Principal Component Analysis

Khoa Vo

February 22, 2020

Abstract

This project will discuss about one of key application of Singular Value Decomposition, Principal Component Analysis and how it will be used to extract information from a large set of data. How noise affect the result of PCA will also be mentioned in this project.

1 Introduction and Overview

We want to explore PCA's practical usefulness and how noise can affect PCA. We perform PCA on the movement of a paint can in 4 different cases: Ideal, Noisy, Horizontal Displacement, and Horizontal Displacement and Rotation. Three cameras at different angles are used to capture the location of the paint can. First, we need to extract data of how the paint can moves in each video and try to sync the starting frame of 3 videos in each case. Also, we need to perform SVD to get the singular values and principal component vectors. Then, we want to see how much energy each component captures to see how the data is redundant.

2 Theoretical Background

2.1 Singular Value Decomposition (SVD)

Singular Value Decomposition is a factorization of a matrix into a number of constitutive components all of which have a specific meaning in application. Specifically, a $m \times n$ matrix $\mathbf{A}$ has Singular Value Decomposition in form of $\mathbf{U\Sigma V^*}$, where $\mathbf{U}$ is a $m \times m$ unitary, $\mathbf{\Sigma}$ is $m \times n$ diagonal matrix with non-negative real numbers σ_i , and $\mathbf{V}$ is a $n \times n$ unitary matrix, i.e., SVD decomposes matrix $\mathbf{A}$ into three transformation: an initial rotation $\mathbf{V}^*$, a scaling $\mathbf{\Sigma}$, and a final rotation $\mathbf{U}$. The singular values σ_i are uniquely determined, and, if matrix $\mathbf{A}$ is square and the σ_i distinct, the singular vectors u_j and v_j are uniquely determined up to complex signs. The SVD can be computed as the following:

$$\begin{aligned}\mathbf{A}^T\mathbf{A} &= (\mathbf{U\Sigma V^*})^T(\mathbf{U\Sigma V^*}) \\ &= \mathbf{V\Sigma U^*U\Sigma V^*} \\ &= \mathbf{V\Sigma^2 V^*} \end{aligned} \tag{1}$$

and

$$\begin{aligned}\mathbf{A}\mathbf{A}^T &= (\mathbf{U}\mathbf{\Sigma}\mathbf{V}^*)\,(\mathbf{U}\mathbf{\Sigma}\mathbf{V}^*)^T \\ &= \mathbf{U}\mathbf{\Sigma}\mathbf{V}^*\mathbf{V}\mathbf{\Sigma}\mathbf{U}^* \\ &= \mathbf{U}\mathbf{\Sigma}^2\mathbf{U}^* \qquad (2)\end{aligned}$$

Multiplying (1) and (2) by $\mathbf{V}$ and $\mathbf{U}$ respectively gives two self-consistent eigenvalue problems.

$$\mathbf{A}^T\mathbf{A}\mathbf{V} = \mathbf{V}\mathbf{\Sigma}^2 \qquad (3)$$

$$\mathbf{A}\mathbf{A}^T\mathbf{U} = \mathbf{U}\mathbf{\Sigma}^2 \qquad (4)$$

If eigenvectors are found, then the orthonormal basis vectors are produced for $\mathbf{U}$ and $\mathbf{V}$. Also, SVD diagonalizes and each singular direction captures highest energy possible. The percentage of energy captured by σ_i can be calculated by:

$$E_i = \frac{{\sigma_i}^2}{{\sigma_1}^2 + {\sigma_2}^2 + ... + {\sigma_r}^2}, \text{ with } r \text{ is the rank of matrix } \mathbf{A} \qquad (5)$$

2.2 Principal Component Analysis (PCA)

One of key applications of Singular Value Decomposition (SVD is Principal Component Analysis (PCA). After the data is collected, two issues might arise: noise and redundancy. Firstly, measurement noise in any data set must be low or else, no information about the system can be extracted, regardless of analysis techniques. The key measurement of this is the so-called signal-to-noise ratio: $SNR = \sigma^2_{signal}/\sigma^2_{noise}$. A high SNR indicates high precision data whereas low SNR suggest noise contaminated data. Secondly, redundancy can happen when the single degree of freedom of data is sampled by more than one camera, these measurements are dependent to each other. To perform a meaningful data analysis, it is crucial to remove this redundancy.

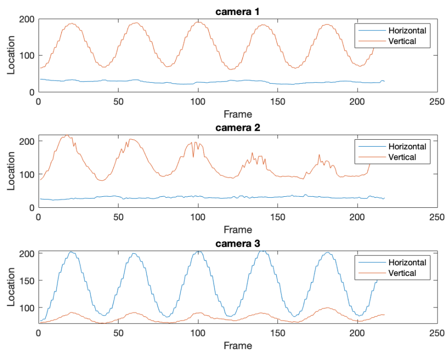
An easy way to identify redundant data is considering the covariance matrix:

$$\mathbf{C_X} = \frac{1}{n-1}\mathbf{A}\mathbf{A}^T \qquad (6)$$

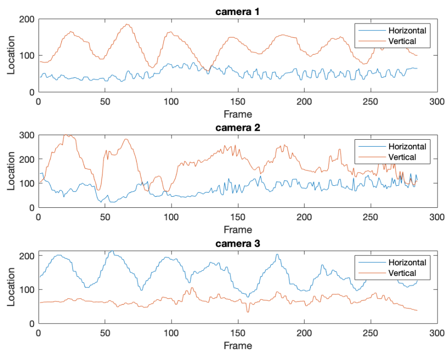
$\mathbf{C_X}$ captures the correlations between data sets. Thus, we can captured whether two data sets are redundant$\mathbf{C_X}$. We want the diagnonals of $\mathbf{C_X}$ are ordered from largest to smallest and off-diagnonals are zeros, this is similar to what SVD does.

3 Algorithm Implementation and Development

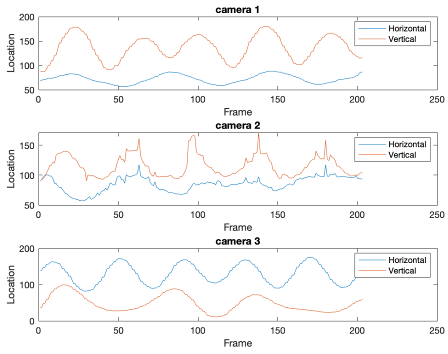
We started by filtering the frames of each video manually. The filtered frames should only contain the vertical and horizontal range of the paint can. Then, we take the advantage of the flashlight on top the paint can to track its movement. In order to do that, we convert the each frame into black and white, by using "rgb2gray", and covert the data into double to perform computation later, by using "double". Then, we find the location of the flash light, which is supposed to have the highest energy in each frame. We can achieve that by finding the index that has highest value. To increase the precision, we can take all the indices that have value



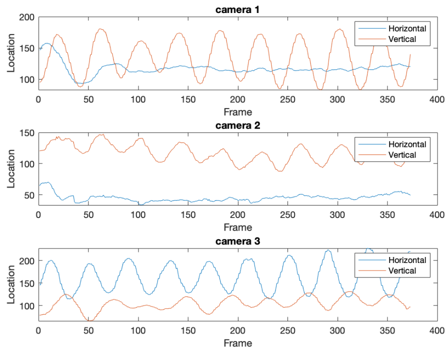
(a) Ideal Case



(b) Noisy Case



(c) Horizontal displacement



(d) Horizontal displacement and rotation

Figure 1: Vertical and Horizontal Positions of Paint Can in 4 Different cases

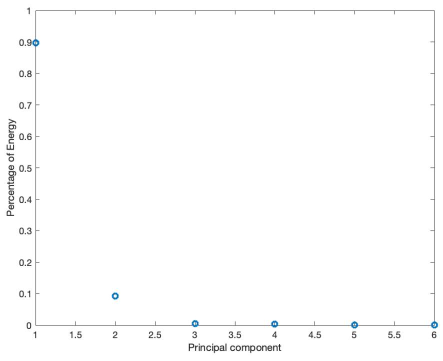
equal to 9/10 of maximum value in each frame and take the mean.

Notice each video has different number of frames and different starting point of the paint can. We want to arrange the data sets so that they have the same dimensions and sync in starting point. In order to that, we locate the paint can at its lowest point (vertical axis) in each video of each case, by finding which frame out of 30 first frames has lowest vertical value. The vertical values of that frame will be the starting point. Then, we trim two data sets which have more frames to have same size as the other one.

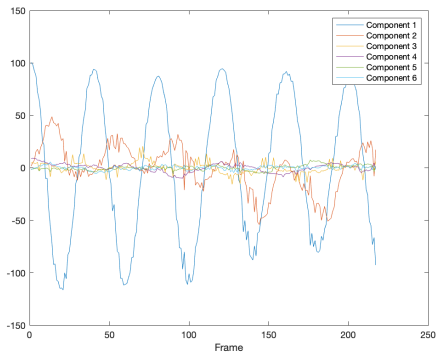
Before performing SVD and exploring PCA, we need to center our data by subtracting the mean of each row. For an unbiased estimator, we apply a normalization constant of $1/\sqrt{n-1}$ to our data. Then, we perform SVD by using "svd" to determine matrices $\mathbf{U}, \mathbf{\Sigma}$, and $\mathbf{V}$. The diagonal variances will be the matrix $\mathbf{\Sigma}^2$. We now can calculate how much energy each singular value captures by using equation (5). Finally, we produce the principal components projection by computing $\mathbf{U}^T$ times the data ($\mathbf{\Sigma}*\mathbf{V}^T$ yields the same result).

4 Computational Results

From Figure 1, we can see that the vertical and horizontal movement of the paint are synced in 3 cameras in 4 different cases.

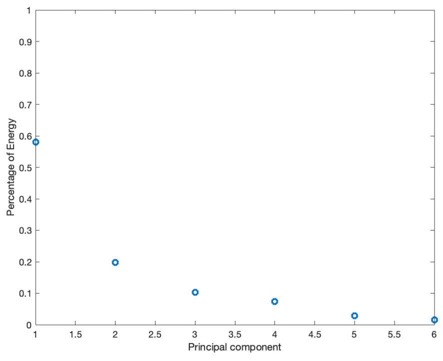


(a) Energy Percentage

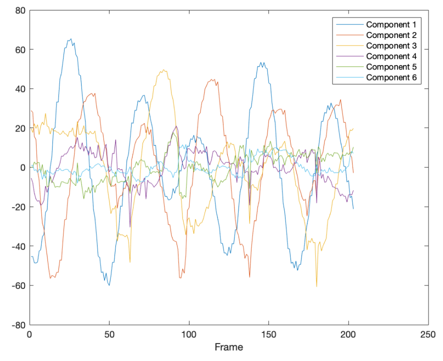


(b) Principal Component Projection on new basis

Figure 2: Energy Percentage and Principal Component Projection of Ideal case



(a) Energy Percentage



(b) Principal Component Projection on new basis

Figure 3: Energy Percentage and Principal Component Projection of Noisy case

4.1 Ideal Case

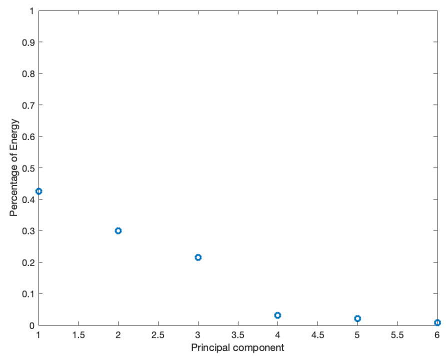
Figure 2.(a) shows that the first principal component captures more than 90% energy of the whole system. Figure 2.(b) also suggests the projection of data onto the first component varies the most. This is because the paint can in this case mostly move in vertical direction, without much noise, all of its motion is captured by the first component.

4.2 Noisy Case

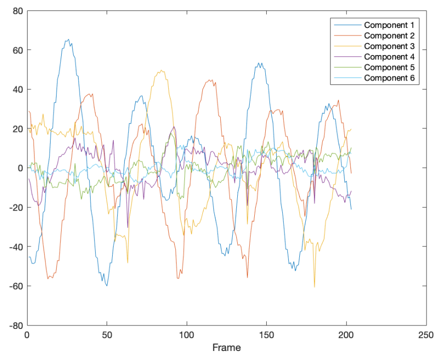
After adding noise to the system, we can see that the energy captured, less than 60% by the first component is not as high as the first case, as seen in Figure 3.(a). This implies that lots of information will be lost if we do not consider more principal components. To capture 90%, we need at least the first 3 principal components. Figure 3.(b) also suggests that the motion of paint can is mostly depended on the first 3 components.

4.3 Horizontal displacement

Figure 4 (a) shows that first 3 component capture most energy in this case. Even though we try to move the paint can in horizontal direct, due to gravity, the paint can in this case has motion similar to a pendulum. That's why we need more than one component to capture most of its energy. Figure 4 (b) suggests the same thing and we can see that the projection on component 5 and 6 do not vary much.

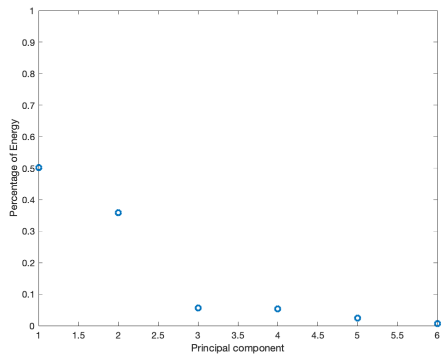


(a) Energy Percentage

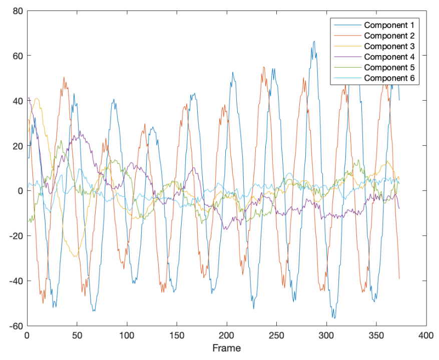


(b) Principal Component Projection on new basis

Figure 4: Energy Percentage and Principal Component Projection of Horizontal displacement case



(a) Energy Percentage



(b) Principal Component Projection on new basis

Figure 5: Energy Percentage and Principal Component Projection of Horizontal displacement and rotation case

4.4 Horizontal displacement and rotation case

In this case, the paint can also has a motion as a pendulum. We need 3 components to capture a good amount of energy for this system. The first and second component are crucial since they captured the a close number of energy. Figure 5 (b) shows that component 1 and component 2 varies almost the same.

5 Summary and Conclusions

In order to perform SVD and explore PCA, we need to extract data and sync them in time (frames). Performing PCA on the data gives us information about how much energy principal component capture. We can see that in the first case, the system is dominant by component 1, or in other cases, is dominant by first 3 components. With that information, we can reduce the dimension of our data sets accordingly. In case 1 and case 2, we know that the movement of the paint can in real life is the same, however, adding noise to case 2 significantly change PCA result.

Appendix A

rgb2gray(): converts the true color image to the grayscale image.
find(): first indices corresponding to the conditions.
svd(): performs a singular value decomposition of matrix.
diag(): return a column vector of the diagonal values of the matrix.

Appendix B

```
%%case1
clear all; close all; clc
load('cam1_1.mat')
load('cam2_1.mat')
load('cam3_1.mat')

for i = 1:size(vidFrames1_1, 4)
    X = double(rgb2gray(vidFrames1_1(:,:,:,i)));
    X_filter = X(200:480,300:400);
    flash = max(X_filter(:));
    [x,y] = find(X_filter >= flash*9/10);
    x1(i)=mean(y);
    y1(i)=mean(x);
end

for i = 1:size(vidFrames2_1, 4)
    X = double(rgb2gray(vidFrames2_1(:,:,:,i)));
    X_filter = X(100:380,250:350);
    flash = max(X_filter(:));
    [x,y] = find(X_filter >= flash*9/10);
    x2(i)=mean(y);
    y2(i)=mean(x);
end

for i = 1:size(vidFrames3_1, 4)
    X = double(rgb2gray(vidFrames3_1(:,:,:,i)));
    X_filter = X(200:350, 250:500);
    %imshow(X_filter);
    [x,y] = find(X_filter >= flash*9/10);
    x3(i)=mean(y);
    y3(i)=mean(x);
end

index1 = find(y1(1:30) == min(y1(1:30)));
index2 = find(y2(1:30) == min(y2(1:30)));
index3 = find(y3(1:30) == min(y3(1:30)));
x1_shift = x1(index1:end);
y1_shift = y1(index1:end);
x2_shift = x2(index2:end);
y2_shift = y2(index2:end);
x3_shift = x3(index3:end);
y3_shift = y3(index3:end);
```

```
figure()

min_frame = min([max(size(x1_shift)) max(size(x2_shift)) max(size(x3_shift))]);
t=1:min_frame;
subplot(3,1,1)
plot(t,x1_shift(1:min_frame),t,y1_shift(1:min_frame))
title('camera 1')
xlabel('Frame'); ylabel('Location')
legend('Horizontal','Vertical','Location','NorthEast')
subplot(3,1,2)
plot(t,x2_shift(1:min_frame),t,y2_shift(1:min_frame))
xlabel('Frame'); ylabel('Location')
legend('Horizontal','Vertical','Location','NorthEast')
title('camera 2')
subplot(3,1,3)
plot(t,x3_shift(1:min_frame),t,y3_shift(1:min_frame))
xlabel('Frame'); ylabel('Location')
legend('Horizontal','Vertical','Location','NorthEast')
title('camera 3')


data = [x1_shift(1:min_frame); y1_shift(1:min_frame); x2_shift(1:min_frame);
    y2_shift(1:min_frame); x3_shift(1:min_frame); y3_shift(1:min_frame)];
[m, n] = size(data);
mn = mean(data, 2);
data = data-repmat(mn,1,n);
[u, s, v] = svd(data/sqrt(n-1), 'econ');
figure(2)
plot(diag(s).^2/sum(diag(s.^2)), 'o', 'Linewidth', 2)
xlabel('Principal component'); ylabel('Percentage of Energy');
ylim([0 1])
Y=u'*data; % produce the principal components projection
figure(3)
plot(Y(1,:)); hold on
plot(Y(2,:))
plot(Y(3,:))
plot(Y(4,:))
plot(Y(5,:))
plot(Y(6,:))
legend('Component 1','Component 2','Component 3','Component 4','Component
    5','Component 6','Location','NorthEast')
xlabel('Frame');

%% case 2
clear all; close all; clc
load('cam1_2.mat')
load('cam2_2.mat')
load('cam3_2.mat')

for i = 1:size(vidFrames1_2, 4)
    X = double(rgb2gray(vidFrames1_2(:,:,:,i)));
    X_filter = X(200:480,300:400);
    flash = max(X_filter(:));
    [x,y] = find(X_filter >= flash*9/10);
    x1(i)=mean(y);
    y1(i)=mean(x);
end

for i = 1:size(vidFrames2_2, 4)
    X = double(rgb2gray(vidFrames2_2(:,:,:,i)));
    X_filter = X(50:480,200:400);
    flash = max(X_filter(:));
    [x,y] = find(X_filter >= flash*9/10);
    x2(i)=mean(y);
    y2(i)=mean(x);
end

for i = 1:size(vidFrames3_2, 4)
    X = double(rgb2gray(vidFrames3_2(:,:,:,i)));
    X_filter = X(200:350, 250:500);
    flash = max(X_filter(:));
    [x,y] = find(X_filter >= flash*9/10);
    x3(i)=mean(y);
    y3(i)=mean(x);
end

index1 = find(y1(1:30) == min(y1(1:30)));
index2 = find(y2(1:30) == min(y2(1:30)));
index3 = find(y3(1:30) == min(y3(1:30)));
x1_shift = x1(index1:end);
y1_shift = y1(index1:end);
x2_shift = x2(index2:end);
y2_shift = y2(index2:end);
x3_shift = x3(index3:end);
y3_shift = y3(index3:end);

min_frame = min([max(size(x1_shift)) max(size(x2_shift)) max(size(x3_shift))]);
t=1:min_frame;
subplot(3,1,1)
plot(t,x1_shift(1:min_frame),t,y1_shift(1:min_frame))
xlabel('Frame'); ylabel('Location')
legend('Horizontal','Vertical','Location','NorthEast')
title('camera 1')

subplot(3,1,2)
plot(t,x2_shift(1:min_frame),t,y2_shift(1:min_frame))
xlabel('Frame'); ylabel('Location')
legend('Horizontal','Vertical','Location','NorthEast')
title('camera 2')

subplot(3,1,3)
plot(t,x3_shift(1:min_frame),t,y3_shift(1:min_frame))
xlabel('Frame'); ylabel('Location')
legend('Horizontal','Vertical','Location','NorthEast')
title('camera 3')
```

```
data = [x1_shift(1:min_frame); y1_shift(1:min_frame); x2_shift(1:min_frame);
    y2_shift(1:min_frame); x3_shift(1:min_frame); y3_shift(1:min_frame)];
[m, n] = size(data);
mn = mean(data, 2);
data = data-repmat(mn,1,n);
[u, s, v] = svd(data/sqrt(n-1), 'econ');
figure(2)
plot(diag(s).^2/sum(diag(s.^2)), 'o', 'Linewidth', 2)
xlabel('Principal component'); ylabel('Percentage of Energy');
ylim([0 1])
Y=u'*data; % produce the principal components projection
figure(3)
plot(Y(1,:)); hold on
plot(Y(2,:))
plot(Y(3,:))
plot(Y(4,:))
plot(Y(5,:))
plot(Y(6,:))
legend('Component 1','Component 2','Component 3','Component 4','Component
    5','Component 6','Location','NorthEast')
xlabel('Frame');

%%case3
clear all; close all; clc
load('cam1_3.mat')
load('cam2_3.mat')
load('cam3_3.mat')
implay(vidFrames1_3)
for i  = 1:size(vidFrames1_3, 4)
    X = double(rgb2gray(vidFrames1_3(:,:,:,i)));
    X_filter = X(200:450,250:400);
    flash = max(X_filter(:));
    [x,y] = find(X_filter >= flash*9/10);
    x1(i)=mean(y);
    y1(i)=mean(x);
end

for i  = 1:size(vidFrames2_3, 4)
    X = double(rgb2gray(vidFrames2_3(:,:,:,i)));
    X_filter = X(170:380,200:350);
    flash = max(X_filter(:));
    [x,y] = find(X_filter >= flash*9/10);
    x2(i)=mean(y);
    y2(i)=mean(x);
end

for i = 1:size(vidFrames3_3, 4)
    X = double(rgb2gray(vidFrames3_3(:,:,:,i)));
    X_filter = X(200:350,250:500);
    flash = max(X_filter(:));
    [x,y] = find(X_filter >= flash*9/10);
    x3(i)=mean(y);
    y3(i)=mean(x);
```

```
end

index1 = find(y1(1:40) == min(y1(1:40)));
index2 = find(y2(1:30) == min(y2(1:30)));
index3 = find(y3(1:30) == min(y3(1:30)));
x1_shift = x1(index1:end);
y1_shift = y1(index1:end);
x2_shift = x2(index2:end);
y2_shift = y2(index2:end);
x3_shift = x3(index3:end);
y3_shift = y3(index3:end);
figure(1)

min_frame = min([max(size(x1_shift.')) max(size(x2_shift.')) max(size(x3_shift))]);
t=1:min_frame;
subplot(3,1,1)
plot(t,x1_shift(1:min_frame),t,y1_shift(1:min_frame))
xlabel('Frame'); ylabel('Location')
legend('Horizontal','Vertical','Location','NorthEast')
title('camera 1')

subplot(3,1,2)
plot(t,x2_shift(1:min_frame),t,y2_shift(1:min_frame))
xlabel('Frame'); ylabel('Location')
legend('Horizontal','Vertical','Location','NorthEast')
title('camera 2')

subplot(3,1,3)
plot(t,x3_shift(1:min_frame),t,y3_shift(1:min_frame))
xlabel('Frame'); ylabel('Location')
legend('Horizontal','Vertical','Location','NorthEast')
title('camera 3')


data = [x1_shift(1:min_frame); y1_shift(1:min_frame); x2_shift(1:min_frame);
   y2_shift(1:min_frame); x3_shift(1:min_frame); y3_shift(1:min_frame)];
[m, n] = size(data);
mn = mean(data, 2);
data = data-repmat(mn,1,n);
[u, s, v] = svd(data/sqrt(n-1), 'econ');
figure(2)
plot(diag(s).^2/sum(diag(s.^2)), 'o', 'Linewidth', 2)
xlabel('Principal component'); ylabel('Percentage of Energy');
ylim([0 1])
Y=u'*data; % produce the principal components projection
figure(3)
plot(Y(1,:)); hold on
plot(Y(2,:))
plot(Y(3,:))
plot(Y(4,:))
plot(Y(5,:))
plot(Y(6,:))
```

```
legend('Component 1','Component 2','Component 3','Component 4','Component
    5','Component 6','Location','NorthEast')
xlabel('Frame');

%%case 4
clear all; close all; clc
load('cam1_4.mat')
load('cam2_4.mat')
load('cam3_4.mat')
for i = 1:size(vidFrames1_4, 4)
    X = double(rgb2gray(vidFrames1_4(:,:,:,i)));
    X_filter = X(200:450,250:420);
    flash = max(X_filter(:));
    [x,y] = find(X_filter >= flash*9/10);
    x1(i)=mean(y);
    y1(i)=mean(x);
end

for i = 1:size(vidFrames2_4, 4)
    X = double(rgb2gray(vidFrames2_4(:,:,:,i)));
    X_filter = X(100:380,200:400);
    flash = max(X_filter(:));
    [x,y] = find(X_filter >= flash*9/10);
    x2(i)=mean(y);
    y2(i)=mean(x);
end

for i = 1:size(vidFrames3_4, 4)
    X = double(rgb2gray(vidFrames3_4(:,:,:,i)));
    X_filter = X(100:300,250:500);
    flash = max(X_filter(:));
    [x,y] = find(X_filter >= flash*9/10);
    x3(i)=mean(y);
    y3(i)=mean(x);
end

index1 = find(y1(1:40) == min(y1(1:40)));
index2 = find(y2(1:30) == min(y2(1:30)));
index3 = find(y3(1:30) == min(y3(1:30)));
x1_shift = x1(index1:end);
y1_shift = y1(index1:end);
x2_shift = x2(index2:end);
y2_shift = y2(index2:end);
x3_shift = x3(index3:end);
y3_shift = y3(index3:end);

figure(1)
min_frame = min([max(size(x1_shift)) max(size(x2_shift)) max(size(x3_shift))]);
t=1:min_frame;
subplot(3,1,1)
plot(t,x1_shift(1:min_frame),t,y1_shift(1:min_frame))
xlabel('Frame'); ylabel('Location')
legend('Horizontal','Vertical','Location','NorthEast')
```

```
title('camera 1')

subplot(3,1,2)
plot(t,x2_shift(1:min_frame),t,y2_shift(1:min_frame))
xlabel('Frame'); ylabel('Location')
legend('Horizontal','Vertical','Location','NorthEast')
title('camera 2')

subplot(3,1,3)
plot(t,x3_shift(1:min_frame),t,y3_shift(1:min_frame))
xlabel('Frame'); ylabel('Location')
legend('Horizontal','Vertical','Location','NorthEast')
title('camera 3')

data = [x1_shift(1:min_frame); y1_shift(1:min_frame); x2_shift(1:min_frame);
    y2_shift(1:min_frame); x3_shift(1:min_frame); y3_shift(1:min_frame)];
[m, n] = size(data);
mn = mean(data, 2);
data = data-repmat(mn,1,n);
[u, s, v] = svd(data/sqrt(n-1), 'econ');
figure(2)
plot(diag(s).^2/sum(diag(s.^2)), 'o', 'Linewidth', 2)
xlabel('Principal component'); ylabel('Percentage of Energy');
ylim([0 1])
Y=u'*data; % produce the principal components projection
figure(3)
plot(Y(1,:)); hold on
plot(Y(2,:))
plot(Y(3,:))
plot(Y(4,:))
plot(Y(5,:))
plot(Y(6,:))
legend('Component 1','Component 2','Component 3','Component 4','Component
    5','Component 6','Location','NorthEast')
xlabel('Frame');
```