

# 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 Decomposition (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,  $\Sigma$  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  $\Sigma$  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  $\Sigma$  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.

$$\text{energy}_n = \frac{\sigma_n^2}{\sigma_1^2 + \sigma_2^2 + \dots + \sigma_r^2} \quad (2)$$

In equation 2,  $\sigma$  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.

$$\text{energy}_N = \frac{\sigma_1^2 + \sigma_2^2 + \dots + \sigma_N^2}{\sigma_1^2 + \sigma_2^2 + \dots + \sigma_r^2} \quad (3)$$

In equation 3,  $\sigma$  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

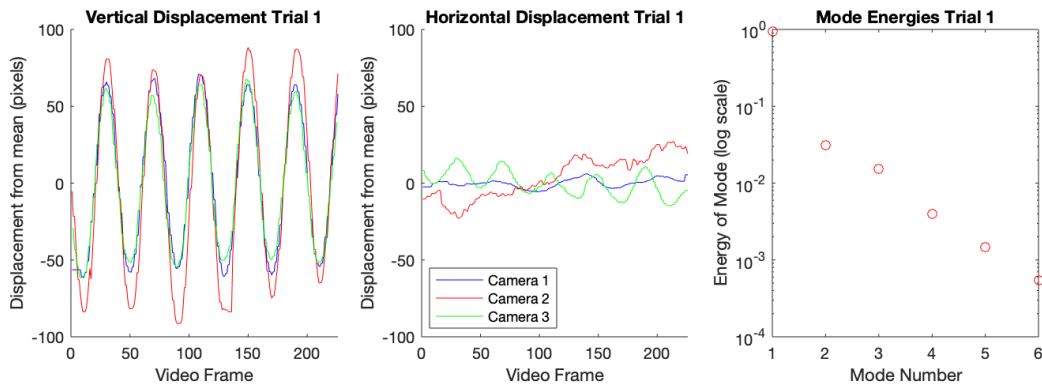
in order to determine the point's subsequent locations. It must first be initialized with a starting image frame and an specific position within that frame (line 28).

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.

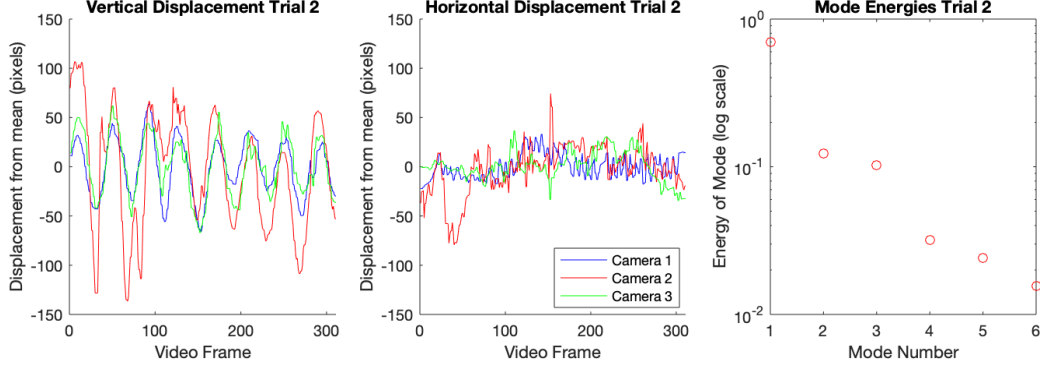
In order to ensure that the data for each trial was being measured on the same scale, we subtracted the mean of each row from that specific row so that all rows were centered around zero (lines 48-49, 91-92, 128-129). Also, by looking at the plots of vertical displacement of the paint can for each camera 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.

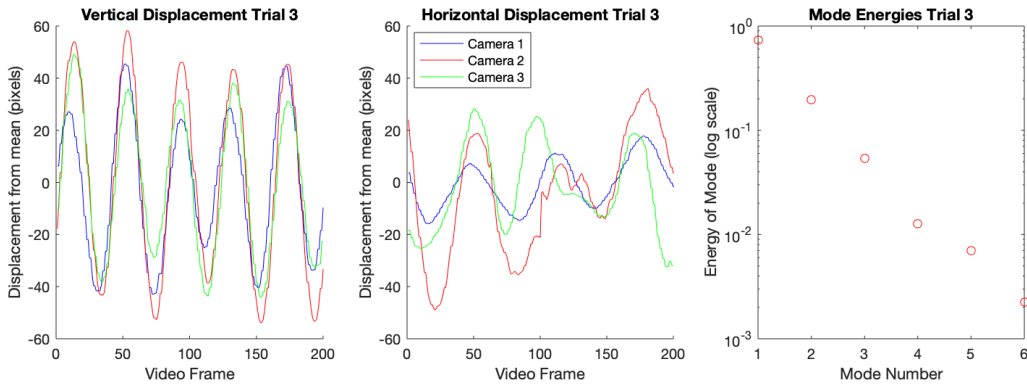


Figures 4, 5, and 6 (from left to right): Trial 2 - noisy case.

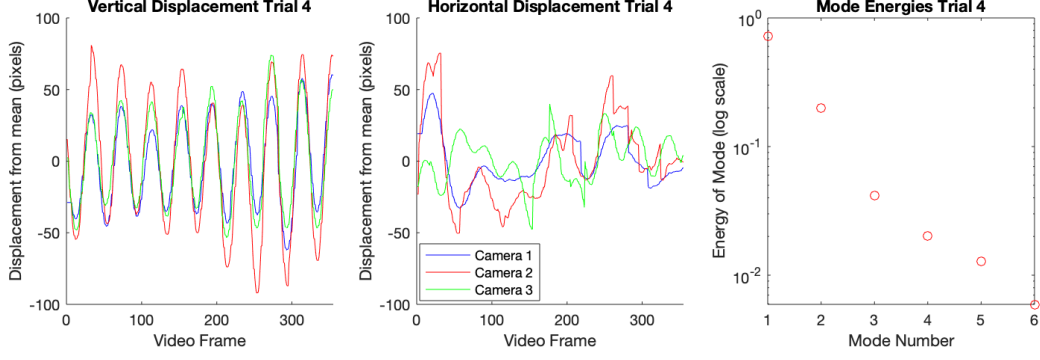
with a log scale on the y axis to account for the large difference in magnitudes of different modes. All 12 figures account for the fact that camera 3 is turned sideways. The horizontal and vertical values gathered with PointTracker are swapped in the plots for this camera.

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  $x - y$



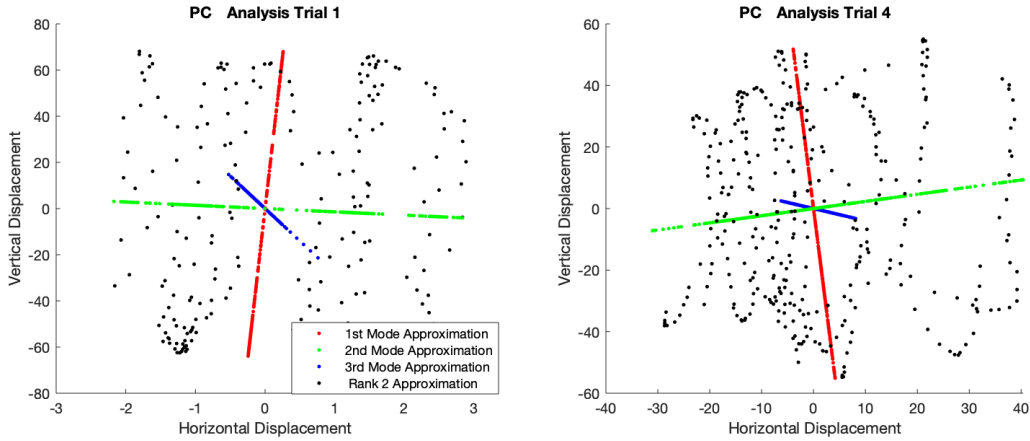
Figures 7, 8, and 9 (from left to right): Trial 3 - intentional horizontal displacement.



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

```
1
2 clear; close all; clc
3
4 % Load video data
5 load('cam1_1.mat')
6 load('cam1_2.mat')
7 load('cam1_3.mat')
8 load('cam1_4.mat')
9 load('cam2_1.mat')
10 load('cam2_2.mat')
11 load('cam2_3.mat')
12 load('cam2_4.mat')
13 load('cam3_1.mat')
14 load('cam3_2.mat')
15 load('cam3_3.mat')
16 load('cam3_4.mat')
17
18 %% Trial 1
19 close all; % closes open figures
20
21 % Trial 1 Camera 1
22 numFrames1 = size(vidFrames1_1,4); % number of video frames for
    camera 1
23 pointTracker = vision.PointTracker; % create PointTracker object
24 points1 = zeros(2, numFrames1); % initialize points1 matrix
25 I = vidFrames1_1(:,:,1); % first video frame for camera 1
26 imshow(I); % use data tips on image (under tools tab) to find
    initial point
27 initial_point = [319 228]; % initial point of flashlight
28 initialize(pointTracker, initial_point, I) % initialize PointTracker
29
30 % find and save points for each time step
31 for ii = 1:numFrames1
32     I = vidFrames1_1(:,:,ii);
33     points1(:,ii) = step(pointTracker, I);
34 end
35
36 %{
37 % shows where points are expected to be
38 for ii = 1:numFrames1
39     I = vidFrames1_1(:,:,ii);
40     imshow(I);
```

```

41     hold on
42     plot(points1(1,ii),points1(2,ii),'r+', 'MarkerSize', 20);
43     pause(0.1)
44 end
45 %}
46
47 % centers points around mean value
48 points1(1,:) = points1(1,:) - mean(points1(1,:));
49 points1(2,:) = points1(2,:) - mean(points1(2,:));
50
51
52 % Trial 1 Camera 2
53 numFrames2 = size(vidFrames2_1,4); % number of video frames for
    camera 2
54 pointTracker = vision.PointTracker; % create PointTracker object
55 points2 = zeros(2, numFrames2); % initialize points2 matrix
56 I = vidFrames2_1(:,:, :,1); % first video frame for camera 2
57 imshow(I); % use data tips on image to find initial point
58 initial_point = [271 278]; % initial point of flashlight
59 initialize(pointTracker,initial_point,I) % initialize PointTracker
60
61 % find and save points for each time step
62 for ii = 1:numFrames2
63     if ii == 20
64         setPoints(pointTracker, [273 110])
65     end
66     if ii == 25
67         setPoints(pointTracker, [277 140])
68     end
69     if ii == 66
70         setPoints(pointTracker, [281 142])
71     end
72     if ii == 186
73         setPoints(pointTracker, [313 158])
74     end
75     I = vidFrames2_1(:,:, :,ii);
76     points2(:,ii) = step(pointTracker, I);
77 end
78
79 %{
80 % shows where points are expected to be
81 for ii = 1:numFrames2
82     I = vidFrames2_1(:,:, :,ii);
83     imshow(I);
84     hold on

```



```

85     plot(points2(1,ii),points2(2,ii),'r+', 'MarkerSize', 20);
86     pause(0.1)
87 end
88 %}
89
90 % centers points around mean value
91 points2(1,:) = points2(1,:) - mean(points2(1,:));
92 points2(2,:) = points2(2,:) - mean(points2(2,:));
93
94
95 % Trial 1 Camera 3
96 numFrames3 = size(vidFrames3_1,4); % number of video frames for
    camera 3
97 pointTracker = vision.PointTracker; % create PointTracker object
98 points3 = zeros(2, numFrames3); % initialize points3 matrix
99 I = vidFrames3_1(:,:, :,1); % first video frame for camera 3
100 imshow(I); % use data tips on image (under tools tab) to find
    initial point
101 initial_point = [317 272]; % initial point of flashlight, got from
    plot
102 initialize(pointTracker,initial_point,I) % initialize PointTracker
103
104 % find and save points at each time step
105 for ii = 1:numFrames3
106     if ii == 97
107         setPoints(pointTracker, [320 258])
108     end
109     if ii == 139
110         setPoints(pointTracker, [341 262])
111     end
112     I = vidFrames3_1(:,:, :,ii);
113     points3(:,ii) = step(pointTracker, I);
114 end
115
116 %{
117 % shows where points are expected to be
118 for ii = 1:numFrames3
119     I = vidFrames3_1(:,:, :,ii);
120     imshow(I);
121     hold on
122     plot(points3(1,ii),points3(2,ii),'r+', 'MarkerSize', 20);
123     pause(0.1)
124 end
125 %}
126

```

```

127 % centers points around mean value
128 points3(1,:) = points3(1,:) - mean(points3(1,:));
129 points3(2,:) = points3(2,:) - mean(points3(2,:));
130
131 % save all points from trial in single matrix
132 trial1 = zeros(6, 226);
133 trial1(1:2, :) = points1(:,1:226);
134 trial1(3:4, :) = points2(:,50:275);
135 trial1(5:6, :) = points3(:,1:226);
136
137 % plots vertical and horizontal displacement of paint can for all
    three
138 % cameras on same figure
139 figure(1)
140 subplot(1,3,1)
141 ylim([-100 100]); xlim([0 226]);
142 xlabel('Video Frame'); ylabel('Displacement from mean (pixels)')
143 title('Vertical Displacement Trial 1');
144 hold on
145 plot(trial1(2,:), 'b')
146 plot(trial1(4,:), 'r')
147 plot(trial1(5,:), 'g')
148 %legend('Camera 1', 'Camera 2', 'Camera 3');
149
150 subplot(1,3,2)
151 ylim([-100 100]); xlim([0 226]);
152 xlabel('Video Frame'); ylabel('Displacement from mean (pixels)')
153 title('Horizontal Displacement Trial 1');
154 hold on
155 plot(trial1(1,:), 'b')
156 plot(trial1(3,:), 'r')
157 plot(trial1(6,:), 'g')
158 legend('Camera 1', 'Camera 2', 'Camera 3');
159
160
161 % Trial 1 Analysis
162
163 % singular value decomposition for trial 1
164 [U1,S1,V1] = svd(trial1, 'econ');
165
166
167 % calculates energy of each rank appromiation
168 S1_vec = diag(S1).';
169 S1_vec = S1_vec.^2;
170 energy1 = S1_vec/sum(S1_vec(1,:));

```

```

171 energy1_sums = cumsum(S1_vec)/sum(S1_vec(1,:));
172
173 % log scale of energy covered by each rank approximation
174 subplot(1,3,3)
175 plot(1:6,energy1,'ro')
176 set(gca,'YScale','log')
177 xlabel('Mode Number'); ylabel('Energy of Mode (log scale)')
178 title('Mode Energies Trial 1');
179
180 % principal components
181 figure(2)
182 hold on
183 X_comp1 = U1(:,1)*S1(1,1)*V1(:,1)';
184 plot3(X_comp1(1,:),X_comp1(2,:),X_comp1(3,:), 'r. ');
185 X_comp2 = U1(:,2)*S1(2,2)*V1(:,2)';
186 plot3(X_comp2(1,:),X_comp2(2,:),X_comp2(3,:), 'g. ');
187 X_comp3 = U1(:,3)*S1(3,3)*V1(:,3)';
188 plot3(X_comp3(1,:),X_comp3(2,:),X_comp3(3,:), 'g. ');
189 X_rank2 = U1(:,1:2)*S1(1:2,1:2)*V1(:,1:2)';
190 plot3(X_rank2(1,:),X_rank2(2,:),X_rank2(3,:), 'b. ');
191 xlabel('Horizontal Displacement'); ylabel('Vertical Displacement')
192 title('PCA Analysis')
193 legend('1st Principal Component', '2nd Principal Component', '
    Rank 2 Approximation');
194
195 %% Trial 2
196 close all; % closes all figures
197
198 % Trial 2 Camera 1
199 numFrames1 = size(vidFrames1_2,4); % number of video frames for
    camera 1
200 pointTracker = vision.PointTracker; % create PointTracker object
201 points1 = zeros(2, numFrames1); % initialize points1 matrix
202 I = vidFrames1_2(:,:,1); % first video frame for camera 1
203 imshow(I); % use data tips on image (under tools tab) to find
    initial point
204 initial_point = [325 310]; % initial point of flashlight
205 initialize(pointTracker,initial_point,I) % initialize PointTracker
206
207 % find and save points for each time step
208 for ii = 1:numFrames1
209     I = vidFrames1_2(:,:,ii);
210     points1(:,ii) = step(pointTracker, I);
211 end
212

```

```

213 %{
214 % shows where points are expected to be
215 for ii = 1:numFrames1
216     I = vidFrames1_2(:, :, :, ii);
217     imshow(I);
218     hold on
219     plot(points1(1, ii), points1(2, ii), 'r+', 'MarkerSize', 20);
220     pause(0.1)
221 end
222 %}
223
224 % centers points around mean value
225 points1(1, :) = points1(1, :) - mean(points1(1, :));
226 points1(2, :) = points1(2, :) - mean(points1(2, :));
227
228
229 % Trial 2 Camera 2
230 numFrames2 = size(vidFrames2_2, 4); % number of video frames for
    camera 2
231 pointTracker = vision.PointTracker; % create PointTracker object
232 points2 = zeros(2, numFrames2); % initialize points2 matrix
233 I = vidFrames2_2(:, :, :, 1); % first video frame for camera 2
234 imshow(I); % use data tips on image (under tools tab) to find
    initial point
235 initial_point = [315 362]; % initial point of flashlight
236 initialize(pointTracker, initial_point, I) % initialize PointTracker
237
238 % find and save points for each time step
239 for ii = 1:numFrames2
240     if ii == 4
241         setPoints(pointTracker, [301 334])
242     end
243     if ii == 59
244         setPoints(pointTracker, [257 82])
245     end
246     if ii == 63
247         setPoints(pointTracker, [231 98])
248     end
249     if ii == 249
250         setPoints(pointTracker, [315 186])
251     end
252     if ii == 291
253         setPoints(pointTracker, [331 136])
254     end
255     if ii == 311

```

```

256         setPoints(pointTracker , [309 208])
257     end
258     if ii == 350
259         setPoints(pointTracker , [391 278])
260     end
261     I = vidFrames2_2(:, :, :, ii);
262     points2(:, ii) = step(pointTracker , I);
263 end
264
265
266 %{
267 % shows where points are expected to be
268 for ii = 1:numFrames2
269     I = vidFrames2_2(:, :, :, ii);
270     imshow(I);
271     hold on
272     plot(points2(1, ii), points2(2, ii), 'r+', 'MarkerSize', 20);
273     pause(0.1)
274 end
275 %}
276
277 % centers points around mean value
278 points2(1,:) = points2(1,:) - mean(points2(1,:));
279 points2(2,:) = points2(2,:) - mean(points2(2,:));
280
281
282 % Trial 2 Camera 3
283 numFrames3 = size(vidFrames3_2,4); % number of video frames for
    camera 3
284 pointTracker = vision.PointTracker; % create PointTracker object
285 points3 = zeros(2, numFrames3); % initialize points3 matrix
286 I = vidFrames3_2(:, :, :, 1); % first video frame for camera 3
287 imshow(I); % use data tips on image (under tools tab) to find
    initial point
288 initial_point = [345 246]; % initial point of flashlight
289 initialize(pointTracker, initial_point, I) % initialize PointTracker
290
291 % find and save points for each time step
292 for ii = 1:numFrames3
293     if ii == 222
294         setPoints(pointTracker , [367 270])
295     end
296     if ii == 247
297         setPoints(pointTracker , [335 260])
298     end

```

```

299     I = vidFrames3_2(:, :, :, ii);
300     points3(:, ii) = step(pointTracker, I);
301 end
302
303 %{
304 % shows where points are expected to be
305 for ii = 1:numFrames3
306     I = vidFrames3_2(:, :, :, ii);
307     imshow(I);
308     hold on
309     plot(points3(1, ii), points3(2, ii), 'r+', 'MarkerSize', 20);
310     pause(0.1)
311 end
312 %}
313
314 % centers points around mean value
315 points3(1, :) = points3(1, :) - mean(points3(1, :));
316 points3(2, :) = points3(2, :) - mean(points3(2, :));
317
318 % save all points from trial in single matrix
319 trial2 = zeros(6, 311);
320 trial2(1:2, :) = points1(:, 4:314);
321 trial2(3:4, :) = points2(:, 30:340);
322 trial2(5:6, :) = points3(:, 7:317);
323
324 % plots vertical and horizontal displacement of paint can for all
    three
325 % cameras on same figure
326 figure(1)
327 subplot(1, 3, 1)
328 ylim([-150 150]); xlim([0 311]);
329 xlabel('Video Frame'); ylabel('Displacement from mean (pixels)')
330 title('Vertical Displacement Trial 2');
331 hold on
332 plot(trial2(2, :), 'b')
333 plot(trial2(4, :), 'r')
334 plot(trial2(5, :), 'g')
335 %legend('Camera 1', 'Camera 2', 'Camera 3');
336
337 subplot(1, 3, 2)
338 ylim([-150 150]); xlim([0 311]);
339 xlabel('Video Frame'); ylabel('Displacement from mean (pixels)')
340 title('Horizontal Displacement Trial 2');
341 hold on
342 plot(trial2(1, :), 'b')

```

```

343 plot(trial2(3,:), 'r')
344 plot(trial2(6,:), 'g')
345 legend('Camera 1', 'Camera 2', 'Camera 3');
346
347
348 % Trial 2 Analysis
349
350 % singular value decomposition for trial 2
351 [U2,S2,V2] = svd(trial2, 'econ');
352
353
354 % calculates energy of each rank approximation
355 S2_vec = diag(S2).';
356 S2_vec = S2_vec.^2;
357 energy2 = S2_vec/sum(S2_vec(1,:));
358 energy2_sums = cumsum(S2_vec)/sum(S2_vec(1,:));
359
360 % log scale of energy covered by each rank approximation
361 subplot(1,3,3)
362 plot(1:6,energy2, 'ro')
363 set(gca, 'YScale', 'log')
364 xlabel('Mode Number'); ylabel('Energy of Mode (log scale)')
365 title('Mode Energies Trial 2');
366
367 % Modes plotted
368 figure(3)
369 plot(1:311,V2(:,1), 'b', 1:311,V2(:,2), '—r', 1:311,V2(:,3), ':k', '
    Linewidth',2)
370
371 % principal components
372 figure(2)
373 hold on
374 X_comp1 = U2(:,1)*S2(1,1)*V2(:,1).';
375 plot3(X_comp1(1,:), X_comp1(2,:), X_comp1(3,:), 'r. ');
376 X_comp2 = U2(:,2)*S2(2,2)*V2(:,2).';
377 plot3(X_comp2(1,:), X_comp2(2,:), X_comp2(3,:), 'g. ');
378 X_rank2 = U2(:,1:2)*S2(1:2,1:2)*V2(:,1:2).';
379 plot3(X_rank2(1,:), X_rank2(2,:), X_rank2(3,:), 'b. ');
380 xlabel('Horizontal Displacement'); ylabel('Vertical Displacement')
381 title('PCA Analysis')
382 legend('1st Principal Component', '2nd Principal Component', '
    Rank 2 Approximation');
383
384 %% Trial 3
385 close all; % closes open figures

```

```

386
387 % Trial 3 Camera 1
388 numFrames1 = size(vidFrames1_3,4); % number of video frames for
    camera 1
389 pointTracker = vision.PointTracker; % create PointTracker object
390 point1 = zeros(2, numFrames1); % initialize points1 matrix
391 I = vidFrames1_3(:,:,1); % first video frame for camera 1
392 imshow(I); % use data tips on image (under tools tab) to find
    initial point
393 initial_point = [317 288]; % initial point of flashlight
394 initialize(pointTracker, initial_point, I) % initialize PointTracker
395
396 % find and save points for each time step
397 for ii = 1:numFrames1
398     I = vidFrames1_3(:,:,ii);
399     points1(:,ii) = step(pointTracker, I);
400 end
401
402 %{
403 % shows where points are expected to be
404 for ii = 1:numFrames1
405     I = vidFrames1_3(:,:,ii);
406     imshow(I);
407     hold on
408     plot(points1(1,ii), points1(2,ii), 'r+', 'MarkerSize', 20);
409     pause(0.1)
410 end
411 %}
412
413 % centers points around mean value
414 points1(1,:) = points1(1,:) - mean(points1(1,:));
415 points1(2,:) = points1(2,:) - mean(points1(2,:));
416
417
418 % Trial 3 Camera 2
419 numFrames2 = size(vidFrames2_3,4); % number of video frames for
    camera 2
420 pointTracker = vision.PointTracker; % create PointTracker object
421 points2 = zeros(2, numFrames2); % initialize points2 matrix
422 I = vidFrames2_3(:,:,1); % first video frame for camera 2
423 imshow(I); % use data tips on image (under tools tab) to find
    initial point
424 initial_point = [249 294]; % initial point of flashlight
425 initialize(pointTracker, initial_point, I) % initialize PointTracker
426

```



```

427 % find and save points for each time step
428 for ii = 1:numFrames2
429     if ii == 10
430         setPoints(pointTracker , [299 298])
431     end
432     if ii == 96
433         setPoints(pointTracker , [307 250])
434     end
435     if ii == 133
436         setPoints(pointTracker , [307 274])
437     end
438     if ii == 260
439         setPoints(pointTracker , [323 232])
440     end
441     I = vidFrames2_3(:, :, :, ii);
442     points2(:, ii) = step(pointTracker , I);
443 end
444
445 %{
446 % shows where points are expected to be
447 for ii = 1:numFrames2
448     I = vidFrames2_3(:, :, :, ii);
449     imshow(I);
450     hold on
451     plot(points2(1, ii), points2(2, ii), 'r+', 'MarkerSize', 20);
452     pause(0.1)
453 end
454 %}
455
456 % centers points around mean value
457 points2(1,:) = points2(1,:) - mean(points2(1,:));
458 points2(2,:) = points2(2,:) - mean(points2(2,:));
459
460
461 % Trial 3 Camera 3
462 numFrames3 = size(vidFrames3_3,4); % number of video frames for
    camera 3
463 pointTracker = vision.PointTracker; % create PointTracker object
464 points3 = zeros(2, numFrames3); % initialize points3 matrix
465 I = vidFrames3_3(:, :, :, 1); % first video frame for camera 3
466 %imshow(I); % use data tips on image (under tools tab) to find
    initial point
467 initial_point = [351 230]; % initial point of flashlight
468 initialize(pointTracker, initial_point, I) % initialize PointTracker
469

```

```

470 % find and save points for each time step
471 for ii = 1:numFrames3
472     I = vidFrames3_3(:,:, :, ii);
473     points3(:, ii) = step(pointTracker, I);
474 end
475
476 %{
477 % shows where points are expected to be
478 for ii = 1:numFrames3
479     I = vidFrames3_3(:,:, :, ii);
480     imshow(I);
481     hold on
482     plot(points3(1, ii), points3(2, ii), 'r+', 'MarkerSize', 20);
483     pause(0.1)
484 end
485 %}
486
487 % centers points around mean value
488 points3(1,:) = points3(1,:) - mean(points3(1,:));
489 points3(2,:) = points3(2,:) - mean(points3(2,:));
490
491 % save all points from trial in single matrix
492 trial3 = zeros(6, 200);
493 trial3(1:2, :) = points1(:, 8:207);
494 trial3(3:4, :) = points2(:, 33:232);
495 trial3(5:6, :) = points3(:, 38:237);
496
497 % plots vertical and horizontal displacement of paint can for all
    three
498 % cameras on same figure
499 figure(1)
500 subplot(1,3,1)
501 ylim([-60 60]); xlim([0 200]);
502 xlabel('Video Frame'); ylabel('Displacement from mean (pixels)')
503 title('Vertical Displacement Trial 3');
504 hold on
505 plot(trial3(2,:), 'b')
506 plot(trial3(4,:), 'r')
507 plot(trial3(5,:), 'g')
508 %legend('Camera 1', 'Camera 2', 'Camera 3');
509
510 subplot(1,3,2)
511 ylim([-60 60]); xlim([0 200]);
512 xlabel('Video Frame'); ylabel('Displacement from mean (pixels)')
513 title('Horizontal Displacement Trial 3');

```

```

514 hold on
515 plot(trial3(1,:), 'b')
516 plot(trial3(3,:), 'r')
517 plot(trial3(6,:), 'g')
518 legend('Camera 1', 'Camera 2', 'Camera 3');
519
520
521 % Trial 3 Analysis
522
523 % singular value decomposition for trial 3
524 [U3,S3,V3] = svd(trial3, 'econ');
525
526 % calculates energy of each rank appromiation
527 S3_vec = diag(S3).';
528 S3_vec = S3_vec.^2;
529 energy3 = S3_vec/sum(S3_vec(1,:));
530 energy3_sums = cumsum(S3_vec)/sum(S3_vec(1,:));
531
532 % log scale of energy covered by each rank approximation
533 subplot(1,3,3)
534 plot(1:6,energy3,'ro')
535 set(gca, 'YScale', 'log')
536 xlabel('Mode Number'); ylabel('Energy of Mode (log scale)')
537 title('Mode Energies Trial 3');
538
539 % principal components
540 figure(2)
541 hold on
542 X_comp1 = U3(:,1)*S3(1,1)*V3(:,1).';
543 plot3(X_comp1(1,:),X_comp1(2,:),X_comp1(3,:), 'r. ');
544 X_comp2 = U3(:,2)*S3(2,2)*V3(:,2).';
545 plot3(X_comp2(1,:),X_comp2(2,:),X_comp2(3,:), 'g. ');
546 X_rank2 = U3(:,1:2)*S3(1:2,1:2)*V3(:,1:2).';
547 plot3(X_rank2(1,:),X_rank2(2,:),X_rank2(3,:), 'b. ');
548 xlabel('Horizontal Displacement'); ylabel('Vertical Displacement')
549 title('PCA Analysis')
550 legend('1st Principal Component', '2nd Principal Component', '
Rank 2 Approximation');
551
552 %% Trial 4
553 close all; % closes open figures
554
555 % Trial 4 Camera 1
556 numFrames1 = size(vidFrames1_4,4); % number of video frames for
camera 1

```

```

557 pointTracker = vision.PointTracker; % create PointTracker object
558 points1 = zeros(2, numFrames1); % initialize points1 matrix
559 I = vidFrames1_4(:,:, :, 1); % first video frame for camera 1
560 imshow(I); % use data tips on image (under tools tab) to find
    initial point
561 initial_point = [400 264]; % initial point of flashlight
562 initialize(pointTracker, initial_point, I) % initialize PointTracker
563
564 % find and save points for each time step
565 for ii = 1:numFrames1
566     if ii == 219
567         setPoints(pointTracker, [374 266])
568     end
569     if ii == 282
570         setPoints(pointTracker, [387 294])
571     end
572     if ii == 308
573         setPoints(pointTracker, [364 324])
574     end
575     I = vidFrames1_4(:,:, :, ii);
576     points1(:, ii) = step(pointTracker, I);
577 end
578
579 %{
580 % shows where points are expected to be
581 for ii = 1:numFrames1
582     I = vidFrames1_4(:,:, :, ii);
583     imshow(I);
584     hold on
585     plot(points1(1, ii), points1(2, ii), 'r+', 'MarkerSize', 20);
586     pause(0.1)
587 end
588 %}
589
590 % centers points around mean value
591 points1(1,:) = points1(1,:) - mean(points1(1,:));
592 points1(2,:) = points1(2,:) - mean(points1(2,:));
593
594
595 % Trial 4 Camera 2
596 numFrames2 = size(vidFrames2_4, 4); % number of video frames for
    camera 2
597 pointTracker = vision.PointTracker; % create PointTracker object
598 points2 = zeros(2, numFrames2); % initialize points2 matrix
599 I = vidFrames2_4(:,:, :, 1); % first video frame for camera 2

```

```

600 imshow(I); % use data tips on image (under tools tab) to find
        initial_point
601 initial_point = [245 246]; % initial point of flashlight
602 initialize(pointTracker, initial_point, I) % initialize PointTracker
603
604 % find and save points for each time step
605 for ii = 1:numFrames2
606     if ii == 6
607         setPoints(pointTracker, [277 214])
608     end
609     if ii == 9
610         setPoints(pointTracker, [301 196])
611     end
612     if ii == 10
613         setPoints(pointTracker, [305 182])
614     end
615     if ii == 39
616         setPoints(pointTracker, [337 266])
617     end
618     if ii == 64
619         setPoints(pointTracker, [267 144])
620     end
621     if ii == 214
622         setPoints(pointTracker, [305 144])
623     end
624     if ii == 268
625         setPoints(pointTracker, [329 158])
626     end
627     if ii == 288
628         setPoints(pointTracker, [307 200])
629     end
630     if ii == 293
631         setPoints(pointTracker, [313 148])
632     end
633     if ii == 302
634         setPoints(pointTracker, [291 114])
635     end
636     if ii == 331
637         setPoints(pointTracker, [285 178])
638     end
639     I = vidFrames2_4(:, :, :, ii);
640     points2(:, ii) = step(pointTracker, I);
641 end
642
643 %{

```

```

644 % shows where points are expected to be
645 for ii = 290:numFrames2
646     I = vidFrames2_4(:, :, :, ii);
647     imshow(I);
648     hold on
649     plot(points2(1, ii), points2(2, ii), 'r+', 'MarkerSize', 20);
650     pause(0.1)
651 end
652 %}
653
654 % centers points around mean value
655 points2(1, :) = points2(1, :) - mean(points2(1, :));
656 points2(2, :) = points2(2, :) - mean(points2(2, :));
657
658
659 % Trial 4 Camera 3
660 numFrames3 = size(vidFrames3_4, 4); % number of video frames for
    camera 3
661 pointTracker = vision.PointTracker; % create PointTracker object
662 points3 = zeros(2, numFrames3); % initialize points3 matrix
663 I = vidFrames3_4(:, :, :, 1); % first video frame for camera 3
664 %imshow(I); % use data tips on image (under tools tab) to find
    initial point
665 initial_point = [361 236]; % initial point of flashlight
666 initialize(pointTracker, initial_point, I) % initialize PointTracker
667
668 % find and save points for each time step
669 for ii = 1:numFrames3
670     if ii == 16
671         setPoints(pointTracker, [323 206])
672     end
673     if ii == 57
674         setPoints(pointTracker, [335 224])
675     end
676     if ii == 194
677         setPoints(pointTracker, [410 204])
678     end
679     if ii == 216
680         setPoints(pointTracker, [347 262])
681     end
682     if ii == 231
683         setPoints(pointTracker, [421 220])
684     end
685     if ii == 244
686         setPoints(pointTracker, [355 204])

```

```

687     end
688     if ii == 263
689         setPoints(pointTracker , [371 222])
690     end
691     if ii == 281
692         setPoints(pointTracker , [381 240])
693     end
694     if ii == 301
695         setPoints(pointTracker , [374 250])
696     end
697     if ii == 322
698         setPoints(pointTracker , [379 230])
699     end
700     I = vidFrames3_4(:, :, :, ii);
701     points3(:, ii) = step(pointTracker , I);
702 end
703
704 %{
705 % shows where points are expected to be
706 for ii = 200:numFrames3
707     I = vidFrames3_4(:, :, :, ii);
708     imshow(I);
709     hold on
710     plot(points3(1, ii), points3(2, ii), 'r+', 'MarkerSize', 20);
711     pause(0.1)
712 end
713 %}
714
715 % centers points around mean value
716 points3(1,:) = points3(1,:) - mean(points3(1,:));
717 points3(2,:) = points3(2,:) - mean(points3(2,:));
718
719 % save all points from trial in single matrix
720 trial4 = zeros(6, 355);
721 trial4(1:2, :) = points1(:, 1:355);
722 trial4(3:4, :) = points2(:, 7:361);
723 trial4(5:6, :) = points3(:, 40:394);
724
725 % plots vertical and horizontal displacement of paint can for all
    three
726 % cameras on same figure
727 figure(1)
728 subplot(1,3,1)
729 ylim([-100 100]); xlim([0 355]);
730 xlabel('Video Frame'); ylabel('Displacement from mean (pixels)')

```

```

731 title('Vertical Displacement Trial 4');
732 hold on
733 plot(trial4(2,:), 'b')
734 plot(trial4(4,:), 'r')
735 plot(trial4(5,:), 'g')
736 %legend('Camera 1', 'Camera 2', 'Camera 3');
737
738 subplot(1,3,2)
739 ylim([-100 100]); xlim([0 355]);
740 xlabel('Video Frame'); ylabel('Displacement from mean (pixels)')
741 title('Horizontal Displacement Trial 4');
742 hold on
743 plot(trial4(1,:), 'b')
744 plot(trial4(3,:), 'r')
745 plot(trial4(6,:), 'g')
746 legend('Camera 1', 'Camera 2', 'Camera 3');
747
748
749 % Trial 4 Analysis
750
751 % singular value decomposition for trial 4
752 [U4,S4,V4] = svd(trial4, 'econ');
753
754
755 % calculates energy of each rank appromiation
756 S4_vec = diag(S4).';
757 S4_vec = S4_vec.^2;
758 energy4 = S4_vec/sum(S4_vec(1,:));
759 energy4_sums = cumsum(S4_vec)/sum(S4_vec(1,:));
760
761 % log scale of energy covered by each rank approximation
762 subplot(1,3,3)
763 plot(1:6,energy4, 'ro')
764 set(gca, 'YScale', 'log')
765 xlabel('Mode Number'); ylabel('Energy of Mode (log scale)')
766 title('Mode Energies Trial 4');
767
768 % principal components
769 figure(2)
770 hold on
771 X_comp1 = U4(:,1)*S4(1,1)*V4(:,1).';
772 plot3(X_comp1(1,:), X_comp1(2,:), X_comp1(3,:), 'r. ');
773 X_comp2 = U4(:,2)*S4(2,2)*V4(:,2).';
774 plot3(X_comp2(1,:), X_comp2(2,:), X_comp2(3,:), 'g. ');
775 X_comp3 = U4(:,3)*S4(3,3)*V4(:,3).';

```



```

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

```