Background Subtraction in Video Streams using Dynamic Mode Decomposition

Timur Zhanabaev

MAST 680 - Assignment 1 - February 7, 2023

Abstract

In the assignment, the dynamic mode decomposition method is implemented to reduce video frames into their modes and eigenvalues, from which the background modes can be subtracted to produce foreground footage. The method works well for footage composing of a static view of a dynamical object of relatively great contrast with it's background, which then segments the dynamical object across frames.

Section I. Introduction and Overview

Considered to be a simple, yet effective method to forecast data in the very short term future, the dynamical mode decomposition finds an approximation of a non-linear or a linear dynamical system, by iterating via linear mappings, called DMD matrix A, for each time step.[1] Further, an eigen-decomposition of the DMD matrix yields the modes of the system, essentially performing a discrete Fourier transform of the dynamical system [2]. The objective is then to filter out the near-zero frequencies of the mode decomposition that makes up the background video-stream.

Section II. Theoretical Background

Dynamic Mode Decomposition

Given a data set $X' \in \mathbb{R}^{M\times N}$. In our scenario, M is the number of pixels and N is the number of snapshots of the video. Following, the $Y \in \mathbb{R}^{M\times N}$ matrix is defined as the snapshot matrix $\triangle t > 0$ time away from the snapshot matrix X'. Then with given data set, we define $X = X' \in \mathbb{R}^{M\times T(t)}$, where $T(t) \in \{1, ..., N-1\}$ and $Y = X' \in \mathbb{R}^{M\times T(t+\triangle t)}$, where $T(t+\triangle t) \in \{2, ..., N\}$.

The point is then to find the DMD matrix that maps snapshots at time t to time $t + \triangle t$, called matrix $A \in \mathbb{R}^{M\times M}$. The minimization problem is then,

$$\text{argmin}_{A\in\mathbb{R}^{M\times M}} ||Y - AX||_F^2, \text{ where } F \text{ denotes the Frobenius norm}$$

This yields the exact solution to A being, $A = YX^\dagger$, where $X^\dagger$ is the Moore-Penrose pseudo-inverse, computed using the singular value decomposition method. In other words,

$$X = U\Sigma V^\top \implies X^\dagger = V\Sigma^\dagger U^\top \tag{1}$$

$$A = YX^\dagger = YV\Sigma^{-1}U^\top \tag{2}$$

Having found the DMD matrix A, now a linear discrete dynamical system, the system can then be iterated from an initial condition $Z_0 \in \mathbb{R}^{M\times 1}$ (the first snapshot of the video), to each next time step by equation,

$$Z_{n+1} = AZ_n \tag{3}$$

Although in practicality, $A \in \mathbb{R}^{M\times M}$ is a large matrix for image files, with resolutions giving millions of pixels, that can easily reach the limits of computer's memory. To circumvent this limitation, the **Exact DMD** [1] method is used, where the DMD matrix is defined instead as,

$$\tilde{A} = U^\top YV\Sigma^{-1} = U^\top AU, \text{, s.t. } \tilde{A} \in \mathbb{R}^{N\times N} \tag{4}$$

The limitation then rests on N the number of snapshots instead.

Mode Decomposition

Following, a subsequent procedure is made on the DMD matrix to find the DMD modes. If the matrix A is diagonalizable, one can obtain N eigenvalue-eigenvector pairs $(\mu_m, \varphi_m) \in \mathbb{C} \times \mathbb{C}^M$. As well, the relationship to the eigen-basis for $\tilde{A}$ is, [1]

$$A\varphi = \mu\varphi \iff \tilde{A}w = \mu w, , \text{ where } \varphi = Uw \tag{5}$$

The system at time n can then be reconstructed directly from the eigenvalues and eigenvector basis and the initial condition Z_0 (first frame),

$$Z_n = \sum_{m=1}^{M} \mu_m^n b_m \varphi_m \tag{6}$$

Where $(\mu_m^n, b_m, \varphi_m) \in \mathbb{C} \times \mathbb{C} \times \mathbb{C}^M$, and hence $Z_n \in \mathbb{C}^M$ The amplitudes of modes can calculated as [2],

$$b = (\varphi^* \varphi)^{-1} \varphi * Z_0 \tag{7}$$

Hence with $|\mu_m| \neq 1$, the values will diverge to 0 or ∞ for a $n \to \infty$.

Then to filter out the background DMD modes, the frequencies ω_m are computed from the eigenvalues μ_m as,

$$\omega_m = \ln(\mu_m)/\triangle t, , \text{ and modulus } |\omega_m| = \sqrt{\text{Re}(\omega_m)^2 + \text{Im}(\omega_m)^2} \tag{8}$$

Background Removal

The objective using DMD is then to remove the static background from a movie. The near-zero frequencies $|\omega_m| \approx 0$ or $|\mu_m| \approx 1$ are then considered to be related to the background modes composing the video. A low-rank matrix is constructed from the modes, defined as,

$$X^{(t)}_{\text{LR-DMD}} = \exp(\omega_p t) b_p \varphi_p \text{ , where } p \in \{p : |\omega_p| \approx 0\} \tag{9}$$

The foreground of the video is then defined as the sparse matrix, with the modulus of low-rank matrix subtracted,

$$X_{\text{Sparse-DMD}} = X - |X_{\text{LR-DMD}}| \tag{10}$$

Further more, the sparse matrix will now possibly contain negative values after subtracting $|X_{\text{LR-DMD}}|$. Hence, the negative residuals correction is made on both matrices, to low-rank and sparse. With matrix R being the negative residuals found in $X_{\text{Sparse-DMD}}$,

$$X_{\text{Sparse-DMD}} \leftarrow X_{\text{Sparse-DMD}} - R \tag{11}$$

$$X_{\text{LR-DMD}} \leftarrow |X_{\text{LR-DMD}}| + R \tag{12}$$

The final video without the background is $|X_{\text{Sparse-DMD}}|$, as to get rid of complex values for actual video-stream.

Section III. Algorithm Implementation and Development

Development

First off, for the video processing. The python package OpenCV, called as `cv2`, is used to load the videos in python and encode the foreground videos. The colored videos are then in 3 color channels RGB. The RGB values are transformed to grayscale values using the weighted method, reducing the 3 color dimensions to 1.

weighted method: `Grayscale` = 0.299 `R` + 0.587 `G` + 0.114 `B`

For the videos, 2 videos are tested on to remove their backgrounds. The first video `monte_carlo.mov` consists of a static view of F1 race cars passing through a race track bend. The resolution of the video is (1872×1112) at around 60 frames per second for approximately 6.3 seconds. The second video `ski_drop.mov` consists of a static aerial view of a skier going down a mountain slope. The resolution of the video is (1466×1604) at around 60 frames per second for approximately 7 seconds.

For the DMD method, 360 frames are then taken from both videos to create a 6 second videos without their backgrounds. For the rest, the `numpy` package is used to perform all the linear algebra; storing the videos in `numpy` matrices, reshaping for DMD, and implementing DMD. Notable premade functions from `numpy.linalg` include; `svd`, `pinv` and `eig` to implement DMD.

DMD variant choice

As a side note, comparing 1080p resolution (2073600 pixels) with a video of 60 fps. A video has to be of 576 seconds (9.6 minutes), for A and $\tilde{A}$ matrices to be of same size.

Hence, instead of calculating A matrix (2081664×2081664 for `monte_carlo.mov`), which will require 1.96 Tb of RAM. The DMD matrix $\tilde{A}$ is instead computed, with it's dimension being 360×360 instead. This turns the method feasible, as the next step is requiring the eigendecomposition of $\tilde{A}$.

Frequencies filtering

While the frame rate is 60Hz for the videos, the eigenvalue moduli are found around a complex circle of $r = 1$, hence $\triangle t = 1/60$ is taken as a time difference. Moreover, the search is done considering the $|\omega_p|$ closest to 0, i.e. $\text{argmin}_p\{||\omega_p||\}$.

Figures

Finally, some plots are produces as well from the development. Using the package `matplotlib`, figures are created of; eigenvalue polar plots, frequencies modulus and their real and imaginary parts side by side, as well as select frames from the footage to showcase background removal.

Algorithm Implementation

For the algorithm implementation, pseudo-code of reduced $\tilde{A}$ DMD [1, 2] is presented below (see section V. for function explanations),

```
DMD algorithm:

initialize X and Y
U, S, Vh = svd(X)
B = Y @ Vh.T @ pinv(np.diag(S)) (.T denotes a transpose)
A = U.T @ B
lambda, eigenvectors = eig(A)
modes = U.T @ eigenvectors
b = pinv( phi.H @ phi) @ phi.H @ X[0] (.H denotes a conjugate transpose)
p = index of near zero omega
BG[t] = exp( (log(lambda[p])/delta_t)*t) * b[p] * modes[p] (background at time t)
X_sparse = X - modulus(BG) (sparse X at time t)
R = X_sparse[X_sparse < 0]
FG_t = X_sparse - R (foreground at time t)
```

Section IV. Computational Results

For the initial results, eigenvalue Argand diagrams and ω frequency plots are presented for `monte_carlo.mov` 1. The modulus of eigenvalues all tend to be lower than 1 (decaying). In `ski_drop.mov`, the eigenvalues are mostly close to 1 2. Observing the final results of foreground footage, this seems to indicate that the background separation is working better for `ski_drop.mov`. The lowest ω seem to be the first by indices, which may indicate the first frames being critical to defining the background.

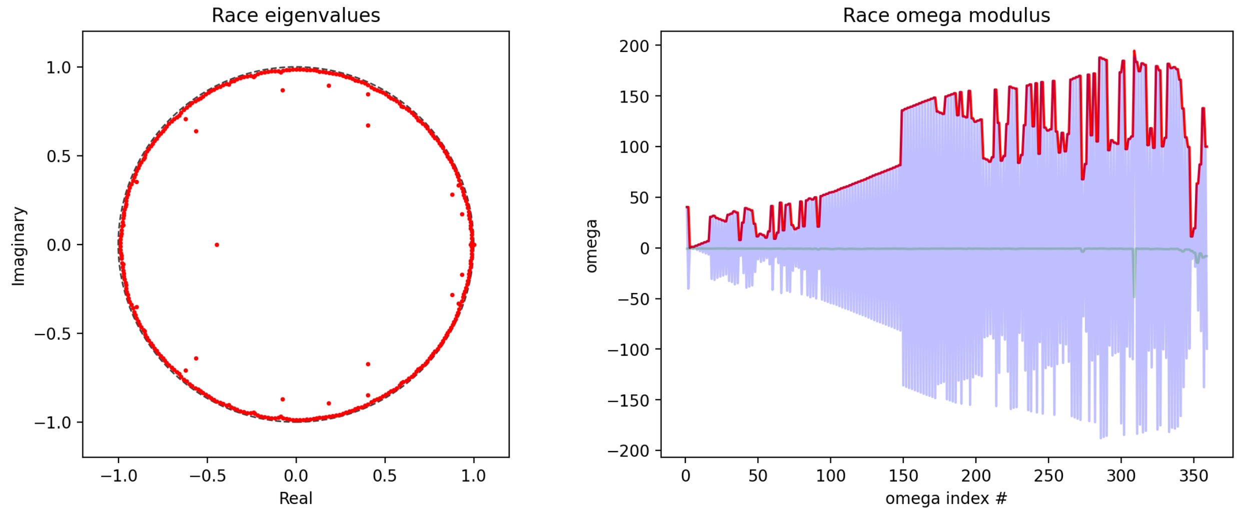

Figure 1: (left) `monte_carlo.mov` eigenvalue Argand diagram. (right) ω plots; modulus (red), Re(ω) (green) and Im(ω) (blue).

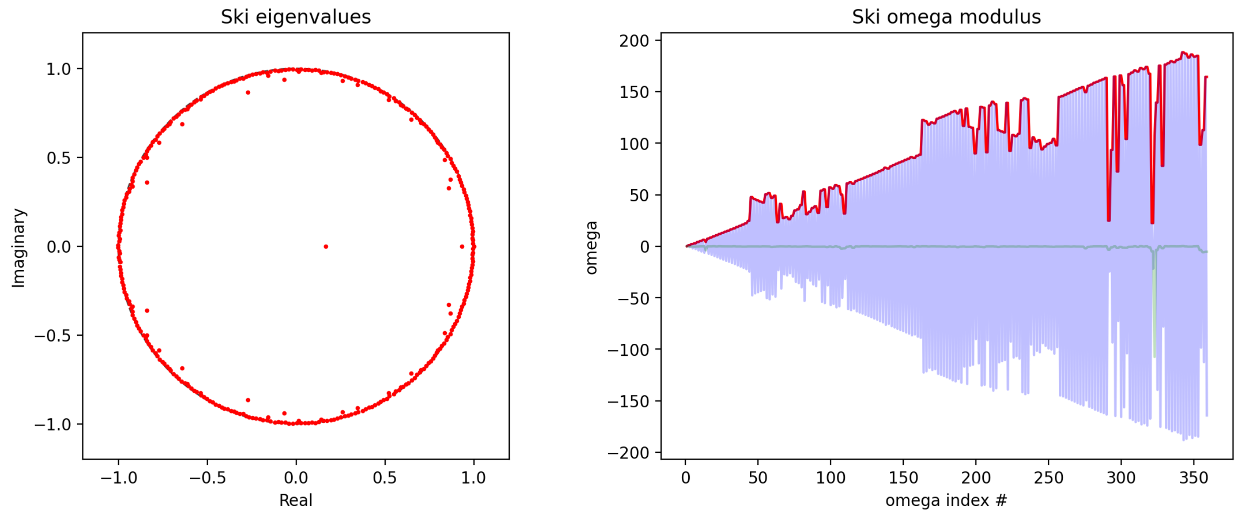

Figure 2: (left) `ski_drop.mov` eigenvalue Argand diagram. (right) ω plots; modulus (red), Re(ω) (green) and Im(ω) (blue).

Finally, some initial frames (frames 1 and 99) are presented for both videos in initial grayscale frames and the processed video's without the background (foreground). Here is shown snapshots for `monte_carlo.mov`. 3 (`ski_drop.mov` foreground figures in the appendix C. 4)

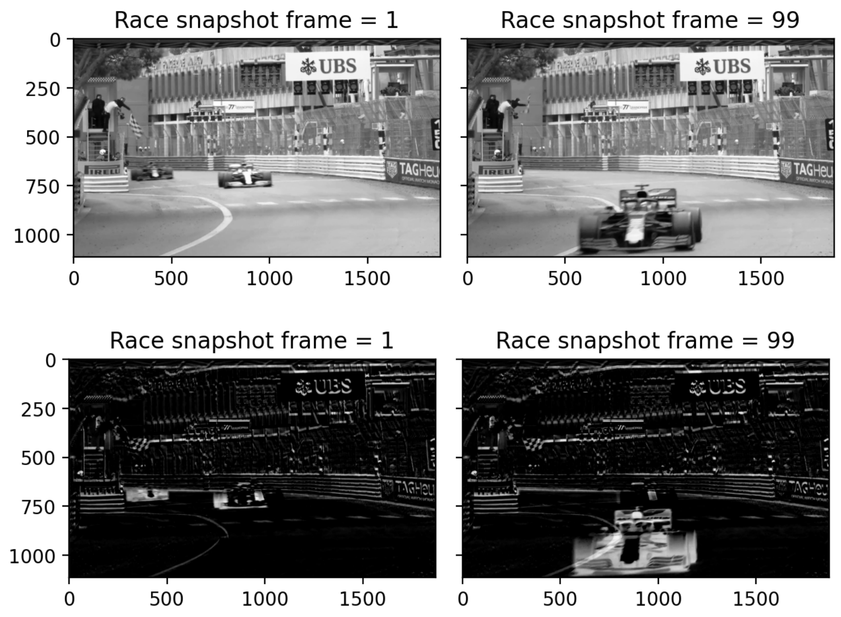

Figure 3: `monte_carlo.mov` snapshots of frames 1 and 99 (top), and snapshots of foreground (bottom).

Section V. Summary and Conclusions

Looking through the final footage, while the method worked decently well, as most of the background is suppressed, with the dynamic object in white. As in the race footage, where the passing car's contrasting region (car's pixel grayscale values being more different from the road's) come out well in the foreground. Since a black moving object in a black screen would simply just be a background itself. However, the video is shaky, so we see the background's edges come out in the foreground video since it becomes a moving object. Finally the worst aspect is the subtracted image of a car in the foreground (the background mode captured one of the car's instance as part of background). On the other hand, the ski footage performs much better. The falling snow's shadow creating contrast between white snow and the skier dropping down the mountain is perfectly separated from the static background. As well, since the camera was not shaking, there is minimal edge formation.

Links to videos

`race video: https://youtu.be/q2k3fxZq-MU`
`ski video: https://youtube.com/shorts/jeI4oqoDOeM`

References

[1] J. H. Tu, C. W. Rowley, D. M. Luchtenburg, S. L. Brunton, and J. Nathan Kutz. On dynamic mode decomposition: Theory and applications. Journal of Computational Dynamics, 1(2):391–421, 2014.

[2] J. Grosek and J. N. Kutz. Dynamic mode decomposition for realtime background/foreground separation in video. arXiv preprint arXiv:1404.7592, 2014.

Appendix A.1 Premade functions used and brief implementation explanation.

The video processing package is not going to be covered here, only the few select functions from `numpy`.

package `numpy.linalg` functions

1. `pinv`: computes pseudoinverse using the svd method. Used instead of the `inv` for computational speed and for non-square matrices.
2. `svd`: computes the $U, \Sigma, V*$ matrices for the DMD framework/components to manipulate DMD modes, etc.
3. `eig`: computes the complex valued eigenvalue, eigenvectors of a matrix.

Appendix A.2 Extra Figures

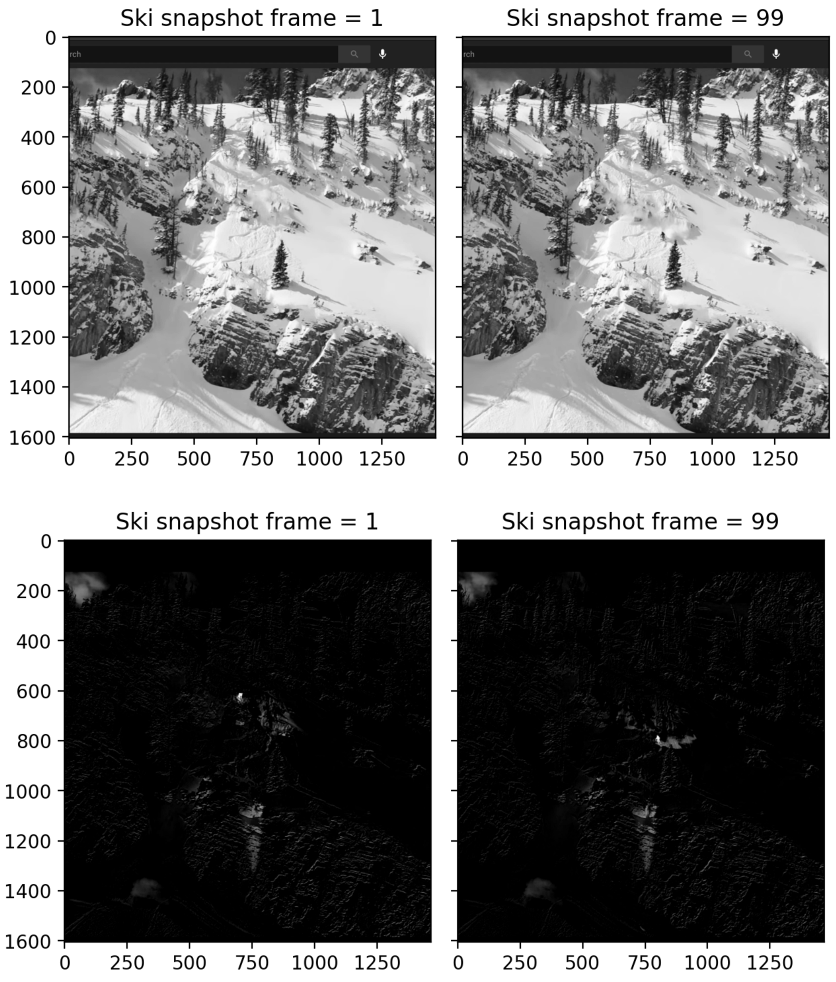

Figure 4: `ski_drop.mov` snapshots of frames 1 and 99 (top), and snapshots of foreground (bottom).

Appendix B. Computer codes