Lucas-Kanade Without Iterative Warping

Alex Rav-Acha
School of Computer Science and Engineering
The Hebrew University of Jerusalem
91904 Jerusalem, Israel
E-Mail: alexis@cs.huji.ac.il

Abstract

A significant part of the parametric motion computation methods are based on the algorithm proposed by Lucas&Kanade [8]. We describe an algorithmic acceleration which can be used to speedup most of those methods, while preserving their accuracy. The method is also generalized to an online multi-image alignment framework which increases the accuracy and robustness of the motion computations with no additional computational cost.

1 Introduction

Although many robust methods exist for motion computation, the direct motion analysis initially proposed by Lucas & Kanade [8], and later adapted for parametric motion computations by [2, 6] is still of wide use [1]. We will denote this method as 'LK' although they addressed in the original paper to the problem of optical flow, while we focus more in this paper on parametric motion. The popularity of this method is due to its stability, simplicity and robustness not only to outliers but also to inaccuracies in the image model (such as changes in the illumination, lens distortion, etc'). For example, it was shown in [9], that when computing a parametric motion of pure translation between two images, it converges to the dominant motion even when there are multiple motions. The advantage of direct methods are mostly expressed in difficult scenes where no clear features exist. In this work we present an algorithmic acceleration which can speedup the motion computations while preserving their accuracy. We also show a generalization of the image alignment to a multi-frame alignment, which increases the robustness and accuracy of the motion computations while preserving both the direct characteristic of the LK method and its efficiency. The ideas shown in this paper can be used for both parametric motion computations and for computing optical flow.

In section 2 we describe how to accelerate parametric motion computations that are based on the LK by avoiding the additional computation needed for the iterative back-warping of the image. In Section 3 we introduce the multi-frame alignment framework, and show how to accelerate it in Section 4. Possible improvements are introduced in Section 5, including an automatic frame weighting and a generalized multi-resolution framework.

2 Accelerating the computations of the LK

2.1 Accelerating Sub-Pixel Translations

Let I_1 and I_2 be two images, and let the transformation between the two images be a pure translation (u, v). As was described in [2], an iteration of the LK method for the case of pure translation is done by solving the equation set $A\begin{bmatrix} u \\ v \end{bmatrix} = b$, where A is the LK matrix given by:

$$A = \begin{bmatrix} \sum\limits_{x,y\in R} I_x(x,y)^2 & \sum\limits_{x,y\in R} I_x(x,y)I_y(x,y) \\ \sum\limits_{x,y\in R} I_x(x,y)I_y(x,y) & \sum\limits_{x,y\in R} I_y(x,y)^2 \end{bmatrix} \tag{1}$$

and b is given by:

$$b = \begin{bmatrix} -\sum\limits_{x,y\in R} I_x(x,y)I_t(x,y) \\ -\sum\limits_{x,y\in R} I_y(x,y)I_t(x,y) \end{bmatrix} \tag{2}$$

Where I_x and I_y are the image derivatives, approximated from the intensities of the image I_2 using derivative kernels, and I_t denotes the derivative in time, and is practically approximated by the difference image $I_2 - I_1$. The summation in the above expressions is over the region of analysis R. This region includes most of the image for many direct methods[2, 6], and a window around a certain pixel for local motion computations[8]. Without loss of generality each pixel received the same weight, but a weighted sum can be used exactly in the same way.

For simplicity of presentation, we sometimes omit the indices (x, y) from the summation expressions in the following equations. In the iterative scheme, given an estimation of the image translation (u, v), the image I_1 is warped (using a back-warping) according to the current motion parameters, and the warped image is used in the next iteration, until convergence.

The LK matrix A does not change per each iterations, and thus can be computed only once (this scheme was described in details by [1]). Whereas the free term does vary per iteration:

$$b^{(n)} = \begin{bmatrix} -\sum I_x I_t \\ -\sum I_y I_t \end{bmatrix} = \begin{bmatrix} -\sum I_x (I_2 - I_1^{(n-1)}) \\ -\sum I_y (I_2 - I_1^{(n-1)}) \end{bmatrix} \quad (3)$$

where $I_1^{(n-1)}$ equals to I_1 after warping according to the current estimated motion between I_1 and I_2. Assuming that the motion is smaller than a pixel (and thus it can be done using interpolation), and under the model of a pure translation, we can write: $I_1^{n-1} = I_1 * m$, where m is a convolver whose size depends on the interpolation scheme. (For example, in bilinear interpolation m is a 2x2 kernel). In this case, we can write:

$$\begin{aligned} \sum_{x,y} I_x(x,y) I_1^{(n-1)}(x,y) &= \sum_{x,y} I_x(x,y)(I_1(x,y) * m) \quad (4) \\ &= \sum_{0 \le i,j \le 1} m_{ij} \sum_{x,y} I_x(x,y)(I_1(x-i, y-j)) \\ &= \sum_{0 \le i,j \le 1} m_{ij} s^1_{i,j} \end{aligned}$$

where $\{s^1_{i,j}\}$ are all scalers. The second component of b can be manipulated in a similar way. Therefore, assuming a sub-pixel translation, the only terms that are changed per iteration are the values of m, while the rest of the terms can be computed in a pre-processing, leaving only a few operations per each iteration of the LK: A number which depends only on the size of the interpolation kernel (which is usually very small) but is independent in the number of pixels in the region of analysis.

The number of operations needed for the pre-processing is similar to the number of operations for each iterations of the LK (only the order of the operations is changed), and the extra number of operations done in each iteration is $O(1)$. [1]

2.2 Accelerating Large Translations

When the relative motion between the two input images is large, there are two changes in the registration scheme that should be addressed:

- The image warping after each iterations can no longer be described as a combination of constant numbers (4 numbers in the case of bilinear interpolation) since the pixel-wise location of the interpolation varies from iteration to iteration.

- For large motions, a multi-resolution framework is usually used. That it, Gaussian pyramids are constructed for both images, and the motion parameters estimated by the low-resolution level are used as an initial estimation for the finer level [2].

It should be noted that when using the multi-resolution scheme, the residual translation in the finer levels is almost always sub-pixel, as the pixel-wise translation is expected to be recovered in the lowest resolution (and enhanced by each level). Nevertheless, we will not like to rely on this observation. Instead, we prepare two tables $T_1(u, v)$ for $\sum I_x I_t$ and $T_2(u, v)$ for $\sum I_y I_t$ (The tables are functions of (u, v) as the two terms are summed over all the pixels in the region of analysis). The number of entries in the tables is determined by the number of possible (pixel-wise) translations. The tables T_1 and T_2 are initialized by null values. Each time we need a value in one of the tables, we compute the relevant term only if its value in the table is null. In this way we guarantee correctness while saving computations whenever possible.(The size of the tables does not influence on the computational complexity, as we access their cells only on demand). Note, for example, that if the translation was a pixel and a fraction, we can still save computations as only some of the terms are new (for bilinear interpolation: half are new, for bi-cubic interpolation only 3 terms out of 9 are new).

Regarding to the multi-resolution framework, we simply construct a table for each level, which means that the minimal number of "actual" iterations done in the proposed method equals to the number of levels in the pyramid (of course, most of the computations are done in the finest resolution level).

2.3 Extensions to other Types of Motions

The acceleration described above cannot be simply generalized to different types of motions, other than pure translation, since general motions can not be modeled by a convolution with a kernel. However, it is very common to approximate the image motion in local regions by a translation. This approximation holds for any smooth image motion. This approximation can be utilized to accelerate general motions by dividing the image into a set of windows, assuming each window have a uniform translation. For example, assume an Affine motion model and let (x, y) belong to a windows whose center is located at image coordinates (x_c, y_c), the image shift $(p(x, y), q(x, y))$ will be

[1] For some platforms, this change in the order of summation is has additional advantages: Instead of working with interpolated values, we process image values (I_1, I_x, I_y), allowing a better precision.

given by:

$$\begin{bmatrix} p(x,y) \\ q(x,y) \end{bmatrix} = \begin{bmatrix} ax_c + by_c + c \\ dx_c + ey_c + f \end{bmatrix} , \quad (5)$$

This motion model has 6 unknowns, and its deviation from the real Affine model is given by

$$\begin{bmatrix} e_x \\ e_y \end{bmatrix} = \begin{bmatrix} a(x_c - x) + b(y_c - y) \\ d(x_c - x) + e(y_c - y) \end{bmatrix} \quad (6)$$

which is usually very small as $a, \ldots, d \approx 0$ and $(x - x_c, y - y_c)$ are only few pixels. (Otherwise, one of the images can be warped using a real Affine transformation after several LK iterations). Since each window is accelerated separately, a significant speedup can be achieved as long as the window size is large compared to the size of the interpolation kernel.

2.4 Numerical Evaluation

The algorithm does not guarantee to reduce the number of iterations to a single iteration if the image motion is larger than a pixel. However, we can hope that practically, when using a multi-resolution framework, the residual motion in the highest-resolution level will be less than a pixel, resulting in a single iteration in this level (which is the most expensive). Note that the proposed acceleration is also valid when the motion is computed only using lower-resolution levels, as is sometimes done in efficient implementations of the LK. In this case, the same speedup is achieved, only that to compute the actual cost of the registration, we consider the size of the highest-resolution level used in the motion analysis instead of the size of the original image.

Simulation with pairs of images of different types showed that for the proposed method only a single iteration was needed in all the image resolutions, accept for the lowest one. Therefore, the speedup achieved with the proposed method was high especially for difficult scenes where the traditional LK converged more slowly than the usual. The speedup in the number of iterations was 3-10 (In many implementations the number of iterations is set to be constant), and the speedup in the total running time was ranged from 2 to 4. The total running time includes the computation time of Gaussian pyramids and the computation of image derivatives. For many applications, these computations are done anyway, making the number of LK iterations be the main measure for the speedup. [2]

[2]We have done the analysis for PCs, which has a very fast memory. In other platform (like DSP cards) the bottle-neck of the computations are usually the number of passes over the image, signifi cantly increasing the benefi t of using our method.

3 Multi-image Online Alignment

In the previous section we described a method for accelerating the LK method. In this section we address the problem of increasing the robustness and stability of the method without paying for it in complexity. For this purpose we introduce a new computation scheme, in which each frame is aligned to several preceding frames and not only to the previous one. Unlike local to global methods [12, 11], the motion computations are done only once per each frame, using the same framework as the traditional LK and avoiding exhaustive (and hard to implement) non-linear optimization (such as bundle adjustment [14]). In addition, we exploit the "true" multi-frame information as we use the gray-values of earlier frame, and not only the transformations computed for those frames. (in a similar way to [10].) The proposed scheme consists of the following steps (to be elaborated later):

1. Select a frame I, and initial from it a set $S = \{I\}$. Set all the pixels in this frame to be valid pixels. (The first frame can be selected arbitrarily to be the first frame in the sequence).

2. Compute motion parameters by aligning a new frame to the existing set S. For each frame in the set S, use only its valid pixels.

3. Mask out pixels that were miss-aligned by the registration, for the future computations. (Mark this pixels as non valid).

4. Add the registered frame to the set S, together with the corresponding validity mask.

5. Return to 2. Repeat until reaching the last frame of the sequence.

In the rest of the section we describe in details the different stages of the multi-image alignment, and in the next section we show how it can be done without increasing the computational complexity.

3.1 Image Alignment Stage

Assume that all the images $I_0 \ldots I_{n-1}$ have already been aligned and let the n^{th} frame be the new frame being aligned. To compute the motion parameters of the new frame, we minimize the error function: (For a multi-image registration, we do not use the term I_t but explicitly write the differences between pairs of images).

$$Err(p,q) = \sum_{k<n} w_k^n \cdot \sum_{x,y} M_k \cdot (p\frac{\partial I_k}{\partial x} + q\frac{\partial I_k}{\partial y} + I_k - I_n)^2, \quad (7)$$

where $M(x,y,k)$ is a validity mask (whose computation is described in the next section) and (p,q) are the displacement of each pixel. For the pure translation case, they are given by:

$$\begin{aligned} p(x,y,n) &= u_k - u_n \\ q(x,y,n) &= v_k - v_n. \end{aligned} \quad (8)$$

Other motion models can also be used. For example, in the case of affine motion model:

$$\begin{bmatrix} p \\ q \\ 1 \end{bmatrix} = \begin{bmatrix} a_n & b_n & c_n \\ d_n & e_n & f_n \\ 0 & 0 & 1 \end{bmatrix}^{-1} \cdot \begin{bmatrix} a_k & b_k & c_k \\ d_k & e_k & f_k \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \quad (9)$$

In either cases, the only unknowns are the motion parameters of the current image (i.e - (u_n, v_n) for pure translation and $(a_n \dots f_n)$ for affine motion model). Therefore the terms p and q are linear in the unknowns, and the error function can be minimized by solving a set of linear equations similar to the traditional parametric LK [2].

Note the use of the derivatives $\frac{\partial I_k}{\partial x}$ and $\frac{\partial I_k}{\partial y}$ which are estimated from each image I_k rather than from the new image I_n. This notation fits well to more general motion models, such as ego-motion or optical flow computations, but the inverse notation can also be used (That is, using the new image as a reference).

The coefficients w_k^n are also used to weight the importance of each frame in the alignment. For example, the weights should probably decrease for images which are sequentially far away from the aligned image.

3.2 Computing the Validity Mask

To increase robustness, we add a step of validation which masks pixels were the alignment is not good. These pixel will not be used for future computations. Note, however that in our method we do not repeat the registration of the current image using only the valid pixels, but only use this validation for future registrations. This scheme is more efficient, as each image is aligned only once.

In the validation stage a mask is computed, which measures for each pixel the SSD error between a window around the pixel, and the corresponding window in the previous frame. This score is normalized with the gradient of that window. In our implementation, we have used a binary mask by thresholding the score:

$$M(x,y,n) = \begin{cases} 1 & if \frac{\sum\limits_{W_{x,y}} (I_n - I_{n-1})^2}{\sum\limits_{w_{x,y}} I_x^2 + I_y^2} < r \\ 0 & otherwise, \end{cases} \quad (10)$$

where the summation is over a window $W_{x,y}$ around (x,y), and r is a threshold (We usually used $r = 1$). This is a conservative scheme to mask out pixels in which the residual energy will likely to bias the registration. The mask $M_n(x,y) = M(x,y,n)$ is used to mask pixels in I_n in all the future alignment in which I_n takes place.

4 Accelerating the Multi-image Alignment

In this section we describe how to speedup the multi-image motion computations. In the first part, we describe how the acceleration method described in Section 2 can be incorporated into the multi-image framework from Section 3. In the second part, we use a recursive update scheme to do the multi-image alignment without actually computing the terms related to all the previous frames for each alignment of a new frame.

4.1 Incorporating the Single-Frame Acceleration into the Multi-image Framework

For clarity of presentation, we focus on the pure translational case with sub-pixel motion. The generalization to other cases is identical to the single-image registration. The the case of pure translation, taking the derivatives of Eq. 7 with respect to u and v and setting them to zero yields the linear set of equations $A \begin{bmatrix} u \\ v \end{bmatrix} = b$ where:

$$A = \sum_{k<n} w_k^n A_k \ , \quad b = \sum_{k<n} w_k^n b_k. \quad (11)$$

A_k is given by:

$$A_k = \begin{bmatrix} \sum\limits_{x,y} M_k \frac{\partial I_k}{\partial x}^2 & \sum\limits_{x,y} M_k \frac{\partial I_k}{\partial x} \frac{\partial I_k}{\partial y} \\ \sum\limits_{x,y} M_k \frac{\partial I_k}{\partial x} \frac{\partial I_k}{\partial y} & \sum\limits_{x,y} M_k \frac{\partial I_k}{\partial y}^2 \end{bmatrix} \quad (12)$$

and b_k is given by:

$$b_k = -\begin{bmatrix} \sum\limits_{x,y} M_k \frac{\partial I_k}{\partial x}(I_k - I_n) \\ \sum\limits_{x,y} M_k \frac{\partial I_k}{\partial y}(I_k - I_n) \end{bmatrix} - A_k \begin{bmatrix} u_k \\ v_k \end{bmatrix} \quad (13)$$

(The right term of b_k does not appear in the single-frame LK, as we usually solve for the residual translation between the two frames. Here, we need to compensate for the different (but known) motion of each image (u_k, v_k).

The 2x2 LK matrices A_k are computed only once per frame. This means that the computation complexity of the LK matrix for the multi-frame case is similar to the single-frame one. As a result, the computation of the free term b is more dominant in the total computational cost, making the proposed acceleration even more important.

Looking at the expression for b, it is evident that the same idea which was used to accelerate the LK computations for the single-frame case can also be used to accelerate the multi-frame case. By writing b as:

$$b = r^1 + \begin{bmatrix} < r^2, I_n > \\ < r^3, I_n > \end{bmatrix} \quad (14)$$

(r^1 is a 2x1 vector while r^2, r^3 are matrices), we can use the same manipulation that was used in Eq. 4 to pass over the images only once, and to update the equation set in $O(1)$ for each iteration.

Similar to the single-frame algorithm, one may use a multi-resolution framework. In this case, a Gaussian pyramid should be computed once for each frame of the sequence for efficiency. Note that as stated before, the same acceleration can be applied in a similar way to more complex motion models.

4.2 A Multi-frame Registration in the Complexity of A Single-Frame

For some natural choices of a frame-weighting scheme for the alignment, additional acceleration can be performed which further reduces the computational cost of the multi-frame case to be equal to a single-frame case regardless of the number of frames used for the alignment. By letting the weights w_k^n to be uniform (that is $w_k^n = 1$ for $n - T < k < n$ and zero elsewhere) or geometrically decreasing ($w_k^{n-1} = q \cdot w_k^n$) we can use an incremental computation to perform the multi-frame registration in the complexity of a single-frame one. The logic behind using a geometrically decreasing weighting scheme is that after compensating for the image motion, closer frames in the sequence tend to be more similar.

To apply the accumulated computations, we must align each new frame to the reference frame. In the case of large motions (more than a pixel), two technical issues must be considered:

- The image should be aligned only according to the sub-pixel part of the motion. The pixel-wise part can be saved and used virtually. (Otherwise, one should have saved panoramic views for each frame, which is not practical).

- For image boundaries (regions which are visible only in the new frame) the accumulated terms are set to zero, so only the values of the new image are used.

In this section we describe in details the incremental computations for the case of geometrically decreasing weights, but the case of a uniform weighting scheme is very similar. The use of an incremental computation for the computation of the LK matrix A is straight-forward from Eq. 11. By writing Eqs. 11 and 12 in the following form:

$$A = \sum_{x,y} \begin{bmatrix} \sum\limits_{k<n} M_k \frac{\partial I_k}{\partial x}^2 & \sum\limits_{k<n} M_k \frac{\partial I_k}{\partial x}\frac{\partial I_k}{\partial y} \\ \sum\limits_{k<n} M_k \frac{\partial I_k}{\partial x}\frac{\partial I_k}{\partial y} & \sum\limits_{k<n} M_k \frac{\partial I_k}{\partial y}^2 \end{bmatrix} , \quad (15)$$

each of the terms in the matrix can be computed recursively as an accumulated sum. For example -

$$\sum_{k<n} M_k \frac{\partial I_k}{\partial x}^2 = q \cdot \sum_{k<n-1} M_k \frac{\partial I_k}{\partial x}^2 + (1-q) \cdot M_n \frac{\partial I_n}{\partial x}^2 \quad (16)$$

Accelerating the computation of b is done in a similar way. Again, we rewrite Eqs. 11 and 13 as following:

$$b = -\sum_{x,y} \begin{bmatrix} \sum\limits_{k<n} M_k \frac{\partial I_k}{\partial x} I_k - (\sum\limits_{k<n} M_k \frac{\partial I_k}{\partial x}) I_n \\ \sum\limits_{k<n} M_k \frac{\partial I_k}{\partial y} I_k - (\sum\limits_{k<n} M_k \frac{\partial I_k}{\partial y}) I_n \end{bmatrix} . \quad (17)$$

(Recall that we align each new frame to the reference frame, and therefore we can assume here without loss of generality that u_k and v_k are zero for all previous frames.) Similar to A, b can be computed recursively by saving the accumulated sums of four images, and updating them with terms that are computed from the new frame.

To conclude, in order to compute the linear system recursively for the geometrically decreasing weighting scheme, one should save 7 accumulated images (three for the computation of A and four for b). These images are updated in each alignment, and no further information should be saved from the past sequence. By combining the recursive computation with the previous acceleration, the multi-frame implementation of the parametric alignment can be done online in a computational complexity of $O(N)$ instead of the naive cost which is $O(N * T * L)$, where N is the number of pixels in each image, T is the number of frames used for the alignment, and L is the number of iterations.

5 Additional Improvements

We describe two additional variations of the method that can improve the accuracy of the motion computations. The first paragraph deals with automatically setting the weights of each frame in the alignment. Unlike previous improvements, using a different weights for each frame requires additional computational complexity, as the acceleration described in Section 4.2 can be applied only for special weighting schemes. The second paragraph generalize the multi-image alignment to a multi-image-multi-resolution alignment, in which different images and different resolutions are used simultaneously for the alignment.

5.1 An Automatic Scheme for Setting the Frame-Weights

In the preceding sections we have suggested to use a set of frames to align each new frame instead of aligning it only to the last frame. Experimental evaluation of this idea showed that using a small number of frames always increased the stability of the registration, but sometimes using more frames gave even better results. On the other hand, it is logical that there is a limit on the number of frames from which it is no longer worthwhile to add more frames to the alignment process. This limit can be determined either due to computational considerations, or according to the temporal variability of the sequence (For example - changes in the illumination of the scene).

It may be of a large benefit if one could automatically determine the number of frames that should be used in the alignment. A more general scheme would be to give a weight for each frame, which describes the influence of this frame on the alignment. Given the weights, the number of frames in each alignment can be determined with respect to the specific application: In applications which demand fast computations, frames with low weights may be omitted from the computations.

As the proposed registration scheme is online, it would be of a great advantage to learn the weights that will be used for aligning the n^{th} frame from preceding alignments. Using a predefined set of weights may not be appropriate, as it must depend on the dynamic properties of the scene. To check whether it is possible to do so, we have performed the following test: We have aligned several video sequences, and for each frame of each sequence we computed its SSD (Sum of Squared Differences) to its K preceding frames (We used 5 frames). From the SSD we have computed alignment scores:

$$w_i = \frac{N}{SSD}$$

where N is the number of overlapping pixels between the two corresponding frames. Then, the K scores where normalized to sum to one. Some vectors of normalized scores are shown below (for two different sequences). The i^{th} row describes the different weights in the alignment of the i^{th} frame (starting from the weight of the earliest frame):

sequence1:
0.161867 , 0.17076 , 0.185054 , 0.191662 , 0.290656
0.171825 , 0.171157 , 0.169407 , 0.200928 , 0.286683
0.172707 , 0.167018 , 0.171996 , 0.198316 , 0.289963
0.17141 , 0.172195 , 0.177726 , 0.19796 , 0.280709
0.172193 , 0.176888 , 0.184121 , 0.195573 , 0.271225
0.173469 , 0.173244 , 0.179744 , 0.190237 , 0.283306

sequence2:
0.0942116 , 0.134753 , 0.161537 , 0.211008 , 0.39849
0.109341 , 0.113662 , 0.154806 , 0.191648 , 0.430543
0.109542 , 0.137191 , 0.15499 , 0.23322 , 0.365057
0.114045 , 0.132609 , 0.166553 , 0.220887 , 0.365905
0.109203 , 0.142641 , 0.161403 , 0.192277 , 0.394475
0.121274 , 0.141872 , 0.16762 , 0.208983 , 0.360251
0.121249 , 0.134765 , 0.161818 , 0.208292 , 0.373876

These vectors look fairly consistent along the sequence (but differ from sequence to sequence). Intuitively, this can be explained by the dynamical nature of the scene - There are scenes which have a large temporal variance (such as crowd), while other scenes have low temporal variance (Such as static scenes). These results suggest that we one can compute the frames weights from the last alignment step (after the motions have been recovered), and use it (shifted) in the next step for aligning the new frame.

5.2 Simultaneous Alignment of Different Resolutions

When using a multiresolution pyramid, the motion parameters are usually computed first using the lowest resolution level, and then the resulting motion parameters are used as an initial estimate to the higher resolution level. A better exploitation of the image information can be obtained by using all the levels simultaneously. Similar schemes were proposed for non-parametric motion computations[3]. In general, we minimize the error function:

$$Err(p,q) = \sum_k w_k^n \sum_{x,y} (pI_x^k + qI_y^k + I_t^k)^2 \tag{18}$$

Here, the upper index denotes the level of the pyramid. ($k = K-1$ denotes the lowest resolution level, and $k = 0$ denotes the highest resolution level which is the original image). p, q are the displacement functions of each pixel in each level. For example, for pure translational model:

$$\begin{array}{l} p(x,y,k) = \frac{u}{2^k} \\ q(x,y,k) = \frac{v}{2^k}. \end{array} \tag{19}$$

To handle large motions, we start by setting $w_k^n = 0$, $k < K-1$, and we use the result of the computation to initialize a finer level, in which w_{K-1}, $w_{K-2} \leq 0$ and so on. Note that the in this multi-resolution alignment does not increase the computational complexity, as we use terms that have already been computed. For example, in the pure translation case, the upper left term of the LK matrix is given by: $\sum\limits_k w_k \sum\limits_{x,y} \frac{\partial I_k}{\partial x}^2$. This means that for each low-resolution level k, one can use the term $\sum\limits_{x,y} \frac{\partial I_k}{\partial x}^2$ which has already been computed in previous iterations, only multiplying it with the weight w_k.

An interesting question that is still open is how can we determine the weights of the different levels automatically. There are several options:

1. Learning the weights per each input sequence.

2. Choosing the weights that maximize some score per frame. (F.g - Minimizing the uncertainty of the alignment which can be described by the covariance matrix).

3. Using geometrically decreasing weights, so that in the finest resolution step, half of the weights will come from the finest resolution level, and half from upper resolution levels.

This question is strongly related to the automatic computation of the frame's weights described in Section 5.1.

6 Appendix1: Fast Stereo Computations

Some methods use the LK method to compute sub-pixel disparity using two or more images (with a known camera motion) [5]. The process can be formulated in the following way: Let k be the index of the frame for which we estimate the disparity. and let $T_{n,k} = (u_n - u_k, v_n - v_k)^t$ be the translation of the optical center of the camera between the n^{th} and the k^{th} frames. Here we assume that the camera can only translate on a planar surface, which fits to many camera settings involving stereo computations (Light Field [4, 7] or Horizontal Parallax [13]).

Following [5], The disparity parameter $d = d(x, y, k)$ in the image location (x, y) minimizes the following error function:

$$Err(d) = \sum_{n \neq k} w_n^d \cdot \sum_{x,y \in W} (d \cdot \bigtriangledown I^t \cdot T_{n,k} + I_n - I_k)^2, \quad (20)$$

Where $\bigtriangledown I$ is the gradient of the image I_k in the location (x, y), and W is a window around (x, y). The minimum of this quadratic equation is obtained by:

$$d = -\frac{\sum_{n \neq k} w_n^d \cdot \sum_{x,y} \bigtriangledown I^t \cdot T_{n,k} \cdot (I_n(x,y) - I_k(x,y))}{\sum_{n \neq k} w_n^d \cdot \sum_{x,y} (\bigtriangledown I^t \cdot T)^2} \quad (21)$$

The weights w_n^d determine the influence of each frame on the disparity estimation.

For each window in I_k, the computation described above is repeated iteratively until convergence, where in each iteration, the relevant regions in all the frames $\{I_n\}$ with $w_n^d \neq 0$ are warped back towards I_k according to $T_{n,k}$ and the current estimate of *d*. Similar to the case of pure translation, the only terms that are changed from iteration to iteration are the warped image of I_n. We can therefore use the same algorithm as described in Section 2 to accelerate the computations, and compute the disparity in a very small number of iterations. Note, that similar to the registration, the computations can be further accelerated using image pyramids.

7 Appendix2: Computing Optical Flow

In this paper we focused on the computation of parametric motion, such as image shift, or Affine motion. The same ideas (both the multi-frame alignment and the accelerations) can be applied also for the computation of optical flow with the Lucas-Kanade method [8]. In this case, the translation of each pixel in estimated using a window around this pixel which is assumed to translate uniformly. The translation of each window can be robustly and efficiently be calculated using the method described in this paper (but instead of summing over the entire image, we sum only over the pixels in the window).

References

[1] Simon Baker and Iain Matthews. Lucas-kanade 20 years on: A unifying framework. *International Journal of Computer Vision*, 56(3):221–255, 2004.

[2] J.R. Bergen, P. Anandan, K.J. Hanna, and R. Hingorani. Hierarchical model-based motion estimation. In *ECCV'92*, pages 237–252, Italy, May 1992.

[3] W.T. Freeman, E.C. Pasztor, and O.T. Carmichael. Learning low-level vision. In *ICCV*, pages 1182–1189, 1999.

[4] Steven J. Gortler, Radek Grzeszczuk, Richard Szeliski, and Michael F. Cohen. The lumigraph. *SIGGRAPH*, 30:43–54, 1996.

[5] M. Irani, P. Anandan, and M. Cohen. Direct recovery of planar-parallax from multiple frames. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 24(11):1528–1534, November 2002.

[6] Michal Irani and Shmuel Peleg. Motion analysis for image enhancement: Resolution, occlusion, and transparency. *J. on Visual Communications and Image Representation*, 4(4):324–335, December 1993.

[7] Marc Levoy and Pat Hanrahan. Light field rendering. *SIGGRAPH*, 30:31–42, 1996.

[8] B. Lucas and T. Kanade. An iterative image registration technique with an application to stereo vision. *IJCAI*, pages 674–679, 1981.

[9] Burt P.J., Hingorani R., and Kolczynski R.J. Mechanisms for isolating component patterns in the sequential analysis of multiple motion. In *Visual Motion, Proceedings of the IEEE Workshop*, pages 187–193, October 1991.

[10] H.S. Sawhney and R. Kumar. True multi-image alignment and its application to mosaicing and lens

distortion correction. *PAMI*, 21(3):245–243, March 1999.

[11] S. H. Sawhney, S. Hsu, and R. Kumar. Robust video mosaicing through topology inference and local to global alignment. In *ECCV*, pages 103–119, 1998.

[12] H. Shum and R. Szeliski. Construction and refinement of panoramic mosaics with global and local alignment. In *ICCV '98*, pages 953–958, Washington, DC, USA, 1998.

[13] Richard Szeliski and Ramin Zabih. An experimental comparison of stereo algorithms. In *Vision Algorithms: Theory and Practice Workshop,*, volume 1883-2000, pages 1–19, September 1999.

[14] P. R. Wolf. *Elements of photogrammetry*. McGraw-Hill, New York, 1974.