AMATH 482: Background Subtraction in Video Streams

Matthew Mangione

March 15, 2021

Abstract

In this assignment, we use dynamic mode decomposition to isolate and reconstruct the foreground and backgrounds of various videos. This decomposition allows us to isolate movement through time, enabling us to extract and subtract the background from the rest of the video. Specifically, we will experiment on two videos of isolated motion: a person skiing down a mountain, and a race-car driving around a turn.

1 Introduction and Overview

In this project, we will subtract the background in selected videos using dynamic mode decomposition, a powerful tool which allows us to isolate and separate the moving objects from the background of stationary videos. This decomposition is formed entirely without a mathematical bias of the input, making it a purely data-driven method. To explore this tool, we will analyze two videos and separate their foregrounds and backgrounds. The first is of a skier travelling down a mountain, and the second is of a race-car speeding down a track. Both videos are largely still besides for the motion captured of the respective moving objects.

2 Theoretical Background

Dynamic mode decomposition (DMD) is the backbone of this project, as it allows us to represent a series of time-dependent data points as a linear function. This makes use of the Koopman operator $\mathbf{A}$ on the data matrix $\mathbf{X}$, which represents each snapshot in time in its $\mathbf{M}$ columns. The Koopman operator is defined as a matrix which returns its input advanced forward in time.

$$x_{j+1} = \mathbf{A}x_j \tag{1}$$

Therefore, $\mathbf{X}$ can be rewritten in terms of $\mathbf{A}$, creating a Krylov space.

$$\mathbf{X_1^{M-1}} = [\mathbf{x_1}, \mathbf{x_2}, \mathbf{x_3}, ..., \mathbf{x_{M-1}}] = [\mathbf{x_1}, \mathbf{A}\mathbf{x_1}, \mathbf{A^2}\mathbf{x_1}, ..., \mathbf{A^{M-2}}\mathbf{x_1}] \tag{2}$$

Dynamic mode decomposition is largely dependent upon the concepts of Singular Value Decomposition (SVD), which transform the vectors of a matrix into distinct orthogonal components based upon its singular values, denoted by σ. More formally, given any matrix A $\in \mathbb{C}^{m\times n}$, this transformation finds the diagonalization of its singular values Σ of A in terms of full rank, unitary matrices U $\in \mathbb{C}^{m\times m}$ and V $\in \mathbb{C}^{n\times n}$.

$$\mathbf{A} = \mathbf{U\Sigma V^*} \tag{3}$$

In dynamic mode decomposition, the SVD is essential to finding the Koopman operator $\mathbf{A}$. This is achieved by taking the SVD of the data matrix $\mathbf{X}$, and reconstructing $\mathbf{A}$ via the similar matrix $\mathbf{S}$, which can be computed as following.

$$\mathbf{S} = \mathbf{U^* X_2^M V \Sigma^{-1}} \tag{4}$$

Because $\mathbf{A}$ and $\mathbf{S}$ are similar, they share the same eigenvalues.

$$\mathbf{Av_k} = \mathbf{Sv_k} = \lambda \mathbf{v_k} \tag{5}$$

Specific to the task of background subtraction, we hope to isolate the dynamic modes that correspond to the moving foreground versus the background of our video. This is accomplished by taking the modes corresponding to ω, defined below, which are approximately 0.

$$\omega = \frac{ln(\lambda_k)}{\Delta t} \tag{6}$$

Therefore, the approximate solution at all future times, denoted $\mathbf{X_{DMD}}$, can be written as following.

$$\mathbf{X_{DMD}} = \sum_{k=1}^{K} b_k(0) U v_k e^{\omega t} \tag{7}$$

Where $\mathbf{b}$ can be found with by $(\mathbf{Uv_k})^{+}\mathbf{x_1}$.

3 Algorithm Implementation and Development

3.1 Preprocessing

Given a video file, a lot has to be done to arrange the data for analysis. First, the file must be converted into a stack of images using the `VideoReader` object and related tools. During this process, each frame must be projected to grayscale, and each each pixel saved as a floating point. This can be done using MATLAB functions `im2gray` and `im2double` respectively. In order to use dynamic mode decomposition, we must rearrange each frame to be stored as a single column vector, such that each column describes an individual frame of the original video.

3.2 Dynamic Mode Decomposition

With each frame of our video neatly stored in the columns of $\mathbf{X}$, we can create $\mathbf{X_1^{M-1}}$ and $\mathbf{X_2^{M}}$, which store all of the frames except either the first or last. Recall that $\mathbf{X_2^{M}}$ can be approximated by multiplying the Koopman operator by $\mathbf{X_1^{M-1}}$. As described previously, the SVD of the first matrix can be calculated using the `svd` command. This will let us approximate the Koopman operator, but also gives us the opportunity to take a low-rank approximation of the data, allowing us to replicate it with only a handful of singular values. From this point, the $\mathbf{S}$ matrix can be calculated using the SVD matrices and $\mathbf{X_2^{M}}$, and its eigenvalues and eigenvectors can be found using the `eig` command. Then, to determine the dynamic modes corresponding to the background video, the ω values can be calculating by taking the `log` of the eigenvalues. The ω and their respective modes are then saved, and the low rank approximation of the data can be calculated according to Equation 7.

3.3 Separation & Formatting

From this point, we have reconstructed the low, rank background approximation of our original video. Therefore, we can calculate the foreground, the moving object, by subtracting our approximation from the original video. With this, we experience a form of computational error, as this process creates several pixel values outside the expected range of [0,1]. To compensate, we use logical indexing to find the negative values of the foreground video, $\mathbf{X_{DMD}^{Sparse}}$. These residuals are saved and appended back into the background and foreground approximations. After correcting this, the reconstructions are ready to be reformatted as videoframes, which is accomplished by using the `reshape` command to reverse the previous transformations of the video images into columns.

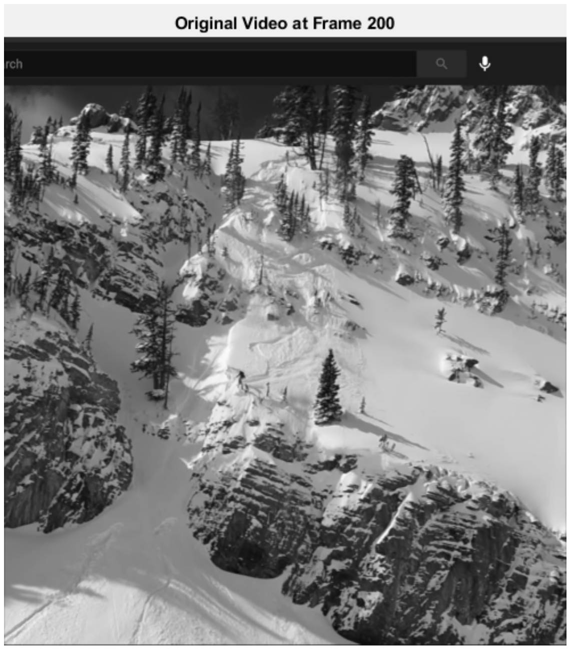

Figure 1: A snapshot of the original video. Notice the skier in the center right of the frame.

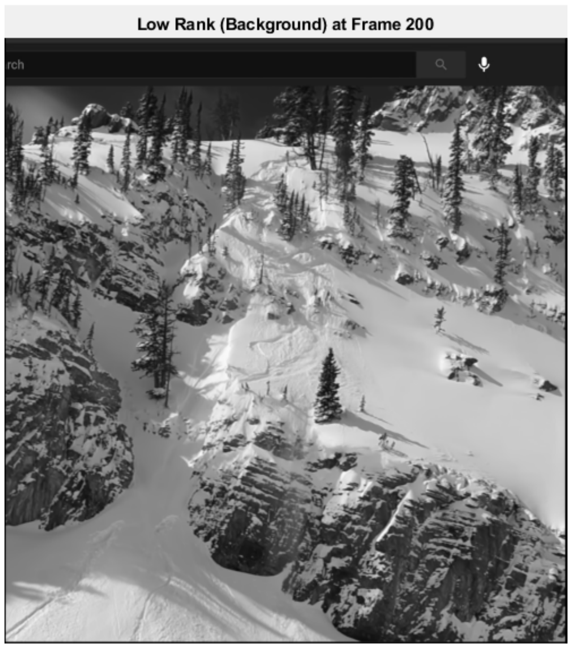

Figure 2: A snapshot of the reconstructed background video. Notice the lack of a skier.

3.4 Skiing Video Results

This video depicts a person skiing down a mountainside. There are many trees and cliffs, but the majority of the frames are filled with a bright white snow. Because of this, there is a lot of contrast, and the video works nicely in grayscale. The Dynamic mode decomposition worked fairly straightforward. The singular values of the data matrix were largely dominated by a single value. Thus, a very low rank approximation could be made, saving a lot of memory on my machine. Given the large resolution of the video matrices, this drastically reduced the loading time for the program. When reconstructed, the low rank matrix was almost perfectly isolated from the moving skier, however, there remained some visual artifacts from some of the trickier movements, like snow falling down the cliff at the end. Adding the residuals helped the sparse matrix (the foreground) significantly, negating some unsavory artifacts. Overall, the videos could be recreated quite successfully, with the combined foreground and background closely reproducing the original.

3.5 Monte Carlo Results

Similarly, our methods were able to separate the foreground and background of the video of the race car. There is a lot more background motion in this video, which impacted the quality of the separations, but it still separated effectively. The run time of this video was very long, so we sampled the videos to only use every other frame. This, in conjunction with the low rank SVD approximation, made the program run much faster. The background and foreground reconstructions decently recreated the original video; there is what seems like a smudge on the background approximation, what is likely a visual remain of the moving cars. Similarly, the partially moving aspects like the fans and the waving flag, are blurred significantly. Because there are many moving parts, the foreground was not as clean as the skier, significantly capturing the flag as well.

4 Summary and Conclusions

In this project, we were able to extract and separate the different components of the two videos. Dynamic mode decomposition is a very interesting and powerful tool that gives us to approximate a time-series as a

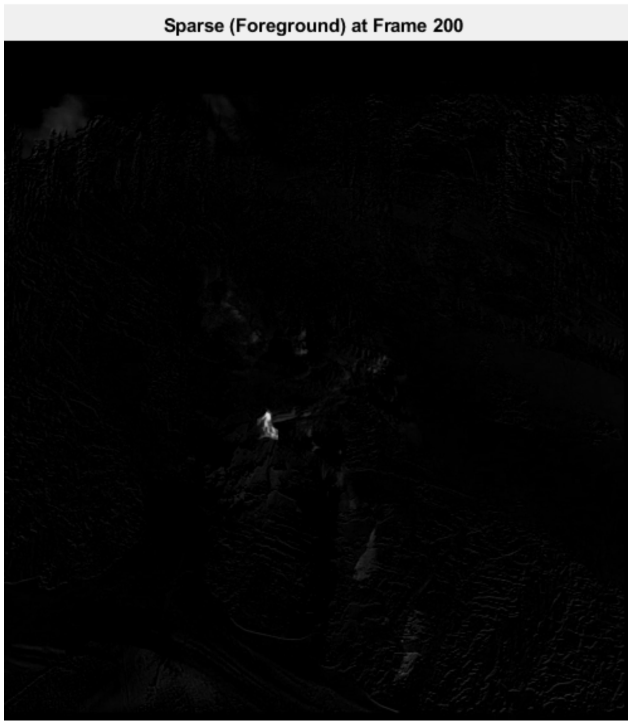

Figure 3: A snapshot of the reconstructed foreground of the video. The skier's motion is captured.

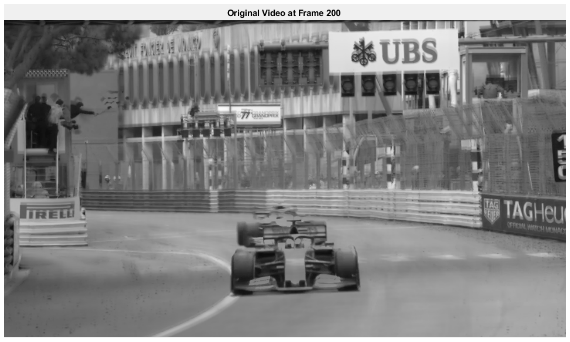

Figure 4: A snapshot of the original video.

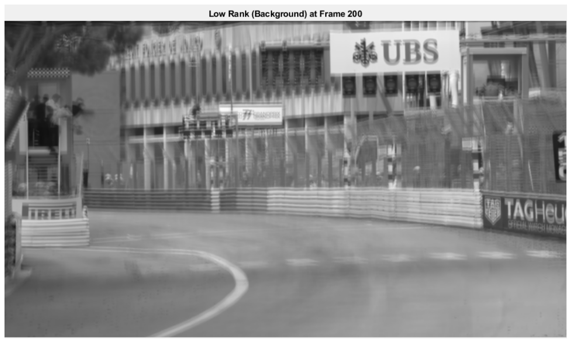

Figure 5: A snapshot of the reconstructed background video.

linear Krylov subspace, allowing us to perform unique tasks on it, like separating the moving components from otherwise stationary videos. Overall, the method excelled in subtracting the background, as was expected. This was far more effective than the methods used in earlier projects, namely assignment 3, where extracting moving components of videos was critical in creating an effective model.

Appendix A MATLAB Functions

- `X = abs(X)` returns the absolute value of the inputted matrix **X**. In the case that **X** is a complex conjugate, this function returns the modulus of **X**
- `v = diag(A)` returns the diagonal of the inputted matrix A stored as a vector.
- `[V, D] = eig(A)` returns the eigenvectors and eigenvalues of inputted matrix **X**.
- `data = rescale(data)` converts the scale of the input to the range of $[0, 1]$.
- `data = reshape(data, n, m)` transforms the shape of a dataset to the dimensions defined as parameters.
- `[U,S,V] = svd(data)` applies singular value decomposition to the dataset and returns the two orthnormal matrices U and V, as well as the diagonal matrix of its singular values, S.

Appendix B MATLAB Code

```
% Matthew Mangione
% AMATH 482: Computational Methods for Data Analysis
% Assignment 5: Video Background Subtraction

%% Load videos / convert to matrices / save for easier access
v_ski = VideoReader('ski_drop_low.mp4');
v_mc = VideoReader('monte_carlo_low.mp4');

% convert each ski video frame into grayscale double matrix
for i = 1:v_ski.numFrames
    ski(:,:, i) = im2double(im2gray(readFrame(v_ski)));
end

% convert each monte carlo video frame into grayscale double matrix
for i = 1:v_mc.numFrames
    mc(:,:, i) = im2double(im2gray(readFrame(v_ski)));
end

%save('ski.mat', 'ski');
%save('mc.mat', 'mc');

%% Reload pre-processed video files // Assign DMD matrices

%load('ski.mat');
%load('mc.mat');

%ski = samplevid(ski, 2);
%playVid(ski)

% reshape each pixel in each image to be a vector, sample video
X = samplevid(reshape(ski, size(ski, 1)*size(ski, 2), size(ski, 3)), 2);
Xs = samplevid(X, 2);

% assign sub matrices
X1 = Xs(:,1:end-1);
X2 = Xs(:,2:end);
t = 1:size(Xs,2);


%% SVD / initialization
[U,Sigma,V] = svd(X1, 'econ');
rank = 10;

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);
Phi = U*eV;

mu = diag(D);
omega = log(mu)/dt;

y0 = Phi\X(:, 1);  % pseudo-inverse initial conditions

% get omega closest to 0
[~, minOmega] = min(abs(omega));

%% plot singular values
%% plot singular values
sv = diag(Sigma) / max(diag(Sigma));

figure(1)
plot(1:5, sv(1:5), 'ro')
title('Singular Values of Video');
xlabel('Index of Singular Value');
ylabel('Singular Value (Normalized)');

%% Recreate DMD solution

lowrank = y0(minOmega).*Phi(:, minOmega).*exp(omega(minOmega).*t);
sparse = Xs - abs(lowrank);

% Create matrix of residuals (negative values)
R = sparse .* (sparse < 0);

% Add residuals back into reconstructions
lowrank = abs(lowrank) + R;
sparse = sparse - R;

%% plots
frame = 200;

figure(2)
fg = reshape(sparse, [height,width, length(t)]);
imshow(uint8(fg(:,:,frame)))
title("Original Video at Frame 200");

figure(3) = reshape(Xs, [height,width, length(t)]);
imshow(uint8(fg(:,:,frame)))
title("Sparse (Foreground) at Frame 200");


figure(4)
bg = reshape(low_rank, [height,width, length(t)]);
imshow(uint8(bg(:,:,frame)))
title("Low Rank (Background) at Frame 200");

%% functions

% samples a video (takes 1 per n frames)
function [svid] = samplevid(vid, jump)
    numFrames = size(vid);
    ind = 1;
    for j = 1:jump:numFrames(2)
        X = vid(:,j);
        svid(:,ind) = X;
        ind = ind + 1;
    end

end
```