IMAGE PROCESSING PROGRAMMING EXERCISES

The following sections contain narrative descriptions of image processing programming exercises. They can be implemented using the PIKS API, PIKSTool Chain or MATLAB software. It is, of course, possible to implement the exercises with other APIs or tools that match the functionality of the PIKS API. In the following sections, suggested source images are indicated by the source image name in parentheses.

CHAPTER 1. PROGRAM GENERATION EXERCISES

E1.1 Develop a program that:

(a) Opens a program session.

(b) Reads an unsigned integer, 8-bit, monochrome source image (`brain`) from a file.

(c) Displays the parameters of the source image.

(d) Displays the source image.

(e) Creates a destination image, which is the complement of the source image.

(f) Displays the destination image.

(g) Closes the program session.

The PIKS API executable `example_complement_monochrome_ND` performs this exercise.

E1.2 Develop a program that:

(a) Creates, in application space, an unsigned integer, 8-bit, pixel array of a source ramp image whose amplitude increases from left-to-right from 0 to 255.

(b) Imports the source image for display.

(c) Creates a destination image by adding value 100 to each pixel

(d) Displays the destination image

The PIKS API executable `example_import_ramp` performs this exercise.

CHAPTER 2. IMAGE MANIPULATION EXERCISES

E2.1 Develop a program that passes a monochrome image (`peppers_mon`) through the log part of the monochrome vision model of Figure 2.4-4. Steps:

(a) Convert an unsigned integer, 8-bit, monochrome source image to floating point datatype.

(b) Scale the source image over the range 1.0 to 100.0.

(c) Compute the source image logarithmic lightness function of Eq. 5.3-4.

(d) Scale the log source image for display.

The PIKS API executable `example_monochrome_vision` performs this exercise.

E2.2 Develop a program that passes an unsigned integer, monochrome image (`jet_mon`) through a lookup table with a square root function. Steps:

(a) Read an unsigned integer, 8-bit, monochrome source image from a file.

(b) Display the source image.

(c) Allocate a 256 level lookup table.

(d) Load the lookup table with a square root function.

(e) Pass the source image through the lookup table.

(f) Display the destination image.

The PIKS API executable `example_lookup_monochrome_ND` performs this exercise.

E2.3 Develop a program that passes a signed integer, monochrome image (`brainscan_SD_double`) through a lookup table with a square root function. Steps:

(a) Read a signed integer, 16-bit, monochrome source image from a file.

(b) Linearly scale the source image over its maximum range and display it.

(c) Allocate a 65,536 level lookup table.

(d) Load the lookup table with a square root function over the source image maximum range.

(e) Pass the source image through the lookup table.

(f) Linearly scale the destination image over its maximum range and display it.

The PIKS API executable `example_lookup_monochrome_SD` performs this exercise.

CHAPTER 3. COLOR SPACE EXERCISES

E3.1 Develop a program that converts a linear *RGB* unsigned integer, 8-bit, color image (`dolls_linear`) to the *XYZ* color space and converts the *XYZ* color image back to the *RGB* color space. Steps:

(a) Display the *RGB* source linear color image.

(b) Display the *R*, *G* and *B* components as monochrome images.

(c) Convert the source image to unit range.

(d) Convert the *RGB* source image to *XYZ* color space.

(e) Display the *X*, *Y* and *Z* components as monochrome images.

(f) Convert the *XYZ* destination image to *RGB* color space.

(g) Display the *RGB* destination image.

The executable PIKS API `example_colour_conversion_RGB_XYZ` performs this exercise.[1]

E3.2 Develop a program that converts a linear *RGB* color image (`dolls_linear`) to the *L*a*b** color space and converts the *L*a*b** color image back to the *RGB* color space. Steps:

(a) Display the *RGB* source linear color image.

(b) Display the *R*, *G* and *B* components as monochrome images.

(c) Convert the source image to unit range.

(d) Convert the *RGB* source image to *L*a*b** color space.

(e) Display the *L**, *a** and *b** components as monochrome images.

(f) Convert the *L*a*b** destination image to *RGB* color space.

(g) Display the *RGB* destination image.

The PIKS API executable `example_colour_conversion_RGB_Lab` performs this exercise.

E3.3 Develop a program that converts a linear *RGB* color image (`dolls_linear`) to a gamma corrected *RGB* color image (`dolls_linear`) and converts the gamma color image back to the linear *RGB* color space. Steps:

(a) Display the *RGB* source linear color image.

(b) Display the *R*, *G* and *B* components as monochrome images.

(c) Convert the source image to unit range.

(d) Perform gamma correction on the linear *RGB* source image.

1.The PIKS API standard utilizes the British version of English language spelling, e.g. colour instead of color.

(e) Display the gamma corrected *RGB* destination image.

(f) Display the *R*, *G* and *B* gamma corrected components as monochrome images.

(g) Convert the gamma corrected destination image to linear *RGB* color space.

(h) Display the linear *RGB* destination image.

The PIKS API executable `example_colour_gamma_correction` performs this exercise.

E3.4 Develop a program that converts a gamma *RGB* color image (`dolls_gamma`) to the *YCbCr* color space and converts the *YCbCr* color image back to the gamma *RGB* color space. Steps:

(a) Display the *RGB* source gamma color image.

(b) Display the *R*, *G* and *B* components as monochrome images.

(c) Convert the source image to unit range.

(d) Convert the *RGB* source image to *YCbCr* color space.

(e) Display the *Y*, *Cb* and *Cr* components as monochrome images.

(f) Convert the *YCbCr* destination image to gamma *RGB* color space.

(g) Display the gamma *RGB* destination image.

The PIKS API executable `example_colour_conversion_RGB_YCbCr` performs this exercise.

E3.5 Develop a program that converts a gamma *RGB* color image (`dolls_gamma`) to the *IHS* color space and converts the *IHS* color image back to the gamma *RGB* color space. Steps:

(a) Display the *RGB* source gamma color image.

(b) Display the *R*, *G* and *B* components as monochrome images.

(c) Convert the source image to unit range.

(d) Convert the *RGB* source image to *IHS* color space.

(e) Display the *I*, *H* and *S* components as monochrome images.

(f) Convert the *IHS* destination image to gamma *RGB* color space.

(g) Display the gamma *RGB* destination image.

The PIK API executable `example_colour_conversion_RGB_IHS` performs this exercise.

CHAPTER 4. IMAGE MEASUREMENT EXERCISES

E4.1 Develop a program that computes the extrema of the *RGB* components of an unsigned integer, 8-bit, color image (`dolls_gamma`). Steps:

(a) Display the source color image.

(b) Compute extrema of the color image and print results for all bands.

The PIKS API executable `example_extrema_colour` performs this exercise.

E4.2 Develop a program that computes the mean and standard deviation of an unsigned integer, 8-bit, monochrome image (`peppers_mon`). Steps:

(a) Display the source monochrome image.

(b) Compute moments of the monochrome image and print results.

The PIKS API executable `example_moments_monochrome` performs this exercise.

E4.3 Develop a program that computes the first-order histogram of an unsigned integer, 8-bit, monochrome image (`brain`) with 16 amplitude bins. Steps:

(a) Display the source monochrome image.

(b) Compute the histogram of the source image.

(c) Display or printout the histogram.

The PIKS API executable `example_histogram_monochrome` performs this exercise.

CHAPTER 5. IMAGE QUANTIZATION EXERCISES

E5.1 Develop a program that re-quantizes an unsigned integer, 8-bit, monochrome image (`peppers_ramp_luminance`) linearly to three bits per pixel and reconstructs it to eight bits per pixel. Steps:

(a) Display the source image.

(b) Perform a right overflow shift by five bits on the source image[1].

(c) Perform a left overflow shift by five bits on the right bit-shifted source image.

(d) Scale the reconstruction levels to 3-bit values.

(e) Display the destination image.

The PIKS API executable `example_linear_quantizer` performs this exercise.

E5.2 Develop a program that quantizes an unsigned integer, 8-bit, monochrome image (`peppers_ramp_luminance`) according to the cube root lightness function of Eq. 6.3-4 and reconstructs it to eight bits per pixel. Steps:

(a) Display the source image.

(b) Scale the source image to unit range.

(c) Perform the cube root lightness transformation.

(d) Scale the lightness function image to 0 to 255.

(e) Perform a right overflow shift by five bits on the source image.

(f) Perform a left overflow shift by five bits on the right bit-shifted source image.

(g) Scale the reconstruction levels to 3-bit values.

(h) Scale the reconstruction image to the lightness function range.

(i) Perform the inverse lightness function.

(j) Scale the inverse lightness function to the display range.

(k) Display the destination image.

The PIKS API executable `example_lightness_quantizer` performs this exercise.

E5.3 Develop a program that re-quantizes an unsigned integer, 8-bit, color image (`candy`) linearly to three bits per pixel per component and reconstructs it to eight bits per pixel per component. Steps:

1.The *right* bit of an unsigned integer is its least significant bit. The *left* bit of an unsigned integer is its most significant bit.

(a) Display the source color image.

(b) Perform a right overflow shift by five bits of all components of the source image.

(c) Perform a left overflow shift by five bits on the right bit-shifted source image.

(d) Display the *R*, *G* and *B* components of the destination color image as monochrome images.

(e) Display the destination image.

The PIKS API executable `example_linear_quantizer_colour` performs this exercise.

CHAPTER 6. REGION-OF-INTEREST EXERCISES

E6.1 Develop a program that forms the complement of an unsigned integer, 8-bit, 512 x 512, monochrome, image (`brainscan`) under region-of-interest control.

Case 1: Full source and destination ROIs.

Case 2: Rectangular source ROI, upper left corner at (50, 100), lower right corner at (300, 350) and full destination ROI.

Case 3: Full source ROI and rectangular destination ROI, upper left corner at (150, 200), lower right corner at (400, 450).

Case 4: Rectangular source ROI, upper left corner at (50, 100), lower right corner at (300, 350) and rectangular destination ROI, upper left corner at (150, 200), lower right corner at (400, 450).

Steps:

(a) Display the source monochrome image.

(b) Create source and destination ROIs.

(c) Complement the source image into the destination image.

(d) Display the destination image.

(e) Create a constant destination image of value 150.

(f) Bind the source ROI to the source image.

(g) Complement the source image into the destination image.

(h) Display the destination image.

(i) Create a constant destination image of value 150.

(j) Bind the destination ROI to the destination image.

(k) Complement the source image into the destination image.

(l) Display the destination image.

(m) Create a constant destination image of value 150.

(n) Bind the source ROI to the source image and bind the destination ROI to the destination image.

(o) Complement the source image into the destination image.

(p) Display the destination image.

The PIKS API executable `example_complement_monochrome_roi` performs this exercise.

CHAPTER 7. CONVOLUTION EXERCISES

E7.1 Develop a program that convolves a test image with a 3×3 uniform impulse response array for three convolution boundary conditions. Steps:

(a) Create a 101×101 pixel, real datatype test image consisting of a 2×2 cluster of amplitude 1.0 pixels in the upper left corner and a single pixel of amplitude 1.0 in the image center. Set all other pixels to 0.0.

(b) Create a 3×3 uniform impulse response array.

(c) Convolve the source image with the impulse response array for the following three boundary conditions: enclosed array, zero exterior, reflected exterior.

(d) Print a 5×5 pixel image array about the upper left corner and image center for each boundary condition and explain the results.

The PIKS API executable `example_convolve_boundary` performs this exercise.

E7.2 Develop a program that convolves an unsigned integer, 8-bit, color image (`dolls_gamma`) with a 5×5 uniform impulse response array. Steps:

(a) Display the source color image.

(b) Fetch the impulse response array from a data object repository.

(c) Convolve the source image with the impulse response array.

(d) Display the destination image.

The PIKS API executable `example_repository_convolve_colour` performs this exercise.

CHAPTER 8. UNITARY AND WAVELET TRANSFORM EXERCISES

E8.1 Develop a program that generates the Fourier transform log magnitude ordered display of Figure 8.2-4*d* for a monochrome image (`smpte_girl_luma`). Steps:

(a) Display the source monochrome image.

(b) Scale the source image to unit amplitude.

(c) Perform a two-dimensional Fourier transform on the unit amplitude source image with the ordered display option.

(d) Scale the log magnitude according to Eq. 8.2-9 where $a = 1.0$ and $b = 100.0$.

(e) Display the Fourier transformed image.

The PIKS API executable `example_transform_fourier` performs this exercise.

E8.2 Develop a program that generates the Hartley transform log magnitude ordered display of Figure 8.3-2c for a monochrome image (`smpte_girl_luma`) by manipulation of the Fourier transform coefficients of the image. Steps:

(a) Display the source monochrome image.

(b) Scale the source image to unit amplitude.

(c) Perform a two-dimensional Fourier transform on the unit amplitude source image with the dc term at the origin option.

(d) Extract the Hartley components from the Fourier components.

(e) Scale the log magnitude according to Eq. 8.2-9 where $a = 1.0$ and $b = 100.0$.

(f) Display the Hartley transformed image.

The PIKS API executable `example_transform_hartley` performs this exercise.

E8.3 Develop a program that generates the Hadamard transform in 8×8 pixel blocks for a monochrome image (`smpte_girl_luma`). Steps:

(a) Display the source monochrome image.

(b) Scale the source image to unit amplitude.

(c) Perform a two-dimensional Hadamard transform in 8×8 pixel blocks on the unit amplitude source image.

(d) Display the Hadamard transformed image.

The PIKS API executable `example_transform_hadamard` performs this exercise.

E8.4 Develop a program that generates the Haar wavelet transform for a monochrome image (`smpte_girl_luma`) following Figure 8.6-4. Steps:

(a) Display the source monochrome image.

(b) Create the Haar low-pass filter impulse array as [0.707 0.707]

(c) Create the Haar high-pass filter impulse array as [-0.707 0.707]

(d) Convolve the rows of the source image with the high-pass filter and subs ample each row by 2.

(e) Convolve the rows of the source image with the low-pass filter and subs ample each row by 2.

(f) Convolve the columns of the upper path array with the high-pass filter and subsample each column by 2 to produce the array $W^D(j, m, n)$.

(g) Convolve the columns of the upper path array with the low-pass filter and subsample each column by 2 to produce the array $W^V(j, m, n)$.

(h) Convolve the columns of the lower path array with the high-pass filter and subsample each column by 2 to produce the array $W^H(j, m, n)$.

(i) Convolve the columns of the lower path array with the low-pass filter and subsample each column by 2 to produce the array $W^A(j, m, n)$.

(j) Compose the four output arrays into a destination image.

(k) Display the scaled magnitude of the destination image.

The PIKS API executable `example_haar_wavelet` performs this exercise.

CHAPTER 9. LINEAR PROCESSING EXERCISES

E9.1 Develop a program that performs fast Fourier transform convolution following the steps of Section 9.3. Execute this program using an 11×11 uniform impulse response array on an unsigned integer, 8-bit, 512×512 monochrome image (`candy_512_luma`) without zero padding. Steps:

(a) Display the source monochrome image.

(b) Scale the source image to unit range.

(c) Perform a two-dimensional Fourier transform of the source image.

(d) Display the clipped magnitude of the source Fourier transform.

(e) Create an 11×11 uniform impulse response array.

(f) Convert the impulse response array to an image and embed it in a 512×512 zero background image.

(g) Perform a two-dimensional Fourier transform of the embedded impulse image.

(h) Display the clipped magnitude of the embedded impulse Fourier transform.

(i) Multiply the source and embedded impulse Fourier transforms.

(j) Perform a two-dimensional inverse Fourier transform of the product image.

(k) Display the destination image.

(l) Printout the erroneous pixels along a mid image row.

The PIKS API executable `example_fourier_filtering` performs this exercise.

E9.2 Develop a program that performs Fourier transform filtering of an unsigned integer, 8-bit, 512 × 512 monochrome image (`peppers_mon`) with a Gaussian low-pass filter. Steps:

(a) Display the source monochrome image.

(b) Create a Gaussian low-pass filter transfer function or fetch it from a repository.

(c) Perform Fourier transform linear filtering on the source image.

(d) Display the filtered destination image.

The PIKS API executable `example_gaussian_low_pass_filtering` performs this exercise.

CHAPTER 10. IMAGE ENHANCEMENT EXERCISES

E10.1 Develop a program that displays the *Q* component of a *YIQ* color image (`dolls_gamma`) over its full dynamic range. Steps:

(a) Display the source monochrome *RGB* image.

(b) Scale the *RGB* image to unit range and convert it to the *YIQ* space.

(c) Extract the *Q* component image.

(d) Compute the amplitude extrema.

(e) Display the Q component.

The PIKS API executable `example_Q_display` performs this exercise.

E10.2 Develop a program to histogram equalize an unsigned integer, 8-bit, monochrome image (`brain`). Steps:

(a) Display the source monochrome image.

(b) Compute and display the source image histogram.

(c) Compute the image cumulative histogram.

(d) Load the image cumulative histogram into a lookup table.

(e) Pass the image through the lookup table.

(f) Compute and display the destination image histogram

(g) Display the enhanced destination image.

The PIKS API executable `example_histogram_equalization` performs this exercise.

E10.3 Develop a program to perform outlier noise cleaning of a noisy unsigned integer, 8-bit, monochrome image (`peppers_replacement_noise`) following the algorithm of Figure 10.3-9. Steps:

(a) Display the source monochrome image.

(b) Compute a 3×3 neighborhood average image by convolution with a uniform impulse array.

(c) Display the neighborhood image.

(d) Create a magnitude of the difference image between the source image and the neighborhood image.

(e) Create a Boolean mask image which is TRUE if the magnitude difference image is greater than a specified error tolerance, e.g. 15%.

(f) Convert the mask image to a ROI and use it to generate the outlier destination image.

(g) Display the destination image.

The PIKS API executable `example_outlier` performs this exercise.

E10.4 Develop a program that performs linear edge crispening of an unsigned integer, 8-bit, color image (`dolls_gamma`) by convolution. Steps:

(a) Display the source color image.

(b) Import the Mask 3 impulse response array defined by Eq.10.4-1*c*.

(c) Convert the source image to 16-bit or larger integer or real datatype.

(d) Convolve the color image with the impulse response array.

(e) Clip the convolved image over the dynamic range of the source image to avoid amplitude undershoot and overshoot.

(f) Display the clipped destination image.

The executable `example_edge_crispening` performs this exercise.

E10.5 Develop a program that performs 7×7 plus-shape median filtering of a noisy unsigned integer, 8-bit, monochrome image (`peppers_uniform_noise`. Steps:

(a) Display the source monochrome image.

(b) Create a 7×7 Boolean mask array.

(c) Perform median filtering.

(d) Display the destination image.

The PIKS API executable `example_filtering_median_plus7` performs this exercise.

E10.6 Develop a program that generates a pseudocolor display of a `ramp` image Steps:

(a) Display the source monochrome image.

(b) Create a pseudocolor lookup table with six maximum amplitude segments starting at blue and ending at magenta.

(c) Pass the source image through the LUT.

(d) Display the destination image.

The PIKS API executable `example_pseudocolor` performs this exercise.

E10.7 Develop a program that generates a false color display of an infrared image (`landsat_ir`) and a blue image (`landsat_blue`). Steps:

(a) Display the infrared band monochrome image.

(b) Display the blue band monochrome image.

(c) Copy the landsat infrared band to the red band of a false color image.

(d) Copy the landsat blue band to the blue band of the false color image.

(e) Create a hybrid image: 0.5[infrared + blue].

(f) Copy the landsat hybrid image to the green band of the false color image.

(g) Display the false color destination image.

The PIKS API executable `example_false_color` performs this exercise.

CHAPTER 11. IMAGE RESTORATION EXERCISES

E11.1 Develop a program that modifies an unsigned integer, 8-bit, monochrome image (`peppers_mon`) to produce an image with zero mean, additive, uniform noise with a signal-to-noise ratio of 10.0. The program should execute for arbitrary size source images. Steps:

(a) Display the source monochrome image.

(b) In application space, create a unit range noise image array using, for example, the C math.h function `rand`.

(c) Import the noise image array.

(d) Display the noise image array.

(e) Scale the noise image array to produce a noise image array with zero mean and a SNR of 10.0.

(f) Compute the mean and standard deviation of the noise image.

(g) Read an unsigned integer, 8-bit monochrome image source image file and normalize it to unit range.

(h) Add the noise image to the source image and clip to unit range.

(i) Display the noisy source image.

The PIKS API executable `example_additive_noise` performs this exercise.

E11.2 Develop a program that modifies an unsigned integer, 8-bit, monochrome image (`peppers_mon`) to produce an image with replacement impulse noise. The program should execute for arbitrary size source images. Steps:

(a) Display the source monochrome image.

(b) In application space, create a unit range noise image array using, for example, the C math.h function `rand`.

(c) Import the noise image array.

(d) Read a source image file and normalize to unit range.

(e) Replace each source image pixel with 0.0 if the noise pixel is less than 1.0%, and replace each source image pixel with 1.0 if the noise pixel is greater than 99%. The replacement operation can be implemented by image copying under ROI control.

(f) Display the noisy source image.

The PIKS API executable `example_replacement_noise` performs this exercise.

E11.3 Develop a program that computes a 512 × 512 Wiener filter transfer function for the blur impulse response array (Mask 3) of Eq. 10.3-2*c* and white noise with a SNR of 10.0 and applies the Wiener filter to a monochrome image (`peppers_blurred_noisy`). Steps:

(a) Create the impulse response array $H_D(x, y)$ or fetch it from a repository.

(b) Convert the impulse response array to an image and embed it in a 512 × 512 zero background array.

(c) Compute the two-dimensional Fourier transform of the embedded impulse response array to obtain $H_D(\omega_x, \omega_y)$.

(d) Form the Wiener filter transfer function according to Eq. 11.2-18:

$$H_R(\omega_x, \omega_y) = \frac{H_D^*(\omega_x, \omega_y)}{|H_D(\omega_x, \omega_y)|^2 + W_N(\omega_x, \omega_y)}$$

(e) Display the magnitude of the Wiener filter transfer function.

(f) Compute and display the Wiener filter destination image.

The PIKS API executable `example_wiener_filter` performs this exercise.

CHAPTER 12. GEOMETRICAL IMAGE MODIFICATION EXERCISES

E12.1 Develop a program that minifies an unsigned integer, 8-bit, monochrome image (`washington_ir`) by a factor of two and rotates the minified image by 45 degrees clockwise about its center using bilinear interpolation. Steps:

(a) Display the source monochrome image

(b) Minify the source image into the center of a zero valued work image of the same size as the source image using bilinear interpolation.

(c) Rotate the work image clockwise about its center into a destination image using bilinear interpolation.

(d) Display the destination image.

The PIKS API executable `example_minify_rotate` performs this exercise.

E12.2 Develop a program that performs shearing of the rows of an unsigned integer, 8-bit, monochrome image (`washington_ir`) using bilinear interpolation such that the last image row is shifted 10% of the row width and all other rows are shifted proportionally. Steps:

(a) Display the source monochrome image.

(b) Shear the source image into a destination image using bilinear interpolation.

(c) Display the destination image.

The PIKS API executable `example_shear` performs this exercise.

E12.3 Develop a program that performs clockwise rotation of an unsigned integer, 8-bit, monochrome image (`aerial`) by 45 degrees using the Catmull and Smith two-pass algorithm. Steps:

(a) Display the source monochrome image.

(b) Perform shearing of each row of the source image into a work image using bilinear interpolation.

(c) Perform shearing of each column of the work image into a destination image using bilinear interpolation.

(d) Display the destination image.

The PIKS API executable `example_rotate_two-pass` performs this exercise.

E12.4 Develop a program that magnifies an unsigned integer, 8-bit, monochrome image source image (`aerial`) by a factor of 2:1 using both nearest neighbor and bilinear interpolation. Compare the results. Steps:

(a) Display the source monochrome image.

(b) Magnify and display the source image by 2:1 into a work image using nearest neighbor interpolation.

(c) Magnify and display the source image by 2:1 into a work image using bilinear interpolation.

(d) Subtract the two magnified images and display the absolute value of the difference.

The PIKS API executable `example_magnify_interpolate` performs this exercise.

CHAPTER 13. MORPHOLOGICAL IMAGE PROCESSING EXERCISES

E13.1 Develop a program that reads a 64×64, Boolean test image (`boolean_test`) and dilates it by one and two iterations with a 3×3 structuring element. Steps:

(a) Read the source image and zoom it by a factor of 8:1.

(b) Create a 3×3 structuring element array.

(c) Dilate the source image with one iteration.

(d) Display the zoomed destination image.

(e) Dilate the source image with two iterations.

(f) Display the zoomed destination image.

The PIKS API executable `example_boolean_dilation` performs this exercise.

E13.2 Develop a program that reads a 64×64, Boolean test image (`boolean_test`) and erodes it by one and two iterations with a 3×3 structuring element. Steps:

(a) Read the source image and zoom it by a factor of 8:1.

(b) Create a 3×3 structuring element array.

(c) Erode the source image with one iteration.

(d) Display the zoomed destination image.

(e) Erode the source image with two iterations.

(f) Display the zoomed destination image.

The PIKS API executable `example_boolean_erosion` performs this exercise.

E13.3 Develop a program that performs gray scale dilation on an unsigned integer, 8-bit, monochrome image (`pcb3`) with a 5×5 zero-value structuring element and a 5×5 TRUE state mask. Steps:

(a) Display the source image.

(b) Create a 5×5 Boolean mask.

(c) Perform grey scale dilation on the source image.

(d) Display the destination image.

The PIKS API executable `example_dilation_grey_ND` performs this exercise.

E13.4 Develop a program that performs gray scale erosion on an unsigned integer, 8-bit, monochrome image (`pcb3`) with a 5×5 zero-value structuring element and a 5×5 TRUE state mask. Steps:

(a) Display the source image.

(b) Create a 5×5 Boolean mask.

(c) Perform grey scale erosion on the source image.

(d) Display the destination image.

The PIKS API executable `example_erosion_grey_ND` performs this exercise.

CHAPTER 14. EDGE DETECTION EXERCISES

E14.1 Develop a program that generates the Sobel edge gradient of a monochrome image (`peppers_mon`) according to Figure 14.2-1 using a square root sum of squares gradient combination. Steps:

(a) Display the source image.

(b) Generate the horizontal and vertical Sobel impulse response arrays or fetch them from a repository.

(c) Convolve the source image with the horizontal Sobel.

(d) Display the Sobel horizontal gradient.

(e) Convolve the source image with the vertical Sobel.

(f) Display the Sobel vertical gradient.

(g) Form the square root sum of squares of the gradients.

(h) Display the Sobel gradient.

The PIKS API executable `example_sobel_gradient` performs this exercise.

E14.2 Develop a program that generates the Laplacian of Gaussian gradient of a monochrome image (`peppers_mon`) for a 11×11 impulse response array and a standard deviation of 2.0. Steps:

(a) Display the source image.

(b) Generate the Laplacian of Gaussian impulse response array.

(c) Convolve the source image with the Laplacian of Gaussian impulse response array.

(d) Display the Laplacian of Gaussian gradient.

The PIKS API executable `example_LoG_gradient` performs this exercise.

CHAPTER 15. IMAGE FEATURE EXTRACTION EXERCISES

E15.1 Develop a program that generates the 7×7 moving window mean and standard deviation features of an unsigned integer, 8-bit, monochrome image (`washington_ir`). Steps:

(a) Display the source image.

(b) Scale the source image to unit range.

(c) Create a 7×7 uniform impulse response array.

(d) Compute the moving window mean with the uniform impulse response array.

(e) Display the moving window mean image.

(f) Compute the moving window standard deviation with the uniform impulse response array.

(g) Display the moving window standard deviation image.

The PIKS API executable `example_amplitude_features` performs this exercise.

E15.2 Develop a program that computes the mean, standard deviation, skewness, kurtosis, energy, and entropy first-order histogram features of an unsigned integer, 8-bit, monochrome image (`ellipse`). Steps:

(a) Display the source image.

(b) Compute the histogram of the source image.

(c) Export the histogram and compute the histogram features.

The PIKS API executable `example_histogram_features` performs this exercise.

E15.3 Develop a program that computes the nine Laws texture features of an unsigned integer, 8-bit, monochrome image (`wool`). Use a 7×7 moving window to compute the standard deviation. Steps:

(a) Display the source image.

(b) Generate the nine Laws impulse response arrays or fetch them from a repository.

(c) For each Laws array:

convolve the source image with the Laws array.

compute the moving window mean of the Laws convolution.

compute the moving window standard deviation of the Laws image.

display the Laws texture features.

The PIKS API executable `example_laws_features` performs this exercise.

CHAPTER 16. IMAGE SEGMENTATION EXERCISES

E16.1 Develop a program that thresholds a monochrome image (`parts`) and displays the thresholded image. Determine the threshold value that provides the best visual segmentation. Steps:

(a) Display the source image.

(b) Threshold the source image into a Boolean destination image.

(c) Display the destination image.

The PIKS API executable `example_threshold` performs this exercise.

E16.2 Develop a program that locates and tags the watershed segmentation local minima in a monochrome image (`artificial7`). Steps:

(a) Display the source image.

(b) Generate a 3 x 3 Boolean mask.

(c) Erode the source image into a work image with the Boolean mask.

(d) Compute the local minima of the work image.

(e) Display the local minima image.

The PIKS API executable `example_watershed` performs this exercise.

CHAPTER 17. SHAPE ANALYSIS EXERCISES

E17.1 Develop a program that computes the scaled second-order central moments of a monochrome image (`ellipse`). Steps:

(a) Display the source image.

(b) Normalize the source image to unit range.

(c) Export the source image and perform the computation in application space in double precision.

The PIKS API executable `example_spatial_moments` performs this exercise.

E17.2 Develop a program that computes the crack code of a binarized monochrome image (`ellipse`). Steps:

(a) Display the source image.

(b) Threshold the source image to binary range.

(c) Compute the crack code and enclosed perimeter of the binary format ellipse.

The PIKS API executable `example_ellipse_perimeter` performs this exercise.

CHAPTER 18. IMAGE DETECTION AND REGISTRATION EXERCISES

E18.1 Develop a program that performs normalized cross-correlation template matching of a monochrome source image (`L_source`) and a monochrome template image (`L_template`) using the convolution operator as a means of correlation array computation. Steps:

(a) Display the source image.

(b) Display the template image.

(c) Rotate the template image 180 degrees and convert it to an impulse response array.

(d) Convolve the source image with the impulse response array to form the numerator of the cross-correlation array.

(e) Display the numerator image.

(f) Square the source image and compute its moving window average energy by convolution with a rectangular impulse response array to form the denominator of the cross-correlation array.

(g) Display the denominator image.

(h) Form the cross-correlation array image.

(i) Display the cross-correlation array image.

(j) Threshold the cross-correlation array image.

(k) Display the thresholded cross-correlation array image.

Note, it is necessary to properly scale the source and template images to obtain valid results. The PIKS API executable `example_template` performs this exercise.

E18.2 Develop a program that executes the cross-correlation operator on the monochrome source image (`washington_ir1`) and a monochrome source image (`washington_ir2`) using the cross-correlation operator. Steps:

(a) Display the first source image.

(b) Display the second source image.

(c) Execute the cross-correlation operator.

(d) Convert the cross-correlation to a destination image.

(e) Display the destination image.

The PIKS API executable `example_cross_correlation` performs this exercise.

CHAPTER 19. POINT PROCESSING IMAGE COMPRESSION EXERCISES

E19.1 Develop a program that generates the Huffman code book for a set of eight symbols. See Appendix 4. Steps:

(a) Read a user generated set of symbol probabilities.

(b) Compute the Huffman code word corresponding to each symbol using the code tree algorithm of Appendix 4.

(c) Print the Huffman code table.

(d) Compute the code book average code length and entropy.

The PIKS API executable `example_huffman_code_table` performs this exercise for the example of Appendix 4.

E19.2 Develop a program that computes the entropy of a monochrome image (`lenna_mon`). Steps:

(a) Display the source image.

(b) Compute the histogram of the source image.

(c) Estimate the grey level probabilities for an eight-level re-quantization of the source image based upon its histogram.

(d) Display the eight-level re-quantized source image.

(e) Compute the entropy of the re-quantized source image.

The PIKS API executable `example_lenna_entropy` performs this exercise.

E19.3 Develop a program that emulates the DPCM image coding system of Figure 19.3-2 using an 8 level tapered quantizer implemented by a lookup table. Process a monochrome image (`lenna_mon`) with the emulated coding system. Steps:

(a) Display the source image.

(b) Generate the quantizer lookup table with decision levels of: 0, 8, 16 and 32 and quantization levels of: 4, 12, 24 and 48.

(c) Create the destination image with the emulator.

(d) Display the eight-level re-quantized source image.

The PIKS API executable `example_DPCM` performs this exercise.

CHAPTER 20. SPATIAL PROCESSING IMAGE COMPRESSION EXERCISES

E20.1 Develop a program that computes the discrete cosine transform of a monochrome image (`lenna_mon`) in 8×8 pixel blocks. Steps:

(a) Display the source image.

(b) Compute the 8×8 pixel DCT.

(c) Display the magnitude of the DCT.

The PIKS API executable `example_8x8_cosine_transform` performs this exercise.

E20.2 Develop a program that simulates the JPEG DCT monochrome image coding system of Figure 20.6-1 without symbol and entropy coding and decoding for a monochrome image (`lenna_mon`). Steps:

(a) Display the source image.

(b) Extract each 8×8 pixel block.

(c) Level shift each block.

(d) Perform a DCT of each block.

(e) Quantize each coefficieent block using the luminance quantization array of Figure 20.6-2.

(f) Perform quantization reconstruction on each quantized coefficient block.

(g) Perform an inverse DCT on each reconstructed coefficient block.

(h) Inverse level shift each reconstructed image block.

(i) Compose each reconstructed image block to a reconstructed image.

(j) Display the reconstructed image.

The PIKS API executable `example_8x8_DCT_quantization` performs this exercise.

E20.3 Develop a program that performs run length coding on a Boolean image (`L_array`.) Steps:

(a) Display the source image.

(b) Compress the source image into a work file using run length coding with a maximum run length of 16.

(c) Decompress the work file to create the reconstructed image,

(d) Display the reconstructed image.

(e) Compute the compression ratio from the source image pixel count and the work file bit count.

The PIKS API executable `example_run_length` performs this exercise.