

Chapter 2

Working with Pixels

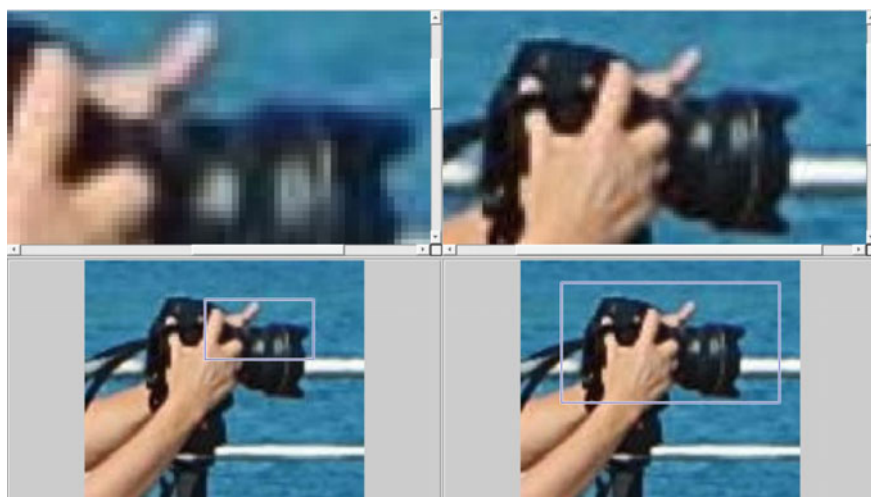


Fig. 2.1 Lower versus Higher resolution image pixels

2.1 Picture Elements

A **pixel** (*aka* picture element) is an element at position (r, c) (row, column) in a digital image I . A pixel represents the smallest constituent element in a digital image. Typically, each pixel in a raster image is represented by a tiny square called a **raster image tile**. Raster image technology has its origins in the raster scan of cathode ray tube (CRT) displays in which images are rendered line-by-line by magnetically steering a focused electron beam. Usually, computer monitors have bitmapped displays

in which each screen pixel corresponds to its **bit depth**, i.e., number of pixels used to render pixel colour channels.

By zooming in on (also, **resample**) an image at different magnification levels, these tiny pixel squares become visible.

Example 2.1 Inspecting Raster Image Pixels.

Four views of a raster image are shown in Fig. 2.1:

- 1° Lower-left panel: hand-held camera with pixel inspection window:



. This is a movable window that makes it possible to inspect different parts of the image.



- 2° Upper-left panel: pixels (zoomed in at 800%) inside the inspection window.

- 3° Lower-right panel: hand-held camera with pixel inspection window:



. This is a second movable window that makes it possible to inspect different parts of the image.



- 4° Upper-left panel: pixels (zoomed in at 400%) inside the inspection window.

See MScript A.10 in Appendix A.2.1 to experiment with other zoomed-in levels and inspect pixels in other images. ■

Example 2.2 Inspecting Raster Image Pixels.

Four views of a raster image are shown in Fig. 2.2:

- 1° Lower-right panel: hand-held camera with pixel inspection window:



. This is a second movable window that makes it possible to inspect different parts of the image.

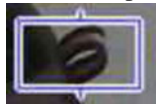


- 2° Upper-left panel: pixels (zoomed in at 100%) inside the inspection window.



Fig. 2.2 Zoom in at 100 and 800%, exhibiting colour image pixels

3° Lower-left panel: hand-held camera with pixel inspection window:



. This is a movable window that makes it possible to inspect different parts of the image.



4° Upper-left panel: pixels (zoomed in at 800%) inside the inspection window.

See MScript A.10 in Appendix A.2.1 to experiment with other zoomed-in levels and inspect pixels in other images. ■ ■

Each colour or greyscale or binary image pixel carries with it several numerical values. There are a number of cases to consider.

- Binary image pixel values:** 1 for a white pixel and 0 for a black pixel.
- Greyscale image pixel values:** Commonly 0–255, for the pixel greyscale intensity. Each greyscale pixel value quantizes the magnitude of white light for a pixel.
- RGB image pixel values:** Each colour pixel value quantizes the magnitude of a particular colour channel brightness for a pixel. A **colour channel** is particular colour component of an image and corresponds to a range of visible light wavelengths. Each color pixel contains intensities for three colour channels. For

colour pixel with a bit depth equal to 8, we have the following range of intensity (brightness) values for each colour channel.

Red: 0–255, for red pixel intensity (brightness).

Green: 0–255, for green pixel intensity (brightness).

Blue: 0–255, for blue pixel intensity (brightness).

Let $I^k(u, v)$ be the intensity of the k colour channel at camera image cell (u, v) , Λ , the set of wavelengths in the visible spectrum, p_0^k , a scaling factor, λ , a particular wavelength, $E_{u,v}(\lambda)$, the amount of incoming light at image cell (u, v) , $\tau^k(\lambda)$, the filter transmittance for the k colour channel, and $s(\lambda)$, the spectral responsivity of a camera optical sensor. The final colour pixel value $I^k(u, v)$ is defined by

$$I^k(u, v) = p_0^k \int_{\Lambda} E_{u,v}(\lambda) \tau^k(\lambda) s(\lambda) d\lambda.$$

In a typical RGB camera, $k \in \{r, g, b\}$. Recently, color pixel values have been used extensively in image segmentation [135, Sect. 2.1, p. 666] and for visual object tracking [32, Sect. 2.1, p. 666]. ■

Separating and Modifying Colour Image Channels

Colour channel values can be separated and pixel values can be modified.

2.2 Separating Colour Image Channels

It is a straightforward task to separate the colour channels in a raster colour image. We illustrate this using notation from Matlab.

Image and Colour Channel Notation

Let img be a $m \times n$ colour image. In that case, the pixels in img can be accessed in the following ways.

$img(:, :)$ = pixel values in all rows and all columns in img .

$img(r, :)$ = all pixel values in row r in img , $1 \leq r \leq m$.

$img(:, c)$ = all pixel values in column c in img , $1 \leq c \leq n$.

$img(:, :, 1)$ = all red channel values in all rows and all columns in img .

$img(:, :, 2)$ = all green channel values in all rows and all columns in img .

$img(:, :, 3)$ = all blue channel values in all rows and all columns in img .

The notation $img(:, :)$, $img(r, :)$, $img(:, c)$ can be used to inspect and change pixel intensities in binary, greyscale or colour images.

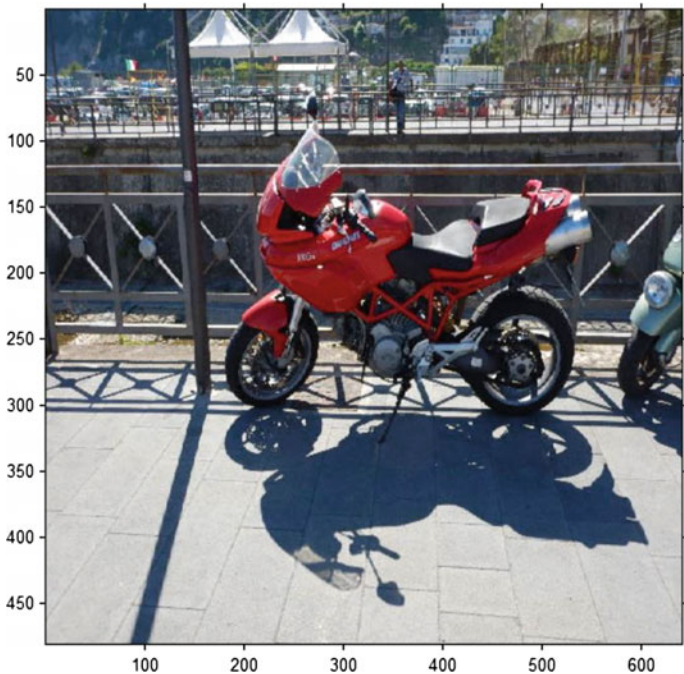


Fig. 2.3 Sample colour image

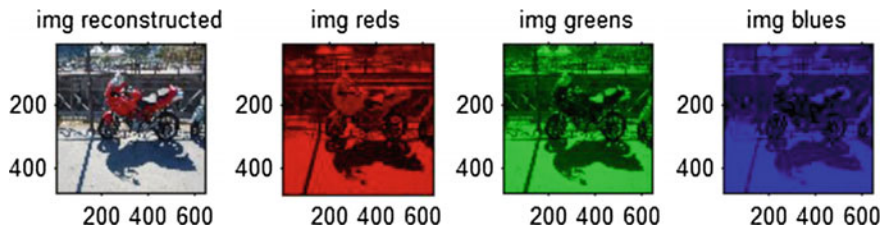


Fig. 2.4 Separated and combined colour image channels

Example 2.3 Separating Colour Image Channel values.

A sample colour image is given in Fig. 2.3. Using the MScript A.11 in Appendix A.2.1, the colour channels exhibited by Fig. 2.3 are separated and then recombined in Fig. 2.4. ■

2.3 Colour to Greyscale Conversion

Ideally, a colour channel value indicates the magnitude of the colour channel light recorded by an optical sensor used in pixel formation of a colour image produced by a digital camera. Let I be a colour image. It is possible to convert the colour image I to a greyscale image I_{gr} using the Matlab function `rgb2gray(I)`.

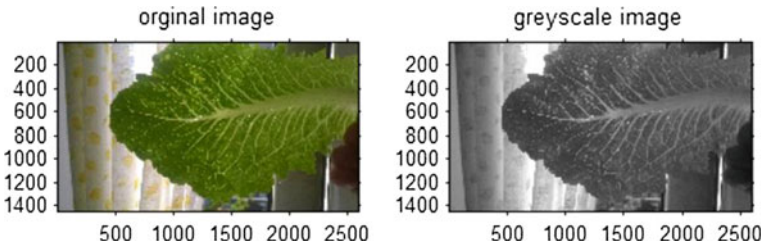


Fig. 2.5 Colour to greyscale conversion

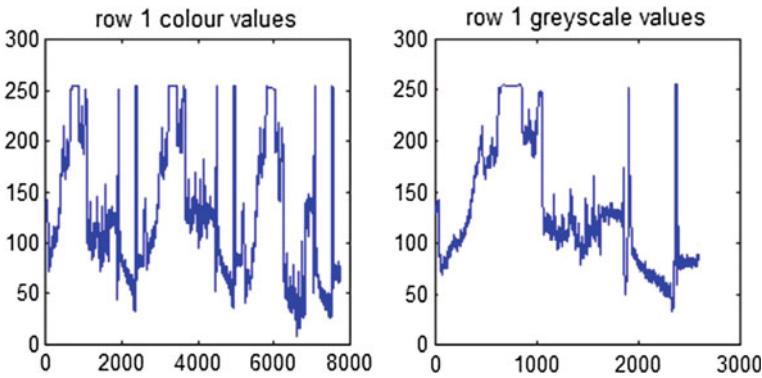
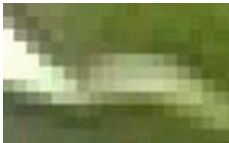


Fig. 2.6 Row of colour and greyscale leaf image intensities

Example 2.4 Figure 2.5 shows the result of converting a leaf colour image to a greyscale image using MScript A.12 in Appendix A.2.3. Contrast between the two forms of images becomes clearer when we zoom in on subimages of the original image and its greyscale counterpart.



Colour Subimage

In this leaf image segment, there is a visible mixture various shades of greens.

Recall that shades of green are obtained by mixing yellow and blue. It is a straightforward task to verify that visible green in an image is rendered digitally with a mixture of red and blue channel intensities.



Colour Subimage

In this leaf image segment, the original mixture various shades of greens is replaced by a mixture of greys. The change from pixel color intensity to greyscale intensity can be seen in the sample pixel values in Fig. 2.6. ■

At pixel level, pixel modification can be carried out by replacing each pixel in a colour or greyscale image I with either the average of the colour channel values or maps of pixels values to real numbers using functions such as $\ln(x)$, $\exp(x)$ or with a weighted sum of the colour channel values. For example, for a greyscale image pixel $I_{gr}(x, y)$ at (x, y) , the pixel intensity of I_{gr} at (x, y) is

$$I_{gr}(x, y) = \alpha I(x, y, r) + \beta I(x, y, g) + \gamma I(x, y, b),$$

where the weighting coefficients α, β, γ approximate the perceptual response of the human eye to the red, green and blue (r, g, b , respectively) colour band values.

There is a NTSC (National Transportation Safety Commission) television standard for greyscale image pixels such that

$$\alpha = 0.2989, \beta = 0.5870, \gamma = 0.1140$$

In Matlab, we write

$$I_{gr}(x, y) = \alpha \cdot I(x, y, 1) + \beta \cdot I(x, y, 2) + \gamma \cdot I(x, y, 3),$$

Problem 2.5 Given a colour image such as $I = \text{rainbow.jpg}$, do the following.

- (step.1) Choose your own values for the weighting coefficients α, β, γ .
- (step.2) Convert a column 30 pixels wide to greyscale intensities in image I . Call the new image I_{gr} .
- (step.3) Display the resulting mixed colour-greyscale image I_{gr} .
- (step.4) ☞ Use **cpselect** to compare the pixels in the 5 wide pixel column in I and I_{gr} . ■

2.4 Algebraic Operations on Pixel Intensities

In this section, we consider various operations on image pixel intensities, resulting in changes in visual appearance of the image. Let g be a digital image (either colour

or greyscale). Let $k \in [0, 255]$. Then new images i_1, i_2, i_3, i_4 are obtained using the image variable in simple algebraic expressions.

Image Algebraic Expressions I.

$$i_1 = g + g,$$

$$i_2 = (0.5)(g + g),$$

$$i_3 = (0.3)(g + g),$$

$$i_4 = \left(g \left(\frac{g}{2}\right)\right) (0.2).$$

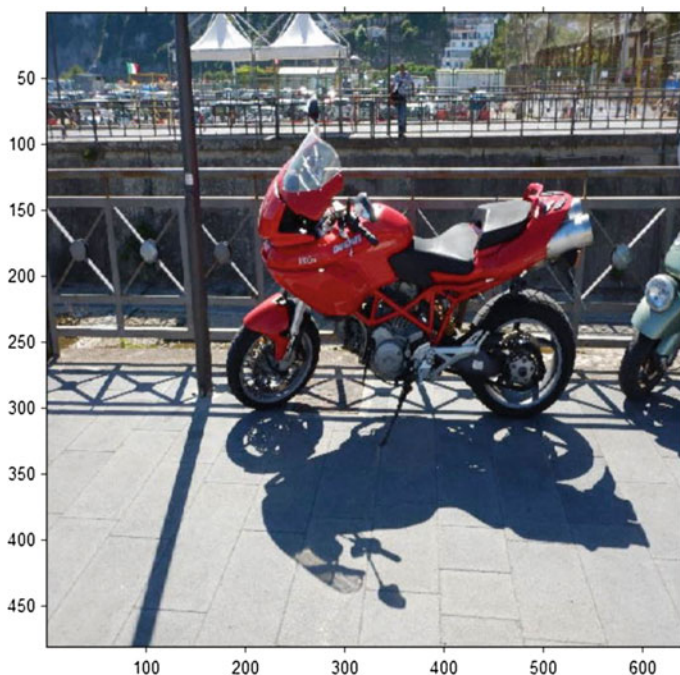


Fig. 2.7 Sample colour image

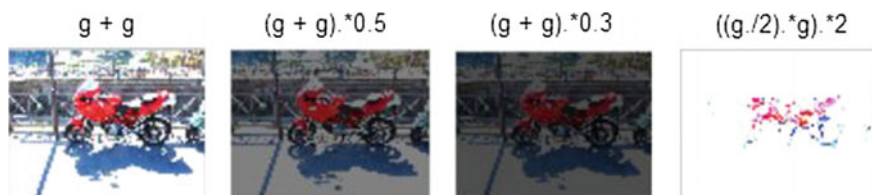


Fig. 2.8 Pixel value intensity changes induced by algebraic expressions I

Example 2.6 Algebraic Operations I on Images.

In algebraic operations I, notice that image g is added to itself. Various algebraic expressions can be put together to modify the pixel values in an image. MScript A.13 in Appendix A.2.3 implements algebraic expressions I on the colour image in Fig. 2.7 to obtain the resulting images shown in Fig. 2.8. For example, the implementation of $g + g$ in the leftmost image i_1 in Fig. 2.8 results in a brighter image (all the pixel intensities in the cycle image g have been doubled). ■

Let h be a colour image and use the following algebraic expressions to change the pixel intensities in h .

Image Algebraic Expressions II

$$i_5 = h + 30,$$

$$i_6 = h - (0.2)h,$$

$$i_7 = |h - (0.2)(h + h)|,$$

$$i_8 = (0.2)(h + (0.5)(h + h)).$$



Fig. 2.9 Sample colour image

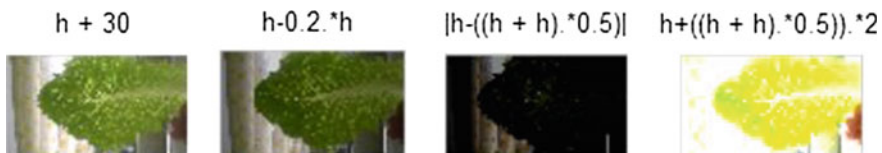


Fig. 2.10 Pixel value intensity changes induced by algebraic expressions II

Example 2.7 Algebraic Operations II on Images.

In algebraic operations II, notice that 30 is added to each of the intensities in image h . MScript A.14 in Appendix A.2.3 implements algebraic expressions II on the leaf colour image in Fig. 2.9 to obtain the resulting images shown in Fig. 2.10. For example, the implementation of $(0.2)(h + (0.5)(h + h))$ in the rightmost image in Fig. 2.10 results in a brighter image (all the pixel intensities in the cycle image are sharply increased). ■

Let img be a colour image and use the following algebraic expressions to change the pixel intensities in img .

Image Algebraic Expressions III.

- $i_9 = (0.8)img(:, :, 1)$ decrease red channel intensities,
- $i_{10} = (0.9)img(:, :, 2)$ slightly decrease green channel intensities,
- $i_{11} = (0.5)img(:, :, 2)$ sharply decrease green channel intensities,
- $i_{12} = (16.5)img(:, :, 3)$ sharply increase blue channel intensities.

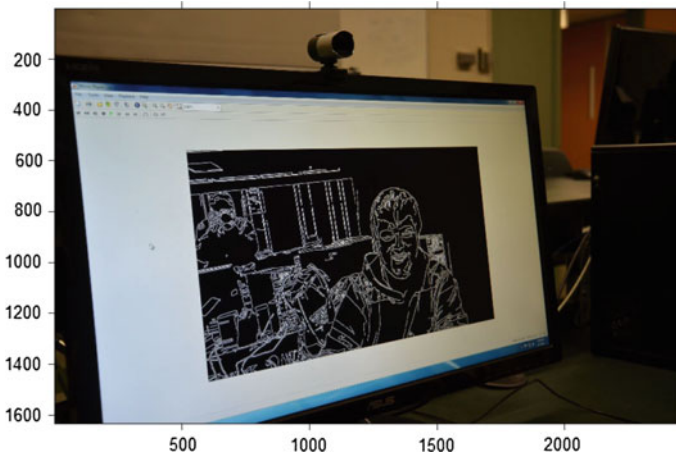


Fig. 2.11 Sample video frame image showing scene edges

Example 2.8 Algebraic Operations III on Images.

In algebraic operations III, notice that the pixel intensities in each colour channel are decreased by varying amounts or increased by a huge amount (in the blue channel). MScript A.15 in Appendix A.2.3 implements algebraic expressions III on the video frame colour image in Fig. 2.11 to obtain the resulting images shown in Fig. 2.12. For

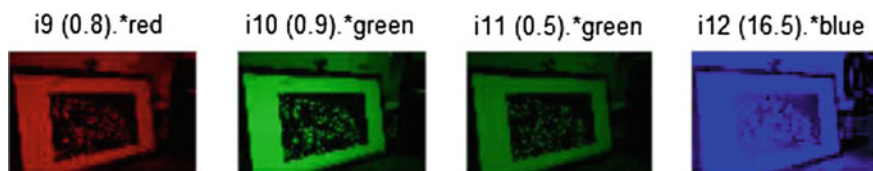


Fig. 2.12 Color channel pixel intensity changes induced by algebraic expressions III

example, the implementation of $(16.5)img(:, :, 3)$ in the rightmost image in Fig. 2.12 results in a brighter blue channel image (all the blue pixel intensities in the video frame image are sharply increased). ■



2.13.1: Colour Array




2.13.2: Sq

Fig. 2.13 Sample Thai grocery shelves

Problem 2.9 Offline Video Frame Colour Channel Changes.

Use the approach to changing image channel intensities in MScript A.15 in Appendix AA.2.3 as a template for offline video processing, do the following.

- 1°  Using Matlab script A.8 in Appendix A.1.5 as a template for offline video processing, change the red channel intensities in each video frame image. **Hint:** Replace the lines of Voronoï tessellation code with lines to code in MScript A.15 to handle and display changes in the green channel of each video frame image.

- 2° Repeat Step 1 to change the green channel intensities in each video frame image.
- 3° Repeat Step 1 to change the blue channel intensities in each video frame image.



Problem 2.10 Real-Time Video Frame Colour Channel Changes.

Use the approach to changing image channel intensities in MScript A.15 in Appendix A.2.3 as a template for real-time video processing, do the following.

- 1° ☕ Using Matlab script A.9 in Appendix A.1.5 as a template for offline video processing, change the red channel intensities in each video frame image. **Hint:** Replace the lines of Voronoï tessellation code with lines to code in Matlab script A.8 to handle and display changes in the green channel of each video frame image in real-time.
- 2° Repeat Step 1 to change the green channel intensities in each video frame image in real-time.
- 3° Repeat Step 1 to change the blue channel intensities in each video frame image in real-time.



Distinct images g and h can be added, provided the images are approximately the same size. To combine pixel values in different images, it is necessary that the distinct images I, \Im have the same dimensions. To get around this same-size images problem, choose any $n \times m$ image img , which is the larger of two images and just copy a second image onto an $n \times m$ array of 1s or 0s (call it *copy*). Then img and *copy* can be combined in various ways.

Example 2.11 Combining Pixel Intensities Across Separate Images.

The images in Fig. 2.13 showing Thai grocery store shelves. These Thai shelf images are both approximately 1.5 MB. MScript A.16 in Appendix A.2.1 illustrates how to combine pixel intensities in pairs of different images. Two Thai grocery shelf images are combined in different ways is the first row of images in Fig. 2.14. The second row of images in Fig. 2.14 are result of algebraic operations on just one of the original images.



Problem 2.12 Choose three different pairs of colour images g, h and do the following.

- 1° 🚲 In Image Algebraic Expressions I, replace g, g with g, h and display the changed images using MScript A.16 in Appendix A.2.1.
- 2° Repeat Step 1 using the Image Algebraic Expressions II.
- 3° Repeat Step 1 using the Image Algebraic Expressions III.



There are many other possibilities besides the constructed images I_1, \dots, I_{12} using the Algebraic Operations I, II and III. For example, one can determine largest red colour value in a selected image row r , using

$$[r, c] = \max(g(\text{row}, :, 1)).$$

Using $g(r, c)$, new images can be constructed by modifying the red channel values using a maximum red channel value.

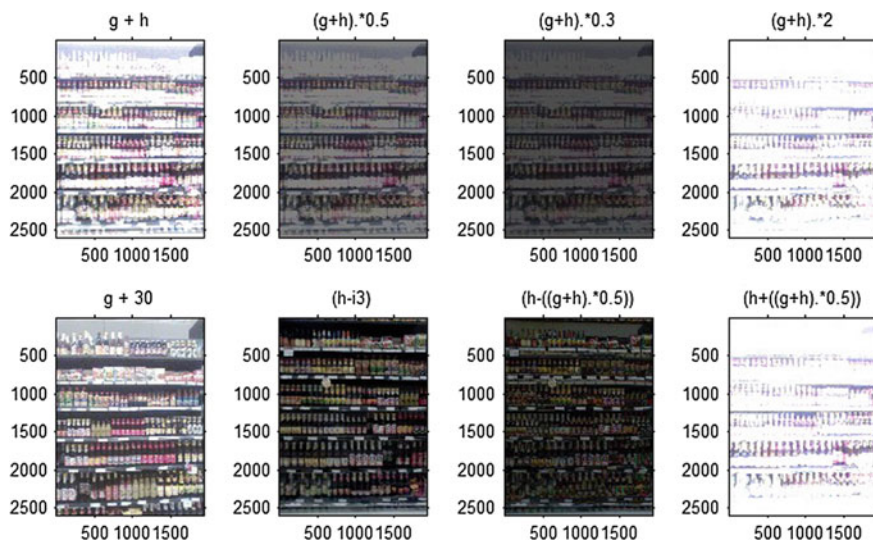


Fig. 2.14 Combining image pixel values using `thai.m`

Example 2.13 Experiment with maximum pixel intensities.

Sample results using MScript A.17 in Appendix refApCh2Sec:PixelValueChanges are shown in Fig. 2.15. This is an external view of the modified red channel intensities

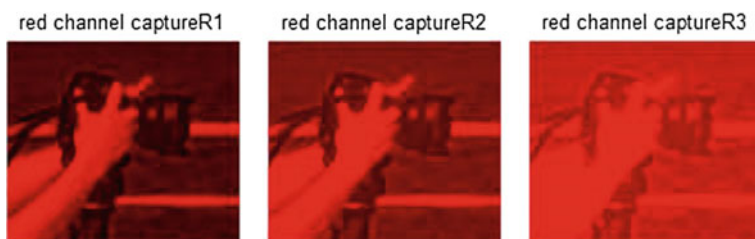


Fig. 2.15 External colour view of max-modified red channel intensities



Fig. 2.16 Internal greyscale view of max-modified red channel intensities

obtained by adding a fraction of a maximum red channel intensity in the first row of an image. Internally, a colour channel is just a greyscale image (not what we would imagine). An internal view of the modified red channel intensities is shown in Fig. 2.16. ■

Internal View of Colour Image Channels

Internally, a colour image channel is viewed as a greyscale image.

Problem 2.14 🚲 Example 2.13 illustrates the addition of fractions of a maximum red channel intensity. For three colour images your own choosing, do the following.

- (step.1) Use the **min** instead of **max** function to find a minimum red channel value for an entire colour image.
- (step.2) Subtract a fraction of the maximum red channel intensity from each of the original red channel intensities.
- (step.3) Display the results both as colour images and greyscale images. ■

Problem 2.15 🚲 Repeat the steps in Problem 2.14 using a minimum colour channel intensity. ■

Finding Image Edges

The hardest thing of all is to find a black cat in a dark room, especially if there is no cat.

—Confucius [114].

2.5 Pixel Selection Illustrated with Edge Pixel Selection

One of the commonest forms pixel selection is in the form of edge pixels. The basic approach is to detect those pixels that are on edges in either in a greyscale image or in a colour channel.

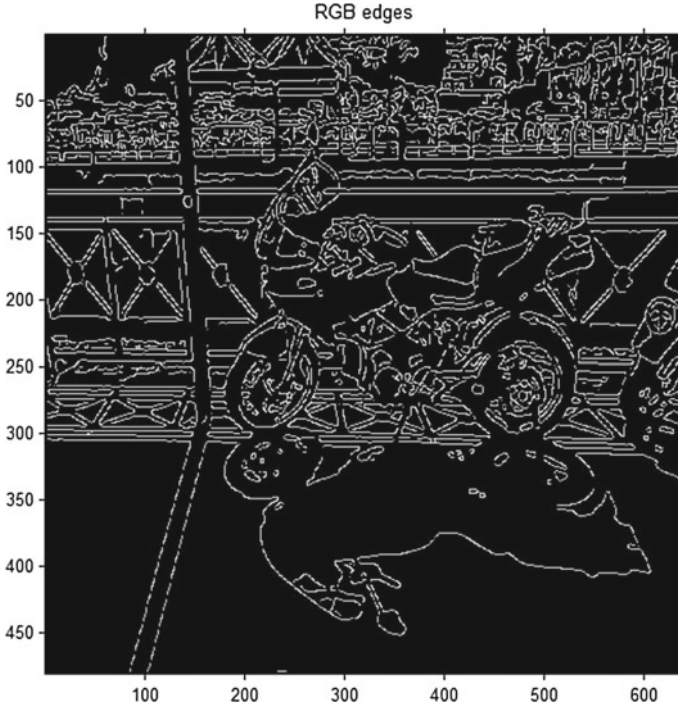


Fig. 2.17 Sample greyscale image edges

Briefly, to find edge pixels, we first find the gradient orientation (gradient angle) of each image pixel, i.e., angle of the tangent to each pixel. Let img be a 2D image and let $img(x, y)$ be a pixel at location (x, y) . Then the **gradient angle** φ of pixel $img(x, y)$ is found in the following way.

$$G_x = \frac{\partial img(x, y)}{\partial x},$$

$$G_y = \frac{\partial img(x, y)}{\partial y},$$

$$\varphi = \tan^{-1} \frac{G_y}{G_x} = \tan^{-1} \left(\frac{\frac{\partial img(x, y)}{\partial y}}{\frac{\partial img(x, y)}{\partial x}} \right).$$

In Canny's approach to edge pixel detection [24], each image is filtered to remove noise, which has the visual effect of smoothing an image. After the gradient orientation for each pixel is found, then a double threshold for an hysteresis interval on orientation angles is introduced by Canny. The basic idea is to choose all pixels with gradient orientations that fall within the hysteresis interval. Edge pixels that fall within the selected hysteresis interval are called **strong edge pixels**. All edge pixels

with gradient angles outside the hysteresis interval are called **weak edge pixels**. The weak edge pixels are ignored.

Algorithm 7: Colour Channel Edges Selection

```

Input : Read digital image img.
1 /* Capture colour image channel edges. */ ;
   Output:  $img \mapsto edgesR, edgesG, edgesB$ .
2 /* Capture red channel pixel intensities. */ ;
3  $gR \leftarrow img(:, :, 1)$ ;
4 /* Capture green channel pixel intensities. */ ;
5  $gG \leftarrow img(:, :, 2)$ ;
6 /* Capture blue channel pixel intensities. */ ;
7  $gB \leftarrow img(:, :, 3)$ ;
8 /* Capture blue channel pixel intensities. */ ;
9  $edge(gR, 'canny') \mapsto imgR$ ;
10  $edge(gG, 'canny') \mapsto imgG$ ;
11  $edge(gB, 'canny') \mapsto imgB$ ;
12 /* Map edge pixel intensities in each channel onto a black channel image bk. */ ;
13  $edgesR \leftarrow cat(3, imgR, bk, bk)$ ;
14  $edgesG \leftarrow cat(3, bk, imgG, bk)$ ;
15  $edgesB \leftarrow cat(3, bk, bk, imgB)$ ;
16 /* Capture modified black image embossed with channel edges. */ ;
17 Display  $edgesR, edgesG, edgesB$ ;

```

Before we separate out the edges from each colour image channel, we consider the conventional approach to separating greyscale image edges embossed as white pixels on a binary image.

Example 2.16 Figure 2.17 shows the result of finding the strong edge pixels in a greyscale image derived from a colour image using MScript A.18 in Appendix A.2.5. The basic approach is to start by converting a colour image to a greyscale image. If we ignore the location of each colour pixel, then a colour image is an example of a 3D image. Mathematically, each pixel p in location (x, y) in a colour image is described by a vector (x, y, r, g, b) in a 5-dimensional Euclidean space, where r, g, b are the colour channel brightness (intensity) values of pixel p . Traditionally, edge detection algorithms require a greyscale image, which is a 2D image in which each pixel intensity is visually a shade of grey ranging from pure white to pure black. After choosing the pixels in a colour channel, then any of the usual edge detection methods can be used on the single colour channel pixels. In this example, we use the edge detection method introduced by John Canny [24].

Here are some of the details.



Colour Subimage

In this cycle image segment, the combined RGB channel pixels are shown.



BW Subimage Edges

In this cycle BW image segment, white edge pixels on a binary subimage are shown. ■

The steps to follow in edge pixel detection in each of the colour channels are given in Algorithm 7. Notice the parallel between the conventional approach to pixel edge detection and colour channel edge detection in Algorithm 7. In both cases, edge pixels (either in white or in colour) are embossed on a black image. Sample strong edge pixels for the red channel of a cycle image are shown in Fig. 2.18.

Example 2.17 Figure 2.19 shows the result of finding the strong edge pixels in the green channel of a colour image using MScript A.18 in Appendix A.2.5. The story starts by selecting all of the pixels in a colour image. Traditionally, edge detection algorithms require a greyscale image. The pixels in a single channel of a colour image have the appearance of a typical 2D greyscale image, except that pixel intensities are pixel colour brightness values in a single channel. After choosing the pixels in a colour channel, then any of the usual edge detection methods can be used on the single colour channel pixels. Here again, we use Canny's edge detection method.

```

>> img = imread('carCycle.jpg'); % select RGB image
>> gR = img(:, :, 1); % select red channel pixels
>> imgR = edge(gR, 'canny'); % select red channel pixels

```

Here are some of the details.

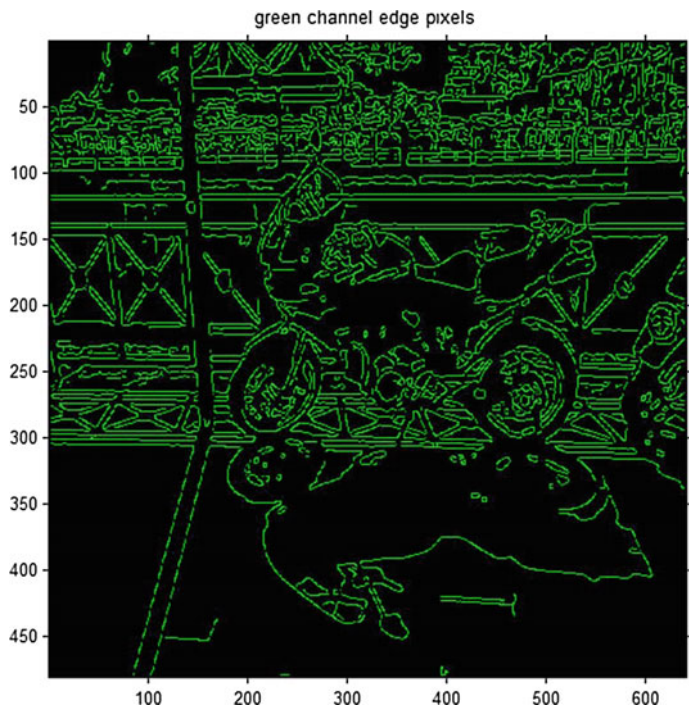


Fig. 2.18 Red channel cycle edges



Colour Channel Subimage

In this cycle image segment, only the green channel pixels are shown.

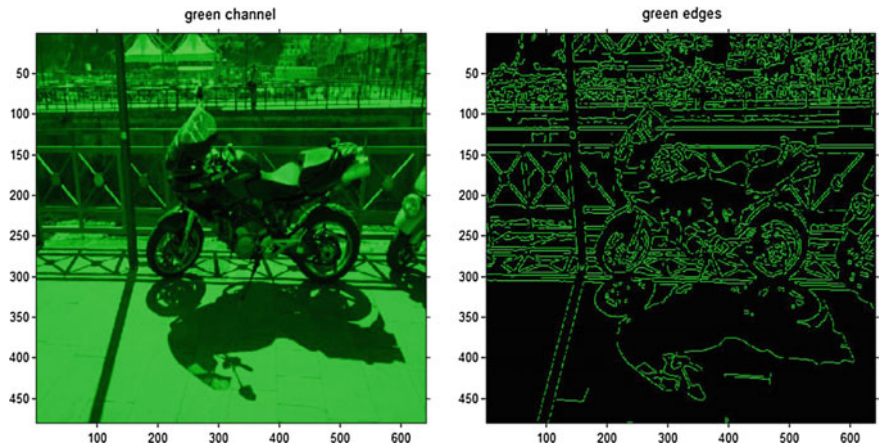


Fig. 2.19 Green channel edges



Colour Channel Subimage Edges

In this cycle image segment, the green channel edge pixels on the wheel subimage are shown. ■

It is possible to combine colour channel edge pixels on a black image.

Example 2.18 Figure 2.20 shows the result of combining the red channel and the green channel edge pixels again using MScript A.18 in Appendix A.2.5. This is accomplished in a straightforward fashion by concatenating the separate images, namely, *imgR* (red channel edges), *imgG* (green channel edges) and *a* (entirely black image).

Here are some of the details.



Binary Edges Subimage

In this cycle image segment, only the green channel pixels are shown.

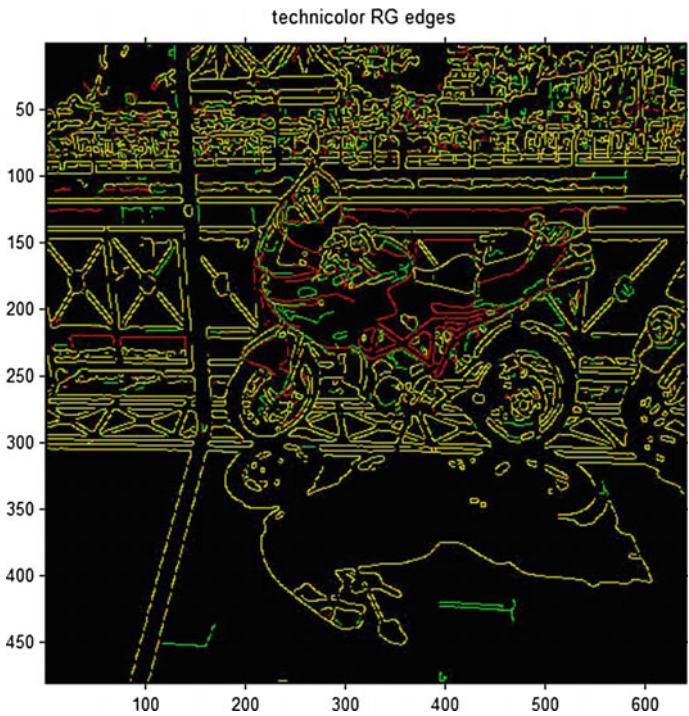
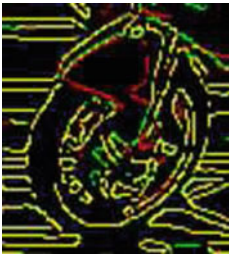


Fig. 2.20 Red green channel cycle edges



Colour Channel Subimage Edges

In this cycle image segment, the red channel and green channel edge pixels on the wheel subimage are shown. Notice that many yellow edges are included in the RG edges. The yellow edge pixels are at the higher (brighter) ends of the Canny hysteresis intervals used to identify strong edge pixels. An entirely different situation will arise, if we consider either RB or GB edge pixels (see Problem 2.19) (Fig. 2.21). ■

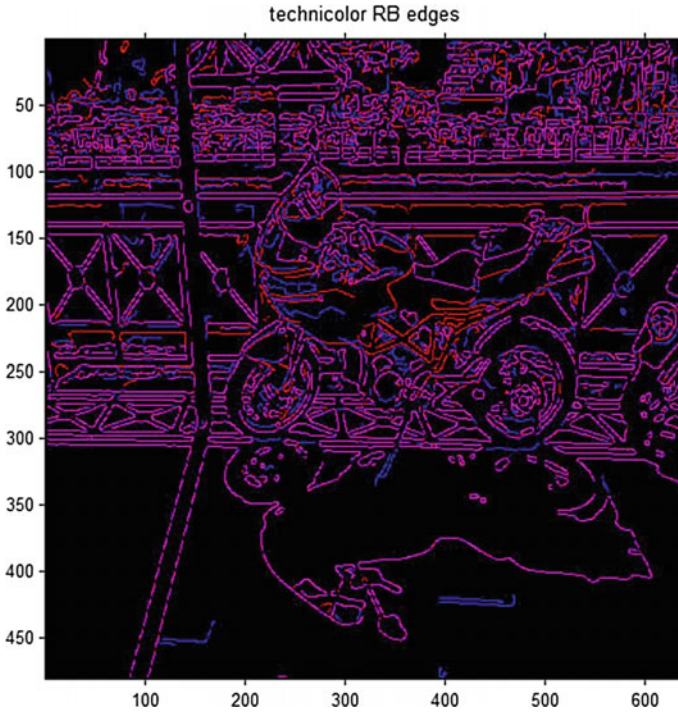


Fig. 2.21 Red blue channel cycle edges

Problem 2.19 Combined Color Channel Edge Pixels.

Extending the approach to combining colour edge pixels in Example 2.18, do the following:

- 1° 🚲 Display a combination of the red channel and blue channel edges on a black image. **Hint:** See how this is done in MScript A.18 in Appendix A.2.5.
- 2° 🚲 Display a combination of the green channel and blue channel edges on a black image.
- 3° ☕ Display a combination of the red, green, and blue channel edges on a black image. ■

Problem 2.20 Offline Video Frame Colour Channel Edges.

Use the approach to changing image channel intensities in MScript A.18 in Appendix A.2.5 as a template for offline video processing, do the following.

- 1° ☕ Using Matlab script A.8 in Appendix A.1.5 as a template for offline video processing, display the red channel edges in each video frame image. **Hint:** Replace the lines of Voronoi tessellation code with lines to code in MScript A.18 to handle and display the red channel edges in each video frame image.

- 2° Repeat Step 1 to handle and display the green channel edges in each video frame image.
- 3° Repeat Step 1 to handle and display the blue channel edges in each video frame image.
- 4° ☕ Using Matlab script A.8 in Appendix A.1.5 as a template for offline video processing, display the combined red and green channel edges in each video frame image. **Hint:** Replace the lines of Voronoï tessellation code with lines to code in MScript A.18 to handle and display the combined red and green channel edges in each video frame image.
- 5° Repeat Step 4 to handle and display the combined red and blue channel edges in each video frame image.
- 6° Repeat Step 4 to handle and display the combined green and blue channel edges in each video frame image. ■

Problem 2.21 Offline Video Frame Combined Colour Channel Edges.

Write a script to display offline the combined RGB channel edges in each video frame image. Do this for two different videos. ■

Problem 2.22 Real-Time Video Frame Colour Channel Edges.

Use the approach to changing image channel intensities in MScript A.18 in Appendix A.2.5 as a template for real-time video processing, do the following.

- 1° Using Matlab script A.9 in Appendix A.1.5 as a template for offline video processing, display the red channel edges in each video frame image. **Hint:** Replace the lines of Voronoï tessellation code with lines to code in Matlab script A.9 to handle and display the red channel edges in each video frame image in real-time.
- 2° Repeat Step 1 to handle and display the green channel edges in each video frame image.
- 3° Repeat Step 1 to handle and display the blue channel edges in each video frame image.
- 4° ☕ Using Matlab script A.8 in Appendix A.1.5 as a template for offline video processing, display the combined red and green channel edges in each video frame image. **Hint:** Replace the lines of Voronoï tessellation code with lines to code in MScript A.18 to handle and display the combined red and green channel edges in each video frame image in real-time.
- 5° Repeat Step 4 to handle and display the combined red and blue channel edges in each video frame image in real-time.
- 6° Repeat Step 4 to handle and display the combined green and blue channel edges in each video frame image in real-time. ■

Problem 2.23 Real-Time Video Frame Combined Colour Channel Edges.

Write a script to display in real-time the combined RGB channel edges in each video frame image. Do this for two different videos. ■

Algorithm 8: Log-Based Image Pixel Changes

Input : Read digital image img .
Output: $img \mapsto \log(img)$.

```

1  $gR \leftarrow img(:, :, 1)$ ;
2 /* Capture red channel pixel intensities. */;
3  $gG \leftarrow img(:, :, 2)$ ;
4 /* Capture green channel pixel intensities. */;
5  $gB \leftarrow img(:, :, 3)$ ;
6 /* Capture blue channel pixel intensities. */;
7  $\log(gR) \mapsto imgR$ ;
8  $\log(gG) \mapsto imgG$ ;
9  $\log(gB) \mapsto imgB$ ;
10 /* Map log of pixel intensities in each channel to a modified channel image. */;
11  $captureModifiedImage \leftarrow cat(3, imgR, imgG, imgB)$ ;
12 /* Capture modified channel intensities in a single image. */;
13 Display  $captureModifiedImage$ ;

```

2.6 Function-Based Image Pixel Value Changes

This section briefly introduces an approach to modifying image pixel values using various functions. We illustrate this approach using the natural log of pixel values over an selected colour channels. The steps to follow in modifying the each of the channel intensities resulting from the log of each colour channel pixel intensity are show in Algorithm 8.

Example 2.24 Figure 2.22 shows the result of a log-based modification of channel pixel intensities in a colour image using Algorithm MScript A.19 in Appendix A.2.6. Here are sample coding steps in the basic approach.

```

>>  $img = imread('carCycle.jpg')$ ; % select RGB image
>>  $gR = img(:, :, 1)$ ; % select red channel pixels
>>  $imgR = \log(double(gR))$ ; % find log of red channel edge pixel intensities
>>  $sf = 0.2$ ; % scaling factor
>>  $imgR = (sf) * \log(double(gR))$ ; % select lower edge pixel intensities

```

Here are some of the details.

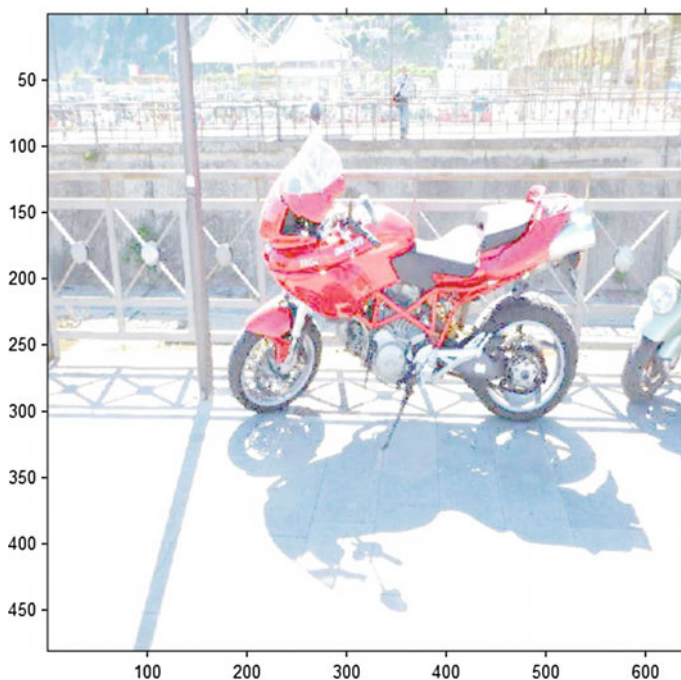


Fig. 2.22 Sample image after log-based pixel intensity changes



Colour Subimage

In this colour image segment, only the front wheel is shown.



Log-Based Colour Subimage

In this colour image segment, the combined log-modified channel intensities are shown. ■

Problem 2.25 Function-Based Colour Channel Intensity Modifications.

Select three colour images of your own choosing and do the following.

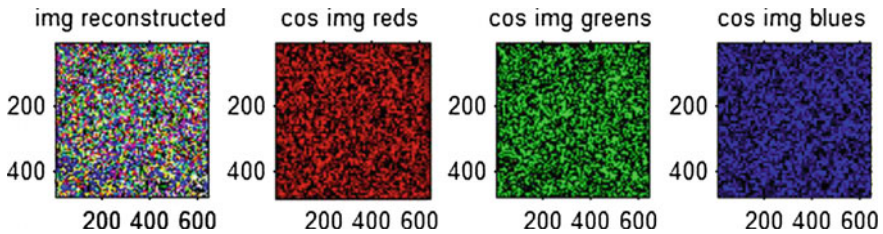


Fig. 2.23 Sample image after cosine-based pixel intensity changes

- 1° Compute the cosine of each colour channel intensity and produce four images like ones in Fig. 2.23. **Hint:** Modify MScript A.19 in Appendix A.2.6 to get the desired result.
- 2° Repeat the preceding step for two different choices of the scaling factor to adjust the brightness of the modified images. For example, 0.2 is the scaling factor in MScript A.19 and 1.8 is the scaling factor used to obtain the results in Fig. 2.23.

Problem 2.26 Colour Channel Edge Information Content.

Select three colour images of your own choosing and do the following.

- 1° Compute the information content of each colour channel edge pixel intensity and produce four images like ones in Fig. 2.23. **Hint:** Find the total number of pixels in each image. Assume that the edge pixel intensities in the digital image img are random. In addition, let the probability $p(img(x, y)) = \frac{1}{x*y}$ for each image intensity $img(x, y)$ for a pixel with coordinates (x, y) , $1 \leq x \leq m$, $1 \leq y \leq n$ in an $n \times m$ image.¹ Then, for each colour channel pixel intensity, compute the colour channel edge pixel **information content** $h(img(x, y, k))$, $k = 1, 2, 3$ of an edge pixel defined by

$$h(img(x, y, k)) := \log_2 \left(\frac{1}{p(img(x, y, k))} \right) \text{ (colour channel pixel info. content).}$$

And, for each colour pixel edge intensity, compute the colour edge pixel **information content** $h(img(x, y))$, $1 \leq x \leq m$, $1 \leq y \leq n$ of an edge pixel defined by

$$h(img(x, y)) := \log_2 \left(\frac{1}{p(img(x, y))} \right) \text{ (pixel information content).}$$

¹Many other ways to compute the probability of a pixel intensity $img(x, y)$ are possible. There is a restriction:

$$\sum_{i=1}^{n*m} p_i(img(r, c)) = 1, 1 \leq r \leq m, 1 \leq c \leq n.$$

- 2° Repeat the preceding steps for two different choices of the scaling factor to adjust the brightness of the modified images. ■

Problem 2.27 Colour Image Entropy and Its Modifications.

Select three colour images of your own choosing and do the following.

- 1° 🚲 Give a formula for the Shannon entropy of a $n \times m$ colour image *img*.
- 2° 🚲 Using the assumptions in Problem 2.26, write a Matlab or Mathematica script to compute and display the Shannon entropy of three colour images of your own choosing.
- 3° ☕ Modify the Matlab script in Step 2 to do the following with the three colour images of your own choosing.
 - 3(a) Change the color image pixel intensities so that the entropy of the image increases.
 - 3(b) Change the color image pixel intensities so that the entropy of the image decreases. ■

2.7 Logical Operations on Images

The logical operations are not, and, or, and xor (exclusive or). This section introduces the use of not, or, and xor (exclusive or) on image pixels. Later, it will be shown how the **and** operation can be combined with what is known as thresholding to separate the foreground from the background of images (see Sect. 2.8).

2.7.1 Complementing and Logical not of Pixel Intensities

For a greyscale image, the complement of the image makes dark areas lighter and bright areas darker. For a binary image *g*, *not(g)* changes background (black) values to white and foreground (white) values to black. The *not(g)* produces the same results as *imcomplement(g)*.

Example 2.28 Mscript A.20 in Appendix A.2.7 illustrates changes in a greyscale image in which the complement of each intensity is complemented and changes in a binary image in which the logical **not** of each pixel intensity is computed. Figure 2.24 shows two modifications every intensity in a greyscale image:

- 1° Complement of each greyscale pixel intensity. Notice how the photographer's coat is now mostly (not entirely) white and dull gray background areas become very dark.
- 2° Addition of the maximum intensity to each greyscale pixel intensity. The result is surprising, since it demonstrated the presence of blurred segments in the original greyscale image.

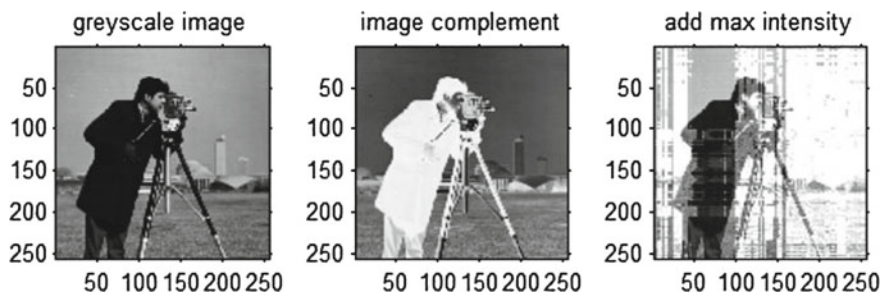


Fig. 2.24 Sample complement and increased greyscale pixel intensities

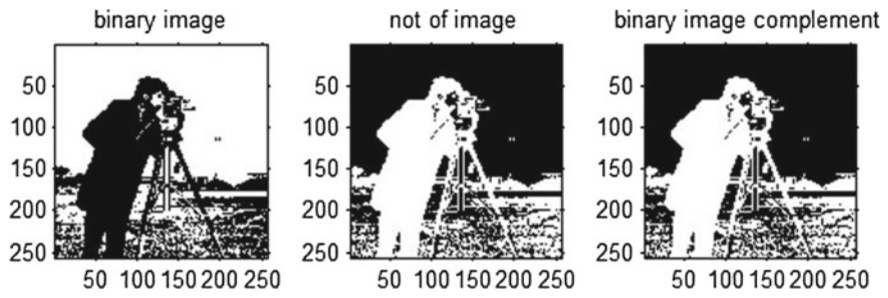


Fig. 2.25 Sample complement and logically negated binary pixel intensities

Figure 2.24 shows two modifications every intensity in a binary image:

- 3° Logical **not** of each greyscale pixel intensity. Notice how all black areas of the binary become white and all white areas become black.
- 4° Complement of each binary pixel intensity. This produces the same result as the complement of the binary image (Fig. 2.25). ■

Table 2.1 XOR

x	y	xor(x,y)
0	0	0
0	1	1
1	0	1
1	1	0

2.7.2 Xor Operation on Pairs of Binary Images

To see what the xor operation does, consider Table 2.1, where x , y are pixel intensities in a binary image. Table 2.1 is modelled after an exclusive or truth table. In Matlab,



2.26.1: Robots at start

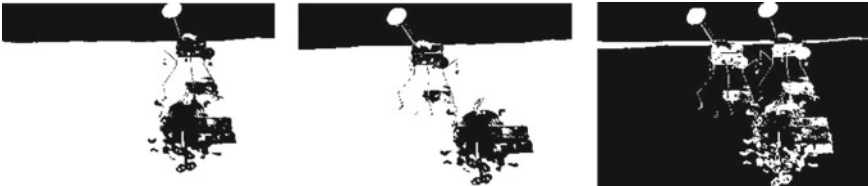
2.26.2: Robots competing

Fig. 2.26 Sample colour images using `robots.m`

the exclusive or operation produces the following sample result on a pair of binary images. To see what happens, consider the following pair of colour images.

```
% constructing new images from old images using xor
% idea from Solomon and Breckon, 2011
clc, close all, clear all

g = imread('race1.jpg'); h = imread('race2.jpg'); % read images
gbw = im2bw(g); hbw = im2bw(h); % convert to binary
check = xor(gbw,hbw); % What's happening?
subplot(1,3,1), imshow(gbw); % display g
subplot(1,3,2), imshow(hbw); % display h
subplot(1,3,3), imshow(check); % display xor(gbw,hbw)
```

Listing 2.1 Matlab code `cars.m` to produce Fig. 2.26.**Fig. 2.27** Sample xor images `robots.m`

Next, a pair of .png colour images in Fig. 2.26 are converted to binary images (every pixel value is either 1 (white) or 0 (black) after applying the `im2bw` function to each image. Then the `xor` function is applied (see Listing 2.2) to the pair of binary images to obtain the result shown in Fig. 2.27.

```
% constructing new images from old images
close all
clear all

%g = imread('birds1.jpg'); h = imread('birds2.jpg'); % What's happening?
g = imread('race1.jpg'); h = imread('race2.jpg'); % read png images
gbw = im2bw(g,0.3); hbw = im2bw(h,0.3); % convert to binary
check = xor(gbw,hbw); % xor binary
intensities
```

```

figure,
subplot(1,3,1), imshow(gbw);           % display gbw
subplot(1,3,2), imshow(hbw);           % display hbw
subplot(1,3,3), imshow(check);         % display xor(gbw,hbw)

```

Listing 2.2 Matlab code to produce Fig. 2.27.

For the sake of completeness, the same experiment is performed on a pair of .jpg colour images showing two different Thai grocery store displays. The interesting thing here is seeing how the xor operation on the displays reveals movements of similar items (bottles) from one display to the other (Fig. 2.28).



Fig. 2.28 Sample Thai Shelf images

```

% constructing new images from old images
clc, clear all, close all           % housekeeping
g = imread('P9.jpg'); h = imread('P7.jpg'); % read jpg images
%
gbw = im2bw(g); hbw = im2bw(h);     % convert to binary
check = xor(gbw,hbw);               % xor binary
intensities
subplot(1,3,1), imshow(gbw);         % display gbw
subplot(1,3,2), imshow(hbw);         % display hbw
subplot(1,3,3), imshow(check);       % display xor(gbw,hbw)

```

Listing 2.3 Matlab code xor2.m to produce Fig. 2.29.

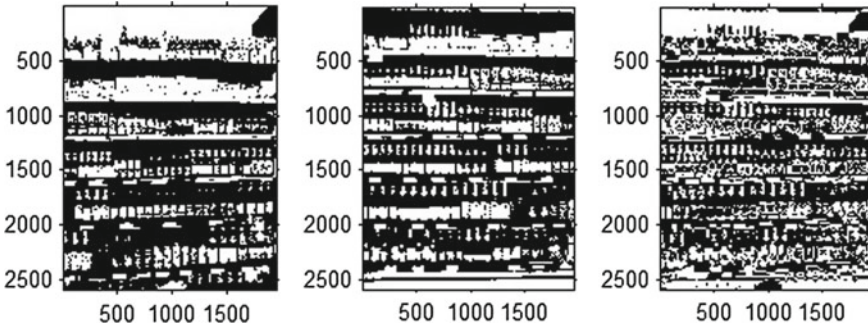


Fig. 2.29 Sample .png colour images

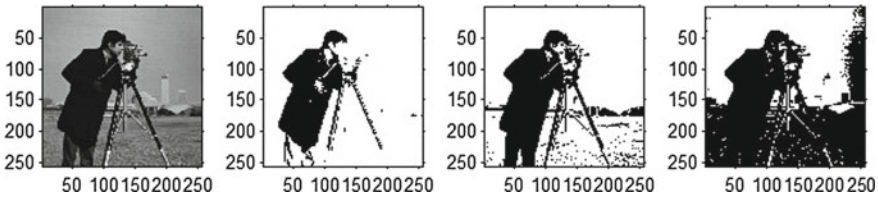


Fig. 2.30 Sample grayscale image thresholding

2.8 Separating Image Foreground From Background

Greyscale and colour images can be transformed into binary (black and white) images, where the pixels in the foreground of an image are black and pixels in the background of an image are white. The separation of image foreground from background is accomplished using a technique called **thresholding**. The thresholding method results in a binary image by changing each background pixel value to 0, if a pixel value is below a threshold, and to 1, if a foreground pixel value is greater than or equal to the threshold. Let $th \in (0, \infty]$ denote a threshold and let \mathfrak{I} denote a greyscale image. Then

$$\mathfrak{I}(x, y) = \begin{cases} 1, & \text{if } \mathfrak{I}(x, y) > th, \\ 0, & \text{otherwise.} \end{cases}$$

```
% Thresholding on greyscale image
clc, clear all, close all % housekeeping
g = imread('cameraman.tif'); % read greyscale image
h1 = im2bw(g, 0.1); % threshold = 0.1
h2 = im2bw(g, 0.4); % threshold = 0.5
h3 = im2bw(g, 0.6); % threshold = 0.5
subplot(1,4,1), imshow(g); % display greyscale image
subplot(1,4,2), imshow(h1); % display transformed image
subplot(1,4,3), imshow(h2); % display transformed image
subplot(1,4,4), imshow(h3); % display transformed image
```

Listing 2.4 Matlab script to produce Fig. 2.30 using `ex_greyth.m`.

Notice that $th = 0.5$ works best in separating the cameraman from the background (in fact, the background is no longer visible in Fig. 2.30 for $th = 0.5$). If there is interest in isolating the foreground of a greyscale image, it is necessary to experiment with different thresholds to obtain the best result. The code used to produce Fig. 2.30 is given in Listing 2.4.

Problem 2.29 Reversing Greyscale Pixel Separation Process.

Partially reverse the thresholding process for a greyscale. Wherever there is a white pixel in a thresholded image, change to white the pixel in the corresponding greyscale image. This reversal process will result in a greyscale where the foreground consists of pixels with varying intensities and the background of the greyscale image is entirely white. This reversal process will be important later, when feature extraction methods are used based on varying pixel intensities. ■

Separating the foreground from the background in colour images can either be done uniformly (treating all three colour channels alike) or finely by thresholding each colour channel individually. Sample results of the uniform separation approach are shown in Fig. 2.31 using the code Listing 2.5.

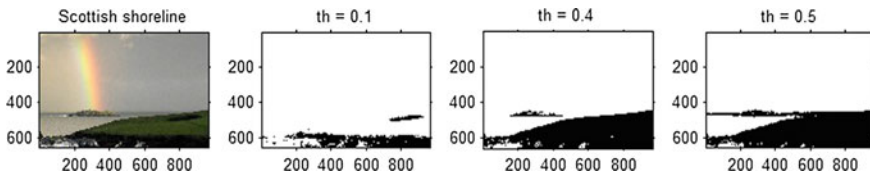


Fig. 2.31 Sample colour image thresholding

```
% Thresholding a colour image
% What's happening?
g = imread('rainbow.jpg'); % read colour image
% g = imread('penguins.jpg'); % read colour image
h1 = im2bw(g,0.1); % threshold = 0.1
h2 = im2bw(g,0.4); % threshold = 0.4
h3 = im2bw(g,0.5); % threshold = 0.5
subplot(1,4,1), imshow(g); title('Scottish shoreline');
subplot(1,4,2), imshow(h1); title('th = 0.1');
subplot(1,4,3), imshow(h2); title('th = 0.4');
subplot(1,4,4), imshow(h3); title('th = 0.5');
```

Listing 2.5 Matlab script to produce Fig. 2.31 using ex_2th.m.

Problem 2.30 Reversing Colour Pixel Separation Process.

Partially reverse the thresholding process for a colour image. Wherever there is a white pixel in a thresholded colour image, change to white the pixel in the corresponding colour image. This reversal process will result in a colour where the foreground consists of pixels with varying intensities for each colour channel for each pixel and the background of the colour image is entirely white. This reversal process will be important later, when feature extraction methods are used based on

varying pixel colour intensities. Common applications of this reversal process are in signature forgery detection and camouflage detection in paintings and in satellite images. ■



Fig. 2.32 Sample colour image

2.9 Conjunction of Thresholded Colour Channels

Another useful technique in separating the foreground from the background in colour images stems from an application of the logical **and** operation. The basic idea is to threshold the pixel intensities in each colour channel and then experiment with the conjunction of the resulting colour changes, either in pairs or the conjunction of all three thresholded colour channels. Let \mathfrak{I} be a colour image, r, g, b colour channels in \mathfrak{I} , and let rth, gth, bth denote thresholds on the red, green, blue colour channels, respectively. Then

- » $rbw = im2bw(\mathfrak{I}(:, :, r), rth)$; thresholded red channel,
- » $gbw = im2bw(\mathfrak{I}(:, :, g), gth)$; thresholded green channel,
- » $bbw = im2bw(\mathfrak{I}(:, :, b), bth)$; thresholded blue channel.

Then using the logical *and* operation, compute

$\gg \text{arg} = \text{and}(\text{rbw}, \text{gbw});$ conjunction of r,g channels,
 $\gg \text{arb} = \text{and}(\text{gbw}, \text{bbw});$ conjunction of g,b channels,
 $\gg \text{agb} = \text{and}(\text{rbw}, \text{bbw});$ conjunction of r,b channels,
 $\gg \text{agb} = \text{and}(\text{and}(\text{rbw}, \text{gbw}), \text{bbw});$ conjunction of r,g,b channels.

The colour image in Fig. 2.32 is an example of macrophotography, showing a closeup of grasshoppers. **Macrophotography** is closeup photography. A **macro lens** is capable of reproduction ratios greater than 1:1. The onscreen reproduction of a 1:1 macroimage results in a photograph greater than a lifesize image. Reproduction ratios much greater than 1:1 is called **photomicroscopy**, usually accomplished with a stereo zoom digital microscope. An application of the conjunction form of thresholding on the macrophotograph of grasshoppers is shown in Fig. 2.33 using the sample code in Listing 2.6. Notice the best separation of the foreground from background is achieved with a conjunction of the thresholded red and blue channels. This is not always the case (see Problem 2.31).

$\gg \text{agb} = \text{and}(\text{rbw}, \text{bbw});$ conjunction of r,b channels.

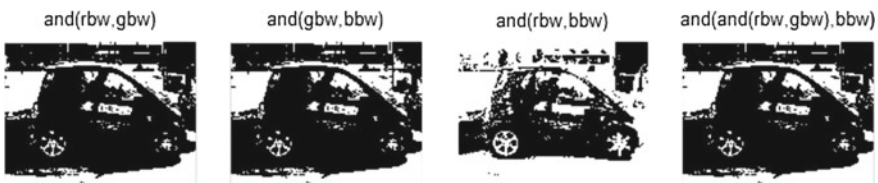


Fig. 2.33 Sample colour image thresholding with conjunction

```

% Thresholding colour channels
clc, clear all, close all      % housekeeping
g = imread('carPoste.jpg');    % read colour image
rth = 0.2989; gth = 0.587; bth = 0.114; % NTSC weights
r = g(:, :, 1); gr = g(:, :, 2); b = g(:, :, 3); % channels
rbw = im2bw(r, rth);          % threshold r
gbw = im2bw(gr, gth);         % threshold g
bbw = im2bw(b, bth);          % threshold b
o1 = and(rbw, gbw); o2 = and(gbw, bbw); o3 = and(rbw, bbw);
o4 = and(and(rbw, gbw), bbw);
subplot(1,4,1), imshow(o1), title('and (rbw, gbw) ');
subplot(1,4,2), imshow(o2), title('and (gbw, bbw) ');
subplot(1,4,3), imshow(o3), title('and (rbw, bbw) ');
subplot(1,4,4), imshow(o4), title('and (and (rbw, gbw) , bbw) ');
  
```

Listing 2.6 Matlab script to produce Fig. 2.33 using ex_2th2.m.

Problem 2.31 Threshold and Conjunction Separation Process.

Use a combination of thresholding and conjunction on the colour channels for several different colour images, starting with **rainbow.jpg** (Scottish rainbow) and **seq4a.jpg** (hand). Do the following.

- (and.1) Vary the weights for the thresholded rgb channels,
- (and.2) Point out which conjunction of thresholded channels gives the best results. The result will be best when there are more details in the foreground.
- (and.3) For a particular colour image, explain why a particular conjunction of colour channels works best.
- (and.4) Besides the **rainbow.jpg** (Scottish rainbow) and **seq4a.jpg** (hand) images, find a third colour image (your choice), where the conjunction of all thresholded colour channels works best. ■

Problem 2.32 Reversing Threshold and Conjunction Separation Process.

Partially reverse the thresholding-conjunction process for a colour image. Wherever there is a white pixel in a binary image resulting from a conjunction of a combination of thresholded colour channels, change to white the pixel in the corresponding colour image. This reversal process will result in a colour where the foreground consists of pixels with varying intensities for each colour channel for each foreground pixel and the background of the colour image will be entirely white. ■

The reversal process from the solutions of Problems 2.30 and 2.32 will also be important later, when feature extraction methods are used based on varying pixel colour intensities. Common applications of this reversal process are in signature forgery detection and camouflage detection in paintings and in satellite images.

2.10 Improving Contrast in an Image

Image contrast can be improved by altering the dynamic range of an image. The **dynamic range** of an image equals the difference between the smallest and largest image pixel values. Transforms can be defined by altering the relation between the

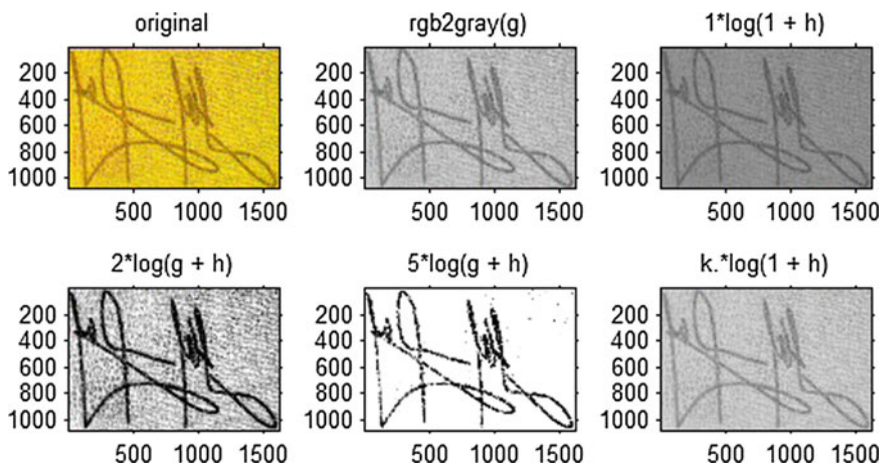


Fig. 2.34 Dynamic range compression with `eg_log1.m`

dynamic range and the greyscale (colour) image pixel values. For example, an image dynamic range can be altered by replacing each pixel value with its logarithm. Let \mathfrak{I} denote an image. Then alter the pixel value at (x, y) using

$$\mathfrak{I}(x, y) = k \log_e(1 + (e^\sigma - 1)\mathfrak{I}(x, y)), \quad (2.1)$$

where (assuming 8 bit pixel values),

$$k = \frac{255}{\log_e(1 + \max(\mathfrak{I}))}.$$

To simplify the implementation of Eq. (2.1), use the following technique to alter all pixel values in \mathfrak{I} .

$$\gg \mathfrak{I} = k.*\log(1 + \text{im2double}(\mathfrak{I}))$$

Next observe that, since \mathfrak{I} is a matrix, **max**(\mathfrak{I}) returns a row vector containing the maximum pixel value from each column. To complete the implementation of k , use

$$\gg k = \text{mean}((255)./ \log(1 + \max(\mathfrak{I}))).$$

The results of a number of different experiments in modifying the dynamic range are shown in Fig. 2.34, using the code in Listing 2.7.

```
% Compressing dynamic range of an image
clc, clear all, close all % housekeeping
g = imread('sig.jpg'); % Read in image
subplot(2,3,1), imshow(g); title('original');
g = rgb2gray(g);
subplot(2,3,2), imshow(g); title('rgb2gray(g)');
g = im2double(g); % pixel values -> double
h = im2double(g); % pixel values -> double
k = (max(max(g)))./(log(1 + max(max(g))));
com1 = 1*log(1 + h); % 1st compression
com2 = 2*log(g + h); % 2nd compression
com3 = 5*log(g + h); % 3rd compression
com4 = k.*log(1 + h); % 4th compression
subplot(2,3,3), imshow(com1); title('1*log(1 + h)');
subplot(2,3,4), imshow(com2); title('2*log(g + h)');
subplot(2,3,5), imshow(com3); title('5*log(g + h)');
subplot(2,3,6), imshow(com4); title('k.*log(1 + h)');
```

Listing 2.7 Matlab script to produce Fig. 2.34 using `eg_log1.m`.

Notice that by increasing the value of the multiplier k , the overall brightness of the image increases.² The best result for the signature image is shown in the third image in row 2 of Fig. 2.34, where **5.*log(g + h)** is used on the image g . A less than satisfactory result is obtained using **k.*log(1 + im2double(g))**. The logarithmic transform in Eq. (2.1) induces a brightening of the foreground by spreading the foreground pixel values over a wider range and a compression of the background pixel range. The

²Many thanks to Patrik Dahlström for pointing out the corrections in `eg_log1.m`.

narrowing of the background pixel range provides a sharper contrast between the background and the foreground.

Problem 2.33 Let g denote either a greyscale or colour image. In Matlab, implement Eq. (2.1) using $(e^\sigma - 1)g(x, y)$ instead of `im2double(g)` and show sample images using several choices of σ . Use the cameraman image as well as the signature image to show the results for different values of σ . ■

Use the Matlab `whos` function to display the information about the current variables in the workspace, e.g., variables k and $com4$ in Listing 2.7. Matlab constructs the double data type in terms of the definition for double precision in IEEE Standard 754, i.e., double precision values require 64 bits (for Matlab, double is the default data type for numbers). The `im2double(g)` function converts pixel intensities in image g to type double.

2.11 Gamma Transform

An alternative to the logarithm approach in compressing the dynamic range of intensities in an image, is the gamma (*raise to a power*) transform. Let I denote a digital image, $I(x, y)$ a pixel located at (x, y) , $k \in \mathbb{N}$ (natural number $1, \dots, \infty$), and $\gamma \in \mathbb{R}^+$ (positive reals). Basically, each pixel value is raised to a power using

$$I(x, y) = k(I(x, y))^\gamma.$$

The constant k provides a means of scaling the transformed pixel values. Here are rules-of-thumb for the choice of γ .

- (rule.1) $\gamma > 1$: Increase contrast between high-value pixel values at the expense of low-valued pixels.
- (rule.2) $\gamma < 1$: Decrease contrast between high-value pixel values at the expense of high-valued pixels.

```
% Gamma transform
clc, clear all, close all           % housekeeping
g = imread('P9.jpg');              % read image
% h = imread('P7.jpg');            % read image
g = im2double(g);
g1 = 2*(g.^(0.5)); g2 = 2*(g.^(1.5)); g3 = 2*(g.^(3.5));
subplot(1,4,1), imshow(g);         % display g
title('Thai shelves');
subplot(1,4,2), imshow(g1);        % gamma = 0.5
title('gamma = 0.5');
subplot(1,4,3), imshow(g2);        % gamma = 1.5
title('gamma = 1.5');
subplot(1,4,4), imshow(g3);        % gamma = 3.5
title('gamma = 3.5');
```

Listing 2.8 Matlab script to produce Fig. 2.35 using `myGamma.m`.

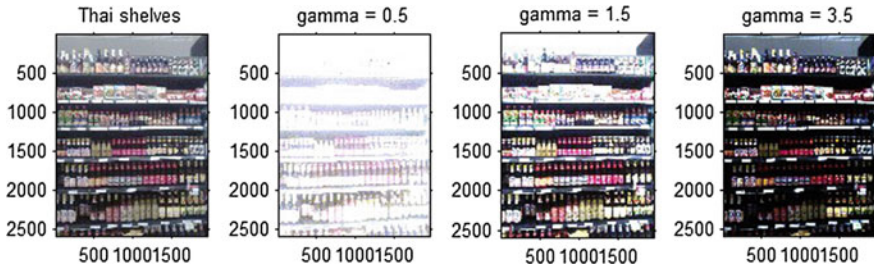


Fig. 2.35 Gamma transformation of a Thai colour image

2.12 Gamma Correction

There is a nonlinear relationship between input voltage and output intensity in monitor displays. This problem can be corrected by preprocessing image intensities with an inverse gamma transform (also called inverse power law transform) using

$$g_{out} = \left(g_{in}^{\frac{1}{\gamma}} \right)^{\gamma+k},$$

where g_{in} is the input image and g_{out} is the output image after gamma correction. Gamma correction can be carried out using the Matlab **imadjust** function as shown in `gamma_adjust.m` with sample results shown in Fig. 2.36. Unlike the results in Fig. 2.35 with the gamma transform, the best result in Fig. 2.36 is obtained with a lower γ value, namely, $\gamma = 1.5$ in Fig. 2.36 as opposed to $\gamma = 3.5$ in Fig. 2.35.

```
% Gamma correction transform
clc, clear all, close all % housekeeping
g = imread('P9.jpg'); % Thai shelves image
%g = imread('sig.jpg'); % Currency signature
g = im2double(g);
g1 = imadjust(g,[0 1],[0 1],0.5); % in/our range [0,1]
g2 = imadjust(g,[0 1],[0 1],1.5); % in/our range [0,1]
g3 = imadjust(g,[0 1],[0 1],3.8); % in/our range [0,1]
subplot(1,4,1), imshow(g); % display g
```

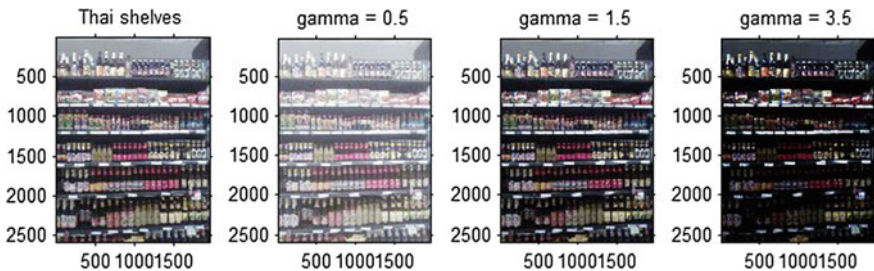


Fig. 2.36 Gamma correction of a Thai colour image

```
title('Thai shelves');  
subplot(1,4,2), imshow(g1);           % gamma = 0.5  
title('gamma = 0.5');  
subplot(1,4,3), imshow(g2);           % gamma = 1.5  
title('gamma = 1.5');  
subplot(1,4,4), imshow(g3);           % gamma = 3.5  
title('gamma = 3.5');
```

Listing 2.9 Matlab script to produce Fig. 2.36 using `gamma_adjust.m`.

Problem 2.34 🚲 Experiment with the currency signature in Fig. 2.34 using both the gamma transform and inverse gamma transform. Which value of γ gives the best result in each case? The best result will be the transformed image that has the clearest signature.

<http://www.springer.com/978-3-319-52481-8>

Foundations of Computer Vision
Computational Geometry, Visual Image Structures and
Object Shape Detection

Peters, J.F.

2017, XVII, 431 p. 354 illus., 301 illus. in color.,

Hardcover

ISBN: 978-3-319-52481-8