

# Principal Component Analysis on a Spring Mass System

Amanda Lee

February 21, 2020

## Abstract

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

## Introduction

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

## Background

### Singular Value Decomposition

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

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

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

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

## Algorithm Implementation

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

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

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

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

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

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

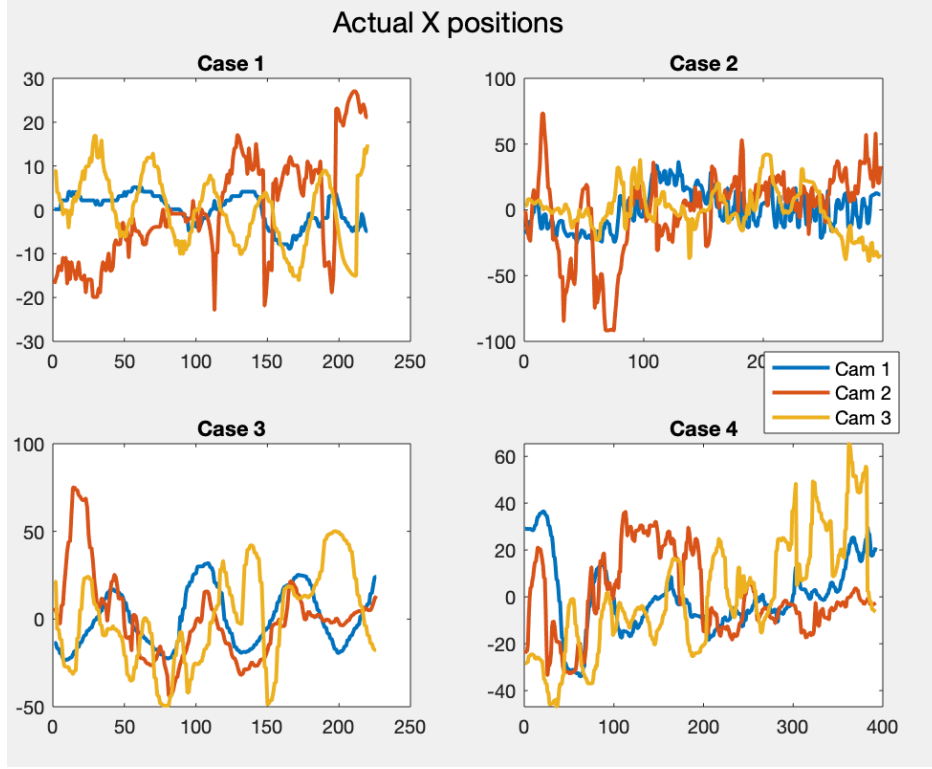


Figure 1: These are the X positions collected from the video data as described above with the means subtracted.

## Results

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

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

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

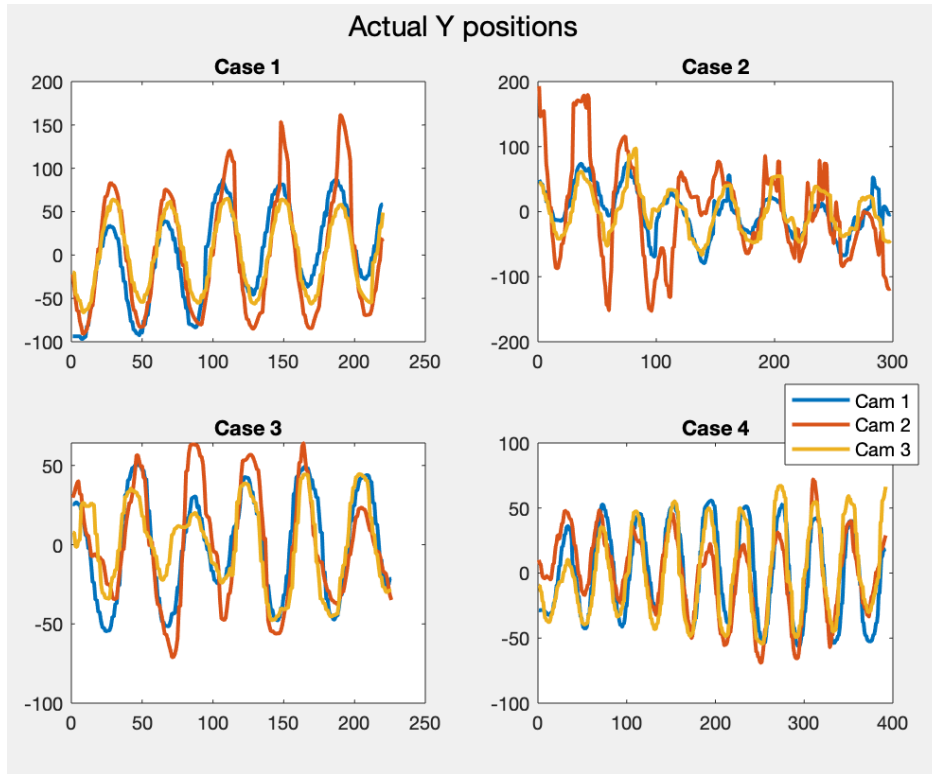


Figure 2: These are the Y positions collected from the video data as described above with the means subtracted.

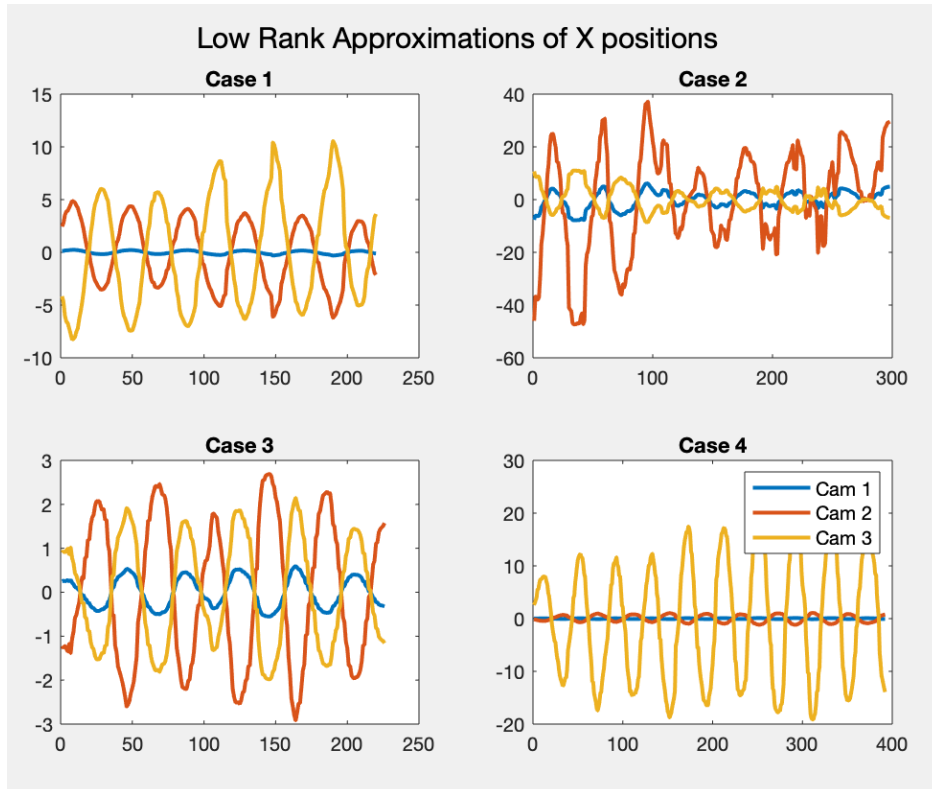


Figure 3: These are the rank 1 approximations of the X positions.

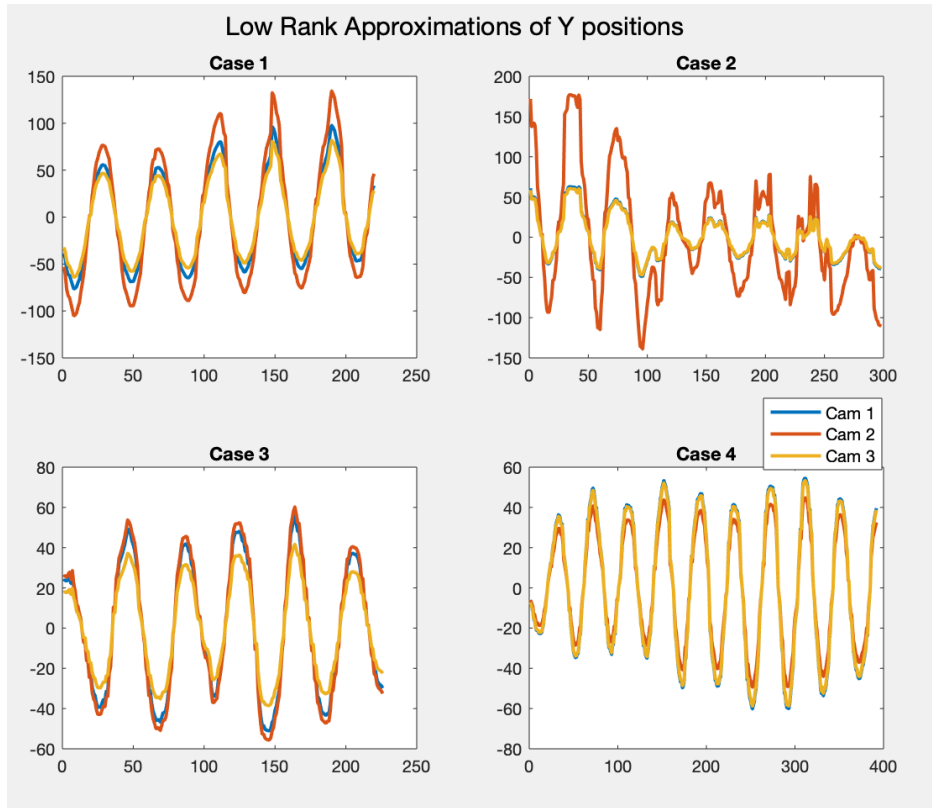


Figure 4: These are the rank 1 approximations of the X positions.

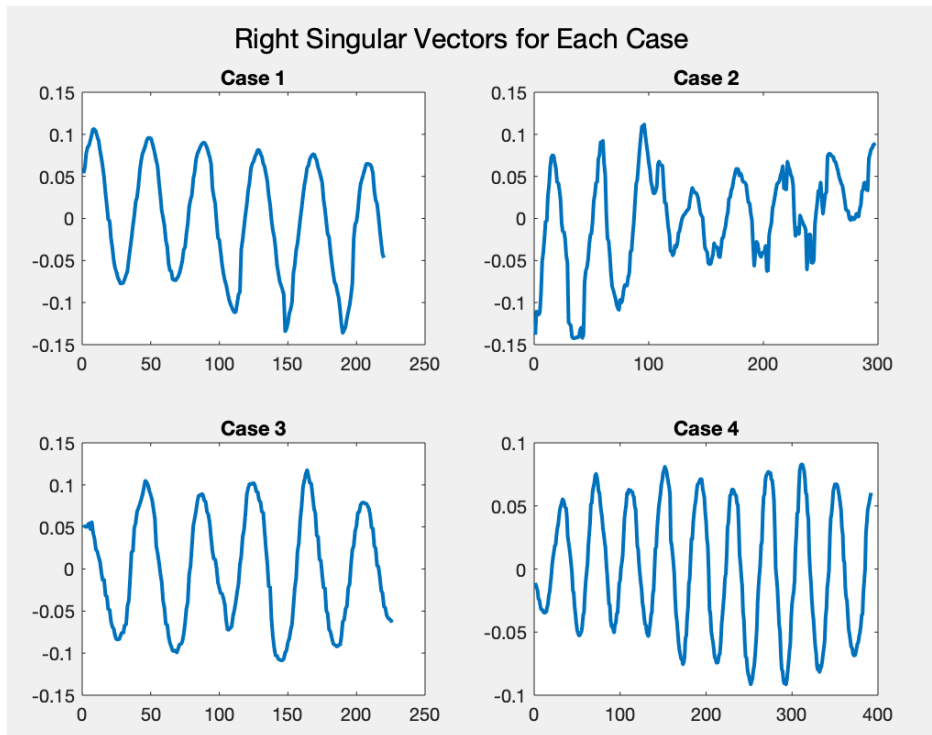


Figure 5: These are the first right singular vectors for each case.

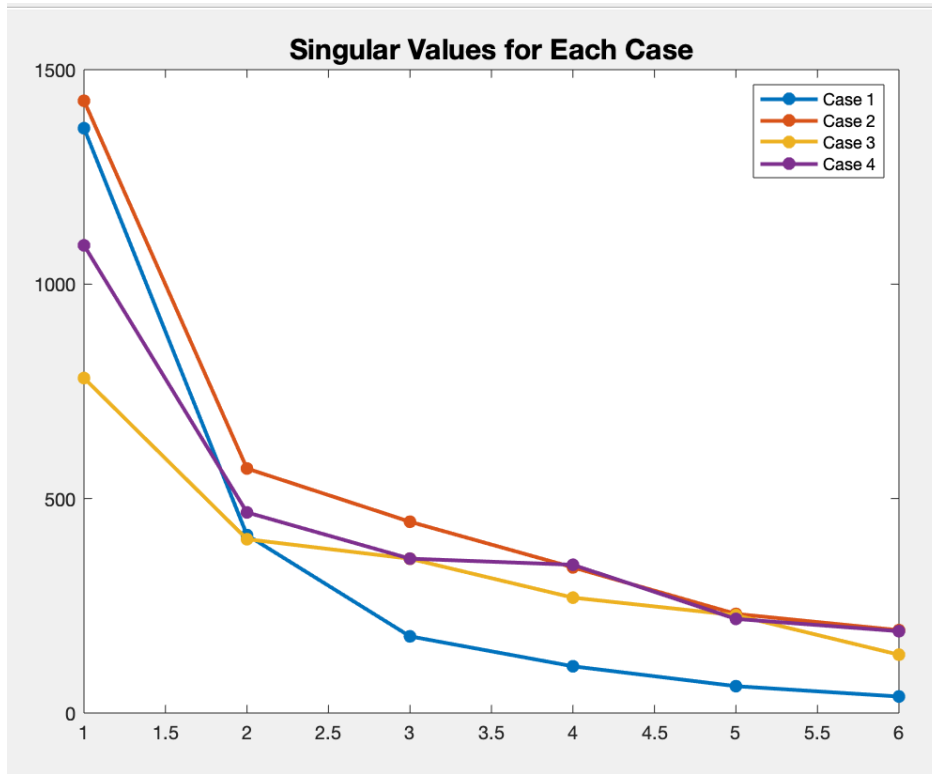


Figure 6: These are the singular values for each case.

## Conclusion

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

## Appendix A

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

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

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

## Appendix B

### getPos1.m

```
1 function [x_inds,y_inds] = getPos1(vid,startx,endx)
2 % Get position data for videos with clear a clear flashlight image
3 numFrames = size(vid,4);
4 % Turn video to greyscale and crop
5 gvid = zeros(480,endx-startx+1,numFrames);
6 for j = 1:numFrames
7     gvid(:, :, j) = rgb2gray(vid(:, startx:endx, :, j));
8 end
9 % Get position data
10 x_inds = zeros(numFrames,1);
11 y_inds = zeros(numFrames,1);
12 for j = 1:numFrames
13     [~,I] = max(gvid(:, :, j), [], 'all', 'linear');
14     [row, col] = ind2sub([480,endx-startx+1],I);
15     x_inds(j) = col;
16     y_inds(j) = row;
17 end
18 end
```

### getPos2.m

```
1 function [x_inds,y_inds] = getPos2(vid,startx,endx,starty,endy)
2 % Get position data for videos with clear a clear flashlight image and
3 % cropping in both dimensions
4 numFrames = size(vid,4);
5 % Turn video to greyscale and crop
6 gvid = zeros(endy-starty+1,endx-startx+1,numFrames);
7 for j = 1:numFrames
8     gvid(:, :, j) = rgb2gray(vid(starty:endy, startx:endx, :, j));
9 end
10 % Get position data
11 x_inds = zeros(numFrames,1);
12 y_inds = zeros(numFrames,1);
13 for j = 1:numFrames
14     [~,I] = max(gvid(:, :, j), [], 'all', 'linear');
15     [row, col] = ind2sub([endy-starty+1,endx-startx+1],I);
16     x_inds(j) = col;
17     y_inds(j) = row;
18 end
19 end
```

### getPos3.m

```
1 function [x_inds,y_inds] = getPos3(thresh,vid,startx,endx,starty,endy)
2 % Get position data for videos using averaging
3 numFrames = size(vid,4);
4 % Turn video to greyscale and crop
```

```

5 % Get average frame
6 avgFrame = zeros(endy-starty+1,endx-startx+1);
7 gvid = zeros(endy-starty+1,endx-startx+1,numFrames);
8 for j = 1:numFrames
9     gvid(:, :, j) = rgb2gray(vid(starty:endy, startx:endx, :, j));
10    avgFrame = avgFrame + im2double(gvid(:, :, j));
11 end
12 avgFrame = avgFrame/numFrames;
13
14 diff = zeros(size(gvid));
15
16 for j = 1:numFrames
17     frame1 = gvid(:, :, j);
18     diff(:, :, j) = frame1-avgFrame;
19 end
20 diff(diff<0) = 0;
21
22 % Get position data
23 x_inds = zeros(numFrames,1);
24 y_inds = zeros(numFrames,1);
25 for j = 1:numFrames
26     [Y,X] = find(diff(:, :, j)>thresh);
27     x_inds(j) = mean(X);
28     y_inds(j) = mean(Y);
29 end
30 end

```

## gettingCoordsScript.m

```

1 clear; clc; close all;
2 % Camera 1 Case 1
3 load('cam1_1.mat')
4 % implay(vidFrames1_1)
5 [xpos_1_1, ypos_1_1] = getPos1(vidFrames1_1, 243,445);
6 numFrames = size(vidFrames1_1,4);
7
8 % Plot position data to look for incongruities
9 frameNums = 1:numFrames;
10 figure(1)
11 plot(frameNums, ypos_1_1)
12 hold on
13 plot(frameNums, xpos_1_1)
14
15 % Remove points that are out of place
16 badFrames = [81, 88, 89, 90, 92, 93, 94, 95, 100, 101, 112, 113, 198, 199,
201, 202, 220, 221];
17
18 for k = length(badFrames):-1:1
19     badFrame = badFrames(k);
20     frameNums(badFrame) = [];
21     xpos_1_1(badFrame) = [];
22     ypos_1_1(badFrame) = [];
23 end
24

```



```

25 % Plot to check
26 figure(2)
27 plot(frameNums, ypos_1_1)
28 hold on
29 plot(frameNums, xpos_1_1)
30
31 % Use interpolation to get data for missing points
32 xinterp = interp1(frameNums, xpos_1_1, badFrames, 'pchip');
33 yinterp = interp1(frameNums, ypos_1_1, badFrames, 'pchip');
34
35 % Add interp points back in
36 xpos_1_1 = xpos_1_1';
37 ypos_1_1 = ypos_1_1';
38 for k = 1:length(badFrames)
39     badFrame = badFrames(k);
40     xpos_1_1 = [xpos_1_1(1:badFrame-1), xinterp(k), xpos_1_1(badFrame:end)]';
41     ypos_1_1 = [ypos_1_1(1:badFrame-1), yinterp(k), ypos_1_1(badFrame:end)]';
42 end
43 frameNums = 1:numFrames;
44
45 % Plot to check
46 figure(3)
47 plot(frameNums, ypos_1_1)
48 hold on
49 plot(frameNums, xpos_1_1)
50 xpos_1_1 = xpos_1_1';
51 ypos_1_1 = ypos_1_1';
52 %% Camera 2 Case 1
53 close all
54 load('cam2_1.mat')
55 %implay(vidFrames2_1)
56 [xpos_2_1, ypos_2_1] = getPos1(vidFrames2_1, 240, 360);
57 numFrames = size(vidFrames2_1, 4);
58
59 % Plot position data to look for incongruities
60 frameNums = 1:numFrames;
61 figure(1)
62 plot(frameNums, ypos_2_1)
63 hold on
64 plot(frameNums, xpos_2_1)
65
66 % Remove points that are out of place
67 badFrames =
    [5,6,7,8,40,41,42,118,119,120,121,122,135,136,137,138,139,140,141,142,145,146,158,159]
68
69 for k = length(badFrames):-1:1
70     badFrame = badFrames(k);
71     frameNums(badFrame) = [];
72     xpos_2_1(badFrame) = [];
73     ypos_2_1(badFrame) = [];
74 end
75 % Plot to check
76 figure(2)
77 plot(frameNums, ypos_2_1)

```

```

78 hold on
79 plot(frameNums, xpos_2_1)
80
81 % Use interpolation to get data for missing points
82 xinterp = interp1(frameNums, xpos_2_1, badFrames, 'pchip');
83 yinterp = interp1(frameNums, ypos_2_1, badFrames, 'pchip');
84
85 % Add interp points back in
86 xpos_2_1 = xpos_2_1';
87 ypos_2_1 = ypos_2_1';
88 for k = 1:length(badFrames)
89     badFrame = badFrames(k);
90     xpos_2_1 = [xpos_2_1(1:badFrame-1), xinterp(k), xpos_2_1(badFrame:end)];
91     ypos_2_1 = [ypos_2_1(1:badFrame-1), yinterp(k), ypos_2_1(badFrame:end)];
92 end
93 xpos_2_1 = xpos_2_1';
94 ypos_2_1 = ypos_2_1';
95 frameNums = 1:numFrames;
96 % Plot to check
97 figure(3)
98 plot(frameNums, ypos_2_1)
99 hold on
100 plot(frameNums, xpos_2_1)
101 %% Camera 3 Case 1
102 close all
103 load('cam3_1.mat')
104 vidFrames3_1 = permute(vidFrames3_1,[2,1,3,4]);
105 %implay(vidFrames3_1)
106
107 [xpos_3_1, ypos_3_1] = getPos2(vidFrames3_1, 215,373,273,455);
108 numFrames = size(vidFrames3_1,4);
109
110 % Plot position data to look for incongruities
111 frameNums = 1:numFrames;
112 figure(1)
113 plot(frameNums, ypos_3_1)
114 hold on
115 plot(frameNums, xpos_3_1)
116
117 % Remove points that are out of place
118 badFrames = [182,197,198,199,200,201,202,203,205,206,207,208,209,210,211];
119
120 % Cut out last area with bad data
121 numFrames = 220;
122 frameNums(numFrames+1:end) = [];
123 xpos_3_1(numFrames+1:end) = [];
124 ypos_3_1(numFrames+1:end) = [];
125
126 for k = length(badFrames):-1:1
127     badFrame = badFrames(k);
128     frameNums(badFrame) = [];
129     xpos_3_1(badFrame) = [];
130     ypos_3_1(badFrame) = [];
131 end
132 % Plot to check

```

```

133 figure(2)
134 plot(frameNums, ypos_3_1)
135 hold on
136 plot(frameNums, xpos_3_1)
137
138 % Use interpolation to get data for missing points
139 xinterp = interp1(frameNums, xpos_3_1, badFrames, 'pchip');
140 yinterp = interp1(frameNums, ypos_3_1, badFrames, 'pchip');
141
142 % Add interp points back in
143 xpos_3_1 = xpos_3_1';
144 ypos_3_1 = ypos_3_1';
145 for k = 1:length(badFrames)
146     badFrame = badFrames(k);
147     xpos_3_1 = [xpos_3_1(1:badFrame-1), xinterp(k), xpos_3_1(badFrame:end)];
148     ypos_3_1 = [ypos_3_1(1:badFrame-1), yinterp(k), ypos_3_1(badFrame:end)];
149 end
150 frameNums = 1:numFrames;
151 xpos_3_1 = xpos_3_1';
152 ypos_3_1 = ypos_3_1';
153 % Plot to check
154 figure(3)
155 plot(frameNums, ypos_3_1)
156 hold on
157 plot(frameNums, xpos_3_1)
158
159 %% Camera 1 Case 2
160 close all
161 load('cam1_2.mat')
162 %implay(vidFrames1_2)
163 [xpos_1_2, ypos_1_2] = getPos2(vidFrames1_2, 243,445,212,387);
164 numFrames = size(vidFrames1_2,4);
165
166 % Plot position data to look for incongruities
167 frameNums = 1:numFrames;
168 figure(1)
169 plot(frameNums, ypos_1_2)
170 hold on
171 plot(frameNums, xpos_1_2)
172
173 % Remove points that are out of place
174 badFrames =
    [11,28,29,40,41,45,50,51,52,63,93,94,120,121,122,123,139,146,152,153,159,160,166,167];
175
176 for k = length(badFrames):-1:1
177     badFrame = badFrames(k);
178     frameNums(badFrame) = [];
179     xpos_1_2(badFrame) = [];
180     ypos_1_2(badFrame) = [];
181 end
182
183 % Plot to check
184 figure(2)
185 plot(frameNums, ypos_1_2)

```

```

186 hold on
187 plot(frameNums, xpos_1_2)
188
189 % Use interpolation to get data for missing points
190 xinterp = interp1(frameNums, xpos_1_2, badFrames, 'pchip');
191 yinterp = interp1(frameNums, ypos_1_2, badFrames, 'pchip');
192
193 % Add interp points back in
194 xpos_1_2 = xpos_1_2';
195 ypos_1_2 = ypos_1_2';
196 for k = 1:length(badFrames)
197     badFrame = badFrames(k);
198     xpos_1_2 = [xpos_1_2(1:badFrame-1), xinterp(k), xpos_1_2(badFrame:end)];
199     ypos_1_2 = [ypos_1_2(1:badFrame-1), yinterp(k), ypos_1_2(badFrame:end)];
200 end
201 frameNums = 1:numFrames;
202 xpos_1_2 = xpos_1_2';
203 ypos_1_2 = ypos_1_2';
204
205 % Plot to check
206 figure(3)
207 plot(frameNums, ypos_1_2)
208 hold on
209 plot(frameNums, xpos_1_2)
210
211 %% Camera 2 Case 2
212 close all
213 load('cam2_2.mat')
214 %implay(vidFrames2_2)
215 [xpos_2_2, ypos_2_2] = getPos1(vidFrames2_2, 182, 457);
216 numFrames = size(vidFrames2_2, 4);
217
218 % Plot position data to look for incongruities
219 frameNums = 1:numFrames;
220 figure(1)
221 plot(frameNums, ypos_2_2)
222 hold on
223 plot(frameNums, xpos_2_2)
224
225 % Remove points that are out of place
226 badFrames =
    [5, 7, 28, 37, 69, 67, 68, 70, 71, 72, 73, 74, 78, 79, 126, 132, 133, 134, 145, 150, 151, 155, 156, 161, 162]
227
228 for k = length(badFrames):-1:1
229     badFrame = badFrames(k);
230     frameNums(badFrame) = [];
231     xpos_2_2(badFrame) = [];
232     ypos_2_2(badFrame) = [];
233 end
234
235 % Plot to check
236 figure(2)
237 plot(frameNums, ypos_2_2)
238 hold on

```

```

239 plot(frameNums, xpos_2_2)
240
241 % Use interpolation to get data for missing points
242 xinterp = interp1(frameNums, xpos_2_2, badFrames, 'pchip');
243 yinterp = interp1(frameNums, ypos_2_2, badFrames, 'pchip');
244
245 % Add interp points back in
246 xpos_2_2 = xpos_2_2';
247 ypos_2_2 = ypos_2_2';
248 for k = 1:length(badFrames)
249     badFrame = badFrames(k);
250     xpos_2_2 = [xpos_2_2(1:badFrame-1), xinterp(k), xpos_2_2(badFrame:end)];
251     ypos_2_2 = [ypos_2_2(1:badFrame-1), yinterp(k), ypos_2_2(badFrame:end)];
252 end
253 frameNums = 1:numFrames;
254 xpos_2_2 = xpos_2_2';
255 ypos_2_2 = ypos_2_2';
256
257 % Plot to check
258 figure(3)
259 plot(frameNums, ypos_2_2)
260 hold on
261 plot(frameNums, xpos_2_2)
262 %% Camera 3 Case 2
263 close all
264 load('cam3_2.mat')
265 vidFrames3_2 = permute(vidFrames3_2, [2, 1, 3, 4]);
266 %implay(vidFrames3_2)
267 [xpos_3_2, ypos_3_2] = getPos2(vidFrames3_2, 184, 323, 275, 474);
268 numFrames = size(vidFrames3_2, 4);
269 % Plot position data to look for incongruities
270 frameNums = 1:numFrames;
271 figure(1)
272 plot(frameNums, ypos_3_2)
273 hold on
274 plot(frameNums, xpos_3_2)
275
276 % Remove points that are out of place
277 badFrames =
    [42, 44, 45, 58, 59, 71, 72, 82, 90, 96, 100, 101, 107, 111, 113, 116, 120, 122, 123, 126, 127, 134, 135, 140, 141, 142, 143, 144, 145, 146, 147, 148, 149, 150, 151, 152, 153, 154, 155, 156, 157, 158, 159, 160, 161, 162, 163, 164, 165, 166, 167, 168, 169, 170, 171, 172, 173, 174, 175, 176, 177, 178, 179, 180, 181, 182, 183, 184, 185, 186, 187, 188, 189, 190, 191, 192, 193, 194, 195, 196, 197, 198, 199, 200, 201, 202, 203, 204, 205, 206, 207, 208, 209, 210, 211, 212, 213, 214, 215, 216, 217, 218, 219, 220, 221, 222, 223, 224, 225, 226, 227, 228, 229, 230, 231, 232, 233, 234, 235, 236, 237, 238, 239, 240, 241, 242, 243, 244, 245, 246, 247, 248, 249, 250, 251, 252, 253, 254, 255, 256, 257, 258, 259, 260, 261, 262, 263, 264, 265, 266, 267, 268, 269, 270, 271, 272, 273, 274, 275, 276, 277, 278, 279, 280, 281, 282, 283, 284, 285, 286, 287, 288, 289, 290, 291, 292, 293, 294, 295, 296, 297, 298, 299, 300, 301, 302, 303, 304, 305, 306, 307, 308, 309, 310, 311, 312, 313, 314, 315, 316, 317, 318, 319, 320, 321, 322, 323, 324, 325, 326, 327, 328, 329, 330, 331, 332, 333, 334, 335, 336, 337, 338, 339, 340, 341, 342, 343, 344, 345, 346, 347, 348, 349, 350, 351, 352, 353, 354, 355, 356, 357, 358, 359, 360, 361, 362, 363, 364, 365, 366, 367, 368, 369, 370, 371, 372, 373, 374, 375, 376, 377, 378, 379, 380, 381, 382, 383, 384, 385, 386, 387, 388, 389, 390, 391, 392, 393, 394, 395, 396, 397, 398, 399, 400, 401, 402, 403, 404, 405, 406, 407, 408, 409, 410, 411, 412, 413, 414, 415, 416, 417, 418, 419, 420, 421, 422, 423, 424, 425, 426, 427, 428, 429, 430, 431, 432, 433, 434, 435, 436, 437, 438, 439, 440, 441, 442, 443, 444, 445, 446, 447, 448, 449, 450, 451, 452, 453, 454, 455, 456, 457, 458, 459, 460, 461, 462, 463, 464, 465, 466, 467, 468, 469, 470, 471, 472, 473, 474, 475, 476, 477, 478, 479, 480, 481, 482, 483, 484, 485, 486, 487, 488, 489, 490, 491, 492, 493, 494, 495, 496, 497, 498, 499, 500, 501, 502, 503, 504, 505, 506, 507, 508, 509, 510, 511, 512, 513, 514, 515, 516, 517, 518, 519, 520, 521, 522, 523, 524, 525, 526, 527, 528, 529, 530, 531, 532, 533, 534, 535, 536, 537, 538, 539, 540, 541, 542, 543, 544, 545, 546, 547, 548, 549, 550, 551, 552, 553, 554, 555, 556, 557, 558, 559, 560, 561, 562, 563, 564, 565, 566, 567, 568, 569, 570, 571, 572, 573, 574, 575, 576, 577, 578, 579, 580, 581, 582, 583, 584, 585, 586, 587, 588, 589, 590, 591, 592, 593, 594, 595, 596, 597, 598, 599, 600, 601, 602, 603, 604, 605, 606, 607, 608, 609, 610, 611, 612, 613, 614, 615, 616, 617, 618, 619, 620, 621, 622, 623, 624, 625, 626, 627, 628, 629, 630, 631, 632, 633, 634, 635, 636, 637, 638, 639, 640, 641, 642, 643, 644, 645, 646, 647, 648, 649, 650, 651, 652, 653, 654, 655, 656, 657, 658, 659, 660, 661, 662, 663, 664, 665, 666, 667, 668, 669, 670, 671, 672, 673, 674, 675, 676, 677, 678, 679, 680, 681, 682, 683, 684, 685, 686, 687, 688, 689, 690, 691, 692, 693, 694, 695, 696, 697, 698, 699, 700, 701, 702, 703, 704, 705, 706, 707, 708, 709, 710, 711, 712, 713, 714, 715, 716, 717, 718, 719, 720, 721, 722, 723, 724, 725, 726, 727, 728, 729, 730, 731, 732, 733, 734, 735, 736, 737, 738, 739, 740, 741, 742, 743, 744, 745, 746, 747, 748, 749, 750, 751, 752, 753, 754, 755, 756, 757, 758, 759, 760, 761, 762, 763, 764, 765, 766, 767, 768, 769, 770, 771, 772, 773, 774, 775, 776, 777, 778, 779, 780, 781, 782, 783, 784, 785, 786, 787, 788, 789, 790, 791, 792, 793, 794, 795, 796, 797, 798, 799, 800, 801, 802, 803, 804, 805, 806, 807, 808, 809, 810, 811, 812, 813, 814, 815, 816, 817, 818, 819, 820, 821, 822, 823, 824, 825, 826, 827, 828, 829, 830, 831, 832, 833, 834, 835, 836, 837, 838, 839, 840, 841, 842, 843, 844, 845, 846, 847, 848, 849, 850, 851, 852, 853, 854, 855, 856, 857, 858, 859, 860, 861, 862, 863, 864, 865, 866, 867, 868, 869, 870, 871, 872, 873, 874, 875, 876, 877, 878, 879, 880, 881, 882, 883, 884, 885, 886, 887, 888, 889, 890, 891, 892, 893, 894, 895, 896, 897, 898, 899, 900, 901, 902, 903, 904, 905, 906, 907, 908, 909, 910, 911, 912, 913, 914, 915, 916, 917, 918, 919, 920, 921, 922, 923, 924, 925, 926, 927, 928, 929, 930, 931, 932, 933, 934, 935, 936, 937, 938, 939, 940, 941, 942, 943, 944, 945, 946, 947, 948, 949, 950, 951, 952, 953, 954, 955, 956, 957, 958, 959, 960, 961, 962, 963, 964, 965, 966, 967, 968, 969, 970, 971, 972, 973, 974, 975, 976, 977, 978, 979, 980, 981, 982, 983, 984, 985, 986, 987, 988, 989, 990, 991, 992, 993, 994, 995, 996, 997, 998, 999, 1000];
278 badFrames = sort(badFrames);
279 for k = length(badFrames):-1:1
280     badFrame = badFrames(k);
281     frameNums(badFrame) = [];
282     xpos_3_2(badFrame) = [];
283     ypos_3_2(badFrame) = [];
284 end
285
286 % Plot to check
287 figure(2)
288 plot(frameNums, ypos_3_2)
289 hold on
290 plot(frameNums, xpos_3_2)
291

```

```

292 % Use interpolation to get data for missing points
293 xinterp = interp1(frameNums, xpos_3_2, badFrames, 'pchip');
294 yinterp = interp1(frameNums, ypos_3_2, badFrames, 'pchip');
295
296 % Add interp points back in
297 xpos_3_2 = xpos_3_2';
298 ypos_3_2 = ypos_3_2';
299 for k = 1:length(badFrames)
300     badFrame = badFrames(k);
301     xpos_3_2 = [xpos_3_2(1:badFrame-1), xinterp(k), xpos_3_2(badFrame:end)];
302     ypos_3_2 = [ypos_3_2(1:badFrame-1), yinterp(k), ypos_3_2(badFrame:end)];
303 end
304 frameNums = 1:numFrames;
305 xpos_3_2 = xpos_3_2';
306 ypos_3_2 = ypos_3_2';
307
308 % Plot to check
309 figure(3)
310 plot(frameNums, ypos_3_2)
311 hold on
312 plot(frameNums, xpos_3_2)
313 %% Camera 1 Case 3
314 close all
315 load('cam1_3.mat')
316 implay(vidFrames1_3)
317 [xpos_1_3, ypos_1_3] = getPos3(80, vidFrames1_3, 238, 404, 223, 421);
318 numFrames = size(vidFrames1_3, 4);
319
320 % Plot position data to look for incongruities
321 frameNums = 1:numFrames;
322 figure(1)
323 plot(frameNums, ypos_1_3)
324 hold on
325 plot(frameNums, xpos_1_3)
326
327 %% Camera 2 Case 3
328 close all
329 load('cam2_3.mat')
330 %implay(vidFrames2_3)
331
332 [xpos_2_3, ypos_2_3] = getPos3(90, vidFrames2_3, 190, 472, 160, 412);
333 numFrames = size(vidFrames2_3, 4);
334
335 % Plot position data to look for incongruities
336 frameNums = 1:numFrames;
337 figure(1)
338 plot(frameNums, ypos_2_3)
339 hold on
340 plot(frameNums, xpos_2_3)
341
342 % Remove points that are out of place
343 badFrames = [2, 3, 4, 5, 41, 42, 43, 49, 163, 164, 199];
344 for k = length(badFrames):-1:1
345     badFrame = badFrames(k);
346     frameNums(badFrame) = [];

```

```

347     xpos_2_3(badFrame) = [];
348     ypos_2_3(badFrame) = [];
349 end
350
351 % Plot to check
352 figure(2)
353 plot(frameNums, ypos_2_3)
354 hold on
355 plot(frameNums, xpos_2_3)
356
357 % Use interpolation to get data for missing points
358 xinterp = interp1(frameNums, xpos_2_3, badFrames, 'pchip');
359 yinterp = interp1(frameNums, ypos_2_3, badFrames, 'pchip');
360
361 % Add interp points back in
362 xpos_2_3 = xpos_2_3';
363 ypos_2_3 = ypos_2_3';
364 for k = 1:length(badFrames)
365     badFrame = badFrames(k);
366     xpos_2_3 = [xpos_2_3(1:badFrame-1), xinterp(k), xpos_2_3(badFrame:end)];
367     ypos_2_3 = [ypos_2_3(1:badFrame-1), yinterp(k), ypos_2_3(badFrame:end)];
368 end
369 frameNums = 1:numFrames;
370 xpos_2_3 = xpos_2_3';
371 ypos_2_3 = ypos_2_3';
372
373 % Plot to check
374 figure(3)
375 plot(frameNums, ypos_2_3)
376 hold on
377 plot(frameNums, xpos_2_3)
378 %% Camera 3 Case 3
379 close all
380 load('cam3_3.mat')
381 vidFrames3_3 = permute(vidFrames3_3,[2,1,3,4]);
382 %implay(vidFrames3_3)
383 [xpos_3_3, ypos_3_3] = getPos3(80, vidFrames3_3, 146, 267, 160, 487);
384 % [xpos_3_3, ypos_3_3] = getPos2(vidFrames3_3, 146, 267, 160, 487);
385 numFrames = size(vidFrames3_3, 4);
386
387 % Plot position data to look for incongruities
388 frameNums = 1:numFrames;
389 figure(1)
390 plot(frameNums, ypos_3_3)
391 hold on
392 plot(frameNums, xpos_3_3)
393
394 %% Camera 1 Case 4
395 close all
396 load('cam1_4.mat')
397 %implay(vidFrames1_4)
398 [xpos_1_4, ypos_1_4] = getPos3(80, vidFrames1_4, 284, 496, 208, 421);
399 numFrames = size(vidFrames1_4, 4);
400
401 % Plot position data to look for incongruities

```

```

402 frameNums = 1:numFrames;
403 figure(1)
404 plot(frameNums, ypos_1_4)
405 hold on
406 plot(frameNums, xpos_1_4)
407
408 % Remove points that are out of place
409 badFrames = [8,111,112,113,114,115,190,191,192,193,194,236,237,238];
410
411 for k = length(badFrames):-1:1
412     badFrame = badFrames(k);
413     frameNums(badFrame) = [];
414     xpos_1_4(badFrame) = [];
415     ypos_1_4(badFrame) = [];
416 end
417
418 % Plot to check
419 figure(2)
420 plot(frameNums, ypos_1_4)
421 hold on
422 plot(frameNums, xpos_1_4)
423
424 % Use interpolation to get data for missing points
425 xinterp = interp1(frameNums, xpos_1_4, badFrames, 'pchip');
426 yinterp = interp1(frameNums, ypos_1_4, badFrames, 'pchip');
427
428 % Add interp points back in
429 xpos_1_4 = xpos_1_4';
430 ypos_1_4 = ypos_1_4';
431 for k = 1:length(badFrames)
432     badFrame = badFrames(k);
433     xpos_1_4 = [xpos_1_4(1:badFrame-1), xinterp(k), xpos_1_4(badFrame:end)];
434     ypos_1_4 = [ypos_1_4(1:badFrame-1), yinterp(k), ypos_1_4(badFrame:end)];
435 end
436 frameNums = 1:numFrames;
437 xpos_1_4 = xpos_1_4';
438 ypos_1_4 = ypos_1_4';
439
440 % Plot to check
441 figure(3)
442 plot(frameNums, ypos_1_4)
443 hold on
444 plot(frameNums, xpos_1_4)
445
446 %% Camera 2 Case 4
447 close all
448 load('cam2_4.mat')
449 %implay(vidFrames2_4)
450 [xpos_2_4, ypos_2_4] = getPos3(80, vidFrames2_4, 153, 443, 86, 371);
451 numFrames = size(vidFrames2_4, 4);
452
453 % Plot position data to look for incongruities
454 frameNums = 1:numFrames;
455 figure(1)
456 plot(frameNums, ypos_2_4)

```



```

457 hold on
458 plot(frameNums, xpos_2_4)
459
460 % Remove points that are out of place
461 badFrames = [34,43,44,59,88,138,193,198,397,398,399,400,401];
462 for k = length(badFrames):-1:1
463     badFrame = badFrames(k);
464     frameNums(badFrame) = [];
465     xpos_2_4(badFrame) = [];
466     ypos_2_4(badFrame) = [];
467 end
468
469 % Plot to check
470 figure(2)
471 plot(frameNums, ypos_2_4)
472 hold on
473 plot(frameNums, xpos_2_4)
474
475 % Use interpolation to get data for missing points
476 xinterp = interp1(frameNums, xpos_2_4, badFrames, 'pchip');
477 yinterp = interp1(frameNums, ypos_2_4, badFrames, 'pchip');
478
479 % Add interp points back in
480 xpos_2_4 = xpos_2_4';
481 ypos_2_4 = ypos_2_4';
482 for k = 1:length(badFrames)
483     badFrame = badFrames(k);
484     xpos_2_4 = [xpos_2_4(1:badFrame-1), xinterp(k), xpos_2_4(badFrame:end)];
485     ypos_2_4 = [ypos_2_4(1:badFrame-1), yinterp(k), ypos_2_4(badFrame:end)];
486 end
487 frameNums = 1:numFrames;
488 xpos_2_4 = xpos_2_4';
489 ypos_2_4 = ypos_2_4';
490
491 % Plot to check
492 figure(3)
493 plot(frameNums, ypos_2_4)
494 hold on
495 plot(frameNums, xpos_2_4)
496 %% Camera 3 Case 4
497 close all
498 load('cam3_4.mat')
499 vidFrames3_4 = permute(vidFrames3_4,[2,1,3,4]);
500 %implay(vidFrames3_4)
501
502 [xpos_3_4, ypos_3_4] = getPos3(65, vidFrames3_4, 117, 297, 298, 529);
503 numFrames = size(vidFrames3_4, 4);
504
505 % Plot position data to look for incongruities
506 frameNums = 1:numFrames;
507 figure(1)
508 plot(frameNums, ypos_3_4)
509 hold on
510 plot(frameNums, xpos_3_4)
511

```

```

512 % Remove points that are out of place
513 badFrames =
    [38,39,103,104,105,160,163,165,167,201,202,231,232,233,280,302,346,347,348,349,390,3
514 for k = length(badFrames):-1:1
515     badFrame = badFrames(k);
516     frameNums(badFrame) = [];
517     xpos_3_4(badFrame) = [];
518     ypos_3_4(badFrame) = [];
519 end
520
521 % Plot to check
522 figure(2)
523 plot(frameNums, ypos_3_4)
524 hold on
525 plot(frameNums, xpos_3_4)
526
527 % Use interpolation to get data for missing points
528 xinterp = interp1(frameNums, xpos_3_4, badFrames, 'pchip');
529 yinterp = interp1(frameNums, ypos_3_4, badFrames, 'pchip');
530
531 % Add interp points back in
532 xpos_3_4 = xpos_3_4';
533 ypos_3_4 = ypos_3_4';
534 for k = 1:length(badFrames)
535     badFrame = badFrames(k);
536     xpos_3_4 = [xpos_3_4(1:badFrame-1), xinterp(k), xpos_3_4(badFrame:end)];
537     ypos_3_4 = [ypos_3_4(1:badFrame-1), yinterp(k), ypos_3_4(badFrame:end)];
538 end
539 frameNums = 1:numFrames;
540 xpos_3_4 = xpos_3_4';
541 ypos_3_4 = ypos_3_4';
542
543 % Plot to check
544 figure(3)
545 plot(frameNums, ypos_3_4)
546 hold on
547 plot(frameNums, xpos_3_4)
548 %% Save position data
549 close all
550 save('posData.mat', '-regexp', '^xpos', '^ypos')

```

## alignData.m

```

1 clear; clc; close all;
2 load('posData.mat')
3
4 %%
5 % Plot first case
6 figure(1)
7 plot(ypos_1_1)
8 figure(2)
9 plot(ypos_2_1)
10 figure(3)

```

```

11 plot(ypos_3_1)
12 %%
13 close all;
14 % Align data
15 xpos_1_1(1:3) = [];
16 ypos_1_1(1:3) = [];
17 xpos_2_1(1:51) = [];
18 ypos_2_1(1:51) = [];
19 %xpos_3_1(1:10) = [];
20 %ypos_3_1(1:10) = [];
21
22 sizes = [length(xpos_1_1),length(xpos_2_1),length(xpos_3_1)];
23 minSize = min(sizes);
24 xpos_1_1(minSize+1:end) = [];
25 ypos_1_1(minSize+1:end) = [];
26 xpos_2_1(minSize+1:end) = [];
27 ypos_2_1(minSize+1:end) = [];
28 xpos_3_1(minSize+1:end) = [];
29 ypos_3_1(minSize+1:end) = [];
30
31 % xpos_1_1 = xpos_1_1 - mean(xpos_1_1);
32 % ypos_1_1 = ypos_1_1 - mean(ypos_1_1);
33 % ypos_2_1 = ypos_2_1 - mean(ypos_2_1);
34 % ypos_3_1 = ypos_3_1 - mean(ypos_3_1);
35
36 figure(1)
37 plot(ypos_1_1)
38 hold on
39 plot(ypos_2_1)
40 plot(ypos_3_1)
41 legend('cam 1', 'cam 2', 'cam3')
42 %%
43 close all;
44 % Plot second case
45 figure(1)
46 plot(ypos_1_2)
47 figure(2)
48 plot(ypos_2_2)
49 figure(3)
50 plot(ypos_3_2)
51
52 %%
53 close all;
54 % Align data
55 xpos_1_2(1:16) = [];
56 ypos_1_2(1:16) = [];
57 xpos_2_2(1:1) = [];
58 ypos_2_2(1:1) = [];
59 xpos_3_2(1:20) = [];
60 ypos_3_2(1:20) = [];
61
62 figure(1)
63 plot(ypos_1_2)
64 hold on
65 plot(ypos_2_2)

```

```

66 plot(ypos_3_2)
67
68 sizes = [length(xpos_1_2),length(xpos_2_2),length(xpos_3_2)];
69 minSize = min(sizes);
70 xpos_1_2(minSize+1:end) = [];
71 ypos_1_2(minSize+1:end) = [];
72 xpos_2_2(minSize+1:end) = [];
73 ypos_2_2(minSize+1:end) = [];
74 xpos_3_2(minSize+1:end) = [];
75 ypos_3_2(minSize+1:end) = [];
76 %%
77 %close all;
78 % Plot third case
79 figure(1)
80 plot(ypos_1_3)
81 hold on
82 plot(ypos_2_3)
83
84 plot(ypos_3_3)
85 legend('cam 1', 'cam 2', 'cam3')
86 %%
87 close all;
88 % Align data
89 xpos_1_3(1:13) = [];
90 ypos_1_3(1:13) = [];
91 xpos_2_3(1:39) = [];
92 ypos_2_3(1:39) = [];
93 xpos_3_3(1:7) = [];
94 ypos_3_3(1:7) = [];
95
96 figure(1)
97 plot(ypos_1_3)
98 hold on
99 plot(ypos_2_3)
100
101 plot(ypos_3_3)
102 legend('cam 1', 'cam 2', 'cam3')
103
104
105 sizes = [length(xpos_1_3),length(xpos_2_3),length(xpos_3_3)];
106 minSize = min(sizes);
107 xpos_1_3(minSize+1:end) = [];
108 ypos_1_3(minSize+1:end) = [];
109 xpos_2_3(minSize+1:end) = [];
110 ypos_2_3(minSize+1:end) = [];
111 xpos_3_3(minSize+1:end) = [];
112 ypos_3_3(minSize+1:end) = [];
113 %%
114 close all;
115 % Plot fourth case
116 figure(1)
117 plot(ypos_1_4)
118 hold on
119 plot(ypos_2_4)
120 plot(ypos_3_4)

```

```

121 legend('cam 1', 'cam 2', 'cam3')
122 %%
123 % Align data
124 %xpos_1_4(1:9) = [];
125 %ypos_1_4(1:9) = [];
126 xpos_2_4(1:9) = [];
127 ypos_2_4(1:9) = [];
128 %xpos_3_4(1) = [];
129 %ypos_3_4(1) = [];
130
131 figure(2)
132 plot(ypos_1_4)
133 hold on
134 plot(ypos_2_4)
135 plot(ypos_3_4)
136 legend('cam 1', 'cam 2', 'cam3')
137
138
139 sizes = [length(xpos_1_4),length(xpos_2_4),length(xpos_3_4)];
140 minSize = min(sizes);
141 xpos_1_4(minSize+1:end) = [];
142 ypos_1_4(minSize+1:end) = [];
143 xpos_2_4(minSize+1:end) = [];
144 ypos_2_4(minSize+1:end) = [];
145 xpos_3_4(minSize+1:end) = [];
146 ypos_3_4(minSize+1:end) = [];
147
148 %%
149 close all
150
151 snapShot1 = [xpos_1_1';ypos_1_1';xpos_2_1';ypos_2_1';xpos_3_1';ypos_3_1'];
152 snapShot2 = [xpos_1_2';ypos_1_2';xpos_2_2';ypos_2_2';xpos_3_2';ypos_3_2'];
153 snapShot3 = [xpos_1_3';ypos_1_3';xpos_2_3';ypos_2_3';xpos_3_3';ypos_3_3'];
154 snapShot4 = [xpos_1_4';ypos_1_4';xpos_2_4';ypos_2_4';xpos_3_4';ypos_3_4'];
155
156 mean1 = mean(snapShot1,2);
157 snapShot1 = snapShot1-mean1;
158
159 mean2 = mean(snapShot2,2);
160 snapShot2 = snapShot2-mean2;
161
162 mean3 = mean(snapShot3,2);
163 snapShot3 = snapShot3-mean3;
164
165 mean4 = mean(snapShot4,2);
166 snapShot4 = snapShot4-mean4;
167
168 figure(1)
169 plot(snapShot2(2,:))
170 hold on
171 plot(snapShot2(4,:))
172 plot(snapShot2(6,:))
173 legend('cam 1', 'cam 2', 'cam3')
174 save('snapShots.mat', '-regexp', '^snapShot')

```

## performPCA.m

```
1 clear; clc; close all;
2 load('snapShots.mat')
3
4 % Perform SVD on each snapshot matrix
5 [U1,S1,V1] = svd(snapShot1, 'econ');
6 [U2,S2,V2] = svd(snapShot2, 'econ');
7 [U3,S3,V3] = svd(snapShot3, 'econ');
8 [U4,S4,V4] = svd(snapShot4, 'econ');
9
10 % Plot singular values of each matrix
11 figure(1)
12 plot(diag(S1),'.-', 'linewidth', 2, 'markersize', 20); hold on
13 plot(diag(S2),'.-', 'linewidth', 2, 'markersize', 20)
14 plot(diag(S3),'.-', 'linewidth', 2, 'markersize', 20)
15 plot(diag(S4),'.-', 'linewidth', 2, 'markersize', 20)
16 title('Singular Values for Each Case', 'fontsize', 15)
17 legend('Case 1', 'Case 2', 'Case 3', 'Case 4')
18
19 % Compute low rank approximations for each matrix
20 n = 1;
21 rankn_1 = U1(:,1:n)*S1(1:n,1:n)*V1(:,1:n)';
22 rankn_2 = U2(:,1:n)*S2(1:n,1:n)*V2(:,1:n)';
23 rankn_3 = U3(:,1:n)*S3(1:n,1:n)*V3(:,1:n)';
24 rankn_4 = U4(:,1:n)*S4(1:n,1:n)*V4(:,1:n)';
25
26 % Plot low rank approximations of positions
27 figure(2)
28 subplot(2,2,1)
29 plot(rankn_1(1,:), 'linewidth', 2); hold on
30 plot(rankn_1(3,:), 'linewidth', 2)
31 plot(rankn_1(5,:), 'linewidth', 2)
32 title('Case 1')
33
34 subplot(2,2,2)
35 plot(rankn_2(1,:), 'linewidth', 2); hold on
36 plot(rankn_2(3,:), 'linewidth', 2)
37 plot(rankn_2(5,:), 'linewidth', 2)
38 title('Case 2')
39
40 subplot(2,2,3)
41 plot(rankn_3(1,:), 'linewidth', 2); hold on
42 plot(rankn_3(3,:), 'linewidth', 2)
43 plot(rankn_3(5,:), 'linewidth', 2)
44 title('Case 3')
45
46 subplot(2,2,4)
47 plot(rankn_4(1,:), 'linewidth', 2); hold on
48 plot(rankn_4(3,:), 'linewidth', 2)
49 plot(rankn_4(5,:), 'linewidth', 2)
50 title('Case 4')
51
52 sgtitle('Low Rank Approximations of X positions', 'fontsize', 15)
53 legend({'Cam 1', 'Cam 2', 'Cam 3'}, 'fontsize', 10)
```

```

54
55 figure(3)
56 subplot(2,2,1)
57 plot(rankn_1(2,:), 'linewidth', 2); hold on
58 plot(rankn_1(4,:), 'linewidth', 2)
59 plot(rankn_1(6,:), 'linewidth', 2)
60 title('Case 1')
61
62 subplot(2,2,2)
63 plot(rankn_2(2,:), 'linewidth', 2); hold on
64 plot(rankn_2(4,:), 'linewidth', 2)
65 plot(rankn_2(6,:), 'linewidth', 2)
66 title('Case 2')
67
68 subplot(2,2,3)
69 plot(rankn_3(2,:), 'linewidth', 2); hold on
70 plot(rankn_3(4,:), 'linewidth', 2)
71 plot(rankn_3(6,:), 'linewidth', 2)
72 title('Case 3')
73
74 subplot(2,2,4)
75 plot(rankn_4(2,:), 'linewidth', 2); hold on
76 plot(rankn_4(4,:), 'linewidth', 2)
77 plot(rankn_4(6,:), 'linewidth', 2)
78 title('Case 4')
79
80 sgtitle('Low Rank Approximations of Y positions', 'fontsize', 15)
81 legend({'Cam 1','Cam 2','Cam 3'}, 'fontsize', 10)
82
83 %%
84 % Plot original positions
85 figure(1)
86 subplot(2,2,1)
87 plot(snapShot1(1,:), 'linewidth', 2); hold on
88 plot(snapShot1(3,:), 'linewidth', 2)
89 plot(snapShot1(5,:), 'linewidth', 2)
90 title('Case 1')
91
92 subplot(2,2,2)
93 plot(snapShot2(1,:), 'linewidth', 2); hold on
94 plot(snapShot2(3,:), 'linewidth', 2)
95 plot(snapShot2(5,:), 'linewidth', 2)
96 title('Case 2')
97
98 subplot(2,2,3)
99 plot(snapShot3(1,:), 'linewidth', 2); hold on
100 plot(snapShot3(3,:), 'linewidth', 2)
101 plot(snapShot3(5,:), 'linewidth', 2)
102 title('Case 3')
103
104 subplot(2,2,4)
105 plot(snapShot4(1,:), 'linewidth', 2); hold on
106 plot(snapShot4(3,:), 'linewidth', 2)
107 plot(snapShot4(5,:), 'linewidth', 2)
108 title('Case 4')

```

```

109
110 sgtitle('Actual X positions', 'fontsize', 15)
111 legend({'Cam 1','Cam 2','Cam 3'}, 'fontsize', 10)
112
113 figure(2)
114 subplot(2,2,1)
115 plot(snapShot1(2,:), 'linewidth', 2); hold on
116 plot(snapShot1(4,:), 'linewidth', 2)
117 plot(snapShot1(6,:), 'linewidth', 2)
118 title('Case 1')
119
120 subplot(2,2,2)
121 plot(snapShot2(2,:), 'linewidth', 2); hold on
122 plot(snapShot2(4,:), 'linewidth', 2)
123 plot(snapShot2(6,:), 'linewidth', 2)
124 title('Case 2')
125
126 subplot(2,2,3)
127 plot(snapShot3(2,:), 'linewidth', 2); hold on
128 plot(snapShot3(4,:), 'linewidth', 2)
129 plot(snapShot3(6,:), 'linewidth', 2)
130 title('Case 3')
131
132 subplot(2,2,4)
133 plot(snapShot4(2,:), 'linewidth', 2); hold on
134 plot(snapShot4(4,:), 'linewidth', 2)
135 plot(snapShot4(6,:), 'linewidth', 2)
136 title('Case 4')
137
138 sgtitle('Actual Y positions', 'fontsize', 15)
139 legend({'Cam 1','Cam 2','Cam 3'}, 'fontsize', 10)
140
141 %%
142 % Plot right singular vectors
143 subplot(2,2,1)
144 plot(V1(:,1:n), 'linewidth', 2)
145 title('Case 1')
146
147 subplot(2,2,2)
148 plot(V2(:,1:n), 'linewidth', 2)
149 title('Case 2')
150
151 subplot(2,2,3)
152 plot(V3(:,1:n), 'linewidth', 2)
153 title('Case 3')
154
155 subplot(2,2,4)
156 plot(V4(:,1:n), 'linewidth', 2)
157 title('Case 4')
158
159 sgtitle('Right Singular Vectors for Each Case', 'fontsize', 15)

```