Principal Component Analysis of Video Recordings
Seong Hyun Han
February 21, 2020

Abstract

In this assignment, principal component analysis is explored and used to empirically extract meaningful behavior of a spring-mass system for four different test case videos. Before principal component analysis can be used, the videos must be processed using various filtering and motion tracking algorithms. After the videos are processed, a data matrix can be constructed. A method called singular value decomposition is used on the constructed data matrix to extract the principal components of the recorded system and their relative singular value energies. This will allow for the reduction of dimensions through the identification of redundant and noisy information.

Sec. I Introduction and Overview

The main motivation for this assignment is to explore and understand the practically of principal component analysis (PCA). In order to realize the usefulness of the algorithm's various aspects, video recordings (converted to MATLAB files) of the displacement of a spring-mass system was analyzed. The mass in this case was a white bucket with a point light-source used to help track the bucket movement. Three cameras at different angles were used to record the motion of the spring-mass system for four tests: 1. Ideal case – mass displacement in z direction, 2. Noisy case – test 1 with camera shake noise, 3. Horizontal displacement – mass motion in both the xy plane and z direction, and 4. Horizontal displacement and rotation – test 3 with mass rotation. The experiments assume that there is no information regarding the spatial orientation or governing equations of the system. Furthermore, the collected data is oversampled in this case (redundant) due to the use of three cameras for a simple oscillatory motion in one dimension (z direction). Moreover, the video data will have inherent noise and perturbations from imperfectly recording the system. Therefore, the goal of this assignment is to empirically extract out meaningful behavior of the system.

Before PCA can be used for the video data, the data must be processed to obtain the position of the white bucket in time (per frame). In order to carry out this task, various image processing techniques will be implemented for each frame, such as masking and threshold binarization, to obtain the (x, y) coordinates of the bucket. The coordinates of the bucket for the three cameras were combined to one (6 × number of frames) data matrix to be used for PCA. PCA will be carried out by using a method called singular value decomposition (SVD). SVD allows for the extraction of the principal components of the data matrix and their relative energies. From this, the most significant principal components can be extracted, allowing us to know the number of dimensions necessary to fully capture the system. Thus, this allows for the reduction of dimensions.

Sec. II. Theoretical Background

In order to approach this assignment, several key mathematical concepts must be understood. The first concept is singular value decomposition (SVD). The cardinal concept of SVD is that it factorizes a given matrix into key constitutive components which all have a specific meaning in applications. The SVD is essentially a transformation that scales and rotates a set of vectors. The scaling and rotation of this transformation can be controlled through the

proper construction of a matrix A [1]. The full SVD of this matrix A can be factorized into the compact matrix notation of

$$A = U\Sigma V^* \quad \text{(Eqn. 1)}$$

$$U \in \mathbb{C}^{m\times m} \text{ is unitary} \quad \text{(Eqn. 2)}$$

$$\Sigma \in \mathbb{R}^{m\times n} \text{ is diagonal} \quad \text{(Eqn. 3)}$$

$$V \in \mathbb{C}^{n\times n} \text{ is unitary.} \quad \text{(Eqn. 4)}$$

The full SVD is generated by adding additional "silent" columns to the U and Σ matrices to make them unitary and allow for compatibility with rank deficient matrices. The matrix U has orthonormal columns containing the left singular vectors of A and the matrix V^* contains the right singular vectors of A. The matrix Σ has r (rank) positive diagonal singular values (σ_j) ordered with the greatest value first and then in descending order by convention (the remaining n - r columns are all zeros) [1].

The SVD can be computed by

$$\begin{aligned} A^TA &= (U\Sigma V^*)^T(U\Sigma V^*) \\ &= V\Sigma U^*U\Sigma V^* \\ &= V\Sigma^2V^* \end{aligned} \quad \text{(Eqn. 5)}$$

$$\begin{aligned} AA^T &= (U\Sigma V^*)(U\Sigma V^*)^T \\ &= U\Sigma V^*V\Sigma U^* \\ &= U\Sigma^2U^* \end{aligned} \quad \text{(Eqn. 6)}$$

$$A^TAV = V\Sigma^2 \quad \text{(Eqn. 7)}$$

$$AA^TU = U\Sigma^2. \quad \text{(Eqn. 8)}$$

By computing the normalized eigenvectors for Eqn. 7 and Eqn. 8, the orthonormal basis vectors for U and V are produced. Moreover, the singular values can be produced by taking the square root of the eigenvalues of these equations. Furthermore, the SVD makes it possible for every matrix to be diagonalized if the proper bases for the domain and range are used. Additionally, this decomposition method allows for the projection of the data onto lower dimensional representations in an algorithmic manner. One key application of SVD is to carry out PCA, where the ideal or simplified behavior of a real data set that is noisy, potentially redundant, and unoptimized can be extracted. This data set can be constructed as a matrix with $X \in \mathbb{R}^{m\times n}$ where m is the number of measurements and n is the number of collected data points. In order to remove redundancy and identify maximal variance in the data set, the covariance matrix can be considered, where covariance shows the dependencies between two variables. The covariance matrix $C_X = \frac{1}{n-1}XX^T$, where 1/(n - 1) is the normalization constant of an unbiased estimator and C_X is a $m \times m$ matrix with the diagonal representing the variance (dynamic of interest) and the off-diagonal terms representing the covariance (redundancy) of the system. Therefore, we want to order the variance from greatest to least and covariances equal to zero. Thus, we want to diagonalize the covariance matrix $C_X = V\Lambda V^{-1}$ where the basis of eigenvectors in V are called principal components and the eigenvalues in Λ gives the variance. This is exactly carried out by SVD, where the meaning of $U\Sigma V^*$ are conserved – U: principal components and Σ: eigenvalues of the scaled data matrix (Eqn. 9 and Eqn. 10) [1].

$$A = \frac{1}{\sqrt{n-1}}X \quad \text{(Eqn. 9)}$$

$$AA^T = \left(\frac{1}{\sqrt{n-1}}X\right)\left(\frac{1}{\sqrt{n-1}}X\right)^T = C_x$$
$$= U\Sigma^2U^* \quad \text{(Eqn. 10)}$$

Sec. III. Algorithm Implementation and Development

This problem is solved in MATLAB by using the methods mentioned above. The coding algorithm can be split into two main components: video processing and SVD implementation. Since the collected data is from real video recordings, the recordings had to be processed to construct a data matrix that SVD can be implemented on. Starting with **Test 1 – Video Processing** in **Appendix B. MATLAB codes**, the test 1 camera 1 video ('cam1_1.mat') was loaded. All of the video frames were stored in vidFrames1_1, a 4-D uint8 data type that stores the RGB values for a 480 × 640 pixel grid (size of video) for each video frame. The image sequences were played using the function *implay()*. This function creates a graphic user interface that allows for the determination of the horizontal and vertical pixel ranges of the bucket location in each video frame. To process each video frame, the frame was turned into a gray scale image using *rgb2gray()* – requirement for *imbinarize()*. To reduce background noise, the frame was masked using 480 × 640 matrices (*crop* - vertical crop and *crop_h* – horizontal crop) with ones at pixel ranges where the bucket was located and zeros everywhere else. In order extract only the bucket and point light source, *imbinarize()* was used to converted each frame into a logical data type (True = 1 and False = 0) based on thresholding of a manually set luminance values from 0 to 1 (white on the bucket and light source correlate to high luminance values). After this threshold was set (0.995) to only capture the bucket, a 2-D median filtering function *medfilt2()* was used filter any other small noise in the video frame. At this point, the bucket is displayed as two components, the light source and metallic/white parts of the mid-bottom of the bucket. To connect these two regions into one, each region was dilated to overlap with each other. This was done by creating a morphological rectangular structure using the *strel()* function, and dilating the logically true regions by the size of the rectangle using the *imdilate()* function. Then the overlap regions were connected together to form a continuous region using the function *bwconncomp()*. Afterwards, the center of the bucket region and its coordinate was found using the function *regionprops()*. The coordinates of the bucket is outputted as a structural data type from using *regionprops()*. Thus, the x and y coordinates of the structs had to be extracted into a cell data type using the function *struct2cell()* and then into each column vector (*x1_loc* and *y1_loc*). The same process was carried out for all three cameras, adjusting the 1. Masking pixel range, 2. Luminance value threshold, and 3. Size of the rectangular dilation structure.

The SVD implementation part is illustrated in **Test 1 – SVD** in **Appendix B. MATLAB codes.** In order to make all of the camera videos in sync, starting out from the minimum displacement position of the bucket, the frame number correlating to the minimum bucket position was found using *min()* (the minimum bucket position for camera 3 is the maximum x position). This frame number was the starting frame for each location vector and then the total length of the location vectors was truncated to the size of the location vector with the shortest length. This was done to make every location vector the same size. For each location vector (e.g., *x1_loc*), the mean of each location vector was subtracted from each element in the vector to store relative displacement values and not give more weight to higher location values. The data matrix used for SVD was constructed by taking the transpose of each location vector, constructing a 6 × number of frames matrix called *X_svd*. The SVD of the data matrix was taken using the function *svd()*, which outputs *U, S* (sigma), and *V* (Eqn. 1). To find the relative energies of the singular

values for each principal component, the square of each singular value was divided by the sum of the squares of all of the singular values (values on the diagonal of S) – if the data matrix was divided by the square root of (n - 1), the energies would correlate to variance (Eqn. 9 and Eqn. 10). Moreover, the data was projected onto the significant principal components by multiplying the transpose of *U* and *X_svd* and plotted relative to time (video frame). These two primary procedures were carried out for all four tests.

Sec. IV. Computational Results

Test 1) From Figure 1, it is evident that most of the singular value energy was captured by one principal component (~97%). The rest of the principal components had near zero energy levels. This outcome is reasonable as the video was ideal in the sense that there was little noise and the bucket was exhibiting simple harmonic motion in one direction. Additionally, the video processing algorithm did an adequate job to track the bucket motion as seen from the top left graph. Furthermore, by projecting our data onto the principal components (bottom left graph), it could be seen that most of the bucket variance was captured in the first principal component as seen by the amplitude of each curve.

Test 2) From Figure 2, it is evident that more than half of the singular value energy was captured by the first principal component (~64%). However, the second and third principal components resulted in energy values (in percentage) of ~17% and ~12%, respectively. This outcome is somewhat reasonable as the camera was shaking a lot when recording the bucket motion. Therefore, along with the simple harmonic motion, the SVD would have taken into consideration the variance due to the noise. Thus, it seems like the first two or three principal components are relevant to capture the behavior of the system, even though the bucket is actually moving in one dimension. Furthermore, by projecting our data onto the principal components (bottom left graph), it could be seen that the majority of the bucket variance was captured in the first principal component with the second and third components capturing the variance primarily introduced through noise.

Test 3) From Figure 3, it is evident that only 47% of the singular value energy was captured by the first principal component. However, the second, third, and fourth principal components resulted in energy values (in percentage) of ~32%, ~12%, and ~7%, respectively. This outcome is somewhat reasonable as the bucket was exhibiting both horizontal pendulum motion along with simple harmonic oscillation. Therefore, two principal components would be needed to adequately capture this behavior as seen in the figure. It seems like inconsistent

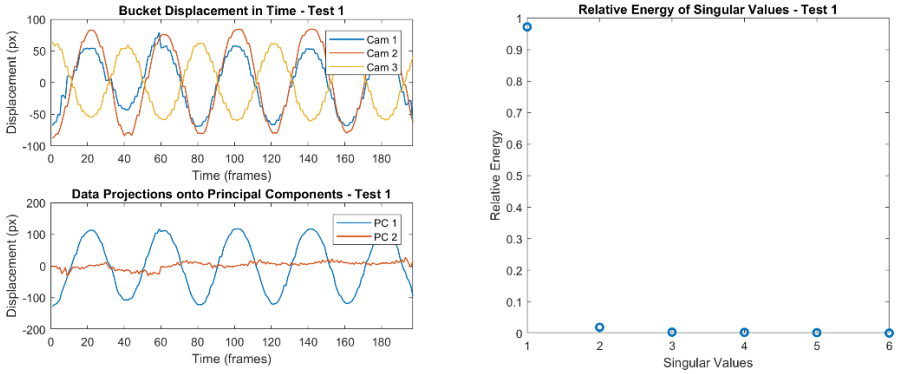

Figure 1. (*top left*) Bucket displacement in time in the z direction – Cam 3's minimum displacement is the maximum displacement of the other two cameras. (*bottom left*) Projection of the data components onto the principal components (PCs) – relevant PCs only shown. (*right*) Relative energy of each singular value (SV): SV 1 = 0.97 and SV 2 = 0.02.

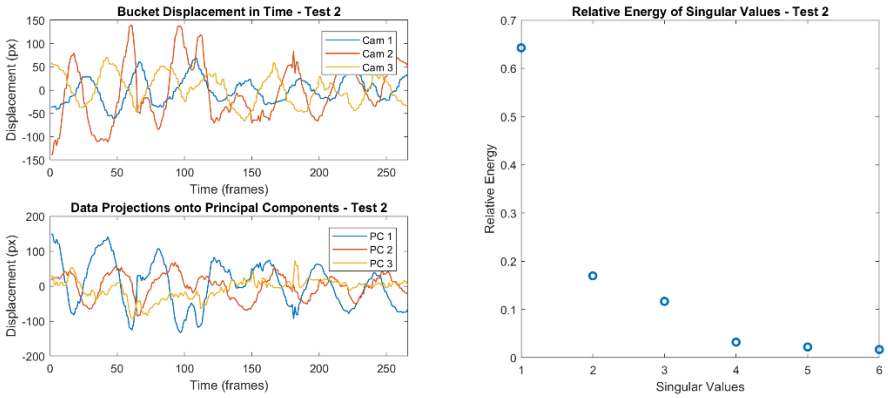

Figure 2. (*top left*) Bucket displacement in time in the z direction – Cam 3's minimum displacement is the maximum displacement of the other two cameras – distortion seen. (*bottom left*) Projection of the data components onto the principal components (PCs) – relevant PCs only shown. (*right*) Relative energy of each singular value (SV): SV 1 = 0.64, SV 2 = 0.17, and SV 3 = 0.12.

horizontal motion along with some noise contributed to increased energy values in the third and fourth principal component values. Furthermore, by projecting our data onto the principal components (bottom left graph), it could be seen that the majority of the bucket variance was captured in the first and second principal component with the third and fourth components capturing the variance likely introduced through noise.

Test 4) From Figure 4, it is evident that only 77% of the singular value energy was captured by the first principal component. However, the second and third principal components resulted in energy values (in percentage) of ~12% and ~6%, respectively. This outcome is somewhat reasonable as the bucket was exhibiting horizontal pendulum motion, simple harmonic oscillation, and rotation. Therefore, three principal components may be needed to adequately capture this behavior. However, due to the short horizontal displacement time and relatively little rotational displacement, the energy values in the second and third principal component values seem to be relatively low. Also, it may be hard to capture three-dimensional motion with 2D videos. Furthermore, by projecting our data onto the principal components (bottom left graph), it could be seen that the majority of the bucket variance was captured in the first principal component with the second and third component capturing a small amount of variance.

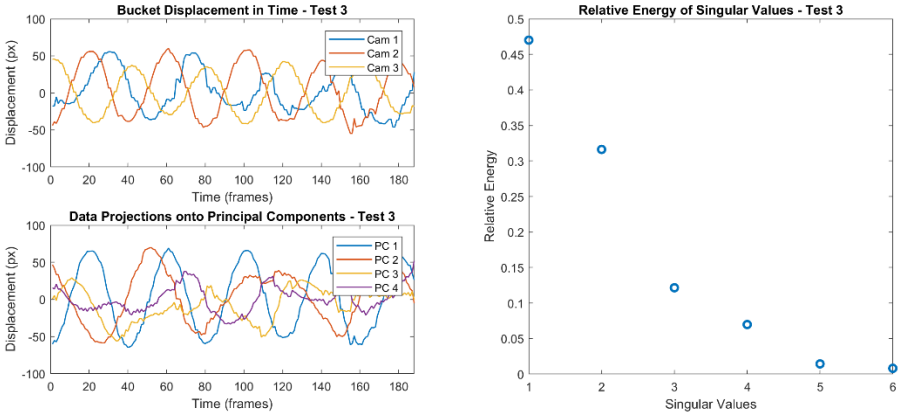

Figure 3. (*top left*) Bucket displacement in time in the z direction – Cam 3's minimum displacement is the maximum displacement of the other two cameras – small shift seen. (*bottom left*) Projection of the data components onto the principal components (PCs) – relevant PCs only shown. (*right*) Relative energy of each singular value (SV): SV 1 = 0.47, SV 2 = 0.32, SV 3 = 0.12, and SV 4 = 0.07.

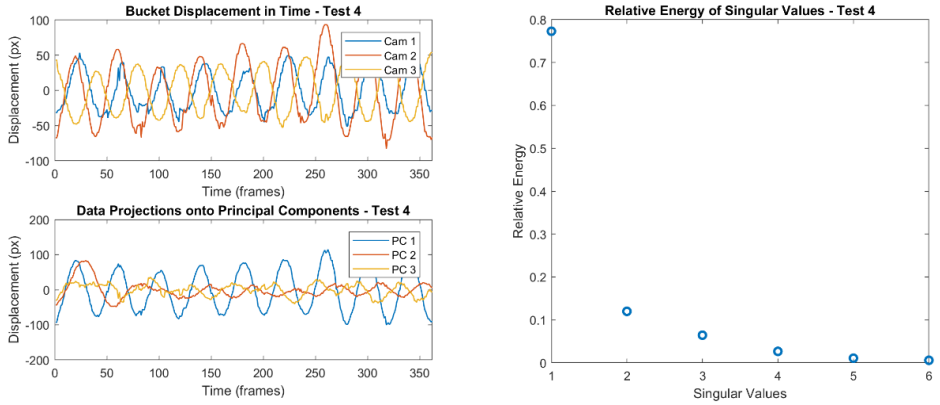

Figure 4. (*top left*) Bucket displacement in time in the z direction – Cam 3's minimum displacement is the maximum displacement of the other two cameras. (*bottom left*) Projection of the data components onto the principal components (PCs) – relevant PCs only shown. (*right*) Relative energy of each singular value (SV): SV 1 = 0.77, SV 2 = 0.12, and SV 3 = 0.06.

Sec. V. Summary and Conclusions

All in all, principal component analysis was carried out using singular value decomposition analysis. Through the use of this algorithm we were able to find the motion of the bucket, singular values, and projection of the data onto the principal components. This assignment allowed us to develop and implement motion tracking algorithms to track a specific object from raw data, process it into a refined data matrix, and find the important dimensions of each test case using SVD. Thus, the dimensionality of the system could be reduced based on the energy of the singular values that each principal component captured. Lastly, this assignment allowed us to realize the significant and impact of PCA in extracting meaningful behavior of an unknown system.

Appendix A. MATLAB functions used and brief implementation explanation

- **implay:** play movies, videos, or image sequences.
- **rgb2gray():** covert RGB image or colormap to grayscale.
- **imbinarize():** binarize grayscale 2D image by thresholding.
- **medfilt2():** 2-D median filtering.
- **strel():** create morphological structuring element.
- **imdilate():** dilate image.
- **bwconncomp():** find connected components in binary image.
- **regionprops():** measure properties of image region – location in this case.
- **struct2cell():** convert structure array to cell array.
- **svd():** take the singular value decomposition of a data matrix.

Appendix B. MATLAB codes

```
%% AMATH 482 Homework 3
% Seong Hyun Han
% 2/20/20

%% Test 1 - Video Processing
%% Cam1_1

%Starter code - load video and store data in matrix X
clear all; close all; clc
load('cam1_1.mat')
% implay(vidFrames1_1) play each fram

[y_res, x_res, rgb, numFrames] = size(vidFrames1_1);
position = [];
for j = 1:numFrames
X = vidFrames1_1(:,:,:,j); % read each frame
gray_X = rgb2gray(X); % change to gray scale

% crop gray scale image to extract only the bucket
[y_len,x_len,~] = size(gray_X);
crop = zeros(y_len,x_len);
c_start = 290;
c_end = 390;
c_width = (c_end - c_start) + 1;
crop(1:y_len, c_start:c_end) = ones(y_len, c_width);
cg_X = gray_X(1:y_len, 1:x_len).*uint8(crop);

% extract only the bucket/light based on
BW = imbinarize(cg_X, 0.995); % binarize grayscale image by thresholding
BW_filt = medfilt2(BW,'symmetric'); % 2D median filtering
se = strel('rectangle',[55;25]); % create morphological structuring element
BW_fd = imdilate(BW_filt, se); % performs dilation using the structuring element

% find connected components in binary image and position
CC = bwconncomp(BW_fd);
S = regionprops(CC, 'Centroid');
position = [position; S];

% imshow(BW_fd);
% drawnow
end

% convert structure array to cell array
pos_cell = struct2cell(position);
x1_loc = [];
y1_loc = y_res.*ones(length(pos_cell),1);
for i = 1:length(pos_cell)
    x1_loc = [x1_loc; pos_cell{1,i}(1,1)];
    y1_loc(i, 1) = y1_loc(i, 1) - pos_cell{1,i}(1,2);
end

figure(1)
subplot(1,3,1)
plot(x1_loc, y1_loc,'.','LineWidth',9)
set(gca,'Xlim',[0,x_res],'Ylim',[0,y_res],'Fontsize',10)
xticks(0:50:x_res)
xlabel('x (px)');
ylabel('y (px)');
title('Bucket Positions - Test 1 Camera 1');

% Cam2_1
load('cam2_1.mat')
% implay(vidFrames2_1)

[y_res, x_res, rgb, numFrames] = size(vidFrames2_1);
position = [];
for j = 1:numFrames
X = vidFrames2_1(:,:,:,j); % read each frame
gray_X = rgb2gray(X); % change to gray scale
```

```
% crop gray scale image to extract only the bucket
[y_len,x_len,~] = size(gray_X);
crop = zeros(y_len,x_len);
c_start = 230;
c_end = 380;
c_width = (c_end - c_start) + 1;
crop(1:y_len, c_start:c_end) = ones(y_len, c_width);
cg_X = gray_X(1:y_len, 1:x_len).*uint8(crop);

% extract only the bucket/light based on
BW = imbinarize(cg_X, 0.97); % binarize grayscale image by thresholding
BW_filt = medfilt2(BW,'symmetric'); % 2D median filtering
se = strel('rectangle',[60;30]); % create morphological structuring element
BW_fd = imdilate(BW_filt, se); % performs dilation using the structuring element

% find connected components in binary image and position
CC = bwconncomp(BW_fd);
S = regionprops(CC, 'Centroid');
position = [position; S];

% imshow(BW_fd);
% drawnow
end

% convert structure array to cell array
pos_cell = struct2cell(position);
x2_loc = [];
y2_loc = y_res.*ones(length(pos_cell),1);
for i = 1:length(pos_cell)
    x2_loc = [x2_loc; pos_cell{1,i}(1,1)];
    y2_loc(i, 1) = y2_loc(i, 1) - pos_cell{1,i}(1,2);
end

subplot(1,3,2)
plot(x2_loc, y2_loc,'.','LineWidth',9)
set(gca,'Xlim',[0,x_res],'Ylim',[0,y_res],'Fontsize',10)
xticks(0:50:x_res)
xlabel('x (px)');
ylabel('y (px)');
title('Bucket Positions - Test 1 Camera 2');

% Cam3_1

load('cam3_1.mat')
% implay(vidFrames3_1)

[y_res, x_res, rgb, numFrames] = size(vidFrames3_1);
position = [];
for j = 1:numFrames
X = vidFrames3_1(:,:,:,j); % read each frame
gray_X = rgb2gray(X); % change to gray scale

% crop gray scale image to extract only the bucket (horizontal)
[y_len,x_len,~] = size(gray_X);
crop_h = zeros(y_len,x_len);
ch_start = 230;
ch_end = 345;
ch_width = (ch_end - ch_start) + 1;
crop_h(ch_start:ch_end, 1:x_len) = ones(ch_width, x_len);
cgh_X = gray_X(1:y_len, 1:x_len).*uint8(crop_h);

% crop gray scale image to extract only the bucket (vertical)
crop = zeros(y_len,x_len);
c_start = 250;
c_end = 500;
c_width = (c_end - c_start) + 1;
crop(1:y_len, c_start:c_end) = ones(y_len, c_width);
cg_X = cgh_X.*uint8(crop);

% extract only the bucket/light based on
```

```
BW = imbinarize(cg_X, 0.90); % binarize grayscale image by thresholding
BW_filt = medfilt2(BW,'symmetric'); % 2D median filtering
se = strel('rectangle',[30;60]); % create morphological structuring element
BW_fd = imdilate(BW_filt, se); % performs dilation using the structuring element

% find connected components in binary image and position
CC = bwconncomp(BW_fd);
S = regionprops(CC, 'Centroid');
position = [position; S];

% imshow(BW_fd);
% drawnow
end

% convert structure array to cell array
pos_cell = struct2cell(position);
x3_loc = [];
y3_loc = y_res.*ones(length(pos_cell),1);
for i = 1:length(pos_cell)
    x3_loc = [x3_loc; pos_cell{1,i}(1,1)];
    y3_loc(i, 1) = y3_loc(i, 1) - pos_cell{1,i}(1,2);
end

subplot(1,3,3)
plot(x3_loc, y3_loc,'.','LineWidth',9)
set(gca,'Xlim',[0,x_res],'Ylim',[0,y_res],'Fontsize',10)
xticks(0:50:x_res)
xlabel('x (px)');
ylabel('y (px)');
title('Bucket Positions - Test 1 Camera 3');

%% Test 1 - SVD
% make all of the camera videos in sync
[~,y1_min] = min(y1_loc(1:50));
[~,y2_min] = min(y2_loc(1:50));
[~,x3_max] = max(x3_loc(1:50));

x1_loc = x1_loc(y1_min:end);
y1_loc = y1_loc(y1_min:end);

x2_loc = x2_loc(y2_min:end);
y2_loc = y2_loc(y2_min:end);

x3_loc = x3_loc(x3_max:end);
y3_loc = y3_loc(x3_max:end);

% find an averaged x and y coordinate system for each camera
% to reduce the effect of the svd giving more weight on just
% absolute location values

% average row values
x1_ave = mean(x1_loc);
x2_ave = mean(x2_loc);
x3_ave = mean(x3_loc);

y1_ave = mean(y1_loc);
y2_ave = mean(y2_loc);
y3_ave = mean(y3_loc);

% find the minimum matrix size
min_size = min(length(x1_loc), length(x2_loc));
min_size = min(min_size, length(x3_loc));

% reduces matrix size to minimum matrix size
x1 = x1_loc(1:min_size);
x2 = x2_loc(1:min_size);
x3 = x3_loc(1:min_size);

y1 = y1_loc(1:min_size);
y2 = y2_loc(1:min_size);
y3 = y3_loc(1:min_size);
```

```
% subtract each row element by its average
for n = 1:length(x1)
    x1(n,1) = x1(n,1) - x1_ave;
    x2(n,1) = x2(n,1) - x2_ave;
    x3(n,1) = x3(n,1) - x3_ave;

    y1(n,1) = y1(n,1) - y1_ave;
    y2(n,1) = y2(n,1) - y2_ave;
    y3(n,1) = y3(n,1) - y3_ave;
end

% Contruct coordinates into a single matrix
X_svd = [x1';y1';x2';y2';x3';y3'];

% Take the SVD of the single matrix using "economy size"
% decomposition - low-dimensional approximations
[U,S,V] = svd(X_svd,'econ');

% Find the energy of the singular values by dividing each singular
% value by the average of all of the singular values
S_col = diag(S);

for j = 1:length(S_col)
    S_col(j,1) = S_col(j,1)*S_col(j,1);
end

S_sum = sum(S_col);
for j = 1:length(S_col)
    S_col(j,1) = S_col(j,1)/S_sum;
end
S_diag = diag(S_col);

% Singular value energy plot
figure(2)
subplot(2,2,[2, 4])
plot(1:length(S_col), S_col, 'o', 'LineWidth', 2)
set(gca,'Fontsize',10)
xlabel('Singular Values');
ylabel('Relative Energy');
title('Relative Energy of Singular Values - Test 1');

% plot of all camera angles showing major variation direction (z)
subplot(2,2,1)
plot(1:length(y1),y1,1:length(y1),y2,1:length(y1),x3,'LineWidth',1)
set(gca,'Xlim',[0,length(y1)],'Ylim',[-100,100],'Fontsize',10)
legend("Cam 1","Cam 2","Cam 3")
xlabel('Time (frames)');
ylabel('Displacement (px)');
title('Bucket Displacement in Time - Test 1');

% plot of data projections onto pricipal components
subplot(2,2,3)
X_proj = U'*X_svd;
plot(1:length(X_proj(1,:)), X_proj(1,:),'LineWidth',1)
hold on
plot(1:length(X_proj(2,:)), X_proj(2,:),'LineWidth',1)
legend("PC 1","PC 2")
set(gca,'Xlim',[0,length(X_proj(1,:))],'Fontsize',10)
xlabel('Time (frames)');
ylabel('Displacement (px)');
title('Data Projections onto Principal Components - Test 1');
%% Test 2 - Video Processing

%% Cam1_2
clear all; close all; clc
load('cam1_2.mat')
% implay(vidFrames1_2)

[y_res, x_res, rgb, numFrames] = size(vidFrames1_2);
position = [];
```

```
for j = 1:numFrames
X = vidFrames1_2(:,:,:,j); % read each frame
gray_X = rgb2gray(X); % change to gray scale

% crop gray scale image to extract only the bucket (horizontal)
[y_len,x_len,~] = size(gray_X);
crop_h = zeros(y_len,x_len);
ch_start = 200;
ch_end = 440;
ch_width = (ch_end - ch_start) + 1;
crop_h(ch_start:ch_end, 1:x_len) = ones(ch_width, x_len);
cgh_X = gray_X(1:y_len, 1:x_len).*uint8(crop_h);

% crop gray scale image to extract only the bucket (vertical)
crop = zeros(y_len,x_len);
c_start = 300;
c_end = 440;
c_width = (c_end - c_start) + 1;
crop(1:y_len, c_start:c_end) = ones(y_len, c_width);
cg_X = cgh_X.*uint8(crop);

% extract only the bucket/light based on
BW = imbinarize(cg_X, 0.92); % binarize grayscale image by thresholding
BW_filt = medfilt2(BW,'symmetric'); % 2D median filtering
se = strel('rectangle',[55;50]); % create morphological structuring element
BW_fd = imdilate(BW_filt, se); % performs dilation using the structuring element

% find connected components in binary image and position
CC = bwconncomp(BW_fd);
S = regionprops(CC, 'Centroid');
position = [position; S];

% imshow(BW_fd);
% drawnow
end

% convert structure array to cell array
pos_cell = struct2cell(position);
x1_loc = [];
y1_loc = y_res.*ones(length(pos_cell),1);
for i = 1:length(pos_cell)
    x1_loc = [x1_loc; pos_cell{1,i}(1,1)];
    y1_loc(i, 1) = y1_loc(i, 1) - pos_cell{1,i}(1,2);
end

figure(8)
subplot(1,3,1)
plot(x1_loc, y1_loc,'.','LineWidth',9)
set(gca,'Xlim',[0,x_res],'Ylim',[0,y_res],'Fontsize',10)
xticks(0:50:x_res)
xlabel('x (px)');
ylabel('y (px)');
title('Bucket Positions - Test 2 Camera 1');

% Cam2_2

load('cam2_2.mat')
% implay(vidFrames2_2)

[y_res, x_res, rgb, numFrames] = size(vidFrames2_2);
position = [];
for j = 1:numFrames
X = vidFrames2_2(:,:,:,j); % read each frame
gray_X = rgb2gray(X); % change to gray scale

% crop gray scale image to extract only the bucket (horizontal)
[y_len,x_len,~] = size(gray_X);
crop_h = zeros(y_len,x_len);
ch_start = 60;
ch_end = 420;
ch_width = (ch_end - ch_start) + 1;
```

```
crop_h(ch_start:ch_end, 1:x_len) = ones(ch_width, x_len);
cgh_X = gray_X(1:y_len, 1:x_len).*uint8(crop_h);

% crop gray scale image to extract only the bucket (vertical)
crop = zeros(y_len,x_len);
c_start = 180;
c_end = 440;
c_width = (c_end - c_start) + 1;
crop(1:y_len, c_start:c_end) = ones(y_len, c_width);
cg_X = cgh_X.*uint8(crop);

% extract only the bucket/light based on
BW = imbinarize(cg_X, 0.965); % binarize grayscale image by thresholding
BW_filt = medfilt2(BW,'symmetric'); % 2D median filtering
se = strel('rectangle',[80;85]); % create morphological structuring element
BW_fd = imdilate(BW_filt, se); % performs dilation using the structuring element

% find connected components in binary image and position
CC = bwconncomp(BW_fd);
S = regionprops(CC, 'Centroid');
position = [position; S];

% imshow(BW_fd);
% drawnow
end

% convert structure array to cell array
pos_cell = struct2cell(position);
x2_loc = [];
y2_loc = y_res.*ones(length(pos_cell),1);
for i = 1:length(pos_cell)
    x2_loc = [x2_loc; pos_cell{1,i}(1,1)];
    y2_loc(i, 1) = y2_loc(i, 1) - pos_cell{1,i}(1,2);
end

subplot(1,3,2)
plot(x2_loc, y2_loc,'.','LineWidth',9)
set(gca,'Xlim',[0,x_res],'Ylim',[0,y_res],'Fontsize',10)
xticks(0:50:x_res)
xlabel('x (px)');
ylabel('y (px)');
title('Bucket Positions - Test 2 Camera 2');

% Cam3_2

load('cam3_2.mat')
% implay(vidFrames3_2)

[y_res, x_res, rgb, numFrames] = size(vidFrames3_2);
position = [];
for j = 1:numFrames
X = vidFrames3_2(:,:,:,j); % read each frame
gray_X = rgb2gray(X); % change to gray scale

% crop gray scale image to extract only the bucket (horizontal)
[y_len,x_len,~] = size(gray_X);
crop_h = zeros(y_len,x_len);
ch_start = 190;
ch_end = 346;
ch_width = (ch_end - ch_start) + 1;
crop_h(ch_start:ch_end, 1:x_len) = ones(ch_width, x_len);
cgh_X = gray_X(1:y_len, 1:x_len).*uint8(crop_h);

% crop gray scale image to extract only the bucket (vertical)
crop = zeros(y_len,x_len);
c_start = 286;
c_end = 495;
c_width = (c_end - c_start) + 1;
crop(1:y_len, c_start:c_end) = ones(y_len, c_width);
cg_X = cgh_X.*uint8(crop);
```

```
% extract only the bucket/light based on
BW = imbinarize(cg_X, 0.90); % binarize grayscale image by thresholding
BW_filt = medfilt2(BW,'symmetric'); % 2D median filtering
se = strel('rectangle',[30;60]); % create morphological structuring element
BW_fd = imdilate(BW_filt, se); % performs dilation using the structuring element

% find connected components in binary image and position
CC = bwconncomp(BW_fd);
S = regionprops(CC, 'Centroid');
position = [position; S];

% imshow(BW_fd);
% drawnow
end

% convert structure array to cell array
pos_cell = struct2cell(position);
x3_loc = [];
y3_loc = y_res.*ones(length(pos_cell),1);
for i = 1:length(pos_cell)
    x3_loc = [x3_loc; pos_cell{1,i}(1,1)];
    y3_loc(i, 1) = y3_loc(i, 1) - pos_cell{1,i}(1,2);
end

subplot(1,3,3)
plot(x3_loc, y3_loc,'.','LineWidth',9)
set(gca,'Xlim',[0,x_res],'Ylim',[0,y_res],'Fontsize',10)
xticks(0:50:x_res)
xlabel('x (px)');
ylabel('y (px)');
title('Bucket Positions - Test 2 Camera 3');

%% Test 2 - SVD
% make all of the camera videos in sync
[~,y1_min] = min(y1_loc(1:50));
[~,y2_min] = min(y2_loc(1:50));
[~,x3_max] = max(x3_loc(1:50));

x1_loc = x1_loc(y1_min:end);
y1_loc = y1_loc(y1_min:end);

x2_loc = x2_loc(y2_min:end);
y2_loc = y2_loc(y2_min:end);

x3_loc = x3_loc(x3_max:end);
y3_loc = y3_loc(x3_max:end);

% find an averaged x and y coordinate system for each camera
% to reduce the effect of the svd giving more weight on just
% absolute location values

% average row values
x1_ave = mean(x1_loc);
x2_ave = mean(x2_loc);
x3_ave = mean(x3_loc);

y1_ave = mean(y1_loc);
y2_ave = mean(y2_loc);
y3_ave = mean(y3_loc);

% find the minimum matrix size
min_size = min(length(x1_loc), length(x2_loc));
min_size = min(min_size, length(x3_loc));

% reduces matrix size to minimum matrix size
x1 = x1_loc(1:min_size);
x2 = x2_loc(1:min_size);
x3 = x3_loc(1:min_size);

y1 = y1_loc(1:min_size);
y2 = y2_loc(1:min_size);
```

```
y3 = y3_loc(1:min_size);

% subtract each row element by its average
for n = 1:length(x1)
    x1(n,1) = x1(n,1) - x1_ave;
    x2(n,1) = x2(n,1) - x2_ave;
    x3(n,1) = x3(n,1) - x3_ave;

    y1(n,1) = y1(n,1) - y1_ave;
    y2(n,1) = y2(n,1) - y2_ave;
    y3(n,1) = y3(n,1) - y3_ave;
end

% Contruct coordinates into a single matrix
X_svd = [x1';y1';x2';y2';x3';y3'];

% Take the SVD of the single matrix using "economy size"
% decomposition - low-dimensional approximations
[U,S,V] = svd(X_svd,'econ');

% Find the energy of the singular values by dividing each singular
% value by the average of all of the singular values
S_col = diag(S);

for j = 1:length(S_col)
    S_col(j,1) = S_col(j,1)*S_col(j,1);
end

S_sum = sum(S_col);
for j = 1:length(S_col)
    S_col(j,1) = S_col(j,1)/S_sum;
end
S_diag = diag(S_col);

% Singular value energy plot
figure(9)
subplot(2,2,[2, 4])
plot(1:length(S_col), S_col, 'o', 'LineWidth', 2)
set(gca,'Fontsize',10)
xlabel('Singular Values');
ylabel('Relative Energy');
title('Relative Energy of Singular Values - Test 2');

% plot of all camera angles showing major variation direction (z)
subplot(2,2,1)
plot(1:length(y1),y1,1:length(y1),y2,1:length(y1),x3,'LineWidth',1)
set(gca,'Xlim',[0,length(y1)],'Ylim',[-150,150],'Fontsize',10)
legend("Cam 1","Cam 2","Cam 3")
xlabel('Time (frames)');
ylabel('Displacement (px)');
title('Bucket Displacement in Time - Test 2');

% plot of data projections onto pricipal components
subplot(2,2,3)
X_proj = U'*X_svd;
plot(1:length(X_proj(1,:)), X_proj(1,:),'LineWidth',1)
hold on
plot(1:length(X_proj(2,:)), X_proj(2,:),'LineWidth',1)
plot(1:length(X_proj(3,:)), X_proj(3,:),'LineWidth',1)
legend("PC 1","PC 2","PC 3")
set(gca,'Xlim',[0,length(X_proj(1,:))],'Fontsize',10)
xlabel('Time (frames)');
ylabel('Displacement (px)');
title('Data Projections onto Principal Components - Test 2');

%% Test 3 - Video Processing

%% Cam1_3
clear all; close all; clc
load('cam1_3.mat')
% implay(vidFrames1_3)
```

```
[y_res, x_res, rgb, numFrames] = size(vidFrames1_3);
position = [];
for j = 1:numFrames
X = vidFrames1_3(:,:,:,j); % read each frame
gray_X = rgb2gray(X); % change to gray scale

% crop gray scale image to extract only the bucket (horizontal)
[y_len,x_len,~] = size(gray_X);
crop_h = zeros(y_len,x_len);
ch_start = 233;
ch_end = 415;
ch_width = (ch_end - ch_start) + 1;
crop_h(ch_start:ch_end, 1:x_len) = ones(ch_width, x_len);
cgh_X = gray_X(1:y_len, 1:x_len).*uint8(crop_h);

% crop gray scale image to extract only the bucket (vertical)
crop = zeros(y_len,x_len);
c_start = 280;
c_end = 401;
c_width = (c_end - c_start) + 1;
crop(1:y_len, c_start:c_end) = ones(y_len, c_width);
cg_X = cgh_X.*uint8(crop);

% extract only the bucket/light based on
BW = imbinarize(cg_X, 0.94); % binarize grayscale image by thresholding
BW_filt = medfilt2(BW,'symmetric'); % 2D median filtering
se = strel('rectangle',[70;55]); % create morphological structuring element
BW_fd = imdilate(BW_filt, se); % performs dilation using the structuring element

% find connected components in binary image and position
CC = bwconncomp(BW_fd);
S = regionprops(CC, 'Centroid');
position = [position; S];

% imshow(BW_fd);
% drawnow
end

% convert structure array to cell array
pos_cell = struct2cell(position);
x1_loc = [];
y1_loc = y_res.*ones(length(pos_cell),1);
for i = 1:length(pos_cell)
    x1_loc = [x1_loc; pos_cell{1,i}(1,1)];
    y1_loc(i, 1) = y1_loc(i, 1) - pos_cell{1,i}(1,2);
end

figure(15)
subplot(1,3,1)
plot(x1_loc, y1_loc,'.','LineWidth',9)
set(gca,'Xlim',[0,x_res],'Ylim',[0,y_res],'Fontsize',10)
xticks(0:50:x_res)
xlabel('x (px)');
ylabel('y (px)');
title('Bucket Positions - Test 3 Camera 1');

% Cam2_3

load('cam2_3.mat')
% implay(vidFrames2_3)

[y_res, x_res, rgb, numFrames] = size(vidFrames2_3);
position = [];
for j = 1:numFrames
X = vidFrames2_3(:,:,:,j); % read each frame
gray_X = rgb2gray(X); % change to gray scale

% crop gray scale image to extract only the bucket (horizontal)
[y_len,x_len,~] = size(gray_X);
crop_h = zeros(y_len,x_len);
```

```
ch_start = 180;
ch_end = 407;
ch_width = (ch_end - ch_start) + 1;
crop_h(ch_start:ch_end, 1:x_len) = ones(ch_width, x_len);
cgh_X = gray_X(1:y_len, 1:x_len).*uint8(crop_h);

% crop gray scale image to extract only the bucket (vertical)
crop = zeros(y_len,x_len);
c_start = 205;
c_end = 426;
c_width = (c_end - c_start) + 1;
crop(1:y_len, c_start:c_end) = ones(y_len, c_width);
cg_X = cgh_X.*uint8(crop);

% extract only the bucket/light based on
BW = imbinarize(cg_X, 0.95); % binarize grayscale image by thresholding
BW_filt = medfilt2(BW,'symmetric'); % 2D median filtering
se = strel('rectangle',[80;85]); % create morphological structuring element
BW_fd = imdilate(BW_filt, se); % performs dilation using the structuring element

% find connected components in binary image and position
CC = bwconncomp(BW_fd);
S = regionprops(CC, 'Centroid');
position = [position; S];

% imshow(BW_fd);
% drawnow
end

% convert structure array to cell array
pos_cell = struct2cell(position);
x2_loc = [];
y2_loc = y_res.*ones(length(pos_cell),1);
for i = 1:length(pos_cell)
    x2_loc = [x2_loc; pos_cell{1,i}(1,1)];
    y2_loc(i, 1) = y2_loc(i, 1) - pos_cell{1,i}(1,2);
end

subplot(1,3,2)
plot(x2_loc, y2_loc,'.','LineWidth',9)
set(gca,'Xlim',[0,x_res],'Ylim',[0,y_res],'Fontsize',10)
xticks(0:50:x_res)
xlabel('x (px)');
ylabel('y (px)');
title('Bucket Positions - Test 3 Camera 2');

% Cam3_3
load('cam3_3.mat')
% implay(vidFrames3_3)

[y_res, x_res, rgb, numFrames] = size(vidFrames3_3);
position = [];
for j = 1:numFrames
X = vidFrames3_3(:,:,:,j); % read each frame
gray_X = rgb2gray(X); % change to gray scale

% crop gray scale image to extract only the bucket (horizontal)
[y_len,x_len,~] = size(gray_X);
crop_h = zeros(y_len,x_len);
ch_start = 150;
ch_end = 355;
ch_width = (ch_end - ch_start) + 1;
crop_h(ch_start:ch_end, 1:x_len) = ones(ch_width, x_len);
cgh_X = gray_X(1:y_len, 1:x_len).*uint8(crop_h);

% crop gray scale image to extract only the bucket (vertical)
crop = zeros(y_len,x_len);
c_start = 263;
c_end = 465;
c_width = (c_end - c_start) + 1;
crop(1:y_len, c_start:c_end) = ones(y_len, c_width);
```

```
cg_X = cgh_X.*uint8(crop);

% extract only the bucket/light based on
BW = imbinarize(cg_X, 0.90); % binarize grayscale image by thresholding
BW_filt = medfilt2(BW,'symmetric'); % 2D median filtering
se = strel('rectangle',[30;60]); % create morphological structuring element
BW_fd = imdilate(BW_filt, se); % performs dilation using the structuring element

% find connected components in binary image and position
CC = bwconncomp(BW_fd);
S = regionprops(CC, 'Centroid');
position = [position; S];

% imshow(BW_fd);
% drawnow
end

% convert structure array to cell array
pos_cell = struct2cell(position);
x3_loc = [];
y3_loc = y_res.*ones(length(pos_cell),1);
for i = 1:length(pos_cell)
    x3_loc = [x3_loc; pos_cell{1,i}(1,1)];
    y3_loc(i, 1) = y3_loc(i, 1) - pos_cell{1,i}(1,2);
end

subplot(1,3,3)
plot(x3_loc, y3_loc,'.','LineWidth',9)
set(gca,'Xlim',[0,x_res],'Ylim',[0,y_res],'Fontsize',10)
xticks(0:50:x_res)
xlabel('x (px)');
ylabel('y (px)');
title('Bucket Positions - Test 3 Camera 3');

%% Test 3 - SVD
% make all of the camera videos in sync
[~,y1_min] = min(y1_loc(1:50));
[~,y2_min] = min(y2_loc(1:50));
[~,x3_max] = max(x3_loc(1:50));

x1_loc = x1_loc(y1_min:end);
y1_loc = y1_loc(y1_min:end);

x2_loc = x2_loc(y2_min:end);
y2_loc = y2_loc(y2_min:end);

x3_loc = x3_loc(x3_max:end);
y3_loc = y3_loc(x3_max:end);

% find an averaged x and y coordinate system for each camera
% to reduce the effect of the svd giving more weight on just
% absolute location values

% average row values
x1_ave = mean(x1_loc);
x2_ave = mean(x2_loc);
x3_ave = mean(x3_loc);

y1_ave = mean(y1_loc);
y2_ave = mean(y2_loc);
y3_ave = mean(y3_loc);

% find the minimum matrix size
min_size = min(length(x1_loc), length(x2_loc));
min_size = min(min_size, length(x3_loc));

% reduces matrix size to minimum matrix size
x1 = x1_loc(1:min_size);
x2 = x2_loc(1:min_size);
x3 = x3_loc(1:min_size);
```

```
y1 = y1_loc(1:min_size);
y2 = y2_loc(1:min_size);
y3 = y3_loc(1:min_size);

% subtract each row element by its average
for n = 1:length(x1)
    x1(n,1) = x1(n,1) - x1_ave;
    x2(n,1) = x2(n,1) - x2_ave;
    x3(n,1) = x3(n,1) - x3_ave;

    y1(n,1) = y1(n,1) - y1_ave;
    y2(n,1) = y2(n,1) - y2_ave;
    y3(n,1) = y3(n,1) - y3_ave;
end

% Contruct coordinates into a single matrix
X_svd = [x1';y1';x2';y2';x3';y3'];

% Take the SVD of the single matrix using "economy size"
% decomposition - low-dimensional approximations
[U,S,V] = svd(X_svd,'econ');

% Find the energy of the singular values by dividing each singular
% value by the average of all of the singular values
S_col = diag(S);

for j = 1:length(S_col)
    S_col(j,1) = S_col(j,1)*S_col(j,1);
end

S_sum = sum(S_col);
for j = 1:length(S_col)
    S_col(j,1) = S_col(j,1)/S_sum;
end
S_diag = diag(S_col);

% Singular value energy plot
figure(16)
subplot(2,2,[2, 4])
plot(1:length(S_col), S_col, 'o', 'LineWidth', 2)
set(gca,'Fontsize',10)
xlabel('Singular Values');
ylabel('Relative Energy');
title('Relative Energy of Singular Values - Test 3');

% plot of all camera angles showing major variation direction (z)
subplot(2,2,1)
plot(1:length(y1),y1,1:length(y1),y2,1:length(y1),x3,'LineWidth',1)
set(gca,'Xlim',[0,length(y1)],'Ylim',[-100,100],'Fontsize',10)
legend("Cam 1","Cam 2","Cam 3")
xlabel('Time (frames)');
ylabel('Displacement (px)');
title('Bucket Displacement in Time - Test 3');

% plot of data projections onto pricipal components
subplot(2,2,3)
X_proj = U'*X_svd;
plot(1:length(X_proj(1,:)), X_proj(1,:),'LineWidth',1)
hold on
plot(1:length(X_proj(2,:)), X_proj(2,:),'LineWidth',1)
plot(1:length(X_proj(3,:)), X_proj(3,:),'LineWidth',1)
plot(1:length(X_proj(4,:)), X_proj(4,:),'LineWidth',1)
legend("PC 1","PC 2","PC 3","PC 4")
set(gca,'Xlim',[0,length(X_proj(1,:))],'Fontsize',10)
xlabel('Time (frames)');
ylabel('Displacement (px)');
title('Data Projections onto Principal Components - Test 3');


%% Test 4 - Video Processing
```

```
%% Cam1_4
clear all; close all; clc
load('cam1_4.mat')
% implay(vidFrames1_4)

[y_res, x_res, rgb, numFrames] = size(vidFrames1_4);
position = [];
for j = 1:numFrames
X = vidFrames1_4(:,:,:,j); % read each frame
gray_X = rgb2gray(X); % change to gray scale

% crop gray scale image to extract only the bucket (horizontal)
[y_len,x_len,~] = size(gray_X);
crop_h = zeros(y_len,x_len);
ch_start = 216;
ch_end = 420;
ch_width = (ch_end - ch_start) + 1;
crop_h(ch_start:ch_end, 1:x_len) = ones(ch_width, x_len);
cgh_X = gray_X(1:y_len, 1:x_len).*uint8(crop_h);

% crop gray scale image to extract only the bucket (vertical)
crop = zeros(y_len,x_len);
c_start = 301;
c_end = 469;
c_width = (c_end - c_start) + 1;
crop(1:y_len, c_start:c_end) = ones(y_len, c_width);
cg_X = cgh_X.*uint8(crop);

% extract only the bucket/light based on
BW = imbinarize(cg_X, 0.95); % binarize grayscale image by thresholding
BW_filt = medfilt2(BW,'symmetric'); % 2D median filtering
se = strel('rectangle',[70;55]); % create morphological structuring element
BW_fd = imdilate(BW_filt, se); % performs dilation using the structuring element

% find connected components in binary image and position
CC = bwconncomp(BW_fd);
S = regionprops(CC, 'Centroid');
position = [position; S];

% imshow(BW_fd);
% drawnow
end

% convert structure array to cell array
pos_cell = struct2cell(position);
x1_loc = [];
y1_loc = y_res.*ones(length(pos_cell),1);
for i = 1:length(pos_cell)
    x1_loc = [x1_loc; pos_cell{1,i}(1,1)];
    y1_loc(i, 1) = y1_loc(i, 1) - pos_cell{1,i}(1,2);
end

figure(21)
subplot(1,3,1)
plot(x1_loc, y1_loc,'.','LineWidth',9)
set(gca,'Xlim',[0,x_res],'Ylim',[0,y_res],'Fontsize',10)
xticks(0:50:x_res)
xlabel('x (px)');
ylabel('y (px)');
title('Bucket Positions - Test 4 Camera 1');

% Cam2_4

load('cam2_4.mat')
% implay(vidFrames2_4)

[y_res, x_res, rgb, numFrames] = size(vidFrames2_4);
position = [];
for j = 1:numFrames
X = vidFrames2_4(:,:,:,j); % read each frame
gray_X = rgb2gray(X); % change to gray scale
```

```
% crop gray scale image to extract only the bucket (horizontal)
[y_len,x_len,~] = size(gray_X);
crop_h = zeros(y_len,x_len);
ch_start = 83;
ch_end = 365;
ch_width = (ch_end - ch_start) + 1;
crop_h(ch_start:ch_end, 1:x_len) = ones(ch_width, x_len);
cgh_X = gray_X(1:y_len, 1:x_len).*uint8(crop_h);

% crop gray scale image to extract only the bucket (vertical)
crop = zeros(y_len,x_len);
c_start = 210;
c_end = 434;
c_width = (c_end - c_start) + 1;
crop(1:y_len, c_start:c_end) = ones(y_len, c_width);
cg_X = cgh_X.*uint8(crop);

% extract only the bucket/light based on
BW = imbinarize(cg_X, 0.97); % binarize grayscale image by thresholding
BW_filt = medfilt2(BW,'symmetric'); % 2D median filtering
se = strel('rectangle',[80;50]); % create morphological structuring element
BW_fd = imdilate(BW_filt, se); % performs dilation using the structuring element

% find connected components in binary image and position
CC = bwconncomp(BW_fd);
S = regionprops(CC, 'Centroid');
position = [position; S];

% imshow(BW_fd);
% drawnow
end

% convert structure array to cell array
pos_cell = struct2cell(position);
x2_loc = [];
y2_loc = y_res.*ones(length(pos_cell),1);
for i = 1:length(pos_cell)
    x2_loc = [x2_loc; pos_cell{1,i}(1,1)];
    y2_loc(i, 1) = y2_loc(i, 1) - pos_cell{1,i}(1,2);
end

subplot(1,3,2)
plot(x2_loc, y2_loc,'.','LineWidth',9)
set(gca,'Xlim',[0,x_res],'Ylim',[0,y_res],'Fontsize',10)
xticks(0:50:x_res)
xlabel('x (px)');
ylabel('y (px)');
title('Bucket Positions - Test 4 Camera 2');

% Cam3_4
load('cam3_4.mat')
% implay(vidFrames3_4)

[y_res, x_res, rgb, numFrames] = size(vidFrames3_4);
position = [];
for j = 1:numFrames
X = vidFrames3_4(:,:,:,j); % read each frame
gray_X = rgb2gray(X); % change to gray scale

% crop gray scale image to extract only the bucket (horizontal)
[y_len,x_len,~] = size(gray_X);
crop_h = zeros(y_len,x_len);
ch_start = 130;
ch_end = 295;
ch_width = (ch_end - ch_start) + 1;
crop_h(ch_start:ch_end, 1:x_len) = ones(ch_width, x_len);
cgh_X = gray_X(1:y_len, 1:x_len).*uint8(crop_h);

% crop gray scale image to extract only the bucket (vertical)
crop = zeros(y_len,x_len);
```

```
c_start = 280;
c_end = 522;
c_width = (c_end - c_start) + 1;
crop(1:y_len, c_start:c_end) = ones(y_len, c_width);
cg_X = cgh_X.*uint8(crop);

% extract only the bucket/light based on
BW = imbinarize(cg_X, 0.905); % binarize grayscale image by thresholding
BW_filt = medfilt2(BW,'symmetric'); % 2D median filtering
se = strel('rectangle',[50;60]); % create morphological structuring element
BW_fd = imdilate(BW_filt, se); % performs dilation using the structuring element

% find connected components in binary image and position
CC = bwconncomp(BW_fd);
S = regionprops(CC, 'Centroid');
position = [position; S];

% imshow(BW_fd);
% drawnow
end

% convert structure array to cell array
pos_cell = struct2cell(position);
x3_loc = [];
y3_loc = y_res.*ones(length(pos_cell),1);
for i = 1:length(pos_cell)
    x3_loc = [x3_loc; pos_cell{1,i}(1,1)];
    y3_loc(i, 1) = y3_loc(i, 1) - pos_cell{1,i}(1,2);
end

subplot(1,3,3)
plot(x3_loc, y3_loc,'.','LineWidth',9)
set(gca,'Xlim',[0,x_res],'Ylim',[0,y_res],'Fontsize',10)
xticks(0:50:x_res)
xlabel('x (px)');
ylabel('y (px)');
title('Bucket Positions - Test 4 Camera 3');

%% Test 4 - SVD
% make all of the camera videos in sync
[~,y1_min] = min(y1_loc(1:50));
[~,y2_min] = min(y2_loc(1:50));
[~,x3_max] = max(x3_loc(1:50));

x1_loc = x1_loc(y1_min:end);
y1_loc = y1_loc(y1_min:end);

x2_loc = x2_loc(y2_min:end);
y2_loc = y2_loc(y2_min:end);

x3_loc = x3_loc(x3_max:end);
y3_loc = y3_loc(x3_max:end);

% find an averaged x and y coordinate system for each camera
% to reduce the effect of the svd giving more weight on just
% absolute location values

% average row values
x1_ave = mean(x1_loc);
x2_ave = mean(x2_loc);
x3_ave = mean(x3_loc);

y1_ave = mean(y1_loc);
y2_ave = mean(y2_loc);
y3_ave = mean(y3_loc);

% find the minimum matrix size
min_size = min(length(x1_loc), length(x2_loc));
min_size = min(min_size, length(x3_loc));

% reduces matrix size to minimum matrix size
```

```
x1 = x1_loc(1:min_size);
x2 = x2_loc(1:min_size);
x3 = x3_loc(1:min_size);

y1 = y1_loc(1:min_size);
y2 = y2_loc(1:min_size);
y3 = y3_loc(1:min_size);

% subtract each row element by its average
for n = 1:length(x1)
    x1(n,1) = x1(n,1) - x1_ave;
    x2(n,1) = x2(n,1) - x2_ave;
    x3(n,1) = x3(n,1) - x3_ave;

    y1(n,1) = y1(n,1) - y1_ave;
    y2(n,1) = y2(n,1) - y2_ave;
    y3(n,1) = y3(n,1) - y3_ave;
end

% Contruct coordinates into a single matrix
X_svd = [x1';y1';x2';y2';x3';y3'];

% Take the SVD of the single matrix using "economy size"
% decomposition - low-dimensional approximations
[U,S,V] = svd(X_svd,'econ');

% Find the energy of the singular values by dividing each singular
% value by the average of all of the singular values
S_col = diag(S);

for j = 1:length(S_col)
    S_col(j,1) = S_col(j,1)*S_col(j,1);
end

S_sum = sum(S_col);
for j = 1:length(S_col)
    S_col(j,1) = S_col(j,1)/S_sum;
end
S_diag = diag(S_col);

% Singular value energy plot
figure(16)
subplot(2,2,[2, 4])
plot(1:length(S_col), S_col, 'o', 'LineWidth', 2)
set(gca,'Fontsize',10)
xlabel('Singular Values');
ylabel('Relative Energy');
title('Relative Energy of Singular Values - Test 4');

% plot of all camera angles showing major variation direction (z)
subplot(2,2,1)
plot(1:length(y1),y1,1:length(y1),y2,1:length(y1),x3,'LineWidth',1)
set(gca,'Xlim',[0,length(y1)],'Ylim',[-100,100],'Fontsize',10)
legend("Cam 1","Cam 2","Cam 3")
xlabel('Time (frames)');
ylabel('Displacement (px)');
title('Bucket Displacement in Time - Test 4');

% plot of data projections onto pricipal components
subplot(2,2,3)
X_proj = U'*X_svd;
plot(1:length(X_proj(1,:)), X_proj(1,:),'LineWidth',1)
hold on
plot(1:length(X_proj(2,:)), X_proj(2,:),'LineWidth',1)
plot(1:length(X_proj(3,:)), X_proj(3,:),'LineWidth',1)
legend("PC 1","PC 2","PC 3")
set(gca,'Xlim',[0,length(X_proj(1,:))],'Fontsize',10)
xlabel('Time (frames)');
ylabel('Displacement (px)');
title('Data Projections onto Principal Components - Test 4');
```

References

1. Kutz, Jose Nathan. *Data-Driven Modeling & Scientific Computation: Methods for Complex Systems & Big Data*. Oxford University Press, 2013.