Video Background Subtraction With Dynamic Mode Decomposition

Arnold Peter Ruymgaart*

University of Washington
apruymgaart@gmail.com github udemy

March 20, 2021

Abstract

In this study we will subtract stationary background from videos of moving objects using Dynamic Mode Decomposition (DMD). We approximate the unknown dynamics with a linear system by organizing the image sequence into two sets related by the linear difference equation allowing us to solve for its matrix by least squares. We then discard the low frequency mode corresponding to the static background.

I. Introduction & Overview

Dynamic Mode Decomposition (DMD) allows us to describe a high dimensional system with unknown dynamics by a low rank approximation of dynamic modes [Tu, 2014]. In turn this allow us to subtract the background mode from a video of a moving object with stationary background [Grosek, 2014].

In this paper we'll briefly review the necessary theory to understand & implement DMD in Section III. Then in Section III we'll explain the algorithm implementation in detail. The results of using DMD to perform background subtraction in two movies are included in Section IV.

II. Theoretical Background

In order to accomplish our background subtraction task, the following principles are needed and briefly reviewed:

A. Linear dynamics

B. Dynamic Mode Decomposition (DMD)

C. Video background subtraction (using DMD)

(A) Linear dynamics. In this report we make the assumption the dynamics observed in the videos can be approximated linearly. For a linear dynamical system, the differential equation and discrete time difference equation are written as follows:

$$\frac{d}{dt}\vec{x} = A\vec{x} \tag{1}$$

$$\vec{x}_{k+1} = A\vec{x}_k \tag{2}$$

*Contact for further information

Such a linear system has the general solution

$$\vec{x}(t) = \sum_j b_j \vec{\phi}_j e^{\lambda_j t} \tag{3}$$

where if A is diagonalizable $A = WAW^{-1}$ the vectors $\vec{\phi}$ are eigenvectors of A.

(B) Dynamic Mode Decomposition (DMD). From the difference equation (Eqn. 2), we have $\vec{x_1} = A\vec{x}_0$ and $\vec{x_2} = A\vec{x}_1$ and so on. We describe the system progression in time as follows

$$(\vec{x}_1\ \vec{x}_2\ \dots\ \vec{x}_n) = (A_0\vec{x}_0\ A_1\vec{x}_1\ \dots\ A_{n-1}\vec{x}_{n-1}) \tag{4}$$

$$X_2 = AX_1 \tag{5}$$

$$X_2X_1^\dagger = A \tag{6}$$

In the last two steps we approximate $A \simeq A_i, \forall i$ by pseudo-inverse (least squares) of X_1. Where psuedoinverse $X_1^\dagger = X_1^*(X_1X_1^*)^{-1}$. So $A = X_2X_1^\dagger$ is the best general match to any A_i that will advance the system one time step having taken into account all steps. In Eqn. 5 above, we organized the time-sequence data vectors $\vec{x}_i$ into two matrices and this is the first step in DMD:

$$X_1 = (\vec{x}_0\ \vec{x}_1\ \dots \vec{x}_{m-1}),\ X_2 = (\vec{x}_1\ \vec{x}_2\ \dots \vec{x}_m) \tag{7}$$

Where X_2 is essentially X_1 "shifted" by one time step.

The psuedoinverse of X_1 is typically huge. If $\vec{x}$ is a (vectorized) movie image, its length may be on the order of 10^6 and dimensions of $X_1^\dagger$ the square of that. So we

will now find a low rank optimal representation of X_1:

$$X_1 = U\Sigma V^* \tag{8}$$

And the inverse of the above equals the psuedoinverse (SVD allows easy calculation it):

$$X_1^\dagger = (U\Sigma V^*)^{-1} = (V^*)^{-1}\Sigma^{-1}U^{-1} = V\Sigma^{-1}U^* \tag{9}$$

Then

$$A = X_2 X_1^\dagger = X_2 V\Sigma^{-1}U^* \tag{10}$$

Now consider matrix A. Any factorization $Y^{-1}AY$ is called a similarity transform of A. Any matrix $\tilde{A}$ that is similar to A has the same eigenvalues. We do *similarity transform* U^*AU in order to represent it in reduced rank:

$$\tilde{A} = U^*AU \tag{11}$$
$$= U^*(X_2 V\Sigma^{-1}U^*)U \tag{12}$$
$$= U^*X_2 V\Sigma^{-1} \tag{13}$$

The final step is an eigendecomposition of $\tilde{A}$ such that $\tilde{A}\vec{w}_i = \lambda_i\vec{w}_i$ and all eigenvectors are arranged into W we have $\tilde{A}W = \Lambda W$. and DMD modes ϕ are

$$\tilde{\phi}_i \equiv U\vec{w}_i \tag{14}$$

In order to use the DMD modes we must transform (project U) them back to the original coordinate system

$$\Phi = X_2 V\Sigma^{-1}W \tag{15}$$

We can now reconstruct the dynamics from low rank approximation. With the general solution Eqn. 3 we can write for one time point:

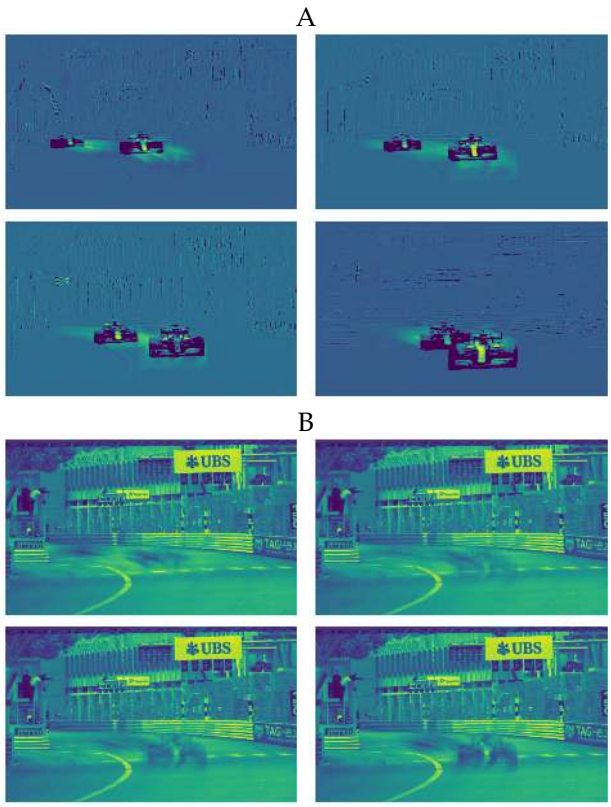
$$\vec{x}(t) = \Phi e^{\Omega t}\vec{b} \tag{16}$$
$$= \sum_{k=1}^{r} \vec{\phi}_k e^{\omega_k t} b_k \tag{17}$$

where $e^{\Omega t}$ is a diagonal matrix with scalar quantities $e^{\omega_k t}b_k$ on the diagonal corresponding to mode k. For example, if we have 2 modes:

$$x(t) = \begin{pmatrix}\vec{\phi}_1 & \vec{\phi}_2\end{pmatrix}\begin{pmatrix}e^{\omega_1 t} & 0\\ 0 & e^{\omega_2 t}\end{pmatrix}\begin{pmatrix}b_1\\ b_2\end{pmatrix} \tag{18}$$
$$= \vec{\phi}_1(b_1 e^{\omega_1 t}) + \vec{\phi}_2(b_2 e^{\omega_2 t}) \tag{19}$$
$$= \vec{\phi}_1(b_1|\omega_1|e^{\theta_1 t}) + \vec{\phi}_2(b_2|\omega_2|e^{\theta_2 t}) \quad (i) \tag{20}$$

(i) complex number in polar form: $\omega_k = p_k + q_k i$ is complex valued, so we can write as $|\omega_k|e^{i\theta_k}$. As time progresses b_1, b_2 and $\vec{\phi}_1, \vec{\phi}_2$ don't change in time. Nor do moduli $|\omega_k|$.

Figure 1: *Selected Frames, Foreground & Background Movie* 1



(A) Foreground of frames $0, 15, 30$ and 45 of the movie *monte_carlo.mov*. **(B)** Background of frames $0, 15, 30$ and 45 of the movie *monte_carlo.mov*.

We can also calculate all time images $\vec{x}(t)$ at once with the matrix product ΦB_{ω^t}:

$$X = (\vec{x}(t_0)\ \vec{x}(t_1)\ \dots) = \Phi B_{\omega^t} \tag{21}$$

where

$$B_{\omega^t} = \begin{pmatrix} b_1 e^{\omega_1 t_0} & b_1 e^{\omega_1 t_1} & \dots \\ b_2 e^{\omega_2 t_0} & b_2 e^{\omega_2 t_1} & \dots \\ \vdots & \vdots & \ddots \end{pmatrix} \tag{22}$$

Since $e^0 = 1$ (and $t_0 = 0$), the diagonal matrix $e^{\Omega 0} = I$ equals identity at $t = 0$ and Eqn. 16 becomes $\vec{x}_0 = \Phi b$. So we can obtain $\vec{b}$ by solving the linear system $\Phi b = \vec{x}(t_0)$. This system is overdetermined so we can use a least squares or QR factorization solver.

(C) Video background subtraction (using DMD) . With the assumption that background does not change in time (or very little), all background should be in low frequency dynamic mode(s). So background subtraction is a matter of splitting the low rank approximation Equation 16 into two parts: one mode with index p

representing background and the rest

$$\vec{x}(t) = \vec{\phi}_p(b_p e^{\omega_p t}) + \sum_{k\neq p}^{r} \vec{\phi}_k(e^{\omega_k t} b_k) \tag{23}$$

where the background mode has frequency near zero $||\omega_p|| \simeq 0$. Video images $\vec{x}$ are expected to be real valued. But the individual summation terms in Eqns. 16 and 23 are complex valued as emphasized in Eqn. 20. If

$$X_{LRdmd} = \vec{\phi}_p(b_p e^{\omega_p t}) \tag{24}$$

where *LRdmd* is low-rank DMD and the rest of the sum $X_{FGdmd} = \sum_{k\neq p}^{r} \vec{\phi}_k(e^{\omega_k t} b_k)$ is foreground video. Then

$$X_{dmd} = X_{LRdmd} + X_{FGdmd} \tag{25}$$

To avoid complex (phase) issues the authors of [Grosek, 2014] suggest obtaining the foreground as follows:

$$X_{FGdmd} = X_{dmd} - |X_{LRdmd}| \tag{26}$$

The latter however, may cause negative values. So we calculate a residual R which equals any negative values in LHS X_{FGdmd} of Eqn. 26 above.

The negative elements are added back as follows:

$$X_{LRdmd} = X_{LRdmd} + R \tag{27}$$
$$X_{FGdmd} = X_{FGdmd} - R \tag{28}$$

III. Algorithm Implementation & Development

The algorithm implementation consists of function scripts *dmd.py* and *videoFunctions.py* included in Appendix A. The most relevant functions will be explained next. The main processing script is *hw*5.*py* and is included in Appendix B.

The **DMD** function implements the DMD algorithm described in Section III B. The Python-numpy and sklearn fast random SVD both return S as a vector. Similarly, the eigenvalues in $eig()$ are returned as a vector. The *diag* command produces a diagonal matrix from a vector argument. The command $lstsq = np.linalg.lstsq$ is a least squares linear solver. In line 9, we use it to solve for b in $\Phi b = \vec{x}_0$

DMD(X)

1: $X_1, X_2 = X[0 : len(X) - 1], X[1 : len(X)]$
2: $X_1, X_2 = X_1^T, X_2^T$
3: $[U, S, Vt] = svd(X_1)$
4: $V = Vt^T$
5: $S_{inv} = diag(1/S)$
6: $A = U^T X_2 V S_{inv}$
7: $\vec{L}, W = eig(A)$
8: $\Phi = X_2 V S_{inv} W$
9: $b = lstsq(\Phi, X_1[:, 0])$
10: **return** $\Phi, A, S, \vec{L}, X_1, X_2, b$

The **dmdDynamics** function reconstructs the low-rank dynamics from the DMD modes, by Eqn. 16 in Section III B. In the below, $\otimes$ is the Hadamard (element-wise) product.

dmdDynamics(X1, L, b, Φ, nv, dt)

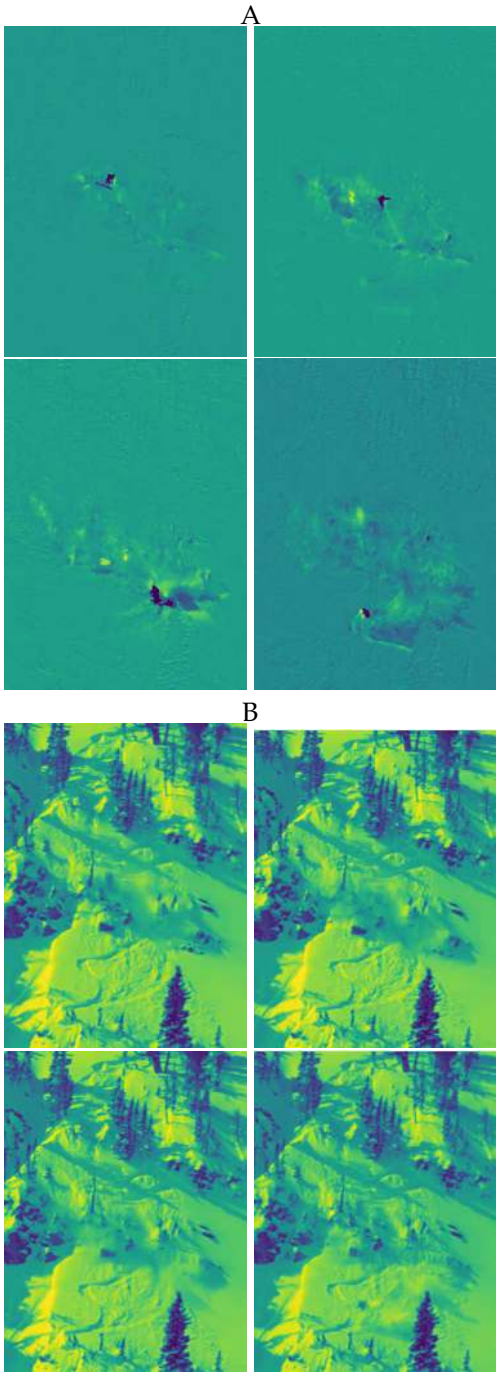
1: $\vec{\omega} = \log(\vec{L})/dt$
2: $T = zeros((nv, X1.shape[1])).astype(complex)$
3: $t = arange(X1.shape[1]) \cdot dt$
4: **for** $k = 0 : len(t)$ **do**:
5: $\quad T[:, k] = \vec{b} \otimes \exp(\vec{\omega} * t[k])$
6: **end for**
7: $Xdmd = \Phi T$
8: **return** $Xdmd, T, \vec{\omega}$

The main processing script *hw*5.*py* processes command line input, reads the files, then computes low rank dynamics by calling the functions *DMD* and *dmdDynamics* and subsequently uses the low rank approximation to subtract the video background as described in III C. From command line input get variables *filename*, start frame $f0$, end frame $f1$, threshold *thresh*, number of singular vectors to keep *nv*. Line 8 calculates the residual R described in Section C (see Eqn. 26).

Main script

1: $images = tnsrFile2numpy(filename)$
2: $X, szX, szY = flattenVideo(images, f0, f1)$
3: $[\Phi, A, S, L, X1, X2, b] = DMD(X, szY, szX, nv)$
4: $Xdmd, T, \vec{\omega} = dmdDynamics(X1, L, b, \Phi, nv, dt)$
5: $BG = abs(Xdmd.T)$
6: $FG = X[0 : len(X) - 1] - BG$
7: $R = copy(FG)$
8: $R[R > 0] = 0.0$
9: $FG = FG - R$
10: $BG = BG + R$
11: $FG = FG/np.max(FG)$
12: $FG[FG < 0.1] = 0.0$

Figure 2: *Selected Frames, Foreground & Background Movie* 2



(A) FG frames $5, 50, 100$ and 145 of the (cropped) movie *ski_drop.mov*. A uniform amount of grey color was added (to otherwise black background) to enhance contrast because the skier figure is dark. **(B)** Background (low rank with 8 modes) of frames $5, 50, 100$ and 145 of the movie *ski_drop.mov* corresponding to the original frames in panel A. Though it is hard to see in the static images, using several modes for the low rank background captures slower moving snow clouds and slight changes in camera angle.

IV. Computational Results

Two movies were provided, both short clips of dynamic objects(s) moving across a mostly static background. The first movie clip was a recording of a Monte Carlo racing event where the dynamic objects were race cars. The second video clip was a recording of a skier, skiing rapidly down a slope.

Both videos were first resized by $1/2$ in each dimension and converted to grey-scale. The ski video was also cropped to remove some area without dynamic movement.

Main script input/output & configuration. The main script is configurable by input commands. Input allows movie file name/path, rank nv and time step dt. Additional input includes start/end frame and threshold value. Thresholding is used to remove any remaining very weak background. The script outputs two movies: LR (the low rank approximation video) and $diff = FG$: the foreground or background subtracted video. Output also produces a selection of static images which are frames of either the LR or FG or original movie (frame numbers selected in input). The output of selected frames of the Monte Carlo LR and FG movie is shown in Figure 1. The output of LR and original of the ski video is shown in figure 2. The low rank LR output movies are included in Github.

Rank selection The number of modes is one of the input options to the main script. The rank selection value nv is the number of singular vectors kept in Eqn. 11. But we note that the rank kept in SVD also determines the number of modes in Eqn. 16 because it determines the size of $\tilde{A}$. For example, if $nv = 2$ is input, $\tilde{A}$ will be 2×2 and there will be two frequency modes as shown in example Eqn 18. So we are keeping all frequencies in this implementation but the number of frequency modes is indirectly selected by nv. We can get the single mode X_{LR-DMD} as in Eqn. 24 by selecting nv=1 as input to the main script.

In both sample video cases, better results were obtained by selecting more than a single low rank (LR background) mode. In the case of the Monte-Carlo video, results are shown for the first 51 frames and 2 LR modes were used. In the ski video 8 LR modes worked best. The first mode captures the static background. The second mode in the first (Monte-Carlo) video captures a small change in camera angle and some other slower movements. The 8 LR modes in the ski video capture a small change in camera zoom as well as slower moving snow clouds generated by the skier.

V. Summary and Conclusions

The least squares approximation becomes less accurate when the video is longer because A in Eqn.5 will be further overdetermined as the length of the image sequence increases. This problem can be overcome by breaking the movie into segments and updating A. The solution used here was to allow start and end frame input in the script.

Low frequency camera movement (like a gradual change in angle) is captured in LR with 2 or more modes. But high frequency movement isn't. This means higher frequency bumps and vibrations are not easily removed. In case of the Monte Carlo racing video, setting the low rank portion to two components captured a small gradual change in camera angle. In the ski video, up to 8 or so components captured a small change in camera zoom as well as slower moving snow dust clouds. Using 8 modes captures a shadow of the skier in areas where they slow down (e.g. when making turns).

Although the LR approximation nicely captured static and slower moving (background) objects and therefore allowed those to be subtracted, both the skier and race cars were darkly textured making them difficult to see on zero background. This was overcome by adding a small base-level (uniformly) to all pixels making the background grey.

References

[Tu, 2014] Tu JH,Rowley CW, Luchtenburg DM, Brunton SL, Kutz JN (2014). On dynamic mode decomposition: Theory and applications *Journal of Computational Dynamics* 1 (2) : 391-421.

[Grosek, 2014] Grosek, J. & Kutz, JN (2014). Dynamic Mode Decomposition for Real-Time Background/Foreground Separation in Video

A. Python functions used

The functions *DMD* and *dmdDynamics* are explained in detail in in Section III and are implemented here in the script *dmd.py*:

```
# AP Ruymgaart
# Python implementation DMD (Dynamic Mode Decomposition) algorithm
import numpy as np
import sklearn
import sklearn.utils
import sklearn.utils.extmath
mx,rsvd,svd = np.matmul, sklearn.utils.extmath.randomized_svd, np.linalg.svd
def sech(x) :  return 1/np.cosh(x)

#========== For this function, images should be in rows in X =========
# (P1) Paper Dynamic mode decomposition with control ;  Proctor, Brunton & Kutz
# this is algorithm 1 in Compressed Dynamic Mode Decomposition for Real-Time Object Detection
# Dynamics operator : x_{t+1} = Ax_{t}
# remember matrix multiplication is associative but not commutative
def DMD(X, szY, szX, nv=18, verbose=True):
        X1, X2 = X[0:len(X)-1], X[1:len(X)]
        X1, X2 = X1.T, X2.T                             # images x are now in columns

        if False:
                [U,S,Vt] = rsvd(X1, nv)                     # rsvd seems to not work
        else:
                [U,S,Vt] = svd(X1, full_matrices=False)     # MATLAB 'econ' mode
                V = Vt.conj().T
                Ur, Sr, Vr = U[:,0:nv], S[0:nv], V[:,0:nv] # Low rank approx

        Sinv = np.diag(1/Sr)
        VSinv = mx(Vr, Sinv)
        UtX2 = mx(Ur.T, X2)
        A = mx(UtX2, VSinv)                 # eqn. 12 P1 Atilde = Ur'*X2*Vr/Sr;
        L, W = np.linalg.eig(A)             # eigenvecs/vals of A
        Phi = mx( mx(X2, VSinv), W)         # eqn 13 in P1
        b = np.linalg.lstsq(Phi, X1[:,0])[0] # X1[:,0] is first image

        if verbose:
                print('DMD\n\t U from svd', U.shape, 'S', S.shape, 'V* from svd', Vt.shape)
                print('\t reduced shapes', Ur.shape, Sr.shape, Vr.shape, 'Nr PC selected', nv)
                print('\t shapes A', A.shape, 'W', W.shape, 'Phi', Phi.shape, 'b', b.shape)

        return [Phi,A,S,L,X1,X2,b]

# Build X for all time steps t
# Dynamic Mode Decomposition for Real-Time Background/Foreground Separation in Video
# Grosek & Kutz DMD reconstruct (eqn.5 in paper)
# x(t) = sum_j b_j phi_j e^{w_j t} where phi_j is a DMD component, b_j is the weight of component phi_j
def dmdDynamics(X1, L, b, Phi, nv=18, dt=0.33, verbose=True):
        omega = np.log(L)/dt
        T = np.zeros((nv, X1.shape[1])).astype(complex)
        t = np.arange(X1.shape[1]) * dt
        for k in range(len(t)):
                e0 = np.exp( omega * t[k] )
                T[:,k] = b * e0         # Hadamard product
        Xdmd = mx(Phi, T)          # outer : (size_im x nv)(nv x time) = (size_im x time)

        if verbose:
                print('dmdDynamics\n\t T size', T.shape, 'nr frames', len(t), len(b))
                print('\t======== omega =========')
                print('\tomega', omega)
                print('\t|omega|', np.abs(omega))

        return Xdmd, T, omega

# build single time x(t) from low rank sum
# does : x(t) = Phi[:,0]*b[0]*exp(om*t) + Phi[:,1]*b[1]*exp(om*t) + ....
def dmdDynamicsLrVec(X1, L, b, Phi, nt, dt, omegaCut=0.0, verbose=True):
        omega = np.log(L)/dt
        t = np.arange(X1.shape[1]) * dt
        LRt = np.zeros((Phi.shape[0])).astype(complex)
        HRt = np.zeros((Phi.shape[0])).astype(complex)
        for k in range(Phi.shape[1]):
                print('\tmode', k, '|omega|', np.abs(omega[k]), b[k])
                R1 = Phi[:,k] * b[k] * np.exp(omega[k]*t[nt])
                if np.abs(omega[k]) > omegaCut:  HRt += R1
```

```
                else:                           LRt += R1
        return LRt, HRt

if __name__ == "__main__": # DMD test examlple, same as MATLAB dmd.m
        xi = np.linspace(-10,10,400)
        t = np.linspace(0,4*np.pi,200)
        dt = t[1] - t[0]
        [Xgrid,T] = np.meshgrid(xi,t)
        f1 = sech(Xgrid+3)*(1.0*np.exp(1j*2.3*T))
        f2 = (sech(Xgrid)*np.tanh(Xgrid))*(2*np.exp(1j*2.8*T))
        f = f1 + f2
        r = 2
        szY, szX, nv = 1, 400, r
        [Phi,A,S,D,X1,X2,b] = DMD(f, szY, szX, r)
        Z, T, omega = dmdDynamics(X1, D, b, Phi, r, dt)
        print('A', A)
        print('b', b)
        print(Z[0:3,0:3])
```

The script videoFunctions.py provides some basic video processing functions that are deemed self-explanatory:

```
import imageio, cv2, numpy as np
from skimage.color import rgb2grey
from skimage.transform import resize
from tensorFiles import *

def video2numpy(fname):
        imgs,img = [], 1
        cap = cv2.VideoCapture(fname)
        while not img is None:
                ret, img = cap.read()
                if not img is None:        imgs.append(img)
        return np.array(imgs)

def flattenVideo(npVid, strt=0, stp=None):
        szX, szY, X = npVid[0].shape[1], npVid[0].shape[0], []
        for im in npVid: X.append(im.reshape(-1))
        X = np.array(X)
        f1, f2 = strt, len(X)
        if not stp is None: f2 = stp
        return X[f1:f2], szX, szY

def reshape2video(flatImgs, szY, szX):
        npMv = []
        for im in flatImgs:
                npMv.append(np.abs(im.reshape(szY, szX)))
        return np.array(npMv)

def halfGreyscale(imgs):
        ret = []
        szy, szx = int(imgs[0].shape[0]/2), int(imgs[0].shape[1]/2)
        for img in imgs: ret.append(resize(rgb2grey(img), (szy, szx)))
        return np.array(ret)

def np2movieFile(imgTensor, fname, fps=30, invert=True):
        if invert: imgTensor = 1.0 - imgTensor
        imgTensor = (imgTensor *255.0).astype('uint8')
        imageio.mimwrite(fname+'.mp4', imgTensor , fps = fps)

def processCmdArgs(arglst):
        cmd = {}
        for c in arglst:
                try:
                        elms = c.split('=')
                        cmd[elms[0]] = elms[1]
                except: pass
        return cmd

if __name__ == "__main__":
        mcVideo = video2numpy("data/monte_carlo.mov")
        mcVideo = halfGreyscale(mcVideo)
        numpy2tnsrFile(mcVideo, "data/monte_carlo.npz", dataType='float32')
        skiVideo = video2numpy("data/ski_drop.mov")
        skiVideo = halfGreyscale(skiVideo)
        numpy2tnsrFile(skiVideo, "data/ski_drop.npz", dataType='float32')
```

B. Python codes

```
# AP Ruymgaart DMD, main script
import numpy as np, time, sys, copy, matplotlib.pyplot as plt
from videoFunctions import *
from tensorFiles import *
from plottingFunctions import *
from dmd import *

#==== input (command line, from run.sh) ====
print('================================== start DMD ================================\nInput:')
cmds = processCmdArgs(sys.argv)
for c in cmds: print('\t', c, cmds[c])
dt,nv,fname,ftype,images,f0,f1,outname,frPlot,binv = None,None,None,None,None,None,None,None,[],None
try:
        nv = int(cmds['modes'])
        dt,thresh = float(cmds['dt']), float(cmds['thresh'])
        fname,ftype,outname = cmds['movie'], cmds['type'], cmds['outname']
        f0,f1 = int(cmds['framestart']), int(cmds['framestop'])
        szplotfr = cmds['plotframes'].split(',')
        binv = cmds['inv'].lower() == 'true'
        for h in szplotfr: frPlot.append(int(h))
except: print('** input error **'), exit()

if ftype == 'npz': images = tnsrFile2numpy(fname)
else: images = video2numpy(fname)
print('Movie-shape=', images.shape, 'dt=',dt, 'Nr modes=', nv, 'file-type=', ftype, 'frame', f0, 'to', f1)
print('Plot frames', frPlot, 'output file name', outname)

#==== DMD & dmdDynamics ====
X, szX, szY = flattenVideo(images, f0, f1)
[Phi,A,S,L,X1,X2,b] = DMD(X, szY, szX, nv)
Xdmd, T, omega = dmdDynamics(X1,L,b,Phi,nv,dt=dt)

#==== foreground/background ====
BG = abs(copy.copy(Xdmd.T))
FG = X[0:len(X)-1] - BG + 0.3 #- subtract low rank BG and add a grey background

print(np.min(FG), np.max(FG))
if False:
        R = copy.copy(FG)
        R[R > 0] = 0.0
        FG = FG - R
        BG = BG + R

for n in range(len(FG)): FG[n] = FG[n]/np.max(FG[n])
FG[FG < thresh] = 0.0 # thresholding (see paper)

if False: #- alternative attempt to select modes, not used now
        omegaCut = 0.0
        Xlr,Xhr = np.zeros(Xdmd.shape), np.zeros(Xdmd.shape)
        for k in range(T.shape[1]):
                LRt, HRt = dmdDynamicsLrVec(X1, L, b, Phi, k, dt, omegaCut=omegaCut)
                Xlr[:,k] = LRt
                Xhr[:,k] = HRt

        L2 = np.abs(Xlr.T)
        H2 = np.abs(Xhr.T)
        lrMv = reshape2video(L2/np.max(L2), szY, szX)
        np2movieFile(lrMv, outname+'_xlr', invert=binv)
        hrMv = reshape2video(H2/np.max(H2), szY, szX)
        np2movieFile(hrMv, outname+'_xhr', invert=binv)

#==== output ====
plotSV(np.log(np.abs(S) + 1), fname=outname+'_logSV.png')
plotSV(np.abs(omega), fname=outname+'_omega.png')
bgMv = reshape2video(BG, szY, szX)
bgMv = bgMv/np.max(bgMv)
np2movieFile(bgMv, outname+'_LR', invert=binv)
fgMv = reshape2video(FG, szY, szX)
np2movieFile(fgMv, outname+'_diff', invert=binv)
origMv = reshape2video(X[0:len(X)-1], szY, szX)
for fr in frPlot:
        plotFrame(bgMv, fr, outname+'_LR_%d' % (fr))
        plotFrame(fgMv, fr, outname+'_diff_%d' % (fr))
        plotFrame(origMv, fr, outname+'_orig_%d' % (fr))
```