

# Amath 482 homework 3: Principal Component Analysis

Andy Zhang

February 20, 2020

## Abstract

This report demonstrates the application of Principal Component Analysis (PCA) on the Mass-Spring system videos, filmed by cameras from several angles in different cases, so that we can determine if redundancies exist and still capture the whole dynamic after dimension reduction. This can be done by applying Singular Value Decomposition (SVD) to the snapshot matrix of data.

## 1 Introduction

PCA is an useful technique to extract low-dimension data from the dynamic, but not having to know the governing rules. Thus, it's a practical tool to deal with large amounts of real-world data. By performing PCA, we are able to extract the features and focus of the data and deal with them efficiently.

Specifically in this report, we are going to check the feasibility of dimension reduction of the data from a Spring-Mass system, filmed by 3 cameras from different angles. Besides, 4 different situations are introduced: the ideal case, that the mass generally moves in z-direction; the noisy case that adds camera shakes to the ideal case; the horizontal displacement case that generates both the harmonic motion in z-direction and pendulum motion in x-y plane; and the last case, that adds rotation of the mass to the horizontal displacement case.

## 2 Theoretical Background

### 2.1 Singular Value Decomposition (Full SVD)

SVD is an useful tool when performing dimension reduction, by decomposing any matrix into 2 orthogonal/unitary basis and 1 stretching/compressing matrix. It takes the form of:

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

where the  $*$  denotes the (conjugate) transpose of the basis  $V$ .

For a  $m \times n$  size matrix  $A$ , both basis matrices  $U$  and  $V$  are unitary/orthogonal, with a size of  $m \times m$  and  $n \times n$  respectively.  $\Sigma$  is a  $m \times n$  diagonal matrix with non-negative diagonal entries known as singular values, sorted from the largest to the smallest. Basically we can understand it as a basis being stretched/compressed and then being applied(rotated) due to another basis.

## 2.2 Principal Component Analysis (PCA)

As described in **Introduction**, PCA is often applied for dimension reduction of data, which knowing the governing rules is unnecessary. First, we rearrange the  $x$  and  $y$  coordinates of the mass generated from each video into a matrix:

$$X = \begin{bmatrix} x_a \\ y_a \\ x_b \\ y_b \\ x_c \\ y_c \end{bmatrix} \quad (2)$$

where the number of columns denote the numbers of data points collected over time (in this case, the number of frames), and the number of rows represent the times of measurements.

We can further construct the covariance matrix  $C_X$  showing the variances between every possible pair of data by:

$$C_X = \frac{1}{n-1} X X^T \quad (3)$$

where  $\frac{1}{n-1}$  is for normalization. The covariance matrix  $C_X$  is a square and symmetric matrix whose diagonal entries represent the variances of particular measurements and the off-diagonals denote the covariances between different measurements. Larger off-diagonals entries imply redundancy in our data.

In order to get an ideal basis that all redundancies have been removed and the largest variances of particular measurements are ordered, the idea of SVD is applied. We first do SVD to  $X$  and create a new basis:

$$Y = U * X \quad (4)$$

that represent the principal components projection. And for the variance of  $Y$ :

$$C_Y = \frac{1}{n-1} Y Y^T = \frac{1}{n-1} \Sigma^2 \quad (5)$$

Thus, an ideal basis is constructed with only diagonal entries in the covariance matrix, that is, all redundancies between each data pairs are removed.

### 3 Algorithm Implementation and Development

#### Prepare data

To start with each case, I load the mat files of videos from three angles. Then I will call  $[X(N), Y(N), \sim, frameNumN(N)] = size(vidFramesN_N)$  to find the sizes of videos, which are all 480 \* 640, and the corresponding number of frames. We insert a  $\sim$  at  $Z$  values because it always generates 3 due to RGB scale of the videos. Since we will transfer the video to gray scale later, we don't need those  $Z$  values. Then I watch each video and determine the approximate range of the paint can's movement.

#### Trace the flashlight in each frame

For each frame of each video, I convert it from RGB to gray scale and the pixels to double precisions, which the largest value 255 represents white and smallest value 0 means black. Since the flashlight is the brightest point most of the time, I will trace it to get the position of the paint can in each frame. By observing each video and applying *pcolor*, we can get the movement interval of the paint can and then set what's outside of that interval to black (0 value), because the whiteboard and reflect light in the background will hinder us from locating the flashlight precisely.

Then, I will get the max value among all pixels in each frame and find a vector of coordinates that are larger than certain **threshold value** multiplies the max value. Averaging this vector gives a much more accurate x and y coordinate for each frame, since it can also average out outliers. Note that the x and y values from camera 3 have to be swapped because it's filming in a reverse angle.

#### Organize the data

Since each video has different numbers of frames, I first rearrange each vector of coordinates so that the can starts from the highest point. Then I find the minimum length of vector among all data and cut off extras that exceed this length. This is an appropriate approach because the mass is in harmonic motion (and pendulum motion), and getting rid of extra data won't affect the pattern. I then plot the horizontal and vertical positions of the mass against each frame to test for different **threshold values** until the plot shows harmonic motions of the mass in all three angles. After arranging the data to a snapshot matrix  $\mathbf{Mx}$ , I subtract mean from each row of data for normalization.

#### Perform PCA on snapshot matrix

I then perform SVD on the snapshot matrix  $\mathbf{Mx}$  divided by  $\sqrt{n-1}$  for normalization, with matlab build-in command  $[U, S, V] = svd(Mx, 'econ')$ , and

construct the principal component matrix  $Y$  by multiplying  $U$  with  $\mathbf{M}\mathbf{x}$ . Following then I plot  $Y$  against each frame so that I can get the normalized position controlled by each mode and check the feasibility of dimension reduction. The energy carried by each mode (totally 6) can also be found by dividing the corresponding singular value squared from the sum of all singular values squared.

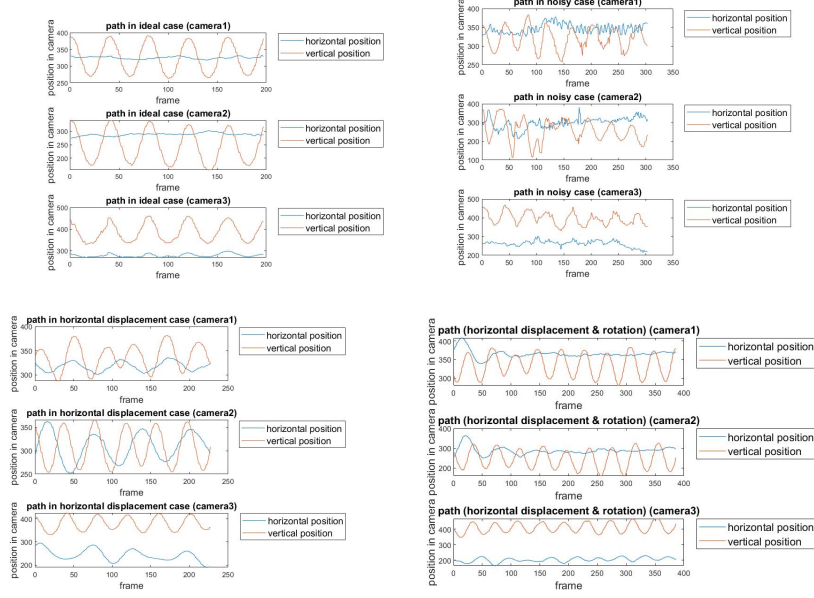


Figure 1: The actual horizontal and vertical positions of the paint can in all videos

## 4 Computational Results

### The horizontal and vertical positions of the paint can from all videos

In **Figure 1**, the actual positions of all cases from all angles are shown. We can see that the mass is in vertical harmonic motion in all cases (except in noisy case) and in horizontal motions in case 3 and 4.

#### Results from ideal case

In the ideal case shown in **Figure 2**, we can clearly see that the first mode is strongly dominant, which takes more than 95% of the whole energy of the data. Thus, it's feasible and necessary for dimension reduction since we can

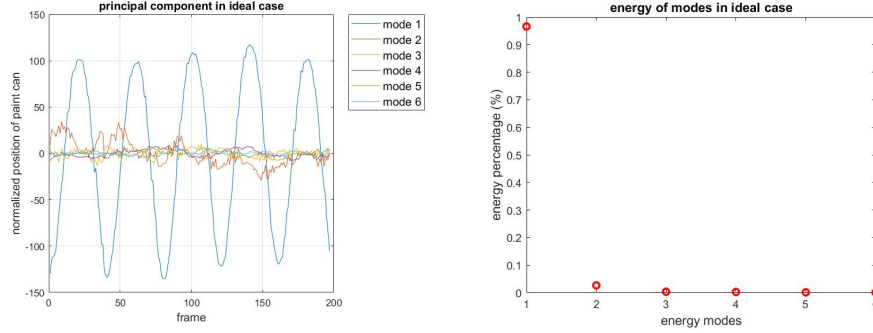


Figure 2: PCA and mode energy in ideal case

also read from the left image that in general, the can is moving only in one direction(z-direction).

### Results from noisy case

Shown in **Figure 3**, the dominant mode only takes about 55% of the whole energy due to camera shakes, and four modes are needed to capture 95% of the whole dynamics. Thus, dimension reduction is not necessary because even the last two modes capture about 5% of the dynamics. In this case, PCA is not performing well as we can read from the left image of position.

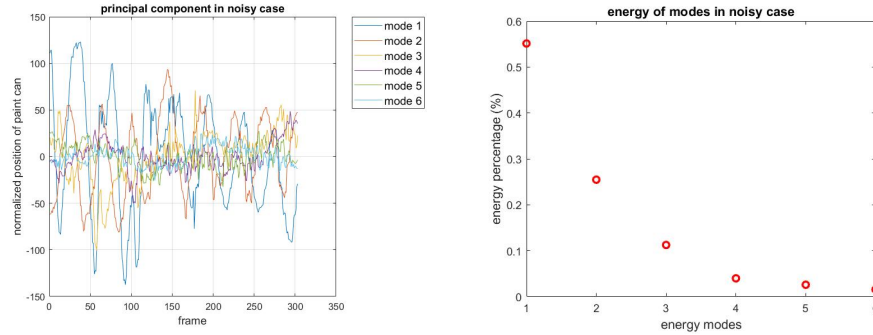


Figure 3: PCA and mode energy in noisy case

### Results from horizontal displacement case

In the horizontal displacement case, the paint can is not only moving in z-direction, but also in x-y plane. Thus, data from 2 different angles are necessary to capture both motions and what's shown in **Figure 4**, that 4 modes take approximately all energy in the dynamics makes sense. We can also see from

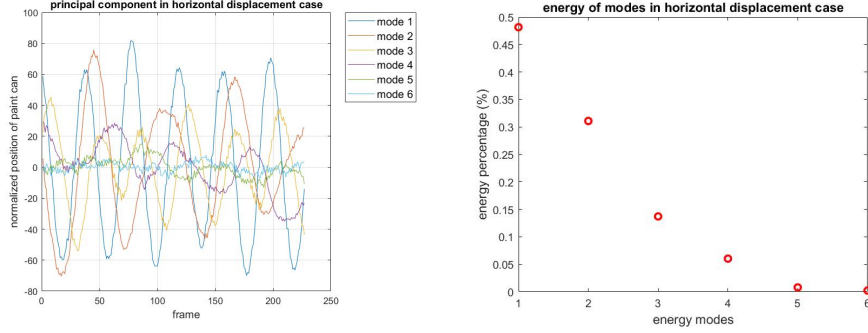


Figure 4: PCA and mode energy in horizontal displacement case

the left image that mode 5 and 6 result in small variations in positions. So we can do dimension reduction in this case so that we capture the whole dynamic by using only two cameras.

## Results from horizontal displacement and rotation case

The last case is the horizontal displacement and rotation of the paint can. However, the can is in pendulum motion only in the first few frames (as shown in **Figure 1**). In the rest of the videos, the can is just in simple vertical harmonic motions. What the results shown in **Figure 5**, is that the first dominant mode takes 65% of the energy and only three modes are needed to represent the whole system. However, we are failing to capture the rotation of can due to fixed angles (linearity). So with that being said, we can do dimension reduction in this case but can only represent the pendulum and harmonic motions by applying a half of the data.

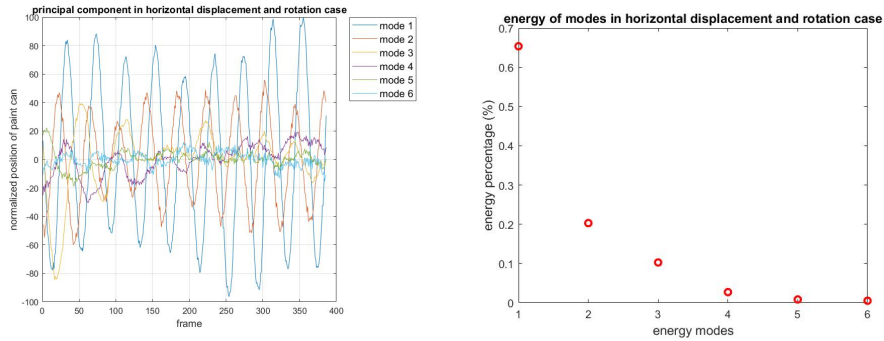


Figure 5: PCA and mode energy in horizontal displacement and rotation case

## 5 Summary and Conclusion

Through the application of PCA on the data in different cases, we can often extract the dominant components and check redundancy, like in the first and third cases. While in the noisy case, the camera shakes interfere with the performance of the PCA, and dimension reduction might be performed in this case only on the premise of applying methods that remove camera shakes; while in the rotation case, the largest singular value, which represents 65% of the dynamics is quite misleading due to the linearity of SVD methods, and thus, we fail to capture rotation through PCA.

## 6 Appendix A

**double(X)** returns the double precision of X.

**rgb2gray(X)** converts RGB image X to grayscale.

**find(M > 0.95 \* max)** finds all the pixels' positions with values that are larger than 95% of the max values of all pixels.

**mean(Mx, 2)** returns the mean values of Mx along 2 dimensions, that is, along each row of **Mx**.

**repmat(mu, 1, n)** creates a matrix with 1 \* n copies of **mu**.

**[U, S, V] = svd(Mx, 'econ')** produces the economy size of SVD, which is efficient for a m-by-n matrix **Mx** that  $m < n$ .

## 7 Appendix B

```
1 clear; close all; clc;
2
3 % Case 1 – ideal
4
5 load('cam1_1.mat')
6 load('cam2_1.mat')
7 load('cam3_1.mat')
8
9
10 [X1(1), Y1(1), ~, frameNum1(1)] = size(vidFrames1_1);
11 [X1(2), Y1(2), ~, frameNum1(2)] = size(vidFrames2_1);
12 [X1(3), Y1(3), ~, frameNum1(3)] = size(vidFrames3_1);
13
14 %{
15 numFrames = size(vidFrames3_1,4);
16 for j = 1:numFrames
17     X = vidFrames3_1(:, :, :, j);
18     imshow(X); drawnow
19 end
20 %}
21 %%
22 % trace the flashlight for each frame for each camera
    angle
23 for j = 1 : frameNum1(1)
24     M1 = double(rgb2gray(vidFrames1_1(:, :, :, j)));
25     M1(:, [1:300, 400:end]) = 0;
26     maxVal = max(M1(:));
27     [yval, xval] = find(M1 >= 0.95*maxVal);
28     x1(j, 1) = mean(xval);
29     y1(j, 1) = mean(yval);
30     %{
31         pcolor(M1)
32         shading interp
33         drawnow
34         colormap(gray)
35     %}
36 end
37
38 for j = 1 : frameNum1(2)
39     M2 = double(rgb2gray(vidFrames2_1(:, :, :, j)));
40     M2(:, [1:200, 400:end]) = 0;
41     maxVal = max(M2(:));
42     [yval, xval] = find(M2 >= 0.95*maxVal);
```



```

43     x2(j, 1) = mean(xval);
44     y2(j, 1) = mean(yval);
45     %{
46     pcolor(M2)
47     shading interp
48     drawnow
49     colormap(gray)
50     %}
51 end
52
53 % rotate back by swapping xval and yval
54 for j = 1 : frameNum1(3)
55     M3 = double(rgb2gray(vidFrames3_1(:, :, :, j)));
56     M3(:, [1:240, 480:end]) = 0;
57     M3([1:200, 350:end], :) = 0;
58     maxVal = max(M3(:));
59     [yval, xval] = find(M3 >= 0.95*maxVal);
60     x3(j, 1) = mean(yval);
61     y3(j, 1) = mean(xval);
62     %{
63     pcolor(M3)
64     shading interp
65     drawnow
66     colormap(gray)
67     %}
68 end
69
70 % cut each data so that the can
71 % starts from the highest point
72 x1 = x1(30 : end);
73 y1 = y1(30 : end);
74 x2 = x2(39 : end);
75 y2 = y2(39 : end);
76 x3 = x3(30 : end);
77 y3 = y3(30 : end);
78
79 % organize data with same length
80 minlength = min([length(y1), length(y2), length(y3)]);
81 x1 = x1(1 : minlength);
82 y1 = y1(1 : minlength);
83 x2 = x2(1 : minlength);
84 y2 = y2(1 : minlength);
85 x3 = x3(1 : minlength);
86 y3 = y3(1 : minlength);
87
88 Mx = [x1, y1, x2, y2, x3, y3];

```

```

89 Mx = Mx';
90 frame = 1:minlength;
91
92 % examine for the path of paint can and appropriate
93 % threshold values
94 figure(1)
95 for i = 1 : 3
96     subplot(3, 1, i)
97     plot(frame, Mx(2 * i - 1, :), frame, Mx(2 * i, :))
98     legend('horizontal position', 'vertical position', 'FontSize', 12, 'Location', 'bestoutside')
99     xlabel('frame', 'FontSize', 12)
100    ylabel('position in camera', 'FontSize', 12)
101    title(['path in ideal case (camera', num2str(i), ')'], 'FontSize', 12)
102 end
103
104 % subtract the mean
105 [m, n] = size(Mx);
106 mu = mean(Mx, 2);
107 Mx = Mx - repmat(mu, 1, n);
108
109
110 % perform PCA
111 [U, S, V] = svd(Mx / sqrt(n-1), 'econ');
112 Y = U' * Mx;
113
114 figure(2)
115 plot(frame, Y)
116 xlabel('frame', 'FontSize', 12)
117 ylabel('normalized position of paint can', 'FontSize', 12)
118 title('principal component in ideal case', 'FontSize', 12)
119 legend('mode 1', 'mode 2', 'mode 3', 'mode 4', 'mode 5', 'mode 6', 'FontSize', 12, 'Location', 'bestoutside')
120 grid on
121
122 figure(3)
123 sig = diag(S);
124 plot(sig.^2/sum(sig.^2), 'ro', 'Linewidth', 2)
125 xticks([0:1:6])
126 xlabel('energy modes', 'FontSize', 12)
127 ylabel('energy percentage (%)', 'FontSize', 12)
128 title('energy of modes in ideal case', 'FontSize', 12)
129

```

```

130
131 %% Case 2 – noisy
132 clear; close all; clc;
133
134 load('cam1_2.mat')
135 load('cam2_2.mat')
136 load('cam3_2.mat')
137
138 [X2(1), Y2(1), ~, frameNum2(1)] = size(vidFrames1_2);
139 [X2(2), Y2(2), ~, frameNum2(2)] = size(vidFrames2_2);
140 [X2(3), Y2(3), ~, frameNum2(3)] = size(vidFrames3_2);
141 %{
142 numFrames = size(vidFrames3_2,4);
143 for j = 1:numFrames
144     X = vidFrames3_2(:, :, :, j);
145     imshow(X); drawnow
146 end
147 %}
148 %%
149 % trace the flashlight for each frame for each camera
    angle
150 for j = 1 : frameNum2(1)
151     M1 = double(rgb2gray(vidFrames1_2(:, :, :, j)));
152     M1(:, [1:300, 400:end]) = 0;
153     M1(1:200, :) = 0;
154     maxVal = max(M1(:));
155     [yval, xval] = find(M1 >= 0.95*maxVal);
156     x1(j, 1) = mean(xval);
157     y1(j, 1) = mean(yval);
158     %{
159         pcolor(M1)
160         shading interp
161         drawnow
162         colormap(gray)
163     %}
164 end
165
166 for j = 1 : frameNum2(2)
167     M2 = double(rgb2gray(vidFrames2_2(:, :, :, j)));
168     M2(:, [1:160, 480:end]) = 0;
169     M2(400:end, :) = 0;
170     maxVal = max(M2(:));
171     [yval, xval] = find(M2 >= 0.96*maxVal);
172     x2(j, 1) = mean(xval);
173     y2(j, 1) = mean(yval);
174     %{

```

```

175     pcolor(M2)
176     shading interp
177     drawnow
178     colormap(gray)
179     %}
180 end
181
182 % rotate back by swopping xval and yval
183 for j = 1 : frameNum2(3)
184     M3 = double(rgb2gray(vidFrames3_2(:, :, :, j)));
185     M3(:, [1:260, 500:end]) = 0;
186     M3([1:160, 360:end], :) = 0;
187     maxVal = max(M3(:));
188     [yval, xval] = find(M3 >= 0.93*maxVal);
189     x3(j, 1) = mean(yval);
190     y3(j, 1) = mean(xval);
191     %{
192     pcolor(M3)
193     shading interp
194     drawnow
195     colormap(gray)
196     %}
197 end
198
199 % cut each data so that the can
200 % starts from the highest point
201 x1 = x1(12 : end);
202 y1 = y1(12 : end);
203 x2 = x2(5 : end);
204 y2 = y2(5 : end);
205 x3 = x3(15 : end);
206 y3 = y3(15 : end);
207
208 % organize data with same length
209 minlength = min([length(y1), length(y2), length(y3)]);
210 x1 = x1(1 : minlength);
211 y1 = y1(1 : minlength);
212 x2 = x2(1 : minlength);
213 y2 = y2(1 : minlength);
214 x3 = x3(1 : minlength);
215 y3 = y3(1 : minlength);
216
217 Mx = [x1, y1, x2, y2, x3, y3];
218 Mx = Mx';
219 frame = 1:minlength;
220

```

```

221 % examine for the path of paint can and appropriate
222 % threshold values
223 figure(1)
224 for i = 1 : 3
225     subplot(3, 1, i)
226     plot(frame, Mx(2 * i - 1, :), frame, Mx(2 * i, :))
227     legend('horizontal position', 'vertical position', '
        FontSize', 12, 'Location', 'bestoutside')
228     xlabel('frame', 'FontSize', 12)
229     ylabel('position in camera', 'FontSize', 12)
230     title(['path in noisy case (camera', num2str(i), ')'],
        'FontSize', 12)
231 end
232
233 % subtract the mean
234 [m, n] = size(Mx);
235 mu = mean(Mx, 2);
236 Mx = Mx - repmat(mu, 1, n);
237
238
239 % perform PCA
240 [U, S, V] = svd(Mx / sqrt(n-1), 'econ');
241 Y = U' * Mx;
242
243 figure(2)
244 plot(frame, Y)
245 xlabel('frame', 'FontSize', 12)
246 ylabel('normalized position of paint can', 'FontSize',
    12)
247 title('principal component in noisy case', 'FontSize',
    12)
248 legend('mode 1', 'mode 2', 'mode 3', 'mode 4', 'mode 5',
    'mode 6', 'FontSize', 12, 'Location', 'bestoutside')
249 grid on
250
251 figure(3)
252 sig = diag(S);
253 plot(sig.^2/sum(sig.^2), 'ro', 'Linewidth', 2)
254 xticks([0:1:6])
255 xlabel('energy modes', 'FontSize', 12)
256 ylabel('energy percentage (%)', 'FontSize', 12)
257 title('energy of modes in noisy case', 'FontSize', 12)
258
259
260 %%
261 clear; close all; clc;

```

```

262
263 % Case 3 – horizontal displacement
264 load( 'cam1_3.mat' )
265 load( 'cam2_3.mat' )
266 load( 'cam3_3.mat' )
267
268
269 [X3(1), Y3(1), ~, frameNum3(1)] = size(vidFrames1_3);
270 [X3(2), Y3(2), ~, frameNum3(2)] = size(vidFrames2_3);
271 [X3(3), Y3(3), ~, frameNum3(3)] = size(vidFrames3_3);
272
273 %%
274 numFrames = size(vidFrames1_3,4);
275 for j = 1:numFrames
276     X = vidFrames1_3(:, :, :, j);
277     imshow(X); drawnow
278 end
279 %%
280 % trace the flashlight for each frame for each camera
    angle
281 for j = 1 : frameNum3(1)
282     M1 = double(rgb2gray(vidFrames1_3(:, :, :, j)));
283     M1(:, [1:270, 400:end]) = 0;
284     M1(1:230, :) = 0;
285     maxVal = max(M1(:));
286     [yval, xval] = find(M1 >= 0.95*maxVal);
287     x1(j, 1) = mean(xval);
288     y1(j, 1) = mean(yval);
289     %{
290         pcolor(M1)
291         shading interp
292         drawnow
293         colormap(gray)
294     %}
295 end
296
297 for j = 1 : frameNum3(2)
298     M2 = double(rgb2gray(vidFrames2_3(:, :, :, j)));
299     M2(:, [1:200, 440:end]) = 0;
300     M2(1:120, :) = 0;
301     maxVal = max(M2(:));
302     [yval, xval] = find(M2 >= 0.98*maxVal);
303     x2(j, 1) = mean(xval);
304     y2(j, 1) = mean(yval);
305     %{
306         pcolor(M2)

```

```

307     shading interp
308     drawnow
309     colormap(gray)
310     %}
311 end
312
313 % rotate back by swapping xval and yval
314 for j = 1 : frameNum3(3)
315     M3 = double(rgb2gray(vidFrames3_3(:, :, :, j)));
316     M3(:, [1:200, 480:end]) = 0;
317     M3([1:120, 360:end], :) = 0;
318     maxVal = max(M3(:));
319     [yval, xval] = find(M3 >= 0.95*maxVal);
320     x3(j, 1) = mean(yval);
321     y3(j, 1) = mean(xval);
322     %{
323     pcolor(M3)
324     shading interp
325     drawnow
326     colormap(gray)
327     %}
328 end
329
330 % cut each data so that the can
331 % starts from the highest point
332 x1 = x1(9 : end);
333 y1 = y1(9 : end);
334 x2 = x2(9 : end);
335 y2 = y2(9 : end);
336 x3 = x3(11 : end);
337 y3 = y3(11 : end);
338
339 % organize data with same length
340 minlength = min([length(y1), length(y2), length(y3)]);
341 x1 = x1(1 : minlength);
342 y1 = y1(1 : minlength);
343 x2 = x2(1 : minlength);
344 y2 = y2(1 : minlength);
345 x3 = x3(1 : minlength);
346 y3 = y3(1 : minlength);
347
348 frame = 1:minlength;
349 Mx = [x1, y1, x2, y2, x3, y3];
350 Mx = Mx';
351
352 % examine for the path of paint can and appropriate

```

```

353 % threshold values
354 figure(1)
355 for i = 1 : 3
356     subplot(3, 1, i)
357     plot(frame, Mx(2 * i - 1, :), frame, Mx(2 * i, :))
358     legend('horizontal position', 'vertical position', '
        FontSize', 12, 'Location', 'bestoutside')
359     xlabel('frame', 'FontSize', 12)
360     ylabel('position in camera', 'FontSize', 12)
361     title(['path in horizontal displacement case (camera'
        , num2str(i), ')'], 'FontSize', 12)
362 end
363
364 % subtract the mean
365 [m, n] = size(Mx);
366 mu = mean(Mx, 2);
367 Mx = Mx - repmat(mu, 1, n);
368
369
370 % perform PCA
371 [U, S, V] = svd(Mx / sqrt(n-1), 'econ');
372 Y = U' * Mx;
373
374 figure(2)
375 plot(frame, Y)
376 xlabel('frame', 'FontSize', 12)
377 ylabel('normalized position of paint can', 'FontSize',
    12)
378 title('principal component in horizontal displacement
    case', 'FontSize', 12)
379 legend('mode 1', 'mode 2', 'mode 3', 'mode 4', 'mode 5',
    'mode 6', 'FontSize', 12, 'Location', 'bestoutside')
380 grid on
381
382 figure(3)
383 sig = diag(S);
384 plot(sig.^2/sum(sig.^2), 'ro', 'Linewidth', 2)
385 xticks([0:1:6])
386 xlabel('energy modes', 'FontSize', 12)
387 ylabel('energy percentage (%)', 'FontSize', 12)
388 title('energy of modes in horizontal displacement case',
    'FontSize', 12)
389
390 %%
391 clear; close all; clc;
392

```



```

393 % Case 4 – horizontal displacement and rotation
394 load('cam1_4.mat')
395 load('cam2_4.mat')
396 load('cam3_4.mat')
397
398
399 [X4(1), Y4(1), ~, frameNum4(1)] = size(vidFrames1_4);
400 [X4(2), Y4(2), ~, frameNum4(2)] = size(vidFrames2_4);
401 [X4(3), Y4(3), ~, frameNum4(3)] = size(vidFrames3_4);
402
403 %%
404 numFrames = size(vidFrames2_4,4);
405 for j = 1:numFrames
406     X = vidFrames2_4(:, :, :, j);
407     imshow(X); drawnow
408 end
409 %%
410 % trace the flashlight for each frame for each camera
    angle
411 for j = 1 : frameNum4(1)
412     M1 = double(rgb2gray(vidFrames1_4(:, :, :, j)));
413     M1(:, [1:300, 480:end]) = 0;
414     M1([1:200, 420:end], :) = 0;
415     maxVal = max(M1(:));
416     [yval, xval] = find(M1 >= 0.95*maxVal);
417     x1(j, 1) = mean(xval);
418     y1(j, 1) = mean(yval);
419     %{
420     pcolor(M1)
421     shading interp
422     drawnow
423     colormap(gray)
424     %}
425 end
426
427 for j = 1 : frameNum4(2)
428     M2 = double(rgb2gray(vidFrames2_4(:, :, :, j)));
429     M2(:, [1:210, 420:end]) = 0;
430     M2([1:60, 400:end], :) = 0;
431     maxVal = max(M2(:));
432     [yval, xval] = find(M2 >= 0.98*maxVal);
433     x2(j, 1) = mean(xval);
434     y2(j, 1) = mean(yval);
435     %{
436     pcolor(M2)
437     shading interp

```

```

438         drawnow
439         colormap(gray)
440     %}
441 end
442
443 % rotate back by swopping xval and yval
444 for j = 1 : frameNum4(3)
445     M3 = double(rgb2gray(vidFrames3_4(:, :, :, j)));
446     M3(:, [1:300, 480:end]) = 0;
447     M3([1:120, 320:end], :) = 0;
448     maxVal = max(M3(:));
449     [yval, xval] = find(M3 >= 0.92*maxVal);
450     x3(j, 1) = mean(yval);
451     y3(j, 1) = mean(xval);
452     %{
453         pcolor(M3)
454         shading interp
455         drawnow
456         colormap(gray)
457     %}
458 end
459
460 % cut each data so that the can
461 % starts from the highest point
462 x1 = x1(7 : end);
463 y1 = y1(7 : end);
464 x2 = x2(5 : end);
465 y2 = y2(5 : end);
466 x3 = x3(1 : end);
467 y3 = y3(1 : end);
468
469 % organize data with same length
470 minlength = min([length(y1), length(y2), length(y3)]);
471 x1 = x1(1 : minlength);
472 y1 = y1(1 : minlength);
473 x2 = x2(1 : minlength);
474 y2 = y2(1 : minlength);
475 x3 = x3(1 : minlength);
476 y3 = y3(1 : minlength);
477
478 frame = 1:minlength;
479 Mx = [x1, y1, x2, y2, x3, y3];
480 Mx = Mx';
481
482 % examine for the path of paint can and appropriate
483 % threshold values

```

```

484 figure(1)
485 for i = 1 : 3
486     subplot(3, 1, i)
487     plot(frame, Mx(2 * i - 1, :), frame, Mx(2 * i, :))
488     legend('horizontal position', 'vertical position', '
        FontSize', 12, 'Location', 'bestoutside')
489     xlabel('frame', 'FontSize', 12)
490     ylabel('position in camera', 'FontSize', 12)
491     title(['path (horizontal displacement & rotation) (
        camera', num2str(i), ')'], 'FontSize', 12)
492 end
493
494 % subtract the mean
495 [m, n] = size(Mx);
496 mu = mean(Mx, 2);
497 Mx = Mx - repmat(mu, 1, n);
498
499 % perform PCA
500 [U, S, V] = svd(Mx / sqrt(n-1), 'econ');
501 Y = U' * Mx;
502
503 figure(2)
504 plot(frame, Y)
505 xlabel('frame', 'FontSize', 12)
506 ylabel('normalized position of paint can', 'FontSize',
    12)
507 title('principal component in horizontal displacement and
    rotation case', 'FontSize', 12)
508 legend('mode 1', 'mode 2', 'mode 3', 'mode 4', 'mode 5',
    'mode 6', 'FontSize', 12, 'Location', 'bestoutside')
509 grid on
510
511 figure(3)
512 sig = diag(S);
513 plot(sig.^2/sum(sig.^2), 'ro', 'Linewidth', 2)
514 xticks([0:1:6])
515 xlabel('energy modes', 'FontSize', 12)
516 ylabel('energy percentage (%)', 'FontSize', 12)
517 title('energy of modes in horizontal displacement and
    rotation case', 'FontSize', 12)

```