Disappearing Act: Applying Dynamic Mode Decomposition to Video

Timothy Lu

Abstract

This report explores the application of Dynamic Mode Decomposition (DMD) to Video files. The aim is to use DMD to separate video streams into background (low-rank) and foreground (sparse) components. DMD is a powerful tool to analyze large state data. Applying singular value decomposition (SVD) to find a low-rank approximation of the data representing background video, it is possible then to subtract the background from the original video to get the foreground video.

1. Introduction and Overview

Background subtraction is an important technology for video streaming. Whether it is an application to video surveillance or sports broadcasting, if video is being streamed, there must be a way to perform operations in real time. The aim for this report is to attempt and asses the effectiveness of one method of performing background subtraction – *Dynamic Mode Decomposition* (DMD). This report will track the results of performing DMD on a few video samples captured by a smart phone. After preforming the DMD, the effectiveness will be qualitatively assessed through a few frames of video.

More generally, DMD is a tool that enables three primary tasks: (I) *diagnostics*, (II) *state estimation and future state prediction*, and (III) *control*. For the purposes of the report, the DMD method provides a method of collecting "snapshots" of data at certain time frames such that one can predict a future state of a given system. Moreover, these snapshots will be used to separate foreground and background of given video streams.

2. Theoretical Background

DMD deals with matrices that maybe difficult to analyze directly due to a large state dimension n. DMD instead "considers a rank-reduced representation in terms of a *proper orthogonal decomposition* (POD)-projected matrix $\widetilde{\boldsymbol{A}}$." This method collects snapshots of data $\boldsymbol{x}_k$ from a dynamical system at times t_k where $k = 1,2,3, \dots, m$. DMD is essentially "a regression of data onto locally linear dynamics $\boldsymbol{x}_{k+1} = \boldsymbol{A}\boldsymbol{x}_k$ where $\boldsymbol{A}$ is chosen to minimize $\left|\left|\boldsymbol{x}_{k+1} - \boldsymbol{A}\boldsymbol{x}_k\right|\right|_2$ over the k snapshots" (Kutz, 2). The cost of the DMD algorithm is simply the SVD of the snapshot matrix of data $\boldsymbol{x}_k$.

The general DMD algorithm is as follows for a matrix $\boldsymbol{X}$ (this is listed in greater detail in Kutz's book on DMD, Chapter 1.3):

1. Take the singular value decomposition of $\boldsymbol{X}$.

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

   the background for SVD is clearly outline in my third report from this series of reports, so please refer to section 2 of that report for more information on SVD.
2. The matrix $\boldsymbol{A}$ is obtained using the pseudo-inverse of $\boldsymbol{X}$ obtained via the SVD

$$\boldsymbol{A} = \boldsymbol{X'V\Sigma^{-1}U}^{*} \tag{2}$$

   although it is more computationally efficient to compute $\widetilde{\boldsymbol{A}}$ or the $r \times r$ projection of the full matrix $\boldsymbol{A}$ onto POD modes

$$\widetilde{\boldsymbol{A}} = \boldsymbol{U}^{*}\boldsymbol{AU} = \boldsymbol{U}^{*}\boldsymbol{X'V\Sigma}^{-1} \tag{3}$$

   $\widetilde{\boldsymbol{A}}$ is a low-dimensional linear model of the dynamical system on POD coordinates:

$$\widetilde{\boldsymbol{x}}_{k+1} = \widetilde{\boldsymbol{A}}\widetilde{\boldsymbol{x}}_{k} \tag{4}$$

   It is possible to reconstruct the high dimensional state using these coordinates $\boldsymbol{x}_{k} = \boldsymbol{U}\widetilde{\boldsymbol{x}}_{k}$.
3. Compute the eigen-decomposition of $\widetilde{\boldsymbol{A}}$:

$$\widetilde{\boldsymbol{A}}\boldsymbol{W} = \boldsymbol{W\Lambda} \tag{5}$$

   where the columns of $\boldsymbol{W}$ are eigenvectors and $\boldsymbol{\Lambda}$ is a diagonal matrix containing the corresponding eigenvalues λ_{k}.
4. We may then reconstruct the eigen-decompoisition of $\boldsymbol{A}$ from $\boldsymbol{W}$ and $\boldsymbol{\Lambda}$. The eigenvalues of $\boldsymbol{A}$ are given by $\boldsymbol{\Lambda}$ and the eigenvectors of $\boldsymbol{A}$ (DMD modes) are given by columns of $\boldsymbol{\Phi}$ defined as:

$$\boldsymbol{\Phi} = \boldsymbol{X'V\Sigma}^{-1}\boldsymbol{W} \tag{6}$$

Moreover, the modes $\boldsymbol{\Phi} = \boldsymbol{UW}$ are *projected DMD modes* whereas the modes in (6) are *exact DMD modes*. The exact modes may be used to find a dynamic mode associated with the zero eigenvalue and with the condition$\phi = \boldsymbol{X'V\Sigma}^{-1} \neq 0$, but in all other cases $\phi = Uw$ should be used. It follows that the future approximation for all times is given by

$$x(t) \approx \Sigma_{k=1}^{r}\boldsymbol{\phi}_{k}\exp(\omega_{k}t)b_{k} = \boldsymbol{\Phi}\exp(\boldsymbol{\Omega}t)\,\boldsymbol{b} \tag{7}$$

where b_k is the initial amplitude of each mode, $\boldsymbol{\Phi}$ is the matrix whose columns are the DMD eigenvectors $\boldsymbol{\phi}_k$, and $\boldsymbol{\Omega} = \text{diag}(\omega)$ is is a diagonal matrix whose entries are the eigenvalues ω_k (Kutz, 10-11).

For our purposes, we calculate the DMD's approximate low-rank reconstruction according to

$$\boldsymbol{X}_{DMD}^{Low-rank} = b_p \varphi_p e^{\omega_p \boldsymbol{t}} \tag{8}$$

Because with DMD:

$$\boldsymbol{X} = \boldsymbol{X}_{DMD}^{Low-Rank} + \boldsymbol{X}_{DMD}^{Sparse} \tag{9}$$

Then the DMD's approximate sparse reconstruction is

$$\boldsymbol{X}_{DMD}^{Sparse} = \Sigma_{j \neq p} b_j \varphi_j e^{\omega_j \boldsymbol{t}} \tag{10}$$

Can be calculated with real-value elements as follows

$$\boldsymbol{X}_{DMD}^{Sparse} = \mathbf{X} - |\boldsymbol{X}_{DMD}^{Low-Rank}| \tag{11}$$

Where $|\cdot|$ is the modulus of each element in the matrix. This may result in $\boldsymbol{X}_{DMD}^{Sparse}$ having negative values in some elements, which would not make sense (negative pixel intensities). These residual negative values can be put into a $n \times m$ matrix $\boldsymbol{R}$ and then added back into $\boldsymbol{X}_{DMD}^{Low-Rank}$

$$\boldsymbol{X}_{DMD}^{Low-Rank} \leftarrow \boldsymbol{R} + |\boldsymbol{X}_{DMD}^{Low-Rank}| \tag{12}$$

$$\boldsymbol{X}_{DMD}^{Sparse} \leftarrow \boldsymbol{X}_{DMD}^{Sparse} - \boldsymbol{R} \tag{13}$$

Thus, we account for the magnitudes of the complex values from the DMD reconstruction while maintaining the constraints set forth in (9) so that none of the pixel intensities are below zero and ensuring that the approximate low-rank and sparse DMD reconstructions are real valued.

3. Algorithm Implementation and Development

As all three videos are produced by the same device, of the same resolution, the procedure for each is the same, and is as follows:

1. Load the video, convert each frame to grayscale for computational efficiency.
2. Reshape the video frames and downsize to 480×270. Then reshape back into column vectors.
3. Take the SVD of the video data to get a low-rank representation of the video through a few modes.
4. Truncate the results of the SVD to contain the first few columns (corresponding to the number of modes) of U and V as well as the values in the Σ matrix. This gives us the low rank matrix in (8). Where Ω and Φ (matrices representing $e^{\omega_j \boldsymbol{t}}$ and φ_j) are constructed from the eigen-decomposition of the Koopman operator A.

5. Fourier modes are calculated by eigen-decomposition on a matrix $\tilde{\Sigma}$ constructed from the SVD of the first snapshot much like the procedure defined in (2) and (3). Resulting in

$$\tilde{\boldsymbol{\Sigma}} \approx \boldsymbol{U} * \boldsymbol{X}_2^m \boldsymbol{V} \boldsymbol{\Sigma}^{-1} \tag{14}$$

and after performing the eigen-decomposition on $\tilde{\boldsymbol{\Sigma}}$, we approximate the j^{th} eigenvalue with

$$\tilde{\boldsymbol{\Sigma}} \boldsymbol{w}_j = \mu_j \boldsymbol{w}_j \tag{15}$$
$$\boldsymbol{\varphi}_j = \boldsymbol{U} \boldsymbol{w}_j$$

which are converted to Fourier modes by

$$\omega_j = \ln \frac{(\mu_j)}{\Delta t} \tag{16}$$

6. We then use the background video ($\boldsymbol{X}_{DMD}^{Low-Rank}$) approximated by these modes and we follow the procedure in (11) and subtract it from our original video to get $\boldsymbol{X}_{DMD}^{Sparse}$, or our foreground video.
7. Finally, we follow the procedures outlined in (12) and (13) to ensure there are no negative pixel intensities.

4. Computational Results

Video 1 – cars on a street

Using one mode, we applied our algorithm to a basic video of cars on a street. The intuition was that the video would be straightforward to handle given a decently static

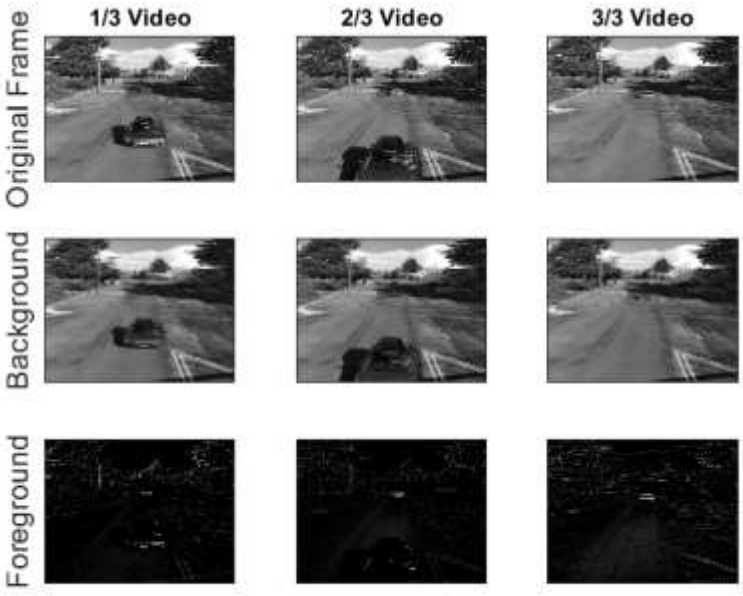

Figure 1 - Video 1, cars moving down a street

Background of the street. However, in including the sides of the road with moving details, we see that the background subtraction fails to properly account for the entirety of the black car moving in the foreground. Rather, the subtraction removes all the white colored details such as the details on that car, as well as the white background from the video leaving the husk of the original car intact in each frame. Here, the background subtraction struggles because of so much detail.

Video 2 - a hand moving across a static background.

Again, we see that some of the background details gets mistaken as part of the foreground. Whether the camera was shaking or there was a breeze, much of this detail is captured as part of the foreground. However, we do see that, perhaps because this video was longer, and perhaps because the object was of one color and texture, the object seems to fade in the frame of the background. Here our algorithm performs slightly better. Although, not perfectly.

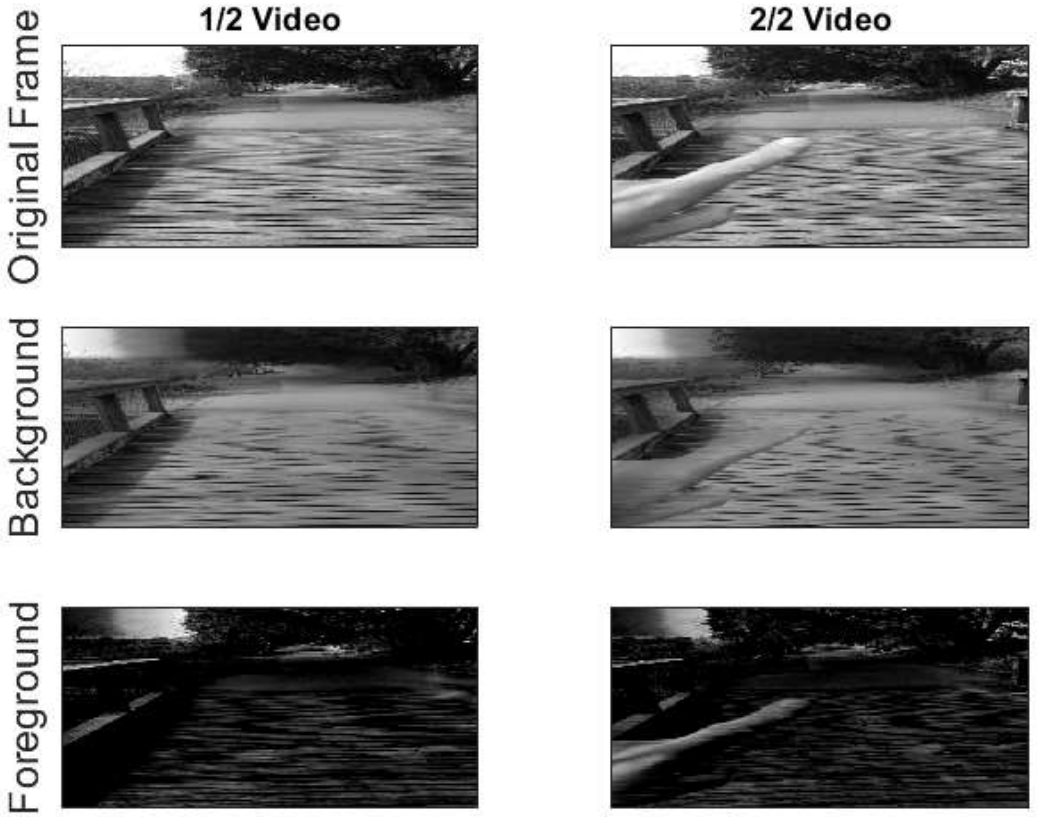

Figure 2 - Video 2 background subtraction

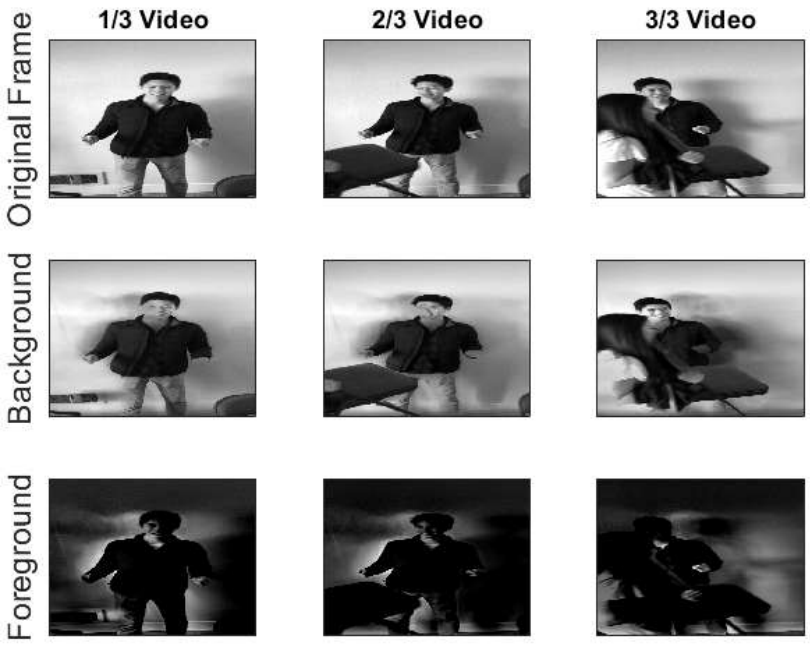

Figure 3 - Video 3, more moving parts

Video 3 – one person moving in front of a static background with another moving in front of him.

We see here that both people are considered as part of the foreground. Although the person moving the chair is mostly left out of the background frames, the person dancing in the background remains there for the most part as he never leaves the frame. What we end up seeing are a blurring of his features and especially his legs in the background frames. Here, our algorithm also did not perform perfectly, but there were interesting results regarding multiple items in the foreground, and many layers of the foreground.

5. Summary and Conclusions

We find that a video can be separated into foreground and background using DMD and Principal Component Analysis. By taking the low-rank representation of the video data, we can perform this separation. However, what we find is that this particular implantation of the algorithm does not work perfectly due to noise. With many moving parts, and with much noise, the algorithm can mistake background and foreground details. Due to a shaking camera, wind causing trees to move, etc. many details in the background registered as details in the foreground. However, With all still, DMD can be very powerful in separating foreground and background as detailed with video 2, and with the white car in video 1. It is possible that if we adjust for noise, DMD can be applied to greater affect.

Appendix A

Command	Implementation
VideoReader hasFrame readFrame	VideoReader reads in a videofile. hasFrame returns true while there are still frames to be read through. readFrame reads a frame from a given video that is read in.
rbg2gray	Used to flatten the video frame from RGB into grayscale. Makes the script computationally lighter.
Reshape	Used to reshape the given matrix to different dimensions. Handled turning images into column vectors.
diag	"diag" gives the diagonal of a given matrix.
imagesc, colormap	Used to plot the frames, original, low rank, and sparse.
svd	Used to get the singular value decomposition of the given matrix. Used to get a low rank approximation of the video frames.
imresize	Used to resize the images down for faster computation
log	Gets the log of the input value.
floor	Rounds to the floor value.
mod	Modulus to account for the iteration.
eig	Gets the eigenvalues and vectors of the matrix input.

Appendix B – MATLAB Code

```
clear all;
close all;
clc;

cd('C:\Users\timot\Desktop\amath 482\hw5');

vid = [];
v = VideoReader('IMG_3803.MOV');
while hasFrame(v)
    fr = readFrame(v);
    fr = rgb2gray(fr);
    fr = reshape (fr, [], 1);
    vid = [vid, fr];
end

[xx,frs] = size(vid);

vid = reshape(vid, 1920,1080,frs);
vid = reshape(vid, 1920*1080,frs);
vid = double(vid);
% vid = imresize(vid,.25);
% vid = double(vid);
% vid = reshape(vid, [480*270,234]);
v1 = vid(:,1:end-1);
v2 = vid(:,2:end);
[U, Sigma, V] = svd(v1, 'econ');

r=1; % # of modes
Sr = Sigma(1:r, 1:r); Ur = U(:, 1:r); Vr = V(:, 1:r);
Stilde = Ur'*v2*Vr*diag(1./diag(Sr));
[eV, D] = eig(Stilde);
mu = diag(D); o = log(mu); Phi = v2*Vr/Sr*eV;

y0 = Phi\vid(:,1);
v_modes = zeros(r,length(v1(1,:)));
for i = 1:length(v1(1,:))
    v_modes(:,i) = (y0.*exp(o*i));
end
x_dmd = Phi*v_modes; x_dmd = abs(x_dmd); x_sparse = v1 - x_dmd;

rmat = x_sparse.*(x_sparse < 0);
x_dmd = rmat + abs(x_dmd);
x_sparse = x_sparse - rmat;

figure(1)

for i = 1:9
    label = '';
    if i == 1
        label = 'Original fr';
    end
    if i == 4
        label = 'Background';
    end
    if i == 7
```

```
        label = 'Foreground';
    end
    subplot(3,3,i)
    vidtype = floor((i-1)/3);
    tf = mod(i,3);
    if tf == 0
        tf = 3;
    end
    if vidtype == 0
        temp = vid(:,tf*15);
    elseif vidtype == 1
        temp = x_dmd(:,tf*15);
    elseif vidtype == 2
        temp = x_sparse(:,tf*15);
    end
    temp = reshape(temp, 1920, 1080);
    imagesc(temp);
    colormap(gray);
    ylabel(label, 'Fontsize',14)
    if i < 4
        title(strcat(num2str(i),'/3 vid'))
    end
    set(gca,'XTick',[], 'YTick', [])
end

%% vid 2
clear all; close all; clc


vid = [];
v = VideoReader('IMG_3789.MOV');
while hasFrame(v)
    fr = readFrame(v);
    fr = rgb2gray(fr);
    fr = reshape (fr, [], 1);
    vid = [vid, fr];
end
vid = reshape(vid, [1920,1080,234]);
vid = imresize(vid,.25);
vid = double(vid);
vid = reshape(vid, [480*270,234]);
v1 = vid(:,1:end-1);
v2 = vid(:,2:end);
[U, Sigma, V] = svd(v1, 'econ');

r=1;
Sr = Sigma(1:r, 1:r); Ur = U(:, 1:r); Vr = V(:, 1:r);
Stilde = Ur'*v2*Vr*diag(1./diag(Sr));
[eV, D] = eig(Stilde);
mu = diag(D); o = log(mu); Phi = v2*Vr/Sr*eV;

y0 = Phi\vid(:,1);
v_modes = zeros(r,length(v1(1,:)));
for i = 1:length(v1(1,:))
    v_modes(:,i) = (y0.*exp(o*i));
end
x_dmd = Phi*v_modes; x_dmd = abs(x_dmd); x_sparse = v1 - x_dmd;
```

```
rmat = x_sparse.*(x_sparse < 0);
x_dmd = rmat + abs(x_dmd);
x_sparse = x_sparse - rmat;

figure(2)

for i = 1:9
    label = '';
    if i == 1
        label = 'Original fr';
    end
    if i == 4
        label = 'Background';
    end
    if i == 7
        label = 'Foreground';
    end
    subplot(3,3,i)
    vidtype = floor((i-1)/3);
    tf = mod(i,3);
    if tf == 0
        tf = 3;
    end
    if vidtype == 0
        temp = vid(:,tf*15);
    elseif vidtype == 1
        temp = x_dmd(:,tf*15);
    elseif vidtype == 2
        temp = x_sparse(:,tf*15);
    end
    temp = reshape(temp, 480, 270);
    imagesc(temp);
    colormap(gray);
    ylabel(label, 'Fontsize',14)
    if i < 4
        title(strcat(num2str(i),'/3 vid'))
    end
    set(gca,'XTick',[], 'YTick', [])
end

%% vid 3
clear all; close all; clc;

limit = 160; % limit the frames read in
vid = [];
v = VideoReader('IMG_3804.MOV');
count = 0;
while (hasFrame(v) && count < limit)
    count = count + 1;
    fr = readFrame(v);
    fr = rgb2gray(fr);
    fr = reshape (fr, [], 1);
    vid = [vid, fr];
end

[throwaway,frs] = size(vid);
```

```
vid = reshape(vid, 1920,1080,frs);
vid = imresize(vid,.25); vid = double(vid); vid = reshape(vid, 480*270,frs);
v1 = vid(:,1:end-1); v2 = vid(:,2:end);
[U, Sigma, V] = svd(v1, 'econ');

r=1;
Sr = Sigma(1:r, 1:r); Ur = U(:, 1:r); Vr = V(:, 1:r);
Stilde = Ur'*v2*Vr*diag(1./diag(Sr));
[eV, D] = eig(Stilde);
mu = diag(D); o = log(mu); Phi = v2*Vr/Sr*eV;

y0 = Phi\vid(:,1);
v_modes = zeros(r,length(v1(1,:)));
for i = 1:length(v1(1,:))
    v_modes(:,i) = (y0.*exp(o*i));
end
x_dmd = Phi*v_modes; x_dmd = abs(x_dmd); x_sparse = v1 - x_dmd;

rmat = x_sparse.*(x_sparse < 0);
x_dmd = rmat + abs(x_dmd);
x_sparse = x_sparse - rmat;

figure(3);

for i = 1:6
    label = '';
    if i == 1
        label = 'Original fr';
    end
    if i == 3
        label = 'Background';
    end
    if i == 5
        label = 'Foreground';
    end
    subplot(3,2,i)
    vidtype = floor((i-1)/2);
    tf = mod(i,2);
    if tf == 0
        tf = 3;
    end
    if vidtype == 0
        temp = vid(:,tf*15);
    elseif vidtype == 1
        temp = x_dmd(:,tf*15);
    elseif vidtype == 2
        temp = x_sparse(:,tf*15);
    end
    temp = reshape(temp, 480, 270);
    imagesc(temp);
    colormap(gray);
    ylabel(label, 'Fontsize',14)
    if i < 3
        title(strcat(num2str(i),'/2 vid'))
    end
    set(gca,'XTick',[], 'YTick', [])
end
```