Foreground-Background Separation: A Dynamic Mode Decomposition Approach

Nathaniel Linden
March 2018 – UW Bioengineering

Abstract

As video recording has become ubiquitous in the past decades, video processing and analysis has become ever important. Often times an analysis procedure will require the separation of a moving foreground object from the static background. This task is usually computationally inefficient and inaccessible for on the fly video processing. Generally video separation makes use a dimensionality reduction algorithm to isolate the foreground from the background. In the following pages I will outline a feature separation algorithm with utilizes the Dynamic Mode Decomposition for isolation of spatial dynamics and reconstruction of the two regions of interest (the foreground and the background).

Introduction

In data analysis, it is often useful to quickly extract the foreground and background from a video feed. More specifically, any object detection/motion tracking algorithm requires separation of the dynamic foreground object from the static background, so the development of a fast and accurate foreground-background separation is motivated. The following technique for this feature extraction makes use of the dynamic mode decomposition to extract the dynamic features (the foreground) from the static (the background).

Defined by Peter J. Schmid, in a 2010 article,[1] as way to extract the spatial and temporal information from a time series of fluid flow data, the dynamic mode decomposition (DMD), is a useful tool for studying dynamical systems. In theory, the DMD algorithm essentially extract low rank dynamics from the time series data and allows for future time predictions to be made. In the original context of fluid dynamics, it allowed for the fluid flow model to be extracted from experimental fluid flow model without prior knowledge of the actual dynamical model.

When applied to the problem of foreground and background feature separation DMD allows the foreground and background to be separated by estimating the background and the foreground with a low rank and a sparse reconstruction, respectively. Here an algorithm for DMD based foreground-background separation is developed and examined on a set of experimental video feeds.

Theoretical

As this foreground-background separation methods makes use of the DMD algorithm, the specifics of the DMD are as follows:

Consider the dynamical system defined by $\frac{dx}{dt} = f(x, t, u)$ where $x(t, u) \in \mathbb{R}^n$ which represents the state of the system at time t, where is an unknown, possibly nonlinear system. This is approximated by: $\frac{dx}{dt} = Ax$, a linear estimate. The solution of the above linear system is known, and

[1] SCMID, P. (2010). Dynamic mode decomposition of numerical and experimental data. *Journal of Fluid Mechanics, 656,* 5-28. Doi:10.1017/

is $x(t,u) = \sum_{k=1}^{n} \varphi_k \exp(\omega_k t)\, b_k$. In order to determine A and a predictor for this solution, A is found to be the Koopman operator which maps from one state of the system to a future state, as follows : $X = [x_1\ x_2 \dots x_{m-1}], X' = [x_2\ x_3 \dots x_m];\ X' = AX$.

The first step of the DMD involves taking the singular value decomposition of X. The components from this will be used for further computation. The SVD yields $= U\Sigma \mathrm{V}^*$. Then determine the number of modes necessary to capture the energy of X, this is r. U, Σ & V can be truncated using r-modes[2].

Step 2 requires solving for A. This can be done simply with $A = X'V\Sigma^{-1}U^*$However, it is more efficient to calculate $\tilde{A}; where\ \tilde{A} = U^*X'\mathrm{V}\Sigma^{-1}$. $\tilde{A}$ is a projection of A into a lower dimension subspace (with dimensions r x r)[3]. It is much more computationally efficient to use $\tilde{A}$.

Step 3 of the is to perform an Eigen decomposition of $\tilde{A}$. This is $\tilde{A}W = W\Lambda$. The columns of W are the eigenvectors of $\tilde{A}$ and Λ is a diagonal matrix with the eigenvalues.

Finally, step 4 of the DMD involves reconstructing the eigenvalues and eigenvectors of A. The eigenvectors are already given by Λ, while the eigen vectors are given by $\phi = X'V\Sigma^{-1}W$.

The time dynamics can then be reconstructed as follows:

$$\omega_k = \frac{\ln(\lambda_k)}{\Delta t}$$

$$\text{Then: } x(t) = \sum_{k=1}^{r} \varphi_k e^{\omega_k t} b_k = \phi e^{\Omega t} b$$

$$b \coloneqq \Phi^{-1} X$$

This estimation of the low rank time dynamics of a system can be used to approximate behavior of current states, and to make future state predictions.

Now we will consider the DMD in the context of video streams. Consider the video matrix $V \in \mathbb{R}^{m\,x\,n}$ where n is the total number of recorded pixels and m is the length of the recording. Np the data of v is converted into the data matrix $X \coloneqq [x_1\ x_2\ x_3 \dots x_m]$ where each vector x_i has size n and represents the observation at each point in time (each from of the video). Using the procedure above the DMD is executed on X and the low rank DMD representation will be defined as follows:

$$X_{DMD}^{Low\ Rank} = \sum_{k=1}^{r} \varphi_k e^{\omega_k t} b_k$$

This low rank DMD approximation will represent the background of the original video. This is due to the fact that the background does not (in theory) vary with time, so it can be represented easily with very few modes (exactly r number of modes). There is little variation through time of each background pixel, so all pixels can be represented in a low dimensional subspace. Next, in order to extract the foreground:

$$X_{DMD}^{Sparse} \coloneqq \sum_{k=r+1}^{n} \varphi_k e^{\omega_k t} b_k$$

This sparse matrix represents the foreground, because there is lots of non-uniform variation of the foreground, so this variation is best captured by a higher dimensional subspace. In theory every dimension which is not representing the static background will capture the dynamics of the foreground. Practically it is challenging to calculate these two matrices, low rank and sparse, without obtaining negative and complex pixel signals[4]. This can be achieved through the following:

$$1)\ X_{DMD}^{Low\ Rank\,\dagger} = \phi e^{\Omega t} b$$

[2] See HW 1 on PCA for details.

[3] A has dimensions n x n where n > r.

[4] Negative and Complex pixel values do not make sense, as pixel signals are always positive [0,255].

2) $X_{DMD}^{Sparse\,\dagger} = X - \left|X_{DMD}^{Low\ Rank\,\dagger}\right|$ (only real components)

3) $R \in \mathbb{R}^{m\,x\,n}$ where R contains the negative values of $X_{DMD}^{Sparse\,\dagger}$ and zero everywhere else.

4) $X_{DMD}^{Low\ Rank} = R + \left|X_{DMD}^{Low\ Rank\,\dagger}\right|$ and $X_{DMD}^{Sparse} = X_{DMD}^{Sparse\,\dagger} - R$

5) Now: $X = X_{DMD}^{Low\ Rank} + X_{DMD}^{Sparse} = X_{DMD}^{Low\ Rank\,\dagger} + X_{DMD}^{Sparse\,\dagger}$

These two new, real and nonzero lowrank and sparse matrices hold the vectors for the video streams of the foreground and backgrounds. The above DMD method provides a quick and relatively efficient means to separate the foreground and background of a video feed and generate two new videos of each.

Algorithm Implementation and Development

DMD Based Video Separation

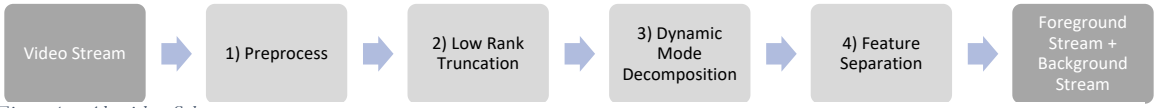

Figure 1 - Algorithm Scheme

In order to separate the two video streams, foreground and background, from an input video the algorithm defined in figure 1 was developed. First, a video stream is recorded on a smart phone or DSLR style camera. Then the video signal is preprocessed, converting from RBG to grayscale and down sampling the data. The down sampling step allows for the overall dimensions of the data to be decrease by one-half without sacrificing feature data; this step increases overall algorithm efficiency.

Next the DMD algorithm is executed in two steps. In the first step, step 2, a low rank truncation, using the concept of PCA analysis[5], is performed to only capture the low rank, unvarying, components of the signal. From here step 3, enacts the DMD algorithm defined in the theoretical section above, returning a not-normalized, lowrank version of the data. Lastly, step 4 performs the foreground background separation by subtracting the lowrank background from the data to gain the sparse foreground. The two signals are normalized to remove any negative or complex pixel signals. The two video separate data feeds are now stored in two separate video streams which can be visualized or further analyzed.

This algorithm has been implemented in a Matlab function called dmd_video_separate. The function takes a video stream as an input and returns the foreground and background video streams. The usage information can be found in appendix A and the source code can be found in appendix B.

Computational Results

The dmd_video_separate algorithm has been tested over a set of six video streams. The videos vary in scope, ranging from a video of vehicles on the highway to one of dogs running in a park. A reference frame from each video can be found in figure 2. The video set has a very diverse background and foreground space, in order to provide many scenarios to examine algorithm performance. The videos range from 3 – 5 seconds long and all are 540x960 pixels in size.

All six videos were run through the dmd_video_seperate, using a single mode truncation for the low rank truncation step. Figure 3, depicts the singular value spectra for all size videos, highlighting necessity for a single mode truncation. Using these low rank representations of the

[5] See PCA homework for details.

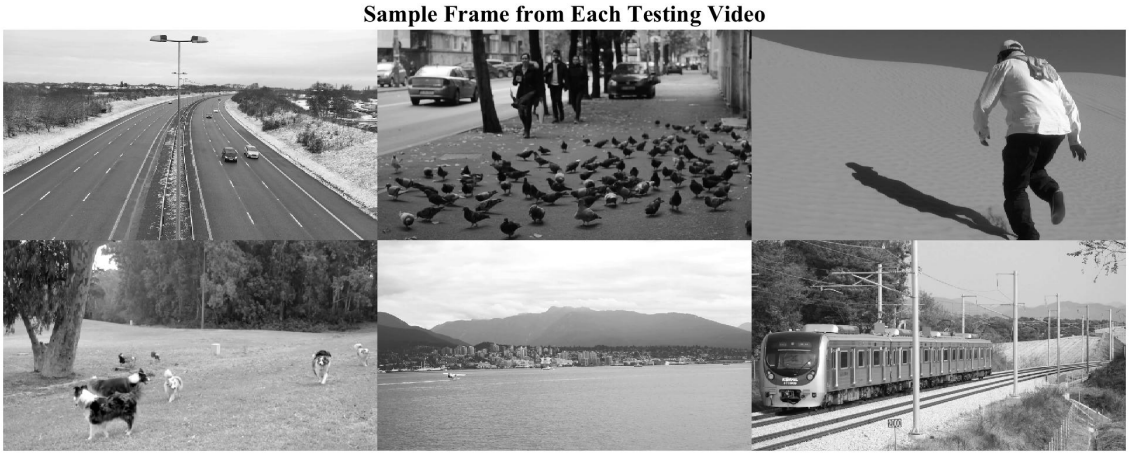

Figure 2 Sample frames from test video streams.

testing videos the DMD as defined above was used to separate the the foreground from the background. The raw results can be found in figure 4, where the original frame is show alongside the background and foreground frames. It is important to note that the foreground images were brightened in order to view the details of the frame. Generally, we can see that the algorithm manages to extract a majority of the foreground from the back ground, however is fails in some instances.

Four of the six videos appear to yield decent feature separation results when fed into the dmd_video_seperate algorithm. The foreground frames show little extraneous noise, although some is still observed, because the background may not be entirely captured by the single-mode low rank truncation. However, the *foreground* features, the person, plane, dogs and train are easily recognized in the foreground image. Furthermore, the background frames capture the background and show little of the original foreground features.

Although four of the six videos, yield decent separation results, two of them, the video of

Figure 4 Experimental results from dmd_video_seperate for the six test videos. The raw image can be found on the far left of each panel. The background is the middle frame. And the foreground is the rightmost frame.

Figure 3 Sample frames from test video streams.

the birds on the street, and the video of the cars on the highway show very poor results. In the case of the bird video, the foreground frame seems to select the street in the background, over the moving birds in the foreground. Furthermore, the foreground video resolves much of the detail of the background, when it is not supposed to. In the case of the car on the highway, the *foreground* features

are extracted quite well (the cars are seen in white), there is lots of extraneous noise which should not have been extracted (lane markings/plants etc). Observation of all frames may raise the question, why does the highway video show worse results that the train video, etc. In the video of the train, most of the noise in the foreground frame is darker gray than the train itself. We see this noise, because the image is brightened 4x to increase clarity of the foreground features. However, this saturation also highlights more background noise.

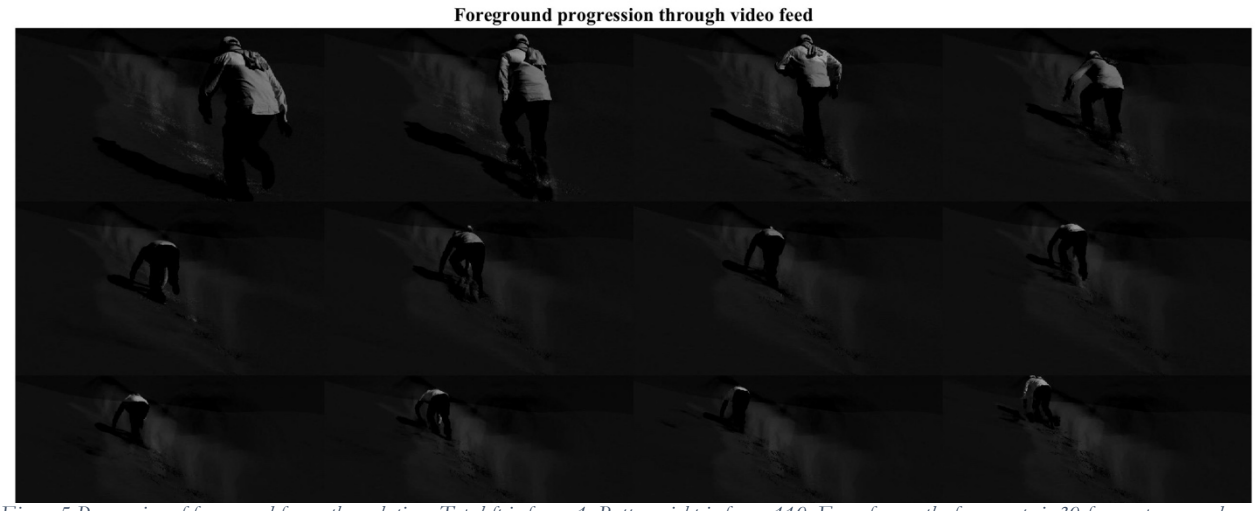

Figure 5 Progression of foreground frame through time. Top left is frame 1. Bottom right is frame 110. For reference the frame rate is 30 frames per second.

In order to observe how the separation algorithms performs across the entire video figure 5 depicts the foreground every 10 frames for one of the test videos. Here frame 1, the top left frame, shows the white shirt of the man in the foreground. As the video progresses, and the man runs into the plane of the frame we observe the white shirt in every frame, despite a general decrease in size. This shows that the algorithm is capable to extract the foreground features from a video with large spatial dynamics.

Overall, the dmd_video_separate algorithm is able to perform foreground/background separation tasks on a majority of the test videos, with some minor shortcomings.

Summary and Conclusions

Overall, an efficient and fairly robust foreground/background separation algorithm has been defined and implemented. The algorithm can be found in the Matlab function dmd_video_separate which is simple to use and take any common video file for the input video stream. All in all, the algorithm performs reasonable well for most background, foreground combinations examined, however it struggles with more heterogeneous backgrounds.

The algorithm performs quite well in scenarios where the foreground object is simple and the background relatively uniform. In these cases, the background is easily represented by one dominant mode, and the foreground moves relatively succinctly. This means that the low rank reconstruction captures a large proportion of the background and avoids potentially capturing the foreground. However, in more complex, more *real-world,* scenarios, the algorithm seems to capture more of the background in the foreground output feed. In a more complex background, there is a higher likelihood that objects move (they are no longer represented by one mode) and that they potentially vary with time. Furthermore, in the more complex case, shadows from the foreground

into the background, or changing illumination from the foreground object (in the case of automobile headlights) will cause the foreground feed to capture background information. All in all, the algorithm proposed is very rudimentary and could use some improvement additions, however, in its current state it succeeds at extracting basic foreground objects from the background in a video feed.

As I have implemented the dmd_video_separate algorithm I am going propose some future modifications which may drastically improve separation results. First, a thresholding step may be adding after the DMD step, in order to clean up and reduce the background noise of the foreground feed. Second, it may be possible to run the foreground feed through the algorithm again, to further extract background. This second background feed could be added back to the original background feed, to attempt to reconstruct a more robust background. Lastly, it may be possible to use more modes to represent the background layer, rather than just 1. I made the assumption that capturing a large proportion of the energy would yield good results, but further exploration into this may yield otherwise.

Appendix A – Matlab Function Repository

```
%% DMD Separation
[foreground, background, originalVideo] = dmd_video_separate('VIDEO FILE
INPUT')
    % This is the only usage option
    %Input is any video file that can be read by a matlab videoreader
    %object
    %Returns: the new foreground and background feeds, along with the
    %orginal video as an image matrix stack
    %NOTE: all outputs are of uint8 type
    %utilizes the DMD algorithm to separate a video feed.

%% Load a video
videoReadObj = VideoReader('INPUT VIDEO')
    %simplest usage
    %use VideoReader functions to read frames and get video feed statistics
    %See matlab documentation for details

%% Subsample an image
smallerImage = imresize(largeImage, [numrows, numcols])
    %simplest usage
    %will retain same image and proporitons, however will reduce pixel
    %count to numrows*numcols by averaging orginal pixels
        %pixel binning
    %see matlab documentation for details
```

Appendix B – Matlab Code-Bank

```
%Nathaniel Linden
%Foreground/Background Separation
%USES THE DMD ALGORITHM
%inputVideo is a video stream with a static background and a dynamic
%foreground
%returns:foregroundFeed and backgroundFeed (video feeds with same (x,y,t)
%dimensions as the input video but containing the foreground and background
%they are also uint8 arrays
function [foregroundFeed, backgroundFeed, fullFeed] =
dmd_video_separate(inputVideo)
    v1 = VideoReader(inputVideo);
    %get video stats
    vidHeight = v1.Height; time = v1.Duration; vidWidth = v1.Width;
    %down-sample dimensions
    sz1 = vidHeight/2; sz2 = vidWidth/2;
    %get frame count
    numFrms = round(v1.FrameRate*time);
    %create time vector
    dt = time/numFrms;
    t = [0:dt:time-dt];

    %load frames and preprocess
    VIDEO = zeros(sz1*sz2,int64(numFrms),'uint8');
    iter = 1;
    while hasFrame(v1)
            %load
            frame = readFrame(v1);
            %convert to grayscale
            frmBW = 0.299*frame(:,:,1) + 0.587*frame(:,:,2) +
0.115*frame(:,:,3);
            %subsample - decrease by 1/2
            frmBW = imresize(frmBW,[sz1,sz2]);
            VIDEO(:,iter) = frmBW(:);
        iter = iter+1;
    end

    %create data matrices
    X = double(VIDEO);
    X1 = X(:,1:end-1);
    X2 = X(:,2:end);

    %take the SVD
    [U, S, V] = svd(X1,'econ');

    %show user singular value spectra and get input for r
    figure(1)
    plot(diag(S)/sum(diag(S)),'ro')

    r = input('How many modes should be used?');
    close 1

    % DMD BODY

    %low rank truncate
    Ur = U(:,1:r);
```

```
    Sr = S(1:r, 1:r);
    Vr = V(:,1:r);

    %get Atilde
    Atilde = Ur' * X2 * Vr/Sr;

    %get eigen space of Atilde
    [W D] = eig(Atilde);
    Phi = X2 * Vr/Sr * W;
    lambda = diag(D);
    omega = log(lambda);

    %regress to b using psuedo inv
    x_1 = X(:,1); %x_1 is first frame
    b = pinv(Phi)*x_1;

    %get time dynamics
    time_dyn = zeros(r,length(t));
    for i = 1:length(t)
        time_dyn(:,i) = b .* exp(omega*t(iter));
    end
    %reconstruct DMD modes
    Xdmd_lr = Phi*time_dyn;

    % ***foreground background separation step***
    Xsparse1 = X - abs(Xdmd_lr); %abs to capture only real parts

    %use R to remove any negative pixel values
    R = Xsparse1;
    R(R > 0) = 0;

    %construct foreground and background
    XlowRank = (R + abs(Xdmd_lr)); %Xlowrank should capture background
    Xsparse = Xsparse1 - R; %Xspase should capture foreground

    %back to videos
    vidSparse = zeros(sz1,sz2,round(numFrms-1));
    vidLowRank = zeros(sz1,sz2,round(numFrms));
    vidFull = zeros(sz1,sz2,round(numFrms));
    for i = 1:numFrms-1
        vidSparse(:,:,i) = reshape(Xsparse(:,i),[sz1,sz2]);
        vidLowRank(:,:,i) = reshape(XlowRank(:,i),[sz1,sz2]);
        vidFull(:,:,i) = reshape(X(:,i),[sz1,sz2]);
    end

    backgroundFeed = uint8(vidLowRank);
    foregroundFeed = uint8(vidSparse);
    fullFeed = uint8(vidFull);
end
```

```
%Nathaniel Linden
%TEST dmd_video_separate


[F722729280, B722729280, full722729280] =
dmd_video_separate('722729280.mp4');
[F716207682, B716207682, full716207682] =
dmd_video_separate('716207682.mp4');
[F636797844, B636797844, full636797844] =
dmd_video_separate('636797844.mp4');
[F703393635, B703393635, full703393635] =
dmd_video_separate('703393635.mp4');
[F717955517, B717955517, full717955517] =
dmd_video_separate('717955517.mp4');
[F796795983, B796795983, full796795983] =
dmd_video_separate('796795983.mp4');
%% Visualize Stuff
%raw videos
f = figure(1);

row1 =
horzcat(full636797844(:,:,1),full703393635(:,:,1),full716207682(:,:,1));
row2 = horzcat(full722729280(:,:,1),
full796795983(:,:,1),full717955517(:,:,1));
grid = vertcat(row1,row2);
imshow(grid);
title('Sample Frame from Each Testing Video','FontSize',20,'FontName','Times
New Roman');

 %%
%good three
f = figure(2);

row1 = horzcat(F716207682(:,:,1), F716207682(:,:,10),F716207682(:,:,20),
F716207682(:,:,30));
row2 = horzcat(F716207682(:,:,40), F716207682(:,:,50),F716207682(:,:,60),
F716207682(:,:,70));
row3 = horzcat(F716207682(:,:,80), F716207682(:,:,90),F716207682(:,:,100),
F716207682(:,:,110));
grid = vertcat(row1, row2, row3);
imshow(grid)
title('Foreground progression through video
feed','FontSize',20,'FontName','Times New Roman')

%%
figure(3)

ha = tight_subplot(3,2,[.01 .01],[.01 .01],[.01 .01]);
axes(ha(1)); imshow(horzcat(full636797844(:,:,100),
B636797844(:,:,100),F636797844(:,:,100)*4))
axes(ha(2)); imshow(horzcat(full703393635(:,:,100),
B703393635(:,:,100),F703393635(:,:,100)*4))
axes(ha(3)); imshow(horzcat(full716207682(:,:,100),
B716207682(:,:,100),F716207682(:,:,100)*4))
axes(ha(4)); imshow(horzcat(full722729280(:,:,100),
B722729280(:,:,100),F722729280(:,:,100)*4))
```

```
axes(ha(5)); imshow(horzcat(full796795983(:,:,100),
B796795983(:,:,100),F796795983(:,:,100)*4))
axes(ha(6)); imshow(horzcat(full717955517(:,:,100),
B717955517(:,:,100),F717955517(:,:,100)*4))
```