AMATH 482 Homework 3: A Spring-Mass System

Matthew Mangione

February 24, 2021

Abstract

In this paper, we analyze the movement of a spring through time based upon three distinct videos of the same event. Rather than using the well-understood mathematical relationships of this system, we opt to solve this computationally using Principal Component Analysis. When we combine all three perspectives of the spring, this will reduce the dimensionality and unique angles of the videos, leaving us with the spring's movement along its strict axes of motion.

1 Introduction and Overview

In this assignment, we are given four sets of three videos of a paint can attached to a spring. Each set of videos provides a unique angle of the event, which are all roughly synchronous with the other videos in a set. This gives us a six dimensional view of a largely one or two dimensional system, which describes the motion of the object with a lot of redundancy. To eliminate some of the unnecessary dimensions and approximate the paint can's motion, we will utilize Principal Component Analysis (PCA) to estimate the true rank of the system. However, before this can be accomplished, the videos of the cans must be analyzed to track the movement of the object in each frame. In order to do so, a light is attached to the paint can to distinguish it from the rest of the image.

Each of the four sets we are given provide a distinct challenge to the analysis of the paint can's motion, and an greater insight into the fundamentals of PCA. The first set is the most basic example of the spring, moving largely along a single dimension with clean videos of the event. The second set is the same motion as the first, except with much shakier and noisier videos, making it more difficult to extract the spring's true motion. The third and fourth video sets are separate examples of the paint can bouncing vertically and horizontally, but in the fourth set, rotation is added to the paint can, at times obscuring the light attached to the paint-can used for tracking the object.

2 Theoretical Background

Principal Component Analysis is largely built upon the concepts of Singular Value Decomposition (SVD), which transform the vectors of a matrix into distinct orthogonal components based upon its singular values, denoted by σ. More formally, given any matrix A $\in \mathbb{C}^{m\times n}$, this transformation finds the diagonalization of its singular values Σ of A in terms of full rank, unitary matrices U $\in \mathbb{C}^{m\times m}$ and V $\in \mathbb{C}^{n\times n}$.

$$A = U\Sigma V^* \tag{1}$$

An important application of these singular values is their use in determining and approximating the rank of a matrix. Because matrices U and V are full-rank, the rank of A must be determined by the rank of Σ. Because Σ is a diagonal matrix, and zeroes in its diagonal correspond to a reduce in rank. Similarly, the rank of a matrix is determined by the number of non-zero singular values. However, in the analysis of real life data, noise will likely be included in its collection, and in practice, any near-zero values will likely still signify the same meaning. Using this notion, we can easily create low rank approximations of A by omitting vectors of U and V corresponding to near-zero singular values.

Central to the ideas of Principle Component Analysis (PCA) is the covariance matrix C_X, which stores the variance between each combination of vectors of matrix X. With the assumption that the mean of each vector is 0, the variance, σ^2, of two vectors is found as below.

$$\sigma^2 = \frac{1}{n-1} x x^T \tag{2}$$

Therefore, with this assumption, the covariance matrix can be defined as below, where diagonal entries relate to the importance of the respective measurements of a vector.

$$C_X = \frac{1}{n-1} X X^T \tag{3}$$

In PCA, we hope to reduce the redundancy of inputted measurements and isolate the important structures in the data. To do this, we will convert our information to the basis of U associated with the SVD.

$$Y = U^* X \tag{4}$$

From which, its covariance is easily calculated.

$$C_Y = \frac{1}{n-1} Y Y^T = \frac{1}{n-1} (U^* X)(U^* X)^T = \frac{1}{n-1} \Sigma^2 \tag{5}$$

3 Algorithm Implementation and Development

Before we can apply PCA on each set of videos, we must first extract the motion of the paint can from every camera perspective. With the assistance of the light atop the paint-can, we will combine a bright pass filter with low level motion detection to isolate the paint can in each frame. In each test, the position of the paint can through time will be captured in each video and then combined to apply PCA. The following procedure is repeated to each set of test videos.

3.1 Preprocessing

Before processing, each video must first be converted into a three dimensional array composed of pixel values for every frame in the video. Using MATLAB functions `im2gray` and `im2double`, each image is converted into grayscale and then the pixel values are converted into fractional values in the range $[0, 1]$. With a correctly formatted set of videos, the number of frames is aligned by comparing the videos manually and inputting the number of frames to delay each video to synchronize them. The total number of frames is then compared and cropped to create videos of equal length. Additionally, due to complications in tracking the object, videos are sampled identically by taking a single frame per every 3-5 frames. Videos may also be cropped to avoid outstanding noise created by the tracking algorithm explained below.

3.2 Tracking

In order to capture the brightness created by the attached flashlight, a highpass filter is applied to each frame of each video. Pixel values below a chosen partition are set to zero, excluding any features of the video which are not bright. Next, with the remaining bright components of the filtered videos, the motion of each pixel is estimated by finding the absolute value of the difference between each frame and the last. This process effectively removes most stationary features in the video, but produces some noise due to minor shifts in the frames. To compensate for this, each image is slightly blurred by applying a Shannon filter to the image in frequency space. Now, with only bright, moving objects remaining in each image, a second high pass filter is applied to remove any noisy data reduced by the Shannon filter. The remaining data in each frame is understood to be the location of the paint can, and an average of the locations of the remaining white pixels is taken to be the location of the can.

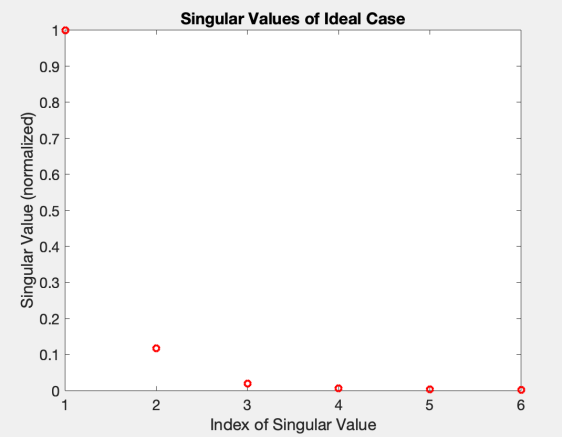

Figure 1: The normalized variance of the principle components expressed in the ideal case.

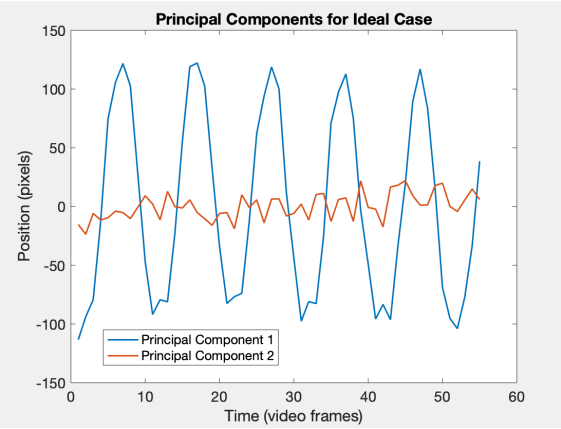

Figure 2: The position of the first two principle components over time for the ideal case.

3.3 Principle Component Analysis

Having now obtained the x,y position of the object throughout each video, a matrix can be created by appending the x and y position vectors from each camera's perspective. From the 2 dimensions from the 3 videos, 6 measurements are described for every remaining video frame. The average of each vector is determined using `mean` and is then subtracted from its vector, ensuring that the average of each measurement is zero in alignment with our earlier assumption. Next, the SVD is computed using the very convenient `svd` MATLAB command. With the U, Σ, and V matrices discovered, our measurements can be projected along the prominent axes present in the data but multiplying our matrix by U^*. The vectors of this matrix are the principal components of our data, and the singular values of Σ relate to the variance of each measurement.

4 Computational Results

4.1 Ideal Case

The first set of videos demonstrate the paint can bouncing vertically with minimal shaking of the cameras. Because of this, the positions of the object were extracted successfully with the tracking algorithm, besides for the inclusion of several duplicate frames which create extraneous data. For this reason, video frames were sampled as described earlier, which lessened this effect. The few remaining 'NaN' values were replaced by interpolation of the surrounding data. With clean data, the singular values and principle components nicely reflected the nature of the motion. The variance of the first principle component, the square of the first singular value, is significantly larger than all others, suggesting the dominance of motion in a single direction, which we know to be true. Similarly, the first principal component displays simple harmonic motion, and the second principle component largely resemble noise of the system, affirming our understanding from the video.

4.2 Noisy Case

The same motion as the ideal case was similarly captured in this set of videos. However, they include much more noise created by the shaking of the cameras. This was a true test for the tracking algorithm implemented, and required significant tuning of its parameters to obtain a decent approximation of the objects motion. This set of videos also required manually determined cropping to isolate the location of the paint can and remove prominent noisy sections. Even then, the SVD of the processed data produced several significant singular values, with the second value almost reaching a fourth of the first variance. Because these videos correspond to the same motion featured in the ideal case, we know that subsequent dimensionality of the principle components are attributed to noise in the system. The significant principle components reflect

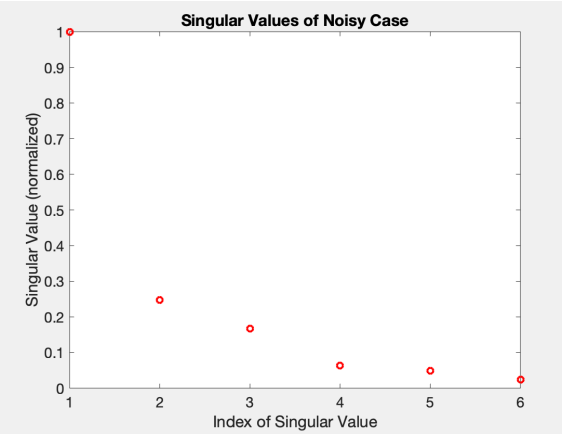

Figure 3: The normalized variance of the principle components expressed in the noisy case.

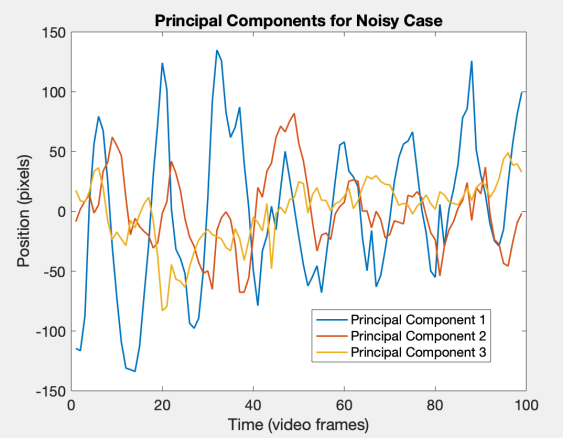

Figure 4: The position of the first three principle components over time for the noisy case.

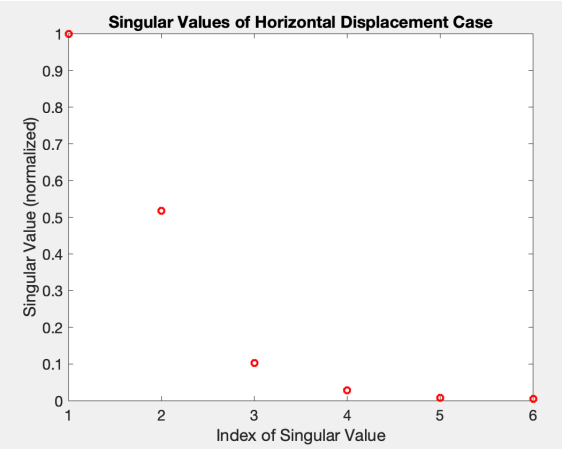

Figure 5: The normalized variance of the principle components expressed in the horizontal displacement case.

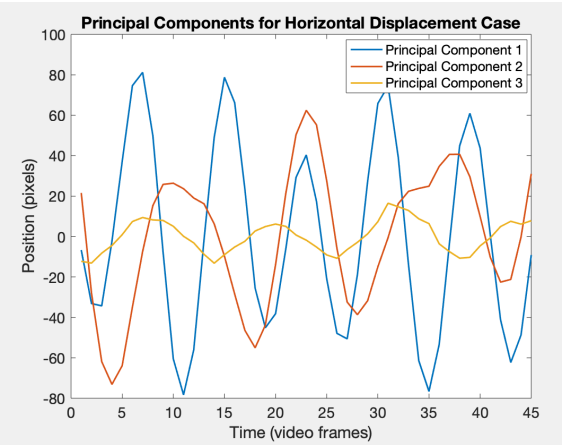

Figure 6: The position of the first three principle components over time for the horizontal displacement case.

this: the first vector displaying moderately clean sinusoidal motion, while the following principle components display the loud noise present in the system.

4.3 Horizontal Displacement Case

This set of videos introduces a second dimension of motion by moving the object vertically and horizontally. Given the previous tools for processing and tracking the videos, not much had to be developed in order to capture the position of the object in this set. The SVD of these measurements create two primary singular values, with normalized variances of 1.0 and 0.5. This is assumed to correspond to the motion in the vertical and horizontal directions respectively. Like the previous cases, there is a strong vertical component which relates to the object's vertical motion. Similarly, second principle component likely relates to the horizontal motion added to this test. While the remaining variances of the principle components are 0.1 or below, the third principle component still expresses sinusoidal motion instead of the typical noise we have seen in the other cases. This is potentially part of the motion of the two prominent axes which was somehow separated. Overall, the first two principal components seem to describe the motion of both dimensions accurately, with the motion of the principle components mirroring the movement of the paint can.

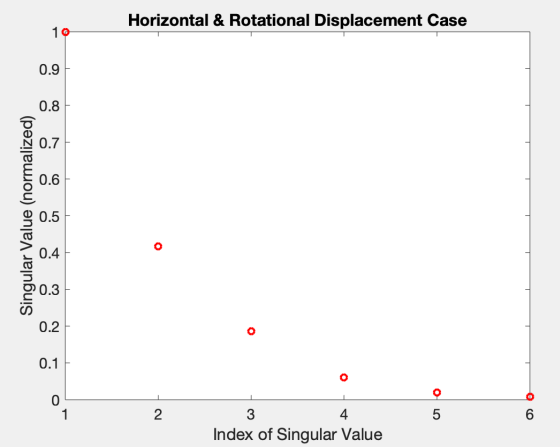

Figure 7: The normalized variance of the principle components expressed in the horizontal displacement and rotation case.

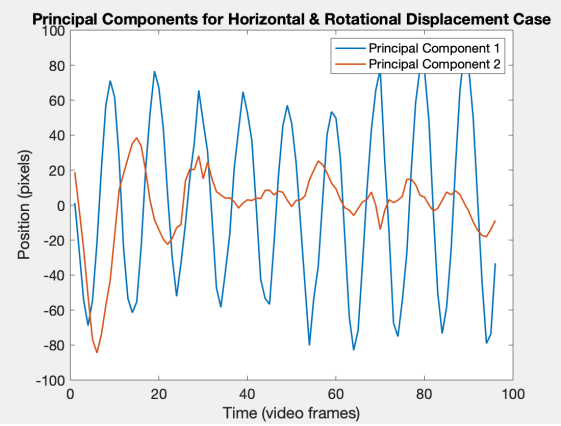

Figure 8: The position of the first three principle components over time for the horizontal displacement and rotation case.

4.4 Horizontal Displacement and Rotation Case

this set of videos expresses the same vertical and horizontal motion described in the last section, but also adds rotation to the paint can, and more importantly, the flashlight. While the vertical bouncing is similar to the previous cases, the horizontal motion is distinct from that of the last case, with the motion decaying much quicker horizontally. Despite the lack of the flashlight present, the tracking method still managed to pick up on the object with a little tuning. This is likely because of the white sections of the paint which were detected as moving. When the SVD was computed, two to three strong singular values were found. Similar to the previous example, the first two likely relate to the vertical and horizontal directions. While the first component expresses a strong sinusoidal motion, the second component displays a decaying swing as was previously described as the horizontal motion of the object, but with a messier signal closer to the end of the video. Perhaps the addition of the abnormally strong third singular value would smooth this motion out. The rotation of the can could have caused this noise, as the lighter parts of the can express horizontal movement from the perspective of the tracking algorithm.

5 Summary and Conclusions

Given three sets of camera perspectives of the same object, we were able to track its motion and reduce the redundancy of the system. These tests highlight the power and limits of the singular value decomposition and principle component analysis. While in most cases we were able to track and deconstruct the objects motion in the vertical and horizontal directions, finding consistent locations of the object proved difficult, and bad input data leads to bad output data. Regardless, most cases produced accurate rank approximations with less than or equal to 3 principal components, reducing the dimensionality of our data by half, and proving itself as an effective tool for organizing, reducing, and interpreting data.

Appendix A MATLAB Functions

- `f = fft2(I)` returns the Discrete Fourier Transform of image `I` along two dimensions, inverting the indices on both sides of the axis.
- `ind = find(I < x)` returns the indices of the image `I` which satisfy the condition expressed as the paramete.
- `I = ifft2(If)` returns the inverse of the two dimensional Discrete Fourier Transform of image `If`.

- `I = im2double(I)` converts an image `I` from $[0, 255]$ integer pixel values to doubles within $[0, 1]$
- `I = im2gray(I)` converts an RGB image to a grayscale image.
- `[x, y] = ind2sub(size, ind)` returns the coordinates corresponding to the single index for the `size` of a given object.

Appendix B MATLAB Code

```
% Matthew Mangione
% AMATH 482: Computational Methods for Data-Analysis
% Assignment 3: Spring-Mass System

%%  Test 1: Ideal Case   ------------------------------------------------

sample_rate = 4; % keeps 1 out of every n frames.

load('cam1_1.mat')
load('cam2_1.mat')
load('cam3_1.mat')

% reduce videos to greyscale
gvid1 = vid2gray(vidFrames1_1);
gvid2 = vid2gray(vidFrames2_1);
gvid3 = vid2gray(vidFrames3_1);

%implay(vidFrames1_1)
clear vidFrames1_1;
clear vidFrames2_1;
clear vidFrames3_1;

% align videos based on delays between videos (determined by eye)
[gvid1, gvid2, gvid3] = cropvids(gvid1, gvid2, gvid3, [1, 15, 6]);

gvid1 = samplevid(gvid1, sample_rate);
gvid2 = samplevid(gvid2, sample_rate);
gvid3 = samplevid(gvid3, sample_rate);

[~, t1] = trackvid(gvid1, .9, 25, .6); % vertical
[~, t2] = trackvid(gvid2, .9, 25, .85); % vertical
[~, t3] = trackvid(gvid3, .9, 25, .5); % horizontal
```

```
%%  Test 2: Noisy case ----------------------------------------------------

sample_rate = 3; % keeps 1 out of every n frames.

load('cam1_2.mat') load('cam2_2.mat') load('cam3_2.mat')


% reduce videos to greyscale
gvid1 = vid2gray(vidFrames1_2);
gvid2 = vid2gray(vidFrames2_2);
gvid3 = vid2gray(vidFrames3_2);

clear vidFrames1_2; clear vidFrames2_2; clear vidFrames3_2;

% align videos based on delays between videos (determined by eye)
[gvid1, gvid2, gvid3] = cropvids(gvid1, gvid2, gvid3, [13, 1, 18]);

gvid1 = samplevid(gvid1, sample_rate);
gvid2 = samplevid(gvid2, sample_rate);
gvid3 = samplevid(gvid3, sample_rate);

gvid1 = cropvid(gvid1, 250, 640, 1, 480);
gvid2 = cropvid(gvid2, 100, 500, 1, 480);
gvid3 = cropvid(gvid3, 250, 640, 100, 300);

[~, t1] = trackvid(gvid1, .95, 25, .4); % vertical
[~, t2] = trackvid(gvid2, .95, 25, .85); % vertical
[~, t3] = trackvid(gvid3, .95, 25, .4); % horizontal

%%  Test 3: Horizontal Displacement----------------------------------------

sample_rate = 5; % keeps 1 out of every n frames.

load('cam1_3.mat') load('cam2_3.mat') load('cam3_3.mat')

% reduce videos to greyscale
gvid1 = vid2gray(vidFrames1_3);
gvid2 = vid2gray(vidFrames2_3);
gvid3 = vid2gray(vidFrames3_3);

clear vidFrames1_3; clear vidFrames2_3; clear vidFrames3_3;

% align videos based on delays between videos (determined by eye)
[gvid1, gvid2, gvid3] = cropvids(gvid1, gvid2, gvid3, [8, 32, 1]);

gvid1 = samplevid(gvid1, sample_rate);
gvid2 = samplevid(gvid2, sample_rate);
gvid3 = samplevid(gvid3, sample_rate);

gvid1 = cropvid(gvid1, 140, 540, 240, 480);
gvid2 = cropvid(gvid2, 140, 500, 150, 480);
gvid3 = cropvid(gvid3, 250, 640, 100, 300);

[~, t1] = trackvid(gvid1, .95, 25, .4); % vertical
[~, t2] = trackvid(gvid2, .95, 25, .85); % vertical
[~, t3] = trackvid(gvid3, .95, 25, .4); % horizontal
```

```
%%  Test 4: Horizontal & Rotation----------------------------------------

sample_rate = 4; % keeps 1 out of every n frames.

load('cam1_4.mat') load('cam2_4.mat') load('cam3_4.mat')

% reduce videos to greyscale
gvid1 = vid2gray(vidFrames1_4);
gvid2 = vid2gray(vidFrames2_4);
gvid3 = vid2gray(vidFrames3_4);

% implay(vidFrames1_4)
% implay(vidFrames2_4)
% implay(vidFrames3_4)

clear vidFrames1_4; clear vidFrames2_4; clear vidFrames3_4;

% align videos based on delays between videos (determined by eye)
[gvid1, gvid2, gvid3] = cropvids(gvid1, gvid2, gvid3, [1, 1, 1]);

gvid1 = samplevid(gvid1, sample_rate);
gvid2 = samplevid(gvid2, sample_rate);
gvid3 = samplevid(gvid3, sample_rate);

gvid1 = cropvid(gvid1, 140, 540, 240, 480);
gvid2 = cropvid(gvid2, 140, 500, 150, 480);
gvid3 = cropvid(gvid3, 250, 640, 100, 300);

[~, t1] = trackvid(gvid1, .9, 25, .6); % vertical
[~, t2] = trackvid(gvid2, .95, 25, .85); % vertical
[~, t3] = trackvid(gvid3, .85, 25, .4); % horizontal

%% Principal Componenent Analysis (Same for all tests)----------------------
%  run this section after pre-processing of specific test.

% fill in empty data with nearest neighbor / average of
% neighbors. function below
t1 = fillNaN(t1);
t2 = fillNaN(t2);
t3 = fillNaN(t3);


X = [t1; t2; t3]; % SWITCH horizontal & VERTICAL
[m, n] = size(X);
row_aves = mean(X, 2);
X = X - row_aves;
%plot(1:length(X(3,:)), X(3,:))

[U, S, V] = svd( X' / sqrt(n - 1), 'econ');
S2 = S(1:m, 1:m).^2;

% create principle components
Y = V' * X;
r1p = U(:,1)*S(1,1)*V(:,1)';
r2p = r1p + U(:,2)*S(2,2)*V(:,2)';
r3p = r2p + U(:,3)*S(3,3)*V(:,3)';
```

```
%% PLOTS

figure(1)
plot(1:m, diag(S2)/max(diag(S2)), 'ro', 'LineWidth', 2)
set(gca, 'fontsize', 14);
title('Horizontal & Rotational Displacement Case');
xlabel('Index of Singular Value');
ylabel('Singular Value (normalized)');

figure(2) % plots PRINCIPLE COMPONENTS (1 should be best)
axes;
plot(1:n ,Y(1,:), 'LineWidth', 1.5); hold on;
plot(1:n ,Y(2,:), 'LineWidth', 1.5);
plot(1:n ,Y(3,:), 'LineWidth', 1.5);
%plot(1:n ,Y(4,:), 'LineWidth', 1.5);

set(gca, 'fontsize', 14);
legend('Principal Component 1','Principal Component 2', 'Principal Component 3');
ylabel('Position (pixels)');
xlabel('Time (video frames)');
title('Principal Components for Horizontal & Rotational Displacement Case');

figure(3)
plot(1:n ,r1p(:,1), 'r', 'LineWidth', 1); hold on;
plot(1:n ,r2p(:,1), 'k', 'LineWidth', 1);
plot(1:n ,r3p(:,1), 'k', 'LineWidth', 1);


figure(4)
plot3(1:n, Y(3,:), Y(1,:), 'LineWidth', 1.5)
set(gca, 'fontsize', 14);
xlabel('Time (video frames)');
ylabel('Horizontal Position (pixels)');
zlabel('Vertical Position (pixels)');
title('Position vs Time');

%% Functions  ------------------------------------------------------------

function [gvid] = vid2gray(rgb_vid)

    numFrames = size(rgb_vid);
    gvid = zeros(numFrames(1), numFrames(2), numFrames(4));

    for j = 1:numFrames(4)
        X = rgb_vid(:,:,:,j);
        gvid(:,:,j) = im2double(rgb2gray(X));
    end
end
```

```
% aligns videos and truncates them based on (delay)
function [vid1, vid2, vid3] = cropvids(vid1, vid2, vid3, delay)


    vid1 = vid1(:,:, delay(1):end);
    vid2 = vid2(:,:, delay(2):end);
    vid3 = vid3(:,:, delay(3):end);

    num_frames = min([size(vid1,3) size(vid2,3) size(vid3,3)]) - 1;

    vid1 = vid1(:,:, 1:(num_frames));
    vid2 = vid2(:,:, 1:(num_frames));
    vid3 = vid3(:,:, 1:(num_frames));
end

% crops a videos length based on inputed pixel range
function [nvid] = cropvid(vid, y1, y2, x1, x2)
    nvid = zeros(x2-x1+1, y2-y1+1, size(vid, 3));
    for frame = 1:size(vid, 3)
        for i = x1:x2
            for j = y1:y2
                nvid(i + 1 - x1, j + 1 - y1, frame) = vid(i, j, frame);
            end
        end
    end
end

% keeps 1 out of every (jump) video frames
function [svid] = samplevid(vid, jump)
    numFrames = size(vid);
    ind = 1;
    for j = 1:jump:numFrames(3)
        X = vid(:,:,j);
        svid(:,:,ind) = X;
        ind = ind + 1;
    end

end

% Non-max supression: any pixels less than cut ([0, 1])
% are set to 0. Input a video.
function [vid] = brightpass(vid, cut)
    numFrames = size(vid);
    for j = 1:numFrames(3)
        vid(:,:,j) = blur(vid(:,:,j), 50);
        for x = 1:numFrames(1)
            for y = 1:numFrames(2)
                if vid(x,y,j) < cut
                    vid(x,y,j) = 0;
                end
            end
        end
    end
end
```

```
% Given a pre-processed grayscale video this function isolates
% the movement of the paint can and returns the averaged coordinates
% of its location through time. To do this, this function filters based
% on brightness, then based on movement, and then takes an average of the
% remaining pixels' location.
function [gvid, track] = trackvid(vid, bright1, blur_cnst, bright2)
    vid = brightpass(vid, bright1);
    numFrames = size(vid);
    gvid = ones(numFrames(1), numFrames(2), numFrames(3)-1);
    track = zeros(2, numFrames(3)-2);
    for j = 2:numFrames(3)
        frame_diff = abs(minus(vid(:,:,j), vid(:,:,j-1)));
        frame_diff = blur(frame_diff, blur_cnst);
        gvid(:,:,j-1) = frame_diff;
    end

    gvid = brightpass(gvid, bright2);
    for j = 1:numFrames(3)-2
        brights = find(gvid(:,:,j) > bright2);
        [x, y] = ind2sub(size(frame_diff), brights);
        if (j > 1 && (length(x) < 15 || length(y) < 15))
            track(:,j) = mean(track(:, 1:j-1), 2);
        end
        track(:,j) = [round(mean(x)); round(mean(y))];
    end
end

% Applies a Shannon Filter to inputted image
% n represents the k number of frequencies allowed
% to pass the filter.
function [frame] = blur(frame, n)
    ff = fft2(frame);
    filt = zeros(size(frame));
    x = double(size(filt,1)/2);
    y = double(size(filt,2)/2);
    [x, y] = size(frame);
    filt([1:n+1 x-n+1:x], [1:n+1 y-n+1:y]) = ones((2*n+1), (2*n+1));
    ff = filt .* ff;
    frame = abs(ifft2(ff));

end

% given a 2 x n vector, interpolates value of NaN
function [t] = fillNaN(t)
    tNaN = find(isnan(t));
    for j = 1:length(tNaN)
        [i,k] = ind2sub(size(t), tNaN(j));
        if k == 1
            t(i,k) = t(i,k+1);
        elseif k == length(t)
            t(i,k) = t(i,k-1);
        else
            t(i,k) = round((t(i,k-1) + t(i,k+1))/2);
        end
    end
end
```