1 Exercises Chapter 3

1.1 Exercise 1

1. Compute an affine matrix for: centre at the middle of the image (image size is 176x144), angle=30°, and scaling=1.

   Answer: From equation chapter 3:

   $$x' = x\,cos(\alpha) + y\,sin(\alpha) \tag{1}$$

   $$y' = -x\,sin(\alpha) + y\,cos(\alpha) \tag{2}$$

   For the given angle, $cos(30) = 0.866$ and $sin(30) = 0.500$. For rotation alone without scaling, with the rotation centre at (0,0), the matrix is:

   $$\begin{vmatrix} 0.866 & 0.500 & 0 \\ -0.500 & 0.866 & 0 \end{vmatrix} \tag{3}$$

   Now one needs to find the displacement of a given point, e.g., where the centre of the image should go to after the transformation. We find that after the rotation the new centre is at:

   $$x' = x\,0.866 + y\,0.500 = 112.208 \tag{4}$$

   $$y' = -x\,0.500 + y\,0.866 = 18.352 \tag{5}$$

   In order to translate the centre (x', y') to the centre of the original image:

   $$x - x' = 176/2 - 112.208 = -24.208 \tag{6}$$

   $$y - y' = 144/2 - 18.352 = 53.646 \tag{7}$$

   And therefore the transform matrix is:

   $$\begin{vmatrix} 0.866 & 0.500 & -24.210 \\ -0.500 & 0.866 & 53.646 \end{vmatrix} \tag{8}$$

2. Write a program using OpenCV that implements rotation, translation, scaling and skewing for a grey-scale image, based on equations from chapter 3.

3. Is there anything wrong with the resulting image? Why?

   Answer: there are "holes" in the resulting image due to rounding up of the position of the pixels. One can solve this problem by using an inverted approach (computing the source pixel (x, y) for every destination pixel (x', y')).

4. Fix the problem using an inverted approach.

5. Use OpenCV approach to achieve the same results. (e.g., use `warpAffine()` with a 2x3 matrix)

   Answer:

Listing 1: Case 1: rotation using *getRotationMatrix2D*()

```
#include "opencv2/opencv.hpp"
char* filename;
using namespace std;
using namespace cv;
Mat image, image2;

int main( int argc, char** argv )
{
    if (argc == 2) { filename=argv[1];}
    else exit(0);
    image = imread( filename, 1 );
    image2.create(image.size(),image.channels());
    Point2f center; center.x=image.cols/2; center.y=image.rows/2;
    double angle=30;
    double scale=1;
    Mat rotmatrix=getRotationMatrix2D(center, angle, scale);
    warpAffine(image, image2, rotmatrix, image.size());//,INTER_LINEAR);
    namedWindow( "Source Image", 0 );
    namedWindow( "Resulting Image", 0 );
    imshow( "Source Image", image );
    imshow( "Resulting Image", image2 );
    waitKey(0);
}
```

Or alternatively, compute the matrix manually:

Listing 2: Case 2: generic affine transform (change the matrix manually)

```
#include "opencv2/opencv.hpp"
char* filename;
using namespace std;
using namespace cv;
Mat image, image2;

int main( int argc, char** argv )
{
    if (argc == 2) { filename=argv[1];}  else exit(0);
    image = imread( filename, 1 );
    image2.create(image.size(),image.channels());
    Mat rotmatrix = (Mat_<double>(2,3) << 0,0,0,0,0,0);
    double scale=2.0;
//to modify the matrix directly:
```

```
15      rotmatrix.at<double>(0,0)=0.866;      rotmatrix.at<double>(0,1)=0.500;
            rotmatrix.at<double>(0,2)=-24.210;
16      rotmatrix.at<double>(1,0)=-0.500;     rotmatrix.at<double>(1,1)=0.866;
            rotmatrix.at<double>(1,2)=53.646;
17      cout << rotmatrix << endl;
18      warpAffine(image, image2, rotmatrix, image.size());//,INTER_LINEAR);
19      namedWindow( "Source Image", 1 );
20      namedWindow( "Resulting Image", 1 );
21      imshow( "Source Image", image );
22      imshow( "Resulting Image", image2 );
23      waitKey(0);
24  }
```

1.2 Exercise 2

1. Write a program that smooths the image using a linear filter. The kernel for the linear filter can be of different sizes (take a parameter for the kernel size).

   Answer:

Listing 3: Exercise 2

```
1  #include "stdio.h"
2  #include "cv.h"
3  #include "cxcore.h"
4  #include "highgui.h"
5
6  //Macros for greyscale
7  #define pixel(image,x,y) \
8  ((uchar *)(image->imageData + (y)*image->widthStep))[(x)*image->nChannels]
9
10 char* filename;
11 IplImage *image = 0, *image2 = 0, *convolved=0;
12 uchar* pixel, *p1,*p2,*p3,*p4,*p5,*p6,*p7,*p8, *p9;
13 //smoothing
14 //float kernel[9]={1,1,1,1,1,1,1,1,1};
15 float kernel[9]={(float)1/9,(float)1/9,(float)1/9,(float)1/9,(float)1/9,(
       float)1/9,(float)1/9,(float)1/9,(float)1/9};
16
17 int main( int argc, char** argv )
18 {
19      if (argc == 3) { filename=argv[1];}
20      else {printf("convolution input output\n");exit(0);}
21      if( (image = cvLoadImage( filename, 1)) == 0 )
22          return -1;
```

```
23      image2 = cvCreateImage(cvSize(image->width,image->height), IPL_DEPTH_8U,
            1);
24      convolved = cvCreateImage(cvSize(image->width,image->height),
            IPL_DEPTH_8U, 1);
25      cvCvtColor(image, image2, CV_BGR2GRAY);
26      //access pixels of the grey-scale image
27      for (int pos_y=0;pos_y<image2->height;pos_y++){
28        for (int pos_x=0;pos_x<image2->width;pos_x++){
29          if(pos_y>=1 && pos_y<(image2->height-1) && pos_x>=1 && pos_x<(image2
                ->width-1) ){
30            int pixelres;
31            pixelres= cvRound(
32              pixel(image2,pos_x-1,pos_y-1)*kernel[0]+
33              pixel(image2,pos_x,pos_y-1)*kernel[1]+
34              pixel(image2,pos_x+1,pos_y-1)*kernel[2]+
35              pixel(image2,pos_x-1,pos_y)*kernel[3]+
36              pixel(image2,pos_x,pos_y)*kernel[4]+
37              pixel(image2,pos_x+1,pos_y)*kernel[5]+
38              pixel(image2,pos_x-1,pos_y+1)*kernel[6]+
39              pixel(image2,pos_x,pos_y+1)*kernel[7]+
40              pixel(image2,pos_x+1,pos_y+1)*kernel[8]
41              );
42            pixel(convolved,pos_x,pos_y)=pixelres;
43            if (pixelres<0) pixel(convolved,pos_x,pos_y)=0;
44            if (pixelres>255) pixel(convolved,pos_x,pos_y)=255;
45          }
46        }
47      }
48      cvSaveImage (argv[2], convolved);
49      cvNamedWindow("original",2);
50      cvNamedWindow("convolved",2);
51      cvShowImage("original",image2);
52      cvShowImage("convolved",convolved);
53      cvWaitKey(0);
54      cvReleaseImage(&image);
55      cvReleaseImage(&image2);
56      return 0;
57  }
```

1.3 Exercise 3

1. Write a program for histogram equalisation.

2. Modify the program for histogram *specification*. Your program should take 6 parameters:

`equalise input.bmp output.bmp x y x' y'`

This should load the input.bmp image and produce the output.bmp file. The equalisation should be specified by the histogram from the rectangular area in the image defined by the points (x,y) and (x',y').

Answer:

Listing 4: Exercise 3-1 and 3-2

```
#include "cv.h"
#include "highgui.h"
#include "stdio.h"
uchar* pixel, *pixel2;
int main(int argc, char *argv[]) {
  int a=0,b=0,c=0,xinit=0,yinit=0,xfinis=0,yfinis=0;
  int hs[256];
  int T[256];
  float TT[256];
  int sumhs[256];
  for(a=0;a<=255;a++) {
    hs[a]=0;
    sumhs[a]=0;
    T[256]=0;
    TT[256]=0;
  }
  /* reading the x1,y1 and x2,y2 that defines the square to be equalized */
  if (argc==7){
    sscanf(argv[3],"%d",&xinit);
    sscanf(argv[4],"%d",&yinit);
    sscanf(argv[5],"%d",&xfinis);
    sscanf(argv[6],"%d",&yfinis);
  }
  else {
    printf("usage: equalize input.tiff output.tiff 1 2 3 4\n");
    return 0;
  }

  /* Opening the files (input.tiff and output.tiff) and loading the image*/
  IplImage *image, *imageout,*image2;
  image=cvLoadImage(argv[1],1);
  image2 = cvCreateImage(cvSize(image->width,image->height),IPL_DEPTH_8U,1);
  cvCvtColor(image, image2, CV_BGR2GRAY);
  printf("image depth %d nchannels %d\n",image2->depth, image2->nChannels);
  imageout = cvCreateImage(cvSize(image2->width,image2->height),IPL_DEPTH_8U
      ,1);

```

```
37    for(a=xinit;a<xfinis;a++) {
38      for(b=yinit;b<yfinis;b++) {
39        pixel=&((uchar*)(image2->imageData+image2->widthStep*b))[a];
40        hs[*pixel]++;
41      }
42    }
43
44    /* Sum of Hs.... */
45    sumhs[0]=hs[0];
46    for(a=1;a<=255;a++) {
47      sumhs[a]=sumhs[a-1]+hs[a];
48    }
49    /* Finding the transfer function, firstly as float, then converting to int
          ...*/
50    for(a=0;a<=255;a++) {
51      TT[a]=(((float)255/(((float)xfinis-(float)xinit)*((float)yfinis-(float)
            yinit)))*(float)sumhs[a]+(float)0.5);
52      T[a]=(int)TT[a];
53    }
54
55    /* Converting the whole picture following the transfer function rules....
          */
56    for(a=0;a<imageout->width;a++) {
57      for(b=0;b<imageout->height;b++) {
58        pixel=&((uchar*)(imageout->imageData+imageout->widthStep*b))[a];
59        pixel2=&((uchar*)(image2->imageData+image2->widthStep*b))[a];
60        *pixel=T[*pixel2];
61      }
62    }
63    cvSaveImage (argv[2], imageout);
64    cvNamedWindow("original",1);//declare window
65    cvShowImage("original",image2);//show window
66    cvNamedWindow("result",2);//declare window
67    cvShowImage("result",imageout);
68    cvWaitKey(0);
69  }
```