

Robust L1-Optimal Camera Paths for Video Stabilization and Retargeting

Matthias Grundmann and Vivek Kwatra

Introduction

In some examples described, systems and methods for providing an optimal stable version of a casually shot video w.r.t. cinematographic inspired criteria is provided. A casually shot video is usually obtained by using handheld devices, such as mobile phones or portable camcorders using little or no stabilization equipment, such as tripods, camera dollies or steady-cams as frequently employed by professional cinematographers. A casually shot video may include low-frequency distortions that occur during handheld panning shots or videos shot by a walking person. In some examples, methods described below may produce an optimal stable video from the original in a post-process.

In general, post-process video stabilization includes the following three main operations:

1. Estimate the original (probably shaky) motion of the camera, i.e. the original camera path
2. Estimate a new steady/smooth camera path
3. Recast the original video from the smooth camera path obtained in step b.

System overview

The system follows the three main operations mentioned in the introduction. First, the original camera path is estimated by:

1. Extracting trackable features (Harris corners) in each frame
2. We perform *local* outlier rejection (to remove spurious matches that would distort our motion estimation) as opposed to *global* outlier rejection to account for multiple independent motion layers.
3. Figure-ground segmentation removes further features from moving objects in the foreground to estimate the pre-dominant background motion.
4. We fit several 2D linear motion models (translation, similarity, affine) to the tracked features describing the motion of the camera between two frames.
5. The motion models are transformed to a common coordinate system and are concatenated, yielding the original camera path.

After the original camera path has been estimated, we optimize for the optimal smooth L1-camera path of the crop window constrained to lie within the original frame. Note, that this poses a constrained optimization problem, which is departure from previous approaches that optimize for the optimal path first and then constrain it to be within the frame-bounds or try to fill in out-of-bound areas.

Finally, we apply the difference transform between original and optimal smooth camera path to the virtual crop window, produces a final smooth version of the original video.

Feature tracking and local outlier rejection

In some examples, feature tracking software (e.g., pyramidal Lucas-Kanade feature tracking as implemented in OpenCV) can be used to obtain a set of feature-pairs capturing the motion between two frames. Naturally, a fraction of the feature tracks may be outliers that need to be removed as they would distort the estimation of the camera motion. To account for independent moving objects, we perform local outlier rejection by leveraging a fast per-frame segmentation and imposing a local 2D translation motion model on each region by running RANSAC (with a fixed number of iterations) on each region. To reduce the overhead introduced by using per-frame segmentation we also implemented a fast draft estimation-mode that replaces the segmentation regions with square regions of identical block size (40 in our case) onto each frame. Enabling the draft mode reduces computation time by around 30 - 40%.

The locally-outlier rejected regions are then classified into independent moving foreground and static background by estimating the fundamental Matrix robustly using RANSAC from the feature tracks. Regions that adhere to the fundamental matrix constraint are labeled background, while regions that violate the constraint are labeled foreground. Note, that this part of our algorithm is not too crucial. We experimented with using it to approximate the local outlier rejection by side-stepping the use of regions obtained from segmentation and directly estimating the fundamental Matrix robustly from the feature tracks. Due to the short base-line between two neighboring frames, the fundamental matrix can be poorly conditioned and results were not convincing. Local outlier rejection by using segmentation or square regions arranged in a grid can also be used.

Camera Path Estimation

We estimate the original camera by fitting several linear motion models to the tracked feature points, resulting in a linear transform for each frame pair.

Specifically, let us denote the features in frame F as $\{x_1, \dots, x_n\}$ and the corresponding features in frame F' as $\{x'_1, \dots, x'_n\}$. We want to find a linear transform T (parametrized by p) minimizing $\min_p \sum_{i=1}^n \|T(x_i; p) - x'_i\|^2$. If T is linear and satisfies that the parametrization for $p = 0$ yields $T = 0$, Taylor expansion for p around zero gives:

$$T(x_i; p) = T(x_i; 0) + D_p T(x_i, 0) * p = D_p T(x_i, 0) * p$$

i.e., we can express the result of $T(\cdot, p)$ applied to x_i simply as matrix multiplication of the Jacobian of T w.r.t. p evaluated at x_i times the parameter vector p . Defining $D_p T(x_i, 0) = J(x_i)$ the minimization is simply

$$\min_p \sum_{i=1}^n \|J(x_i) * p - x'_i\|^2$$

which can be solved via normal equations. To avoid numerical instabilities, we scale all feature points by the inverse of the diameter of the frame-size (transform S), effectively resulting into a condition number for the system of normal equation less than 100 in our experiments. The resulting transform between two frames is then $S^{-1} * T * S$.

The system can estimate the following three linear motion models:

- translation: $T(x; t) = x + t$
- similarity: $T(x; t, a, b) = [a \ -b; b \ a] * x + t$
- affine: $T(x; t, a, b, c, d) = [a \ b; c \ d] * x + t$

The final camera path is obtained by concatenating the frame-pair transforms $G_1, G_2, \dots, G_{m-1}$, where m denotes the number of frames and $G_i (=T_i^{-1})$ the inverse of the transform between frames F_i and F_{i+1} (the coordinate transform is always the inverse of the feature point transform). Note, that the transform G_i is computed w.r.t. to the coordinate system defined by frame F_i . Therefore, we need to transform each G_i to a common coordinate system to be able to concatenate them, we arbitrary choose the coordinate system of G_1 .

Specifically the camera path $C = (C_1, \dots, C_n)$ is iteratively estimated as:

$$C_1 = G_1 \text{ and}$$

$$C_n = (C_{n-1} * G_{n-1} * C_{n-1}^{-1}) * C_{n-1} = C_{n-1} * G_{n-1}.$$

As the estimation of per-frame models can lead to an accumulation of error over time (noticeable as slight wobble in the stabilized results), we track each frames w.r.t. to its previous N frames, where N is fixed and usually between 3 - 5 (this is a speed vs. accuracy trade-off). The optimal algorithm could estimate all parameters for all frames jointly, similar to Bundle Adjustment. However, the concatenation of parameters is not linear anymore, requiring a more complex non-linear optimization method which is not fast enough for real-time performance. We therefore opt for a simpler linear update scheme, here described, for example, for $N = 3$ and the first three frames. From the tracking we obtain three transforms: $T_{F_1 \rightarrow F_2}$, $T_{F_1 \rightarrow F_3}$ and $T_{F_2 \rightarrow F_3}$. We simply initialize T_1 with $T_{F_1 \rightarrow F_2}$, but T_2 is estimated jointly from $T_{F_2 \rightarrow F_3}$ and $T_{F_1 \rightarrow F_3} * T_1^{-1}$. The generalization to higher of numbers of tracked frames N is straight forward.

The optimal L1 camera path

From a cinematographic standpoint, a pleasant viewing experience is conveyed by the use of static cameras, panning ones mounted on tripods and cameras placed onto a dolly. Changes between these shot types can be obtained by the introduction of a cut or smooth transitions using ease-in and out.

In some examples, we want that the optimal camera path adheres to the above mentioned cinematographic properties, but choose not to introduce additional cuts beyond the ones already contained in the original video. Our optimal camera path P can be partitioned into three segments, where only one is present at each time t :

1. A constant path, representing a static camera, i.e. $DP(t) = 0$
2. A path of constant velocity, representing a panning or a dolly shot, i.e. $D(DP(t)) = 0$
3. A path of constant acceleration, representing the ease-in and out transition between static and panning cameras. Thus, $D(D(DP(t))) = 0$.

In some examples, while a path of constant velocity is also a path of constant acceleration, but not vice versa, one might think that a path exclusively made from segments that are constant or have constant velocity is optimal as well by our definition. However, in this case, the connection of two different segments may have essentially infinite acceleration, which can be noticeable as a sudden jerk in the stabilized video.

In some examples, the so far described optimal camera path formulation can be cast as an L1 optimization problem, with imposed constraints to assure to the validity of the crop window. Given a continuous but shaky camera path C , we express the desired smooth path as $P(t) = C(t) * B(t)$, where $B(t) = C(t)^{-1} * P(t)$ can be interpreted as the “camera stabilization transform” (or crop transform) which is applied to the crop window at each frame to obtain the final stabilized video. The optimization serves to find the optimal stable camera path $P(t)$ minimizing the objective: $a*|D(P)| + b*|DD(P)| + c*|DDD(P)|$, with linear weights a, b and c , such that the crop window transformed by $B(t)$ is contained within the frame transformed by $C(t)$ for all t . The previous “containment” constraint can be expressed linearly as a function of $P(t)$, by requiring that each corner of the transformed crop window has to be the left of each boundary-line of the transformed frame.

Linear programming: This results in a constrained L1 minimization problem which can be solved using linear programming. For example if we were only concerned with translation, i.e.

If $P(t)$ is restricted to be a translation, then the goal is to

Minimize: $a*|P(t+1) - P(t)| + b*|P(t+1) - 2*P(t) + P(t-1)| + c * |P(t+2) - 3*P(t+1) - 3*P(t) + P(t-1)|$
such that for each crop window corner w :

Let $q = (q_x, q_y) = P(t) * C(t)^{-1} * w$ be the point obtained by transforming w by $A(t)^{-1}$.

Then

$q_x \geq 0$ [constraint imposed by left line ($x=0$)]

$q_x \leq 1$ [constraint imposed by left line ($x=1$)]

$q_y \geq 0$ [constraint imposed by left line ($y=0$)]

$q_y \leq 1$ [constraint imposed by left line ($y=1$)]

[assuming the frame window is normalized between 0 and 1].

A known method of converting this into a linear program is to rewrite

Min $|x_1| + |x_2|$

as

Min $e_1 + e_2$

s.t.

$e_1 \geq 0$

$e_2 \geq 0$

$-e_1 \leq x_1 \leq e_1$

$-e_2 \leq x_2 \leq e_2$

where e_i 's are known as slack variables.

Note that the objective now becomes linear in the number of slack variables and this optimization can be solved using linear programming. The weights for each term (like a, b, c above) can be incorporated as:

Min $w_1*|x_1| + w_2*|x_2|$

-->

Min $w_1 * e_1 + w_2 * e_2$

s.t.

$e_1 \geq 0$

$e_2 \geq 0$

$-e_1 \leq x_1 \leq e_1$

$-e_2 \leq x_2 \leq e_2$

This gives a weight to error residuals proportional to w_1, w_2 , etc.

This formulation can be extended to other linear transforms such as similarity and affine transforms (for $P(t)$) easily, since all terms above are linear in $P(t)$ or parameters of $P(t)$.

Sparsity: L1 optimization has the property that the result is “sparse”, i.e. it will attempt to satisfy many properties 1-3 along the path *exactly* (i.e. the derivatives would be exactly 0 for most segments), while L2 optimization would try to minimize them on average (i.e. all derivatives will be small but not exactly 0). What this means is that if $D(p) = 0$, it will lead to static camera; if $DD(p) = 0$, it will lead to linear motion, and where $DDD(p) = 0$, it will have a constant acceleration. It therefore fits the model of the above described optimal camera path.

Smoothness constraints for similarity and affine transformations

When the camera motion model is pure translation, minimizing derivatives of $P(t)$ is straightforward. When it is a similarity transform however, it is a combination of scale, rotation and translation. Our model imposes smoothness on the parameters of $P(t)$, which are a, b, t (see similarity equation above). While t directly corresponds to translation, scale, and rotation are related to a, b as:

Scale, $s = \sqrt{a^2 + b^2}$
Rotation angle, $\theta = \arctan(b/a)$

Constraining smoothness on a, b may not always guarantee that rotation and scale are smooth especially if we want them to be highly smooth. Since imposing smoothness constraints directly on s and θ would be non-linear, we instead constrain just a and b but use the estimated camera path to ensure that s and θ do not deviate too much from that. Specifically, we want:

Scale:
 $\Rightarrow \text{scale_low} < s^2 = a^2 + b^2 < \text{scale_hi}$ [constraint on squared scale]
To linearize this, we use a_0, b_0 from the estimated camera transform $C(t)$, to get:
 $\text{scale_low} < a \cdot a_0 + b \cdot b_0 < \text{scale_hi}$

The low and hi bounds are calculated as
 $\text{scale_low} = (a_0^2 + b_0^2)/k$
 $\text{scale_hi} = (a_0^2 + b_0^2) \cdot k$
where $k \approx 1.05$

Rotation:
Noting that for rotation:
 $a = s \cdot \cos(\theta)$
 $b = s \cdot \sin(\theta)$
We constraint the $\sin(\theta)$ and $\cos(\theta)$ to be bounded near the estimated camera rotation:
 $\cos(a_0/s_0 - \epsilon_1) < a < \cos(a_0/s_0 + \epsilon_1)$
 $\sin(b_0/s_0 - \epsilon_2) < b < \sin(b_0/s_0 + \epsilon_2)$

where ϵ_1 and ϵ_2 are small positive or negative numbers (sign chosen so lower bound is less than upper bound).

Direct stabilization of feature tracks and/or multiple camera paths

The techniques, as presented, first estimate 2D camera path transforms $C(t)$. It then trusts these to be the true paths and optimizes for a new path $P(t)$, which is smooth. It then computes the camera stabilization crop transform $B(t)$, which stabilizes $C(t)$ resulting in $P(t)$. In other examples, this approach additionally or alternatively includes:

1. Instead of solving for the path in two steps by first estimating $C(t)$ and then optimizing for the path $P(t)$, we can roll both steps into one and directly optimize for the stabilization transform from feature correspondences.
2. The estimation of camera path $C(t)$ only uses frame pairs and therefore only relies on the first order derivative. We describe a procedure based on N frames above, but that still requires concatenation of camera derivatives and can lead to error accumulation. Another approach is to work purely with derivatives (1st to 3rd order) of $C(t)$ or equivalently feature tracks.
3. The 2D camera transformation path is effectively the inverse of the *average* feature path transformation. The average is only computed over robust feature points, but it still employs L2 minimization, which results in the average. Instead we can roll the estimation into the optimization and compute a path that attempts to simultaneously stabilize all (or several) feature points. For example, using L1 minimization over all features (for the translation model) results in “median” of feature tracks as opposed to their average, which is more robust.

We formulate the problem in this new setting as directly estimating a “frame stabilization transform” $A(t)$ at each frame. $A(t)$ transforms each video frame (and equivalently features in the frame) as opposed to the camera path or crop window. Mathematically,

$$A(t) = B(t)^{-1}$$

i.e. the frame stabilization transform is the inverse of the camera stabilization transform.

$A(t)$ can also be interpreted as the shake in the camera path, which causes the feature points to move in the opposite direction of the camera. If the shake is instead applied to the feature points, it should stabilize them. To see this, consider a scenario where $A(0) = I$ (identity: no shake). $A(1)$ = translation of *camera* by 10 pixels to the right [$A(1)^*(x,y) = (x + 10,y)$]. $A(1)$ will cause a feature point to move to the *left* by 10 pixels, e.g. (0,0) frame 0 moves to (-10,0) in frame 1. Applying $A(1)$ to (-10,0) moves it back to (0,0), thereby stabilizing it. Given this observation, we can now express stabilized feature points $G(t)$ as a function of original features $F(t)$.

$$G_k(t) = A(t) * F_k(t) \text{ [k: feature point index]}$$

OR

$$G_{ki} = A_i * F_{ki} \text{ [i: time index]}$$

We now impose the smoothness criteria on G_k 's:

Minimize:

$$\sum_t \sum_k a * |DG_k(t)| + b * |DDG_k(t)| + c * |DDDG_k(t)|$$

$$\sum_t \sum_k \{$$

$$a * |A(t+1) * F_k(t+1) - A(t) * F_k(t)| +$$

$$b * |A(t+1) * F_k(t+1) - 2 * A(t) * F_k(t) + A(t-1) * F_k(t-1)| +$$

$$c * |A(t+2) * F_k(t+2) - 3 * A(t+1) * F_k(t+1) - 3 * A(t) * F_k(t) + A(t-1) * F_k(t-1)|$$

$$\} \text{ --- (1)}$$

Using the Taylor series expansion of A (as above), we get
 $A(t)F_k(t) = J_k(t)p$, where $J_k(t) = D_p A(t)F_k(t)$ is the Jacobian of the transformed feature point k w.r.t parameters $p(t)$.

resulting in:

Minimize:

$$\begin{aligned} & \text{Sum}_t \text{Sum}_k \{ \\ & a|J_k(t+1)p(t+1) - J_k(t)p(t)| + \\ & b|J_k(t+1)p(t+1) - 2J_k(t)p(t) + J_k(t-1)p(t-1)| + \\ & c|J_k(t+2)p(t+2) - 3J_k(t+1)p(t+1) - 3J_k(t)p(t) + J_k(t-1)p(t-1)| \\ & \} \text{--- (2)} \\ & = \text{Sum}_t \text{Sum}_k a|D(Jp)| + b|DD(Jp)| + c|DDD(Jp)| \end{aligned}$$

Note that this is a summation over all feature points as opposed to just the camera path. The constraints required to ensure that the crop window stays within the original frame need to be handled differently in this case though. If we were to apply the constraints as before, we would get equations of the form:

$$0 < B(t)w < 1 \Rightarrow 0 < A(t)^{-1}w < 1$$

where w are the crop window corners. This, however is expressed in terms of inverse of A, which is non-linear (specifically quadratic) in the parameters of A. To preserve our linear framework, we make the assumption that the rotation and scale components of A are small enough that if we would transform the *frame window corners*, they would not move by more than the distance to the nearest crop window corner. This assumption allows us to apply a stronger constraint on A. We force the transformed frame window corners to lie in the rectangle formed by the original frame window corner and the nearest crop window corners as its opposite end points. The new constraints are then expressed as follows:

Let $(r_x, r_y) = A(t) * v$, where v is one of the original frame window corner: one of $\{(0,0), (0,1), (1,0), (1,1)\}$.

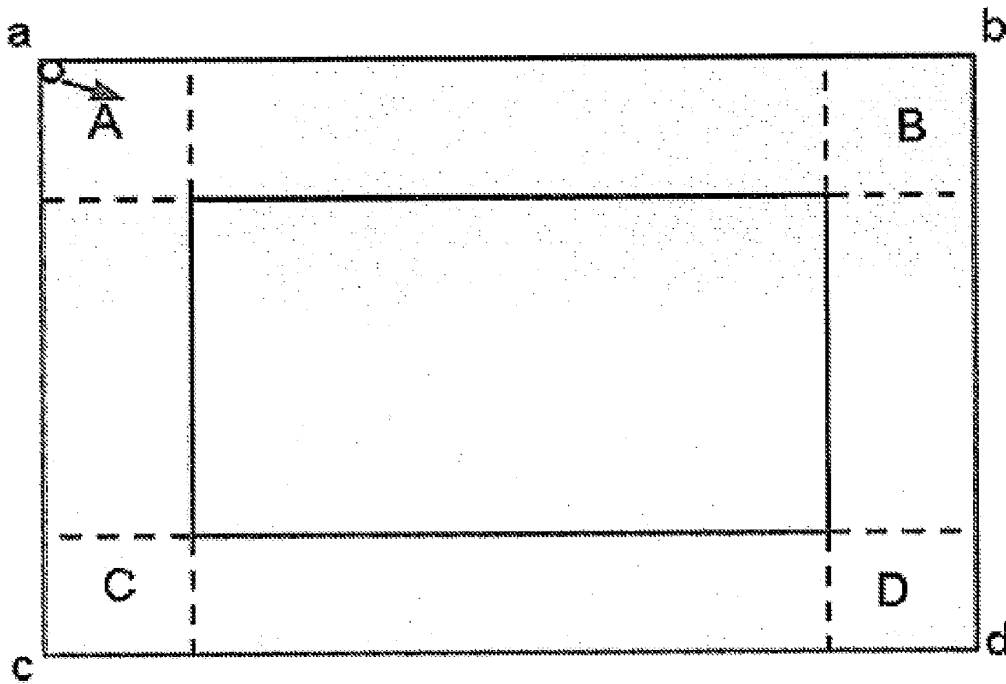
Also let the for crop window (fixed based on the desired scale of the crop mask) be bounded on left, top, bottom, right by cl, ct, cb, cr respectively. Then the constraints are:

$$\begin{aligned} & A(0,0)(x) < cl \\ & A(0,0)(y) < ct \\ & A(1,0)(x) > cr \\ & A(1,0)(y) < ct \\ & A(0,1)(x) < cl \\ & A(0,1)(y) > cb \\ & A(1,1)(x) > cr \\ & A(1,1)(y) > cb \\ & \text{where } A(x,y) = A(t) * (x,y) \end{aligned}$$

An equivalent way of writing this is:

$$n^t A v \leq 0, \text{--- (3)}$$

where n is the line representation of each bounding line of the crop-rectangle, with a sign such that the inequality corresponds to putting $A*v$ inside the cropping rectangle.



Constraints: a cannot only move beyond the box labeled A (cannot cross to right or bottom of bounding dotted lines). The constraints in the previous formulation allowed motion beyond dotted line as long as the transformed frame window contained the smaller crop mask (interior rectangle). However since rotation and scale are expected to be small, these more restrictive constraints are reasonable.

Efficient solution (going back to feature transforms): While the above formulation from feature tracks works, it can be too slow to compute as such, because the resulting linear programming problem may become huge, adding one slack variable per feature per frame. We make the solution more efficient by observing that all feature points corresponding to a frame pair (for D_p), triplets DD_p etc. affect the objective parameters directly only for that pair, etc. Therefore the system can be decoupled by first solving it individually on a per-frame set (pair for D_p , triplet for DD_p , quadruple for DDD_p etc) basis. This is followed by the path optimization over all frames. The per-frame-set solution has the effect of collapsing the per feature point terms into a single term or a small number of terms.

The collapsing procedure for a frame set works by recalling the camera transformation based method above but using the feature path formulation just described. The observation is that the feature paths themselves convey a transformation (or a set of transformations), which are consistent with the feature path. So if we limit the transformations to translation, then each feature path conveys a translation transform. If we admit higher order transforms such as similarity, then a single feature path is insufficient to uniquely determine the postulated transformation but rather votes for the set of transformations consistent with it. If we limit ourselves to a single transformation between a pair of frames, then we can first compute the transformation most consistent with all features and plug it back into the equations above. Consider just the frame pair term $|D(J_p)|$ between frames 1 and 2. Let T be the single transformation the best relates features from frame 1 to frame 2, i.e.

$$F_k2 = T * F_k1$$

Then the first term in equation (1) becomes

$$\text{Sum_k } |A2 * T * F_k1 - A1 * F_k1|$$

This can be converted to the Jacobian form in (2) as

$$\text{Sum_k } |J(T * F_k1) * p2 - J(F_k1) * p1|$$

Now, note that J is linear in the coordinates of feature points F_k . Specifically, if A is the similarity transform, then for $F_k = (x, y)$

$$T * F1_k = [a \ -b; b \ a] * (x, y) + [dx; dy] = [ax - by + dx; bx + ay + dy] := [x' \ y']$$

$$\text{Then } J(F2_k) = J(T * F1_k) = \begin{bmatrix} 1 & 0 & x' & -y' \\ 0 & 1 & y' & x' \end{bmatrix}$$

which now can be rewritten as

$$J(T * F1_k) = J(F1_k) * M(T)$$

$$\text{with: } J(f1_k) = \begin{bmatrix} 1 & 0 & x & -y \\ 0 & 1 & y & x \end{bmatrix}$$

using a parametrization of $p1 = [dx_1, dy_1, a_1, b_1]$

and

$$M(T) = \begin{bmatrix} 1 & 0 & dx & -dy \\ 0 & 1 & dy & dx \\ 0 & 0 & a & -b \\ 0 & 0 & b & a \end{bmatrix}$$

plugging this back into D-term of (2):

$$\begin{aligned} & \text{Sum_k } |J(T * F1_k) * p2 - J(F1_k) * p1| \\ &= \text{Sum_k } |J(F1_k) * M(T) * p2 - J(F1_k) * p1| \\ &= \text{Sum_k } |J(F1_k) (M(T) * p2 - p1)| \\ &\leq |M(T) * p2 - p1| * \text{Sum_k } |J(F1_k)| \end{aligned}$$

Since $\text{Sum_k } |J(f1_k)|$ aggregates all features and is therefore independent of the optimization parameters p , we can essentially ignore it [or use it as a weight; although since $J(f1_k)$ is only a function of feature locations, using it as weight will only be a function of which features happen to be selected and is therefore probably misguided].

Effective outcome of this derivation is that we ignore the individual features and optimize only for $|M * p2 - p1|$, where M depends on the precomputed transform T from all features. M is a matrix that relates the parameters $p1$ and $p2$ based on T . This reduces to minimization of the L1 norm over linear functions of parameters p . The same derivation can be extended to DD and DDD.

It is possible to perform the minimization in a higher dimensional space than $p1$ and $p2$ resides in, e.g. to embed the similarity (4 DOF -- Degrees of Freedom) into an affine transform (6 DOF). We start by deriving the embedding for a 4 DOF similarity (parametrized by s) into an affine (6 DOF) or homography (8 DOF) governed by the frame transform T . If T is affine, we want to minimize

$$|Ma(T) * Pa * s2 - Pa * s1|$$

where Pa is a permutation matrix transforming the similarity $s1 = [dx_1; dy_1; a_1; b_1]$ into an affine transform $p1 = [dx_1; dy_1; a_1; -b_1; b_1; a_1]$, i.e.

$$Pa = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

The matrix $Ma(T)$ is a 6x6 matrix that relates 6 DOF affine parameters between two frames based on the affine transform T :

If

$$Tx = [a \ b; c \ d] x + [dx \ dy]$$

then

$$Ma(t) = \begin{bmatrix} 1 & 0 & dx & dy & 0 & 0 \\ 0 & 1 & 0 & 0 & dx & dy \\ 0 & 0 & a & c & 0 & 0 \\ 0 & 0 & b & d & 0 & 0 \\ 0 & 0 & 0 & 0 & a & c \\ 0 & 0 & 0 & 0 & b & d \end{bmatrix}$$

Note that while the input $s1$ and $s2$ only have 4 DOF, the minimization is performed in a 6 dimensional space. This results in an over-constrained system where the 4 parameters are optimized to best fit 6 potentially competing constraints. Note that these 6 constraints are being fed into the path optimization and the actual parameters are then chosen based on the best overall fit.

In a similar fashion, we can compute the matrices Ph and $Mh(T)$ for a homography T . We want to minimize

$$|Mh(T)*Ph*s2 - Ph*s1|$$

The matrix for the transformation of $s1$ into a homography can be obtained by adding two zero rows to Pa . For a homography parametrized as:

$$T(h) = \begin{bmatrix} h_1 & h_2 & h_3 \\ h_4 & h_5 & h_6 \\ h_7 & h_8 & 1 \end{bmatrix}$$

We get

$$Mh(t) = \begin{bmatrix} h_1 & h_4 & h_7 & 0 & 0 & 0 & 0 & 0 & 0 \\ h_2 & h_5 & h_8 & 0 & 0 & 0 & 0 & 0 & 0 \\ h_3 & h_6 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & h_1 & h_4 & h_7 & 0 & 0 & 0 \\ 0 & 0 & 0 & h_2 & h_5 & h_8 & 0 & 0 & 0 \\ 0 & 0 & 0 & h_3 & h_6 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & h_1 & h_4 & h_7 \\ 0 & 0 & 0 & 0 & 0 & 0 & h_2 & h_5 & h_8 \\ 0 & 0 & 0 & 0 & 0 & 0 & h_3 & h_6 & 1 \end{bmatrix}$$

When using minimization with homographies a complicating factor is that homographies are only defined up to scale. We removed this ambiguity by setting the last element to 1, and we have that the last element of $H(T) * h'$ equals $h_3 * h_7' + h_6 * h_8' + 1 := w$. In case we use for p_1 a similarity or an affine transform, $h_7' = h_8' = 0$ and it follows that $w = 1$. [It also implies that we can erase column 7 and 8 of $H(T)$. Further there is constant error of $h_7 + h_8 + 1$ which can not optimized for.]

However, if we optimize for a full 8 DOF homography, in general w will not be equal to one. We therefore either need to divide by w , giving $|1/w * H(T) * h_2 - h_1|$ which is not linear anymore or multiply h_1 by w , giving $|H(T) * h_2 - w * h_1|$ which is quadratic w.r.t. the parameters h_1 and h_2 . The coefficients modelling the current perspective h_7' and h_8' are in our experiments on the order of $O(1e-3)$, while estimated translations h_2 and h_3 are $O(1e2)$ giving $w = 1 + O(1e-1)$. In this case we suggest to solve the problem by functional iteration. We derive w from the current parameters of h_2 , treating it as a fixed scalar, optimizing for h_1 and h_2 . Then we set w again from the current parameters of h_2 , iterating between the evaluation of w and optimization of h_1 and h_2 until convergence. Note that this is guaranteed to converge by the functional mapping theorem if the change in error divided by the change of parameters is bounded by 1.

Note that this is similar to first computing the single camera transform between frames (as we do before), followed by path optimization, but with two differences: Firstly, here we work with the forward feature transform (as opposed to its inverse, camera transform). Secondly, the path optimization is over smoothness of feature paths implied by the transform. It turns out, this is equivalent to smoothing the parameters of the forward transform, while earlier we were smoothing parameters of the cumulative camera transform (which is the inverse of the forward transform multiplied by the cumulative transform). In one example, an advantage of the forward transform method is that, because it is not cumulative, it does not accumulate error. Secondly for second and third order terms, DD and DDD, smoothing feature paths is more consistent with the physical interpretation of expected camera motions than the smoothing of camera parameters themselves.

Physical implications of camera motion on feature paths

In some examples, it can be shown that the following 3D camera motions result in equivalent smooth motion of feature points with certain assumptions:

1. Camera translation, assuming distance from camera to objects $\gg$ velocity in any direction.
 - Static camera = static feature points
 - Constant 3D velocity lateral to camera = constant feature point velocity

- Constant 3D velocity in depth $\approx$ constant feature point velocity [assumes depth (Z) is large]
- Same for acceleration

2. Camera zoom

- Constant velocity zoom = constant feature point velocity
- same for acceleration

3. Camera rotation

- Feature point motion derivatives diminish as a square of angular velocity. If angular velocity is small $\Rightarrow$ higher order derivatives tend to 0. therefore minimizing $|DDD_x|$ reasonable. Another interpretation is that rotational motion can be well approximated by a series of parabolic motion segments.

Multiple transforms

One extension of this is to do something intermediate between using all features explicitly and the single transform between features. Specifically, we can use a small number of transforms between a frame pair, where each of these transforms may correspond to different regions in the image that may be moving differently. Each region would correspond to a different transform, and therefore a different M matrix. The D-term would become

$$\sum_k |M_k \cdot p_1 - p_2|,$$

where k now refers to the region transforms. Note that the number of such transforms is expected to be much lower (say 3 to 4) than the number of features (of the order of hundreds), and therefore much more tractable and stable.

One could also weight these transforms differently depending upon various factors, such as

- Foreground/background separation: Better to stabilize foreground than background (or stabilize the foreground, if we want to track it).
- Size of regions: Better to stabilize larger regions than smaller ones
- Textured-ness: Better to stabilize textured regions over untextured ones, any errors/artifacts more visible in textured regions.

The path optimization is then supposed to smooth compute a stabilization which minimizes the L1 norm of path smoothness over all transforms. This effectively would lead to the algorithm "picking" a set of transforms to smooth while leaving others as unstable. The choice of which transforms are smoothed would be determined by a combination of their individual importance (weights) and how easily they can be stabilized over time.

More general feature transform

Another extension is that since we decouple the feature transform T from the stabilization transform A, we can allow T to be more general than A. i.e. we can use an affine or homography transform for T, while restricting A to be a similarity transform. This would allow the T transforms to result in a better alignment of features from frame-to-frame. Since A is still similarity, it would not introduce any distortions that the affine or homography transforms can introduce, but try to "fit" them as best as possible.

Content-aware Video Stabilization and Video Retargeting

Our framework allows us to easily add new constraints in the system in a similar way as we add constraints for forcing the cropping window to be inside the frame. Specifically if we want to use certain content aware signals, such as a face (from face detector) or other salient (or user-marked) objects/regions, which should stay within the frame, we can incorporate that into our constraints. Generally, we think of these content-aware constraints being specified as regions which should either stay in the cropped frame completely (hard constraint) or as much as possible (soft constraint). Additionally, these constraints need not be specified in all frames as the path smoothness will automatically propagate the constraints from key-frames to all other frames to a large extent. If a region is represented using a bounding polygon, such as a bounding box, then a sufficient constraint for containment is that each vertex of the polygon should lie within the cropping window. If v is a vertex of the polygon, then in the forward feature path optimization framework, the constraint is opposite of the one in (3):

$$n^t \cdot A(t) \cdot v \geq 0$$

This is a hard constraint, forcing the region of interest to always lie inside the cropping window. However sometimes this may lead to an infeasible system where all constraints may not be simultaneously satisfied. A more relaxed constraint is a soft “one-sided” constraint which penalizes any vertices that move out of the cropping window. Recall that we can convert L1 minimization to a linear program by adding slack variables, which are minimized, and modified constraints that bound the original constraint within $(-slack_variable, +slack_variable)$. We follow the same procedure here but only bound the constraints from one side.

Specifically we add to the objective:

$$\sum_k w_k \cdot d_k$$

where k indexes a combination of polygon vertex v and crop rectangle line n and the constraints:

$$n_k^t \cdot A(t) \cdot v_k \geq -d_k \quad (5)$$

$$d_k \geq 0$$

Content-aware constraints may also be used for retargeting in addition/in place of stabilization. Retargeting refers to the notion of modifying a video to fit a device with a different resolution aspect-ratio than the original video. We can use our framework for the same by simply modifying the cropping window to match the aspect ratio of the target device. We can employ the content aware constraints above to ensure that salient content stays within this crop window. Optionally, we can drop the stabilization of feature points if we only need retargeting. This implies instead of estimating the forward feature transform T between every pair of frames and using the resulting M matrices, we simply set $M = I$ (identity), which corresponds to minimizing

$$|p_2 - p_1| \text{ instead of } |M \cdot p_2 - p_1| \text{ or (including higher order terms):}$$

$$\text{Min: } |Dp| + |DDp| + |DDDp|$$

i.e. we want the the frame transforms induced by A to be smooth over time as opposed to the features themselves. Note that if all constraints are satisfied already, this may result in $A = I$ for all frames. Otherwise, this would find A 's that fit the constraints while smoothly interpolating between keyframes.

If there are multiple salient objects, then the L1 minimization (via constraints as in (5)) again acts to satisfy as many constraints as possible by “selects” the one that can be satisfied better and satisfies them well as opposed to trying to satisfy all of them in the average sense. One could use weighting here to prefer different salient regions differently.

Automatic crop window size estimation

In one example, the crop window scale needs to be specified upfront. One possible downside is that even if the video is already stabilized, it still may lead to a cropped result. Therefore, a good choice of the scale is provided upfront. Another option is to automatically set that scale automatically. One way to achieve this is to first perform stabilization and then expand the crop window to the maximum possible size that still fits within the transformed frame windows over all frames. i.e. Compute $A(t)*v$, where v are the frame window corners, for all frames t . Then find the largest rectangle (represented by lines m), such that, $m^t * A(t) * v \geq 0$. Note that since m is axis aligned, computing this rectangle can be done by simply looking at the min and max values of the frame window coordinates over all times.

One issue with this approach is that since we do not impose any constraints into the framework upfront that may help the crop-window to “expand” later on, the chosen paths may end up being such that the crop window cannot expand at all. We can alleviate this by adding constraints similar to the one-sided constraints above, such that frame transforms $A(t)$, that result in as large a frame window as possible are encouraged. The corresponding objective and constraints will be of the form:

Min $\sum_k w_k * d_k$

such that

$r^t * A(t) * v \geq d_k$

$d_k \geq 0$

k varies over a combination of the four frame window corners and frame window rectangle lines r .

[ONLY use the nearest two frame window lines for each window corner, because otherwise the farther lines will impose huge unnecessary penalty]

d_k are slack variables representing how far each corner moves away from the original frame rectangle.

w_k are weights (typically very small: we only want them to impact the solution to the extent that the crop window is as centered as possible so we can expand it later).

Learning weights from professional footage

Instead of setting the relative weights between D , DD , and DDD by hand, one can optionally learn them from professional footage. The idea is that professional videos have the desirable combination of the different kinds of camera motions. Therefore if we take a professional video, and artificially add jitter to its camera path, our stabilization algorithm should be able to retrieve the original smooth camera path. Therefore the learned parameters are those that result in a close match to the original path. Since we have 3 parameters, we can do an exhaustive exploration of the search space, similar to the exploration of thresholding parameters in Support

Vector Machines (SVM). Steps:

1. Take professional footage and compute camera / feature path by estimating T ($\leftarrow$ optimal path ground truth)
2. Jitter path ($\leftarrow$ shaky path, to be corrected)
3. Parameter search (weights in objective function). Divide parameters into a $N \times N \times N$ grid and compute the path for each combination of parameters. Pick the one with least error / best match

to optimal path. The error is not simply distance to optimal path, but needs to model smoothness of original path as well.

Example System Overview

Figure 1 below is a block diagram illustrating a system view of a video hosting service 100 having a camera path translation system 102. Multiple users/viewers use client 110A-N to send video hosting requests to the video hosting service 100, such as uploading videos to a video hosting website, and receive the requested services from the video hosting service 100. The video hosting service 100 communicates with one or more clients 110A-N via a network 130. The video hosting service 100 receives the video hosting service requests from clients 110A-N, which may be received over wired or wireless connections

Turning to the individual entities illustrated on Figure 1, each client 110 is used by a user to request video hosting services. For example, a user uses a client 110 to send a request for uploading a video for sharing, or playing a video. The client 110 can be any type of computer device, such as a personal computer (e.g., desktop, notebook, tablet, laptop) computer, as well as devices such as a mobile telephone, personal digital assistant, IP enabled video player. The client 110 typically includes a processor, a display device (or output to a display device), a local storage, such as a hard drive or flash memory device, to which the client 110 stores data used by the user in performing tasks, and a network interface for coupling to the system 100 via the network 130.

A client 110 also has a video player 120 (e.g., the Flash™ player from Adobe Systems, Inc., or a proprietary one) for playing a video stream. The video player 120 may be a standalone application, or a plug-in to another application such as a network browser. Where the client 110 is a general purpose device (e.g., a desktop computer, mobile phone), the player 120 is typically implemented as software executed by the computer. Where the client 110 is dedicated device (e.g., a dedicated video player), the player 120 may be implemented in hardware, or a combination of hardware and software. All of these implementations are functionally equivalent in regards to the present invention. The player 120 includes user interface controls (and corresponding application programming interfaces) for selecting a video feed, starting, stopping, and rewinding a video feed. Also, the player 120 can include in its user interface a video display format selection configured to indicate a video display format (e.g., a standard definition TV or a high-definition TV). Other types of user interface controls (e.g., buttons, keyboard controls) can be used as well to control the playback and video format selection functionality of the player 120.

The network 130 enables communications between the clients 110 and the video hosting service 100. In one embodiment, the network 130 is the Internet, and uses standardized internetworking communications technologies and protocols, known now or subsequently developed that enable the clients 110 to communicate with the video hosting service 100.

The video hosting service 100 comprises a camera path translation system 102, a video server 104, an ingest server 106 and a video database 108. The ingest server 106 receives user uploaded videos and stores the videos in the video database 108. The video server 104 serves the videos from the video database 108 in response to user video hosting service requests. The video database 108 stores user uploaded videos and videos processed by the

camera path translation system 102. In one embodiment, the video database 108 stores a large video corpus.

The camera path translation system 102 may include a camera path estimation engine 200 that may estimate a path of a camera based on motion of objects within a received video. The camera path translation system 102 also may include a video stabilization and retargeting engine 400 that may estimate a new steady/smooth camera path. The camera path translation system 102 may also include a video translation engine 600 that may recast the received video from a viewpoint of the smooth camera path determined by the video stabilization and retargeting engine 400, using any of the examples described above.