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:

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

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×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. Aftering 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 `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 y2 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.

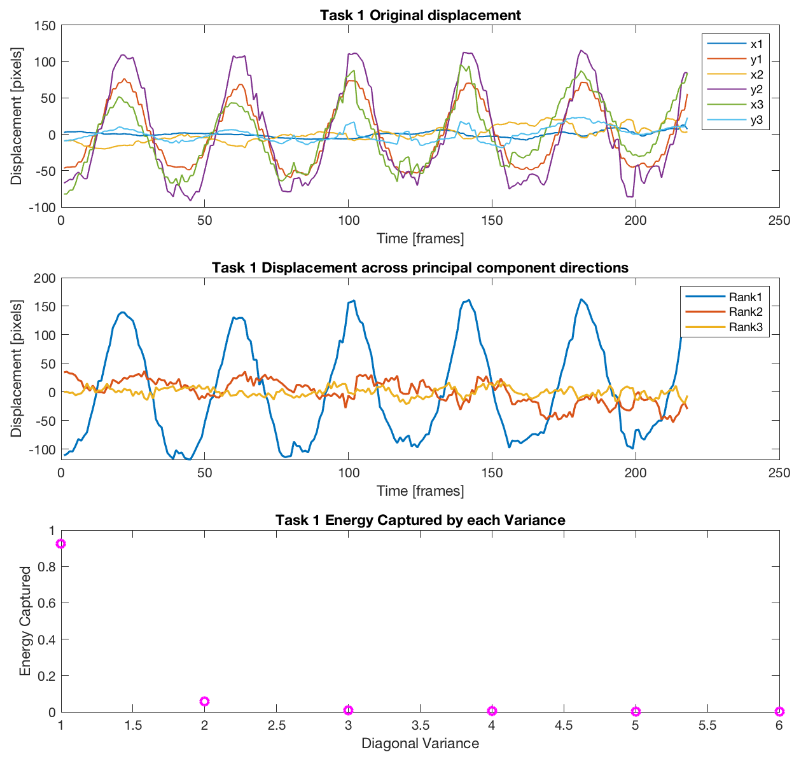

Figure 1: 3 plots for Task 1 (see titles for details).

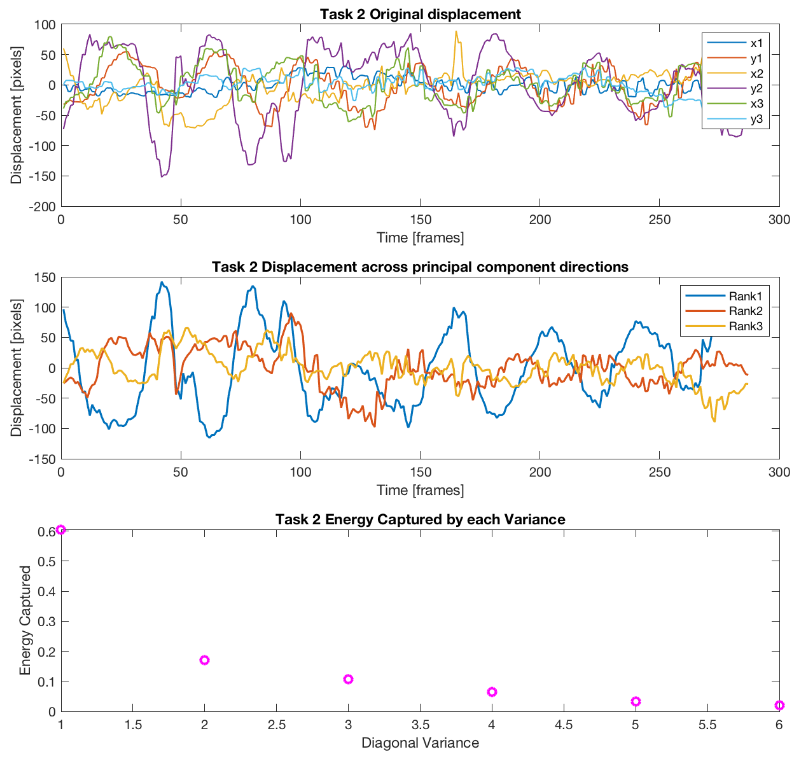

Figure 2: 3 plots for Task 2 (see titles for details).

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.

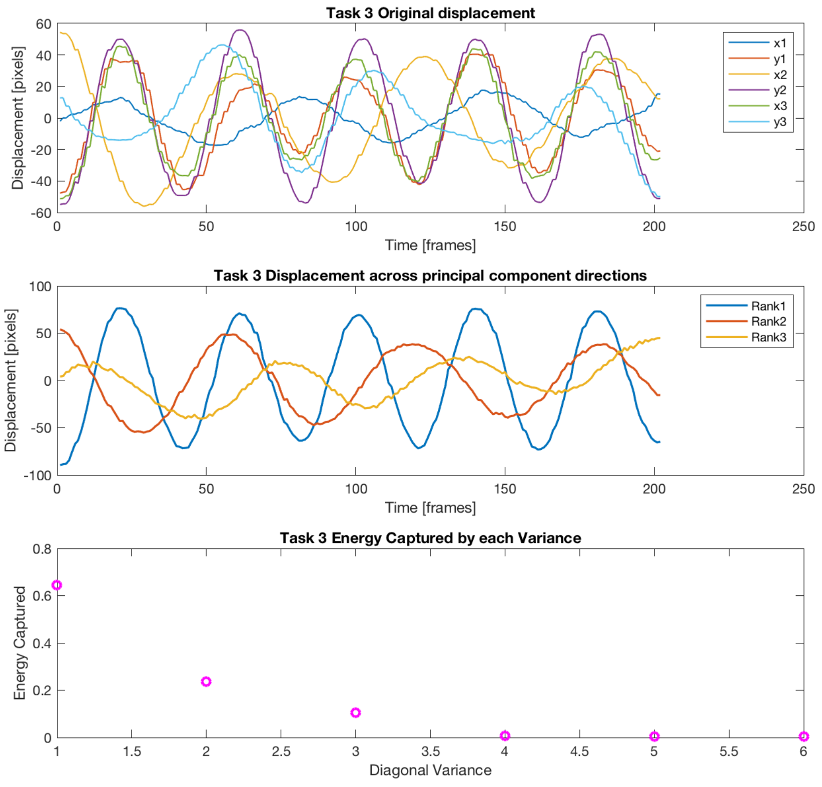

Figure 3: 3 plots for Task 3 (see titles for details).

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.

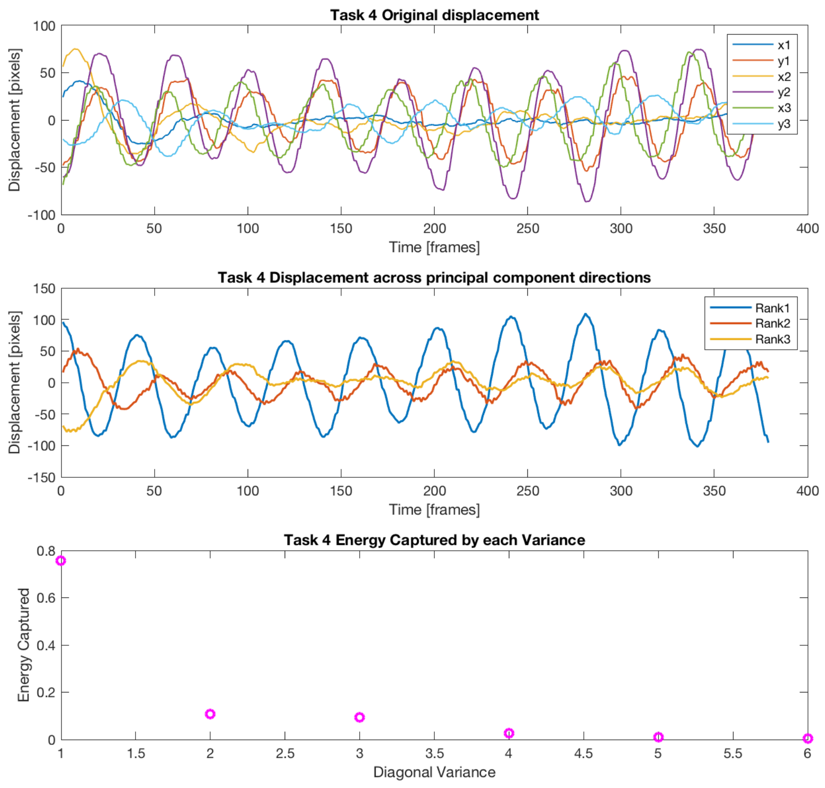

Figure 4: 3 plots for Task 4 (see titles for details).

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

```matlab
%% T1
%% T1
clear; close all; clc;
load('cam1_1.mat')
load('cam2_1.mat')
load('cam3_1.mat')
framenum1 =size(vidFrames1_1,4);
framenum2 =size(vidFrames2_1,4);
framenum3 =size(vidFrames3_1,4);
raw1 = zeros(framenum1,2);
raw2 = zeros(framenum2,2);
raw3 = zeros(framenum3,2);

%%
for j = 1:framenum1
    X1 = imcrop(vidFrames1_1(:,:,:,j),[310,220,50,200]);
    X1 = im2double(rgb2gray(X1));
    %imshow(X1); drawnow
    if max(X1,[],'all')<0.988
```

```
        max(X1,[],'all')
    end
    [Y, X] = find(X1≥0.988);
    raw1(j,:) = [mean(X), mean(Y)];
end
for j = 1:framenum2
    X2 = imcrop(vidFrames2_1(:,:,:,j),[260,70,65,300]);
    X2 = im2double(rgb2gray(X2));
    %imshow(X2); drawnow
    if max(X2,[],'all')<0.988
        max(X2,[],'all')
    end
    [Y, X] = find(X2≥0.988);
    raw2(j,:) = [mean(X), mean(Y)];
end
for j = 1:framenum3
    X3 = imcrop(vidFrames3_1(:,:,:,j),[260,230,230,70]);
    X3 = im2double(rgb2gray(X3));
    %imshow(X3); drawnow
    if max(X3,[],'all')<0.97
        max(X3,[],'all')
    end
    [Y, X] = find(X3≥0.97);
    raw3(j,:) = [mean(X),mean(Y)];
end
% % find where to trim
% subplot(311)
% plot(1:length(raw1),raw1(:,2))
% subplot(312)
% plot(1:length(raw2),raw2(:,2))
% subplot(313)
% plot(1:length(raw3),raw3(:,1))
%%
[M,I] = min(raw1(1:20,2));
trimmed1  = raw1(I:end,:);
[M,I] = min(raw2(1:20,2));
trimmed2  = raw2(I:end,:);
[M,I] = min(raw3(1:20,1));
trimmed3  = raw3(I:end,:);
trimmedtime=min([length(trimmed1),length(trimmed2),length(trimmed3)]);
data1 = trimmed1(1:trimmedtime, :);
data2 = trimmed2(1:trimmedtime, :);
data3 = trimmed3(1:trimmedtime, :);
pltdata = [data1';data2';data3'];
[m,n]=size(pltdata); % compute data size
mn=mean(pltdata,2); % compute mean for each row
pltdata=pltdata-repmat(mn,1,n); % subtract mean
[U,S,V]=svd(pltdata/sqrt(n-1)); % perform the SVD
lambda=diag(S).^2; % produce diagonal variances
Y= U' * pltdata; % produce the principal components projection

%%
figure()
subplot(311)
plot(1:n, pltdata(1,:),'Linewidth',1)
```

```
hold on
plot(1:n, pltdata(2,:),'Linewidth',1)
plot(1:n, pltdata(3,:),'Linewidth',1)
plot(1:n, pltdata(4,:),'Linewidth',1)
plot(1:n, pltdata(5,:),'Linewidth',1)
plot(1:n, pltdata(6,:),'Linewidth',1)
ylabel("Displacement [pixels]"); xlabel("Time [frames]");
title("Task 1 Original displacement");
legend('x1','y1','x2','y2','x3','y3')
subplot(312)
plot(1:trimmedtime, Y(1,:),'Linewidth',1.5)
hold on
plot(1:trimmedtime, Y(2,:),'Linewidth',1.5)
plot(1:trimmedtime, Y(3,:),'Linewidth',1.5)
ylabel("Displacement [pixels]"); xlabel("Time [frames]");
title('Task 1 Displacement across principal component directions');
legend('Rank1','Rank2','Rank3')
subplot(313)
plot(1:6, lambda/sum(lambda), 'mo', 'Linewidth', 2);
title('Task 1 Energy Captured by each Variance');
xlabel('Diagonal Variance'); ylabel('Energy Captured');
set(gcf,'Position',[0,0,700,700])
print('Task1','-dpng')


%% T2
clear; close all; clc;
load('cam1_2.mat')
load('cam2_2.mat')
load('cam3_2.mat')
framenum1 =size(vidFrames1_2,4);
framenum2 =size(vidFrames2_2,4);
framenum3 =size(vidFrames3_2,4);
raw1 = zeros(framenum1,2);
raw2 = zeros(framenum2,2);
raw3 = zeros(framenum3,2);

%%
for j = 1:framenum1
    X1 = imcrop(vidFrames1_2(:,:,:,j),[320,220,70,170]);
    X1 = im2double(rgb2gray(X1));
    %imshow(X1); drawnow
    if max(X1,[],'all')<0.98
        max(X1,[],'all')
    end
    [Y, X] = find(X1≥0.98);
    raw1(j,:) = [mean(X), mean(Y)];
end
for j = 1:framenum2
    X2 = imcrop(vidFrames2_2(:,:,:,j),[220,65,175,295]);
    X2 = im2double(rgb2gray(X2));
    %imshow(X2); drawnow
    if max(X2,[],'all')<0.975
        max(X2,[],'all')
    end
    [Y, X] = find(X2≥0.975);
    raw2(j,:) = [mean(X), mean(Y)];
end
for j = 1:framenum3
    X3 = imcrop(vidFrames3_2(:,:,:,j),[260,200,230,100]);
    X3 = im2double(rgb2gray(X3));
    %imshow(X3); drawnow
    if max(X3,[],'all')<0.964
        max(X3,[],'all')
    end
    [Y, X] = find(X3≥0.964);
    raw3(j,:) = [mean(X),mean(Y)];
end
% % find where to trim
% subplot(311)
% plot(1:length(raw1),raw1(:,2))
% subplot(312)
% plot(1:length(raw2),raw2(:,2))
% subplot(313)
% plot(1:length(raw3),raw3(:,1))

%%
[M,I] = min(raw1(1:40,2));
trimmed1  = raw1(I:end,:);
[M,I] = min(raw2(1:40,2));
trimmed2  = raw2(I:end,:);
[M,I] = min(raw3(1:40,1));
trimmed3  = raw3(I:end,:);
trimmedtime=min([length(trimmed1),length(trimmed2),length(trimmed3)]);
data1 = trimmed1(1:trimmedtime, :);
data2 = trimmed2(1:trimmedtime, :);
data3 = trimmed3(1:trimmedtime, :);
pltdata = [data1';data2';data3'];
[m,n]=size(pltdata); % compute data size
mn=mean(pltdata,2); % compute mean for each row
pltdata=pltdata-repmat(mn,1,n); % subtract mean
[U,S,V]=svd(pltdata/sqrt(n-1)); % perform the SVD
lambda=diag(S).^2; % produce diagonal variances
Y= U'*pltdata; % produce the principal components projection

%%
figure()
subplot(311)
plot(1:trimmedtime, pltdata(1,:),'Linewidth',1)
hold on
plot(1:trimmedtime, pltdata(2,:),'Linewidth',1)
hold on
plot(1:trimmedtime, pltdata(3,:),'Linewidth',1)
plot(1:trimmedtime, pltdata(4,:),'Linewidth',1)
plot(1:trimmedtime, pltdata(5,:),'Linewidth',1)
plot(1:trimmedtime, pltdata(6,:),'Linewidth',1)
ylabel("Displacement [pixels]"); xlabel("Time [frames]");
title("Task 2 Original displacement");
legend('x1','y1','x2','y2','x3','y3')
subplot(312)
```

```
plot(1:trimmedtime, Y(1,:),'Linewidth',1.5)
hold on
plot(1:trimmedtime, Y(2,:),'Linewidth',1.5)
plot(1:trimmedtime, Y(3,:),'Linewidth',1.5)
ylabel("Displacement [pixels]"); xlabel("Time [frames]");
title('Task 2 Displacement across principal component directions');
legend('Rank1','Rank2','Rank3')
subplot(313)
plot(1:6, lambda/sum(lambda), 'mo', 'Linewidth', 2);
title('Task 2 Energy Captured by each Variance');
xlabel('Diagonal Variance'); ylabel('Energy Captured');
set(gcf,'Position',[0,0,700,700])
print('Task2','-dpng')


%% T3
clear all; close all; clc;
load('cam1_3.mat')
load('cam2_3.mat')
load('cam3_3.mat')
framenum1 =size(vidFrames1_3,4);
framenum2 =size(vidFrames2_3,4);
framenum3 =size(vidFrames3_3,4);
raw1 = zeros(framenum1,2);
raw2 = zeros(framenum2,2);
raw3 = zeros(framenum3,2);

%%
for j = 1:framenum1
    X1 = imcrop(vidFrames1_3(:,:,:,j),[280,220,110,170]);
    X1 = im2double(rgb2gray(X1));
    %imshow(X1); drawnow
    if max(X1,[],'all')<0.98
        max(X1,[],'all')
    end
    [Y, X] = find(X1≥0.98);
    raw1(j,:) = [mean(X), mean(Y)];
end
for j = 1:framenum2
    X2 = imcrop(vidFrames2_3(:,:,:,j),[230,180,175,205]);
    X2 = im2double(rgb2gray(X2));
    %imshow(X2); drawnow
    if max(X2,[],'all')<0.975
        max(X2,[],'all')
    end
    [Y, X] = find(X2≥0.975);
    raw2(j,:) = [mean(X), mean(Y)];
end
for j = 1:framenum3
    X3 = imcrop(vidFrames3_3(:,:,:,j),[260,180,210,150]);
    X3 = im2double(rgb2gray(X3));
    %imshow(X3); drawnow
    if max(X3,[],'all')<0.964
        max(X3,[],'all')
    end
    [Y, X] = find(X3≥0.964);
    raw3(j,:) = [mean(X),mean(Y)];
end
% % find where to trim
% subplot(311)
% plot(1:length(raw1),raw1(:,2))
% subplot(312)
% plot(1:length(raw2),raw2(:,2))
% subplot(313)
% plot(1:length(raw3),raw3(:,1))

%%
[M,I] = min(raw1(1:40,2));
trimmed1  = raw1(I:end,:);
[M,I] = min(raw2(1:40,2));
trimmed2  = raw2(I:end,:);
[M,I] = min(raw3(1:40,1));
trimmed3  = raw3(I:end,:);
trimmedtime=min([length(trimmed1),length(trimmed2),length(trimmed3)]);
data1 = trimmed1(1:trimmedtime, :);
data2 = trimmed2(1:trimmedtime, :);
data3 = trimmed3(1:trimmedtime, :);
pltdata = [data1';data2';data3'];
[m,n]=size(pltdata); % compute data size
mn=mean(pltdata,2); % compute mean for each row
pltdata=pltdata-repmat(mn,1,n); % subtract mean
[U,S,V]=svd(pltdata/sqrt(n-1)); % perform the SVD
lambda=diag(S).^2; % produce diagonal variances
Y= U'*pltdata; % produce the principal components projection

%%
figure()
subplot(311)
plot(1:trimmedtime, pltdata(1,:),'Linewidth',1)
hold on
plot(1:trimmedtime, pltdata(2,:),'Linewidth',1)
plot(1:trimmedtime, pltdata(3,:),'Linewidth',1)
plot(1:trimmedtime, pltdata(4,:),'Linewidth',1)
plot(1:trimmedtime, pltdata(5,:),'Linewidth',1)
plot(1:trimmedtime, pltdata(6,:),'Linewidth',1)
ylabel("Displacement [pixels]"); xlabel("Time [frames]");
title("Task 3 Original displacement");
legend('x1','y1','x2','y2','x3','y3')
subplot(312)
plot(1:trimmedtime, Y(1,:),'Linewidth',1.5)
hold on
plot(1:trimmedtime, Y(2,:),'Linewidth',1.5)
plot(1:trimmedtime, Y(3,:),'Linewidth',1.5)
ylabel("Displacement [pixels]"); xlabel("Time [frames]");
title('Task 3 Displacement across principal component directions');
legend('Rank1','Rank2','Rank3')
subplot(313)
plot(1:6, lambda/sum(lambda), 'mo', 'Linewidth', 2);
title('Task 3 Energy Captured by each Variance');
xlabel('Diagonal Variance'); ylabel('Energy Captured');
```

```
set(gcf,'Position',[0,0,700,700])
print('Task3','-dpng')


%% T4
clear; close; clc;
load('cam1_4.mat')
load('cam2_4.mat')
load('cam3_4.mat')
framenum1 =size(vidFrames1_4,4);
framenum2 =size(vidFrames2_4,4);
framenum3 =size(vidFrames3_4,4);
raw1 = zeros(framenum1,2);
raw2 = zeros(framenum2,2);
raw3 = zeros(framenum3,2);

%%
for j = 1:framenum1
    X1 = imcrop(vidFrames1_4(:,:,:,j),[330,225,110,165]);
    X1 = im2double(rgb2gray(X1));
    %imshow(X1); drawnow
    if max(X1,[],'all')<0.9685
        max(X1,[],'all')
    end
    [Y, X] = find(X1≥0.9685);
    raw1(j,:) = [mean(X), mean(Y)];
end
for j = 1:framenum2
    X2 = imcrop(vidFrames2_4(:,:,:,j),[220,100,210,235]);
    X2 = im2double(rgb2gray(X2));
    %imshow(X2); drawnow
    if max(X2,[],'all')<0.976
        max(X2,[],'all')
    end
    [Y, X] = find(X2≥ 0.976);
    raw2(j,:) = [mean(X), mean(Y)];
end
for j = 1:framenum3
    X3 = imcrop(vidFrames3_4(:,:,:,j),[310,130,205,150]);
    X3 = im2double(rgb2gray(X3));
    %imshow(X3); drawnow
    if max(X3,[],'all')<0.917
        max(X3,[],'all')
    end
    [Y, X] = find(X3≥0.917);
    raw3(j,:) = [mean(X),mean(Y)];
end
% % find where to trim
% subplot(311)
% plot(1:length(raw1),raw1(:,2))
% subplot(312)
% plot(1:length(raw2),raw2(:,2))
% subplot(313)
% plot(1:length(raw3),raw3(:,1))

```

```
%%
[M,I] = min(raw1(1:25,2));
trimmed1  = raw1(I:end,:);
[M,I] = min(raw2(1:25,2));
trimmed2  = raw2(I:end,:);
[M,I] = min(raw3(1:25,1));
trimmed3  = raw3(I:end,:);
trimmedtime=min([length(trimmed1),length(trimmed2),length(trimmed3)]);
data1 = trimmed1(1:trimmedtime, :);
data2 = trimmed2(1:trimmedtime, :);
data3 = trimmed3(1:trimmedtime, :);
pltdata = [data1';data2';data3'];
[m,n]=size(pltdata); % compute data size
mn=mean(pltdata,2); % compute mean for each row
pltdata=pltdata-repmat(mn,1,n); % subtract mean
[U,S,V]=svd(pltdata/sqrt(n-1)); % perform the SVD
lambda=diag(S).^2; % produce diagonal variances
Y= U'*pltdata; % produce the principal components projection

%%
figure()
subplot(311)
plot(1:trimmedtime, pltdata(1,:),'Linewidth',1)
hold on
plot(1:trimmedtime, pltdata(2,:),'Linewidth',1)
plot(1:trimmedtime, pltdata(3,:),'Linewidth',1)
plot(1:trimmedtime, pltdata(4,:),'Linewidth',1)
plot(1:trimmedtime, pltdata(5,:),'Linewidth',1)
plot(1:trimmedtime, pltdata(6,:),'Linewidth',1)
ylabel("Displacement [pixels]"); xlabel("Time [frames]");
title("Task 4 Original displacement");
legend('x1','y1','x2','y2','x3','y3')
subplot(312)
plot(1:trimmedtime, Y(1,:),'Linewidth',1.5)
hold on
plot(1:trimmedtime, Y(2,:),'Linewidth',1.5)
plot(1:trimmedtime, Y(3,:),'Linewidth',1.5)
ylabel("Displacement [pixels]"); xlabel("Time [frames]");
title('Task 4 Displacement across principal component directions');
legend('Rank1','Rank2','Rank3')
subplot(313)
plot(1:6, lambda/sum(lambda), 'mo', 'Linewidth', 2);
title('Task 4 Energy Captured by each Variance');
xlabel('Diagonal Variance'); ylabel('Energy Captured');
set(gcf,'Position',[0,0,700,700])
print('Task4','-dpng')
```