CHAPTER 11

Embedded Computer Vision: Running OpenCV Programs on the Raspberry Pi

Embedded computer vision is a very practical branch of computer vision that concerns itself with developing or modifying vision algorithms to run on embedded systems—small mobile computers like smartphone processors or hobby boards. The two main considerations in an embedded computer vision system are judicious use of processing power and low battery consumption. As an example of an embedded computing system, we will consider the Raspberry Pi (Figure 11-1), an ARM processor–based small open-source computer that is rapidly gaining popularity among hobbyists and researchers alike for its ease of use, versatile capabilities, and surprisingly low cost in spite of good build and support quality.

Figure 11-1. *The Raspberry Pi board*

Raspberry Pi

The Raspberry Pi is a small computer that can run a Linux-based operating system from a SD memory card. It has a 700 MHz ARM processor and a small Broadcom VideoCore IV 250 MHz GPU. The CPU and GPU share 512 MB of SDRAM memory, and you can change the sharing of memory between each according to your use pattern. As shown in Figure 11-1, the Pi has one Ethernet, one HDMI, two USB 2.0 ports, 8 general-purpose input/output pins, and a UART to interact with other devices. A 5 MP camera board has also been recently released to promote the use of Pi in small-scale computer vision applications. This camera board has its own special parallel connector to the board; hence, it is expected to support higher frame-rates than a web camera attached to one of the USB ports. The Pi is quite power-efficient, too—it can run off a powered USB port of your computer or your iPhone's USB wall charger!

Setting Up Your New Raspberry Pi

Because the Pi is just a processor with various communication ports and pins on it, if you are planning to buy one make sure you order it with all the required accessories. These mainly include connector cables:

- Ethernet cable for connection to the Internet
- USB-A to USB-B converter cable for power supply from a powered USB port
- HDMI cable for connection to a monitor
- Camera module (optional)
- USB expander hub if you plan to connect more than two USB devices to the Pi
- SD memory card with a capacity of at least 4 GB for installing the OS and other programs

Although several Linux-based operating systems can be installed on the Pi, Raspbian (latest version "wheezy" as of writing), which is a flavor of Debian optimized for the Pi, is recommended for beginners. The rest of this chapter will assume that you installed Raspbian on your Pi, because it works nicely out-of-the-box and has a large amount community support. Once you install the OS and connect a monitor, keyboard, and mouse to the Pi, it becomes a fully functional computer! A typical GUI-based setup is shown in Figure 11-2.

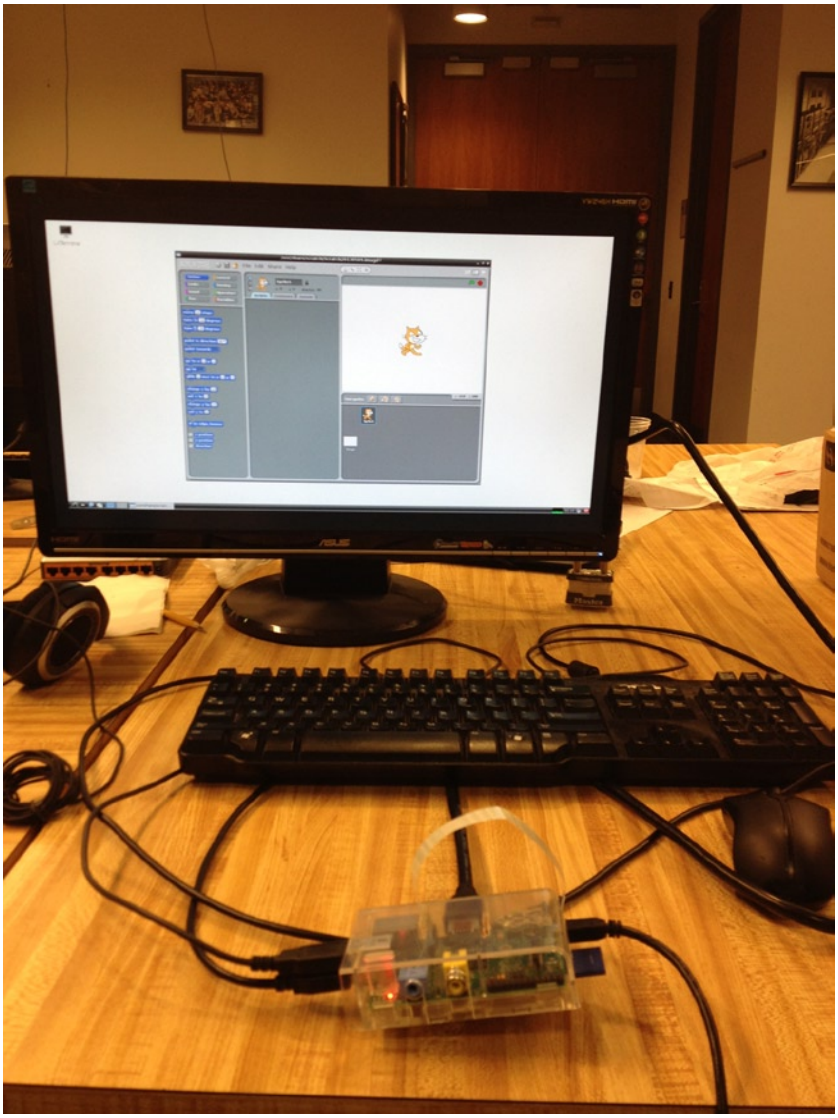

Figure 11-2. *Raspberry Pi connected to keyboard, mouse, and monitor—a complete computer!*

Installing Raspbian on the Pi

You will need a blank SD card with a capacity of at least 4 GB and access to a computer for this straightforward process. If you are a first-time user, it is recommended that you use the specially packaged New Out Of Box Software (NOOBS) at `www.raspberrypi.org/downloads`. Detailed instructions for using it can be found at the quick-start guide at `www.raspberrypi.org/wp-content/uploads/2012/04/quick-start-guide-v2_1.pdf`. In essence, you:

- Download the NOOBS software to your computer
- Unzip it on the SD card

- Insert the SD card into the Pi and boot it up with a keyboard and monitor attached to USB and HDMI, respectively
- Follow the simple on-screen instructions, making sure that you choose Raspbian as the operating system that you want to install

Note that this creates a user account "pi" with the password "raspberry." This account also has sudo access with the same password.

Initial Settings

After you install Raspbian, reboot and hold "Shift" to enter the `raspi-config` settings screen, which is shown in Figure 11-3.

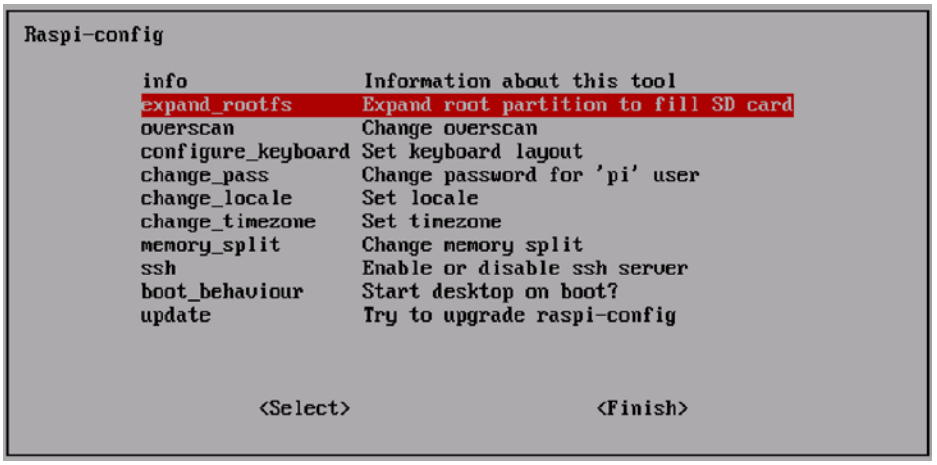

Figure 11-3. *The `raspi-config` screen*

- First, you should use the `'expand_rootfs'` option to enable the Pi to use your entire memory card.
- You can also change the RAM memory sharing settings using the `'memory_split'` option. My recommendation is to allocate as much as possible to the CPU, because OpenCV does not support the VideoCore GPU yet, so we will end up using the ARM CPU for all our computations.
- You can also use the `'boot_behavior'` option in `raspi-config` specify whether the Pi boots up into the GUI or a command line. Obviously, maintaining the GUI takes up processing power, so I recommend that you get comfortable with the Linux terminal and switch off the GUI. The typical method of access to a Pi that has been set to boot into the command line is by SSH-ing into it with X-forwarding. This essentially means that you connect to the Pi using Ethernet and have access to the Pi's terminal from a terminal on your computer. The X-forwarding means that the Pi will your computer's screen to render any windows (for example, OpenCV `imshow()` windows). Adafruit has a great guide for first-time SSH users at `http://learn.adafruit.com/downloads/pdf/adafruits-raspberry-pi-lesson-6-using-ssh.pdf`.

Installing OpenCV

Because Raspbian is a Linux flavor with Debian as its package-management system, the process to install OpenCV remains exactly the same as that for installing on 64-bit systems, as outlined in Chapter 2. Once you install OpenCV you should be able to run the demo programs just as you were able to on your computer. The demos that require live feed from a camera will also work if you attach a USB web camera to the Pi.

Figure 11-4 shows the OpenCV video homography estimation demo (`cpp-example-video_homography`) running from a USB camera on the Pi. As you might recall, this function tracks corners in frames and then finds a transformation (shown by the green lines) with reference to a frame that the user can choose. If you will run this demo yourself, you will observe that the 700 MHz processor in the Pi is quite inadequate to process 640x480 frames in such demos—the frame rate is very low.

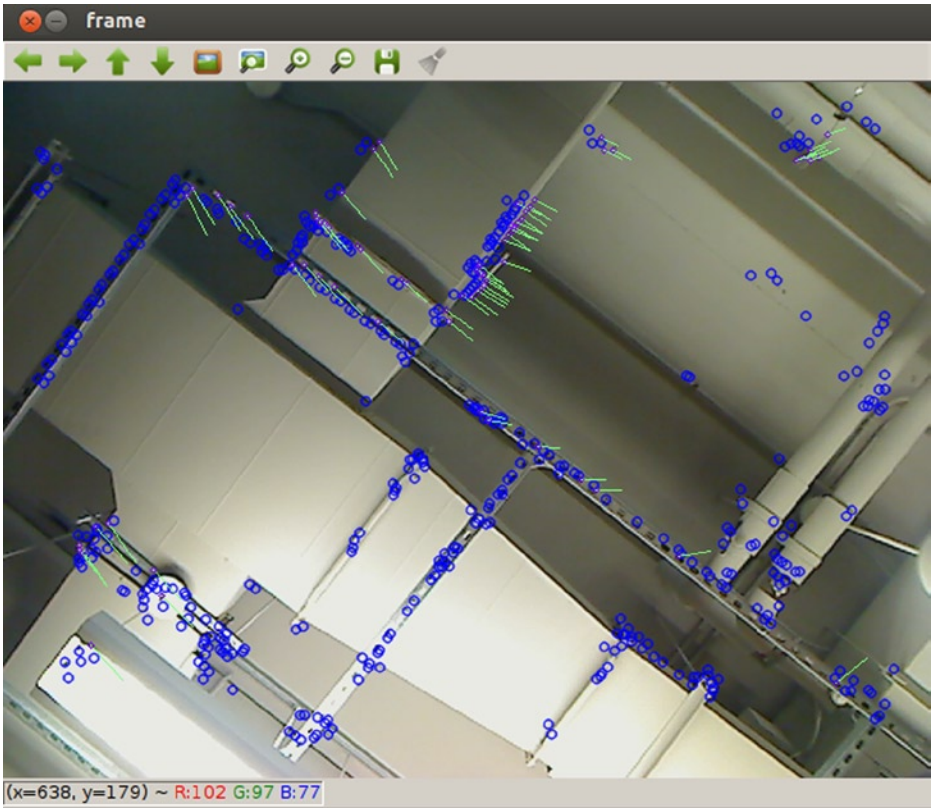

Figure 11-4. *OpenCV built-in video homography demo being run on the Raspberry Pi*

Figure 11-5 shows the OpenCV convex hull demo (`cpp-example-convexhull`) that chooses a random set of points and computes the smallest convex hull around them being executed on the Pi.

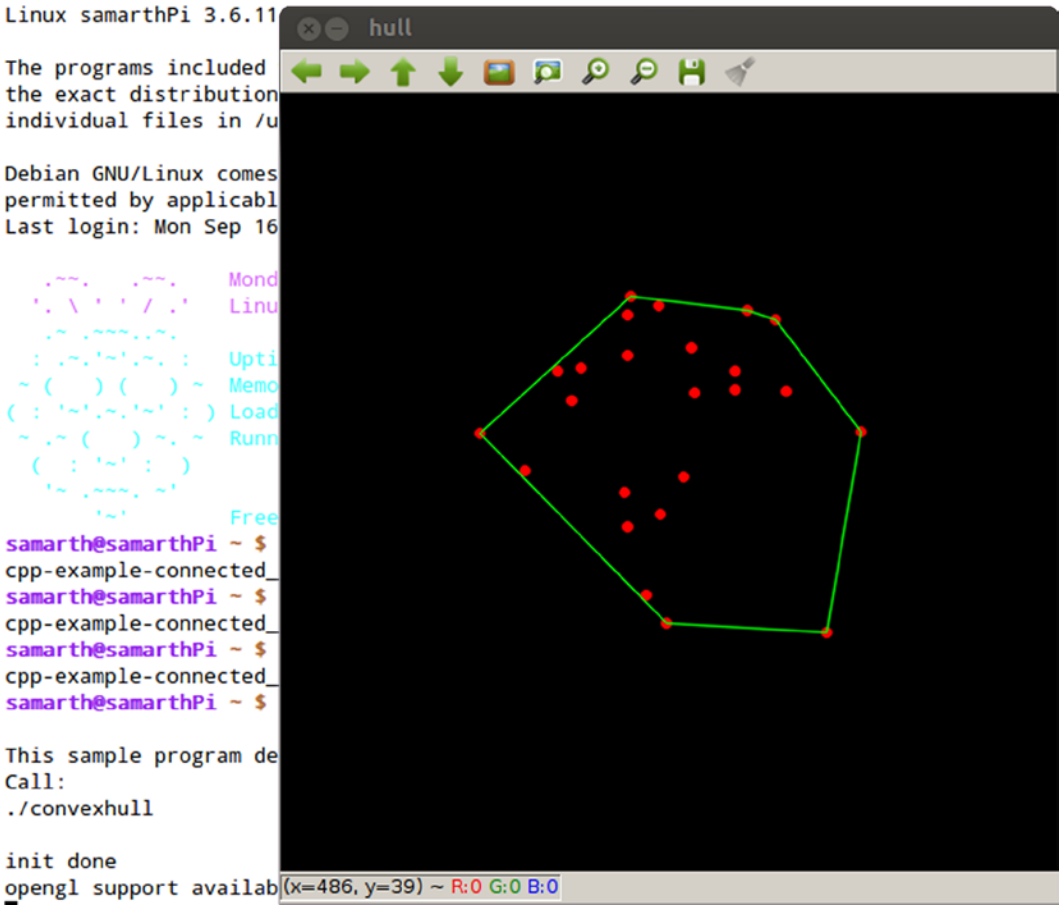

Figure 11-5. *OpenCV convex hull demo being run on the Pi*

Camera board

The makers of the Raspberry Pi also released a custom camera board recently to encourage image processing and computer vision applications of the Pi. The board is small and weighs just 3 grams, but it boasts a 5 Mega pixel CMOS sensor. It connects to the Pi using a ribbon cable and looks like Figure 11-6.

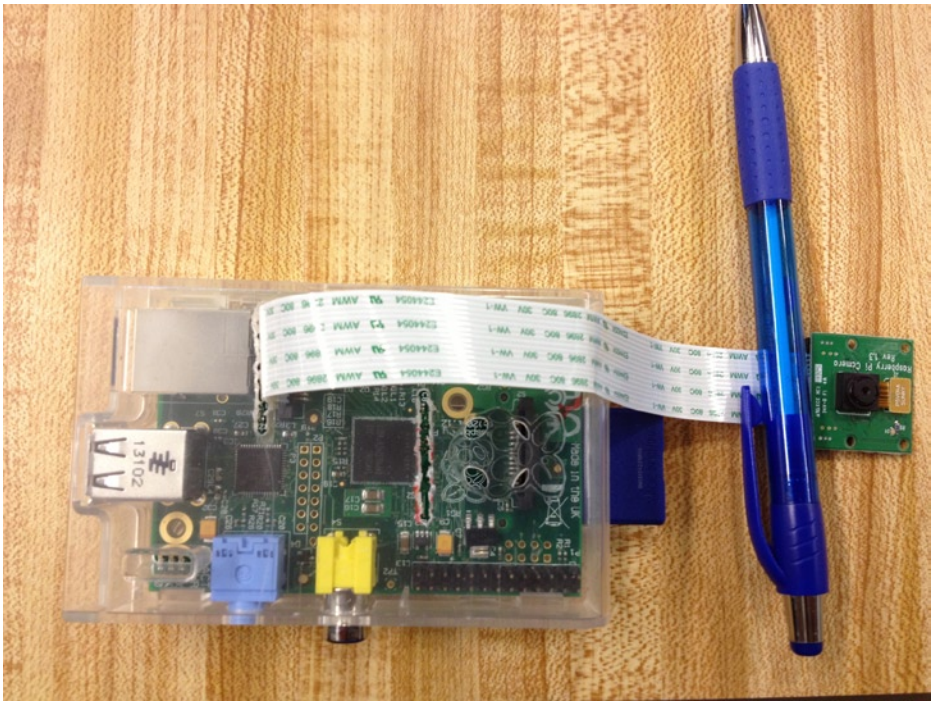

Figure 11-6. *The Raspberry Pi with the camera board attached to it*

Once you connect it by following the instructional video at `www.raspberrypi.org/camera`, you need to enable it from the `raspi-config` screen (which can be brought up by pressing and holding "Shift" at boot-up or typing `sudo raspi-config` at a terminal). The preinstalled utilities `raspistill` and `raspivid` can be used to capture still images and videos, respectively. They offer a lot of capture options that you can tweak. You can learn more about them by going through the documentation at `https://github.com/raspberrypi/userland/blob/master/host_applications/linux/apps/raspicam/RaspiCamDocs.odt`. The source code for these applications is open source, so we will see how to use it to get the board to interact with OpenCV.

Camera Board vs. USB Camera

You might ask, why use a camera board when you can use a simple USB camera like the ones you use on desktop computers? Two reasons:

1. The Raspberry Pi camera board connects to the Pi using a special connector that is designed to transfer data in parallel. This makes is faster than a USB camera
2. The camera board has a better sensor than a lot of other USB cameras that cost the same (or even more)

The only concern is that because the board is not a USB device, OpenCV does not recognize it out-of-the-box. I made a small C++ wrapper program that grabs the bytes from the camera buffer and puts them into an OpenCV `Mat` by reusing some C code that I found on Raspberry Pi forums. This code makes use of the open-sourced code released by developers of the camera drivers. This program allows you to specify the size of the frame that you want to capture and a boolean flag that indicates whether the captured frame should be colored or grayscale. The camera returns

image information in the form of the Y channel and subsampled U and V channels of the (YUV color space). To get a grayscale image, the wrapper program just has to set the Y channel as the data source of an OpenCV `Mat`. Things get complicated (and slow) if you want color. To get a color image, the following steps have to be executed by the wrapper program:

- Copy the Y, U, and V channels into OpenCV Mats
- Resize the U and V channels because the U and V returned by the camera are subsampled
- Perform a YUV to RGB conversion

These operations are time-consuming, and hence frame rates drop if you want to stream color images from the camera board.

By contrast, a USB camera captures in color by default, and you will have to spend extra time converting it to grayscale. Hence, unless you know how to set the capture mode of the camera to grayscale, grayscale images from USB cameras are costlier than color images.

As we have seen so far, most computer vision applications operate on intensity and, hence, just require a grayscale image. This is one more reason the use the camera board than a USB camera. Let us see how to use the wrapper code to grab frames from the Pi camera board. Note that this process requires OpenCV to be installed on the Pi.

- If you enabled your camera from `raspi-config`, you should already have the source code and libraries for interfacing with the camera board in `/opt/vc/`. Confirm this by seeing if the commands `raspistill` and `raspivid` work as expected.
- Clone the official MMAL Github repository to get the required header files by navigating to any directory in your home folder in a terminal and running the following command. Note that this directory will be referenced as USERLAND_DIR in the CMakeLists.txt file in Listing 11-1, so make sure that you replace USERLAND_DIR in that file with the full path to this directory. If you do not have Git installed on your Raspberry Pi you can install it by typing `sudo apt-get install git` in a terminal.)

  ```
  git clone https://github.com/raspberrypi/userland.git
  ```

- The wrapper program will work with the CMake build environment. In a separate folder (called DIR further on) make folders called “src” and “include.” Put Picam.cpp and cap.h files from code 10-1 into the “src” and “include” folders, respectively. These constitute the wrapper code
- To use the wrapper code, make a simple file like main.cpp shown in code 10-1 to grab frames from the camera board, show them in a window and measure the frame-rate. Put the file in the “src” folder
- The idea is to make an executable that uses functions and classes from both the main.cpp and PiCapture.cpp files. This can be done by using a CMakeLists.txt file like the one shown in code 10-1 and saving it in DIR
- To compile and build the executable, run in DIR

  ```
  mkdir build
  cd build
  cmake ..
  make
  ```

A little explanation about the wrapper code follows. As you might have realized, the wrapper code makes a class called `PiCapture` with a constructor that takes in the width, height, and boolean flag (true means color images are grabbed). The class also defines a method called `grab()` that returns an OpenCV `Mat` containing the grabbed image of the appropriate size and type.

Listing 11-1. Simple CMake project illustrating the wrapper code to capture frames from the Raspberry Pi camera board

main.cpp:

```
//Code to check the OpenCV installation on Raspberry Pi and measure frame rate
//Author: Samarth Manoj Brahmbhatt, University of Pennsyalvania

#include "cap.h"
#include <opencv2/opencv.hpp>
#include <opencv2/highgui/highgui.hpp>

using namespace cv;
using namespace std;

int main() {
    namedWindow("Hello");

    PiCapture cap(320, 240, false);

    Mat im;
    double time = 0;
    unsigned int frames = 0;
    while(char(waitKey(1)) != 'q') {
        double t0 = getTickCount();
        im = cap.grab();
        frames++;
        if(!im.empty()) imshow("Hello", im);
        else cout << "Frame dropped" << endl;

        time += (getTickCount() - t0) / getTickFrequency();
        cout << frames / time << " fps" << endl;
    }

    return 0;
}
```

cap.h:

```
#include <opencv2/opencv.hpp>
#include "interface/mmal/mmal.h"
#include "interface/mmal/util/mmal_default_components.h"
#include "interface/mmal/util/mmal_connection.h"
#include "interface/mmal/util/mmal_util.h"
#include "interface/mmal/util/mmal_util_params.h"

class PiCapture {
    private:
        MMAL_COMPONENT_T *camera;
        MMAL_COMPONENT_T *preview;
        MMAL_ES_FORMAT_T *format;
        MMAL_STATUS_T status;
        MMAL_PORT_T *camera_preview_port, *camera_video_port, *camera_still_port;
        MMAL_PORT_T *preview_input_port;
```

```
        MMAL_CONNECTION_T *camera_preview_connection;
        bool color;
    public:
        static cv::Mat image;
        static int width, height;
        static MMAL_POOL_T *camera_video_port_pool;
        static void set_image(cv::Mat _image) {image = _image;}
        PiCapture(int, int, bool);
        cv::Mat grab() {return image;}
};

static void color_callback(MMAL_PORT_T *, MMAL_BUFFER_HEADER_T *);
static void gray_callback(MMAL_PORT_T *, MMAL_BUFFER_HEADER_T *);
```

PiCapture.cpp:

```
/*
 * File:   opencv_demo.c
 * Author: Tasanakorn
 *
 * Created on May 22, 2013, 1:52 PM
 */

// OpenCV 2.x C++ wrapper written by Samarth Manoj Brahmbhatt, University of Pennsylvania

#include <stdio.h>
#include <stdlib.h>

#include <opencv2/opencv.hpp>

#include "bcm_host.h"

#include "interface/mmal/mmal.h"
#include "interface/mmal/util/mmal_default_components.h"
#include "interface/mmal/util/mmal_connection.h"
#include "interface/mmal/util/mmal_util.h"
#include "interface/mmal/util/mmal_util_params.h"

#include "cap.h"

#define MMAL_CAMERA_PREVIEW_PORT 0
#define MMAL_CAMERA_VIDEO_PORT 1
#define MMAL_CAMERA_CAPTURE_PORT 2

using namespace cv;
using namespace std;

int PiCapture::width = 0;
int PiCapture::height = 0;
MMAL_POOL_T * PiCapture::camera_video_port_pool = NULL;
Mat PiCapture::image = Mat();
```

```
static void color_callback(MMAL_PORT_T *port, MMAL_BUFFER_HEADER_T *buffer) {
    MMAL_BUFFER_HEADER_T *new_buffer;

    mmal_buffer_header_mem_lock(buffer);
    unsigned char* pointer = (unsigned char *)(buffer -> data);
    int w = PiCapture::width, h = PiCapture::height;
    Mat y(h, w, CV_8UC1, pointer);
    pointer = pointer + (h*w);
    Mat u(h/2, w/2, CV_8UC1, pointer);
    pointer = pointer + (h*w/4);
    Mat v(h/2, w/2, CV_8UC1, pointer);
    mmal_buffer_header_mem_unlock(buffer);
    mmal_buffer_header_release(buffer);

    if (port->is_enabled) {
        MMAL_STATUS_T status;

        new_buffer = mmal_queue_get(PiCapture::camera_video_port_pool->queue);

        if (new_buffer)
            status = mmal_port_send_buffer(port, new_buffer);

        if (!new_buffer || status != MMAL_SUCCESS)
            printf("Unable to return a buffer to the video port\n");
    }

    Mat image(h, w, CV_8UC3);

    resize(u, u, Size(), 2, 2, INTER_LINEAR);
    resize(v, v, Size(), 2, 2, INTER_LINEAR);
    int from_to[] = {0, 0};
    mixChannels(&y, 1, &image, 1, from_to, 1);
    from_to[1] = 1;
    mixChannels(&v, 1, &image, 1, from_to, 1);
    from_to[1] = 2;
    mixChannels(&u, 1, &image, 1, from_to, 1);
    cvtColor(image, image, CV_YCrCb2BGR);

    PiCapture::set_image(image);
}

static void gray_callback(MMAL_PORT_T *port, MMAL_BUFFER_HEADER_T *buffer) {
    MMAL_BUFFER_HEADER_T *new_buffer;

    mmal_buffer_header_mem_lock(buffer);
    unsigned char* pointer = (unsigned char *)(buffer -> data);
    PiCapture::set_image(Mat(PiCapture::height, PiCapture::width, CV_8UC1, pointer));
    mmal_buffer_header_release(buffer);
```

```
    if (port->is_enabled) {
        MMAL_STATUS_T status;

        new_buffer = mmal_queue_get(PiCapture::camera_video_port_pool->queue);

        if (new_buffer)
            status = mmal_port_send_buffer(port, new_buffer);

        if (!new_buffer || status != MMAL_SUCCESS)
            printf("Unable to return a buffer to the video port\n");
    }
}

PiCapture::PiCapture(int _w, int _h, bool _color) {
    color = _color;
    width = _w;
    height = _h;

    camera = 0;
    preview = 0;
    camera_preview_port = NULL;
    camera_video_port = NULL;
    camera_still_port = NULL;
    preview_input_port = NULL;
    camera_preview_connection = 0;

    bcm_host_init();

    status = mmal_component_create(MMAL_COMPONENT_DEFAULT_CAMERA, &camera);
    if (status != MMAL_SUCCESS) {
        printf("Error: create camera %x\n", status);
    }

    camera_preview_port = camera->output[MMAL_CAMERA_PREVIEW_PORT];
    camera_video_port = camera->output[MMAL_CAMERA_VIDEO_PORT];
    camera_still_port = camera->output[MMAL_CAMERA_CAPTURE_PORT];

    {
        MMAL_PARAMETER_CAMERA_CONFIG_T cam_config = {
            { MMAL_PARAMETER_CAMERA_CONFIG, sizeof (cam_config)}, width, height, 0, 0,
width, height, 3, 0, 1, MMAL_PARAM_TIMESTAMP_MODE_RESET_STC };
        mmal_port_parameter_set(camera->control, &cam_config.hdr);
    }

    format = camera_video_port->format;

    format->encoding = MMAL_ENCODING_I420;
    format->encoding_variant = MMAL_ENCODING_I420;

    format->es->video.width = width;
    format->es->video.height = height;
    format->es->video.crop.x = 0;
```

```
    format->es->video.crop.y = 0;
    format->es->video.crop.width = width;
    format->es->video.crop.height = height;
    format->es->video.frame_rate.num = 30;
    format->es->video.frame_rate.den = 1;

    camera_video_port->buffer_size = width * height * 3 / 2;
    camera_video_port->buffer_num = 1;

    status = mmal_port_format_commit(camera_video_port);

    if (status != MMAL_SUCCESS) {
        printf("Error: unable to commit camera video port format (%u)\n", status);
    }

    // create pool form camera video port
    camera_video_port_pool = (MMAL_POOL_T *) mmal_port_pool_create(camera_video_port,
camera_video_port->buffer_num, camera_video_port->buffer_size);

    if(color) {
        status = mmal_port_enable(camera_video_port, color_callback);
        if (status != MMAL_SUCCESS)
            printf("Error: unable to enable camera video port (%u)\n", status);
        else
            cout << "Attached color callback" << endl;
    }
    else {
        status = mmal_port_enable(camera_video_port, gray_callback);
        if (status != MMAL_SUCCESS)
            printf("Error: unable to enable camera video port (%u)\n", status);
        else
            cout << "Attached gray callback" << endl;
    }

    status = mmal_component_enable(camera);

    // Send all the buffers to the camera video port
    int num = mmal_queue_length(camera_video_port_pool->queue);
    int q;

    for (q = 0; q < num; q++) {
        MMAL_BUFFER_HEADER_T *buffer = mmal_queue_get(camera_video_port_pool->queue);

        if (!buffer) {
            printf("Unable to get a required buffer %d from pool queue\n", q);
        }

        if (mmal_port_send_buffer(camera_video_port, buffer) != MMAL_SUCCESS) {
            printf("Unable to send a buffer to encoder output port (%d)\n", q);
        }
    }
```

```
    if (mmal_port_parameter_set_boolean(camera_video_port, MMAL_PARAMETER_CAPTURE, 1) !=
MMAL_SUCCESS) {
        printf("%s: Failed to start capture\n", __func__);
    }

    cout << "Capture started" << endl;
}
```

CMakeLists.txt:

```
cmake_minimum_required(VERSION 2.8)

project(PiCapture)

SET(COMPILE_DEFINITIONS -Werror)

find_package( OpenCV REQUIRED )

include_directories(/opt/vc/include)
include_directories(/opt/vc/include/interface/vcos/pthreads)
include_directories(/opt/vc/include/interface/vmcs_host)
include_directories(/opt/vc/include/interface/vmcs_host/linux)
include_directories(USERLAND_DIR)
include_directories("${PROJECT_SOURCE_DIR}/include")

link_directories(/opt/vc/lib)
link_directories(/opt/vc/src/hello_pi/libs/vgfont)

add_executable(main src/main.cpp src/PiCapture.cpp)

target_link_libraries(main mmal_core mmal_util mmal_vc_client bcm_host ${OpenCV_LIBS})
```

From now on you can use the same strategy to grab frames from the camera board—include `cam.h` in your source file, and make an executable that uses both your source file and `PiCapture.cpp`.

Frame-Rate Comparisons

Figure 11-7 shows frame-rate comparisons for a USB camera versus the camera board. The programs used are simple; they just grab frames and show them using `imshow()` in a loop. In case of the camera board, the boolean flag in the argument list to the `PiCapture` constructor is used to switch from color to grayscale. For the USB camera, grayscale images are obtained by using `cvtColor()` on the color images.

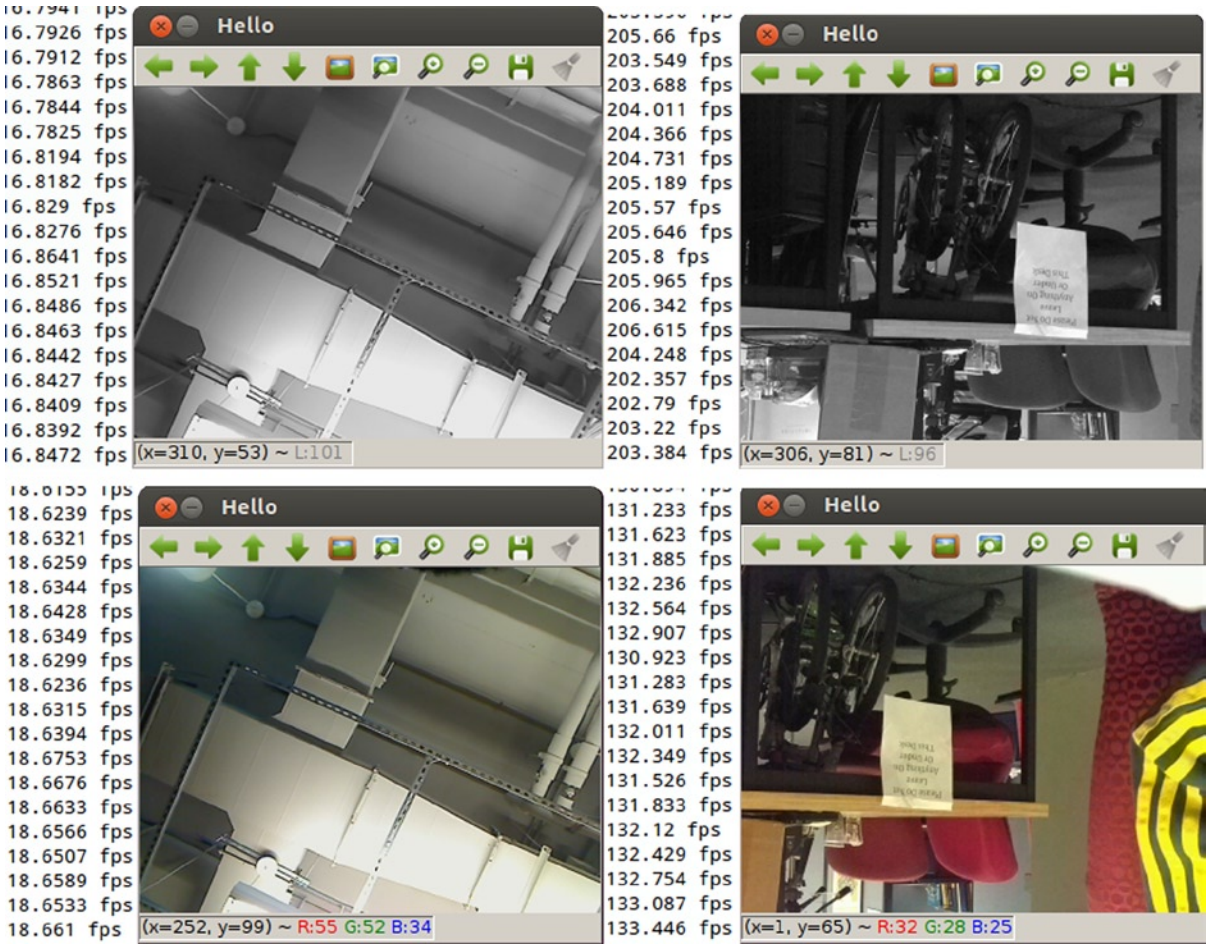

Figure 11-7. *Frame-rate tests. Clockwise from top left: Grayscale from USB camera, grayscale from camera board, color from camera board, and color from USB camera*

The frame-rate calculations are done as before using the OpenCV functions `getTickCount()` and `getTickFrequency()`. However, sometimes the parallel processes going on in `PiCapture.cpp` seem to be messing up the tick counts, and the frame rates measured for the camera board are reported to be quite higher than they actually are. When you run the programs yourself, you will find that both approaches give almost the same frame rate (visually) for color images, but grabbing from the camera board is much faster than using a USB camera for grayscale images.

Usage Examples

Once you set everything up everything on your Pi, it acts pretty much like your normal computer and you should be able to run all the OpenCV code that you ran on your normal computer, except that it will run slower. As usage examples, we will check frame rates of the two types of object detectors we have developed in this book, color-based and ORB keypoint-based.

Color-based Object Detector

Remember the color-based object detector that we perfected in Chapter 5 (Listing 5-7)? Let us see how it runs for USB camera versus color images grabbed using the wrapper code. Figure 11-8 shows that for frames of size 320 x 240, the detector runs about 50 percent faster if we grab frames using the wrapper code!

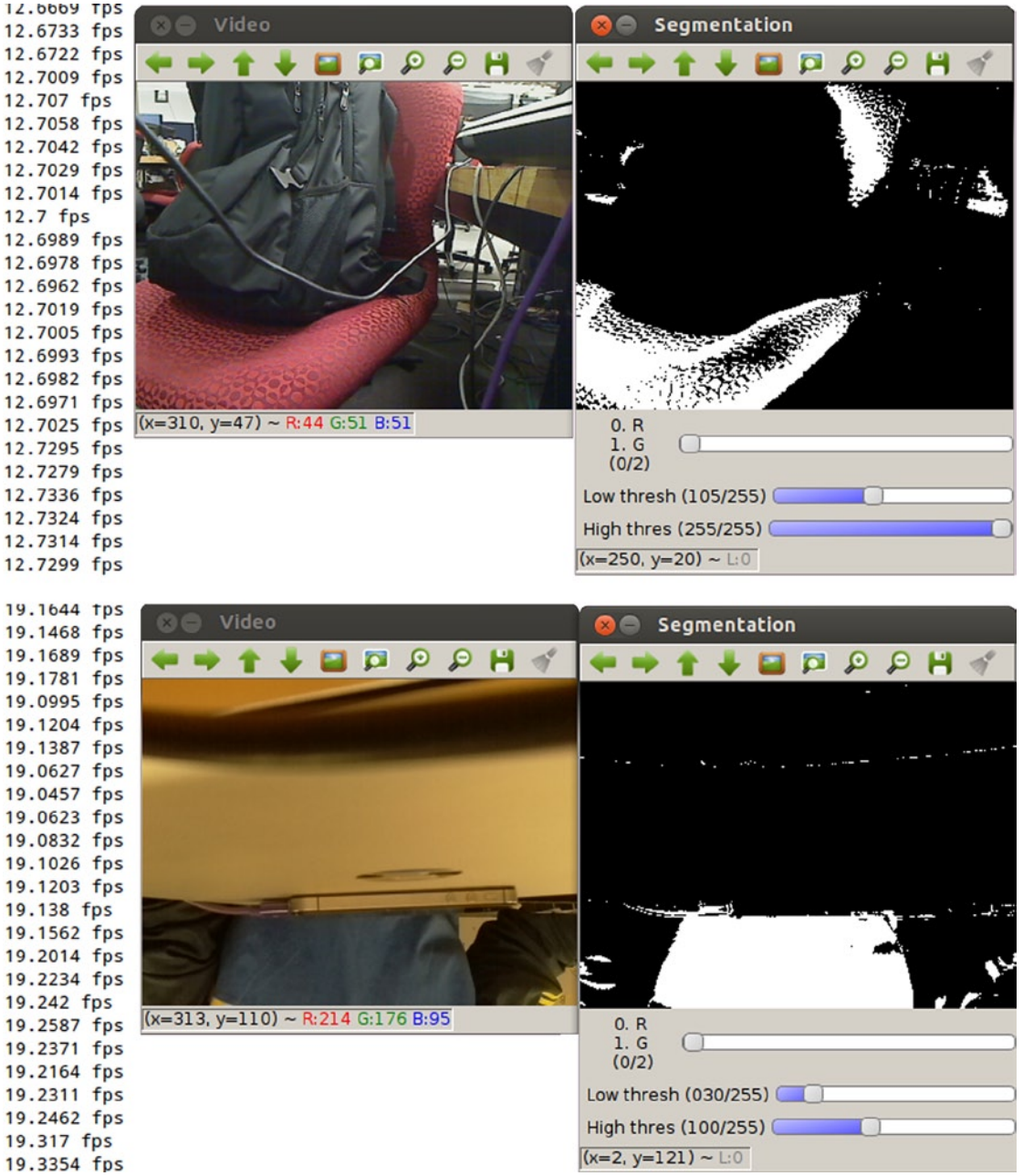

Figure 11-8. *Color-based object detector using color frames from USB camera (top) and camera board (bottom)*

ORB Keypoint-based Object Detector

The ORB keypoint-based object detector that we developed in Chapter 8 (Listing 8-4) requires a lot more computation than computing histogram back-projections, and so it is a good idea to see how the two methods perform when the bottleneck is not frame-grabbing speed but frame-processing speed. Figure 11-9 shows that grabbing 320 x 240 grayscale frames (remember, ORB keypoint detection and description requires just a grayscale image) using the wrapper code runs about 14 percent faster than grabbing color frames from the USB camera and manually converting them to grayscale images.

Figure 11-9. *ORB-based object detector using frames grabbed from a USB camera (top) and camera board (bottom)*

Summary

With this chapter, I conclude our introduction to the fascinating field of embedded computer vision. This chapter gave you a detailed insight into running your own vision algorithms on the ubiquitous Raspberry Pi, including the use of its swanky new camera board. Feel free to let your mind wander and come up with some awesome use-cases! You will find that knowing how to make a Raspberry Pi understand what it sees is a very powerful tool, and its applications know no bounds.

There are lots of embedded systems other than the Pi out there in the market that can run OpenCV. My advice is to pick one that best suits your application and budget, and get really good at using it effectively rather than knowing something about every embedded vision product out there. At the end of the day, these platforms are just a means to an end; it is the algorithm and idea that makes the difference.