DMD and Video Background Subtraction

Luotong Kang

Abstract

In this report, we'll try to separate videos into background video and foreground video. We define the moving objects in video clips as foreground. Using Dynamic Mode Decomposition (DMD). With the help of SVD, we could implement separation and reconstruction on **ski_drop.mov** and **monte_carlo.mov**.

1 Introduction

Even with high-dimensional nonlinear dynamical systems, we can find a basis of spatial modes for which the time dynamics are just exponential functions. Thus, we can deal with spatial-temporal data sets as linear mapping using data-based technique DMD.

We are going to analyze two video clips. **ski_drop.mov** contains a skier foreground, and **monte_carlo.mov** shows several racing cars' moving. We'll see how DMD spectrum of frequencies can be used to subtract background modes.

2 Theoretical Background

2.1 Setup

We'll assume that the readers have some knowledge about SVD. We assume that we have snapshots of spatio-temporal data. We define N as the number of spatial points saves per unit time snapshot and M as the number of snapshots taken. With a regularly spaced intervals $t_{m+1} = t_m + \Delta t, m = 1, \ldots, M-1, \Delta t = 0$, we denote the snapshots as

$$U(\mathbf{x}, t_m) = \begin{bmatrix} U(x_1, t_m) \\ U(x_2, t_m) \\ \vdots \\ U(x_n, t_m) \end{bmatrix}$$

Thus, the setup of the data matrix would be

$$\mathbf{X} = \begin{bmatrix} U(\mathbf{x}, t_1) & U(\mathbf{x}, t_2) & \ldots & U(\mathbf{x}, t_M) \end{bmatrix}$$

We denote columns j through k of $\mathbf{X}$ as

$$\mathbf{X}_j^k = \begin{bmatrix} U(\mathbf{x}, t_j) & U(\mathbf{x}, t_{j+1}) & \ldots & U(\mathbf{x}, t_j) \end{bmatrix}$$

2.2 DMD Procedure

The DMD approximates modes of the **Koopman operator**, a linear, time-independent operator such that $\mathbf{x}_{j+1} = \mathbf{A}\mathbf{x}_j$, where the j indicates the specific data collection time and $\mathbf{A}$ is the linear operator and $\mathbf{x_j}$ is an N-dimensional data points at time j. This global operator shows mechanism of DMD.

Using the Koopman operator, we can form the basis for Krylov subspace and map:

$$\mathbf{X}_1^{M-1} = [\mathbf{x}_1 \quad \mathbf{A}\mathbf{x}_1 \quad \mathbf{A}^2\mathbf{x}_1 \quad \dots \quad \mathbf{A}^{M-2}\mathbf{x}_1]$$
$$\mathbf{X}_2^M = \mathbf{A}\mathbf{X}_1^{M-1} + \mathbf{r}e_{M-1}^T$$

where e_{M-1} is the vector with all zeros except a 1 at the $(M-1)$st component, and $\mathbf{r}$ is the residual vector to account final point $\mathbf{x}_M$ which is not in Krylov basis.

Using SVD of $\mathbf{X}_1^{M-1}$ and the fact that $\mathbf{U}^*\mathbf{r} = 0$, we do operations to get:

$$\mathbf{U}^*\mathbf{A}\mathbf{U} = \mathbf{U}^*\mathbf{X}_2^M\mathbf{V}\boldsymbol{\Sigma}^{-1} = \tilde{\mathbf{S}}$$

Since $\tilde{\mathbf{S}}$ and $\mathbf{A}$ are related by applying a matrix on one side and its inver on the other, they are similar, which implies that they have same eigenvalues. So we would want to do eigenvalue expansion. We can obtain the DMD modes

$$\psi_k = \mathbf{U}\mathbf{y}_k.$$

Expanding in eigenbasis, we get DMD solutions

$$\mathbf{x}_{DMD}(t) = \sum_{k=1}^{K} b_k\psi_k e^{\omega_k t} = \Psi \text{diag}(e^{\omega_k t})\mathbf{b},$$

where $K = \text{rank}|\mathbf{X}_1^{M-1}|$, b_k is the initial amplitude of each mode, $\omega_k = \frac{\ln(\mu_k)}{\Delta t}$ ($\mu_k =$ eigenvalues of $\tilde{\mathbf{S}}$), and Ψ is formed by the ψ_k columns. We solve $\mathbf{b} = \Psi^\dagger\mathbf{x}_1$ using $t = 0$.

2.3 Background Subtraction

Assume that ω_p, where $p \in \{1, 2, \dots, \ell\}$, satisfies $\|\omega_p\| \approx 0$, and that $\|\omega_j\| \,\forall j \neq p$ is bounded away from zero. We obtain the following:

$$\mathbf{X}_{\text{DMD}} = \underbrace{b_p\varphi_p e^{\omega_p \mathbf{t}}}_{\text{Background Video}} + \underbrace{\sum_{j\neq p} b_j\varphi_j e^{\omega_j \mathbf{t}}}_{\text{Foreground Video}},$$

which will produce a $\mathbf{X}_{DMD}$ matrix with terms being complex. Calculate the DMD's approximate low-rank reconstruction according to $\mathbf{X}_{\text{DMD}}^{\text{Low-Rank}} = b_p\varphi_p e^{\omega_p \mathbf{t}}$. We can solve

$$\mathbf{X}_{\text{DMD}}^{\text{Sparse}} = \sum_{i\neq n} b_j\varphi_j e^{\omega_j \mathbf{t}} \text{ via } \mathbf{X}_{\text{DMD}}^{\text{Sparse}} = \mathbf{X} - \left|\mathbf{X}_{\text{DMD}}^{\text{Low-Rank}}\right|.$$

Since we may obtain senseless negative values in sparse matrix, we put these negative values into a residual $n \times m$ matrix R make adjustments:

$$\mathbf{X}_{\text{DMD}}^{\text{Low-Rank}} \leftarrow \mathbf{R} + \left|\mathbf{X}_{\text{DMD}}^{\text{Low-Rank}}\right|$$
$$\mathbf{X}_{\text{DMD}}^{\text{Sparse}} \leftarrow \mathbf{X}_{\text{DMD}}^{\text{Sparse}} - \mathbf{R}$$

Finally, we can check the constraint by showing the image $\mathbf{X}_{\text{DMD}}^{\text{Low-Rank}} + \mathbf{X}_{\text{DMD}}^{\text{Sparse}}$

3 Algorithm Implementation and Development

Lines 1-15: We first load the video data and put all frames (each frame is a image, also known as snapshots of DMD) in grayscale. In order to have a better processing speed, we can compress the images using `imsize(vidFrames(:,:,:,j),1/comfact)` in the loop, where `comfact` represents the factor of scaling down resolution. Then, using the setup format described in Section 2.1, we reshape the data matrix before DMD.

Lines 16-48: By looking at energies captured by the diagonal variances from SVD basis, we can truncate our matrices to low rank versions. Using formula in Section 2.2, we can easily compute $\tilde{\mathbf{S}}$ using `U'*X2*V*diag(1./diag(Sigma))`, where `Sigma,U,V` is the truncated version of SVD results. We can check by looking at omega values after DMD to see how much of the background we were capturing with additional modes.

Lines 49-55: Using pseudoinverse to get initial conditions, we can calculate DMD modes in a loop and then obtain DMD solutions `Xlowrank`.

Lines 56-66: Using the theories in Section 2.3, we calculate the low rank matrix and add back R to get background video matrix. Also, we can try to generate foreground video matrix by subtracting R from matrix `fdata-abs(Xlowrank)`. Next, we reshape the matrix into spatial-temporal setup in order to make the plotting easier.

Lines 67-83: Choose a frame to plot. We can make comparison by plotting original video, background/foreground videos for one frame. We can check the constraint by plotting a reconstructed version (background matrix + foreground matrix), which should give us a looking that's similar to original video.

Lines 85-167: Repeat the above algorithm for second video **monte_carlo.mov**. The only difference might be the resolution setup, which is determined by pixel numbers.

4 Computational Results

4.1 ski_drop.mov

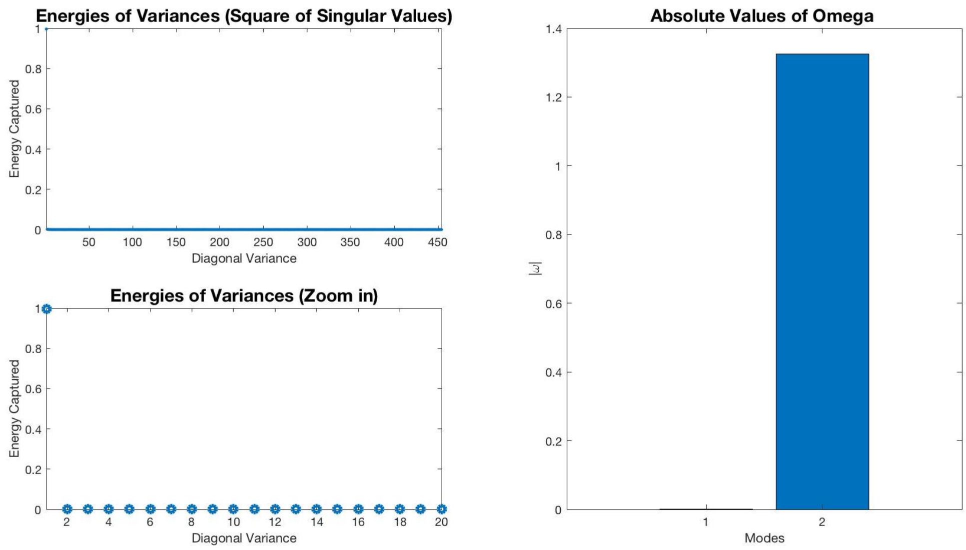

Figure 1: **ski_drop.mov**: Energy of variances (left).
Absolute values of omega of each selected modes (right).

Since the first diagonal variance captures the most energy (Figure 1), one mode is efficient for low rank approximation. For safety, we use one more mode and solve low-rank dynamics using DMD. We also see that the absolute value of our first omega value is extremely close to 0, while the next one is much higher. The omega close to 0 indicates that it is a background mode. Thus, two modes is again being verified as sufficient.

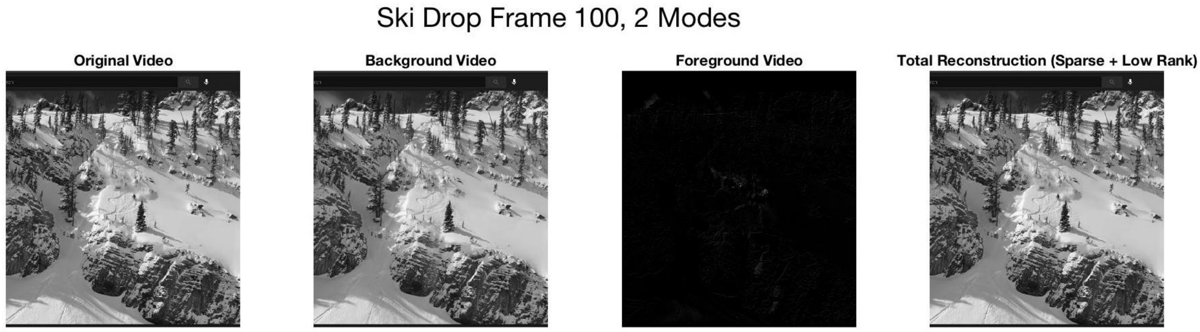

Figure 2: Comparison of original **ski_drop.mov** video and background/foreground video at frame 100. Check for reconstruction in the right.

From Figure 2 we can see that we've successfully separate the background from the original video with the skier disappear in the second graph. However, the foreground video is not that clear. Although the skier is dark and distinct, it only occupies a little portion of the snapshot. Also, the speed of skier is high and snow is falling along with the motion. Thus there a white slash in foreground video which is larger than the skier figure size. From the reconstruction in the right, we know that the background and foreground match to the same video.

4.2 monte_carlo.mov

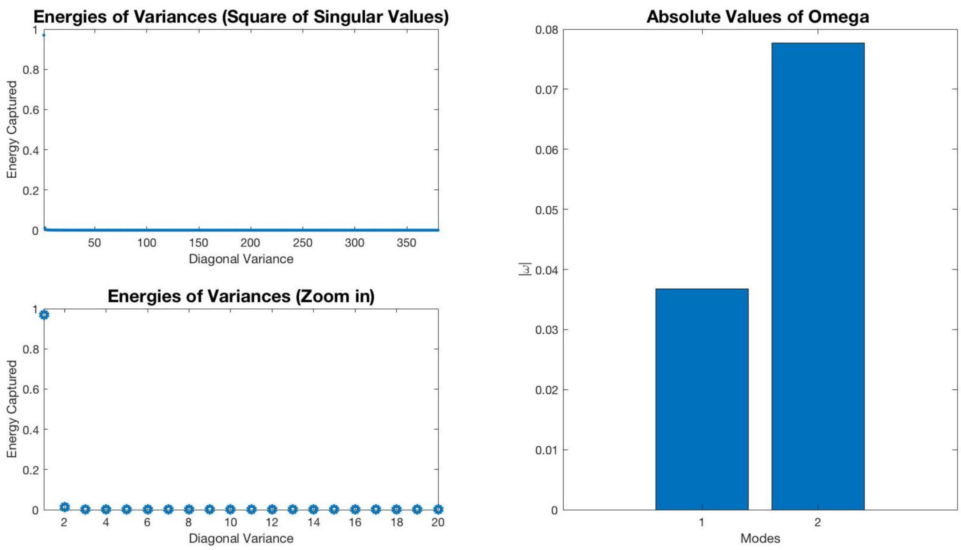

Figure 3: **monte_carlo.mov**: Energy of variances (left).
Absolute values of omega of each selected modes (right).

Since the first diagonal variance captures the most energy (Figure 3), one mode is efficient for low rank approximation. For safety, we use one more mode and solve low-rank dynamics using DMD. We also see that the absolute value of first omega value is much lower than the second one, showing that the first mode captures the background but we may still need the second mode.

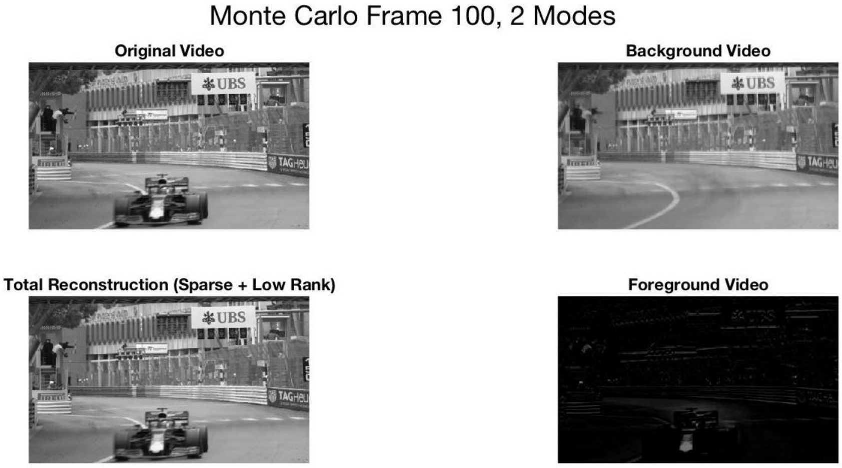

Figure 4: Comparison of original **monte_carlo.mov** video and background/foreground video at frame 100. Check for reconstruction in the right.

From Figure 2 we can see that we've successfully separate the background from the original video with the cars disappear in the second graph. However, the foreground video is again not that clear. The distinctive characteristic of the the car (white front) is obvious, but sharp boundaries like advertising board still add noises on foreground video. From the reconstruction in the right, we know that the background and foreground match to the same video.

5 Summary and Conclusions

Since the boundary of foreground is obvious and the camera is not shaking in both two video clips, we've done a great job generating the background videos. However, noises like fast motion and sharp boundaries in background make foreground videos looks blurry, especially when seeing the foreground as "black base" image. We cannot deny though the impressive power of DMD spectrum of frequencies. Without even knowing the governing equations, DMD takes advantages of low dimensionality in the data to create low-rank approximation of the linear mapping (Koopman operator) that best iterates through the snapshots of data.

6 Appendices

6.1 MATLAB functions used

NOT listed chronologically

`VideoReader()`: read files containing video data. The object contains information about the video file and enables you to read data from the video. Property `.Framerate` returns (average) number of video frames per second, specified as a numeric scalar.
`zeros(size),ones(size)`: creates matrices with all entries being zero/one with indicated size.
`plot()`: creates a 2-D line plot of the data in Y versus the corresponding values in X.
`bar(y)`: creates a bar graph with one bar for each element in y.
`set(gca,), set(gcf,), title(), sgtitle(), xlabel(), ylabel(), yticks(), yticklabels(), subplot(), hold on`: plotting commands.
`size(A)`: returns a row vector whose elements are the lengths of the corresponding dimensions of A.
`length(A)`: returns the length of the largest array dimension in X.
`sum(V)`: V is a vector, returns the sum of the elements.
`diag(A)`: returns a column vector of the main diagonal elements of A.
`[U,S,V] = svd(A)`: returns numeric unitary matrices U and V with the columns containing the singular vectors, and a diagonal matrix S containing the singular values. The matrices satisfy the condition A = U*S*V'.
`[V,D] = eig(A)`: The columns of V present eigenvectors of A. The diagonal matrix D contains eigenvalues.
`log(X)`: returns the natural logarithm of X.
`mldivide,\`: solves the system of linear equations.
`abs(X)`: returns the absolute value of each element in array X. If X is complex, it returns the complex magnitude.
`imshow(I)`: displays the grayscale image I in a figure.
`imresize(A,[numrows numcols])`: returns image that has the number of rows and columns specified by the two-element vector.
`rgb2gray()`: converts the truecolor image RGB to the grayscale image I.
`double()`: converts the values to double precision.
`reshape(A,sz)`: reshapes A using the size vector sz.
`strcat(s1,...,sN)`: horizontally concatenates s1,...,sN.

6.2 MATLAB codes

```
%% Vid 1
clear; close all; clc;
vid = VideoReader('ski_drop.mov');
dt = 1/vid.Framerate;
vidFrames = read(vid);
numFrames = size(vidFrames(1,1,1,:),4);
t=linspace(0,vid.Duration,numFrames);
[nx,ny]=size(vidFrames(:,:,:,1),[1,2]);
```

```
%% Grascale, Compress and Reshape
comfact = 2;
fdata = zeros([nx*ny/(comfact^2),numFrames]);
for j=1:numFrames
    X=double(rgb2gray(imresize(vidFrames(:,:,:,j),1/comfact)));
    fdata(:,j) = reshape(X,[nx*ny/(comfact^2),1]);
end
%% DMD
% Create DMD matrices
X1 = fdata(:,1:end-1);
X2 = fdata(:,2:end);
% SVD of X1 and Computation of ¬S
[U_untr, Sigma_untr, V_untr] = svd(X1,'econ');
lam= diag(Sigma_untr).^2;
figure(1)
subplot(221)
plot(lam/sum(lam), '.');
set(gca,'Xlim',[1,length(lam)])
title("Energies of Variances (Square of Singular Values)", ...
    'Fontsize', 15);
ylabel("Energy Captured"); xlabel("Diagonal Variance");
subplot(223)
plot(lam/sum(lam), 'o','Linewidth',3);
set(gca,'Xlim',[1,20])
title("Energies of Variances (Zoom in)",'Fontsize', 15);
ylabel("Energy Captured"); xlabel("Diagonal Variance");
% Truncate to rank-r
r = 2;
U = U_untr(:, 1:r);
Sigma = Sigma_untr(1:r, 1:r);
V = V_untr(:, 1:r);
S = U'*X2*V*diag(1./diag(Sigma)); % low-rank dynamics
[eV, D] = eig(S); % compute eigenvalues + eigenvectors
mu = diag(D); % extract discrete-time eigenvalues
omega = log(mu)/dt; % continuous-time eigenvalues
Phi = U*eV;
subplot(2,2,[2,4])
bar(abs(omega))
title("Absolute Values of Omega ",'Fontsize', 15);
xlabel("Modes"); ylabel("|\omega|");
set(gcf,'Position',[0,0,1000,500])
%% DMD Reconstruction
y0 = Phi\X1(:,1); % pseudoinverse to get initial conditions
u_modes = zeros(r,length(t));
for iter = 1:length(t)
   u_modes(:,iter) = y0.*exp(omega*t(iter));
end
Xlowrank = Phi*u_modes;
%% Create Sparse and Nonsparse
Xsparse = fdata - abs(Xlowrank);
R = 2.*(Xsparse<0);
X_back = R + abs(Xlowrank);
X_fore = Xsparse - R;
X_reconstructed =  X_fore + X_back;
%% Reshape plotting data
```

```
Orig =  reshape(fdata, [nx/comfact,ny/comfact,length(t)]);
Lowrank_addR = reshape(X_back, [nx/comfact,ny/comfact,length(t)]);
Sparse_subR =  reshape(X_fore, [nx/comfact,ny/comfact,length(t)]);
Reconstructed =  reshape(X_reconstructed, ...
    [nx/comfact,ny/comfact,length(t)]);
%% Plot
framenum = 100;
figure(2)
subplot(141)
imshow(uint8(Orig(:,:,framenum)))
title("Original Video");
subplot(142)
imshow(uint8(Lowrank_addR(:,:,framenum)))
title("Background Video");
subplot(144)
imshow(uint8(Reconstructed(:,:,framenum)))
title("Total Reconstruction (Sparse + Low Rank)");
subplot(143)
imshow(uint8(Sparse_subR(:,:,framenum)))
title('Foreground Video');
sgtitle(strcat("Ski Drop Frame ", int2str(framenum), ", ", ...
    int2str(r)," Modes"), 'Fontsize', 20)
set(gcf,'Position',[0,0,1100,300])

%% Vid 2
clear; close all; clc;
vid = VideoReader('monte_carlo.mov');
dt = 1/vid.Framerate;
vidFrames = read(vid);
numFrames = size(vidFrames(1,1,1,:),4);
t=linspace(0,vid.Duration,numFrames);
[nx,ny]=size(vidFrames(:,:,:,1),[1,2]);
%% Grascale, Compress and Reshape
comfact = 4;
fdata = zeros([nx*ny/(comfact^2),numFrames]);
for j=1:numFrames
    X=double(rgb2gray(imresize(vidFrames(:,:,:,j), 1/comfact)));
    fdata(:,j) = reshape(X,[nx*ny/(comfact^2),1]);
end
%% DMD
% Create DMD matrices
X1 = fdata(:,1:end-1);
X2 = fdata(:,2:end);
% SVD of X1 and Computation of ¬S
[U_untr, Sigma_untr, V_untr] = svd(X1,'econ');
lam= diag(Sigma_untr).^2;
figure(3)
subplot(221)
plot(lam/sum(lam), '.');
set(gca,'Xlim',[1,length(lam)])
title("Energies of Variances (Square of Singular Values)", ...
    'Fontsize', 15);
ylabel("Energy Captured"); xlabel("Diagonal Variance");
subplot(223)
plot(lam/sum(lam), 'o','Linewidth',3);
```

```
set(gca,'Xlim',[1,20])
title("Energies of Variances (Zoom in)",'Fontsize', 15);
ylabel("Energy Captured"); xlabel("Diagonal Variance");
% Truncate to rank-r
r = 2;
U = U_untr(:, 1:r);
Sigma = Sigma_untr(1:r, 1:r);
V = V_untr(:, 1:r);
S = U'*X2*V*diag(1./diag(Sigma)); % low-rank dynamics
[eV, D] = eig(S); % compute eigenvalues + eigenvectors
mu = diag(D); % extract discrete-time eigenvalues
omega = log(mu)/dt; % continuous-time eigenvalues
Phi = U*eV;
subplot(2,2,[2,4])
bar(abs(omega))
title("Absolute Values of Omega ",'Fontsize', 15);
xlabel("Modes"); ylabel("|\omega|");
set(gcf,'Position',[0,0,1000,500])
%% DMD Reconstruction
y0 = Phi\X1(:,1); % pseudoinverse to get initial conditions
u_modes = zeros(r,length(t));
for iter = 1:length(t)
   u_modes(:,iter) = y0.*exp(omega*t(iter));
end
Xlowrank = Phi*u_modes;
%% Create Sparse and Nonsparse
Xsparse = fdata - abs(Xlowrank);
R = 2.*(Xsparse<0);
X_back = R + abs(Xlowrank);
X_fore = Xsparse - R;
X_reconstructed =  X_fore + X_back;
%% Reshape plotting data
Orig =  reshape(fdata, [nx/comfact,ny/comfact,length(t)]);
Lowrank_addR = reshape(X_back, [nx/comfact,ny/comfact,length(t)]);
Sparse_subR =  reshape(X_fore, [nx/comfact,ny/comfact,length(t)]);
Reconstructed =  reshape(X_reconstructed, ...
    [nx/comfact,ny/comfact,length(t)]);
%% Plot
framenum = 100;
figure(4)
subplot(221)
imshow(uint8(Orig(:,:,framenum)))
title("Original Video");
subplot(222)
imshow(uint8(Lowrank_addR(:,:,framenum)))
title("Background Video");
subplot(223)
imshow(uint8(Reconstructed(:,:,framenum)))
title("Total Reconstruction (Sparse + Low Rank)");
subplot(224)
imshow(uint8(Sparse_subR(:,:,framenum)))
title('Foreground Video');
sgtitle(strcat("Monte Carlo Frame ", int2str(framenum), ", ", ...
    int2str(r)," Modes"), 'Fontsize', 18)
set(gcf,'Position',[0,0,800,350])
```