ILLINOIS INSTITUTE OF TECHNOLOGY

COMPUTER VISION

CS512

Assignment 1

JiadaTu (ID: A20306906)

ILLINOIS INSTITUTE OF TECHNOLOGY

Language && Lib && OS

C++ && OpenCV && OS X

Functions

a) Read image or capture image from camera as 3 channel color image
b) Save current processed image
c) Convert the image to grayscale
d) Show different channels of the image
e) Convert the image to grayscale and smooth it
f) Convert the image to grayscale and convolute with x or y derivative filter
g) Convert the image to grayscale and compute magnitude of the gradient
h) Convert the image to grayscale and plot the gradient vectors(line segments) every N pixels
i) Rotate the image
j) If capture image from camera, do the capture and processes above continuously on time dimension
k) Show help document

Architecture design

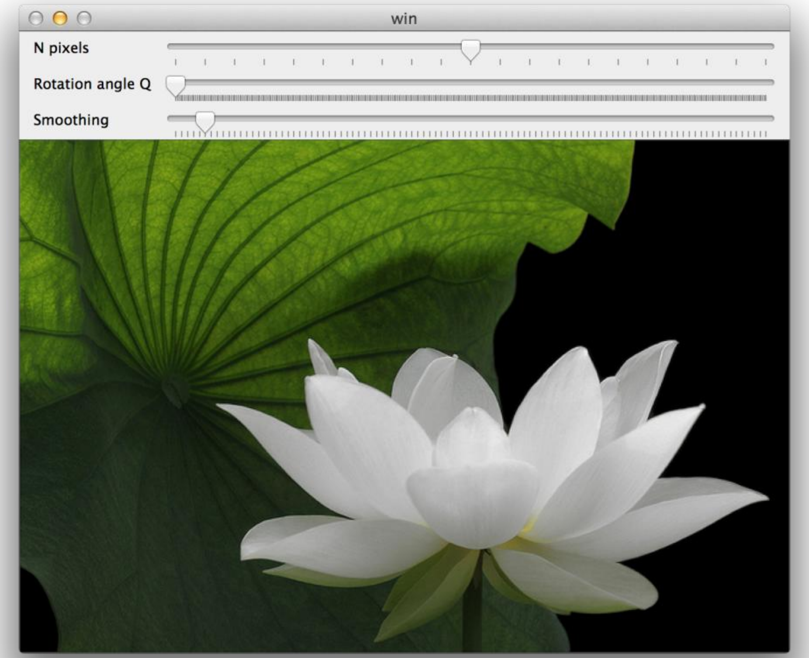
Since I use the support code, the architecture has already been designed. After read the image, the program wait for command in a while loop. If user press nothing, keep loops, else do some operations base on the key pressed. First interpret the first level command(key pressed) to second level command, and do the operations base on the second level command. If user drag on the track bar, try to refresh(a second level command) the image base on what operation is processed on the image and the value on that track bar.

Implementation && I/O results && Correctness

1. Read / capture image

Read the image (imread()) with the file path passed by second argument. If it is not passed, use the camera (VideoCapture) to capture images per few milliseconds. If so, the later processing operations such as rotation will be acted on this image sequence, like a video.

Result:



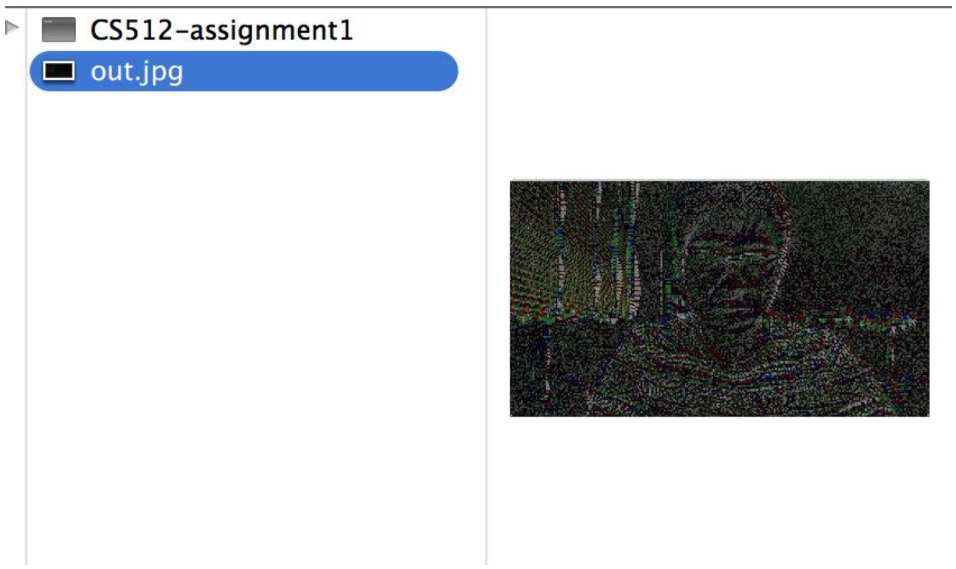
Correctness prove:

not necessary

2. Write image

Just imwrite() function. The file will be saved to the folder where the app located.

Result(gradient plot with N=5):



Correctness prove:

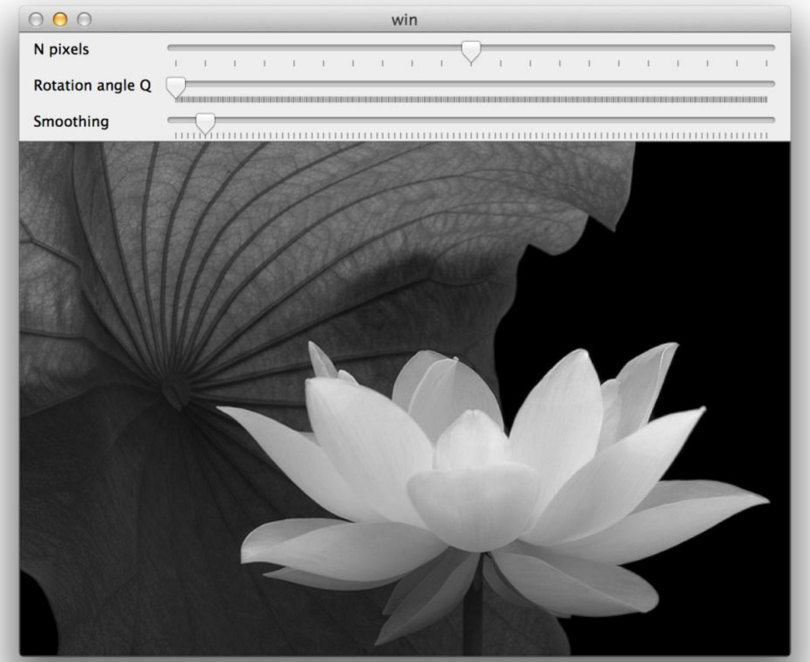
not necessary

3. Grayscale

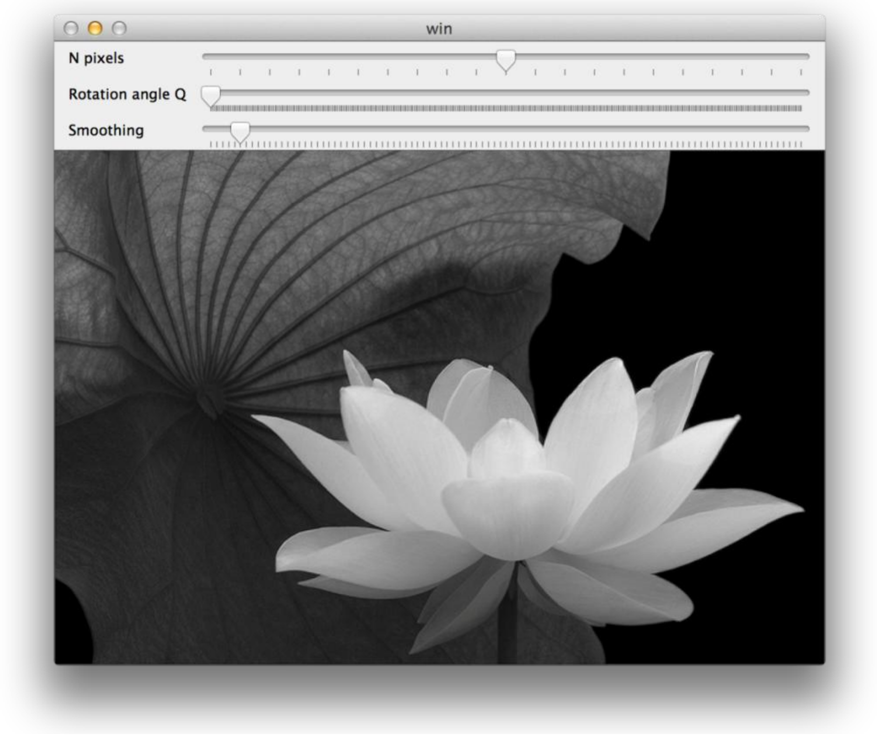
By using the formula: Gray=Red*0.299+Green*0.587+Blue*0.114, it's easy to turn RGB image into grayscale. The grayscale image has only one channel.

Result:

Grayscale by OpenCV:

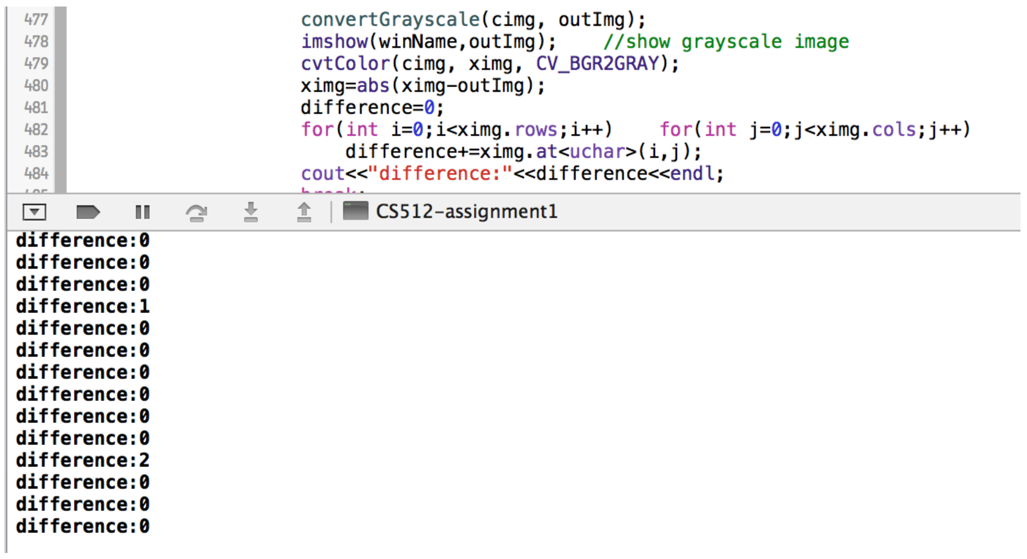


Grayscale by myself:



Correctness prove:

we compute the absolute difference between each pixel of result from OpenCV and own code, and add them. We use the camera, so we can get differences of several image at one time. Here are the difference:



```
convertGrayscale(cimg, outImg);
imshow(winName,outImg);     //show grayscale image
cvtColor(cimg, ximg, CV_BGR2GRAY);
ximg=abs(ximg-outImg);
difference=0;
for(int i=0;i<ximg.rows;i++)     for(int j=0;j<ximg.cols;j++)
    difference+=ximg.at<uchar>(i,j);
cout<<"difference:"<<difference<<endl;
```

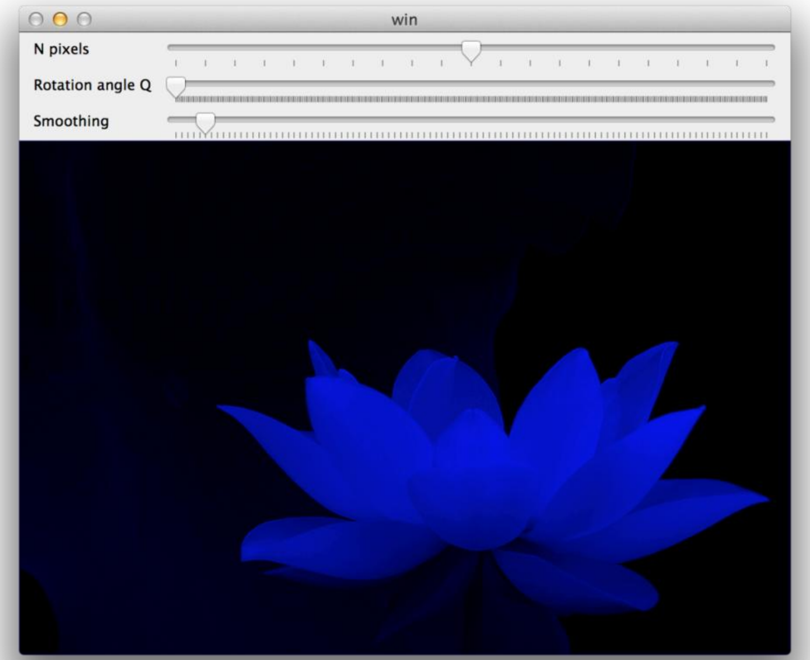
```
difference:0
difference:0
difference:0
difference:1
difference:0
difference:0
difference:0
difference:0
difference:0
difference:0
difference:2
difference:0
difference:0
difference:0
```

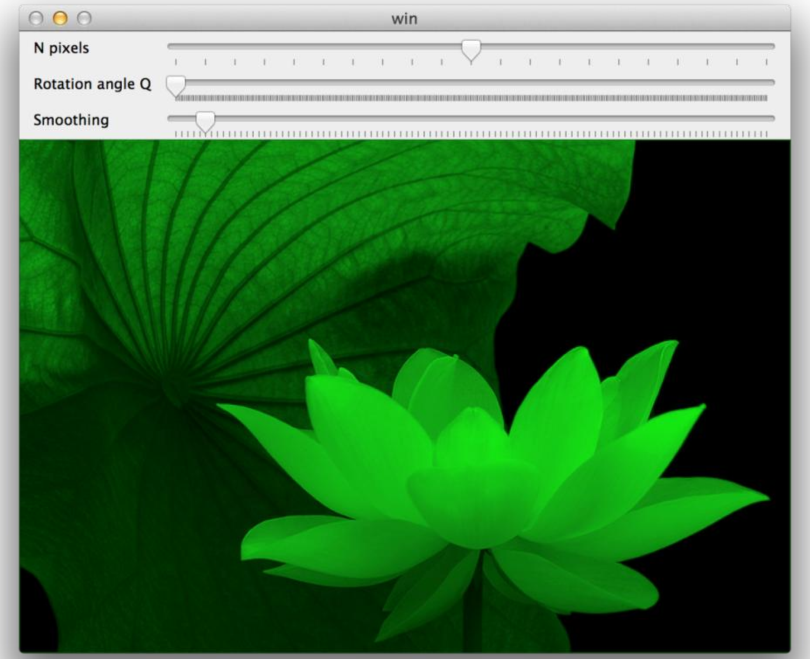
Some differences are not 0, maybe because the precise of the parameter we use in the convertion is different from OpenCV. But a 1280*720 picture only has 2 in absolute difference is acceptable.

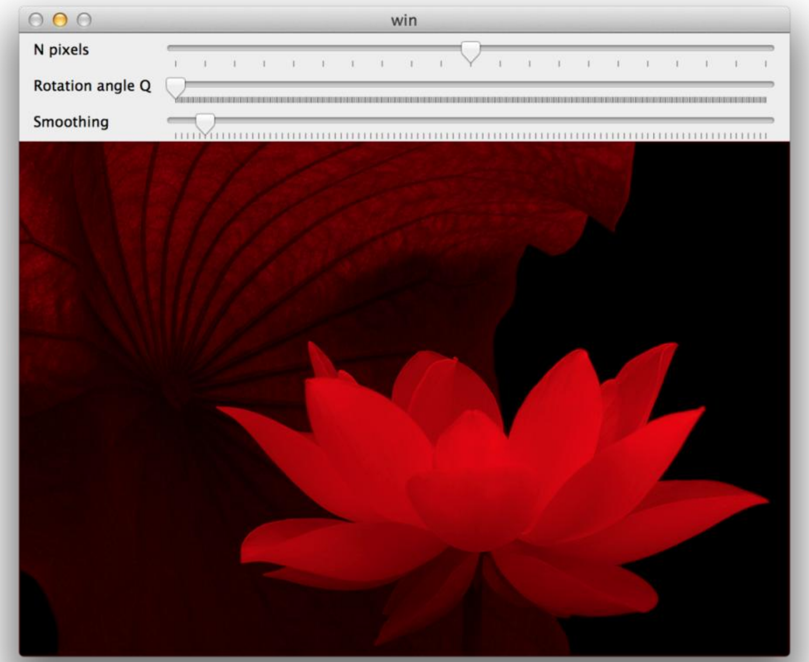
4. Show single channel

We assume the image inputed has 3 channels with range [0,255]. By setting the values of other two channels to 0, we can show single channel of the image (Using "Mat::at<Vec3b>.val[i]").

Result:







Correctness prove:

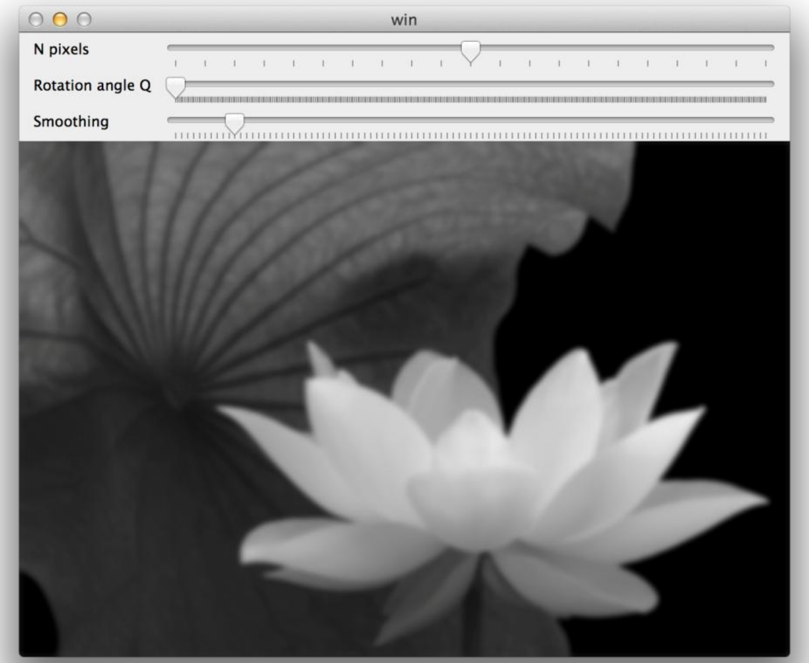
not necessary. But by observing the result we can think it's correct.

5. Smoothing

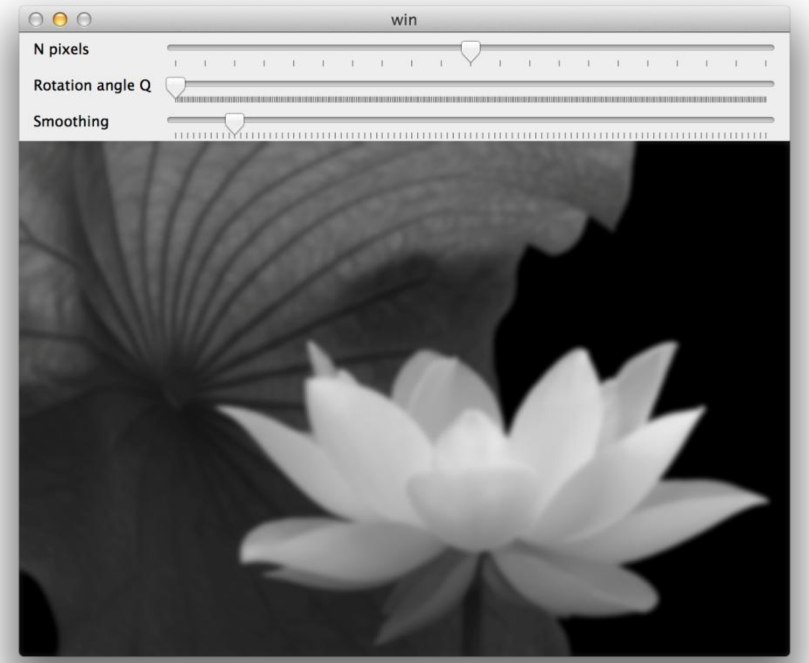
I use the Gaussian filter to do the smoothing. The border I added to the image for the convolution is described in “problems faced”. I separate the blur into blur-on-x and blur-on-y, which increase the performance. I didn't use the filter2D() but write the simple convolution code by my own, resulting to a poor performance (compare to GaussianBlur offer by OpenCV when the size of the filter is over 20). The track bar control the size of the gaussian filter matrix. Since the filter size must be odd, we set the (filter size+=1-(filter size%2)). We set the sigma to (filter size/5.0)

Result:

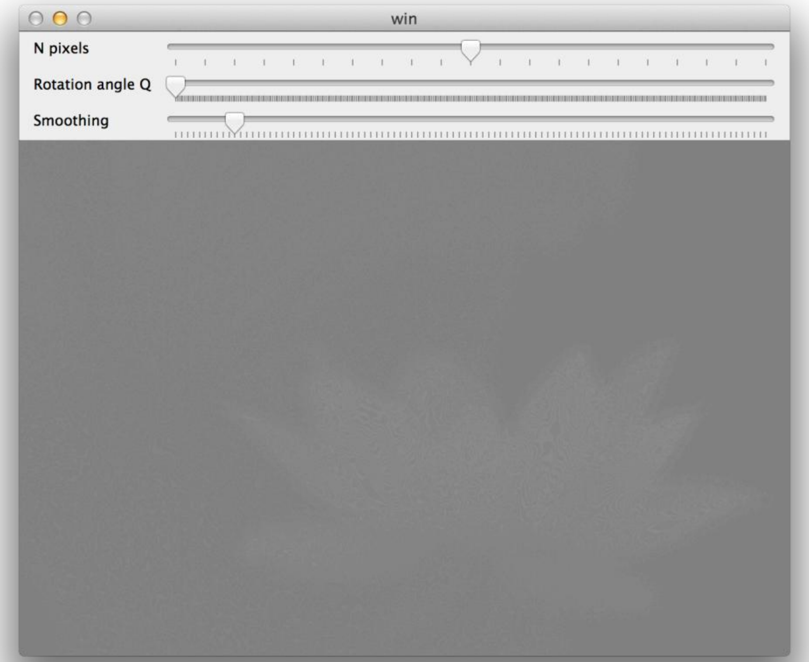
With filter size 11, by OpenCV:



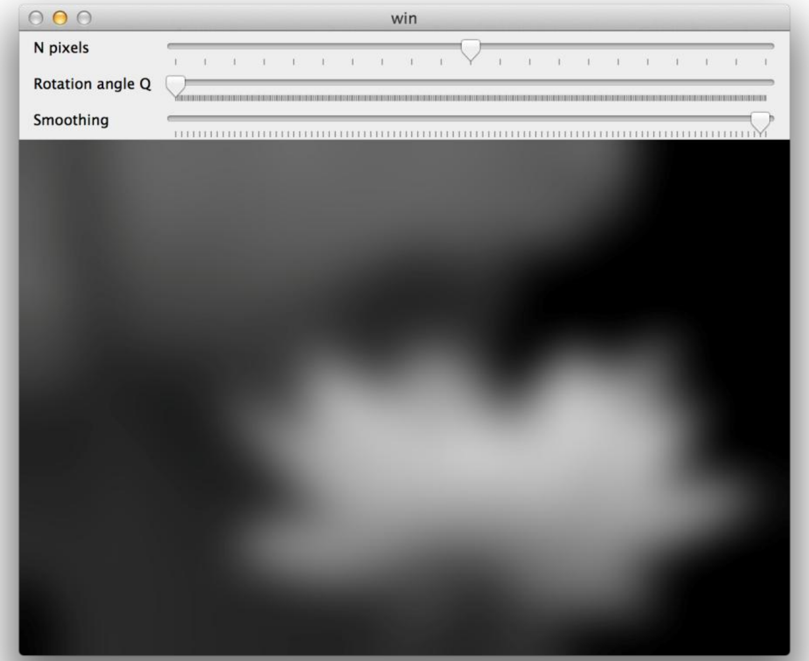
With filter size 11, by myself:



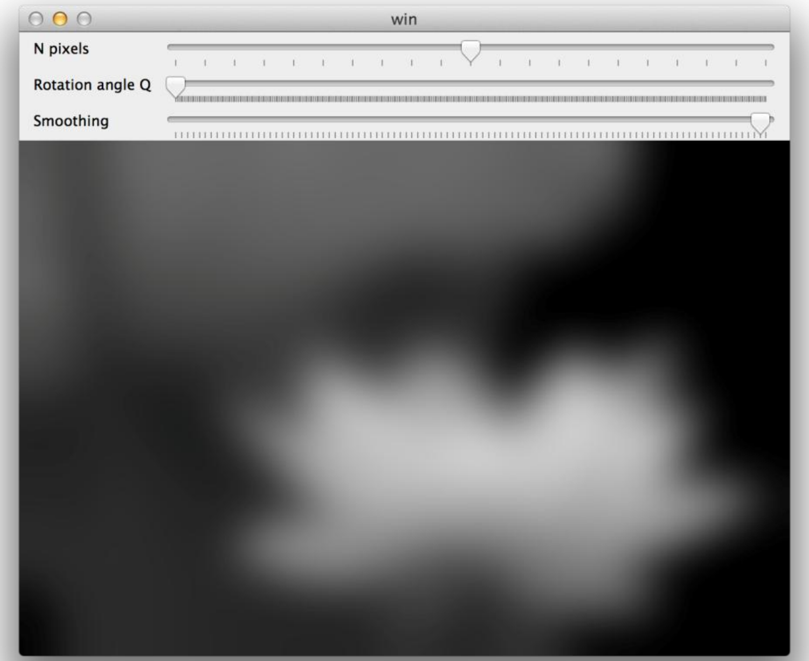
Difference (*5+127.5) (press 'd'):



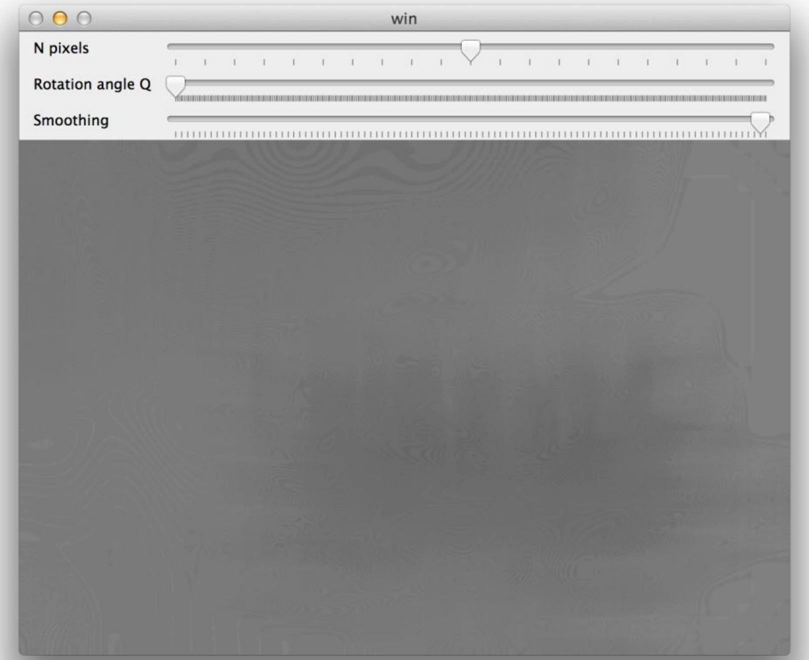
With filter size 99, by OpenCV:



With filter size 99, by myself:



Difference (*5+127.5) (press 'd'):



Correctness prove:

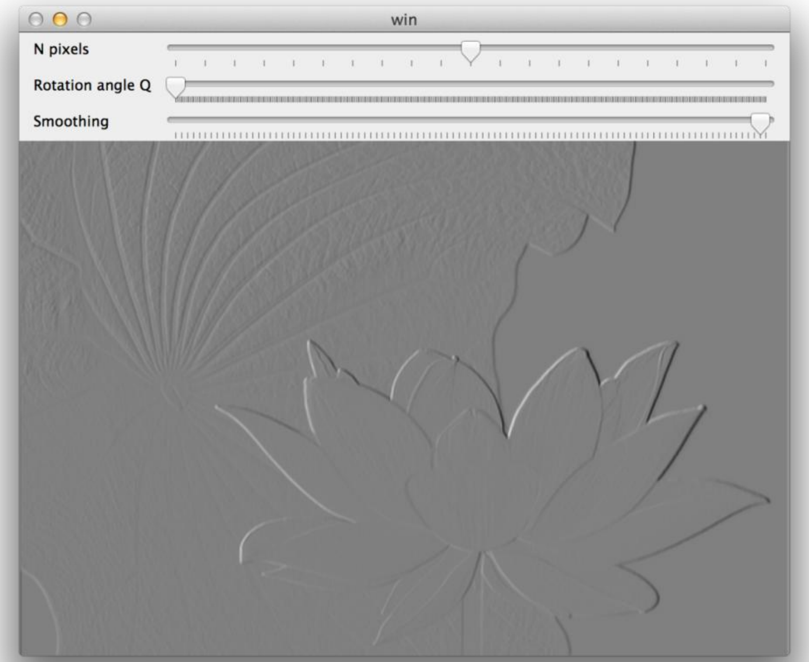
We can see the difference is not zero. But as I will say in "problem faced": I believe the gaussian blur used by OpenCV is a little different(maybe for performance issue), because the brightness of the whole picture is changed in some result of GaussianBlur by OpenCV. Also, the result of my own code and OpenCV on gaussian blur is mainly the same. So I believe my code is correct.

6. x/y derivative filter

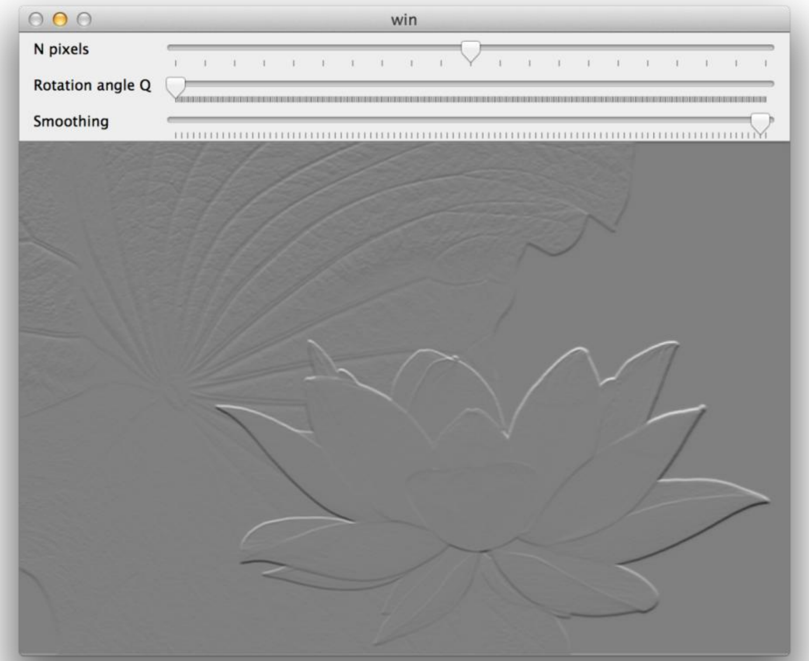
I use [-1,0,1; -1,0,1; -1,0,1] for x filter and [-1,-1,-1; 0,0,0; 1,1,1] for y filter. Since the result value is between [-765,765], I normalize it by add 765 to it and then divided by 6, which maps to [0,255].

Result:

x filter:



y filter:



Correctness prove:

not necessary, because we use filter2D() to do the convolution. But by observing the result, we can see the value on edge is reasonable.

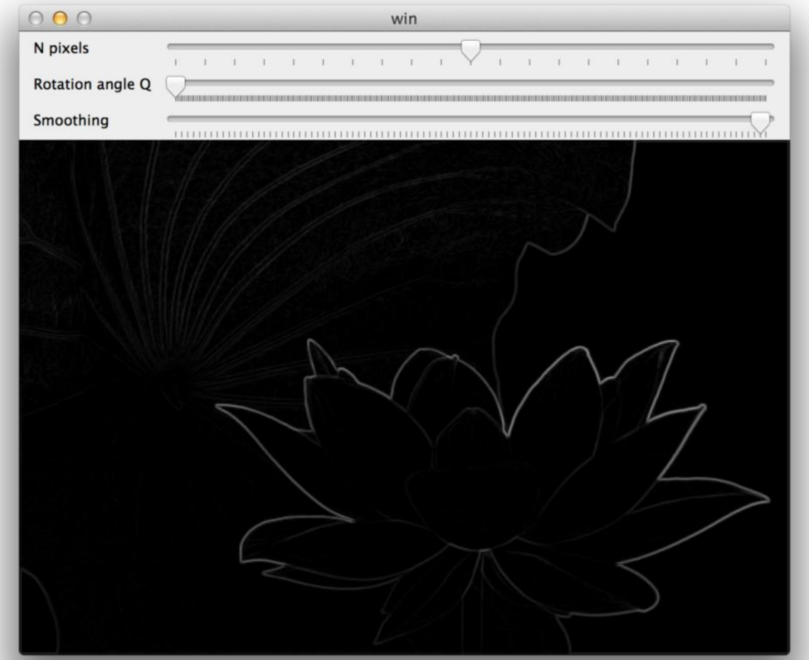
7. Magnitude of the gradient

This operation uses the result of x/y derivative filter operation. Assume dx and dy are x/y derivation(normalized) of a point, then its gradient magnitude is

sqrt((|dx-127.5|^2+|dy-127.5|^2)*2)

which normalized it to [0,255].

Result:



Correctness prove:

Since x/y derivative filter operation is correct, if the equation I use is correct, then it's correct. By observation, the result is quiet reasonable.

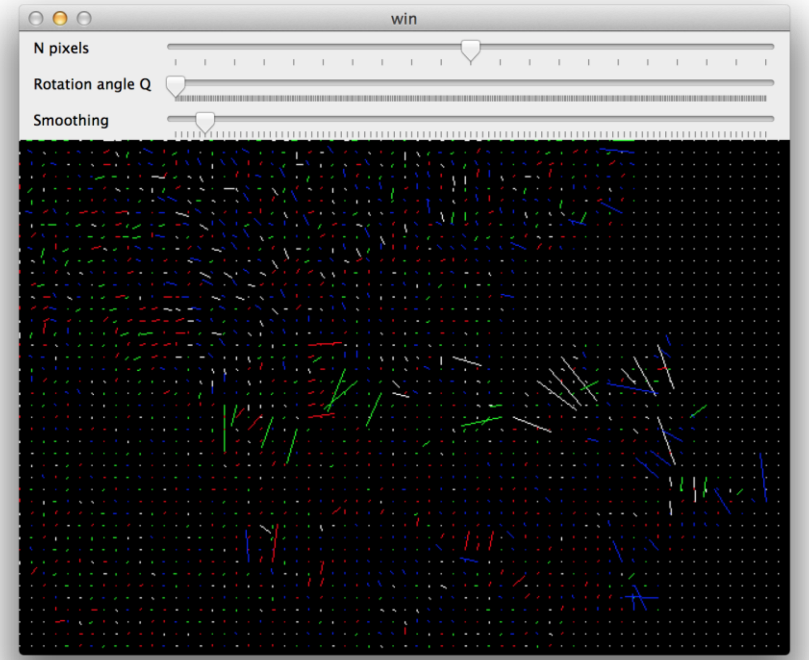
8. Gradient vectors plot

This operation also uses the result of x/y derivative filter operation. Assume dx and dy are x/y derivation(normalized) of a point with position (i,j), N is the pixel gap value controlled by track bar, and dlx=N/2*(dx-127.5)/127.5; dly=N/2*(dy-127.5)/127.5, then the line segment of this point will be: [(i,j) to (i+dlx,j+dly)]. But the line segments are not that obvious according to the experiment result, so I make the line segment [(i,j) to (i+12*dlx,j+12*dly)]. The line segment is orthogonal to the edge. The advantage of this is we don't have to calculate length K of the line segment, it's already there.

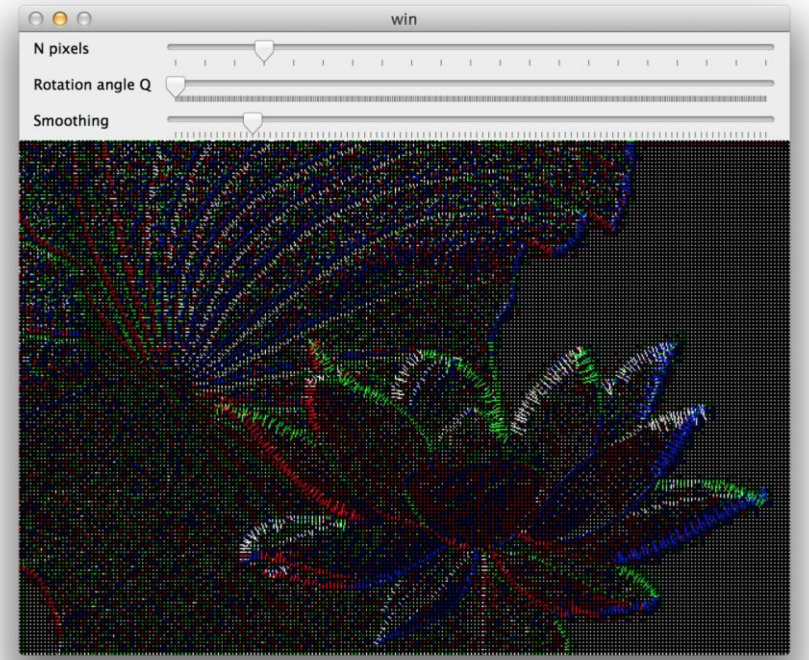
Since line segment has no arrow, I give different color for different directions: (255,0,0) to the second quadrant, (0,255,0) to the third quadrant, (0,0,255) to the first quadrant, (255,255,255) to the fourth quadrant.

Result:

With N=10:


win
N pixels
Rotation angle Q
Smoothing

With N=3:


win
N pixels
Rotation angle Q
Smoothing

Correctness prove:

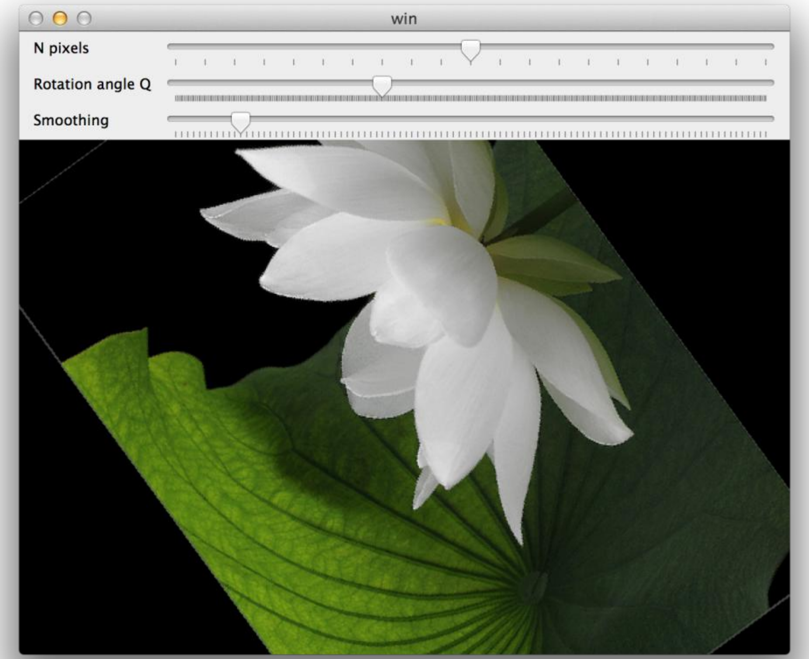
Since x/y derivative filter operation is correct, if the equation I use is correct, then it's correct. By observation, the result is quiet reasonable. The vectors are orthogonal to the edge.

9. Rotation

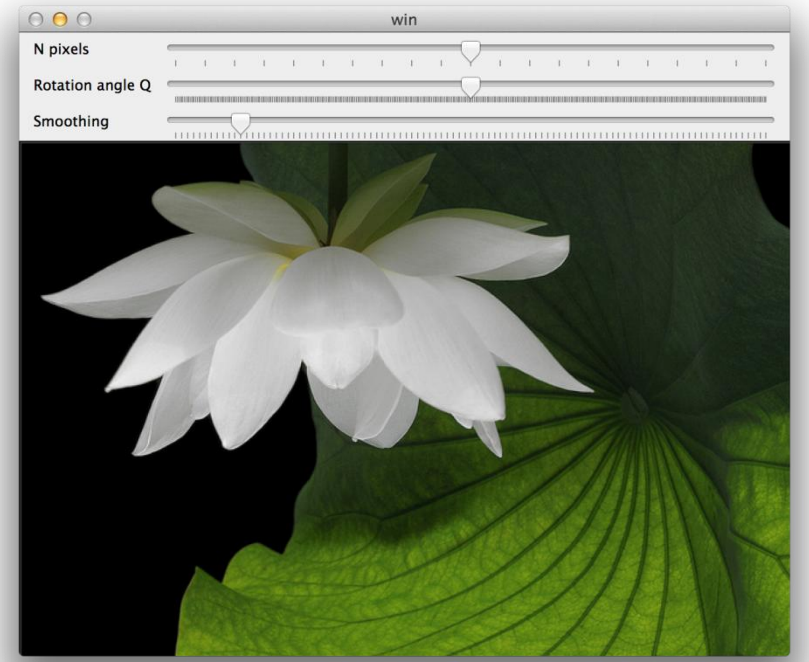
I use matrix multiplication in 2DH space to calculate the corresponding position of each point before the rotation, that means we use inverse map to make sure there are no holes in the result image. The step is: move the image to the origin, do anti-rotation, move the image back.

Result:

Rotate 126 degree:



Rotate 180 degree:



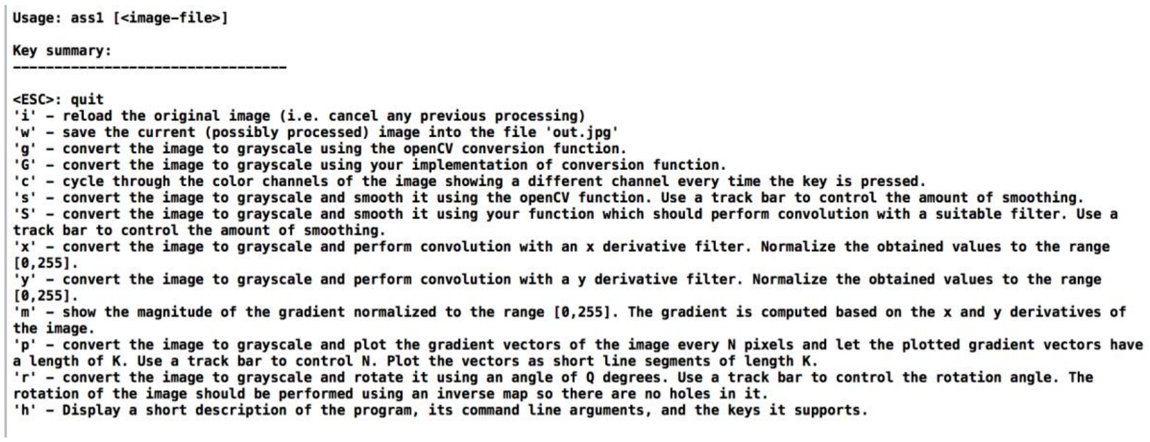
Correctness prove:

By observation, the result is quiet reasonable. There's no hole.

10. Help document

Press 'h' key and the help document will be showed on the command line console.

Result:

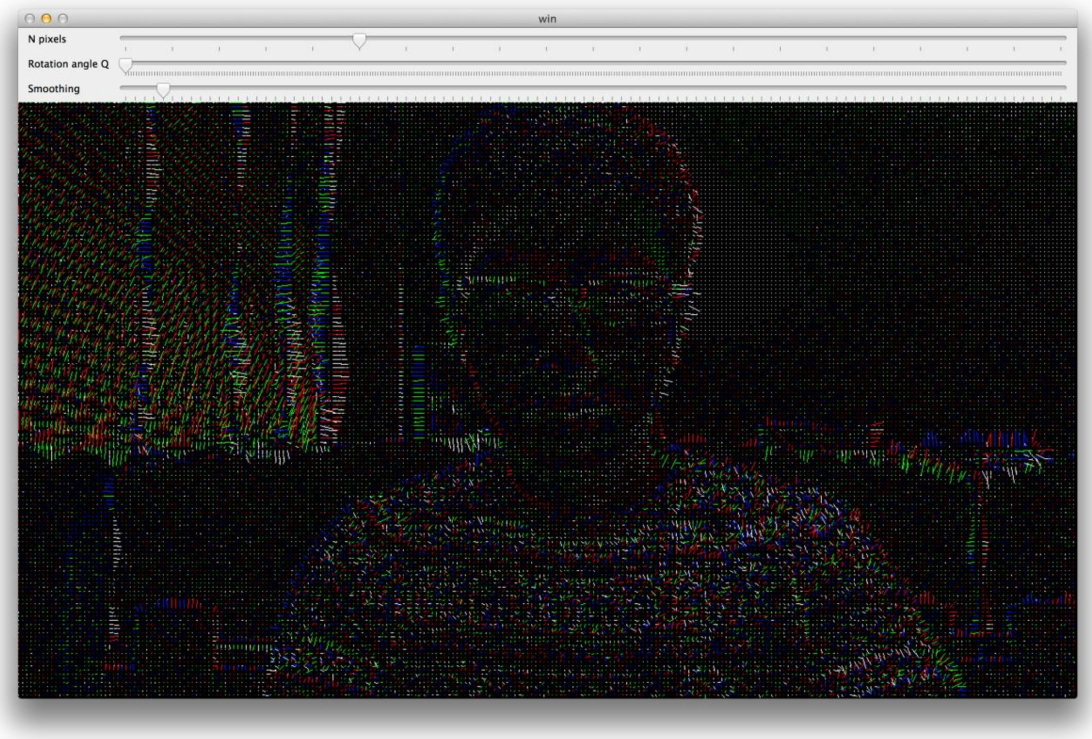


```
Usage: ass1 [<image-file>]

Key summary:
---------------------------------

<ESC>: quit
'i' - reload the original image (i.e. cancel any previous processing)
'w' - save the current (possibly processed) image into the file 'out.jpg'
'g' - convert the image to grayscale using the openCV conversion function.
'G' - convert the image to grayscale using your implementation of conversion function.
'c' - cycle through the color channels of the image showing a different channel every time the key is pressed.
's' - convert the image to grayscale and smooth it using the openCV function. Use a track bar to control the amount of smoothing.
'S' - convert the image to grayscale and smooth it using your function which should perform convolution with a suitable filter. Use a
track bar to control the amount of smoothing.
'x' - convert the image to grayscale and perform convolution with an x derivative filter. Normalize the obtained values to the range
[0,255].
'y' - convert the image to grayscale and perform convolution with a y derivative filter. Normalize the obtained values to the range
[0,255].
'm' - show the magnitude of the gradient normalized to the range [0,255]. The gradient is computed based on the x and y derivatives of
the image.
'p' - convert the image to grayscale and plot the gradient vectors of the image every N pixels and let the plotted gradient vectors have
a length of K. Use a track bar to control N. Plot the vectors as short line segments of length K.
'r' - convert the image to grayscale and rotate it using an angle of Q degrees. Use a track bar to control the rotation angle. The
rotation of the image should be performed using an inverse map so there are no holes in it.
'h' - Display a short description of the program, its command line arguments, and the keys it supports.
```

Other:
Camera with gradient plot with N=5:



Performance

To get the CPU time usage, we use a timer class download from website which use gettimeofday() for linux like OS, QueryPerformanceCounter() for Windows. For each processing operation, we only count the core operation on processing one image, which means we don't count the “convert to grayscale” part on smoothing. Since the program is simple, I don't think we need to count the space usage, but I record the memory usage given by Xcode, just for reference. Since my program does not call Mat::release(), so we just record the lowest memory usage.

Experiment platform:

OS:	OS X 10.9.1
Processor:	2.3GHz Intel Core i7
Memory:	16GB
Hard disk:	SSD

Result:

Operation	Image size	CPU time usage/image		Memory usage	
		OpenCV version	Own function	OpenCV version	Own function
Do nothing (read)	1280*720 (camera)	-	-	50.2MB	50.2MB
	640*428	-	-	13.4MB	13.4MB
GrayScale	1280*720 (camera)	0.673ms	17.571ms	51.5MB	52.1MB
	640*428	0.337ms	6.863ms	14.0MB	14.2MB
Color channel	1280*720 (camera)	-	10.019ms	-	50.3MB
	640*428	-	4.157ms	-	14.5MB
Gaussian Blur with size 3	1280*720 (camera)	1.502ms	63.319ms/ 21.553ms(using filter2D)	51.5MB	58.3MB
	640*428	0.362ms	20.114ms 8.606ms(using filter2D)	14.8MB	21.1MB
Gaussian Blur size 11	1280*720 (camera)	4.691ms	136.656ms 27.220ms(using filter2D)	51.6MB	58.4MB
	640*428	1.284ms	40.629ms 10.016ms(using filter2D)	15.0MB	20.4MB

Gaussian Blur size 99	1280*720 (camera)	31.774ms	1032.560ms 93.803ms(using filter2D)	51.9MB	59.8MB
	640*428	11.959ms	298.858ms 35.178ms(using filter2D)	15.3MB	22.7MB
X/Y filter(3*3)	1280*720 (camera)	-	21.864ms	-	59.2MB
	640*428	-	8.580ms	-	16.4MB
Gradient magnitude	1280*720 (camera)	-	83.555ms	-	59.3MB
	640*428	-	24.936ms	-	18.6MB
Plot gradient vector with N=1	1280*720 (camera)	-	140.233ms	-	59.1MB
	640*428	-	40.883ms	-	23.2MB
Plot gradient vector with N=5	1280*720 (camera)	-	37.038ms	-	58.6MB
	640*428	-	12.118ms	-	23.2MB
Plot gradient vector with N=20	1280*720 (camera)	-	33.136ms	-	58.6MB
	640*428	-	10.841ms	-	23.2MB
Rotation	1280*720 (camera)	-	53.604ms	-	50.5MB
	640*428	-	16.157ms	-	27.0MB

Obviously, the code write by myself is much slower than the OpenCV function, especially gaussian blur, which don't use the filter2D(). After using filter2D, the time performance is up to 11 times better, but still several times slow then OpenCV. Except smoothing, plot with N=1 takes most time, then Gradient magnitude, then Rotation, then X/Y filter(3*3), then GrayScale, then Color channel. The order is reasonable, since drawing line will takes a lot of time; gradient magnitude takes time to calculate; rotation takes three matrix multiplication which is more complicate than x/y filter; GrayScale needs simple calculate when color channel doesn't.

Problems faced

1. In gaussian blur, we need to add border to the image to make the convolution result the same size as the original image. So what value should we add? My choice is the mirror reflection of the image value. For example, for the left border of the image: assume the x position of the left border is negative and the position of the points of original image do not changed(start from (0,0)), then position (-i,j) has the same value as position (i,j). This choice make the border of the blur result more native.

2. There are many matrix(or image) operations in the program. If I use loops to do the operation for each elements in the matrix(or image), the program will be slow. So I try to use the matrix operation which the OpenCV offered to increase the running speed. For example, in the rotation: At first I do the position calculate one time for each point (assume the position after rotation is (px, py, 1)), turning out with low speed. So I make a matrix which contains all points' position in its columns and do the matrix multiplications just one time. This increasing the speed a lot.

3. I write my own convolution and make-border code in gaussian blur. They are not optimized, so my gaussian blur is slow, although I have already used the separated version(blur on two direction) of gaussian blur. For better performance, I use filter2D() in x/y derivative filter convolution.

4. The result of my gaussian blur is a little different from the GaussianBlur offered by OpenCV. If you press 'd', the program will show you the difference(by multiple 5 and plus 127.5). I observed the difference and result, finding that the GaussianBlur offered by OpenCV changes the brightness of the original image a little when my gaussian blur don't. Resulting that I think my gaussian blur is right although it's different from the GaussianBlur().

5. It's not a problem but I want to mention it. Since we use integer matrix to save the image, all medium result of operation such as rotation, gaussian blur and gradient magnitude must be saved in floating point matrix.