Priyanshi Rastogi
AMATH 482
2/18/2020

PCA
By: Priyanshi Rastogi

1. **Abstract:** In this report we are analyzing different motion videos by using Singular Value Decomposition (SVD) to demonstrate difference aspects of the Principal Component Analysis (PCA). We will see its practical usefulness and the effects of noise on the results that are produced using the PCA algorithms. We will see how it produces different results based on noisier videos, videos with added shakiness and ideal scenario videos (with minimum noisiness or shakiness).

2. **Introduction/Overview:** We are given four different test scenarios with three different camera angle point of views in each video. Each video displays a paint can moving in the x, y, and/or z direction with a flashlight on top of the paint can to help track the location of it. Test 1 is the ideal case where there is only movement in the z direction. Test 2 is the noisy case where camera shake is introduced so movement in not just the z direction but also the x direction. Test 3 is where instead of the paint can (released off-center) moving completely in the vertical direction (z axis), the displacement is happening in the horizontal direction (y axis) causing motion in the x direction and as well as in the z direction. Test 4 is similar to test 3 but instead of moving just back and forth in the y direction the paint can is rotating, thus producing motion in the x-y plane, rotation and motion in the z direction. We used a built in tool vision.PointTracker to capture the location of the paint with the help of the flashlight to help us get the points of the location and then used SVD to further use PCA on the resulting findings.

3. **Theoretical Background:** We extracted the points of the paint can's motion using the built in Matlab tool vision.PointTracker for each camera angle for each test. In each case our goal was to get the simple harmonic motion. Thus, we use the process of PCA to extract out the ideal or simplified behavior (Kutz).
   Once we got the points, we had to use Singular Value Decomposition to get this Principal Component Analysis. The SVD equation is:

$$\mathbf{A} = \boldsymbol{U\Sigma V^T} \quad \ldots (1)$$

   Where A is the snapshot matrix (consisting of our points of the location of the paint can) and where U and V are orthonormal matrices, containing the left eigenvectors and the right eigenvectors respectively. Here the rows of V^T tell you how much the singular values change over time. Σ consists of the singular values of A (meaning the eigenvalues of $A^T A$). Conceptually, in our case, U contains the principal components telling us by how much (Σ) each principal component direction is being stretched. Σ in this case, holds the scalars (or the singular values) by how much we would want to do that stretching. Computing each singular value's energy tells us information that we can use to compute a rank approximation as well as tells us how much energy is in each mode. We want to value the modes with the highest energies because that tells us that most of our data lies within those first 1, 2, 3, or how many ever of those axes, thus telling us what might possibly be accounting for redundancies or noise.

   Energy Equation (Gin's version): $\sigma_1{}^2 + \sigma_2{}^2 + \cdots \sigma_N{}^2 / \sigma_1{}^2 + \sigma_2{}^2 + \cdots \sigma_r{}^2 \ldots (2)$

Here all of the sigma values are just the singular values of the matrix Σ. N is the rank approximation number we are using, and r is the rank of the whole matrix. Computing the energies is able to tell us how much of the system's energy is in each mode and tells us that we can use a lower rank matrix to get the majority of our data, eliminating any redundancies or noise in the given data. In this version of the equation we just squared each sigma value to amplify/scale up the values.

4. Algorithm Implementation and Development:

(Since the procedure for finding the tracked points of the flashlight is the same for each test case and each camera angle, we are just describing it for one (test case 1 cam 1) and it the same method applies to all test cases & camera angles, thus referring to a set of lines for a method we used in one test & camera angle applies similarly to all test cases & camera angles just with different values)

1. First, we load in our data (again going to refer generally to one section, since each section is just a repeat of another except the initial point values are changing and the videoframes are changing with respect to each camera angle and each test). (see lines 1, 20, 39, …)
2. We then installed the toolbox required in order to use vision.PointTracker. It requires us to create a pointTracker object, initialize a point, and then use vision.PointTracker's built in step function to start tracking the point given to it at each frame in the loop, (see lines 2-9).
3. We then save the location of the point being tracked at each frame into a matrix with X,Y coordinates of the point's location so that we can use those point locations (for each test) and, later, put it into a snapshot matrix. (see line 9)
4. The point tracker tool sometimes loses the point that it was tracking, so in order to reset it to track the point of the flashlight, we watched each frame and plotted the point that was being tracked on it and paused it whenever the point would be way off the flashlight. When we paused it, we got the time at which we needed to reset the point as well as (by using the built in DataGrips tool) were able to find where the flashlight was at that time and reset the points location to that. (see lines 85-93, 300-355, etc.)
5. After we got all the (x,y) points tracking of the flashlight for each camera angle for each test, we plotted the y displacement for all three cameras over time (for each test) and then were able to compare and see where to trim the graph (or our points data) (See figures 1-8). We were able to tell this by picking the same length of time for all three angles in order to make sure it fit within the matrix, as well as making sure that the start and the end points that we were trimming at were in the same direction as the start and the end of the other camera angles. After playing around and comparing the lengths/directions of the start and end points we were able to find a time for each camera angle (for each test) to trim it at. We then plotted the X displacement over the trimmed time as well as the Y-displacement over the trimmed time for each test case. (see figures 1-8, see lines 15-16, 34-35, etc. and see lines 376-455 for plotting).
6. Once we trimmed down the length of our data, we were able to put it into a snapshot matrix (a matrix for each test with x points stacked over y points and the columns indicating time values). We wanted to make sure that we subtracted the mean of each row (mean of all the X points and mean of all Y points) from each

corresponding row because we didn't want the position of where we started our video for each test to result in different data values (see lines 17-18, 36-37, etc.)We then used MATLAB's built in SVD function to find the reduced SVD form for each snapshot matrix (see lines 348-370).

7. After calculating the reduced SVD we were able to get the singular values for each matrix and then plotted the energy for each singular value. We used a log scale ('loglog) to plot these because using the log scale shows us the differences between the small values much clearer than if we just used the normal 'plot' command. (using Professor Gin's equation of energy, see eq. #2) (see figure 9, see lines 457-end)

5. **Computational Results:** If we look at our y-displacement vs. time plots for each test we are able to see graphs that mimic sin graphs due to the amount of displacement happening in that direction (see figures 2, 4, 6, and 8). Similarly, if we look at the x-displacement vs. time graphs we are able to see that in some scenarios, like camera 1 & 2 test 1, the x-displacement didn't really have a change in the graph because there wasn't much changing happening in that direction except for what accounted for some noise or shakiness (see figure 1). However, if we compare it to, for ex., test 4's x-displacement graphs we can see that it shows much more of a change in the x-displacement which makes sense because test 4 had not only horizontal displacement but rotation and shakiness as well causing all directions to have changes in displacement (see figure 7).

   From our energy plot we are able to see that for test one, we end up getting something that looks like a rank 1 approximation because there is one very high singular value and then the jump between the first and the second is very drastic. The first energy value came out to be around .9758 which means that the first mode captures 97.58% of the data. This shows that most of the displacement was happening in one direction for this test. Similarly, for test 2, we can see that the it looks almost like a rank 3 approximation. There is approx. 60.93% of the system's energy in the first mode ~16.26% in the second mode and ~15.18% in the third mode. Because the energies in the second and third modes are fairly low this tells us that it is coming from either noise of just the shakiness that occurred when taking the video. Test 3 gave us about a rank 3 approximation again, showing us that the first mode had ~54.14% of the system's energy, ~26.31% in the second mode and ~16.48% in the third mode. This tells us that there was not just displacement in the vertical direction but as well as in the horizontal direction because of that 26.31% and 16.48%. Test 4 gave us energies of ~64.88%, ~13.35%, ~11.64%, for the first, second, and third modes respectively giving us a rank approximation of about 3 (see figure 9). Since some rotation was happening it is a little more difficult to tell from these numbers whether that was completely rotation, or the values were affected by noise as well. Since we had to keep adjusting our point tracker and resetting it, it didn't track it completely flawlessly (but still as close as possible) so this might have also caused the other values to go up resulting in some slight error. Since we found out that our rank approximations were either 1,2, or 3 then if we took the first three (or two or 1) columns of U multiplied it with the same number of the singular values and again multiplied it with the same number of rows of V^T then we would be able to get where most of our data is coming from directionally without the redundancies or noise. U, which contains the principal components, tell us in which direction the stretching is happening in but since our dimension is in 6D it is hard to interpret or see that.

Priyanshi Rastogi
AMATH 482
2/18/2020

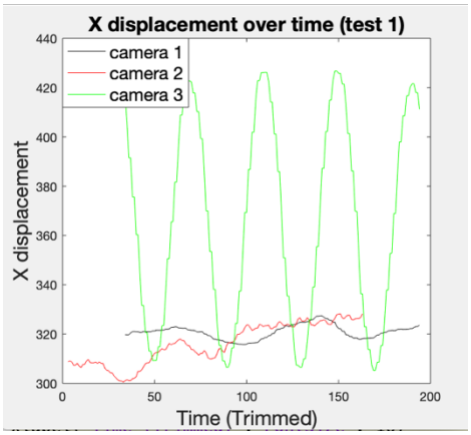

Figure 1: shows the x-displacement over time for cameras 1, 2, 3 for test 1

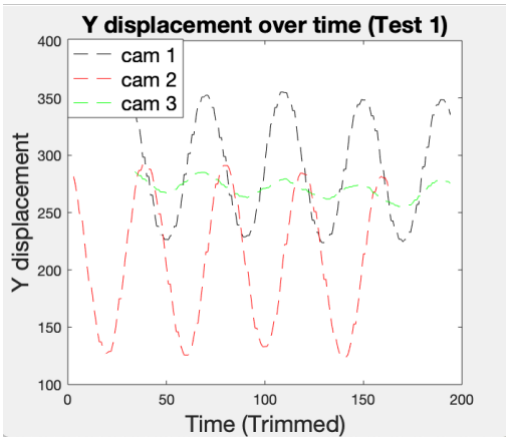

Figure 2: shows the y-displacement over time for cameras 1, 2, 3 for test 1

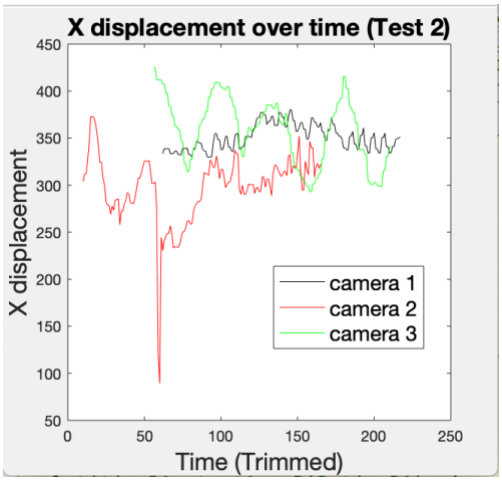

Figure 3: shows the x-displacement over time for cameras 1, 2, 3 for test 2

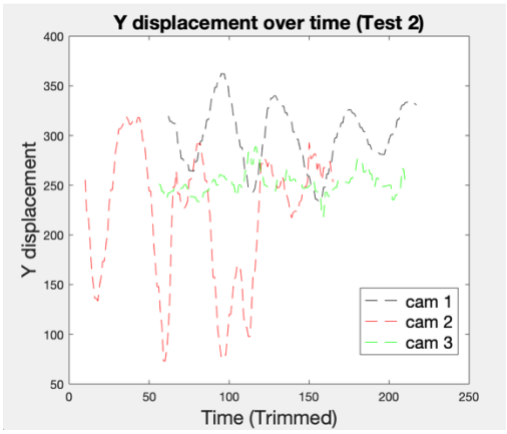

Figure 4: shows the y-displacement over time for cameras 1, 2, 3 for test 2

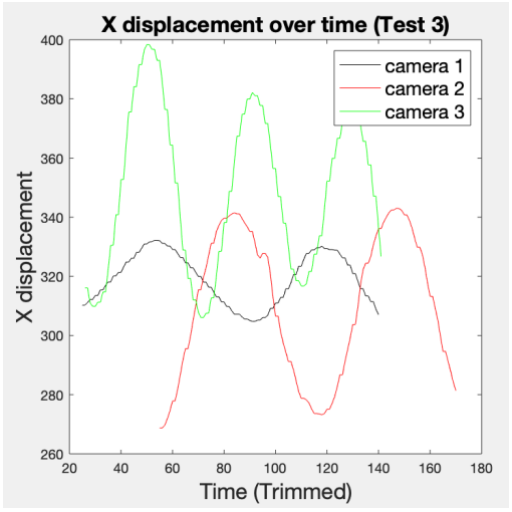

Figure 5: shows the x-displacement over time for cameras 1, 2, 3 for test 3

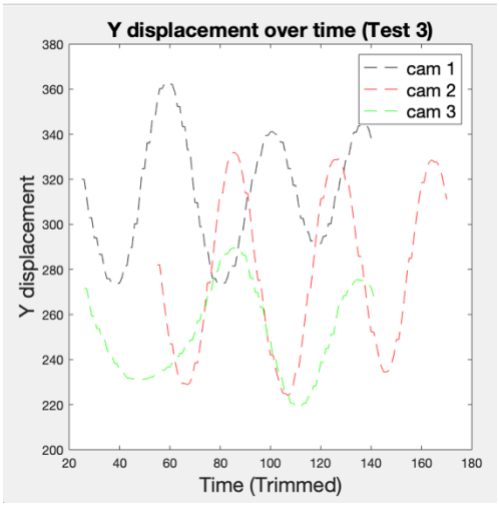

Figure 6: shows the y-displacement over time for cameras 1, 2, 3 for test 3

Priyanshi Rastogi
AMATH 482
2/18/2020

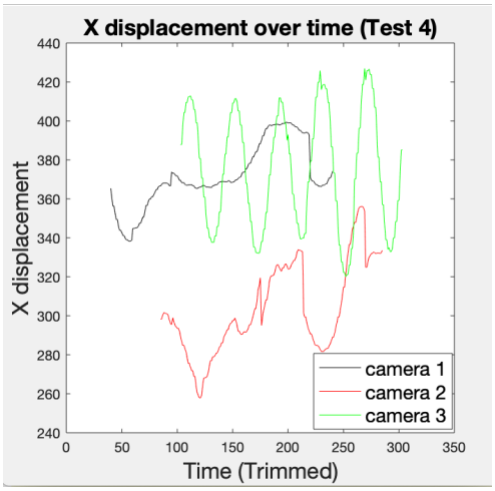

Figure 7: shows the x-displacement over time for cameras 1, 2, 3 for test 4

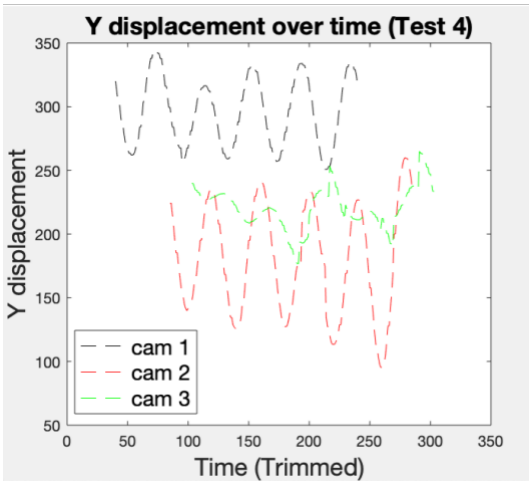

Figure 8: shows the Y-displacement over time for cameras 1, 2, 3 for test 4

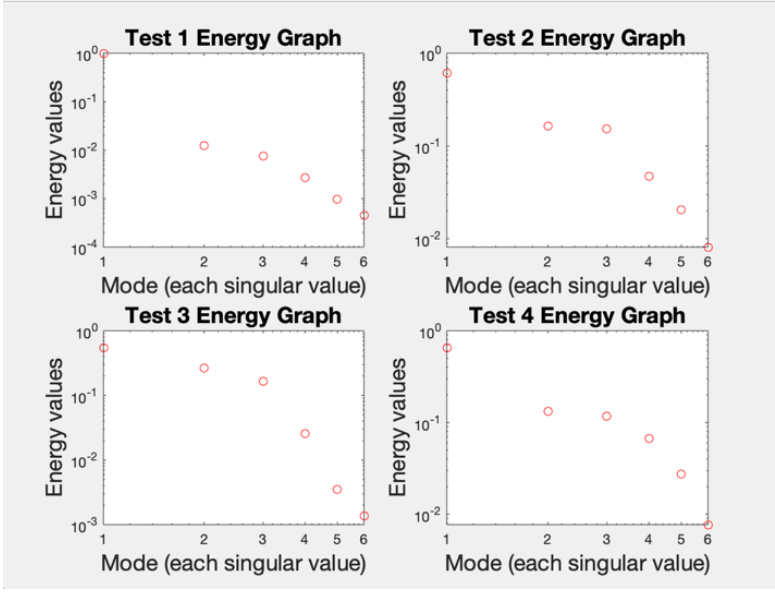

Figure 9: shows the energies of each singular value for each test

6. Summary and Conclusions: Overall, we were able to see how SVD is able to tell us about how our data is being transformed directionally. We were able to see that U (orthogonal basis for R^6 in this case) contained the principal components and told us in which direction the most displacement was happening when multiplied with the singular values that were found in Σ. If we then further multiply this by V^T we get a projection of our data onto the least rank possible (which we were able to see by calculating the energies of each singular value). From the energies of the modes, we saw that certain tests like test 1 clearly had a rank 1 approximation meaning that there was only a significant change in displacement happening in one direction (the z direction) whereas test 3 for example showed us a pretty clear rank 3 approximation which meant that there was a change in displacement in 3 different directions but with one direction containing the most amount of the change and the rest of the modes (that we didn't include in the rank approximation) accounting for noise/redundancies giving us the ideal angle to look at for the videos.

Priyanshi Rastogi
AMATH 482
2/18/2020

7. Appendix A-Function Descriptions:

-loglog(x,y): does a log log scale plot on each parameter passed in
-vision.PointTracker: The point tracker object tracks a set of points using the Kanade-Lucas-Tomasi (KLT), feature-tracking algorithm.
-setPoints(pointTracker, [x,y]): sets the point to specified x,y (or in higher dimensions) location
- plot(x,y): plots passed in parameters x over y
- subplot(rows, columns, index): puts the plot in the specifced index of the subplot (that is of dimension rows x columns)
- zeros(n,m): creates a matrix of size nxm with all zeros as its values.
- length(v): returns the length of the passed in parameter
-size(x): Gives you the dimension of the passed in parameter
-initialize(pointTracker,points,I): initializes points to track and sets the initial video frame. The initial locations points, must be an *M*-by-2 array of [x y] coordinates. The initial video frame, I, must be a 2-D grayscale or RGB image and must be the same size and data type as the video frames passed to the step method.
-step(pointTracker, X); returns the new location of the point being tracked in frame X

8. Appendix B:

```
%% CAMERA 1 (camera 1 test 1) cam1_1 DONEE
load('cam1_1.mat')
pointTracker = vision.PointTracker;
numFrames = size(vidFrames1_1,4);
points = [319,228];
initialize(pointTracker, points, vidFrames1_1(:,:,:,1));
test1cam1 = zeros(2,numFrames);
for j = 1:numFrames
      X = vidFrames1_1(:,:,:,j);
      test1cam1(:,j) = step(pointTracker, X);
      %     imshow(X);
      %     hold on
      %     plot(test1cam1(1,j), test1cam1(2,j), 'ro')
      %     pause(0.1)
      end
      time11 = [34:194]; %trimmed length of 160
      trimmed1test1 = test1cam1(:,time11);
      trimmed1test1(1,:) = trimmed1test1(1,:) -
mean(trimmed1test1(1,:));
      trimmed1test1(2,:) = trimmed1test1(2,:) -
mean(trimmed1test1(2,:));
      %% CAMERA 2 Test 1 cam2_1 DONEE
      load('cam2_1.mat')
      pointTracker = vision.PointTracker;
      numFrames = size(vidFrames2_1,4);
      points = [309,288]; %based on ginput(1)
      initialize(pointTracker, points, vidFrames2_1(:,:,:,1));
      test1cam2 = zeros(2,numFrames);
      for j = 1:numFrames
```

Priyanshi Rastogi
AMATH 482
2/18/2020

```
            X = vidFrames2_1(:,:,:,j);
            test1cam2(:,j) = step(pointTracker, X);
            %imshow(X);
            %hold on
            %plot(test1cam2(1,j), test1cam2(2,j), 'ro')
            %pause(0.1)
      end
      time21 = [3:163];
      trimmed2test1 = test1cam2(:,time21);
      trimmed2test1(1,:) = trimmed2test1(1,:) -
   mean(trimmed2test1(1,:));
      trimmed2test1(2,:) = trimmed2test1(2,:) -
   mean(trimmed2test1(2,:));
      %% CAMERA 3 test 1 cam3_1 DONEE
      load('cam3_1.mat')
      pointTracker = vision.PointTracker;
      numFrames = size(vidFrames3_1,4);
      points = [335,280]; %based on ginput(1)
      initialize(pointTracker, points, vidFrames3_1(:,:,:,1));
      test1cam3 = zeros(2,numFrames);
      for j = 1:numFrames
            X = vidFrames3_1(:,:,:,j);
            test1cam3(:,j) = step(pointTracker, X);
            %     imshow(X);
            %     hold on
            %     plot(test1cam3(1,j), test1cam3(2,j), 'ro')
            %     pause(0.1)
      end
      time31 = [34:194];
      trimmed3test1 = test1cam3(:,time31);
      trimmed3test1(1,:) = trimmed3test1(1,:) -
   mean(trimmed3test1(1,:));
      trimmed3test1(2,:) = trimmed3test1(2,:) -
   mean(trimmed3test1(2,:));
      %% CAMERA 1 Test 2 cam1_2 DONEE
      load('cam1_2.mat')
      pointTracker = vision.PointTracker;
      numFrames = size(vidFrames1_2,4);
      points = [325,310]; %based on ginput(1)
      initialize(pointTracker, points, vidFrames1_2(:,:,:,1));
      test2cam1 = zeros(2,numFrames);
      for j = 1:numFrames
            X = vidFrames1_2(:,:,:,j);
            test2cam1(:,j) = step(pointTracker, X);
            %     imshow(X);
            %     hold on
            %     plot(test2cam1(1,j), test2cam1(2,j), 'ro')
            %     pause(0.1)
      end
```

Priyanshi Rastogi
AMATH 482
2/18/2020

```
72.     time12 = [62:217]; %chosen time length: 155
73.     trimmed1test2 = test2cam1(:,time12);
74.     trimmed1test2(1,:) = trimmed1test2(1,:) -
   mean(trimmed1test2(1,:));
75.     trimmed1test2(2,:) = trimmed1test2(2,:) -
   mean(trimmed1test2(2,:));
76.     %% CAMERA 2 Test 2 cam2_2 DONEE
77.     load('cam2_2.mat')
78.     pointTracker = vision.PointTracker;
79.     numFrames = size(vidFrames2_2,4);
80.     points = [313,360]; %based on ginput(1)
81.     initialize(pointTracker, points, vidFrames2_2(:,:,:,1));
82.     test2cam2 = zeros(2,numFrames);
83.     for j = 1:numFrames
84.           X = vidFrames2_2(:,:,:,j);
85.           if (j == 61)
86.           setPoints(pointTracker, [245,84])
87.           end
88.           if (j == 257)
89.           setPoints(pointTracker, [325,144])
90.           end
91.           if (j == 297)
92.           setPoints(pointTracker, [323,104])
93.           end
94.           test2cam2(:,j) = step(pointTracker, X);
95.     %     imshow(X);
96.     %     hold on
97.     %     plot(test2cam2(1,j), test2cam2(2,j), 'ro')
98.     %     pause(0.1)
99.     end
100.    time22 = [10:165];
101.    trimmed2test2 = test2cam2(:,time22);
102.    trimmed2test2(1,:) = trimmed2test2(1,:) -
   mean(trimmed2test2(1,:));
103.    trimmed2test2(2,:) = trimmed2test2(2,:) -
   mean(trimmed2test2(2,:));
104.    %% CAMERA 3 Test 2 cam3_2 DONEE
105.    load('cam3_2.mat')
106.    pointTracker = vision.PointTracker;
107.    numFrames = size(vidFrames3_2,4);
108.    points = [351,250]; %based on ginput(1)
109.    initialize(pointTracker, points, vidFrames3_2(:,:,:,1));
110.    test2cam3 = zeros(2,numFrames);
111.    for j = 1:numFrames
112.          X = vidFrames3_2(:,:,:,j);
113.          if (j == 196)
114.                setPoints(pointTracker, [315,248])
115.          end
116.          if (j == 260)
```

Priyanshi Rastogi
AMATH 482
2/18/2020

```
                setPoints(pointTracker, [374,258])
            end
            test2cam3(:,j) = step(pointTracker, X);
            %imshow(X);
            %hold on
            %plot(test2cam3(1,j), test2cam3(2,j), 'ro')
            %pause(0.1)
    end
    time32 = [56:211];
    trimmed3test2 = test2cam3(:,time32);
    trimmed3test2(1,:) = trimmed3test2(1,:) -
  mean(trimmed3test2(1,:));
    trimmed3test2(2,:) = trimmed3test2(2,:) -
  mean(trimmed3test2(2,:));
    %% CAMERA 1 Test 3 cam1_3 DONEE
    load('cam1_3.mat')
    pointTracker = vision.PointTracker;
    numFrames = size(vidFrames1_3,4);
    points = [339,313]; %based on ginput(1)
    initialize(pointTracker, points, vidFrames1_3(:,:,:,1));
    test3cam1 = zeros(2,numFrames);
    for j = 1:numFrames
          X = vidFrames1_3(:,:,:,j);
          test3cam1(:,j) = step(pointTracker, X);
                %imshow(X);
                %hold on
                %plot(test3cam1(1,j), test3cam1(2,j), 'ro')
                %pause(0.1)
    end
    time13 = [25:140];
    trimmed1test3 = test3cam1(:,time13);
    trimmed1test3(1,:) = trimmed1test3(1,:) -
  mean(trimmed1test3(1,:));
    trimmed1test3(2,:) = trimmed1test3(2,:) -
  mean(trimmed1test3(2,:));
    %% CAMERA 2 Test 3 cam2_3 DONEE
    load('cam2_3.mat')
    pointTracker = vision.PointTracker;
    numFrames = size(vidFrames2_3,4);
    points = [265,328]; %based on ginput(1)
    initialize(pointTracker, points, vidFrames2_3(:,:,:,1));
    test3cam2 = zeros(2,numFrames);
    for j = 1:numFrames
          X = vidFrames2_3(:,:,:,j);
          test3cam2(:,j) = step(pointTracker, X);
          imshow(X);
          hold on
          plot(test3cam2(1,j), test3cam2(2,j), 'ro')
          pause(0.1)
```

Priyanshi Rastogi
AMATH 482
2/18/2020

```
    end
    % time23 = [55:170];
    % trimmed2test3 = test3cam2(:,time23);
    % trimmed2test3(1,:) = trimmed2test3(1,:) -
  mean(trimmed2test3(1,:));
    % trimmed2test3(2,:) = trimmed2test3(2,:) -
  mean(trimmed2test3(2,:));
    %% CAMERA 3 Test 3 cam3_3 DONEE
    load('cam3_3.mat')
    pointTracker = vision.PointTracker;
    numFrames = size(vidFrames3_3,4);
    points = [369,236]; %based on ginput(1)
    initialize(pointTracker, points, vidFrames3_3(:,:,:,1));
    test3cam3 = zeros(2,numFrames);
    for j = 1:numFrames
          X = vidFrames3_3(:,:,:,j);
          test3cam3(:,j) = step(pointTracker, X);
          %     imshow(X);
          %     hold on
          %     plot(test3cam3(1,j), test3cam3(2,j), 'ro')
          %     pause(0.1)
    end
    time33 = [26:141];
    trimmed3test3 = test3cam3(:,time33);
    trimmed3test3(1,:) = trimmed3test3(1,:) -
  mean(trimmed3test3(1,:));
    trimmed3test3(2,:) = trimmed3test3(2,:) -
  mean(trimmed3test3(2,:));
    %% CAMERA 1 Test 4 cam1_4 DONE
    load('cam1_4.mat')
    pointTracker = vision.PointTracker;
    numFrames = size(vidFrames1_4,4);
    points = [400,268]; %based on ginput(1)
    initialize(pointTracker, points, vidFrames1_4(:,:,:,1));
    test4cam1 = zeros(2,numFrames);
    time = 1:numFrames;
    for j = 1:numFrames
          X = vidFrames1_4(:,:,:,j);
          if (j == 33)
                setPoints(pointTracker, [394,338])
          end
          if (j == 60)
                setPoints(pointTracker, [344,276])
          end
          if (j == 95)
                setPoints(pointTracker, [375,256])
          end
          if (j == 220)
                setPoints(pointTracker, [371,266])
```

Priyanshi Rastogi
AMATH 482
2/18/2020

```
207.          end
208.          if (j == 289)
209.                setPoints(pointTracker, [381,252])
210.          end
211.    test4cam1(:,j) = step(pointTracker, X);
212.    %     imshow(X);
213.    %     hold on
214.    %     plot(test4cam1(1,j), test4cam1(2,j), 'ro')
215.    %     pause(0.1)
216.    end
217.    time14 = [40:240];
218.    trimmed1test4 = test4cam1(:,time14);
219.    trimmed1test4(1,:) = trimmed1test4(1,:) -
   mean(trimmed1test4(1,:));
220.    trimmed1test4(2,:) = trimmed1test4(2,:) -
   mean(trimmed1test4(2,:));
221.    %% CAMERA 2 Test 4  cam2_4 DONE
222.    load('cam2_4.mat')
223.    pointTracker = vision.PointTracker;
224.    numFrames = size(vidFrames2_4,4);
225.    points = [247,260]; %based on ginput(1)
226.    initialize(pointTracker, points, vidFrames2_4(:,:,:,1));
227.    test4cam2 = zeros(2,numFrames);
228.    for j = 1:numFrames
229.          X = vidFrames2_4(:,:,:,j);
230.          if (j == 10)
231.                setPoints(pointTracker, [305,188])
232.          end
233.          if (j == 16)
234.                setPoints(pointTracker, [351,148])
235.          end
236.          if (j == 35)
237.                setPoints(pointTracker, [369,246])
238.          end
239.          if (j == 50)
240.          setPoints(pointTracker, [283,194])
241.          end
242.          if (j == 65)
243.                setPoints(pointTracker, [265,148])
244.          end
245.          if (j == 68)
246.                setPoints(pointTracker, [261,170])
247.          end
248.          if (j == 70)
249.                setPoints(pointTracker, [269,196])
250.          end
251.          if (j == 73)
252.                setPoints(pointTracker, [283,220])
253.          end
```

Priyanshi Rastogi
AMATH 482
2/18/2020

```
254.          if (j == 80)
255.                setPoints(pointTracker, [291,248])
256.          end
257.          if (j == 96)
258.                setPoints(pointTracker, [299,150])
259.          end
260.          if (j == 176)
261.                setPoints(pointTracker, [295,142])
262.          end
263.          if (j == 194)
264.                setPoints(pointTracker, [321,206])
265.          end
266.          if (j == 214)
267.                setPoints(pointTracker, [303,144])
268.          end
269.          if (j == 270)
270.                setPoints(pointTracker, [323,186])
271.          end
272.          if (j == 290)
273.                setPoints(pointTracker, [307,170])
274.          end
275.          if (j == 299)
276.                setPoints(pointTracker, [293,112])
277.          end
278.          if (j == 320)
279.          setPoints(pointTracker, [277,258])
280.    end
281.    test4cam2(:,j) = step(pointTracker, X);
282.    %   imshow(X);
283.    %   hold on
284.    %   plot(test4cam2(1,j), test4cam2(2,j), 'ro')
285.    %   pause(0.1)
286.    end
287.    time24 = [85:285];
288.    trimmed2test4 = test4cam2(:,time24);
289.    trimmed2test4(1,:) = trimmed2test4(1,:) -
   mean(trimmed2test4(1,:));
290.    trimmed2test4(2,:) = trimmed2test4(2,:) -
   mean(trimmed2test4(2,:));
291.    %% CAMERA 3 Test 4  cam3_4 DONE
292.    load('cam3_4.mat')
293.    pointTracker = vision.PointTracker;
294.    numFrames = size(vidFrames3_4,4);
295.    points = [361,230]; %based on ginput(1) after imshow for cam2_1
296.    initialize(pointTracker, points, vidFrames3_4(:,:,:,1));
297.    test4cam3 = zeros(2,numFrames);
298.    for j = 1:numFrames
299.          X = vidFrames3_4(:,:,:,j);
300.          if (j == 25)
```

```
301.                setPoints(pointTracker, [375,194])
302.          end
303.          if (j == 40)
304.                setPoints(pointTracker, [377,210])
305.          end
306.          if (j == 54)
307.                setPoints(pointTracker, [326,222])
308.          end
309.          if (j == 65)
310.                setPoints(pointTracker, [377,206])
311.          end
312.          if (j == 185)
313.                setPoints(pointTracker, [375,194])
314.          end
315.          if (j == 192)
316.                setPoints(pointTracker, [411,196])
317.          end
318.          if (j == 200)
319.                setPoints(pointTracker, [393,215])
320.          end
321.          if (j == 217)
322.                setPoints(pointTracker, [354,257])
323.          end
324.          if (j == 230)
325.                setPoints(pointTracker, [416,223])
326.          end
327.          if (j == 260)
328.                setPoints(pointTracker, [358,218])
329.          end
330.          if (j == 270)
331.                setPoints(pointTracker, [421,214])
332.          end
333.          if (j == 291)
334.                setPoints(pointTracker, [335,265])
335.          end
336.          test4cam3(:,j) = step(pointTracker, X);
337.          %imshow(X);
338.          %hold on
339.          %plot(test4cam3(1,j), test4cam3(2,j), 'ro')
340.          %pause(0.1)
341.    end
342.    time34 = [103:303];
343.    trimmed3test4 = test4cam3(:,time34);
344.    trimmed3test4(1,:) = trimmed3test4(1,:) -
   mean(trimmed3test4(1,:));
345.    trimmed3test4(2,:) = trimmed3test4(2,:) -
   mean(trimmed3test4(2,:));
346.    %% SVD Calculations
347.    %Test 1 Matrix:
```

Priyanshi Rastogi
AMATH 482
2/18/2020

```
348.     X1 = zeros(6,161);
349.     X1(1:2,:) = trimmed1test1(:,:);
350.     X1(3:4,:) = trimmed2test1(:,:);
351.     X1(5:6,:) = trimmed3test1(:,:);
352.     [U1,S1,V1] = svd(X1,'econ');
353.     %Test 2 Matrix:
354.     X2 = zeros(6,156);
355.     X2(1:2,:) = trimmed1test2(:,:);%SUBTRACT FROM THE MEAN OF EACH
   INDIVIDUAL ROW to avoid different shifts affect
356.     X2(3:4,:) = trimmed2test2(:,:);
357.     X2(5:6,:) = trimmed3test2(:,:);
358.     [U2,S2,V2] = svd(X2,'econ');
359.     %Test 3 Matrix:
360.     X3 = zeros(6,116);
361.     X3(1:2,:) = trimmed1test3(:,:);
362.     X3(3:4,:) = trimmed2test3(:,:);
363.     X3(5:6,:) = trimmed3test3(:,:);
364.     [U3,S3,V3] = svd(X3,'econ');
365.     %Test 4 Matrix:
366.     X4 = zeros(6,201);
367.     X4(1:2,:) = trimmed1test4(:,:);
368.     X4(3:4,:) = trimmed2test4(:,:);
369.     X4(5:6,:) = trimmed3test4(:,:);
370.     [U4,S4,V4] = svd(X4,'econ');
371.     %plotting the singular values:
372.     s1values = diag(S1).';
373.     s2values = diag(S2).';
374.     s3values = diag(S3).';
375.     s4values = diag(S4).';
376.     %Plotting the X values of all camera of test 1 over time:
377.     % plot(time11, test1cam1(1,time11), '-k')
378.     % hold on
379.     % plot(time21, test1cam2(1,time21), '-r')
380.     % hold on
381.     % plot(time31, test1cam3(1,time31), '-g')
382.     % title('X displacement over time (test 1)','FontSize', 16)
383.     % xlabel('Time (Trimmed)','FontSize', 16)
384.     % ylabel('X displacement','FontSize', 16)
385.     % legend('camera 1','camera 2','camera 3','FontSize', 14)

386.     %Plotting Y values of test 1 for all camera angles over time:
387.     % plot(time11, test1cam1(2,time11), '--k')
388.     % hold on
389.     % plot(time21, test1cam2(2,time21), '--r')
390.     % hold on
391.     % plot(time31, test1cam3(2,time31), '--g')
392.     % title('Y displacement over time (Test 1)','FontSize', 16)
393.     % xlabel('Time (Trimmed)','FontSize', 16)
394.     % ylabel('Y displacement','FontSize', 16)
```

```
% legend('cam 1', 'cam 2', 'cam 3','FontSize', 14)

%Plotting the X values of all camera of test 2 over time:
% plot(time12, test2cam1(1,time12), '-k')
% hold on
% plot(time22, test2cam2(1,time22), '-r')
% hold on
% plot(time32, test2cam3(1,time32), '-g')
% title('X displacement over time (Test 2)','FontSize', 16)
% xlabel('Time (Trimmed)','FontSize', 16)
% ylabel('X displacement','FontSize', 16)
% legend('camera 1','camera 2','camera 3','FontSize', 14)

%Plotting Y values of test 2 for all camera angles over time:
% plot(time12, test2cam1(2,time12), '--k')
% hold on
% plot(time22, test2cam2(2,time22), '--r')
% hold on
% plot(time32, test2cam3(2,time32), '--g')
% title('Y displacement over time (Test 2)','FontSize', 16)
% xlabel('Time (Trimmed)','FontSize', 16)
% ylabel('Y displacement','FontSize', 16)
% legend('cam 1', 'cam 2', 'cam 3','FontSize', 14)

%Plotting the X values of all camera of test 3 over time:
% plot(time13, test3cam1(1,time13), '-k')
% hold on
% plot(time23, test3cam2(1,time23), '-r')
% hold on
% plot(time33, test3cam3(1,time33), '-g')
% title('X displacement over time (Test 3)','FontSize', 16)
% xlabel('Time (Trimmed)','FontSize', 16)
% ylabel('X displacement','FontSize', 16)
% legend('camera 1','camera 2','camera 3','FontSize', 14)

%Plotting Y values of test 3 for all camera angles over time:
% plot(time13, test3cam1(2,time13), '--k')
% hold on
% plot(time23, test3cam2(2,time23), '--r')
% hold on
% plot(time33, test3cam3(2,time33), '--g')
% title('Y displacement over time (Test 3)','FontSize', 16)
% xlabel('Time (Trimmed)','FontSize', 16)
% ylabel('Y displacement','FontSize', 16)
% legend('cam 1', 'cam 2', 'cam 3','FontSize', 14)

%Plotting the X values of all camera of test 4 over time:
% plot(time14, test4cam1(1,time14), '-k')
% hold on
```

```
% plot(time24, test4cam2(1,time24), '-r')
% hold on
% plot(time34, test4cam3(1,time34), '-g')
% title('X displacement over time (Test 4)','FontSize', 16)
% xlabel('Time (Trimmed)','FontSize', 16)
% ylabel('X displacement','FontSize', 16)
% legend('camera 1','camera 2','camera 3','FontSize', 14)

%Plotting Y values of test 4 for all camera angles over time:
% plot(time14, test4cam1(2,time14), '--k')
% hold on
% plot(time24, test4cam2(2,time24), '--r')
% hold on
% plot(time34, test4cam3(2,time34), '--g')
% title('Y displacement over time (Test 4)','FontSize', 16)
% xlabel('Time (Trimmed)','FontSize', 16)
% ylabel('Y displacement','FontSize', 16)
% legend('cam 1', 'cam 2', 'cam 3','FontSize', 14)


%Plotting Energies:
% subplot(2,2,1)
% loglog([1:6],
(s1values.^2)/(s1values(1).^2+s1values(2).^2+s1values(3).^2+s1values(4)
.^2+s1values(5).^2+s1values(6).^2),'ro')
% title('Test 1 Energy Graph','FontSize', 16)
% xlabel('Mode (each singular value)','FontSize', 16)
% ylabel('Energy values','FontSize', 16)
% hold on
% subplot(2,2,2)
% loglog([1:6],
(s2values.^2)/(s2values(1).^2+s2values(2).^2+s2values(3).^2+s2values(4)
.^2+s2values(5).^2+s2values(6).^2),'ro')
% title('Test 2 Energy Graph','FontSize', 16)
% xlabel('Mode (each singular value)','FontSize', 16)
% ylabel('Energy values','FontSize', 16)
% hold on
% subplot(2,2,3)
% loglog([1:6],
(s3values.^2)/(s3values(1).^2+s3values(2).^2+s3values(3).^2+s3values(4)
.^2+s3values(5).^2+s3values(6).^2),'ro')
% title('Test 3 Energy Graph','FontSize', 16)
% xlabel('Mode (each singular value)','FontSize', 16)
% ylabel('Energy values','FontSize', 16)
% hold on
% subplot(2,2,4)
% loglog([1:6],
(s4values.^2)/(s4values(1).^2+s4values(2).^2+s4values(3).^2+s4values(4)
.^2+s4values(5).^2+s4values(6).^2),'ro')
% title('Test 4 Energy Graph','FontSize', 16)
```

Priyanshi Rastogi
AMATH 482
2/18/2020

```
478.    % xlabel('Mode (each singular value)','FontSize', 16)
479.    % ylabel('Energy values','FontSize', 16)
```