Principal Component Analysis of Simple Harmonic Motion

Alyssa Mell

Abstract

Using Principal Component Analysis (PCA), we study a total of twelve videos, separated into four trials, with three cameras filming each time. These videos show a paint can undergoing simple harmonic motion, pendulum motion, rotation, and random motion. First, we extract the mass positions of the paint can from each of the video frames using vision.PointTracker in MATLAB. Then, we study the differences in the singular value decomposition of the position matrices for each of the four trials. Finally, we draw conclusions about the PCA, its usefulness, and how it is affected by these various types of motion and noise.

Sec. I. Introduction and Overview

PCA is a useful tool in analyzing the main components of the variance of a data set. It is also useful when trying to reduce the size of a large data set by finding the best rank approximation. Here, the Singular Value Decompositon (SVD) method is used to conduct PCA. The SVD of a matrix separates that matrix into its singular values and corresponding left and right eigenvectors. The singular values of this decomposition allow one to determine how much energy is contained in each of the principal components, and the eigenvectors provide information about the directions of those components.

We use the SVD on the data sets for each video in this paper in order to determine the rank of each position matrix and, consequently, the number of principal components needed to span the main data set for each trial. The first trial is an ideal case with motion in only the z direction. The second trial is a noisy (shaky camera) version of the ideal case. In the third trial, the paint can is intentionally displaced in both the horizontal $x - y$ plane, and the vertical z direction. The fourth trial is the same as the third with the addition of rotation. Through the use of PCA, we make inferences about the differences in these trials and determine how and if these reflect our existing knowledge of the motion in each video.

Sec. II. Theoretical Background

The main method used for data analysis in this paper is Singular Value Decomposition (SVD), which aids in Principal Component Analysis (PCA). Equation 1 shows the SVD of a matrix A.

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

In equation 1, A is the original matrix, Σ is a diagonal matrix containing the singular values of A, U contains the left eigenvectors of A, and V^T contains the right eigenvectors of A. The number of large singular values in Σ gives information about the rank of A, or the number of linearly independent rows contained in A. In order to determine what constitutes a "large" singular value, the energy contained in each mode of the original matrix can be found using equation 2. An individual mode approximation is given by the multiplication of U_n, $\Sigma_{n,n}$, and V_n^T, where the subscript n represents the specific mode chosen. Only the n^{th} column of U and V, and the n^{th} diagonal value contained in Σ are used in the approximation for each mode. The energy of a mode represents the amount of the original data contained in that approximation of the matrix.

$$\text{energy}_n = \frac{\sigma_n^2}{\sigma_1^2 + \sigma_2^2 + ... + \sigma_r^2} \tag{2}$$

In equation 2, σ represents the singular value, n represents the specific number of the chosen singular value, and r is the total number of singular values. Equation 2 can be altered to give the energy of a specific rank approximation of A. This is shown in equation 3.

$$\text{energy}_N = \frac{\sigma_1^2 + \sigma_2^2 + ... + \sigma_N^2}{\sigma_1^2 + \sigma_2^2 + ... + \sigma_r^2} \tag{3}$$

In equation 3, σ represents the singular value, N represents the number of the rank approximation, and r is the total number of singular values. This equation gives the total energy contained by a given rank approximation of A. A rank 1 approximation for a given matrix A is found by multiplying the first singular value of S by the eigenvectors contained in the first column of U and the first column of V^T. A rank 2 approximation is found by doing the same with the second singular value of S and the eigenvectors contained in the second columns of U and V^T, and then summing this with the rank 1 approximation. A rank n approximation is found by doing the same multiplication and addition with n singular values and eigenvectors.

The principal components of a matrix A are found by multiplying U and S. The singular values contained in S provide a scale for the principal components, and the vectors contained in U provide the direction of those components. The size and direction of these components provide key information about the main directions of variance of the data, or the orthogonal modes of the data.

Sec. III. Algorithm Implementation and Development

In this section there will often be references to the MATLAB code which is located in Appendix B. The specific line of the code that is being referenced will be stated at the end of the sentence.

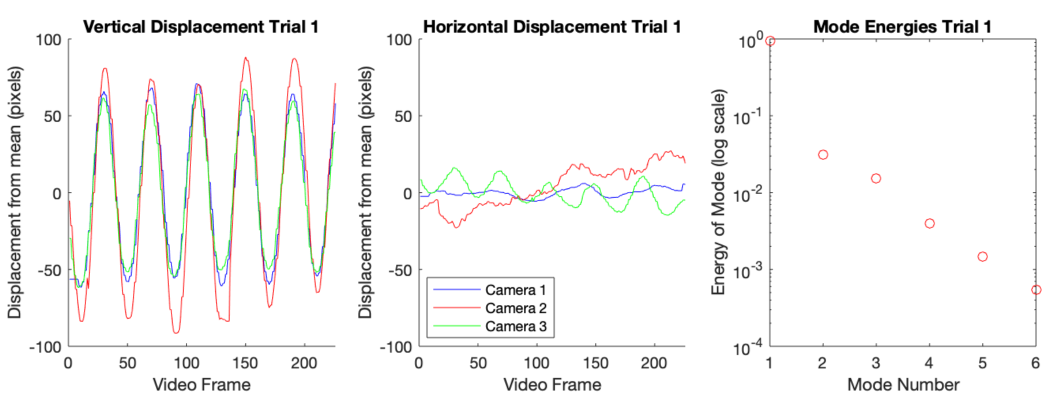
The main method for solving this problem was repeated for all three cameras in each of the four trials, so we will give a general procedure for how the data was gathered and analyzed for a single camera and trial. In order to solve this problem with the following method in MATLAB, the Computer Vision toolbox must be installed. In this toolbox, a tool called vision.PointTracker was used. This tool uses the Kanade-Lucas-Tomasi algorithm in order to track a given point. This algorithm studies the motion or change in each image frame in order to determine the point's subsequent locations. It must first be initialized with a starting image frame and an specific position within that frame (line 28).

Since we are analyzing videos, we first separated them into their individual frames and then created a `for` loop to track and update the point each time. The paint can in the videos had a flashlight on top of it, so this lit point in each image frame made for a natural tracking location. However, some videos were buggier than others and the loop had to be paused and manually updated to reset the point to the correct location using a function called setPoints(). In order to determine where these bugs occurred, the videos were played with the expected point overlaid on top of each frame. When the point moved too far from the flashlight it was updated with the setPoints() function. If the bright part of the flashlight went out of sight temporarily, the point was set to be as close as possible to its actual location. The data for the vertical and horizontal positions of the paint can at each frame, and for each of the three cameras, was saved into a large matrix of six rows for each trial. This matrix is called a snapshot matrix because each column represents a snapshot in time containing the pixel position of the paint can at that given time for each camera.

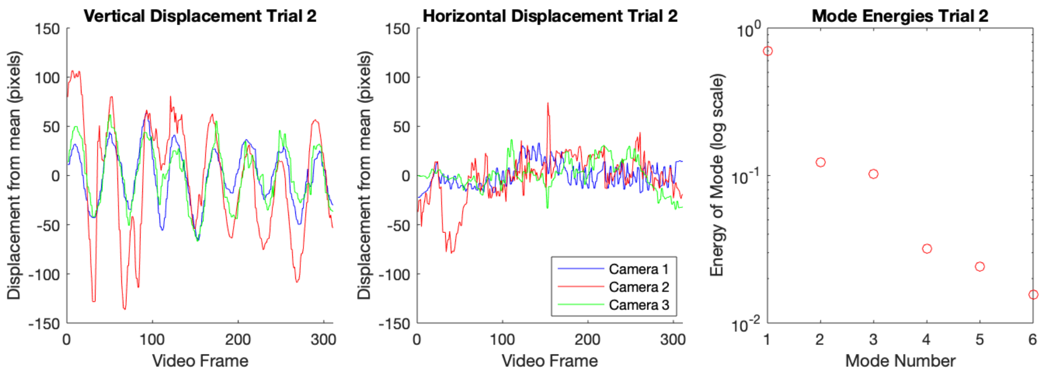
In order to ensure that the data for each trial was being measured on the same scale, we subtracted the mean of each row from that specific row so that all rows were centered around zero (lines 48-49, 91-92, 128-129). Also, by looking at the plots of vertical displacement of the paint can for each camera of each trial (as well as horizontal displacement in trials 3 and 4) we were able to ensure that the correct subset of points were chosen. This extra step was necessary because some of the cameras began recording at different times, so it was important to ensure that only data from overlapping times was being used. The SVD (equation 1) of this data was then taken and the energy of each mode was found (lines 164-171). The plots of the position of the paint can for each trial and the corresponding energies of each mode are provided in Sec IV. below.

Sec. IV. Computational Results

Figures 1 through 12 contain visual representations of the displacement of the paint can and the corresponding energies of the modes for each trial. The mode energy plots are measured



Figures 1, 2, and 3 (from left to right): Trial 1 - ideal case.

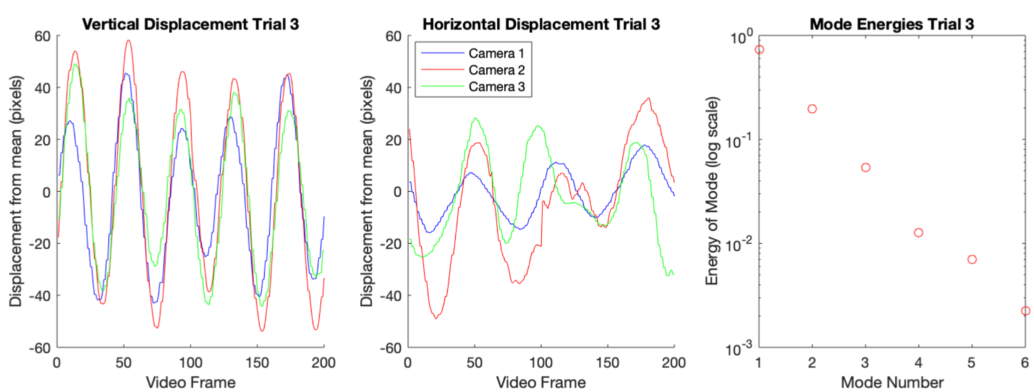


Figures 4, 5, and 6 (from left to right): Trial 2 - noisy case.

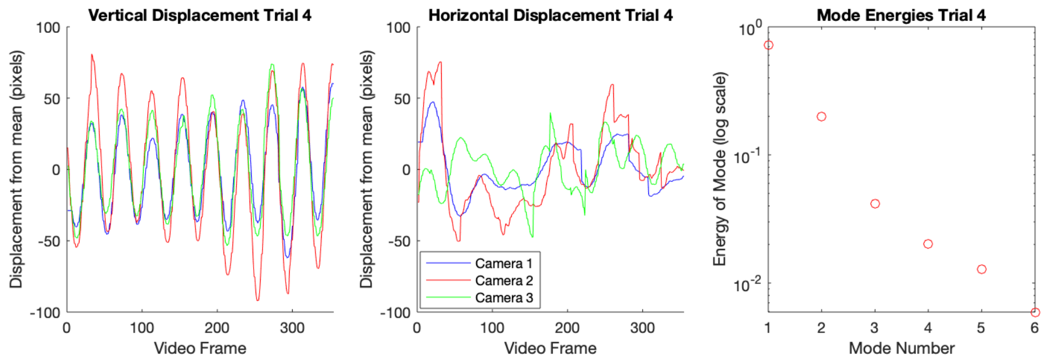
with a log scale on the y axis to account for the large difference in magnitudes of different modes. All 12 figures account for the fact that camera 3 is turned sideways. The horizontal and vertical values gathered with PointTracker are swapped in the plots for this camera.

Figures 1 and 2 and figures 4 and 5 show the vertical and horizontal displacement of the paint can in trial 1 (ideal case) and trial 2 (noisy case) respectively. Figures 3 and 6 use equation 2 to show the energies contained in each mode of these two trials. The first three modes in trial 1 contain 94.71%, 3.15%, and 1.54% of the total energy. In trial 2, these values are 70.26%, 12.31%, and 10.30%. For trial 1, a rank 1 approximation is sufficient in covering a main portion of the data, but in trial 2, due to the noise, there are three significant energy modes. The largest energy mode for each of these trials reflects the large vertical motion of the paint can in the z direction.

Figures 7 and 8 and figures 10 and 11 show the vertical and horizontal displacement for trial 3 (the case with intentional horizontal displacement) and trial 4 (the case with intentional rotation and horizontal displacement) respectively. Figures 9 and 12 use equation 2 to show the energies contained in each of these two trials. The first three modes in trial 3 contain 72.92%, 19.52%, and 5.38% of the total energy. In trial 4, these values are 72.04%, 19.92%, and 4.16%. The large first two energies reflect the fact that the data has two large two principal components, one pointing in the vertical z direction and one going along the $x-y$



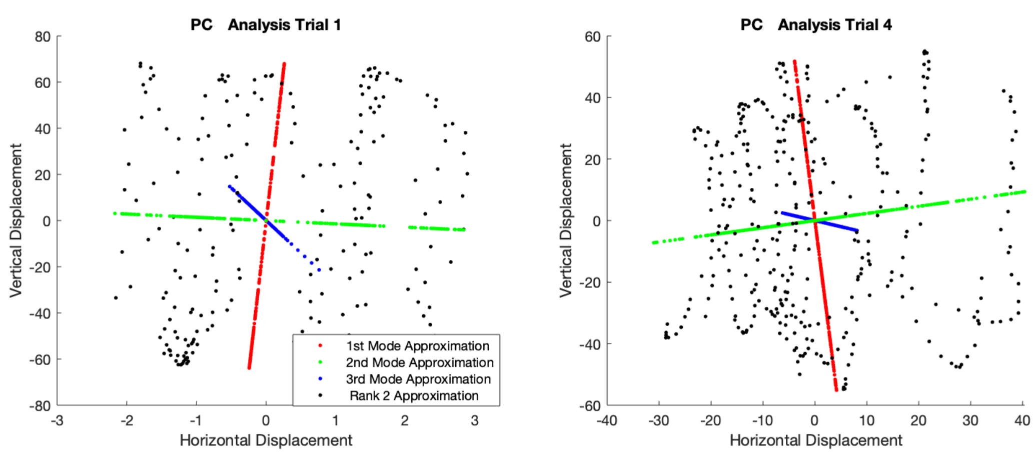
Figures 7, 8, and 9 (from left to right): Trial 3 - intentional horizontal displacement.



Figures 10, 11, and 12 (from left to right): Trial 4 - intentional rotation and horizontal displacement.

plane. In comparison to figure 8, the rotation has affected figure 11 and added additional noise to the plot. Also, in figure 12, the lowest modes (4, 5, and 6) are larger than in figure 9. This reflects the fact that there is additional movement in the videos with the added rotation of the paint can, causing the smaller modes to contain more data.

Figure 13 provides information about the vectors contained in U and V from the singular value decomposition. To reduce repetitive plots, we have only included figures of this for trials 1 and 4. In six dimensions, it is difficult to plot the values contained in U and V directly, but when used to create rank and mode approximations, it is easier to analyze these values. U contains important information regarding the main directions of variance of the data as seen in figure 13 where the variance of the modes are plotted in red, green, and blue. The main directions of the principal components for all trials is directly in the z direction (red) and the $x-y$ plane (green). The black points in figure 13 show the rank 2 approximation for the points contained in trials 1 and 4. The matrix V provides information necessary to project the values of the original matrix, A, onto a given principal component of A. The largest principal components point in the direction of the greatest variance where the largest number of points will be clearly visible when projected onto this axis.



Figures 13: Principal component analysis for trials 1 and 4.

Sec. V. Summary and Conclusions

After analyzing these four different trials taken from various camera angles and involving different types of motion, we can determine some useful information about Principal Component Analysis. This method of analysis provides information about the variance of a given data set as well as the possible noisiness when compared to an ideal case. If the smallest energies of the modes for a given data set are much larger than those in the ideal case, then it is likely that the case being studied has some noise. Also, the number of large energy modes tells us about the rank of the given matrix and how many principal component axes it requires to fully capture the data. The types and directions of motion largely contribute to the number of modes required to accurately cover the data sets. As expected, more directions of motion and additional noise caused the number of significant modes to increase.

Appendix A. MATLAB functions

Listed below are the important MATLAB functions used in this paper, along with a brief explanation of how they are implemented.

- pointTracker = `vision.PointTracker` : returns an object pointTracker that tracks a set of points using the Kanade-Lucas-Tomasi algorithm. Initial point location must be specified.
- `initialize`(pointTracker, POINTS, I) : sets initial point and video frame for the pointTracker.
- POINTS = `step`(pointTracker, I) : tracks the point in the input frame, I, and returns a 2x1 matrix containing the location of the point in the frame I.
- `setPoints`(pointTracker, POINTS) : manually sets the point at a given step of the iteration, useful when a point needs to be reset or centered.
- `imshow`(I) : displays the given image, I.
- [U,S,V] = `svd`(X, 'econ') : finds the "economy size" singular value decomposition of X. If x is m-by-n, where m < n, only the first m columns of V are computed and S is m-by-m.
- `diag`(X) : returns the main diagonal values of X as a vector.
- `cumsum`(X) : computes the cumulative sum of the elements in X.

Appendix B. MATLAB Code

```
clear; close all; clc

% Load video data
load('cam1_1.mat')
load('cam1_2.mat')
load('cam1_3.mat')
load('cam1_4.mat')
load('cam2_1.mat')
load('cam2_2.mat')
load('cam2_3.mat')
load('cam2_4.mat')
load('cam3_1.mat')
load('cam3_2.mat')
load('cam3_3.mat')
load('cam3_4.mat')

%% Trial 1
close all; % closes open figures

% Trial 1 Camera 1
numFrames1 = size(vidFrames1_1,4); % number of video frames for
    camera 1
pointTracker = vision.PointTracker; % create PointTracker object
points1 = zeros(2, numFrames1); % initialize points1 matrix
I = vidFrames1_1(:,:,:,1); % first video frame for camera 1
%imshow(I); % use data tips on image (under tools tab) to find
    initial point
initial_point = [319 228]; % initial point of flashlight
initialize(pointTracker,initial_point,I) % initialize PointTracker

% find and save points for each time step
for ii = 1:numFrames1
    I = vidFrames1_1(:,:,:,ii);
    points1(:,ii) = step(pointTracker, I);
end

%{
% shows where points are expected to be
for ii = 1:numFrames1
    I = vidFrames1_1(:,:,:,ii);
    imshow(I);
```

```
    hold on
    plot(points1(1,ii),points1(2,ii),'r+', 'MarkerSize', 20);
    pause(0.1)
end
%}

% centers points around mean value
points1(1,:) = points1(1,:) - mean(points1(1,:));
points1(2,:) = points1(2,:) - mean(points1(2,:));


% Trial 1 Camera 2
numFrames2 = size(vidFrames2_1,4); % number of video frames for
    camera 2
pointTracker = vision.PointTracker; % create PointTracker object
points2 = zeros(2, numFrames2); % initialize points2 matrix
I = vidFrames2_1(:,:,:,1); % first video frame for camera 2
%imshow(I); % use data tips on image to find initial point
initial_point = [271 278]; % initial point of flashlight
initialize(pointTracker,initial_point,I) % initialize PointTracker

% find and save points for each time step
for ii = 1:numFrames2
    if ii == 20
        setPoints(pointTracker, [273 110])
    end
    if ii == 25
        setPoints(pointTracker, [277 140])
    end
    if ii == 66
        setPoints(pointTracker, [281 142])
    end
    if ii == 186
        setPoints(pointTracker, [313 158])
    end
    I = vidFrames2_1(:,:,:,ii);
    points2(:,ii) = step(pointTracker, I);
end

%{
% shows where points are expected to be
for ii = 1:numFrames2
    I = vidFrames2_1(:,:,:,ii);
    imshow(I);
    hold on
```

```
    plot(points2(1,ii),points2(2,ii),'r+', 'MarkerSize', 20);
    pause(0.1)
end
%}

% centers points around mean value
points2(1,:) = points2(1,:) - mean(points2(1,:));
points2(2,:) = points2(2,:) - mean(points2(2,:));


% Trial 1 Camera 3
numFrames3 = size(vidFrames3_1,4); % number of video frames for camera 3
pointTracker = vision.PointTracker; % create PointTracker object
points3 = zeros(2, numFrames3); % initialize points3 matrix
I = vidFrames3_1(:,:,:,1); % first video frame for camera 3
%imshow(I); % use data tips on image (under tools tab) to find initial point
initial_point = [317 272]; % initial point of flashlight, got from plot
initialize(pointTracker,initial_point,I) % initialize PointTracker

% find and save points at each time step
for ii = 1:numFrames3
    if ii == 97
        setPoints(pointTracker, [320 258])
    end
    if ii == 139
        setPoints(pointTracker, [341 262])
    end
    I = vidFrames3_1(:,:,:,ii);
    points3(:,ii) = step(pointTracker, I);
end

%{
% shows where points are expected to be
for ii = 1:numFrames3
    I = vidFrames3_1(:,:,:,ii);
    imshow(I);
    hold on
    plot(points3(1,ii),points3(2,ii),'r+', 'MarkerSize', 20);
    pause(0.1)
end
%}
```

```
% centers points around mean value
points3(1,:) = points3(1,:) - mean(points3(1,:));
points3(2,:) = points3(2,:) - mean(points3(2,:));

% save all points from trial in single matrix
trial1 = zeros(6, 226);
trial1(1:2, :) = points1(:,1:226);
trial1(3:4, :) = points2(:,50:275);
trial1(5:6, :) = points3(:,1:226);

% plots vertical and horizontal displacement of paint can for all three
% cameras on same figure
figure(1)
subplot(1,3,1)
ylim([-100 100]); xlim([0 226]);
xlabel('Video Frame'); ylabel('Displacement from mean (pixels)')
title('Vertical Displacement Trial 1');
hold on
plot(trial1(2,:) , 'b')
plot(trial1(4,:), 'r')
plot(trial1(5,:), 'g')
%legend('Camera 1', 'Camera 2', 'Camera 3');

subplot(1,3,2)
ylim([-100 100]); xlim([0 226]);
xlabel('Video Frame'); ylabel('Displacement from mean (pixels)')
title('Horizontal Displacement Trial 1');
hold on
plot(trial1(1,:) , 'b')
plot(trial1(3,:), 'r')
plot(trial1(6,:), 'g')
legend('Camera 1', 'Camera 2', 'Camera 3');


% Trial 1 Analysis

% singular value decomposition for trial 1
[U1,S1,V1] = svd(trial1, 'econ');


% calculates energy of each rank appromiation
S1_vec = diag(S1).';
S1_vec = S1_vec.^2;
energy1 = S1_vec/sum(S1_vec(1,:));
```

```
energy1_sums = cumsum(S1_vec)/sum(S1_vec(1,:));

% log scale of energy covered by each rank approximation
subplot(1,3,3)
plot(1:6,energy1,'ro')
set(gca, 'YScale', 'log')
xlabel('Mode Number'); ylabel('Energy of Mode (log scale)')
title('Mode Energies Trial 1');

% principal components
figure(2)
hold on
X_comp1 = U1(:,1)*S1(1,1)*V1(:,1).';
plot3( X_comp1(1,:),X_comp1(2,:),X_comp1(3,:), 'r.');
X_comp2 = U1(:,2)*S1(2,2)*V1(:,2)';
plot3( X_comp2(1,:),X_comp2(2,:),X_comp2(3,:), 'g.');
X_comp3 = U1(:,3)*S1(3,3)*V1(:,3)';
plot3( X_comp3(1,:),X_comp3(2,:),X_comp3(3,:), 'g.');
X_rank2 = U1(:,1:2)*S1(1:2,1:2)*V1(:,1:2)';
plot3( X_rank2(1,:),X_rank2(2,:),X_rank2(3,:), 'b.');
xlabel('Horizontal Displacement'); ylabel('Vertical Displacement')
title('PCA Analysis')
legend('1st Principal Component', '2nd Principal Component', '
    Rank 2 Approximation');

%% Trial 2
close all; % closes all figures

% Trial 2 Camera 1
numFrames1 = size(vidFrames1_2,4);  % number of video frames for
    camera 1
pointTracker = vision.PointTracker; % create PointTracker object
points1 = zeros(2, numFrames1); % initialize points1 matrix
I = vidFrames1_2(:,:,:,1); % first video frame for camera 1
%imshow(I); % use data tips on image (under tools tab) to find
    initial point
initial_point = [325 310]; % initial point of flashlight
initialize(pointTracker,initial_point,I) % initialize PointTracker

% find and save points for each time step
for ii = 1:numFrames1
    I = vidFrames1_2(:,:,:,ii);
    points1(:,ii) = step(pointTracker, I);
end
```

```
%{
% shows where points are expected to be
for ii = 1:numFrames1
    I = vidFrames1_2(:,:,:,ii);
    imshow(I);
    hold on
    plot(points1(1,ii),points1(2,ii),'r+', 'MarkerSize', 20);
    pause(0.1)
end
%}

% centers points around mean value
points1(1,:) = points1(1,:) - mean(points1(1,:));
points1(2,:) = points1(2,:) - mean(points1(2,:));


% Trial 2 Camera 2
numFrames2 = size(vidFrames2_2,4);  % number of video frames for camera 2
pointTracker = vision.PointTracker; % create PointTracker object
points2 = zeros(2, numFrames2); % initialize points2 matrix
I = vidFrames2_2(:,:,:,1); % first video frame for camera 2
%imshow(I); % use data tips on image (under tools tab) to find initial point
initial_point = [315 362]; % initial point of flashlight
initialize(pointTracker,initial_point,I) % initialize PointTracker

% find and save points for each time step
for ii = 1:numFrames2
    if ii == 4
        setPoints(pointTracker, [301 334])
    end
    if ii == 59
        setPoints(pointTracker, [257 82])
    end
    if ii == 63
        setPoints(pointTracker, [231 98])
    end
    if ii == 249
        setPoints(pointTracker, [315 186])
    end
    if ii == 291
        setPoints(pointTracker, [331 136])
    end
    if ii == 311
```

```
        setPoints(pointTracker, [309 208])
    end
    if ii == 350
        setPoints(pointTracker, [391 278])
    end
    I = vidFrames2_2(:,:,:,ii);
    points2(:,ii) = step(pointTracker, I);
end


%{
% shows where points are expected to be
for ii = 1:numFrames2
    I = vidFrames2_2(:,:,:,ii);
    imshow(I);
    hold on
    plot(points2(1,ii),points2(2,ii),'r+', 'MarkerSize', 20);
    pause(0.1)
end
%}

% centers points around mean value
points2(1,:) = points2(1,:) - mean(points2(1,:));
points2(2,:) = points2(2,:) - mean(points2(2,:));


% Trial 2 Camera 3
numFrames3 = size(vidFrames3_2,4);  % number of video frames for camera 3
pointTracker = vision.PointTracker; % create PointTracker object
points3 = zeros(2, numFrames3); % initialize points3 matrix
I = vidFrames3_2(:,:,:,1); % first video frame for camera 3
%imshow(I); % use data tips on image (under tools tab) to find initial point
initial_point = [345 246]; % initial point of flashlight
initialize(pointTracker,initial_point,I) % initialize PointTracker

% find and save points for each time step
for ii = 1:numFrames3
    if ii == 222
        setPoints(pointTracker, [367 270])
    end
    if ii == 247
        setPoints(pointTracker, [335 260])
    end
```

```
    I = vidFrames3_2(:,:,:,ii);
    points3(:,ii) = step(pointTracker, I);
end

%{
% shows where points are expected to be
for ii = 1:numFrames3
    I = vidFrames3_2(:,:,:,ii);
    imshow(I);
    hold on
    plot(points3(1,ii),points3(2,ii),'r+', 'MarkerSize', 20);
    pause(0.1)
end
%}

% centers points around mean value
points3(1,:) = points3(1,:) - mean(points3(1,:));
points3(2,:) = points3(2,:) - mean(points3(2,:));

% save all points from trial in single matrix
trial2 = zeros(6, 311);
trial2(1:2, :) = points1(:,4:314);
trial2(3:4, :) = points2(:,30:340);
trial2(5:6, :) = points3(:,7:317);

% plots vertical and horizontal displacement of paint can for all
    three
% cameras on same figure
figure(1)
subplot(1,3,1)
ylim([-150 150]); xlim([0 311]);
xlabel('Video Frame'); ylabel('Displacement from mean (pixels)')
title('Vertical Displacement Trial 2');
hold on
plot(trial2(2,:) , 'b')
plot(trial2(4,:), 'r')
plot(trial2(5,:), 'g')
%legend('Camera 1', 'Camera 2', 'Camera 3');

subplot(1,3,2)
ylim([-150 150]); xlim([0 311]);
xlabel('Video Frame'); ylabel('Displacement from mean (pixels)')
title('Horizontal Displacement Trial 2');
hold on
plot(trial2(1,:) , 'b')
```

```
plot(trial2(3,:), 'r')
plot(trial2(6,:), 'g')
legend('Camera 1', 'Camera 2', 'Camera 3');


% Trial 2 Analysis

% singular value decomposition for trial 2
[U2,S2,V2] = svd(trial2, 'econ');


% calculates energy of each rank appromiation
S2_vec = diag(S2).';
S2_vec = S2_vec.^2;
energy2 = S2_vec/sum(S2_vec(1,:));
energy2_sums = cumsum(S2_vec)/sum(S2_vec(1,:));

% log scale of energy covered by each rank approximation
subplot(1,3,3)
plot(1:6,energy2,'ro')
set(gca, 'YScale', 'log')
xlabel('Mode Number'); ylabel('Energy of Mode (log scale)')
title('Mode Energies Trial 2');

% Modes plotted
figure(3)
plot(1:311,V2(:,1),'b',1:311,V2(:,2),'--r',1:311,V2(:,3),':k','Linewidth',2)

% principal components
figure(2)
hold on
X_comp1 = U2(:,1)*S2(1,1)*V2(:,1).';
plot3( X_comp1(1,:),X_comp1(2,:),X_comp1(3,:), 'r.');
X_comp2 = U2(:,2)*S2(2,2)*V2(:,2)';
plot3( X_comp2(1,:),X_comp2(2,:),X_comp2(3,:), 'g.');
X_rank2 = U2(:,1:2)*S2(1:2,1:2)*V2(:,1:2)';
plot3( X_rank2(1,:),X_rank2(2,:),X_rank2(3,:), 'b.');
xlabel('Horizontal Displacement'); ylabel('Vertical Displacement')
title('PCA Analysis')
legend('1st Principal Component', '2nd Principal Component', 'Rank 2 Approximation');

%% Trial 3
close all; % closes open figures
```

```
% Trial 3 Camera 1
numFrames1 = size(vidFrames1_3,4);  % number of video frames for
    camera 1
pointTracker = vision.PointTracker; % create PointTracker object
point1 = zeros(2, numFrames1); % initialize points1 matrix
I = vidFrames1_3(:,:,:,1); % first video frame for camera 1
%imshow(I); % use data tips on image (under tools tab) to find
    initial point
initial_point = [317 288]; % initial point of flashlight
initialize(pointTracker,initial_point,I) % initialize PointTracker

% find and save points for each time step
for ii = 1:numFrames1
    I = vidFrames1_3(:,:,:,ii);
    points1(:,ii) = step(pointTracker, I);
end

%{
% shows where points are expected to be
for ii = 1:numFrames1
    I = vidFrames1_3(:,:,:,ii);
    imshow(I);
    hold on
    plot(points1(1,ii),points1(2,ii),'r+', 'MarkerSize', 20);
    pause(0.1)
end
%}

% centers points around mean value
points1(1,:) = points1(1,:) - mean(points1(1,:));
points1(2,:) = points1(2,:) - mean(points1(2,:));


% Trial 3 Camera 2
numFrames2 = size(vidFrames2_3,4);  % number of video frames for
    camera 2
pointTracker = vision.PointTracker; % create PointTracker object
points2 = zeros(2, numFrames2); % initialize points2 matrix
I = vidFrames2_3(:,:,:,1); % first video frame for camera 2
%imshow(I); % use data tips on image (under tools tab) to find
    initial point
initial_point = [249 294]; % initial point of flashlight
initialize(pointTracker,initial_point,I) % initialize PointTracker

```

```
% find and save points for each time step
for ii = 1:numFrames2
    if ii == 10
        setPoints(pointTracker, [299 298])
    end
    if ii == 96
        setPoints(pointTracker, [307 250])
    end
    if ii == 133
        setPoints(pointTracker, [307 274])
    end
    if ii == 260
        setPoints(pointTracker, [323 232])
    end
    I = vidFrames2_3(:,:,:,ii);
    points2(:,ii) = step(pointTracker, I);
end

%{
% shows where points are expected to be
for ii = 1:numFrames2
    I = vidFrames2_3(:,:,:,ii);
    imshow(I);
    hold on
    plot(points2(1,ii),points2(2,ii),'r+', 'MarkerSize', 20);
    pause(0.1)
end
%}

% centers points around mean value
points2(1,:) = points2(1,:) - mean(points2(1,:));
points2(2,:) = points2(2,:) - mean(points2(2,:));


% Trial 3 Camera 3
numFrames3 = size(vidFrames3_3,4);  % number of video frames for camera 3
pointTracker = vision.PointTracker; % create PointTracker object
points3 = zeros(2, numFrames3); % initialize points3 matrix
I = vidFrames3_3(:,:,:,1); % first video frame for camera 3
%imshow(I); % use data tips on image (under tools tab) to find initial point
initial_point = [351 230]; % initial point of flashlight
initialize(pointTracker,initial_point,I) % initialize PointTracker

```

```
% find and save points for each time step
for ii = 1:numFrames3
    I = vidFrames3_3(:,:,:,ii);
    points3(:,ii) = step(pointTracker, I);
end

%{
% shows where points are expected to be
for ii = 1:numFrames3
    I = vidFrames3_3(:,:,:,ii);
    imshow(I);
    hold on
    plot(points3(1,ii),points3(2,ii),'r+', 'MarkerSize', 20);
    pause(0.1)
end
%}

% centers points around mean value
points3(1,:) = points3(1,:) - mean(points3(1,:));
points3(2,:) = points3(2,:) - mean(points3(2,:));

% save all points from trial in single matrix
trial3 = zeros(6, 200);
trial3(1:2, :) = points1(:,8:207);
trial3(3:4, :) = points2(:,33:232);
trial3(5:6, :) = points3(:,38:237);

% plots vertical and horizontal displacement of paint can for all
    three
% cameras on same figure
figure(1)
subplot(1,3,1)
ylim([-60 60]); xlim([0 200]);
xlabel('Video Frame'); ylabel('Displacement from mean (pixels)')
title('Vertical Displacement Trial 3');
hold on
plot(trial3(2,:) , 'b')
plot(trial3(4,:), 'r')
plot(trial3(5,:), 'g')
%legend('Camera 1', 'Camera 2', 'Camera 3');

subplot(1,3,2)
ylim([-60 60]); xlim([0 200]);
xlabel('Video Frame'); ylabel('Displacement from mean (pixels)')
title('Horizontal Displacement Trial 3');
```

```
hold on
plot(trial3(1,:) , 'b')
plot(trial3(3,:), 'r')
plot(trial3(6,:), 'g')
legend('Camera 1', 'Camera 2', 'Camera 3');


% Trial 3 Analysis

% singular value decomposition for trial 3
[U3,S3,V3] = svd(trial3, 'econ');

% calculates energy of each rank appromiation
S3_vec = diag(S3).';
S3_vec = S3_vec.^2;
energy3 = S3_vec/sum(S3_vec(1,:));
energy3_sums = cumsum(S3_vec)/sum(S3_vec(1,:));

% log scale of energy covered by each rank approximation
subplot(1,3,3)
plot(1:6,energy3,'ro')
set(gca, 'YScale', 'log')
xlabel('Mode Number'); ylabel('Energy of Mode (log scale)')
title('Mode Energies Trial 3');

% principal components
figure(2)
hold on
X_comp1 = U3(:,1)*S3(1,1)*V3(:,1).';
plot3( X_comp1(1,:),X_comp1(2,:),X_comp1(3,:), 'r.');
X_comp2 = U3(:,2)*S3(2,2)*V3(:,2)';
plot3( X_comp2(1,:),X_comp2(2,:),X_comp2(3,:), 'g.');
X_rank2 = U3(:,1:2)*S3(1:2,1:2)*V3(:,1:2)';
plot3( X_rank2(1,:),X_rank2(2,:),X_rank2(3,:), 'b.');
xlabel('Horizontal Displacement'); ylabel('Vertical Displacement')
title('PCA Analysis')
legend('1st Principal Component', '2nd Principal Component', 'Rank 2 Approximation');

%% Trial 4
close all; % closes open figures

% Trial 4 Camera 1
numFrames1 = size(vidFrames1_4,4);  % number of video frames for camera 1
```

```
pointTracker = vision.PointTracker; % create PointTracker object
points1 = zeros(2, numFrames1); % initialize points1 matrix
I = vidFrames1_4(:,:,:,1); % first video frame for camera 1
%imshow(I); % use data tips on image (under tools tab) to find initial point
initial_point = [400 264]; % initial point of flashlight
initialize(pointTracker,initial_point,I) % initialize PointTracker

% find and save points for each time step
for ii = 1:numFrames1
    if ii == 219
        setPoints(pointTracker, [374 266])
    end
    if ii == 282
        setPoints(pointTracker, [387 294])
    end
    if ii == 308
        setPoints(pointTracker, [364 324])
    end
    I = vidFrames1_4(:,:,:,ii);
    points1(:,ii) = step(pointTracker, I);
end

%{
% shows where points are expected to be
for ii = 1:numFrames1
    I = vidFrames1_4(:,:,:,ii);
    imshow(I);
    hold on
    plot(points1(1,ii),points1(2,ii),'r+', 'MarkerSize', 20);
    pause(0.1)
end
%}

% centers points around mean value
points1(1,:) = points1(1,:) - mean(points1(1,:));
points1(2,:) = points1(2,:) - mean(points1(2,:));


% Trial 4 Camera 2
numFrames2 = size(vidFrames2_4,4);  % number of video frames for camera 2
pointTracker = vision.PointTracker; % create PointTracker object
points2 = zeros(2, numFrames2); % initialize points2 matrix
I = vidFrames2_4(:,:,:,1); % first video frame for camera 2
```

```
%imshow(I); % use data tips on image (under tools tab) to find
    initial point
initial_point = [245 246]; % initial point of flashlight
initialize(pointTracker,initial_point,I) % initialize PointTracker

% find and save points for each time step
for ii = 1:numFrames2
    if ii == 6
        setPoints(pointTracker, [277 214])
    end
    if ii == 9
        setPoints(pointTracker, [301 196])
    end
    if ii == 10
        setPoints(pointTracker, [305 182])
    end
    if ii == 39
        setPoints(pointTracker, [337 266])
    end
    if ii == 64
        setPoints(pointTracker, [267 144])
    end
    if ii == 214
        setPoints(pointTracker, [305 144])
    end
    if ii == 268
        setPoints(pointTracker, [329 158])
    end
    if ii == 288
        setPoints(pointTracker, [307 200])
    end
    if ii == 293
        setPoints(pointTracker, [313 148])
    end
    if ii == 302
        setPoints(pointTracker, [291 114])
    end
    if ii == 331
        setPoints(pointTracker, [285 178])
    end
    I = vidFrames2_4(:,:,:,ii);
    points2(:,ii) = step(pointTracker, I);
end

%{
```

```
% shows where points are expected to be
for ii = 290:numFrames2
    I = vidFrames2_4(:,:,:,ii);
    imshow(I);
    hold on
    plot(points2(1,ii),points2(2,ii),'r+', 'MarkerSize', 20);
    pause(0.1)
end
%}

% centers points around mean value
points2(1,:) = points2(1,:) - mean(points2(1,:));
points2(2,:) = points2(2,:) - mean(points2(2,:));


% Trial 4 Camera 3
numFrames3 = size(vidFrames3_4,4);  % number of video frames for
    camera 3
pointTracker = vision.PointTracker; % create PointTracker object
points3 = zeros(2, numFrames3); % initialize points3 matrix
I = vidFrames3_4(:,:,:,1); % first video frame for camera 3
%imshow(I); % use data tips on image (under tools tab) to find
    initial point
initial_point = [361 236]; % initial point of flashlight
initialize(pointTracker,initial_point,I) % initialize PointTracker

% find and save points for each time step
for ii = 1:numFrames3
    if ii == 16
        setPoints(pointTracker, [323 206])
    end
    if ii == 57
        setPoints(pointTracker, [335 224])
    end
    if ii == 194
        setPoints(pointTracker, [410 204])
    end
    if ii == 216
        setPoints(pointTracker, [347 262])
    end
    if ii == 231
        setPoints(pointTracker, [421 220])
    end
    if ii == 244
        setPoints(pointTracker, [355 204])
```

```
    end
    if ii == 263
        setPoints(pointTracker, [371 222])
    end
    if ii == 281
        setPoints(pointTracker, [381 240])
    end
    if ii == 301
        setPoints(pointTracker, [374 250])
    end
    if ii == 322
        setPoints(pointTracker, [379 230])
    end
    I = vidFrames3_4(:,:,:,ii);
    points3(:,ii) = step(pointTracker, I);
end

%{
% shows where points are expected to be
for ii = 200:numFrames3
    I = vidFrames3_4(:,:,:,ii);
    imshow(I);
    hold on
    plot(points3(1,ii),points3(2,ii),'r+', 'MarkerSize', 20);
    pause(0.1)
end
%}

% centers points around mean value
points3(1,:) = points3(1,:) - mean(points3(1,:));
points3(2,:) = points3(2,:) - mean(points3(2,:));

% save all points from trial in single matrix
trial4 = zeros(6, 355);
trial4(1:2, :) = points1(:,1:355);
trial4(3:4, :) = points2(:,7:361);
trial4(5:6, :) = points3(:,40:394);

% plots vertical and horizontal displacement of paint can for all
    three
% cameras on same figure
figure(1)
subplot(1,3,1)
ylim([-100 100]); xlim([0 355]);
xlabel('Video Frame'); ylabel('Displacement from mean (pixels)')
```

```
title('Vertical Displacement Trial 4');
hold on
plot(trial4(2,:) , 'b')
plot(trial4(4,:), 'r')
plot(trial4(5,:), 'g')
%legend('Camera 1', 'Camera 2', 'Camera 3');

subplot(1,3,2)
ylim([-100 100]); xlim([0 355]);
xlabel('Video Frame'); ylabel('Displacement from mean (pixels)')
title('Horizontal Displacement Trial 4');
hold on
plot(trial4(1,:) , 'b')
plot(trial4(3,:), 'r')
plot(trial4(6,:), 'g')
legend('Camera 1', 'Camera 2', 'Camera 3');


% Trial 4 Analysis

% singular value decomposition for trial 4
[U4,S4,V4] = svd(trial4, 'econ');


% calculates energy of each rank appromiation
S4_vec = diag(S4).';
S4_vec = S4_vec.^2;
energy4 = S4_vec/sum(S4_vec(1,:));
energy4_sums = cumsum(S4_vec)/sum(S4_vec(1,:));

% log scale of energy covered by each rank approximation
subplot(1,3,3)
plot(1:6,energy4,'ro')
set(gca, 'YScale', 'log')
xlabel('Mode Number'); ylabel('Energy of Mode (log scale)')
title('Mode Energies Trial 4');

% principal components
figure(2)
hold on
X_comp1 = U4(:,1)*S4(1,1)*V4(:,1).';
plot3( X_comp1(1,:),X_comp1(2,:),X_comp1(3,:), 'r.');
X_comp2 = U4(:,2)*S4(2,2)*V4(:,2)';
plot3( X_comp2(1,:),X_comp2(2,:),X_comp2(3,:), 'g.');
X_comp3 = U4(:,3)*S4(3,3)*V4(:,3)';
```

```
plot3( X_comp3(1,:),X_comp3(2,:),X_comp3(3,:), 'g.');
X_rank2 = U4(:,1:2)*S4(1:2,1:2)*V4(:,1:2)';
plot3( X_rank2(1,:),X_rank2(2,:),X_rank2(3,:), 'b.');
xlabel('Horizontal Displacement'); ylabel('Vertical Displacement')
title('PCA Analysis')
legend('1st Principal Component', '2nd Principal Component', '
    Rank 2 Approximation');
```