MAST 680 Assignment 2: Learning Dynamics from Video

Will Sze

March 07, 2023

Abstract

Many phenomena in the world are complex and some may have dynamics which have yet to be discovered. A method of identifying models called SINDy may be used to obtain a simple yet accurate model of a system. In this paper, SINDy is used to attempt to learn the dynamics of a mass-spring system. Videos of different angles were given and the mass positions were obtained. PCA was also needed to extract additional an additional component. The model produced by SINDy was very close to the known system, however, there are still some discrepancies.

1 Introduction and Overview

Traditional approaches to modelling physical systems have been to find the governing equations analytically through fundamental principles. The models could then be verified through experimentation. However, many physical phenomenons are complex and finding their governing equations that accurately model their behaviour may be challenging[3]. It is not until the 1980s that people started to characterize dynamic systems through the use of time series data[3]. There have been many more advances in this field. One of them is the Sparse Identification of Non-Linear Dynamics or SINDy for short. The SINDy method allows us to uncover the underlying dynamics of a system with few terms through sparse regression[1].

In this current paper, the SINDy method was implemented to learn the dynamics of a mass-spring system. Two sets of videos were given: one with relatively low noise and the other with the noise from the camera shaking. Each set contains three videos representing the different camera angles of the system.

2 Theoretical Background

Before working with SINDy, the steps leading to the method will be defined. A model of a dynamical system in its spatial coordinates (x,y,z) is usually described as equation 1.

$$\dot{x} = f(t, x, y, z); \quad \dot{y} = g(t, x, y, z); \quad \dot{z} = h(t, x, y, z) \tag{1}$$

where $\dot{x}$, $\dot{y}$ and $\dot{z}$ are the time derivatives of the spatial coordinates; f, g and h are functions in terms of the spatial coordinates and elapse time t. equation 1 can be reformulated into matrices where similar operations to DMD seen in assignment 1 can be performed. The left-hand side will be represented by the matrix Y containing the time derivatives of the collected data matrix X. The data matrix X contains N columns representing the coordinates collected at each time step and the rows representing each variable. Since the data are collected discretely, the time derivatives may be estimated using the finite difference method $\frac{dx_n}{dt} \approx \frac{x_{n+1}-x_n}{\Delta t}$ where $n \in \{0, 1, 2, ..., N-1\}$ and Δt is the time difference between each measurement. The functions on the right-hand side can be approximated with the sum of different basic functions with coefficients. These different functions called θ form a dictionary $\mathcal{D}$ defined by the user.

$$\mathcal{D} = \{\theta_1, \theta_2, ..., \theta_K\} \tag{2}$$

The subscript K represents the number of user-defined functions. These functions can be linear, polynomials, sine functions, exponential, etc. of x, y or z. Equation 3 illustrates the matrix reformulation for a single time frame to save on space, but it can be extended to a matrix of multiple time frames.

$$\begin{aligned}\dot{x} &\approx a_1\theta_1 + a_2\theta_2 + a_3\theta_3 + ... + a_K\theta_K \\ \dot{y} &\approx b_1\theta_1 + b_2\theta_2 + b_3\theta_3 + ... + b_K\theta_K \\ \dot{z} &\approx c_1\theta_1 + c_2\theta_2 + b_3\theta_3 + ... + c_K\theta_K\end{aligned} \rightarrow \begin{bmatrix}\dot{x}\\ \dot{y}\\ \dot{z}\end{bmatrix} \approx \begin{bmatrix}a_1 & a_2 & a_3 & ... & a_K\\ b_1 & b_2 & b_3 & ... & b_K\\ c_1 & c_2 & c_3 & ... & b_K\end{bmatrix}\begin{bmatrix}\theta_1\\ \theta_2\\ \theta_3\\ ...\\ \theta_K\end{bmatrix} \rightarrow Y \approx \Xi\Theta(X) \tag{3}$$

Since Y is estimated from the data and $\Theta(X)$ is user-defined, only Ξ is unknown. As with DMD, the matrices are not square thus the coefficient matrix Ξ should be obtained such that it minimizes the difference between both sides of the approximate equality. Just as with DMD, the pseudo-inverse, denoted with the $\dagger$ symbol, will be used to find the coefficient matrix Ξ.

$$\Xi = Y\Theta(X)^{\dagger} \tag{4}$$

Performing equation 4, a model can be fitted to describe the system approximately. However, it is advantageous to find a system in which the model is easier to work with yet remains accurate.

The user-defined dictionary, in the beginning, may contain some insignificant terms. This will be shown by coefficient values of smaller magnitudes. The SINDy method helps iron out the terms that are more significant. The method imposes a sparsity threshold λ to collect the coefficients higher than this value. It is an iterative method imposing this constraint and re-evaluating equation 4 to obtain a new set of coefficients. The iteration stops when the new coefficients does not change from the previous one. This is different than just finding Ξ once and picking the coefficients higher than the threshold. In doing so, the model is less accurate due to some loss of information. In contrast, the SINDy method allows most of the information to be kept while best fitting the model to the smallest amount of terms.

Up to now, it is assumed the data matrix X is noise free. However, in reality, the data is messy. Thus finding the matrix Y with the finite derivative method can lead to inaccuracies. Any noise present will be amplified as shown in section 3 of [4]. Instead of using finite derivatives, an integration formulation can be used to average out the data. Starting from any of the formulations in equation 3, both sides are integrated to obtain equation 5.

$$X(t) - X(0) \approx \Xi\int_0^t \Theta(X(s))\,ds \tag{5}$$

The variable s is only a change of variable for integration. X(t) is the position at time t and X(0) is the initial position or simply the first data. The integration of discrete data sets can be estimated by performing a simple Riemann Sum. This means at each moment in time, we multiply the function by Δt and add the value to the total approximating the area under the curve.

$$\mathcal{I} = \int_0^{t_n} \Theta(X(s))\,ds = \sum_{i=1}^{n} \Delta t\Theta(X(t_n)) \tag{6}$$

Rewriting equation 5 by replacing $X(t) - X(0)$ by $\tilde{X}$ and using I from equation 6, the same form of equation 3 can be obtained. Thus the pseudo-inverse can be used to find Ξ.

$$\Xi = \tilde{X}\mathcal{I}^{\dagger} \tag{7}$$

The SINDy method can be performed the same way as stated earlier to find a sparse representation of the system.

Regarding the problem of a one-dimensional damped mass-spring system travelling in the x direction, the dynamics of the system, in state-space representation, can be shown to be equation 8 using Newton's second law.

$$\begin{aligned}\dot{z_1} &= z_2\\ \dot{z_2} &= -\frac{K}{M}z_1 - \frac{c}{M}z_2\end{aligned} \tag{8}$$

z_1 represents the x-position, z_2 represents the x-velocity, K is the spring constant in $[N/m]$, M is the mass of the system in $[kg]$ and c is the damping coefficient in $[Ns/m]$. For a single-dimension problem with the involvement of acceleration, such as a mass-spring system, the use of different camera angles to collect data along with a Principal Component Analysis (PCA) may be used to uncover the velocity component of the system.

PCA, in essence, redefines the data in a way that variables become uncorrelated. To do this, the variance and covariance of the mean subtracted data are used. Variance and covariance, in statistics, are measures of the spread and correlation of between measurements respectively. These variances and covariances can be calculated in a single matrix (C_x) by multiplying X with its transpose.

$$C_x = \frac{1}{N-1} X X^T \tag{9}$$

The factor $\frac{1}{N-1}$ comes from the calculation of variances in the sampled population where N is the number of data points. C_x is a symmetric matrix thus its eigenvectors can be found. These vectors are called the principal components. Since the eigenvectors are orthogonal to each other, they can be projected to the data using equation 10 to create new variables which are uncorrelated. Furthermore, the eigenvalues provide the magnitude of correlation associated with the principal component direction.

$$X_{new} = U^T X \tag{10}$$

X_{new} is the new uncorrelated data and U is a matrix containing the eigenvectors. For the mass-spring system, applying PCA will uncover two dominant principal components which provide the position as well as the velocity directions of the system. The SINDy procedure described earlier could then be used on the resulting PCA data to find the model.

3 Algorithm Implementation and Development

To solve the problem, the algorithms were written in Octave. The scripts can be found in Appendix B. As mentioned in assignment 1, Octave scripts are interchangeable with Matlab with few exceptions. Any pre-made functions used will be described in Appendix A. The problem was first solved by extracting the position of the mass-spring system in each of the video files in a set, performing PCA on the data obtained and then finally applying SINDy to find the underlying dynamics. The algorithms described in the coming sections were performed on the first video set containing non-noisy videos and will not work on the second set containing noisy videos. A separate approach will be suggested to tackle the noisy video set in the future.

3.1 Locating the Mass

The first step was to load the video files into Octave. The video was stored as a matrix with 4 dimensions. Each of the dimensions represents, in order, the vertical pixels, the horizontal pixels, the RGB values and the different frames. The video can be viewed by taking the values of the first three dimensions and viewing the frames in succession. The video was turned into a grayscale for ease of computation. The data was then converted into a single matrix of dimensions M and N, with M representing the number of pixels in a frame and N representing the number of frames.

To locate the mass in each of the frames, a simple background-subtraction method was used. There are many different ways of recreating the background in a simple manner [2]. Here, we used, at each pixel, the median or mode value of the pixel intensity from all the frames to construct the background depending on which method gave the better result. This will give a static image of the background since it is capturing the pixels appearing the most and groups all N pixels in a row into a single frame. The background was then subtracted from the original gray video, leaving only the objects in motion visible. A threshold intensity of 50 was set to remove some of the noise. Higher intensities were set to 255 and the lower intensities were set to 0 to produce a pure black-and-white image. This method worked well for the first and third videos but had some issues with the second due to a larger level of camera shake. In this case, the median of the pixel intensity was calculated in sections of 4 frames to provide a background that is adapting to the video.

The background subtracting method gave a bright area on each frame which has a larger surface area than the noise elements that appeared. Thus the x and y centroid positions of the mass in each frame can be calculated by taking the mean pixel intensities of each column (x-centroid) and row (y-centroid) and then using equation 11.

$$x_c = \frac{\sum I_{col}x}{\sum I}; \;\; y_c = \frac{\sum I_{row}y}{\sum I} \tag{11}$$

x_c and y_c are the centroid positions and I are the mean pixel intensity values at each x and y pixel position. This equation is analogous to the mass centroid calculation of a physical system. If the noise is large, the centroid could be shifted. To counter this, the pixel range containing the mass was manually selected in the first frame, and the pixels outside this boundary were turned black. The centroid in this boundary could be found using equation 11. Using previous positions, a new slightly larger boundary can be set in the next frame. Repeating the steps leads to a fairly accurate tracking algorithm.

Algorithm 1: Locating the Mass

```
Import data from cami_j.mat, i ∈ {1,2,3}, j ∈ {1,2}
for k = 1 : Numberofframes do
  Convert to grayscale
  Data(:,k) = Reshape grayscale k^th frame
end for
for k = 1 : M do
  Background(k,:) = median(I,:)
end for
Subtract background and apply threshold
for k = 1 : N do
  Reshape k^th frame of Data into frame width × height
  if k == 1, create boundary around mass
  Else, take the previous frame location and create a new bounded region
  Calculate the mean intensities of each row and col of frame
  Calculate the x and y centroid positions and collect value into array
  Reshape back into MxN Data
end for
Save collected positions into pos.mat file
```

3.2 PCA Implementation

Once the x and y positions have been obtained for the three videos, their mean values were subtracted and the values were brought together in a single matrix stacking one on top of the other. This matrix is of size $6 \times N$. PCA was then performed on this matrix. Instead of finding C_x and computing its eigenvectors, SVD will be used. To do this, a matrix A was created by taking X and dividing by $\sqrt{N-1}$. Since multiplying A by its transpose will find C_x, Performing SVD on A will give the principal components U but now the vectors are in order of importance. Thus we can use equation 10 to readily find the new coordinates using just the first two principal components. Lastly, a filtering function was used to reduce the noise in the new data. Furthermore, only a section (125 frames) in the middle of the data was kept.

Algorithm 2: Implementation of PCA

```
Import data from pos.mat
Create matrix X by stacking the 6 positions
Create A by dividing elements of X by √(N − 1)
[U,S,V] = svd(A,'econ')
X_new = U'X
```

3.3 SINDy Implementation

The X_{new} obtained from PCA contains two rows of information. The first (z_1) indicates the position and the second (z_2) indicates the velocity directions. Here, we use the variable z instead of x to be consistent with equation 8. Since the second principal component does not actually give the exact magnitude of the velocity, this principal component was multiplied by a scale factor. This scale factor was found using the finite derivative method on the position and taking the ratio of the maximum values between the finite derivative and the second principal component. To further reduce the effect of noise, the integral method was implemented instead of the time derivatives. The dictionary was defined as the set of polynomials up to degree 2. The higher degree terms would allow us to see how SINDy performs.

$$D = \{1, z_1, z_2, z_1^2, z_1 z_2, z_2^2\} \tag{12}$$

Θ was created by performing element-wise multiplication of the data from X_{new}. Each row of Θ would be a different function. By applying equation 5, 6 and 7 the first set of coefficients was found. A time step Δt of 1 was used as it curiously provided a better result. A loop was then created to continuously select indices which contain coefficients larger than the sparsity threshold value and re-apply equation 7 with those indices until the coefficients stay the same as those from the last iteration. The sparsity threshold was taken as 0.1 to obtain the closest result to the actual system. The system of equations was displayed with very small values in the coefficients omitted. Finally, the SINDy algorithm was validated against a dataset of a known solution of the damped mass-spring system which it was able to obtain the exact system of 8 even with small artificially induced noise.

Algorithm 3: Implementation of SINDy

```
Create X̃ by subtracting X_new(0) from X_new(:, 2 : end)
Perform Θ(i, :) = D_i using element-wise operation of X_new
Calculate I
coeff = X̃ pinv(I)
while |coeff − prevCoeff| ≠ 0 or first iteration do
    Find indices of coeff smaller than threshold λ and make them 0 in the new coeff
    Calculate new coeff using the indices with coeff bigger than λ using pinv
end while
Display system on screen
```

4 Computational Results

The data obtained from the mass locating algorithm showed almost perfect sinusoidal waves as seen in figure 1. This indicates that the first algorithm performs well in capturing the movement of the mass. In one of the videos, the mass seemed to be out of phase with the other two. This phase shift allowed PCA to discover both the position and the velocity components. These were the 2 dominant principal components. We can see from the bottom portion of figure 1 that the blue curve (second principal component) resembles the velocity of the red curve. If the second principal component data was scaled to where the amplitude matches the amplitude of the actual derivative of the first principal component, then the exact velocity is uncovered (see Appendix C for complementary figures). This confirms that it is the velocity component.

Performing SINDy method on the filtered data with a time step (dt) of 1 and a sparsity threshold (λ) of 0.1, we arrive at a system close to the theoretical system of equation 8.

$$\begin{aligned} \dot{z_1} &= 0.986784 z_2 \\ \dot{z_2} &= 0.150838 - 0.130678 z_1 \end{aligned} \tag{13}$$

However, there are still quite a few differences. The first is that the coefficient in front of z_2 is not exactly 1. This difference could be due to a combination of the remaining noise in the data, the pseudoinverse which can only best fit the data as well as the sparsity threshold. The second is that there is a constant term

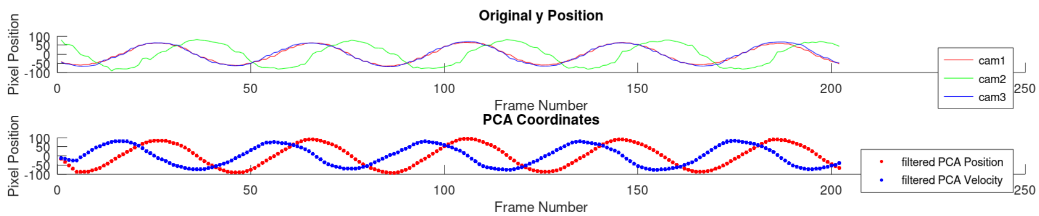

Figure 1: Comparison between the mean adjusted y-positions obtained with the mass locating algorithm (TOP) and the PCA variables (BOTTOM)

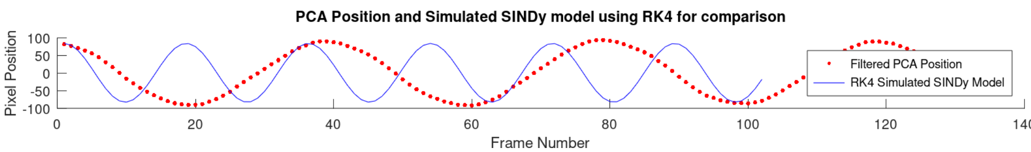

Figure 2: PCA position compared with the simulated SINDy model

which appears in the $\dot{z_2}$ equation. This may be due to noise in the data as well. Particularly, the blue velocity curve from figure 1 appears less smooth than the red position curve. Theoretically, the constant means there is a constant external force (other than gravity and the spring) acting on the mass. The noise could be producing this effect. The third discrepancy is that there is no z_2 term in $\dot{z_2}$. However, this may not be surprising since the decay in the sine wave seems to be negligible. Lastly, and perhaps less obvious if the theoretical system was unknown, the coefficient in front of z_1 shows that the system is oscillating at a higher frequency than what the data shows. In fact, the coefficient K/M represents the square of the natural frequency $\omega_n^2 = \frac{K}{M}$. If we simulate the result with an initial condition matching the data and plotting it with the first principal component, then we see a faster oscillation than the original (figure 2).

This difference might be due to the sparsity threshold being higher than the actual natural frequency. However, this should mean that the z_1 would be completely gone. Lowering the threshold resulted in some undesirable coefficients appearing which further increases the discrepancies. The time step could also be an area of concern since 1 second between each frame seems slow and unrealistic. However, decreasing this value only increases the differences.

Note that the described algorithm to locate the mass would not be able to extract the positions of the second set of videos which contain a larger amount of noise. Due to time constraints, a new tracking algorithm was not able to be developed. However, one could use a cross-correlation technique by comparing each frame to an image of the mass to locate the position based on the highest correlation index [5]. Based on current results with the low-noise videos, it is suspected that SINDy would have a hard time extracting the model of equation 8 without excessive filtering of data.

5 Summary and Conclusions

In summary, the SINDy method is used to obtain a simple yet accurate model of a dynamical system. This technique was examined on a mass-spring system. PCA was first applied to data collected from three perspectives of this one-dimension system to uncover the velocity component. The results from SINDy with a sparsity parameter of 0.1 and a time difference of 1 second resulted in a better model than other combinations. However, there were still some discrepancies possibly due to noise or outliers in the data. The reasons for the differences concerning the frequencies are not yet understood.

References

[1] Steven L. Brunton, Joshua L. Proctor, and J. Nathan Kutz. “Discovering governing equations from data by sparse identification of nonlinear dynamical systems”. In: *Proceedings of the National Academy of*

Sciences 113.15 (2016), pp. 3932–3937. DOI: 10.1073/pnas.1517384113. eprint: https://www.pnas.org/doi/pdf/10.1073/pnas.1517384113. URL: https://www.pnas.org/doi/abs/10.1073/pnas.1517384113.

[2] First Principles of Computer Vision. *Change Detection — Object Tracking*. May 2021. URL: https://www.youtube.com/watch?v=0ANl4bZAxjI.

[3] Amin Ghadami and Bogdan I. Epureanu. "Data-driven prediction in dynamical systems: recent developments". In: (2022). DOI: 10.1098/rsta.2021.0213. URL: https://royalsocietypublishing.org/doi/10.1098/rsta.2021.0213.

[4] Jason J. Bramburger. *Data-Driven Methods for Dynamic Systems*. Concordia University, 2023.

[5] MathWorks. *xcorr2 2-D cross-correlation*. URL: https://www.mathworks.com/help/signal/ref/xcorr2.html. (accessed: 07.03.2023).

Appendix A Premade Functions Used

The following premade functions are found in Octave which are the same as in Matlab. The last function was created during a course on Numerical Simulations. It has the same function as ODE45.

- `grayImage = rgb2gray(readFrame(vid))` converts each pixel containing of RGB values to a single value by a weighted sum. These values represent the grayscale of each pixel.
- `reshape(X, M, N)` converts the matrix X into a matrix of size M × N.
- `movie(h, mov, 1, frameRate)` displays the frames inside `mov` structure in the figure box `h`. The number 1 indicates the number of loops and `frameRate` indicates the video fps.
- `[U,S,V] = svd(X,'econ')` returns the unitary matrices `U` and `V`, and the singular values of matrix X. The argument `'econ'` removes all zeros rows or columns to make the matrices compact.
- `round(x)` rounds the value `x` down to the nearest integer.
- `imshow(mov(k).cdata,mov(k).colormap)` displays the k^{th} frame in `mov` structure with its color map onto a figure window.
- `pinv(X)` returns the pseudoinverse ($V\Sigma^{-1}U^*$) of `X` matrix.
- `abs(x)` returns the norm of `x`.
- `sort(X)` returns a vector sorted from smallest to largest of vector `X`.
- `linspace(x,y,n)` produces an array from x to y in n equal spaces.
- `filter(x,y,A)` filters array A according to a transfer function x/y. For a moving average filter, x is an array having a length of the window size and its values are 1 over the window size. y is simply 1.
- `RK4_Sys_Inputs(f, to, yo, n, tf, inputs)` solves numerically system function f with initial time to, initial conditions yo, number of divisions, final time tf and function inputs.

Appendix B Octave Code

B.1 Example Mass Location Code

```
%Mass position extracting on Cam 1
%Will Sze
%40096561

```

```
clear all; close all; clc
pause(0.1);

nfigure = 1;
FrameInterest = 1;

fprintf('\nImporting Video...\n');
tic
  %Load data into Octave
  load cam1_1.mat;

  %Collects video dimensions
  [vidHeight1, vidWidth1, nRGB1, vidNFrames1] = size(vidFrames1_1);
  FrameRate = 60;

  %Movie Struct to collect all original frames to run movie using movie function
  mov1 = struct('cdata', zeros(vidHeight1,vidWidth1,3,'uint8'), 'colormap',[]);
  mov1_Gray = struct('cdata', zeros(vidHeight1,vidWidth1,1,'uint8'), 'colormap',gray(255));
  mov1_BW = struct('cdata', zeros(vidHeight1,vidWidth1,1,'uint8'), 'colormap',gray(255));
  mov1_int = struct('cdata', zeros(vidHeight1*vidWidth1,1,'uint8'), 'colormap',gray(255));

  for k = 1:vidNFrames1
    mov1(k).cdata = vidFrames1_1(:,:,:,k);
  end
toc

%View original video1
fprintf('\nDisplaying Original Video1...\n');
tic
   hf = figure(nfigure);
   set(hf, 'position', [150 150 vidWidth1 vidHeight1]);
   movie(hf, mov1, 1, FrameRate);
   nfigure = nfigure + 1;
toc

fprintf('\nDisplaying Video1 image at frame %d in the movie...\n', FrameInterest);
tic
  hf = figure(nfigure);
  set(hf, 'position', [150 150 vidWidth1 vidHeight1]);
  imshow(mov1(FrameInterest).cdata,mov1(FrameInterest).colormap);
  nfigure = nfigure + 1;
toc


%Converting to grayscale1
tic
  for k = 1:vidNFrames1
    mov1_Gray(k).cdata = rgb2gray(mov1(k).cdata);
```

```
    mov1_Gray(k).colormap = gray(255);
  end
toc


%View GrayScale video1
fprintf('\nDisplaying Gray Scale Video1...\n');
tic
   hf = figure(nfigure);
   set(hf, 'position', [150 150 vidWidth1 vidHeight1]);
   movie(hf, mov1_Gray, 1, FrameRate);
   nfigure = nfigure + 1;
toc

fprintf('\nDisplaying Video1 image at frame %d in the movie...\n',
    FrameInterest);
tic
  hf = figure(nfigure);
  set(hf, 'position', [150 150 vidWidth1 vidHeight1]);
  imshow(mov1_Gray(FrameInterest).cdata,mov1_Gray(FrameInterest).colormap)
     ;
  nfigure = nfigure + 1;
toc



Data1 = zeros(vidWidth1*vidHeight1, vidNFrames1);

%---------- Tracking algorithm -------------%
fprintf('\nCreating Data1...\n');
tic
  for k = 1:vidNFrames1
    mov1_int(k).cdata = reshape(mov1_Gray(k).cdata,[],1);
    mov1_int(k).colormap = gray(255);
    Data1(:,k) = [mov1_int(k).cdata];
  end
toc
[M N] = size(Data1);


IntensityThreshold = 50;
%Mode Background Method
fprintf('\nCreating background on Data1...\n');
tic
  Background = zeros(M,N);
  for i = 1:M
    Background(i,:) = mode(Data1(i,:));
  end
  Data_Test2 = Data1 - Background;

  Data_Test2(Data_Test2>IntensityThreshold) = 255;
  Data_Test2(Data_Test2<=IntensityThreshold) = 0;
toc

```

```
%Horizontal and Vertical Intensities
fprintf('\nLocating the mass...\n');
cc_F_Interest = 1;
x_pos = [];
y_pos = [];
x_Intensity = zeros(vidNFrames1,vidWidth1);
y_Intensity = zeros(vidNFrames1,vidHeight1);
for i = 1:vidNFrames1

  tic
  Frame_Analysis = reshape(Data_Test2(:,i),vidHeight1, vidWidth1);
  Gray_Frame = reshape(Data1(:,i),vidHeight1, vidWidth1);

  %Set Boundary on first frame
  if(i == 1)
    Frame_Analysis(:,1:300) = 0;
    Frame_Analysis(:,390:vidWidth1) = 0;
    Frame_Analysis(1:210,:) = 0;
    Frame_Analysis(320:vidHeight1,:) = 0;
  end
  %Use previous position to locate next boundary
  if(i ~= 1)
    Y_UpperBound = round(vidHeight1-y_pos(i-1))+60;
    Y_LowerBound = round(vidHeight1-y_pos(i-1))-60;
    if(Y_UpperBound > vidHeight1)
      Y_UpperBound = vidHeight1;
    end
    if(Y_LowerBound < 1)
      Y_LowerBound = 1;
    end
    Frame_Analysis(1:Y_LowerBound,:) = 0;
    Frame_Analysis(Y_UpperBound:vidHeight1,:) = 0;

    X_UpperBound = round(x_pos(i-1))+50;
    X_LowerBound = round(x_pos(i-1))-50;
    if(X_UpperBound > vidWidth1)
      X_UpperBound = vidWidth1;
    end
    if(X_LowerBound < 1)
      X_LowerBound = 1;
    end
    Frame_Analysis(:,1:X_LowerBound) = 0;
    Frame_Analysis(:,X_UpperBound:vidWidth1) = 0;
  end

  %Calculating Centroid
  for j = 1:vidWidth1
      x_Intensity(i,j) = mean(Frame_Analysis(:,j));
  end
  for j = 1:vidHeight1
    y_Intensity(i,vidHeight1+1-j) = mean(Frame_Analysis(j,:));
  end

  %X-centroid
```

```
  SumWeightPos = 0;
  SumWeight = 0;
  for j = 1:vidWidth1
    SumWeightPos = SumWeightPos + j*x_Intensity(i,j);
    SumWeight = SumWeight + x_Intensity(i,j);
  end
  x_pos = [x_pos,SumWeightPos/SumWeight];

  %Y-centroid
  SumWeightPos = 0;
  SumWeight = 0;
  for j = 1:vidHeight1
    SumWeightPos = SumWeightPos + j*y_Intensity(i,j);
    SumWeight = SumWeight + y_Intensity(i,j);
  end
  y_pos = [y_pos,SumWeightPos/SumWeight];

  %Visualize Point in video
  Frame_Analysis(:,round(x_pos(i))) = 180;
  Gray_Frame(:,round(x_pos(i))) = 0;
  Frame_Analysis(vidHeight1 - round(y_pos(i)),:) = 180;
  Gray_Frame(vidHeight1 - round(y_pos(i)),:) = 0;

  Data_Test2(:,i) = reshape(Frame_Analysis,vidHeight1*vidWidth1,1);
  Data1(:,i) = reshape(Gray_Frame,vidHeight1*vidWidth1,1);
  toc
end


fprintf('\nPlotting intensity curves at frame %d in the movie...\n',
   cc_F_Interest);
tic
  figure(nfigure);
  plot(x_Intensity(cc_F_Interest,:),'-o');
  nfigure = nfigure + 1;

  y = linspace(1,vidHeight1,vidHeight1);
  figure(nfigure);
  plot(y_Intensity(cc_F_Interest,:),y,'-o');
  nfigure = nfigure + 1;
toc


%View Results
tic
  for k = 1:vidNFrames1
    mov1_int(k).cdata = [cast(Data_Test2(:,k),'uint8')];
    mov1_BW(k).cdata = reshape(mov1_int(k).cdata,vidHeight1, vidWidth1);
    mov1_BW(k).colormap = gray(255);

  end
toc

fprintf('\nDisplaying Black and White Video1...\n');
```

```
tic
   hf = figure(nfigure);
   set(hf, 'position', [150 150 vidWidth1 vidHeight1]);
   movie(hf, mov1_BW, 1, FrameRate);
   nfigure = nfigure + 1;
toc

fprintf('\nDisplaying Video1 image at frame %d in the movie...\n',
    FrameInterest);
tic
  FrameInterest = cc_F_Interest;
  hf = figure(nfigure);
  set(hf, 'position', [150 150 vidWidth1 vidHeight1]);
  imshow(mov1_BW(FrameInterest).cdata,mov1_BW(FrameInterest).colormap);
  nfigure = nfigure + 1;
toc



tic
  for k = 1:vidNFrames1
    mov1_int(k).cdata = [cast(Data1(:,k),'uint8')];
    mov1_Gray(k).cdata = reshape(mov1_int(k).cdata,vidHeight1, vidWidth1);
    mov1_Gray(k).colormap = gray(255);

  end
toc

fprintf('\nDisplaying Gray with centroid of Video1...\n');
tic
   hf = figure(nfigure);
   set(hf, 'position', [150 150 vidWidth1 vidHeight1]);
   movie(hf, mov1_Gray, 1, FrameRate);
   nfigure = nfigure + 1;
toc
```

B.2 PCA and SINDy Implementation Code

```
%PCA and SINDy Implementation
%Will Sze
%40096561
clear all; close all; clc;

%Loading Data from extracted location of each camera
load cam1_1_xPos2.mat
load cam1_1_yPos2.mat
cam1_xPos = x_pos;
cam1_yPos = y_pos;

load cam2_1_xPos.mat
load cam2_1_yPos.mat
cam2_xPos = x_pos(1:length(cam1_xPos));
cam2_yPos = y_pos(1:length(cam1_yPos));
```

```
load cam3_1_xPos.mat
load cam3_1_yPos.mat
cam3_xPos = x_pos(1:length(cam1_xPos));
cam3_yPos = y_pos(1:length(cam1_yPos));

clear 'x_pos';
clear 'y_pos';

nfigure = 1;

%Plots all three camera y positions
figure(nfigure);
hold on
  plot(cam1_yPos);
  plot(cam2_yPos);
  plot(cam3_yPos);
hold off
nfigure++;

%Removes first few data
Start = 25;
X_og(1,:) = cam1_xPos(Start:end);
X_og(2,:) = cam1_yPos(Start:end);
X_og(3,:) = cam2_xPos(Start:end);
X_og(4,:) = cam2_yPos(Start:end);
X_og(5,:) = cam3_xPos(Start:end);
X_og(6,:) = cam3_yPos(Start:end);

for i = 1:6
  X_og_mean(i) = mean(X_og(i,:));
  X_PCA(i,:) = X_og(i,:) - X_og_mean(i);
end

%Plotting the mean subtracted data
figure(nfigure);
hold on
  plot(X_PCA(2,:),'r.','MarkerSize', 10);
  plot(X_PCA(4,:),'g.','MarkerSize', 10);
  plot(X_PCA(6,:),'b.','MarkerSize', 10);
hold off
nfigure++;


%--------- Applying PCA -----------%
A = 1/sqrt(length(X_og)-1)*X_PCA;

%A_AT = A*A'; %Alternative
%[eig_Vect, eig_Val] = eig(A_AT); %Alternative

[U,S,V] = svd(A,'econ');
Rank_S = 2; %First 2 principal components
Y_PCA = U(:,1:Rank_S)'*X_PCA;

```

```
%Ploting Original mean subtracted data and the data obtained from PCA
figure(nfigure);

  subplot (2, 1, 1)
  hold on
  plot(X_PCA(2,:),'r','MarkerSize', 10);
  plot(X_PCA(4,:),'g','MarkerSize', 10);
  plot(X_PCA(6,:),'b','MarkerSize', 10);
  hold off

  title("Original y Position");
  h = legend ("cam1","cam2","cam3");
  xlabel ("Frame Number");
  ylabel ("Pixel Position");

  subplot (2, 1, 2)
  hold on
  plot(Y_PCA(1,:),'r.','MarkerSize', 10);
  plot(Y_PCA(2,:),'b.','MarkerSize', 10);
  hold off

  title("PCA Coordinates");
  h = legend ("Unfiltered PCA Position","Unfiltered PCA Velocity");
  xlabel ("Frame Number");
  ylabel ("Pixel Position");

nfigure++;

%Comparing with a true sine curve
t = linspace(0,202,203);
y_sin = 90*sin(2*pi/40*t-pi/2-0.5);

figure(nfigure)

  hold on
  plot(Y_PCA(1,:),'r.','MarkerSize', 10);
  plot(y_sin,'b','MarkerSize', 10);

  %Filtering
  windowSize = 5;
  b = (1/windowSize)*ones(1,windowSize);
  a = 1;

  Y_PCA(1,:) = filter(b,a,Y_PCA(1,:));
  Y_PCA(2,:) = filter(b,a,Y_PCA(2,:));

  plot(Y_PCA(1,:),'g','MarkerSize', 10);
  hold off

  title("Comparison Between PCA Position, true sine curve and filtered PCA
      Position");
  h = legend ("PCA Position","True sine curve", "Filtered PCA Position");
  xlabel ("Frame Number");
  ylabel ("Pixel Position");
```

```
nfigure++;

%Plotting the filtered PCA Data
figure(nfigure);

  subplot (2, 1, 1)
  hold on
  plot(X_PCA(2,:),'r','MarkerSize', 10);
  plot(X_PCA(4,:),'g','MarkerSize', 10);
  plot(X_PCA(6,:),'b','MarkerSize', 10);
  hold off

  title("Original y Position");
  h = legend ("cam1","cam2","cam3");
  xlabel ("Frame Number");
  ylabel ("Pixel Position");

  subplot (2, 1, 2)
  hold on
  plot(Y_PCA(1,:),'r.','MarkerSize', 10);
  plot(Y_PCA(2,:),'b.','MarkerSize', 10);
  hold off

  title("PCA Coordinates");
  h = legend ("filtered PCA Position","filtered PCA Velocity");
  xlabel ("Frame Number");
  ylabel ("Pixel Position");


nfigure++;

save("Y_PCA.mat",'Y_PCA');


%--------- Applying SINDy -----------%
Framerate = 1;
dt = 1/Framerate;
N_Start = 28;
N_End = 153;
X_SINDy(1,:) = Y_PCA(1,N_Start:(N_End-1));
X_SINDy(2,:) = Y_PCA(2,N_Start:(N_End-1));

Y_SINDy_dxdt = (Y_PCA(:,N_Start+1:(N_End))-Y_PCA(:,N_Start:(N_End-1)))/dt;
    %derivative
Y_SINDy = Y_PCA(:,N_Start+1:(N_End)) - Y_PCA(:,N_Start).*ones([size(Y_PCA
   (:,N_Start+1:(N_End)))]);

dxdt_Max = max(Y_SINDy_dxdt(1,:));
scaling = dxdt_Max/max(X_SINDy(2,:))
X_SINDy(2,:) = Y_PCA(2,N_Start:(N_End-1))*scaling;

%Plotting the principal components and their derivatives
figure(nfigure);
```

```
  hold on
  plot(X_SINDy(1,:),'r.','MarkerSize', 10);
  plot(X_SINDy(2,:),'b.','MarkerSize', 10);
  plot(Y_SINDy_dxdt(1,:),'g','MarkerSize', 10);
  hold off

  title("PCA Coordinates and derivative comparison");
  h = legend ("Filtered PCA Position","Filtered & Scaled PCA Velocity","
     Derivative PCA Position");
  xlabel ("Frame Number");
  ylabel ("Pixel Position");

nfigure++;

figure(nfigure);

  hold on
  plot(X_SINDy(1,:),'r.','MarkerSize', 10);
  plot(Y_SINDy_dxdt(1,:),'g.','MarkerSize', 10);
  hold off

  title("PCA Position and its derivative");
  h = legend ("PCA Position","Derivative PCA Position");
  xlabel ("Frame Number");
  ylabel ("Pixel Position");

nfigure++;

figure(nfigure);

  hold on
  plot(X_SINDy(2,:),'b.','MarkerSize', 10);
  plot(Y_SINDy_dxdt(2,:),'o.','MarkerSize', 10);
  hold off

  title("PCA Velocity and its derivative");
  h = legend ("PCA Velocity","Derivative PCA Velocity");
  xlabel ("Frame Number");
  ylabel ("Pixel Position");

nfigure++;

% Defining Dictionary (1, x1, x2, x1^2, x1*x2, x2^2);
Theta = ones([1,length(X_SINDy)]);
Theta(2:3,:) = X_SINDy;
Theta(4,:) = X_SINDy(1,:).^2;
Theta(5,:) = X_SINDy(1,:).*X_SINDy(2,:);
Theta(6,:) = X_SINDy(2,:).^2;

%Applying integration method
ThetaInt = zeros([size(Theta)]);
ThetaInt(:,1) = dt*Theta(:,1);

```

```
%reman sum
for i = 2:length(Theta(1,:))
  ThetaInt(:,i) = ThetaInt(:,i-1)+dt*Theta(:,i);
end

coeff = Y_SINDy*pinv(ThetaInt);

%SINDy method begins %adapted from Jason Bramburger's SINDy example code:
    https://github.com/jbramburger/DataDrivenDynSyst/tree/main/Identifying
    %20Nonlinear%20Dynamics
lam = 0.1;
k = 1;

Coeff_New = coeff;
Err = sum(sum((abs(coeff - Coeff_New)))); %Coefficient checking
while k == 1 || Err > 0;

  coeff = Coeff_New; %Save new coeff
  smallinds = (abs(coeff)<lam);
  Coeff_New(smallinds) = 0; %Set new coeff small indices to 0;

  for i = 1:2
    biginds = ~smallinds(i,:);
    Coeff_New(i,biginds) = Y_SINDy(i,:)*pinv(ThetaInt(biginds,:)); %Find
        new coeff for big indices;
  end

  k = k + 1;
  Err = sum(sum((abs(coeff - Coeff_New))));
end

%Display results
fprintf('Expected Output: \n')
fprintf('z1_dot = z2 \n')
fprintf('z2_dot = -Az2 - Bz1 \n')
fprintf('B is K/m (2pi/T)^2 should be around 0.025\n')

mons2 = {''; 'z1'; 'z2'; 'z1^2'; 'z1z2'; 'z2^2'};

fprintf('\nSINDy Output with lam = %d', lam)
fprintf(' and dt = %d:\n', dt)
fprintf('z1_dot = ')
for i = 1:length(Coeff_New)

  if Coeff_New(1,i) < -1e-5;
    mons2_get = mons2{i};
    fprintf(" - %d %s", abs(Coeff_New(1,i)),mons2_get);
  elseif Coeff_New(1,i) > 1e-5
    mons2_get = mons2{i};
    fprintf(" + %d %s", abs(Coeff_New(1,i)),mons2_get);
  end

end
fprintf('\n')
```

```
fprintf('z2_dot = ')
for i = 1:length(Coeff_New)

  if Coeff_New(2,i) < -1e-5
    mons2_get = mons2{i};
    fprintf(" - %d %s", abs(Coeff_New(2,i)),mons2_get);
  elseif Coeff_New(2,i) > 1e-5
    mons2_get = mons2{i};
    fprintf(" + %d %s", abs(Coeff_New(2,i)),mons2_get);
  end

end
fprintf('\n')

%Simulate results and plot
fprintf('\nSimulating SINDy discovered model...\n')
[t_sim,Y_PCA_Sim] = RK4_Sys_Inputs(@f, 0, [85,0], 126, 125, 0);
figure(nfigure);

hold on
plot(Y_PCA(1,N_Start:(N_End-1)),'r.','MarkerSize', 10);
plot(Y_PCA_Sim(:,1),'b','MarkerSize', 10);
hold off

title("PCA Position and Simulated SINDy model using RK4 for comparison");
h = legend ("Filtered PCA Position","RK4 Simulated SINDy Model");
xlabel ("Frame Number");
ylabel ("Pixel Position");

nfigure++;
```

Appendix C Complementary Figures

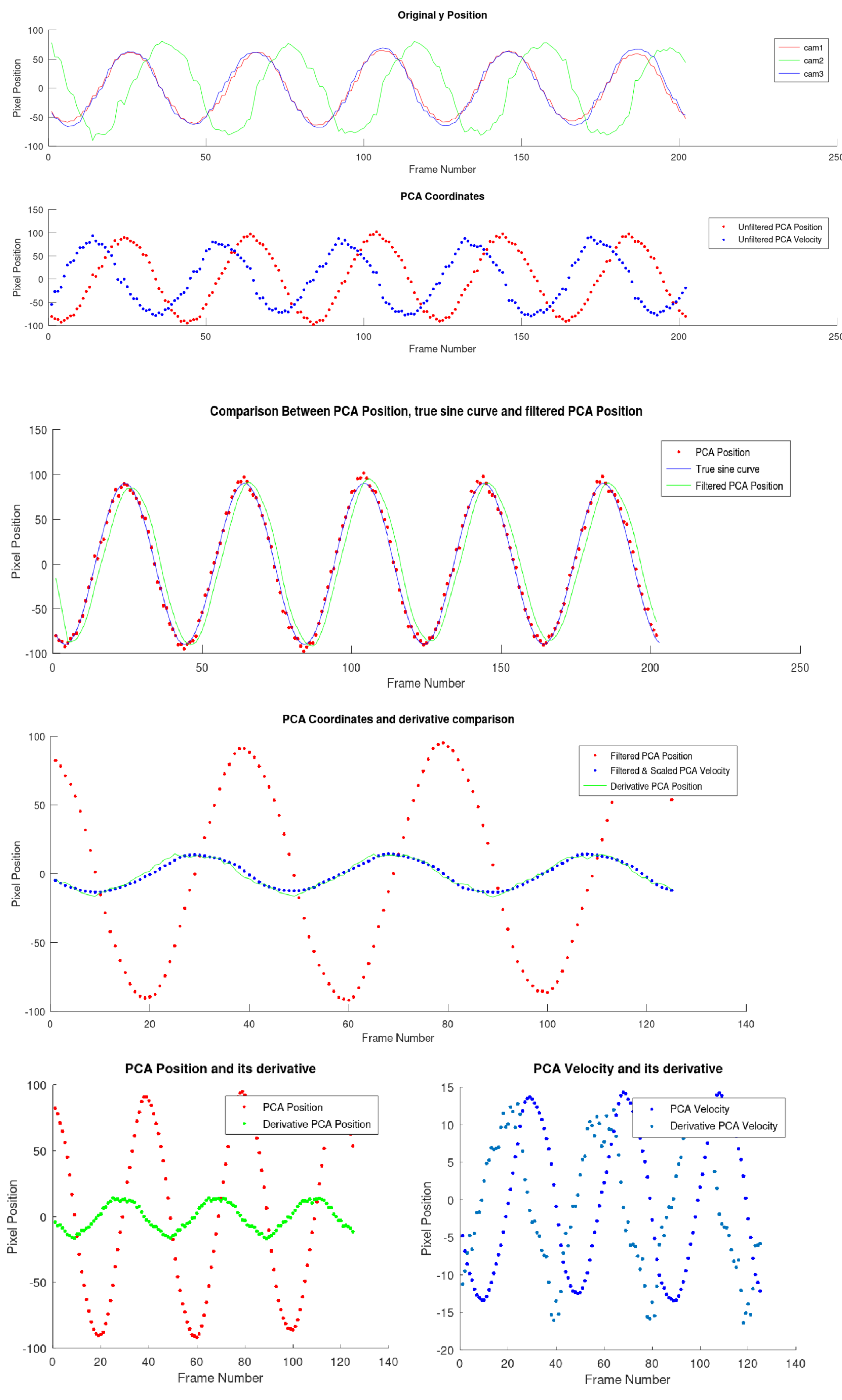

Figure 3: From top to bottom: plot of the original mean subtracted y positions of three different camera angles, plot of the resulting PCA component variables; plot comparing the filtered PCA positions with the true sine curves and the raw PCA positions; plot of the scaled velocity principal component with the true velocity superimposed, plot of the time derivatives of the PCA component variables.