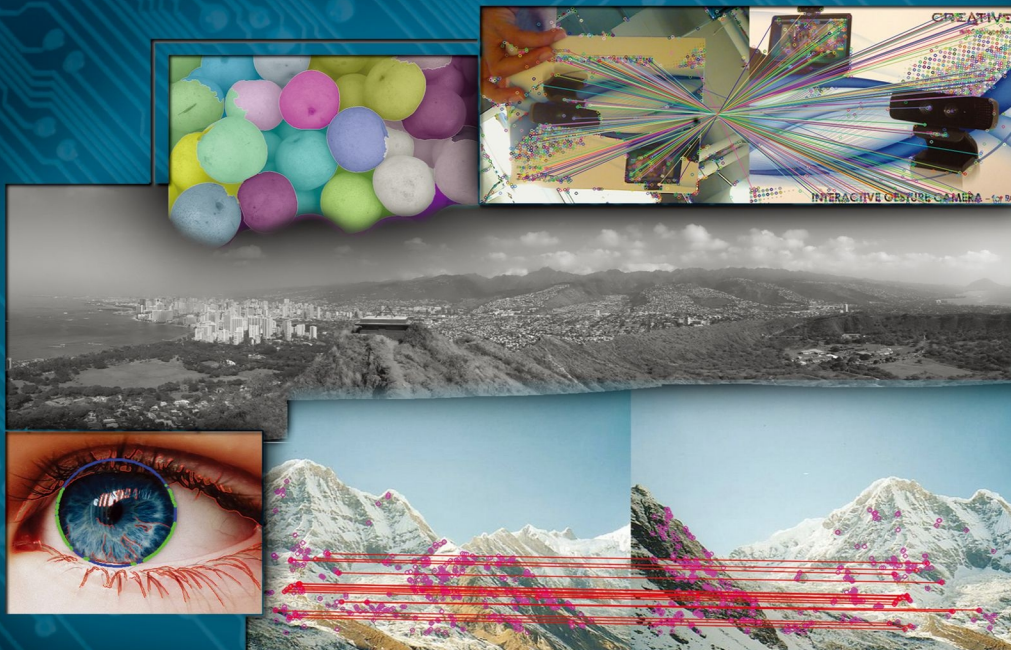




TECHNOLOGY IN ACTION™

Practical OpenCV

*HANDS ON PROJECTS FOR COMPUTER
VISION ON THE WINDOWS, LINUX AND
RASPBERRY PI PLATFORMS*



Samarth Brahmhatt

For your convenience Apress has placed some of the front matter material after the index. Please use the Bookmarks and Contents at a Glance links to access them.



Apress®

Contents at a Glance

About the Author	xiii
About the Technical Reviewer	xv
Acknowledgments	xvii
■ Part 1: Getting Comfortable	1
■ Chapter 1: Introduction to Computer Vision and OpenCV	3
■ Chapter 2: Setting up OpenCV on Your Computer	7
■ Chapter 3: CV Bling—OpenCV Inbuilt Demos.....	13
■ Chapter 4: Basic Operations on Images and GUI Windows.....	23
■ Part 2: Advanced Computer Vision Problems and Coding Them in OpenCV	39
■ Chapter 5: Image Filtering.....	41
■ Chapter 6: Shapes in Images.....	67
■ Chapter 7: Image Segmentation and Histograms.....	95
■ Chapter 8: Basic Machine Learning and Object Detection Based on Keypoints	119
■ Chapter 9: Affine and Perspective Transformations and Their Applications to Image Panoramas.....	155
■ Chapter 10: 3D Geometry and Stereo Vision.....	173
■ Chapter 11: Embedded Computer Vision: Running OpenCV Programs on the Raspberry Pi.....	201
Index.....	219

PART 1



Getting Comfortable



Introduction to Computer Vision and OpenCV

A significant share of the information that we get from the world while we are awake is through sight. Our eyes do a wonderful job of swiveling about incessantly and changing focus as needed to see things. Our brain does an even more wonderful job of processing the information stream from both eyes and creating a 3D map of the world around us and making us aware of our position and orientation in this map. Wouldn't it be cool if robots (and computers in general) could see, and understand what they see, as we do?

For robots, seeing in itself is less of a problem—cameras of all sorts are available and quite easy to use. However, to a computer with a camera connected to it, the camera feed is technically just a time-varying set of numbers.

Enter computer vision.

Computer vision is all about making robots intelligent enough to take decisions based on what they see.

Why Was This Book Written?

In my opinion, robots today are like personal computers 35 years ago—a budding technology that has the potential to revolutionize the way we live our daily lives. If someone takes you 35 years ahead in time, don't be surprised to see robots roaming the streets and working inside buildings, helping and collaborating safely with humans on a lot of daily tasks. Don't be surprised also if you see robots in industries and hospitals, performing the most complex and precision-demanding tasks with ease. And you guessed it right, to do all this they will need highly efficient, intelligent, and robust vision systems.

Computer vision is perhaps the hottest area of research in robotics today. There are a lot of smart people all around the world trying to design algorithms and implement them to give robots the ability to interpret what they see intelligently and correctly. If you too want to contribute to this field of research, this book is your first step.

In this book I aim to teach you the basic concepts, and some slightly more advanced ones, in some of the most important areas of computer vision research through a series of projects of increasing complexity. Starting from something as simple as making the computer recognize colors, I will lead you through a journey that will even teach you how to make a robot estimate its speed and direction from how the objects in its camera feed are moving.

We shall implement all our projects with the help of a programming library (roughly, a set of prewritten functions that can execute relevant higher-level tasks) called OpenCV.

This book will familiarize you with the algorithm implementations that OpenCV provides via its built-in functions, theoretical details of the algorithms, and the C++ programming philosophies that are generally employed while using OpenCV. Toward the end of the book, we will also discuss a couple of projects in which we employ OpenCV's framework for algorithms of our own design. A moderate level of comfort with C++ programming will be assumed.

OpenCV

OpenCV (Open-source Computer Vision, opencv.org) is the Swiss Army knife of computer vision. It has a wide range of modules that can help you with a lot of computer vision problems. But perhaps the most useful part of OpenCV is its architecture and memory management. It provides you with a framework in which you can work with images and video in any way you want, using OpenCV's algorithms or your own, without worrying about allocating and deallocating memory for your images.

History of OpenCV

It is interesting to delve a bit into why and how OpenCV was created. OpenCV was officially launched as a research project within Intel Research to advance technologies in CPU-intensive applications. A lot of the main contributors to the project included members of Intel Research Russia and Intel's Performance Library Team. The objectives of this project were listed as:

- Advance vision research by providing not only open but also optimized code for basic vision infrastructure. (No more reinventing the wheel!)
- Disseminate vision knowledge by providing a common infrastructure that developers could build on, so that code would be more readily readable and transferable.
- Advance vision-based commercial applications by making portable, performance-optimized code available for free—with a license that did not require the applications to be open or free themselves.

The first alpha version of OpenCV was released to the public at the IEEE Conference on Computer Vision and Pattern Recognition in 2000. Currently, OpenCV is owned by a nonprofit foundation called OpenCV.org.

Built-in Modules

OpenCV's built-in modules are powerful and versatile enough to solve most of your computer vision problems for which well-established solutions are available. You can crop images, enhance them by modifying brightness, sharpness and contrast, detect shapes in them, segment images into intuitively obvious regions, detect moving objects in video, recognize known objects, estimate a robot's motion from its camera feed, and use stereo cameras to get a 3D view of the world—to name just a few applications. If, however, you are a researcher and want to develop a computer vision algorithm of your own for which these modules themselves are not entirely sufficient, OpenCV will still help you a lot by its architecture, memory-management environment, and GPU support. You will find that your own algorithms working in tandem with OpenCV's highly optimized modules make a potent combination indeed.

One aspect of the OpenCV modules that needs to be emphasized is that they are highly optimized. They are intended for real-time applications and designed to execute very fast across a variety of computing platforms from MacBooks to small embedded fitPCs running stripped down flavors of Linux.

OpenCV provides you with a set of modules that can execute roughly the functionalities listed in Table 1-1.

Table 1-1. *Built-in modules offered by OpenCV*

Module	Functionality
Core	Core data structures, data types, and memory management
Imgproc	Image filtering, geometric image transformations, structure, and shape analysis
Highgui	GUI, reading and writing images and video
Video	Motion analysis and object tracking in video
Calib3d	Camera calibration and 3D reconstruction from multiple views
Features2d	Feature extraction, description, and matching
Objdetect	Object detection using cascade and histogram-of-gradient classifiers
ML	Statistical models and classification algorithms for use in computer vision applications
Flann	Fast Library for Approximate Nearest Neighbors—fast searches in high-dimensional (feature) spaces
GPU	Parallelization of selected algorithms for fast execution on GPUs
Stitching	Warping, blending, and bundle adjustment for image stitching
Nonfree	Implementations of algorithms that are patented in some countries

In this book, I shall cover projects that make use of most of these modules.

Summary

I hope this introductory chapter has given you a rough idea of what this book is all about! The readership I have in mind includes students interested in using their knowledge of C++ to program fast computer vision applications and in learning the basic theory behind many of the most famous algorithms. If you already know the theory, and are interested in learning OpenCV syntax and programming methodologies, this book with its numerous code examples will prove useful to you also.

The next chapter deals with installing and setting up OpenCV on your computer so that you can quickly get started with some exciting projects!

CHAPTER 2



Setting up OpenCV on Your Computer

Now that you know how important computer vision is for your robot and how OpenCV can help you implement a lot of it, this chapter will guide you through the process of installing OpenCV on your computer and setting up a development workstation. This will also allow you to try out and play with all the projects described in the subsequent chapters of the book. The official OpenCV installation wiki is available at <http://opencv.willowgarage.com/wiki/InstallGuide>, and this chapter will build mostly upon that.

Operating Systems

OpenCV is a platform independent library in that it will install on almost all operating systems and hardware configurations that meet certain requirements. However, if you have the freedom to choose your operating system I would advise a Linux flavor, preferably Ubuntu (the latest LTS version is 12.04). This is because it is free, works as well as (and sometimes better than) Windows and Mac OS X, you can integrate a lot of other cool libraries with your OpenCV project, and if you plan to work on an embedded system such as the Beagleboard or the Raspberry Pi, it will be your only option.

In this chapter I will provide setup instructions for Ubuntu, Windows, and Mac OSX but will mainly focus on Ubuntu. The projects themselves in the later chapters are platform-independent.

Ubuntu

Download the OpenCV tarball from <http://sourceforge.net/projects/opencvlibrary/> and extract it to a preferred location (for subsequent steps I will refer to it as `OPENCV_DIR`). You can extract by using the Archive Manager or by issuing the `tar -xvf` command if you are comfortable with it.

Simple Install

This means you will install the current stable OpenCV version, with the default compilation flags, and support for only the standard libraries.

1. If you don't have the standard build tools, get them by

```
sudo apt-get install build-essential checkinstall cmake
```
2. Make a build directory in `OPENCV_DIR` and navigate to it by

```
mkdir build
cd build
```
3. Configure the OpenCV installation by

```
cmake ..
```


4. Compile the source code by

```
make
```

5. Finally, put the library files and header files in standard paths by

```
sudo make install
```

Customized Install (32-bit)

This means that you will install a number of supporting libraries and configure the OpenCV installation to take them into consideration. The extra libraries that we will install are:

- FFmpeg, gstreamer, x264 and v4l to enable video viewing, recording, streaming, and so on
 - Qt for a better GUI to view images
1. If you don't have the standard build tools, get them by

```
sudo apt-get install build-essential checkinstall cmake
```

2. Install gstreamer

```
sudo apt-get install libgstreamer0.10-0 libgstreamer0.10-dev gstreamer0.10-tools
gstreamer0.10-plugins-base libgstreamer-plugins-base0.10-dev gstreamer0.10-plugins-good
gstreamer0.10-plugins-ugly gstreamer0.10-plugins-bad gstreamer0.10-ffmpeg
```

3. Remove any installed versions of ffmpeg and x264

```
sudo apt-get remove ffmpeg x264 libx264-dev
```

4. Install dependencies for ffmpeg and x264

```
sudo apt-get update
sudo apt-get install git libfaac-dev libjack-jackd2-dev libmp3lame-dev
libopencore-amrnb-dev libopencore-amrwb-dev libsdl1.2-dev libtheora-dev
libva-dev libvdpau-dev libvorbis-dev libx11-dev libxfixes-dev libxvidcore-dev
texi2html yasm zlib1g-dev libjpeg8 libjpeg8-dev
```

5. Get a recent stable snapshot of x264 from

<ftp://ftp.videolan.org/pub/videolan/x264/snapshots/>, extract it to a folder on your computer and navigate into it. Then configure, build, and install by

```
./configure --enable-static
make
sudo make install
```

6. Get a recent stable snapshot of ffmpeg from <http://ffmpeg.org/download.html>, extract it to a folder on your computer and navigate into it. Then configure, build, and install by

```
./configure --enable-gpl --enable-libfaac --enable-libmp3lame
--enable-libopencore-amrnb --enable-libopencore-amrwb --enable-libtheora
--enable-libvorbis --enable-libx264 --enable-libxvid --enable-nonfree
--enable-postproc --enable-version3 --enable-x11grab
make
sudo make install
```

7. Get a recent stable snapshot of v4l from <http://www.linuxtv.org/downloads/v4l-utils/>, extract it to a folder on your computer and navigate into it. Then build and install by

```
make
sudo make install
```

8. Install `cmake-curses-gui`, a semi-graphical interface to CMake that will allow you to see and edit installation flags easily

```
sudo apt-get install cmake-curses-gui
```

9. Make a build directory in `OPENCV_DIR` by

```
mkdir build
cd build
```

10. Configure the OpenCV installation by

```
ccmake ..
```

11. Press 'c' to start configuring. CMake-GUI should do its thing, discovering all the libraries you installed above, and present you with a screen showing the installation flags (Figure 2-1).

```

samarth@samarth-inspiron-N5010: ~/Documents/opencv-2.4.5/build
Page 1 of 3
ANT_EXECUTABLE          ANT_EXECUTABLE-NOTFOUND
BUILD_DOCS              ON
BUILD_EXAMPLES         OFF
BUILD_JASPER            OFF
BUILD_JPEG              OFF
BUILD_OPENEXR           OFF
BUILD_PACKAGE          ON
BUILD_PERF_TESTS       ON
BUILD_PNG               OFF
BUILD_SHARED_LIBS      ON
BUILD_TBB               OFF
BUILD_TESTS            ON
BUILD_TIFF              OFF
BUILD_WITH_DEBUG_INFO  ON
BUILD_ZLIB              OFF
BUILD_opencv_apps      ON
BUILD_opencv_calib3d   ON
BUILD_opencv_contrib   ON
BUILD_opencv_core      ON
BUILD_opencv_features2d ON
BUILD_opencv_flann     ON
BUILD_opencv_gpu       ON
BUILD_opencv_highgui   ON
BUILD_opencv_imgproc   ON
BUILD_opencv_legacy    ON
BUILD_opencv_ml        ON
BUILD_opencv_nonfree   ON
BUILD_opencv_objdetect ON
BUILD_opencv_photo     ON
BUILD_opencv_stitching ON
BUILD_opencv_superres  ON
BUILD_opencv_ts        ON
BUILD_opencv_video     ON
BUILD_opencv_videostab ON

ANT_EXECUTABLE: Path to a program.
Press [enter] to edit option
Press [c] to configure
Press [h] for help          Press [q] to quit without generating
Press [t] to toggle advanced mode (Currently Off)
CMake Version 2.8.7

```

Figure 2-1. Configuration flags when you start installing OpenCV

12. You can navigate among the flags by the up and down arrows, and change the value of a flag by pressing the Return key. Change the following flags to the values shown in Table 2-1.

Table 2-1. Configuration flags for installing OpenCV with support for other common libraries

FLAG	VALUE
BUILD_DOCS	ON
BUILD_EXAMPLES	ON
INSTALL_C_EXAMPLES	ON
WITH_GSTREAMER	ON
WITH_JPEG	ON
WITH_PNG	ON
WITH_QT	ON
WITH_FFMPEG	ON
WITH_V4L	ON

- Press 'c' to configure and 'g' to generate, and then build and install by

```
make
sudo make install
```

- Tell Ubuntu where to find the OpenCV shared libraries by editing the file `opencv.conf` (first time users might not have that file—in that case, create it)

```
sudo gedit /etc/ld.so.conf.d/opencv.conf
```

- Add the line `"/usr/local/lib"` (without quotes) to this file, save and close. Bring these changes into effect by

```
sudo ldconfig /etc/ld.so.conf
```

- Similarly, edit `/etc/bash.bashrc` and add the following lines to the bottom of the file, save, and close:

```
PKG_CONFIG_PATH=$PKG_CONFIG_PATH:/usr/local/lib/pkgconfig
export PKG_CONFIG_PATH
```

Reboot your computer.

Customized Install (64-bit)

If you have the 64-bit version of Ubuntu, the process remains largely the same, except for the following changes.

- During the step 5 to configure `x264`, use this command instead:

```
./configure --enable-shared --enable-pic
```

- During the step 6 to configure `ffmpeg`, use this command instead:

```
./configure --enable-gpl --enable-libfaac --enable-libmp3lame
--enable-libopencore-amrnb --enable-libopencore-amrwb --enable-libtheora
--enable-libvorbis --enable-libx264 --enable-libxvid --enable-nonfree
--enable-postproc --enable-version3 --enable-x11grab --enable-shared --enable-pic
```

Checking the Installation

You can check the installation by putting the following code in a file called `hello_opencv.cpp`. It displays an image, and closes the window when you press “q”:

```
#include <iostream>
#include <opencv2/highgui/highgui.hpp>
using namespace std;
using namespace cv;

int main(int argc, char **argv)
{
    Mat im = imread("image.jpg", CV_LOAD_IMAGE_COLOR);
    namedWindow("Hello");
    imshow("Hello", im);

    cout << "Press 'q' to quit..." << endl;
    while(1)
    {
        if(char(waitKey(1)) == 'q') break;
    }
    destroyAllWindows();
    return 0;
}
```

1. Open up that directory in a Terminal and give the following command to compile the code:

```
g++ 'pkg-config opencv --cflags' hello_opencv.cpp -o hello_opencv 'pkg-config opencv --libs'
```

2. Run the compiled code by

```
./hello_opencv
```

Note that you need to have an image called “image.jpg” in the same directory for this program to run.

Installing Without Superuser Privileges

Many times you do not have superuser access privileges to the computer you are working on. You can still install and use OpenCV, if you tell Ubuntu where to look for the library and header files. In fact, this method of using OpenCV is recommended over the previous method, as it does not “pollute” the system directories with conflicting versions of OpenCV files according to the official OpenCV installation Wiki page. Note that installing extra libraries such as Qt, Ffmpeg, and so on will still require superuser privileges. But OpenCV will still work without these add-ons. The steps involved are:

1. Download the OpenCV tarball and extract it to a directory where you have read/write rights. We shall call this directory `OPENCV_DIR`. Make the following directories in `OPENCV_DIR`

```
mkdir build
cd build
mkdir install-files
```

2. Configure your install as mentioned previously. Change the values of flags depending on which extra libraries you have installed in the system. Also, set the value of `CMAKE_INSTALL_PREFIX` to `OPENCV_DIR/build/install-files`.

3. Continue the same making process as the normal install, up to step 12. Then, run `make install` instead of `sudo make install`. This will put all the necessary OpenCV files in `OPENCV_DIR/build/install-files`.
4. Now, edit the file `~/.bashrc` (your local `bashrc` file over which you should have read/write access) and add the following lines to the end of the file, then save and close

```
export INCLUDE_PATH=<path-to-OPENCV_DIR>/build/install-files/include:$INCLUDE_PATH
export LD_LIBRARY_PATH=<path-to-OPENCV_DIR>/build/install-files/lib:$LD_LIBRARY_PATH
export PKG_CONFIG_PATH=<path-to-OPENCV_DIR>/build/install-files/lib/pkgconfig:$PKG_CONFIG_PATH
```

where `<path-to-OPENCV_DIR>` can for example be `/home/user/libraries/opencv/`.

1. Reboot your computer.
2. You can now compile and use OpenCV code as mentioned previously, like a normal install.

Using an Integrated Development Environment

If you prefer to work in an IDE rather than a terminal, you will have to configure the IDE project to find your OpenCV library files and header files. For the widely used Code::Blocks IDE, very good instructions are available at <http://opencv.willowgarage.com/wiki/CodeBlocks>, and the steps should be pretty much the same for any other IDE.

Windows

Installation instructions for Windows users are available at <http://opencv.willowgarage.com/wiki/InstallGuide> and they work quite well. Instructions for integration with MS Visual C++ are available at <http://opencv.willowgarage.com/wiki/VisualC++>.

Mac OSX

Mac OSX users can install OpenCV on their computers by following instructions at http://opencv.willowgarage.com/wiki/Mac_OS_X_OpenCV_Port.

Summary

So you see how much more fun installing software in Linux than it is in Windows and Mac OS X! Jokes apart, going through this whole process will give valuable insight to beginners about the internal workings of Linux and the use of Terminal. If, even after following the instructions to the dot, you have problems installing OpenCV, Google your error. Chances are very high that someone else has had that problem, too, and they have asked a forum about it. There are also a number of websites and detailed videos on YouTube explaining the installation process for Linux, Windows, and Mac OS X.

CHAPTER 3



CV Bling—OpenCV Inbuilt Demos

Now that you (hopefully) have OpenCV installed on your computer, it is time to check out some cool demos of what OpenCV can do for you. Running these demos will also serve to confirm a proper install of OpenCV.

OpenCV ships with a bunch of demos. These are in the form of C, C++, and Python code files in the `samples` folder inside `OPENCV_DIR` (the directory in which you extracted the OpenCV archive while installing; see Chapter 2 for specifics). If you specified the flag `BUILD_EXAMPLES` to be `ON` while configuring your installation, the compiled executable files should be present ready for use in `OPENCV_DIR/build/bin`. If you did not do that, you can run your configuration and installation as described in Chapter 2 again with the flag turned on.

Let us take a look at some of the demos OpenCV has to offer. Note that you can run these demos by

```
./<demo_name> [options]
```

where `options` is a set of command line arguments that the program expects, which is usually the file name. The demos shown below have been run on images that ship with OpenCV, which can be found in `OPENCV_DIR/samples/cpp`.

Note that all the commands mentioned below are executed after navigating to `OPENCV_DIR/build/bin`.

Camshift

Camshift is a simple object tracking algorithm. It uses the intensity and color histogram of a specified object to find an instance of the object in another image. The OpenCV demo first requires you to draw a box around the desired object in the camera feed. It makes the required histogram from the contents of this box and then proceeds to use the camshift algorithm to track the object in the camera feed. Run the demo by navigating to `OPENCV_DIR/build/bin` and doing

```
./cpp-example-camshiftdemo
```

However, camshift *always* tries to find an instance of the object. If the object is not present, it shows the nearest match as a detection (see Figure 3-4).

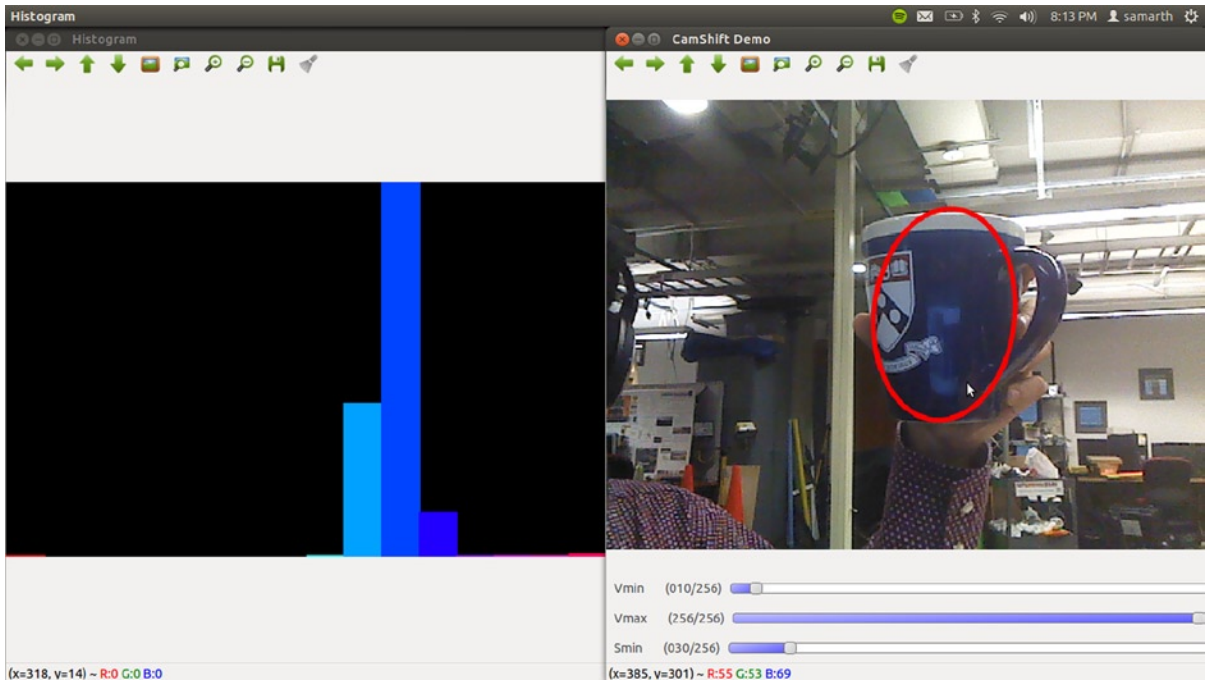


Figure 3-1. Camshift object tracking—specifying the object to be tracked

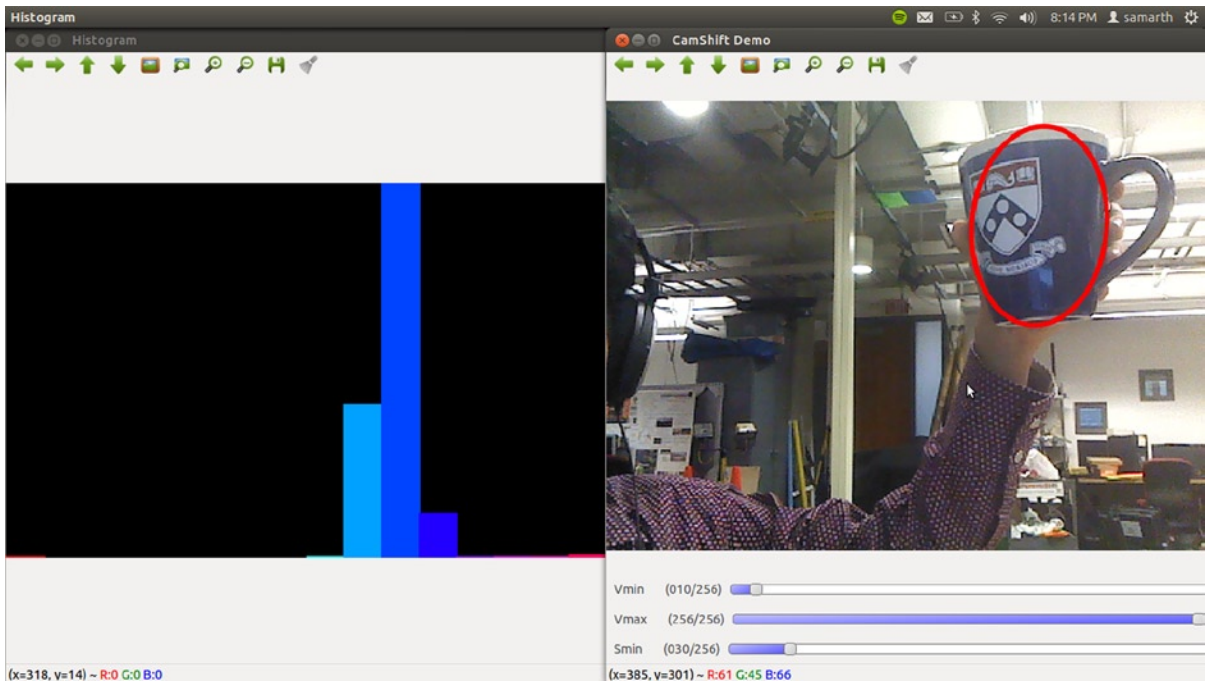


Figure 3-2. Camshift object tracking

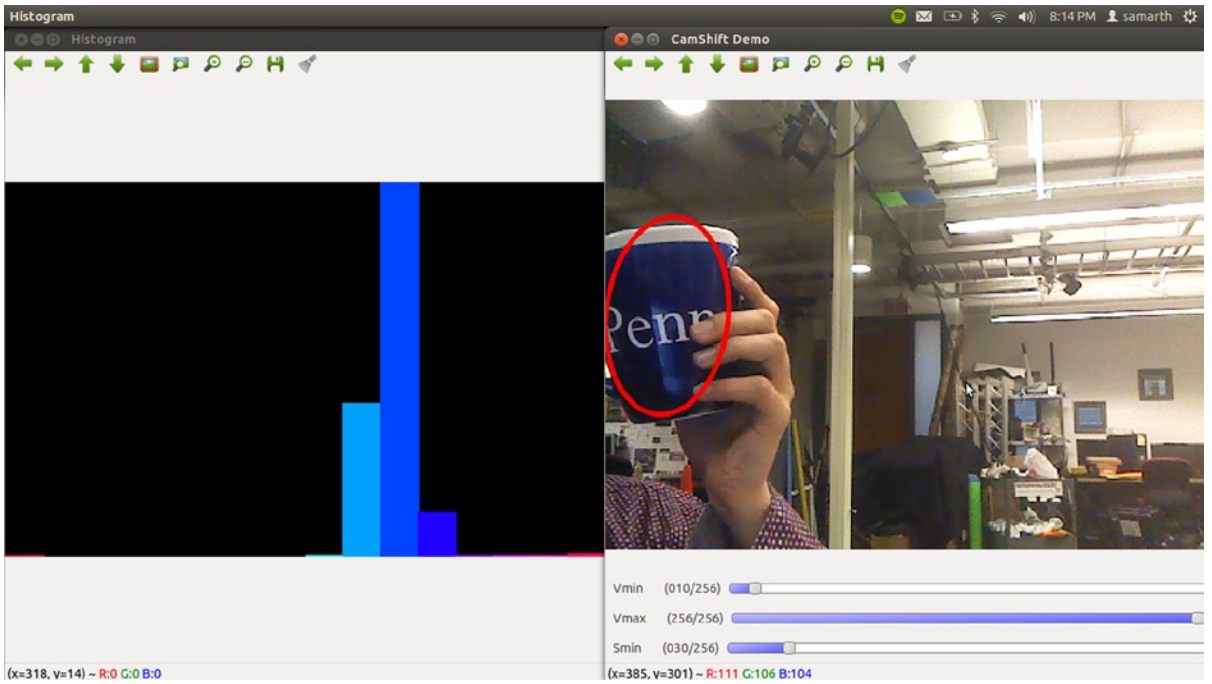


Figure 3-3. Camshift object tracking

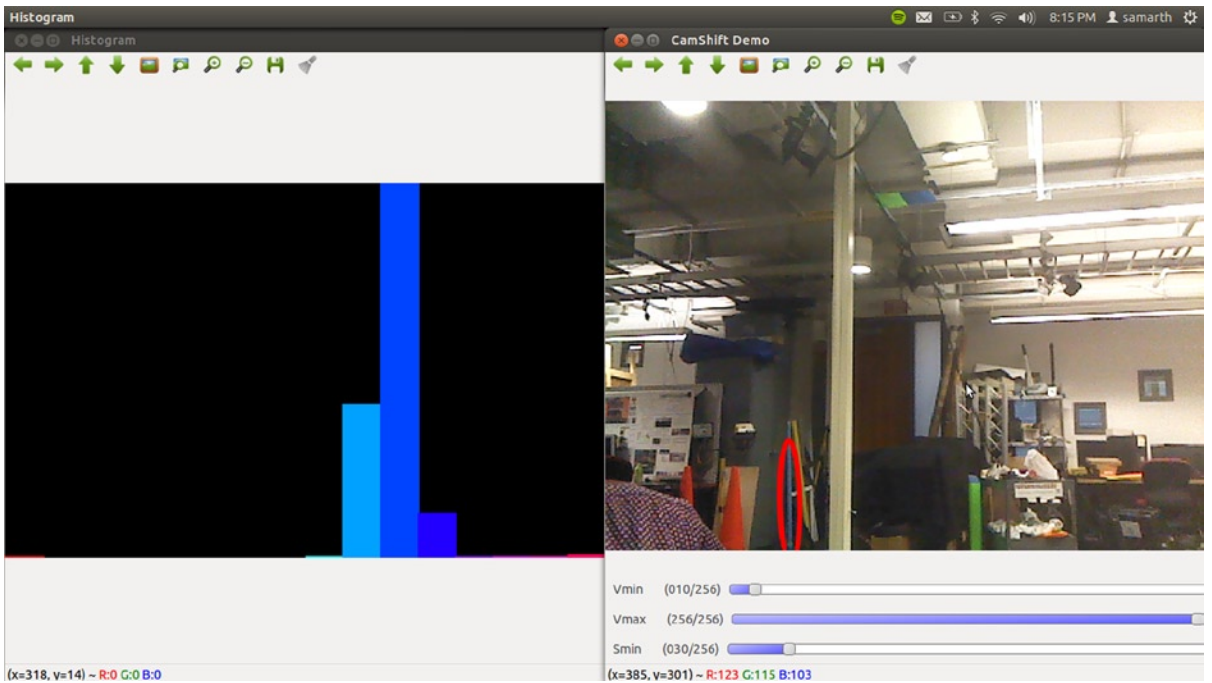


Figure 3-4. Camshift giving a false positive

Stereo Matching

The `stereo_matching` demo showcases the stereo block matching and disparity calculation abilities of OpenCV. It takes two images (taken with a left and right stereo camera) as input and produces an image in which the disparity is grey color-coded. I will devote an entire chapter to stereo vision later on in the book. Meanwhile, a short explanation of disparity: when you see an object using two cameras (left and right), it will appear to be at slightly different horizontal positions in the two images. The difference of the position of the object in the right frame with respect to the left frame is called disparity. Disparity can give an idea about the depth of the object, that is, its distance from the cameras, because disparity is inversely proportional to distance. In the output image, pixels with higher disparity are lighter. (Recall that higher disparity means lesser distance from the camera.) You can run the demo on the famous Tsukuba images by

```
./cpp-example-stereo_match OPENCV_DIR/samples/cpp/tsukuba_l.png OPENCV_DIR/samples/cpp/tsukuba_r.png
```

where `OPENCV_DIR` is the path to `OPENCV_DIR`

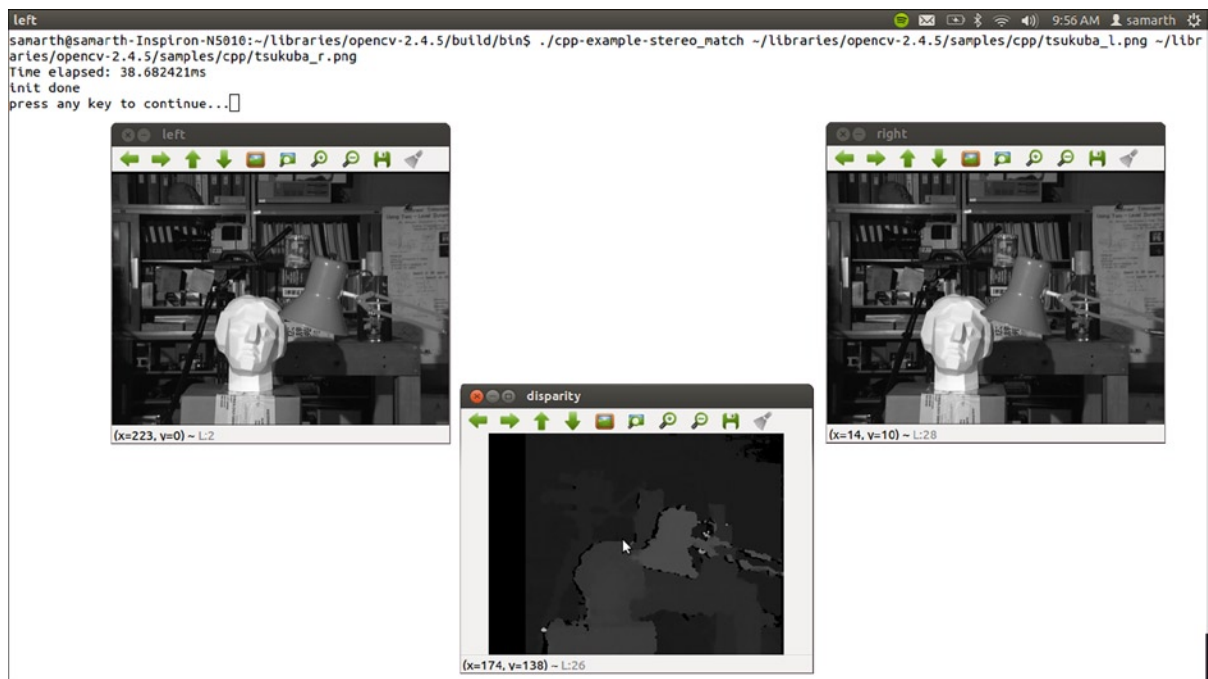


Figure 3-5. OpenCV stereo matching

Homography Estimation in Video

The `video_homography` demo uses the FAST corner detector to detect interest points in the image and matches BRIEF descriptors evaluated at the keypoints. It does so for a “reference” frame and any other frame to estimate the homography transform between the two images. A homography is simply a matrix that transforms points from one plane to another. In this demo, you can choose your reference frame from the camera feed. The demo draws lines in the direction of the homography transform between the reference frame and the current frame. You can run it by

```
./cpp-example-video_homography 0
```

where `0` is the device ID of the camera. `0` usually means the laptop's integrated webcam.

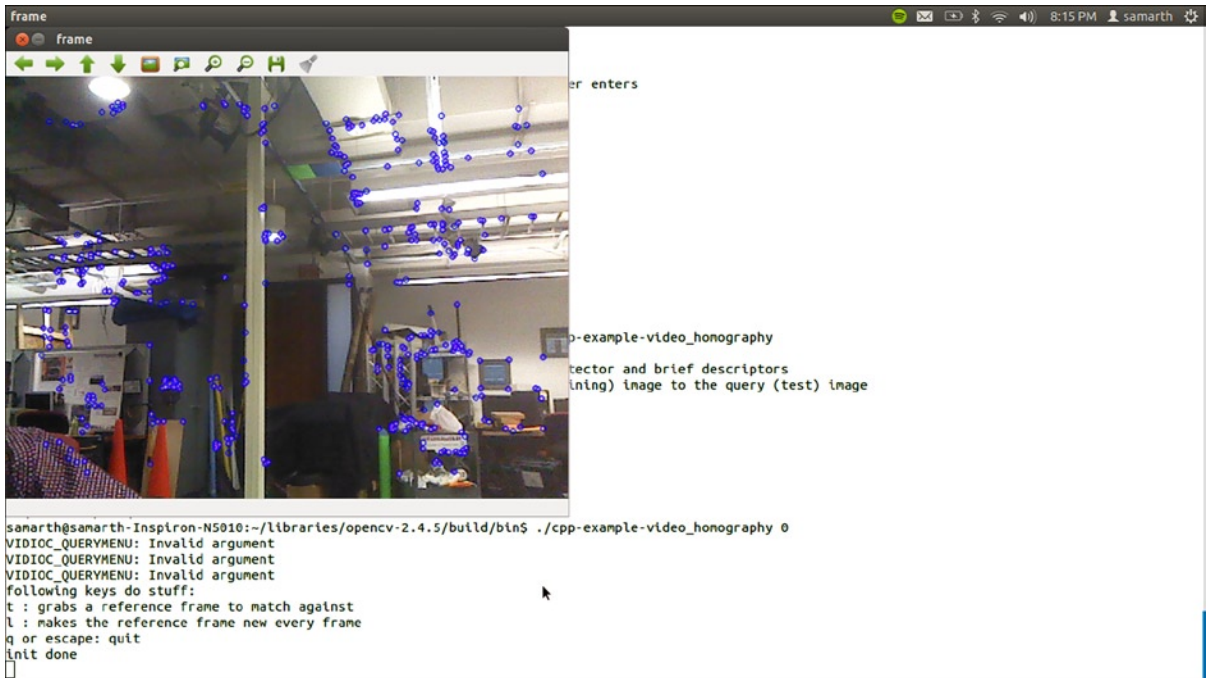


Figure 3-6. The reference frame for homography estimation, also showing FAST corners

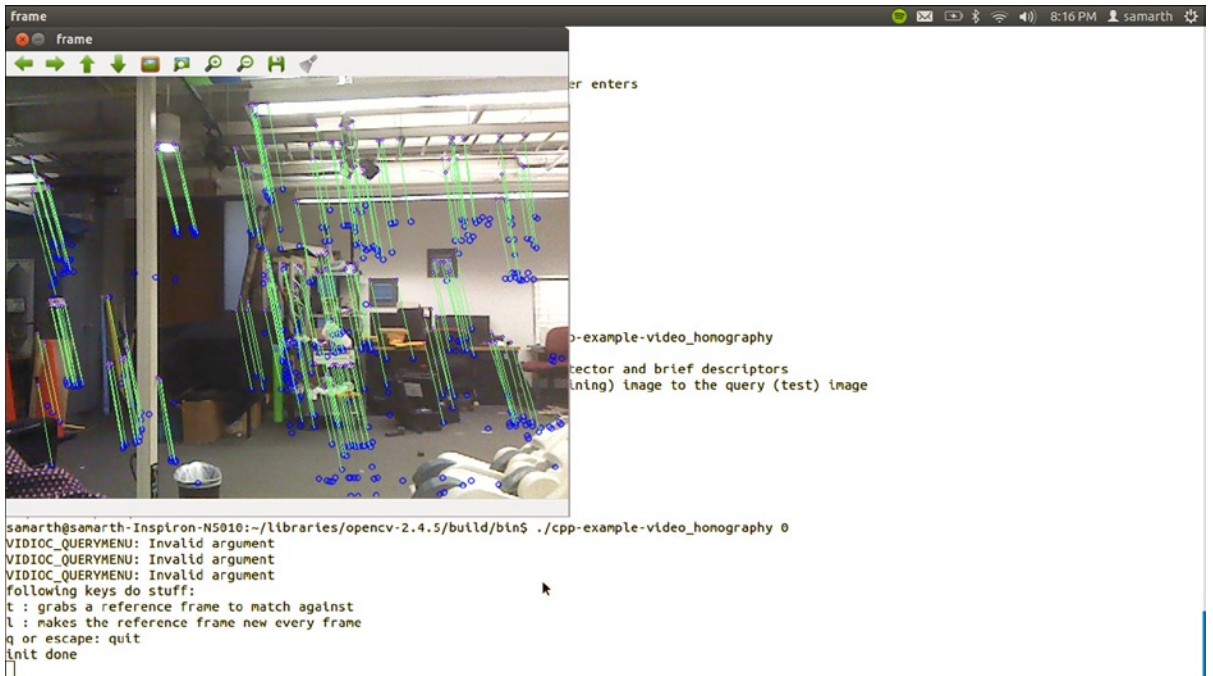


Figure 3-7. Estimated homography shown by lines

Circle and Line Detection

The `houghcircles` and `houghlines` demos in OpenCV detect circles and lines respectively in a given image using the Hough transform. I shall have more to say on Hough transforms in Chapter 6. For now, just know that the Hough transform is a very useful tool that allows you to detect regular shapes in images. You can run the demos by

```
./cpp-example-houghcircles OPENCV_DIR/samples/cpp/board.jpg
```

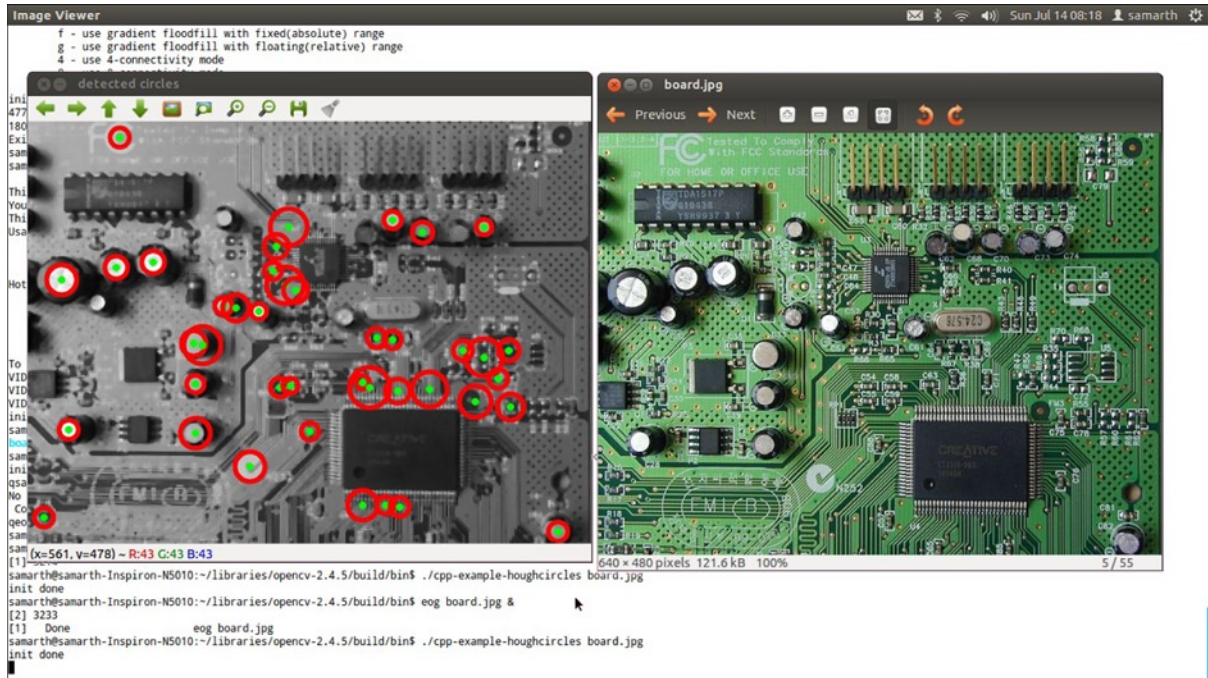


Figure 3-8. Circle detection using Hough transform

and

```
./cpp-example-houghlines OPENCV_DIR/samples/cpp/pic1.png
```



```

source
Usage:
./canshiftdemo [camera number]

Hot keys:
ESC - quit the program
c - stop the tracking
b - switch to/from backprojection view
h - show/hide object histogram
p - pause video

To initialize tracking, select t
VIDIOC_QUERYMENU: Invalid argume
VIDIOC_QUERYMENU: Invalid argume
VIDIOC_QUERYMENU: Invalid argume
lnit done
samarth@samarth-Inspiron-N5010:~
This program demonstrated the use
to track planar objects by compu

usage: ./cpp-example-video_honogr
The following keys do stuff:
t : grabs a reference frame to
l : makes the reference frame
q or escape: quit
samarth@samarth-Inspiron-N5010:~
VIDIOC_QUERYMENU: Invalid argume
VIDIOC_QUERYMENU: Invalid argume
VIDIOC_QUERYMENU: Invalid argume
following keys do stuff:
t : grabs a reference frame to match against
l : makes the reference frame new every frame
q or escape: quit
lnit done
samarth@samarth-Inspiron-N5010:~/libraries/opencv-2.4.5/build/bin$ ./cpp-example-houghcircles ~/libraries/opencv-2.4.5/samples/cpp/board.jpg
lnit done
samarth@samarth-Inspiron-N5010:~/libraries/opencv-2.4.5/build/bin$ ./cpp-example-houghlines ~/libraries/opencv-2.4.5/samples/cpp/pic1.png
lnit done

```

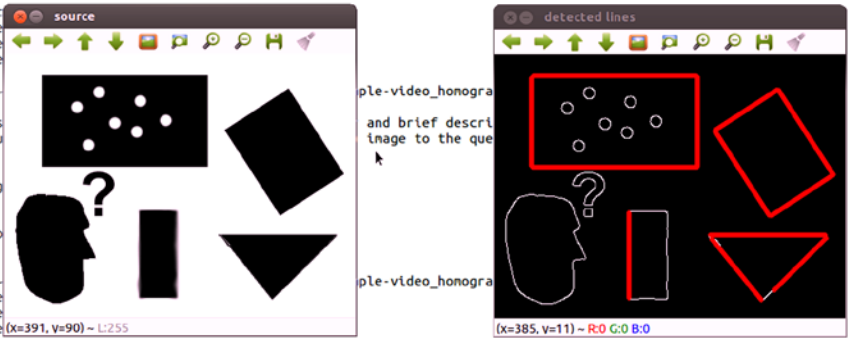


Figure 3-9. Line detection using Hough transform

Image Segmentation

The `meanshift_segmentation` demo implements the meanshift algorithm for image segmentation (distinguishing different “parts” of the image). It also allows you to set various thresholds associated with the algorithm. Run it by

```
./cpp-example-meanshift_segmentation OPENCV_DIR/samples/cpp/tsukuba_1.png
```

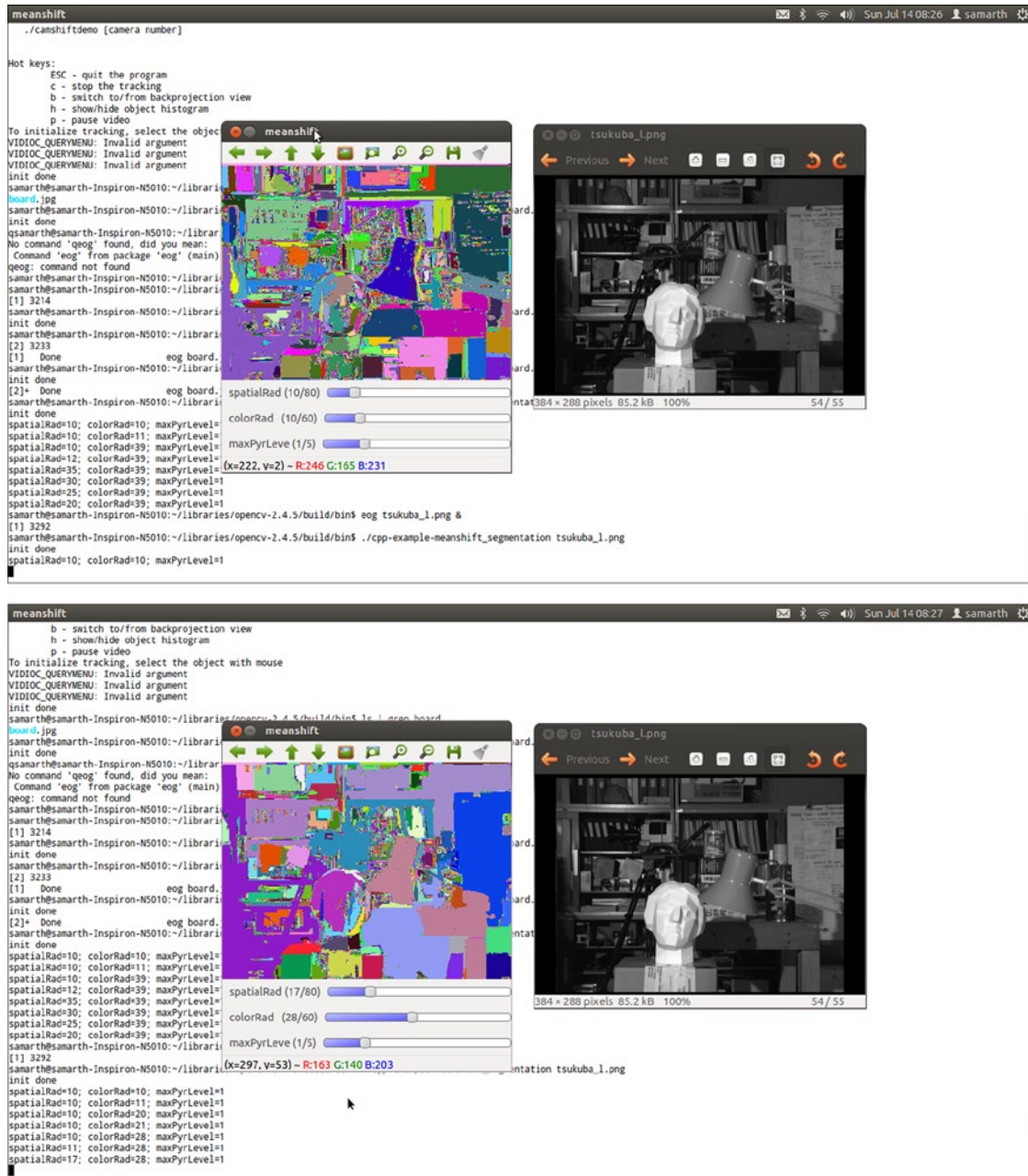


Figure 3-10. Image segmentation using the meanshift algorithm

As you can see, various regions in the image are colored differently.

Bounding Box and Circle

The `minarea` demo finds the smallest rectangle and circle enclosing a set of points. In the demo, the points are selected from within the image area randomly.

```
./cpp-example-minarea
```

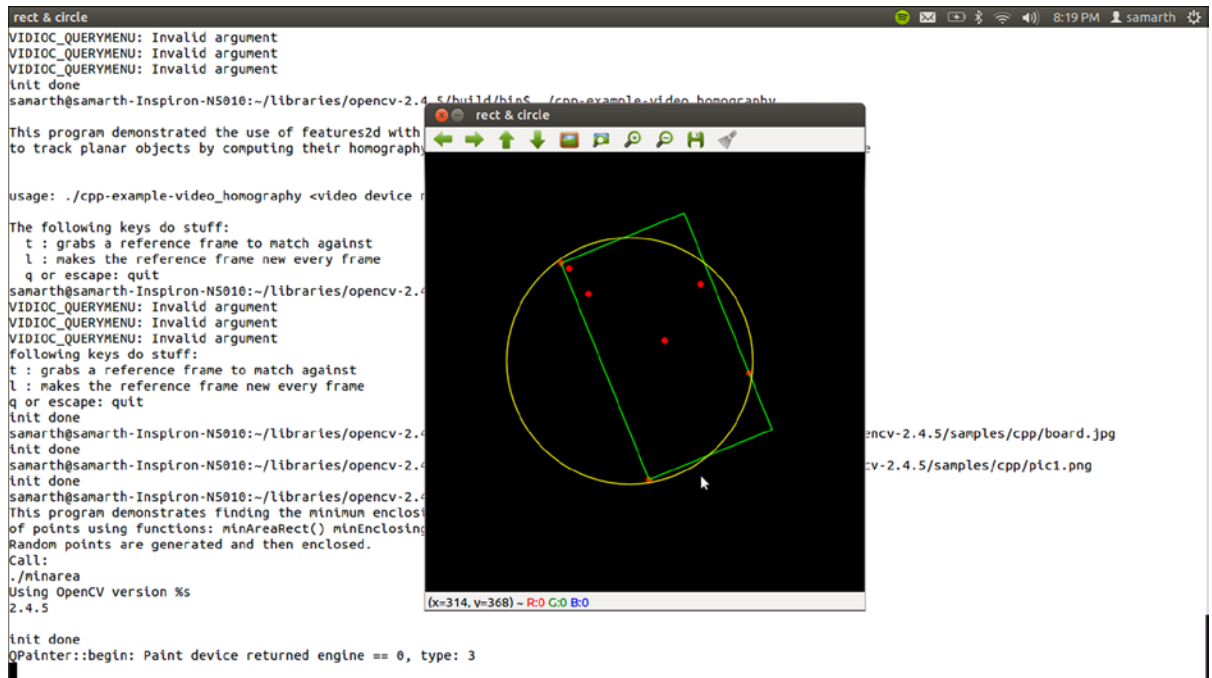


Figure 3-11. Bounding box and circle

Image Inpainting

Image inpainting is replacing certain pixels in the image with surrounding pixels. It is mainly used to repair damages to images such as accidental brush-strokes. The OpenCV `inpaint` demo allows you to vandalize an image by making white marks on it and then runs the inpainting algorithm to repair the damages.

```
./cpp-example-inpaint OPENCV_DIR/samples/cpp/fruits.jpg
```