AMATH 582 Homework 3

Kyle Schultz*

February 21, 2020

Abstract

In this project, we use principal component analysis (PCA) to analyze the motion of an object captured with three different cameras. We achieve this by tracking object in each camera feed and then performing SVD on trajectories from all cameras. This allows find the dominant modes of movement from all the cameras.

1 Introduction and Overview

We are given videos of a paint can connected to a spring taken from 3 different angles. The paint can therefore undergoes simple harmonic motion. The paint can is subjected to additional movements, and the cases are described below:

1. Ideal case: movement only in z-direction
2. Noisy case: movement only in z-direction, but with significant camera shake
3. Horizontal displacement: the mass is released of center and moves in x-y plane as well as z-direction
4. Horizontal displacement and: movement in x-y plane, z-direction and rotation about z-axis

To analyze the dominant modes in which the paint can is moving, we first must track the paint can frame by frame. This is achieved by using the built in MatLab computer vision toolbox for feature tracking. This is repeated for each camera angle, giving us 6 coordinates in which the paint can moves (2 coordinates per camera). We then perform SVD on this coordinate matrix to find the principal components, which represent the dominant directions of motion.

A significant part of this project was tracking the paint can. The video feeds were low resolution and the only marker was a lamp added to the top of the can. However, we found using MatLab's feature detection and tracking to work well for tracking the centroid of the paint can. In case 4 where the can is rotating, this method did not work since now a single feature needed to be tracked to capture the rotation. For this, we wrote an algorithm that classified points when they were the same color as a feature on the can. We used the pink part of the light, and tracked pixels that were sufficiently similar in color.

2 Theoretical Background

2.1 Singular Value Decomposition

Singular value decomposition (SVD) is an algorithm that expresses any matrix in terms of three matrices: U, S, V (equation 1).

$$A = USV^* \tag{1}$$

In equation 1, matrices U and V are unitary and S has real, positive values on its diagonal. In our case, we have an A matrix with dimension $6x216$, where we have 6 rows reprenting the (x, y) location of the paint

*Code available on Github: https://github.com/kyleschultz0/

can for 3 different cameras. Our U, S, and V are therefore of dimension $6x6$, $6x216$, and $216x216$ respectively. The columns of U can be interpreted as the dominant modes of the data matrix A (i.e. directions the paint can displaces along). V can be interpreted as how the data projects onto the modes in U, and S are the singular values. The singular values give us a weighting of how important the dominant modes in U are.

3 Algorithm Implementation and Development

3.1 Tracking the Paint Can

The first part of this algorithm is tracking the paint can. Although the purpose of this project is to explore PCA, tracking the paint can and determining its position in each of the camera feeds was a significant amount of the work. Algorithm 1 was used for cases 1-3, and algorithm 2 was used in case 4. Algorithm 1 tracked the centroid of the paint can, which would not capture rotation and hence algorithm 2 was developed. The general outline of the program is:

1. Load videos
2. Convert .mat videos to .avi for compatibility with CV tools
3. Get user defined rectangular region with features to track
4. Apply algorithm 1 or 2 (see below) to track points and obtain position coordinates
5. Concatenate position coordinates from 3 cameras, subtract average, and perform SVD

Algorithm 1: Tracking object centroid using corners

```
Find corner points in first frame
Initialize tracker
for i = 1 : NumberFrames do
  Load frame i
  Track points from frame i − 1 to i
  xpostion = mean(pointsx)
  ypostion = mean(pointsy)
  Store xposition and yposition i
end for
```

Corner features were detected using `detectMinEigenFeatures` and points were tracked using `vision.PointTracker`. Both of these functions are discussed in appendix A. An example of a tracked frame is shown in figure 1.

In algorithm 2, a user defined rectangle convering the pink part of the lamp is selected, and the pixel color is used to track points on the can. Specifically, the standard deviation and mean of the pixel RGB values is used to create an interval which will discriminate if pixels in a given frame belong to the feature.

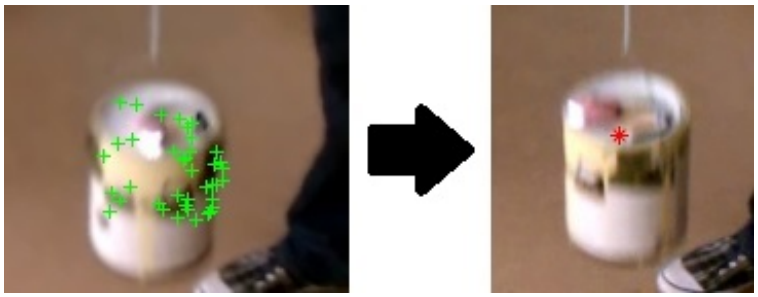

Figure 1: Corner tracking used in algorithm 1 (cases 1-3)

Algorithm 2: Creating frequency spectrum from song samples

```
Compute mean and standard deviation of RGB values in region
Create interval of accepted RGB values
for i = 1 : NumberFrames do
  num = 0
  for j = 1 : FrameHeight do
    for k = 1 : FrameWidth do
      if frame i pixel (j, k) is in accepted interval then
        num = num + 1
        pointx num = k
        pointy num = y
      end if
    end for
  end for
  pointx = mean of num pointx, pointy = mean of num pointy
  store pointx i, pointy i
end for
```

4 Computational Results

In the first test, we were able to get accurate tracking of the paint can. In figure 2, the singular values are plotted and we see that the the first singular value is an order of magnitude larger than the second. We further see that the displacement along mode 1 is a sinusoid. This is what we expect: since the can is moving only up and down, there should be one principal component which aligns with the z-direction. Second, simple harmonic motion is sinusoidal.

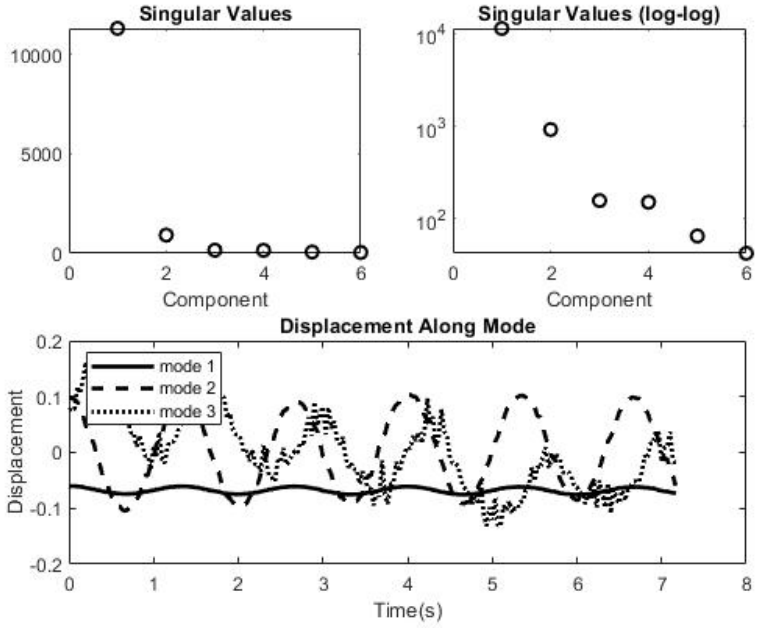

Figure 2: Ideal case, singular values and projection of displacement onto modes

We can also see some interesting results in figure 3. This image shows the past positions of the can as red points, and the blue line represents the direction of the first principal component. The SVD is giving us a direction is almost in the z-direction.

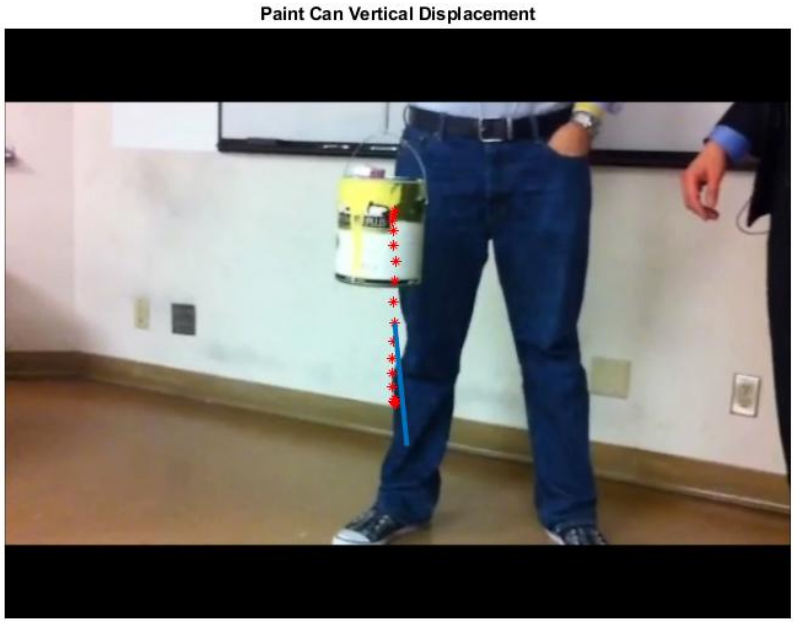

Figure 3: Image of paint can with previous positions in red. Blue line in direction of first principal component

Figure 4 shows the SVD data for the shaky case. Compared to figure 2, the 2nd-6th singular values are larger, however the first mode clearly dominates. We can also see that the projections onto the modes have become noisier, they are no longer smooth sinusoids. This makes sense since there is random noise added to the measurements. The new projections look like white noise superimposed on sines.

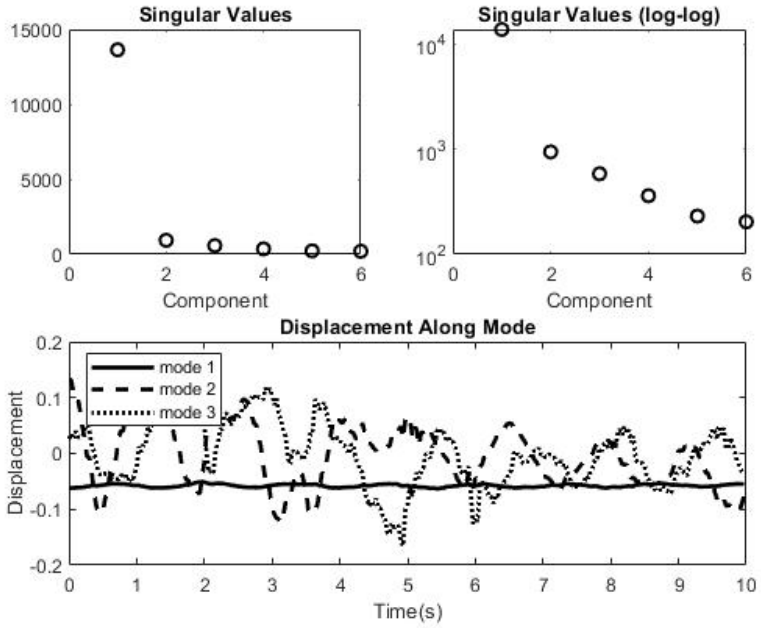

Figure 4: Shaky case, singular values and projection of displacement onto modes

Figure 5 shows the SVD data for the case with horizontal displacement. The first two modes are prominent, both being more than double the third mode. Physically, we have two dominant directions in the video feed: up and down and side to side. Thus, it makes sense we would have two dominant modes. Figure 6 confirms this, as the first principle component points in the z-direction while the second points in the x-y plane. A reason why the first two modes are not more dominant is because there is a lot of noise in this system. The tracking algorithm we wrote as well as camera shake generate considerable noise.

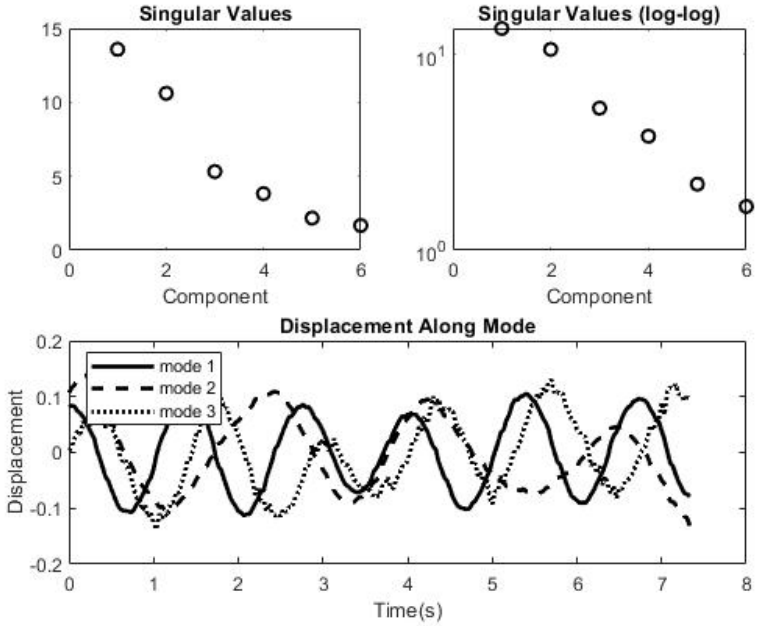

Figure 5: Horizontal motion case, singular values and projection of displacement onto modes

Figure 6: Image of paint can with previous positions in red. Blue line in direction of first principal component, orange line in direction of second

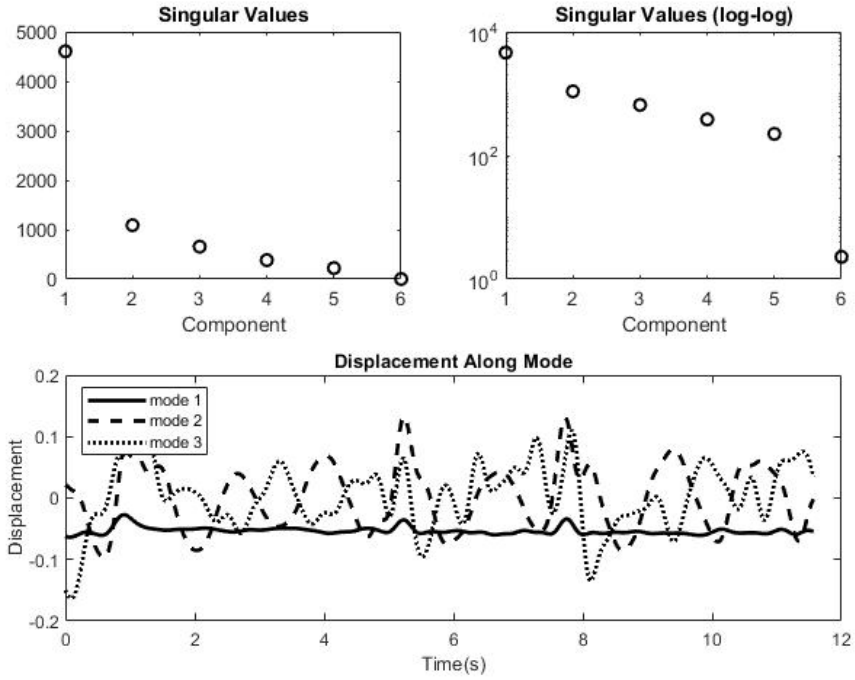

Figure 7: Horizontal and rotating case, singular values and projection of displacement onto modes

Figure 7 shows the case when the can is both rotating and moving horizontally as well as vertically. We expect there to be 3 prominent modes: one for vertical movement, one for horizontal movement, and one for rotation. In figure 7, there is only one mode that is significant compared to the others. The reason that we do not see a significant mode associated with the horizontal movement is likely because the horizontal movement is minute compared to that in test 3, and it damps out quicker. Our algorithm for tracking points also had to change since the can was rotating and we could not track the light or edges. This algorithm was worse that used in parts 1-3 and thus introduced additional noise.

5 Summary and Conclusions

In this project we successfully tracked an object in a video feed to its position as a function of time. We did this for three cameras and then performed PCA on the resulting data. This intuitively gave us one dominant principal component in the z-direction for the first two cases. In the third case, there is motion in the x-y plane as well as in the z-direction. Our PCA yielded two dominant modes as we would expect. We did not obtain such successful results for the fourth case. This is likely because there was too much noise from our tracking algorithm and thus we could not identify any modes other than the high magnitude z-axis motion. Nonetheless, these techniques proved powerful for reducing the dimension of the motion along principal components.

Appendix A MATLAB Functions

Below are some of the most critical functions in our MATLAB implementation. Included are details pertinent to how we used them.

- `points = detectMinEigenFeatures(I)` returns the corner points of feature defined by I, where I is a grayscale image. This uses the minimum eigenvalue algorithm developed by Shi and Tomasi to find feature points.

- `pointTracker = vision.PointTracker` returns a pointTracker that will track points between video frames.
- `objectRegion = getPosition(imrect)` returns the corner points of a user prescribed rectangle. This is used to specify the region for tracking.

Appendix B MATLAB Code

```
clear all; close all; clc;
load Testdata

clc; clear all; close all

%% Ideal case
%% Create .avi from given .mat file to use with CV toolbox

load("cam1_1.mat");
videoFWriter = vision.VideoFileWriter('cam1_1.avi');
size1_1 = size(vidFrames1_1);

for i=1:size1_1(4)
  videoFrame = vidFrames1_1(:, :, :, i);
  videoFWriter(videoFrame);
end
release(videoFWriter);

load("cam2_1.mat");
videoFWriter = vision.VideoFileWriter('cam2_1.avi');
size2_1 = size(vidFrames2_1);

for i=1:size2_1(4)
  videoFrame = vidFrames2_1(:, :, :, i);
  videoFWriter(videoFrame);
end
release(videoFWriter);

load("cam3_1.mat");
videoFWriter = vision.VideoFileWriter('cam3_1.avi');
size3_1 = size(vidFrames3_1);

for i=1:size3_1(4)
  videoFrame = vidFrames3_1(:, :, :, i);
  videoFWriter(videoFrame);
end
release(videoFWriter);

%%  Create object region

videoFileReader1_1 = vision.VideoFileReader('cam1_1.avi');
videoPlayer1_1 = vision.VideoPlayer('Position',[100,100,680,520]);
objectFrame1_1 = videoFileReader1_1();
figure; imshow(objectFrame1_1);
objectRegion1_1=round(getPosition(imrect));
```

```
videoFileReader2_1 = vision.VideoFileReader('cam2_1.avi');
videoPlayer2_1 = vision.VideoPlayer('Position',[100,100,680,520]);
objectFrame2_1 = videoFileReader2_1();
figure; imshow(objectFrame2_1);
objectRegion2_1=round(getPosition(imrect));

videoFileReader3_1 = vision.VideoFileReader('cam3_1.avi');
videoPlayer3_1 = vision.VideoPlayer('Position',[100,100,680,520]);
objectFrame3_1 = videoFileReader3_1();
figure; imshow(objectFrame3_1);
objectRegion3_1=round(getPosition(imrect));



%% Track Points

% Camera 1 ------------------------------------------
points = detectMinEigenFeatures(rgb2gray(objectFrame1_1),'ROI',objectRegion1_1);
tracker = vision.PointTracker('MaxBidirectionalError',1);
initialize(tracker,points.Location,objectFrame1_1);

i = 0;
while ~isDone(videoFileReader1_1)
    i = i+1;
    frame = videoFileReader1_1();
    [points,validity] = tracker(frame);
    points = points(validity, :);
    x1_1(i) = round(sum(points(:,1))/length(points(:,1)));
    y1_1(i) = round(sum(points(:,2))/length(points(:,2)));
    [points,validity] = tracker(frame);
    out = insertMarker(frame,points(validity, :),'+');
    videoPlayer1_1(out);
    pause(0.01)
end

for i = 1:size1_1(4)-1
    imshow(vidFrames1_1(:, :, :, i))
    hold on
    plot(x1_1(i), y1_1(i), 'r*')
    hold off
    pause(0.01)
end

% Camera 2 ------------------------------------------
points = detectMinEigenFeatures(rgb2gray(objectFrame2_1),'ROI',objectRegion2_1);
tracker = vision.PointTracker('MaxBidirectionalError',1);
initialize(tracker,points.Location,objectFrame2_1);

i = 0;
while ~isDone(videoFileReader2_1)
    i = i+1;
    frame = videoFileReader2_1();
    [points,validity] = tracker(frame);
```

```
    points = points(validity, :);
    x2_1(i) = round(sum(points(:,1))/length(points(:,1)));
    y2_1(i) = round(sum(points(:,2))/length(points(:,2)));
    [points,validity] = tracker(frame);
    out = insertMarker(frame,points(validity, :),'+');
    videoPlayer2_1(out);
    pause(0.01)
end

for i = 1:size2_1(4)-1
    imshow(vidFrames2_1(:, :, :, i))
    hold on
    plot(x2_1(i), y2_1(i), 'r*')
    hold off
    pause(0.01)
end

% Camera 3 -----------------------------------------
points = detectMinEigenFeatures(rgb2gray(objectFrame3_1),'ROI',objectRegion3_1);
tracker = vision.PointTracker('MaxBidirectionalError',1);
initialize(tracker,points.Location,objectFrame3_1);

i = 0;
while ~isDone(videoFileReader3_1)
    i = i+1;
    frame = videoFileReader3_1();
    [points,validity] = tracker(frame);
    points = points(validity, :);
    x3_1(i) = round(sum(points(:,1))/length(points(:,1)));
    y3_1(i) = round(sum(points(:,2))/length(points(:,2)));
    [points,validity] = tracker(frame);
    out = insertMarker(frame,points(validity, :),'+');
    videoPlayer3_1(out);
    pause(0.01)
end

for i = 1:size3_1(4)-1
    imshow(vidFrames3_1(:, :, :, i))
    hold on
    plot(x3_1(i), y3_1(i), 'r*')
    hold off
    pause(0.01)
end


%% Aligning video feeds

frame_rate = 30;

t1 = (1:1:(size1_1(4)-1))/frame_rate;
t2 = (1:1:(size2_1(4)-1))/frame_rate;
t3 = (1:1:(size3_1(4)-1))/frame_rate;

figure(1)
```

```
plot(t1, x1_1, t1, y1_1)
legend("x", "y")

figure(2)
plot(t2, x2_1, t2, y2_1)
legend("x", "y")


figure(3)
plot(t3, x3_1, t3, y3_1)
legend("x", "y")

x1_1_truncated = x1_1(10:225) - mean(x1_1(10:225));
y1_1_truncated = y1_1(10:225) - mean(y1_1(10:225));
t1_truncated = t1(10:225) - t1(10);

x2_1_truncated = x2_1(18:233) - mean(x2_1(18:233));
y2_1_truncated = y2_1(18:233) - mean(y2_1(18:233));
t2_truncated = t2(18:233) - t2(18);

x3_1_truncated = x3_1(8:223) - mean(x3_1(8:223));
y3_1_truncated = y3_1(8:223) - mean(y3_1(8:223));
t3_truncated = t3(8:223) - t3(8);

figure(4)
plot(t1_truncated, x1_1_truncated, t1_truncated, y1_1_truncated)
legend("x", "y")

figure(5)
plot(t2_truncated, x2_1_truncated, t2_truncated, y2_1_truncated)
legend("x", "y")

figure(6)
plot(t3_truncated, x3_1_truncated, t3_truncated, y3_1_truncated)
legend("x", "y")

%% Computing SVD

X = [x1_1_truncated; y1_1_truncated; x2_1_truncated;...
    y2_1_truncated; x3_1_truncated; y3_1_truncated];

[U,S,V] = svd(X);

figure(7)
subplot(2,2,1)
plot(t1_truncated, x1_1_truncated, t1_truncated, y1_1_truncated)
ylim([240 400])
for j=1:3
  ff=U(:,1:j)*S(1:j,1:j)*V(:,1:j)'; % modal projections
  subplot(2,2,j+1)
  plot(t1_truncated, ff(1,:), t1_truncated, ff(2,:))
  ylim([240 400])
end
```

```
figure(9)
sig=diag(S);

subplot(2,2,1), plot(sig,'ko','Linewidth',[1.5])
% axis([0 25 0 50])
% set(gca,'Fontsize',[13],'Xtick',[0 5 10 15 20 25])
% text(20,40,'(a)','Fontsize',[13])

subplot(2,2,2), semilogy(sig,'ko','Linewidth',[1.5])
% axis([0 25 10^(-18) 10^(5)])
% set(gca,'Fontsize',[13],'Ytick',[10^(-15) 10^(-10) 10^(-5) 10^0 10^5],...
%    'Xtick',[0 5 10 15 20 25]);
% text(20,10^0,'(b)','Fontsize',[13])

subplot(2,1,2)
plot(t1_truncated,V(:,1),'k',t2_truncated,V(:,2),'k--',t3_truncated,V(:,3),'k:','Linewidth',[2])
% set(gca,'Fontsize',[13])
legend('mode 1','mode 2','mode 3','Location','NorthWest')
% text(0.8,0.35,'(c)','Fontsize',[13])

%% SVD Directions

c = 200;
figure(10)
imshow(vidFrames2_1(:, :, :, 100))
hold on
for i = 100:120
    plot(x2_1(i), y2_1(i), 'r*')
end

plot([x2_1(110), x2_1(110) + c*U(1, 4)], [y2_1(110), y2_1(110) + c*U(1, 3)], 'linewidth', 3)
hold off

title("Paint Can Vertical Displacement")



clc; clear all; close all

%% shaky case
%% Create .avi from given .mat file to use with CV toolbox

load("cam1_2.mat");
videoFWriter = vision.VideoFileWriter('cam1_2.avi');
size1_1 = size(vidFrames1_2);

for i=1:size1_1(4)
  videoFrame = vidFrames1_2(:, :, :, i);
  videoFWriter(videoFrame);
end
release(videoFWriter);

load("cam2_2.mat");
```

```
videoFWriter = vision.VideoFileWriter('cam2_2.avi');
size2_1 = size(vidFrames2_2);

for i=1:size2_1(4)
  videoFrame = vidFrames2_2(:, :, :, i);
  videoFWriter(videoFrame);
end
release(videoFWriter);

load("cam3_2.mat");
videoFWriter = vision.VideoFileWriter('cam3_2.avi');
size3_1 = size(vidFrames3_2);

for i=1:size3_1(4)
  videoFrame = vidFrames3_2(:, :, :, i);
  videoFWriter(videoFrame);
end
release(videoFWriter);

%%  Create object region

videoFileReader1_1 = vision.VideoFileReader('cam1_2.avi');
videoPlayer1_1 = vision.VideoPlayer('Position',[100,100,680,520]);
objectFrame1_1 = videoFileReader1_1();
figure; imshow(objectFrame1_1);
objectRegion1_1=round(getPosition(imrect));

videoFileReader2_1 = vision.VideoFileReader('cam2_2.avi');
videoPlayer2_1 = vision.VideoPlayer('Position',[100,100,680,520]);
objectFrame2_1 = videoFileReader2_1();
figure; imshow(objectFrame2_1);
objectRegion2_1=round(getPosition(imrect));

videoFileReader3_1 = vision.VideoFileReader('cam3_2.avi');
videoPlayer3_1 = vision.VideoPlayer('Position',[100,100,680,520]);
objectFrame3_1 = videoFileReader3_1();
figure; imshow(objectFrame3_1);
objectRegion3_1=round(getPosition(imrect));




%% Track Points

% Camera 1 ----------------------------------------
points = detectMinEigenFeatures(rgb2gray(objectFrame1_1),'ROI',objectRegion1_1, 'MinQuality', 0.001);
tracker = vision.PointTracker('MaxBidirectionalError',3);
initialize(tracker,points.Location,objectFrame1_1);

i = 0;
while ~isDone(videoFileReader1_1)
    i = i+1;
    frame = videoFileReader1_1();
    [points,validity] = tracker(frame);
```

```
    points = points(validity, :);
    x1_1(i) = round(sum(points(:,1))/length(points(:,1)));
    y1_1(i) = round(sum(points(:,2))/length(points(:,2)));
    [points,validity] = tracker(frame);
    out = insertMarker(frame,points(validity, :),'+');
    videoPlayer1_1(out);
    pause(0.01)
end

for i = 1:size1_1(4)-1
    imshow(vidFrames1_2(:, :, :, i))
    hold on
    plot(x1_1(i), y1_1(i), 'r*')
    hold off
    pause(0.01)
end

% Camera 2 -----------------------------------------
points = detectMinEigenFeatures(rgb2gray(objectFrame2_1),'ROI',objectRegion2_1, 'MinQuality', 0.001);
tracker = vision.PointTracker('MaxBidirectionalError',80);
initialize(tracker,points.Location,objectFrame2_1);

i = 0;
while ~isDone(videoFileReader2_1)
    i = i+1;
    frame = videoFileReader2_1();
    [points,validity] = tracker(frame);
    points = points(validity, :);
    x2_1(i) = round(sum(points(:,1))/length(points(:,1)));
    y2_1(i) = round(sum(points(:,2))/length(points(:,2)));
    [points,validity] = tracker(frame);
    out = insertMarker(frame,points(validity, :),'+');
    videoPlayer2_1(out);
    pause(0.01)
end

for i = 1:size2_1(4)-1
    imshow(vidFrames2_2(:, :, :, i))
    hold on
    plot(x2_1(i), y2_1(i), 'r*')
    hold off
    pause(0.01)
end

% Camera 3 -----------------------------------------
points = detectMinEigenFeatures(rgb2gray(objectFrame3_1),'ROI',objectRegion3_1, 'MinQuality', 0.001);
tracker = vision.PointTracker('MaxBidirectionalError',5);
initialize(tracker,points.Location,objectFrame3_1);

i = 0;
while ~isDone(videoFileReader3_1)
    i = i+1;
    frame = videoFileReader3_1();
    [points,validity] = tracker(frame);
```

```
    points = points(validity, :);
    x3_1(i) = round(sum(points(:,1))/length(points(:,1)));
    y3_1(i) = round(sum(points(:,2))/length(points(:,2)));
    [points,validity] = tracker(frame);
    out = insertMarker(frame,points(validity, :),'+');
    videoPlayer3_1(out);
    pause(0.01)
end

for i = 1:size3_1(4)-1
    imshow(vidFrames3_2(:, :, :, i))
    hold on
    plot(x3_1(i), y3_1(i), 'r*')
    hold off
    pause(0.01)
end


%% Aligning video feeds

frame_rate = 30;

t1 = (1:1:(size1_1(4)-1))/frame_rate;
t2 = (1:1:(size2_1(4)-1))/frame_rate;
t3 = (1:1:(size3_1(4)-1))/frame_rate;

figure(1)
plot(t1, x1_1, t1, y1_1)
legend("x", "y")

figure(2)
plot(t2, x2_1, t2, y2_1)
legend("x", "y")


figure(3)
plot(t3, x3_1, t3, y3_1)
legend("x", "y")

x1_1_truncated = x1_1(11:310);
y1_1_truncated = y1_1(11:310);
t1_truncated = t1(11:310) - t1(11);

x2_1_truncated = x2_1(2:301);
y2_1_truncated = y2_1(2:301);
t2_truncated = t2(2:301) - t2(2);

x3_1_truncated = x3_1(16:315);
y3_1_truncated = y3_1(16:315);
t3_truncated = t3(16:315) - t3(16);

figure(4)
plot(t1_truncated, x1_1_truncated, t1_truncated, y1_1_truncated)
legend("x", "y")
```

```
figure(5)
plot(t2_truncated, x2_1_truncated, t2_truncated, y2_1_truncated)
legend("x", "y")

figure(6)
plot(t3_truncated, x3_1_truncated, t3_truncated, y3_1_truncated)
legend("x", "y")

%% Computing SVD

X = [x1_1_truncated; y1_1_truncated; x2_1_truncated;...
    y2_1_truncated; x3_1_truncated; y3_1_truncated];

[U,S,V] = svd(X);

figure(7)
subplot(2,2,1)
plot(t2_truncated, x2_1_truncated, t2_truncated, y2_1_truncated)
ylim([240 400])
for j=1:3
  ff=U(:,1:j)*S(1:j,1:j)*V(:,1:j)'; % modal projections
  subplot(2,2,j+1)
  plot(t1_truncated, ff(3,:), t1_truncated, ff(4,:))
  ylim([240 400])
end

figure(9)
sig=diag(S);

subplot(2,2,1), plot(sig,'ko','Linewidth',[1.5])
% axis([0 25 0 50])
% set(gca,'Fontsize',[13],'Xtick',[0 5 10 15 20 25])
% text(20,40,'(a)','Fontsize',[13])

subplot(2,2,2), semilogy(sig,'ko','Linewidth',[1.5])
% axis([0 25 10^(-18) 10^(5)])
% set(gca,'Fontsize',[13],'Ytick',[10^(-15) 10^(-10) 10^(-5) 10^0 10^5],...
%     'Xtick',[0 5 10 15 20 25]);
% text(20,10^0,'(b)','Fontsize',[13])

subplot(2,1,2)
plot(t1_truncated,V(:,1),'k',t2_truncated,V(:,2),'k--',t3_truncated,V(:,3),'k:','Linewidth',[2])
% set(gca,'Fontsize',[13])
legend('mode 1','mode 2','mode 3','Location','NorthWest')
% text(0.8,0.35,'(c)','Fontsize',[13])

clc; clear all; close all

%% horizontal case
%% Create .avi from given .mat file to use with CV toolbox

load("cam1_3.mat");
videoFWriter = vision.VideoFileWriter('cam1_3.avi');
```

```
size1_3 = size(vidFrames1_3);

for i=1:size1_3(4)
  videoFrame = vidFrames1_3(:, :, :, i);
  videoFWriter(videoFrame);
end
release(videoFWriter);

load("cam2_3.mat");
videoFWriter = vision.VideoFileWriter('cam2_3.avi');
size2_3 = size(vidFrames2_3);

for i=1:size2_3(4)
  videoFrame = vidFrames2_3(:, :, :, i);
  videoFWriter(videoFrame);
end
release(videoFWriter);

load("cam3_3.mat");
videoFWriter = vision.VideoFileWriter('cam3_3.avi');
size3_3 = size(vidFrames3_3);

for i=1:size3_3(4)
  videoFrame = vidFrames3_3(:, :, :, i);
  videoFWriter(videoFrame);
end
release(videoFWriter);
%%  Create object region

videoFileReader1_3 = vision.VideoFileReader('cam1_3.avi');
videoPlayer1_3 = vision.VideoPlayer('Position',[100,100,680,520]);
objectFrame1_3 = videoFileReader1_3();
figure; imshow(objectFrame1_3);
objectRegion1_3=round(getPosition(imrect));

videoFileReader2_3 = vision.VideoFileReader('cam2_3.avi');
videoPlayer2_3 = vision.VideoPlayer('Position',[100,100,680,520]);
objectFrame2_3 = videoFileReader2_3();
figure; imshow(objectFrame2_3);
objectRegion2_3=round(getPosition(imrect));

videoFileReader3_3 = vision.VideoFileReader('cam3_3.avi');
videoPlayer3_3 = vision.VideoPlayer('Position',[100,100,680,520]);
objectFrame3_3 = videoFileReader3_3();
figure; imshow(objectFrame3_3);
objectRegion3_3=round(getPosition(imrect));



%% Track Points

% Camera 1 ------------------------------------------
points = detectMinEigenFeatures(rgb2gray(objectFrame1_3),'ROI',objectRegion1_3, 'MinQuality', 0.1, 'Fil
```

```
tracker = vision.PointTracker('MaxBidirectionalError',10);
initialize(tracker,points.Location,objectFrame1_3);

i = 0;
while ~isDone(videoFileReader1_3)
    i = i+1;
    frame = videoFileReader1_3();
    [points,validity] = tracker(frame);
    points = points(validity, :);
    x1_1(i) = round(sum(points(:,1))/length(points(:,1)));
    y1_1(i) = round(sum(points(:,2))/length(points(:,2)));
    [points,validity] = tracker(frame);
    out = insertMarker(frame,points(validity, :),'+');
    videoPlayer1_3(out);
    pause(0.01)
end

for i = 1:size1_3(4)-1
    imshow(vidFrames1_3(:, :, :, i))
    hold on
    plot(x1_1(i), y1_1(i), 'r*')
    hold off
    pause(0.01)
end

% Camera 2 -----------------------------------------
points = detectMinEigenFeatures(rgb2gray(objectFrame2_3),'ROI',objectRegion2_3, 'MinQuality', 1, 'Filte
tracker = vision.PointTracker('MaxBidirectionalError',5);
initialize(tracker,points.Location,objectFrame2_3);

i = 0;
while ~isDone(videoFileReader2_3)
    i = i+1;
    frame = videoFileReader2_3();
    [points,validity] = tracker(frame);
    points = points(validity, :);
    x2_1(i) = round(sum(points(:,1))/length(points(:,1)));
    y2_1(i) = round(sum(points(:,2))/length(points(:,2)));
    [points,validity] = tracker(frame);
    out = insertMarker(frame,points(validity, :),'+');
    videoPlayer2_3(out);
    pause(0.01)
end

for i = 1:size2_3(4)-1
    imshow(vidFrames2_3(:, :, :, i))
    hold on
    plot(x2_1(i), y2_1(i), 'r*')
    hold off
    pause(0.01)
end

% Camera 3 -----------------------------------------
points = detectMinEigenFeatures(rgb2gray(objectFrame3_3),'ROI',objectRegion3_3, 'MinQuality', 0.3, 'Fil
```

```
tracker = vision.PointTracker('MaxBidirectionalError',10);
initialize(tracker,points.Location,objectFrame3_3);

i = 0;
while ~isDone(videoFileReader3_3)
    i = i+1;
    frame = videoFileReader3_3();
    [points,validity] = tracker(frame);
    points = points(validity, :);
    x3_1(i) = round(sum(points(:,1))/length(points(:,1)));
    y3_1(i) = round(sum(points(:,2))/length(points(:,2)));
    [points,validity] = tracker(frame);
    out = insertMarker(frame,points(validity, :),'+');
    videoPlayer3_3(out);
    pause(0.01)
end

for i = 1:size3_3(4)-1
    imshow(vidFrames3_3(:, :, :, i))
    hold on
    plot(x3_1(i), y3_1(i), 'r*')
    hold off
    pause(0.01)
end

%% Aligning video feeds

frame_rate = 30;

t1 = (1:1:(size1_3(4)-1))/frame_rate;
t2 = (1:1:(size2_3(4)-1))/frame_rate;
t3 = (1:1:(size3_3(4)-1))/frame_rate;

figure(1)
plot(t1, x1_1, t1, y1_1)
legend("x", "y")

figure(2)
plot(t2, x2_1, t2, y2_1)
legend("x", "y")


figure(3)
plot(t3, x3_1, t3, y3_1)
legend("x", "y")

x1_1_truncated = x1_1(16:236) - mean(x1_1(16:236));
y1_1_truncated = y1_1(16:236) - mean(y1_1(16:236));
t1_truncated = t1(16:236) - t1(16);

x2_1_truncated = x2_1(21:241) - mean(x2_1(21:241));
y2_1_truncated = y2_1(21:241) - mean(y2_1(21:241));
t2_truncated = t2(21:241) - t2(21);
```

```
x3_1_truncated = x3_1(11:231) - mean(x3_1(11:231));
y3_1_truncated = y3_1(11:231) - mean(y3_1(11:231));
t3_truncated = t3(11:231) - t3(11);

figure(4)
plot(t1_truncated, x1_1_truncated, t1_truncated, y1_1_truncated)
legend("x", "y")

figure(5)
plot(t2_truncated, x2_1_truncated, t2_truncated, y2_1_truncated)
legend("x", "y")

figure(6)
plot(t3_truncated, x3_1_truncated, t3_truncated, y3_1_truncated)
legend("x", "y")

%% Computing SVD

X = [x1_1_truncated/max(x1_1_truncated); y1_1_truncated/max(y1_1_truncated); x2_1_truncated/max(x2_1_tru
    y2_1_truncated/max(y2_1_truncated); x3_1_truncated/max(x3_1_truncated); y3_1_truncated/max(x3_1_trun

[U,S,V] = svd(X);

figure(7)
subplot(2,2,1)
plot(t2_truncated, x2_1_truncated, t2_truncated, y2_1_truncated)
ylim([200 400])
for j=1:3
  ff=U(:,1:j)*S(1:j,1:j)*V(:,1:j)'; % modal projections
  subplot(2,2,j+1)
  plot(t1_truncated, ff(3,:), t1_truncated, ff(4,:))
  ylim([200 400])
end

figure(9)
sig=diag(S);

subplot(2,2,1), plot(sig,'ko','Linewidth',[1.5])
% axis([0 25 0 50])
% set(gca,'Fontsize',[13],'Xtick',[0 5 10 15 20 25])
% text(20,40,'(a)','Fontsize',[13])

subplot(2,2,2), semilogy(sig,'ko','Linewidth',[1.5])
% axis([0 25 10^(-18) 10^(5)])
% set(gca,'Fontsize',[13],'Ytick',[10^(-15) 10^(-10) 10^(-5) 10^0 10^5],...
%     'Xtick',[0 5 10 15 20 25]);
% text(20,10^0,'(b)','Fontsize',[13])

subplot(2,1,2)
plot(t1_truncated,V(:,1),'k',t2_truncated,V(:,2),'k--',t3_truncated,V(:,3),'k:','Linewidth',[2])
% set(gca,'Fontsize',[13])
legend('mode 1','mode 2','mode 3','Location','NorthWest')
% text(0.8,0.35,'(c)','Fontsize',[13])
```

```
%%

c = 100;
figure(10)
imshow(vidFrames2_3(:, :, :, 100))
hold on
for i = 20:120
    plot(x2_1(i), y2_1(i), 'r*')
end

plot([x2_1(100), x2_1(100) + c*U(3, 1)], [y2_1(100), y2_1(100) + c*U(4, 1)], 'linewidth', 3)
plot([x2_1(100), x2_1(100) + c*U(3, 2)], [y2_1(100), y2_1(100) + c*U(4, 2)], 'linewidth', 3)
hold off

title("Paint Can Vertical and Horizontal Displacement")

clc; clear all; close all

%% Rotating case
%% Create .avi for CV

load("cam1_4.mat");
vidFrames1_4 = vidFrames1_4(179:179+218, 321:321+165, :, :);
videoFWriter = vision.VideoFileWriter('cam1_4.avi');
size1_4 = size(vidFrames1_4);
videoFileReader1_4 = vision.VideoFileReader('cam1_4.avi');

for i=1:size1_4(4)
  videoFrame = vidFrames1_4(:, :, :, i);
  videoFWriter(videoFrame);
end

load("cam2_4.mat");
vidFrames2_4 = vidFrames2_4(90:90+300, 160:160+290, :, :);
videoFWriter = vision.VideoFileWriter('cam2_4.avi');
size2_4 = size(vidFrames2_4);
videoFileReader2_4 = vision.VideoFileReader('cam2_4.avi');


for i=1:size2_4(4)
  videoFrame = vidFrames2_4(:, :, :, i);
  videoFWriter(videoFrame);
end
release(videoFWriter);

load("cam3_4.mat");
vidFrames3_4 = vidFrames3_4(139:139+136, 300:319+160, :, :);
videoFWriter = vision.VideoFileWriter('cam3_4.avi');
size3_4 = size(vidFrames3_4);
videoFileReader3_4 = vision.VideoFileReader('cam3_4.avi');


for i=1:size3_4(4)
  videoFrame = vidFrames3_4(:, :, :, i);
```

```
  videoFWriter(videoFrame);
end
release(videoFWriter);

%% Get object region

release(videoFWriter);
videoPlayer1_4 = vision.VideoPlayer('Position',[100,100,680,520]);
objectFrame1_4 = videoFileReader1_4();
figure; imshow(objectFrame1_4);
objectRegion1_4=round(getPosition(imrect));

release(videoFWriter);
videoPlayer2_4 = vision.VideoPlayer('Position',[100,100,680,520]);
objectFrame2_4 = videoFileReader2_4();
figure; imshow(objectFrame2_4);
objectRegion2_4=round(getPosition(imrect));

release(videoFWriter);
videoPlayer3_4 = vision.VideoPlayer('Position',[100,100,680,520]);
objectFrame3_4 = videoFileReader3_4();
figure; imshow(objectFrame3_4);
objectRegion3_4=round(getPosition(imrect));

%% Find matching RGB values from region
figure(1)

% Camera 1 ----------------------------------------
RGB_rect1 = vidFrames1_4(objectRegion1_4(2):objectRegion1_4(2)+objectRegion1_4(4),...
    objectRegion1_4(1):objectRegion1_4(1)+objectRegion1_4(3), :, 1);

RGB_rect1 = double(RGB_rect1);

RGB_rect_size1 = size(RGB_rect1);
length_rect1 = RGB_rect_size1(1)*RGB_rect_size1(2);

R_mean1 = sum(RGB_rect1(:, :, 1), 'all')/length_rect1;
R_std1 = 0.45*sqrt(var(RGB_rect1(:, :, 1),0,'all'));
R_min1 = R_mean1 - R_std1; R_max1 = R_mean1 + R_std1;

G_mean1 = sum(RGB_rect1(:, :, 2), 'all')/length_rect1;
G_std1 = 0.45*sqrt(var(RGB_rect1(:, :, 2),0,'all'));
G_min1 = G_mean1 - G_std1; G_max1 = G_mean1 + G_std1;

B_mean1 = sum(RGB_rect1(:, :, 3), 'all')/length_rect1;
B_std1 = 0.45*sqrt(var(RGB_rect1(:, :, 3),0,'all'));
B_min1 = B_mean1 - B_std1; B_max1 = B_mean1 + B_std1;


for i = 1:size1_4(4)
    num = 0;
    for j = 1:size1_4(1)
        for k = 1:size1_4(2)
            if (vidFrames1_4(j, k, 1, i)>R_min1) && (vidFrames1_4(j, k, 1, i)<R_max1)...
```

```
                    && (vidFrames1_4(j, k, 2, i)>G_min1) && (vidFrames1_4(j, k, 2, i)<G_max1)...
                    && (vidFrames1_4(j, k, 3, i)>B_min1) && (vidFrames1_4(j, k, 3, i)<B_max1)
                num = num+1;
                points1(num,1,i) = k;
                points1(num,2,i) = j;
            end
        end
    end
    if num == 0
        points1(1,1,i) = 1;
        points1(1,2,i) = 1;
    end
end

for i = 1:size1_4(4)
    imshow(vidFrames1_4(:, :, :, i))
    hold on
    scatter(points1(:, 1, i), points1(:,2,i), 'r*')
    hold off
    pause(0.01)
end

% Camera 2 ---------------------------------------
RGB_rect2 = vidFrames2_4(objectRegion2_4(2):objectRegion2_4(2)+objectRegion2_4(4),...
    objectRegion2_4(1):objectRegion2_4(1)+objectRegion2_4(3), :, 1);

RGB_rect2 = double(RGB_rect2);

RGB_rect_size2 = size(RGB_rect2);
length_rect2 = RGB_rect_size2(1)*RGB_rect_size2(2);

R_mean2 = sum(RGB_rect2(:, :, 1), 'all')/length_rect2;
R_std2 = 0.45*sqrt(var(RGB_rect2(:, :, 1),0,'all'));
R_min2 = R_mean2 - R_std2; R_max = R_mean2 + R_std2;

G_mean2 = sum(RGB_rect2(:, :, 2), 'all')/length_rect2;
G_std2 = 0.45*sqrt(var(RGB_rect2(:, :, 2),0,'all'));
G_min2 = G_mean2 - G_std2; G_max = G_mean2 + G_std2;

B_mean2 = sum(RGB_rect2(:, :, 3), 'all')/length_rect2;
B_std2 = 0.45*sqrt(var(RGB_rect2(:, :, 3),0,'all'));
B_min2 = B_mean2 - B_std2; B_max = B_mean2 + B_std2;


for i = 1:size2_4(4)
    num = 0;
    for j = 1:size2_4(1)
        for k = 1:size2_4(2)
            if (vidFrames2_4(j, k, 1, i)>R_min2) && (vidFrames2_4(j, k, 1, i)<R_max)...
                    && (vidFrames2_4(j, k, 2, i)>G_min2) && (vidFrames2_4(j, k, 2, i)<G_max)...
                    && (vidFrames2_4(j, k, 3, i)>B_min2) && (vidFrames2_4(j, k, 3, i)<B_max)
                num = num+1;
                points2(num,1,i) = k;
                points2(num,2,i) = j;
```

```
            end
        end
    end
    if num == 0
        points2(1,1,i) = 1;
        points2(1,2,i) = 1;
    end
end

for i = 1:size2_4(4)
    imshow(vidFrames2_4(:, :, :, i))
    hold on
    scatter(points2(:, 1, i), points2(:,2,i), 'r*')
    hold off
    pause(0.01)
end

% Camera 3 ---------------------------------------

RGB_rect3 = vidFrames3_4(objectRegion3_4(2):objectRegion3_4(2)+objectRegion3_4(4),...
    objectRegion3_4(1):objectRegion3_4(1)+objectRegion3_4(3), :, 1);

RGB_rect3 = double(RGB_rect3);

RGB_rect_size3 = size(RGB_rect3);
length_rect3 = RGB_rect_size3(1)*RGB_rect_size3(2);

R_mean3 = sum(RGB_rect3(:, :, 1), 'all')/length_rect3;
R_std3 = 1.2*sqrt(var(RGB_rect3(:, :, 1),0,'all'));
R_min3 = R_mean3 - R_std3; R_max3 = R_mean3 + R_std3;

G_mean3 = sum(RGB_rect3(:, :, 2), 'all')/length_rect3;
G_std3 = 1.2*sqrt(var(RGB_rect3(:, :, 2),0,'all'));
G_min3 = G_mean3 - G_std3; G_max3 = G_mean3 + G_std3;

B_mean3 = sum(RGB_rect3(:, :, 3), 'all')/length_rect3;
B_std3 = 1.2*sqrt(var(RGB_rect3(:, :, 3),0,'all'));
B_min3 = B_mean3 - B_std3; B_max3 = B_mean3 + B_std3;


for i = 1:size3_4(4)
    num = 0;
    for j = 1:size3_4(1)
        for k = 1:size3_4(2)
            if (vidFrames3_4(j, k, 1, i)>R_min3) && (vidFrames3_4(j, k, 1, i)<R_max3)...
                    && (vidFrames3_4(j, k, 2, i)>G_min3) && (vidFrames3_4(j, k, 2, i)<G_max3)...
                    && (vidFrames3_4(j, k, 3, i)>B_min3) && (vidFrames3_4(j, k, 3, i)<B_max3)
                num = num+1;
                points3(num,1,i) = k;
                points3(num,2,i) = j;
            end
        end
    end
    if num == 0
```

```
        points3(1,1,i) = 1;
        points3(1,2,i) = 1;
    end
end

for i = 1:size3_4(4)
    imshow(vidFrames3_4(:, :, :, i))
    hold on
    scatter(points3(:, 1, i), points3(:,2,i), 'r*')
    hold off
    pause(0.01)
end

%% Plotting

frame_rate = 30;

t1 = (1:1:(size1_4(4)))/frame_rate;
t2 = (1:1:(size2_4(4)))/frame_rate;
t3 = (1:1:(size3_4(4)))/frame_rate;

for i = 1:size1_4(4)
    x1(i) = sum(points1(:,1,i))/sum(points1(:,1,i)~=0);
    y1(i) = sum(points1(:,2,i))/sum(points1(:,2,i)~=0);
end

for i = 1:size2_4(4)
    x2(i) = sum(points2(:,1,i))/sum(points2(:,1,i)~=0);
    y2(i) = sum(points2(:,2,i))/sum(points2(:,2,i)~=0);
end

for i = 1:size3_4(4)
    x3(i) = sum(points3(:,1,i))/sum(points3(:,1,i)~=0);
    y3(i) = sum(points3(:,2,i))/sum(points3(:,2,i)~=0);
end

figure(1)
plot(t1, x1, t1, y1)

figure(2)
plot(t2, x2, t2, y2)

figure(3)
plot(t3, x3, t3, y3)

%% Filter

figure(5)
x1f = lowpass(x1,2,frame_rate);
y1f = lowpass(y1,2,frame_rate);
plot(t1, x1f, t1, x1)
legend("Filtered", "Raw")

figure(6)
```

```
x2f = lowpass(x2,2,frame_rate);
y2f = lowpass(y2,2,frame_rate);
plot(t2, x2f, t2, x2)
legend("Filtered", "Raw")

figure(7)
x3f = lowpass(x3,2,frame_rate);
y3f = lowpass(y3,2,frame_rate);
plot(t3, x3f, t3, x3)
legend("Filtered", "Raw")


%% SVD




x1_1_truncated = x1f(33:380) - mean(x1f(33:380));
y1_1_truncated = y1f(33:380) - mean(y1f(33:380));
t1_truncated = t1(33:380) - t1(33);

x2_1_truncated = x2f(20:367) - mean(x2f(20:367));
y2_1_truncated = y2f(20:367) - mean(y2f(20:367));
t2_truncated = t2(20:367) - t2(20);

x3_1_truncated = x3f(32:379) - mean(x3f(32:379));
y3_1_truncated = y3f(32:379) - mean(y3f(32:379));
t3_truncated = t3(32:379) - t3(32);


X = [x1_1_truncated; y1_1_truncated; x2_1_truncated;...
    y2_1_truncated; x3_1_truncated; y3_1_truncated];

[U,S,V] = svd(X);

figure(8)
subplot(2,2,1)
plot(t2_truncated, x2_1_truncated, t2_truncated, y2_1_truncated)
for j=1:3
  ff=U(:,1:j)*S(1:j,1:j)*V(:,1:j)'; % modal projections
  subplot(2,2,j+1)
  plot(t1_truncated, ff(3,:), t1_truncated, ff(4,:))
end

figure(9)
sig=diag(S);

subplot(2,2,1), plot(sig,'ko','Linewidth',[1.5])
% axis([0 25 0 50])
% set(gca,'Fontsize',[13],'Xtick',[0 5 10 15 20 25])
% text(20,40,'(a)','Fontsize',[13])

subplot(2,2,2), semilogy(sig,'ko','Linewidth',[1.5])
% axis([0 25 10^(-18) 10^(5)])
```

```
% set(gca,'Fontsize',[13],'Ytick',[10^(-15) 10^(-10) 10^(-5) 10^0 10^5],...
%    'Xtick',[0 5 10 15 20 25]);
% text(20,10^0,'(b)','Fontsize',[13])

subplot(2,1,2)
plot(t1_truncated,V(:,1),'k',t2_truncated,V(:,2),'k--',t3_truncated,V(:,3),'k:','Linewidth',[2])
% set(gca,'Fontsize',[13])
legend('mode 1','mode 2','mode 3','Location','NorthWest')
% text(0.8,0.35,'(c)','Fontsize',[13])

%% SVD Directions

c = 200;
figure(10)
imshow(vidFrames1_4(:, :, :, 100))
hold on
for i = 100:120
    plot(x1f(i), y1f(i), 'r*')
end

plot([x1f(110), x1f(110) + c*U(1, 1)], [y1f(110), y1f(110) + c*U(2, 2)], 'linewidth', 3)
plot([x1f(110), x1f(110) + c*U(1, 2)], [y1f(110), y1f(110) + c*U(2, 2)], 'linewidth', 3)
plot([x1f(110), x1f(110) + c*U(1, 3)], [y1f(110), y1f(110) + c*U(2, 3)], 'linewidth', 3)

hold off

title("Paint Can Vertical Displacement")
```