Background Subtraction in Video Streams

Zhiyang Cheng

March 17, 2021

1 Introduction and Overview

In this report we are using the Dynamic mode decomposition method. Dynamic mode decomposition (DMD) is a dimensional reduction algorithm. We would apply it on the given video clips ski drop low.mp4 and monte carlo low.mp4 which contain a foreground and background object. We want to separate the video stream to both the foreground video and a background.

2 Theoretical Background

2.1 The Singular Value Decomposition - SVD

The Singular Value Decomposition (SVD) of a matrix is a factorization of that matrix into three matrices.

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

The singular value decomposition of an m$\times$ n complex matrix M is a factorization of the form $U\Sigma V^*$.
U is an $m \times m$ complex unitary matrix, Σ is an $m \times n$ rectangular diagonal matrix with non-negative real numbers on the diagonal, and V is an $n \times n$ complex unitary matrix.

2.2 Dynamic mode decomposition (DMD)

Dynamic mode decomposition (DMD) is a dimensional reduction algorithm. Given a time series of data, DMD computes a set of modes each of which is associated with a fixed oscillation frequency and decay/growth rate. For linear systems in particular, these modes and frequencies are analogous to the normal modes of the system, but more generally, they are approximations of the modes and eigenvalues of the composition operator (also called the Koopman operator).
The DMD spectrum of frequencies can be used to subtract background modes. Specifically, assume that p, where p in 1,2,..., satisfies wp = 0, and that wj for any j that is not equal to p is bounded away from zero. Thus,

$$X_{DMD} = b_p \varphi_p e^{w_p t} + \sum_{j \neq 1} b_j \varphi_j e^{w_j t} \tag{2}$$

Assuming that X in Rn$\times$m, then a proper DMD reconstruction should also produce X_{DMD} in Rn$\times$m. However, each term of the DMD reconstruction is complex, though they sum to a real-valued matrix. This poses a problem when separating the DMD terms into approximate low-rank and sparse reconstructions because real-valued outputs are desired and knowing how to handle the complex elements can make a significant difference in the accuracy of the results. Consider calculating the DMD's approximate low-rank reconstruction according to

$$X_{DMD}^{LowRank} = b_p \varphi_p e^{w_p t} \tag{3}$$

Since it should be true that

$$X = X_{DMD}^{LowRank} + X_{DMD}^{Sprase} \tag{4}$$

then the DMD's approximate sparse reconstruction,

$$X_{DMD}^{Sprase} = \sum_{j \neq 1} b_j \varphi_j e^{w_j t} \tag{5}$$

However, this may result in X_{DMD}^{Sprase} having negative values in some of its elements, which would not make sense in terms of having negative pixel intensities. These residual negative values can be put into a n×m matrix R and then be added back into $X_{DMD}^{LowRank}$ as follows:

$$X_{DMD}^{LowRank} \leftarrow R + X_{DMD}^{LowRank} \tag{6}$$

$$X_{DMD}^{Sprase} \leftarrow X_{DMD}^{Sprase} - R \tag{7}$$

This way the magnitudes of the complex values from the DMD reconstruction are accounted for, while maintaining the important constraints that

$$X = X_{DMD}^{LowRank} + X_{DMD}^{Sprase} \tag{8}$$

so that none of the pixel intensities are below zero, and ensuring that the approximate low-rank and sparse DMD reconstructions are real-valued.

3 Algorithm Implementation and Development

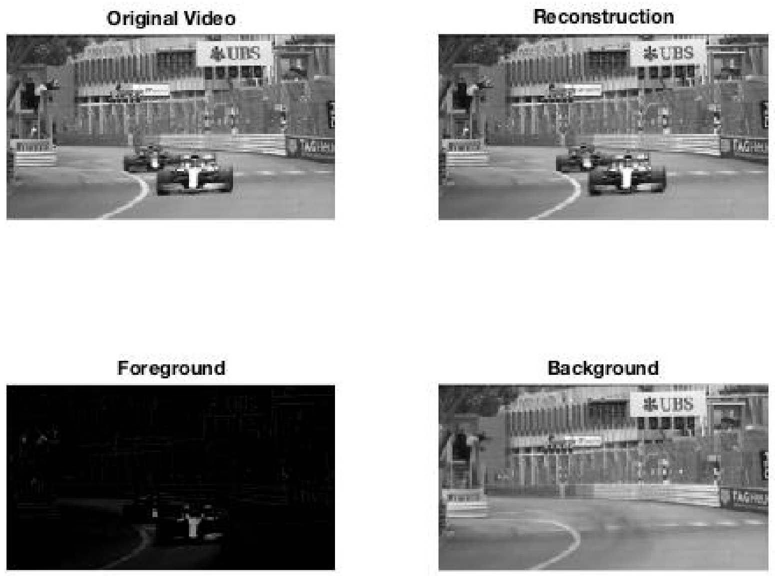
3.1 Setting up and Analyze

We can first Load the data from the provided video clips. Through the command videoreader we can obtain the length and the number of the frames. We then convert video to gray scale using the matlab command. Next we will use the svd command to take SVD on X and then use low rank approximation to get the rank. For this assignment, we use 2 for numbers of the significant modes. We then preform a eigen-decomposition on a a matrix who is similar to S getting its eigenvalue and eigen vector. We then calculate the DMD solutions. We lastly subtract the background video clips from the original video clips to get the foreground video clips. In the end plot out what we have for original video, reconstruction video, background video, and foreground video.

4 Computational Results

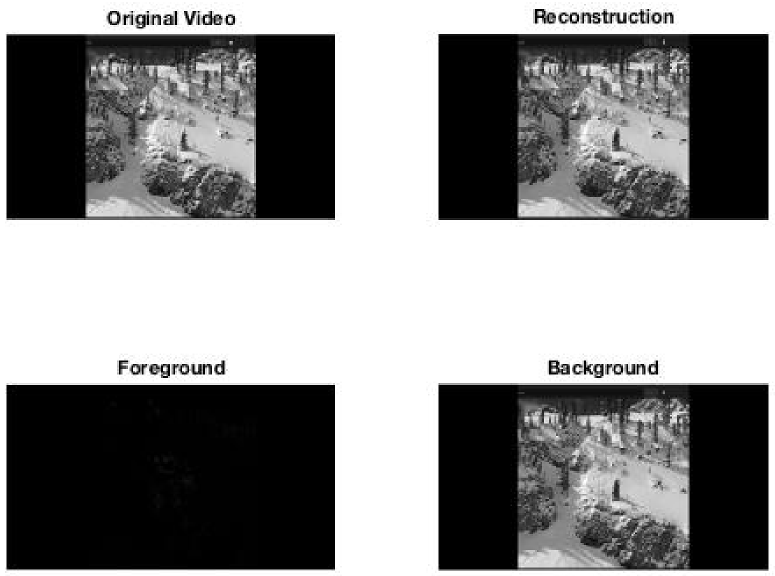
4.1 monte carlo.mov

Based on the graph below, we have four pictures for the original video monte carlo.mov, reconstruction video, background video, and foreground video. And based on the pictures we can see that the foreground (the car) separates from the background pretty well. With a similar construction towards the original video, DMD did a good job for this one.



4.2 ski drop.mov

Based on the graph below, we have four pictures for the original video ski drop.mov, reconstruction video, background video, and foreground video. Based on the pictures we can see that the the foreground (the tourist) does not separate from the background fully. It might because of the relatively smaller size of our target so DMD did not do a good job as the previous one.



5 Summary and Conclusions

In this assignment, we use SVD capturing energy and Dynamic mode decomposition (DMD) to separate our foreground and background given the original video clips. For both video, I am satisfied with the results for the reconstructions which both look similar to the original video. In terms of separating, dynamic mode decomposition did a better job using monte carlo.mov. But I think DMD works well for computing a set of modes each of which is associated with a fixed oscillation frequency and decay/growth rate.

6 References

https://en.wikipedia.org/wiki/Singular$_{v}alue_{d}ecomposition$
$https://en.wikipedia.org/wiki/Dynamic_{m}ode_{d}ecomposition: \quad :text = Dynamic$

Appendix A MATLAB Functions

- `[Videoreader` creates a object containing the information likes the length of the input video data
- `[U,S,V] = svd(x)` performs a singular value decomposition of matrix x
- `rgb2gray(x)` covert image x to gray scale of dimension width * height

Appendix B MATLAB Code

```
clear all;
close all;
clc

% Load Data Import
v1 = VideoReader('monte_carlo_low.mp4');
v2 = VideoReader('v2_drop_low.mp4')

dt = 1/v1.Framerate;
t = 0:dt:v1.Duration;
vid1 = read(v1);
num = get(v1, 'numberOfFrames');

dt = 1/v2.Framerate;
t = 0:dt:v2.Duration;
vid2 = read(v2);
num = get(v2, 'numberOfFrames');

for i = 1:num
    mov(i).cdata = vid2(:,:,:,i);
    mov(i).colormap = [];
end

images = [];

images1 = images(:,1:end-1);
images2 = images(:,2:end);

[U,Sigma,V] = svd(images1,'econ');

energy = 0;
modes = 0;
while energy < 0.9
    modes = modes + 1
    energy = energy + (sig(modes).^2/sum(sig.^2))
end

r = 2;
Ur = U(:,1:r);
Sigmar = Sigma(1:r,1:r);
Vr = V(:,1:r);

S = Ur'*images2*Vr*diag(1./diag(Sigmar));


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


y0 = Phi\images1(:,1);
```

```
imagesmodes = zeros(length(y0),length(t)-1);
for iter = 1:(length(t)-1)
    imagesmodes(:,iter) = y0.*eimagesp(omega*t(iter));
end
imadmd = Phi*imagesmodes;


sparse = images1-abs(imadmd);

R = sparse.*(sparse<0);

images_low = R + abs(imadmd);
imagesp = sparse-R;
images_r = images_low + imagesp;

% image_con = reshape(images_r, [135,240,378]);
% image_back = reshape(images_dmd, [135,240,378]);
% image_fore = reshape(images_sparse, [135,240,378]);
% image_ori = reshape(images1, [135,240,378]);

image_con=reshape(images_r, [135,240,453]);
image=uint8(real(image_con));
imshow(image);drawnow


image_back=reshape(imadmd, [135,240,453]);
image=uint8(real(image_back));
imshow(image);drawnow


image_fore=reshape(imagesp, [135,240,453]);
image=uint8(real(image_fore));
imshow(image);drawnow


image_ori=reshape(images1, [135,240,453]);
image=uint8(real(image_ori));
imshow(image);drawnow
```