레벨 Lm에서 계산됩니다. 그 계산 결과는 (레벨 Lm-1에서) 픽셀 변위에 대한 초기 추측의 형태로 상위 레벨 Lm-1로 전파됩니다. 초기 추측을 감안할 때, rened 옵티컬 플로우는 레벨 Lm - 1에서 계산되고, 결과는 레벨 Lm - 2로 전달되어 레벨 0 (원본 이미지)까지 확대됩니다. 이제 더 많은 수학적 세부 사항에서 2 개의 제네릭 레벨 L + 1과 L 사이의 재귀 연산을 설명합시다. 레벨 L에서의 옵티컬 플로우에 대한 초기 추측을 가정하면, g ^ L = [g ^ L_x g ^ L_ y] ^ T는 레벨 Lm에서 레벨 L + 1까지의 계산에서 사용 가능합니다. 그런 다음, 레벨 L에서 옵티컬 플로우를 계산하기 위해, 새로운 이미지 매칭 에러 함수 E^L를 최소화하는 잔여 픽셀 변위 벡터 dL = [d^L_x d^L_y]^T:

Observe that the window of integration is of constant size (2w_x + 1) X (2w_y + 1) for all values of L. Notice that the initial guess flow vector g^L is used to pre-translate the image patch in the second image J. That way, the residual flow vector d^L = [d^L_x d^L_y]T is small and therefore easy to compute through a standard Lucas Kanade step.
통합의 창은 L의 모든 값에 대해 일정한 크기 (2w_x + 1) X (2w_y + 1)임을 관찰하십시오. 초기 추측 흐름 벡터 g ^ L은 두 번째 이미지의 이미지 패치를 사전 변환하는 데 사용됩니다 J. 그런 식으로, 잔류 유동 벡터는 작은 루카스 Kanade 단계를 통해 계산하기 쉽고 따라서 작습니다.

The details of computation of the residual optical flow d^L will be described in the next section 2.3. For now, let usassume that this vector is computed (to close the main loop of the algorithm). Then, the result of this computation is propagated to the next level L- 1 by passing the new initial guess g^L-1 of expression:
잔여 옵티컬 플로우 d ^ L의 계산에 대한 자세한 내용은 다음 섹션 2.3에서 설명합니다. 지금 당장이 벡터가 계산된다는 것을 알려주십시오 (알고리즘의 메인 루프를 닫습니다). 그 다음,이 계산의 결과는 다음과 같이 표현의 새로운 초기 추측 g ^ L-1을 전달하여 다음 레벨 L-1로 전달됩니다.

$$\mathbf{g}^{L-1} = 2\left(\mathbf{g}^{\mathrm{L}} + \mathbf{d}^L\right). \quad (7)$$

The next level optical flow residual vector d^L-1 is then computed through the same procedure. This vector, computed by optical flow computation (decribed in Section 2.3), minimizes the functional E^L-1(d^L-1) (equation 6). This procedure goes on until the nest image resolution is reached (L = 0). The algorithm is initialized by setting the initial guess for level Lm to zero (no initial guess is available at the deepest level of the pyramid):
다음 레벨의 옵티컬 플로우 잔류 벡터 dL-1은 동일한 절차를 통해 계산된다. 옵티컬 플로우 계산 (2.3 절에서 설명)에 의해 계산 된이 벡터는 함수 E ^ L-1 (d ^ L-1) (식 6)을 최소화합니다. 이 절차는 네스트 이미지 해상도에 도달 할 때까지 계속됩니다 (L = 0). 알고리즘은 레벨 Lm에

대한 초기 추측을 0으로 설정하여 초기화됩니다 (피라미드의 가장 깊은 레벨에서 초기 추측을 사용할 수 없음).

$$\mathbf{g}^{L_m} = [0 \;\; 0]^T. \tag{8}$$

The final optical flow solution d (refer to equation 1) is then available after the nest optical flow computation:
final 옵티컬 플로우 솔루션 d (방정식 1 참조)는 옵티컬 플로우 계산 후 사용 가능합니다.

$$\mathbf{d} = \mathbf{g}^0 + \mathbf{d}^0. \tag{9}$$

Observe that this solution may be expressed in the following extended form:
이 솔루션은 다음과 같은 확장 된 형식으로 표현 될 수 있음을 관찰하십시오:

$$\mathbf{d} = \sum_{L=0}^{L_m} 2^L \mathbf{d}^L. \tag{10}$$

The clear advantage of a pyramidal implementation is that each residual optical flow vector d^L can be kept very small while computing a large overall pixel displacement vector d. Assuming that each elementary optical flow computation step can handle pixel motions up to $d_{max}$, then the overall pixel motion that the pyramidal implementation can handle becomes $d_{max}$ final = (2Lm+1 - 1) $d_{max}$. For example, for a pyramid depth of $L_m$ = 3, this means a maximum pixel displacement gain of 15! This enables large pixel motions, while keeping the size of the integration window relatively small.
피라미드 구현의 분명한 이점은 큰 전체 픽셀 변위 벡터 d를 계산하는 동안 각각의 잔류 옵티컬 플로우 벡터 d ^ L을 매우 작게 유지할 수 있다는 것입니다. 각각의 기본 옵티컬 흐름 계산 단계가 픽셀 움직임을 최대 $d_{max}$까지 처리 할 수 있다고 가정하면 피라미드 구현이 처리 할 수있는 전체 픽셀 모션은 $d_{max}$ final = (2Lm + 1 - 1) $d_{max}$가됩니다. 예를 들어 피라미드 깊이가 Lm = 3 인 경우 최대 픽셀 변위 gain은 15!입니다. 이는 통합 윈도우의 크기를 비교적 작게 유지하면서 큰 픽셀 모션을 가능하게합니다.

## 2.3 Iterative Optical Flow Computation (Iterative Lucas-Kanade)

Let us now describe the core optical flow computation. At every level L in the pyramid, the goal is finding the vector dL that minimizes the matching function E^L defined in equation 6. Since the same type of operation is per-formed for all levels L, let us now drop the superscripts L and dene the new images A and B as follows:
코어 옵티컬 플로우 계산을 설명합시다. 피라미드의 모든 레벨 L에서, 목표는 방정식 6에서 정의 된 일치 함수 E^L을 최소화하는 벡터 dL을 찾는 것입니다. 모든 레벨 L에 대해 동일한 유형의 연산이 수행되므로, 위 첨자 L과 dene 새 이미지 A와 B는 다음과 같습니다.

.

$$\forall (x,y) \in [p_x - \omega_x - 1, p_x + \omega_x + 1] \times [p_y - \omega_y - 1, p_y + \omega_y + 1],$$
$$A(x,y) \doteq I^L(x,y), \tag{11}$$
$$\forall (x,y) \in [p_x - \omega_x, p_x + \omega_x] \times [p_y - \omega_y, p_y + \omega_y],$$
$$B(x,y) \doteq J^L(x + g_x^L, y + g_y^L). \tag{12}$$

Observe that the domains of denition of A(x; y) and B(x; y) are slightly different. Indeed, A(x; y) is defined over a window of size $(2w_x+3)(2w_y+3)$ instead of $(2w_x+1)(2w_y+1)$. This difference will become clear when computing spatial derivatives of A(x; y) using the central difference operator (eqs. 19 and 20). For clarity purposes, let us change the name of the displacement vector v = [Vx Vy]T = d^L, as well as the image position vector p = [Px Py]T = u^L.

Following that new notation, the goal is to nd the displacement vector v = [Vx Vy]^T that minimizes the matching function:

A (x; y)와 B (x; y)의 정의 영역이 약간 다름을 관찰하십시오. 실제로 A (x; y)는 (2wx + 1) (2wy + 1) 대신에 크기 (2wx + 3) (2wy + 3)의 창에 정의됩니다. 이 차이는 중앙 차분 연산자 (식 19 및 식 20)를 사용하여 A (x; y)의 공간 도함수를 계산할 때 명확 해집니다. 명확함을 위해 이미지 위치 벡터 p = [Px Py] T = u ^ L뿐 아니라 변위 벡터 v = [Vx Vy] T = d ^ L의 이름을 변경합시다.

새로운 표기법에 따라 목표는 일치 함수를 최소화하는 변위 벡터 v = [Vx Vy] ^ T를 찾는 것입니다.

$$\varepsilon(\overline{\nu}) = \varepsilon(\nu_x, \nu_y) = \sum_{x=p_x-\omega_x}^{p_x+\omega_x} \sum_{y=p_y-\omega_y}^{p_y+\omega_y} (A(x,y) - B(x+\nu_x, y+\nu_y))^2. \tag{13}$$

A standard iterative Lucas-Kanade may be applied for that task. At the optimimum, the rst derivative of E with respect to V is zero:

Lucas-Kanade 표준 반복 작업이이 작업에 적용될 수 있습니다. 최적 조건에서 V에 대한 E의 첫 번째 미분은 0입니다.

$$\left.\frac{\partial \varepsilon(\overline{\nu})}{\partial \overline{\nu}}\right|_{\overline{\nu}=\overline{\nu}_{\text{opt}}} = [0 \;\; 0]. \tag{14}$$

After expansion of the derivative, we obtain:

파생 상품을 확장 한 후 다음을 얻습니다.

$$\frac{\partial \varepsilon(\overline{\nu})}{\partial \overline{\nu}} = -2 \sum_{x=p_x-\omega_x}^{p_x+\omega_x} \sum_{y=p_y-\omega_y}^{p_y+\omega_y} (A(x,y) - B(x+\nu_x, y+\nu_y)) \cdot \left[ \frac{\partial B}{\partial x} \;\; \frac{\partial B}{\partial y} \right]. \tag{15}$$

Let us now substitute B(x + Vx ,y +V y) by its rst order Taylor expansion about the point = [0 0]^T (this has a good chance to be a valid approximation since we are expecting a small displacement vector, thanks to the pyramidal scheme):

이제 B(x + Vx ,y +V y) 를 point = [0 0] ^ T에 대한 첫 번째 테일러 전개로 대체 해 봅시다 (피라미드 방식 덕분에 작은 변위 벡터를 기대하기 때문에 유효한 근사값이 될 수 있습니다).

$$\frac{\partial \varepsilon(\overline{\nu})}{\partial \overline{\nu}} \approx -2 \sum_{x=p_x-\omega_x}^{p_x+\omega_x} \sum_{y=p_y-\omega_y}^{p_y+\omega_y} \left( A(x,y) - B(x,y) - \begin{bmatrix} \frac{\partial B}{\partial x} & \frac{\partial B}{\partial y} \end{bmatrix} \overline{\nu} \right) \cdot \begin{bmatrix} \frac{\partial B}{\partial x} & \frac{\partial B}{\partial y} \end{bmatrix}. \quad (16)$$

Observe that the quatity A(x; y) - B(x; y) can be interpreted as the temporal image derivative at the point [x y]^T :
A (x; y) - B (x; y)는 점 [x y]에서 시간적 이미지 도함수로 해석 될 수 있음을 관찰하라. T :

$$\forall (x,y) \in [p_x - \omega_x, p_x + \omega_x] \times [p_y - \omega_y, p_y + \omega_y],$$

$$\delta I(x,y) \doteq A(x,y) - B(x,y). \quad (17)$$

The matrix [@B/@x @B/@y] is merely the image gradient vector. Let's make a slight change of notation:
행렬 [@ B / @ x @ B / @ y]는 단지 이미지 그래디언트 벡터입니다. 표기법을 약간 변경해 보겠습니다.

$$\nabla I = \begin{bmatrix} I_x \\ I_y \end{bmatrix} \doteq \begin{bmatrix} \frac{\partial B}{\partial x} & \frac{\partial B}{\partial y} \end{bmatrix}^T. \quad (18)$$

Observe that the image derivatives Ix and Iy may be computed directly from the rst image A(x; y) in the (2Wy + 1) X (2wy + 1) neighborhood of the point p independently from the second image B(x; y) (the importance of this observation will become apparent later on when describing the iterative version of the flow computation). If a central dfference operator is used for derivative, the two derivative images have the following expression:
이미지 도함수 Ix 및 Iy는 제 2 이미지 B (x; y)와 독립적으로 포인트 p의 (2Wy + 1) X (2wy + 1) 이웃의 첫 번째 이미지 A (x,y )이다. (이 관찰의 중요성은 나중에 유량 계산의 반복 버전을 설명 할 때 분명해질 것이다). 중심 도수 연산자가 도함수에 사용되면 두 도함수 이미지는 다음과 같은 표현을 갖습니다.

$$\forall (x,y) \in [p_x - \omega_x, p_x + \omega_x] \times [p_y - \omega_y, p_y + \omega_y],$$

$$I_x(x,y) = \frac{\partial A(x,y)}{\partial x} = \frac{A(x+1,y) - A(x-1,y)}{2}, \quad (19)$$

$$I_y(x,y) = \frac{\partial A(x,y)}{\partial y} = \frac{A(x,y+1) - A(x,y-1)}{2}. \quad (20)$$

In practice, the Sharr operator is used for computing image derivatives (very similar to the central difference operator).
Following this new notation, equation 16 may be written:
Sharr 연산자는 이미지 파생물을 계산하는 데 사용됩니다 (중앙 차등 연산자와 매우 유사합니다).
이 새로운 표기법에 따라, 등식 16이 쓰여질 수 있습니다 :

$$\frac{1}{2}\frac{\partial \varepsilon(\overline{\nu})}{\partial \overline{\nu}} \approx \sum_{x=p_x-\omega_x}^{p_x+\omega_x} \sum_{y=p_y-\omega_y}^{p_y+\omega_y} \left(\nabla I^T \overline{\nu} - \delta I\right) \nabla I^T, \tag{21}$$

$$\frac{1}{2}\left[\frac{\partial \varepsilon(\overline{\nu})}{\partial \overline{\nu}}\right]^T \approx \sum_{x=p_x-\omega_x}^{p_x+\omega_x} \sum_{y=p_y-\omega_y}^{p_y+\omega_y} \left(\begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix} \overline{\nu} - \begin{bmatrix} \delta I I_x \\ \delta I I_y \end{bmatrix}\right). \tag{22}$$

Denote

$$G \doteq \sum_{x=p_x-\omega_x}^{p_x+\omega_x} \sum_{y=p_y-\omega_y}^{p_y+\omega_y} \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix} \quad \text{and} \quad \overline{b} \doteq \sum_{x=p_x-\omega_x}^{p_x+\omega_x} \sum_{y=p_y-\omega_y}^{p_y+\omega_y} \begin{bmatrix} \delta I I_x \\ \delta I I_y \end{bmatrix}. \tag{23}$$

Then, equation 22 may be written:
그러면, 수학 식 22가 작성 될 수있다 :

$$\frac{1}{2}\left[\frac{\partial \varepsilon(\overline{\nu})}{\partial \overline{\nu}}\right]^T \approx G\overline{\nu} - \overline{b}. \tag{24}$$

Therefore, following equation 14, the optimimum optical flow vector is
따라서, 수학 식 14에 따르면, 최적의 옵티컬 플로우 벡터는

$$\overline{\nu}_{\text{opt}} = G^{-1}\overline{b}. \tag{25}$$

This expression is valid only if the matrix G is invertible. That is equivalent to saying that the image A(x; y) contains gradient information in both x and y directions in the neighborhood of the point p.
이 표현식은 행렬 G가 역전 일 때만 유효합니다. 이는 이미지 A (x; y)가 점 p 근처에서 x 및 y 방향으로 모두 기울기 정보를 포함한다고 말하는 것과 같습니다.

This is the standard Lucas-Kanade optical flow equation, which is valid only if the pixel displacement is small (in order ot validate the rst order Taylor expansion). In practice, to get an accurate solution, is is necessary to iterate multiple times on this scheme (in a Newton-Raphson fashion).
이것은 Lucas-Kanade 표준 광학 흐름 방정식으로 픽셀 변위가 작은 경우에만 유효합니다 (첫 번째 테일러 확장의 유효성을 검사하기 위해). 실제로, 정확한 해결책을 얻으려면이 계획에 대해 Newton-Raphson 방식으로 여러 번 반복해야합니다.

Now that we have introduced the mathematical background, let us give the details of the iterative version of the algorithm. Recall the goal: find the vector V  that minimizes the error functional E(V) introduced in equation 13.
이제 수학적 배경을 소개 했으므로 알고리즘의 반복 버전에 대한 세부 정보를 제공합니다. 목표를 상기하자 : 식 13에서 소개 된 오차 함수 E (V)를 최소화하는 벡터 V를 찾는다.

Let k be the iterative index, initialized to 1 at the very rst iteration. Let us describe the algorithm recursively: at a generic iteration k >= 1, assume that the previous computations from iterations 1, 2,..... , k - 1 provide an initial guess V^k-1 = [V^k-1_x  V^k-1_x ]^T for the pixel displacement V. Let $B_k$ be the new translated image according to that initial guess V^k-1:

k를 반복 인덱스로하고, 첫 번째 반복에서 1로 초기화합니다. 알고리즘을 재귀 적으로 기술해보자. 일반 iteration k> = 1에서 iterations 1, 2, ....., k - 1의 이전 계산은 픽셀 변위 V에 대한 초기 추측 V^k-1 = [V^k-1_x  V^k-1_x ]^T  를 제공한다고 가정한다. 초기 추측에 따라 Bk를 새로운 번역 된 이미지라고하자. V ^ k-1 :

$$\forall (x,y) \in [p_x - \omega_x, p_x + \omega_x] \times [p_y - \omega_y, p_y + \omega_y],$$

$$B_k(x,y) = B(x + \nu_x^{k-1}, y + \nu_y^{k-1}). \tag{26}$$

The goal is then to compute the residual pixel motion vector n^k = [n^k_x n^k_y ] that minimizes the error function
따라서 목표는 오차 함수를 최소화하는 잔여 픽셀 모션 벡터n^k = [n^k_x n^k_y ] 를 계산하는 것입니다

$$\varepsilon^k(\overline{\eta}^k) = \varepsilon(\eta_x^k, \eta_y^k) = \sum_{x=p_x-\omega_x}^{p_x+\omega_x} \sum_{y=p_y-\omega_y}^{p_y+\omega_y} \left(A(x,y) - B_k(x+\eta_x^k, y+\eta_y^k)\right)^2. \tag{27}$$

The solution of this minimization may be computed through a one step Lucas-Kanade optical flow computation (equation 25)
이 최소화의 해는 하나의 단계 Lucas-Kanade 옵티컬 플로우 계산 (방정식 25)을 통해 계산 될 수있다.

$$\overline{\eta}^k = G^{-1}\overline{b}_k, \tag{28}$$

where the 2 x 1 vector $b_k$ is dened as follows (also called image mismatch vector):
여기서 2 x 1 벡터 bk는 다음과 같이 정의됩니다 (이미지 불일치 벡터라고도 함).

$$\overline{b}_k = \sum_{x=p_x-\omega_x}^{p_x+\omega_x} \sum_{y=p_y-\omega_y}^{p_y+\omega_y} \begin{bmatrix} \delta I_k(x,y) I_x(x,y) \\ \delta I_k(x,y) I_y(x,y) \end{bmatrix}, \tag{29}$$

where the k^th image difference 8Ik are dfiened as follows:
k^th 이미지 차이 (8Ik)는 다음과 같이 표현된다 :

$$\forall (x,y) \in [p_x - \omega_x, p_x + \omega_x] \times [p_y - \omega_y, p_y + \omega_y],$$

$$\delta I_k(x,y) = A(x,y) - B_k(x,y). \tag{30}$$

Observe that the spatial derivatives $I_x$ and $I_y$ (at all points in the neighborhood of p) are computed only once at the beginning of the iterations following equations 19 and 20. Therefore the 2 X 2 matrix G also remains constant throughout the iteration loop (expression given in equation 23). That constitutes a clear computational advantage.

식 (19) 및 (20)에 계속 반복이 시작될 때 공간 미분 Ix 및 Iy (p 근방의 모든 점)가 1 회만 계산되는 것을 관찰한다.따라서 2 × 2 행렬 G도 반복 루프 (식 23에 주어진 식)을 통해 일정하게 유지한다. 그것은 명확한 계산상의 장점을 구성한다.

The only quantity that needs to be recomputed at each step k is the vector b^k that really captures the amount of residual difference between the image patches after translation by the vector V^k-1. Once the residual optical flow 7^k is computed through equation 28, a new pixel displacement guess V^k is computed for the next iteration step k + 1:
각 단계 k에서 재 계산 될 필요가있는 유일한 양은 벡터 V ^ k-1에 의한 변환 후에 이미지 패치들 간의 잔차 차이의 양을 실제로 포착하는 벡터 b ^ k이다. 수학 식 28을 통해 잔류 옵티컬 플로우가 계산되면, 다음 반복 단계 k + 1에 대해 새로운 픽셀 변위 추정치 V ^ k가 계산된다 :

$$\overline{v}^k = \overline{v}^{k-1} + \overline{\eta}^k. \qquad (31)$$

The iterative scheme goes on until the computed pixel residual 7^k is smaller than a threshold (for example 0.03 pixel), or a maximum number of iteration (for example 20) is reached. On average, 5 iterations are enough to reach convergence. At the rst iteration (k = 1) the initial guess is initialized to zero:
반복 계획은 계산 된 픽셀 잔차 7 ^ k가 임계 값 (예를 들면 0.03 픽셀)보다 작거나 반복의 최대 개수 (예 : 20)에 도달 할 때까지 계속된다. 평균 수렴에 도달하기 위해 5 회 반복 충분합니다. 첫 번째 반복 (k = 1) 첫 번째 추측은 0으로 초기화됩니다.

$$\overline{v}^0 = [0 \;\; 0]^T. \qquad (32)$$

Assuming that K iterations were necessary to reach convergence, the nal solution for the optical flow vector V = d^L is:
수렴에 도달하기 위해 K 회 반복이 필요하다고 가정하면 광학 흐름 벡터 V = d ^ L에 대한 최종 해는 다음과 같습니다.

$$\overline{v} = \mathrm{d}^L = \overline{v}^K = \sum_{k=1}^{K} \overline{\eta}^k. \qquad (33)$$

This vector minimizes the error functional described in equation 13 (or equation 6). This ends the description of the iterative Lucas-Kanade optical flow computation. The vector d^L is fed to equation 7 and this overall procedure is repeated at all subsequent levels L -1,L - 2,...., 0 (see section 2.2).
이 벡터는 식 13 (또는 식 6)에 설명 된 오류 기능을 최소화합니다. 이것은 Lucas-Kanade 반복 옵티컬 플로우 계산의 설명을 종료한다. 벡터 d ^ L은 방정식 7에 공급되며이 전체 과정은 모든 후속 레벨 L -1, L - 2, ..., 0에서 반복된다 (2.2 절 참조).

## 2.4 Summary of the pyramidal tracking algorithm

Let is now summarize the entire tracking algorithm in a form of a pseudo-code. Find the details of the equations in the main body of the text (especially for the domains of denition).

이제 전체 추적 알고리즘을 의사 코드 형태로 요약 해 보겠습니다. 텍스트 본문의 방정식을 더 찾고 있습니다 (특히 부정 도메인의 경우).

Goal: Let u be a point on image I. Find its corresponding location v on image J
목표 : u를 이미지 I의 한 점으로 놓고 이미지 J의 해당 위치 v를 찾습니다.

Build pyramid representations of I and J:
I 및 J의 피라미드 표현 만들기 :

$$\{I^L\}_{L=0,\dots,L_m} \text{ and } \{J^L\}_{L=0,\dots,L_m} \qquad \text{(eqs. 2,3,4)}$$

Initialization of pyramidal guess:
피라미드 추측의 초기화 :

$$\mathbf{g}^{L_m} = [g_x^{L_m} \;\; g_x^{L_m}]^T = [0 \;\; 0]^T \qquad \text{(eq. 8)}$$

for L = Lm down to 0 with step of -1
L = Lm에 대해 -1의 단계로 0까지 내려 간다.

Location of point u on image I^L:
이미지상의 포인트 u의 위치 I ^ L :

$$\mathbf{u}^L = [p_x \;\; p_y]^T = \mathbf{u}/2^L \qquad \text{(eq. 5)}$$

Derivative of I^L with respect to x:
x에 관한 I^L의 유도 :

$$I_x(x,y) = \frac{I^L(x+1,y) - I^L(x-1,y)}{2} \qquad \text{(eqs. 19,11)}$$

Derivative of I^L with respect to y:
y에 관한 I^L의 유도 :

$$I_y(x,y) = \frac{I^L(x,y+1) - I^L(x,y-1)}{2} \qquad \text{(eqs. 20,11)}$$

Spatial gradient matrix:
공간 그라디언트 행렬 :

$$G = \sum_{x=p_x-\omega_x}^{p_x+\omega_x} \sum_{y=p_y-\omega_y}^{p_y+\omega_y} \begin{bmatrix} I_x^2(x,y) & I_x(x,y)\,I_y(x,y) \\ I_x(x,y)\,I_y(x,y) & I_y^2(x,y) \end{bmatrix} \qquad \text{(eq. 23)}$$

Initialization of iterative L-K:
반복적 인 L-K의 초기화 :

$$\overline{\nu}^0 = [0 \;\; 0]^T \qquad \text{(eq. 32)}$$

for k = 1 to K with step of 1 (or until 7^k < accuracy threshold)
스텝 1 (또는 7 ^ k <정확도 임계 값)까지의 k = 1에서 K까지

Image difference:
이미지 차이 :

$$\delta I_k(x,y) = I^L(x,y) - J^L(x + g_x^L + \nu_x^{k-1}, y + g_y^L + \nu_y^{k-1}) \qquad \text{(eqs. 30,26,12)}$$

Image mismatch vector:
이미지 불일치 벡터 :

$$\overline{b}_k = \sum_{x=p_x-\omega_x}^{p_x+\omega_x} \sum_{y=p_y-\omega_y}^{p_y+\omega_y} \begin{bmatrix} \delta I_k(x,y)\, I_x(x,y) \\ \delta I_k(x,y)\, I_y(x,y) \end{bmatrix} \qquad \text{(eq. 29)}$$

Optical flow (Lucas-Kanade):
광학 흐름 (Lucas-Kanade) :

$$\overline{\eta}^k = G^{-1}\, \overline{b}_k \qquad \text{(eq. 28)}$$

Guess for next iteration:
다음 반복을위한 추측 :

$$\overline{\nu}^k = \overline{\nu}^{k-1} + \overline{\eta}^k \qquad \text{(eq. 31)}$$

end of for-loop on k
k에 대한 for-loop의 끝

Final optical flow at level L:
레벨 L에서 최종 광학 흐름 :

$$\mathbf{d}^L = \overline{\nu}^K \qquad \text{(eq. 33)}$$

Guess for next level L - 1:
다음 단계에 대한 추측 L - 1 :

$$\mathbf{g}^{L-1} = [g_x^{L-1} \;\; g_y^{L-1}]^T = 2\,(\mathbf{g^L} + \mathbf{d}^L) \qquad \text{(eq. 7)}$$

end of for-loop on L
L에 대한 for-loop의 끝

Final optical flow vector:

최종 옵티컬 플로우 벡터 :

$$\mathbf{d} = \mathbf{g}^0 + \mathbf{d}^0 \qquad \text{(eq. 9)}$$

Location of point on J:
J상의 지점 위치 :

$$\mathbf{v} = \mathbf{u} + \mathbf{d}$$

Solution: The corresponding point is at location v on image J
해결책 : 해당 지점이 이미지 J의 위치 v에 있습니다.

# 2.5 Subpixel Computation

It is absolutely essential to keep all computation at a subpixel accuracy level. It is therefore necessary to be able to compute image brightness values at locations between integer pixels (refer for example to equations 11,12 and 26).
모든 계산을 서브 픽셀 정확도 수준으로 유지하는 것은 절대적으로 중요합니다. 따라서 정수 픽셀 사이의 위치에서 이미지 밝기 값을 계산할 수 있어야합니다 (예 : 수식 11, 12 및 26참조).

In order to compute image brightness at subpixel locations we propose to use bilinear interpolation.
서브 픽셀 위치에 이미지 밝기를 계산하기 위해, 우리는 이중 선형 보간을 이용하는 것을 제안한다.

Let L be a generic pyramid level. Assume that we need the image value I^L(x, y) where x and y are not integers.
L을 일반 피라미드 레벨이라고합시다. x와 y가 정수가 아닌 이미지 값 I ^ L (x, y)가 필요하다고 가정합니다.

Let x0 and y0 be the integer parts of x and y (larger integers that are smallers than x and y).
x0와 y0을 x와 y의 정수 부분 (x와 y보다 작은 더 큰 정수)이라하자.

Let ax and ay be the two reminder values (between 0 and 1) such that:
ax와 ay가 0과 1 사이의 두 가지 알림 값이되도록 다음과 같이합니다.

$$x = x_o + \alpha_x, \qquad (34)$$
$$y = y_o + \alpha_y. \qquad (35)$$

Then I^L(x; y) may be computed by bilinear interpolation from the original image brightness values:
그리고 I L (x; y)는 원본 이미지 밝기 값의 쌍 선형 보간에 의해 계산 될 수있다.

$$I^L(x,y) = (1-\alpha_x)(1-\alpha_y)I^L(x_o,y_o) + \alpha_x(1-\alpha_y)I^L(x_o+1,y_o) + (1-\alpha_x)\alpha_y I^L(x_o,y_o+1) + \alpha_x\alpha_y I^L(x_o+1,y_o+1).$$

Let us make a few observations that are associated to subpixel computation (important implementation issues).
서브 픽셀 계산과 관련된 몇 가지 관찰을 해보겠습니다 (중요한 구현 문제).

Refer to the summary of the algorithm given in section 2.4. When computing the two image derivatives I^x(x; y) and I^y(x; y) in the neigborhood (x, y) E [Px-Wx, Px +Wx]X[Py -Wy, Py +Wy], it is necessary to have the brightness values of I^L(x, y) in the neighborhood (x, y) E[Px - Wx - 1, Px +Wx + 1] X [Py - Wy - 1, Py + Wy + 1] (see equations 19, 20).

2.4 절에 주어진 알고리즘의 요약을 참조하십시오 .neigborhood (x, y) E [Px-Wx, Px +Wx]X[Py -Wy, Py +Wy]에서 두 개의 이미지 도함수 I ^ x (x; y)와 I ^ y (x; y)를 계산할 때 (x, y) E[Px - Wx - 1, Px +Wx + 1] X [Py - Wy - 1, Py + Wy + 1] 근처에 I^L(x, y)의 밝기 값을 가져야합니다 (방정식 19, 20).

Of course, the coordinates of the central point p = [Px Py]^T are not garanteed to be integers. Call pxo and pyo the integer parts of px and py. Then we may write:
물론 중심점 p = [Px Py] ^ T의 좌표는 정수가 될 수 있습니다. pyo와 pyc는 px와 py의 정수 부분을 호출하십시오. 그런 다음 우리는 다음과 같이 쓸 수 있습니다.

$$p_x = p_{x_o} + p_{x_\alpha}, \quad (36)$$
$$p_y = p_{y_o} + p_{y_\alpha}, \quad (37)$$

where px and py are the associated reminder values between 0 and 1. Therefore, in order to compute the image patch I^L(x, y) in the neighborhood (x, y) E [Px - Wx - 1, Px + Wx + 1] X [Py - Wy - 1, Py + Wy + 1] through bilinear interpolation, it is necessary to use the set of original brightness values I^L(x, y) in the integer grid patch (x, y) E[Pxo- Wx - 1, Pxo + Wx + 2]X [Pyo-Wy - 1, Pyo + !y + 2] (recall that !x and !y are integers).
여기서, px 및 py는 0과 1 사이의 연관된 리마인더 값이다. 따라서, 바이 리니어 보간을 통해 이웃 (x, y) E [Px - Wx - 1, Px + Wx + 1] X [Py - Wy - 1, Py + Wy + 1] 에서 이미지 패치를 계산하기 위해서는, 원래의 밝기 값의 정수 그리드 패치 (x, y)의 I^L(x, y) E [Pxo-Wx-1, Pxo + Wx + 2] X [Pyo-Wy-1, Pyo +W y + 2] (x와 y는 정수 ) 세트를 사용할 필요가있다

A similar situation occurs when computing the image difference δIk(x, y) in the neigborhood (x, y) ∈ [px −ωx, px + ωx] × [py − ωy, py + ωy] (refer to section 2.4). Indeed, in order to compute δIk(x, y), it is required to have the values J^L(x + g^ L_ x + v k−1 x , y + g^L_y + v^ k−1_ y ) for all (x, y) ∈ [px − ωx, px + ωx] × [py − ωy, py + ωy], or, in other words, the values of J L(α, β) for all (α, β) ∈ [px + g L x + v k−1 x − ωx, px + g^L_ x + v^k−1_x + ωx] × [py + g ^L_y + v^k−1_y − ωy, py + g^L_ y + v^k−1_ y + ωy]CCC. Of course, px + g^L_ x + v^k−1_x and py + g^L_ y + v^k−1_ y are not necessarily integers. Call qxo and qyo the integer parts of px + g^L_ x + v^k−1_x and py + g^L_ y +v^k−1 _y :

유사 상황은 근수 (x, y) ∈ [px-ωx, px + ωx] × [py-ωy, py + ωy] (2.4 절 참조)에서 이미지 차이 δIk (x, y)를 계산할 때 발생한다.
실제로 δIk (x, y)를 계산하기 위해서는 모든 J^L(x + g^ L_ x + v k−1 x , y + g^L_y + v^ k−1_ y )(AAA) for all (x, y) ∈ [px − ωx, px + ωx] × [py − ωy, py + ωy] 에 대해 J^L(x + g^ L_ x + v k−1 x , y + g^L_y + v^ k−1_ y ) 값을 가져야하며, 즉 모든 (α, β) ∈ [px + g L x + v k−1 x − ωx, px + g^L_ x + v^k−1_x + ωx] × [py + g ^L_y + v^k−1_y − ωy, py + g^L_ y + v^k−1_ y + ωy] 에 대해 JL (α, β)의 값을 가져야합니다.
물론, px + g ^ L_x + v ^ k-1_x와 py + g ^ L_y + v ^ k-1_y는 반드시 정수가 아닙니다. qxo와 qyo를 px + g ^ L_x + v ^ k-1_x의 정수 부분과 py + g ^ L_y + v ^ k-1_y의 정수 부분을 호출하십시오.

$$p_x + g_x^L + v_x^{k-1} = q_{x_o} + q_{x_\alpha}, \quad (38)$$
$$p_y + g_y^L + v_y^{k-1} = q_{y_o} + q_{y_\alpha}, \quad (39)$$

where qxα and qyα are the associated reminder values between 0 and 1. Then, in order to compute the image patch J L(α, β) in the neighborhood (α, β) ∈ [px+g L x +v k−1 x −ωx, px+g L x +v k−1 x +ωx]×[py+g L y +v k−1 y −ωy, py+g L y +v k−1 y +ωy] , it is necessary to use the set of original brightness values J L(α, β)in the integer grid patch (α, β) ∈ [qxo − ωx, qxo + ωx + 1] × [qyo − ωy, qyo + ωy + 1].

여기서 qxα 및 qyα는 0과 1 사이의 관련된 미리 알림 값입니다. 그런 다음 이웃 (α, β) ∈ [px+g L x +v k−1 x −ωx, px+g L x +v k−1 x +ωx]×[py+g L y +v k−1 y −ωy, py+g L y +v k−1 y +ωy] 에서 이미지 패치 JL (α, β)를 계산하려면 정수형 그리드 패치(α, β) ∈ [qxo − ωx, qxo + ωx + 1] × [qyo − ωy, qyo + ωy + 1] 에서 원래 밝기 값 J L(α, β) 의 세트를 사용할 필요가 있습니다 .

# 2.6 Tracking features close to the boundary of the images

It is very useful to observe that it is possible to process points that are close enough to the image boundary to have a portion of their integration window outside the image.
이미지 바운더리에 충분히 근접한 지점을 처리하여 이미지의 외부에 통합 창의 일부를 갖는 것이 가능하다는 것을 관찰하는 것은 매우 유용하다.

This observation becomes very significant as the the number of pyramid levels Lm is large.
이 관찰은 피라미드 레벨의 수 Lm이 클수록 매우 중요합니다.

Indeed, if we always enforce the complete $(2\omega_x + 1) \times (2\omega_y + 1)$ window to be within the image in order to be tracked, then there is “forbidden band” of width ωx (and ωy) all around the images I L. If Lm is the height of the pyramid, this means an effective forbidden band of width 2Lm ωx (and 2Lm ωy) around the original image I. For small values of ωx, ωy and Lm, this might not constitute a significant limitation, howerver, this may become very troublesome for large integration window sizes, and more importantly, for large values of Lm. Some

numbers: ωx = ωy = 5 pixels, and Lm = 3 leads to a forbidden band of 40 pixels around the image!

사실, 우리가 추적하기 위해 이미지 내에 있어야하는 완전한 (2ωx + 1) × (2ωy + 1) 창을 강제로 수행한다면, 모든 이미지 IL 주위에 폭 ωx (및 ωy)의 "금지 된 밴드"가 있습니다 . Lm이 피라미드의 높이 인 경우, 이는 원 이미지 I 주위의 폭 2Lm ωx (및 2Lm ωy)의 유효 금지대를 의미한다. ωx, ωy 및 Lm의 작은 값에 대해 이것은 상당한 제한을 구성하지 않을 수 있지만, 이것은 큰 통합 창 크기, 더 중요한 것은 Lm 값이 클 경우 매우 번거로울 수 있습니다. 일부 숫자는 ωx = ωy = 5 픽셀이고 Lm = 3은 이미지 주위에 40 픽셀의 금지 된 밴드를 만듭니다!

In order to prevent this from happenning, we propose to keep tracking points whose integration windows partially fall outside the image (at any pyramid level). In this case, the tracking procedure reamins the same, expect that the summations appearing in all expressions (see pseudo-code of section 2.4) should be done only over the valid portion of the image neigborhood, i.e. the portion of the neighborhood that have valid entries for Ix(x, y), Iy(x, y) and δIk(x, y) 7 (see section 2.4).
이를 방지하기 위해, 통합 윈도우가 부분적으로 이미지 외부에있는 추적 점 (피라미드 레벨에서)을 유지할 것을 제안합니다.
이 경우, 추적 절차는 동일하게 진행되며, 모든 표현식 (2.4 절의 의사 코드 참조)에 나타나는 합계가 이미지 neigborhood의 유효한 부분 인 i.e에서만 수행되어야한다고 예상합니다. Ix (x, y), Iy (x, y) 및 δIk (x, y) 7 (섹션 2.4 참조)에 대한 유효 엔트리를 갖는 이웃의 부분.

Observe that doing so, the valid summation regions may vary while going through the Lucas-Kanade iteration loop (loop over the index k in the pseudo-code - Sec. 2.4).
이렇게함으로써 루아스 - 카나 데 (Lucas-Kanade) 반복 루프 (의사 코드의 색인 k에 대한 루프 - 초 2.4)를 통과하는 동안 유효한 합계 영역이 달라질 수 있음을 관찰하십시오.

Indeed, from iteration to iteration, the valid entry of the the image difference δIk(x, y) may vary as the translation vector [g L x + v k−1 x g L y + v k−1 y ] T vary.
실제로, 반복에서 반복까지, 이미지 차이 δIk (x, y)의 유효 엔트리는 변환 벡터 [gLx + vk-1 × gLy + vk-1y] T가 변함에 따라 변할 수있다.

In addition, observe that when computing the mismatch vector bk and the gradient matrix G, the summations regions must be identical (for the mathematics to remain valid).
또한 불일치 벡터 bk와 그라디언트 행렬 G를 계산할 때 합산 영역이 동일해야합니다 (수학이 유효하도록 유지되어야 함).

Therefore, in that case, the G matrix must be recomputed within the loop at each iteration. The differential patches Ix(x, y) and Iy(x, y) may however be computed once before the iteration loop
따라서이 경우 G 반복은 매 반복마다 루프 내에서 다시 계산되어야합니다. 그러나 차동 패치 Ix (x, y) 및 Iy (x, y)는 반복 루프 전에 한번 계산 될 수있다

Of course, if the point p = [px py] T (the center of the neighborhood) falls outside of the image I L, or if its corresponding tracked point [px + g L x + v k−1 x py + g L y + v k−1 y ]

falls outside of the image J L, it is reasonable to declare the point “lost”, and not pursue tracking for it.
물론, 점 P = [px py] T (이웃의 중심)가 이미지 IL의 바깥에 떨어지거나 해당 추적 점 [px + gLx + vk-1xpy + gLy + v k-1 y]가 이미지 JL의 외부로 떨어지면, "lost"점을 선언하고 추적하지 않는 것이 합리적입니다.

# 2.7 Declaring a feature “lost”

There are two cases that should give rise to a “lost” feature. The first case is very intuitive: the point falls outside of the image. We have discussed this issue in section 2.6.
"lost"기능을 야기시키는 두 가지 경우가 있습니다. 첫 번째 경우는 매우 직관적입니다.이 점은 이미지 외부에 있습니다. 이 문제는 2.6 절에서 논의했습니다.

The other case of loss is when the image patch around the tracked point varies too much between image I and image J (the point disappears due to occlusion).
다른 손실의 경우는 추적 점 주변의 이미지 패치가 이미지 I와 이미지 J 사이에서 너무 많이 변하는 경우입니다 (점은 교합으로 사라집니다).

Observe that this condition is a lot more challenging to quantify precisely. For example, a feature point may be declared “lost” if its final cost function E(d) is larger than a threshold (see equation 1).
이 상태는 정확하게 정량화하기가 훨씬 더 어렵습니다. 예를 들어, 최종 비용 함수 E (d)가 임계 값 (식 1 참조)보다 큰 경우 특징점을 "lost"로 선언 할 수 있습니다.

A problem comes in when having decide about a threshold. It is particularly sensitive when tracking a point over many images in a complete sequence.
임계 값에 대해 결정할 때 문제가 발생합니다. 전체 시퀀스에서 많은 이미지 위에 점을 추적할 때 특히 민감합니다.

Indeed, if tracking is done based on consecutive pairs of images, the tracked image patch (used as reference) is implicitely initialized at every track.
실제로 연속 이미지의 쌍에 따라 추적이 행해지는 경우, 추적 된 이미지 패치 (참조로 사용되는)은 각 트랙에서 암시 적으로 초기화된다.

Consequently, the point may drift throughout the extended sequence even if the image difference between two consecutive images is very small.
결과적으로 두 개의 연속적인 이미지 사이의 이미지 차이가 매우 작더라도 확장 된 시퀀스 전체에서 점이 드리프트 될 수 있습니다.

This drift problem is a very classical issue when dealing with long sequences.
이 드리프트 문제는 긴 시퀀스를 처리 할 때 매우 고전적인 문제입니다.

One approach is to track feature points through a sequence while keeping a fixed reference for the appearance of the feature patch (use the first image the feature appeared). Following this technique, the quantity (d) has a lot more meaning.
한 가지 방법은 기능 패치의 모양에 대한 고정 된 참조를 유지하면서 시퀀스를 통해 특징 지점을 추적하는 것입니다 (기능이 표시된 첫 번째 이미지 사용). 이 기술에 따라, 양 (d)는 훨씬 더 의미가 있습니다.

While adopting this approach however another problem rises: a feature may be declared “lost” too quickly. One direction that we envision to answer that problem is to use affine image matching for deciding for lost track (see Shi and Tomasi in [1]).
그러나 이 방법을 채택하는 동안 또 다른 문제가 발생합니다. 즉, 기능이 너무 빨리 "lost"될 수 있습니다. 이 문제를 해결하기 위해 우리가 생각하는 한 가지 방향은 잃어버린 선로를 결정할 때 아핀 이미지 매칭을 사용하는 것입니다 ([1]의 Shi and Tomasi 참조).

The best technique known so far is to combine a traditional tracking approach presented so far (based on image pairs) to compute matching, with an affine image matching (using a unique reference for feature patch) to decide for false track.
지금까지 알려진 가장 좋은 기술은 거짓 추적을 결정하기 위해 (이미지 패치를 기반으로 한) 고유 한 참조를 사용하는 어필 이미지 매칭을 사용하여 지금까지 제시된 기존의 추적 접근 방식을 이미지 매칭에 결합하여 매칭을 계산하는 것입니다.

More information regarding this technique is presented by Shi and Tomasi in [1]. We believe that eventually, this approach should be implemented for rejection. It is worth observing that for 2D tracking itself, the standard tracking scheme presented in this report performs a lot better (more accurate) that affine tracking alone.
이 기법에 관한 더 자세한 정보는 Shi와 Tomasi가 [1]에 발표했다. 우리는 결국이 접근법을 거부해야한다고 생각합니다. 2D 추적 자체의 경우이 보고서에 제시된 표준 추적 체계가 아핀 추적만으로 훨씬 더 (더 정확하게) 수행된다는 것을 관찰 할 가치가 있습니다.

The reason is that affine tracking requires to estimate a very large number of parameters: 6 instead of 2 for the standard scheme. Many people have made that observation (see for example http://www.stanford.edu/ ssorkin/cs223b/final.html). Therefore, affine tracking should only be considered for building a reliable rejection scheme on top of the main 2D tracking engine.
그 이유는 아핀 추적이 매우 많은 수의 매개 변수를 추정해야하기 때문입니다. 즉, 표준 체계에 대해 2 대신에 6을 추정합니다. 많은 사람들이 이러한 관찰을했습니다 (예를 들어 http://www.stanford.edu/ssorkin/cs223b/ final.html 참조). 따라서, 어 파인 트래킹은 메인 2D 트래킹 엔진 위에 신뢰성있는 거부 기법을 구축하기 위해서만 고려되어야합니다.

# 3 Feature Selection

So far, we have described the tracking procedure that takes care of following a point u on an image I to another location v on another image J. However, we have not described means to select the point u on I in the first place.
지금까지 우리는 이미지 I의 점 u를 다른 이미지 J의 다른 위치 v로 따라가는 추적 절차를 설명했습니다. 그러나 처음에는 I 위에서 점 U를 선택하는 방법을 설명하지 않았습니다.

This step is called feature selection. It is very intuitive to approach the problem of feature selection once the mathematical ground for tracking is led out. Indeed, the central step of tracking is the computation of the optical flow vector η k (see pseudo-code of algorithm in section 2.4).
이 단계를 피쳐 선택이라고합니다. 트래킹을위한 수학적 기반을 이끌어 내면 기능 선택 문제에 접근하는 것은 매우 직관적입니다. 사실, 추적의 중심 단계는 옵티컬 플로우 벡터 η k의 계산입니다 (2.4 절의 알고리즘의 의사 코드 참조).

At that step, the G matrix is required to be invertible, or in other words, the minimum eigenvalue of G must be large enough (larger than a threshold). This characterizes pixels that are “easy to track”.
이 단계에서 G 행렬은 역전 될 필요가 있습니다. 즉 G의 최소 고유치는 충분히 커야합니다 (임계 값보다 커야합니다). 이것은“easy to track” 픽셀을 특징으로합니다.

Therefore, the process of selection goes as follows:
따라서 선택 과정은 다음과 같습니다.
1. Compute the G matrix and its minimum eigenvalue λm at every pixel in the image I.
1. 이미지 I의 모든 픽셀에서 G 매트릭스와 최소 고유치 λm을 계산합니다.

2. Call λmax the maximum value of λm over the whole image.
2. 전체 이미지에 대해 λmax의 최대 값을 λmax라고 부릅니다.

3. Retain the image pixels that have a λm value larger than a percentage of λmax. This percentage can be 10% or 5%.
3. λm 값이 λmax의 백분율보다 큰 이미지 픽셀을 유지합니다. 이 백분율은 10 % 또는 5 %가 될 수 있습니다.

4. From those pixels, retain the local max. pixels (a pixel is kept if its λm value is larger than that of any other pixel in its 3 × 3 neighborhood).

5. Keep the subset of those pixels so that the minimum distance between any pair of pixels is larger than a given threshold distance (e.g. 10 or 5 pixels).
4.이 픽셀들에서 로컬 최대 값을 유지하십시오. 픽셀 (λm 값이 3 × 3 근처의 다른 픽셀의 픽셀보다 큰 경우 픽셀은 유지됩니다).

5. 모든 픽셀 쌍 사이의 최소 거리가 주어진 임계 거리 (e.g. 10 또는 5 픽셀)보다 커지도록 해당 픽셀의 하위 집합을 유지합니다.

After that process, the remaining pixels are typically “good to track”. They are the selected feature points that are fed to the tracker.
이 프로세스가 끝나면 나머지 픽셀은 일반적으로 “good to track”. 추적기에 공급되는 선택된 특징점입니다.

The last step of the algorithm consisting of enforcing a minimum pairwise distance between pixels may be omited if a very large number of points can be handled by the tracking engine. It all depends on the computational performances of the feature tracker.

추적 엔진에 의해 매우 많은 수의 포인트가 처리 될 수있는 경우 픽셀 간의 최소 쌍 방향 거리를 적용하는 알고리즘의 마지막 단계는 생략 될 수 있습니다. 그것은 모두 피쳐 트래커의 계산 성능에 달려 있습니다.

Finally, it is important to observe that it is not necessary to take a very large integration window for feature selection (in order to compute the G matrix). In fact, a 3 × 3 window is sufficient ωx = ωx = 1 for selection, and should be used. For tracking, this window size (3 × 3) is typically too small (see section 1).
마지막으로, (G 행렬을 계산하기 위해) 피쳐 선택을위한 매우 큰 통합 윈도우를 취할 필요는 없다는 것을 관찰하는 것이 중요합니다. 사실, 3 × 3 창은 선택을 위해 ωx = ωx = 1로 충분하므로 사용해야합니다. 추적을 위해이 창 크기 (3x3)는 일반적으로 너무 작습니다 (섹션 1 참조).

References

[1] Jianbo Shi and Carlo Tomasi, “Good features to track”, Proc. IEEE Comput. Soc. Conf. Comput. Vision and Pattern Recogn., pages 593–600, 1994.