AMATH582 HW4 SINGULAR VALUE DECOMPOSITION AND PRINCIPAL COMPONENT ANALYSIS

SHUANG WU

Feb. 19th

Abstract

As the data analysis becomes more and more popular, the size of the raw data becomes larger and larger. Also, the types of the measurement of the data are different for different people. If a big data is presented, we want to know if there exists redundancy in the data. First, the redundancies will lead to more time and other cost. More important, the results of the analysis will change because of the redundancies. In this project, the Single Value Decomposition (SVD) and Principle Component Analysis (PCA) methods will be used to find redundancies along with a simple experiment.

1 INTRODUCTION

First, I need introduce a simple experiment of how to measure the data. In this experiment, a person holds a spring and a can is attached to the bottom of this spring. At the top of the can, there is a light source. If the can is released, it will oscillate up and down. The experiment also set up three cameras in three different positions to recode the movement of the can. The angle between each two cameras is not 90 degree. After the set up, it is clear that if I release the can and open the three cameras, maybe not simultaneously. I can have a bunch of x-y coordinates data from each camera for each unit time, or each frame. There are 4 cases, which are 4 different types of experiments. The first case is the ideal case, which only has small displacements of the mass in the z direction, so the entire motion is in the z direction. The second case is a noisy case. This time, a similar case is repeated like the first case but along with some camera shakes. The third case is the horizontal displacement case. In this case, the can is released off-center to make it not only moves in z direction but also in x and y directions. The last case is the third case along with some rotations for the can.

2 THEORETICAL BACKGROUND

2.1 *Singular Value Decomposition (SVD)*

Before talking about the SVD, I will first introduce the basic idea of matrix transformation.

If I need to do a matrix multiplication like$Ax = y$, the fact is that I rotate the matrix first and then stretch (compress) the matrix, so the result is y.

The rotation matrix:

$$A_r = \begin{bmatrix} cos\theta & -sin\theta \\ sin\theta & cos\theta \end{bmatrix}$$

And the scale matrix:

$$A_s = \begin{bmatrix} \alpha & 0 \\ 0 & \alpha \end{bmatrix}$$

After understand the idea, I can continue talking about the SVD. SVD is the factorization of matrix into a number of constitutive components, and each component has specific meaning in application.

Formula:

$$AV = \hat{U}\hat{\Sigma}$$

For V is an $n * n$ unitary matrix, U is a $m * n$ orthonormal columns, and Σ is a $n * n$ diagonal matrix. Because the V is unitary, I can move the V to the other side to get the reduced singular value decomposition formula:

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

This is named as reduced SVD instead of full SVD, because the size for U is $m * n$. If I make the U unitary, I will have the full SVD. The easiest way is adding $m - n$ columns to U that are orthonormal to the existing set in $\hat{U}$ and also adding $m - n$ rows of zeros to $\hat{\Sigma}$. After these two steps, I will have an unitary U along with the diagonal matrix Σ. The full SVD formula is :

$$A = U\Sigma V^*$$

for $U \in \mathbb{C}^{m*m}$ is unitary, $V \in \mathbb{C}^{n*n}$ is unitary, and $\Sigma \in \mathbb{R}^{m*n}$ is diagonal with the entry is the positive largest to the positive smallest, or, 0.

There is a very important property that every matrix $A \in \mathbb{C}m * n$ has a SVD.

There are also some benefits if we complete the SVD at first, and then do the computing. For example, if I want to calculate the m power of some matrix A, it will take a lot of time to do the basic matrix multiplication. By using the SVD, it is easier: $A^m = S\Sigma^m S^{-1}$

2.2 *Principle Components Analysis(PCA)*

I will only talk three steps about how to do the PCA and more details will be discussed in the next section.

1. Organize the data into a matrix $A \in \mathbb{C}^{m*n}$ where m is the number of measurement types and n is the number of measurements taken from each type.
2. Subtract off the mean for each measurement type or row.
3. Compute the SVD and singular values of the covariance matrix to determine the principal components.

3 ALGORITHM IMPLEMENTATION

3.1 *Case1: The Ideal case*

1. Load video and set up some coefficients
   Because the video has already been transformed to data, so I just need to load the data. The size of the data is $480 * 640 * 3 * 226$ for the first camera in the first case. $480 * 640$ is the size of the corresponding xy-coordinate , the number 3 means that I have the data in RGB color base. 226 means I have a total of 226 images. To make this problem easier, I only deal with the blue color base. The reason that I choose

blue instead of the other two colors is that under the blue base, the light I mentioned before is clearer than in other color bases.

After loading the data, the structure for the data is `uint8`, the structure for images. I first transferred the whole data into `double` and then create a variable to store the size information of the data matrix, named `size1`. I also created another variable named `numFrames`, which contains the last element in the `size1`. This variable stands how many images in the data matrix will be used for the end-loop number. Then, I set x and y with the pixels from the first image and `meshgrid` them, which will be used in the filter later. Set sigma = 0.00001, which is the size of the filter window, and d = 5, which is the max number correction.

2. Find the initial position of the light
   This step seems inefficient, but until now, this is the only way that I can find the actual position of the light. I plot the first image with grid on and then zoom in the light part to find the actual x-y coordinate for that point. The coordinate will be the center position used in the first filter.

   Because from each camera in each case, the position for the light is different, I find the initial position for 12 times and save the coordinate to b and c. This step need to be improve if possible.

3. Build the big for loop

   (a) The start and end number
   The start number is 1, which is obvious. The end number I already defined before, which is just `numFrames`, the number of the total frames from the data matrix.

   (b) The if-statement with filter
   Before starting the if-statement, I created a new variable to store the image under blue base from each frame, so the size of this new variable should be (480*640) pixels of the image.

   Then, I use the `if` statement here to check if this is the first time or not for the loop. If it is the first time, the center position for the filter is b and c that I found before. If this is not the first time, use the center position found from last loop. So the two filter formulas will be:

$$F = e^{(-\sigma*(K_x-b)^2-\sigma*(K_y-c)^2)}$$

   for the first time filter and

$$F = e^{(-\sigma*(K_x-nyc1(j-1))^2-\sigma*(K_y-nxc1(j-1))^2)}$$

   for the rest filter. The variables `nxc` and `nyc` store the new center position calculated from each loop.

   Someone may ask why I use `nyc` for `Kx` and `nxc` for `Ky`, this is an important part need to be very careful, because the `meshgrid` need the structure like this.

   (c) Multiple the filter and find the new center position
   After having the filter ready, just multiple the filter along with the

variable created before that stores the image for each frame. To find the center position, the key point is the light. If I do not have the light on the can, I may be unable to finish this problem. But with the light, I know the white transfer to the data should have the value 255, which is also the maximum value that can represent a color. Thus, after applying the filter, I find the maximum value from the new matrix. Also, because of the filter, the area outside the filter window will be 0, which represents the color black. So, I do not need to worry about the other white stuffs like the white board or white wall which may drag my filter to other place.

After finding the maximum value, I want to find the coordinate(s) for the maximum value. I correct the maximum value by subtracting the coefficient d, which is introduced before to find more coordinates instead of one coordinates. The idea here is because the can is also white, I need a big area of coordinates to make sure the center in still on the light instead of on the can.

(d) While-loop for coordinate correction
Before starting this part, I need to mention that the light is not just a point but a small area. What I mean is that if I have the center position at x and y for the light. I know the value for (x, y) before applying the filter should be 255, the white color. Also, the average values around the point, i.e., the area (x-5:x+5, y-5:y+5) should also near to 255, maybe only 250, because the whole the area have the same light.

By using this idea, I use a `while` loop to make the correction for the center position found before. If the average value of the new center position area is less than the maximum value subtract some number, e.g. 15, I know it maybe the can area instead of the light area, and then I just find the new center position with a smalled max value again. Because different cameras for different cases, some cameras may always have a very bright light area, which means I can set the subtract number very small, like 5. However, some cameras may recode some light area maybe less bright than the can area. In these cases I need to make the subtract number larger, like 45, to make sure the center position will not move to the can area even though the center position area may have a very low average value.

(e) Save the new center position
After the above steps, I will have a new center coordinate, which will be used for the next filter to make sure the filter moves along with the light or the can. I also need save all the coordinates from the `for` loop, because this is the position I extract from the data matrix and will be used later for `SVD` and `PCA`.

(f) `For` loop until end
If all the previous steps are correct, the `for` loop will do the rest tasks for me and I will have two valuable variables, one is named `nxc1` and one is named `nyc1`. Each of the two variables has the size of $1 * numFrames$, which is the x coordinates and y coordinates for the light from the first camera.

4. For Camera 2 and Camera 3

The same step as for Camera 1 is used for Camera 2 and Camera 3. The adjustments are the subtracted numbers mentioned before, and before starting the `forloop`, I need to find the initial center position used in the first filter.

After finishing all the work, I will have 6 vectors, x coordinates and y coordinates for the light of each of the 3 cameras.

5. PCA
   (a) Choose the size
   I found the length for the 6 vectors are not the same. The easiest way is to choose the minimum value of the length. Then, select from the beginning to the end so that all the 6 vectors are in the same length.
   (b) Build the big matrix M
   After having the data ready, build the matrix M as following:
   $$M = \begin{bmatrix} nxc1 \\ nyc1 \\ nxc2 \\ nyc2 \\ nxc3 \\ nyc3 \end{bmatrix}$$
   which will be the basic matrix I need to do the PCA.
   (c) Subtract mean from each row
   Like what I talked in the theory part, subtract each row with its mean value.
   (d) Covariance
   Find the Covariance matrix for the new matrix from the step above.
   (e) SVD
   Use the `svd` command from `Matlab` to do the SVD for the covariance matrix.
   (f) Find the singular value and calculate the energy.
   After the SVD, the diagonal matrix s will have all the singular values for each measurement. Calculate the energy use the singular value of corresponding model divided by the summation of all the singular values.

3.2 *Case2: noisy case*

Apply the similar steps as before, I only need to change the data set, the initial center position coordinates, and maybe the number used for the subtraction.

3.3 *Case3: horizontal displacement*

Apply the similar steps as before, I only need to change the data set, the initial center position coordinates, and maybe the number used for the subtraction.

3.4 *Case4: horizontal displacement and rotation*

Apply the similar steps as before, I only need to change the data set, the initial center position coordinates, and maybe the number used for the subtraction.

4 COMPUTATION RESULTS

4.1 *Case1: The ideal case*

From figure 1, we notice that the simple oscillation from three cameras matches with the ideal case. Because in the ideal case, we do not have other affects, and the only thing left is the oscillation.

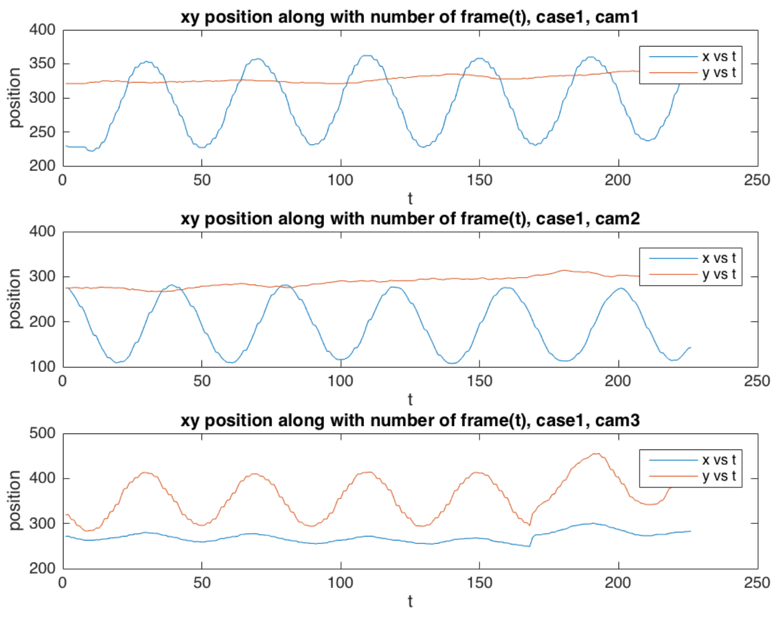

Figure 1: From top to bottom, the x-y positions along with the frame number(t) for camera 1, camera 2 and camera 3 for case 1

From figure 2, I can conclude that the first two models are very useful, and the rest can be considered as redundancies. Also, if look directly from the value of the energy: the energy for model 1 is 0.6238, and the total energy for the first two models is 0.9703, so, just the first two models will be enough. Also, because of the ideal case, this redundancy make seance.

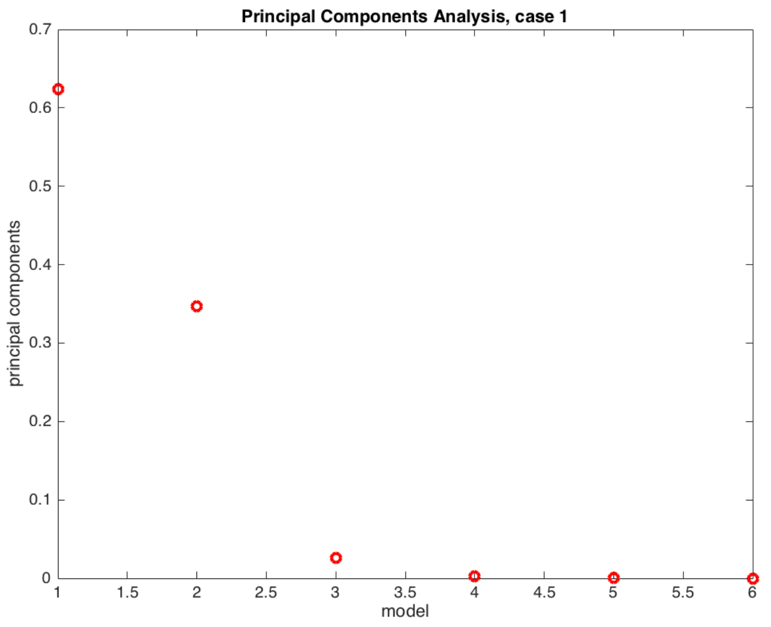

Figure 2: PCA case 1, energy value for each model

4.2 *Case2: noisy case*

From figure 3, we notice that the oscillation is not simple from three cameras which matches with the noisy case. Because when the camera is shaken, the oscillation will also shakes like what happens in this figure.

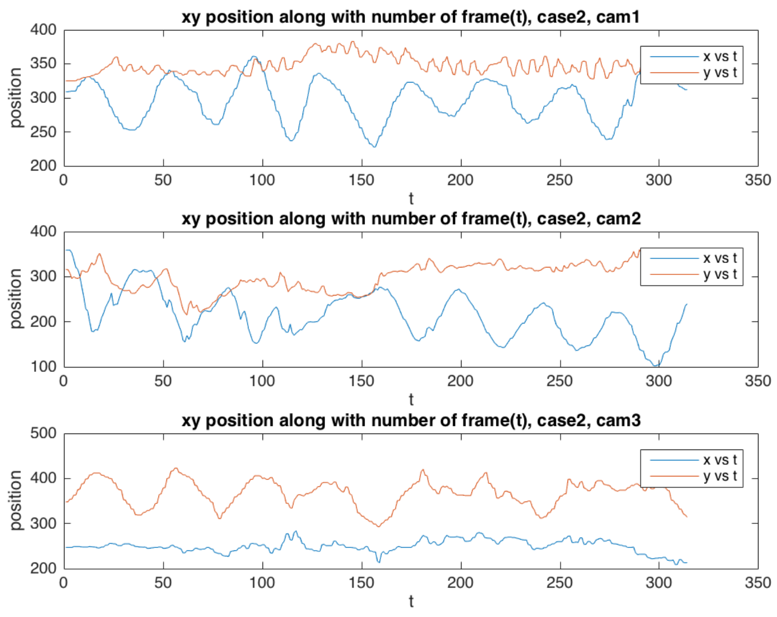

Figure 3: From top to bottom, the x-y position along with the frame number(t) for camera 1, camera 2 and camera 3 for case 2

From figure 4, I can conclude that the first three models are very useful, and the rest can be considered as redundancies. Also, if look directly from the value of the energy, the energy for model 1 is 0.4312, the sum of energy for the first two models is 0.8219, and the sum of energy for the first three model is 0.9355. Because this is the noisy case, it make seance that I need more data to know what happens to this motion.

4.3 *Case3: horizontal displacement*

From figure 5, we notice that the oscillation is simple from three cameras along with some pendulum motion, which matches with the displacement case. Because when the can is released off-center, it will create the pendulum motion instead of the simple oscillate motion. Compare with the Figure 1, it is easier to find that for the Camera 1 and Camera 2, there is a position change for the y coordinate in this case. The forth model can be also considered as useful in this case.

From figure 6, I can conclude that the first four models are very useful, and the rest can be considered as redundancy. Also if look directly from the value of energy, the energy for model 1 is 0.4727, the sum of energy for the first two models is 0.8232, the sum of energy for the first three model is 0.9226, and the sum of energy for the first four model is 0.9838. Because this is the displacement case. I need more and more data to decide what actually happened for the motion. Compare with figure 4, they are very same.

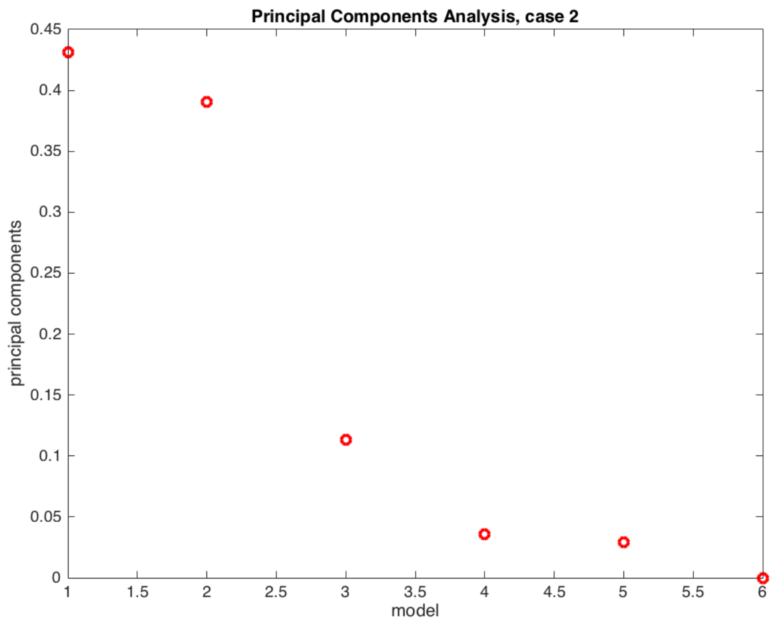

Figure 4: PCA case 2, energy value for each model

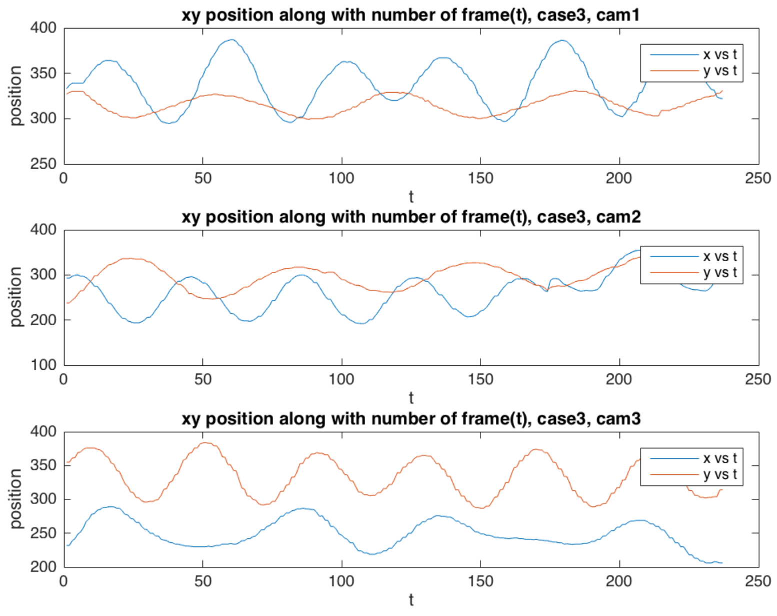

Figure 5: From top to bottom, x-y position along with the frame number(t) for camera 1, camera 2 and camera 3 for case 3

4.4 *Case4: horizontal displacement and rotation*

From figure 7, we notice that the oscillation is simple from three cameras along with some pendulum motions, which match with the displacement and rotation case. Also, if we compare with figure 5, both figure are almost the same, except for this case the y coordinate from Camera1 and Camera2 do not have very strong pendulum motion. This may caused by the rotation of the can.

From figure 8, I can conclude that the first three models are very useful, and the rest can be consider as redundancy. Also, if look directly from the value of energy, the energy for model 1 is 0.6290, the sum of energy for the first two

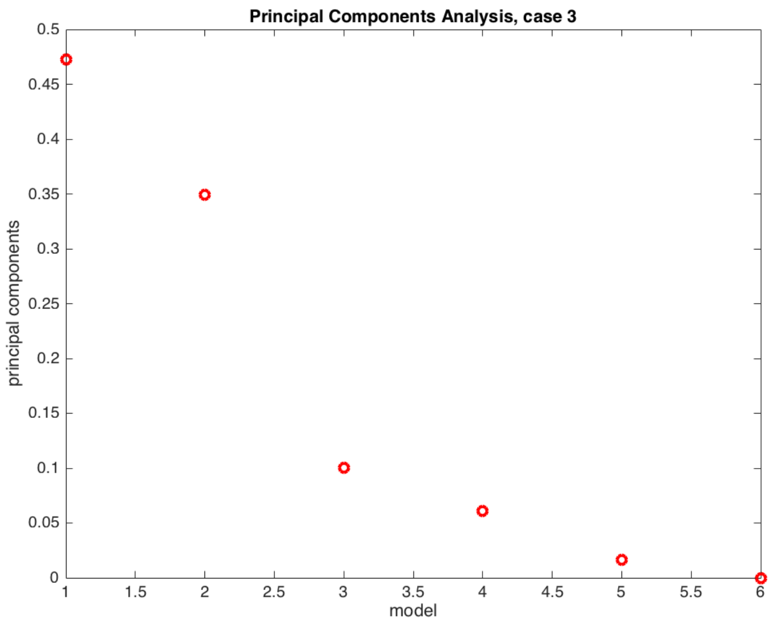

Figure 6: PCA case 3, energy value for each model

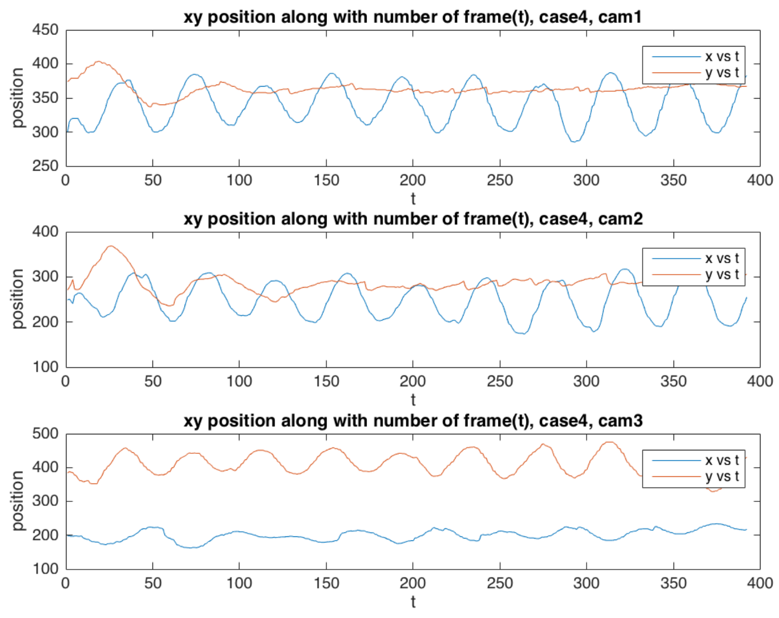

Figure 7: From top to bottom, x-y position along with the frame number(t) for camera 1, camera 2 and camera 3 for case 4

models is 0.9183, and the sum of energy for the first three model is 0.9635. Compare this case to case 3, although I have both displacement and rotation together, the model seems better than case 3. I only need select 3 models in case 4 instead of select 4 models in case 3. I think the reason is because of the rotation. The rotation may reduce the pendulum motion and make the data looks better.

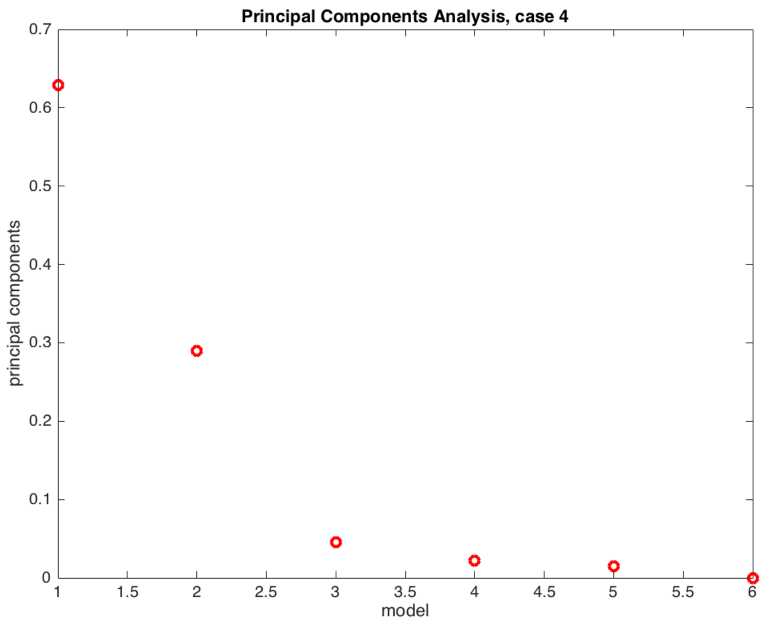

Figure 8: PCA case 4, energy value for each model

5 SUMMARY AND CONCLUSION

By applying the SVD and PCA, we can easily know which measurements may be the redundancy. By removing redundant data, we can decrease the cost and increase the accuracy of the result even though I have no idea about the background of the database.

6 APPENDIX I

Functions Used and Brief Implementation Explanations

1. `load`: load the data matrix
2. `size`: find the size of some variable
3. `meshgrid`: build 2-D system
4. `double`: transfer the structure to double
5. `round`:round towards nearest decimal or integer
6. `max`: find the maximum value
7. `[xc,yc] = find()`: find the coordinate for something
8. `median`: calculate the median value
9. `uint8`: transfer the structure to uint8
10. `imshow`: show the image(should be in uint8 structure)
11. `min`: find the minimum value
12. `mean`: calculate the mean value
13. `cov`: find the covariance matrix of some matrix
14. `[u,s,v]=svd()`: found the SVD for some matrix
15. `diag`: pull out the diagonal value
16. `sum`: calculate the summation

7 APPENDIX II

MATLAB codes

7.1 *Case1: Ideal case*

```
        clear all; close all; clc
sigma=0.0001;
d=5;
%% Cam1_1
load('cam1_1.mat')
a=(vidFrames1_1);
size1=size(a);
numFrames=size1(4);
x=1:640; y=1:480;
[Kx,Ky]=meshgrid(x,y);
% imshow(uint8(vidFrames3_1(180:320,190:500,:,1)))
b=325;
c=247;
for j = 1:numFrames
    X = double(a(:, :, 3, j));
    % imshow(uint8(Xnew(180:320,190:500)))
    if (j==1)
        F=exp(-sigma*(Kx-b).^2-sigma*(Ky-c).^2);
    else
        F=exp(-sigma*(Kx-nyc1(j-1)).^2-sigma*(Ky-nxc1(j-1)).^2);
    end
    xf=round(F.*X);
    Xmax=max(xf(:));
    [xc,yc] = find(xf>=Xmax-d);
    while (sum(sum(xf(round(median(xc))-3:round(median(xc)+3)...
            ,round(median(yc))-3:round(median(yc)+3))))/49<Xmax-15)
        Xmax=Xmax-1;
        [xc,yc] = find(xf>=Xmax-d);
    end
    nxc1(j)=median(xc);
    nyc1(j)=median(yc);
%         imshow(uint8(xf));
%         drawnow
end
%% Cam2_1
load('cam2_1.mat')
a=(vidFrames2_1);
size1=size(a);
numFrames=size1(4);
x=1:640; y=1:480;
[Kx,Ky]=meshgrid(x,y);
% imshow(uint8(vidFrames3_1(180:320,190:500,:,1)))
b=275;
c=275;
for j = 1:numFrames
    X = double(a(:, :, 3, j));
    % imshow(uint8(Xnew(180:320,190:500)))
    if (j==1)
        F=exp(-sigma*(Kx-b).^2-sigma*(Ky-c).^2);
    else
        F=exp(-sigma*(Kx-nyc2(j-1)).^2-sigma*(Ky-nxc2(j-1)).^2);
    end
    xf=round(F.*X);
    Xmax=max(xf(:));
    [xc,yc] = find(xf>=Xmax-d);
    while (sum(sum(xf(round(median(xc))-3:round(median(xc)+3)...
            ,round(median(yc))-3:round(median(yc)+3))))/49<Xmax-15)
        Xmax=Xmax-1;
        [xc,yc] = find(xf>=Xmax-d);
    end
    nxc2(j)=median(xc);
    nyc2(j)=median(yc);
%         imshow(uint8(xf));
%         drawnow
end
%% Cam3_1
load('cam3_1.mat')
a=(vidFrames3_1);
size1=size(a);
numFrames=size1(4);
x=1:640; y=1:480;
[Kx,Ky]=meshgrid(x,y);
% imshow(uint8(vidFrames3_1(180:320,190:500,:,1)))
b=315;
c=272;
for j = 1:numFrames
    X = double(a(:, :, 3, j));
    % imshow(uint8(Xnew(180:320,190:500)))
    if (j==1)
        F=exp(-sigma*(Kx-b).^2-sigma*(Ky-c).^2);
    else
        F=exp(-sigma*(Kx-nyc3(j-1)).^2-sigma*(Ky-nxc3(j-1)).^2);
    end
    xf=round(F.*X);
    Xmax=max(xf(:));
    [xc,yc] = find(xf>=Xmax-d);
    while (sum(sum(xf(round(median(xc))-3:round(median(xc)+3)...
            ,round(median(yc))-3:round(median(yc)+3))))/49<Xmax-5)
        Xmax=Xmax-1;
        [xc,yc] = find(xf>=Xmax-d);
    end
    nxc3(j)=median(xc);
    nyc3(j)=median(yc);
%         imshow(uint8(xf));
%         drawnow
end
%%
s1=size(nxc1,2);
s2=size(nxc2,2);
s3=size(nxc3,2);
minsize=min([s1 s2 s3]);
t=1:minsize;
figure(1)
subplot(3,1,1)
plot(t,nxc1(1:minsize),t,nyc1(1:minsize))
xlabel('t') % x-axis label
ylabel('position') % x-axis label
legend('x vs t','y vs t')
title('xy position along with number of frame(t), case1, cam1')
subplot(3,1,2)
plot(t,nxc2(1:minsize),t,nyc2(1:minsize))
xlabel('t') % x-axis label
```

```
ylabel('position') % x-axis label
legend('x vs t','y vs t')
title('xy position along with number of frame(t), case1, cam2')
subplot(3,1,3)
plot(t,nxc3(1:minsize),t,nyc3(1:minsize))
xlabel('t') % x-axis label
ylabel('position') % x-axis label
legend('x vs t','y vs t')
title('xy position along with number of frame(t), case1, cam3')
%%
M=[nxc1(1:minsize);
    nyc1(1:minsize);
    nxc2(1:minsize);
    nyc3(1:minsize);
    nxc3(1:minsize);
    nyc3(1:minsize)];
for i=1:minsize
    newM(:,i)=M(:,i)-mean(M,2);
end
[u,s,v]=svd(cov(newM'));
% for j=1:3
% ff=u(:,1:j)*s(1:j,1:j)*v(:,1:j)'; % modal projections
% end
sig=diag(s);
% cy=sig.^2/(128-1);
energy1=sig(1)/sum(sig);
energy2=sum(sig(1:2))/sum(sig);
energy3=sum(sig(1:3))/sum(sig);
figure(2)
plot(sig/sum(sig),'ro','Linewidth',[2])
% semilogy(cy,'ko','Linewidth',[1.5])
xlabel('model') % x-axis label
ylabel('principal components') % x-axis label
title('Principal Components Analysis, case 1')
% axis([0 7 0 1])
% subplot(2,2,2), semilogy(sig,'ko','Linewidth',[1.5])
% subplot(2,1,2)
% plot(x,u(:,1),'k',x,u(:,2),'k--',x,u(:,3),'k:','Linewidth',[2])
% plot(y,v(:,1),'k',y,v(:,2),'k--',y,v(:,3),'k:','Linewidth',[2])
```

7.2 *Case2: noisy case*

```
    clear all; close all; clc
sigma=0.0001;
d=5;
%% Cam1_2
load('cam1_2.mat')
a=(vidFrames1_2);
size1=size(a);
numFrames=size1(4);
x=1:640; y=1:480;
[Kx,Ky]=meshgrid(x,y);
% imshow(uint8(vidFrames3_1(180:320,190:500,:,1)))
b=325;
c=307;
for j = 1:numFrames
    X = double(a(:, :, 3, j));
    % imshow(uint8(Xnew(180:320,190:500)))
    if (j==1)
        F=exp(-sigma*(Kx-b).^2-sigma*(Ky-c).^2);
    else
        F=exp(-sigma*(Kx-nyc1(j-1)).^2-sigma*(Ky-nxc1(j-1)).^2);
    end
    xf=round(F.*X);
    Xmax=max(xf(:));
    [xc,yc] = find(xf>=Xmax-d);
    while (sum(sum(xf(round(median(xc))-3:round(median(xc)+3)...
            ,round(median(yc))-3:round(median(yc)+3))))/49<Xmax-5)
        Xmax=Xmax-1;
        [xc,yc] = find(xf>=Xmax-d);
    end
    nxc1(j)=median(xc);
    nyc1(j)=median(yc);
%         imshow(uint8(xf));
%         drawnow
end
%% Cam2_2
load('cam2_2.mat')
a=(vidFrames2_2);
size1=size(a);
numFrames=size1(4);
x=1:640; y=1:480;
[Kx,Ky]=meshgrid(x,y);
% imshow(uint8(vidFrames3_1(180:320,190:500,:,1)))
b=315;
c=362;
for j = 1:numFrames
    X = double(a(:, :, 3, j));
    % imshow(uint8(Xnew(180:320,190:500)))
    if (j==1)
        F=exp(-sigma*(Kx-b).^2-sigma*(Ky-c).^2);
    else
        F=exp(-sigma*(Kx-nyc2(j-1)).^2-sigma*(Ky-nxc2(j-1)).^2);
    end
    xf=round(F.*X);
    Xmax=max(xf(:));
    [xc,yc] = find(xf>=Xmax-d);
    while (sum(sum(xf(round(median(xc))-3:round(median(xc)+3)...
            ,round(median(yc))-3:round(median(yc)+3))))/49<Xmax-15)
        Xmax=Xmax-1;
        [xc,yc] = find(xf>=Xmax-d);
    end
    nxc2(j)=median(xc);
    nyc2(j)=median(yc);
%         imshow(uint8(xf));
%         drawnow
end
%% Cam3_2
load('cam3_2.mat')
a=(vidFrames3_2);
size1=size(a);
numFrames=size1(4);
x=1:640; y=1:480;
[Kx,Ky]=meshgrid(x,y);
% imshow(uint8(vidFrames3_1(180:320,190:500,:,1)))
b=347;
c=247;
```

```
for j = 1:numFrames
    X = double(a(:, :, 3, j));
    % imshow(uint8(Xnew(180:320,190:500)))
    if (j==1)
        F=exp(-sigma*(Kx-b).^2-sigma*(Ky-c).^2);
    else
        F=exp(-sigma*(Kx-nyc3(j-1)).^2-sigma*(Ky-nxc3(j-1)).^2);
    end
    xf=round(F.*X);
    Xmax=max(xf(:));
    [xc,yc] = find(xf>=Xmax-d);
    while (sum(sum(xf(round(median(xc))-3:round(median(xc)+3)...
            ,round(median(yc))-3:round(median(yc)+3))))/49<Xmax-15)
        Xmax=Xmax-1;
        [xc,yc] = find(xf>=Xmax-d);
    end
    nxc3(j)=median(xc);
    nyc3(j)=median(yc);
%         imshow(uint8(xf));
%         drawnow
end
%%
s1=size(nxc1,2);
s2=size(nxc2,2);
s3=size(nxc3,2);
minsize=min([s1 s2 s3]);
t=1:minsize;
figure(1)
subplot(3,1,1)
plot(t,nxc1(1:minsize),t,nyc1(1:minsize))
xlabel('t') % x-axis label
ylabel('position') % x-axis label
legend('x vs t','y vs t')
title('xy position along with number of frame(t), case2, cam1')
subplot(3,1,2)
plot(t,nxc2(1:minsize),t,nyc2(1:minsize))
xlabel('t') % x-axis label
ylabel('position') % x-axis label
legend('x vs t','y vs t')
title('xy position along with number of frame(t), case2, cam2')
subplot(3,1,3)
plot(t,nxc3(1:minsize),t,nyc3(1:minsize))
xlabel('t') % x-axis label
ylabel('position') % x-axis label
legend('x vs t','y vs t')
title('xy position along with number of frame(t), case2, cam3')
%%
ed=1+255;
M=[nxc1(1:ed);
    nyc1(1:ed);
    nxc2(1:ed);
    nyc3(1:ed);
    nxc3(1:ed);
    nyc3(1:ed)];
for i=1:ed
    newM(:,i)=abs(M(:,i)-mean(M,2));
end
[u,s,v]=svd(cov(newM'));
% for j=1:3
% ff=u(:,1:j)*s(1:j,1:j)*v(:,1:j)'; % modal projections
% end
sig=diag(s);
% cy=sig.^2/(128-1);
energy1=sig(1)/sum(sig);
energy2=sum(sig(1:2))/sum(sig);
energy3=sum(sig(1:3))/sum(sig);
energy4=sum(sig(1:4))/sum(sig);
energy5=sum(sig(1:5))/sum(sig);
figure(2)
plot(sig/sum(sig),'ro','Linewidth',[2])
% semilogy(cy,'ko','Linewidth',[1.5])
xlabel('model') % x-axis label
ylabel('principal components') % x-axis label
title('Principal Components Analysis, case 2')
% axis([0 7 0 1])
% subplot(2,2,2), semilogy(sig,'ko','Linewidth',[1.5])
% subplot(2,1,2)
% plot(x,u(:,1),'k',x,u(:,2),'k--',x,u(:,3),'k:','Linewidth',[2])
% plot(y,v(:,1),'k',y,v(:,2),'k--',y,v(:,3),'k:','Linewidth',[2])
```

7.3 *Case3: horizontal displacement*

```
        clear all; close all; clc
sigma=0.0001;
d=5;
%% Cam1_3
load('cam1_3.mat')
a=(vidFrames1_3);
size1=size(a);
numFrames=size1(4);
x=1:640; y=1:480;
[Kx,Ky]=meshgrid(x,y);
% imshow(uint8(vidFrames3_1(180:320,190:500,:,1)))
b=320;
c=325;
for j = 1:numFrames
    X = double(a(:, :, 3, j));
    % imshow(uint8(Xnew(180:320,190:500)))
    if (j==1)
        F=exp(-sigma*(Kx-b).^2-sigma*(Ky-c).^2);
    else
        F=exp(-sigma*(Kx-nyc1(j-1)).^2-sigma*(Ky-nxc1(j-1)).^2);
    end
    xf=round(F.*X);
    Xmax=max(xf(:));
    [xc,yc] = find(xf>=Xmax-d);
    while (sum(sum(xf(round(median(xc))-3:round(median(xc)+3)...
            ,round(median(yc))-3:round(median(yc)+3))))/49<Xmax-15)
        Xmax=Xmax-1;
        [xc,yc] = find(xf>=Xmax-d);
    end
    nxc1(j)=median(xc);
    nyc1(j)=median(yc);
%         imshow(uint8(xf));
%         drawnow
end
%% Cam2_3
load('cam2_3.mat')
```

```
a=(vidFrames2_3);
size1=size(a);
numFrames=size1(4);
x=1:640; y=1:480;
[Kx,Ky]=meshgrid(x,y);
% imshow(uint8(vidFrames3_1(180:320,190:500,:,1)))
b=235;
c=294;
for j = 1:numFrames
    X = double(a(:, :, 3, j));
    % imshow(uint8(Xnew(180:320,190:500)))
    if (j==1)
        F=exp(-sigma*(Kx-b).^2-sigma*(Ky-c).^2);
    else
        F=exp(-sigma*(Kx-nyc2(j-1)).^2-sigma*(Ky-nxc2(j-1)).^2);
    end
    xf=round(F.*X);
    Xmax=max(xf(:));
    [xc,yc] = find(xf>=Xmax-d);
    while (sum(sum(xf(round(median(xc))-3:round(median(xc)+3)...
            ,round(median(yc))-3:round(median(yc)+3))))/49<Xmax-35)
        Xmax=Xmax-1;
        [xc,yc] = find(xf>=Xmax-d);
    end
    nxc2(j)=median(xc);
    nyc2(j)=median(yc);
%         imshow(uint8(xf));
%         drawnow
end
%% Cam3_3
load('cam3_3.mat')
a=(vidFrames3_3);
size1=size(a);
numFrames=size1(4);
x=1:640; y=1:480;
[Kx,Ky]=meshgrid(x,y);
% imshow(uint8(vidFrames3_1(180:320,190:500,:,1)))
b=352;
c=230;
for j = 1:numFrames
    X = double(a(:, :, 3, j));
    % imshow(uint8(Xnew(180:320,190:500)))
    if (j==1)
        F=exp(-sigma*(Kx-b).^2-sigma*(Ky-c).^2);
    else
        F=exp(-sigma*(Kx-nyc3(j-1)).^2-sigma*(Ky-nxc3(j-1)).^2);
    end
    xf=round(F.*X);
    Xmax=max(xf(:));
    [xc,yc] = find(xf>=Xmax-d);
    while (sum(sum(xf(round(median(xc))-3:round(median(xc)+3)...
            ,round(median(yc))-3:round(median(yc)+3))))/49<Xmax-15)
        Xmax=Xmax-1;
        [xc,yc] = find(xf>=Xmax-d);
    end
    nxc3(j)=median(xc);
    nyc3(j)=median(yc);
%         imshow(uint8(xf));
%         drawnow
```

```
end
%%
s1=size(nxc1,2);
s2=size(nxc2,2);
s3=size(nxc3,2);
minsize=min([s1 s2 s3]);
t=1:minsize;
figure(1)
subplot(3,1,1)
plot(t,nxc1(1:minsize),t,nyc1(1:minsize))
xlabel('t') % x-axis label
ylabel('position') % x-axis label
legend('x vs t','y vs t')
title('xy position along with number of frame(t), case3, cam1')
subplot(3,1,2)
plot(t,nxc2(1:minsize),t,nyc2(1:minsize))
xlabel('t') % x-axis label
ylabel('position') % x-axis label
legend('x vs t','y vs t')
title('xy position along with number of frame(t), case3, cam2')
subplot(3,1,3)
plot(t,nxc3(1:minsize),t,nyc3(1:minsize))
xlabel('t') % x-axis label
ylabel('position') % x-axis label
legend('x vs t','y vs t')
title('xy position along with number of frame(t), case3, cam3')
%%
ed=1+220;
M=[nxc1(1:ed);
    nyc1(1:ed);
    nxc2(1:ed);
    nyc3(1:ed);
    nxc3(1:ed);
    nyc3(1:ed)];
for i=1:ed
    newM(:,i)=abs(M(:,i)-mean(M,2));
end
[u,s,v]=svd(cov(newM'));
% for j=1:3
% ff=u(:,1:j)*s(1:j,1:j)*v(:,1:j)'; % modal projections
% end
sig=diag(s);
% cy=sig.^2/(128-1);
energy1=sig(1)/sum(sig);
energy2=sum(sig(1:2))/sum(sig);
energy3=sum(sig(1:3))/sum(sig);
energy4=sum(sig(1:4))/sum(sig);
energy5=sum(sig(1:5))/sum(sig);
figure(2)
plot(sig/sum(sig),'ro','Linewidth',[2])
% semilogy(cy,'ko','Linewidth',[1.5])
xlabel('model') % x-axis label
ylabel('principal components') % x-axis label
title('Principal Components Analysis, case 3')
% axis([0 7 0 1])
% subplot(2,2,2), semilogy(sig,'ko','Linewidth',[1.5])
% subplot(2,1,2)
% plot(x,u(:,1),'k',x,u(:,2),'k--',x,u(:,3),'k:','Linewidth',[2])
% plot(y,v(:,1),'k',y,v(:,2),'k--',y,v(:,3),'k:','Linewidth',[2])
```

7.4 *Case4: horizontal displacement and rotation*

```
        clear all; close all; clc
sigma=0.0001;
d=5;
%% Cam1_4
load('cam1_4.mat')
a=(vidFrames1_4);
size1=size(a);
numFrames=size1(4);
x=1:640; y=1:480;
[Kx,Ky]=meshgrid(x,y);
% imshow(uint8(vidFrames3_1(180:320,190:500,:,1)))
b=390;
c=275;
for j = 1:numFrames
    X = double(a(:, :, 3, j));
    % imshow(uint8(Xnew(180:320,190:500)))
    if (j==1)
        F=exp(-sigma*(Kx-b).^2-sigma*(Ky-c).^2);
    else
        F=exp(-sigma*(Kx-nyc1(j-1)).^2-sigma*(Ky-nxc1(j-1)).^2);
    end
    xf=round(F.*X);
    Xmax=max(xf(:));
    [xc,yc] = find(xf>=Xmax-d);
    while (sum(sum(xf(round(median(xc))-3:round(median(xc)+3)...
            ,round(median(yc))-3:round(median(yc)+3))))/49<Xmax-15)
        Xmax=Xmax-1;
        [xc,yc] = find(xf>=Xmax-d);
    end
    nxc1(j)=median(xc);
    nyc1(j)=median(yc);
%         imshow(uint8(xf));
%         drawnow
end
%% Cam2_4
load('cam2_4.mat')
a=(vidFrames2_4);
size1=size(a);
numFrames=size1(4);
x=1:640; y=1:480;
[Kx,Ky]=meshgrid(x,y);
% imshow(uint8(vidFrames3_1(180:320,190:500,:,1)))
b=250;
c=250;
for j = 1:numFrames
    X = double(a(:, :, 3, j));
    % imshow(uint8(Xnew(180:320,190:500)))
    if (j==1)
        F=exp(-sigma*(Kx-b).^2-sigma*(Ky-c).^2);
    else
        F=exp(-sigma*(Kx-nyc2(j-1)).^2-sigma*(Ky-nxc2(j-1)).^2);
    end
    xf=round(F.*X);
    Xmax=max(xf(:));
    [xc,yc] = find(xf>=Xmax-d);
    while (sum(sum(xf(round(median(xc))-3:round(median(xc)+3)...
             ,round(median(yc))-3:round(median(yc)+3))))/49<Xmax-35)
        Xmax=Xmax-1;
        [xc,yc] = find(xf>=Xmax-d);
    end
    nxc2(j)=median(xc);
    nyc2(j)=median(yc);
%         imshow(uint8(xf));
%         drawnow
end
%% Cam3_4
load('cam3_4.mat')
a=(vidFrames3_4);
size1=size(a);
numFrames=size1(4);
x=1:640; y=1:480;
[Kx,Ky]=meshgrid(x,y);
% imshow(uint8(vidFrames3_1(180:320,190:500,:,1)))
b=360;
c=205;
for j = 1:numFrames
    X = double(a(:, :, 3, j));
    % imshow(uint8(Xnew(180:320,190:500)))
    if (j==1)
        F=exp(-sigma*(Kx-b).^2-sigma*(Ky-c).^2);
    else
        F=exp(-sigma*(Kx-nyc3(j-1)).^2-sigma*(Ky-nxc3(j-1)).^2);
    end
    xf=round(F.*X);
    Xmax=max(xf(:));
    [xc,yc] = find(xf>=Xmax-d);
    while (sum(sum(xf(round(median(xc))-3:round(median(xc)+3)...
             ,round(median(yc))-3:round(median(yc)+3))))/49<Xmax-15)
        Xmax=Xmax-1;
        [xc,yc] = find(xf>=Xmax-d);
    end
    nxc3(j)=median(xc);
    nyc3(j)=median(yc);
%         imshow(uint8(xf));
%         drawnow
end
%%
s1=size(nxc1,2);
s2=size(nxc2,2);
s3=size(nxc3,2);
minsize=min([s1 s2 s3]);
t=1:minsize;
figure(1)
subplot(3,1,1)
plot(t,nxc1(1:minsize),t,nyc1(1:minsize))
xlabel('t') % x-axis label
ylabel('position') % x-axis label
legend('x vs t','y vs t')
title('xy position along with number of frame(t), case4, cam1')
subplot(3,1,2)
plot(t,nxc2(1:minsize),t,nyc2(1:minsize))
xlabel('t') % x-axis label
ylabel('position') % x-axis label
legend('x vs t','y vs t')
```

```
title('xy position along with number of frame(t), case4, cam2')
subplot(3,1,3)
plot(t,nxc3(1:minsize),t,nyc3(1:minsize))
xlabel('t') % x-axis label
ylabel('position') % x-axis label
legend('x vs t','y vs t')
title('xy position along with number of frame(t), case4, cam3')
%%
ed=1+390;
M=[nxc1(1:ed);
    nyc1(1:ed);
    nxc2(1:ed);
    nyc3(1:ed);
    nxc3(1:ed);
    nyc3(1:ed)];
for i=1:ed
    newM(:,i)=M(:,i)-mean(M,2);
end
[u,s,v]=svd(cov(newM'));
% for j=1:3
% ff=u(:,1:j)*s(1:j,1:j)*v(:,1:j)'; % modal projections
% end
sig=diag(s);
% cy=sig.^2/(128-1);
energy1=sig(1)/sum(sig);
energy2=sum(sig(1:2))/sum(sig);
energy3=sum(sig(1:3))/sum(sig);
energy4=sum(sig(1:4))/sum(sig);
energy5=sum(sig(1:5))/sum(sig);
figure(2)
plot(sig/sum(sig),'ro','Linewidth',[2])
% semilogy(cy,'ko','Linewidth',[1.5])
xlabel('model') % x-axis label
ylabel('principal components') % x-axis label
title('Principal Components Analysis, case 4')
% axis([0 7 0 1])
% subplot(2,2,2), semilogy(sig,'ko','Linewidth',[1.5])
% subplot(2,1,2)
% plot(x,u(:,1),'k',x,u(:,2),'k--',x,u(:,3),'k:','Linewidth',[2])
% plot(y,v(:,1),'k',y,v(:,2),'k--',y,v(:,3),'k:','Linewidth',[2])
```