CAMERA JITTER COMPENSATION

Huseyin Ozkan Jonathan Tang

May 4, 2008

Boston University

Department of Electrical and Computer Engineering

Technical report No. ECE-2008-02

BOSTON
UNIVERSITY

CAMERA JITTER COMPENSATION

Huseyin Ozkan Jonathan Tang

Boston University
Department of Electrical and Computer Engineering
8 Saint Mary's Street
Boston, MA 02215
www.bu.edu/ece

May 4, 2008

Technical report No. ECE-2008-02

Summary

This project was completed as a part of the course EC 720 entitled “Digital Video Processing”. The objective of this project was to compensate the jitter found in the PTZ cameras on the roof of Photonics. Since the jitter inherent in the capture process can be viewed as global motion, we implemented a global motion compensation method which is based on perspective projective modeling of the motion. Because this motion model ignores the dept variations in a scene, we assumed that the observation in a video is sufficiently away from the camera center so that assuming planarity in the scenes is reasonable. The implemented technique in this project was designed to minimize the mean square error (MSE) between the frames of a video sequence and a chosen reference frame among them with which all of the other frames are to be aligned. To achieve this, gradient descent algorithm was applied because of the nonlinearity of the minimization. And lastly, since the jitter in the cameras is due to small amounts of camera movements: track-boom-pan-tilt-roll, it may be approximated, to a great amount, as mostly global translations and less rotations in the imaging plane of the cameras. This, indeed, was proved by our simulation results.

Contents

List of figures

1 Introduction

Camera Jitter is an issue with practically all video acquisition setups. The result is shaky video that is both unpleasing to the eye and detrimental to the performance of local motion estimation, motion detection, and a host of other video processing algorithms. Successfully compensating for the jitter will smooth out the video and improve the performance of the aforementioned algorithms. Our project focuses on compensating the jitter in the cameras installed on the roof of the Photonics building.

Fig. 1 PTZ Cameras

Fig. 1 shows the PTZ cameras that overlook Commonwealth Avenue and MIT. Note the rigid mounting structure. This particular setup will restrict the motion of the cameras and this highly constrained movement can be accurately modeled by a relatively simple model of motion.

1.1 Literature Review

In [1] a basic layout for the global motion estimation is proposed. Our work mimics the main ideas of this paper with a few small variations that will be

discussed in the **Problem Statement** and **Implementation** sections. In the paper, there were four stages implemented to achieve global motion compensation. First the input images were downsized. On each of the downsized image pairs an initial matching was performed using a modified n-step search. The initial estimate provides a starting point for the gradient descent algorithm. Upon convergence of the gradient descent algorithm at one level, the motion parameters were scaled to correspond to the up-sampled image pairs and the gradient descent algorithm was repeated. This procedure was repeated once more forming a 3 level pyramid approach to global motion estimation. Convergence of the gradient descent algorithm was achieved when one of two conditions was met: The update term of the previous iteration was less than some threshold or the number of iterations surpassed some threshold.

The ideas in [2] provided valuable insight in the actual implementation of the gradient descent algorithm, however; since we implemented our algorithm using MATLAB's pre-built functions, most of the techniques proposed in this paper were not necessary.

The interpolating kernel proposed in [3] provided the foundation of our gradient descent algorithm. The derivative of the kernel,

$$u(s) = \begin{cases} \frac{3}{2}|s|^3 - \frac{5}{2}|s|^2 + 1 & 0 < |s| < 1 \\ -\frac{1}{2}|s|^3 + \frac{5}{2}|s|^2 - 4|s| + 2 & 1 < |s| < 2 \\ 0 & 2 < |s|. \end{cases}$$

can be used to create the Hessian matrix needed for the update term in the gradient descent algorithm that will be discussed in **Implementation**.

1.2 Problem Statement

Our task is to compensate the jitter found in the PTZ cameras shown in Fig. *1* above. Due to the highly constrained movement of the cameras combined with the long distance at which we are capturing video, we expect that the jitter inherent in the capture process consists mostly of translation. To prove this point

we will assume different degrees of motion complexity and compare the results of each implementation.

2 Implementation

In this project, the implemented technique is designed to minimize the mean square error (MSE) between the chosen reference frame and the other "motion compensated" frames in a given video sequence which is suffering from camera jittering. MSE is given as follows:

$$E=\sum_{i=1}^{N} e_i^{\,2}, \text{where } e_i = I'(x_i', y_i') - I(x_i, y_i)$$

In the above expression,

(x_i, y_i) denotes the spatial coordinates of the i^{th} pixel in the referance frame and similarly, (x'_i, y'_i) denotes the location of corresponding pixel in the compansated frame. The summation is carried out over N pairs of pixels.

Hence, basically, an error minimization problem was attempted to solve a defined mapping between the pairs $(x_i, y_i) \& (x'_i, y'_i)$. This mapping stems from a perspective motion model which is, in the most general case, an 8-parameter non-linear model.

2.1 Motion Model

In the project, the relationship between the pairs $(x_i, y_i) \& (x'_i, y'_i)$ or the camera jittering was described by a perspective projective motion model which is defined as follows:

$$x_i' = (a_0 + a_2 x_i + a_3 y_i)/(a_6 x_i + a_7 y_i + 1)$$
$$y_i' = (a_1 + a_4 x_i + a_5 y_i)/(a_6 x_i + a_7 y_i + 1)$$

where $(a_0,...,a_7)$ are the motion parameters. Obviously, the model here is not dependent on the depth variations, whereas, perspective projection suggests that it must be so. Therefore, as the model implies, the depth variations in the 3-D space were neglected under the following assumptions:

- The scene in the video sequence is far away from the camera center.
- The objects in the scene (actually, the scene itself) are almost planar.

So that, the above model does not suffer from depth issues and becomes a suitable approximation.

Moreover; when the set up of the Photonics cameras are considered and when some sample video sequences are carefully watched, it is easy to conclude that the cameras go under a motion which is a combination of small amounts of track-boom-pan-tilt-zoom (and maybe a little bit of roll). All of these camera movements, under the above assumptions and when they are not large movements, can be regarded as a combination of simple global translation, rotation, and zoom on the imaging plane. So for such cases, it is useful to apply the following reductions to the above 8-parameter motion model:

2-parameter model: If one sets $a_2 = a_5 = 1,\ a_3 = a_4 = a_6 = a_7 = 0$, the most general case reduces to 2-paramter model which only accounts for simple global translation on the imaging plane.

Similarly,

4-parameter model: If one does the necessary eliminations (a2=a5, a3=-a4, a6=a7=0), then the 8-parameter model turns out to be 4-parameter model which only accounts for simple global translation-rotation and zooming activity on the imaging plane.

Hence, in the project, 2-parameter and 4-parameter models are assumed and an MSE minimization algorithm implemented on each.

2.2 Gradient Descent

Based on the above motion model, the goal of the project was solving for the motion parameters via minimizing the following cost function which is essentially the MSE between the chosen reference frame and compensated frames in a given video sequence suffering from camera jittering.

$$E=\sum_{i=1}^{N} e_i^{\,2}, \text{where } e_i = I'(x_i', y_i') - I(x_i, y_i)$$

The dependency of this cost function on motion parameters are resulted from the relation between the pairs $(x_i, y_i) \& (x'_i, y'_i)$ which is given by the above motion model. It is easy to notice that, this cost function is a non-linear function of the motion parameters and so minimization with respect to these parameters is not trivial. Therefore, gradient descent algorithm was applied to solve this non-linear minimization problem. This can be given by the following iterative procedure:

$$\mathbf{a}^{(t+1)} = \mathbf{a}^{(t)} + \mathbf{H}^{-1}\mathbf{b}$$

where $\mathbf{a}^{(t+1)}$ and $\mathbf{a}^{(t)}$ denote the motion parameters, $(a_0,...,a_n)$, suggested by the iterations t+1 and t respectively. $\mathbf{H}$ is an *nxn* matrix equals to one-half times of the Hessian matrix of E and is usually referred to as the curvature matrix, b is an n-element vector equals to minus one-half times the gradient of E, and n refers to the number of the parameters of the model. More specifically,

$$H_{kl} = \frac{1}{2}\sum_{i=1}^{N} \frac{\delta^2 e_i^2}{\delta a_k \delta a_l} \approx \sum_{i=1}^{N} \frac{\delta e_i}{\delta a_k}\frac{\delta e_i}{\delta a_l}$$

$$b_k = -\frac{1}{2}\sum_{i=1}^{N} \frac{\delta e_i^2}{\delta a_k} = -\sum_{i=1}^{N} e_i \frac{\delta e_i}{\delta a_k}$$

The approximation in the calculation of the Hessian matrix of E does hold when the second order derivatives are significantly smaller then the first order derivatives and the latter equality in the calculation of gradient vector of E does hold when the cost function is in the form of ‘sum of squares’. Both are the valid cases in this project and so with these modified formulas, the gradient descent is

called *Newton-Ralphson method* which is widely used in error minimization problems.

In this iterative loop, the $\mathbf{H}^{-1}\mathbf{b}$ is the "updating term" and at each iteration it is used for renewing the latest updated motion parameters or in other words, is standing for the direction at which the gradient descent algorithm is tracing on the multidimensional surface of the cost function to locate a minima. The stopping criterion of this iterative procedure is as follows:

- At each iteration, if the magnitude of the update term is sufficiently small (meaning that it is checked with some predefined threshold and if it is below that threshold) the iteration is ended. This implies that, the convergence issue of the gradient descent was considered at this point. Being sufficiently small is equivalent to convergence.

- However, the algorithm does not have to be convergent. In this case, to get rid of the infinite loop issue, a maximum number of iteration was defined and above which the algorithm is not allowed to go.

This iterative procedure starts with an initial guess which is an extremely important issue. This is because the gradient descent algorithm does not have to find the global minima but it is very likely for the algorithm that it finds a local minimum of the cost function. However, in the project, the goal was minimizing the cost function (MSE) as much as possible which requires finding the global minima. The gradient descent algorithm will find the closest minimum which way be a local minimum. To ensure convergence on the global minimum, the initial estimate must start the motion parameters within the basin of the global minimum. This is shown in Fig. *2* below. If the initial estimate outputs a motion parameter indicated by the blue dot, the gradient descent will find the global minimum. If the initial estimate outputs a motion parameter indicated by a red dot, the gradient descent algorithm will find a local minimum.

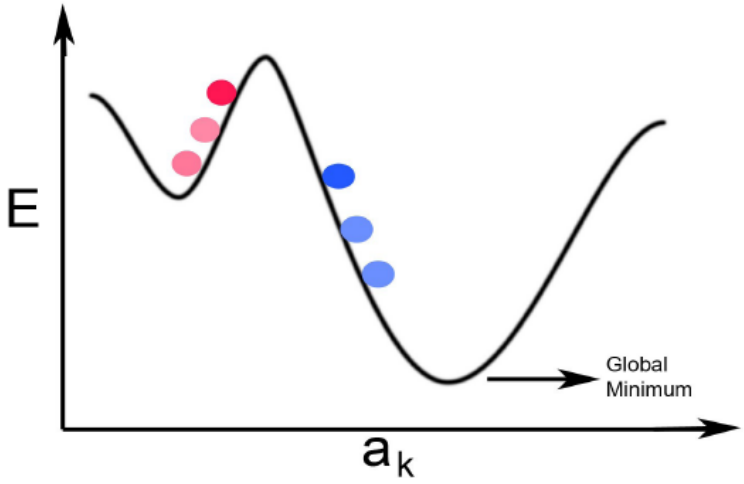

Fig. 2 Basin of Global Minimum

The initial estimate is found using the phase correlation method described in [4] and shown below. We chose this method instead of the n step search proposed in [1] because phase correlation has proven to be a quick and accurate method of finding shifts in an image. The formulation of phase correlation method for full pixel accuracy is as follows:

$$\psi_1(\boldsymbol{x}) = \psi_2(\boldsymbol{x} + \boldsymbol{d})$$

$$\Psi_1(\boldsymbol{f}) = \Psi_2(\boldsymbol{f}) \cdot e^{(j2\pi \boldsymbol{d}^T \boldsymbol{f})}$$

$$\tilde{\Psi}(\boldsymbol{f}) = \frac{\Psi_1(\boldsymbol{f})\Psi_2'(\boldsymbol{f})}{\left\|\Psi_1(\boldsymbol{f})\Psi_2'(\boldsymbol{f})\right\|} = e^{(j2\pi \boldsymbol{d}^T \boldsymbol{f})}$$

$$PCF(\boldsymbol{x}) = FT^{-1}[\tilde{\Psi}(\boldsymbol{f})] = \delta(\boldsymbol{x} + \boldsymbol{d})$$

2.3 Algorithm

The final picture of the algorithm implemented in this project is seen in Fig. *3* below:

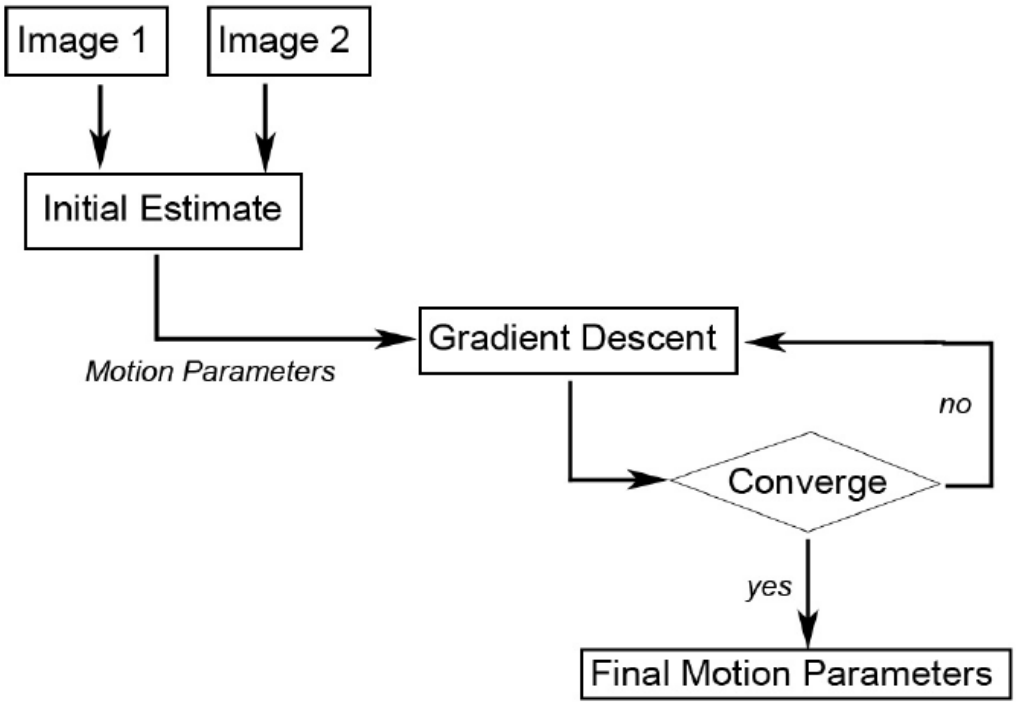

Fig. 3 Algorithm Block Diagram

In the above block diagram, “image1” may be the reference frame and “image2” may be any frame from the video sequence. Phase correlation method is applied to these images to get the initial guess for the gradient descent algorithm which applies an iterative procedure for minimizing the cost function (MSE) with respect to motion model parameters. So at the end, this iterative procedure gives the final, latest updated motion model parameters between the images that the whole algorithm started with. Essentially, this motion model defines a motion vector field between frames which is applied to the frame that is wanted to be aligned with the reference frame. One important thing here is that, the motion vectors do not have to point to integer locations. In other words, the motion vector field offered by the motion model used in this project is defined on Real Numbers. For this reason, in the process of compensation of the frames, bi-cubic interpolation was used to estimate the intensity values at non integer locations. This surely required an intense computation. However, this can be overcome by down-sampling the whole video sequence before everything and then up-sampling after everything. In this project, time and computational

complexity was not considered but we believe down-sampling and up-sampling will improve computation speed.

Before proceeding with the experimental results, a test result obtained when the algorithm (4-parameter model) is run with the Barbara image and rotated & translated Barbara image (rotation : 5 degree counter clock wise and translation: 5 pixels in both vertical and horizontal directions) is shown in Fig. *4* below:

Fig. 4. Barbara Image

It is interesting to note that depth plays a very little role in the accuracy of our compensation for “Barbara”. For instance, although the book shelf in the image is behind Barbara (where perspective projection suggests that the shelf must move less than Barbara under actual camera motion), both Barbara and the book shelf was moved by the same amount. Therefore, the assumptions of the motion model do hold perfectly here. For this reason, the MSE for compensated image is so small when compared to the MSE for the uncompensated one. This result may seem to be not interesting since the assumptions hold perfectly, but it was important to see what the algorithm (4-parameter) is capable of doing at most. Here, 4-parameter motion model was used and so it was able to compensate the rotation however and obviously, neither the phase correlation method nor the 2-parameter model is able to do that.

3 Experimental Results

We compare three different algorithms in this paper. The first algorithm is phase correlation (PC) at full pixel accuracy. This corresponds to using our initial estimate without improvements from the gradient descent algorithm. The second algorithm is PC combined with the 2 parameter model which describes motion at sub-pixel accuracy. This algorithm, like PC alone, can only find image matches that differ by pure translation. The third algorithm is PC combined with the 4 parameter model which incorporates the possibility of translation, zoom and rotation as movements of the camera.

Two quantitative error metrics we used to compare the efficacy of each algorithm were mean-squared-error (MSE) and pixel progression variance (PPV). MSE is calculated by taking the sum of squared differences (SSD) between the current frame and a reference frame and dividing that value by the number of pixels in the image and the number of frames in the sequence. This is shown below.

$$MSE = \frac{1}{N}\frac{1}{M}\sum_{i=1}^{N} e_i^{\,2}, \; where \; e_i = I'(x_i', y_i') - I_r(x_i, y_i), \; M = number \; of \; frames$$

Since squared error is what we are minimizing, MSE was a natural choice for a numerical metric. Comparatively, lower MSE corresponds to more accurate algorithms.

PPV describes the behavior of individual pixels through the entire image sequence. Fig. *5* shown below shows the intensity fluctuations of the pixel located at row = 121, col = 177 (middle of the image). The mean of the waveform was subtracted so that the fluctuations are centered on zero. This particular pixel is well behaved with a standard deviation of only 1.23. In practice we append the intensity waveform (with the mean subtracted) of each pixel to the end of the previous pixel for all pixels in the image creating a very long waveform.

The variance or standard deviation is then taken over the complete waveform and labeled as PPV (or the square root of PPV as the case may be).

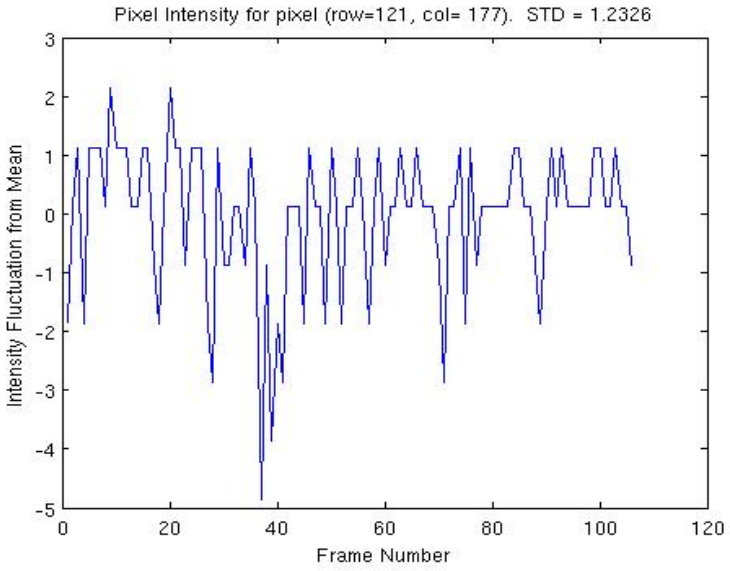

Fig. 5 Pixel Progression Variance

The results of the MSE and PPV for our three different algorithms are shown in Figure 6 below.

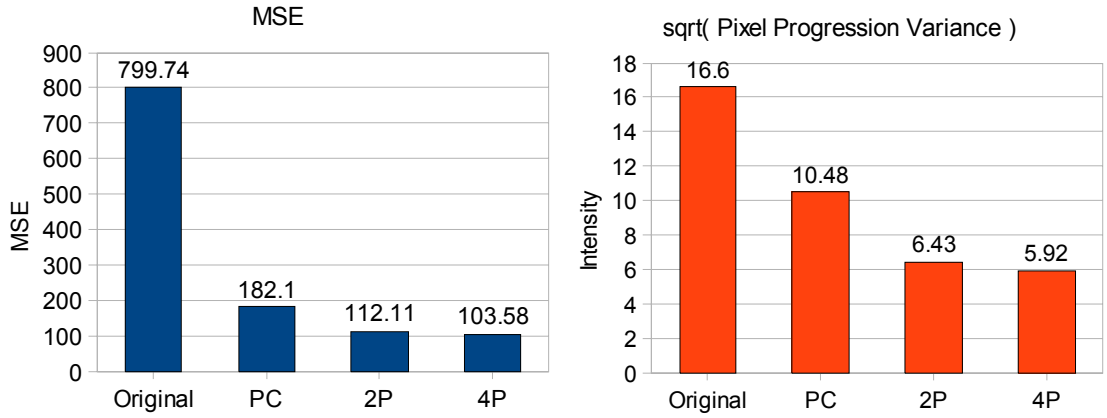

Fig. 6 MSE and PPV

The original video sequence has a very large MSE and PPV as expected due to the jittery nature of the video. PC at full pixel accuracy reduces the MSE

and PPV significantly. The 2 parameter model further reduces MSE and PPV and the 4 parameter model reduces these measures further. Note the small gain between the 2 parameter and 4 parameter models. This small gain indicates the existence of zoom and rotation is very small and that the global motion in the PTZ cameras is mostly translation.

4 Conclusions

The results described in the previous section indicate that the motion in the PTZ cameras on the roof of the Photonics Building suffer from jitter that consists primarily of translation. Zoom and rotation exist but their effects are miniscule when compared to the effects of translation.

The primary deficiency in our implementation is run time. We have implemented the gradient descent algorithm very inefficiently and so it takes roughly 1 hour to compensate the jitter in 105 frames. By taking advantage of advanced filter techniques as opposed to calculating values on a pixel-by-pixel basis, the run time of our algorithm can be significantly reduced, possibly to times that allow for real-time implementation.

5 Appendix

```
% the MATLAB code for the project camera jittering compensation
% by Jonathan and Huseyin

% *************** main **************
clear all
close all
clc
load ('data.mat');
iternum = 15;
scale = 1;
teta = 0;
n = 105;
parameters = zeros(n,4);
image1 = im_mat(:,:,1);
[X Y] = grids(image1);
for i = 1:n
image2 = im_mat(:,:,i);
r = imcorr(image1,image2); % initial coming from phase correlation
initial = maxima(r,1,X,Y);
initials = [initial(2),initial(1),1,0];
parameters(i,:) = oneframe(image2,image1,initials,iternum,scale); %frame by
frame
processing
end
% ************************ functions *********************
% 1)
function result = oneframe(image1,image2,initials,number,scale)
a = initials';
E = 1000;
for iternum =1:number
[update temp] = Hinvb(image1,image2,a(1),a(2),a(3),a(4));
if (temp > E && scale > 1)
a = a_prev;
scale = ceil(scale / 2)
else
update = update * scale;
a_prev = a;
a = a + update;
E = temp
end
result = a;
end
% 2)
function [result1 result2] = Hinvb(image1,image2,a0,a1,a2,a3
[m n] = size(image2);
mx = ceil(max(abs(a0),abs(a1)))+3;
H = zeros(4,4);
b = zeros(4,1);
E = 0;
count = 0;
for i =1:m
for j = 1:n
result = degis(i,j,a0,a1,a2,a3);
k = result(1);
l = result(2);
if ((k > 2 && k < m-1) && (l > 2 && l <n-1))
e = interpolate(image1,[k,l]) - image2(i,j);
[da0 da1] = derv(image1,[k,l]);
da2 = da0*j+da1*i;
da3 = da0*i-da1*j;
H = H + ([da0 da1 da2 da3]' * [da0 da1 da2 da3]);
```

```
b = b + -1*e*[da0 da1 da2 da3]';
E = E + e^2;
count = count+1;
end
end
end
E =E / count;
E = sqrt(E);
result1 = pinv(H)*b;
result2 = E;
% 3)
function [result] = interpolate(image,point)
m = floor(point(1));
n = floor(point(2));
s = point(1) - m;
A1 = (-1*s^3+2*s^2-s) / 2 ;
A2 = (3*s^3-5*s^2+2) / 2 ;
A3 = (-3*s^3+4*s^2+s) / 2 ;
A4 = (s^3-s^2) / 2;
resultx = [];
for i=0:3
resultx = [resultx,
image(m-1,n-1+i)*A1+image(m,n-1+i)*A2+image(m+1,n-1+i)*A3+image(m+2,n-
1+i)*A4];
end
s = point(2) - n;
A1 = (-1*s^3+2*s^2-s) / 2 ;
A2 = (3*s^3-5*s^2+2) / 2 ;
A3 = (-3*s^3+4*s^2+s) / 2 ;
A4 = (s^3-s^2) / 2;
result = [resultx(1)*A1+resultx(2)*A2+resultx(3)*A3+resultx(4)*A4];
% 4)
function [dx dy] = derv(image,point)
[resultx resulty] = int(image,point);
s = point(2) - floor(point(2));
D1 = (-3*s^2+4*s-1) * 1/2 ;
D2 = (9*s^2-10*s) * 1/2;
D3 = (-9*s^2+8*s+1) * 1/2;
D4 = (3*s^2-2*s) * 1/2;
dx = resultx*[D1 D2 D3 D4]';
s = point(1) - floor(point(1));
D1 = (-3*s^2+4*s-1) * 1/2 ;
D2 = (9*s^2-10*s) * 1/2;
D3 = (-9*s^2+8*s+1) * 1/2;
D4 = (3*s^2-2*s) * 1/2;
dy = resulty*[D1 D2 D3 D4]';
% 5)
function [resultx resulty] = int(image,point)
m = floor(point(1));
n = floor(point(2));
s = point(1) - m;
A1 = (-1*s^3+2*s^2-s) / 2 ;
A2 = (3*s^3-5*s^2+2) / 2 ;
A3 = (-3*s^3+4*s^2+s) / 2 ;
A4 = (s^3-s^2) / 2;
resultx = [];
for i=0:3
resultx = [resultx,
image(m-1,n-1+i)*A1+image(m,n-1+i)*A2+image(m+1,n-1+i)*A3+image(m+2,n-
1+i)*A4];
end
s = point(2) - n;
A1 = (-1*s^3+2*s^2-s) / 2 ;
A2 = (3*s^3-5*s^2+2) / 2 ;
A3 = (-3*s^3+4*s^2+s) / 2 ;
A4 = (s^3-s^2) / 2;
resulty = [];
```

```
for i=0:3
resulty = [resulty,
image(m-1+i,n-1)*A1+image(m-1+i,n)*A2+image(m-1+i,n+1)*A3+image(m-1+i,
n+2)*A4];
end
% 6)
function result = degis(m,n,a0,a1,a2,a3)
l = a0 + a2*n + a3*m;
k = a1 - a3*n + a2*m;
result = [k l];
% 7)
function result = apply_motion(data,parameters)
[m n num] = size(data);
reference = data(:,:,1);
result = zeros(m,n,num);
for frame = 1:num
frame
for i=1:m
for j=1:n
a0 = parameters(frame,1);
a1 = parameters(frame,2);
a2 = parameters(frame,3);
a3 = parameters(frame,4);
image = data(:,:,frame);
temp = degis(i,j,a0,a1,a2,a3);
k = temp(1);
l = temp(2);
if(floor(k) > 1 && floor(k) < m-1 && floor(l) > 1 && floor(l) < n-1)
result(i,j,frame) = interpolate(image,[k,l]);
end
end
end
end
% 8)
function result = imcorr(image1, image2)
f1 = fft2(image1);
f2 = fft2(image2);
f = exp(i*(angle(f2) - angle(f1)));
f = real(ifft2(f));
result = ifftshift(f);
% 9)
function result = maxima(data,n,X,Y)
result = zeros(n,2);
mn = min(min(data));
x = X(:);
y = Y(:);
for i = 1:n
temp = find(data == max(max(data)));
temp = temp(1);
result(i,:) = [y(temp),x(temp)];
data(temp) = mn;
end


function [MSE,frStd]=MSEfrVar(imseq)
% MSE and Inter-Frame Variance
% assumes a Matlab movie input
n=105; % number of frames
sth = 338;
% h = size(imseq(10).cdata(:,:,1),1);
% w = size(imseq(10).cdata(:,:,1),2);
im_mat = zeros(size(imseq(10).cdata(:,:,1),1),size(imseq(10).cdata
(:,:,1),2)-sth,n);
for i=1:n
im_mat(:,:,i)=imseq(i).cdata(:,sth+1:size(imseq(10).cdata
(:,:,1),2),1);
end
```

```
h = size(im_mat,1);
w = size(im_mat,2);
N = w*h;
MSEs=zeros(1,n-1);
for i=2:n
MSEs(i) = sum(sum((im_mat(:,:,i)-im_mat(:,:,1)).^2))/(N);
end
MSE = sum(MSEs)/n;
k=1;
imseqLine=zeros(1,size(im_mat,1)*size(im_mat,2)*size(im_mat,3));
pline = zeros(1,n);
for j=1:size(im_mat,1)
for i=1:size(im_mat,2)
pline(:) = im_mat(j,i,:);
imseqLine(k:k+n-1) = pline - mean(pline);
k=k+n;
end
end
x = 1:length(imseqLine);
%x2 = 1:n:length(imseqLine);
frStd = std(imseqLine);
subplot(3,1,1)
plot(imseqLine(round((w/2))*round((h/2))*n:round((w/2))*round((h/2))
*n+n))
title(['Pixel Intensity for pixel (row=',num2str(round(h/2)),', col=
',num2str(round(w/2)),'). STD = ',num2str(std(imseqLine(round((w/2))
*round((h/2))*n:round((w/2))*round((h/2))*n+n)))])
xlabel('Frame Number')
ylabel('Intensity Fluctuation from Mean')
subplot(3,1,2)
plot(imseqLine(1:length(imseqLine)))
title(['Pixel Intensity STD: ',num2str(frStd)])
subplot(3,1,3)
plot(imseqLine(1:1000:length(imseqLine)))
title('Pixel Intensity (subsampled)')%,num2str(frStd)])
```

References

[1] F. Dufaux and J. Konrad. "Efficient, Robust and Fast Global Motion Estimation for Video Coding". IEEE Transactions on Image Processing, Vol. 9. No.3, March 2000.

[2] W.H. Press, B.P. Flannery, S.A. Teukolsky, and W.T. Vetterling, *Numerical Recipes in C: The Art of Scientific Computing,* Cambridge Univ. Press, Cambridge, 1988.

[3] Keys, Robert G. "Cubic Convolution Interpolation for Digital Image Processing". IEEE Transactions on Acoustics, Speech, and Signal Processing Vol. ASSP-29, No.6, Decemer 1981.

[4] Yao Wang, Jorn Ostermann, and Ya-Qin Zhang, *Video Processing and Communications*. Prentice-Hall, Inc. New Jersey, 2002.