Table of Contents

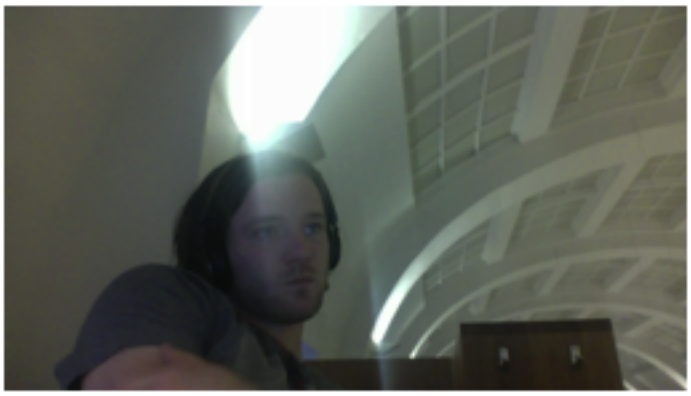
Convert Movie AVI to MPG and clock time

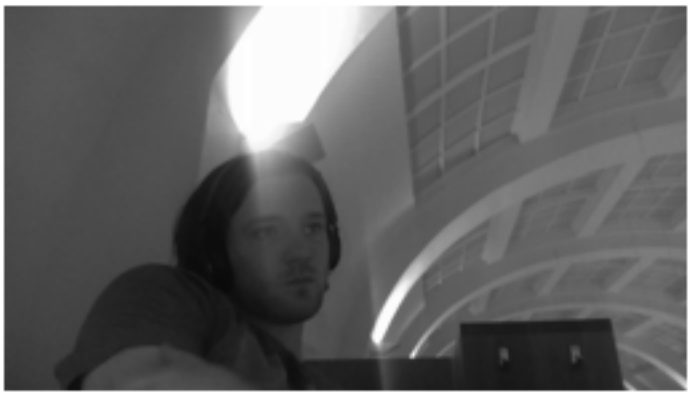
```
% First we open a Video reader/writer.
readerObj = VideoReader('testABCD.mp4');
writerObj = VideoWriter('testABCDAVI.avi', 'Uncompressed AVI');

% We scale down the original video to save computing time.
height = readerObj.Height * 0.2;
width = readerObj.Width * 0.2;
mov(1:readerObj.NumberOfFrames) = struct('data', zeros(height, width, 3, 'uint8'),

tic %Starts counting
open(writerObj);
for k = 1:readerObj.NumberOfFrames
    mov(k).data = imresize(read(readerObj,k), [height, width]);
    writeVideo(writerObj, mov(k).data);
end
close(writerObj);
disp('Time it took to convert MPG to AVI:');
disp(toc); %Display eleapsed time
```

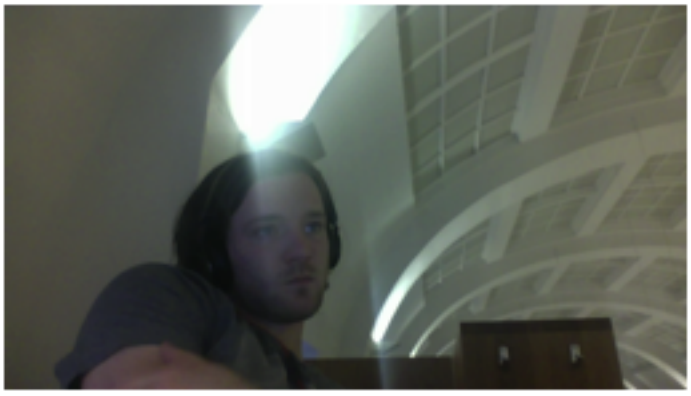
```
Time it took to convert MPG to AVI:
   25.2367
```

Show RGB -> YUV (Y) -> RGB

```
randomIndex = randi([1 readerObj.NumberOfFrames]);
randomFrameRGB = mov(randomIndex).data;
randomFrameYUV = frameRGB2YUV(randomFrameRGB);

randomFrameY = uint8(randomFrameYUV(:,:,1));
randomFrameBackToRGB = frameYUV2RGB(randomFrameYUV);

figure,
imshow(randomFrameRGB);
figure,
imshow(randomFrameY);
figure,
imshow(randomFrameBackToRGB);
```



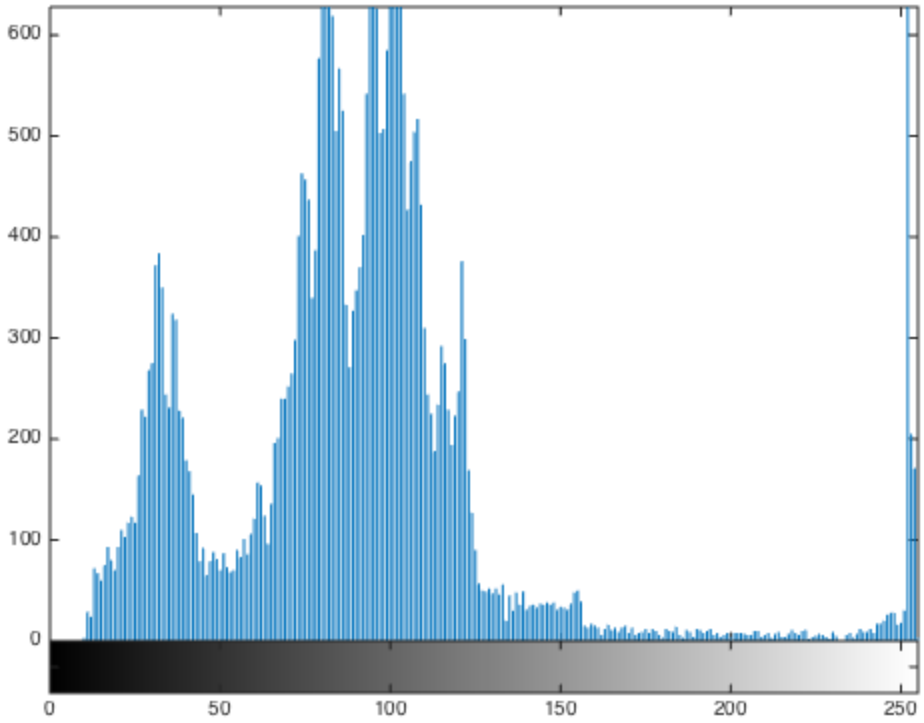




Huffman encoding / decoding

```
figure, imhist(randomFrameY);
disp('Histogram of frame used for Huffman Codebook:');
disp(entropy(randomFrameY));

codeBook = huffmanCodebook(randomFrameY, 0, 256);

randomIndex = randi([1 readerObj.NumberOfFrames]);
randomFrameYUV = frameRGB2YUV(mov(randomIndex).data);
randomFrameY = uint8(randomFrameYUV(:,:,1));

disp('Entropy before huffman encoding');
disp(entropy(randomFrameY));

randomFrameYHuffmanEncoded = huffmanEncoder(randomFrameY, codeBook);
disp('Entropy after huffman encoding:');
disp(entropy(randomFrameYHuffmanEncoded));

randomFrameYHuffmanDecoded = huffmanDecoder(randomFrameYHuffmanEncoded, codeBook,

if isequal(randomFrameY, randomFrameYHuffmanDecoded)
    disp('No data lost');
end
```

```
Histogram of frame used for Huffman Codebook:
    6.8281

Entropy before huffman encoding
    6.8166

Entropy after huffman encoding:
    0.9981

No data lost
```



Differential encoding/decoding

```
% First we make a random 200 x 200 matrix.
frame = uint8(rand(200, 250) * 255);
[height, width] = size(frame);

% Then we encode it with differential-encoding
frameDiffEncoded = diffEncoder(frame);

% Then we decode it with differntial-decoding.
frameDiffDecoded = diffDecoder(frameDiffEncoded, height, width);

% And to prove that the encoding works without data loss:

if isequal(frame, frameDiffDecoded)
    disp('No data loss');
end
```

```
No data loss
```

Published with MATLAB® R2014b

```
function YUVFrame  = frameRGB2YUV( RGBFrame )
```

Instruction

this function simply transforms a RGBframe to its corresponding YUVframe input RGBframe, size == [frameHeight * framewidth * 3]

```
% output
% YUVframe, size == [frameHeight * framewidth * 3]
% For Question 5 in specification, U and V should have only 25% size of Y

RGBFrame = double(RGBFrame); %Otherwise values are clipped in the converting proce
R = RGBFrame(:,:,1);
G = RGBFrame(:,:,2);
B = RGBFrame(:,:,3);

%Constans from lectures.
Y = 0.299 * R + 0.587 * G + 0.114*B;  %Process each channel individually.
U = 0.492 * (B-Y);
V = 0.877 * (R-Y);

YUVFrame = cat(3, Y, U, V); %Put it back together.

end
```

Published with MATLAB® R2014b

huffmanCodebook

```
function [ CodeBook ] = huffmanCodebook( imgBW, min, max )

% This function generates Huffman dictionary (aka codebook) from a BW image
% Input: imgBW, which is a black and white image (UINT8). min/max, which is
% the min and max value of pixel intensity.
% Output: CodeBook, which is a 'table' for converting 256 grayscale levels to
% the corresponding codewords
% Note: Some of the codewords might be 'null' due to the absence of certain pixel
% You can add the input/output arguments if needed.
```

Entropy and probability distribution

```
[height,width] = size(imgBW);
if (width>1)
    imgBW = reshape(imgBW, [1,(height*width)]);
end

[freq,symbols] = histcounts(imgBW, min:max);
symbols(end) = [];

P = freq./sum(freq);
```

Create codebook

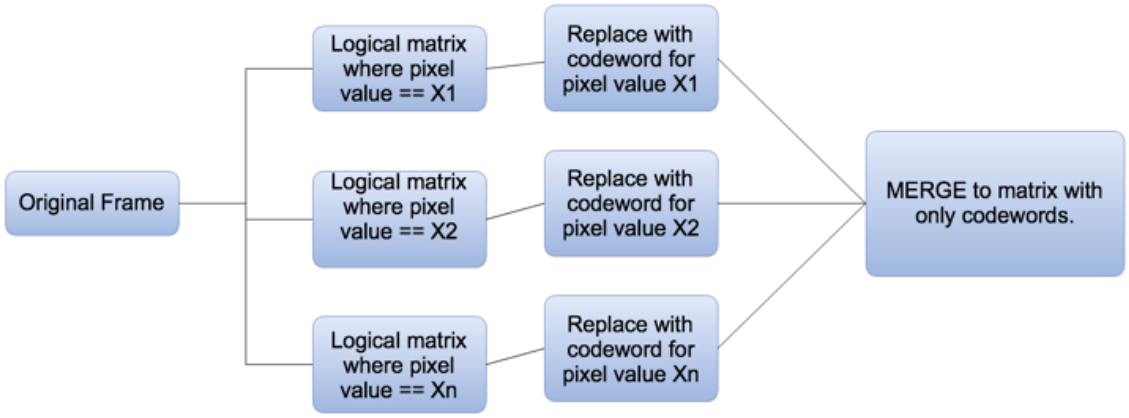
```
tempbook = cell(length(symbols),2); % List of codewords, same length as number of
for i=1:length(symbols)
    tempbook{i,1} = symbols(i);
end

mEvents = num2cell(1:length(P)); % Changing list for events, merged and single
mProb = P; % Changing list for probabilities, merged and single

% Loop until there is only 1 merged value left
while(length(mEvents) > 1)

    % Sort the list of merged/single probabilities
    % and retrieve both a list of probabilites and index values of the
    % events
    [sProb, sInd] = sort(mProb);

    % Get the events with the smallest probabilites from merged list
    % events are in a cell array with all the sub-events in the cell
    smallestE1 = mEvents{sInd(1)};
    smallestE2 = mEvents{sInd(2)};

    % Add a 0 to the codeword of the smallest probability
    % and all the sub-probabilities "under" the chosen node
    for i = 1:length(smallestE1)
        tempbook{smallestE1(i), 2} = [0, tempbook{smallestE1(i),2}];
    end
```

```
        % Add a 1 to the codeword of the second smallest probability
        % and all the sub-probabilities "under" the chosen node
        for i = 1:length(smallestE2)
            tempbook{smallestE2(i), 2} = [1, tempbook{smallestE2(i),2}];
        end

        % Add the events to the merged event cell array, as a cell of
        % both the events to be able to add code to them
        mEvents{end+1} = [smallestE1, smallestE2];

        % Add the sum of both the probabilites to the merged probabilities to
        % be able to sort them into the "tree"
        mProb(end+1) = sProb(1)+sProb(2);

        % Remove the single events and probabilities as they now exists as
        % merged cells
        mEvents(sInd(1:2)) = [];
        mProb(sInd(1:2)) = [];

    end

    CodeBook = tempbook;
```

Published with MATLAB® R2014b

```
function [ binaryVector ] = huffmanEncoder( imgBW, codeBook )

%This function encodes a black n white image into a single binary vector
% Input: 'bwImg' is black image, e.g. a Y frame of a YUV image
%        'CodeBook', is the codebook/dictionary generated by huffmanCodebook

% Output: encoded bwImg, a binary vector formed by concatenated huffman codewords

[height, width] = size(imgBW);

placeholder = cell(height, width);

for i = 1: length(codeBook)
    codeWord = codeBook(i, 2);
    logical = imgBW == codeBook{i,1}; %Targets only the pixels where the values ar
    placeholder(logical) = codeWord;
end

binaryVector = reshape(placeholder, 1, []);
binaryVector = cell2mat(binaryVector); %[ [1,2,3,4], [1,2,3,4] ] -> [1,2,3,4,1,2,3
```

Published with MATLAB® R2014b



huffmanDecoder

```
function [ decodedImg ] = huffmanDecoder( binaryVector, CodeBook, height, width)

% This function decodes a single binary vector into a black n white image
% Input: 'binaryVector' is a binary vector formed by concatenated huffman
%       codewords representing, e.g. a Y frame.
%         'codebook' is the generated by huffmanCodebook
%         'width' is the width of the image to be decoded
%         'height' is the height of the image to be decoded
% Output: decodedImage, Black and White, type uint8
```

Code

```
% Pre-allocate imagematrix, as a vector
tempimagevector = zeros(height*width, 1);

% Assign variables for faster computing
codebooklength = length(CodeBook);
binvectorlength = length(binaryVector);


% An index for the imagevector, to assign values into a pre-allocated
% vector for faster computing
pixelpos = 1;

% Loop until all elements are checked
i = 1;
while(i <= binvectorlength)

    % Reset the found variable
    found = 0;

    % Reset the codeword that is checked
    codeword = [];

    % Loop the binaries and add the next binary to the checked codeword if
    % the current doesn't exist in the codebook
    while(found == 0 && i <= binvectorlength)

        % Add the current binary to the checked codeword
        codeword(end+1) = binaryVector(i);

        % Loop the codebook to check the current codeword
        % if it exists, for-loop breaks and found-switch is true which
        % breaks the second while loop, that resets the codeword checked.
        for n=1:codebooklength
            if(isequal(codeword,CodeBook{n,2}))
                tempimagevector(pixelpos,1) = CodeBook{n,1};
                found = 1;
                pixelpos = pixelpos + 1;
                break;
            end
```

```
            end

            % Go to next index even if the codeword has been found or not
            i = i+1;

        end
    end

    % Reshape the vector to the right size as given by the input argument
    decodedImg = reshape(tempimagevector, [height,width]);
    decodedImg = uint8(decodedImg);

    end
```

Published with MATLAB® R2014b

```
function [ encodedVector ] = diffEncoder( imgBW )
```

diffEncoder

This function encodes a black and white image into a single vector using differential coding Input: BWImg: black and white image i.e. a Y frame

Output: encodedVector: the BWImg encoded into a single vector through differential-encoding.

```
[height, width] = size(imgBW);       % We need to know how many columns there is.
encodedImg = zeros(height, width);   % Placeholder for the encoded-img.

imgBW = double(imgBW);                  %Original type is uint8. uint8 cant be neg

for colIndex = 1: width                %For every column..

    if colIndex == 1                    %If first column, save as reference-value
        encodedImg(:, 1) = imgBW(:,1);

    else                                %Calculate difference between indexed colu
        encodedImg(:, colIndex) = imgBW(:, colIndex) - imgBW(:, colIndex - 1);

    end
end

encodedVector = reshape(encodedImg, [], 1); %Reshape the matrix C x R into a singl
```

Published with MATLAB® R2014b

diffDecoder

decodes the previously encoded image.

```
function [ decodedImg ] = diffDecoder( encodedVector, height, width )
%   encodedVector : nx1 vector with the image differential-encoded
%   columns : the amount of columns in the original image.
% ouput:
%   decodedImg: a NxR image decoded from the encodedVector.


% Reshape the vector into a matrix, so we can do column-wise operations.
encodedImg = reshape(encodedVector, height, width);
encodedImg = double(encodedImg);

decodedImg = zeros(height, width);   %Placeholder

for colIndex = 1: width               %Loop over all columns.

    if colIndex == 1                    %Reference value at the start of the rows,
        decodedImg(:, 1) = encodedImg(:,1);

    else                                %Take the last value and add the differenc
        decodedImg(:, colIndex) = decodedImg(:, colIndex - 1) + encodedImg(:, colI

    end
end

decodedImg = uint8(decodedImg);         %the encodedVector was type 'double' to su
end
```

Published with MATLAB® R2014b