Exercise Guidelines for 30330 - Image Analysis on Microcomputers

Fall 2015

Authors:

Mathias Benn, Jonas B. Bjarnø, Troelz Denver and Alessandro S. Massaro
mb@space.dtu.dk, jbb@space.dtu.dk, td@space.dtu.dk, asm@space.dtu.dk

Maintained by:
David A. K. Pedersen,
dakp@space.dtu.dk

Using the image analysis laboratory

Hardware

Workstations

The workstations are IA-32 based computers running the Windows OS. The computers are equipped with 1 or 2 GB of RAM.

Cameras

The cameras are found at the workstations, and are to be used for the exercises 1-5 in this manual. The camera drivers are installed on the workstations, and the cameras are configurable through the small systray icon named "Camera Properties". We will get back to this dialogue later. The cameras available are:

- **Philips SPC900 NC CCD Web camera**. This camera features a VGA format CCD (640x480), a Sony ICX098, with square pixels and a Bayer color mask. The lenses are monofocal (max aperture of F2.2), with a focus knob.
- **Philips monochrome cameras**. A couple of these cameras are found in the corner of the lab, and are available for intermediate resolution capture of static scenes. A selection of lenses is available. Ask the tutor for help.
- **Olympus digital SLR camera**. For special purposes, this camera allows the capture of high-definition images, also in RAW (uncompressed) formats. Ask the tutor regarding loan of this camera for your project.

***** IMPORTANT NOTE: Do NOT bring any items from the image analysis lab out of the lab, without prior approval from your tutor. Failing to respect this may have consequences for your exam *****

Camera Interface to workstations

The cameras interface to the workstation through a USB 2.0 cable. Thus, all camera data (settings and image data) are transferred at a speed of maximum 480 MBits/s. The camera driver takes care of unpacking the data to images, available for use in applications.

Other hardware

In the lab, miscellaneous supporting gear can be found, including camera supports and other optical gear. If you need particular items for your exam projects, please ask your tutor.

Software

Prerequisites

In order to benefit most from these exercises, some previous experience with C and/or C++ programming is clearly an advantage. However, we also find that there is no better method of learning than that of "doing". So, don't despair if parts of the code presented here is not clear - get the help you need from us or your fellow students to understand it.

You may also find the web useful - there are plenty of good tutorials that can get you started programming in a day's time or so. Google is your friend...

Using MS Visual Studio C++

The environment used in the exercises, and optionally for the subsequent final projects, is Microsoft Visual C++. Both versions 9.0 (VC++ 2008) and 10 (VC++ 2010) are installed on the computers in the image analysis laboratory. Within the environment, development of a variety of applications can take place - both simple data processing (console) applications, and more fancy widget-based applications. The application type for this course will be a mixture of the two.

In order to benefit most from the exercises, it is recommended that you acquire some familiarity with the environment beforehand.

The Intel Open Computer Vision Library - OpenCV

Along with MS Visual Studio C++, the Intel OpenCV libraries are installed. Version 2.10 is installed on the workstations. The OpenCV library is documented by itself on the

workstations; the entry point being c:\Program Files\OpenCV\docs\index.htm. The library provides an easy interface for the student to:

- load image files from a variety of known image formats
- acquire images directly from camera devices with a Windows Media capable driver
- display images or image channels in windows
- drbaw primitive geometric shapes in image windows
- process images using a plethora of library functions
- save images and image information in a variety of known image formats

Danger! Warning! The OpenCV library is a very powerful tool, which may encourage you to attempt solving rather complex problems. Use it with great care, and - for your own sake - be sure to understand the principles of the underlying code before applying it to your project. The price for taking short cuts using pre-compiled functions that "does-it-all" may be that your understanding of image analysis does not improve much during the course. Which will - trust us - be reflected in the final report from you, which is what we use for evaluating your work performed during the course. Therefore, *we urge you to follow the exercise guidelines rather strictly, and not stray off when tempted to make your project more fancy than needed to solve the exercise*. We want you to focus on the image analysis - not the programming of it.

Setup a simple MS Visual Studio C++ project

1. Start up Microsoft Visual Studio C++. The application should start sort of "empty". If a wizard pops up, please close it and do as we say.
2. In the menu, click 'File->New->Project...'
3. In the resulting dialogue, choose 'Win32 console application'. Provide the project name and location in the fields below. Please put the project somewhere under the 'My Documents' folder of the account that you are using (note: a directory will be created for your application).
4. In the next screen, choose "Console application" and check "Empty project". Hit finish and the project is created.
5. In the "Workspace" field to the left, an overview of your project resides. In the bottom, you can choose from "Class View" or "Solution Explorer". Choose "Solution Explorer" to see a breakdown of the project tree in source-, header and resource files.
6. Right-click on "Source Files" and click on "Add->New Item…". Choose "C++ File (.cpp)", provide the name of the new file, and hit OK. An empty file will be appear in an editor window, and added to the project.
7. In the empty file, write the minimalistic C programme below. Save the file, and hit F7 to build the application.
8. Hit CTRL+F5 to run the application. A simple console window will appear with the text you chose for the printf statement.
9. Hit any key to get on with your life.

Simple hello world programme:

---------- SNIP ----------
```
#include <stdio.h>

int main(int argc, char *argv[])
{
    printf("Hello world\n");
    return 0;
}
```
---------- SNAP ----------

Specify directories for OpenCV include- and library files

The workstations are intended to be ready-to-go, a setup procedure is required to enable OpenCV in your own Visual Studio project. The procedure below should enable this.

- Set paths for the specific project: Project -> *project_name* properties:
    - Set the Configuration for *Debug* and leave the Platform at Win32
        - Configuration Properties -> VC++ Directories
            - Ad these paths to the Include directories
                - C:\OpenCV2.1\include\openVC
            - Ad these paths to the Library directories
                - C:\OpenCV2.1\Lib
            - Ad these paths to the source directories
                - C:\OpenCV2.1\src\cv
                - C:\OpenCV2.1\src\cvaux
                - C:\OpenCV2.1\src\cxcore
                - C:\OpenCV2.1\src\highgui
                - C:\OpenCV2.1\src\ml
        - Configuration Properties -> Linker -> Input
            - Additional Dependencies
                - cv210d.lib
                - cxcore210d.lib
                - highgui210d.lib

Note:
Make sure OpenCV's path (e.g. ";C:\OpenCV2.1\bin") is in your system PATH variable (Environment variable found in Control Panel->System).

Simple OpenCV hello world programme:

---------- SNIP ----------

```
// opencv_helloworld.cpp : Defines the entry point for the console
application.

#include <stdio.h>
#include "cv.h"
#include "cxcore.h"
#include "highgui.h"
#include <conio.h>

int main(int argc, char* argv[])
{
    const char* wName="Hello world!"; // window name
    cvNamedWindow(wName, 0); // create simple window

    CvCapture* capture = 0;

    double capProp=0;

    IplImage *frame, *frame_copy=0; // pointers to images

    capture = cvCaptureFromCAM(0); // initialize capture device

    if (capture)
    {
        for (;;)
        {
            if ( !cvGrabFrame( capture ))
                break;
            frame = cvRetrieveFrame( capture );

            if ( !frame )
                break;
            if ( !frame_copy )
            {
                printf("Frame settings:\n Width: %d\n Height: %d\n",
                        frame->width, frame->height);
                frame_copy = cvCreateImage (cvSize(frame->width,frame-
>height), IPL_DEPTH_8U, frame->nChannels);
                cvResizeWindow(wName,frame->width,frame->height);
            }
            if ( frame->origin == IPL_ORIGIN_TL )
                cvCopy ( frame, frame_copy, 0 );
            else
                cvFlip ( frame, frame_copy, 0);

            cvShowImage(wName, frame_copy);

            if (cvWaitKey(5)>0)
                break;

        }
```

```
    }

    cvReleaseImage( &frame_copy );
    cvDestroyWindow("Hello World");

    return 0;
}
```
---------- SNAP ----------

Adding files to the project

Source- or header files can be added to the project ad libitum, under "Project->Add to Project->Files..."

Loading an existing project

An existing project can be loaded into the MS Visual Studio using the "File->Open..." menu.

Other Software Installed on the Workstations:

A nice and fast image display program is IrfanView. IrfanView also has a capture interface that allows capturing e.g. 100 or 1000 subsequent images from the web cam. Use this program to grab VGA-format image sequences.
The GNU Image Manipulation Program (GIMP) is available for image editing at the workstations.

Exercises

Exercise 1: Image acquisition and display

The purpose of this exercise is to get acquainted with the working environment of the image analysis laboratory, and with the basic input/output primitives provided by the OpenCV library.

Preparations

You should go through the relevant sections of the previous chapter on using the Image Analysis laboratory before proceeding with the exercise.

Part 1: Loading images from files

The first part of the exercise is to load an image from a file located on the hard drive on the computer. The team should be able to:

- Create a simple console program that takes a file name as argument, and provides simple information about the file, such as file name, size, image format and, image size, and color format. An example image can be found at c:\Documents and Settings\30330\Exercises\Exercise1\ariane5_1b.jpg or at CampusNet.

Useful OpenCV data structures:

- `struct IplImage`

Useful OpenCV functions are:

- `cvCreateImage`
- `cvLoadImage`

Make sure to understand the IplImage structure fully, and especially, how image data is stored and accessed within the structure. At this point, we do not care much about the image storage file formats - be it JPEG, PNG, PGM, PPM, PNM, BMP, DIB, RAS or TIFF (all supported by OpenCV).

Questions

- What are the supported color formats of OpenCV images?
- How does the memory organization differ between interleaved and planar images?
- How would you access a single value of a color channel in a pixel, in either interleaved or planar images?
- How would you invert a single color channel of the image?

Part 2: Image display

In order to display the acquired image, you need to use the highgui library functions. They should already be available, if you successfully completed the preparation steps. Remember to add the line

```
#include <highgui.h>
```

to the include statements in the beginning of your program.

Useful OpenCV functions are:

- `cvNamedWindow`
- `cvMoveWindow`
- `cvResizeWindow`
- `cvShowImage`
- `cvWaitKey`

Questions:

- Who is really in control of window sizing?
- How would you display a color image in grayscale?
- Try to show graphically a histogram of pixel values. A histogram for an 8-bit color channel can be created as follows:

```
unsigned char *channel; // pre-allocated array holding image data
for the color channel or the grayscale image.
unsigned char value=0; // index value for the histogram (not really
needed)
int histogram[256]; // histogram array - remember to set to zero
initially
int width; // say, 320
int height; // say, 240

int k=256;
while (k-- > 0)
   histogram[k] = 0; // reset histogram entry

for (i=0;i<width*height;i++)
{
    value = channel[i];
    histogram[value] += 1;
}
```

Useful OpenCV functions for drawing lines on an image window are

- `cvLine`
- `cvRectangle (e.g., to set background color for drawing space)`

Note: You may want to use a separate window/image for drawing the histogram.

Part 3: Image acquisition from cameras

In order to acquire a new image from a camera, a capture device needs to be declared in your program. Also, you need to provide storage space for the images captured. The OpenCV library indexes the capture devices by the order they are registered within Windows. This means that if you have only one camera/frame grabber attached, the device will have an index of 0. If you have *N* devices attached, you should be able to capture from devices from 0 to *N*-1.

The OpenCV Hello World program given above can be used as a template in order to study how image acquisition works, as seen from the OpenCV environment. However, it is not possible to play with camera settings directly from within OpenCV. Rather, you can use the systray icon, or the program "Philips VideoLounge" to alter the camera settings. Oddly, it is not possible to alter camera settings such as gain- or shutter time directly using OpenCV function calls. You have to rely on some external control tool.

- Before attempting to process the images acquired, please be sure to take the following steps, using the systray camera control tool:
    1. In the "General" tab, disable all features, if any are enabled.
    2. In the "Picture" tab, disable "Full Automatic Control". A few additional options are now available.
        1. Disable "Auto exposure". The shutter speed / gain sliders are now available.
        2. Disable "Auto white balance". The Red/Blue sliders are now available for setting the color temperature of the images.
- Create a program that shows one 8-bit grayscale image at a time (so that you e.g. have to press a key before the camera grabs the next image), and displays a histogram of the image acquired. A useful OpenCV function is `cvConvertImage`.
- Study how moving the shutter speed slider alone alters the histogram. What is your interpretation of this? **Be sure to leave the contrast/brightness settings unaltered while you play around with shutter and gain settings.**
- With a low (fast) shutter speed setting, study how the gain slider alters the histogram. What happens to the noise in the image? (a live image may help you to see this).
- If you have time, change the program to 24-bit color image capture (RGB). Display each color channel in a separate window (declare three 8-bit grayscale images for each channel, and copy the image data to each channel image). Also, create windows to display histograms for each color channel. Observe how the white balance red- and blue sliders modify the image histograms. What is your interpretation of this?

Questions:

- Do you understand how shutter and gain differs from contrast and brightness?

Part 4: Image storage

The last part of the exercise serves to demonstrate how images can easily be stored from within OpenCV. The storage format is determined simply by the extension of the argument file name to the cvSaveImage function. The possible storage formats are:

Format name	File extension	Description
Windows Bitmaps	BMP, DIB	Don't use these unless you have to. They are raw bitmap formats, with headers that make them tedious to use directly in a programming context.
JPEG files	JPEG, JPG, JPE	Compressed format based on frequency analysis of image content. Lossy compression, so don't use if you want pixel-accurate saving of your source images.
Portable Network Graphics	PNG	Losslessly compressed format, with support for image layers, transparency, masks etc. To take full advantage of the format possibilities, the libpng library should be used.
Portable Bitmap Formats	PBM, PGM, PPM	1-, 8- or 24-bit raw image storage. Extremely portable, since the image header is ASCII text - almost impossible to misinterpret. For more info, look at http://netpbm.sourceforge.net/
Sun rasters	SR, RAS	Native UNIX image format, originally invented by Sun Microsystems. Available, but superfluous
Tagged Image Format Files	TIFF, TIF	General image container format - supports e.g. multiple images within one file, compressed or uncompressed. Also supports floating-point color depth of images.

Add image storage to your program, using either the OpenCV image storage function, cvSaveImage, or a storage method of your own. To save e.g. a portable graymap file (PGM), the following code can be used

```
unsigned char *image;
int width, height;

FILE *FP=fopen("file.pgm",'w');

fprintf(FP,"P5 %d %d 255\n",width, height);
fprintf(FP,"# This is a comment. You can have as many as you want, as
long as they begin with the hash character and ends with a newline\n");
fprintf(FP,"# Useful things to put in comments could be image time
```

```
stamp, scene description etc.\n");
fwrite(image,width,height,FP);
fclose(FP);
```

Exercise 2: Binary Images

The scope of this exercise is to get acquainted with binary images. For this, we will look at three important subtopics: (1) creation of binary images, (2) calculating image moments based on binary images, and (3) applying binary image analysis to a simple motion detection study. There are several ways of creating binary images - for this exercise, we will use only one, namely image thresholding.

Part 1: Image thresholding

In order to create a binary image to work with, perform the following steps:

- Create a program that loads and displays the image "C:\Documents and Settings\30330\My documents\Exercises\Exercise 2\pen.pgm". The file contains a 640x480 pixel 8-bit grayscale image, showing a pen on a table.
- Create and show a histogram of the image. What is the predominant feature in the histogram?
- Create a copy of the image and show the copy in a window next to the original image
- From the histogram, choose and apply a threshold *t* so that a pixel in the image copy will have the following value:
    - `255 if p >= t` (where *p* is the original pixel value)
    - `0 if p < t`
- Display the binary image next to the original. Try to adjust the threshold and see how this affects your binary image.

Part 2: Center of mass

Using the binary image, it is possible to compute the center of mass of objects, once identified.

- Calculate the center of mass of the binary object identified in part 1. Place a cross in the original image, at the center of mass coordinates. Does the location match the center of mass of the physical object? Hint: In a binary image, the center of mass *x*- and *y* coordinates can be computed simply by averaging the *x*- and *y* coordinates of the non-zero pixel values.
- Study the sensitivity of the center of mass location towards changes in the threshold parameter *t*. How would you optimize a scene for object identification and measurement using binary images?

Part 3: Image moments

Given the location of the center of mass, it is possible to compute the central moments of an object about an axis through this point. Recall that the $(p+q)^{th}$ central moment is defined by

$$\mu_{pq} = \sum_x \sum_y (x - \bar{x})^p (y - \bar{y})^q f(x, y)$$

- where *x'*,*y'* are the center of mass coordinates. Often, the reduced central moment $\mu_{pq}' = \mu_{pq}/\mu_{00}$ is used rather than the central moment itself.

The second central moment is the analogue to the moment of inertia. The second order moments are computed using (*p*,*q*) = (2,0), (*p*,*q*) = (0,2), or (*p*,*q*)=(1,1) in the formula above.

- Calculate the reduced central moments of second order (μ_{20}', μ_{02}' and μ_{11}') of the binary image from part 2, using the center of mass coordinates found in part 2.

It can be shown that the *principal axis* of the object forms an angle θ with the x-axis, given by

$$\tan 2\theta = \frac{2\mu_{11}'}{\mu_{20}' - \mu_{02}'}$$

- Calculate the angle θ. Draw a line through the center of mass of the object. What does the line represent? How sensitive is the line orientation to changes of the threshold parameter *t* ?

Part 4: Moment invariances

In recognition problems, invariant properties of images or part of images are often sought. If an invariant property can be identified for two images, the images are often similar (e.g., an object appears in both images, the only difference being a translation or rotation).

The raw image moments M_{pq} are generally not invariant under a coordinate transformation. The central moments, however, are invariant under translation (*x* -> *x*-x_0, *y*->*y*-y_0), but not under e.g. rotation (*x*,*y*) -> (*x* cos θ + *y* sin θ, -*x* sin θ + *y* cos θ) or scaling (*x*->*ax*,*y*->*by*).

However, there exists a set of moments of order 2 or higher ($p+q \geq 2$), that are invariant to both translation and scale, defined by

$$\eta_{ij} = \frac{\mu_{ij}}{\mu_{00}^{\left(1+\frac{i+j}{2}\right)}}$$

An additional set of moments that are additionally invariant under rotation is the *Hu set of invariant moments*, of which the first is given by

$$I_1 = \eta_{20} + \eta_{02}$$

- Using the web cam, capture two images of a simple object (e.g., a pen), where the only change between the images is that object is rotated by a certain angle between the two images. Calculate the first Hu moment for the two images, and judge whether the rotational invariance holds. If time allows, repeat the exercise with a translation AND a rotation of the pen between the two images.

Exercise 3: Image filters

Image filtering is often used as a pre-processing step, in order to reduce noise, or emphasize certain image features. The following exercises will demonstrate this.

A filter is generally implemented using a kernel of filter coefficients. The actual filtering is then a convolution of the image with the kernel. This is equivalent to a moving window operation. Depending on the kernel size, the filter works in a certain neighborhood of the pixel.

A general expression for filtering an image *p* into the result image *q* using the (2*n*+1)x(2m+1) kernel *c* is

$$p_{ij} = \sum_{k=i-n}^{i+n} \sum_{l=j-m}^{j+m} c_{kl} q_{kl}$$

This means that the ij'th pixel of the new image becomes a (linear) function of the neighboring (2*n*+1)x(2m+1) pixels.

When working with filters, be sure not to overwrite the original image with the results from the filtering process.

If the filter is to conserve overall image brightness, the following condition must be satisfied:

$$\sum_{k=-n}^{n} \sum_{l=-m}^{m} c_{kl} = 1$$

This condition can be used to normalize all filter kernels. Other criteria for normalization can of course be used as well.

Part 1: Lowpass filters

- Load or capture an example image that you will work with throughout this exercise. The image should be loaded/captured as 8-bit grayscale, at a resolution of 640x480 pixels.

- Make a low pass filter with the size of 3 x 3 (*n*=1, *m*=1) pixels. The value of the pixel (*i*, *j*) can for instance be determined with the average filter:
  $p_{ij} = \frac{1}{(2n+1)^2} \sum_{k=i-n}^{i+n} \sum_{l=j-n}^{j+n} q_{kl}$, $c_{kl} = 1/(2n+1)^2$
  where *q* is the original image and *p* is the lowpass filtered image.

- Compare the speed of your implementation of a low pass filter with the `cvFilter2D` function in OpenCV. Use the OpenCV functions `cvGetTickCount` and `cvGetTickFrequency`

Questions:

- How could the values in the rim of the image be calculated?
- What would be the coefficients of the identity filter that makes $p_{ij} = q_{ij}$?.

Part 2: Highpass filters

- Try to make a high pass filter instead (by changing filter coefficients). Hint: differentiation emphasizes high frequency content.

Part 3: Fractile filters (non-linear!)

A fractile filter assigns the center pixel a value depending on the percentile *p* chosen for the filter. E.g., for a window of pixels, the center pixel gets assigned the value that ensures that *p* % of the pixels in the window are less bright than the center pixel.

- On a piece of paper, draw a histogram of imaginary pixel values within a window, and explain the working of the fractile filter with your own words.

- Try to make a fractile filter of a variable size (e.g., the *median* = the 50% fractile). Explain why the fractile filter generally does a better job than the average filter when it comes to
    - (a) edge conservation
    - (b) salt- and pepper noise reduction.

Part 4: Laplace-Gaussian filter

The implementation of the Laplace-Gaussian filter attempts to mimic the double differential operator ∇^2. This operator has a number of interesting properties when applied to image data.

- Create a Laplace-Gaussian high pass filter using the 9x9 filter below:

$g_{i,j}$ =

0	0	1	2	2	2	1	0	0
0	1	5	10	12	10	5	1	0
1	5	15	19	16	19	15	5	1
2	10	19	-19	-64	-19	19	10	2
2	12	16	-64	-148	-64	16	12	2
2	10	19	-19	-64	-19	19	10	2
1	5	15	19	16	19	15	5	1
0	1	5	10	12	10	5	1	0
0	0	1	2	2	2	1	0	0

- Apply the filter to
  - The image you have worked with until now
  - The image stream from the web camera. Evaluate the speed of the filtering process vs. the frame rate from the camera.

Part 5: Odd-shaped filters

- Try implementing a triangular filter that emphasizes image gradients in a certain direction.

Exercise 4: Contours

The objective of this exercise is to detect the contour of an object. In image analysis, it is generally a good idea to show the result of your analysis graphically. This implies that whenever you have to test methods or results, show it superimposed on the original image.

The contour may be found by one of the following methods:

Gradient Method

A simple method, is to calculate the numerical value of the gradient. It is then assumed that the contour consists of pixels with a gradient larger than a certain value.

Laplacian Method

The contour may also be identified by the Laplacian operator. Explain how this approach differs from the gradient method. Explain also why the contour information is not "connected".

Contour Search

A third possibility is to identify the contour in a binary image. Use the code below to perform a contour search in a binary image:

```
unsigned char *pic; // placeholder for image data
int randx[MAX_RAND], randy[MAX_RAND];
int newpos, local_tresh, draw_type;
draw_type=0;
newpos=pos; // pos equals the starting position in the image ( =
y*Width+x)
while(newpos>=0L && newpos<end)
{
rimx[count]=newpos%B; // save current position in list
rimy[count]=newpos/B;
count++;
draw_type=(draw_type+6)%8; // Select next search direction
switch(draw_type)
{
case 0: if(pic[newpos+1 ]>local_tresh) {newpos+=1;draw_type=0;break;}
case 1: if(pic[newpos+B+1]>local_tresh)
{newpos+=B+1;draw_type=1;break;}
case 2: if(pic[newpos+B ]>local_tresh) {newpos+=B ;draw_type=2;break;}
case 3: if(pic[newpos+B-1]>local_tresh) {newpos+=B-
1;draw_type=3;break;}
case 4: if(pic[newpos-1 ]>local_tresh) {newpos-=1 ;draw_type=4;break;}
case 5: if(pic[newpos-B-1]>local_tresh) {newpos-
=B+1;draw_type=5;break;}
case 6: if(pic[newpos-B ]>local_tresh) {newpos-=B ;draw_type=6;break;}
```

```
case 7: if(pic[newpos-B+1]>local_tresh) {newpos-=B-
1;draw_type=7;break;}
case 8: if(pic[newpos+1 ]>local_tresh) {newpos+=1 ;draw_type=0;break;}
case 9: if(pic[newpos+B+1]>local_tresh)
{newpos+=B+1;draw_type=1;break;}
case 10: if(pic[newpos+B ]>local_tresh) {newpos+=B ;draw_type=2;break;}
case 11: if(pic[newpos+B-1]>local_tresh) {newpos+=B-
1;draw_type=3;break;}
case 12: if(pic[newpos-1 ]>local_tresh) {newpos-=1 ;draw_type=4;break;}
case 13: if(pic[newpos-B-1]>local_tresh) {newpos-
=B+1;draw_type=5;break;}
case 14: if(pic[newpos-B ]>local_tresh) {newpos-=B ;draw_type=6;break;}
}

// If we are back at the beginning, we declare success
if(newpos==pos)
      break;

// Abort if the contour is too complex.
if(count>=MAX_RAND)
      break;
}
```

- Draw an example on a piece of paper, in order to observe the effect.
- Implement the above algorithm on a binary image.
- Implement a similar contour search algorithm based on a 4 or 6 neighbor connectivity.
- How would you store the contour information with relation to its parent object?

Exercise 5: Correspondence Problem

In image analysis one often encounter the need to know where certain objects (set of pixels) located in one image are located in a second image of the same scene, which differ with respect to the first image in attitude or position. This kind of problem is generally known as the correspondence problem.

Solving this problem will often end up with a transformation that transforms pixel positions in the first image to pixel positions in the second image. One such type of transformation is the affine transformation:

$$x' = p_0 + p_1 x + p_2 y$$
$$y' = q_0 + q_1 x + q_2 y$$

where (x, y) are pixel positions in the first image and (x', y') are pixel positions in the second image. The affine transformation may describe both translation, rotation, scaling and skewing between the two images.

Only three pair of points are necessary in order to give the six equations necessary to solve for the 6 parameters. However, often one would choose 30-50 points in each image, obtain a constellation of these points that matches each other and perform a least squares fit of the affine transformation parameters.

This exercise is about selecting these points (pixel positions) in both images, and to implement the affine transformation using the obtained coordinate matches. There are generally two different methods to select these points: a) Correlation Methods and b) Token/Feature Matching Methods.

The first one is the one you are expected to work with in this exercise. The correlation method works by extracting small areas of pixels (e.g. 10×10) in the first image and search for an area in the second image having similar gray-level pixel values. Below are shown two simple measures that can be used to determine the match:

$$m_1 = \sum_{x,y} |f - g|$$
$$´m_2 = \sum_{x,y} (f - g)^2$$

- where the image regions to be compared are denoted f and g, respectively.

- Start by extracting one small area in the first image and search for a corresponding area in the second image. Use the image pair `PIC1_R.PIC` and `PIC1_L.PIC`.
- Next consider how to choose the areas used in the first image and change the program to find 30-50 corresponding areas. Remember to visualize the match for instance by framing and numbering the areas in both images. For each match,

store the x,x',y,y' coordinates for the center pixels in the match areas. Also consider a way of sorting the match areas by confidence of the match.

- Write up and solve the equation relating the matched coordinate pairs (x,y) <-> (x',y') to the parameters p_i,q_i of the affine transformation. Assume that the problem is well-determined; e.g. that you only need 3 coordinate pairs for determining the transformation parameters.
- Apply the affine transformation (or its reverse) to either of the images, and check (using e.g. a difference image) if your transformation is correct. Hint:
    - Create a new image of suitable dimensions (e.g., twice as big as the original
    - For each pixel (x,y) in the old image, calculate the corresponding coordinates (x',y'). If they are outside the boundaries of the new image, simply skip the pixel.
    - Assign the intensity of the original image at (x,y) to the new image at (x',y').

Questions:

- Ideally, how should the match coordinates be distributed in the image?
- What configurations of matched coordinates cannot be used to derive the affine transformation parameters?

Besides the geometric transformation of the two images there may also be differences due to lighting conditions and the cameras vantage point. This means that false matches may be obtained but you should be able to correspond the majority of the selected areas manually.

More questions:

- Consider how to design a filter that rejects false matches.
- What can be changed in order to minimize influence from changing light conditions?

Document last modified: September 07, 2015 by DAKP