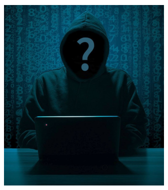
Illinois Institute of Technology Department of Electrical Engineering

# Image Stabilization

## Midterm Group Project

ECE 481: Image Processing

**- 7 -**



Clive Gomes | Dristi Chaudhuri | Nanjeeba Chowdhury

## Purpose of the Project

Image Enhancement is a family of techniques or methods that works to improve upon images. This could mean anything from improving contrast to reducing noise. Any or all methods used to improve images will fall under this branch. The goal of image enhancement, especially in this case, it is to improve the image to provide a more subjectively pleasing experience for human viewing. The goal of this project was to take an old movie, given by the professor, and to make as many repairs and enhancements, thereby making it look as good as possible using tools learned in class.

The project was done in various steps in order to achieve these goals. First, the group was required to identify what was wrong with the movie, and what would need great improvement. Before this step could be completed, the group needed to go through all of the methods that were taught in class and to see what the options were to choose from. The chosen methods to fix the old movie were decided based on what methods were taught in class. After the creation of the different filters to fix the movie, the group had to decide what order to apply the methods in. After all the methods were applied in a designated order, the movie was able to be seen with great improvements. While, even more improvements could be made to the movie, time did not allow for such enhancements . Several aspects could have been improved with more time.

### ***Requirements:***

The requirements of this project include listing aspects of the movie that the group tried to improve as well as a flowchart or block diagram of the series of image processing steps that the group chose. The group also needed to indicate the purpose of each step and why it was chosen. The next requirement was writing out the thought process that led to the final solution, including things that didn't work. Next, was what would have been attempted if there had been more time. Another requirement of the project was to describe which aspects of the project each person contributed to. Lastly, the requirements include providing all of the code in a zip file. It had to be designed so that the whole thing can be run as a script that does the entire process. The last thing the script needed to do was display the output movie. The script needs to be called filmRestore.m, so that it can easily be run.

## A First Look at the Movie

After a first glance at the movie, several observations were made. The movie has several problems associated with it. Some of these are listed below:

1. The movie is very blurry
2. Several random white lines pop up in different places
3. The movie shakes a lot (especially in the up and down motion)
4. The image frames are just not clear.

At first glance, it was much more difficult to tell what the exact problems were. It took a few times of replaying the movie to understand how to break down the problems associated with it. For example, the blurriness of the movie could be reduced by improving the sharpness of the movie. In the image below, it can be seen how the movie has random white lines and dots appearing all over. The movie is also very shaky as the image keeps moving up and down.

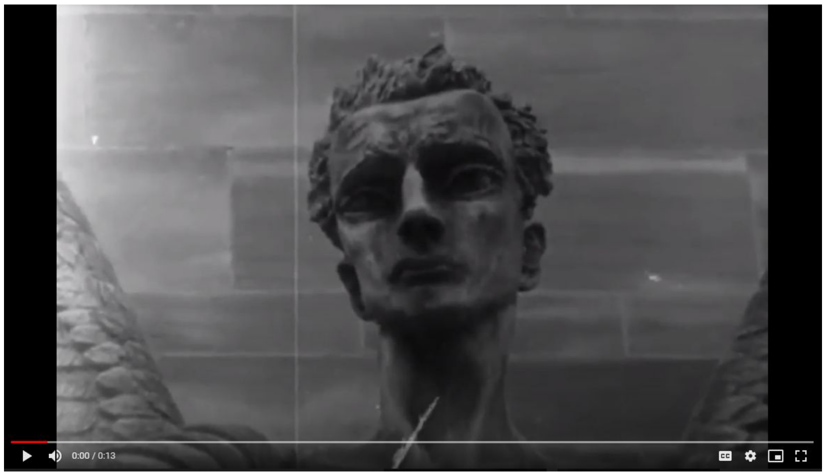


Figure 1. Screenshot of original Movie given to work with/improve upon

## Problems Addressed in this Project

As discussed previously, there were several issues with the provided video, many of which could be tackled using techniques in image enhancement. Among the many issues, our team tackled issues with stabilization, edge enhancement, impulse noise reduction, contrast improvement and scratch removal.

The most observable issue was the shaking of the video and, hence, image stabilization was the most crucial. To do this, the motion had to be analyzed to recognize the displacement in each dimension. Upon doing so, each frame had to be shifted to counteract the appropriate displacement in the dimensions. Furthermore, the frame shifting results in an overlap of pixels and hence, appropriate cropping was necessary to have the output include only the valid pixels in all frames.

To tackle the issues of blurriness, the image required sharpening. The edge enhancement was successfully done through using the unmasking method described in detail in the "Edge Enhancement" section. Furthermore, histogram equalization was applied to improve contrast while the median filter was used to reduce impulse noise. The final issue tackled in the project was the scratch removal. In the provided image, there is a clear disruption in the image as a line passes through the image vertically. Through selecting the appropriate regions, the noise in the column is removed and interpolation of the accurate data is used to replace the corrupted section of data.

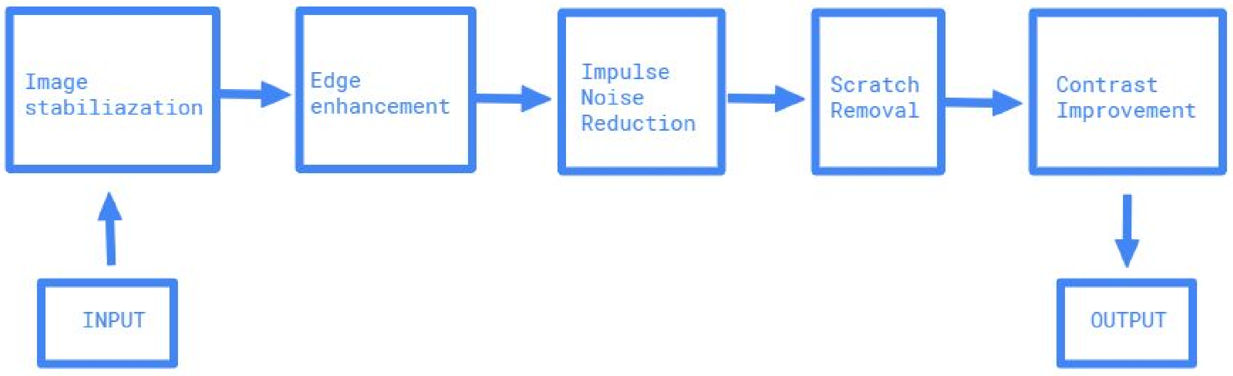


Figure 2: Illustrative process of image processing techniques

# Image Processing Theory

This section briefly describes the theory behind the various Image Processing methods used in this project.

## *Image Stabilization*

The process of stabilizing a shaky video can be broken down into 3 steps: (1) motion estimation; (2) frame shifting; and (3) movie cropping. Each of these steps are discussed below.

### Motion Estimation

The goal of this first step is to figure out what motion is taking place (how part or all of the image frames is moving). Accordingly, a specific region (range of y & x coordinates) is taken as a reference and its position in the overall image is compared between subsequent movie frames. Consider the following movie frames:

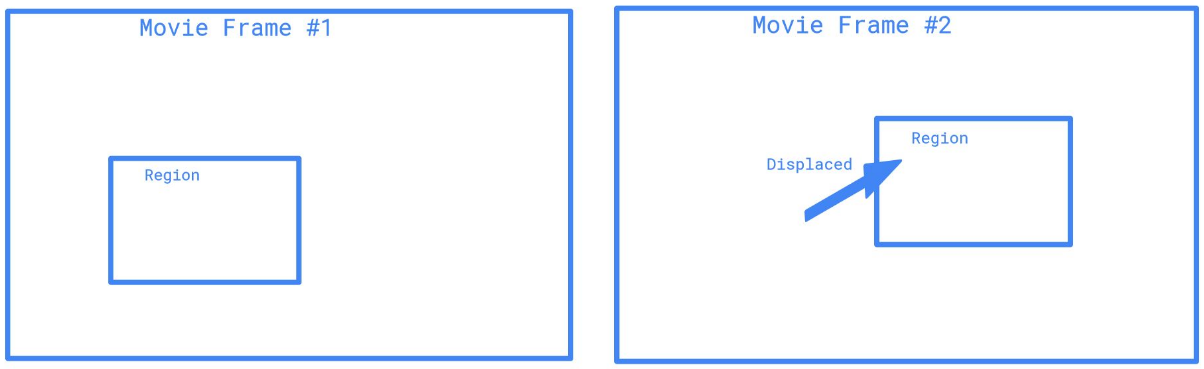


Figure 3: Movie Frame #1 (Left), Movie Frame #2 with Displaced Region (Right).

The region under consideration has displaced to a new position from movie frame #1 to frame #2. Therefore, in order to fix this shaking, the y & x displacements need to be found first.

Accordingly, frame #1 of the movie is taken as a reference and the displacements of all other frames are taken with respect to it. Each movie frame is shifted around, and the difference (error) between the two images is computed using the following formula:

$$error = \sum_y \sum_x \left( f(n,y,x) - f(n+1,y,x) \right)$$

The displacement that gives the lowest error is saved for each frame, and the displacement vector containing y & x displacements for all movie frames w.r.t. the first frame is obtained.

## Frame Shifting

Having obtained the displacement vector, each frame in the movie is now shifted by the appropriate y & x amount. This makes the region under consideration approximately overlap within consecutive movie frames as can be seen below:

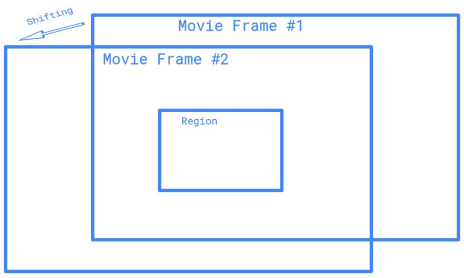


Figure 4: Frame Shifting

## Movie Cropping

There is still an important issue that needs to be addressed. As can be seen in the previous screenshot, moving the image frame would leave a black border where no image pixels exist. To fix this, the movie needs to be cropped so that the movie only contains the region containing valid pixels in all frames (the overlapped area in the previous screenshot).

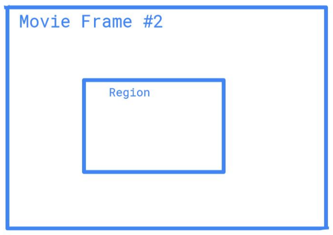


Figure 5: Movie Cropping

## *Edge Enhancement (Sharpening)*

To sharpen the image, "Unsharp Masking" was used. In this method, a high-pass version of the original image is first obtained, and this high-pass version is added to the original image based on a multiplication parameter $\lambda$.

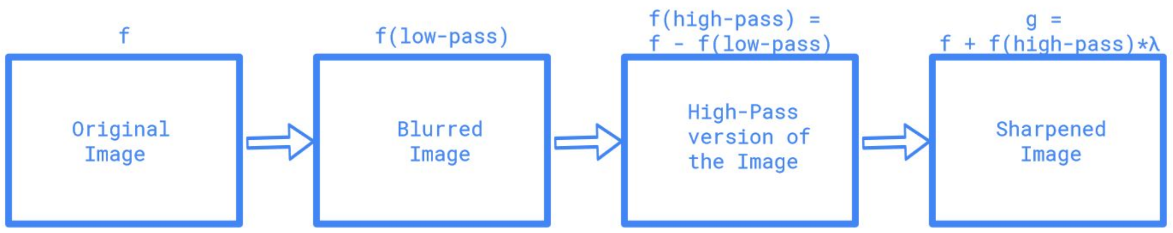


Figure 6: Unsharp Masking Process

In this method, a low-pass (blurred version) of the original image is first obtained by convolving the image with a gaussian point spread function. An appropriate value for the radius parameter 'R' of the gaussian function must be selected. Next, this blurred image is subtracted from the original image to get a high-pass version of the original image (image - high-pass content = low-pass content). Finally, the high-pass version of the image is multiplied with the parameter $\lambda$ and then added to the original image, thus forming a sharpened version of it.

## *Impulse Noise Reduction*

Impulse noises are unwanted short duration noises that degrade images or videos. These can generally be caused during interference or disturbances during acquisition or transmission. Hence, the removal of this noise plays an important role in image processing.

For the removal of impulse noise, a median filter was implemented. In the input video, near pixels have very similar values since they change slowly over space except for edges, corners etc. The implemented median filter moves through the video pixel by pixel and sorts values into numerical order and replaces each value with the median value of neighboring pixels. This allows for the effective removal of the noise while preserving edges, corners etc.

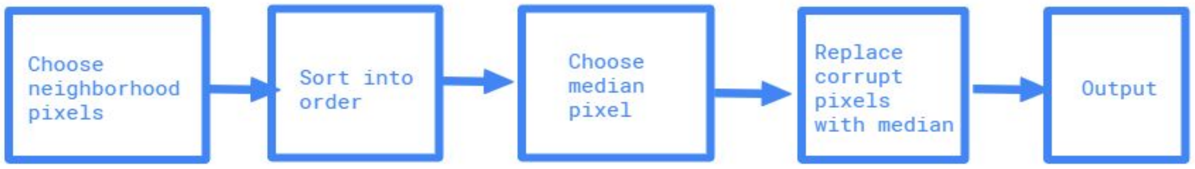

Choose neighborhood pixels
Sort into order
Choose median pixel
Replace corrupt pixels with median
Output

Figure 7: Illustration of the impulse noise reduction process through implementation of the median filter

## *Contrast Improvement (Manipulation)*

Contrast Improvement/Manipulation is a form of image enhancement that is called point processing, which is where each pixel is processed separately or point-by-point. In point processing, there is a function applied v=T(u) to each pixel where u= a pixel value before processing and v= that pixel's value after processing. In Matlab specifically, the equation becomes g = T(f) where g is the whole output image and f is the whole input image.

The term **contrast** refers to the amount of color or grayscale differentiation that exists between various image features in both analog and digital images. One of the most common defects of photographic or electronic images is poor contrast resulting from either poor lighting conditions like an underexposed or an overexposed scene, or a sensor nonlinearity or small dynamic range compared to the dynamic range of the captured scene like a back-lit scene.

Image contrast can often be improved by a look-up table (LUT) operation that rescales the amplitude of each pixel. For example, in an underexposed image with the histogram centered around small code values, a LUT that stretches that range can result in the desired image enhancement.

The average brightness of an image (fmean) can be calculated using histogram equalization. If there exists a brightness variable B, and a contrast variable C, then a linear transformation is created that maps f = (fmean - B) to g = fmean with an inverse slope of (1 - C/100).

In histogram equalization specifically, it attempts to improve contrast by redistributing the histogram in a roughly uniform fashion. It tries to make the histogram flat. It is usually done automatically and results in a nonlinear LUT that is derived from the histogram of the image. It often reduces the total number of output levels. For every input code value f, the output code value g is determined so that the area of the two regions are as close as possible. Like the cumulative distribution in statistics, v = T(u) = the integral from 0 to u of h(u') du'. For continuous brightness u, this will cause the output image's histogram to be h'(v) =1. For pixels with integer values, the equation v = the summation j=o to u of h(j) is used. After this, the intensity normalization for display is used.

## *Scratch Removal*

In the input image provided, a white line is presented in the background of the image. This is a representation of a noise that is generally caused by scratches in the film or during acquisition. When the average of the columns is graphed, this can be sought as an exaggerated peak caused by the noise.

Through selecting the appropriate window of data for the noise, corrupted data within the window can be removed and replaced with interpolated data based on the surrounding data. This patch of interpolated data allows for an accurate estimation of the region to replace the corrupted data point i.e. the scratch.

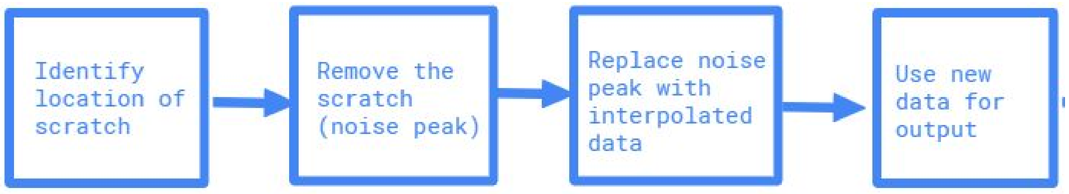

Identify location of scratch
Remove the scratch (noise peak)
Replace noise peak with interpolated data
Use new data for output

Figure 8: Illustration of the scratch removal process

# MATLAB Code

Now that the theory behind the various Image Processing methods have been discussed, their implementation in MATLAB is detailed in this section.

## *Image Stabilization*

The theory section on "Image Stabilization" talks about the process of correcting motion in consecutive image frames in the video. This section discusses the MATLAB implementation of this method. In order to do so, the following steps were followed:

**Step 1: Creating a Reference Movie**

To compute the y & x displacements of each movie frame, the first frame of the movie was used as the "reference" image. Accordingly, in order to speed up the computation, a reference "movie" (3D matrix) was created by taking the first frame of the movie and duplicating it as many times as there are frames in the original movie.

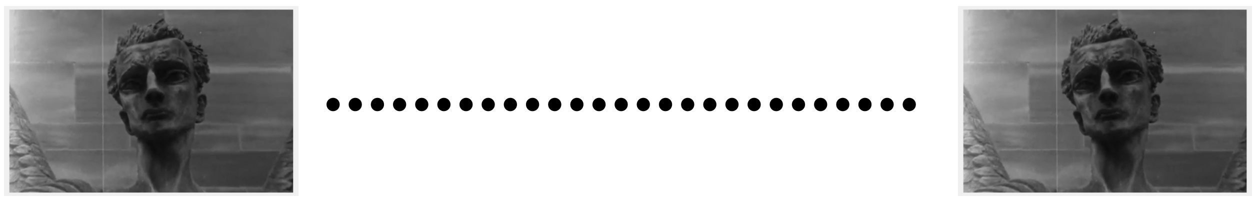


Figure 9: Reference Movie (Formed by Duplicating Frame #1 N Times)

**Step 2: Getting the Region Under Consideration for all Movie Frames**

To perform the Image Stabilization process, a region smaller than the actual size of the image frames in the movie must be selected. Accordingly, the entire movie was cropped to the following region: *ymin = 35, ymax = 940, xmin = 35, xmax = 1465*. To perform this computation faster, a 3D crop function *"crop3d()"* was created that crops the entire movie (3D matrix) in one go.

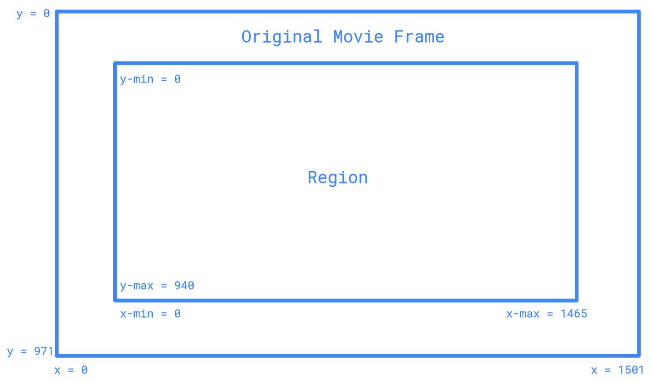


Figure 10: Original Movie Frame & Region Dimensions

## Step 3: Computing the Displacement Vector

In a loop, a certain region of the reference movie (of the same size as that of the region) was cropped and then subtracted from the cropped original movie to compute the error. The y & x displacements having the minimum error for each movie frame were stored in a displacement vector. The range of displacements taken under consideration were *y = -27:6 and x = -15:1*. These values were selected as the maximum and minimum y & x displacements obtained by running the stabilization function for all possible values were *y = -26 & 5* and *x = -14 & 0*.

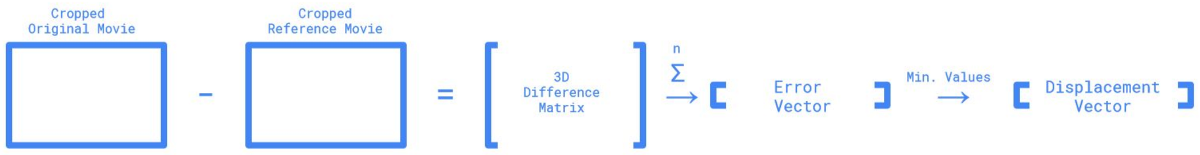


Figure 11: Computing the Displacement Vector

## Step 4: Shifting Movie Frames

Having obtained the displacement vector, each of the image frames of the original movie were shifted by the appropriate amount as given in the displacement vector. To do this, a separate function *shiftVideo()* was implemented. This function takes in the original movie and the displacement vector and shifts each frame using the *shiftImg()* function created. This function zero pads the portion of the frame outside the original image, leaving a black border in some places (see the bottom edge of the screenshot below).

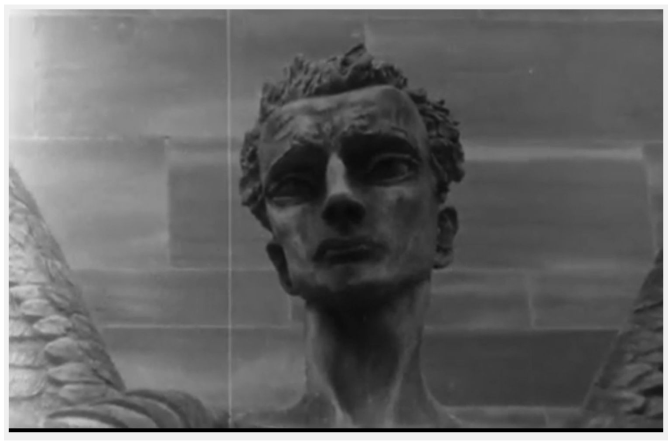


Figure 12: Image Frame with Black Border

**Step 5: Cropping the Final Movie**

The final movie was cropped using *ymin = 35, ymax = 940, xmin = 35, xmax = 1465*. This effectively removed the black borders formed in the previous step. Compare the following screenshot with the previous one.

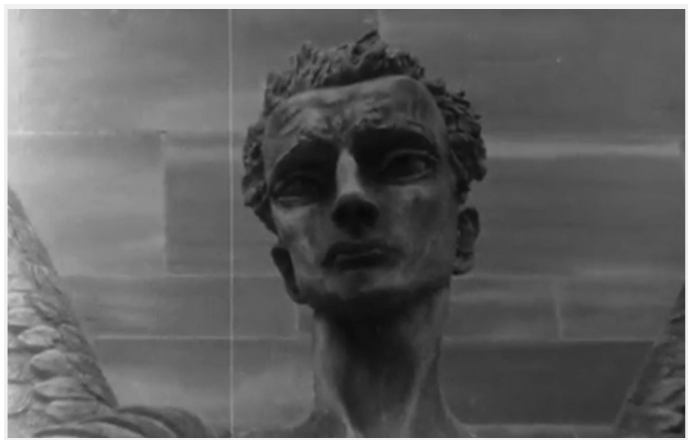


Figure 13: Image Frame with Black Border Removed

On running the stabilization operation on the entire movie, the average execution time achieved was approximately **35 minutes.**

## *Edge Enhancement (Sharpening)*

The steps listed in the theory section of this report were followed to implement the sharpening function in MATLAB. This section discusses the code & output at each stage of execution.

### Step 1: Defining the Gaussian Point Spread Function

To create a gaussian psf, the *"fspecial('gaussian',hsize,sigma)"* MATLAB function was used. The radius parameter 'R' inputted to the sharpening function was used to define the gaussian psf. Additionally, the gaussian matrix was zero padded to be the same size as the image frames in the video.

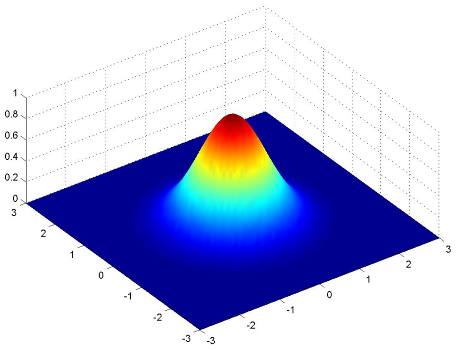


Figure 14: Gaussian Function

### Step 2: Obtaining a Low-Pass (Blurred) Image

Having the gaussian psf ready, it is now convolved with the original image to obtain the blurred version of the image. This was done by first computing the fourier transform of the gaussian psf & the original image using the *"fft2()"* MATLAB function, multiplying all elements in the two matrices, and then taking the inverse fourier transform of it (using the *"ifft2()"* MATLAB function). A screenshot of the blurred version of an image frame from the video (using R = 5) is given below:

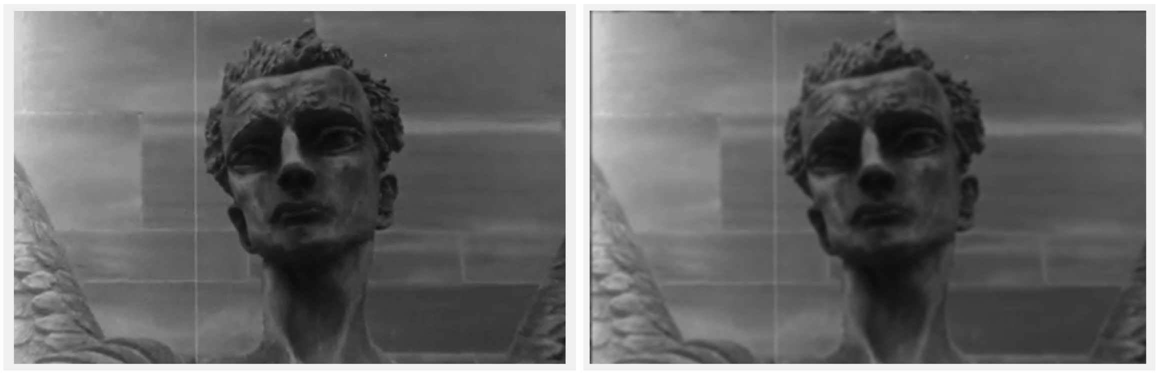


Figure 15: Original Image (Left), Low-Pass (Blurred) Version of the Image (Right).

## Step 3: Obtaining the High-Pass Version of the Image

Next, the high-pass version of the image is obtained by subtracting the low-pass image from the original image. A screenshot of the high-pass version of the image frame used in the previous example is given below:

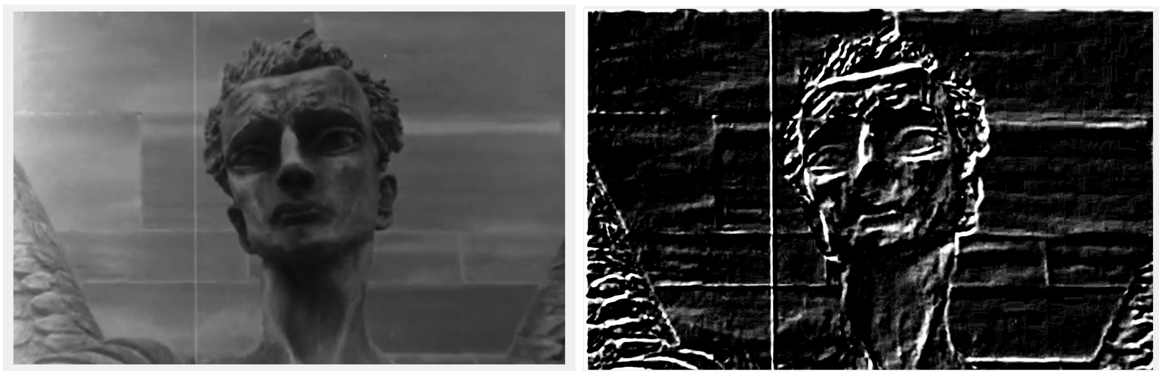


Figure 16: Original Image (Left), High-Pass Version of the Image (Right).

## Step 4: Sharpening the Image

Finally, the high-pass version of the image is multiplied by the parameter $\lambda$ (using $\lambda = 1$) and then added to the original image. A screenshot of the final, sharpened image is given below:

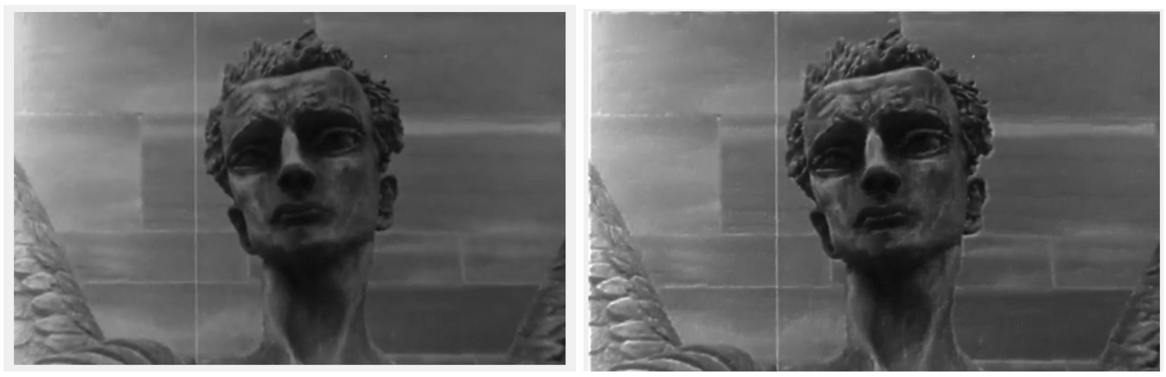


Figure 17: Original Image (Left), Sharpened Image (Right).

On running the sharpening operation on the entire movie, the average execution time achieved was approximately **75 seconds.**

## *Impulse Noise Reduction*

For the removal of the impulse noise, a median filter was implemented. Due to the 3D aspect of the filter, three for loops were required. As seen in the code attached, the three for loops allow for the sorting of values in each dimension pixel by pixel. The median is then calculated using MATLAB's in-built median function. Through replacing values by the median of its neighboring values in each dimension, the impulse noises are removed. Figure (18) demonstrates the sample result.

While the median filter does not remove large patches of noise, it successfully removes the small patches. This is because the median filter's process accounts for the preservation of edges and corners and large patches of noise are recognized as such because of the significant changes in value.

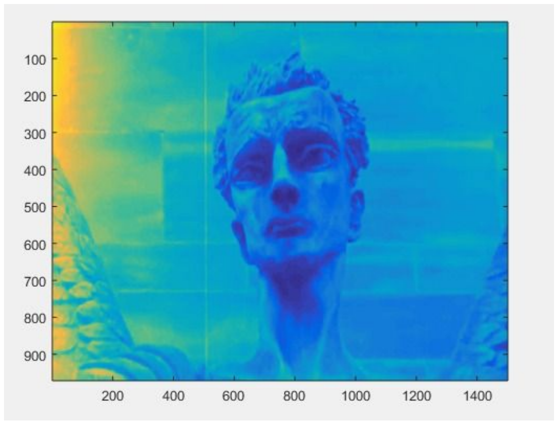


Figure 18: After median filter application

## *Contrast Improvement*

For this part of the code, in order to improve or manipulate the contrast of the image, histogram equalization was used. Rather than using the function that is already in Matlab, a version of it was created by writing out the function. This was done by first loading the data into the new function created called histogrameq. The loaded data is in the form of a x b x c and the data needs to be in an a x b format (2D image format). This was done by using the squeeze function. From here, the rows and columns were established as well as creating arrays for the frequency, pdf, cdf, cumulative, and outpic. Then, a for loop was created with increasing the frequency and the pixels. This loop was tracing the rows and columns of the image. The intensity level wanted is 256-1 so that's why the intensity level is initialized to 255. A second for loop was created for the cumulative function as well as a third for loop for the final result. The second loop is for the pdf and cdf of this histogram, while the third loop traces the final rows and columns of the image after the pdf and cdf is used. The final part of the code was moving the a x b format back into the a x b x c format so that it could be displayed as a movie rather than a single image. The code was tested using a single image (the first image) to see how the result looks. In reality, the squeeze function would be for the whole f.mat file and the concat at the end would uncompress all the images back into the full movie. The code takes approximately a minute to run.

Problems associated with this part of the code:

This code is performing histogram equalization, but it isn't really improving the quality of the image. The image is trying to equalize the dark and white parts of the image, but it's making the dark parts darker and the light parts lighter when looking at the final image that is created. If there was more time, this code would be adapted to equalize the image better, so that the contrast actually improved.

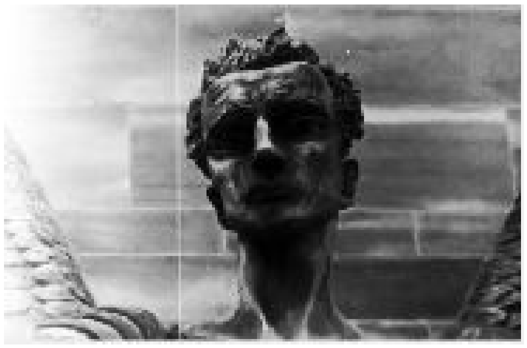


Figure 19. How the histogram equalization function makes the image look after applied.

## *Scratch removal*

The steps listed in the theory section of this report were followed to implement the scratch removal in MATLAB. This section discusses the code & output at each stage of execution as well as obstacles encountered in applying the code to multiple frames (i.e. the video).

### Step 1: Identifying region of interest

To identify the region that we were interested in, we analyzed the average of the column values to observe the exaggerated peak caused by the noise and identified the values of interest. Through analyzing the output and observing the peak's changes through different time frames, we selected the frame of the video that had the maximum peak (n= frame= 20, selected). Similarly, analyzing Figure (20. a) through MATLAB's cursor tool allowed us to narrow the affected columns and hence, allowed us to select the regions for removal and replacement.

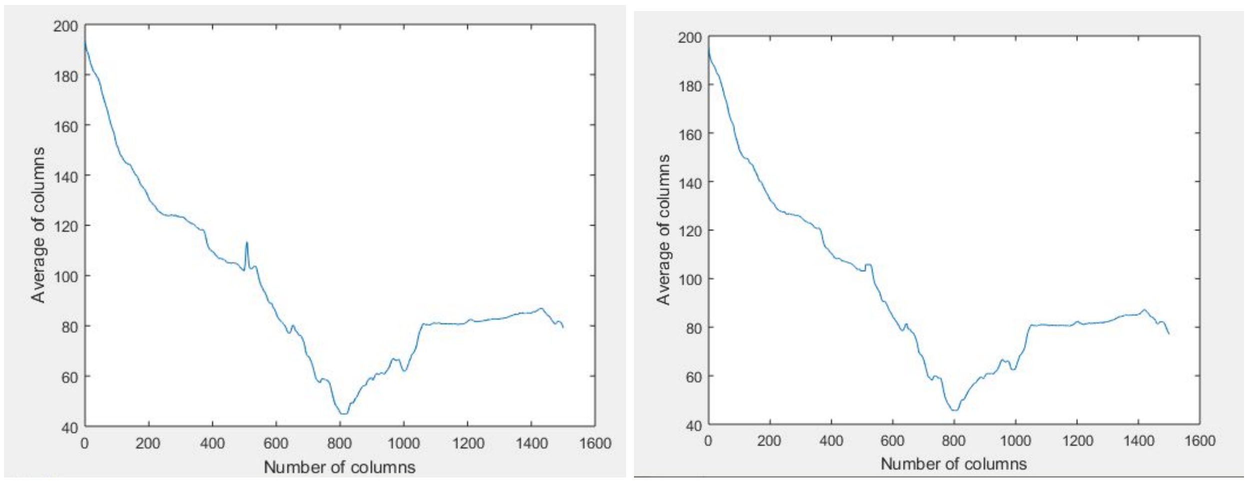


Figure 20: (a) Graph to identify location of scratch
(b) Data for the region of scratch after replacement with interpolated data

**Step 2: Remove and replace with patch**

Upon recognizing the set of corrupted values, the values in these regions are removed. Through interpolating the data before and after the removed section, the output includes an estimated set of accurate data to replace the scratch.

**Problems:**

The peak was analyzed for over multiple time frames and the frame with the maximum observable peak was selected for scratch removal. However, while applying the interpolation through different time frames, many obstacles affected the successful implementation. Since different time frames of the video have varying levels of noise, the code could not be applied over multiple frames due to the need for varying region selection.

## The Final Movie & Future Improvements

After all the code has been created, the final movie can be seen with great improvements. The final movie overall is much more pleasing to the human eye. It no longer has problems of blurriness, random white lines/dots popping up, and the shakiness of the movie moving up and down has been fixed. The improvements can be seen in the image below.

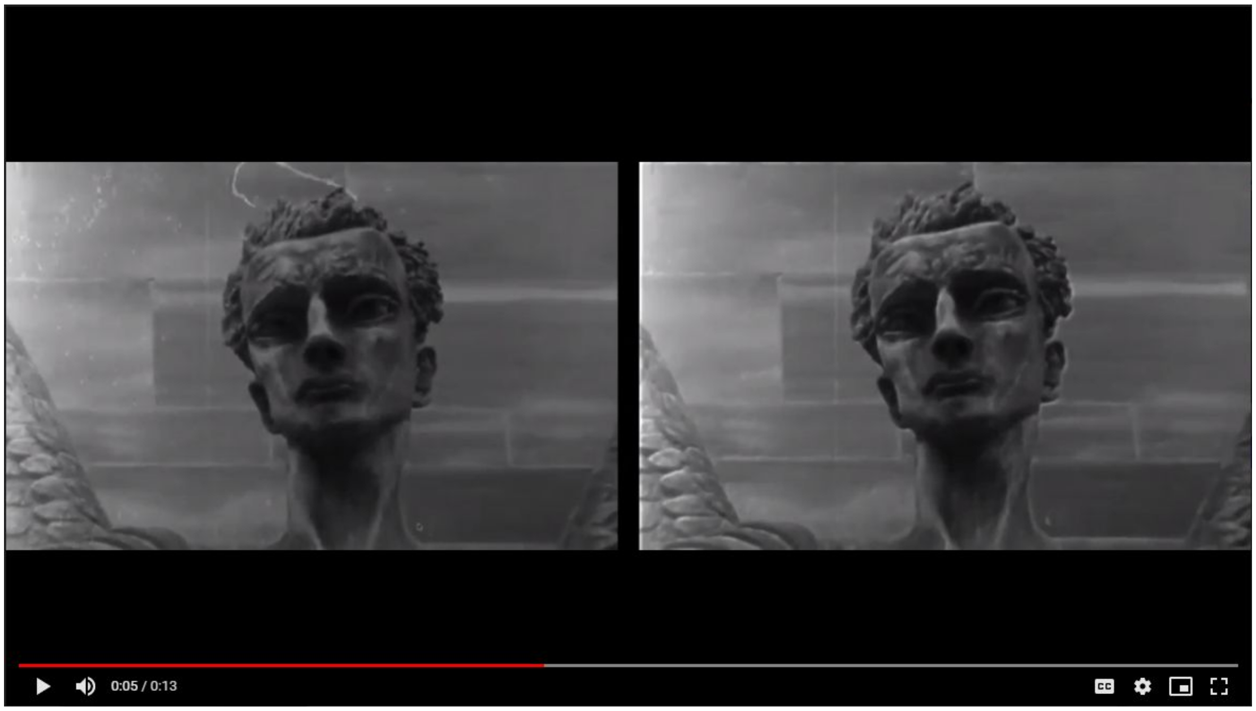


Figure 21.This is a comparison side by side of the original movie (left) vs the new improved movie (right)

If there was more time, even more improvements could have been made to the video. These future improvements include optimizing the group’s code so that it runs quicker and smoother. Currently, it takes about an hour to run everything together. The code tends to take a little too long to run at the moment. Other improvements include improving scratch removal and contrast manipulation. These were attempted, but not all the way perfected. These could be improved to help fix the movie in greater detail. Overall, the project was attempted using all of the resources available to the group to the best of the group’s ability. If there was more time, further improvements could have been made, but overall this is the attempt that was made by the group.

# Appendix

## *Individual Contributions:*

Clive Gomes →

- ★ Code:
    - ○ Stabilization (*stabilize.m, shiftImg.m, shiftVideo.m*)
    - ○ Edge Enhancement (*sharpen.m*)
    - ○ Helper Functions (*crop3d.m, exportAvi.m, playmat.m*)
    - ○ Overall Movie Restoration Function (*filmRestore.m*)
- ★ Documentation:
    - ○ Image Stabilization (Image Processing Theory & MATLAB Code)
    - ○ Edge Enhancement (Image Processing Theory & MATLAB Code)
    - ○ Appendix

Dristi Chaudhuri →

- ★ Code:
    - ○ Contrast Improvement (*histogrameq.m*)
- ★ Documentation:
    - ○ Purpose of the Project
    - ○ A First Look at the Movie
    - ○ Contrast Improvement (Image Processing Theory & MATLAB Code)
    - ○ The Final Movie & Future Improvements

Nanjeeba Chowdhury →

- ★ Code:
    - ○ Median Filter (*removeNoise.m*)
    - ○ Scratch Removal (*removeScratch.m*)
- ★ Documentation:
    - ○ Impulse Noise Reduction (Image Processing Theory & MATLAB Code)
    - ○ Scratch Removal (Image Processing Theory & MATLAB Code)
    - ○ Problems Addressed in this Project

## *MATLAB Code:*

### crop3d.m

```
% Crop all frames in the video up to given frame boundaries
function g = crop3d(f,frameBounds)
        g = f; % Make a copy of the input video
        dim = size(f); % Get dimensions of the video

        % Crop the image with given frame boundaries
        if (frameBounds(1,2) < dim(2));g(:,frameBounds(1,2)+1:end,:) = [];end % crop bottom
        if (frameBounds(1,1) > 1);g(:,1:frameBounds(1,1)-1,:) = []; end % crop top
        if (frameBounds(2,2) < dim(3));g(:,:,frameBounds(2,2)+1:end) = [];end % crop right
        if (frameBounds(2,1) > 1);g(:,:,1:frameBounds(2,1)-1) = [];end % crop left
end
```

### exportAvi.m

```
% Export the video f as 'fname.avi'
function exportAvi(fname,f)
        % Create and open a video
        v = VideoWriter(fname);
        open(v);

        % Get dimensions of the video
        fsize = size(f);

        % Write frames to the video
        for n = 1:fsize(1)
        writeVideo(v,reshape(f(n,:,:),fsize(2),fsize(3)));
        end

        % Close video
        close(v);
end
```

### filmRestore.m

```
% ECE 481 Project - Image Stabilization
% Team Members: Clive Gomes, Dristi Chaudhuri, Nanjeeba Chowdhury
% Code Estimated Runtime: ~50mins

% Load original movie
load f;

% Define frame boundaries
ymin = 35;
ymax = 940;
xmin = 35;
```

```
xmax = 1465;
fbounds = [[ymin ymax];[xmin xmax]];

% Define range of displacements to test
flim = [[-27 6];[-15 1]];


% Stabilize Movie (~35mins)
g = stabilize(f,fbounds,flim);


% Remove Impulse Noise (~15mins)
g = removeNoise(g_stabilized(1:39,:,:),5);


% Sharpen Movie (~1mins)
g = sharpen(g,1,5);


% Create Final Movie (~30sec)

% Crop Original Movie to Same Size as Final Movie
f = crop3d(f,fbounds);

% Create Spacing Matrix
fsize = size(f);
gap = zeros(fsize(1),fsize(2),50);

% Concatenate Videos
g = cat(3,f,gap,g_stable_noise_sharp);

% Export AVI
exportAvi('finalMovie.avi',g);
```

## histogrameq.m

```
function histogrameq()
load f;

original = squeeze(f(1,:,:)); %%squeezing the matrix to be 2D
[rows,columns,~] = size(original); %converting to rows and columns array
finalResult = uint8(zeros(rows,columns));
pixelNumber = rows*columns; %creating the pixel
frequency = zeros(256,1); %creating an array of values to be used later
pdf = zeros(256,1); %creating an array of values to be used later
cdf = zeros(256,1); %creating an array of values to be used later
cumulative = zeros(256,1); %creating an array of values to be used later
outpic = zeros(256,1); %creating an array of values to be used later

%loop tracing rows and columns
for i = 1:1:rows
        for j = 1:1:columns
        val = original(i,j);
        frequency(val+1) = frequency(val+1)+1; %increasing the array
        pdf(val+1) = frequency(val+1)/pixelNumber;
        end
end

sum = 0 ;
%we want the 256 - 1 that's why we initailzed the intensityLevel
%with 255 instead of 256
intensityLevel = 255;

%loop for pdf and cdf of histogram
for i = 1:1:size(pdf)
        sum = sum + frequency(i);
        cumulative(i) = sum;
        cdf(i) = cumulative(i)/ pixelNumber;
        outpic(i) = round(cdf(i) * intensityLevel);
end

%loop tracing rows and columns
for i = 1:1:rows
        for j = 1:1:columns
        finalResult(i,j) = outpic(original(i,j) + 1); %creating the final result array
        end
end
%%concat(3,f1,f2...); would be used to convert the 2D array back into 3D to
%%play the movie
subplot(1,2,1),imshow(finalResult(:,:,1));
```

## playmat.m

```
% Play a video in MATLAB with the given frame rate.
function playmat(f,rate)
        fsize = size(f); % Get dimensions of the video
        for n = 1:fsize(1)
        imagesc(squeeze(f(n,:,:))); % Get nth image frame
        pause(1/rate); % Pause for amount given by frame rate
        end
end
```

## removeNoise.m

```
% Remove Impulse Noise from Video using Median Filter
function g = removeNoise(f,R)
        % Get dimensions of the video
        [l,m,n] = size(f);

        % Create blank output image
        g = zeros(l,m,n);
        g = uint8(g);

        % Perform the noise removal operation
        for i = 1:l
        for j = 1:m
        for k=1:n
        % Define max and minimum values x and y coordinates
        % Minimum x coordinate has to be greater than or equal to 1
        xmin = max(1,i-R);
        xmax = min(l,i+R);
        ymin = max(1,j-R);
        ymax = min(m,j+R);
        zmin= max(1,k-1);
        zmax = min(n,k+1);
        medianfilt = f(xmin:xmax, ymin:ymax, zmin:zmax);
        %Now the new intensity of pixel at (i,j) will be median
        % of this matrix
        g(i,j,k) = median(medianfilt(:));
        end
        end
        end

end
```

### removeScratch.m

```
%%
close all; clear all; clc;
load f;
p= squeeze(mean(f,2)); %get average of columns for all frames
%Plot the p graph for all frames as a movie -- change the speed by changing the value in pause()
dim = size(p);
x = [1:dim(2)];
fig = figure(1);
%for n=1:dim(2); %for loop removed after viewing video and recognizing
%frame with highest peak
n=20; %Selected one frame after viewing video
clf(fig);
y= squeeze(p(n,:));
plot(x,y); xlabel('Number of columns'); ylabel('Average of columns');
pause(0.01);
%end

xgood= [1:495, 527:1501];%selected the normal regions based on figure 1 values
xpatch= [496:526]; %area needing patch
pgood = squeeze(p(20,xgood)); %frame= 20
ppatch= interp1(xgood, pgood, xpatch, 'spline'); %interpolation using spline
p_final= [pgood(1:495), ppatch(1:31), pgood(496:1470)]; %replace corrupted area with patch
figure(2); plot(x,p_final); xlabel('Number of columns'); ylabel('Average of columns');%plot new
figure with patch
```

### sharpen.m

```
% Sharpen all Image Frames in a Video using Unsharp Masking
function g = sharpen(f,lambda,R)
```

```
        g = f; % Make a copy of the input video
        fsize = size(f); % Get dimensions of the video

        % Get Gaussian PSF
        h = fspecial('gaussian',2*R+1,R);

        % Get Sizes of h
        hsize = size(h);

        % Zero Pad h
        h = cat(2,h,zeros(hsize(1),fsize(3)-hsize(2)));
        h = cat(1,h,zeros(fsize(2)-hsize(1),fsize(3)));

        % Sharpen All Image Frames in Video
        for n=1:fsize(1)
        % Get nth Image Frame
        fn = squeeze(f(n,:,:));

        % Perform Convolution to Get Blurred Image
        H = fft2(h);
        F = fft2(squeeze(fn));
        fn_blurred = ifft2(H.*F);

        % Performing Unsharp Masking
        g(n,:,:) = reshape(fn +((fn - uint8(fn_blurred)).*lambda),[1, fsize(2), fsize(3)]);
        end

end
```

**shiftImg.m**

```
% Shift a single image by the amount given in d. Zero pad the image.
function g = shiftImg(f,d)
        g = f; % Make a copy of the input image
        fsize = size(f); % Get dimensions of the image

        % Shift Up
        if d(1) < 0
        for i = d(1):-1
        g(1,:) = []; % Delete first row
        g = [g;zeros(1,fsize(2))]; % Zero pad bottom
        end
        % Shift Down
        elseif d(1) > 0
        for i = d(1):-1:1
        g(end,:) = []; % Delete last row
        g = [zeros(1,fsize(2));g]; % Zero pad top
        end
        end
        % Shift Left
        if d(2) < 0
        for i = d(2):-1
```

```
            g(:,1) = []; % Delete first column
            g = cat(2,g,zeros(fsize(1),1)); % Zero pad right
            end
            % Shift Right
            elseif d(2) > 0
            for i = d(2):-1:1
            g(:,end) = []; % Delete last column
            g = cat(2,zeros(fsize(1),1),g); % Zero pad left
            end
            end

end
```

**shiftVideo.m**

```
% Shift all image frames in a video by the amount given in the displacement
% vector d
function g = shiftVideo(f,d)
            g = f; % Make a copy of the input video
            fsize = size(f); % Get dimensions of the video

            % Shift each frame n by given amount d(n)
            for n=1:fsize(1)
            g(n,:,:) = shiftImg(squeeze(g(n,:,:)),d(n,:));
            end
end
```

**stabilize.m**

```
% Stabilize a video by finding the displacement vector, shifting individual
% frames by the appropriate amount, and cropping the final video to remove
% the black borders.
function g = stabilize(f,fbounds,flim)
            g = f; % Make a copy of the input video
            fsize = size(f); % Get dimensions of the video

            % Create a displacement vector (d) containing [error, y-displacement,
            % x-displacement] for each image frame. -1 is the default value.
            d = [-1 -1 -1];
            d1 = d;
            % Also create a 3d image matrix (f1) by duplicating the first (reference)
            % frame equal to the number of frames in the video.
            f1 = squeeze(f(1,:,:));
            f2 = f1;
            for i=1:fsize(1)-1
            f1 = cat(3,f1,f2);
            d = cat(3,d,d1);
            end
            d = squeeze(permute(d,[3 1 2]));
            f1 = permute(f1,[3 1 2]);

            % Crop all image frames in the original video to get the region defined
```

```
        % by fbounds
        f2 = crop3d(f,fbounds);

        % Loop through the y & x displacements given by flim
        for y=flim(1,1):flim(1,2)
        for x=flim(2,1):flim(2,2)
        % Compute the error for all frames in the video
        error = sum(abs(crop3d(f1,(fbounds+[[y y];[x x]])) - f2),[2 3]);
        % Save the y & x displacements for the min error for each frame
        for n=1:fsize(1)
        if (error(n) < d(n,1)) | | (d(n,1) < 0)
                d(n,1) = error(n);
                d(n,2) = y;
                d(n,3) = x;
        end
        end
        end
        end

        % Remove the errors from the displacement vector leaving only the
        % [y-displacement, x-displacement] part
        d(:,1) = [];

        % Crop the final video to remove the black borders
        g = crop3d(shiftVideo(f,d),fbounds);

end
```