AMATH 482 Homework 5:
Background Subtraction in Video Streams

Malikah Nathani
March 17th, 2021

Abstract

The objective of this paper is to examine the practical application of Dynamic Mode Decomposition. Specifically, by examining the videos of moving objects and separate the videos into its foreground and background. We compare the extracted foreground and background videos to the original to further investigate Dynamic Mode Decomposition and explore the true usefulness of this data analysis tool.

1 Introduction and Overview

Dynamic Mode Decomposition (DMD) is vast area of study as it can analyze nonlinear systems incredibly well. For this paper, we will be applying this analytical method to separate a video into its foreground and background elements. Taking advantage of low-dimensionality is incredibly powerful, and this problem can serve as a basis for more complicated issues allowing us to further understand the true power of DMD.

In this paper, we are given two videos, one of a skier going down a mountain and another of racecars driving in Monte Carlo. First, we are tasked to create a dataset of this video and then use DMD to break apart the video to create two videos, one of the background and one of the foreground.

In Section 2, I will go about explaining the DMD in more detail while section 3 will go over the implementation of this method and the algorithm used to solve the problem. Sections 4 and 5 will go over the computed results and my conclusions respectively. Finally, I will also present the MATLAB functions used and provide my code.

2 Theoretical Background

2.1 Dynamic Mode Decomposition (DMD)

We first construct the Koopman Operator (explained in 2.2) that best explains the data we have collected, we consider the matrix

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

We use the shorthand x_j to denote the data at time *j*. We can also rewrite this where the columns are formed by applying the powers of *A* to the vector x_1, this forms the basis for the Krylov subspace.

$$X_1^{M-1} = [x_1 \; Ax_1 \; \dots \; A^{M-2}x_1\,]$$

We rewrite the above matrix to receive

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

Where e_{M-1} is the vector with all zeros except at the (M-1) component. Then, we use the SVD to write

$$X_1^{M-1} = U\Sigma V^*$$

To get

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

It is important to note that we are going to choose *A* in a way such that the columns of X_2^M can we written as linear combinations of the columns of *U*. We then want to create the matrix $\tilde{S}$ such that

$$\tilde{S} = U^* X_2^M V \Sigma^{-1}$$

This is the matrix we want to find the eigenvectors and eigenvalues for to be able to successfully compute the DMD. This also gives us the eigenvectors of *A*, called the DMD modes. We expand in our eigenbasis to describe continual multiplications by A as

$$x_{DMD}(t) = \sum_{k=1}^{K} b_k \psi_k e^{\omega_k t} = \Psi diag(e^{\omega_k t})b$$

Where *K* is the rank of X_1^{M-1}, b_k is the initial amplitude of each mode, and the matrix Ψ contains the eigenvectors ψ_k and its columns. ω_k represents the ln of the eigenvalues of $\tilde{S}$.

2.2 The Koopman Operator

DMD is successful by approximating the modes of the Koopman Operator. This operator is linear and time-independent such that

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

The *j* indicates the specific data collection at that time and *A* is the linear operator that is mapping the data from time *j* to *j+1*. The vector x_j is N-dimensional containing the data points collected at time *j*. Although the system may not be linear, we use linear mapping for each time step.

3 Algorithm Implementation and Development

This section details the implementations of the methods above and any other steps taken to solve this problem in MATLAB. For the actual code used, please see Appendix B. An important note is that I will go over the implementation for one video, as the process is simply replicated for the other case.

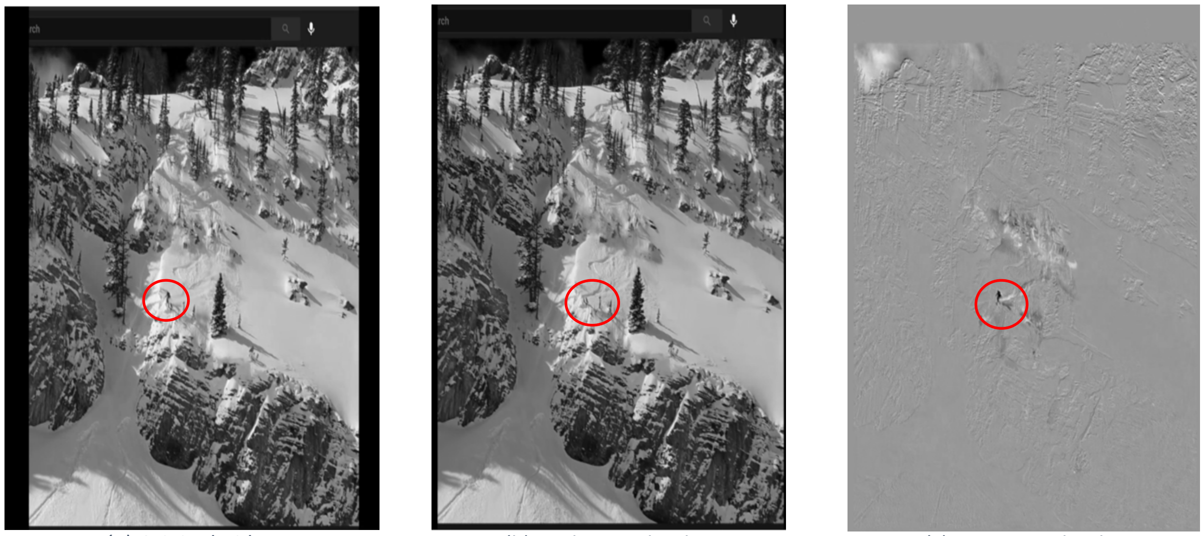
First, we load the video into MATLAB using the `VideoReader` function. To be able to perform DMD on the dataset, we must turn our video into a matrix where the number of frames in the video is the number of columns. We then construct two subset matrices, `X1` and `X2`, of our data matrix A. These subsets are the data matrix however, they are offset by one time step. `X1` does not have the last time step data and `X2` does not have the data for the first time step.

Afterwards, we employ Singular Value Decomposition on our `X1` data matrix to allow us to find the eigenvalues to use later on. We find a matrix `S` where the columns of one of our matrices can be written as linear combinations of the columns of `U`. We then determine the eigenvalues of this specific matrix. We set a vector `mu` which contains the DMD eigenvalues and use these values to covert to the `omega` vector, which is by taking the log of the eigenvalues, transforming them into another form. We finally also construct a `Phi` vector containing the eigenvectors of the `S` matrix multiplied by the columns of `U`.

To only use the background modes when constructing our DMD solution, we want the omega values that are closest to zero. Therefore, we set a threshold of 0.001 and find wherever the absolute value of the `omega` values are under that threshold. We now want to use this `new omega` vector as it will allow us to easily extract the background.

We calculate our DMD solution for each frame in the video using the DMD function mentioned in Section 2. We add all these values to a matrix where the columns represent each frame of the background. To find the foreground, we simply subtract the absolute values of the background from the original video. We then reshape these matrices back into the format where we can view them as a picture.

4 Computational Results



(a) Original Video *(b) Background Video* *(c) Foreground Video*

Figure 1 Original, Background, and Foreground Video of Ski Drop at frame 200

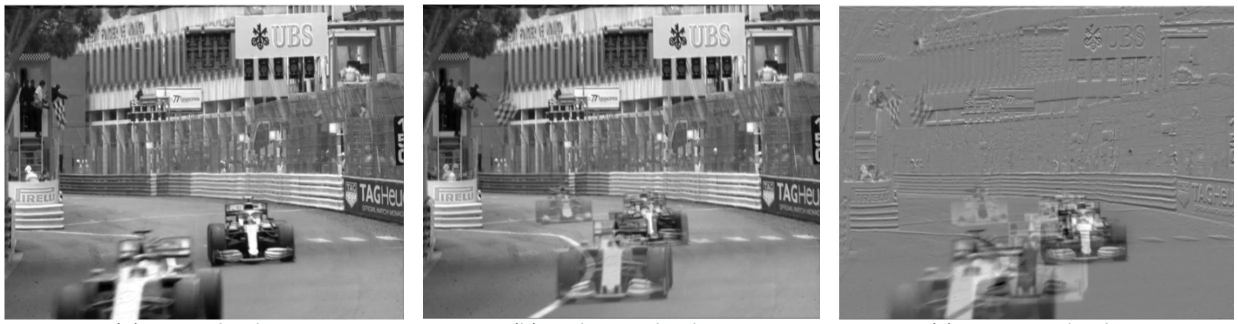

(a) Original Video (b) Background Video (c) Foreground Video

Figure 2 Original, Background, and Foreground Video of Racecars at frame 226

5 Summary and Conclusions

As seen in Figure 1, in this snapshot we have successfully used DMD to isolate the foreground and the background of the video. For this snapshot, in the Background video, we see the mountain, it's ridges, and the trees but we do not see the skier, whereas in the Foreground video, we only see the skier. When comparing this snapshot of the video to the original, we clearly see success for the DMD.

Looking at Figure 2, the separation is not as clear as it was in the ski video. In the Background snapshot, we still see the cars although they are much slighter than in the original and in the Foreground snapshot the cars are not as defined. It is unclear what might have caused this issue, perhaps it was the video itself or a bug in the code. With further investigation, we can maybe uncover the reason behind this and thus produce a better result and learn more about the DMD. Although there are some issue present, I would still say the DMD is pretty successful, we cannot always have perfect results especially when working with real life data.

We have gained a lot of knowledge about DMD while working through this problem, but it is clear that there is much more work and study that can be done to understand this field further. With further study we can understand DMD at a deeper level and implement this data analysis tool to more unique applications.

Appendix A. MATLAB functions used

- `v = VideoReader(filename):` creates object `v` to read video data from the file named `filename`.
- `sz = size(A):` returns a row vector whose elements are the lengths of the corresponding dimensions of `A`.
- `B = reshape(A, sz):` reshapes `A` using the size vector, `sz`, to define `size(B)`.
- `[U,S,V] = svd(A, 'econ'):` performs a singular value economy-size decomposition of matrix `A`, such that A = U*S*V' and removes extra rows or columns of zeros from the diagonal matrix of singular values.
- `E = eig(A):` returns a column vector containing the eigenvalues of square matrix `A`.
- `D = diag(V):` returns a square diagonal matrix with the elements of vector `v` on the main diagonal.

- `plot(X, Y):` creates a 2-D line plot of the data in `Y` versus the corresponding values in `X`.
- `Y = abs(X):` returns the absolute value of each element in array `X`. If `X` is complex, returns the complex magnitude.
- `pcolor(X, Y, C):` creates a pseudocolor plot using the values in the matrix `C`, while `X` and `Y` specify the coordinates for their respective vertices.
- `B = flip(A):` returns array `B` the same size as `A`, but with the order of the elements reversed.
- `y = exp(x)`: returns the exponential for each element in array `x`.

Appendix B. MATLAB codes

```
%%
sd = VideoReader('ski_drop_low.mp4');
%%
sd_vid = read(sd);
[n m s s2] = size(sd_vid);
for j = 1: s2
    sd_reshape = double(reshape(sd_vid(:,:,1,j), n*m, 1));
    vid_sd(:,j) = sd_reshape;
end

X = vid_sd;
X1 = X(:,1:end-1);
X2 = X(:, 2:end);

[U,Sig,V] = svd(X1, 'econ');
S = U'*X2*V*diag(1./diag(Sig));

[eV, D] = eig(S);
mu = diag(D);
omega = log(mu);
Phi = U*eV;

t = 0.001;
b = find(abs(omega) < t);
omega_b = omega(b);
y0 = Phi(:,b)\X1(:,1);

u_modes = zeros(length(y0), 454);
for i = 1:454
   u_modes(:,i) = y0.*exp(omega_b);
end
u_dmd = Phi(:,b)*u_modes;

X_fg = X - abs(u_dmd);
R = zeros(length(X_fg), size(X_fg,2));
u_dmd = R + abs(u_dmd);
X_fg = X_fg - R;
X_fg = reshape(X_fg, [n m size(X,2)]);
u_dmd = reshape(u_dmd, [n m size(X,2)]);
%%
c_f = 200;
figure(1)
```

```
pcolor(flip(reshape(X(:, c_f), [n m]))); shading interp, colormap(gray)
figure(2)
pcolor(flip(u_dmd(:,:,c_f))); shading interp, colormap(gray)
figure(3)
pcolor(flip(X_fg(:,:,c_f))); shading interp, colormap(gray)
%%
mc = VideoReader('monte_carlo_low.mp4');

mc_vid = read(mc);
[n m s s2] = size(mc_vid);
for j = 1: s2
    mc_reshape = double(reshape(mc_vid(:,:,1,j), n*m, 1));
    vid_mc(:,j) = mc_reshape;
end

X = vid_mc;
X1 = X(:,1:end-1);
X2 = X(:, 2:end);

[U,Sig,V] = svd(X1, 'econ');
S = U'*X2*V*diag(1./diag(Sig));

[eV, D] = eig(S);
mu = diag(D);
omega = log(mu);
Phi = U*eV;

t = 0.001;
b = find(abs(omega) < t);
omega_b = omega(b);

y0 = Phi(:,b)\X1(:,1);

u_modes = zeros(length(y0), 379);
for i = 1:379
   u_modes(:,i) = y0.*exp(omega_b);
end
u_dmd = Phi(:,b)*u_modes;

X_fg = X - abs(u_dmd);
R = zeros(length(X_fg), size(X_fg,2));
u_dmd = R + abs(u_dmd);
X_fg = X_fg - R;
X_fg = reshape(X_fg, [n m size(X,2)]);
u_dmd = reshape(u_dmd, [n m size(X,2)]);
%%
c_f = 226;
figure(1)
pcolor(flip(reshape(X(:, c_f), [n m]))); shading interp, colormap(gray)
figure(2)
pcolor(flip(u_dmd(:,:,c_f))); shading interp, colormap(gray)
figure(3)
pcolor(flip(X_fg(:,:,c_f))); shading interp, colormap(gray)
```