Dynamic Mode Decomposition to Separate a Video

Johnny Van

March 12, 2021

Abstract

We used Dynamic Mode Decomposition in order to separate the foreground and background of two given videos. One video is of cars racing on a track, and the other is of a skier going down a mountain. Using DMD, we will have created a video of the foreground and a video of the background for each given video,

1 Introduction and Overview

We are given two videos, one is a video of cars racing on a track, and the other is a video of a person skiing down a mountain. Using Dynamic Mode Decomposition(DMD), we will separate the foreground and background of each video, so at the end of this project, for each video, we will have created two videos, one that shows only the foreground, and the other shows only the background. We explore the concepts of low rank and sparse DMD reconstructions and how they are related to the background and foreground of a video.

2 Theoretical Background

2.1 Dynamic Mode Decomposition Background

Suppose that we have a high-dimensional data set, but we don't have any knowledge of the governing equations for our algorithm for analysis? This leads to the idea of data-based algorithms which is fairly new concept.

An example of such an algorithm is described in the textbook, [1], as the **Dynamic Mode Decomposition** or DMD. The objective of this method is to take advantage of the low-dimensionality of the experimental data without using a set of governing equations. Basically, DMD forecasts our data which means it predicts what our data will look like in the future. DMD finds a basis of spatial modes, which don't have to be orthogonal like in PCA, for which the time dynamics are just exponential functions. Sometimes, we have complex exponents, but in this case, we are mainly dealing with real exponents. With exponential functions, the only things that happen in time is osciallations, decay, and/or growth. DMD essentially turns the problem into a linear system of differential equations.

Let's assume that we have snapshots of spatio-temporal data, which means the data evolves in both space and time. Let N be the number of spatial points saved per snapshot. Let $\mathcal{M}$ be the number of snap shots taken. It is important that the time data or snapshots are collected at regularly space intervals.

$$t_{m+1} = t_m + \delta t, \qquad m = 1, ..., M-1, \qquad \delta t > 0 \tag{1}$$

Then the snapshots are denoted,

$$U(x, t_m = \begin{bmatrix} U(x_1, t_m) \\ U(x_2, t_m) \\ U(x_3, t_m) \\ U(x_4, t_m) \end{bmatrix} \tag{2}$$

for each m = 1,..., M. We can then just use these snapshots to form the columns of the data matrix X.

$$X = \begin{bmatrix} U(x, t_1) & U(x, t_2) & ... & U(x, t_M) \end{bmatrix} \tag{3}$$

and the matrix X_j^k which represents columns j through k of the full matrix X.

$$X_j^k = \begin{bmatrix} U(x, t_j) & U(x, t_{j+1}) & ... & U(x, t_k) \end{bmatrix} \tag{4}$$

2.2 Koopman Operator

DMD essentially approximates the modes of the Koopman operator. Let's call the Koopman operator as A, and A is a linear, time-independent operator such that

$$x_{j+1} = Ax_j \tag{5}$$

where j indicates the 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 the vector of data points collected at time j, so we are applying A to this vector in order to advance time forward by δt. Conceptually, we are performing a linear mapping from one timestep to another, even if the system dynamics are likely nonlinear. This operator is global which means it is not restricted to an area of space or time.

2.3 Dynamic Mode Decomposition

Let's consider the matrix,

$$X_1^{M-1} = \begin{bmatrix} x_1 & x_2 & ... & x_{M-1} \end{bmatrix} \tag{6}$$

where x_j is a snapshot of data at the time t_j. With the Koopman operator, we can rewrite this as,

$$X_1^{M-1} = \begin{bmatrix} x_1 & Ax_1 & A^2x_1 ... & A^{M-2}x_1 \end{bmatrix} \tag{7}$$

We basically applied the powers of operator A to the vector x_1, and these formed the basis for the **Krylov subspace**. Now, we somewhat have a way of relating the first M-1 snapshots to x_1 or the first/initial snapshot using A. We can rewrite the above matrix to get,

$$X_2^M = AX_1^{M-1} + re_{M-1}^T \tag{8}$$

where e_{M-1} is a vector of all zeros except at the (M-1)st component. The main idea around this equation is that after we apply A to each column of X_1^{M-1}, there is residual error denoted as r. Currently, A is unknown and it is our objective to find it. Generally, matrices are understood by their eigenvalues and eigenvectors alone, so we are going to find A directly by finding other matrices with the same eigenvalues, which then the eigenvectors are founds easily after doing so.
First, let's perform the SVD on X_1^{M-1}, such that $X_1^{M-1} = U\Sigma V^*$. Then let's rewrite equation (9) to be

$$X_2^M = AU\Sigma V^* + re_{M-1}^T \tag{9}$$

From here we choose A so that the columns of X_2^M can be written as linear combinations of the columns of U. This is similar to how they can be written as linear combinations to the POD modes, which means the vector r must be orthogonal to the POD basis, or $U^*r = 0$. we multiply equation (10) by U^* on the left side,

$$U^*X_2^M = U^*AU\Sigma V^* \tag{10}$$

Then isolate for U^*AU by multiplying by V and then Σ^{-1} on the right to get,

$$U^*AU = U^*X_2^M V\Sigma^{-1} \tag{11}$$

The right side of equation (12) can be denoted as $\tilde{S}$.

Note: We want to achieve dimensionality reduction with these data-driven methods, so you don't need to use the entire matrix of Σ, you just need to use Σ with the first K nonzero singular values on the diagonal.

Notice that $\tilde{S}$ and A are related if we apply a matrix on one side and its inverse on the other, which infers they are similar. Similar matrices share a lot of properties such as the same eigenvalues. If y is an eigenvector of $\tilde{S}$, then Uy is an eigenvector of A. Therefore the eigenvector/eigenvalue pairs of $\tilde{S}$ are,

$$\tilde{S}y_k = \mu_k y_k \tag{12}$$

Thus, the eigenvectors of A, or the DMD modes, are

$$\psi_k = Uy_k \tag{13}$$

Finally, we can just expand in our eigenbasis to get,

$$x_{DMD}(t) = \sum_{k=1}^{K} b_k \psi_k e^{w_k t} = \Psi diag(e^{w_k t} b) \tag{14}$$

K is the rank of X_1^{M-1}, b_k are the initial amplitudes of each mode, and the matrix, Ψ, contains the eigenvectors, ψ_k, as its columns.In this case, we write the time dynamics as exponential functions, meaning that $\omega_k = ln(\mu_k)/\delta t$. The vector μ contains the DMD eigenvalues, and ω_k is another form of of these eigenvalues. The reason why we need this different form is because we want our DMD reconstruction to look more like a solution to a continuous time ODE and not discrete. To compute b_k, we would use the formula:

$$b = \Psi^{\dagger} * x_1 \tag{15}$$

where $\Psi^{\dagger}$ is the psuedoinverse of Ψ and x_1 is our initial condition/snapshot.

2.4 Low Rank and Sparse DMD Approximations

The spectrum of frequencies from Dynamic Mode Decomposition can be used to subtract background modes. Assume that ω_p, where $p \in 1, 2, ...\updownarrow$ cause $||\omega_p|| \approx 0$, and that $||\omega_j|| \ \forall j \neq p$ is bounded away from zero. Then

$$X_{DMD} = b_p \phi_p e^{\omega_p t} + \sum_{j \neq p} b_j \phi_j e^{\omega_j t} \tag{16}$$

The reason this form is necessary is because each term of the DMD reconstruction is complex even though they sum to a real-valued matrix. This makes it difficult to separate the DMD terms into an approximate low-rank and sparse reconstruction because we want real-valued outputs and handling the complex elements incorrectly can significantly reduce the accuracy of the low-rank/sparse reconstructions. Let's suppose the DMD's approximate low-rank reconstruction is given by equation (16).

$$X_{DMD}^{Low-Rank} = b_p \phi_p e^{\omega_p t} \tag{17}$$

Then it is true that our original data X is related to the low-rank and sparese reconstruction by:

$$X = X_{DMD}^{Low-Rank} + X_{DMD}^{Sparse} \tag{18}$$

Instead of calculating the DMD's sparse reconstruction with this equation:

$$X_{DMD}^{Sparse} = \sum_{j \neq p} b_j \phi_j e^{\omega_j t} \tag{19}$$

We can simply use the equation:

$$X_{DMD}^{Sparse} = X - |X_{DMD}^{Low-Rank}| \tag{20}$$

However, this can cause X_{DMD}^{Sparse} to have some negative values which do not make sense due to how these values represent the pixel's intensity. The solution to this is to take all of these residual negative values into a matrix called $\mathbf{R}$, and then we can add it back to the low rank reconstruction and subtract it from the sparse reconstruction.

$$X_{DMD}^{Low-Rank} \leftarrow \mathbf{R} + |X_{DMD}^{Low-Rank}| \tag{21}$$

$$X_{DMD}^{Sparse} \leftarrow X_{DMD}^{Sparse} - \mathbf{R} \tag{22}$$

This method allows for the magnitudes of the complex values are accurately accounted for and that the approximate low-rank and sparse DMD reconstructions are real-valued.

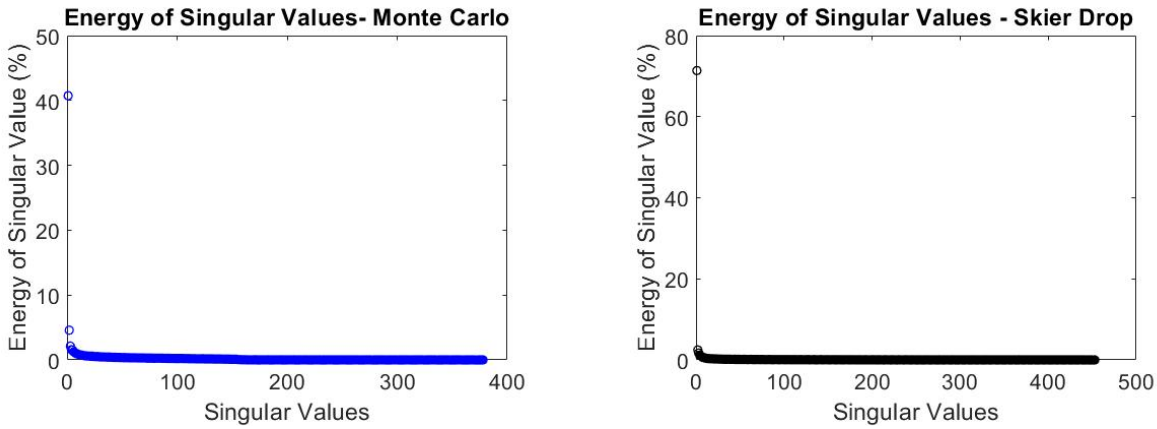

Figure 1: The left plot shows the energy percentages of each singular value given by the SVD of Monte Carlo video data. Similarly, the right plot shows energy of singular values from SVD of the skier video data.

3 Algorithm Implementation and Development

3.1 Algorithm 1: Dynamic Mode Decomposition of Videos

Load in the videos into MATLAB, they should come in as a 4-D matrix which is width x height x RGB x frame. The videos are split up evenly into frames, and these would be our "snapshots" that we refer to in DMD. Use the snapshots to create the data matrix X, then using equation (7) and equation (8) respectively to create the matrices, X_1^{M-1} and X_2^{M-1}.

1. Compute the SVD of the matrix X_1^{M-1}. Choose what rank you will use and redefine U,S,V so that they only contain the number of columns corresponding to the rank you chosen.

2. Find the matrix $\tilde{S}$ which is the right side of equation (11), and then find its eigenvalues and eigenvectors. Then calculate Ψ which is the U matrix multiplied by eigenvectors of $\tilde{S}$.

3. Calculate ω which are the eigenvalues of $\tilde{S}$ divided by the time interval between frames. Use the initial snapshot x_1 and the pseduoinverse of Ψ to find the coefficients b_k.

4. Plot the omega values and find the omega values closest to zero. Then find the values of Ψ and b_k at those same indexes. Redefine *omega*, Ψ, b_k to be only these particular values.

5. Using equation (17), compute the low rank DMD reconstruction matrix. Then compute the sparse DMD reconstruction by subtracting the absolute value of low rank reconstruction from X_1^{M-1}.

6. Isolate the negative values of X_{sparse} into a matrix $\mathbf{R}$. Add R onto the low rank approximation and subtract $\mathbf{R}$ from the sparse approximation, then plot these frames to see how well defined they are.

7. Compute the DMD reconstruction by adding the low rank and sparse reconstruction together. Then plot a frame of the DMD reconstruction.

4 Computational Results

4.1 Monte Carlo Separation

Using Algorithm 1 create the low rank DMD and the sparse DMD solution gave us great results. The rank I chose on step 3 of Algorithm 1 was 62 because that accounted for about 80% of the energy of the original data. In this assignment, I tend to use ranks that account for $70\% - 80\%$ of the energy of original data because there is only a slight, almost unnoticeable difference in the accuracy of our results. For comparison,

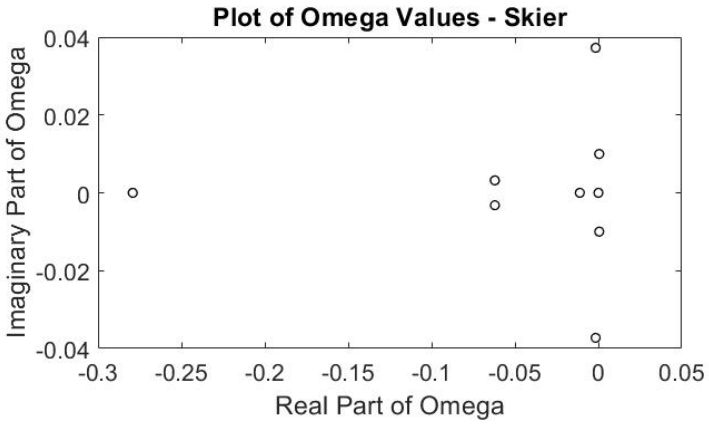

Figure 2: A plot of the omega values that was created while performing DMD on the skier video. The plot compares the real parts to complex parts of the omegas.

a rank of 62 accounts for 80% of data's energy and a rank of 126 accounts for 95% of data's energy. Our program runs significantly faster with a rank of 62 while still achieving a great separation.
In Figure 6 in Appendix A, the plot of the omega values is shown, and there are exactly 62 omegas values since we chose a rank of 62 in step 3 of Algorithm 1. In order to find the omega values that have real parts close to 0, we have to set a threshold to define what value is "close to 0". So in this case, I set my threshold to be omega values with absolute values less than 0.1, so everything to the right of this threshold in the plot are considered the low rank DMD modes. Then from there, I followed the rest of the steps of Algorithm 1 to separate the video into a foreground and background.

So we can see our results in Figure 3, where we look at the 75th frame of both the background and foreground video of Monte Carlo. In the left figure,we can see the race track without any trace of the race cars. In the right figure, most of the figure is black and we can see only the cars. This makes sense because the spare reconstruction should only contain pixels of the moving cars and the low rank reconstruction contains only information about the background of the video. Therefore, our attempt at separation was a success. In Figure 5 in Appendix A, you can compare the background, foreground, and DMD reconstruction to the original video.

One thing of note is that adding $\mathbf{R}$ onto the low rank reconstruction gave poor results. We noticed that adding $\mathbf{R}$ caused the cars to appear in the low rank reconstruction which is not good since our objective is to separate the background from the foreground. However, subtracting $\mathbf{R}$ from sparse reconstruction did not seem to alter results significantly. In both the Monte Carlo video and the skier video, I decided not to add $\mathbf{R}$ onto their low rank reconstructions in order to get a clean separation.

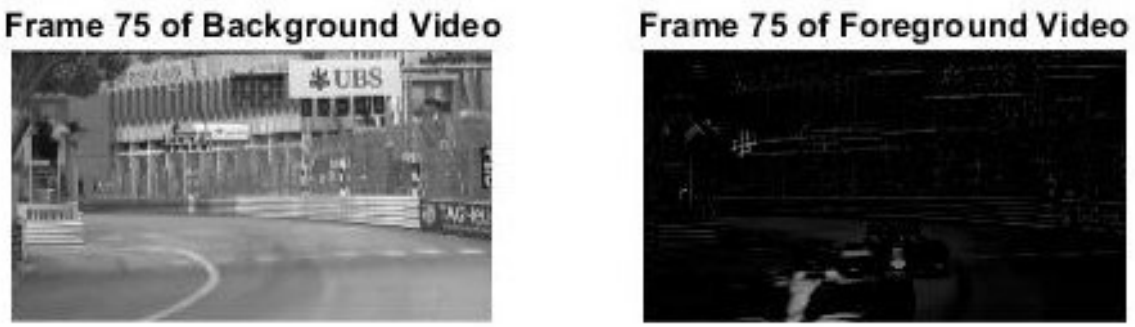

Figure 3: The left figure shows the background of the video in the 75th frame after we separated it from the original Monte Carlo video. Similarly, the right figure shows the foreground of the video at 75th frame.

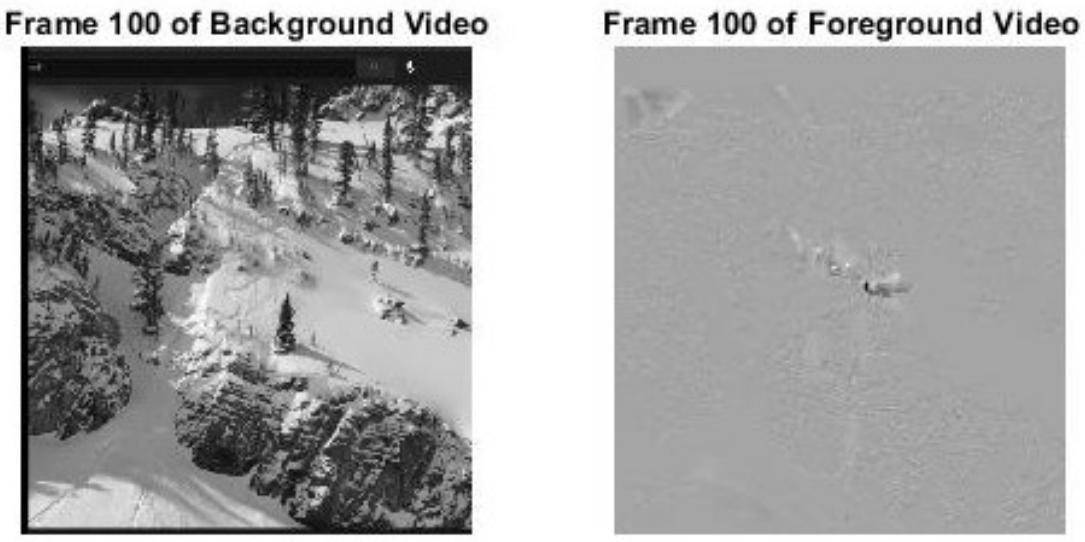

Figure 4: The left figure shows the background of the video in the 100th frame after we separated it from the original skier video. Similarly, the right figure shows the foreground of the video at 100th frame.

4.2 Skier Drop Video

Using Algorithm 1, we achieved a clean separation of the skier video. In Figure 4, we can see that the left figure shows the image of the mountain without any trace of the skier. Then in the right figure, everything is greyed out except the tiny body of the skier in the middle of the frame. When playing the foreground video, we were able to deduce the exact movements of the skier, down to his arm movements which is an amazing result. In Figure 7, you can compare the background and foreground to the original video.

For separating the skier video, I used a rank of 9 which corresponds to 80% of energy of original data. Using a rank-truncation of 9, we followed the steps of the algorithm and defined omega. Figure 2 shows the plot of omegas corresponding to the skier data. Since our objective is to find the omegas closest to zero, we set our threshold to omegas less than absolute value of 0.05, so the 6 rightmost omega values should correspond to the background/low rank DMD modes.

Similarly to the Monte Carlo separation, we found that adding $\mathbf{R}$ to the low rank reconstruction caused the body of the skier to appear which is not ideal in our goal of separation. For the sparse reconstruction, we found that when plotting the frames, the skier is not clearly visible as his body is black and most of frame is black as well. The solution I have found is that if we subtract each column of the sparse reconstruction by its minimum and divide it by the column's maximum minus its minimum, we are able to rescale this matrix and plotting the frames will allow us the see the skier as shown in Figure 4.

5 Summary and Conclusions

Dynamic Mode Decomposition was able to cleanly separate the foreground and background from two videos, Monte Carlo and the skier video. This is an amazing feat as this method could be applied to any video and have other potential applications. However, this project was only limited to separating grayscale videos as our attempt to separate RGB videos was unsuccessful. It will be a good idea to come back to this project and attempt that once more. All in all, using Dynamic Mode Decomposition is an excellent tool that separates the foreground and background from a video which could be useful for processing data for other projects.

References

[1] Jose Nathan Kutz. *Data-driven modeling & scientific computation: methods for complex systems & big data*. Oxford University Press, 2013.

Appendix A Additional Figures

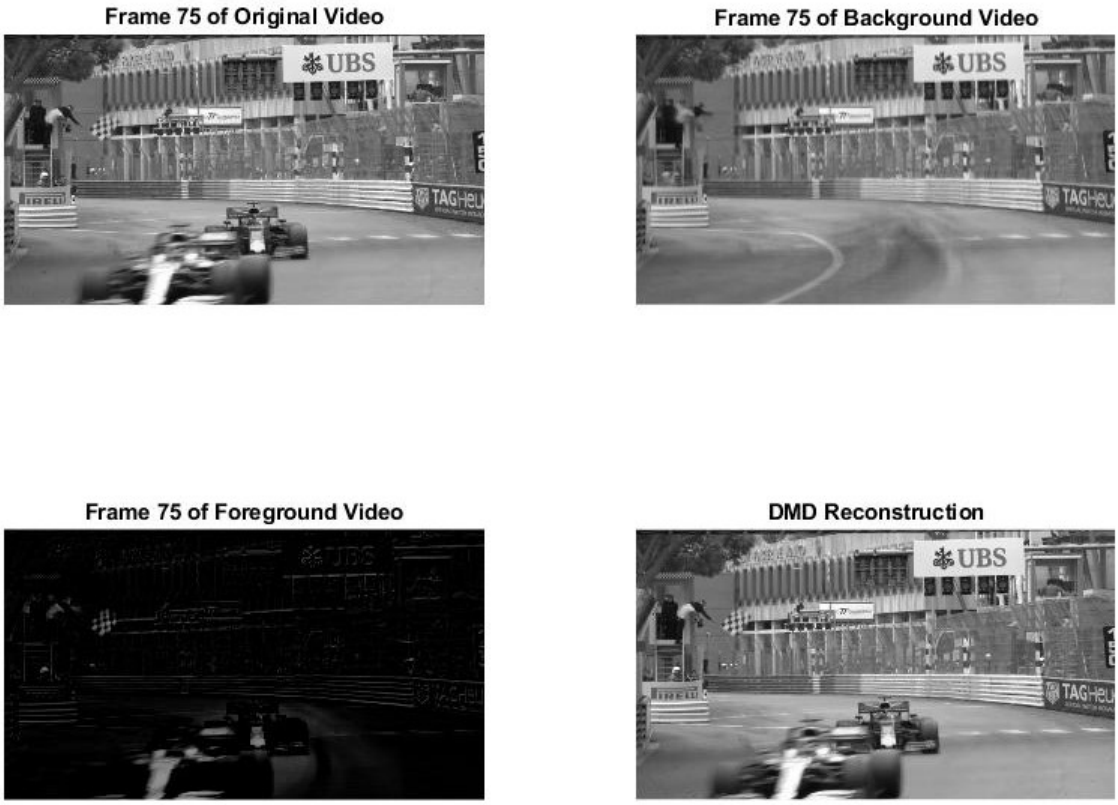

Figure 5: This figure shows the 75th frame of the original, background, foreground, and DMD reconstruction video of the Monte Carlo video.

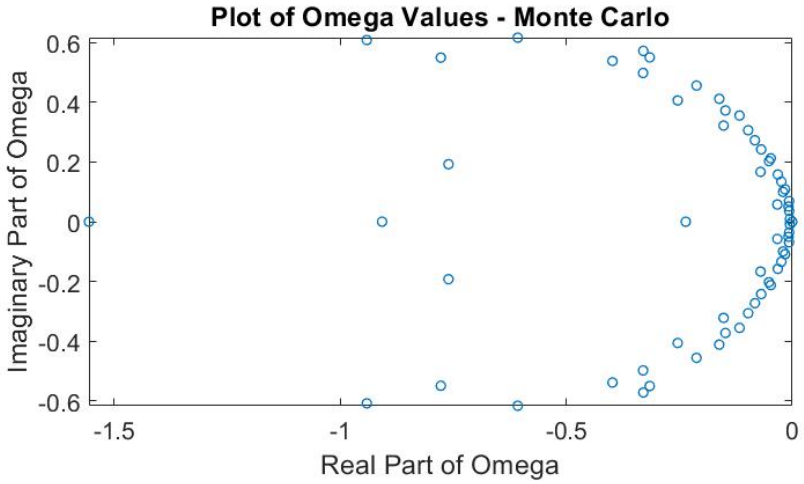

Figure 6: A confusion matrix showing the mislabeled images made by 10-digit SVM classifier on test data.

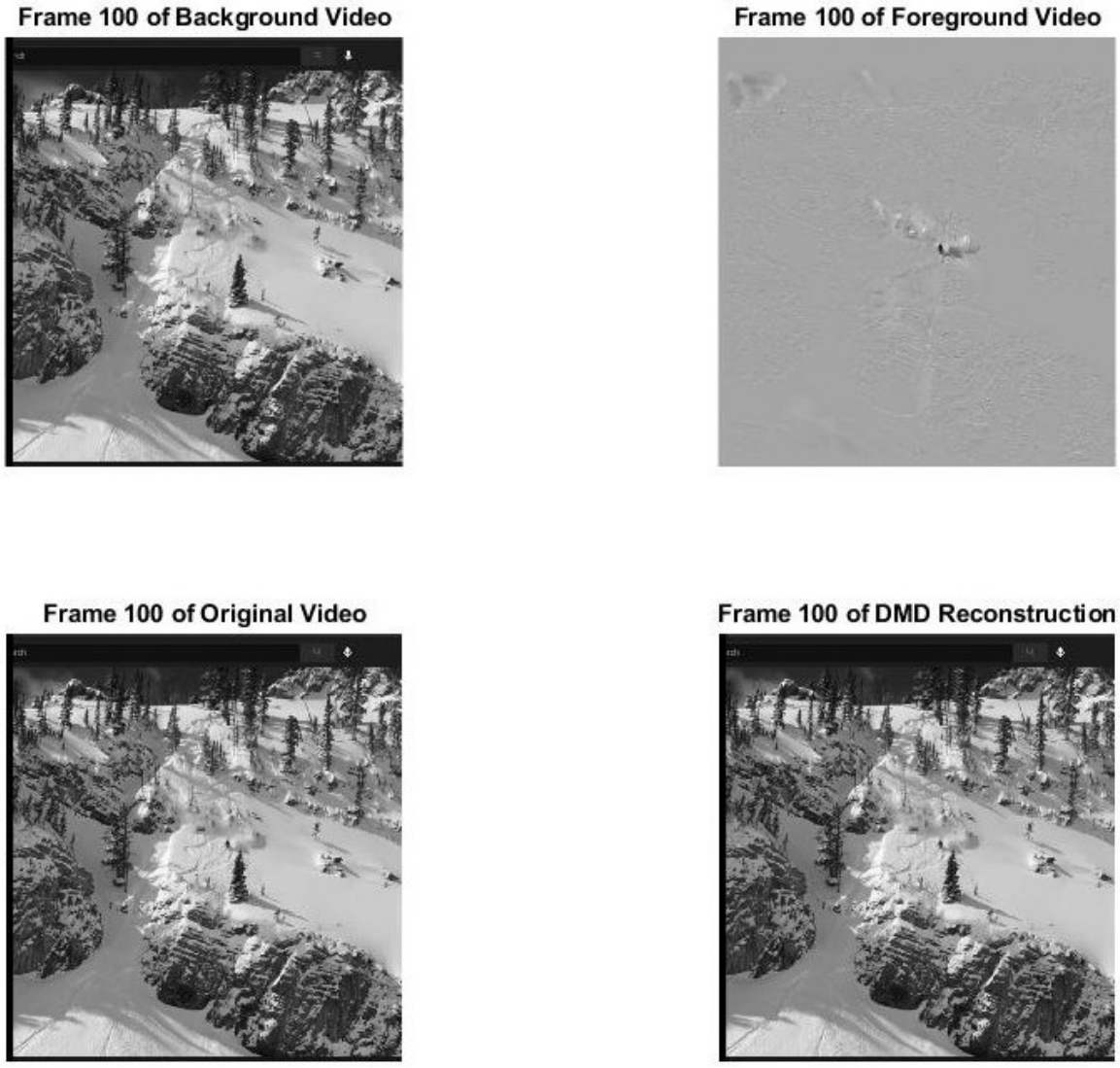

Figure 7: This figure shows the 100th frame of the background, foreground, original and DMD reconstruction video of the skier video.

Appendix B MATLAB Functions

- `y = reshape(X, [ m n ])` reshapes X to the size defined by m and n. So [m n] represents reshaping X to be a matrix with m rows and n columns.
- `k = find(X)` returns a vector containing the linear indices of each nonzero value in matrix X. If you use `k = find(X < 0)`, this will return indices of all negative elements in X.
- `video = read(v)` reads all frames of the video object v and returns a 4-D matrix that is width by height by RGB values by frame.
- `v = videoReader('X')` creates an object called v that reads the video data from the file X.
- `Y = uint8(X)` converts the values of X into uint8 values.
- `rgb2gray(X)` converts the RGB image, X, into a grayscale image where each pixel has a value indicating how bright that pixel is.

- `[`I_1, I_2`] = ind2sub(sz, linearIndex)` returns I_1, I_2 which represents the equivalent multidimensional subscripts corresponding to the linearIndex for a multidimensional array of size sz.
- `[Max, Index] = max(x(:))` returns the max of an array of values and gives its index.
- `imshow(X)` returns the frame of the video, X, as a figure.
- `[m,n] = size(X)` stores the number of rows of X as m and the number of columns of X as n.
- `repmat(mean, 1, n)` returns an array containing n copies of mean in the row dimension specified by the 1.
- `[U,S,V] = svd(A, 'econ')` performs the singular value decomposition of matrix A and stores the decompsition as the three matrices U,S,V such that A = U*S*V'. The 'econ' means that the command returns an economy-size decomposition of A.

Appendix C MATLAB Code

```
%% Part 1 - Follow this for both Monte Carlo and Skier Video

%% Load in the video
v= VideoReader('monte_carlo_low.mp4');  % Load Monte Carlo or Skier
video = read (v) ;
frames = get(v,'numberOfFrames');
[width height rgbSize frames] = size(video);

%% Reshape each frame of the video into a column vector and form a
%  matrix where each column is a frame.

X = [];
for i = 1:frames
    x = rgb2gray(video(:,:,:,i));
    x = double(reshape(x(:,:),width*height,1));
    X= [X x];
end

%% Define X1 and X2 and perform SVD on X1
X_1 = X(:, 1:end-1);
X_2 = X(:, 2:end);

[U1,S1,V1] = svd(X_1, 'econ');

%% Plot the Singular Values
plot(diag(S1)/sum(diag(S1))*100, 'bo', 'Linewidth', [1]);
set(gca, 'Fontsize', 16);
xlabel('Singular Values');
ylabel('Energy of Singular Value (%)')
title("Energy of Singular Values- Monte Carlo")

diagonal_S = diag(S1)/sum(diag(S1))*100;
sum_sing = 0;
i = 0;
while (sum_sing < 80.0)
    i = i+1;
    sum_sing = sum_sing + diagonal_S(i);
end

%% Choosing how many dimensions we should use from SVD of X1 before
%  performing DMD.

rank=i;
U=U1(:,1:rank); S =S1(1:rank,1:rank); V = V1(:,1:rank);

%% Defining S_tilde, Phi, omega, y_0
S_tilde = U' * X_2 * V * diag(1./diag(S));
```

Listing 1: Example code from external file.

```
[eigVec, eigVal] = eig(S_tilde);

Phi=U*eigVec;
mu=diag(eigVal);
dt = 1;
omega=log(mu)/dt;

b = Phi\X_1(:,1);

%% Plotting Omega values
plot(omega, 'o', 'Linewidth', [1]);
xlabel("Real Part of Omega")
ylabel("Imaginary Part of Omega")

%% Finding omega values close to 0 and their repsective Phi, y_0 values

time_step = size(X_1, 2);
t = (0:time_step-1)*dt;

bg_mode = find(abs(omega) < 0.01);
omega_bg = omega(bg_mode);
Phi_bg = Phi(:,bg_mode);
b_bg = b(bg_mode);

%% Creating the DMD low rank reconstruction
u_modes = zeros(length(y0_bg), size(X_1, 2));

for iter = 1:time_step
    u_modes(:,iter) =(b_bg .*exp(omega_bg*t(iter)));
end
u_dmd = Phi_bg*u_modes;   % DMD resconstruction with chosen modes

% End of Part 1 Code

%% Calculating sparse reconstruction - Monte Carlo - Part 2 Code

X_s = X_1 - abs(u_dmd);
R = X_s .* (X_s < 0);

X_lowr = uint8(abs(u_dmd));
X_sparse = uint8(X_s - R);
dmd_reconstruction = uint8(abs(u_dmd) + R) + X_sparse;

%% Calculating Sparse Reconstruction - Skier Video - Part 2 Code

X_s = X_1 - abs(u_dmd);
R = X_s .* (X_s < 0);

X_lr = uint8(abs(u_dmd));
X_sparse = (X_s - min(X_s)) ./ (max(X_s) - min(X_s));
dmd_reconstruction = uint8(abs(u_dmd) + R) + uint8(X_s -R);
```

Listing 2: Example code from external file.

```
%% Playing frames of low rank reconstruction - background
close all;
for i = 1:378
    image_low_rank(:,:,i) = (reshape(X_lowr(:,i),width,height));
end

for i = 1:378
    imshow(image_low_rank(:,:,i));
    i
    drawnow
end

%% Playing frames of sparse reconstruction - foreground
close all;
for i = 1:378
    image_sparse(:,:,i) = (reshape(X_sparse(:,i),width,height));
end

for i = 1:378
    imshow(image_sparse(:,:,i));
    i
    drawnow
end

%%
```

Listing 3: Example code from external file.