AMATH 482 - Winter Quarter
Homework 4: Background Subtraction in Video Streams

Vicky Norman

March 7, 2018

Abstract

This project shows how Dynamic Mode Decomposition can be used to separate the background and foreground of an image by tracking the movement between frames. The algorithm theory and implementatin are explained and the results are presented and discussed for two different cases.

1 Introduction and Overview

This project explains the implementation of the Dynamic Mode Decomposition to separate the still background from the moving foreground of two different videos. The videos have the same background but feature different moving objects. Video 1 features a girl in the foreground moving back and forth, and video 2 features a car and a bus driving along the street.
The theory behind the DMD algorithm is explained in full and there is a discussion of how the algorithm is implemented in MATLAB and a presentation of the results achieved. The conclusion explains that the algorithm works best for videos with objects that move quickly in the foreground of the frame and discusses possible uses for this algorithm.

2 Theoretical Background

The Dynamic Mode Decomposition approximates low-dimensional modes of the linear, time-independent Koopman operator to estimate the potential non-linear dynamics of a system. The method relies on data being evenly spaced by Δt consisting of m samples in time with n data points at each given time. So for each $\mathbf{x}_j$ we have n data points collected at time $t_j \in \{j = 1, 2, ..., m\}$. We can form two matrices as follows,

$$\begin{aligned} \mathbf{X}_1^{m-1} &= [\mathbf{x}_1 \quad \mathbf{x}_2 \quad \dots \quad \mathbf{x}_{m-1}] \\ \mathbf{X}_2^{m} &= [\mathbf{x}_2 \quad \mathbf{x}_3 \quad \dots \quad \mathbf{x}_{m}]. \end{aligned} \tag{1}$$

The Koopman operator $\mathbf{A}$ maps data from time j to data at time $j+1$ such that $\mathbf{x}_{j+1} = \mathbf{A}\mathbf{x}_j$. In estimating the Koopman operator the DMD finds $\mathbf{A}$ that best represents the data in $\mathbf{X}_1^{m-1}$ in the following way,

$$\mathbf{X}_1^{m-1} = [\mathbf{x}_1 \quad \mathbf{A}\mathbf{x}_1 \quad \mathbf{A}^2\mathbf{x}_1 \quad \dots \quad \mathbf{A}^{m-1}\mathbf{x}_2]. \tag{2}$$

Therefore we can draw the relation $\mathbf{A}\mathbf{X}_1^{m-1} = \mathbf{X}_2^m$ where the last data point $\mathbf{x}_m$ is calculated by,

$$\mathbf{x}_m = \sum_{j=1}^{m-1} k_j \mathbf{x}_j + \mathbf{r}, \tag{3}$$

where k_j are the coefficients for the Krylov space basis vector and $\mathbf{r}$ is the residual error orthogonal to the Krylov space.

Representing $\mathbf{X}_2^m$ in terms of matrix $\mathbf{X}_1^{m-1}$ gives the result in equation 4 where $\mathbf{e}_{m-1}$ is the $(m-1)$ unit vector and * implies we take the conjugate transpose operator.

$$\mathbf{X}_2^m = \mathbf{X}_1^{m-1}\mathbf{S} + \mathbf{r}\cdot\mathbf{e}_{m-1}^*, \quad \mathbf{S} = \begin{vmatrix} 0 & & \dots & 0 & k_1 \\ 1 & 0 & & \dots & k_2 \\ 0 & 1 & \ddots & 0 & \vdots \\ \vdots & & \ddots & 0 & k_{m-2} \\ 0 & & \dots & 1 & k_{m-1} \end{vmatrix}_{n\times(m-1)} . \tag{4}$$

To reduce the dimensionality of the system we can employ the SVD method used in previous homeworks. The matrix $\mathbf{X}_1^{m-1}$ can be represented by the SVD in the form $\mathbf{U\Sigma V}^*$ where $\mathbf{U}\in\mathbb{C}^{n\times l}$, $\mathbf{V}\in\mathbb{C}^{(m-1)\times l}$ are unitary and $\mathbf{\Sigma}\in\mathbb{C}^{l\times l}$ is diagonal. The dimension l is chosen to truncate the matrix and reduce the rank of $\mathbf{X}_1^{m-1}$.

If we assume the residual error is extremely small, i.e. $||\mathbf{r}|| << 1$ we can draw an important conclusion,

$$\begin{aligned} \mathbf{X}_2 &\approx \mathbf{X}_1^{m-1}\mathbf{S} \\ \mathbf{X}_2 &\approx \mathbf{U\Sigma V}^*\mathbf{S} \\ \mathbf{S} &\approx \mathbf{V\Sigma}^{-1}\mathbf{U}^*\mathbf{X}_2^m. \end{aligned} \tag{5}$$

In this way we can formulate $\tilde{\mathbf{S}}$ using the similarity transform $\mathbf{U\Sigma}^{-1}$ as $\tilde{\mathbf{S}} \approx \mathbf{U}^*\mathbf{X}_2^m\mathbf{V\Sigma}^{-1}$, this is mathematically similar to $\mathbf{S}$.

Since $\mathbf{X}_2 \approx \mathbf{X}_1^{m-1}\mathbf{S}$ some of the eigenvalues of the matrix $\mathbf{S}$ approximate the eigenvalues of the Koopman operator $\mathbf{A}$. Because of this relation we can use $\tilde{\mathbf{S}}$ to determine the eigenvalues, therefore

$$\begin{aligned} \mathbf{AU} &\approx \mathbf{U}\tilde{\mathbf{S}} = \mathbf{UW\Omega W}^{-1} \\ \mathbf{A}(\mathbf{UW}) &\approx (\mathbf{UW})\mathbf{\Omega} \\ \mathbf{A\Phi} &\approx \mathbf{\Phi\Omega} \end{aligned} \tag{6}$$

from the eigendecomposition of $\tilde{\mathbf{S}}$. Solving the eigenvalue problem $\tilde{\mathbf{S}}\mathbf{w}_j = \mu_j\mathbf{w}_j$ where $j = 1, 2, \dots, l$ gives us the DMD eigenvalues and eigenvectors of $\tilde{\mathbf{S}}$. The jth DMD basis $\boldsymbol{\varphi}_j = \mathbf{Uw}_j$ make up the columns of matrix Φ.

Now to reconstruct the data using DMD we have the relation $\mathbf{x}_{DMD}(t) = \mathbf{A}^t\mathbf{x}_1$ for any time t after the initial data vector $\mathbf{x}_1$ collected at $t = 0$. Using the approximate eigenvalue decomposition of the Koopman operator produces the final useful equation,

$$\mathbf{x}_{DMD}(t) = \sum_{j=1}^{l} b_j\boldsymbol{\varphi}_j e^{w_j t} = \mathbf{\Phi\Omega}^t\mathbf{b}, \tag{7}$$

with

$$\mathbf{\Omega} = \begin{vmatrix} e^{w_1} & 0 & \dots & 0 \\ 0 & e^{w_2} & \dots & \vdots \\ \vdots & & \ddots & 0 \\ 0 & \dots & 0 & e^{w_l} \end{vmatrix}, \mathbf{b} \approx \mathbf{\Phi}^{-1}\mathbf{x}_1. \tag{8}$$

The vector $\mathbf{b}$ contains the initial amplitudes of the modes and at $t = 0$, where the first data point is taken the equation reduces to $\mathbf{x}_{DMD}(t_1) = \mathbf{x}_1 = \mathbf{\Phi b}$.

3 Algorithm Implementation and Development

Since the DMD algorithm relies on the data being evenly spaced in time it lends itself perfectly to video data since this is naturally spaced in time. For this purpose 'time' will be represented by frame number, i.e. t_1 represents the first frame, and so on until t_m which is the last frame of the video. Each from is represented by a vector of length

n since there are n pixels per frame. Using equation 7 we can reconstruct each frame of the video with the DMD algorithm, validity of the reconstruction depends on how well the video meets the assumptions and criteria of the DMD algorithm.

The diagonal matrix $\mathbf{\Omega}^t$ dictates how each frame changes over time, this is used to reconstruct the subsequent frames. Any frame that does not change over time, or changes very slowly will have an associated Fourier mode (w_j) that is located near the origin and almost equal to zero. This is the key principle which allows us to separate the background (still) from the foreground (moving) in a video clip. If we assume that $||w_p|| \approx 0$ where $p \in \{1, 2, \ldots, l\}$ and $||w_j|| \forall j \neq p$ is bounded away from zero then the DMD can be split into two sections representing the background and foreground of the video.

$$\mathbf{x}_{DMD}(t) = \underbrace{b_p \boldsymbol{\varphi}_p e^{w_p \boldsymbol{t}}}_{\text{background}} + \underbrace{\sum_{j \neq p} b_j \boldsymbol{\varphi}_j e^{w_j \mathbf{t}}}_{\text{foreground}} \tag{9}$$

Implementing this algorithm in MATLAB means we can easily perform an SVD and matrix manipulation to find the DMD. The stages of the algorithm are as follows:

- Reshape data matrix $\mathbf{X}$ so that each column represents one frame from the video,
- Create matrices $\mathbf{X}_1$ and $\mathbf{X}_2$,
- Compute the SVD of $\mathbf{X}_1$,
- Truncate the matrices $\mathbf{U}, \mathbf{S}$ and $\mathbf{V}$ to length l,
- Compute $\tilde{\mathbf{S}} = \mathbf{U}^* \mathbf{X2V\Sigma}^{-1}$ and DMD modes $\mathbf{\Phi}, \Omega$,
- Produce final $\mathbf{X}_{DMD}$ and separate background from foreground as in equation 9.

Since some of the values in the background reconstruction of the DMD have complex values we take the the absolute value of this matrix this creates some residual error as the complex part of the reconstruction is lost. To combat this we find the difference between $\mathbf{X}_{DMD}$ and the absolute value of this. We then add the absolute value of this back onto the foreground part of the DMD. The algorithm was run on two separate videos and the results are presented below.

4 Computational Results

After implementing the DMD algorithm in MATLAB the code was tested with two different video clips. The results produced are presented as follows; the plot of the diagonal values of Σ highlight how many modes are important in the DMD reconstruction, then three frames of each video are presented in three different forms for analysis. First the original frame is shown, then the DMD reconstruction of the frame which should contain just the background picture from the frame, lastly the original frame minus the background is presented to show how the movement in the video can be isolated. //

Figures 1 and 2 show the magnitude of the singular values for videos 1 and 2. From these figures we can see that both videos have one mode that contains roughly 35-40% of the data, meaning that it could be possible to reconstruct the data to a degree from just one mode. However the accuracy from this is unlikely to be as good as we desire and since the size of the videos are relatively small it will not take too much computational power to include more of the modes. In the following results the largest 20 modes have been used for the reconstruction.

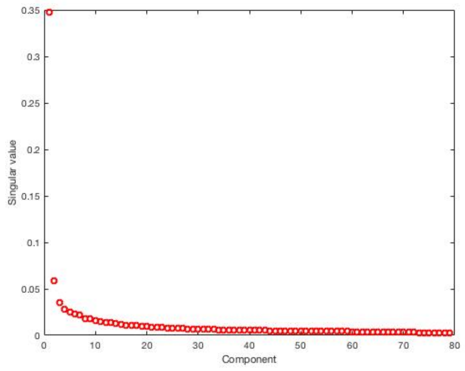

Figure 1: Magnitude of singular values for video 1.

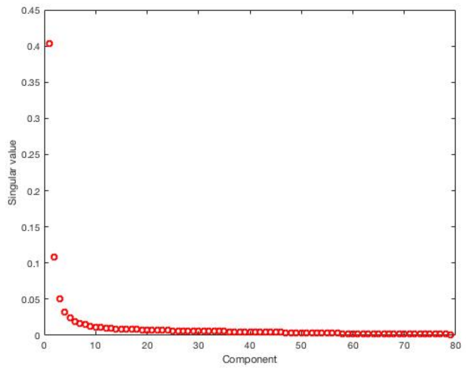

Figure 2: Magnitude of singular values for video 2.

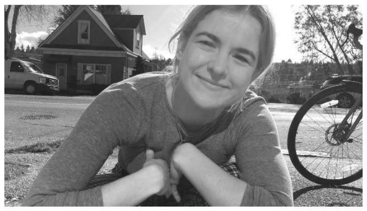

Figure 3: Video 1, frame 5 original.

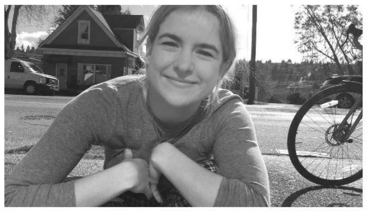

Figure 4: Video 1, frame 10 original.

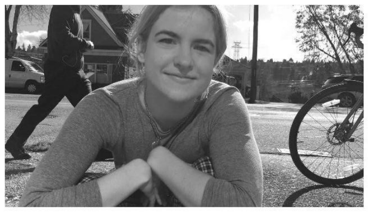

Figure 5: Video 1, frame 50 original.

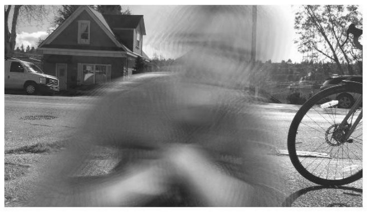

Figure 6: Video 1, frame 5 background.

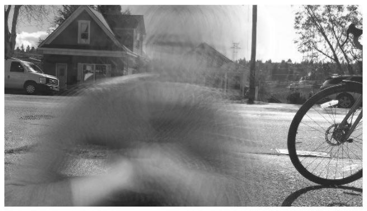

Figure 7: Video 1, frame 10 background.

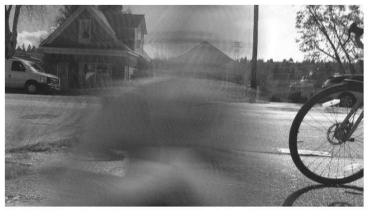

Figure 8: Video 1, frame 50 background.

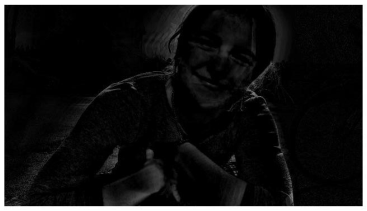

Figure 9: Video 1, frame 5 movement.

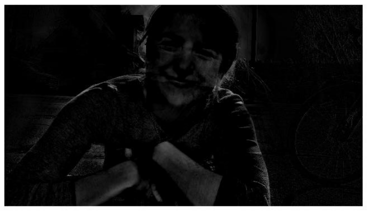

Figure 10: Video 1, frame 10 movement.

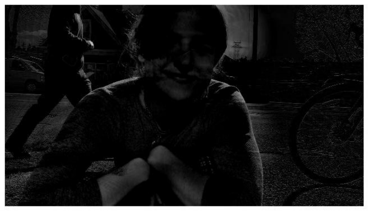

Figure 11: Video 1, frame 50 movement.

Figures 3-5 show the original frames from video 1. These show that the girl in the foreground is moving from side to side and there is also a man walking in the background in figure 5. After performing the DMD algorithm on this data one would expect to be able to isolate the movement in the video so that the only visible part of the image is the moving figure(s). The $\mathbf{X}_{DMD}$ matrix should contain just the background (still) part of each frame. Figures 6-8 show frames 5,10 and 50 of the reconstructed DMD, here we can see that although blurred the figure in the foreground is still present. This is most likely because some of the movement is too slow to be picked up by the algorithm. The results of figures 9-ref50 do however show just the outline of the moving figures. Notice especially the man in the figures from frame 50 is not present in the background image (8) but his movement can be seen in figure (11).

The second video features a white car driving along a road, followed by a bus driving in the opposite direction. The moving objects have been circled in figures 12-14. Again, figures 15-17 show the background, or stationary part of each frame, in all of these frames we can see that the vehicles are not present, there is a slight dark area where the bus is in figure 17. Looking at just the movement in figures 1820 we can see the movement of the car is highlighted and the area where the bus has driven through.

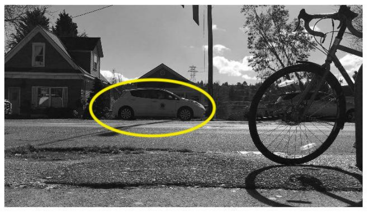
Figure 12: Video 2, frame 5 original.

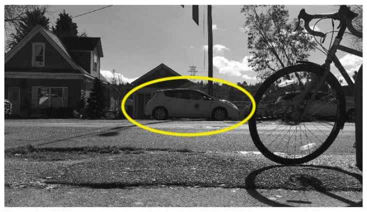
Figure 13: Video 2, frame 10 original.

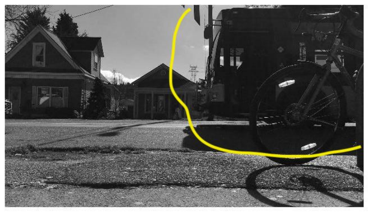
Figure 14: Video 2, frame 50 original.

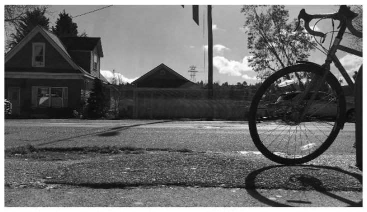
Figure 15: Video 2, frame 5 background.

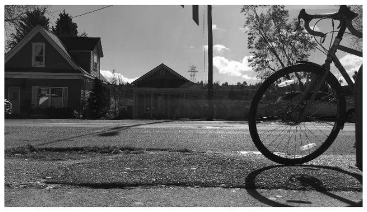
Figure 16: Video 2, frame 10 background.

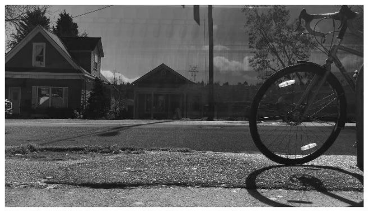
Figure 17: Video 2, frame 50 background.

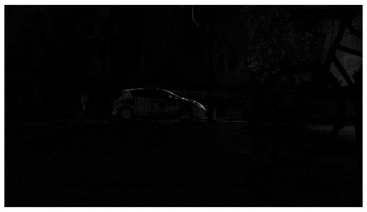
Figure 18: Video 2, frame 5 movement.

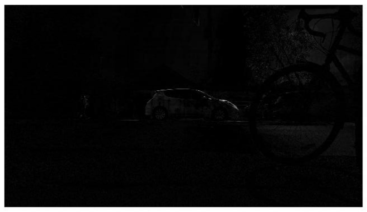
Figure 19: Video 2, frame 10 movement.

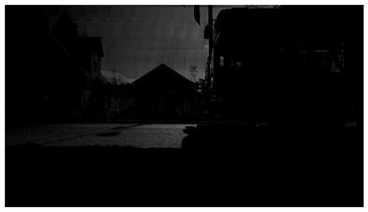
Figure 20: Video 2, frame 50 movement.

5 Summary and Conclusions

The results of this process show that the DMD algorithm is fairly effective in isolating the moving components of a video. The DMD algorithm produced good results for both videos, the separation of the background from the foreground worked better when the objects in the video were moving faster.

This algorithm has many different practical applications such as monitoring CCTV surveillance and removing moving images from videos that were unintentionally in shot. The algorithm is very effective and can be applied to many different videos and is computationally efficient.

References

[1] *Dynamic Mode Decomposition for Real-Time Background/Foreground Separation in Video*, J. Grosek and J. Nathan Kutz

[2] *Dynamic Mode Decomposition: Data-Driven Modeling of Complex Systems*, J. Nathan Kutz

[3] *https://www.mathworks.com/*

6 Appendix

A

- [U,S,V] = svd(A,'econ') produces an economy-size decomposition of m-by-n matrix A.[3]
- B = reshape(A,sz) reshapes A using the size vector, sz, to define size(B). For example, reshape(A,[2,3]) reshapes A into a 2-by-3 matrix. sz must contain at least 2 elements, and prod(sz) must be the same as numel(A).[3]
- Y = uint8(X) converts the values in X to type uint8. Values outside the range [0,28-1] map to the nearest endpoint. [3]
- Y = abs(X) returns the absolute value of each element in array X. If X is complex, abs(X) returns the complex magnitude.[3]

B

Appendix B MATLAB codes.
Main code to run algorithm on files

```
v= VideoReader('woozo.MOV');
v2= VideoReader('vid2.MOV');
vid1 = read (v) ;
% vid1 = read (v2) ;
[n m h1 h2] = size(vid1);
frames = 80;

%%

X = [];
for i = 1:frames
    x = double(reshape(vid1(:,:,1,i),n*m,1));
    X= [X x];
end

%%

X1=X(:, 1:end-1);
X2=X(:, 2:end);
r = 20;
dt = 1;


%%
size(X)
[Phi,omega,lambda,b,Xdmd, S, time_dynamics] = DMD(X1,X2,r,dt);

%%

plot(diag(S)/sum(diag(S)), 'ro','Linewidth',[2]);
xlabel('Component')
ylabel('Singular value')

%%

X_lr = uint8(abs(Xdmd));

r = Xdmd-abs(Xdmd);
```

```
%%

X_s = uint8(X1 - abs(Xdmd) -r);

%%

for i = 1:frames-1
    imshow((reshape(X_s(:,i),n, m))')
    drawnow
end
```

DMD function algorithm adapted from Dynamic Mode Decomposition: Data-Driven Modeling of Complex Systems[2].

```
function [Phi,omega,lambda,b,Xdmd, S, time_dynamics] = DMD(X1,X2,r,dt, time_dynamics)
% function [Phi,omega,lambda,b,Xdmd] = DMD(X1,X2,r,dt)
% Computes the Dynamic Mode Decomposition of X1, X2
%
% INPUTS:
% X1 = X, data matrix
% X2 = X', shifted data matrix
% Columns of X1 and X2 are state snapshots
% r = target rank of SVD
% dt = time step advancing X1 to X2 (X to X')
%
% OUTPUTS:
% Phi, the DMD modes
% omega, the continuous-time DMD eigenvalues
% lambda, the discrete-time DMD eigenvalues
% b, a vector of magnitudes of modes Phi
% Xdmd, the data matrix reconstrcted by Phi, omega, b
%% DMD
[U, S, V] = svd(X1, 'econ');
r = min(r, size(U,2));
Ur = U(:, 1:r); % truncate to rank-r
Sr = S(1:r, 1:r);
Vr = V(:, 1:r);
Atilde = Ur' * X2 * Vr / Sr; % low-rank dynamics
[Wr, D] = eig(Atilde);
Phi = X2 * Vr / Sr * Wr; % DMD modes
lambda = diag(D); % discrete-time eigenvalues
omega = log(lambda)/dt; % continuous-time eigenvalues
%% Compute DMD mode amplitudes
x1 = X1(:, 1);
b = Phi\x1;
%% DMD reconstruction
mm1 = size(X1, 2); % mm1 = m - 1
time_dynamics = zeros(r, mm1);
t = (0:mm1-1)*dt; % time vector
for iter = 1:mm1
    time_dynamics(:,iter) = (b.*exp(omega*t(iter)));
end
Xdmd = Phi * time_dynamics;


end
```