Mandatory exercise 4
Signal and Image Processing 2012

Jens P. Raaby
frn617@diku.dk

September 30, 2012

Question 4.1 - Gamma Correction

To implement the user interface I used the GUIDE tool in Matlab. The image (in the form of an M x N array of pixels) is passed to the tool via an argument, for example

```
gammatool(imread('barbara.tif'));
```

The result of loading the tool is shown in the figure 1, and the source code is listed in appendix A. I set the slider component to take input in the range -1 to 1, which are applied as exponents of 10 to give gamma values in the range 0.1 to 10.

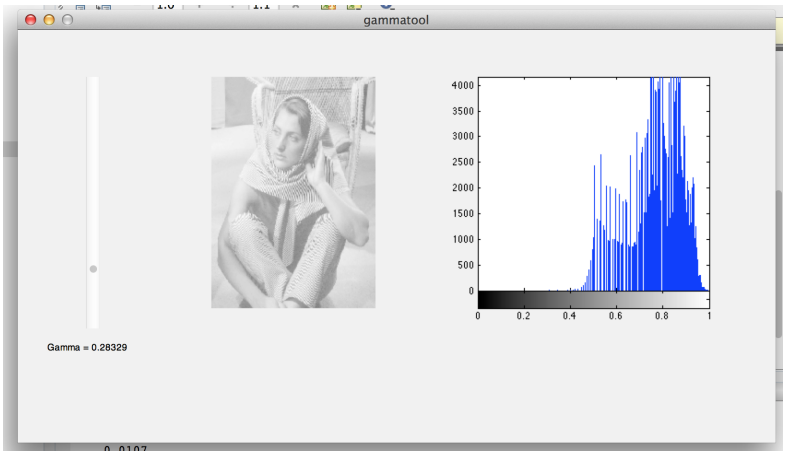


Figure 1: Demonstration of the Gamma tool developed for Q4.1

Question 4.2 - Monotonic intensity changes

The definition of a strictly decreasing function $g(r)$ is that when $r_2 < r_1$, $g(r_2) < g(r_1)$. The cumulative probability distribution always sums to 1, but in the decreasing case has the opposite gradient and thus the probability density function is different. When the function g is strictly decreasing and has an inverse, then that inverse function is also decreasing.

The domain and range of the inverse function can thus be expressed:

$$g^{-1}((-\infty, s]) = [g^{-1}(s), \infty) \tag{1}$$

Since $s = g(r)$ and $r = g^{-1}(s)$,

$$P_S(s) = P(R \in [g^{-1}(s), \infty)) \tag{2}$$

$$P_S(g(r)) = P_R(r) \Leftrightarrow P_S(s) = P_R(g^{-1}(s)) \tag{3}$$

We have the definition of $P_R(t)$ (it always takes value 1), and since we know the integration intervals in this decreasing case from above:

$$P_R(t) = \int_t^{\infty} p_r(x)dx + \int_{-\infty}^{t} p_r(x)dx = 1 \tag{4}$$

$$P_R(t) = \int_{-\infty}^{\infty} p_r(x)dx = 1 \tag{5}$$

Hence from (3) and (4):

$$P_S(t) = 1 - P_R(t) = 1 - P_R(g^{-1}(s)) \tag{6}$$

This follows because P_R can be expressed as the sum of the two integrals, and it always takes value 1. Since we know P_R is the integral of a decreasing function, and we know its range from (2), then this is rearranged as $1 - P_R(t)$.

Now the derivatives can be calculated:

$$\frac{d(P_S(t))}{dr} = \frac{d(P_S(g(r)))}{dr} = \frac{d(1 - P_R(r))}{dr} \tag{7}$$

Beginning with the right hand side:

$$\frac{d(1 - P_R(r))}{dr} = -p_R(r) \tag{8}$$

The left side can be derived using the chain rule:

$$\frac{d(P_S(g(r)))}{dr} = p_S(s)\frac{d(g(r))}{dr} = p_S(s)\frac{ds}{dr} = -p_R(r) \tag{9}$$

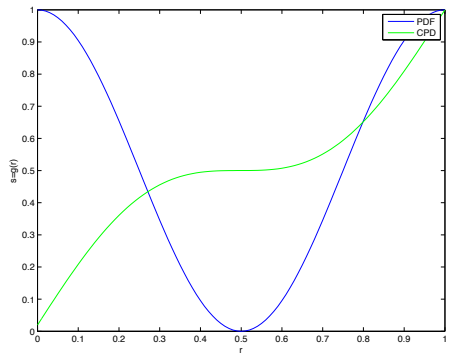
$$g'(s) = \frac{ds}{dr} = \frac{-p_R(r)}{p_S(s)} \tag{10}$$

$$g'(s) = \frac{-p_R(r)}{p_S(g(r))} \tag{11}$$

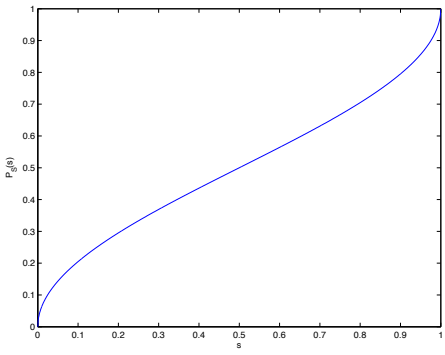
This is the complement of the equation given in the slides for a strictly increasing intensity change. The general formulation for a strictly monotonic intensity change is thus the:

$$g'(s) = |\frac{p_R(r)}{p_S(g(r))}| \tag{12}$$

In other words, the derivative of $g(s)$ is the probability density function of the input divided by the probability density function of the transformed input, and it will take a negative value if the change is monotonically decreasing, positive if increasing.



(a) Plot of $g(r)$ and its CPD



(b) CPD of S, $P_S(s)$

Figure 2: Plots for question 4.3

Question 4.3

We have the function over the interval $R \in [0, 1]$:

$$g(r) = \frac{1}{2}(1 + cos(2\pi r)) \tag{13}$$

To find the CPD of S in terms of the CPD of R, (13) must be rearranged in terms of r:

$$s = g(r) = \frac{1}{2}(1 + cos(2\pi r)) \tag{14}$$

$$2s = 1 + cos(2\pi r) \tag{15}$$

$$2s - 1 = cos(2\pi r) \tag{16}$$

$$cos^{-1}(2s - 1) = 2\pi r \tag{17}$$

$$r = \frac{cos^{-1}(2s - 1)}{2\pi} \tag{18}$$

In the given interval, this gives 2 roots, given some value for s. This can be inferred by inspecting the graph of $g(r)$ in figure 2a. Hence the next step is to determine these roots, r_1 and r_2, in terms of s.

The two roots r_1, r_2 are separated by the symmetry of the PDF around value $r = 0.5$. r_1 is given by (18), and r_2 is the symmetric case:

$$r_2 = \frac{2\pi - cos^{-1}(2s - 1)}{2\pi} = 1 - \frac{cos^{-1}(2s - 1)}{2\pi}$$

This is essentially a 'shift' of r_1 (the period of the function is 1, and of cos is 2π).

In order to express the cumulative probability distribution of S in terms of the CPD of R,

the set notation is used for the intermediate steps below:

$$
\begin{aligned}
P_S(s) &= P(S \leq s) && (19)\\
&= P(\{\omega | S(\omega) \leq s\}) && (20)\\
&= P(\{\omega | g(R(\omega)) \leq s\}) && (21)\\
&= P(\{\omega | g(R(\omega)) \in [r_1, r_2]\}) && (22)\\
&= P(\{\omega | g(R(\omega)) \in [\frac{cos^{-1}(2s-1)}{2\pi}, 1 - \frac{cos^{-1}(2s-1)}{2\pi}]\}) && (23)\\
&= P(\{\omega | g(R(\omega)) \leq \frac{cos^{-1}(2s-1)}{2\pi}\}) \cap P(\{\omega | g(R(\omega)) \leq \frac{1 - cos^{-1}(2s-1)}{2\pi}\}) && (24)\\
&= P_R(1 - \frac{cos^{-1}(2s-1)}{2\pi}) - P_R(\frac{cos^{-1}(2s-1)}{2\pi}) && (25)
\end{aligned}
$$

Note that this indicates that $P_S(s)$ is zero when $P_R(r_2) = P_R(r_1)$. The result of plotting $P_S(s)$ is shown in figure 2b.

Comparing the results

In figure 3 the intensity transformation $g(r)$ was applied to an image containing a (dithered) linear gradient from white to black. The histograms are shown below. Comparing the histogram afterwards (figure 3d) to the CPD of the function (shown in figure 2b) shows a similar straight section in the mid-tones of the image. The low and high tones (close to 0 and 1) in the transformed histogram are similar to the original histogram (i.e. higher intensity than the mid-tones). What is interesting from observing the image itself is that the midtones have been dropped towards black, and the dark tones have been boosted/swapped for white (figure 3b). This correlates with the shape of the function $g(r)$ shown in the blue plot in rigure 2a; the light tones (r near 1) are not significantly changed.

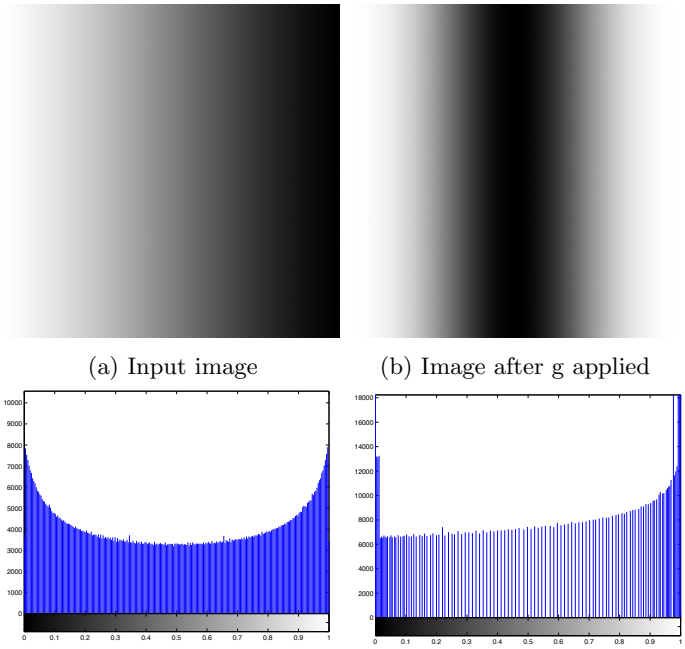
The CPD P_S is characterised by the predominantly flat central section with a steeper gradient close to the limits (0,1). The transformed histogram is also characterised by an almost linear central section with steep gradients towards the end. The central region is less dense than the original image's histogram, because the midtones have been altered by the transformation - they still remain present because there is still a gradient visible in the image, but it is a steeper change in tone (so there are gaps in the histogram).

Question 4.4 - Histogram equalisation and matching

Histogram equalisation is applied by the function histogram_equalise(image) defined in histogram_equalise.m (see appendix B).

The equalisation method involves non-whole nubers, because the CPD is calculated (and it is a sum of probabilities and therefore a fraction at each point in the range $(0, 1)$). These must be quantized back to integers in the equalised image and therefore some of the information will be lost. There is not an obvious way to remove this problem. One workaround could be to extend the range of intensities to 16 bit rather than 8 bit, but a lot of display hardware cannot show this many intensity levels. Another possible method to improve the situation would be to smooth the histogram of the original image (perhaps by blurring slightly) to give a more continuous input PDF. There would still be some rounding error but spikes in the histogram would be less likely.

The images in figure 4 show the result of histogram equalisation on the 'Barbara' image. It is clear from the histograms that the equalised image has a more spread out range of intensities,



(a) Input image (b) Image after g applied

(c) Histogram of original image (d) Histogram after g applied

Figure 3: Results of applying g

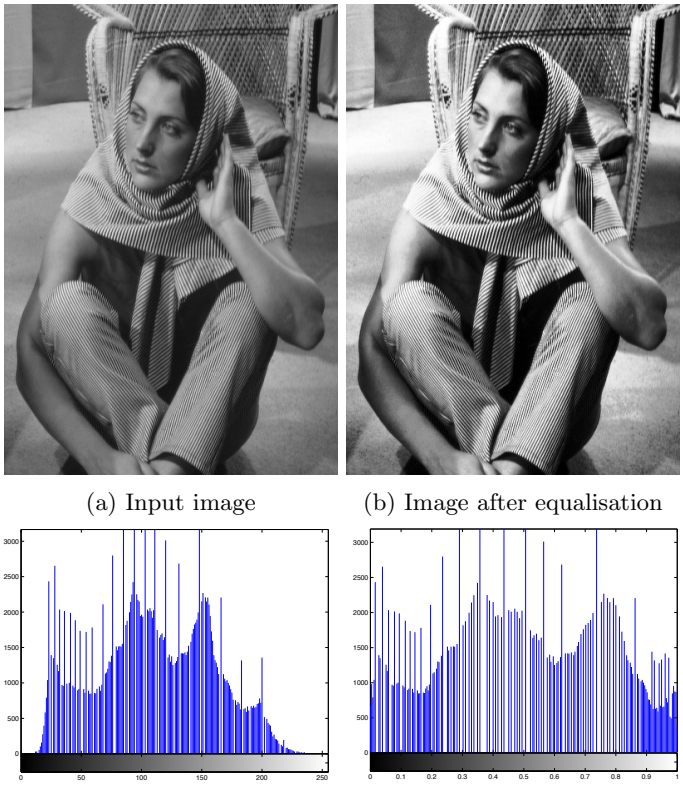
but the pattern is not perfectly uniform, and both gaps and spikes are exaggerated by the transformation.

Histogram matching is defined in histogram_match.m, as listed in appendix C.

The potential problems caused by quantization include matching an image to a histogram which is not monotonic in its intensity changes. The result is that there is not a 1-1 mapping between an intensity on the matched image and one in the original image (in other words, the matching has compressed the data). This also means it is not possible to restore the image back to its original intensities accurately, because there will be an ambiguity in the mapping of intensities.

In my example shown in figure 5e, the start and end of the histogram is almost flat (and thus potentially non-monotonically changing). This means there is an ambiguity in the matching process, and some of the data will be incorrectly mapped and detail lost. The cliff face in the upper middle part of the image in figure 5c has less contrast than in the original version, caused by this quantization error. If any given input intensity matches equally to two or more of the specimen intensities, then a decision must be made. This means that information is essentially lost or corrupted in the transformation. Gonzalez and Woods (page 133) suggest choosing the smallest value by convention. This might be a temporary solution, but is not always ideal.

Images with many pixels with intensities close to zero may have a problem being matched to a specimen histogram (this is the case with my example in figure 5a). The potential solution is to apply an additional histogram matching operation which smooths the transitions of dark pixels. The matching can be applied after this first transformation. This is the approach explained by Gonzalez and Woods (pp. 137-8).



(a) Input image (b) Image after equalisation

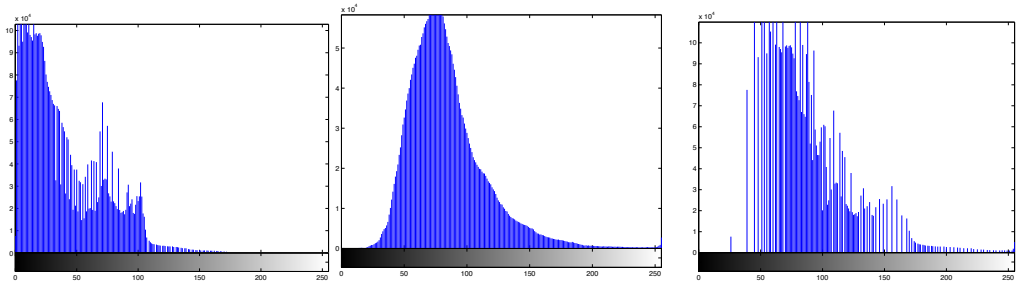
(c) Histogram of original image (d) Equalised histogram

Figure 4: Results of applying histogram equalisation

Question 4.5 - RGB to HSI and back again

My implementation of RGB to HSI and the inverse are listed in appendix D

(a) Input image (b) Specimen image (c) Image after matching



(d) Original histogram (e) Specimen histogram (f) Matched histogram

Figure 5: Results of applying histogram matching. Images taken by the author

A Gammatool.m source

```
function varargout = gammatool(varargin)
% GAMMATOOL MATLAB code for gammatool.fig
%      GAMMATOOL, by itself, creates a new GAMMATOOL or raises the existing
%      singleton*.
%
%      H = GAMMATOOL returns the handle to a new GAMMATOOL or the handle to
%      the existing singleton*.
%
%      GAMMATOOL('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in GAMMATOOL.M with the given input arguments.
%
%      GAMMATOOL('Property','Value',...) creates a new GAMMATOOL or raises the
%      existing singleton*.  Starting from the left, property value pairs are
%      applied to the GUI before gammatool_OpeningFcn gets called.  An
%      unrecognized property name or invalid value makes property application
%      stop.  All inputs are passed to gammatool_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help gammatool

% Last Modified by GUIDE v2.5 26-Sep-2012 21:06:04

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @gammatool_OpeningFcn, ...
                   'gui_OutputFcn',  @gammatool_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if (nargin < 1)
    error('Specify an image as input');
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before gammatool is made visible.
function gammatool_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to gammatool (see VARARGIN)

% Load the image from the arguments
handles.origImage = varargin{1};
handles.currentImage = handles.origImage;
axes(handles.imageAxes);
imshow(handles.currentImage);
axes(handles.histAxes);
imhist(handles.currentImage);
% Set the current data value.
% handles.current_data = handles.peaks;
% surf(handles.current_data)
```

```
% Choose default command line output for gammatool
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes gammatool wait for user response (see UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = gammatool_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;


% --- Executes on slider movement.
function tag_gammaslider_Callback(hObject, eventdata, handles)
% hObject    handle to tag_gammaslider (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
newGamma = logconvert(get(hObject,'Value'));
newImage = gammaTransform(handles.origImage,newGamma);
imshow(handles.imageAxes,newImage);%,'Parent',handles.imageAxes);
axes(handles.histAxes);
imhist(newImage);
% Hints: get(hObject,'Value') returns position of slider
%        get(hObject,'Min') and get(hObject,'Max') to determine range of slider


% --- Executes during object creation, after setting all properties.
function tag_gammaslider_CreateFcn(hObject, eventdata, handles)
% hObject    handle to tag_gammaslider (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: slider controls usually have a light gray background.
if isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end


% --- Executes on slider movement.
function gammaSlider_Callback(hObject, eventdata, handles)
% hObject    handle to gammaSlider (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Convert the linear slider scale to logarithmic
newGamma = logconvert(get(hObject,'Value'));
% Compute gamma transform of image
newImage = gammaTransform(handles.origImage,newGamma);
axes(handles.imageAxes);
imshow(newImage);
axes(handles.histAxes);
imhist(newImage);
gammaString = sprintf('Gamma = %0.5f',newGamma);
set(handles.gammaVal,'String',gammaString);

% --- Executes during object creation, after setting all properties.
function gammaSlider_CreateFcn(hObject, eventdata, handles)
% hObject    handle to gammaSlider (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: slider controls usually have a light gray background.
```

```
if isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end


function converted = logconvert( inputVal )
% LOGCONVERT Take a linear input value INPUTVAL and return 10^INPUTVAL
    converted = 10^inputVal;

function image_ = gammaTransform( image, gamma )
% GAMMATRANSFORM Takes an IMAGE and a GAMMA value and returns the
% transformed image.
    image_ = im2double(image); % One could also use rangeTransform 0 1 here
    image_ = image_ .^ gamma;
```

B histogram_equalise.m source

```
function equalised = histogram_equalise( image )
%HISTOGRAM_EQUALISE Equalises the histogram of IMAGE

    [M N] = size(image);
    h = imhist(image);
    n = numel(image);

    im_vec = reshape(image,1,n);

    % Build the cumulative distribution of intensity hists
    cumulative = zeros(1,length(h));
    cumulative(1) = h(1);
    for j = 2:length(h)
        cumulative(j) = cumulative(j-1) + h(j);
    end

    % Build the equalised image vector
    eq_vec = (zeros(size(im_vec)));
    for i=1:n
       eq_vec(i) = cumulative(im_vec(i))/n;
    end

    equalised = reshape(eq_vec,M,N);

end
```

C histogram_match.m source

```
function matched = histogram_match( image, histogram )
%HISTOGRAM_MATCH Matches the histogram of the supplied IMAGE to the
%HISTOGRAM

     [M N] = size(image);

    n = numel(image);

    % normalise supplied histograms (sum to 1)
    h = imhist(image);
    h = h./sum(h);
    h_s = length(h);
    h2 = histogram;
    h2 = h2/sum(h2);
    h2_s = length(h2);

    % Build the cumulative distribution of image histogram and new histogram
```

```
    cumulative = zeros(1,h_s);
    cumulative_new = zeros(1,h2_s);

    cumulative(1) = h(1);
    cumulative_new(1) = h2(1);
    for j = 2:h_s
        cumulative(j) = cumulative(j-1) + h(j);
        cumulative_new(j) = cumulative_new(j-1) + h2(j);
    end

    % Find the indices in the specimen histogram which are closest to the
    % values in the original histogram
    lookup = zeros(h_s+1);
    j = 1; % j is indexed from 1-256,
    %         - 0 is possible but matlab indices must start at 1
    for i=1:256
        if (cumulative(i) <= cumulative_new(j))
            lookup(i)=j;
        else
            while j<=255 && (cumulative(i) > cumulative_new(j))
                j=j+1;
            end
            if (cumulative_new(j) - cumulative(i)) > (cumulative(i) - cumulative_new(↩
                j-1))
                j = j-1;
                lookup(i) = j;
            else
                lookup(i) = j;
            end
        end
    end

    % process the pixels of the original image as a vector
    im_vec = reshape(image,1,n);
    t_vec = zeros(size(im_vec));
    for k=1:n
        t_vec(k) = lookup(im_vec(k)+1); % plus one because 0 is min. val
    end
    % ensure the returned image is in integer form:
    matched = cast(reshape(t_vec,M,N),'uint8');
end
```

D RGBtoHSI.m

```
function hsi_image = RGBtoHSI( rgb_image )
%RGBTOHSI Converts a given image in RGB to HSI representation

    [M N P] = size(rgb_image);
    n = numel(rgb_image)/P;
    hsi_image = zeros(M, N, 3);

    % separate the channels and convert to normalised double:
    if P==3
        R = im2double(rgb_image(:,:,1));
        G = im2double(rgb_image(:,:,2));
        B = im2double(rgb_image(:,:,3));
    end


    % for each channel, vectorise
    R_v = reshape(R,1,n);
    G_v = reshape(G,1,n);
    B_v = reshape(B,1,n);

    summed = R_v + G_v + B_v;
    i = summed/3;
    I = reshape(i,M,N);
```

```
    s = summed - 3*min([R_v; G_v; B_v],[],1);

    problematic = find(summed==0);
    summed(problematic) = 0.1; % avoid divide by zero!
    s = s./(summed);

    S = reshape(s,M,N);


    % handle the 360 subtraction using the bigger indices:
    bigger = find(B_v-G_v > 0);
    h = 0.5*((2*R_v) - G_v - B_v);
    denom = (R_v.^2) + (B_v.^2) + (G_v.^2) - (R_v.*G_v) - (R_v.*B_v) - (B_v.*G_v);
    h = h./(sqrt(denom));
    h = acos(h);
    h(bigger) = 360 - h(bigger);
    H = reshape(h,M,N);



    hsi_image(:,:,1) = H;
    hsi_image(:,:,2) = S;
    hsi_image(:,:,3) = I;
```

E HSItoRGB.m

```
function rgb_image = HSItoRGB( hsi_image )
%HSItoRGB Converts a given image in HSI to RGB representation

    [M N P] = size(hsi_image);
    n = numel(hsi_image)/P;

    % separate the channels and convert to normalised double:
    if P==3
        H = (hsi_image(:,:,1));
        S = (hsi_image(:,:,2));
        I = (hsi_image(:,:,3));
    end

    % for each channel, vectorise
    H_v = reshape(H,1,n);
    S_v = reshape(S,1,n);
    I_v = reshape(I,1,n);


    % apply the formulae from the slides

    % divide the pixels by hue
    sub120 = find(H_v<120);
    sub240 = find(H_v>=120 & H_v<240);
    rest   = find(H_v>=240 & H_v<360);

    % handle each case of hue
    B(sub120) = I_v(sub120).*(1-S_v(sub120));
    R(sub120) = I_v(sub120).*(1+(S_v(sub120).*cos(H_v(sub120))./cos(60-H_v(sub120))))↩
        ;
    G(sub120) = 3*I_v(sub120) - R(sub120) - B(sub120);

    H_v(sub240) = H_v(sub240) - 120;
    R(sub240) = I_v(sub240).*(1-(S_v(sub240)));
    G(sub240) = I_v(sub240).*(1+(S_v(sub240).*cos(H_v(sub240)))./cos(60-H_v(sub240)))↩
        ;
    B(sub240) = 3*I_v(sub240) - R(sub240) - G(sub240);

    H_v(rest) = H_v(rest) - 240;
    G(rest) = I_v(rest).*(1-S_v(rest));
    B(rest) = I_v(rest).*(1+(S_v(rest).*cos(H_v(rest)))./cos(60-H_v(rest)));
    R(rest) = 3*I_v(rest) - B(rest) - G(rest);
```

```
    rgb_image(:,:,1) = reshape(R,M,N);
    rgb_image(:,:,2) = reshape(G,M,N);
    rgb_image(:,:,3) = reshape(B,M,N);
```