# AMATH 582 Homework 3

Leon Yan

February 21, 2020

**Abstract**

This report serves to illustrate how we use the Singular Value Decomposition(SVD) techniques to remove the noise and the redundancy in our position data of an oscillating paint can. For the full materials including code and homework description, please go to url: `https://github.com/unitraveleryy/Amath582HW`

## 1 Introduction and Overview

In this homework, We are given the video data of an oscillating paint can from different perspectives. The ultimate goal is to extract the paint can's trajectory data in these noisy, redundant data. In different set of video data, the paint can has different oscillating characteristics and different noisy level.

In task I, Ideal case: Consider a small displacement of the mass in the z direction and the ensuing oscillations. In this case, the entire motion is in the z directions with simple harmonic motion being observed. In task II, the same motion in task I, but we deliberately introduce some camera shaking to serve as the noise to corrupt the data.

In task III, horizontal displacement: In this case, the mass is released off the center so as to produce motion in the $x, y$ plane as well as the z direction. Thus there is both a pendulum motion and a simple harmonic oscillations. In task IV, the same motion in task III, we added extra rotation momentum to the paint can regarding z direction.

### 1.1 Data Description

Each RGB video stream is stored in the 4-D .mat file. The name "cam x_y" is interpreted as the camera x's data in y-th task. The first two dimensions correspond to the pixel coordinates. The third one is the different RGB channel. The last one represents the time stamp.

## 2 Theoretical Background

### 2.1 Singular Value Decomposition

As we learned from our textbook [1], A singular value decomposition (SVD) is a factorization of a matrix into a number of constitutive components all of which have a specific meaning in applications. From a pure linear algebra point of view, the SVD is essentially a transformation that stretches/compresses and rotates a given set of vectors. The full SVD of an arbitrary matrix $A \in \mathbb{C}^{m\times n}$ takes the form:

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

where $\mathbf{U} \in \mathbb{C}^{m\times m}$ and $\mathbf{V} \in \mathbb{C}^{n\times n}$ are unitary matrices; $\mathbf{\Sigma} \in \mathbb{R}^{m\times n}$ is diagonal kind, the diagonal entries of which are non-negative and ordered from the largest to smallest arranged as $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_p \geq 0$ where $p = \min(m, n)$. Further, we define the total energy from mode 1 to mode $i$ is defined by the fraction below:

$$E_{1\sim i} = \frac{\sum_{n=1}^{i} \sigma_n}{\sum_{n=1}^{p} \sigma_n}, \tag{2}$$

which can measure how important the modes are.

### 2.2 Principle Component Analysis via SVD

Given the data matrix $X \in \mathbb{R}^{m \times n}$, each row is a time-series measurement specified by $n$ time stamps. By doing the SVD of the data matrix $X$, we can project the data into $Y$ in terms of the new orthonormal basis $U$ such that the measurements in different measurement have zero covariance and picking out the principle components determined by the its variance to do the analysis. By picking out the principle components, we are exactly extracting the principle information described in a concise coordinates and removing the noise information at the same time. Below are the proof to explain why we can get the covariance of the new data matrix $Y$ to be zero when we express $X$ in terms of $U$ basis. When we are projecting data matrix $X$ in terms of basis $U$:

$$Y = U^{-1}X = U^*X, \tag{3}$$

since the $U$ is the unitary matrix, i.e., $U^{-1} = U^*$.

Then $C_Y$, the covariance of $Y$ can be computed and simplified as below by doing the SVD of $X = U\Sigma V^*$:

$$\begin{aligned} C_Y &= \frac{1}{n-1}YY^T \\ &= \frac{1}{n-1}(U^*X)(U^*X)^T \\ &= \frac{1}{n-1}U^*(XX^T)U \\ &= \frac{1}{n-1}U^*U\Sigma^2UU^* \end{aligned}$$

Then we have:

$$C_Y = \frac{1}{n-1}\Sigma^2. \tag{4}$$

# 3 Algorithm Implementation and Development

Below is my algorithm implementation and development of doing the PCA for the paint can's trajectory:

1. frame by frame, find and use the coordinates of the brightest pixel to represent the paint can's position in that frame
2. Collecting all the three camera's video stream
3. Crop each camera's data properly; align the starting and ending
4. Stack the data vertically so as to form a 6-by-n data matrix $X$
5. Doing the SVD of X, get the unitary matrices $U$ and $V$
6. Find the principle components (dominant singular values) and corresponding basis direction
7. Projecting the data matrix $X$ in the dominant direction to see how it changes versus time.

The relevant detailed implementation can be seen in Algorithm 1.

# 4 Computational Results

For Task I, we can see the PCA in Fig. 1. It's clear from the analysis, there is just 1 principle harmonic oscillating movement, the energy of which is 71.2%, based on the formula mentioned in Section. 2. Further, by projecting the data matrix X into the principle direction analyzed by PCA, by plotting the change of coordinates in terms of the principle modes, we can see the real pixel translation of the paint can, as demonstrated in Fig. 2. By splitting the first mode (the first 6-by-1 column vector in matrix U generated by SVD) into 3 different 2-by-1 vector, we can draw the them separately in three different camera. These

**Algorithm 1:** PCA to extract the paint can's trajectory

**Input:** frame data from `cam x_y.mat`

**Output:** simplified paint can's trajectory

*Initialisation, data cleaning* :

Locate the paint can's position in each frame by using the brightest pixel's coordinates, extracted as 2-by-n matrix

Visually compare the different camera's data, align them in time domain

Stack all them up vertically as data matrix X

*PCA via SVD*

Doing the SVD of $X = U\Sigma V^*$

Select significant i-th singular value and the corresponding columns in $U$ as the mode direction and in $V$ as the time evolution of that mode

Project the data $X$ in terms of the basis selected and plot the corresponding $Y$ in each frame

vectors point out the major movement, i.e., the direction of the oscillation. You can see the illustrations in Fig. 3. However, the similar process can not be done in Task III and Task IV, because the number of modes exceeds the camera's mode capacity 2. (The video frame is just 2-d, which can only capture at most two modes.) It's hard for me to interpret the physical meaning of the modes in those situations. :( Could you plz offer some insights?

For Task II, the PCA results are shown in Fig. 4. Compared with the distribution of the singular values in Task I, it's apparent that the following singular values' magnitude are larger due to the deliberately shaking camera, which is mentioned in the homework descriptions for different tasks. The real oscillating component (largest singular value) only constitutes 40.42% in total energy. The real pixel translation of the paint can is demonstrated in Fig. 5.

For Task III, the PCA result can be seen in Fig. 6. There are two principle modes, which constitute the 79.4% of the total energy. Both acting in the sinusoidal pattern, as seen in Fig. 7 and Fig. 8. The corresponding directions are plotted in camera 1's frame as an example, illustrated by Fig. 9. That is, grabbing the first 2-by-1 vector in the first two columns of U generated by SVD.

For Task IV, we plotted the PCA in Fig. 10. There are two principle modes, which constitute the 71.84% of the total energy. Compared with the PCA in Task III, we can see that the purposely camera shaking movements do really affect the energy distribution, which makes the principle modes less significant. And the second harmonic oscillating is quite deformed in noisy task IV case, which in task III PCA, we can see clear harmonic behavior for two principle modes. And the real pixel translation plots for these two principle direction can be seen in Fig. 11 and Fig. 11.

# 5 Summary and Conclusions

To conclude, in this report, we successfully use the SVD to do the PCA for paint can's complex noisy movements. In SVD, $X = U\Sigma V^*$, we can obtain principle direction in matrix $U$ and corresponding time behavior in matrix $V$. The $\Sigma$ tells us how many principle modes exist in the data matrix, and we can decrease the data matrix's rank by eliminating un-important singular values.

# References

[1] Jose Nathan Kutz. *Data-driven modeling & scientific computation: methods for complex systems & big data*. Oxford University Press, 2013.

# Appendix A MATLAB Functions

Below are the descriptions of the MATLAB function that I used in this report:

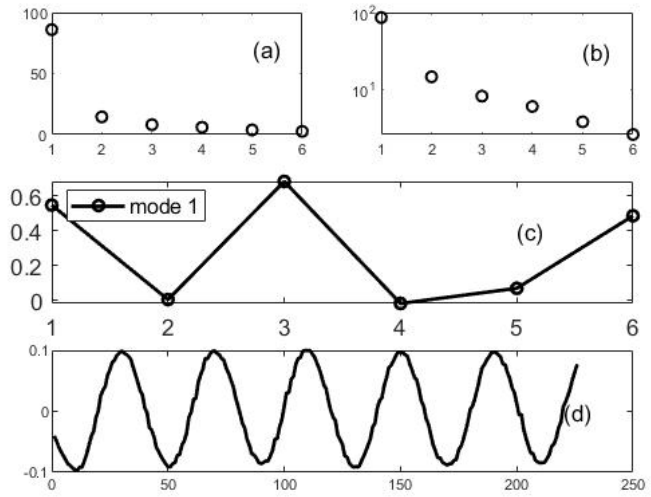


Figure 1: The PCA for Task I, (a) is the plot of the singular values; (b) is the same but in the log scale; (c) is the plot of the principle mode; (d) is the corresponding time behavior.

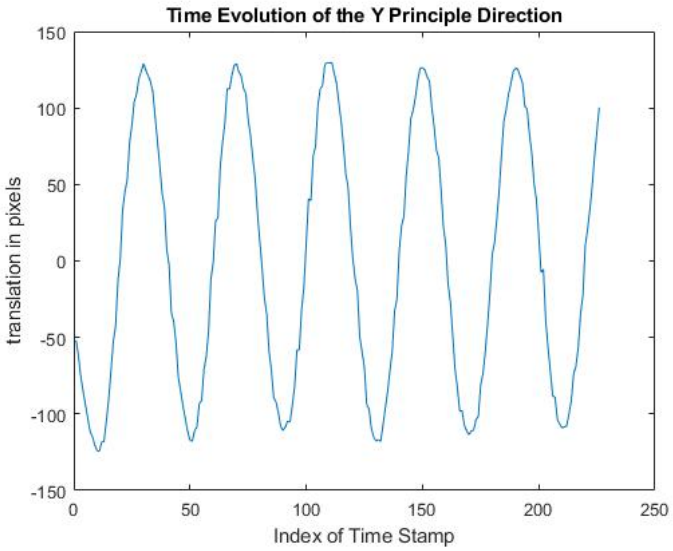


Figure 2: The principle translation of the paint can's movement in pixel values.

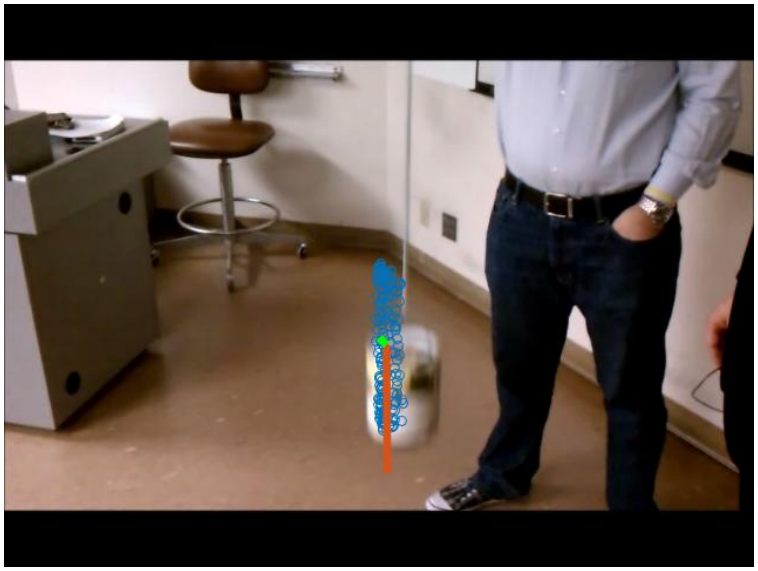

Figure 3: First two Mode directions in camera 1 perspective. The red and green direction correspond to the biggest and second biggest singular value respectively. The length of the direction is scaled to the same extent. The blue circles are the overall paint can's positions.

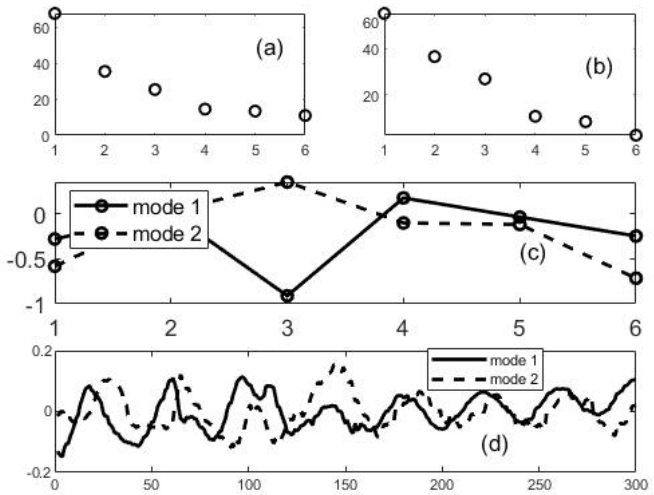


Figure 4: The PCA for Task I, (a) is the plot of the singular values; (b) is the same but in the log scale; (c) is the plot of the principle mode; (d) is the corresponding time behavior.

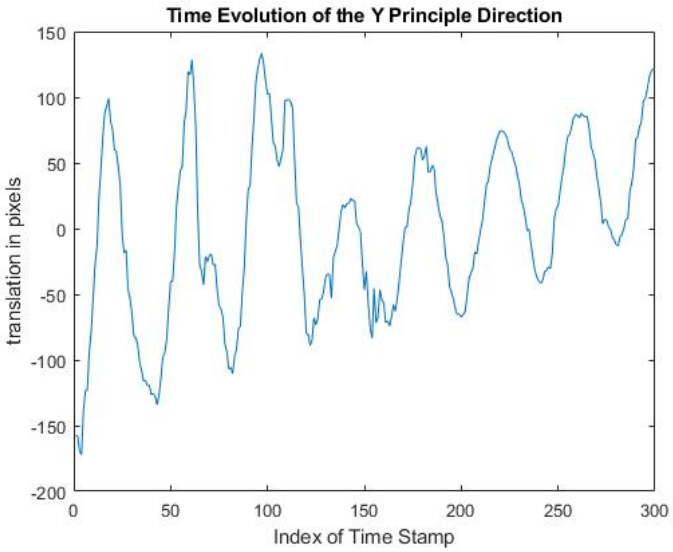


Figure 5: The principle translation of the paint can's movement in pixel values.

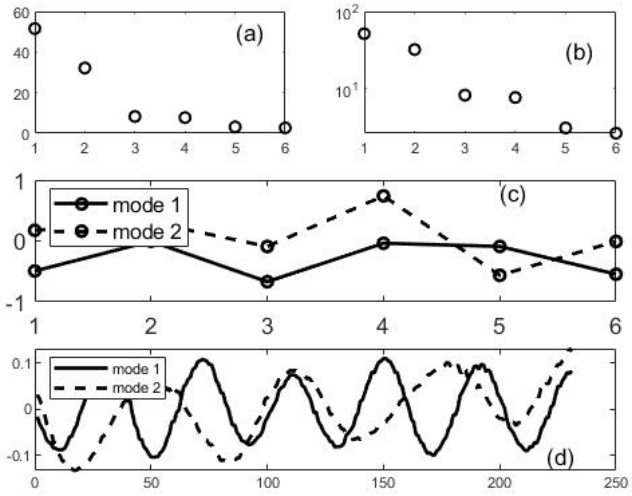


Figure 6: The PCA for Task I, (a) is the plot of the singular values; (b) is the same but in the log scale; (c) is the plot of the principle mode; (d) is the corresponding time behavior.

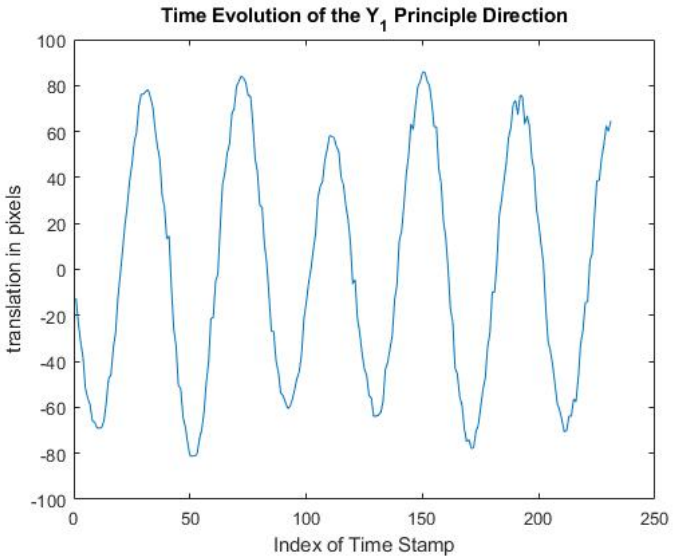


Figure 7: The first principle translation of the paint can's movement in pixel values.

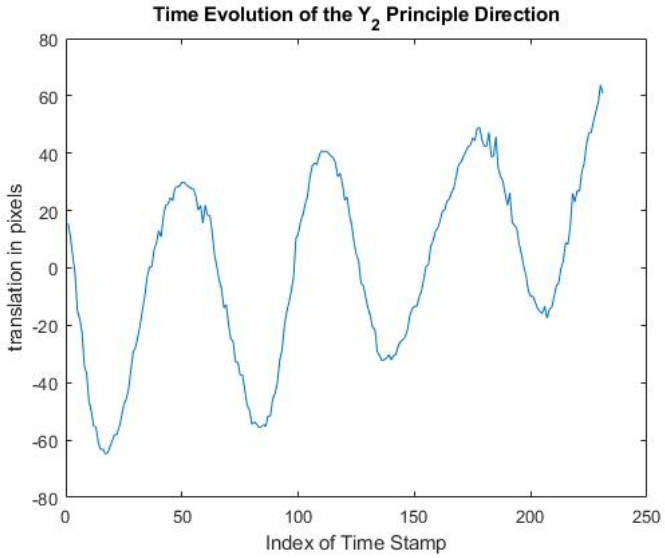


Figure 8: The second principle translation of the paint can's movement in pixel values.

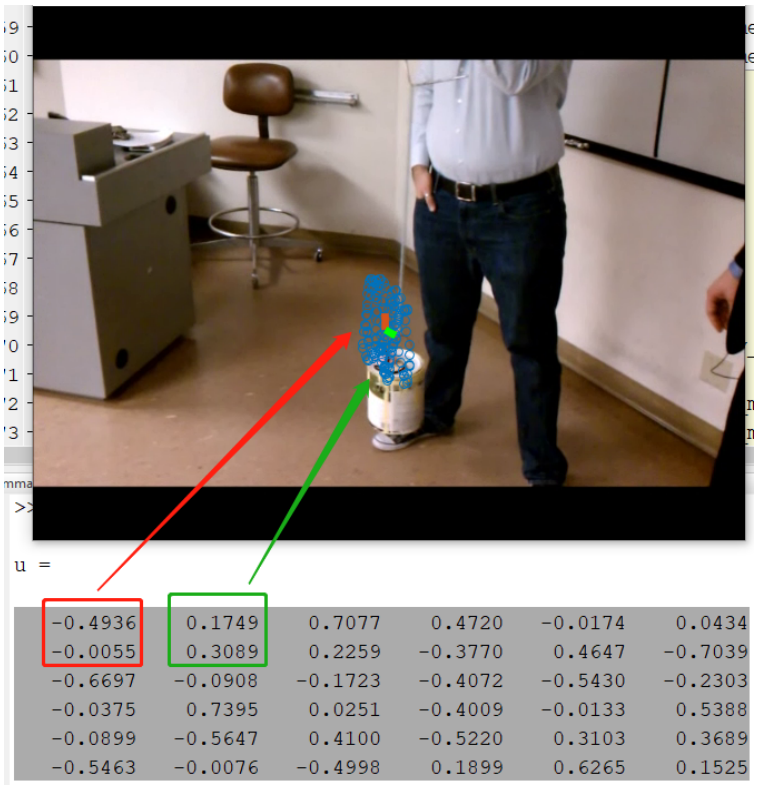


Figure 9: First two Mode directions in camera 1 perspective. The red and green direction correspond to the biggest and second biggest singular value respectively. The length of the direction is scaled to the same extent. The blue circles are the overall paint can's positions.

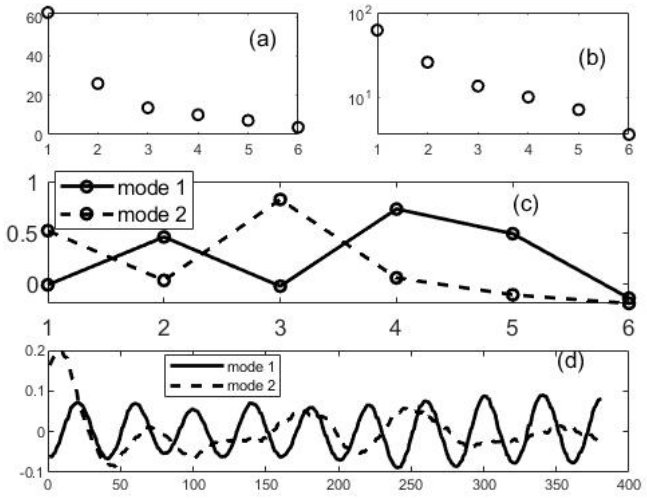


Figure 10: The PCA for Task I, (a) is the plot of the singular values; (b) is the same but in the log scale; (c) is the plot of the principle mode; (d) is the corresponding time behavior.

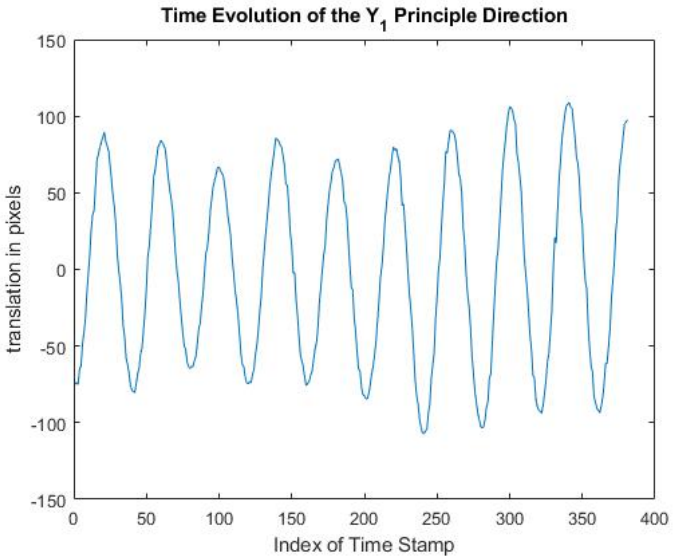


Figure 11: The first principle translation of the paint can's movement in pixel values.

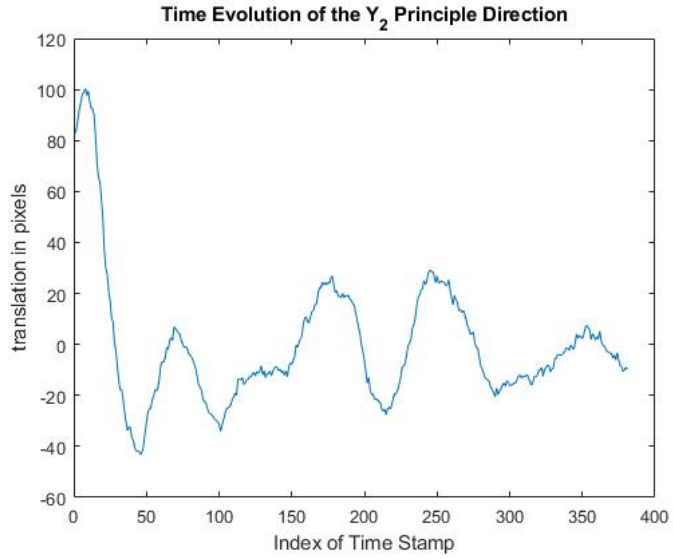


Figure 12: The second principle translation of the paint can's movement in pixel values.

- `B = repmat(A,n)` returns an array containing n copies of A in the row and column dimensions. The size of B is size(A)*n when A is a matrix.
- `x = diag(A)` returns a column vector of the main diagonal elements of A.
- `[U,S,V] = svd(A)` performs a singular value decomposition of matrix A, such that A = U*S*V'.
- `semilogy(Y)` creates a plot using a base 10 logarithmic scale for the y-axis and a linear scale for the x-axis. It plots the columns of Y versus their index.
- `h = imrect` begins interactive placement of a rectangle on the current axes, and returns an imrect object.
- `Num = getPosition(Group)` returns the position of an slmetric.dashboard.Group object within an array that holds Metrics Dashboard objects.
- `RGB = insertShape(I,shape,position)` returns a true color image with shape inserted. The input image, I, can be either a true color or grayscale image.

# Appendix B MATLAB Code

```
cam1 = importdata('cam1_1.mat');
cam2 = importdata('cam2_1.mat');
cam3 = importdata('cam3_1.mat');
figure();
% demo code for extracting the paint can position in cam 1 data
xy_cam1 = zeros(2,size(cam1,4));%
for ii = 1:size(cam1,4)%
    frame = cam1(:,:,:,ii);
    frame = rgb2gray(frame);
    imshow(frame);
    objectRegion = round(getPosition(imrect));
    objectImage = insertShape(frame,'Rectangle',objectRegion,'Color','red');
    imshow(objectImage);
    pause(0.5)
    dataAnaly = frame(objectRegion(2):(objectRegion(2)+objectRegion(4)),...
                      objectRegion(1):(objectRegion(1)+objectRegion(3)));
    [~,I] = max(dataAnaly(:));
    [r,c] = ind2sub(size(dataAnaly),I)
    r = r + objectRegion(2)
    c = c + objectRegion(1)

    xy_cam1(:,ii) = [r;c];%

    pointImage = insertMarker(objectImage,[c r],'+','Color','green');
    imshow(pointImage);

    pause(0.5);
end
```

Listing 1: MATLAB code for paint can's position extraction

```
%% demo of task I's PCA
xy_cam1 = importdata('pos_rec/xy_cam1_1.mat');
xy_cam2 = importdata('pos_rec/xy_cam2_1.mat');
xy_cam3 = importdata('pos_rec/xy_cam3_1.mat');
% visually align the video data
xy_cam2 = xy_cam2(:,10:226+9); xy_cam3 = xy_cam3(:,1:226);

X = [xy_cam1;xy_cam2;xy_cam3];
[m,n] = size(X);
mn = mean(X,2);
X = X - repmat(mn,1,n);
[u,s,v]=svd(X/sqrt(n-1));
Y = u' * X;
sig = diag(s);
sig(1)/sum(sig) % compute the energy

figure()
sig=diag(s);
subplot(3,2,1), plot(1:6,sig,'ko','Linewidth',[1.5])
text(5,70,'(a)','Fontsize',[13])
subplot(3,2,2), semilogy(1:6,sig,'ko','Linewidth',[1.5])
text(5,10^1.5,'(b)','Fontsize',[13])

subplot(3,1,2) % spatial modes
plot(1:6,u(:,1),'ko-','Linewidth',[2])
set(gca,'Fontsize',[13])
legend('mode 1','Location','NorthWest')
text(5,0.4,'(c)','Fontsize',[13])
subplot(3,1,3) % time behavior
plot(1:length(xy_cam1),v(:,1),'k','Linewidth',[2])
text(220,0,'(d)','Fontsize',[13])

figure();plot(Y(1,:)); title("Time Evolution of the Y Principle Direction");
xlabel('Index of Time Stamp');ylabel('translation in pixels');
```

Listing 2: MATLAB code for doing PCA

```
%% demo task I
cam1 = importdata('cam1_1.mat');
cam2 = importdata('cam2_1.mat');
cam3 = importdata('cam3_1.mat');
xy_cam1 = importdata('pos_rec/xy_cam1_1.mat');
xy_cam2 = importdata('pos_rec/xy_cam2_1.mat');
xy_cam3 = importdata('pos_rec/xy_cam3_1.mat');

frame = cam1(:,:,:,100);
% plot all paint can's trajectory
figure();imshow(frame);hold on;scatter(xy_cam1(2,:),xy_cam1(1,:));
xy_mean = mean(xy_cam1,2);
% plot two principle directions
plot([xy_mean(2) xy_mean(2)+0.038*20],[xy_mean(1) xy_mean(1)+5.452*20],'LineWidth',5);
plot([xy_mean(2) xy_mean(2)-0.306*20],[xy_mean(1) xy_mean(1)-0.3158*20],'g','LineWidth',5);
```

Listing 3: MATLAB code for visualizing movement points