Background Subtraction in Video Streams

Yukai Yan

March 17, 2021

Abstract

In this paper, we will use the Dynamic Mode Decomposition (DMD) method on the video clips containing a foreground and background object and separate the video stream to both the foreground video and background. The DMD algorithm takes advantage of low dimensionality in the data to create a low-rank approximation of the linear mapping that iterates the best snapshots of the data, and we will verify how it performs on background subtraction.

1 Introduction and Overview

1.1 The problem

In this paper, we will perform the analysis on two given videos. The first video *Ski_drop.mov* contains snapshots of an athlete skiing downhill. The second video *monte_carlo.mov* contains videos of F1 cars passing the finishing line. The DMD spectrum of frequencies can be used to subtract background modes. The paper aims to learn the DMD method and its performance by verifying both the foreground video and the background video.

Figure 1: A snapshot of two videos

1.2 Approach the problem

Working in high-dimensional spaces can be undesirable for many reasons. The raw data can record redundant information and possibly be sparse due to their dimensionality. Therefore, we want to transform data from a high-dimensional space to a low-dimensional space but retain their properties. This process is known as Dimensional Reduction in data analysis. The primary technique for dimensional reduction used in this paper is called Dynamic Mode Decomposition. It will compute a set of modes, each of which is associated with a fixed oscillation frequency and decay/growth rate. DMD differs from principle component analysis since its modes are not orthogonal. Therefore, DMD-based representations can be less parsimonious than those generated by PCA.

1.3 Overview

We will discuss singular value decomposition, and Dynamic Mode decomposition in Section Two. In Section Three, we will develop the solution and implement the corresponding algorithms. Section Four will include the computational results derived from the approach, as the summary and conclusion will be in Section Five.

2 Theoretical Background

2.1 Singular Value Decomposition

In matrix form, the singular value decomposition of the matrix A with size $m \times n$ can be represented with:

$$A = U\Sigma V^*$$

Where U is an $m \times m$ complex unitary matrix, Σ is an $m \times n$ rectangular diagonal matrix with non-negative real numbers on the diagonal, and V is an $n \times n$ complex unitary matrix. The singular value decomposition of matrix A is connected to the eigenvalue decomposition of AA^T. To account for the $\frac{1}{n-1}$ factor, consider:

$$A = \frac{1}{\sqrt{n-1}}X$$

Then, for principle component analysis, we have:

$$C_X = \frac{1}{n-1}XX^T = AA^T = U\Sigma^2U^T$$

So, the eigenvalues of the covariance matrix are the squared of the scaled singular values. Since we used a change of basis to work in the basis of the principal components, the data in the new coordinate and its covariance can be obtained by:

$$Y = U^TX, C_Y = \Sigma^2$$

2.2 Dynamic Mode Decomposition

The Dynamic Mode Decomposition approximates the mode of Koopman Operator. The Koopman operator **A** is a linear, time-independent operator s.t.

$$x_{j+1} = Ax_j$$

Where j indicates the specific data collection time and A is the linear operator that maps the data from time t_j to t_{j+1}. The vector x_j is an N-dimensional vector of the data points collect at time j. To construct the appropriate Koopman Operator that best represents the data collected, we will consider the matrix

$$X_1^{M-1} = [x_1, x_2, x_3, \ldots, x_{M-1}]$$

Where we use the shorthand x_j to denote a snapshot of the data at time t_j. The Koopman operator allows us to rewrite this as:

$$X_1^{M-1} = [x_1, Ax_1, A^2x_1, \ldots, A^{M-2}x_1], \qquad X_2^M = AX_1^{M-1} + re_{M-1}^T$$

Where e_{M-1} is the vector with all zeros except a 1 at the (M-1)st component. The residual vector r account for the error of the final point X_M that wasn't included in Krylov basis. Note that A is unknown and our goal is to find it. If we use the SVD to write $X_1 = U\Sigma V^*$, then from above we have

$$X_2^M = AU\Sigma V^* + re_{M-1}^T$$

We are going to choose A in such a way that columns in X_2^M can be written as linear combinations of the columns of U. The residual error r must be orthogonal to the POD basis, giving that $U^* \times r = 0$. Then, from above we have:

$$U^*X_2^M = U^*AU\Sigma V^*, \qquad U^*AU = U^*X_2^MV\Sigma^{-1} = S$$

Note that S and A are related by applying a matrix on one side and its inverse on the other. If y is an eigenvector of S, then U_y is an eigenvector of A. Then

$$Sy_k = \mu_k y_k, \qquad \psi_k = Uy_k$$

ψ_k is the eigenvectors of A or the *DMD modes*, k is the rank of X_1^{M-1}. Now, if we expand in our eigenbasis:

$$X_{DMD}(t) = \sum_{k=1}^{K} b_k\psi_k e^{w_k t}$$

Where b_k is the initial amplitude of each mode. Taking $t = 0$ we have $x_1 = \Psi b$ and thus $b = \Psi^\dagger x_1$, where $\Psi^\dagger$ is the pseudoinverse of the matrix Ψ

2.3 DMD spectrum of frequencies

The DMD spectrum of frequencies can be used to subtract background modes. Assume that w_p, where $p \in \{1, 2, \ldots, \ell\}$ satisfies $||w_p|| \approx 0$, and that $||w_j|| \forall j \neq p$ is bounded away from zero. Thus,

$$X_{DMD} = b_p \varphi_p e^{w_p t} + \sum_{j \neq p} b_j \varphi_j e^{w_j t}$$

The first part is the Background Video and the second part is the Foreground Video. However, there is a problem separating the DMD terms into approximate low-rank and sparse reconstructions because real-valued outputs are desired and knowing how to handle the complex elements can make a significant difference in the accuracy of the results. Consider calculating the DMD's approximate low-rank reconstruction according to

$$X_{DMD}^{Low-Rank} = b_p \varphi_p e^{w_p t}, \qquad X = X_{DMD}^{Low-Rank} + X_{DMD}^{Sparse}$$

Then the DMD's approximate sparse reconstruction can be calculated with real-value elements:

$$X_{DMD}^{Sparse} = \sum_{j \neq p} b_j \varphi_j e^{w_j t} = X - |X_{DMD}^{Low-Rank}|$$

Where $|.|$ yields the modulus of each element within the matrix. However, this may result in $X_{DMD}^{Sparse}|$ having negative values in some of its elements, which would not make sense in terms of having negative pixel intensities. These residual negative values can be put into a $n \times m$ matrix R and then be added back into $X_{DMD}^{Low-Rank}$ as follows:

$$X_{DMD}^{Low-Rank} = R + |X_{DMD}^{Low-Rank}|$$
$$X_{DMD}^{Sparse} = X_{DMD}^{Sparse} - R$$

This way the magnitudes of the complex values from the DMD reconstruction are accounted for, while maintaining the important constraints of X

3 Algorithm Implementation and Development

Our goal is to separate the video stream to both the foreground video and a background. Using Dynamic Mode decomposition, the code can be written in three main parts:

1. clean workspace and load video into frames
2. Apply Dynamic Mode Decomposition
3. Optimize DMD spectrum of frequencies

Note that the code implemented in MatLab is included in Appendix B.

Algorithm 1: clean workspace and load video into frames

Clean work space
Load video data.
for *each frame in the video* **do**
  Convert from rgb to gray.
  Convert to double precision.
  Reshape the frame from 3d to 2d.
end
Store frames in *frames*.
Define dt as the inverse of the frame rate.
Define t as $1 : dt : videoDuration$

Convert data into 2d space allows us to perform the Dynamic Mode Decomposition. The conversion can be done from x, y, t plane to z, t plane, where z is the combined data of x, y

Algorithm 2: Apply Dynamic Mode Decomposition

Define X_1 to be $frames(:, 1 : end - 1)$.
Define X_2 to be $frames(:, 2 : end)$.
Obtain $[U, Sig, V]$ by performing Singular Value Decomposition on X_1.
Apply Low rank approximation on matrix U, Sig, V
Obtain S as stated in section 2.2
Find the eigenValue eV and the eigenvector D of S
μ is the diagonal of D, ω is the log of μ divided by dt, and Ψ is the multiplication of U and eV.

Now we have defined all terms needed to perform the DMD. Note that we can use pseudoinverse to get initial conditions b as stated in section 2.2.

Algorithm 3: Optimize DMD spectrum of frequencies

Define $X_{lowrank}$ to be the result of DMD reshaped back to 3d space.
Define X_{frames} to be the original frame - $X_{lowrank}$.
Put negative values of X_{frames} in R
Redefine $X_{lowrank}$ and X_{frames} as stated in section 2.3

The results of the background video and foreground video are included in Section 4.

4 Computational Results

4.1 Video 1: Ski drop

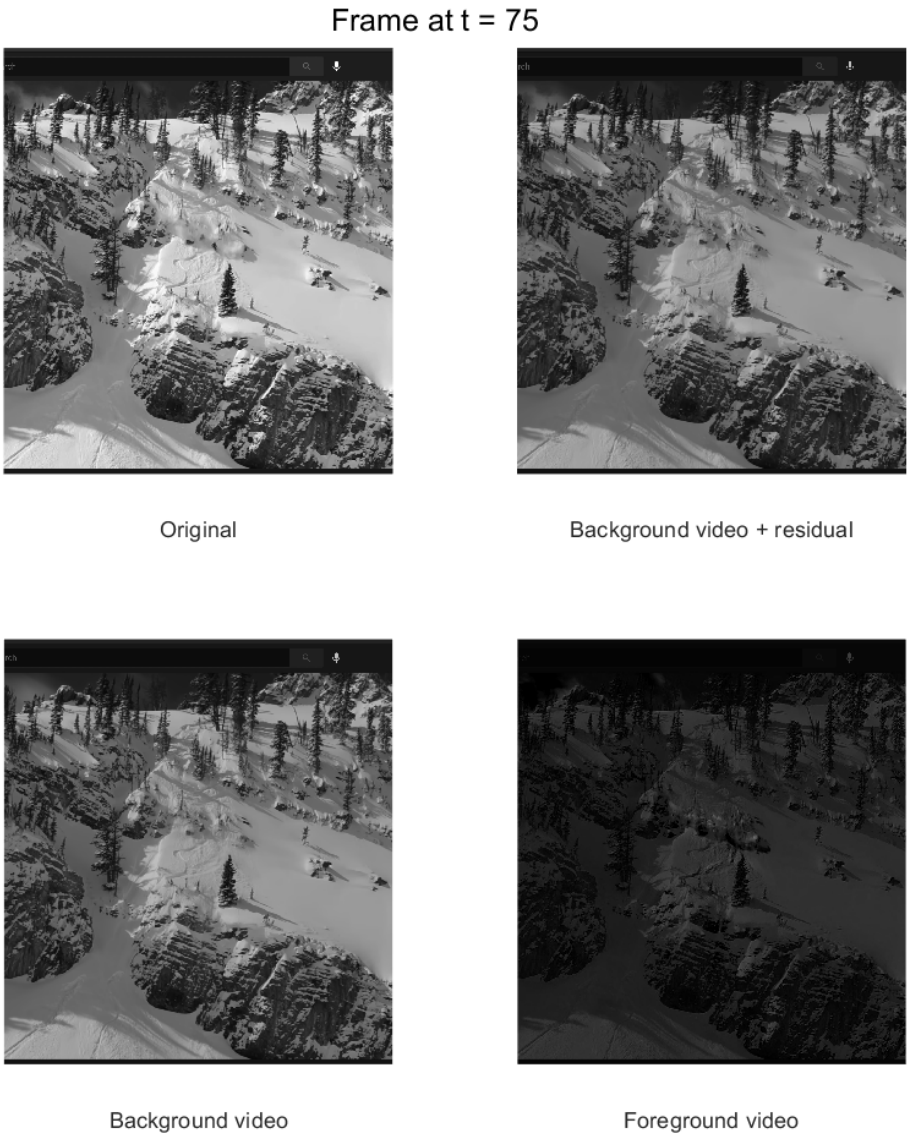

Figure 2: A snapshot of videos when $t = 75$

As you might have seen in the graph, Dynamic mode decomposition successfully separates the video. There are no athletes in the background video. DMD also removes the movement of the snow, as if no one has been there before. With residuals added back to the background video, the athlete appears again. The athlete is also barely caught by the foreground video, indicated as the dark moving spot in the figure.

4.2 Video 2: Monte Carlo

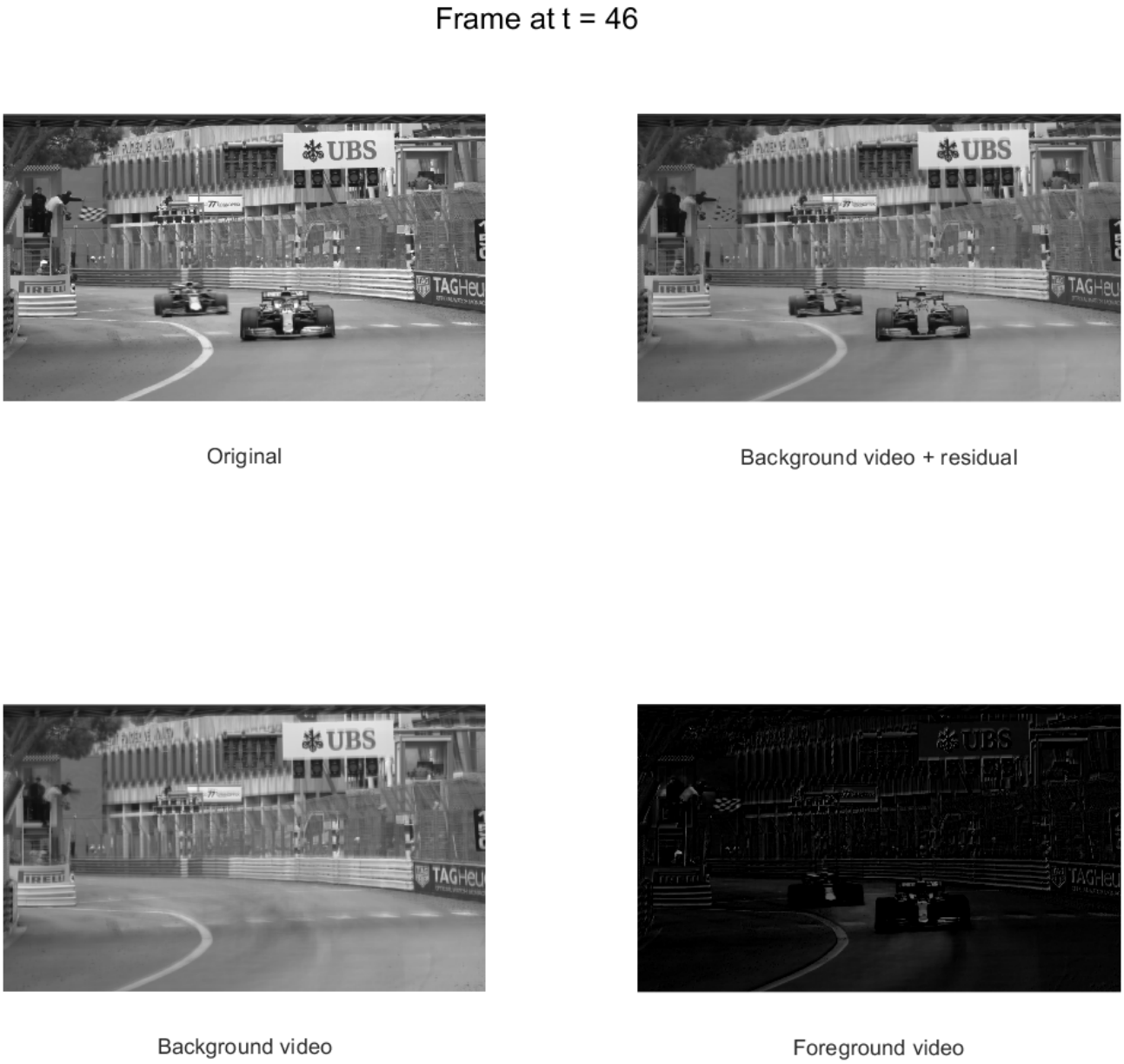

Figure 3: A snapshot of videos when $t = 45$

Like the previous video, the movement of the F1 car was captured by the foreground video. The F1 car was also removed in the background video as if no vehicle is passing by at that time. However, We can still barely see the trace of F1 cars on the runway. Note that as a side effect, the flag is also removed in the background video, as if the person is waving nothing at the finish line.

5 Summary and Conclusions

The dynamic mode decomposition aims to take advantage of low-dimensionality in experimental data without having to rely on a given set of governing equations. As demonstrated in this paper, we see that DMD allows us to capture the moving object, thus separating the video and obtaining its background. Although the moving object's trace can still be visible in the background, DMD is a powerful tool for forecasting the data and subtracting background modes. More work can be done on adjusting ranks, combine DMD with other algorithms to obtain a clear separation of the video.

Appendices

Appendix A: MatLab functions used and brief explanations

The following list includes the MatLab functions used in Appendix B. It also contains a brief implementation explanation of their functionalities. More information can be found at MatLab Documentation at MathWorks.

- *v = VideoReader(filename)*: creates object v to read video data from the file named filename.
- *video = readFrame(v)*: reads the next available video frame from the file associated with v.
- *I = rgb2gray(RGB)*: converts the truecolor image RGB to the grayscale image I.
- *I2 = im2double(I)*: converts the image I to double precision.
- *szdim = size(A,dim)*: returns the length of dimension *dim* when *dim* is a positive integer scalar.
- *B = reshape(A,sz)*: reshapes A using the size vector, sz, to define size(B).
- *[U,S,V] = svd(A,'econ')*: produces an economy-size decomposition of m-by-n matrix A
- *D = diag(v)*: returns a square diagonal matrix with the elements of vector v on the main diagonal.
- *[V,D] = eig(A)*: returns diagonal matrix D of eigenvalues and matrix V whose columns are the corresponding right eigenvectors, so that $A * V = V * D$.
- *Y = log(X)*: returns the natural logarithm ln(x) of each element in array X.
- *L = length(X)*: returns the length of the largest array dimension in X.
- *Y = exp(X)*: returns the exponential e^x for each element in array X.
- *Y = abs(X)*: returns the absolute value of each element in array X.
- *imshow(I)*: displays the grayscale image I in a figure.

Appendix B: MatLab Code

```
%% clean and load data
clear variables; close all; clc;

% v = VideoReader('monte_carlo.mov');
v = VideoReader('ski_drop.mov');

rank = 5;

dt = 1/v.Framerate;
t = 1:dt:v.Duration;

x = v.Height;
y = v.Width;
if (strcmp(v.name,'ski_drop.mov'))
    frames = zeros(x*y,length(t) - 15);
    t = t(16:end);
else
    frames = zeros(x*y,length(t));
end

index = 1;
for i = 1:length(t)
    frame = readFrame(v);
    if (strcmp(v.name,'ski_drop.mov') && i <= 15) % clean data
        continue
    end
    frame = rgb2gray(frame);
    frame = im2double(frame);

    frame = reshape(frame,x*y,1);
    frames(:,index) = frame;
    index = index + 1;
end

%% Dynamic Mode Decomposition
X1 = frames(:,1:end-1);
X2 = frames(:,2:end);
[U, Sigma, V] = svd(X1,'econ');
U = U(:,1:rank);
Sigma = Sigma(1:rank,1:rank);
V = V(:,1:rank);

S = U'*X2*V*diag(1./diag(Sigma));
[eV, D] = eig(S);  % compute eigenvalues + eigenvectors
mu = diag(D);      % extract eigenvalues
omega = log(mu)/dt;
Phi = U*eV;

y0 = Phi\X1(:,1);  % pseudoinverse to get initial conditions
u_modes = zeros(length(y0),length(t));
for iter = 1:length(t)
    u_modes(:,iter) = y0.*exp(omega*t(iter));
end
u_dmd = Phi*u_modes;

%% optimize DMD spectrum of frequencies and display data
X = reshape(frames,x,y,size(frames,2));
X_lowrank = reshape(u_dmd,x,y,size(u_dmd,2));
```

```
X_sparse = X - abs(X_lowrank);
R = X_sparse;
R(R>0) = 0;

X_lowrank_residual = R + abs(X_lowrank);
X_sparse = X_sparse - R;
figure(1)
for i = 1:size(X_lowrank,3)
    subplot(2,2,1)
    imshow(X(:,:,i));
    xlabel('Original')

    subplot(2,2,2)
    imshow(X_lowrank_residual(:,:,i));
    xlabel('Background video + residual')

    subplot(2,2,3)
    imshow(abs(X_lowrank(:,:,i)));
    xlabel('Background video')

    subplot(2,2,4)
    imshow(X_sparse(:,:,i));
    xlabel('Foreground video')

    sgtitle(['Frame at t = ', num2str(i)]);
    drawnow

end
```