Introduction to Matlab

Problem 1)

Code File: **p3.m**

% Parse the data

[x, fs] = audio2bin('music.mp3');

sound(x, fs);

% Read formatted data

[x, fs] = bin2audio('music.bin');

sound(x, fs);

% Time range of interest

time = (0:(length(x) - 1)) / fs;

t1 = 1;

t2 = 1.01;

% Plot the time domain

subplot(2, 2, 1);

plot(time, x);

xlabel('time [seconds]', 'FontSize', 18);

ylabel('amplitude', 'FontSize', 18);

set(gca, 'FontSize', 16);

grid on;

xlim([t1 t2]);

% Plot the FFT

i1 = round(t1 \* fs);

i2 = round(t2 \* fs);

nfft = 2^12; % FFT size

freq = (((0:(nfft-1)) / nfft) - 0.5) \* fs; % frequency [Hz]

X = fft(x(i1:i2,:), nfft); % Discrete Fourier Transform

subplot(2, 2, 2);

plot(freq, 20\*log10(abs(fftshift(X)))); % use log axis

xlabel('frequency [Hz]', 'FontSize', 18);

ylabel('magnitude [dB]', 'FontSize', 18);

set(gca, 'FontSize', 16);

grid on;

% Plot the spectrograms

nfft = 2^8;

overlap = round(0.8\*nfft);

window = hamming(nfft);

subplot(2, 2, 3);

spectrogram(x(:,1), window, overlap, nfft, fs);

set(gca, 'FontSize', 16);

grid on;

subplot(2, 2, 4);

spectrogram(x(:,2), window, overlap, nfft, fs);

set(gca, 'FontSize', 16);

grid on;

% View on oscilloscope

win\_sec = 0.05; % window length [seconds]

win\_sam = round(win\_sec\*fs); % window length [samples]

step\_sec = 0.001; % step length [seconds]

step\_sam = round(step\_sec\*fs); % step length [samples]

figure;

han = plot(time(1:win\_sam), x(1:win\_sam));

drawnow;

ylim(0.1\*[-1, 1]);

for i = win\_sam:step\_sam:length(x)

ind = ((i - win\_sam + 1):i);

set(han, 'XData', time(ind), 'YData', x(ind));

xlim(time(ind([1, end])));

drawnow;

pause(0.05);

end

Code File: **audio2bin.m**

function [x, Fs] = audio2bin(fin, fout)

% Construct output file name from input

if (nargin < 2)

fout = [fin(1:max(strfind(fin, '.'))), 'bin'];

end

% Path correction

fin = [pwd,'\','Resources\',fin];

fout = [pwd,'\','Output\',fout];

fprintf('Input file = %s\n', fin);

fprintf('Output file = %s\n', fout);

% Read the audio file

[x, Fs] = audioread(fin);

fid = fopen(fout, 'wb');

% Determine data dimensions

channels = size(x, 2); % columns in x (size of second dimension)

samples = size(x, 1); % rows in x

% Write the header

% ndim = 1 (audio)

% nchan = channels

% dim0 = samples

% dim1 = Fs (for audio files)

% dim2 = 0 (not used for audio)

fwrite(fid, [1, channels, samples, Fs, 0], 'int');

% Arrange channels by row so the (:) will coalesce correctly

%transpose = x.';

% Output the data

%fwrite(fid, transpose(:), 'float');

% Alternatively, use a simple for loop (so what, it's not efficient?)

for i = 1:samples

for j = 1:channels

fwrite(fid, x(i, j), 'float');

end

end

% Release the file handler

fclose(fid);

end

Code File: **bin2audio.m**

function [x, Fs] = bin2audio(fin, fout)

% Construct output file name from input

if (nargin < 2)

fout = [fin(1:max(strfind(fin, '.'))), 'wav'];

end

% Path correction

fin = [pwd,'\','Output\',fin];

fout = [pwd,'\','Output\',fout];

fprintf('Input file = %s\n', fin);

fprintf('Output file = %s\n', fout);

% Read the header

% ndim = 1 (audio)

% nchan = channels

% dim0 = samples

% dim1 = Fs (for audio files)

% dim2 = 0 (not used for audio)

fid = fopen(fin, 'rb');

ndim = fread(fid, 1, 'int');

channels = fread(fid, 1, 'int');

samples = fread(fid, 1, 'int');

Fs = fread(fid, 1, 'int');

dim2 = fread(fid, 1, 'int');

% Read the data

[a, ~] = fread(fid, inf, 'float');

% Release the file handler

fclose(fid);

% Just use nested loops this time

x = zeros(samples, channels);

for i = 1:samples

for j = 1:channels

% i-1 because matlab is stupid with 1-based indexing

x(i, j) = a((i - 1)\*channels + j);

end

end

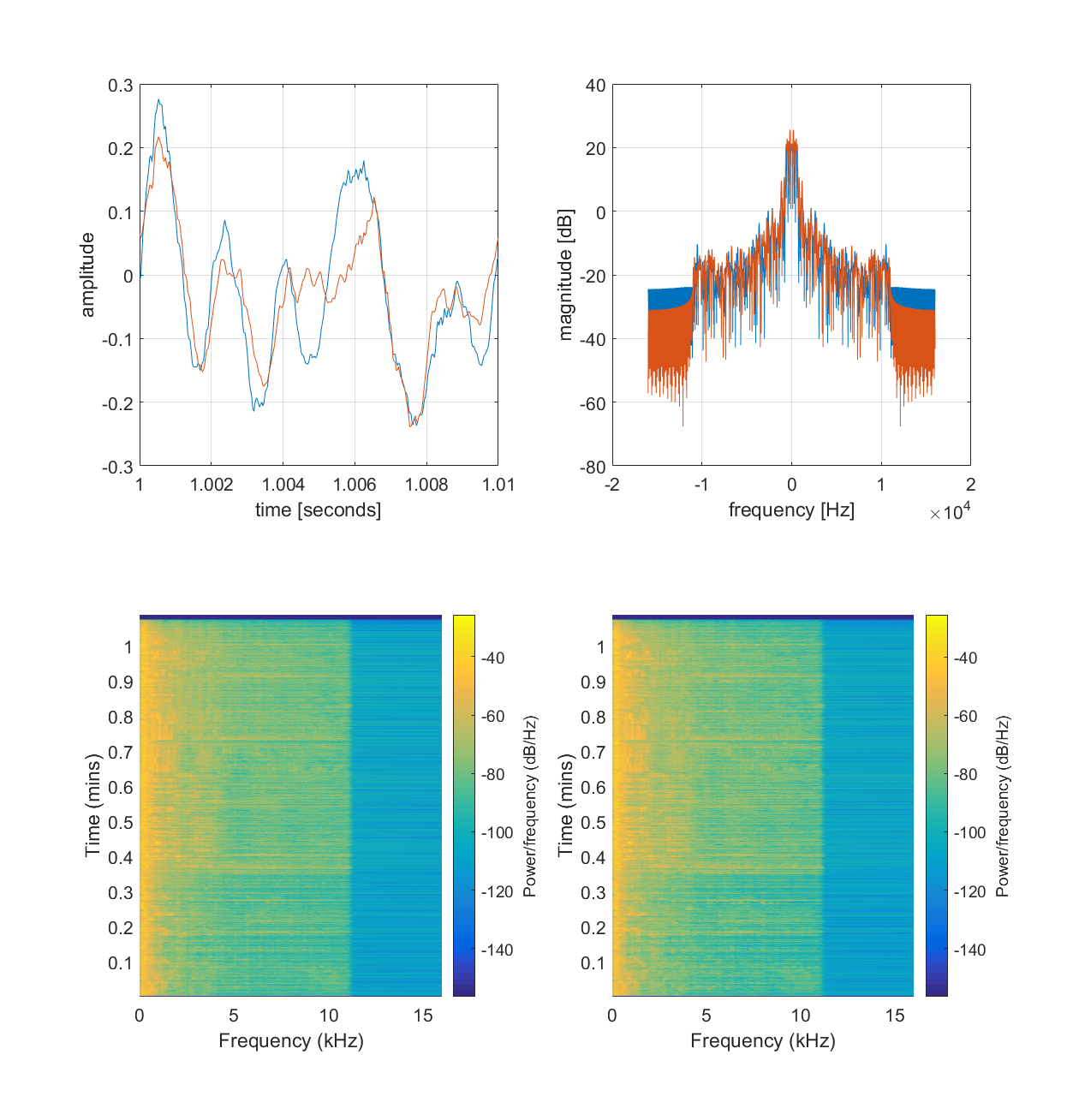
% Write out the data

audiowrite(fout, x, Fs);

end

In hindsight, I probably could have just done: x(i, j) = fread(fid, 1, ‘float’);

oh well….



Basically, Matlab was able to parse an audio file into a series of sampled values that I can later perform computation on in C.

Problem 2)

Code File: **p2.m**

% Parse the data

[x1] = image2bin('liftingbody.png');

[x2] = image2bin('coloredchips.png');

% Read formatted data

[x1] = bin2image('liftingbody.bin');

[x2] = bin2image('coloredchips.bin');

figure;

% Show gray-scale

subplot(1,2,1);

imagesc(x1, [100, 200]);

axis image;

colormap(gray);

colorbar;

% Show color

subplot(1,2,2);

image(x2);

axis image;

Code File: **image2bin.m**

function [x] = image2bin(fin, fout)

% Construct output file name from input

if (nargin < 2)

fout = [fin(1:max(strfind(fin, '.'))), 'bin'];

end

% Path correction

fin = [pwd,'\','Resources\',fin];

fout = [pwd,'\','Output\',fout];

fprintf('Input file = %s\n', fin);

fprintf('Output file = %s\n', fout);

% read the image file

x = imread(fin);

% determine data dimensions

[M, N, colors] = size(x); % rows, cols, 3 for color, 1 for grayscale

% Write the header

% ndim = 2 (image)

% nchan = colors (RGB = 3, gray-scale = 1)

% dim0 = M

% dim1 = N

% dim2 = 0 (not used for images)

fid = fopen(fout, 'wb');

fwrite(fid, [2, colors, M, N, 0], 'int');

% Loop over pixel(i = row, j = col) -> R, G, B

for i = 1:M

for j = 1:N

for k = 1:colors

fwrite(fid, x(i, j, k), 'float');

end

end

end

% Release the file handler

fclose(fid);

end

Code File: **bin2image.m**

function [x] = bin2image(fin, fout)

% Construct output file name from input

if (nargin < 2)

fout = [fin(1:max(strfind(fin, '.'))), 'png'];

end

% Path correction

fin = [pwd,'\','Output\',fin];

fout = [pwd,'\','Output\',fout];

fprintf('Input file = %s\n', fin);

fprintf('Output file = %s\n', fout);

% Read the header

% ndim = 2 (image)

% nchan = colors (RGB = 3, gray-scale = 1)

% dim0 = M

% dim1 = N

% dim2 = 0 (not used for images)

fid = fopen(fin, 'rb');

ndim = fread(fid, 1, 'int');

colors = fread(fid, 1, 'int');

M = fread(fid, 1, 'int');

N = fread(fid, 1, 'int');

dim2 = fread(fid, 1, 'int');

% Read the data

[a, ~] = fread(fid, inf, 'float');

% Release the file handler

fclose(fid);

% Just use nested loops this time

x = zeros(M, N, colors);

for i = 1:M

for j = 1:N

for k = 1:colors

% 3 dimensional array access with 1-based indexing

x(i, j, k) = a((i - 1)\*N\*colors + (j - 1)\*colors + k);

end

end

end

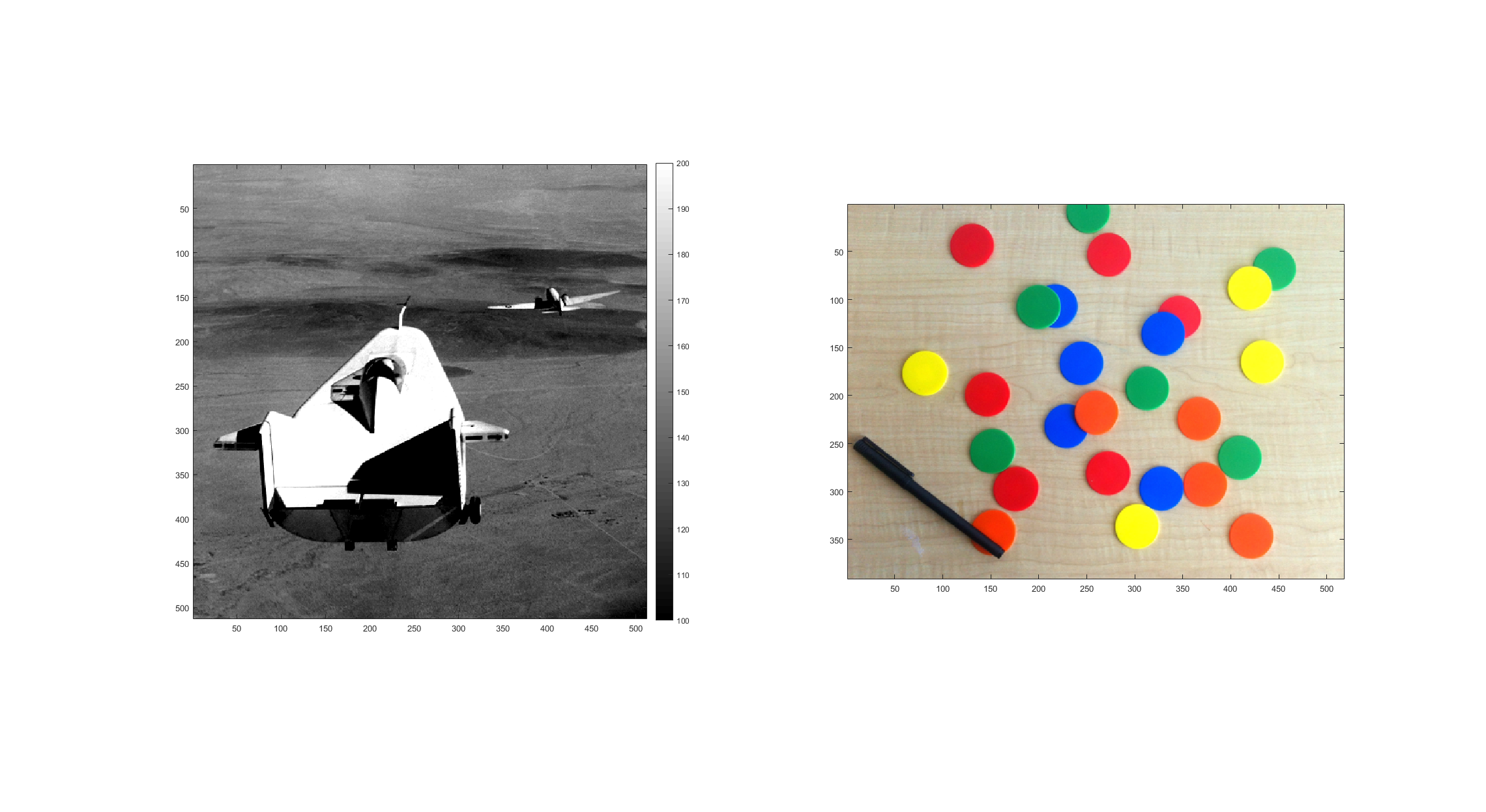
% Because imwrite is stupid, and requires type uint8

x = uint8(x);

% Write out the data

imwrite(x, fout, 'png');

end



In this case we were to parse images. The data processing is the same procedure as for audio processing, but the dimensions of the data we work with are different. Additionally, gray-scale uses one byte per pixel where standard color uses three.

Problem 3)

Code File: **p3.m**

% Parse the data

[vid] = video2bin('xylophone.mp4');

% Read formatted data

[file] = bin2video('xylophone.bin');

implay(file)

Code File: **video2bin.m**

function [vid] = video2bin(fin, fout)

% Construct output file name from input

if (nargin < 2)

fout = [fin(1:max(strfind(fin, '.'))), 'bin'];

end

% Path correction

fin = [pwd,'\','Resources\',fin];

fout = [pwd,'\','Output\',fout];

fprintf('Input file = %s\n', fin);

fprintf('Output file = %s\n', fout);

% read the video file

vid = VideoReader(fin);

frames = vid.NumberOfFrames; % apparently this is deprecated too?! >:|

M = vid.Height;

N = vid.Width;

Fs = vid.FrameRate;

%colors = vid.BitsPerPixel / 8;

colors = 3; % we can't store it so we can't support differences here

%x = read(vid); % read all frames - who cares that this is deprecated

%[M, N, colors, frames] = size(x);

% Write the header

% ndim = 3 (video)

% nchan = Fs

% dim0 = M

% dim1 = N

% dim2 = frames

fid = fopen(fout, 'wb');

fwrite(fid, [3, Fs, M, N, frames], 'int');

% Loop over pixel(i = row, j = col) -> R, G, B

for f = 1:frames

x = read(vid, f); % Don't care that this is deprecated, matlab sucks

for i = 1:M

for j = 1:N

for k = 1:colors

fwrite(fid, x(i, j, k), 'float');

end

end

end

end

% Release the file handler

fclose(fid);

end

Code File: **bin2video.m**

function [fout] = bin2video(fin, fout)

% Construct output file name from input

if (nargin < 2)

fout = [fin(1:max(strfind(fin, '.'))), 'mp4'];

end

% Path correction

fin = [pwd,'\','Output\',fin];

fout = [pwd,'\','Output\',fout];

fprintf('Input file = %s\n', fin);

fprintf('Output file = %s\n', fout);

% Read the header

% ndim = 3 (video)

% nchan = Fs

% dim0 = M

% dim1 = N

% dim2 = frames

fid = fopen(fin, 'rb');

ndim = fread(fid, 1, 'int');

Fs = fread(fid, 1, 'int');

M = fread(fid, 1, 'int');

N = fread(fid, 1, 'int');

frames = fread(fid, 1, 'int');

colors = 3; % We can't support gray-scale videos

%Write the data

size\_frame = M \* N \* colors;

vid = VideoWriter(fout, 'MPEG-4');

vid.FrameRate = Fs;

open(vid);

% Just use nested loops this time

for f = 1:frames

% Read frame by frame

[a, ~] = fread(fid, size\_frame, 'float');

x = zeros(M, N, colors);

for i = 1:M

for j = 1:N

for k = 1:colors

% 3 dimensional array access with 1-based indexing

x(i, j, k) = a(...

(i - 1) \* N \* colors + ...

(j - 1) \* colors + ...

k);

end

end

end

% Convert to uint8 just like images

x = uint8(x);

writeVideo(vid, x);

end

% Release the file handlers

fclose(fid);

close(vid);

end

This is where I really had grief with Matlab. The VideoWriter object has deprecated NumberOfFrames, which we really need to do this efficiently. Additionally, being able to set the nth frame of a video is deprecated and must be done sequentially. I can see this being a nuisance for some forms of data processing. Good thing we’ll do that in C (I hope).