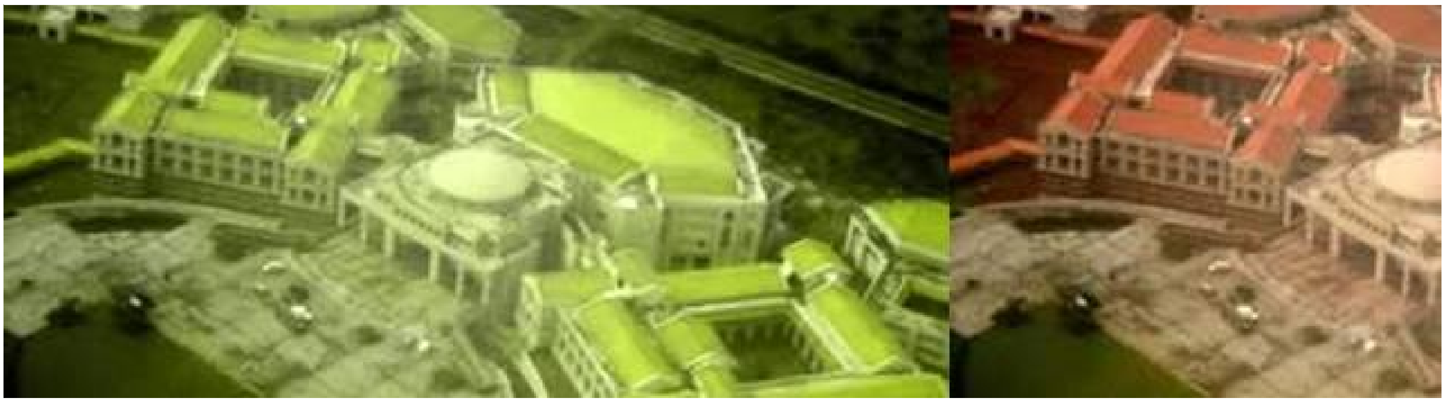


Filtering images

By Utkarsh | Published: February 10, 2010

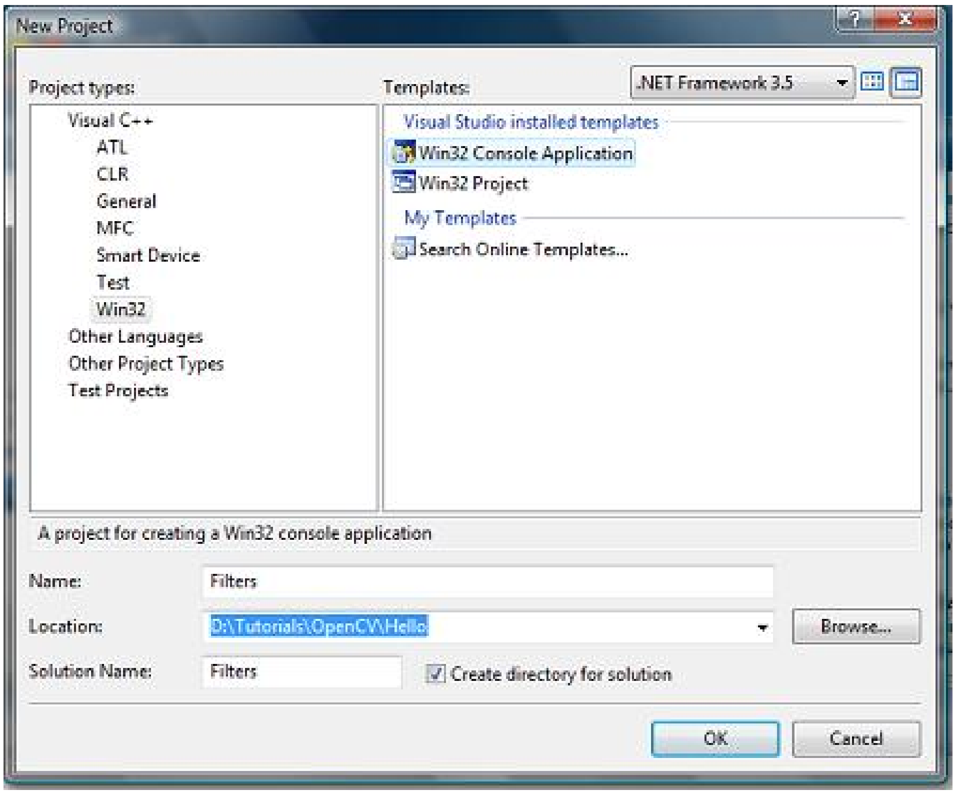
Suka 2.575 orang menyukai ini.

Introduction

In the previous tutorial, Hello World with Images, you learned how to load an image. In this tutorial, we'll take it a step further. You'll do photoshop style "editing" of the image in real time. Things like increasing the brightness, equalizing the histogram, blurring, removing noise, etc. And doing all of this requires only a few lines of code!

Step 1: Creating the application

Create a new Win32 Console Application, and name it "Filters". Save it wherever you like. Accept the defaults and click Finish.



A project for creating a Win32 console application

Name: Filters

Location: D:\Tutorials\OpenCV\Hello

Solution Name: Filters

[✓] Create directory for solution

This generates a project. Open the file Filters.cpp. We'll modify its code.

At the top, include the CV headers:

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

In the main function,

```
int main()
{
```

Add these lines to the main function:

```
    IplImage* img = cvLoadImage("C:\\orangeman.jpg");

    cvNamedWindow("Original");
    cvShowImage("Original", img);

    // We add processing code here

    cvWaitKey(0);
    cvReleaseImage(&img);
    return 0;
}
```

What we're doing here is, loading the file "C:\orangeman.jpg" into img. Then we create a window, and display the image in it. This is the original image. And then wait till eternity for a key press. As soon as a key is pressed, the image will be released and the program will terminate.

Now add links to the various OpenCV libraries as shown in Step 3 of the Hello World with OpenCV tutorial.

Step 2: Trying out things

We'll just experiment with the various "tools" available. This will demonstrate the power of OpenCV, and you'll learn about OpenCV as well.

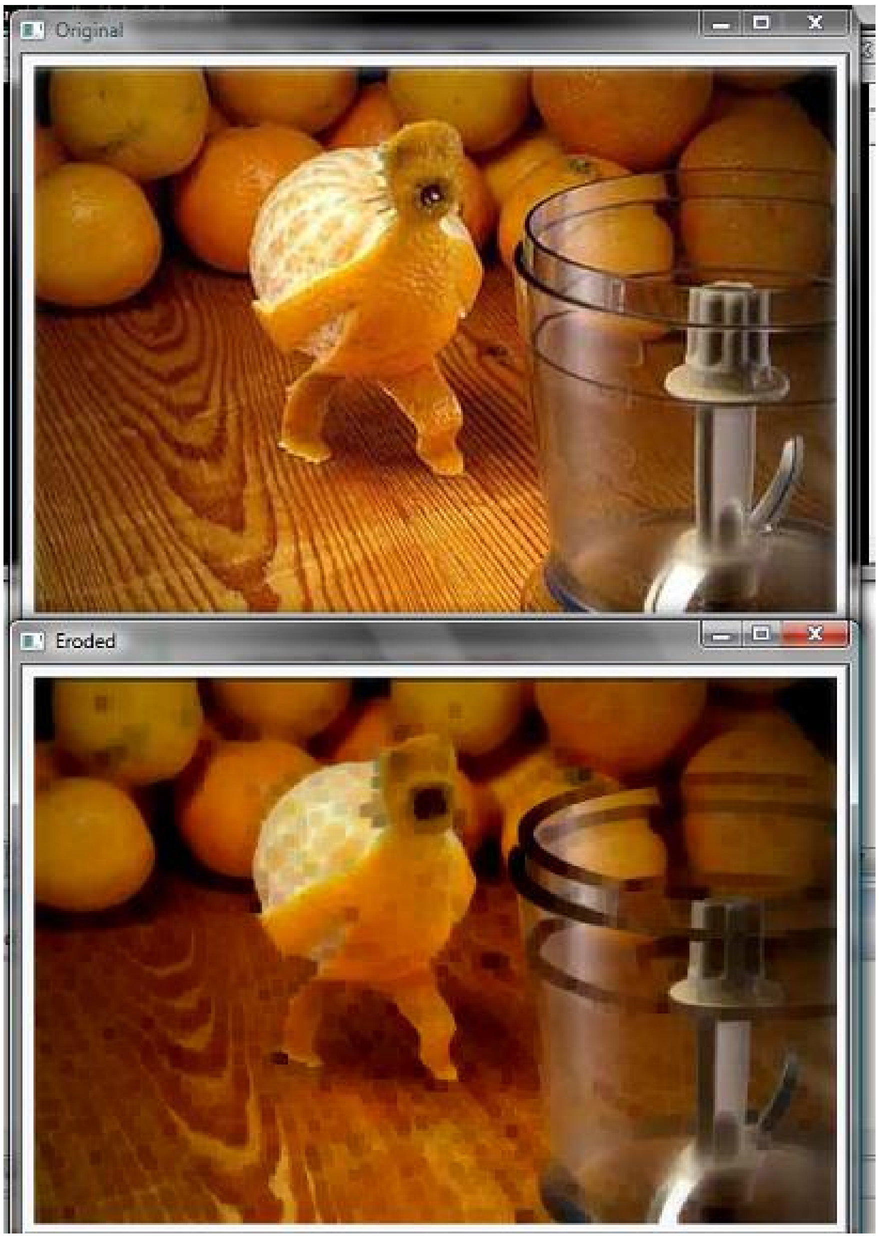
Part A: Eroding the image

Replace the "// We add processing code" herewith the following code:

```
    cvErode(img, img, 0, 2);

    cvNamedWindow("Eroded");
    cvShowImage("Eroded", img);
```

Notice the cvErode? This function takes in image specified in the first parameter, and outputs an eroded image into the second parameter. The third parameter is mostly for advanced uses, so I won't go into the details about that. The fourth parameter is the number of times you want to erode the image. In my case, I tell it to erode the image twice.

Once we've eroded the image, we create a new window titled "Eroded" and display the image in there. Here's the output I got:



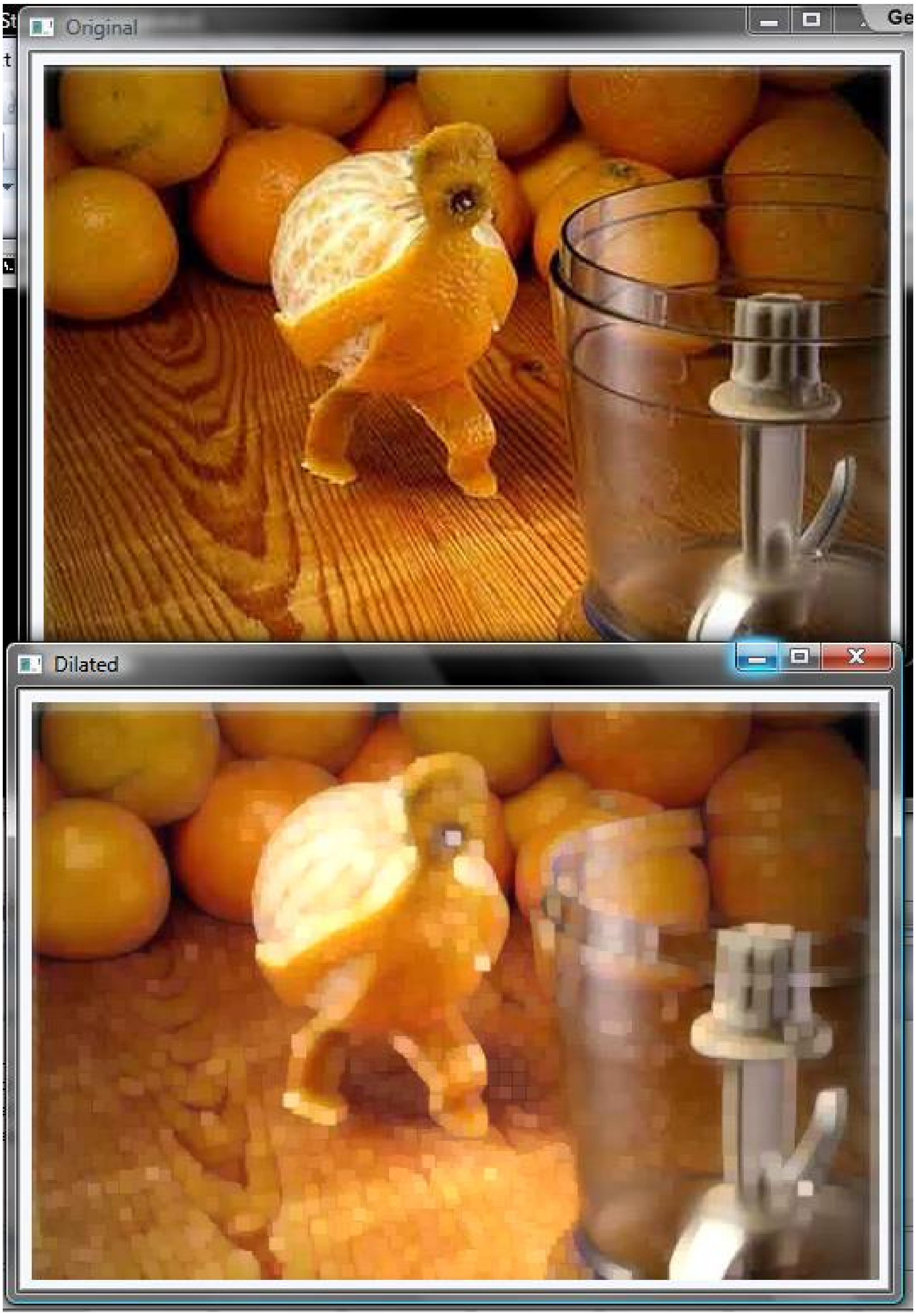
Note: cvErode is an in-place function… meaning the input and output images can be same. Not many functions in OpenCV are in-place. Most of them require that the input and output images are different.

Part B: Dilating the image

Dilating the image is as simple as eroding it. Just replace "Erode" in the above code with a "Dilate"!! Here's the complete code:

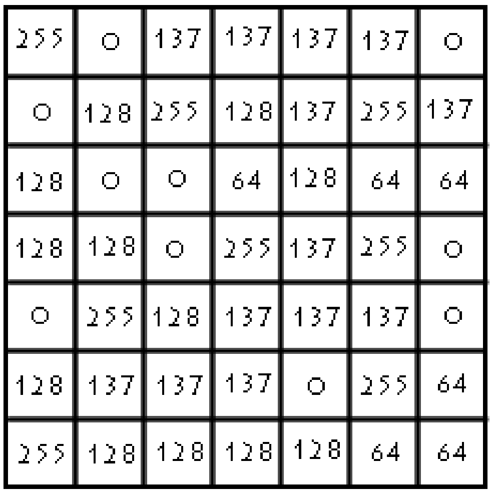
```
int main()
{
    IplImage* img = cvLoadImage("C:\\orangeman.jpg");
    cvNamedWindow("Original");
    cvShowImage("Original", img);

    cvDilate(img, img, 0, 2);
    cvNamedWindow("Dilated");
    cvShowImage("Dilated", img);

    cvWaitKey(0);
    cvReleaseImage(&img);
    return 0;
}
```

Again, cvDilate is an in-place function. Here's the result I got on my image:



Part C: Brightness

OpenCV stores images in the form of matrices. Yes, the matrices you studied in high school. If you didn't, a matrix is a grid of values… something like this:



255	0	137	137	137	137	0
0	128	255	128	137	255	137
128	0	0	64	128	64	64
128	128	0	255	137	255	0
0	255	128	137	137	137	0
128	137	137	137	0	255	64
255	128	128	128	128	64	64

In OpenCV's matrix, each pixel can have a value from 0 to 255. So, each pixel has 8-bits of memory alloted to it. If the image is grayscale, it has just one such matrix. So you often hear of 8-bit grayscale images.

If the image is coloured (say a JPG image), it will have 3 matrices. One matrix for the Red component, one for the Green and one for Blue (or, technically speaking, three 8-bit planes). This means each pixel has 3×8 bits alloted to it (hence, you hear of a 24-bit image).

So to increase the brightness of an image, you just add some value to the entire image. If it is a coloured image, you add that value to each of the components.

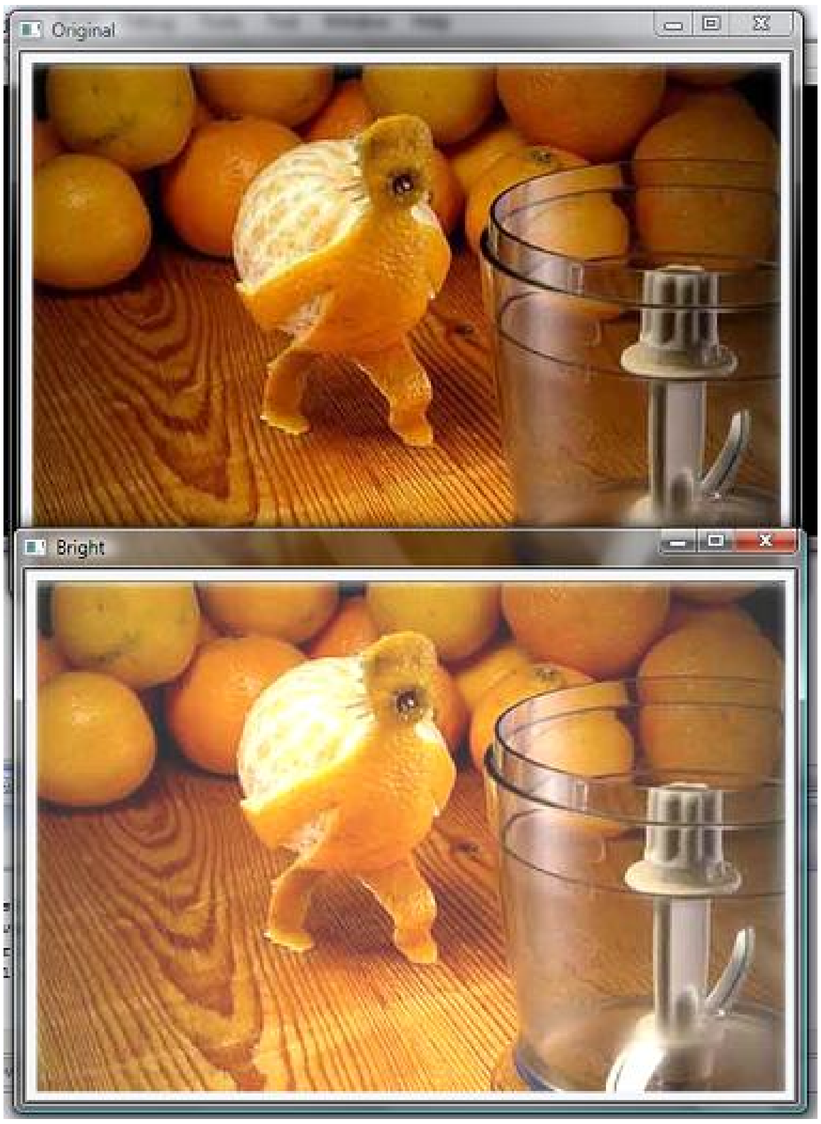
To add, we use the cvAddS function. This function adds a scalar to each element of the matrix. There also exists a cvAdd function, but that function adds two matrices (not a matrix and a value).

Replace the // We add processing code herewith the following code:

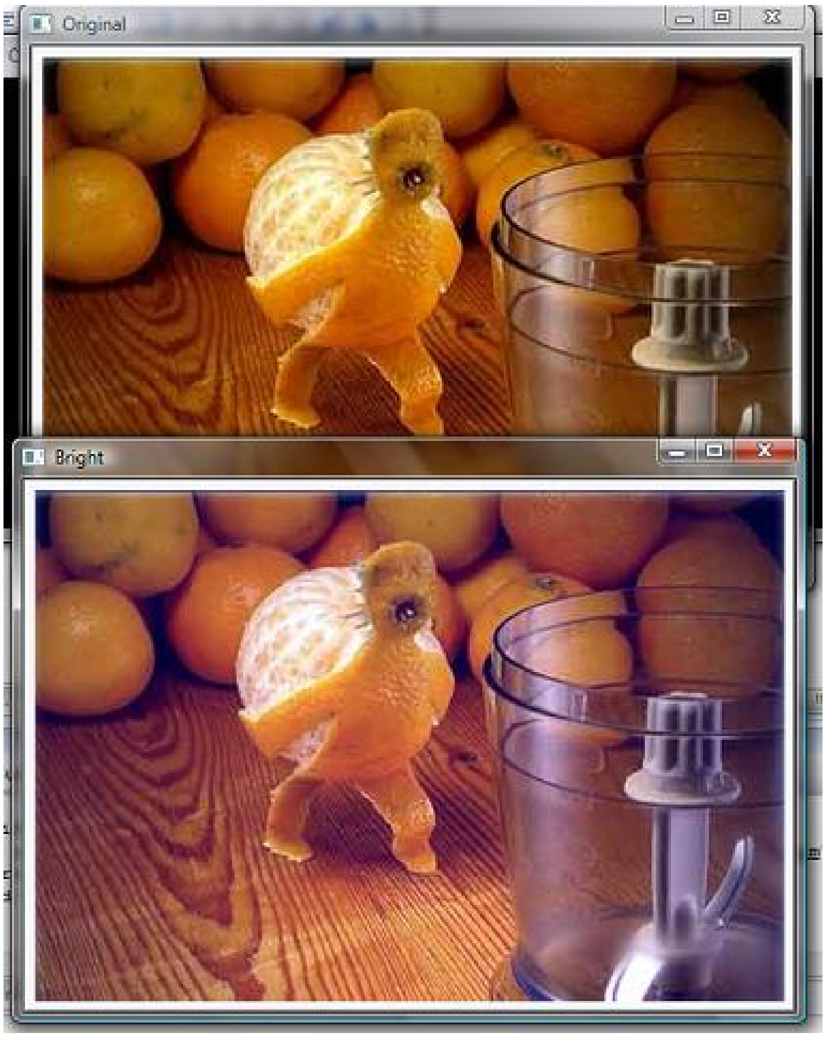
```
cvAddS(img, cvScalar(50,50,50), img);
cvNamedWindow("Bright");
cvShowImage("Bright", img);
```

The cvAddS function takes 3 parameters. The first is the source image. The last is the destination image. The second parameter is the value you want to add to each pixel.

Because this is a coloured image, it will have 3 matrices. And we need to add the constant to each of the 3 planes. So we specify the value to be added for each plane using cvScalar(50, 50, 50).



Had this been a grayscale image (with only one matrix), we'd do just a cvScalar(50). However, if you did a cvScalar(50) to a coloured image, you'd get wierd results…. something like this:



Part D: Contrast

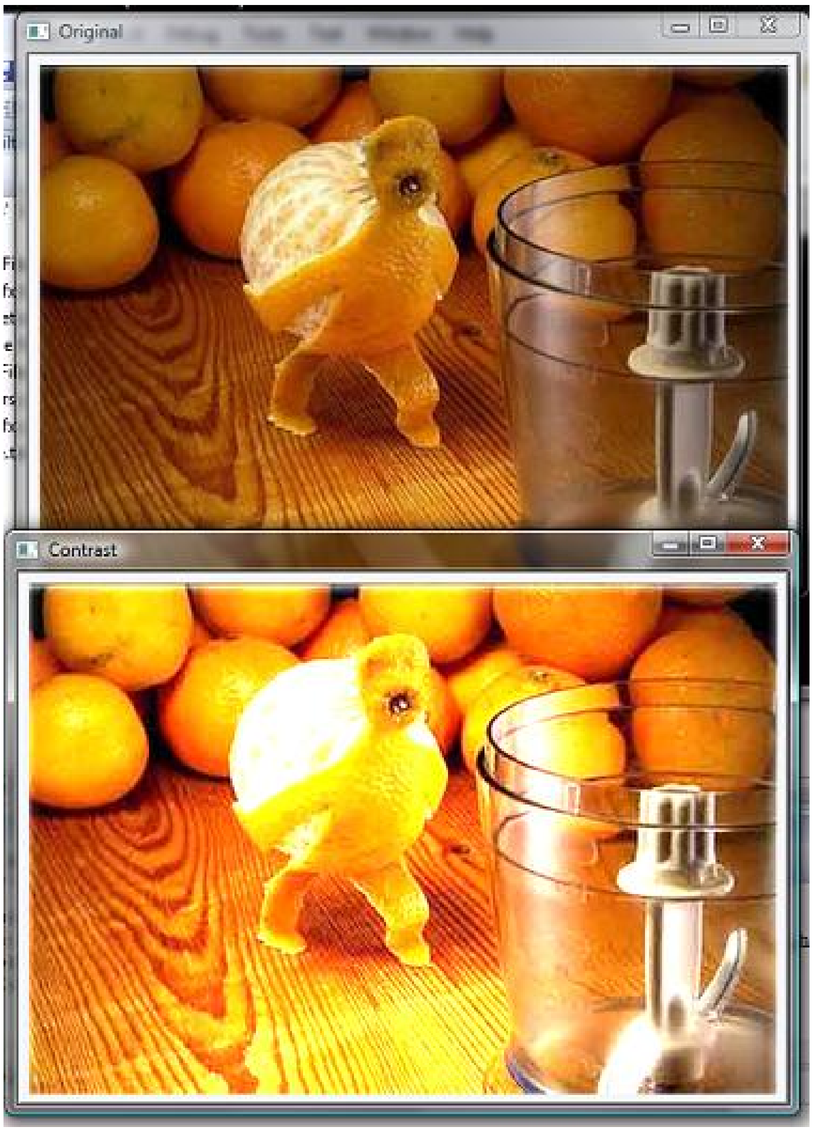
For brightness, you added values. For contrast, you multiply.

For adding scalars (values), you use cvAddS. For multiplying, you use cvScale. Here's the code for it… replace the // We add processing code herewith the following code:

```
cvScale(img, img, 2);
cvNamedWindow("Contrast");
cvShowImage("Contrast", img);
```

cvScale's first parameter is the source image, the second is the destination image. And the last parameter is the multiplication value. In this case, each pixel is multiplied by 2. If you multiplied by 1, you won't see any change.

Here's the output I got:

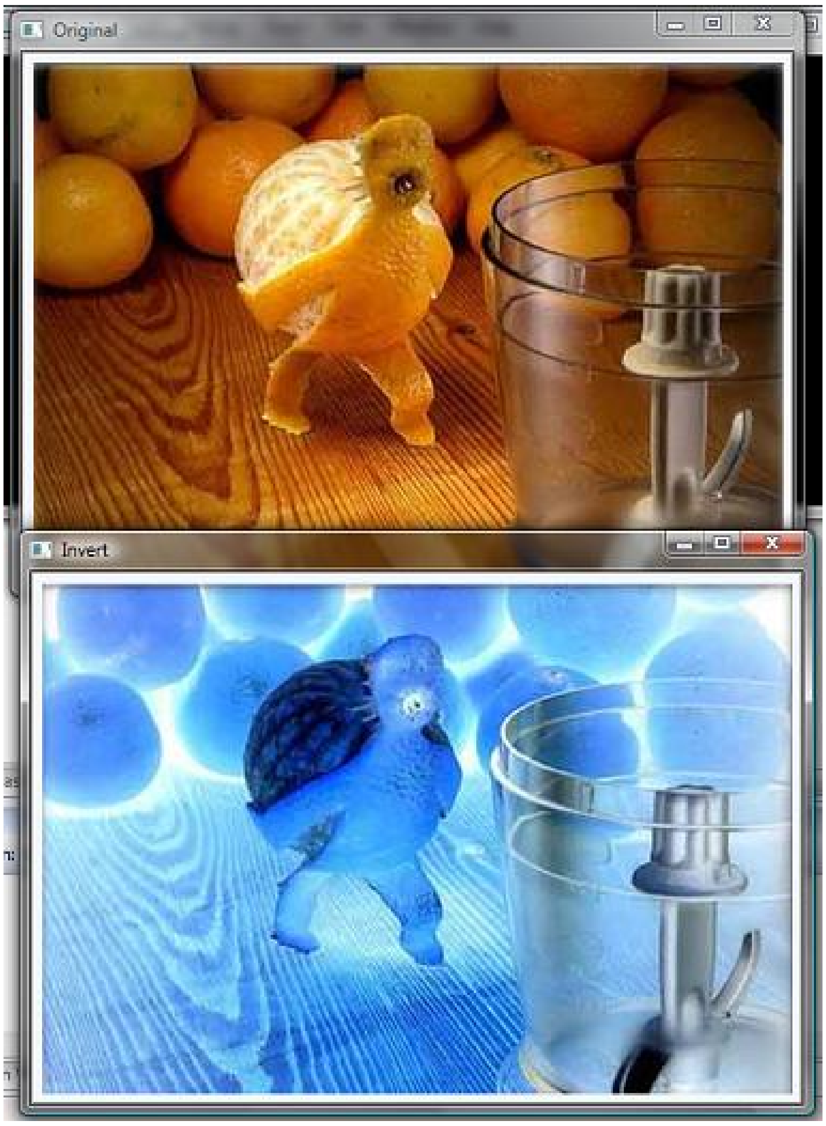


Part E: Negative

Inverting an image is equivalent to doing a logical NOT operation on each element in the matrix. Here's code for doing it… which I'm sure you can work out on your own now:

```
cvNot(img, img);
cvNamedWindow("Invert");
cvShowImage("Invert", img);
```

The cvNot performs a logical NOT on each pixel of the source image (the first parameter) and stores the output in the destination image (the second parameter). Then we create a window, and display the processed image in it:



Why not just use Photoshop?

This is a question that is often asked… after all, photoshop has a lot more fancy effects… plastic wrap, lens flare and what not.

With Photoshop, you can modify one image and save it. Or at best, modify a batch of images and save them. But thats it.

With OpenCV, you can take the images in realtime (from a camera) and do things with those images in real time. This is something you simply cannot do with Photoshop.

Conclusion

I hope you learned quite a bit in this tutorial. For any feedback or criticism, please do **leave a comment below**! In the next tutorial, you'll see how to take input directly from a webcam!

Next Parts

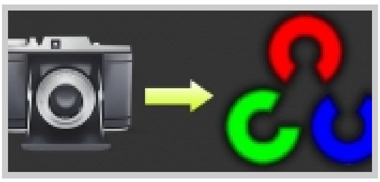
This post is a part of an article series on OpenCV for Beginners

Issues? Suggestions? Visit the Github issue tracker for AI Shack

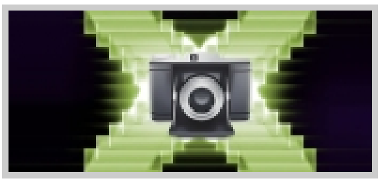
Back to top

Suka 2.575 orang menyukai ini.

Hello World! with Images



Capturing images



Capturing images with DirectX

7 Comments

Uni
Posted September 28, 2010 at 9:18 pm | *Permalink*

Did this next. Seemed simple enough for my simpleton brain! 😀

Thanks very much!!

Reply

anay
Posted October 16, 2010 at 6:23 pm | *Permalink*

nice work boss 😀

Reply

seygyi
Posted November 2, 2010 at 4:24 pm | *Permalink*

I am a newbie to OpenCV. Would you please advise me how to deblur an image?

Reply

Utkarsh
Posted November 12, 2010 at 1:29 am | *Permalink*

Well, that’s a complex thing to do. You’d have to write your own functions to do that.

Reply

Leanne
Posted June 17, 2011 at 3:21 pm | *Permalink*

Thanks for all the guidance here. It’s good for me, as a newbie for OpenCV 😀 Hope to read more on your updates on OpenCV2.2…

Reply

Varun
Posted January 16, 2012 at 11:12 am | *Permalink*

I am a newbie to the OpenCv 😀 And I wanted to learn it b’coz of SOP (I guess u know what it is) :). Your tutorials are very easy and interesting making life simpler….

Thanks a lot….

Reply

Utkarsh
Posted January 20, 2012 at 12:44 am | *Permalink*

SOP’s are fun 😐 Just make sure you mention about AI Shack/me to your prof 😛

Reply

Post a Comment

Your email is *never* published nor shared. Required fields are marked *

Name *

Email *

Website

Comment

You may use these HTML tags and attributes `<a href="" title=""> <abbr title=""> <acronym title=""> <b> <blockquote cite=""> <cite> <code> <del datetime=""> <em> <i> <q cite=""> <strike> <strong>`

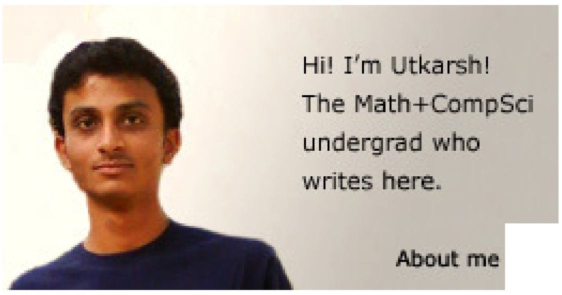
Post Comment

Free updates

Email Address

More info

1000+ subscribers!



Popular Posts

- Scanning QR Codes
- The Canny Edge Detector
- Implementing Canny Edges from scratch
- Image Moments
- Predator: Tracking + Learning

More on AI SHack

Popular **Beginners** **Utkarsh's favs**

- Scanning QR Codes
- The Canny Edge Detector
- Implementing Canny Edges from scratch
- Image Moments
- Predator: Tracking + Learning

About me

My name is Utkarsh Sinha, and I'm an undergraduate student, pursuing B.E. Computer Science + M.Sc. Mathematics. Here, I help you understand ideas in Artificial Intelligence, using a not so techy and mathematical language. And in the process, learn more about Artificial Intelligence myself.

Read more at the about page