Background Subtraction in Video Streams

Paul Abboud

***Abstract*—In this paper, we are tasked with separating a video into two distinct components. Specifically, we will subtract the background from the foreground to isolate frequencies and produce separate videos for the foreground and background. To do this, we implement Dynamic Mode Decomposition (DMD) which allows us to decompose the video stream into modes with fixed frequency oscillations.**

I. Introduction

In this application of Dynamic Mode Decomposition (DMD), we will compute a low-rank approximation of video streams in order to isolate the foreground from the background. DMD is especially useful in video processing because it encodes the dynamics with respect to time. This is in contrast to Principal Component Analysis (PCA) which lacks the vital time dynamics present in video streams. Using Singular Value Decomposition (SVD) in conjunction with DMD will allow us to decompose the system while still understanding how the video progresses over time.

II. Theoretical Background

To understand the issue at hand, the following will discuss the key ideas behind Singular Value Decomposition and Dynamic Mode Decomposition.

A. Singular Value Decomposition

A Singular Value Decomposition decomposes a matrix into three components such that

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

where $U \in \mathbb{R}^{m\times m}$ and $V \in \mathbb{R}^{m\times n}$ are unitary matrices and $\Sigma \in \mathbb{R}^{m\times n}$ is diagonal. U is a matrix with orthonormal columns, V is an orthonormal matrix and S has non-negative singular values in decreasing order along the diagonal. Their product $A\hat{x}$ represents an orthogonal scaling by $V^T\hat{x}$ then an axis-aligned scaling by $S(V^T\hat{x})$ and finally applying resulting coefficients in the orthonormal basis by $U(S(V^T\hat{x}))$. The result is a rank approximation of the original data separated into principal components where each component describes a portion of the system.

B. Dynamic Mode Decomposition

Dynamic Mode Decomposition is able to predict data in time without relying on a dynamical system of equations utilizing only experimental data. DMD aims to approximate the nonlinear dynamics of a system by a Koopman Operator. This operator is used to linearize and approximate high-dimensional nonlinear data. To implement DMD, we take snapshots of the data in time such that

$$X_1^{M-1} = [x_1, x_2, ..., x_{M-1}] \tag{2}$$

where x_j denotes each snapshot in time at t_j and M is the number of snapshots taken. Using the Koopman Operator

$$x_{j+1} = Ax_j \tag{3}$$

where A is the linear transformation that maps data from time t_j to t_{j+1}. Thus we can rewrite Eq.(2) as

$$X_1^{M-1} = [x_1, Ax_1, ..., A^{M-2}x_1] \tag{4}$$

which amazingly allows us to understand all the snapshots from only x_1 and the Koopman Operator (A). We can rewrite Eq.(4) into matrix form by

$$X_2^M = AX_1^{M-1} + re_{M-1}^T \tag{5}$$

where the vector r is the residual since x_M isn't accounted for in X_1^{M-1} and A applied to X_1^{M-1} maps to X_2^M. Now we can perform SVD on X_1^{M-1} such that

$$X_2^M = AU\Sigma V^T + re_{M-1}^T \tag{6}$$

Note that we can used a reduce Σ such that it contains only the first $K \geq 1$ nonzero values on the diagonal. Combined with the fact that we know K is equivalent to the rank of X_1^{M-1} we can reduce the dimension of the data given that K is small compared to N and M. We have $U \in \mathbb{C}^{N\times K}$, $V \in \mathbb{C}^{M-1\times K}$ and $\Sigma \in \mathbb{R}^{K\times K}$ where N is the number of spatial points per snapshot. Now we can assemble a matrix similar to A denoted by $\tilde{S}$, meaning $\tilde{S}$ has the same eigenvalues as A.

$$\tilde{S} = UX_2^M V\Sigma^{-1} \tag{7}$$

Now we can compute the eigenvectors of $\tilde{S}$ and then of A by

$$\tilde{S}y_k = \mu_k y_k \tag{8}$$

$$\psi = Uy_k \tag{9}$$

Expanding into the eigenbasis we have

$$x_{DMD}(t) = \sum_{k=1}^{K} b_k\psi_k e^{\omega_k t} \tag{10}$$

where b_k are the initial amplitudes of each mode and $\omega_k = \frac{ln(\mu_k)}{\Delta t}$. Δt is the difference in time step from t_j to t_{j+1}. In this application, we can represent the DMD by two modes: the background video and the foreground video. This is described by the following equation

$$X_{DMD} = b_p\phi_p e^{\omega_p t} + \sum_{j\neq p} b_j\phi_j e^{\omega_j t} \tag{11}$$

where the first term is the background video and the second term is the foreground video. We assume $\|\omega_p\| \approx 0$ and that

$\|\omega_j\| \forall j \neq p$ is not near zero. The output should be real-valued, however, each term in the sum is complex which can affect accuracy if handled incorrectly. Separating the DMD into the following

$$X = X_{DMD}^{LowRank} + X_{DMD}^{Sparse} \tag{12}$$

we have $X_{DMD}^{LowRank}$ representing the background and X_{DMD}^{Sparse} representing the foreground. To avoid inaccuracies from complex terms we can compute X_{DMD}^{Sparse} by

$$X_{DMD}^{Sparse} = X - |X_{DMD}^{LowRank}| \tag{13}$$

where $|...|$ takes the modulus of each element to avoid issues with complex elements. The result may have negative values in some entries, but negative pixel intensities are not realistic. So we can perform the following alteration

$$X_{DMD}^{LowRank} \longleftarrow R + |X_{DMD}^{LowRank}| \tag{14}$$

$$X_{DMD}^{Sparse} = X_{DMD}^{Sparse} - R \tag{15}$$

where R is an $n \times m$ matrix of negative value residuals. Now the result accounts for complex values (low-rank and sparse DMD are real-valued) while maintaining that no pixels have negative intensities.

III. Algorithm Implementation

To begin, we load in the video data and process it into a matrix to perform computations.

```
frames = VideoReader('ski_drop_low.mp4');
X = zeros(vidWidth*vidHeight, totalFrames);
for i = 1:1:totalFrames
    current_Frame2 = rgb2gray(read(frames,i));
    current_Frame = reshape(current_Frame2,vidWidth*vidHeight,1);
    X(:,i) = double(current_Frame);
end
```

We then compute the SVD on X_1^{M-1}. We can use this to analyze the necessary number of modes to sufficiently approximate the original data.

```
X1 = X(:,1:end-1);
X2 = X(:,2:end);
[U,Sigma,V] = svd(X1, 'econ');
```

Then computing $\tilde{S}$ by Eq.(7) and using Matlab to find the eigenvalues and eigenvectors of $\tilde{S}$.

```
r = 57;
U2 = U(:,1:r);
S2 = Sigma(1:r,1:r);
V2 = V(:,1:r);
S = U2'*X2*V2/S2;
```

Now we can compute the eigenvalues and eigenvectors of the Koopman Operator, A, by Eq.(9). We then select modes near zero for our low-rank DMD and calculate b_k, the initial amplitudes of each mode.

```
phi = U2*E;
mu = diag(D);
w = log(mu)/dt;
b = find(abs(w) < 1e-2);
w_b = w(b);
phi_b = phi(:,b);
```

We can now construct our low-rank DMD using Eq.(11), this is the background stream.

```
X1_n = X1(:,1);
y = phi_b \ X1_n;
M = length(t_frame);
modes = zeros(length(w_b), M);
for j = 1:M
    modes(:,j) = (y.*exp(w_b*t_frame(j)));
end
X_low = phi_b*modes;
```

Finally, we can construct the foreground video using the low-rank DMD to compute the sparse DMD by Eq.(13-15).

```
X_sparse = X - abs(X_low);
R = X_sparse.*(X_sparse < 0);
X_low = R + abs(X_low);
X_sparse = X_sparse - R;
```

We have now successfully isolated the background and foreground videos by computing the low-rank DMD and sparse DMD.

IV. Computational Results

In this application, we performed SVD on X_1^{M-1}. Figures 1 and 2 show the proportion of energy captured at each rank. For the Monte Carlo video, we found that 90% of the energy is captured in the first 99 modes and for the Skiing video, the first 57 modes captured 90% of the data. This is a great reduction of dimension which will help when computing the DMD. Then we successfully computed the low-rank and sparse DMD to separate the background and foreground video. Figure 3 shows the reconstruction at frames 100 and 200 for the Monte Carlo video. It's apparent that the application of DMD was successful in this video stream. We can see that in both frames we have almost completely eliminated the car from the background reconstruction. In the foreground, the car is completely visible, however, we can still see the road below the car and edges of the background behind the car. These inaccuracies in reconstruction are most likely a result of the residual negative pixel intensities found in R. In the skiing video, it's more difficult to see the effects of background subtraction which is illustrated in Figure 4. We can see the skier in frame 200 at the center of the image. In frame 400, the location of the skier in the foreground is more apparent

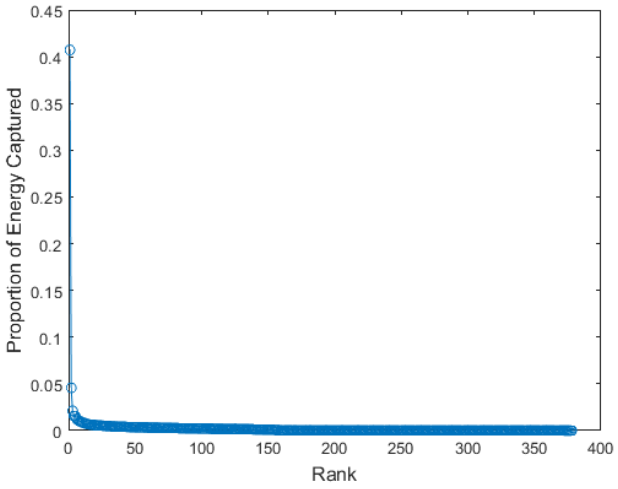

Fig. 1. Energy Captured at Each Dimension in Monte Carlo Video

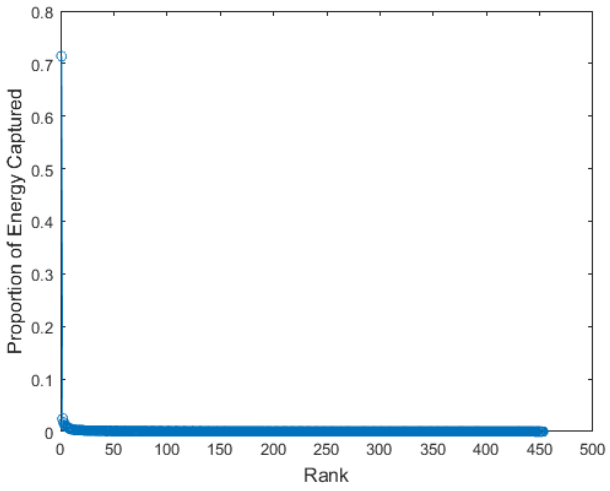

Fig. 2. Energy Captured at Each Dimension in Skiing Video

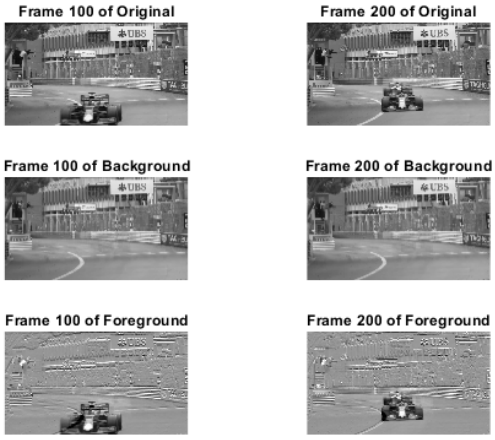

Fig. 3. Frame Reconstruction of Monte Carlo Video

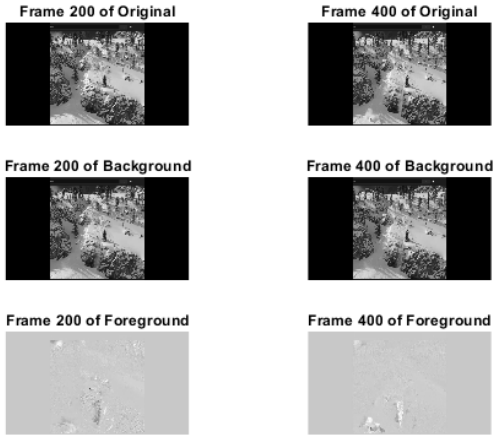

Fig. 4. Frame Reconstruction of Skiing Video

at the bottom of the image. The skier also does not appear in the background image. Overall, the application of DMD for separating the background and foreground streams was successful given the frame reconstruction in Figures 3 and 4.

V. Summary of Results

In this paper, we implemented Singular Value Decomposition and Dynamic Mode Decomposition onto two video streams. We used the SVD to find a low-rank approximation of the original data. Applying additional computations, we found a low-rank DMD which represented the background video. We then used this to compute the foreground video, or the sparse DMD. This lead to successfully isolating the foreground video from the background video which was illustrated in frame reconstructions. DMD has proven to be a powerful tool in approximating the dynamics and separating the components of video streams where behavior over time is essential to reconstruction.

Appendix A

MATLAB Functions

VideoReader(file): Creates an object the reads video data from the provided file.

rgb2gray(image): Converts an RGB image to a grayscale image by eliminating the hue and saturation information.

svd(A): Performs a singular value decomposition on the passed matrix A.

eig(A): Computes the eigenvalues and eigenvectors of the passed matrix A.

Appendix B

MATLAB Code

```
close all; clear all; clc

% frames = VideoReader('monte_carlo_low.mp4');
frames = VideoReader('ski_drop_low.mp4');
vidHeight = frames.Height;
vidWidth = frames.Width;
totalFrames = frames.NumberOfFrames;

X = zeros(vidWidth*vidHeight, totalFrames);
for i = 1:1:totalFrames
    current_Frame2 = rgb2gray(read(frames,i));
    current_Frame = reshape(current_Frame2,vidWidth*vidHeight,1);
    X(:,i) = double(current_Frame);
end

t_current = linspace(0, frames.CurrentTime, totalFrames+1);
t_frame = t_current(1:end-1);
dt = t_frame(2) - t_frame(1);

X1 = X(:,1:end-1);
X2 = X(:,2:end);
[U,Sigma,V] = svd(X1, 'econ');

figure(1)
plot(diag(Sigma)/sum(diag(Sigma)), '-o')
xlabel('Rank','Fontsize',12)
ylabel('Proportion of Energy Captured','Fontsize',12)

% r = 99;
r = 57;
U2 = U(:,1:r);
S2 = Sigma(1:r,1:r);
V2 = V(:,1:r);
S = U2'*X2*V2/S2;
[E,D] = eig(S);
phi = U2*E;

mu = diag(D);
w = log(mu)/dt;
b = find(abs(w) < 1e-2);
w_b = w(b);
phi_b = phi(:,b);

X1_n = X1(:,1);
y = phi_b \ X1_n;
M = length(t_frame);
modes = zeros(length(w_b), M);
for j = 1:M
    modes(:,j) = (y.*exp(w_b*t_frame(j)));
end

X_low = phi_b*modes;
% X_sparse = X - abs(X_low);
% R = X_sparse.*(X_sparse < 0);
% X_low = R + abs(X_low);
% X_sparse = X_sparse - R;
X_low = abs(X_low);
X_sparse = X - abs(X_low) + 200;

% capture = [100, 200];
capture = [200, 400];
figure(2)
subplot(3,1,1)
fig1 = reshape(uint8(X(:,capture(1))), vidHeight, vidWidth);
imshow(fig1)
title(['Frame ', num2str(capture(1)),' of Original'])
subplot(3,1,2)
fig2 = reshape(uint8(X_low(:,capture(1))), vidHeight, vidWidth);
imshow(fig2)
title(['Frame ', num2str(capture(1)),' of Background'])
subplot(3,1,3)
fig3 = reshape(uint8(X_sparse(:,capture(1))), vidHeight, vidWidth);
imshow(fig3)
title(['Frame ', num2str(capture(1)),' of Foreground'])

figure(3)
subplot(3,1,1)
fig1 = reshape(uint8(X(:,capture(2))), vidHeight, vidWidth);
imshow(fig1)
title(['Frame ', num2str(capture(2)),' of Original'])
subplot(3,1,2)
fig2 = reshape(uint8(X_low(:,capture(2))), vidHeight, vidWidth);
imshow(fig2)
title(['Frame ', num2str(capture(2)),' of Background'])
subplot(3,1,3)
fig3 = reshape(uint8(X_sparse(:,capture(2))), vidHeight, vidWidth);
imshow(fig3)
```

```
88 title(['Frame ', num2str(capture(2)),'
       of Foreground'])
```