Fall 2009 ELEC301 Group Project Report: Video Stabilization

By:

Jeffrey Bridge
Robert Brockman
Stamatios Mastrogiannis

Fall 2009 ELEC301 Group Project Report: Video Stabilization

By:
Jeffrey Bridge
Robert Brockman
Stamatios Mastrogiannis

Online:
< http://cnx.org/content/col11152/1.1/ >

C O N N E X I O N S

Rice University, Houston, Texas

Collection structure revised: December 21, 2009
PDF generated: February 6, 2011
For copyright and attribution information for the modules contained in this collection, see p. 23.

Table of Contents

Chapter 1

Introduction[1]

Introduction
A common problem in dealing with Unmanned Aerial Vehicles (UAVs) is image stabilization. If an operator wishes to control the craft in real-time, a camera mounted on the UAV is often a good solution. This video feed, if left in its original state, has varying amounts of jitter, which in turn makes operating the craft more difficult and makes the footage of the flight much less pleasant to watch. We decided that we could stabilize the video without using any additional hardware-based assistance (such as gyroscopes) with the digital signal processing techniques we've learned over the semester. Our first approach to solving this problem was to correlate each video frame with the previous one, but this proved to be less than optimal ; there exists a faster, more accurate technique. Enter KLT feature tracking and Serial Affine Transformation. We used a freely-available KLT feature tracker from Stan Birchfield, then prototyped our affine transformation techniques in MATLAB. We have started porting our work to C, and in the future we expect this sort of solution to be fully implemented on GPUs for real-time processing.

[1] This content is available online at <http://cnx.org/content/m33246/1.1/>.

Chapter 2

Background[1]

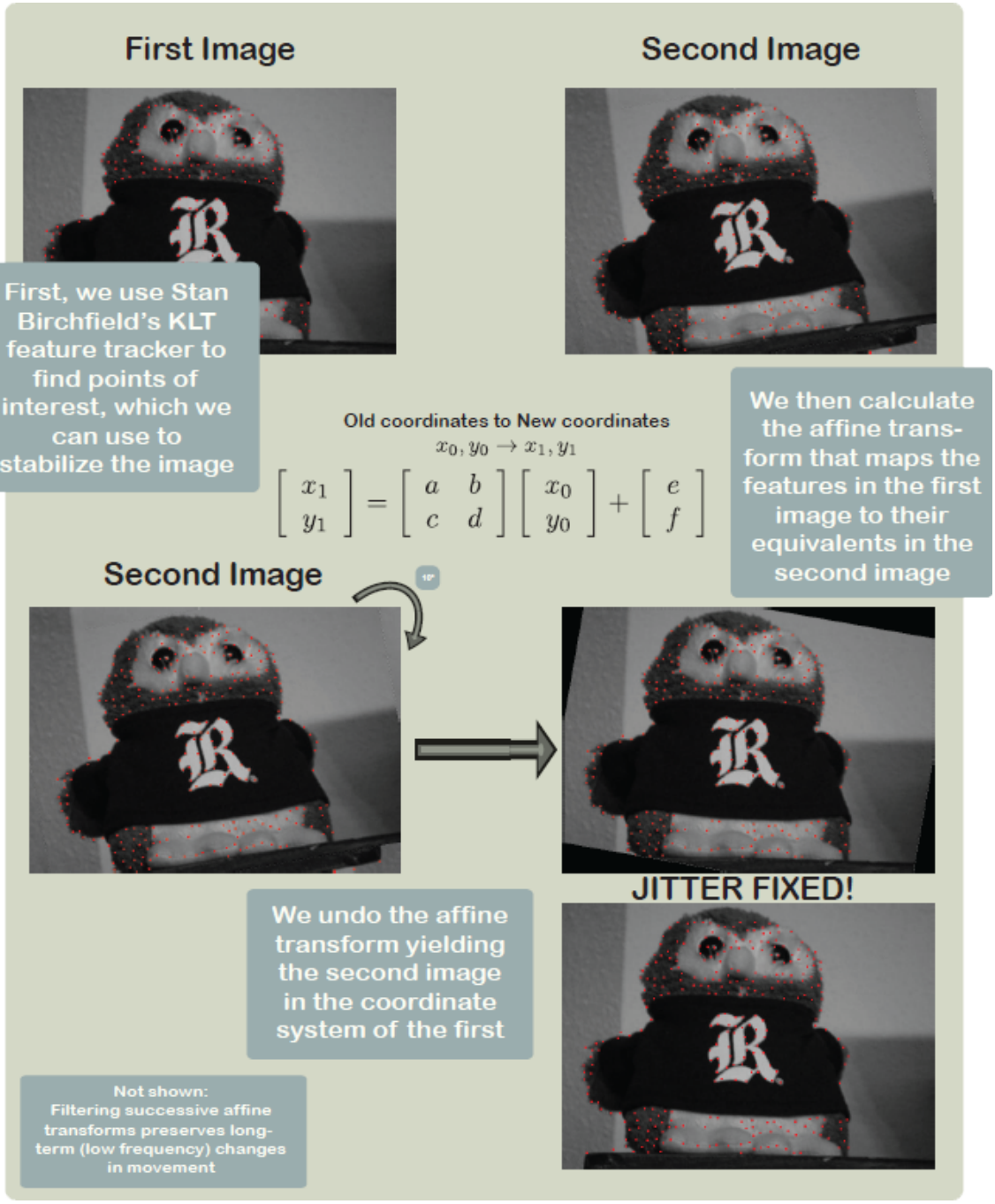
Background
Image stabilization can be done in many different ways. Kanade-Lucas-Tomasi (KLT) feature tracking[2] is one of the computationally inexpensive ways, in comparison to 2-D correlation and even SIFT. We chose Stan Birchfield's implementation because it is written in C and we found it easy to interface to in comparison with other open-source implementations.

When we have a set of common features between two images, we can 'undo' the transformation that makes the second image's features reside in a different location than the first, creating a new image whose features have similar locations to those in the first image.

In order to accomplish this, we use a series of least-squares affine transformations on the set of features to determine the 'best' values for the un-affine we perform to correct the later image. After this, we then filter the resulting affine transformation, keeping the low-frequency movement (such as panning) and removing the high-frequency jitter.

Pictorally, the process is as such:

[1] This content is available online at <http://cnx.org/content/m33247/1.1/>.
[2] http://en.wikipedia.org/wiki/Kanade-Lucas-Tomasi_Feature_Tracker


First Image
Second Image
First, we use Stan Birchfield's KLT feature tracker to find points of interest, which we can use to stabilize the image
Old coordinates to New coordinates
$x_0, y_0 \rightarrow x_1, y_1$
$\begin{bmatrix} x_1 \\ y_1 \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} x_0 \\ y_0 \end{bmatrix} + \begin{bmatrix} e \\ f \end{bmatrix}$
We then calculate the affine transform that maps the features in the first image to their equivalents in the second image
Second Image
JITTER FIXED!
We undo the affine transform yielding the second image in the coordinate system of the first
Not shown:
Filtering successive affine transforms preserves long-term (low frequency) changes in movement

Chapter 3

Procedures[1]

3.1 Affine Transform Estimation

We wish to approximate the movement of the feature points by an affine transform, because it can account for rotation, zooming, and panning, all of which are common features in videos. The coordinates of a feature in the old frame are written as (x_0, y_0) and in the new frame as (x_1, y_1). Then an affine transform can be written as:

$$\begin{bmatrix} x_1 \\ y_1 \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} x_0 \\ y_0 \end{bmatrix} + \begin{bmatrix} e \\ f \end{bmatrix} \tag{3.1}$$

However, this form needs some modification to deal with multiple point pairs at once, and needs rearranging to find a, b, c, d, e, and f. It can be easily verified that the form below is equivalent to the one just given:

$$\begin{bmatrix} x_0 & y_0 & 0 & 0 & 1 & 0 \\ 0 & 0 & x_0 & y_0 & 0 & 1 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \\ d \\ e \\ f \end{bmatrix} = \begin{bmatrix} x_1 \\ y_1 \end{bmatrix} \tag{3.2}$$

With this form, it is easy to add multiple feature points by stacking two additional rows on the left and on the right. Denoting the pairs of points as $\left(\left(x_0^{(1)}, y_0^{(1)}\right), \left(x_1^{(1)}, y_1^{(1)}\right)\right)$, $\left(\left(x_0^{(2)}, y_0^{(2)}\right), \left(x_1^{(2)}, y_1^{(2)}\right)\right)$,

[1] This content is available online at <http://cnx.org/content/m33251/1.1/>.

$\left(\left(x_0^{(3)}, y_0^{(3)}\right), \left(x_1^{(3)}, y_1^{(3)}\right)\right)$, etc, the matrices will now look like:

$$\begin{bmatrix} x_0^{(1)} & y_0^{(1)} & 0 & 0 & 1 & 0 \\ 0 & 0 & x_0^{(1)} & y_0^{(1)} & 0 & 1 \\ x_0^{(2)} & y_0^{(2)} & 0 & 0 & 1 & 0 \\ 0 & 0 & x_0^{(2)} & y_0^{(2)} & 0 & 1 \\ x_0^{(3)} & y_0^{(3)} & 0 & 0 & 1 & 0 \\ 0 & 0 & x_0^{(3)} & y_0^{(3)} & 0 & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \end{bmatrix} \begin{bmatrix} a \\ b \\ c \\ d \\ e \\ f \end{bmatrix} = \begin{bmatrix} x_1^{(1)} \\ y_1^{(1)} \\ x_1^{(2)} \\ y_1^{(2)} \\ x_1^{(3)} \\ y_1^{(3)} \\ \vdots \end{bmatrix} \tag{3.3}$$

So long as there are more than three points, the system of equations will be overdetermined. Therefore the objective is to find the solution $[a, b, c, d, e, f]$ in the least squares sense. This is done using the pseudoinverse of the matrix on the left.

3.2 Filtering

The affine transforms produced above only relate one video frame to the one immediately after it. The problem with this is that if the video is jerky, it will take several consecutive frames to have a good idea of what the average position of the camera is during this time. Then the difference between the current location and the moving-average location can be used to correct the current frame to be in this average position.

When the features are tracked frame-to-frame, it constitutes an implicit differentiation in terms of measuring the overall movement of the camera. In order to track changes across many frames, we sequentially accumulate the frame-to-frame differences. This is akin to an integral operator. Unfortunately, when integrating imperfect data, errors will build up linearly in time, and that is true here. However, since the stream of integrated affine transforms is not used directly, these errors are not as important.

Once the stream of integrated affine transforms is generated, the goal is to undo high-frequency motions, while leaving the low-frequency motions intact. This is done by treating the coefficients of the stream of integrated affine transforms as independent, and applying six high pass filters, one for each stream of coefficients. Although this technique works, it is hoped that a more elegant way of handling the filtering may be developed in the future.

Since a high pass filter is being used, it is important to not have large phase offsets created by the filter. If the transform which ideally stabilized frame #5 was instead applied to frame #10, and so forth, the delay would wholly invalidate the offsets, and the resulting video would be more jerky than before, instead of less. Therefore, we decided to use the zero phase filtering technique of applying a filter in both the forward and reverse time directions sequentially. This is handled by the Matlab function filtfilt.

Initially, we tried various order-4 to order-8 IIR filters with cutoff frequencies around 0.1 pi. However, the unit step response of nearly all IIR filters involves a significant amount of overshoot and ringing. Since our signal is best viewed as a time-domain signal instead of a frequency-domain signal, we sought to avoid this overshoot. Therefore, we switched to a truncated Gaussian FIR filter, which averages across a bit more than one second worth of video at a time. This removed the overshoot and ringing which had been visible with the IIR filters.

In the algorithm we used, the high pass filter is implicitly generated by using a low pass filter, then subtracting the low-pass version from the original. It would be mathematically equivalent to simply change the impulse response of the filter and skip the subtraction step.

The last wrinkle is that for affine transforms, the identity transform has the a and d coefficients equal to one, instead of zero. The high pass filter will create a stream of transforms which are centered around having all the coefficients zero. Therefore, after the high pass filter, we added 1 back to the a and d coefficients of the stream of affine transforms, so they would be centered on the identity transform.

Chapter 4

Results[1]

Results: Output Quality
We successfully used Stan Birchfeld's KLT tracker with our implementation of affine transforms in MATLAB to stabilize the sample UAV video that Aswin provided us. The video is of six cars at the end of a runway with the plane slowly circling them. There is some jitter, and evidence of a couple of dropped frames. Our filter completely removes these, but it also eliminates the perspective change caused by the movement of the plane. This introduces considerable distortion after more than about 10 seconds. High pass filtering of the affine transformation series does remove the jitter while preserving the overall motion.

UAV Footage Stabilized with KLT + Affine Transforms

This media object is a video file. Please view or download it at<uav_source.avi>
(a)
This media object is a video file. Please view or download it at<uav_stable.avi>
(b)

Figure 4.1: Source footage provided by Aswin Sankaranarayanan.

We wanted a more serious test of the jitter reduction, with more sudden motion. To do this we wrote some MATLAB code that takes an individual frame and generates a sequence of frames based on it, each with a random displacement from the original. The effect is that of a VERY jerky camera. The KLT-affine transform combination undoes this severe jitter quite nicely. We then superimposed a circular motion on top of the jitter to see if the filtered affine transformation series would preserve it while still removing the jitter. It does an acceptable job at this, although there are a few visible kinks.

Shifted Image Sequence Stabilized with KLT + Filtered Affine Transforms

This media object is a video file. Please view or download it at<owl_source.avi>
(a)
This media object is a video file. Please view or download it at<owl_stable.avi>
(b)

Figure 4.2

[1] This content is available online at <http://cnx.org/content/m33248/1.1/>.

Additional testing revealed that although the KLT tracker we used does a good job on tracking features through sudden translations, it cannot effectively deal with large sudden rotations. It loses track of all features in these cases. Hopefully this will not be an issue for our ultimate application, or we will be able to compensate for the rotation using additional input from gyros.

We also experimented with stabilizing jerky footage from movies, such as the opening scene to Saving Private Ryan. This works quite well! We invite you to test out our code on DVD-quality video and see what you think of the results. (At some point we plan to "stabilize" The Blair Witch Project so those of us prone to motion sickness can watch it without becoming ill.) Of course the output needs to be cropped somewhat to eliminate the black border caused by shifting the image: we cannot create data from nothing!

Results: Speed

Our code is not nearly fast enough for real-time use. Greyscale output at 640x480 resolution runs at about one-third of realtime, whereas color output at the same resolution is about one-tenth real time, on the 2GHz Intel Core2 Duo laptop used for testing. The biggest bottleneck right now seems to be in the interpolation used to assign pixel intensity values for the corrected frames. The KLT tracker itself is the next slowest component. Hopefully converting the code to C and/or offloading some of the work to the GPU will improve performance.

Chapter 5

Sources[1]

We'd like to thank Aswin Sankaranarayanan at Rice DSP for pointing us in the correct direction early in our work and steering us away from trouble. Also, a key piece of the project required using Stan Birchfeld's KLT feature tracker[2] and the interface code he wrote to easily move the table of features into MATLAB.

[1] This content is available online at <http://cnx.org/content/m33249/1.1/>.
[2] http://www.ces.clemson.edu/~stb/klt/

Chapter 6

The Team[1]

Jeffrey A. Bridge
Studying for a BS Electrical Engineering at Rice University in 2011. I am interested in spaceflight and hope to do more aerospace related research in the future.
Robert T. Brockman II
Rice University Computer Science, Lovett '11. I'm interested in artificial intelligence and neuroscience and hope to do graduate work in one of those fields.
Stamatios Mastrogiannis
Rice University ECE, Brown '11. I'm interested in bionics, cybernetics, and anything that brings man and machine closer together. I plan on going into medical research to further these fields.

[1] This content is available online at <http://cnx.org/content/m33250/1.1/>.

Chapter 7

Code[1]

The main pieces of code used to accomplish the stabilization are shown below. There are several addition files needed for the complete program, which are available for download instead of being shown inline:

- tracker.c[2]
- im2_jpeg.c[3]
- imload_bw.m[4]
- write_jpeg_bw.m[5]
- write_jpeg_col.m[6]

l2aff.m

```
    % Least Squares Affine Transformation
% ELEC 301 Group Project
% 11/29/2009
% Jeffrey Bridge, Robert Brockman II, Stamatios Mastrogiannis
%
% Calculate the least squares affine transformation for two corresponding
% sets of pixel locations.
% px inputs are of the form:
%[ x_1 y_1
%  x_2 y_2
%  :   :
%  x_N y_N ]
%
% [x'] = [a, b] * [x] + [e]
% [y']   [c, d]   [y]   [f]

function Aff = l2aff(pxold, pxnew)
    b = reshape(pxnew.', [], 1);
    A = makenice(pxold);
    x = pinv(A) * b; % Was psinv, our version of computing the pseudoinv
    Aff = [x(1), x(2), x(5); ...
           x(3), x(4), x(6)];
```

[1] This content is available online at <http://cnx.org/content/m33253/1.1/>.
[2] See the file at <http://cnx.org/content/m33253/latest/tracker.c>
[3] See the file at <http://cnx.org/content/m33253/latest/im2_jpeg.c>
[4] See the file at <http://cnx.org/content/m33253/latest/imload_bw.m>
[5] See the file at <http://cnx.org/content/m33253/latest/write_jpeg_bw.m>
[6] See the file at <http://cnx.org/content/m33253/latest/write_jpeg_col.m>

```
return

function A = makenice(pxold)
    [r, c] = size(pxold);
    A = zeros(2*r, 6);
    for k=1:r
        x = pxold(k,1);
        y = pxold(k,2);
        %correspond to a, b, c, d, e, f
        A(2*k-1, :) = [x, y, 0, 0, 1, 0];
        A(2*k  , :) = [0, 0, x, y, 0, 1];
    end
return
```

aff_mul.m

```
    % ELEC 301 Group Project
% 2009 December 12
% Jeffrey Bridge, Robert Brockman II, Stamatios Mastrogiannis
%
% Combine two affine transforms into one
%
% Aff = [a b e
%        c d f]
%
% [x'] = [a, b] * [x] + [e]
% [y']   [c, d]   [y]   [f]

function Aff = aff_mul(Aff2, Aff1)
a1 = Aff1(1,1);
b1 = Aff1(1,2);
c1 = Aff1(2,1);
d1 = Aff1(2,2);
e1 = Aff1(1,3);
f1 = Aff1(2,3);

a2 = Aff2(1,1);
b2 = Aff2(1,2);
c2 = Aff2(2,1);
d2 = Aff2(2,2);
e2 = Aff2(1,3);
f2 = Aff2(2,3);

Aff = [...
    a2*a1 + b2*c1, ...
    a2*b1 + b2*d1, ...
    a2*e1 + e2; ...
    c2*d1 + c2*a1, ...
    c2*b1 + d1*d2, ...
    d2*f1 + f2];
return
```

stabilize.m

```
    % Perform video stabilization on a set of jpeg images
% ELEC 301 Group Project
% 11/29/2009
% Jeffrey Bridge, Robert Brockman II, Stamatios Mastrogiannis
%
% Uses KLT features generated via track_destabilize.sh
%  or track_movie.sh
% Reads destabilized stream of jpegs from stabilize_input
% Outputs stabilized stream of jpegs to stabilize_output
%
% Use view_stabilize.sh to play back results
%
function stabilize()

% Read feature table.  x and y contain coordinates of each feature
% for each frame.  val is used to determine whether a feature has been
% replaced.
[x,y,val] = klt_read_featuretable('stabilize_input/features.txt');
% x, y are sets of column vectors, which we like.

% Extract number of features and frames from feature table.
[nFeatures, nFrames] = size(x);

invalid_inds = [];

% Each frame will have an affine transformation which allows it
% to be transformed back into the coordinates of the original frame.
% (These transforms will then be filtered to keep low-speed drift.)
Affs = zeros(nFrames,6);

% Affine transformation starts out as the identity transformation.
myAff = [1 0 0; 0 1 0];

% Iterate over all input frames
for n = 2:nFrames
    fprintf('processing features for frame %d...', n);

    % Position of features in previous frame.
    pxold = [ x(:,n-1) y(:,n-1) ];
    % Position of features in new frame.
    pxnew = [ x(:,n) y(:,n)];

    % Features which have replaced those that have left the scene
    % have non-zero values in the feature table.  These must be excluded
    % from computing our affine transformation
    ind = find(val(:,n) ~= 0);
    invalid_inds = ind;

    % These are the indices of valid rows in our feature table
    valid_inds = setdiff([1:nFeatures].', invalid_inds);
    fprintf(' only %d features left\n', length(valid_inds));
```

```
    % Extract valid features.
    valid_pxold = pxold(valid_inds,:);
    valid_pxnew = pxnew(valid_inds,:);

    % Compute affine transformation which minimizes least squares
    % difference in distances between features in the previous frame
    % vs. the new frame transformed back to the original coordinates.
    aff = l2aff(valid_pxold, valid_pxnew);

    % Combine this "frame-by-frame" transformation with those from
    % all previous frames to get an affine transformation that will
    % transform the current frame into the coordinate system of the
    % FIRST frame.

    myAff = aff_mul(aff, myAff);

    % Make the resulting transform into a vector for ease of filtering
    % and add it to the array of transforms for each frame.
    Affs(n,:) = reshape(myAff,1,[]);
end

% High-pass filter the series of affine transformations to allow low
% frequency movement (panning, etc.) to show up in the final output.
%
% We do this by first low-pass filtering the series and then subtracting
% the result from the original.
%%{
switch 2 % Choose a filter
    case 1 % Butterworth filter
        [b, a] = butter(4,.05);
    case 2 % Gaussian filter
        b = exp(-linspace(-3,3,41).^2/2);
        b = b / sum(b);
        a = [1];
    otherwise
        error('Bad filter number');
end
filter_a = a;
filter_b = b;

% Pad beginning of transformation series with identity transforms
% to eliminate startup distortion.
eyeAff = [1 0 0 1 0 0];
prepCount = 1;
filtinAffs = [eyeAff(ones(prepCount,1),:); Affs(2:end,:)];

% LFP the affine transforms TWICE, the second time in time-reversed
% sequence.  This eliminates phase distortion caused by the filter.
LpAffs = filtfilt(filter_b, filter_a, filtinAffs);
LpAffs = LpAffs(prepCount:end,:); % Remove padding
```

```
% HPF by subtracting LPF'd series from original.
Affs = Affs - LpAffs;

% Add back 1's in corners of rotation matrix component of transform
% removed by LPF. (Add back in identity transform)
Affs(:,1) = Affs(:,1) + 1;
Affs(:,4) = Affs(:,4) + 1;
%}

% Apply affine transforms to each frame to provide video stabilization.
%%{
for n = 2:nFrames
    % Get transform back into matrix form.
    aff = reshape(Affs(n,:),2,3);

    fprintf('interpolating image %d...\n', n);
    disp(aff);

    filename = sprintf('stabilize_input/D%08d.jpg', n);

    % Black and white output is 3x faster to compute.
    if 1
        A = imread(filename);

        Ar = single(A(:,:,1));
        Ag = single(A(:,:,2));
        Ab = single(A(:,:,3));

        %B is image in coordinate system of first frame.
        Br = im_unaff(Ar, aff);
        Bg = im_unaff(Ag, aff);
        Bb = im_unaff(Ab, aff);
        B = cat(3,Br,Bg,Bb);
        write_jpeg_col(B,sprintf('stabilize_output/S%08d.jpg',n));
    else
        A = imload_bw(filename);
        B = im_unaff(A, aff);
        write_jpeg_bw(B,sprintf('stabilize_output/S%08d.jpg',n));
    end
end
%}
return
```

destabilize.m

```
    % Generate Synthetic unstable test data
% ELEC 301 Group Project
% 11/29/2009
% Jeffrey Bridge, Robert Brockman II, Stamatios Mastrogiannis

function destabilize()
```

```
% Load a big source image, and split it into colors
filename = 'destabilize_input.jpg';
A = imread(filename);
Ar = single(A(:,:,1));
Ag = single(A(:,:,2));
Ab = single(A(:,:,3));

% Size of output image to generate, a subset of the source image
output_w = 560;
output_h = 400;

% Center of source image
[r,c] = size(Ar);
center_row = r/2;% - 50;
center_col = c/2;

% Number of output frames to generate
N = 300;

% Standard deviation of jerky movement in pixels
dev = 5;

% Parameters controlling slow drift
drift_radius = 10;
drift_period = 100;

for n = 1:N
    fprintf('Generating destabilized image %d...\n', n);

    % Add in slow drift of the image center
    drift_rows = drift_radius * sin(n*2*pi/drift_period);
    drift_cols = drift_radius * cos(n*2*pi/drift_period);

    % Add in fast random jerky movements
    offset_rows = floor(randn(1) * dev);
    offset_cols = floor(randn(1) * dev);

    % Calculate current image boundaries
    left = floor(center_col + drift_cols - output_w/2 + offset_cols);
    right = left + output_w - 1;
    top = floor(center_row + drift_rows - output_h/2 + offset_rows);
    bottom = top + output_h - 1;

    % Grab an offset portion of the larger image
    Br = Ar(top:bottom, left:right);
    Bg = Ag(top:bottom, left:right);
    Bb = Ab(top:bottom, left:right);

    % Save it to its own file
    B = cat(3,Br,Bg,Bb);
    write_jpeg_col(B,sprintf('destabilize_output/D%08d.jpg',n));
```

```
    % Play back with view_destabilize.sh
end

return
```

im_unaff.m

```
    % IMage UNdo an AFFine transformation
% ELEC 301 Group Project
% 11/29/2009
% Jeffrey Bridge, Robert Brockman II, Stamatios Mastrogiannis
%
% --- INPUTS ---
% Z = image matrix (2D grid of intensities)
% Aff = affine transformation
%   [a b  e
%    c d  f]
% [x'] = [a b]*[x] + [e]
% [y']   [c d] [y]   [f]
%
% --- OUTPUTS ---
% ZI = output image matrix

function ZI = im_unaff(Z, Aff)
% Extract size of image.
[r,c] = size(Z);

% Extract affine transformation coefficients.
Aa = Aff(1,1);
Ab = Aff(1,2);
Ac = Aff(2,1);
Ad = Aff(2,2);
Ae = Aff(1,3);
Af = Aff(2,3);

% generate new sets of grid points
[X0,Y0] = meshgrid(1:c, 1:r);
% XI(c,r) and YI(c,r) contain where to look in Z for the correct
% intensity value to place in the new image ZI at coordinates (r,c).
XI = Aa*X0 + Ab*Y0 + Ae;
YI = Ac*X0 + Ad*Y0 + Af;

% Since XI and YI contain non-integer values, a simple lookup will not
% suffice.  We must perform interpolation.
ZI = interp2(Z, XI, YI);

return
```

Chapter 8

Future Work[1]

Future Work
Now that we basic algorithms down, the focus should be on improving the speed so we can get real-time stabilized video feed while operating our UAV. This means converting the code to C. It may also be necessary to use KLT trackers that use the video card GPU, as well as writing an equivalent of the MATLAB interp2 that does the same.

While taking the first steps towards this conversion, we realized that our video stabilizer would make a pretty cool GStreamer plugin. GStreamer[2] is a media framework for the open-sourse Gnome desktop environment. With it, we will be able route video sources of many kinds through our stabilizer and then on to our choice of video sinks. We have already figured out how to implement a "null" plugin that just copies frames from the source to the sink already, so once our algorithms are in C using GStreamer should be easy.

If these improvements can be made, the next step will be to test the code out with live footage from our own UAV.

[1] This content is available online at <http://cnx.org/content/m33254/1.1/>.
[2] http://gstreamer.freedesktop.org/

Index of Keywords and Terms

Keywords are listed by the section with that keyword (page numbers are in parentheses). Keywords do not necessarily appear in the text of the page. They are merely associated with that section. *Ex.* apples, § 1.1 (1) **Terms** are referenced by the page they appear on. *Ex.* apples, 1

Attributions

Collection: *Fall 2009 ELEC301 Group Project Report: Video Stabilization*
Edited by: Jeffrey Bridge, Robert Brockman, Stamatios Mastrogiannis
URL: http://cnx.org/content/col11152/1.1/
License: http://creativecommons.org/licenses/by/3.0/

Module: "Introduction"
By: Robert Brockman
URL: http://cnx.org/content/m33246/1.1/
Page: 1
Copyright: Robert Brockman
License: http://creativecommons.org/licenses/by/3.0/

Module: "Background"
By: Stamatios Mastrogiannis
URL: http://cnx.org/content/m33247/1.1/
Pages: 3-4
Copyright: Stamatios Mastrogiannis
License: http://creativecommons.org/licenses/by/3.0/

Module: "Procedures"
By: Jeffrey Bridge
URL: http://cnx.org/content/m33251/1.1/
Pages: 5-6
Copyright: Jeffrey Bridge
License: http://creativecommons.org/licenses/by/3.0/

Module: "Results"
By: Robert Brockman, Jeffrey Bridge, Stamatios Mastrogiannis
URL: http://cnx.org/content/m33248/1.1/
Pages: 7-8
Copyright: Robert Brockman, Jeffrey Bridge, Stamatios Mastrogiannis
License: http://creativecommons.org/licenses/by/3.0/

Module: "Sources"
By: Robert Brockman
URL: http://cnx.org/content/m33249/1.1/
Page: 9
Copyright: Robert Brockman
License: http://creativecommons.org/licenses/by/3.0/

Module: "The Team"
By: Stamatios Mastrogiannis
URL: http://cnx.org/content/m33250/1.1/
Page: 11
Copyright: Stamatios Mastrogiannis
License: http://creativecommons.org/licenses/by/3.0/

Module: "Code"
By: Jeffrey Bridge
URL: http://cnx.org/content/m33253/1.1/
Pages: 13-19

Module: "Future Work"
By: Robert Brockman
URL: http://cnx.org/content/m33254/1.1/
Page: 21

Fall 2009 ELEC301 Group Project Report: Video Stabilization
The final report for the Rice University Fall 2009 ELEC301 Group Project on video stabilization, by Jeff Bridge, Robert Brockman II, and Stamatios Mastrogiannis.