Principal Component Analysis on a Spring Mass System

Amanda Lee

Febuary 21, 2020

Abstract

I perform PCA on position data of a mass in a spring mass system recorded from different angles for different cases of motion.

Introduction

There is a spring mass system that is recorded with three cameras, each at a different location. There are four sets of recordings. In the first set, the mass is only moving in the z direction and there is minimal noise, or shaking of the cameras. In the second case, the motion of the mass is the same, but the cameras are shaking. In the third case, the mass was released off center, creating a pendulum movement along with the simple harmonic motion. In the fourth case, the movement is the same as the third case but with the addition of the rotation of the mass. The goal is to use principal component analysis to get a simple description of the motion of the mass. For the first two cases, the motion of mass is essentially the same and the information is oversampled with three cameras. Before I can use principal component analysis, however, I will have to extract the position information of the mass from every video recording.

Background

Singular Value Decomposition

Singular Value Decomposition (SVD) is a way of factorizing a matrix into different components that make it useful for principal component analysis (PCA). For the reduced SVD, A matrix, A, is broken down such that

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

If A is a full rank mxn matrix, then U is a mxm unitary matrix, Σ is a diagonal matrix, and V^* is a nxm unitary matrix. The entries in Σ are called the singular values of A and are assumed to be nonnegative. The singular values are also arranged from largest to smallest. The value of the singular value corresponds to the amount of information contained in the corresponding bases. The SVD allows one to create low dimensional approximations of the matrix.

SVD is similar to eigenvalue decomposition, except instead of diagonalizing the matrix along one basis, the SVD diagonalizes along two bases, V^* and U. In addition, SVD can be done on any size of matrix. The columns of U are orthogonal unit vectors called the left singular vectors of A, and the columns of V are orthogonal unit vectors called the right singular vectors of A.

Algorithm Implementation

To get the position information of paint can in each video, I developed two methods.Both methods involve turning the video to grayscale so that there is less information I need to work with.

The first method depends on the video having a good view of the flashlight that is on top of the can. I crop out areas where the mass does not show up. Then, to get the position of the paint can, I get the index of the maximum value in each frame. I then plotted the position information to check how smooth the data was.

The third method was used when the flashlight was not very visible. I found the average frame of each video, then subtracted the average from each frame. This has the effect of getting rid of most of the background in the video, leaving mostly the paint can. Then, I set a minimum brightness value and took the average of the positions of all the pixels brighter than that threshold. This method lessened the effect of rotation on the position recorded in the fourth case.

Using these methods, I was not able to get good position information for all frames, so I discarded information at certain frames that did not match up with the rest of the data and used interpolation to fill in data at those frames.

The next step was to align the position information for each video to the position information for the other videos in the same case. The method I used was to look at the plots of the y positions over time and sync up the peaks and troughs of the videos. Once I had the data aligned, truncated the vectors to make them the same length. Finally, I subtracted the mean of each position vector from itself.

Once I had the snapshot matrices, I could perform SVD on each matrix and look at the low rank approximations of the position data.

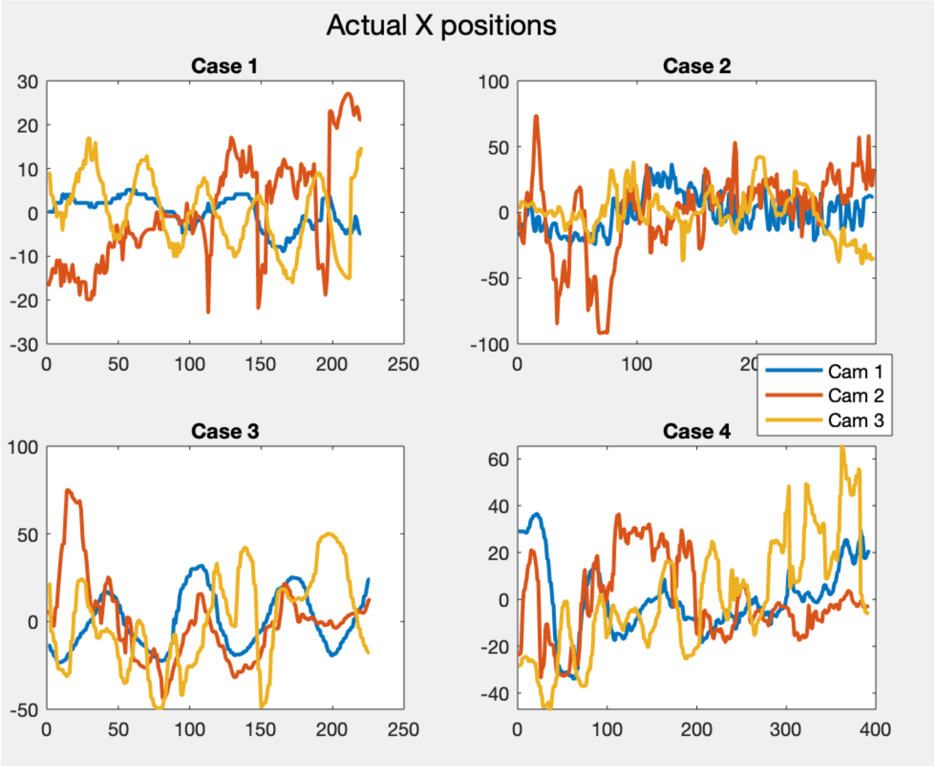

Figure 1: These are the X positions collected from the video data as described above with the means subtracted.

Results

The methods I used for getting the position information of the mass worked well for all cases except for the noisy case, as seen in figures 1 and 2. For the noisy case, it was difficult to figure out which coordinates I originally collected were mistakes and which were just a result of the noise. Because of this, I was able to get nice rank 1 approximations for the first, third, and fourth cases that show the simple harmonic motion of the mass as seen in figures 3 and 4.

We can also see that the rank 1 approximations are just scaled versions of the first right singular vector, which is consistent with the math used to get the approximation (figure 5).

The singular values also gave an insight into the motion of the mass. If then motion of the mass was perfectly one dimensional, then there would only be one singular value. In figure 6, we can see that for case 1, the case with the simplest motion, the first singular value is much larger than the rest of the singular values for that case. The second case, which has the same motion but with added noise, has a somewhat larger first singular value, with the other values also being much smaller. This means that the SVD was able to isolate the one dimensional motion of the mass despite the noise. Case three and four have a smaller first singular value, which is likely because of the horizontal displacement of the mass.

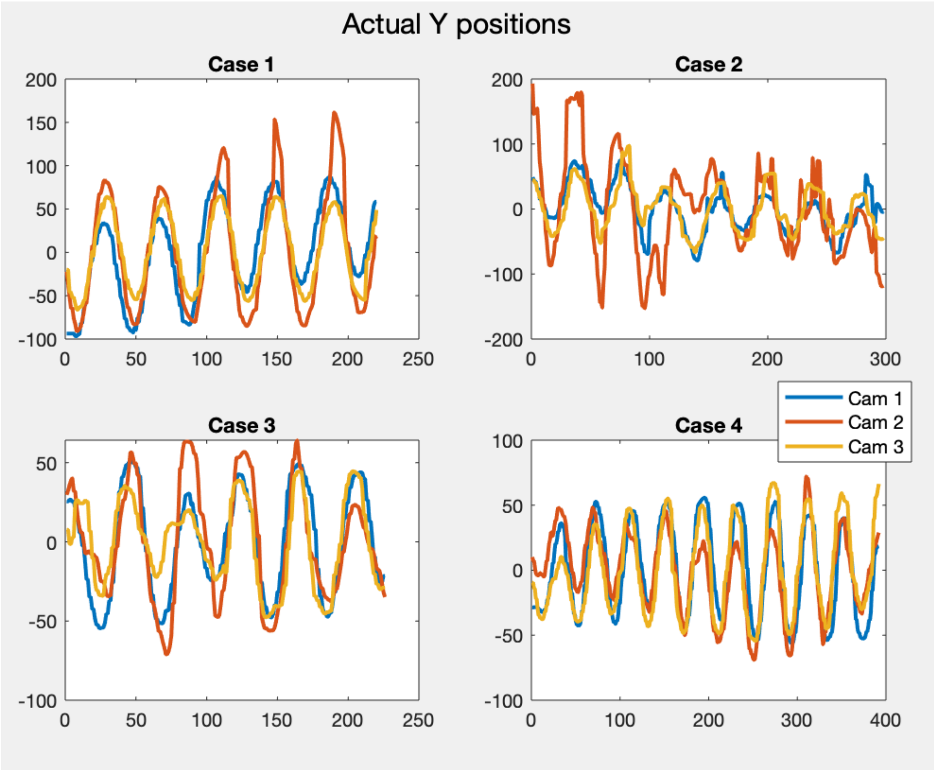

Figure 2: These are the Y positions collected from the video data as described above with the means subtracted.

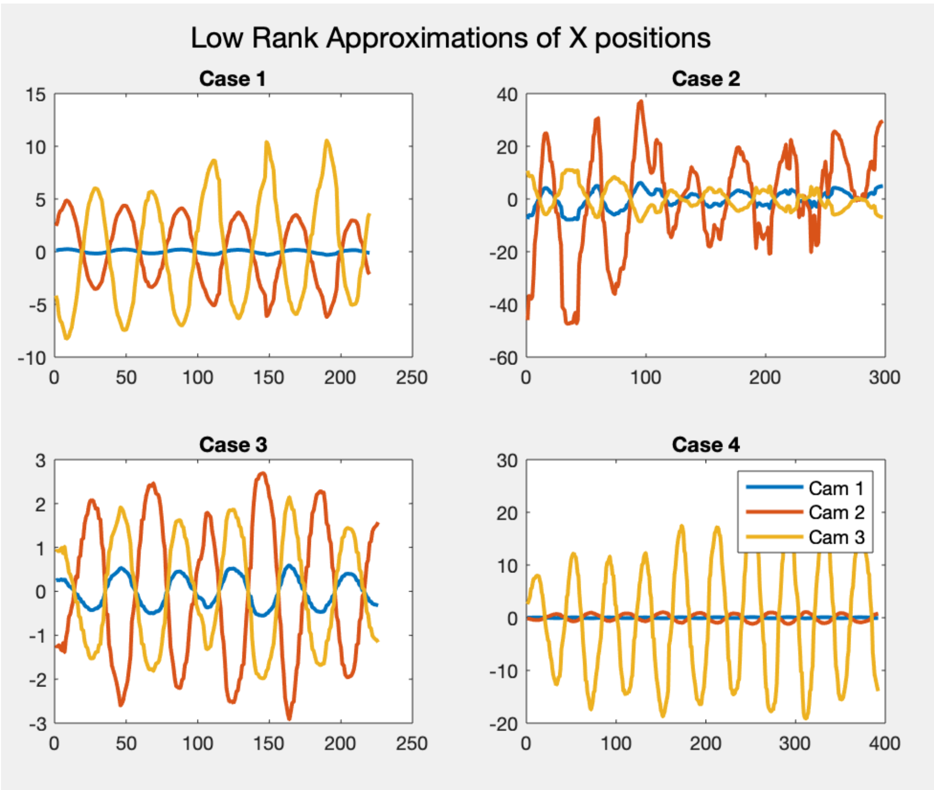

Figure 3: These are the rank 1 approximations of the X positions.

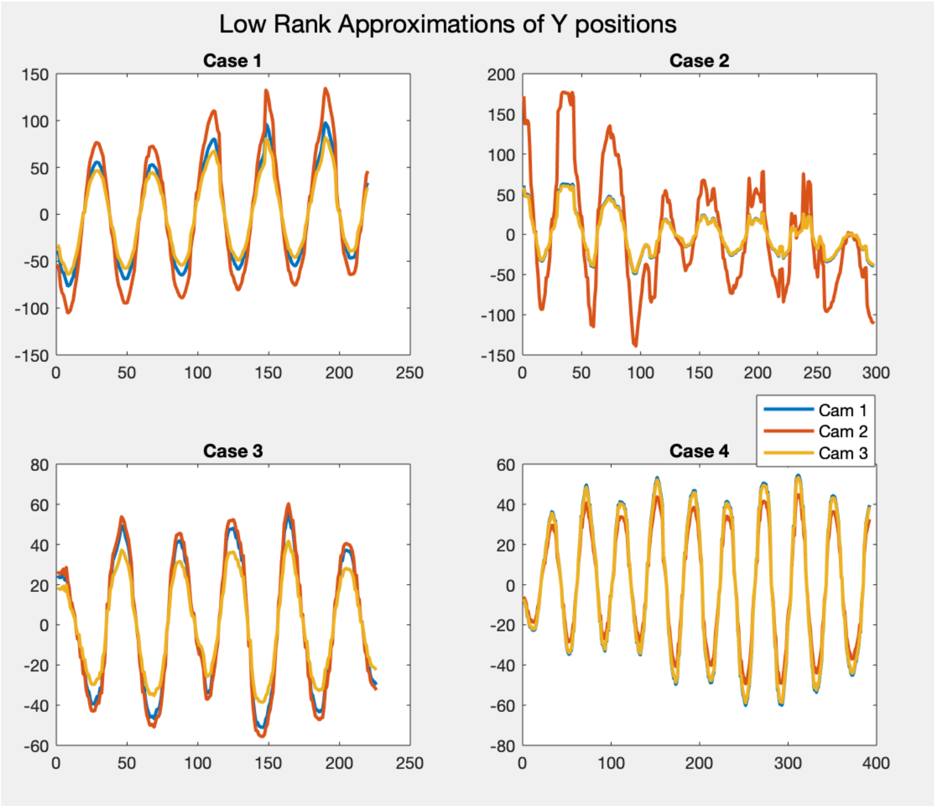

Figure 4: These are the rank 1 approximations of the X positions.

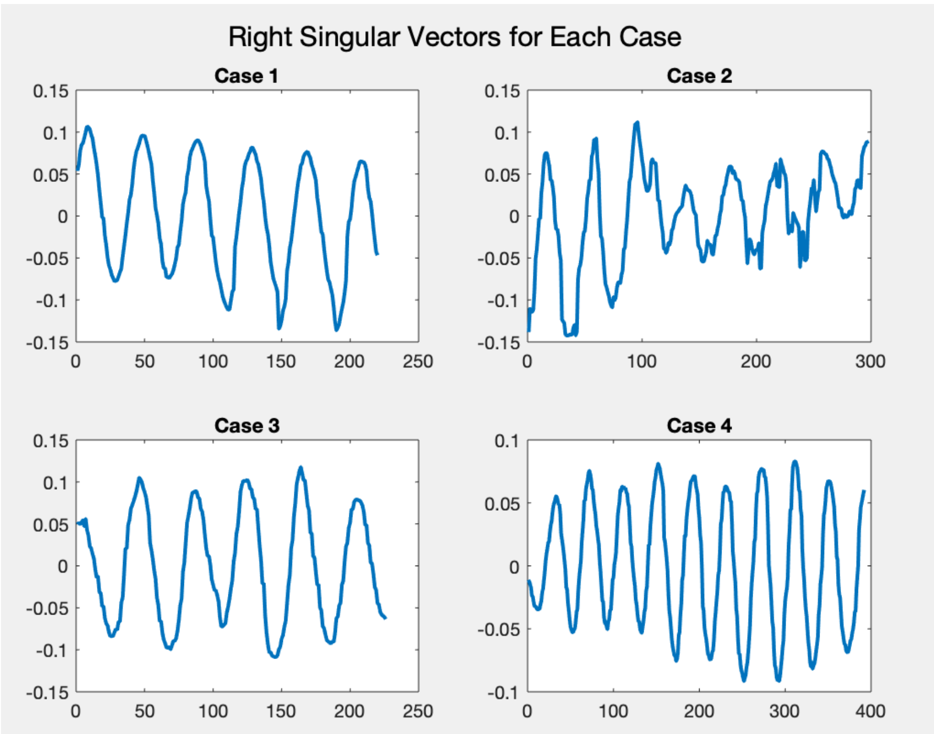

Figure 5: These are the first right singular vectors for each case.

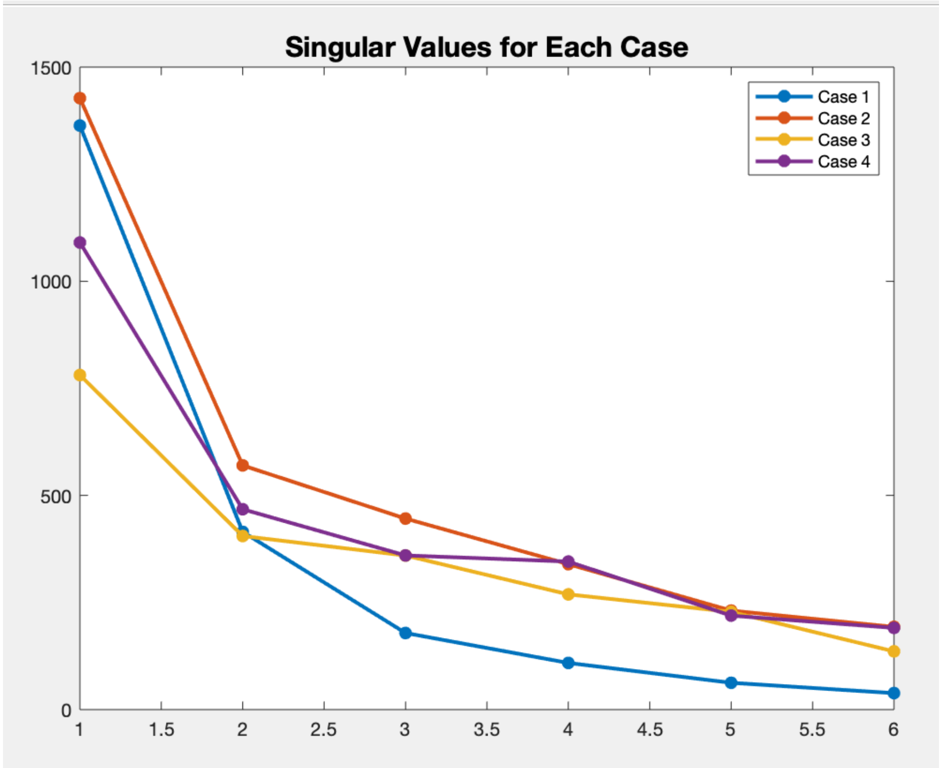

Figure 6: These are the singular values for each case.

Conclusion

Through PCA, I was able to infer information about the motion of the mass in each case. For the first two cases with one dimensional motion, the singular values reflected this fact. The last two cases had more complex motion, resulting in a smaller first singular value, indicating that the up and down motion of the mass was not as significant in these cases. The first right singular vectors also revealed the simple harmonic motion in each case.

Appendix A

`[U,S,V] = svd(A,'econ')` - Performs the reduced SVD where `A=U*S*V'`.

`implay(I)` - Plays an image sequence. This function is used to help with syncing positions across cameras and for knowing which areas to crop out for position gathering.

`vq = interp1(x,v,xq,'pchip')` - Returns interpolated values within `v` with the corresponding the sample points, `x`, at the specified points, `xq`, using cubic interpolation. This is used to fill in data at frames where the position finding methods failed.

Appendix B

getPos1.m

```
function [x_inds,y_inds] = getPos1(vid,startx,endx)
% Get position data for videos with clear a clear flashlight image
numFrames = size(vid,4);
% Turn video to greyscale and crop
gvid = zeros(480,endx-startx+1,numFrames);
for j = 1:numFrames
    gvid(:,:,j) = rgb2gray(vid(:,startx:endx,:,j));
end
% Get position data
x_inds = zeros(numFrames,1);
y_inds = zeros(numFrames,1);
for j = 1:numFrames
    [~,I] = max(gvid(:,:,j), [], 'all', 'linear');
    [row, col] = ind2sub([480,endx-startx+1],I);
    x_inds(j) = col;
    y_inds(j) = row;
end
end
```

getPos2.m

```
function [x_inds,y_inds] = getPos1(vid,startx,endx,starty,endy)
% Get position data for videos with clear a clear flashlight image and
% cropping in both dimensions
numFrames = size(vid,4);
% Turn video to greyscale and crop
gvid = zeros(endy-starty+1,endx-startx+1,numFrames);
for j = 1:numFrames
    gvid(:,:,j) = rgb2gray(vid(starty:endy,startx:endx,:,j));
end
% Get position data
x_inds = zeros(numFrames,1);
y_inds = zeros(numFrames,1);
for j = 1:numFrames
    [~,I] = max(gvid(:,:,j), [], 'all', 'linear');
    [row, col] = ind2sub([endy-starty+1,endx-startx+1],I);
    x_inds(j) = col;
    y_inds(j) = row;
end
end
```

getPos3.m

```
function [x_inds,y_inds] = getPos3(thresh,vid,startx,endx,starty,endy)
% Get position data for videos using averaging
numFrames = size(vid,4);
% Turn video to greyscale and crop
```

```
% Get average frame
avgFrame = zeros(endy-starty+1,endx-startx+1);
gvid = zeros(endy-starty+1,endx-startx+1,numFrames);
for j = 1:numFrames
    gvid(:,:,j) = rgb2gray(vid(starty:endy,startx:endx,:,j));
    avgFrame = avgFrame + im2double(gvid(:,:,j));
end
avgFrame = avgFrame/numFrames;

diff = zeros(size(gvid));

for j = 1:numFrames
    frame1 = gvid(:,:,j);
    diff(:,:,j) = frame1-avgFrame;
end
diff(diff<0) = 0;

% Get position data
x_inds = zeros(numFrames,1);
y_inds = zeros(numFrames,1);
for j = 1:numFrames
    [Y,X] = find(diff(:,:,j)>thresh);
    x_inds(j) = mean(X);
    y_inds(j) = mean(Y);
end
end
```

gettingCoordsScript.m

```
clear; clc; close all;
% Camera 1 Case 1
load('cam1_1.mat')
% implay(vidFrames1_1)
[xpos_1_1,ypos_1_1] = getPos1(vidFrames1_1, 243,445);
numFrames = size(vidFrames1_1,4);

% Plot position data to look for incongruities
frameNums = 1:numFrames;
figure(1)
plot(frameNums,ypos_1_1)
hold on
plot(frameNums,xpos_1_1)

% Remove points that are out of place
badFrames = [81, 88, 89, 90, 92, 93, 94, 95, 100, 101, 112, 113, 198, 199,
    201, 202, 220, 221];

for k = length(badFrames):-1:1
    badFrame = badFrames(k);
    frameNums(badFrame) = [];
    xpos_1_1(badFrame) = [];
    ypos_1_1(badFrame) = [];
end

```

```
% Plot to check
figure(2)
plot(frameNums,ypos_1_1)
hold on
plot(frameNums,xpos_1_1)

% Use interpotation to get data for missing points
xinterp = interp1(frameNums, xpos_1_1, badFrames, 'pchip');
yinterp = interp1(frameNums, ypos_1_1, badFrames, 'pchip');

% Add interp points back in
xpos_1_1 = xpos_1_1';
ypos_1_1 = ypos_1_1';
for k = 1:length(badFrames)
    badFrame = badFrames(k);
    xpos_1_1 = [xpos_1_1(1:badFrame-1), xinterp(k), xpos_1_1(badFrame:end)];
    ypos_1_1 = [ypos_1_1(1:badFrame-1), yinterp(k), ypos_1_1(badFrame:end)];
end
frameNums = 1:numFrames;

% Plot to check
figure(3)
plot(frameNums,ypos_1_1)
hold on
plot(frameNums,xpos_1_1)
xpos_1_1 = xpos_1_1';
ypos_1_1 = ypos_1_1';
%% Camera 2 Case 1
close all
load('cam2_1.mat')
%implay(vidFrames2_1)
[xpos_2_1,ypos_2_1] = getPos1(vidFrames2_1, 240,360);
numFrames = size(vidFrames2_1,4);

% Plot position data to look for incongruities
frameNums = 1:numFrames;
figure(1)
plot(frameNums,ypos_2_1)
hold on
plot(frameNums,xpos_2_1)

% Remove points that are out of place
badFrames =
    [5,6,7,8,40,41,42,118,119,120,121,122,135,136,137,138,139,140,141,142,145,146,158,15

for k = length(badFrames):-1:1
    badFrame = badFrames(k);
    frameNums(badFrame) = [];
    xpos_2_1(badFrame) = [];
    ypos_2_1(badFrame) = [];
end
% Plot to check
figure(2)
plot(frameNums,ypos_2_1)
```

```
hold on
plot(frameNums,xpos_2_1)

% Use interpotation to get data for missing points
xinterp = interp1(frameNums, xpos_2_1, badFrames, 'pchip');
yinterp = interp1(frameNums, ypos_2_1, badFrames, 'pchip');

% Add interp points back in
xpos_2_1 = xpos_2_1';
ypos_2_1 = ypos_2_1';
for k = 1:length(badFrames)
    badFrame = badFrames(k);
    xpos_2_1 = [xpos_2_1(1:badFrame-1), xinterp(k), xpos_2_1(badFrame:end)];
    ypos_2_1 = [ypos_2_1(1:badFrame-1), yinterp(k), ypos_2_1(badFrame:end)];
end
xpos_2_1 = xpos_2_1';
ypos_2_1 = ypos_2_1';
frameNums = 1:numFrames;
% Plot to check
figure(3)
plot(frameNums,ypos_2_1)
hold on
plot(frameNums,xpos_2_1)
%% Camera 3 Case 1
close all
load('cam3_1.mat')
vidFrames3_1 = permute(vidFrames3_1,[2,1,3,4]);
%implay(vidFrames3_1)

[xpos_3_1,ypos_3_1] = getPos2(vidFrames3_1, 215,373,273,455);
numFrames = size(vidFrames3_1,4);

% Plot position data to look for incongruities
frameNums = 1:numFrames;
figure(1)
plot(frameNums,ypos_3_1)
hold on
plot(frameNums,xpos_3_1)

% Remove points that are out of place
badFrames = [182,197,198,199,200,201,202,203,205,206,207,208,209,210,211];

% Cut out last area with bad data
numFrames = 220;
frameNums(numFrames+1:end) = [];
xpos_3_1(numFrames+1:end) = [];
ypos_3_1(numFrames+1:end) = [];

for k = length(badFrames):-1:1
    badFrame = badFrames(k);
    frameNums(badFrame) = [];
    xpos_3_1(badFrame) = [];
    ypos_3_1(badFrame) = [];
end
% Plot to check
```

```
figure(2)
plot(frameNums,ypos_3_1)
hold on
plot(frameNums,xpos_3_1)

% Use interpotation to get data for missing points
xinterp = interp1(frameNums, xpos_3_1, badFrames, 'pchip');
yinterp = interp1(frameNums, ypos_3_1, badFrames, 'pchip');

% Add interp points back in
xpos_3_1 = xpos_3_1';
ypos_3_1 = ypos_3_1';
for k = 1:length(badFrames)
    badFrame = badFrames(k);
    xpos_3_1 = [xpos_3_1(1:badFrame-1), xinterp(k), xpos_3_1(badFrame:end)];
    ypos_3_1 = [ypos_3_1(1:badFrame-1), yinterp(k), ypos_3_1(badFrame:end)];
end
frameNums = 1:numFrames;
xpos_3_1 = xpos_3_1';
ypos_3_1 = ypos_3_1';
% Plot to check
figure(3)
plot(frameNums,ypos_3_1)
hold on
plot(frameNums,xpos_3_1)

%% Camera 1 Case 2
close all
load('cam1_2.mat')
%implay(vidFrames1_2)
[xpos_1_2,ypos_1_2] = getPos2(vidFrames1_2, 243,445,212,387);
numFrames = size(vidFrames1_2,4);

% Plot position data to look for incongruities
frameNums = 1:numFrames;
figure(1)
plot(frameNums,ypos_1_2)
hold on
plot(frameNums,xpos_1_2)

% Remove points that are out of place
badFrames =
    [11,28,29,40,41,45,50,51,52,63,93,94,120,121,122,123,139,146,152,153,159,160,166,167


for k = length(badFrames):-1:1
    badFrame = badFrames(k);
    frameNums(badFrame) = [];
    xpos_1_2(badFrame) = [];
    ypos_1_2(badFrame) = [];
end

% Plot to check
figure(2)
plot(frameNums,ypos_1_2)
```

```matlab
hold on
plot(frameNums,xpos_1_2)

% Use interpotation to get data for missing points
xinterp = interp1(frameNums, xpos_1_2, badFrames, 'pchip');
yinterp = interp1(frameNums, ypos_1_2, badFrames, 'pchip');

% Add interp points back in
xpos_1_2 = xpos_1_2';
ypos_1_2 = ypos_1_2';
for k = 1:length(badFrames)
    badFrame = badFrames(k);
    xpos_1_2 = [xpos_1_2(1:badFrame-1), xinterp(k), xpos_1_2(badFrame:end)];
    ypos_1_2 = [ypos_1_2(1:badFrame-1), yinterp(k), ypos_1_2(badFrame:end)];
end
frameNums = 1:numFrames;
xpos_1_2 = xpos_1_2';
ypos_1_2 = ypos_1_2';

% Plot to check
figure(3)
plot(frameNums,ypos_1_2)
hold on
plot(frameNums,xpos_1_2)

%% Camera 2 Case 2
close all
load('cam2_2.mat')
%implay(vidFrames2_2)
[xpos_2_2,ypos_2_2] = getPos1(vidFrames2_2, 182,457);
numFrames = size(vidFrames2_2,4);

% Plot position data to look for incongruities
frameNums = 1:numFrames;
figure(1)
plot(frameNums,ypos_2_2)
hold on
plot(frameNums,xpos_2_2)

% Remove points that are out of place
badFrames =
    [5,7,28,37,69,67,68,70,71,72,73,74,78,79,126,132,133,134,145,150,151,155,156,161,162

for k = length(badFrames):-1:1
    badFrame = badFrames(k);
    frameNums(badFrame) = [];
    xpos_2_2(badFrame) = [];
    ypos_2_2(badFrame) = [];
end

% Plot to check
figure(2)
plot(frameNums,ypos_2_2)
hold on
```

```
plot(frameNums,xpos_2_2)

% Use interpotation to get data for missing points
xinterp = interp1(frameNums, xpos_2_2, badFrames, 'pchip');
yinterp = interp1(frameNums, ypos_2_2, badFrames, 'pchip');

% Add interp points back in
xpos_2_2 = xpos_2_2';
ypos_2_2 = ypos_2_2';
for k = 1:length(badFrames)
    badFrame = badFrames(k);
    xpos_2_2 = [xpos_2_2(1:badFrame-1), xinterp(k), xpos_2_2(badFrame:end)];
    ypos_2_2 = [ypos_2_2(1:badFrame-1), yinterp(k), ypos_2_2(badFrame:end)];
end
frameNums = 1:numFrames;
xpos_2_2 = xpos_2_2';
ypos_2_2 = ypos_2_2';

% Plot to check
figure(3)
plot(frameNums,ypos_2_2)
hold on
plot(frameNums,xpos_2_2)
%% Camera 3 Case 2
close all
load('cam3_2.mat')
vidFrames3_2 = permute(vidFrames3_2,[2,1,3,4]);
%implay(vidFrames3_2)
[xpos_3_2,ypos_3_2] = getPos2(vidFrames3_2, 184,323,275,474);
numFrames = size(vidFrames3_2,4);
% Plot position data to look for incongruities
frameNums = 1:numFrames;
figure(1)
plot(frameNums,ypos_3_2)
hold on
plot(frameNums,xpos_3_2)

% Remove points that are out of place
badFrames =
    [42,44,45,58,59,71,72,82,90,96,100,101,107,111,113,116,120,122,123,126,127,134,135,14

badFrames = sort(badFrames);
for k = length(badFrames):-1:1
    badFrame = badFrames(k);
    frameNums(badFrame) = [];
    xpos_3_2(badFrame) = [];
    ypos_3_2(badFrame) = [];
end

% Plot to check
figure(2)
plot(frameNums,ypos_3_2)
hold on
plot(frameNums,xpos_3_2)

```

```
% Use interpotation to get data for missing points
xinterp = interp1(frameNums, xpos_3_2, badFrames, 'pchip');
yinterp = interp1(frameNums, ypos_3_2, badFrames, 'pchip');

% Add interp points back in
xpos_3_2 = xpos_3_2';
ypos_3_2 = ypos_3_2';
for k = 1:length(badFrames)
    badFrame = badFrames(k);
    xpos_3_2 = [xpos_3_2(1:badFrame-1), xinterp(k), xpos_3_2(badFrame:end)];
    ypos_3_2 = [ypos_3_2(1:badFrame-1), yinterp(k), ypos_3_2(badFrame:end)];
end
frameNums = 1:numFrames;
xpos_3_2 = xpos_3_2';
ypos_3_2 = ypos_3_2';

% Plot to check
figure(3)
plot(frameNums,ypos_3_2)
hold on
plot(frameNums,xpos_3_2)
%% Camera 1 Case 3
close all
load('cam1_3.mat')
implay(vidFrames1_3)
[xpos_1_3,ypos_1_3]  = getPos3(80,vidFrames1_3,238,404,223,421);
numFrames = size(vidFrames1_3,4);

% Plot position data to look for incongruities
frameNums = 1:numFrames;
figure(1)
plot(frameNums,ypos_1_3)
hold on
plot(frameNums,xpos_1_3)

%% Camera 2 Case 3
close all
load('cam2_3.mat')
%implay(vidFrames2_3)

[xpos_2_3,ypos_2_3]  = getPos3(90,vidFrames2_3,190,472,160,412);
numFrames = size(vidFrames2_3,4);

% Plot position data to look for incongruities
frameNums = 1:numFrames;
figure(1)
plot(frameNums,ypos_2_3)
hold on
plot(frameNums,xpos_2_3)

% Remove points that are out of place
badFrames = [2,3,4,5,41,42,43,49,163,164,199];
for k = length(badFrames):-1:1
    badFrame = badFrames(k);
    frameNums(badFrame) = [];
```

```
    xpos_2_3(badFrame) = [];
    ypos_2_3(badFrame) = [];
end

% Plot to check
figure(2)
plot(frameNums,ypos_2_3)
hold on
plot(frameNums,xpos_2_3)

% Use interpotation to get data for missing points
xinterp = interp1(frameNums, xpos_2_3, badFrames, 'pchip');
yinterp = interp1(frameNums, ypos_2_3, badFrames, 'pchip');

% Add interp points back in
xpos_2_3 = xpos_2_3';
ypos_2_3 = ypos_2_3';
for k = 1:length(badFrames)
    badFrame = badFrames(k);
    xpos_2_3 = [xpos_2_3(1:badFrame-1), xinterp(k), xpos_2_3(badFrame:end)];
    ypos_2_3 = [ypos_2_3(1:badFrame-1), yinterp(k), ypos_2_3(badFrame:end)];
end
frameNums = 1:numFrames;
xpos_2_3 = xpos_2_3';
ypos_2_3 = ypos_2_3';

% Plot to check
figure(3)
plot(frameNums,ypos_2_3)
hold on
plot(frameNums,xpos_2_3)
%% Camera 3 Case 3
close all
load('cam3_3.mat')
vidFrames3_3 = permute(vidFrames3_3,[2,1,3,4]);
%implay(vidFrames3_3)
[xpos_3_3,ypos_3_3]  = getPos3(80,vidFrames3_3,146,267,160,487);
%[xpos_3_3,ypos_3_3]  = getPos2(vidFrames3_3,146,267,160,487);
numFrames = size(vidFrames3_3,4);

% Plot position data to look for incongruities
frameNums = 1:numFrames;
figure(1)
plot(frameNums,ypos_3_3)
hold on
plot(frameNums,xpos_3_3)

%% Camera 1 Case 4
close all
load('cam1_4.mat')
%implay(vidFrames1_4)
[xpos_1_4,ypos_1_4]  = getPos3(80,vidFrames1_4,284,496,208,421);
numFrames = size(vidFrames1_4,4);

% Plot position data to look for incongruities
```

```
frameNums = 1:numFrames;
figure(1)
plot(frameNums,ypos_1_4)
hold on
plot(frameNums,xpos_1_4)

% Remove points that are out of place
badFrames = [8,111,112,113,114,115,190,191,192,193,194,236,237,238];

for k = length(badFrames):-1:1
    badFrame = badFrames(k);
    frameNums(badFrame) = [];
    xpos_1_4(badFrame) = [];
    ypos_1_4(badFrame) = [];
end

% Plot to check
figure(2)
plot(frameNums,ypos_1_4)
hold on
plot(frameNums,xpos_1_4)

% Use interpotation to get data for missing points
xinterp = interp1(frameNums, xpos_1_4, badFrames, 'pchip');
yinterp = interp1(frameNums, ypos_1_4, badFrames, 'pchip');

% Add interp points back in
xpos_1_4 = xpos_1_4';
ypos_1_4 = ypos_1_4';
for k = 1:length(badFrames)
    badFrame = badFrames(k);
    xpos_1_4 = [xpos_1_4(1:badFrame-1), xinterp(k), xpos_1_4(badFrame:end)];
    ypos_1_4 = [ypos_1_4(1:badFrame-1), yinterp(k), ypos_1_4(badFrame:end)];
end
frameNums = 1:numFrames;
xpos_1_4 = xpos_1_4';
ypos_1_4 = ypos_1_4';

% Plot to check
figure(3)
plot(frameNums,ypos_1_4)
hold on
plot(frameNums,xpos_1_4)

%% Camera 2 Case 4
close all
load('cam2_4.mat')
%implay(vidFrames2_4)
[xpos_2_4,ypos_2_4] = getPos3(80,vidFrames2_4,153,443,86,371);
numFrames = size(vidFrames2_4,4);

% Plot position data to look for incongruities
frameNums = 1:numFrames;
figure(1)
plot(frameNums,ypos_2_4)
```

```
hold on
plot(frameNums,xpos_2_4)

% Remove points that are out of place
badFrames = [34,43,44,59,88,138,193,198,397,398,399,400,401];
for k = length(badFrames):-1:1
    badFrame = badFrames(k);
    frameNums(badFrame) = [];
    xpos_2_4(badFrame) = [];
    ypos_2_4(badFrame) = [];
end

% Plot to check
figure(2)
plot(frameNums,ypos_2_4)
hold on
plot(frameNums,xpos_2_4)

% Use interpotation to get data for missing points
xinterp = interp1(frameNums, xpos_2_4, badFrames, 'pchip');
yinterp = interp1(frameNums, ypos_2_4, badFrames, 'pchip');

% Add interp points back in
xpos_2_4 = xpos_2_4';
ypos_2_4 = ypos_2_4';
for k = 1:length(badFrames)
    badFrame = badFrames(k);
    xpos_2_4 = [xpos_2_4(1:badFrame-1), xinterp(k), xpos_2_4(badFrame:end)];
    ypos_2_4 = [ypos_2_4(1:badFrame-1), yinterp(k), ypos_2_4(badFrame:end)];
end
frameNums = 1:numFrames;
xpos_2_4 = xpos_2_4';
ypos_2_4 = ypos_2_4';

% Plot to check
figure(3)
plot(frameNums,ypos_2_4)
hold on
plot(frameNums,xpos_2_4)
%% Camera 3 Case 4
close all
load('cam3_4.mat')
vidFrames3_4 = permute(vidFrames3_4,[2,1,3,4]);
%implay(vidFrames3_4)

[xpos_3_4,ypos_3_4]  = getPos3(65,vidFrames3_4,117,297,298,529);
numFrames = size(vidFrames3_4,4);

% Plot position data to look for incongruities
frameNums = 1:numFrames;
figure(1)
plot(frameNums,ypos_3_4)
hold on
plot(frameNums,xpos_3_4)

```

```
% Remove points that are out of place
badFrames =
    [38,39,103,104,105,160,163,165,167,201,202,231,232,233,280,302,346,347,348,349,390,3

for k = length(badFrames):-1:1
    badFrame = badFrames(k);
    frameNums(badFrame) = [];
    xpos_3_4(badFrame) = [];
    ypos_3_4(badFrame) = [];
end

% Plot to check
figure(2)
plot(frameNums,ypos_3_4)
hold on
plot(frameNums,xpos_3_4)

% Use interpotation to get data for missing points
xinterp = interp1(frameNums, xpos_3_4, badFrames, 'pchip');
yinterp = interp1(frameNums, ypos_3_4, badFrames, 'pchip');

% Add interp points back in
xpos_3_4 = xpos_3_4';
ypos_3_4 = ypos_3_4';
for k = 1:length(badFrames)
    badFrame = badFrames(k);
    xpos_3_4 = [xpos_3_4(1:badFrame-1), xinterp(k), xpos_3_4(badFrame:end)];
    ypos_3_4 = [ypos_3_4(1:badFrame-1), yinterp(k), ypos_3_4(badFrame:end)];
end
frameNums = 1:numFrames;
xpos_3_4 = xpos_3_4';
ypos_3_4 = ypos_3_4';

% Plot to check
figure(3)
plot(frameNums,ypos_3_4)
hold on
plot(frameNums,xpos_3_4)
%% Save position data
close all
save('posData.mat', '-regexp', '^xpos','^ypos')
```

alignData.m

```
clear; clc; close all;
load('posData.mat')

%%
% Plot first case
figure(1)
plot(ypos_1_1)
figure(2)
plot(ypos_2_1)
figure(3)
```

```
plot(ypos_3_1)
%%
close all;
% Align data
xpos_1_1(1:3) = [];
ypos_1_1(1:3) = [];
xpos_2_1(1:51) = [];
ypos_2_1(1:51) = [];
%xpos_3_1(1:10) = [];
%ypos_3_1(1:10) = [];

sizes = [length(xpos_1_1),length(xpos_2_1),length(xpos_3_1)];
minSize = min(sizes);
xpos_1_1(minSize+1:end) = [];
ypos_1_1(minSize+1:end) = [];
xpos_2_1(minSize+1:end) = [];
ypos_2_1(minSize+1:end) = [];
xpos_3_1(minSize+1:end) = [];
ypos_3_1(minSize+1:end) = [];

% xpos_1_1 = xpos_1_1 - mean(xpos_1_1);
% ypos_1_1 = ypos_1_1 - mean(ypos_1_1);
% ypos_2_1 = ypos_2_1 - mean(ypos_2_1);
% ypos_3_1 = ypos_3_1 - mean(ypos_3_1);

figure(1)
plot(ypos_1_1)
hold on
plot(ypos_2_1)
plot(ypos_3_1)
legend('cam 1', 'cam 2', 'cam3')
%%
close all;
% Plot second case
figure(1)
plot(ypos_1_2)
figure(2)
plot(ypos_2_2)
figure(3)
plot(ypos_3_2)

%%
close all;
% Align data
xpos_1_2(1:16) = [];
ypos_1_2(1:16) = [];
xpos_2_2(1:1) = [];
ypos_2_2(1:1) = [];
xpos_3_2(1:20) = [];
ypos_3_2(1:20) = [];

figure(1)
plot(ypos_1_2)
hold on
plot(ypos_2_2)
```

```
plot(ypos_3_2)

sizes = [length(xpos_1_2),length(xpos_2_2),length(xpos_3_2)];
minSize = min(sizes);
xpos_1_2(minSize+1:end) = [];
ypos_1_2(minSize+1:end) = [];
xpos_2_2(minSize+1:end) = [];
ypos_2_2(minSize+1:end) = [];
xpos_3_2(minSize+1:end) = [];
ypos_3_2(minSize+1:end) = [];
%%
%close all;
% Plot third case
figure(1)
plot(ypos_1_3)
hold on
plot(ypos_2_3)

plot(ypos_3_3)
legend('cam 1', 'cam 2', 'cam3')
%%
close all;
% Align data
xpos_1_3(1:13) = [];
ypos_1_3(1:13) = [];
xpos_2_3(1:39) = [];
ypos_2_3(1:39) = [];
xpos_3_3(1:7) = [];
ypos_3_3(1:7) = [];

figure(1)
plot(ypos_1_3)
hold on
plot(ypos_2_3)

plot(ypos_3_3)
legend('cam 1', 'cam 2', 'cam3')


sizes = [length(xpos_1_3),length(xpos_2_3),length(xpos_3_3)];
minSize = min(sizes);
xpos_1_3(minSize+1:end) = [];
ypos_1_3(minSize+1:end) = [];
xpos_2_3(minSize+1:end) = [];
ypos_2_3(minSize+1:end) = [];
xpos_3_3(minSize+1:end) = [];
ypos_3_3(minSize+1:end) = [];
%%
close all;
% Plot fourth case
figure(1)
plot(ypos_1_4)
hold on
plot(ypos_2_4)
plot(ypos_3_4)
```

```
legend('cam 1', 'cam 2', 'cam3')
%%
% Align data
%xpos_1_4(1:9) = [];
%ypos_1_4(1:9) = [];
xpos_2_4(1:9) = [];
ypos_2_4(1:9) = [];
%xpos_3_4(1) = [];
%ypos_3_4(1) = [];

figure(2)
plot(ypos_1_4)
hold on
plot(ypos_2_4)
plot(ypos_3_4)
legend('cam 1', 'cam 2', 'cam3')


sizes = [length(xpos_1_4),length(xpos_2_4),length(xpos_3_4)];
minSize = min(sizes);
xpos_1_4(minSize+1:end) = [];
ypos_1_4(minSize+1:end) = [];
xpos_2_4(minSize+1:end) = [];
ypos_2_4(minSize+1:end) = [];
xpos_3_4(minSize+1:end) = [];
ypos_3_4(minSize+1:end) = [];

%%
close all

snapShot1 = [xpos_1_1';ypos_1_1';xpos_2_1';ypos_2_1';xpos_3_1';ypos_3_1'];
snapShot2 = [xpos_1_2';ypos_1_2';xpos_2_2';ypos_2_2';xpos_3_2';ypos_3_2'];
snapShot3 = [xpos_1_3';ypos_1_3';xpos_2_3';ypos_2_3';xpos_3_3';ypos_3_3'];
snapShot4 = [xpos_1_4';ypos_1_4';xpos_2_4';ypos_2_4';xpos_3_4';ypos_3_4'];

mean1 = mean(snapShot1,2);
snapShot1 = snapShot1-mean1;

mean2 = mean(snapShot2,2);
snapShot2 = snapShot2-mean2;

mean3 = mean(snapShot3,2);
snapShot3 = snapShot3-mean3;

mean4 = mean(snapShot4,2);
snapShot4 = snapShot4-mean4;

figure(1)
plot(snapShot2(2,:))
hold on
plot(snapShot2(4,:))
plot(snapShot2(6,:))
legend('cam 1', 'cam 2', 'cam3')
save('snapShots.mat', '-regexp', '^snapShot')
```

performPCA.m

```
clear; clc; close all;
load('snapShots.mat')

% Perform SVD on each snapshot matrix
[U1,S1,V1] = svd(snapShot1, 'econ');
[U2,S2,V2] = svd(snapShot2, 'econ');
[U3,S3,V3] = svd(snapShot3, 'econ');
[U4,S4,V4] = svd(snapShot4, 'econ');

% Plot singular values of each matrix
figure(1)
plot(diag(S1),'.-', 'linewidth', 2, 'markersize', 20); hold on
plot(diag(S2),'.-', 'linewidth', 2, 'markersize', 20)
plot(diag(S3),'.-', 'linewidth', 2, 'markersize', 20)
plot(diag(S4),'.-', 'linewidth', 2, 'markersize', 20)
title('Singular Values for Each Case', 'fontsize', 15)
legend('Case 1', 'Case 2', 'Case 3', 'Case 4')

% Compute low rank approximations for each matrix
n = 1;
rankn_1 = U1(:,1:n)*S1(1:n,1:n)*V1(:,1:n)';
rankn_2 = U2(:,1:n)*S2(1:n,1:n)*V2(:,1:n)';
rankn_3 = U3(:,1:n)*S3(1:n,1:n)*V3(:,1:n)';
rankn_4 = U4(:,1:n)*S4(1:n,1:n)*V4(:,1:n)';

% Plot low rank approximations of positions
figure(2)
subplot(2,2,1)
plot(rankn_1(1,:), 'linewidth', 2); hold on
plot(rankn_1(3,:), 'linewidth', 2)
plot(rankn_1(5,:), 'linewidth', 2)
title('Case 1')

subplot(2,2,2)
plot(rankn_2(1,:), 'linewidth', 2); hold on
plot(rankn_2(3,:), 'linewidth', 2)
plot(rankn_2(5,:), 'linewidth', 2)
title('Case 2')

subplot(2,2,3)
plot(rankn_3(1,:), 'linewidth', 2); hold on
plot(rankn_3(3,:), 'linewidth', 2)
plot(rankn_3(5,:), 'linewidth', 2)
title('Case 3')

subplot(2,2,4)
plot(rankn_4(1,:), 'linewidth', 2); hold on
plot(rankn_4(3,:), 'linewidth', 2)
plot(rankn_4(5,:), 'linewidth', 2)
title('Case 4')

sgtitle('Low Rank Approximations of X positions', 'fontsize', 15)
legend({'Cam 1','Cam 2','Cam 3'}, 'fontsize', 10)
```

```
figure(3)
subplot(2,2,1)
plot(rankn_1(2,:), 'linewidth', 2); hold on
plot(rankn_1(4,:), 'linewidth', 2)
plot(rankn_1(6,:), 'linewidth', 2)
title('Case 1')

subplot(2,2,2)
plot(rankn_2(2,:), 'linewidth', 2); hold on
plot(rankn_2(4,:), 'linewidth', 2)
plot(rankn_2(6,:), 'linewidth', 2)
title('Case 2')

subplot(2,2,3)
plot(rankn_3(2,:), 'linewidth', 2); hold on
plot(rankn_3(4,:), 'linewidth', 2)
plot(rankn_3(6,:), 'linewidth', 2)
title('Case 3')

subplot(2,2,4)
plot(rankn_4(2,:), 'linewidth', 2); hold on
plot(rankn_4(4,:), 'linewidth', 2)
plot(rankn_4(6,:), 'linewidth', 2)
title('Case 4')

sgtitle('Low Rank Approximations of Y positions', 'fontsize', 15)
legend({'Cam 1','Cam 2','Cam 3'}, 'fontsize', 10)

%%
% Plot original positions
figure(1)
subplot(2,2,1)
plot(snapShot1(1,:), 'linewidth', 2); hold on
plot(snapShot1(3,:), 'linewidth', 2)
plot(snapShot1(5,:), 'linewidth', 2)
title('Case 1')

subplot(2,2,2)
plot(snapShot2(1,:), 'linewidth', 2); hold on
plot(snapShot2(3,:), 'linewidth', 2)
plot(snapShot2(5,:), 'linewidth', 2)
title('Case 2')

subplot(2,2,3)
plot(snapShot3(1,:), 'linewidth', 2); hold on
plot(snapShot3(3,:), 'linewidth', 2)
plot(snapShot3(5,:), 'linewidth', 2)
title('Case 3')

subplot(2,2,4)
plot(snapShot4(1,:), 'linewidth', 2); hold on
plot(snapShot4(3,:), 'linewidth', 2)
plot(snapShot4(5,:), 'linewidth', 2)
title('Case 4')
```

```
sgtitle('Actual X positions', 'fontsize', 15)
legend({'Cam 1','Cam 2','Cam 3'}, 'fontsize', 10)

figure(2)
subplot(2,2,1)
plot(snapShot1(2,:), 'linewidth', 2); hold on
plot(snapShot1(4,:), 'linewidth', 2)
plot(snapShot1(6,:), 'linewidth', 2)
title('Case 1')

subplot(2,2,2)
plot(snapShot2(2,:), 'linewidth', 2); hold on
plot(snapShot2(4,:), 'linewidth', 2)
plot(snapShot2(6,:), 'linewidth', 2)
title('Case 2')

subplot(2,2,3)
plot(snapShot3(2,:), 'linewidth', 2); hold on
plot(snapShot3(4,:), 'linewidth', 2)
plot(snapShot3(6,:), 'linewidth', 2)
title('Case 3')

subplot(2,2,4)
plot(snapShot4(2,:), 'linewidth', 2); hold on
plot(snapShot4(4,:), 'linewidth', 2)
plot(snapShot4(6,:), 'linewidth', 2)
title('Case 4')

sgtitle('Actual Y positions', 'fontsize', 15)
legend({'Cam 1','Cam 2','Cam 3'}, 'fontsize', 10)

%%
% Plot right singular vectors
subplot(2,2,1)
plot(V1(:,1:n), 'linewidth', 2)
title('Case 1')

subplot(2,2,2)
plot(V2(:,1:n), 'linewidth', 2)
title('Case 2')

subplot(2,2,3)
plot(V3(:,1:n), 'linewidth', 2)
title('Case 3')

subplot(2,2,4)
plot(V4(:,1:n), 'linewidth', 2)
title('Case 4')

sgtitle('Right Singular Vectors for Each Case', 'fontsize', 15)
```