Basic Image Processing Tasks with Open CV

For one of our customers in the **scientific domain** we do a lot of integration of pieces of hardware into the existing measurement- and control network. A good part of these are **2D detectors** and scientific **CCD cameras**, which have all sorts of interfaces like Ethernet , firewire and frame grabber cards. Our task is then to write some **glue software** that makes the camera available and controllable for the scientists.

One standard requirement for us is to do some **basic image processing** and analytics. Typically, this entails **flipping** the image horizontally and/or vertically, **rotating** the image around some multiple of 90 degrees, and calculating some **statistics** like standard deviation.

The starting point there is always some **image data in memory** that has been acquired from the camera. Most of the time the image data is either **gray values (8, or 16 bit), or RGB(A)**.

As we are generally not falling victim to the NIH syndrome we use open source image processing library's . The first one we tried was **CImg**, which is a header-only (!) C++ library for image processing. The header-only part is very cool and handy, since you just have to *#include <CImg.h>* and you are done. No further dependencies. The immediate **downside**, of course, is long compile times. We are talking about > 40000 lines of C++ template code!

The bigger issue we had with CImg was that for **multi-channel images** the memory layout is like this: *R1R2R3R4.....G1G2G3G4....B1B2B3B4*. And since the images from the camera usually come **interlaced** like *R1G1B1R2G2B2...* we always had to do tricks to use CImg on these images correctly. These tricks killed us eventually in terms of **performance**, since some of these 2D detectors produce lots of megabytes of image data that have to be processed in **real time**.

So **OpenCV**. Their headline was already very promising:

OpenCV (Open Source Computer Vision) is a library of programming functions for real time computer vision.

Especially the words "real time" look good in there. But let's see.

Image data in OpenCV is represented by instances of class *cv::Mat*, which is, of course, short for Matrix. From the documentation:

The class Mat represents an n-dimensional dense numerical single-channel or multi-channel array. It can be used to store real or complex-

valued vectors and matrices, grayscale or color images, voxel volumes, vector fields, point clouds, tensors, histograms.

Our standard requirements stated above can then be implemented like this (gray scale, 8 bit image):

```
1    void processGrayScale8bitImage(uint16_t width, uint16_t height,
2                                   const double& rotationAngle,
3                                   uint8_t* pixelData)
4    {
5      // create cv::Mat instance
6      // pixel data is not copied!
7      cv::Mat img(height, width, CV_8UC1, pixelData);
8
9      // flip vertically
10     // third parameter of cv::flip is the so-called flip-code
11     // flip-code == 0 means vertical flipping
12     cv::Mat verticallyFlippedImg(height, width, CV_8UC1);
13     cv::flip(img, verticallyFlippedImg, 0);
14
15     // flip horizontally
16     // flip-code > 0 means horizontal flipping
17     cv::Mat horizontallyFlippedImg(height, width, CV_8UC1);
18     cv::flip(img, horizontallyFlippedImg, 1);
19
20     // rotation (a bit trickier)
21     // 1. calculate center point
22     cv::Point2f center(img.cols/2.0F, img.rows/2.0F);
23     // 2. create rotation matrix
24     cv::Mat rotationMatrix =
25       cv::getRotationMatrix2D(center, rotationAngle, 1.0);
26     // 3. create cv::Mat that will hold the rotated image.
27     // For some rotationAngles width and height are switched
28     cv::Mat rotatedImg;
29     if ( (rotationAngle / 90.0) % 2 != 0) {
30       // switch width and height for rotations like 90, 270 degrees
31       rotatedImg =
32         cv::Mat(cv::Size(img.size().height, img.size().width),
33                 img.type());
34     } else {
35       rotatedImg =
36         cv::Mat(cv::Size(img.size().width, img.size().height),
```

```
37              img.type());
38    }
39    // 4. actual rotation
40    cv::warpAffine(img, rotatedImg,
41                rotationMatrix, rotatedImg.size());
42
43    // save into TIFF file
44    cv::imwrite("myimage.tiff", gray);
45  }
```

The cool thing is that almost the same code can be used for our other image types, too. The only difference is the **image type** for the cv::Mat constructor:

8-bit gray scale: CV_U8C1
16bit gray scale: CV_U16C1
RGB : CV_U8C3
RGBA: CV_U8C4

Additionally, the whole thing is **blazingly fast**! All performance problems gone. Yay!

Getting basic **statistical values** is also a breeze:

```
1   void calculateStatistics(const cv::Mat& img)
2   {
3     // minimum, maximum, sum
4     double min = 0.0;
5     double max = 0.0;
6     cv::minMaxLoc(img, &min, &max);
7     double sum = cv::sum(img)[0];
8
9     // mean and standard deviation
10    cv::Scalar cvMean;
11    cv::Scalar cvStddev;
12    cv::meanStdDev(img, cvMean, cvStddev);
13  }
```

All in all, the OpenCV experience was very positive, so far. They even support CMake. Highly recommended!