

PCA and a Spring-Mass System

Luotong Kang

Abstract

Using Singular Value Decomposition (SVD), we are able to do Principal Component Analysis, in which we geometrically change the basis of data matrices. This time, we are going to look at the Spring-Mass System in 4 cases and compare the performance of PCA with different types noisy data.

1 Introduction

Principle Component Analysis is a powerful tool for scientist to discover the characteristics of multi-dimensional data low-rank approximations. In this report, we separate videos of spring-mass harmonic motion into 2-D frames through time.

We are going to compare four different scenarios have been analyzed. Each scenario is filmed by 3 cameras in different angles and different starting time. Although movement of mass of all four data sets is effected by some noise (background color, unstable angles, etc.), it is reasonable to consider Task 1 as ideal case. In other 3 tasks, we meet obstacles like swinging motion of mass.

2 Theoretical Background

2.1 Singular Value Decomposition

We can always decomposite $\mathbf{A} \in \mathbb{C}^{m \times n}$ as the product of three matrices:

$$\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^*$$

where $\mathbf{U} \in \mathbb{R}^{m \times m}$ and $\mathbf{V} \in \mathbb{R}^{n \times n}$ are unitary and $\mathbf{\Sigma} \in \mathbb{R}^{m \times n}$ is diagonal. The values σ_n on the diagonal of $\mathbf{\Sigma}$, ordered in descending manner, are the singular values of \mathbf{A} . They are also square roots of eigenvalues of $\mathbf{A}\mathbf{A}^*$. The column vectors of \mathbf{U} and \mathbf{V} are called left singular vectors (eigenvectors of $\mathbf{A}\mathbf{A}^*$) and right singular vectors respectively (eigenvectors of $\mathbf{A}^*\mathbf{A}$).

Geometrically, we are separating the linear transformation into two rotation of basis (principal semiaxes) and one scaling along basis. Based on geometric understanding, SVD has unique properties comparing to eigenvalue decomposition, making it proper for PCA:

- SVD can be performed on matrices of any size.
- SVD gives orthogonal bases.

2.2 Principle Component Analysis

Using the knowledge of statistics, we represent redundancy of data one population using sample variance (we always assume that we have subtracted the mean off from our data):

$$\sigma^2 = \frac{1}{n-1} \mathbf{a} \mathbf{a}^T$$

where \mathbf{a} is the information data, and n is the length of \mathbf{a} . $(n-1)$ is divided to avoid bias from small samples.

When looking at the statistical dependence of two data sets a and b with same length, we use covariance:

$$\sigma_{\mathbf{ab}}^2 = \frac{1}{n-1} \mathbf{a} \mathbf{b}^T,$$

in which large covariance correspond to high redundancy.

We use the same idea in PCA, in which we compute all the variances and covariances between row of \mathbf{X} (data matrix) with the covariance matrix

$$\mathbf{C}_x = \frac{1}{n-1} \mathbf{X} \mathbf{X}^T = \mathbf{A} \mathbf{A}^T = \mathbf{U} \mathbf{\Sigma}^2 \mathbf{U}^T, \quad \mathbf{A} = \frac{1}{\sqrt{n-1}} \mathbf{X}$$

Therefore, doing SVD to A help us analyze X . After this change of basis, we can compute data in the new coordinates

$$\mathbf{Y} = \mathbf{U}^T \mathbf{X}, \quad \mathbf{C}_Y = 0$$

We can also compute best rank N approximation using first N singular values and singular vectors.

3 Algorithm Implementation and Development

We are given the solution for Spring-mass System $z(t) = A \cos(\omega t + \phi)$. However, instead of saying z direction or $x-y$ direction, we want to denote position of mass using (x_1, y_1) for camera 1, (x_2, y_2) for camera 2 and (x_3, y_3) for camera 3. Since camera 3 is filmed in a different angle, we know that displacement of y_1, y_2, x_3 represent movement is z direction, and the other three element arrays represents $x-y$ direction. The length of the 6 element array is determined by synchronized frame numbers. The algorithm is very similar in all four cases, so I'll only cover the general procedure once.

We first load all videos of all 3 cameras and derive the frame number of each video using `size` command. We treat each frame as an image and handle the each video using a loop. In the loop, we first convert images into grey-scale using `rgb2gray`.

The most complicated part of algorithm development is tracking the position of mass in each frame. We are going to using the strategy of locating the pixels using color information in a filtered region. After playing the video in the loop using `imshow()`, we know that the mass we are tracking is white. Thus we rescale the image matrix to the range $[0, 1]$, in which we know larger an element is, the whiter its corresponding pixel will be. To filter out the region of the motion occurs, we'll use `imcrop()` for each frame. By exploring for several time, the cropped area should be as narrow as possible to avoid noisy

information from background color. Note the bright points on top of the mass is very helpful (when we can observe it) because we know that pixel will have a value close to 1. To judge whether pixels are representing the mass or not, we need to see whether it is "white enough", in other words, whether their value is close enough to 1. The threshold we set should satisfy the logical condition `max(X,[],'all')>Threshold`, making sure that we can find mass in every frame. Then, we use `[Y,X]=find(X>=Threshold)` to grab the coordinates of pixels with values higher than/equal to the threshold we've set. There might be more than one pixels getting over the threshold, so we take `mean` to the coordinates in order to keep only one representative pixel in each frame. Repeat this process for 3 cameras and align all coordinates to 3 matrix (each has size= $2 \times \text{frame number of corresponding video}$). The columns of those matrices are $x_1, y_1, x_2, y_2, x_3, x_4$ we need for one task.

Next, we are going to trim the 3 data matrix to same length and synchronized motion. After plotting y_1, y_2, x_3 , we know the approximate time (in frames) that first local min occurs. Using `min()`, we make the first frame in each of the videos to be the frame that provides the local min pixel. After identifying the shortest matrix, we trim other two matrices to have make all coordinates arrays have the same length n .

Stack the 3 matrix, we get the data matrix (\mathbf{X} in Section 2) as the input of PCA. After subtracting the mean from the the data and scale it by $\frac{1}{\sqrt{n-1}}$ element-wise, we can compute SVD using `[U,S,V]=svd()`. To draw displacement from approximation with different ranks, we produce the principal components projection by computing $\mathbf{U}' * \mathbf{X}$. We can also represent energy of those approximation by extracting squared diagonal of \mathbf{S} . We plot displacement from mean of original data on the top of figure, displacement from approximations in the middle, and the energy of approximations on the bottom of the figure.

4 Computational Results

4.1 Task 1: Ideal Case

In this case, we have little noise from motion itself, the only thing we care about is to correctly set the range of hue. Because most redundancy is eliminated, Rank 1 approximation is sufficient in terms of capturing the energy in this system (see bottom graph of Figure 1). We can also get this conclusion by interpreting the middle graph, in which the Rank 2 and Rank 3 curves always appears around displacement = 0. Thus, the projection will basically not change the location of data points.

4.2 Task 2: Noisy Case

With shaking cameras, we get those messy zigzags on the top graph while in Task 1 we get smooth curves. However, the consistence between y_2 and Rank1 indicates that the approximation is not too bad. There's still 60.57% energy captured by Rank1 approximation. Also, we can see the sinusoidal shape especially for Rank2 displacement. It is also true that the first singular value is significantly larger than all the others, marking the essence of 1D motion.

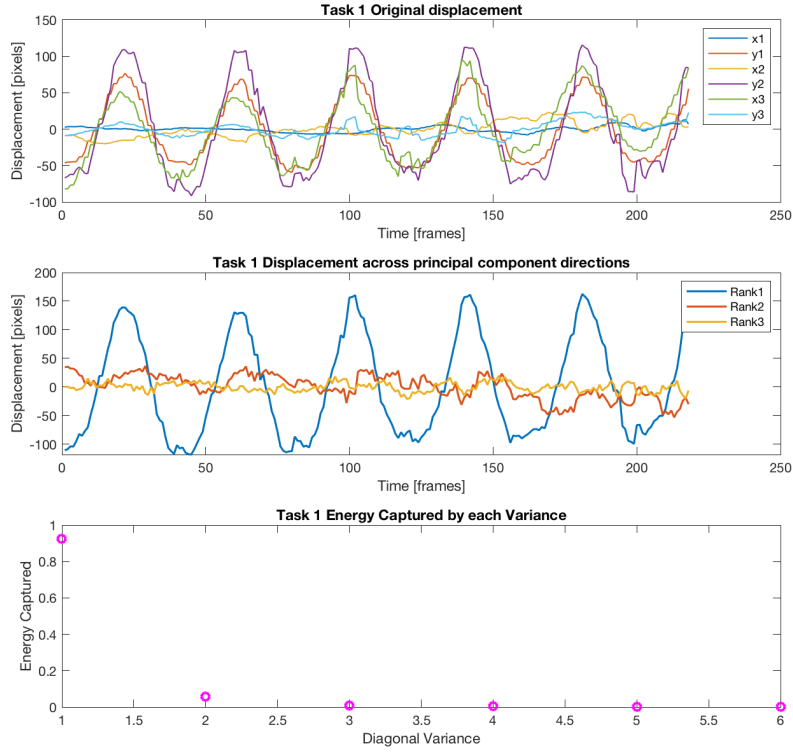


Figure 1: 3 plots for Task 1 (see titles for details).

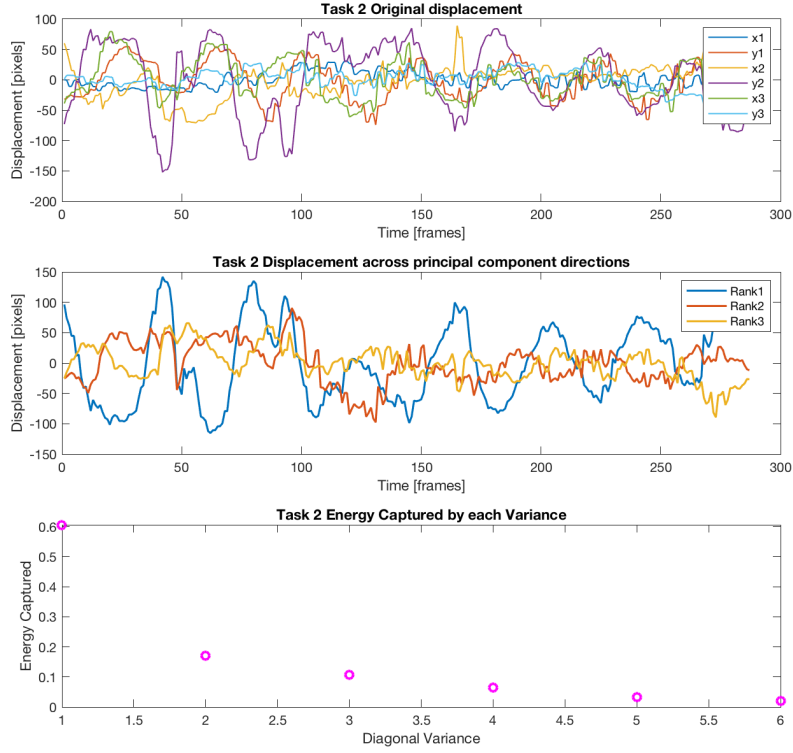


Figure 2: 3 plots for Task 2 (see titles for details).

4.3 Task 3: Horizontal Displacement

Similar to Task 1, we get very smooth sinusoidal curve since the "sharp" movement caused by intentional shaking disappears. However, we didn't get better performance than Task 2 in terms of energy captured by Rank 1 approximation. This is caused by the pendulum motion. With angular velocity changing, pendulum motion is also harmonic oscillation that can be described by sine functions, which explained $x1, x2, y3$ curve on the top graph. If we want to capture $> 90\%$ energy, we need to do Rank 3 approximation.

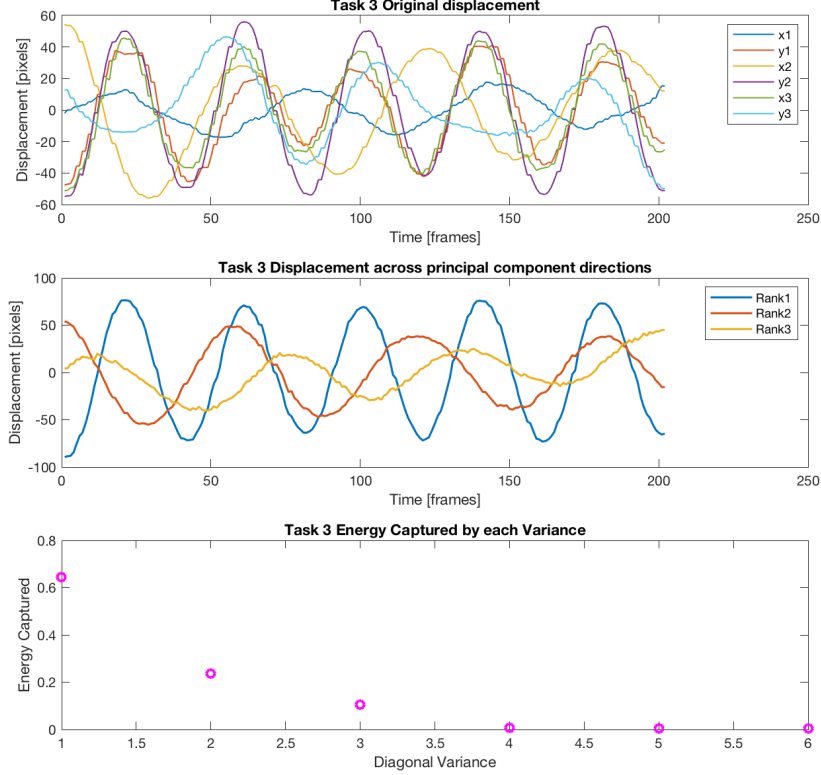


Figure 3: 3 plots for Task 3 (see titles for details).

4.4 Task 4: Horizontal Displacement and Rotation

In this Task, we see a graph very similar to the one observed for Task 3. We have both oscillation, rotation, and pendulum motion. Thus, we need first 3 diagonal variance to get a relative precise approximation. However, although with of the coexistence of rotation and pendulum motions, the amplitude is not as big as in Task 3 and rotation factors little to the main direction, so the Rank 1 approximation performs better comparing in the previous task.

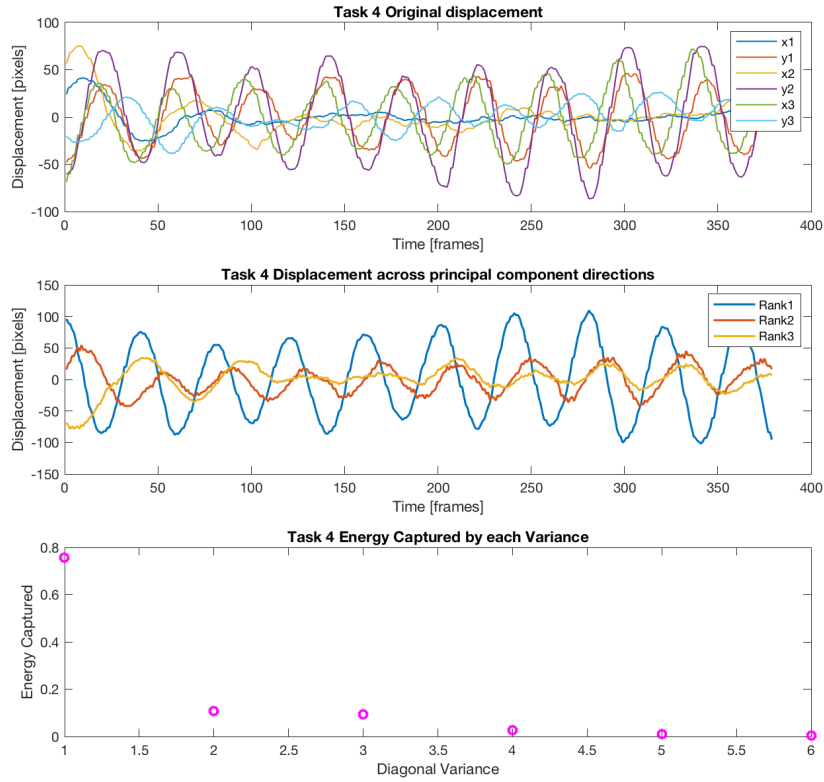


Figure 4: 3 plots for Task 4 (see titles for details).

5 Summary and Conclusions

Using Principal Components of multi-dimensional data set, we can analyze real-world phenomenon in terms of redundancy and direction. Energy captured is obvious after applying Singular Value Decomposition. From the four tasks, we can see the power of PCA even with those noise added.

6 Appendices

6.1 MATLAB functions used

The commands are NOT listed chronologically.

`load()`: loads the data stored in file.

`zeros(size)`, `ones(size)`: creates matrices with all entries being zero/one with indicated size.

`set(gca,)`, `set(gcf,)`, `title()`, `xlabel()`, `ylabel()`, `yticks()`, `yticklabels()`: edit component labels/axes of the graph.

`im2double()`: converts the image to double precision, rescales the output from integer data types to the range [0, 1].

`rgb2gray()`: converts the truecolor image to the grayscale image.

`imcrop(I,[x,y,width,height])`: crops the image according to the position and dimensions specified in the crop rectangle. `x,y` specifies the position in spatial coordinates.

`[Y,X]=find()`: returns the row and column subscripts of each nonzero element in array `X` using any of the input arguments in previous syntaxes.

`max(A,[],'all')`: finds the maximum over all elements of `A`.

`min(V)`: returns the minimum of vector `V`.

`size(A)`: returns a row vector whose elements are the lengths of the corresponding dimensions of `A`.

`repmat(A,n)`: returns an array containing `n` copies of `A` in the row and column dimensions. The size of array is `size(A)*n` when `A` is a matrix.

`mean(V)`: `V` is a vector, returns the mean of the elements.

`sum(V)`: `V` is a vector, returns the sum of the elements.

`diag(A)`: returns a column vector of the main diagonal elements of `A`.

6.2 MATLAB codes

```
1 %% T1
2 %% T1
3 clear; close all; clc;
4 load('cam1_1.mat')
5 load('cam2_1.mat')
6 load('cam3_1.mat')
7 framenum1 =size(vidFrames1_1,4);
8 framenum2 =size(vidFrames2_1,4);
9 framenum3 =size(vidFrames3_1,4);
10 raw1 = zeros(framenum1,2);
11 raw2 = zeros(framenum2,2);
12 raw3 = zeros(framenum3,2);
13
14 %%
15 for j = 1:framenum1
16     X1 = imcrop(vidFrames1_1(:, :, :, j), [310,220,50,200]);
17     X1 = im2double(rgb2gray(X1));
18     imshow(X1); drawnow
19     if max(X1,[],'all')<0.988
```

```

20         max(X1,[], 'all')
21     end
22     [Y, X] = find(X1>=0.988);
23     raw1(j,:) = [mean(X), mean(Y)];
24 end
25 for j = 1:framenum2
26     X2 = imcrop(vidFrames2_1(:,:,j), [260,70,65,300]);
27     X2 = im2double(rgb2gray(X2));
28     %imshow(X2); drawnow
29     if max(X2,[], 'all')<0.988
30         max(X2,[], 'all')
31     end
32     [Y, X] = find(X2>=0.988);
33     raw2(j,:) = [mean(X), mean(Y)];
34 end
35 for j = 1:framenum3
36     X3 = imcrop(vidFrames3_1(:,:,j), [260,230,230,70]);
37     X3 = im2double(rgb2gray(X3));
38     %imshow(X3); drawnow
39     if max(X3,[], 'all')<0.97
40         max(X3,[], 'all')
41     end
42     [Y, X] = find(X3>=0.97);
43     raw3(j,:) = [mean(X), mean(Y)];
44 end
45 % % find where to trim
46 % subplot(311)
47 % plot(1:length(raw1),raw1(:,2))
48 % subplot(312)
49 % plot(1:length(raw2),raw2(:,2))
50 % subplot(313)
51 % plot(1:length(raw3),raw3(:,1))
52 %%
53 [M,I] = min(raw1(1:20,2));
54 trimmed1 = raw1(I:end,:);
55 [M,I] = min(raw2(1:20,2));
56 trimmed2 = raw2(I:end,:);
57 [M,I] = min(raw3(1:20,1));
58 trimmed3 = raw3(I:end,:);
59 trimmedtime=min([length(trimmed1),length(trimmed2),length(trimmed3)]);
60 data1 = trimmed1(1:trimmedtime, :);
61 data2 = trimmed2(1:trimmedtime, :);
62 data3 = trimmed3(1:trimmedtime, :);
63 pltdata = [data1';data2';data3'];
64 [m,n]=size(pltdata); % compute data size
65 mn=mean(pltdata,2); % compute mean for each row
66 pltdata=pltdata-repmat(mn,1,n); % subtract mean
67 [U,S,V]=svd(pltdata/sqrt(n-1)); % perform the SVD
68 lambda=diag(S).^2; % produce diagonal variances
69 Y= U' * pltdata; % produce the principal components projection
70
71 %%
72 figure()
73 subplot(311)
74 plot(1:n, pltdata(1,:), 'Linewidth',1)

```



```

75 hold on
76 plot(1:n, pltdata(2,:), 'Linewidth',1)
77 plot(1:n, pltdata(3,:), 'Linewidth',1)
78 plot(1:n, pltdata(4,:), 'Linewidth',1)
79 plot(1:n, pltdata(5,:), 'Linewidth',1)
80 plot(1:n, pltdata(6,:), 'Linewidth',1)
81 ylabel("Displacement [pixels]"); xlabel("Time [frames]");
82 title("Task 1 Original displacement");
83 legend('x1', 'y1', 'x2', 'y2', 'x3', 'y3')
84 subplot(312)
85 plot(1:trimmedtime, Y(1,:), 'Linewidth',1.5)
86 hold on
87 plot(1:trimmedtime, Y(2,:), 'Linewidth',1.5)
88 plot(1:trimmedtime, Y(3,:), 'Linewidth',1.5)
89 ylabel("Displacement [pixels]"); xlabel("Time [frames]");
90 title('Task 1 Displacement across principal component directions');
91 legend('Rank1', 'Rank2', 'Rank3')
92 subplot(313)
93 plot(1:6, lambda/sum(lambda), 'mo', 'Linewidth', 2);
94 title('Task 1 Energy Captured by each Variance');
95 xlabel('Diagonal Variance'); ylabel('Energy Captured');
96 set(gcf, 'Position', [0,0,700,700])
97 print('Task1', '-dpng')
98
99
100 %% T2
101 clear; close all; clc;
102 load('cam1_2.mat')
103 load('cam2_2.mat')
104 load('cam3_2.mat')
105 framenum1 =size(vidFrames1_2,4);
106 framenum2 =size(vidFrames2_2,4);
107 framenum3 =size(vidFrames3_2,4);
108 raw1 = zeros(framenum1,2);
109 raw2 = zeros(framenum2,2);
110 raw3 = zeros(framenum3,2);
111
112 %%
113 for j = 1:framenum1
114     X1 = imcrop(vidFrames1_2(:,:, :, j), [320,220,70,170]);
115     X1 = im2double(rgb2gray(X1));
116     %imshow(X1); drawnow
117     if max(X1, [], 'all')<0.98
118         max(X1, [], 'all')
119     end
120     [Y, X] = find(X1>=0.98);
121     raw1(j, :) = [mean(X), mean(Y)];
122 end
123 for j = 1:framenum2
124     X2 = imcrop(vidFrames2_2(:,:, :, j), [220,65,175,295]);
125     X2 = im2double(rgb2gray(X2));
126     %imshow(X2); drawnow
127     if max(X2, [], 'all')<0.975
128         max(X2, [], 'all')
129     end

```

```

130     [Y, X] = find(X2>=0.975);
131     raw2(j,:) = [mean(X), mean(Y)];
132 end
133 for j = 1:framenum3
134     X3 = imcrop(vidFrames3_2(:,:,j), [260,200,230,100]);
135     X3 = im2double(rgb2gray(X3));
136     %imshow(X3); drawnow
137     if max(X3,[], 'all')<0.964
138         max(X3,[], 'all')
139     end
140     [Y, X] = find(X3>=0.964);
141     raw3(j,:) = [mean(X), mean(Y)];
142 end
143 % % find where to trim
144 % subplot(311)
145 % plot(1:length(raw1),raw1(:,2))
146 % subplot(312)
147 % plot(1:length(raw2),raw2(:,2))
148 % subplot(313)
149 % plot(1:length(raw3),raw3(:,1))
150
151 %%
152 [M,I] = min(raw1(1:40,2));
153 trimmed1 = raw1(I:end,:);
154 [M,I] = min(raw2(1:40,2));
155 trimmed2 = raw2(I:end,:);
156 [M,I] = min(raw3(1:40,1));
157 trimmed3 = raw3(I:end,:);
158 trimmedtime=min([length(trimmed1),length(trimmed2),length(trimmed3)]);
159 data1 = trimmed1(1:trimmedtime, :);
160 data2 = trimmed2(1:trimmedtime, :);
161 data3 = trimmed3(1:trimmedtime, :);
162 pltdata = [data1';data2';data3'];
163 [m,n]=size(pltdata); % compute data size
164 mn=mean(pltdata,2); % compute mean for each row
165 pltdata=pltdata-repmat(mn,1,n); % subtract mean
166 [U,S,V]=svd(pltdata/sqrt(n-1)); % perform the SVD
167 lambda=diag(S).^2; % produce diagonal variances
168 Y= U'*pltdata; % produce the principal components projection
169
170 %%
171 figure()
172 subplot(311)
173 plot(1:trimmedtime, pltdata(1,:), 'Linewidth',1)
174 hold on
175 plot(1:trimmedtime, pltdata(2,:), 'Linewidth',1)
176 hold on
177 plot(1:trimmedtime, pltdata(3,:), 'Linewidth',1)
178 plot(1:trimmedtime, pltdata(4,:), 'Linewidth',1)
179 plot(1:trimmedtime, pltdata(5,:), 'Linewidth',1)
180 plot(1:trimmedtime, pltdata(6,:), 'Linewidth',1)
181 ylabel("Displacement [pixels]"); xlabel("Time [frames]");
182 title("Task 2 Original displacement");
183 legend('x1','y1','x2','y2','x3','y3')
184 subplot(312)

```

```

185 plot(1:trimmedtime, Y(1,:), 'Linewidth',1.5)
186 hold on
187 plot(1:trimmedtime, Y(2,:), 'Linewidth',1.5)
188 plot(1:trimmedtime, Y(3,:), 'Linewidth',1.5)
189 ylabel("Displacement [pixels]"); xlabel("Time [frames]");
190 title('Task 2 Displacement across principal component directions');
191 legend('Rank1', 'Rank2', 'Rank3')
192 subplot(313)
193 plot(1:6, lambda/sum(lambda), 'mo', 'Linewidth', 2);
194 title('Task 2 Energy Captured by each Variance');
195 xlabel('Diagonal Variance'); ylabel('Energy Captured');
196 set(gcf, 'Position', [0,0,700,700])
197 print('Task2', '-dpng')
198
199
200 %% T3
201 clear all; close all; clc;
202 load('cam1_3.mat')
203 load('cam2_3.mat')
204 load('cam3_3.mat')
205 framenum1 =size(vidFrames1_3,4);
206 framenum2 =size(vidFrames2_3,4);
207 framenum3 =size(vidFrames3_3,4);
208 raw1 = zeros(framenum1,2);
209 raw2 = zeros(framenum2,2);
210 raw3 = zeros(framenum3,2);
211
212 %%
213 for j = 1:framenum1
214     X1 = imcrop(vidFrames1_3(:,:, :, j), [280,220,110,170]);
215     X1 = im2double(rgb2gray(X1));
216     %imshow(X1); drawnow
217     if max(X1, [], 'all')<0.98
218         max(X1, [], 'all')
219     end
220     [Y, X] = find(X1>=0.98);
221     raw1(j, :) = [mean(X), mean(Y)];
222 end
223 for j = 1:framenum2
224     X2 = imcrop(vidFrames2_3(:,:, :, j), [230,180,175,205]);
225     X2 = im2double(rgb2gray(X2));
226     %imshow(X2); drawnow
227     if max(X2, [], 'all')<0.975
228         max(X2, [], 'all')
229     end
230     [Y, X] = find(X2>=0.975);
231     raw2(j, :) = [mean(X), mean(Y)];
232 end
233 for j = 1:framenum3
234     X3 = imcrop(vidFrames3_3(:,:, :, j), [260,180,210,150]);
235     X3 = im2double(rgb2gray(X3));
236     %imshow(X3); drawnow
237     if max(X3, [], 'all')<0.964
238         max(X3, [], 'all')
239     end

```

```

240     [Y, X] = find(X3>=0.964);
241     raw3(j,:) = [mean(X),mean(Y)];
242 end
243 % % find where to trim
244 % subplot(311)
245 % plot(1:length(raw1),raw1(:,2))
246 % subplot(312)
247 % plot(1:length(raw2),raw2(:,2))
248 % subplot(313)
249 % plot(1:length(raw3),raw3(:,1))
250
251 %%
252 [M,I] = min(raw1(1:40,2));
253 trimmed1 = raw1(I:end,:);
254 [M,I] = min(raw2(1:40,2));
255 trimmed2 = raw2(I:end,:);
256 [M,I] = min(raw3(1:40,1));
257 trimmed3 = raw3(I:end,:);
258 trimmedtime=min([length(trimmed1),length(trimmed2),length(trimmed3)]);
259 data1 = trimmed1(1:trimmedtime, :);
260 data2 = trimmed2(1:trimmedtime, :);
261 data3 = trimmed3(1:trimmedtime, :);
262 pltdata = [data1';data2';data3'];
263 [m,n]=size(pltdata); % compute data size
264 mn=mean(pltdata,2); % compute mean for each row
265 pltdata=pltdata-repmat(mn,1,n); % subtract mean
266 [U,S,V]=svd(pltdata/sqrt(n-1)); % perform the SVD
267 lambda=diag(S).^2; % produce diagonal variances
268 Y= U'*pltdata; % produce the principal components projection
269
270 %%
271 figure()
272 subplot(311)
273 plot(1:trimmedtime, pltdata(1,:), 'Linewidth',1)
274 hold on
275 plot(1:trimmedtime, pltdata(2,:), 'Linewidth',1)
276 plot(1:trimmedtime, pltdata(3,:), 'Linewidth',1)
277 plot(1:trimmedtime, pltdata(4,:), 'Linewidth',1)
278 plot(1:trimmedtime, pltdata(5,:), 'Linewidth',1)
279 plot(1:trimmedtime, pltdata(6,:), 'Linewidth',1)
280 ylabel("Displacement [pixels]"); xlabel("Time [frames]");
281 title("Task 3 Original displacement");
282 legend('x1', 'y1', 'x2', 'y2', 'x3', 'y3')
283 subplot(312)
284 plot(1:trimmedtime, Y(1,:), 'Linewidth',1.5)
285 hold on
286 plot(1:trimmedtime, Y(2,:), 'Linewidth',1.5)
287 plot(1:trimmedtime, Y(3,:), 'Linewidth',1.5)
288 ylabel("Displacement [pixels]"); xlabel("Time [frames]");
289 title('Task 3 Displacement across principal component directions');
290 legend('Rank1', 'Rank2', 'Rank3')
291 subplot(313)
292 plot(1:6, lambda/sum(lambda), 'mo', 'Linewidth', 2);
293 title('Task 3 Energy Captured by each Variance');
294 xlabel('Diagonal Variance'); ylabel('Energy Captured');

```

```

295 set(gcf, 'Position', [0,0,700,700])
296 print('Task3', '-dpng')
297
298
299 %% T4
300 clear; close; clc;
301 load('cam1_4.mat')
302 load('cam2_4.mat')
303 load('cam3_4.mat')
304 framenum1 =size(vidFrames1_4,4);
305 framenum2 =size(vidFrames2_4,4);
306 framenum3 =size(vidFrames3_4,4);
307 raw1 = zeros(framenum1,2);
308 raw2 = zeros(framenum2,2);
309 raw3 = zeros(framenum3,2);
310
311 %%
312 for j = 1:framenum1
313     X1 = imcrop(vidFrames1_4(:,:,j), [330,225,110,165]);
314     X1 = im2double(rgb2gray(X1));
315     %imshow(X1); drawnow
316     if max(X1,[], 'all')<0.9685
317         max(X1,[], 'all')
318     end
319     [Y, X] = find(X1>=0.9685);
320     raw1(j,:) = [mean(X), mean(Y)];
321 end
322 for j = 1:framenum2
323     X2 = imcrop(vidFrames2_4(:,:,j), [220,100,210,235]);
324     X2 = im2double(rgb2gray(X2));
325     %imshow(X2); drawnow
326     if max(X2,[], 'all')<0.976
327         max(X2,[], 'all')
328     end
329     [Y, X] = find(X2>= 0.976);
330     raw2(j,:) = [mean(X), mean(Y)];
331 end
332 for j = 1:framenum3
333     X3 = imcrop(vidFrames3_4(:,:,j), [310,130,205,150]);
334     X3 = im2double(rgb2gray(X3));
335     %imshow(X3); drawnow
336     if max(X3,[], 'all')<0.917
337         max(X3,[], 'all')
338     end
339     [Y, X] = find(X3>=0.917);
340     raw3(j,:) = [mean(X),mean(Y)];
341 end
342 % % find where to trim
343 % subplot(311)
344 % plot(1:length(raw1),raw1(:,2))
345 % subplot(312)
346 % plot(1:length(raw2),raw2(:,2))
347 % subplot(313)
348 % plot(1:length(raw3),raw3(:,1))
349

```

```

350 %%
351 [M,I] = min(raw1(1:25,2));
352 trimmed1 = raw1(I:end,:);
353 [M,I] = min(raw2(1:25,2));
354 trimmed2 = raw2(I:end,:);
355 [M,I] = min(raw3(1:25,1));
356 trimmed3 = raw3(I:end,:);
357 trimmedtime=min([length(trimmed1),length(trimmed2),length(trimmed3)]);
358 data1 = trimmed1(1:trimmedtime, :);
359 data2 = trimmed2(1:trimmedtime, :);
360 data3 = trimmed3(1:trimmedtime, :);
361 pltdata = [data1';data2';data3'];
362 [m,n]=size(pltdata); % compute data size
363 mn=mean(pltdata,2); % compute mean for each row
364 pltdata=pltdata-repmat(mn,1,n); % subtract mean
365 [U,S,V]=svd(pltdata/sqrt(n-1)); % perform the SVD
366 lambda=diag(S).^2; % produce diagonal variances
367 Y= U'*pltdata; % produce the principal components projection
368
369 %%
370 figure()
371 subplot(311)
372 plot(1:trimmedtime, pltdata(1,:), 'Linewidth',1)
373 hold on
374 plot(1:trimmedtime, pltdata(2,:), 'Linewidth',1)
375 plot(1:trimmedtime, pltdata(3,:), 'Linewidth',1)
376 plot(1:trimmedtime, pltdata(4,:), 'Linewidth',1)
377 plot(1:trimmedtime, pltdata(5,:), 'Linewidth',1)
378 plot(1:trimmedtime, pltdata(6,:), 'Linewidth',1)
379 ylabel("Displacement [pixels]"); xlabel("Time [frames]");
380 title("Task 4 Original displacement");
381 legend('x1', 'y1', 'x2', 'y2', 'x3', 'y3')
382 subplot(312)
383 plot(1:trimmedtime, Y(1,:), 'Linewidth',1.5)
384 hold on
385 plot(1:trimmedtime, Y(2,:), 'Linewidth',1.5)
386 plot(1:trimmedtime, Y(3,:), 'Linewidth',1.5)
387 ylabel("Displacement [pixels]"); xlabel("Time [frames]");
388 title('Task 4 Displacement across principal component directions');
389 legend('Rank1', 'Rank2', 'Rank3')
390 subplot(313)
391 plot(1:6, lambda/sum(lambda), 'mo', 'Linewidth', 2);
392 title('Task 4 Energy Captured by each Variance');
393 xlabel('Diagonal Variance'); ylabel('Energy Captured');
394 set(gcf, 'Position', [0,0,700,700])
395 print('Task4', '-dpng')

```