Programming exercise 2: OpenCV

Credit:http://docs.opencv.org/2.4/doc/tutorials/tutorials.html

1. Load, display images

Image Representation

Create a cv::Mat object can be done in several ways. Create an empty folder, add a *cvmat.cpp* file inside the empty folder and put the following code in your *cvmat.cpp* file.

```
#include "opencv2/opencv.hpp"
#include <iostream>

int main(void)
{
    // cv::Mat(int row,int col,int type) Constructor
    // allocate a 2*2 matrix of unsigned char type
    // with 1 channel
    cv::Mat m0(2,2, CV_8UC1);
    std::cout << "m0 = \n " << m0 << std::endl;

    // a 2*2 matrix of unsigned char type
    // with 3 channels
    cv::Mat m1(2,2, CV_8UC3);
    std::cout << "m1 = \n " << m1 << std::endl;

    // a 3*2 matrix of unsigned char type
    // with 3 channels
    // assigne values for each channel
    // using cv::Scalar
    cv::Mat m2(3,2, CV_8UC3, cv::Scalar(1,0,2));
    std::cout << "m2 = \n " << m2 << std::endl;

    return 0;
}
```

Next thing is to add a cmake file which link to OpenCV library for compiling. In the same folder, create a *CMakeLists.txt* and add the following:

```
cmake_minimum_required(VERSION 2.8)
project( cvmat )
find_package( OpenCV REQUIRED )
add_executable( cvmat cvmat.cpp )
target_link_libraries( cvmat ${OpenCV_LIBS} )
```

To build, follow the same steps as before:

1. `mkdir build`
2. `cd build`
3. `cmake ..`
4. `make`

Then run the executable: `./cvmat`

	Column 0	Column 1	Column ...	Column m
Row 0	0,0	0,1	...	0, m
Row 1	1,0	1,1	...	1, m
Row ...	...,0	...,1	...	..., m
Row n	n,0	n,1	n,...	n, m

Figure 1: OpenCV coordinate.

Check the result and figure out how the rows and columns are index in OpenCV. OpenCV uses the 0-based row index (or y-coordinate) goes first and the 0-based column index (or x-coordinate) follows it. Change the parameters to produce different matrices. Furthermore, try to create a program that gives the similar output as following:

```
m1 =
 [126,   0, 255, 126,   0, 255;
 126,   0, 255, 126,   0, 255]
m2 =
 [  1,   0,   2,   1,   0,   2;
    1,   0,   2,   1,   0,   2;
    1,   0,   2,   1,   0,   2]
m3 =
 [1.1, 0.1, 2.1, 1.1, 0.1, 2.1;
 1.1, 0.1, 2.1, 1.1, 0.1, 2.1]
```

In order to get more insight to the `cv::Mat` class, test the following code and see the results.

```
#include "opencv2/opencv.hpp"
#include <iostream>

int main(void)
{
    // creat a mat object m0 and assign some value
    cv::Mat m0(2,2, CV_8UC1,cv::Scalar(8));
```

```
    // creat a second mat object, and copy the
    // content from m0
    cv::Mat m_tmp(m0);

    // m0 and m_tmp display
    std::cout << "m0 = \n " << m0 << std::endl;
    std::cout << "m_tmp = \n " << m_tmp << std::endl;

    // do some change to m0
    m0.at<uchar>(0,0) = 4;

    // check the resulted output
    std::cout << "m0 = \n " << m0 << std::endl;
    std::cout << "m_tmp = \n " << m_tmp << std::endl;

    return 0;
}
```

Note: During developing your code, remember to re-compile the updated source code using `make`, then execute using `./NameOfExecutable`.

Reference link `http://docs.opencv.org/2.4/doc/tutorials/introduction/linux_gcc_cmake/linux_gcc_cmake.html#linux-gcc-usage`.

Load and display an image

Here's an example of how to load an image from folder. A sample image of this exercise can be found in Commsy room.

```
#include "opencv2/opencv.hpp"

int main(int argc, char** argv)
{
    cv::Mat image;
    // read the image data in the file "gray.png"
    // and store it in 'image'
    image = cv::imread(argv[1]);
    cv::imshow("Gray Image", image);

    //wait infinite time for a keypress
    cv::waitKey(0);

    return 0;
}
```

Add corresponding *CMakeLists.txt* to compile and then run the code(`./LoadImage gray.png`) to view the output. Zoom in the displayed image to see each pixel intensity value. Furthermore, try to access the color information (3 channels) using `cv::Vec3b intensity = image.at<cv::Vec3b>(y,x);`, at (rows,cols),(100,200),(200,100)...

Input from Webcam

Here's an example of how to get video stream from a webcam, create an empty folder, add a *DisplayImage.cpp* file inside the empty folder and put the following code in your *DisplayImage.cpp* file.

```
#include "opencv2/opencv.hpp"

int main(void)
{
    cv::VideoCapture cap;
    // open the default camera, use something
    // different from 0 otherwise;
    if(!cap.open(0))
        return 0;
    for(;;)
    {
          cv::Mat frame;
          cap >> frame;
          // end of video stream
          if( frame.empty() ) break;
          cv::imshow("this is you, smile! :)", frame);
          // stop capturing by pressing ESC
          if( cv::waitKey(1) == 27 ) break;
    }

    return 0;
}
```

Add corresponding *CMakeLists.txt* to compile and then run the code to view video stream from your webcam.

2. Modify a image

Create a program that:

- Read the input stream from the webcam
- Convert the RGB image into gray scale image using `cvtColor`

- Display the input image in one window, display gray scale image in another window

Helpful link `http://docs.opencv.org/2.4/doc/tutorials/introduction/load_save_image/load_save_image.html`.

3. Histogram

Histograms are collected counts of data organized into a set of predefined bins.

Create a program that:

- Read the input stream from the webcam, and Convert the RGB image into gray scale image
- Display the histogram, as well as the original and gray scale image

Furthermore, you could also split the original image into R, G, B channel and display the histogram respectively with in the same window.

Helpful link `http://docs.opencv.org/2.4/doc/tutorials/imgproc/histograms/histogram_calculation/histogram_calculation.html`.

4.Thresholding

Thresholding is the simplest segmentation method.

Create a program that:

- Read the input stream from the webcam, and Convert the RGB image into gray scale image
- Threshold the gray scale image using `Binary`, `Threshold Truncated` and `Threshold to Zero`.

Helpful link `http://docs.opencv.org/2.4/doc/tutorials/imgproc/threshold/threshold.html`

5. Filtering

Smoothing:

Create a program that:

- Read the input stream from the webcam
- Smooth the input image stream using `Homogeneous Blur`, and display output.

- Try different smoothing/blurring method such as `Gaussian blur`, `Median blur`.

Helpful link `http://docs.opencv.org/2.4/doc/tutorials/imgproc/gausian_median_blur_bilateral_filter/gausian_median_blur_bilateral_filter.html#smoothing`

6. Canny edge detection

The Canny Edge detector was developed by John F. Canny in 1986. Also known to many as the optimal detector, Canny algorithm aims to satisfy three main criteria:

- **Low error rate**: Meaning a good detection of only existent edges.
- **Good localization**: The distance between edge pixels detected and real edge pixels have to be minimized.
- **Minimal response**: Only one detector response per edge.

Create a program that:

- Read the input stream from the webcam, and Convert the RGB image into gray scale image.
- Detect the edges using Canny edge detection and display the output.

Helpful link `http://docs.opencv.org/2.4/doc/tutorials/imgproc/imgtrans/canny_detector/canny_detector.html`

7. Morphological Operations

Morphological Operations are a set of operations that process images based on shapes by applying a structuring element to an input image and generate an output image. The most basic morphological operations are two: Erosion and Dilation.

Dilation, erosion and more:

Create a program that:

- Read the input stream from the webcam, and Convert the RGB image into gray scale image.
- Threshold the gray scale image using `Threshold Truncated`
- Apply dilation and erosion on the thresholded image and display the result.
- Try more morphology operations (`Opening and Closing`) using `morphologyEx(...)` and show the result.

Helpful link `http://docs.opencv.org/2.4/doc/tutorials/imgproc/erosion_dilatation/erosion_dilatation.html`.

8. Image Pyramid

An image pyramid is a collection of images - all arising from a single original image - that are successively down-sampled until some desired stopping point is reached.

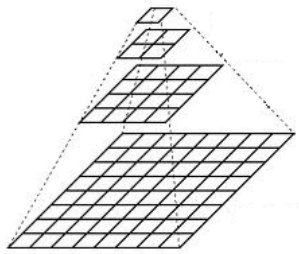

Figure 2: An image pyramid.

Create a program that:

- Read the input stream from the webcam.
- Create up and down pyramid of the input stream and display the results.
- Store one set of images in the pyramid in a vector when user input 's' from keyboard (`cv::waitKey()`). Here's an example on how to use `std::vector`:

```
// load images
cv::Mat image1 = imread("location_image1");
cv::Mat image2 = imread("location_image2");
// create a vector of cv::Mat
std::vector<cv::Mat> various_images;
// add element into vector
various_images.push_back(image1);
various_images.push_back(image2);
```

Furthermore, try to loop through the created vector and display the store images one by one. While displaying the image pyramid from vector, try to let user control the display event by give keyboard input `waitKey(0)`.

Helpful link `http://docs.opencv.org/2.4/doc/tutorials/imgproc/pyramids/pyramids.html`.

9. Switch between thresholding methods

Create a program that:

- Read the input stream from the webcam, and Convert the RGB image into gray scale image.

- Read a keyboard (`cv::waitKey()`) input from user, 'b', 't' or 'z'.
- If user input 'b', threshold the image with `Threshold binary`.
- If user input 't', threshold the image with `Threshold truncate`.
- If user input 'z', threshold the image with `Threshold to zero`.
- Display output

Hint: *`switch()` statement,* *`cv::threshold()`*

10. More on intensities

Create a program that:

- Read the input stream from the webcam.
- Split the input image into R, G and B channels.
- Create a gray scale image that express the difference between R and G channel.
- Display the resulted gray scale image.