# Video Stabilization and Enhancement

Hany Farid and Jeffrey B. Woodward
Department of Computer Science
Dartmouth College
Hanover NH 03755

## Abstract

We describe a simple and computationally efficient approach for video stabilization and enhancement. By combining multiple low-quality video frames, it is possible to extract a high-quality still image. This technique is particularly helpful in identifying people, license plates, etc. from low-quality video surveillance cameras.

# 1 Introduction

The poor image quality of many video surveillance cameras effectively renders them useless for the purposes of identifying a person, a license plate, etc. Under certain conditions, however, it may be possible to combine multiple video frames for such identification tasks. To this end, we describe a simple and computationally efficient technique for video stabilization and enhancement. The efficacy of this technique is shown on synthetic and real video sequences.

# 2 Video Stabilization

The goal of video stabilization is to create a new video sequence where the motion between frames (or parts of a frame) has effectively been removed. To this end, differential motion estimation has proven highly effective at computing inter-frame motion [3, 2, 5]. We describe one such approach and then describe how a full video sequence is stabilized.

We begin by modeling the motion between two sequential frames, $f(x, y, t)$ and $f(x, y, t-1)$ with a 6-parameter affine transform:

$$f(x, y, t) \quad = \quad f(m_1 x + m_2 y + m_5, m_3 x + m_4 y + m_6, t-1), \tag{1}$$

where $m_1$, $m_2$, $m_3$, $m_4$ form the $2 \times 2$ affine matrix $A$ and $m_5$ and $m_6$ the translation vector $\vec{T}$ given by:

$$A = \begin{pmatrix} m_1 & m_2 \\ m_3 & m_4 \end{pmatrix} \qquad \text{and} \qquad \vec{T} = \begin{pmatrix} m_5 \\ m_6 \end{pmatrix}. \tag{2}$$

With relatively few parameters, the affine model captures a sufficiently rich range of motions such as translation, scale, and rotation. The basic framework described here can easily be extended to accommodate higher-order polynomial models that capture a richer set of motions (or even a lower-order model that models the inter-frame motion with only a translation). In order to estimate the affine parameters, we define the following quadratic error function to be minimized:

$$E(\vec{m}) \quad = \quad \sum_{x,y \in \Omega} [f(x, y, t) - f(m_1 x + m_2 y + m_5, m_3 x + m_4 y + m_6, t-1)]^2, \tag{3}$$

where $\vec{m}^T = (\, m_1 \quad \ldots \quad m_6 \,)$ and where $\Omega$ denotes a user specified ROI. Since this error function is non-linear in its unknowns, it cannot be minimized analytically. To simplify the minimization, we approximate this error function using a first-order truncated Taylor series expansion:

$$\begin{aligned} E(\vec{m}) \quad &\approx \quad \sum_{x,y \in \Omega} [f - (f + (m_1 x + m_2 y + m_5 - x) f_x - (m_3 x + m_4 y + m_6 - y) f_y - f_t)]^2 \\ &= \quad \sum_{x,y \in \Omega} [f_t - (m_1 x + m_2 y + m_5 - x) f_x - (m_3 x + m_4 y + m_6 - y) f_y]^2 \\ &= \quad \sum_{x,y \in \Omega} \left[ k - \vec{c}^T \vec{m} \right]^2, \end{aligned} \tag{4}$$

where, for notational convenience, the spatial/temporal parameters are dropped, and where the scalar $k$ and vector $\vec{c}$ are given as:

$$k \quad = \quad f_t + x f_x + y f_y \tag{5}$$

$$\vec{c}^T \quad = \quad (\, x f_x \quad y f_x \quad x f_y \quad y f_y \quad f_x \quad f_y \,). \tag{6}$$

The quadratic error function is now linear in its unknowns, $\vec{m}$, and can therefore be minimized analytically by differentiating with respect to $\vec{m}$:

$$\frac{dE(\vec{m})}{d\vec{m}} = \sum_{\Omega} 2\vec{c}\left[k - \vec{c}^T\vec{m}\right], \tag{7}$$

setting the result equal to zero, and solving for $\vec{m}$ to yield:

$$\vec{m} = \left[\sum_{\Omega} \vec{c}\vec{c}^T\right]^{-1}\left[\sum_{\Omega} \vec{c}k\right]. \tag{8}$$

This solution assumes that the first term, a $6 \times 6$ matrix, is invertible. This can usually be guaranteed by integrating over a large enough ROI ($\Omega$) with sufficient image content.

Central to the above motion estimation are the calculation of the spatial/temporal derivatives. Given a pair of frames $f(x, y, t)$ and $f(x, y, t-1)$, these derivatives are computed as follows:

$$f_x(x, y, t) = (0.5f(x, y, t) + 0.5f(x, y, t-1)) \star d(x) \star p(y) \tag{9}$$

$$f_y(x, y, t) = (0.5f(x, y, t) + 0.5f(x, y, t-1)) \star p(x) \star d(y) \tag{10}$$

$$f_t(x, y, t) = (0.5f(x, y, t) - 0.5f(x, y, t-1)) \star p(x) \star p(y), \tag{11}$$

where $\star$ denotes the convolution operator, and $d(\cdot)$ and $p(\cdot)$ are 1-D separable filters:

$$d(x) = \begin{pmatrix} 0.5 & -0.5 \end{pmatrix} \qquad \text{and} \qquad p(x) = \begin{pmatrix} 0.5 & 0.5 \end{pmatrix}, \tag{12}$$

and where $p(y)$ and $d(y)$ are the same filters oriented vertically instead of horizontally.

The required spatial/temporal derivatives have finite support thus fundamentally limiting the amount of motion that can be estimated. A coarse-to-fine scheme is adopted in order to contend with larger motions [4, 1]. A $L$-level Gaussian pyramid is built for each frame, $f(x, y, t)$ and $f(x, y, t-1)$. The motion estimated at pyramid level $l$ is used to warp the frame at the next higher level $l-1$, until the finest level of the pyramid is reached (the full resolution frame at $l = 1$). In so doing, large motions are estimated at the coarse levels, and are iteratively refined through the pyramid levels. The repeated warping through the pyramid levels can introduce significant blurring and therefore adversely effect the motion estimation at the higher pyramid levels. This can be avoided by always operating on the original frame. Specifically, if the estimated motion at pyramid level $l$ is $m_1$, $m_2$, $m_3$, $m_4$, $m_5$, and $m_6$, then the original frame should be warped with the affine matrix $A$ and the translation vector $\vec{T}$ given by:

$$A = \begin{pmatrix} m_1 & m_2 \\ m_3 & m_4 \end{pmatrix} \qquad \text{and} \qquad \vec{T} = \begin{pmatrix} 2^{l-1}m_5 \\ 2^{l-1}m_6 \end{pmatrix}. \tag{13}$$

As we work through each level of the pyramid, the original frame will have to be repeatedly warped according to the motion estimates at each pyramid level. These individual warps can be accumulated so that the original frame need only undergo one warp. Specifically, two affine matrices $A_1$ and $A_2$ and corresponding translation vectors $\vec{T_1}$ and $\vec{T_2}$ are combined as follows:

$$A = A_2A_1 \qquad \text{and} \qquad \vec{T} = A_2\vec{T_1} + \vec{T_2}, \tag{14}$$

which is equivalent to applying $A_1$ and $\vec{T_1}$ followed by $A_2$ and $\vec{T_2}$. The warped original frame is then decomposed into the next required pyramid level, and the process repeated through all pyramid levels.

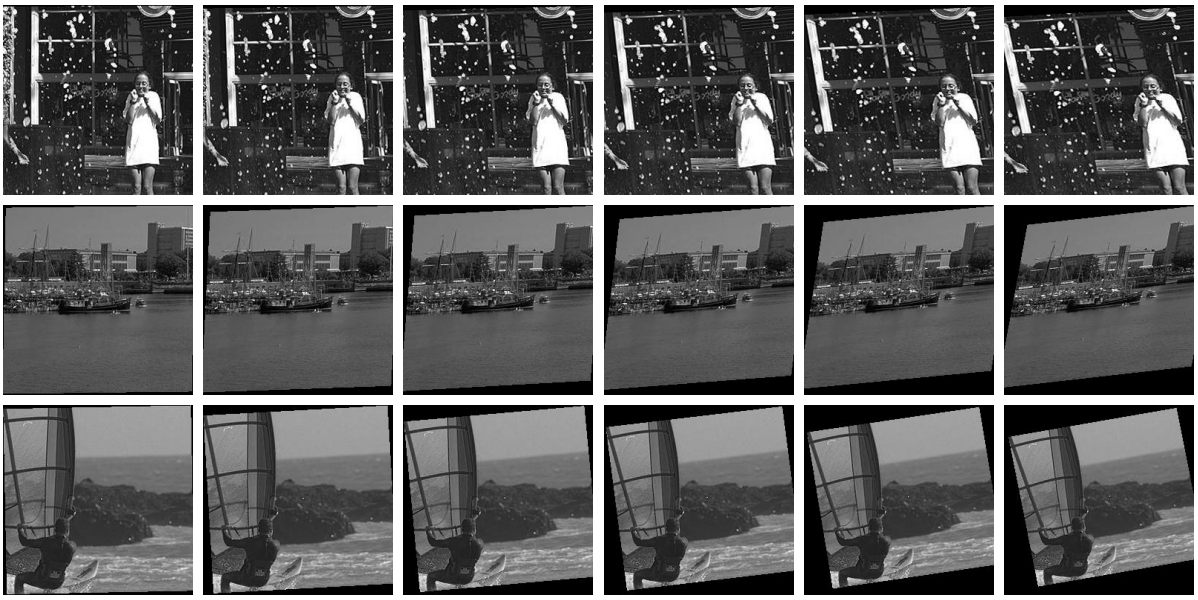

**Figure 1:** Three of one hundred simulated video sequences undergoing a global affine transformation (every other frame of the 11-frame sequence is shown).

To this point, we have described how to estimate the inter-frame motion between two frames. For a video sequence $f(x, y, t)$, $t \in [1, N]$, the inter-frame motion is computed between all pairs of neighboring frames, $f(x, y, t)$ and $f(x, y, t-1)$, for $t \in [2, N]$ to yield motion estimates $\vec{m}_t$. With the ROI ($\Omega$ in Equation (8)) specified on the first frame, the ROI must be warped on each successive frame to account for the underlying motion between frames. The ROI at time $t$ ($t > 2$) is therefore simply warped according to the previous frame's estimated motion. With the motion estimated between all pairs of neighboring frames, each frame is warped to align with the last frame of the sequence. Successive warps are combined according to Equation (14). See Appendix A for a complete Matlab implementation of this video stabilization algorithm.

## 3 Video Enhancement

Once stabilized, the video sequence is combined to yield a single high quality image. We assume that the corrupting noise on each video frame is identically and independently drawn (iid) from a zero-mean distribution. Under this assumption a temporal mean or median filter is the optimal approach to removing the noise – the temporal mean minimizes the L2 norm and the temporal median minimizes the L1 norm between the true and estimated value at each pixel. Specifically, denote $f(x, y, t)$, $t \in [1, N]$ as the original video sequence, and $\hat{f}(x, y, t)$ as the stabilized video. The enhanced frame is estimated by simply computing a pixel-wise mean or median across $\hat{f}(x, y, t)$. Although typically comparable, one temporal filter may give better results than the other depending on the specific noise characteristics. See Appendix B for a Matlab implementation.

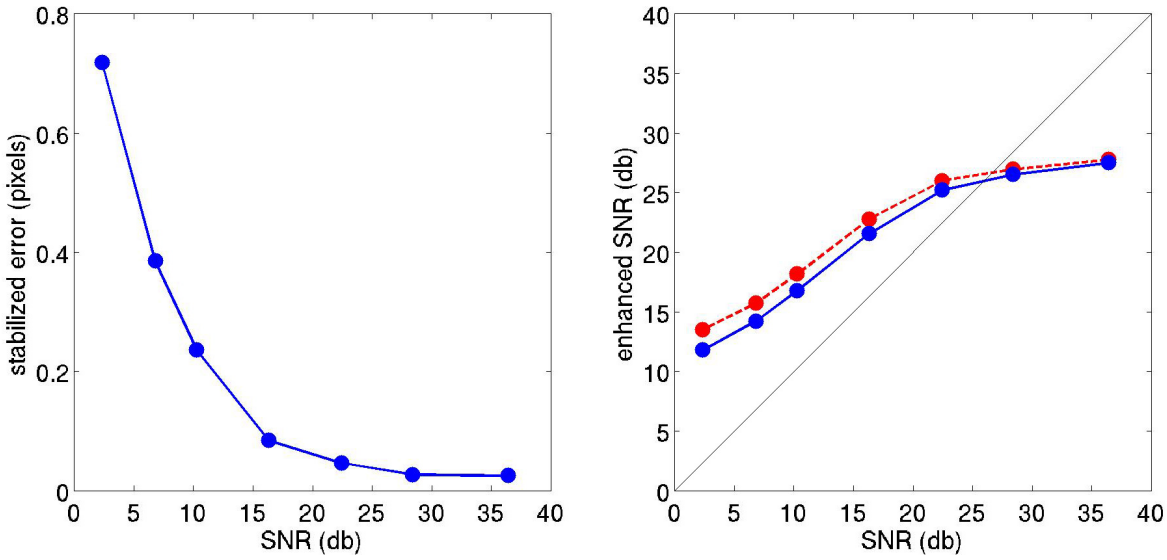


**Figure 2:** Shown is (a) the average misalignment in pixels as a function of signal to noise ratio (SNR); and (b) the signal to noise ratio of the enhanced image (the red dashed curve denotes the results of the temporal mean filter, and blue solid curve denotes the results of the temporal median filter).

# 4 Results

To test the efficacy of the video stabilization algorithm, simulated video sequences were generated as follows. The first frame was a $256 \times 256$ central crop of a grayscale image. The motion between frames followed the affine model, Equation (2), where the affine and translation parameters were randomly generated to yield an average 2-pixel displacement between successive frames. Shown in Figure 1 are three examples of 11-frame sequences (only every other frame is shown). The ROI was a central $240 \times 240$ region. One hundred sequences were generated and stabilized. Prior to stabilization, the average frame to frame displacement was $2.0$ pixels. After stabilization, the average pixel displacement was $0.02$ pixels. Visually, little to no motion could be seen in the stabilized video.

Since we are interested in enhancing noisy video, it is natural to test the efficacy of the stabilization in the face of noise. Random zero-mean Gaussian noise, with varying signal to noise ratio (SNR), was added to each frame of a video sequence prior to stabilization. The same one hundred sequences as described above were stabilized. Shown in Figure 2(a) is the average pixel displacement after stabilization as a function of SNR, and shown in Figure 2(b) is the signal to noise ratio of the enhanced frame. Note that even at very high levels of noise, the stabilization is effective to sub-pixel levels, and the enhanced frame is of significantly higher signal to noise ratio relative to the original. Shown in Figure 3 are three examples of the final enhanced frames.

And lastly, two real video sequences with complex motions were further corrupted with noise, stabilized and enhanced. Shown in Figure 4 is one frame of the original video and the enhancement results. In these examples, the motion estimation was applied to a grayscale version of each color frame, and the stabilization and enhancement applied separately to each color channel. The ROI was a rectangular region encompassing the license plate and face.

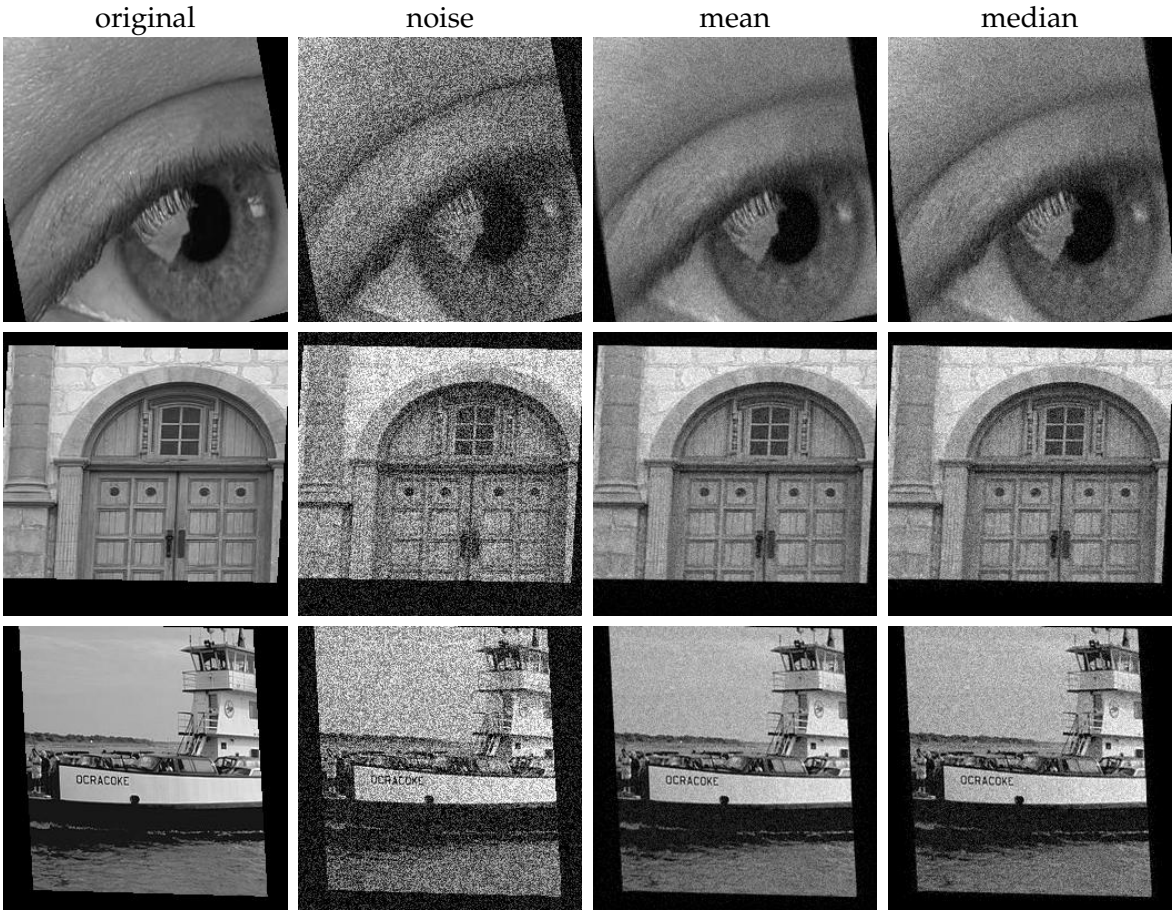


**Figure 3:** Shown is an original frame, this frame plus noise ($\sim$ 30 dB) and the results of stabilization and temporal mean and median filtering.

## 5 Discussion

We have described a simple and computationally efficient technique for video stabilization and enhancement. The motion between video frames is modeled as a global affine transform whose parameters are estimated using standard differential motion techniques. A temporal mean or median filter is then applied to this stabilized video to yield a single high quality frame. We have shown the effectiveness of this technique on both synthetically generated and real video. This technique should prove useful in enhancing the quality of low-grade video surveillance cameras.

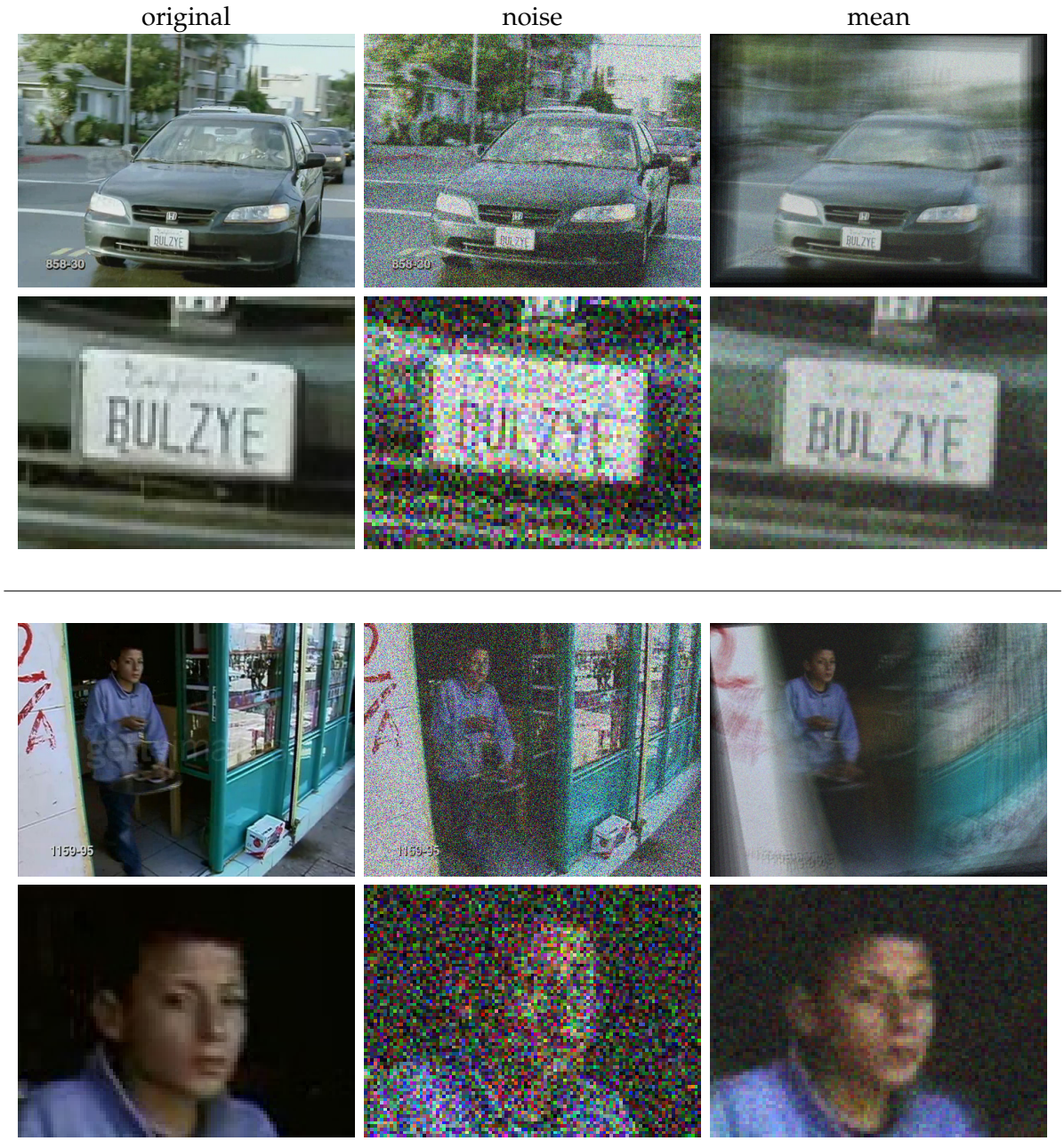


**Figure 4:** Shown is an original frame, this frame plus noise and the results of stabilization and temporal mean filter. Also shown is a magnified view of the license plate (top) and face (bottom).

## 6 Acknowledgments

This work was supported by a gift from Adobe Systems, Inc., a gift from Microsoft, Inc., a grant from the United States Air Force (FA8750-06-C-0011), and by the Institute for Security Technology Studies at Dartmouth College under grant 2005-DD-BX-1091 from the Bureau of Justice Assistance and Award Number 2006-CS-001-000001 from the U.S. Department of Homeland Security. Points of view or opinions in this document are those of the author and do not represent the official position or policies of the U.S. Department of Justice, the U.S. Department of Homeland Security, or any other sponsor.

## References

[1] P. Anandan. A computational framework and an algorithm for the measurement of visual motion. *International Journal of Computer Vision*, 2(3):283–310, 1989.

[2] J.L. Barron, D.J. Fleet, and S.S. Beauchemin. Performance of optical flow techniques. *International Journal of Computer Vision*, 12(1):43–77, Feb. 1994.

[3] B.K.P. Horn. *Robot Vision*. MIT Press, Cambridge, MA, 1986.

[4] B.D. Lucas and T. Kanade. An iterative image registration technique with an application to stereo vision. In *International Joint Conference on Artificial Intelligence*, pages 674–679, Vancouver, 1981.

[5] E.P. Simoncelli. *Handbook of Computer Vision and Applications*, chapter Bayesian Multi-scale Differential Optical Flow, pages 397–420. Academic Press, 1999.

# Appendix A

Below is Matlab code for video stabilization. The function `videostabilize` takes as input: (1) a data structure `frames` with field `im` that contains the original video sequence; (2) the number of Gaussian pyramid levels $L$; and (3) a binary image `roi` that specifies the region of interest, with value of 1 for those pixels to be considered in the motion estimation, and 0 otherwise. It is assumed that each frame is a grayscale image. The output of this function is a data structure `stable` with field `im` that contains the stabilized video sequence, and a data structure `motion` with fields `A` and `T` that contain the estimated affine and translation parameters.

```
% ------------------------------------------------------------------------
%%% STABILIZE VIDEO

function[ motion, stable ] = videostabilize( frames, roi, L )

   N = length( frames );
   roiorig = roi;

   %%% ESTIMATE PAIRWISE MOTION
   Acum = [1 0 ; 0 1];
   Tcum = [0 ; 0];
   stable(1).roi = roiorig;
   for k = 1 : N-1
      [A,T] = opticalflow( frames(k+1).im, frames(k).im, roi, L );
      motion(k).A = A;
      motion(k).T = T;
      [Acum,Tcum] = accumulatewarp( Acum, Tcum, A, T );
      roi = warp( roiorig, Acum, Tcum );
   end

   %%% STABILIZE TO LAST FRAME
   stable(N).im = frames(N).im;
   Acum = [1 0 ; 0 1];
   Tcum = [0 ; 0];
   for k = N-1 : -1 : 1
      [Acum,Tcum] = accumulatewarp( Acum, Tcum, motion(k).A, motion(k).T );
      stable(k).im = warp( frames(k).im, Acum, Tcum );
   end
% ------------------------------------------------------------------------
%%% ALIGN TWO FRAMES (f2 to f1)

function[ Acum, Tcum ] = opticalflow( f1, f2, roi, L )

   f2orig = f2;
   Acum = [1 0 ; 0 1];
   Tcum = [0 ; 0];

   for k = L : -1 : 0
      %%% DOWN-SAMPLE
      f1d = down( f1, k );
      f2d = down( f2, k );
      ROI = down( roi, k );

      %%% COMPUTE MOTION
      [Fx,Fy,Ft] = spacetimederiv( f1d, f2d );
```

```
      [A,T] = computemotion( Fx, Fy, Ft, ROI );
      T = (2^k) * T;
      [Acum,Tcum] = accumulatewarp( Acum, Tcum, A, T );

      %%% WARP ACCORDING TO ESTIMATED MOTION
      f2 = warp( f2orig, Acum, Tcum );
   end
% --------------------------------------------------------------------
%%% COMPUTE MOTION

function[ A, T ] = computemotion( fx, fy, ft, roi )

   [ydim,xdim] = size(fx);
   [x,y] = meshgrid( [1:xdim]-xdim/2, [1:ydim]-ydim/2 );

   %%% TRIM EDGES
   fx   = fx( 3:end-2, 3:end-2 );
   fy   = fy( 3:end-2, 3:end-2 );
   ft   = ft( 3:end-2, 3:end-2 );
   roi  = roi( 3:end-2, 3:end-2 );
   x    = x( 3:end-2, 3:end-2 );
   y    = y( 3:end-2, 3:end-2 );

   ind  = find( roi > 0 );
   x    = x(ind); y = y(ind);
   fx   = fx(ind); fy = fy(ind); ft = ft(ind);
   xfx  = x.*fx; xfy = x.*fy; yfx  = y.*fx; yfy = y.*fy;

   M(1,1) = sum( xfx .* xfx ); M(1,2) = sum( xfx .* yfx ); M(1,3) = sum( xfx .* xfy );
   M(1,4) = sum( xfx .* yfy ); M(1,5) = sum( xfx .* fx );  M(1,6) = sum( xfx .* fy );
   M(2,1) = M(1,2);            M(2,2) = sum( yfx .* yfx ); M(2,3) = sum( yfx .* xfy );
   M(2,4) = sum( yfx .* yfy ); M(2,5) = sum( yfx .* fx );  M(2,6) = sum( yfx .* fy );
   M(3,1) = M(1,3);            M(3,2) = M(2,3);            M(3,3) = sum( xfy .* xfy );
   M(3,4) = sum( xfy .* yfy ); M(3,5) = sum( xfy .* fx );  M(3,6) = sum( xfy .* fy );
   M(4,1) = M(1,4);            M(4,2) = M(2,4);            M(4,3) = M(3,4);
   M(4,4) = sum( yfy .* yfy ); M(4,5) = sum( yfy .* fx );  M(4,6) = sum( yfy .* fy );
   M(5,1) = M(1,5);            M(5,2) = M(2,5);            M(5,3) = M(3,5);
   M(5,4) = M(4,5);            M(5,5) = sum( fx .* fx );   M(5,6) = sum( fx .* fy );
   M(6,1) = M(1,6);            M(6,2) = M(2,6);            M(6,3) = M(3,6);
   M(6,4) = M(4,6);            M(6,5) = M(5,6);            M(6,6) = sum( fy .* fy );

   k = ft + xfx + yfy;
   b(1) = sum( k .* xfx );     b(2) = sum( k .* yfx );
   b(3) = sum( k .* xfy );     b(4) = sum( k .* yfy );
   b(5) = sum( k .* fx );      b(6) = sum( k .* fy );

   v = inv(M) * b';
   A = [v(1) v(2) ; v(3) v(4)];
   T = [v(5) ; v(6)];
```

```
% --------------------------------------------------------------------
%%% WARP IMAGE

function[ f2 ] = warp( f, A, T )

   [ydim,xdim] = size( f );
   [xramp,yramp] = meshgrid( [1:xdim]-xdim/2, [1:ydim]-ydim/2 );

   P       = [xramp(:)' ; yramp(:)'];
   P       = A * P;
   xramp2  = reshape( P(1,:), ydim, xdim ) + T(1);
   yramp2  = reshape( P(2,:), ydim, xdim ) + T(2);
   f2      = interp2( xramp, yramp, f, xramp2, yramp2, 'bicubic' ); % warp
   ind     = find( isnan(f2) );
   f2(ind) = 0;
% --------------------------------------------------------------------
%%% BLUR AND DOWNSAMPLE (L times)

function[ f ]  = down( f, L );

   blur = [1 2 1]/4;
   for k = 1 : L
      f = conv2( conv2( f, blur, 'same' ), blur', 'same' );
      f = f(1:2:end,1:2:end);
   end
% --------------------------------------------------------------------
%%% SPACE/TIME DERIVATIVES

function[ fx, fy, ft ] = spacetimederiv( f1, f2 )

   %%% DERIVATIVE FILTERS
   pre   = [0.5 0.5];
   deriv = [0.5 -0.5];

   %%% SPACE/TIME DERIVATIVES
   fpt = pre(1)*f1 + pre(2)*f2; % pre-filter in time
   fdt = deriv(1)*f1 + deriv(2)*f2; % differentiate in time

   fx = conv2( conv2( fpt, pre', 'same' ), deriv, 'same' );
   fy = conv2( conv2( fpt, pre, 'same' ), deriv', 'same' );
   ft = conv2( conv2( fdt, pre', 'same' ), pre, 'same' );
% --------------------------------------------------------------------
%%% ACCUMULATE WARPS

function[ A2, T2 ] = accumulatewarp( Acum, Tcum, A, T )

   A2 = A * Acum;
   T2 = A*Tcum + T;
```

## Appendix B

Below is Matlab code for enhancing a stabilized video sequence. The function `videoenhance` takes as input a data structure `stable` with field `im` that contains a stabilized video sequence (the output of `videostabilize`, Appendix A). The output of this function are two images `tempmean` and `tempmedian` that contain the results of temporally filtering the stabilized video.

```
% --------------------------------------------------------------------
%%% ENHANCE STABILIZED VIDEO

function[ tempmean, tempmedian ] = videoenhance( stable )

   N = length( stable );
   [ydim,xdim] = size( stable(1).im );

   %%% BUILD A 3-D IMAGE STACK FROM THE INPUT SEQUENCE
   stack = zeros( ydim, xdim, N )
   for k = 1 : N
      stack(:,:,k) = stable(k).im;
   end

   %%% FILTER
   tempmean   = mean( stack, 3 );
   tempmedian = median( stack, 3 );
```