CHAPTER 7

Using OpenCV

Open source Computer Vision (OpenCV) is a set of cross-platform libraries containing functions that provide computer vision in real time.

OpenCV is huge framework and there are some basic functions needed to capture and process videos and images so that they can communicate with input devices, such as a webcams. This chapter introduces the basic concepts needed to build powerful applications with your Intel Galileo board. The project will focus on how to connect a webcam to Intel Galileo, how the webcam works in Linux, how to capture pictures and videos, how to change the pictures with OpenCV algorithms, and how to detect and recognize faces and emotions.

BSP (board support package) SD card images of the Intel Galileo board support OpenCV and allow projects like the one in this chapter to be developed.

Several programs and tasks will be executed in this project. They are divided into Video4Linux and OpenCV categories as follows:

1. Identify the capabilities of webcam with V4L2.
2. Capture pictures using V4L2.
4. Capture and process images with OpenCV.
5. Incorporate edge detection in your pictures with OpenCV.
6. Incorporate face and eye detection with OpenCV.
7. Detect emotions with OpenCV.

Note that the V4L2 examples use C and the OpenCV examples are written in C++ and Python. This is done to illustrate the performance of OpenCV in different languages and its cross-platform capabilities.

OpenCV Primer

OpenCV was developed by Intel research and is now supported by Willow Garage under the open source BSD license.

But what is computer vision and what is used for? Computer vision is the ability to provide methods and algorithms that help computers interpret the environment around them. Human eyes are able to capture the environment around us stereographically.
They send the images to our brains, which interpret the images with a sense of depth, format, and dimension to all the components that compose an image.

For example, when you look at a dog in a park, you can tell how far the dog is from you, where exactly the dog is, whether you know the dog and his name, the format of the objects in the park such as sandboxes, trees, and parked cars, if it is going to rain or not, and so on.

A three-month old baby can identify objects and faces in a process that looks so natural for human beings.

What about computers? How do we program computers to use the same kind of analysis and come to the same conclusions when analyzing a simple picture of the park?

Several mathematical models, static data, and machine learned methodologies have being developed that allow computers to “see” the world and understand the environment around them.

Robots use computer vision to assemble cars, recognize people, help patients in hospitals, and replace astronauts in dangerous missions in the space. In the future they will be able to replace soldiers in the battlefield, perform surgeries with precision, and more.

The OpenCV libraries offer a powerful infrastructure that enables developers to create sophisticated computer vision applications, abstracting all mathematic, static, and machine learning models out of the application context.

It is important to understand how V4L2 works because sometimes OpenCV throws some “mysterious” messages related to issues with V4L2. If you focus exclusively in OpenCV, it will be difficult to understand what is going on and how to fix these issues. These “mysterious” messages are related to V4L2 and not to OpenCV, which can be confusing.

If you need more details about how the algorithms works, visit the OpenCV website (opencv.org) and improve your knowledge with books dedicated exclusively to OpenCV and image processing.

Project Details

This project requires a webcam to serve as Intel Galileo’s “eyes” to capture pictures and videos and apply algorithms using OpenCV. If you are using Intel Galileo, you will also need an OTG-USB adapter in order to connect the webcam because, unlike Intel Galileo Gen2, Intel Galileo does not have an OTG-USB connector.

You’ll need to generate a custom BSP image that contains all the tools and software packages that will be used. You can also download the BSP image from the code folder and copy it to the micro SD card, which will save you hours building with Yocto. The tools and ipks packages used in this chapter require more space than the SPI images can support, thus a micro SD card is necessary.

To focus directly on the OpenCV examples, it is necessary to understand the capabilities of your webcam, like the resolution, encodes, and frames per second that are supported. Understanding these capabilities using V4L2 will prevent you from wasting hours trying to decipher errors that in fact do not come from OpenCV but from V4L2.
Materials List

If you are using the Intel Galileo Gen 2, you need a webcam that’s compatible with the UVC standard. Intel Galileo Gen 2 has an OTG USB connector and you can connect the camera directly to the board.

Table 7-1 lists this project’s materials. If you are using the Intel Galileo only (first generation), you need to buy an OTG USB 2.0 adaptor similar to the one shown in Figure 7-1.

Table 7-1. Materials List

<table>
<thead>
<tr>
<th>Quantity</th>
<th>Components</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Webcam Logitech C270</td>
</tr>
<tr>
<td>1</td>
<td>OTG-USB 2.0 adaptor with Micro-USB male to USB A Female (only for Intel Galileo)</td>
</tr>
<tr>
<td>1</td>
<td>Micro SD card, 4GB to a maximum of 32GB</td>
</tr>
</tbody>
</table>

Figure 7-1. OTG-USB 2.0 adaptor with micro-USB male to USB A female

The Logitech webcam C270 is the best bet for this project because it is an affordable camera (US$ 26.00), complies with the USB Video Class (UVC), and works with the programs presented in this chapter. The Logitech webcam C270 is shown in Figure 7-2.
Avoid using the OTG-USB adaptor with an L connector due to space constraints with other connectors in the board.

For more details about UVC, read the next section.

**USB Video Class**

UVC is a standard that defines how the device streams video and pictures through a USB port. It uses a driver named uvcvideo, which is supported by the BSP SD card software releases. In this case, the device is a simple webcam, but there are other types of devices that support UVC, such as transcoders, video recorders, camcorders, and so on.

If you have a different webcam and you want to use it with this project, just check if the webcam is UVC compliant on the website [http://www.ideasonboard.org/uvc/](http://www.ideasonboard.org/uvc/) and check the Supported Devices section, as shown in Figure 7-3.
This website will tell you if the webcam works with the `uvcvideo` driver. If it does, it will be classified as “Device Works” or “Device Works with Issues.”

However, even when the device is classified as working, that does not mean you can trust that information completely. Developers often end up setting the camera to a lower resolution or decreasing the frames per second in order to make the code work, even when the webcam is reported as one that works.

**Preparing the BSP Software Image and Toolchain**

As mentioned, it is necessary to prepare a custom BSP image and save the deployed files in a micro SD card in order to run the examples in this chapter. The toolchain is also necessary to align with eGlibc enabled in the build.

The procedure to create a BSP image and toolchain based on the Yocto project was discussed in Chapter 2, so if you have not read Chapter 2 yet, now is a good time to do so.

Alternatively, if you do not want to learn how to generate the image you can download all files into the `/code/SDcard` folder of this chapter and copy it to your micro SD card. Doing so will save you hours.

Once you have copied the files to the micro SD card, insert in the micro SD card slot (review Chapter 1) and boot the board.
Using eGlibc for Video4Linux Image

The standard BSP SD card release is based on tiny uClibc but there is some problem with V4L. The solution is to build the full Intel Galileo SD card image based on eGlibc. To do this, open the file `.../meta-clanton-distro/recipes-multimedia/v4l2apps/v4l-utils_0.8.8.bbappend` and comment all three lines using your favorite text editor:

```bash
#FILESEXTRAPATHS_prepend := "${THISDIR}/files:"
#SRC_URI += "file://uclibc-enable.patch"
#DEPENDS += "virtual/libiconv"
```

Increasing the rootfs Size

Some packages related to tools and development libs for OpenCV and V4L will be used, which means you need to increase the rootfs size.

To do this, edit the `.../meta-clanton-distro/recipes-core/image/image-full.bb` file by changing the following lines (see the items in bold):

```bash
IMAGE_ROOTFS_SIZE = "507200"
IMAGE_FEATURES += "package-management dev-pkgs"
IMAGE_INSTALL += "autoconf automake binutils binutils-symlinks cpp cpp-symlinks gcc gcc-symlinks g++ g++-symlinks gettext make libstdc++ libstdc++-dev file coreutils"
```

In the first line, rootfs (IMAGE_ROOTFS_SIZE) is increased to 5GB. A new image feature (IMAGE_FEATURES) is enhanced with the integration of the development packages (dev-pkgs). If IMAGE_INSTALL is added, a series of development tools will be part of the image (g++, make, and so on).

Disabling GPU Support on OpenCV

The Quark SoC used on Intel Galileo does not contain any GPUs (Graphics Processing Units). OpenCV can be compiled so that it enables or disables GPU through a definition called CUDA. For Intel Galileo, GPU must be disabled.

To disable GPU, you need to edit two files: `.../meta-oe/meta-oe/recipes-support/opencv/opencv_2.4.3.bb` and `.../meta-clanton-distro/recipes-support/opencv/opencv_2.4.3.bbappend`. Make the same changes to both of the lines in EXTRA_OECMAKE (see the items in bold):

```bash
EXTRA_OECMAKE = "-DPYTHON_NUMPY_INCLUDE_DIR:PATH=${STAGING_LIBDIR}/ ${PYTHON_DIR}/site-packages/numpy/core/include \ -DBUILD_PYTHON_SUPPORT=ON \ -DWITH_FFMPEG=ON \ -DWITH_CUDA=OFF \ -DBUILD_opencv_gpu=OFF \ -DWITH_GSTREAMER=OFF \```

324
Building the SD Image and Toolchain

You use the same procedure explained in Chapter 2 to build the image here. For your quick reference, the build process commands are:

```bash
cd meta-clanton*
./setup.sh
source poky/oe-init-build-env yocto_build
```

To build the full SD image, the bitbake is:

```
bitbake image-full-galileo
```

To create the toolchain, use this command:

```
bitbake image-full-galileo -c populate_sdk
```

Development Library Packages

You might encounter the following message when running OpenCV programs on Intel Galileo:

```
error while loading shared libraries: libopencv_gpu.so.2.4: cannot open shared object file: No such file or directory
```

If you do, it means some libraries are not properly installed on Intel Galileo. You can install the packages (the ipk files) individually as well.

The code folder contains a tarball named ipk.tar.gz with all the ipk files needed for OpenCV and V4L. Copy that file to Intel Galileo and install the libraries using opkg. To decompress and install the ipk files for OpenCV and V4L, you use the following command:

```bash
root@clanton: #tar -zxvf ipk.tar.gz
root@clanton: #cd ipk
root@clanton: #opkg install libopencv-gpu2.4_2.4.3-r2_i586.ipk
libopencv-stitching2.4_2.4.3-r2_i586.ipk libopencv-ts2.4_2.4.3-r2_i586.ipk
libopencv videostab2.4_2.4.3-r2_i586.ipk libv4l-dev_0.8.8-r2_i586.ipk
libv4l-dbgsym_0.8.8-r2_i586
```
Connecting the Webcam

After inserting the micro SD card and booting the board, you need to load the uvcvideo driver and connect your webcam.

Open a terminal shell and type the following command to load the driver:

```
root@clanton:~# modprobe uvcvideo
```

If you cannot load the `uvcvideo` module driver, it means you have a problem with the custom BSP image. Review the build process or use the micro SD card files provided with this chapter.

The driver must be loaded with success. Then, connect your webcam and you should see messages similar to these:

```
root@clanton:~# [31372.707513] USB Video Class driver (1.1.1)
[31374.420165] usb 2-1: new high-speed USB device number 3 using ehci-pci
[31474.930869] input: UVC Camera
[31474.930869] as /devices/pci0000:00/0000:00:14.3/usb2/2-1/2-1:1.0/input/input2
```

The kernel message `input:UVC Camera` confirms the webcam is in compliance with UVC.

To determine on which device the camera was installed, type the following command:

```
root@clanton:~# ls /dev/video*
/dev/video0
```

In this case the device was properly installed and it is mapped as `/dev/video0`. The last number (0 in this case) won’t always be 0. When you connect the webcam the driver can assign any integer number. For example, if you have a USB host and connect two cameras, one might be `/dev/video0` and the other might be `/dev/video1`. If you keep connecting more and more webcams to the USB host, each one is mapped and then integer will increase, such as `/dev/video2`, `/dev/video3`, and so on.

If you have a single webcam currently mapped as `/dev/video0` and for some reason the webcam crashes and was not released properly, the next time you connect, it might be mapped as `/dev/video1`.

This information sounds irrelevant, but keep it in your mind because it will be useful when working with OpenCV later.
Chapter 7 ■ Using OpenCV

Introduction to Video4Linux

Video4Linux, also called V4L, is a set of API and drivers developed to allow Linux to communicate with devices that receive and transmit audio and video. With V4L it is possible to communicate with video cameras, TVs and radio cards, codecs converters, streaming devices, and remote controllers.

This chapter focuses exclusively on the C270 webcam using the second version of this API called Video4Linux 2 (V4L2).

V4L2 includes several bug fixes and new API functions that were not covered in the first release, V4L.

The code samples regarding V4L2 in this chapter come from the official V4L2 website with minor changes to adapt to the C270 webcam. For more detail regarding the API, visit http://linuxtv.org/downloads/v4l-dvb-apis.

Before discussing the technical details about how the API works, let’s explore the C270 webcam using a tool based on V4L2, called v4l2-ctl.

It is important to use your camera with this tool before writing your own code or using OpenCV. It will help to understand the settings necessary to make this API work. If you are using a different camera, you will be able to exercise your camera to understand the adaptations that will be necessary for the code examples in this chapter.

Exploring the Webcam Capabilities with V4L2-CTL

Before you start using OpenCV it is important to understand the following aspects of your camera:

- The encode/pixel formats supported
- The resolutions supported to capture images
- The resolutions supported to capture video
- The frames per second (fps) supported in different encode modes
- The resolutions that really work

If these packages were properly installed you should have a command-line tool called v4l2-ctl. This tool not only informs the capabilities of your camera but is able to set/change some properties.

For example, with Logitech C270, you should connect the uvcvideo driver properly. You can type v4l2-ctl --all to check the current capabilities.

root@clanton:~# v4l2-ctl --all
Driver Info (not using libv4l2):
  Driver name : uvcvideo
  Card type   : UVC Camera (046d:0825)
  Bus info    : usb-0000:00:14.3-1
  Driver version: 3.8.7
The current encode is set to MJPG, which is a motion JPEG streamer, and the resolution is 640/480 pixels. The current frame per seconds (fps) is 30 and video cropping is set to the actual video resolution of 640/480, as informed by Crop Capability Video Capture.
Even if you are using the same webcam model, you might see different settings. These are the current settings of my webcam and considering this webcam supports other resolutions and encodes, yours might use different settings.

Changing and Reading Camera Properties

In the previous command, other properties like brightness, contrast, and saturation are used. You can change a property using the \texttt{--set-ctrl} argument with \texttt{v4l2-ctl} tool. Suppose you want to change the contrast attribute from 32 to 40. To do so, type the following in your terminal:

\texttt{\textbackslash root@clanton:\textasciitilde \# v4l2-ctl \textasciitilde set-ctrl=contrast=40}

You can instead use the delimiter =:

\texttt{\textbackslash root@clanton:\textasciitilde \# v4l2-ctl \textasciitilde set-ctrl contrast=40}

To read an individual property, use \texttt{--get-ctrl} rather than \texttt{--all}, which lists all the properties. For example:

\texttt{\textbackslash root@clanton:\textasciitilde \# v4l2-ctl \textasciitilde get-ctrl contrast\n\textbf{contrast}: 40}

You can also use the \texttt{-L} argument to get the list of controls. See the following example:

\texttt{\textbackslash root@clanton:\textasciitilde \# v4l2-ctl -L\n\textbf{brightness} (int) : min=0 max=255 step=1 default=128\hspace{1cm}\textbf{value}=128\n\textbf{contrast} (int) : min=0 max=255 step=1 default=32\hspace{1cm}\textbf{value}=40\n\textbf{saturation} (int) : min=0 max=255 step=1 default=32\hspace{1cm}\textbf{value}=32\n\textbf{white_balance_temperature_auto} (bool) : default=1 \textbf{value}=1\n\textbf{gain} (int) : min=0 max=255 step=1 default=64\hspace{1cm}\textbf{value}=64\n\textbf{power_line_frequency} (menu) : min=0 max=2 default=2 \textbf{value}=2\hspace{1cm}0 : Disabled\hspace{2.5cm}1 : 50 Hz\hspace{2.5cm}2 : 60 Hz\n\textbf{white_balance_temperature} (int) : min=0 max=10000 step=10 default=4000\hspace{0.5cm}\textbf{value}=4000 flags=inactive\n\textbf{sharpness} (int) : min=0 max=255 step=1 default=24\hspace{1cm}\textbf{value}=24\n\textbf{backlight_compensation} (int) : min=0 max=1 step=1 default=0 \textbf{value}=0}
 CHAPTER 7 ■ USING OPENCV

exposure_auto (menu): min=0 max=3 default=3 value=3
1: Manual Mode
3: Aperture Priority Mode

exposure_absolute (int): min=1 max=10000 step=1 default=166
value=166 flags=inactive

exposure_auto_priority (bool): default=0 value=1

Pixel Formats and Resolution

To check the encodes that your webcam supports with v4l2-ctl, use v4l2-ctl --list-formats.

root@clanton:~# v4l2-ctl --list-formats
ioctl: VIDIOC_ENUM_FMT

Index  : 0
Type   : Video Capture
Pixel Format: 'YUYV'
Name   : YUV 4:2:2 (YUYV)

Index  : 1
Type   : Video Capture
Pixel Format: 'MJPG' (compressed)
Name   : MJPEG

As you can see, the webcam supports two types of encodes in this case: YUYV (index 0) and Motion JPEG (index 1) can both capture video as shown by the field Type.

When the v4l2-ctl --all command was previously executed, the current settings were pointing to MJPG.

The webcam C270 supports 1280x720 resolution. Use the following command to change the current 640x480:

root@clanton:~# v4l2-ctl --set-fmt-video width=1920,height=780,pixelformat=0

The resolution needs to be changed to 1920x720 and the pixel format with index 0 represents YUYV, as demonstrated by the byte command --list-formats.

To determine the current pixel format and resolution, you can run the following command (alternative to --all), which is used before in order to have summarized information:

root@clanton:~# v4l2-ctl --get-fmt-video
Format Video Capture:
  Width/Height : 1280/720
  Pixel Format  : 'YUYV'
  Field        : None
  Bytes per Line: 2560
  Size Image   : 1843200
  Colorspace   : SRGB
To check all the resolutions supported by the webcam on each encode as the frame per seconds supported, you must run the `v4l2-ctl --list-formats-ext` command.

```
root@clanton:~# v4l2-ctl --list-formats-ext
ioctl: VIDIOC_ENUM_FMT
Index : 0
Type : Video Capture
Pixel Format: 'YUYV'
Name   : YUV 4:2:2 (YUYV)

Size: Discrete 640x480
  Interval: Discrete 0.033 s (30.000 fps)
  Interval: Discrete 0.040 s (25.000 fps)
  Interval: Discrete 0.050 s (20.000 fps)
  Interval: Discrete 0.067 s (15.000 fps)
  Interval: Discrete 0.100 s (10.000 fps)
  Interval: Discrete 0.200 s (5.000 fps)

Size: Discrete 160x120
  Interval: Discrete 0.033 s (30.000 fps)
  Interval: Discrete 0.040 s (25.000 fps)
  Interval: Discrete 0.050 s (20.000 fps)
  Interval: Discrete 0.067 s (15.000 fps)
  Interval: Discrete 0.100 s (10.000 fps)
  Interval: Discrete 0.200 s (5.000 fps)

Size: Discrete 1184x656
  Interval: Discrete 0.100 s (10.000 fps)
  Interval: Discrete 0.200 s (5.000 fps)

Size: Discrete 1280x720
  Interval: Discrete 0.133 s (7.500 fps)
  Interval: Discrete 0.200 s (5.000 fps)

Size: Discrete 1280x960
  Interval: Discrete 0.133 s (7.500 fps)
  Interval: Discrete 0.200 s (5.000 fps)

Index : 1
Type   : Video Capture
Pixel Format: 'MJPG' (compressed)
Name   : MJPEG

Size: Discrete 640x480
  Interval: Discrete 0.033 s (30.000 fps)
  Interval: Discrete 0.040 s (25.000 fps)
  Interval: Discrete 0.050 s (20.000 fps)
  Interval: Discrete 0.067 s (15.000 fps)
  Interval: Discrete 0.100 s (10.000 fps)
  Interval: Discrete 0.200 s (5.000 fps)
```
Size: **Discrete 160x120**  
Interval: Discrete 0.033 s (30.000 fps)  
Interval: Discrete 0.040 s (25.000 fps)  
Interval: Discrete 0.050 s (20.000 fps)  
Interval: Discrete 0.067 s (15.000 fps)  
Interval: Discrete 0.100 s (10.000 fps)  
Interval: Discrete 0.200 s (5.000 fps)  

Size: **Discrete 1184x656**  
Interval: Discrete 0.033 s (30.000 fps)  
Interval: Discrete 0.040 s (25.000 fps)  
Interval: Discrete 0.050 s (20.000 fps)  
Interval: Discrete 0.067 s (15.000 fps)  
Interval: Discrete 0.100 s (10.000 fps)  
Interval: Discrete 0.200 s (5.000 fps)  

Size: **Discrete 1280x720**  
Interval: Discrete 0.033 s (30.000 fps)  
Interval: Discrete 0.040 s (25.000 fps)  
Interval: Discrete 0.050 s (20.000 fps)  
Interval: Discrete 0.067 s (15.000 fps)  
Interval: Discrete 0.100 s (10.000 fps)  
Interval: Discrete 0.200 s (5.000 fps)  

Size: **Discrete 1280x960**  
Interval: Discrete 0.033 s (30.000 fps)  
Interval: Discrete 0.040 s (25.000 fps)  
Interval: Discrete 0.050 s (20.000 fps)  
Interval: Discrete 0.067 s (15.000 fps)  
Interval: Discrete 0.100 s (10.000 fps)  
Interval: Discrete 0.200 s (5.000 fps)  

This command provides a long list for webcam C270 with several resolutions supported for each pixel format. Note that each resolution has a list of frames per second supported in case you capture video. At a glance, it looks like the table contains the same resolutions and frames per second, but in fact there are a few differences. For example, the resolution 1184x656 with pixel format YUYV only supports capturing video in 10 and 5 fps, whereas the same resolution in MJPG supports 30, 25, 20, 15, 10, and 5 fps. If these kinds of details are unnoticed and the wrong settings are made, the video or picture capture will fail and cause problems when working with OpenCV.

To change the frames per second using v4l2-ctl, use the set-parm argument. For example, you can set 30 fps using the following:

```
root@clanton:~# v4l2-ctl --set-parm=30  
```
Frame rate set to 30.000 fps
Capturing Videos and Images with libv4l2

The process for capturing video and images using V4L2 is quite similar because it involves the same IOCTL calls and the same sequence. When the webcam is single-planar that means one single video frame has a single buffer address as a starting point. Some devices are multi-planar, which means a single video frame requires more than one start address.

The sequence for capturing the video is quite the same, but the difference is in how the software saves the frames—individually or as an entire stream.

Copy the images between the kernel and the userspace. Remember that the applications run in the userspace context. There are three I/O operations supported in V4L2 API:

- **Memory mapped buffers (mmap)**: The buffers are allocated in the kernel space. The device determines the number of buffers that can be allocated as the size of the buffers. Usually the `mmap()` function and the application must inform this method using `V4L2_MEMORY_MMAP` before it queries the buffers. This is the method used with the webcam C270.

- **Userspace pointers**: The buffers are allocated to the userspace context using the regular `malloc()` or `calloc()` functions. In this case, the `V4L2_MEMORY_USERPTR` is used to query the buffer.

- **Direct Read/Write**: The application in this case can read/write the buffer directly. Thus, no mapped memory (mmap) or userspace memory allocation is necessary (malloc/calloc).

In all three methods, the device determines which ones are supported. You must also set the resolution, the pixel format, and the frames per second.

A Program for Capturing Video

The code used in this section is from “Appendix D: Video Capture Example” of the *Linux Media Infrastructure API* documentation at [http://linuxtv.org/downloads/v4l-dvb-apis/capture-example.html](http://linuxtv.org/downloads/v4l-dvb-apis/capture-example.html) (with a few changes to support the C270 motion JPEG stream).

All communication between the userspace and the kernel driver is made through IOCTL calls. Therefore, it’s a good idea to have a function to do that.

```c
static int xioctl(int fh, int request, void *arg)
{
    int r;

    do {
        r = ioctl(fh, request, arg);
    } while (-1 == r && EINTR == errno);

    return r;
}
```
This function will be used several times in all IOCTL calls that send IOCTL to the kernel using the `ioctl()` function.

Figure 7-4 represents the sequence for video and images.

**Figure 7-4.** Flowchart related to capturing video using V4L2
Each step in the flowchart is explained with the respective snippet:

1. **Open the device:** You use a file descriptor to open the device. In this case, a string representing the device is used. The string contains the proper device name in /dev. For example, the string might be "/dev/video0". The O_NONBLOCK option prevents the software from remaining blocked when the buffers are read (this is explained in more detail in the dequeue process in Step 9).

   ```c
   static void open_device(void)
   {
     ...
     ...
      ...
      
      fd = open(dev_name, O_RDWR /* required */ | O_NONBLOCK, 0);
      
      if (-1 == fd) {
         fprintf(stderr, "Cannot open '%s': %d, %s\n",
                  dev_name, errno, strerror(errno));
         exit(EXIT_FAILURE);
      }
   }
   ```

2. **Initiate the device:** If the device opens properly, you need to ask the device about its capabilities using VIDIOC_QUERYCAP.

   ```c
   struct v4l2_capability cap;
   
   ...
   ...
   ...
   
   if (-1 == xioctl(fd, VIDIOC_QUERYCAP, &cap)) {
      if (EINVAL == errno) {
         fprintf(stderr, "%s is no V4L2 device\n",
                  dev_name);
         exit(EXIT_FAILURE);
      } else {
         errno_exit("VIDIOC_QUERYCAP");
      }
   }
   ```
3. **Reset any image cropping set**: Use VIDIOC_S_CROP set to its default values. If the devices do not support cropping, you can ignore the errors because the image will always have the same resolution.

```c
struct v4l2_crop crop;
struct v4l2_cropcap cropcap;
...
...
cropcap.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
if (0 == xioctl(fd, VIDIOC_CROPCAP, &cropcap)) {
    crop.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
    crop.c = cropcap.defrect; /* reset to default */
    if (-1 == xioctl(fd, VIDIOC_S_CROP, &crop)) {
        switch (errno) {
        case EINVAL: /* Cropping not supported. */
            break;
        default: /* Errors ignored. */
            break;
        }
    }
}
```

4. **Set pixel format and resolution**: If VIDIOC_S_FMT is used, the device will assume the setting passed to IOCTL with the structure v4l2_format. Otherwise, if VIDIOC_G_FMT is used, the current programmed settings are used. In this case you can use the v4l2-ctl tool as explained before. This is the only part of the code that changed from the original code on the V4L2 website. The force_format variable, when set to true, forces the format to motion JPEG stream with 1280x720 resolution. Otherwise, use the current setting of your camera, which can be changed using the v4ctl tool.

```c
fmt.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
if (force_format) {
    fmt.fmt.pix.width       = 1280;
    fmt.fmt.pix.height      = 720;
    fmt.fmt.pix.pixelformat = V4L2_PIX_FMT_MJPEG;
    fmt.fmt.pix.field       = V4L2_FIELD_NONE;
```
if (-1 == xioctl(fd, VIDIOC_S_FMT, &fmt))
    errno_exit("VIDIOC_S_FMT");
/* Note VIDIOC_S_FMT may change width and height. */
} else {
    /* Preserve original settings as set by v4l2-ctl for example */
    if (-1 == xioctl(fd, VIDIOC_G_FMT, &fmt))
        errno_exit("VIDIOC_G_FMT");
}

5. Allocate buffers: With the v4l2_requestbuffers structure
   and more precisely using the count field, it is passed to
   the device using IOCTL VIDIOC_REQBUFS, which requires a
certain number of buffers to be used by the device to store the
images. In the case of webcam C270, the maximum number of
buffers is five. More than five and the webcam will report
“out of memory” and VIDIOC_REQBUFS will fail. It’s best to set
at least two buffers. If the device accepts the number of buffers
asked, due to the VIDIOC_REQBUFS call, the size of the buffer
is reported. Then you allocate the buffers in the userspace
context to allow the device to fill them. The allocation might
be done using regular functions like calloc() and malloc().

struct v4l2_requestbuffers req;

CLEAR(req);

req.count = 5;
req.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
req.memory = V4L2_MEMORY_MMAP;

if (-1 == xioctl(fd, VIDIOC_REQBUFS, &req)) {
    if (EINVAL == errno) {
        fprintf(stderr, "%s does not support "
                "memory mapping\n", dev_name);
        exit(EXIT_FAILURE);
    } else {
        errno_exit("VIDIOC_REQBUFS");
    }
}

if (req.count < 2) {
    fprintf(stderr, "Insufficient buffer memory on %s\n",
             dev_name);
    exit(EXIT_FAILURE);
}
buffers = calloc(req.count, sizeof(*buffers));

if (!buffers) {
    fprintf(stderr, "Out of memory\n");
    exit(EXIT_FAILURE);
}

6. Query buffers' statuses and map the memory for each:
For each buffer allocated, it is necessary to query their status using VIDIOC_QUERYBUF. In response to VIDIOC_QUERYBUF, the offset of the buffer from the start device as the length of each buffer is reported. With this information, the function mmap() must be called to map the virtual memory that will be shared between the userspace and the device.

   for (n_buffers = 0; n_buffers < req.count; ++n_buffers) {
        struct v4l2_buffer buf;

        CLEAR(buf);

        buf.type       = V4L2_BUF_TYPE_VIDEO_CAPTURE;
        buf.memory     = V4L2_MEMORY_MMAP;
        buf.index      = n_buffers;

        if (-1 == xioctl(fd, VIDIOC_QUERYBUF, &buf))
            errno_exit("VIDIOC_QUERYBUF");

        buffers[n_buffers].length = buf.length;
        buffers[n_buffers].start =
            mmap(NULL /* start anywhere */,
                 buf.length,
                 PROT_READ | PROT_WRITE /* required */,
                 MAP_SHARED /* recommended */,
                 fd, buf.m.offset);

        if (MAP_FAILED == buffers[n_buffers].start)
            errno_exit("mmap");
    }

7. Enqueue the buffers: Considering that the C270 webcam operated with the mmap method, it is necessary to exchange each buffer obtained with VIDIOC_REQBUFS, with the driver using VIDIOC_QBUF. The VIDIOC_QBUF enqueues each buffer.

   case IO_METHOD_MMAP:
        for (i = 0; i < n_buffers; ++i) {
            struct v4l2_buffer buf;
CLEAR(buf);
buf.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
buf.memory = V4L2_MEMORY_MMAP;
buf.index = i;

if (-1 == xioctl(fd, VIDIOC_QBUF, &buf))
    errno_exit("VIDIOC_QBUF");
}

8. **Start the capture:** With all buffers ready, it is necessary to start the streaming with VIDIOC_STREAMON.

if (-1 == xioctl(fd, VIDIOC_STREAMON, &type))
    errno_exit("VIDIOC_STREAMON");

9. **Dequeue the buffers to read the frames:** By calling VIDIOC_DQBUF, the buffers are dequeued and the frames can be read in case of success. Note that there is no specific order for the buffers, so as result beside the data, the buffer index is informed. It is necessary to keep waiting for all the buffers to be dequeued and a while loop can be implemented for this purpose. However, to avoid the blocking operation in context of the userspace, the select() function is used. Note that if you try to read a specific buffer that's not available in memory and if the device was opened with O_NONBLOCK, the error EAGAIN is thrown during the VIDIOC_DQBUF call. Otherwise it remains blocked until the buffer is ready.

```c
static void mainloop(void)
{
    unsigned int count;

    count = frame_count;

    while (count-- > 0) {
        for (;;) {
            fd_set fds;
            struct timeval tv;
            int r;

            FD_ZERO(&fds);
            FD_SET(fd, &fds);

            /* Timeout. */
            tv.tv_sec = 2;
            tv.tv_usec = 0;

            r = select(fd + 1, &fds, NULL, NULL, &tv);
```
if (-1 == r) {
    if (EINTR == errno)
        continue;
    errno_exit("select");
}

if (0 == r) {
    fprintf(stderr, "select timeout\n");
    exit(EXIT_FAILURE);
}

if (read_frame())
    break;
/* EAGAIN - continue select loop. */
}

... ...

static int read_frame(void)
{
    struct v4l2_buffer buf;
    unsigned int i;
    ...
    ...
    ...
    if (-1 == xioctl(fd, VIDIOC_DQBUF, &buf)) {
        switch (errno) {
        case EAGAIN:
            printf("EAGAIN\n");
            return 0;

        case EIO:
            printf("EIO\n");
            /* Could ignore EIO, see spec. */
            /* fall through */

        default:
            printf("default\n");
            errno_exit("VIDIOC_DQBUF");
        }
    }
}
10. **Stop the stream**: Simply call VIDIOC_STREAMOFF.

```c
    type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
    if (-1 == xioctl(fd, VIDIOC_STREAMOFF, &type))
        errno_exit("VIDIOC_STREAMOFF");
```

11. **Free the buffers and unmap the memory**: The memory virtually mapped with mmap() must be unmapped with the munmap() function and the memory allocated to the buffer must be freed with the free() function.

```c
    static void uninit_device(void)
    {
        unsigned int i;

        switch (io) {
            ...

            case IO_METHOD_MMAP:
                for (i = 0; i < n_buffers; ++i)
                    if (-1 == munmap(buffers[i].start, buffers[i].length))
                        errno_exit("munmap");
                break;
            ...
            ...

        }

        free(buffers);
    }
```

12. **Close the device**: The device file descriptor is closed.

```c
    static void close_device(void)
    {
        if (-1 == close(fd))
            errno_exit("close");

        fd = -1;
    }
```

The code excerpts used in this section are from “Appendix D: Video Capture Example” of the Linux Media Infrastructure API documentation at http://linuxtv.org/downloads/v4l-dvb-apis/capture-example.html (with a few changes to support the C270 motion JPEG stream). The complete program highlighting the changes is provided in Appendix C “Video Capturing,” Listing C-1.
Building and Transferring the Video Capture Program

With the toolchain properly installed on your computer, open a terminal shell and type the following:

```
mcramon@ubuntu:~/ $ cd <YOUR BASE TOOLCHAIN PATH>
mcramon@ubuntu:~/xcompiler$ source environment-setup-*
```

This command will set all the variables of the environment in your system to the current installation of your toolchain.

To test if it is working, considering the programs created in this book to understand how V4L is written in C, run the following command in your computer shell:

```
mcramon@ubuntu:~/xcompiler$ ${CC} --version
```

```
i586-poky-linux-gcc (GCC) 4.7.2
Copyright (C) 2012 Free Software Foundation, Inc.  
This is free software; see the source for copying conditions.  There is NO 
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
```

The GCC compiler represented by `${CC}` was properly set and you are ready to build the program.

To build the program, you need to run the following:

```
${CC} -O2 -Wall 'pkg-config --cflags --libs libv4l2'
galileo_video_capture.c -o galileo_video_capture
```

Note that `pkg-config` is being used to inform the `libv4l2` what will be used in the compilation.

Now you can transfer the `galileo_video_capture` program using your favorite program, as explained in Chapter 5. For example, if you are using an Ethernet cable or a WiFi card in Intel Galileo and your operation system is Linux/MacOSX, you can use `scp`. If your computer is Windows, you can use WinSCP. Here’s an example using `scp`.

On the Intel Galileo terminal shell, create a direct connection like the following:

```
root@clanton:~# ifconfig eth0 192.254.1.1 netmask 255.255.255.0 up
```

Then on your Linux/MacOSX, transfer the file:

```
m cram on@ubuntu:~scp galileo_capture root@192.254.1.1:/home/root/.
```

If you are using a customized BSP image with the development tools present on your SD card image, it is easier because you can compile directly through Intel Galileo’s terminal shell. There is no needed to transfer files once the executable is already in the file system.
Running the Program and Capturing Videos

The program is executed in the command line and must run using the Intel Galileo terminal shell. The program accepts some arguments that are used to configure the capture type and redirect the content capture.

Among the arguments supported, this code supports all three IO methods and the method to be used during the capture is passed as a simple argument during the program execution. The arguments are as follows:

- **-m**: Memory mapped; used for the C270 webcam
- **-u**: Userspace pointers
- **-r**: Direct read/write

Another important argument is **-f**. If this argument is not set, the current camera settings are used to capture the video. This means you can change them using the `v4l2-ctl` tool, as explained.

If **-f** is used, the capture is forced to use a width of 1280, a height of 720, and the pixel format that supports the Motion JPEG stream (MJPEG). This is the only change from the original code, which supports a different format.

You also can define the number of frames you want to be part of your video with the argument **-c**. You need to use **-c <NUMBER OF FRAMES>**.

Finally, the **-o** argument redirects the content captured to an output that can be redirected to a file.

With all these arguments in mind, suppose you want to capture a video with 100 frames, force the resolution to 1280X720 using Motion JPEG encode, and create an output file named `video.mjpeg`. For this, you can execute the program with the following arguments using the Intel Galileo terminal shell:

```
root@clanton:~# ./galileo_video_capture -m -f -c 100 -o > video.mjpeg
```

Each dot “.” represents a frame captured in the output.

Now use a different setting using the `v4l2-ctl` command tool and run the same command line as before, but omit the **-f**. Let’s reduce the video resolution.

```
root@clanton:~# v4l2-ctl --set-fmt-video width=320,height=176,pixelformat=1
```

Then run the command to accept this configuration, by omitting the **-f** option.

```
root@clanton:~# ./galileo_video_capture -m -c 100 -o > video2.mpjpeg
```

You will have a new video but with a different resolution. If your device is not enumerated as /dev/video0, it is necessary to use the -d </dev/video*> option. For example, suppose your device is enumerated as /dev/video1. Your command line must then be:

```
root@clanton:~# ./galileo_video_capture -d /dev/video1 -m -c 100 -o > video2.mpjpeg
```

**Converting and Playing Videos**

If you read and worked through the previous section, you have two videos collected with the Motion JPEG encode and with different resolutions. If you transfer these files to your computer and try to play them you, will have a very sad surprise. They cannot be played because they are created for streaming, which means some specific headers in the files are missing.

To include the headers in the files, you must use an external tool. The recommended tools is ffmpeg, for three reasons:

- It’s vastly maintained by the open source community
- It can run directly on Intel Galileo SD image or on your computer
- It supports different encoders

In case you prefer to run it on your computer, the instructions for downloading and installing ffmpeg in different operational systems are found at [http://www.ffmpeg.org/download.html](http://www.ffmpeg.org/download.html). If your computer runs Linux, the easier way is to install it from static releases present at [http://ffmpeg.gusari.org/static/](http://ffmpeg.gusari.org/static/) and then run the following commands:

```
mcramon@ubuntu~$:~/mkdir ffmpeg;cd ffmpeg
mcramon@ubuntu~$:~/wget http://ffmpeg.gusari.org/static/64bit/ffmpeg.static.64bit.2014-03-02.tar.gz
mcramon@ubuntu~$:~/tar -zxvf ffmpeg.static.64bit.2014-03-02.tar.gz
```

ffmpeg will be available in the same directory you extracted the files.

On your personal computer or in Intel Galileo, you need to execute ffmpeg to convert the videos to a “playable” format in most systems. For example, to convert the first and second videos captured, you can run the following:

```
mcramon@ubuntu:~/video_samples$ ffmpeg -f mjpeg -i video.mjpeg -c:v copy video.mp4
```

ffmpeg version N-63717-g4e3fe65 Copyright (c) 2000-2014 the FFmpeg developers
built on Jun 3 2014 01:10:16 with gcc 4.4.7 (Ubuntu/Linaro 4.4.7-1ubuntu2)
configuration: --disable-yasm --enable-cross-compile --arch=x86
--target-os=linux
Chapter 7 ■ Using OpenCV

libavutil: 52.89.100 / 52.89.100
libavcodec: 55.66.100 / 55.66.100
libavformat: 55.42.100 / 55.42.100
libavdevice: 55.13.101 / 55.13.101
libavfilter: 4.5.100 / 4.5.100
libswscale: 2.6.100 / 2.6.100
libswresample: 0.19.100 / 0.19.100

Input #0, mjpeg, from 'video.mjpeg':
Duration: N/A, bitrate: N/A
Stream #0:0: Video: mjpeg, yuvj422p(pc), 1280x720, 25 fps, 25 tbr, 1200k tbn, 25 tbc
Output #0, mp4, to 'video.mp4':
Metadata:
encoder: Lavf55.42.100
Stream #0:0: Video: mjpeg (l[0][0][0] / 0x006C), yuvj422p, 1280x720, q=2-31, 25 fps, 1200k tbn, 1200k tbc
Stream mapping:
Stream #0:0 -> #0:0 (copy)
Press [q] to stop, [?] for help
frame= 100 fps=0.0 q=-1.0 Lsize= 2871kB time=00:00:03.96 bitrate=5939.9kbits/s
video:2870kB audio:0kB subtitle:0kB other streams:0kB global headers:0kB
muxing overhead: 0.041544%

mcramon@ubuntu:~/video_samples$ ffmpeg -f mjpeg -i video2.mjpeg -vcodec copy video2.mp4
ffmpeg version N-63717-g4e3fe65 Copyright (c) 2000-2014 the FFmpeg developers
built on Jun 3 2014 01:10:16 with gcc 4.4.7 (Ubuntu/Linaro 4.4.7-1ubuntu2)
configuration: --disable-yasm --enable-cross-compile --arch=x86
--target-os=linux
libavutil: 52.89.100 / 52.89.100
libavcodec: 55.66.100 / 55.66.100
libavformat: 55.42.100 / 55.42.100
libavdevice: 55.13.101 / 55.13.101
libavfilter: 4.5.100 / 4.5.100
libswscale: 2.6.100 / 2.6.100
libswresample: 0.19.100 / 0.19.100

Input #0, mjpeg, from 'video2.mjpeg':
Duration: N/A, bitrate: N/A
Stream #0:0: Video: mjpeg, yuvj422p(pc), 320x176, 25 fps, 25 tbr, 1200k tbn, 25 tbc
Output #0, mp4, to 'video2.mp4':
Metadata:
encoder: Lavf55.42.100
Stream #0:0: Video: mjpeg (l[0][0][0] / 0x006C), yuvj422p, 320x176, q=2-31, 25 fps, 1200k tbn, 1200k tbc
Stream mapping:

Stream #0:0 -> #0:0 (copy)
Press [q] to stop, [?] for help
frame= 100 fps=0.0 q=-1.0 Lsize= 843kB time=00:00:03.96
bitrate=1743.1kbits/s
video:841kB audio:0kB subtitle:0kB other streams:0kB global headers:0kB
muxing overhead: 0.137993%

Basically, -f indicates that the file is encoded as the Motion JPEG pixel format; the -i indicates an input file, and -vcodec copy maintains the same encode and quality, but adds the frames to a MP4 container.

All videos with the extensions of MJPEG and MP4 used in this section are present in the code/video_samples folder of this chapter. Thus you can exercise this conversion independently, with or without a webcam.

Videos with MP4 extensions can be played on your computer. If you are using Ubuntu, you can play them directly using the Movie Player. If you are using Windows or MacOSX, you can play them with the VLC player or using QuickTime (see Figure 7-5).

Figure 7-5. Videos captured by Intel Galileo reproduced in VLC and Movie Player

By comparing the files before and after the inclusion of headers by ffmpeg, you can see that the frames in general were maintained in the same content and only the headers were added to the end of MP4 file and a few bytes in the beginning. Figure 7-6 shows the header added to the end of the original MJPEG file.
A Program to Capture Images

The process for capturing images using V4L2 is the same as with videos except that frame-by-frame images should be saved in the file system.

The code used in this section is based on the software presented in “Appendix E. Video Grabber Example Using libv4l” of the *Linux Media Infrastructure API* documentation, with some changes to accommodate the webcam C270 and to choose different encodes. The complete program can be found at [http://linuxtv.org/downloads/v4l-dvb-apis/v4l2grab-example.html](http://linuxtv.org/downloads/v4l-dvb-apis/v4l2grab-example.html).

The software has some requirements that must be explained before you review the code:

- It is also a command-line program similar to the software used to capture videos that runs in the Intel Galileo’s terminal shell.
- It only supports the mapped memory IO method, and it is a simplified version of the software used to capture videos.

Remember that the software used to capture video was created to cover the whole scenario necessary to communicate with different types of devices. Limiting the software to memory mapped IO devices enables the code to be incredibly simplified. No argument in the command line is necessary because it is hard coded.
• It accepts different resolutions related to width and height. It accepts the `-w` or `--width` arguments for the image’s width and the `-H` or `--height` arguments for the image’s height. If the options are omitted, the default resolution is 1280x720 (width and height, respectively). If some resolution not supported by the webcam is defined through these arguments, the closest resolution is automatically selected. libv4l2 compares the ones supported by the camera and a warning message is displayed to the user in the terminal shell.

• It can select two different encodes, YUYV or RGB24. If the `-y` or `--yuyv` arguments are used, the YUYV encode is used; otherwise if this argument is omitted, the RGB24 encode is used by default.

• It is possible to set the number of images that will be stored in the files using the `-c` or `--count` argument, followed by the number of images desired. If this option is omitted, the number of images stored in the file system will be 10 by default. The image’s names will have the prefix `out`, followed by three decimals formatting the image order with the extension .ppm. For example, out000.ppm, out001.ppm, and so on.

You might wonder how it is possible to set the pixel format to RGB24 if it was demonstrated using `v4l2-ctl --list-formats` that the webcam C270 only supports MJPEG and YUYV? Actually, RGB24 is not supported by the webcam C270, but libv4l2 supports to conversion from YUYV to RGB24 and BGR24. In other words, even when you set the pixel format to RGB24 or BGR24, more precisely `V4L2_PIX_FMT_RGB24` and `V4L2_PIX_FMT_BGR24`, if your camera does not offer such formats natively, YUYV is considered and the V4L2 library makes the conversion from YUYV to RGB24 or BGR24 for you.

The following snippet demonstrates the selection of pixel format controlled by the variable `isYUYV`.

```c
CLEAR(fmt);
fmt.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
fmt.fmt.pix.width       = width;
fmt.fmt.pix.height      = height;

if (!isYUYV) {
    printf("Encode RGB24\n");
    fmt.fmt.pix.pixelformat = V4L2_PIX_FMT_RGB24;
} else {
    printf("Encode YUYV\n");
    fmt.fmt.pix.pixelformat = V4L2_PIX_FMT_YUYV;
}
```
fmt.fmt.pix.field = V4L2_FIELD_INTERLACED;
xioctl(fd, VIDIOC_S_FMT, &fmt);
if (fmt.fmt.pix.pixelformat != V4L2_PIX_FMT_RGB24 &&
    fmt.fmt.pix.pixelformat != V4L2_PIX_FMT_YUYV) {
    printf("Libv4l didn’t accept RGB24 or YUYV format. Can't
proceed.\n");
    exit(EXIT_FAILURE);
}
    if ((fmt.fmt.pix.width != width) || (fmt.fmt.pix.height != height))
        printf("Warning: driver is sending image at %dx%d\n",
            fmt.fmt.pix.width, fmt.fmt.pix.height);

If YUYV or RGB24 is selected, the image will have the same extension, .ppm. The
PPM file extension means “portable pixmap” and consists of a file with an ASCII header
following by a sequence of raw bytes.
A valid example of a PPM header file is:
   P6
   1280 720 255
   The string P6 is called the “magic identifier” and it can be P3 as well. Then the next
line contains the image width and height, represented by 1280 and 720 in this example.
The number 255 is the maximum value of the color component RGB, so it might vary
between 0 to 255. Thus it’s used to delimit the maximum color range in the image.
The following code lines can be used to create the file with fopen() and write this
string sequence in this file using fprintf().
   fout = fopen(out_name, "w");
   ...
   ...
   fprintf(fout, "P6\n%d %d 255\n", fmt.fmt.pix.width, fmt.fmt.pix.height);

The maximum color component in the header is retriwed by the image returned by the
webcam and that’s why it is not part of fprintf() function.
If you selected the RGB24 and the image provided by libv4l2 is already converted
from YUYV to RGB24, you simply need to write the header text for the PPM image and
attach the binary data received when the buffers were “dequeued” through VIDIOC_DQBUF.
   xioctl(fd, VIDIOC_STREAMON, &type);
   for (i = 0; i < images_count; i++) {
        do {
            ...
            ...
            CLEAR(buf);
            buf.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
            buf.memory = V4L2_MEMORY_MMAP;
            xioctl(fd, VIDIOC_DQBUF, &buf);
If YUYV is selected, you must convert from YUYV to RGB24 in order to create the .ppm file.

In the case of webcam C270, the YUYV is 4:2:2, which means 4 bytes per 2 pixels; or 1 pixel per 2 bytes. RGB24 represents 24 bits per pixel or 3 bytes per pixel. Thus, to convert YUYV 4:2:2 to RGB24 each 2 bytes (1 pixel) from YUYV 4:2:2 originates to 3 bytes (pixel) to RGB24.

It means the function that converts the image needs to allocate a buffer because the resultant image in RGB24 is 1.5 times bigger than the YUYV 4:2:2 image.

So, if in RGB24 each pixel is represented by three bytes, the buffer that will receive the image allocated must be evaluated. You get the total number of pixels and multiply that by three bytes each. The following code represents this logic:

```c
// each pixel 3 bytes in RGB 24
int size = fmt.fmt.pix.width * fmt.fmt.pix.height * sizeof(char) * 3;
unsigned char * data = (unsigned char *) malloc(size);
```

To make this conversion, the code example uses a function extracted from OpenCV, copied without any changes under Intel licenses. The function is called `yuyv_to_rgb24()` and it is from the file `cvicap_v4l.cpp`. For reference, you can see the whole file at https://code.ros.org/trac/opencv/browser/trunk/opencv/src/highgui/cvicap_v4l.cpp?rev=284.

The following snippet makes the conversion function call by passing the image's dimensions (`fmt.fmt.pix.width` and `fmt.fmt.pix.height`), the initial buffer address (`buffers[buf.index].start`) to the current frame returned, and the destination buffer allocated (`data`).

```c
... ... ... 
fwrite(buffers[buf.index].start, buf.bytesused, 1, fout);
... ... ...
fclose(fout);

xioctl(fd, VIDIOC_QBUF, &buf);
```
The software is shown in Listing D-1. The portions of the code relevant to this section are in bold.

The code excerpts used in this section are based on the software presented in “Appendix E. Video Grabber Example Using libv4l” of the Linux Media Infrastructure API documentation, with some changes to accommodate the webcam C270 and to choose different encodes (http://linuxtv.org/downloads/v4l-dvb-apis/v4l2grab-example.html). The complete program highlighting the changes discussed in this section are provided in Appendix D “Picture Grabber,” Listing D-1.

Building and Transferring the Picture Grabber

The procedure for building and transferring the file is the same as the one used in the section “Building and Transferring the Video Capture Program” of this chapter, except for the command line used to compile the picture grabber program.

Type the following into the command line with the toolchain properly set.

```
${CC} -O2 -Wall 'pkg-config --cflags --libs libv4l2'
picture_grabber.c -o picture_grabber
```

Then transfer the file to your Intel Galileo board.

Running the Program and Capturing Images

The program is executed in the command line and can be used with Intel Galileo terminal shell. The program accepts some arguments explained in the previous section.

The first step is to capture five images in RGB24 format with a resolution of 352x288.

```
root@clanton:~# ./picture_grabber -W 352 -H 288 -c 5
Encode RGB24
Creating image: out000.ppm
Creating image: out001.ppm
Creating image: out002.ppm
Creating image: out003.ppm
Creating image: out004.ppm
```
As a result, five images with the prefix out and extension ppm are created. Copy these images to your computer and open them using an image viewer.

If you try to use a resolution not supported by the webcam, the V4L2 library will adjust to the closest one supported by the camera and a warning message will appear informing you of the real resolution used. For example, 300x200 is not supported:

```bash
root@clanton:~# ./picture_grabber -W 300 -H 200 -c 5
Encode RGB24
Warning: driver is sending image at 176x144
Creating image: out000.ppm
Creating image: out001.ppm
Creating image: out002.ppm
Creating image: out003.ppm
Creating image: out004.ppm
```

If you try to run the problem using YUYV encode, just add the -y or --yuyv argument to it:

```bash
root@clanton:~# ./picture_grabber -W 352 -H 288 -c 5 -y
Encode YUYV
Creating image: out000.ppm
Creating image: out001.ppm
Creating image: out002.ppm
Creating image: out003.ppm
Creating image: out004.ppm
```

**Working with OpenCV**

At this moment you have explored your webcam, you understand how to load the drivers, adjust the resolutions and other settings on your camera, you are aware of the type of formats (encodes) that are supported, and you have idea how Video4Linux works.

Now it is time to start exercising some applications created using OpenCV.

As mentioned in the beginning of this chapter, the topic of OpenCV is worthy of a whole book, and there are several books available with this purpose. The idea here is to learn what is possible with Intel Galileo and OpenCV, to compare performance between C++ and Python, and to identify when a problem is related to OpenCV or if it is related to the wrong settings in Video4Linux.

---

**Note**  The examples demonstrated in this chapter are in C++ and Python. OpenCV also supports C, which is not explored here. This is because the C++ interface created for OpenCV is simpler than the C language interface, which requires you to manage memory allocations.
Building Programs with OpenCV

To build programs that run OpenCV, you must follow the same process you followed when compiling the program for Video4Linux in the previous sections. In other words, it is necessary to set the toolchain and run the proper command line.

The procedure is the same as the one outlined in the “Building and Transferring the Video Capture Program” section of this chapter, except the command line changes a little bit because the programs are written in C++ instead of C, and it is necessary to invoke the OpenCV libs instead of the V4L2.

For example, to build the program used in the next section listed as Listing 7-1 and named opencv_capimage.cpp, use the following line:

```
${CXX} -O2 'pkg-config --cflags --libs opencv' opencv_capimage.cpp -o opencv_capimage
```

`${CXX}` invokes the C++ compiler (g++) of the toolchain and `pkg-config` invokes the OpenCV libs.

Once this compiles, you just transfer the program to Intel Galileo if the toolchain is not installed directly on your Intel Galileo.

Capturing an Image with OpenCV

Capturing an image using OpenCV is very simple because all the complexity is abstracted by OpenCV, which uses V4L2 as a baseline.

Figure 7-7 shows the flowchart used to capture images and videos and process the images.
Figure 7-7. OpenCV flowchart to capture a frame
Listing 7-1 shows an example of how to capture an image and store it in the file system as a JPEG file.

Listing 7-1. opencv_capimage.cpp

```cpp
#include <opencv2/opencv.hpp>
using namespace cv;
using namespace std;

int main()
{
    VideoCapture cap(-1);
    //check if the file was opened properly
    if(!cap.isOpened())
    {
        cout << "Webcam could not be opened succesfully" << endl;
        exit(-1);
    }
    else
    {
        cout << "p n" << endl;
    }
    int w = 960;
    int h = 544;
    cap.set(CV_CAP_PROP_FRAME_WIDTH, w);
    cap.set(CV_CAP_PROP_FRAME_HEIGHT, h);
    Mat frame;
    cap >> frame;
    imwrite("opencv.jpg", frame);
    cap.release();
    return 0;
}
```

Reviewing opencv_capimage.cpp

This first example uses a few objects based in the following classes. VideoCapture is used to create the capture objects that open and configure the devices, capture images and videos, and release the devices when they are not in use any more. Mat receives frames read and works with some algorithms to process the images. It can apply filters, change colors, and transform the images according to mathematical and statistical algorithms. In the next example, Mat is used only to read the image. However, in the next couple of examples, Mat will be used to process images as well.

To understand the code in Listing 7-1, you need a quick overview of each class used.
VideoCapture:: VideoCapture

The first thing to do is to use the VideoCapture class to create a video capture object and open the device or some video stored in the files system.

For more information regarding the VideoCapture class, see http://docs.opencv.org/modules/highgui/doc/reading_and_writing_images_and_video.html.

In the case of the webcam, you will create the object with the parameter -1 in the constructor, as follows:

```cpp
VideoCapture cap(-1);
```

The value -1 means, “open the current device enumerated in the system,” so if you have the camera enumerated as /dev/video0 or /dev/video1, the webcam will be opened anyway. Otherwise, if you want to be specific regarding which device to open, you have to pass to the constructor the index of the enumerated device. For example, to open the device /dev/video0, you must pass the number 0 to the constructor like this:

```cpp
VideoCapture cap(0);
```

If you’re using Intel Galileo and one camera, I recommend you use -1 to avoid problems with camera enumeration indexes versus the hardcoded number you use in the constructor.

VideoCapture::isOpened()

You can check if the webcam was opened and initiated with success by invoking the isOpened() method. It returns a Boolean as true if the webcam was opened and false if not.

VideoCapture::set(const int prop, int value)

This method sets a property (prop) to a specific value (value). You can set the image’s width, height, frames per second, and several other properties. In the code example, the video width and height are set to 960x544:

```cpp
int w = 960;
int h = 544;
cap.set(CV_CAP_PROP_FRAME_WIDTH, w);
cap.set(CV_CAP_PROP_FRAME_HEIGHT, h);
```

For more information about the properties supported, visit http://nullege.com/codes/search/opencv.highgui.CV_CAP_PROP_FPS.
VideoCapture::read(Mat & image) or operator >> (Mat & image)

This method reads the image from the device. It grabs the image in one single call. The return is a Mat object that is explained shortly.

This example uses the operator >>:

Mat frame;
cap >> frame;

VideoCapture::release()

Once the video is captured, if the destructor of the object is not called, you must release the camera by invoking the release() method.

cap.release();

At a glance, you can see how simple this is, compared to the software used when we were focusing on Video4Linux.

cv::Mat::Mat

Mat is an awesome class used for matrix operations and it is constantly used in OpenCV applications. Mat is used to organize images in the format of matrixes responsible for saving details of each pixel, including color intensity, position in the image, image dimension, and so on.

The Mat class is organized into two parts—one part contains the image headers with generic information about the image and the second part contains the sequence of bytes representing the image.

In the code example, Mat is called only as Mat instead of as cv::Mat because the namespace was defined in the beginning of the code:

using namespace cv;

Also, in the code example, there is a Mat object created with the simple constructors available in the class:

Mat frame;

In the next examples, other methods will be used and properly discussed. For now, keep in mind what the Mat class is for and this simple constructor.

For more details regarding the Mat class, visit http://docs.opencv.org/modules/core/doc/basic_structures.html#mat-mat. The tutorial maintained by docs.opencv.org is also recommended at http://docs.opencv.org/doc/tutorials/core/mat_the_basic_image_container/mat_the_basic_image_container.html.
cv::imwrite( const string& filename, InputArray img, const vector<int>& params=vector<int>())

This method saves an image to the file system. In the code example, the file is opencv.jpg, the input array is intrinsically casted as my Mat class with the object frame, and the optional vector of the params argument is omitted.

Mat frame;
cap >> frame;
imwrite("opencv.jpg", frame);

In this case, with the omission of the vector of the params argument, the encode used to capture the image is based on the file extension .jpg. Remember that the camera does not support capturing images in the JPEG format. It captures streaming in Motion JPEG but JPEG is not extracted from Motion JPEG because there is a segment called DHT that’s not present in this stream (check out [http://www.digitalpreservation.gov/formats/fdd/fdd000063.shtml](http://www.digitalpreservation.gov/formats/fdd/fdd000063.shtml)). You can extract a series of JPEG images using ffmpeg from a Motion JPEG streaming file, but they will not be viewable in any image software due to the DHT segment missing.

In other words, when the file extension is specified and not supported by the webcam, the OpenCV framework converts the file.

The extensions supported besides JPEG are PNG, PPM, PGM, and PBM.

The docs.opencv.org site maintains a nice tutorial about how to load, modify, and save an image at [http://docs.opencv.org/doc/tutorials/introduction/load_save_image/load_save_image.html](http://docs.opencv.org/doc/tutorials/introduction/load_save_image/load_save_image.html).

Running opencv_capimage.cpp

Compile the code and transfer the file to Intel Galileo. Make sure the uvcvideo driver is loaded and the webcam is connected to the USB port (read the section called “Connecting the Webcam” in this chapter). Finally, smile at your webcam and run the software:

root@clanton:~# ./opencv_capimage
VIDIOC_QUERYMENU: Invalid argument
VIDIOC_QUERYMENU: Invalid argument
VIDIOC_QUERYMENU: Invalid argument
VIDIOC_QUERYMENU: Invalid argument
VIDIOC_QUERYMENU: Invalid argument
VIDIOC_QUERYMENU: Invalid argument
VIDIOC_QUERYMENU: Invalid argument
VIDIOC_QUERYMENU: Invalid argument
Webcam is OK! I found it!

VIDIOC_QUERYMENU: Invalid argument
VIDIOC_QUERYMENU: Invalid argument
VIDIOC_QUERYMENU: Invalid argument
VIDIOC_QUERYMENU: Invalid argument
You should have a file named opencv.jpg in the same folder. Now, you might asking what the VIDIOC_QUERYMENU: Invalid argument message mean. Such messages are not related to OpenCV and there is nothing wrong with the code. It is simply OpenCV using the Video4Linux framework to understand the capabilities and controls offered by the webcam. When some control or capability is not offered, V4L informs you with these warning messages.

If you do not want to see these messages, you can redirect them using a stderr stream to a null device. For example:

root@clarton:~# ./opencv_capimage 2> /dev/null
Webcam is OK! I found it!

The Same Software Written in Python

You can use Python with OpenCV because the Python Opencv development packages are part of the BSP SD card images introduced in this chapter.

The program in Listing 7-1 can easily be converted to Python, as demonstrated by Listing 7-2.

Listing 7-2. opencv_capimage.py

```python
import cv2
import cv
import sys

cap = cv2.VideoCapture(-1)

w, h = 960, 544
cap.set(cv.CV_CAP_PROP_FRAME_WIDTH, w)
cap.set(cv.CV_CAP_PROP_FRAME_HEIGHT, h)

if not cap.isOpened():
    print "Webcam could not be opened successfully"
sysexit(-1)
else:
    print "Webcam is OK! I found it!"

ret, frame = cap.read()

cv2.imwrite('pythontest.jpg', frame)

cap.release()
```
As you can see, the objects are quite the same. To run the software, transfer to Intel Galileo board and run the following in the terminal shell:

```
root@clanton:~# python opencv_capimage.py 2> /dev/null
Webcam is OK! I found it!
```

However, the examples in this chapter are written in C++. This is because code written in C++ runs significantly faster than the same code written in Python.

**Performance of OpenCV C++ versus OpenCV Python**

To check for performance issues, suppose you have the Python program shown in Listing 7-2 and the C++ program shown in Listing 7-1 properly installed on Intel Galileo. You can measure performance using the bash terminal with the command `date +%s`, which returns the number of seconds passed since 00:00:00 1970-01-01 UTC. Execute the program and evaluate the time difference.

First, run the Python program with the following command:

```
root@clanton:~# s=$(date +%s);python opencv_capimage.py; echo $(expr 'date +%s' - $s)
Webcam is OK! I found it!
8
```

Python took eight seconds to take the picture. Do the same thing with the C++ program:

```
root@clanton:~# s=$(date +%s);./opencv_capimage 2> /dev/null; echo $(expr 'date +%s' - $s)
Webcam is OK! I found it!
4
```

The same program written in C++ took only four seconds. Even the programs running in the userspace context suffer some time execution variation because it’s not a real-time system. The OpenCV applications created in C++ are much faster than the same applications running in Python.

**Processing Images**

In the previous section, you captured images from the webcam and saved them into the files system, but no image processing was done. The next examples explore some of the infinite possibilities of image processing using OpenCV. Some of them use a huge algorithm in the background and it is not in the scope of this book to discuss the details of each one. However, references are included for more information.
Detecting Edges

For the first example of image processing, you'll learn about how to detect images using the Canny edge algorithm developed by John F. Canny in 1986.

OpenCV has a function called Canny() that implements such an algorithm. For details about this algorithm, see http://docs.opencv.org/doc/tutorials/imgproc/imgtrans/canny_detector/canny_detector.html.

With a few changes to Listing 7-1, the Canny algorithm is shown in Listing 7-3.

Listing 7-3. opencv_capimage_canny.cpp

```cpp
#include <opencv2/opencv.hpp>
using namespace cv;
using namespace std;

int main()
{
    VideoCapture cap(-1);

    // check if the file was opened properly
    if(!cap.isOpened())
    {
        cout << "Webcam could not be opened successfully" << endl;
        exit(-1);
    }
    else
    {
        cout << "Webcam is OK! I found it!\n" << endl;
    }

    int w = 960;
    int h = 544;
    cap.set(CV_CAP_PROP_FRAME_WIDTH, w);
    cap.set(CV_CAP_PROP_FRAME_HEIGHT, h);

    Mat frame;
    cap >> frame;

    // converts the image to grayscale
    Mat frame_in_gray;
    cvtColor(frame, frame_in_gray, CV_BGR2GRAY);

    // process the Canny algorithm
    cout << "processing image with Canny..." << endl;
    int threshold1 = 0;
    int threshold2 = 28;
    Canny(frame_in_gray, frame_in_gray, threshold1, threshold2);
```
// saving the images in the files system

    cout << "Saving the images..." << endl;
    imwrite("captured.jpg", frame);
    imwrite("captured_with_edges.jpg", frame_in_gray);

    // release the camera
    cap.release();

    return 0;
}

**Reviewing opencv_capimage_canny.cpp**

In this example the following is changed in Listing 7-3:

1. A new static method called cvtColor() is added.
2. The Canny() function is used for image processing.

The image originally captured by the camera and the image processed with Canny algorithm are both stored in the file system as captured.jpg and captured_with_edges.jpg using the imwrite() function explained previously.

```cpp
cvoid cv::cvtColor(InputArray src, OutputArray dst, int code, int dstCn=0)

Converts the image space color to another one. In the following code example:

```cpp
Mat frame_in_gray;
cvtColor(frame, frame_in_gray, CV_BGR2GRAY);
```

The input image was the one captured by the webcam and stored in the Mat object frame. The frame_in_gray object was created to receive the image converted in gray space color as requested by code CV_BGR2GRAY.

For more detail about the cvtColor() function and color in general, visit [http://docs.opencv.org/modules/imgproc/doc/miscellaneous_transformations.html#cvtcolor](http://docs.opencv.org/modules/imgproc/doc/miscellaneous_transformations.html#cvtcolor).

```cpp
cvoid cv::Canny(InputArray image, OutputArray edges, double threshold1, double threshold2, int apertureSize=3, bool L2gradient=false)
```

The Canny function takes the image input array as a source image and transforms the edges into sharp ones. It stores the output array in edges. The input and output image in the example is the same object (frame_in_gray); for best effect, a grayscale image is used.
The apertureSize argument is the size of the Sobel operator used in the algorithm (see http://en.wikipedia.org/wiki/Sobel_operator for more details) and the code keeps the default value of 3.

The L2gradient argument is a Boolean; when it’s true, the image gradient magnitude is used and when it’s false, only the normative equation is considered. This example used the default value of false.

Two hysteresis thresholds are represented by the arguments threshold1 and threshold2 and the values 0 and 28 were used, respectively. These values are based on my experiments with changing these values until I got results I considered good. You can change these values and check the effects you get.

```c
int threshold1 = 0;
int threshold2 = 28;
Canny(frame_in_gray, frame_in_gray, threshold1, threshold1);
```

The official documentation about Canvas function is found on this link “http://docs.opencv.org/modules/imgproc/doc/feature_detection.html?highlight=canny#canny.”

Running opencv_capimage_canny.cpp

Compile the code and transfer the file to Intel Galileo. Make sure the uvcvideo driver is loaded and the webcam is connected to the USB port (read the section entitled “Connecting the Webcam” in this chapter). Point your webcam to some object rich in edges, like the image shown in Figures 7-8 and 7-9.

```
root@clanton:~# ./opencv_capimage_canny 2> /dev/null
Webcam is OK! I found it!

processing image with Canny...
Saving the images...
```

*Figure 7-8. Original image before the Canny effect*
You should have two images stored in the file system as `captured.jpg` and `captured_with_edges.jpg`.

**Face and Eyes Detection**

This next example detects multiples faces and eyes in a picture captured using the webcam. The class used to detect the faces and eyes is named `CascadeClassifier`.

The basic concept is that this class loads some XML files that use the `classifier model`. In the code, two files are loaded—called `haarcascade_frontalface_alt.xml` and `haarcascade_eye.xml`—during the creation of the `CascadeClassifier` objects. Each file brings a series of models that defines how specific objects are represented in the image based on the sum of intensity of pixels in a series of rectangles. The difference of these sums is evaluated in the image. Both files have characteristics about faces and eyes from an image and the class `CascadeClassifier` can determine the detections when the method `detectMultiScale()` is invoked.


When a face is detected a rectangle will be drawn around the face and when the eyes are detected circles will be drawn around the eyes. These drawing are done using very basic draw functions in OpenCV called `rectangle()` and `circle()`.
Listing 7-4 shows the code for this example.

Listing 7-4. opencv_face_and_eyes_detection.cpp

```cpp
#include <opencv2/opencv.hpp>
#include "opencv2/core/core.hpp"

using namespace cv;
using namespace std;

String face_cascade_name = "haarcascade_frontalface_alt.xml";
String eye_cascade_name = "haarcascade_eye.xml";

void faceDetect(Mat img);
CascadeClassifier face_cascade;
CascadeClassifier eyes_cascade;

int main(int argc, const char *argv[]) {
  if( !face_cascade.load( face_cascade_name ) )
  {
    cout << face_cascade_name << " not found!! aborting..." << endl;
    exit(-1);
  }

  if( !eyes_cascade.load( eye_cascade_name ) )
  {
    cout << eye_cascade_name << " not found!! aborting..." << endl;
    exit(-1);
  }

  // 0 is the ID of the built-in laptop camera, change if you want to use
  // other camera
  VideoCapture cap(-1);

  //check if the file was opened properly
  if(!cap.isOpened())
  {
    cout << "Capture could not be opened succesfully" << endl;
    return -1;
  }
  ```
else
{
    cout << "camera is ok\n" << endl;
}

int w = 432;
int h = 240;
cap.set(CV_CAP_PROP_FRAME_WIDTH, w);
cap.set(CV_CAP_PROP_FRAME_HEIGHT, h);

Mat frame;
cap >> frame;

cout << "processing the image...." << endl;

faceDetect(frame);
imwrite("face_and_eyes.jpg", frame);

// release the camera
cap.release();

cout << "done!" << endl;
return 0;
}

void faceDetect(Mat img)
{
    std::vector<Rect> faces;
    std::vector<Rect> eyes;
    bool two_eyes = false;
    bool any_eye_detected = false;

    //detecting faces
    face_cascade.detectMultiScale(img, faces, 1.1, 2, 0|CV_HAAR_SCALE_IMAGE,
    Size(30, 30) );

    if (faces.size() == 0)
    {
        cout << "Try again.. I did not dectected any faces..." << endl;
        return;
    }

    // it is possible to face more than one human face in the image
    for( size_t i = 0; i < faces.size(); i++ )
    {

// rectangle in the face
rectangle( img, faces[i], Scalar(255, 100, 0), 4, 8, 0);

Mat frame_gray;
cvtColor( img, frame_gray, CV_BGR2GRAY );

// cropping only the face in region defined by faces[i]
std::vector<Rect> eyes;
Mat faceROI = frame_gray( faces[i] );

// In each face, detect eyes
eyes_cascade.detectMultiScale( faceROI, eyes, 1.1, 2, 0 |
| CV_HAAR_SCALE_IMAGE, Size(30, 30) );

for( size_t j = 0; j < eyes.size(); j++ )
{
    Point center( faces[i].x + eyes[j].x + eyes[j].width*0.5,
faces[i].y + eyes[j].y + eyes[j].height*0.5 );
    int radius = cvRound( (eyes[j].width + eyes[j].height)*0.25 );
    circle( img, center, radius, Scalar(255, 0, 0), 4, 8, 0 );
}

Reviewing opencv_face_and_eyes_detection.cpp

In this example there are a few new components:

- Introduction of the CascadeClassifier class.
- Usage of the Point class
- The cvRound() function
- Usage of the rectangle() and circle() functions
- The Rect class and vectors

The following sections provide an explanation of each item used in the code.

cv::CascadeClassifier::CascadeClassifier()

Creates the CascadeClassifier object. In the example code, two objects are created, one
to detect the face and the other to detect the eyes.

CascadeClassifier face_cascade;
CascadeClassifier eyes_cascade;
cv::CascadeClassifier::load(const string & filename)

 Loads the file with the classifier to the object. In the code, two classifiers were used, one to detect the face and the other to detect the eyes.

 if( !face_cascade.load( face_cascade_name ) )
 {
   cout << face_cascade_name << " not found!! aborting..." << endl;
   exit(-1);
 }

 if( !eyes_cascade.load( eye_cascade_name ) )
 {
   cout << eye_cascade_name << " not found!! aborting..." << endl;
   exit(-1);
 }

 void cv::CascadeClassifier::detectMultiScale(const Mat& image, vector<Rect>& objects, double scaleFactor=1.1, int minNeighbors=3, int flags=0, Size minSize=Size(), Size maxSize=Size())

 The `detectMultiScale()` method is where the magic happens in terms of detections. A description of each argument follows:

- **image** is the image source.
- **vector<Rect>& objects** is a vector of rectangles and is where the object detects are in the image.
- **scaleFactor** is a factor that determines if the image is reduced.
- **minNeighbors** determines how many neighbors each candidate rectangle has. If 0 is passed, there is a risk of other objects in the image being detected incorrectly, which results in false positives in the detection. For example, if you have a clock on your wall it might be detected as a face (a false positive). During my practical experiments, specifying 2 or 3 is good. More than 3 and there is a risk of losing true positives and faces not being detected properly.
- **flags** is related to the type of optimization. `CV_HAAR_SCALE_IMAGE` tells the algorithm to be in charge of the scaled image. This flag accepts `CV_HAAR_DO_CANNY_PRUNNING`, which skips flat regions, `CV_HAAR_FIND_BIGGEST_OBJECT` if there is interest in finding the biggest object in the image, and `CV_HAAR_DO_ROUGH_SEARCH`, which must be used only with `CV_HAAR_FIND_BIGGEST_OBJECT` like "0|CV_HAAR_DO_ROUGH_SEARCH |CV_HAAR_FIND_BIGGEST_OBJECT".
• SizeminSize defines the minimum object size and objects smaller than this are ignored. If it’s not defined this argument is not considered.

• maxSize defines the maximum object size and objects bigger than this are ignored. If it’s not defined this argument is not considered.

```cpp
//detecting faces
face_cascade.detectMultiScale( img, faces, 1.1, 2, 0|CV_HAAR_SCALE_IMAGE, Size(30, 30) );

...  
... 

//In each face, detect eyes
eyes_cascade.detectMultiScale( faceROI, eyes, 1.1, 2, 0 |CV_HAAR_SCALE_IMAGE, Size(30, 30) 

In this code, the scaling factor used is 1.1, minNeighbors is 2 (a kind of hint), the flags are optimized for performance using CV_HAAR_SCALE_IMAGE, and the minimum size of the object to detect is 30x30 pixels. No maximum size is defined, so you can put your face very close to the webcam.

The code detected the faces in the image. For each face that’s detected, a rectangle is drawn delimiting the region.

// rectangle in the face
rectangle( img, faces[i], Scalar( 255, 100, 0 ), 4, 8, 0 );

The resulting regions containing the faces detected are stored in vector<Rect> faces. For example, faces[0] is the first face in the picture. If there is more than one person, you will have face[1], face[2], and so on. The object type Rect means rectangle so the faces vector is a group of rectangles without graphical objects. They are objects that store the initial coordinates (upper-left points) in (Rec.x,Rec.y) and the width (Rec.w) and height (Rec.h) of the rectangle in the object class.

For each region detected, a new image is created with the image content delimited by the rectangle, which forms a small area. This small area is called the ROI (Region of Interest). For best performance and to normalize the image in the eye, detection is converted to grayscale using the cvColor() function.

Mat frame_gray;
cvtColor( img, frame_gray, CV_BGR2GRAY );
// cropping only the face in region defined by faces[i]
std::vector<Rect> eyes;
Mat faceROI = frame_gray( faces[i] );
In this small area that contains only the face, the cascade classifier tries to identify the eyes. For each eye detected, a circle is drawn. So the while the face uses the whole image to be detected, the eyes are detected only in the face regions. This optimizes the algorithm.

```cpp
// In each face, detect eyes
eyes_cascade.detectMultiScale( faceROI, eyes, 1.1, 2, 0 |
| CV_HAAR_SCALE_IMAGE, Size(30, 30) );
```

The resultant regions containing the eyes is stored in `vector<Rect> eyes`. This process is done with the `for` loops in this code:

```cpp
for( size_t i = 0; i < faces.size(); i++ )
{
    ...
    ...
    for( size_t j = 0; j < eyes.size(); j++ )
    {
        ...
        ...
    }
}
```

To draw the circles around the eyes, the `Point` class was used. It extracts information from `vector<Rect> eyes` and stores the exact center of the eyes (the central coordinates):

```cpp
Point center( faces[i].x + eyes[j].x + eyes[j].width*0.5, faces[i].y +
eyes[j].y + eyes[j].height*0.5 );
int radius = cvRound( (eyes[j].width + eyes[j].height)*0.25 );
circle( img, center, radius, Scalar( 255, 0, 0 ), 4, 8, 0 );
```

Thus, the `Point center` object is based on the rectangle's dimension of the current face, identified by the center point of the eye and the variable `radius`. Using the function `cvRound()`, it determines the radius to be drawn around the eyes.

With those two information, it is possible to draw a circle using the function `circle()`.

Figure 7-10 shows this code's sequence.
main

detectMultiScale() for faces

i=0
This is the index of the faces counter

Draw a rectangle on each face

Create a new image delimited by the rectangle area (ROI)

Convert the image to gray color space

detectMultiScale() for eyes

J=0
This is the index of the eyes counter

Draw a circle on each eye detected

Are there more eyes in the current face?

YES

j=j+1
Move to next eye

NO

Are there more faces in the current image?

YES

i=i+1
Move to next face

NO

END

Figure 7-10. Flowchart for detecting faces and eyes
Running opencv_face_and_eyes_detection.cpp

Compile the code and transfer the file to Intel Galileo. Make sure the uvcvideo driver was loaded and the webcam is connected to the USB port (read the section called “Connecting the Webcam” in this chapter) and copy the haarcascade_frontalface_alt.xml and haarcascade_eye.xml files to the same location you transferred the executable program. Stay in front of camera and look in the direction of the lens. Then run the software:

root@clanton:~# ./opencv_face_and_eyes_detection 2> /dev/null

camera is ok
processing the image....
done!

A image named face_and_eyes.jpg is created in the files system wilt all faces and eyes detected, as shown in Figure 7-11.

![Figure 7-11. More than one face and multiples eyes have been detected](image)

Emotions Classification

The methods shown in this section and some of the scripts are based on the work of Phillip Wagner in article “Gender Classification with OpenCV,” which you can find at [http://docs.opencv.org/trunk/modules/contrib/doc/facerec/tutorial/facerec_gender_classification.html](http://docs.opencv.org/trunk/modules/contrib/doc/facerec/tutorial/facerec_gender_classification.html). Phillip Wagner kindly granted permission for the code adaptation and the techniques explored, keeping all the code under the BSD licenses as his original work in this book.

The original code was changed in order to:

- Run on Intel Galileo and classify emotions instead of genders.
- Use faces and eyes detection directly from the images captured by the webcam.
- Crop the images dynamically based on human anatomy.
The emotions classifications in this example are divided into three categories:

- Happy
- Sad
- Curious

The idea is that you take pictures with the webcam, and Intel Galileo will try to describe your emotional state.

You need to create a database with images of you showing each emotional state. This database will contain images that use specific algorithms explained later. These images are used as references to allow Intel Galileo to determine your emotions while you look at the webcam through a model named fisherface.

The database in this chapter is based on my face, but there are instructions for recreating the database based on your face. If you run the program using this section, there is a remote chance that it will recognize your emotions (if you are lucky enough to look like me). Okay, if you look like me, you are not necessarily lucky (ha ha).

**Preparing the Desktop**

You need to create a database with a few pictures of you. The process for generating this database is explained in detail in conjunction with some scripts that run in Python.

It’s necessary to have Python installed on your computer, with the pillow and setuptools modules installed.

Pillow is used to treat images using Python scripts and setuptools is a dependence that pillow requests. You should install the setuptools module first.

Pillow can be downloaded from https://pypi.python.org/pypi/Pillow and the setuptools module can be downloaded from https://pypi.python.org/pypi/setuptools. Both sites include information on how to install these modules on Linux, Windows, and MacOSX.

You will also need an image editor because it’s necessary to take some pictures of your face with different emotions and identify the coordinates of the center of each of your eyes. You can use Paint in Windows, Gimp on Linux/OSX and Windows, or any other software that allows you to move the mouse cursor in the image and obtain the coordinates.

You can download Gimp from http://www.gimp.org/.

**Creating the Database**

Follow these steps to create the database:

1. Obtain the initial images.
2. Crop the images.
3. Organize the images in directories.
4. Create the CSV file.

Let’s look at each step in more detail.
Obtaining the Initial Images

This example uses three emotions—happy, sad, and surprised. That means the database must contain at least three pictures of you of each state.

Such pictures must be obtained using your webcam. It doesn’t matter if you obtain the images with Intel Galileo using the code examples described previously, or if you connect the webcam to your computer and take the pictures using other software. The most important thing is to take at least three pictures of each emotion—sad, surprised, and happy—totalizing nine pictures. I recommend you take these pictures at a resolution of 1280x1024 or 1280x720. The images will be cropped and reduced and it is important to maintain the images with good definition after these changes.

In the initial_pictures subfolder of the code folder of this chapter, there are some pictures of me of each emotion. For each picture the pixel coordinates of the center of my eyes were taken—see Table 7-2.

Table 7-2. Central Coordinate of Each Eye on Each Emotional State

<table>
<thead>
<tr>
<th>Picture</th>
<th>Left Eye Center (x,y)</th>
<th>Right Eye Center (x,y)</th>
</tr>
</thead>
<tbody>
<tr>
<td>serious_01.jpg</td>
<td>528, 423</td>
<td>770, 431</td>
</tr>
<tr>
<td>serious_02.jpg</td>
<td>522, 412</td>
<td>758, 415</td>
</tr>
<tr>
<td>serious_03.jpg</td>
<td>518, 423</td>
<td>754, 425</td>
</tr>
<tr>
<td>smile_01.jpg</td>
<td>516, 377</td>
<td>753, 379</td>
</tr>
<tr>
<td>smile_02.jpg</td>
<td>533, 374</td>
<td>763, 380</td>
</tr>
<tr>
<td>smile_03.jpg</td>
<td>518, 379</td>
<td>749, 381</td>
</tr>
<tr>
<td>surprised_01.jpg</td>
<td>516, 356</td>
<td>754, 355</td>
</tr>
<tr>
<td>surprised_02.jpg</td>
<td>548, 364</td>
<td>793, 364</td>
</tr>
<tr>
<td>surprised_03.jpg</td>
<td>528, 377</td>
<td>770, 378</td>
</tr>
</tbody>
</table>

Be expressive when you take the pictures. Otherwise, it will be more difficult for the program to guess your emotional states.

Cropping the Images

The next step is to crop the images, removing the ears and hair, and try to generate 20x20 images with only the faces showing. The Python script that was initially created for gender classification was adapted to emotion classification, as shown in Listing 7-5.
Listing 7-5. align_faces.py

#!/usr/bin/env python
# Software License Agreement (BSD License)
#
# Copyright (c) 2012, Philipp Wagner
# All rights reserved.
#
# Redistribution and use in source and binary forms, with or without
# modification, are permitted provided that the following conditions
# are met:
#
# * Redistributions of source code must retain the above copyright
#   notice, this list of conditions and the following disclaimer.
# * Redistributions in binary form must reproduce the above
#   copyright notice, this list of conditions and the following
#   disclaimer in the documentation and/or other materials provided
#   with the distribution.
# * Neither the name of the author nor the names of its
#   contributors may be used to endorse or promote products derived
#   from this software without specific prior written permission.
#
# THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
# "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
# LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS
# FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
# COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
# INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING,
# BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
# LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER
# CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
# LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN
# ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
# POSSIBILITY OF SUCH DAMAGE.
#
# Manoel Ramon 06/11/2014- changed the code to support images used
# as example of emotion classification
#
import sys, math, Image

def Distance(p1, p2):
    dx = p2[0] - p1[0]
    return math.sqrt(dx*dx+dy*dy)
def ScaleRotateTranslate(image, angle, center = None, new_center = None, scale = None, resample=Image.BICUBIC):
    if (scale is None) and (center is None):
        return image.rotate(angle=angle, resample=resample)
    nx,ny = x,y = center
    sx,sy=1.0
    if new_center:
        (nx,ny) = new_center
    if scale:
        (sx,sy) = (scale, scale)
    cosine = math.cos(angle)
    sine = math.sin(angle)
    a = cosine/sx
    b = sine/sx
    c = x-nx*a-ny*b
    d = -sine/sy
    e = cosine/sy
    f = y-nx*d-ny*e
    return image.transform(image.size, Image.AFFINE, (a,b,c,d,e,f), resample=resample)

def CropFace(image, eye_left=(0,0), eye_right=(0,0), offset_pct=(0.2,0.2),
    dest_sz = (70,70)):
    # calculate offsets in original image
    offset_h = math.floor(float(offset_pct[0])*dest_sz[0])
    offset_v = math.floor(float(offset_pct[1])*dest_sz[1])
    # get the direction
    eye_direction = (eye_right[0] - eye_left[0], eye_right[1] - eye_left[1])
    # calculate rotation angle in radians
    rotation = -math.atan2(float(eye_direction[1]),float(eye_direction[0]))
    # distance between them
    dist = Distance(eye_left, eye_right)
    # calculate the reference eye-width
    reference = dest_sz[0] - 2.0*offset_h
    # scale factor
    scale = float(dist)/float(reference)
    # rotate original around the left eye
    image = ScaleRotateTranslate(image, center=eye_left, angle=rotation)
    # crop the rotated image
    crop_xy = (eye_left[0] - scale*offset_h, eye_left[1] - scale*offset_v)
    crop_size = (dest_sz[0]*scale, dest_sz[1]*scale)
    image = image.crop((int(crop_xy[0]), int(crop_xy[1]),
        int(crop_xy[0]+crop_size[0]), int(crop_xy[1]+crop_size[1])))
    # resize it
    image = image.resize(dest_sz, Image.ANTIALIAS)
    return image
if __name__ == '__main__':

    image = Image.open("serious_01.jpg")
    CropFace(image, eye_left=(528, 423), eye_right=(770, 431), offset_pct=(0.2,0.2)).save("serious01_20_20_70_70.jpg")

    image = Image.open("serious_02.jpg")
    CropFace(image, eye_left=(522, 412), eye_right=(758, 415), offset_pct=(0.2,0.2)).save("serious02_20_20_70_70.jpg")

    image = Image.open("serious_03.jpg")
    CropFace(image, eye_left=(518, 423), eye_right=(754, 425), offset_pct=(0.2,0.2)).save("serious03_20_20_70_70.jpg")

    image = Image.open("smile_01.jpg")
    CropFace(image, eye_left=(516, 377), eye_right=(753, 379), offset_pct=(0.2,0.2)).save("smile01_20_20_70_70.jpg")

    image = Image.open("smile_02.jpg")
    CropFace(image, eye_left=(533, 374), eye_right=(763, 380), offset_pct=(0.2,0.2)).save("smile02_20_20_70_70.jpg")

    image = Image.open("smile_03.jpg")
    CropFace(image, eye_left=(518, 379), eye_right=(749, 381), offset_pct=(0.2,0.2)).save("smile03_20_20_70_70.jpg")
image = Image.open("smile_03.jpg")
CropFace(image, eye_left=(518,379), eye_right=(749,381), offset_pct=(0.2,0.2)).save("smile03_20_20_70_70.jpg")

#surprised_01.jpg
#left -> 516,356
#right -> 754,355

image = Image.open("surprised_01.jpg")
CropFace(image, eye_left=(516,356), eye_right=(754,355), offset_pct=(0.2,0.2)).save("surprised01_20_20_70_70.jpg")

#surprised_02.jpg
#left -> 548, 364
#right -> 793, 364

image = Image.open("surprised_02.jpg")
CropFace(image, eye_left=(548,364), eye_right=(793,364), offset_pct=(0.2,0.2)).save("surprised02_20_20_70_70.jpg")

#surprised_03.jpg
#left -> 528, 377
#right -> 770, 378

image = Image.open("surprised_03.jpg")
CropFace(image, eye_left=(528,377), eye_right=(770,378), offset_pct=(0.2,0.2)).save("surprised03_20_20_70_70.jpg")

If you use the same filenames in your pictures, the only thing that you must change are the coordinates of your eyes for each picture. Then you copy the script into the same folder your pictures are in and run this in the computer shell:

mcramon@ubuntu:~/.tmp/opencv/emotion/mypics$ python align_faces.py

A series of images with the suffix _20_20_70_70 is created:

mcramon@ubuntu:~/.tmp/opencv/emotion/mypics$ ls *20*
serious01_20_20_70_70.jpg smile01_20_20_70_70.jpg surprised01_20_20_70_70.jpg
serious02_20_20_70_70.jpg smile02_20_20_70_70.jpg surprised02_20_20_70_70.jpg
serious03_20_20_70_70.jpg smile03_20_20_70_70.jpg surprised03_20_20_70_70.jpg

If you use different filenames and a different number of pictures, you need to change the script accordingly.
Do not worry about the details of this code; only keep in mind that this script uses the pillow module to create an image object that, using the CropFace() function, crops the image according to the scale reduction. For example, to crop the image file surprised_02.jpg to a scale of 20% x 20%, the following line of code is necessary:

```python
image = Image.open("surprised_02.jpg")
CropFace(image, eye_left=(548,364), eye_right=(793,364), offset_pct=(0.2,0.2)).save("surprised02_20_20_70_70.jpg")
```

As a result, all the images will contain only your face, as shown in Figure 7-12.

![Figure 7-12. Emotion pictures cropped](image)

The next step is to transfer these cropped images to Intel Galileo. A quick way to do that if you are using Linux, MacOSX, or Windows Cygwin and have Intel Galileo with a valid IP address on your network is to use scp. Run the following in the command line in the directory containing your images:

```bash
mcramon@ubuntu:~/.tmp/opencv/emotion/mypics$ for i in $(ls *20*);do scp $i root@192.254.1.1:/home/root/; done
```

All the images are transferred to the /home/root directory.

Organizing the Images in Directories

With the images transferred to Intel Galileo, organize the images by creating a directory for each type of emotion and transfer the pictures to the corresponding directory. For example, use the mkdir command to create the serious, smile, and surprised
directories. Move each picture with the `mv` command to the corresponding directory. The result is something like this:

```
├── serious
│   ├── serious01_20_20_70_70.jpg
│   ├── serious02_20_20_70_70.jpg
│   ├── serious03_20_20_70_70.jpg
│   ├── smile
│   │   ├── smile01_20_20_70_70.jpg
│   │   ├── smile02_20_20_70_70.jpg
│   │   ├── smile03_20_20_70_70.jpg
│   │   └── surprised
│   │       ├── surprised01_20_20_70_70.jpg
│   │       ├── surprised02_20_20_70_70.jpg
│   │       └── surprised03_20_20_70_70.jpg
└── surprised
```

Creating the CSV File

The last step in creating the database is to create a CSV (comma-separated values) file. This is a simple text file that describes the exact location of each image and categorizes each image by emotion based on the directory.

An example of a CSV file is shown in Listing 7-6.

**Listing 7-6. my_csv.csv**

```
/home/root/emotion/pics/smile/smile01_20_20_70_70.jpg;0
/home/root/emotion/pics/smile/smile02_20_20_70_70.jpg;0
/home/root/emotion/pics/smile/smile03_20_20_70_70.jpg;0
/home/root/emotion/pics/surprised/surprised01_20_20_70_70.jpg;1
/home/root/emotion/pics/surprised/surprised02_20_20_70_70.jpg;1
/home/root/emotion/pics/surprised/surprised03_20_20_70_70.jpg;1
/home/root/emotion/pics/serious/serious01_20_20_70_70.jpg;2
/home/root/emotion/pics/serious/serious02_20_20_70_70.jpg;2
/home/root/emotion/pics/serious/serious03_20_20_70_70.jpg;2
```

Note that each image is delimited by `;` with an index that represents the emotional state of the picture. In Listing 7-6, 0 represents smiling, 1 represents surprise, and 2 represents seriousness.

The script that helps create CSV files is shown in Listing 7-7.

**Listing 7-7. create_csv.py**

```
#!/usr/bin/env python
# Software License Agreement (BSD License)
#
# Copyright (c) 2012, Philipp Wagner
# All rights reserved.
#
```
# Redistribution and use in source and binary forms, with or without
# modification, are permitted provided that the following conditions
# are met:
#
# * Redistributions of source code must retain the above copyright
# notice, this list of conditions and the following disclaimer.
# * Redistributions in binary form must reproduce the above
# copyright notice, this list of conditions and the following
# disclaimer in the documentation and/or other materials provided
# with the distribution.
# * Neither the name of the author nor the names of its
# contributors may be used to endorse or promote products derived
# from this software without specific prior written permission.
#
# THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
# "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
# LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS
# FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
# COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
# INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING,
# BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
# LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER
# CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
# LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN
# ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
# POSSIBILITY OF SUCH DAMAGE.

import sys
import os.path

# This is a tiny script to help you creating a CSV file from a face
# database with a similar hierarchie:
#
# philipp@mango:~/.facerec/data/at$ tree
# .
# | -- README
# | -- s1
# |   | -- 1.pgm
# |   | -- ...
# |   | -- 10.pgm
# | -- s2
# |   | -- 1.pgm
# |   | -- ...
# |   | -- 10.pgm
# ...
# | -- s40
# |  | -- 1.pgm
# |  |  | -- ...
# |  |  | -- 10.pgm
#

if __name__ == "__main__":

    if len(sys.argv) != 2:
        print "usage: create_csv <base_path>"
        sys.exit(1)

    BASE_PATH=sys.argv[1]
    SEPARATOR=";"

    label = 0
    for dirname, dirnames, filenames in os.walk(BASE_PATH):
        for subdirname in dirnames:
            subject_path = os.path.join(dirname, subdirname)
            for filename in os.listdir(subject_path):
                abs_path = "%s/%s" % (subject_path, filename)
                print "%s%s%d" % (abs_path, SEPARATOR, label)
                label = label + 1

Transfer this file to Intel Galileo and run the following command line:

github create_csv.py <the ABSOLUTE directory path> > <your file name>

For example:

    root@clanton:~/emotion# python create_csv.py $(pwd)/pics/ > my_csv.csv

And check the file:

    root@clanton:~/emotion# cat my_csv.csv

    /home/root/emotion/pics/smile/smile01_20_20_70_70.jpg;0
    /home/root/emotion/pics/smile/smile02_20_20_70_70.jpg;0
    /home/root/emotion/pics/smile/smile03_20_20_70_70.jpg;0
    /home/root/emotion/pics/surprised/surprised01_20_20_70_70.jpg;1
    /home/root/emotion/pics/surprised/surprised02_20_20_70_70.jpg;1
    /home/root/emotion/pics/surprised/surprised03_20_20_70_70.jpg;1
    /home/root/emotion/pics/serious/serious01_20_20_70_70.jpg;2
    /home/root/emotion/pics/serious/serious02_20_20_70_70.jpg;2
    /home/root/emotion/pics/serious/serious03_20_20_70_70.jpg;2

382
The Code for Emotion Classification

The code for emotion classification uses a class called FaceRecognizer, which is responsible for reading your models. In other words, it reads the pictures and each state index in the database and, using a model called fisherface, feeds (or trains) the model in order to be able to predict emotions.

The code in this section is based on the code presented in Listing 7-7. Listing 7-8 shows the code with the new parts in bold.

Listing 7-8. opencv_emotion_classification.cpp

/include <opencv2/opencv.hpp>
#include <stdio.h>
#include "opencv2/imgproc/imgproc.hpp"
#include "opencv2/core/core.hpp"
#include "opencv2/contrib/contrib.hpp"
#include "opencv2/highgui/highgui.hpp"
```cpp
#include <iostream>
#include <fstream>
#include <sstream>

using namespace cv;
using namespace std;

String face_cascade_name = "haarcascade_frontalface_alt.xml";
String eye_cascade_name = "haarcascade_eye.xml";

Mat faceDetect(Mat img);
CascadeClassifier face_cascade;
CascadeClassifier eyes_cascade;

using namespace cv;
using namespace std;

enum EmotionState_t {
    SMILE = 0,  // 0
    SURPRISED,  // 1
    SERIOUS,  // 2
};

static void read_csv(const string& filename, vector<Mat>& images,
                     vector<int>& labels, char separator = ';') {
    std::ifstream file(filename.c_str(), ifstream::in);
    if (!file) {
        string error_message = "No valid input file was given, please check the given filename.";
        CV_Error(CV_StsBadArg, error_message);
    }
    string line, path, classlabel;
    while (getline(file, line)) {
        stringstream liness(line);
        getline(liness, path, separator);
        getline(liness, classlabel);
        if (!path.empty() && !classlabel.empty()) {
            images.push_back(imread(path, 0));
            labels.push_back(atoi(classlabel.c_str()));
        }
    }
}
```

384
int main(int argc, const char *argv[])
{

    EmotionState_t emotion;

    // Check for valid command line arguments, print usage
    // if no arguments were given.
    if (argc < 2) {
        cout << "usage: " << argv[0] << " <csv.ext> <output_folder> " << endl;
        exit(1);
    }

    if( !face_cascade.load( face_cascade_name ) ){ printf("--(!)Error
loading\n"); return -1; };
    if( !eyes_cascade.load( eye_cascade_name ) ){ printf("--(!)Error
loading\n"); return -1; };

    // 0 is the ID of the built-in laptop camera, change if you want to use
    other camera
    VideoCapture cap(-1);

    //check if the file was opened properly
    if(!cap.isOpened())
    {
        cout << "Capture could not be opened succesfully" << endl;
        return -1;
    }
    else
    {
        cout << "camera is ok.. Stay 2 ft away from your camera\n" << endl;
    }

    int w = 432;
    int h = 240;
    cap.set(CV_CAP_PROP_FRAME_WIDTH, w);
    cap.set(CV_CAP_PROP_FRAME_HEIGHT, h);

    Mat frame;
    cap >>frame;

    cout << "processing the image...." << endl;

    Mat testSample = faceDetect(frame);
// Get the path to your CSV.
string fn_csv = string(argv[1]);
// These vectors hold the images and corresponding labels.
vector<Mat> images;
vector<int> labels;
// Read in the data. This can fail if no valid
// input filename is given.
try
{
    read_csv(fn_csv, images, labels);
} catch (cv::Exception& e) {
    cerr << "Error opening file \"" << fn_csv << ":\". Reason: \\n    << e.msg << endl;
    // nothing more we can do
    exit(1);
}
// Quit if there are not enough images for this demo.
if(images.size() <= 1)
{
    string error_message = "This demo needs at least 2 images to
work. Please add more images to your data set!";
    CV_Error(CV_StsError, error_message);
}
// Get the height from the first image. We'll need this
// later in code to reshape the images to their original
// size:
int height = images[0].rows;

// The following lines create an Fisherfaces model for
// face recognition and train it with the images and
// labels read from the given CSV file.
// If you just want to keep 10 Fisherfaces, then call
// the factory method like this:
//    cv::createFisherFaceRecognizer(10);
// However it is not useful to discard Fisherfaces! Please
// always try to use _all_ available Fisherfaces for
// classification.
// If you want to create a FaceRecognizer with a
// confidence threshold (e.g. 123.0) and use _all_
// Fisherfaces, then call it with:
//    cv::createFisherFaceRecognizer(0, 123.0);
Ptr<FaceRecognizer> model = createFisherFaceRecognizer();
model->train(images, labels);

// The following line predicts the label of a given test image:
int predictedLabel = model->predict(testSample);

// To get the confidence of a prediction call the model with:
//
// int predictedLabel = -1;
// double confidence = 0.0;
// model->predict(testSample, predictedLabel, confidence);
//
string result_message = format("Predicted class = %d", predictedLabel);
cout << result_message << endl;

// giving the result
switch (predictedLabel)
{
    case SMILE:
        cout << "You are happy!" << endl;
        break;
    case SURPRISED:
        cout << "You are surprised!" << endl;
        break;
    case SERIOUS:
        cout << "You are serious!" << endl;
        break;
}

return 0;

cap.release();

return 0;
}

Mat faceDetect(Mat img)
{
    std::vector<Rect> faces;
    std::vector<Rect> eyes;
    bool two_eyes = false;
    bool any_eye_detected = false;
//detecting faces
face_cascade.detectMultiScale( img, faces, 1.1, 2, 0 | CV_HAAR_SCALE_IMAGE,
Size(30, 30) );

if (faces.size() == 0)
{
    cout << "Try again.. I did not detected any faces..." << endl;
    exit(-1);  // abort everything
}

Point p1 = Point(0,0);
for( size_t i = 0; i < faces.size(); i++ )
{

    // we cannot draw in the image !!! otherwise will mess with the prediction
    // rectangle( img, faces[i], Scalar( 255, 100, 0 ), 4, 8, 0 );

    Mat frame_gray;
    cvtColor( img, frame_gray, CV_BGR2GRAY );

    // croping only the face in region defined by faces[i]
    std::vector<Rect> eyes;
    Mat faceROI = frame_gray( faces[i] );

    //In each face, detect eyes
    eyes_cascade.detectMultiScale( faceROI, eyes, 1.1, 3, 0 |
    CV_HAAR_SCALE_IMAGE, Size(30, 30) );

    for( size_t j = 0; j < eyes.size(); j++ )
    {
        Point center( faces[i].x + eyes[j].x + eyes[j].width*0.5,
                        faces[i].y + eyes[j].y + eyes[j].height*0.5 );
        // we cannot draw in the image !!! otherwise will mess with the prediction
        // int radius = cvRound( (eyes[j].width + eyes[j].height)*0.25 );
        // circle( img, center, radius, Scalar( 255, 0, 0 ), 4, 8, 0 );

        if (j==0)
        {
            p1 = center;
            any_eye_detected = true;
        }
    }
else
{
    two_eyes = true;
}
}

cout << "SOME DEBUG" << endl;
cout << "-------------------------" << endl;
cout << "faces detected:" << faces.size() << endl;
cout << "x: " << faces[0].x << endl;
cout << "y: " << faces[0].y << endl;
cout << "w: " << faces[0].width << endl;
cout << "h: " << faces[0].height << endl;

Mat imageInRectangle;
imageInRectangle = img(faces[0]);
Size recFaceSize = imageInRectangle.size();

cout << recFaceSize << endl;

// for debug
imwrite("imageInRectangle.jpg", imageInRectangle);

int rec_w = 0;
int rec_h = faces[0].height * 0.64;

// checking the (x,y) for cropped rectangle
// based in human anatomy
int px = 0;
int py = 2 * 0.125 * faces[0].height;

Mat cropImage;

cout << "faces[0].x:" << faces[0].x << endl;
p1.x = p1.x - faces[0].x;
cout << "p1.x:" << p1.x << endl;
if (any_eye_detected)
{
    if (two_eyes)
    {
        cout << "two eyes detected" << endl;
        // we have detected two eyes
        // we have p1 and p2
        // left eye
        px = p1.x / 1.35;
    }
    else
    {
        // only one eye was found.. need to check if the
        // left or right eye
        // we have only p1
        if (p1.x > recFaceSize.width/2)
        {
            // right eye
            cout << "only right eye detected" << endl;
            px = p1.x / 1.75;
        }
        else
        {
            // left eye
            cout << "only left eye detected" << endl;
            px = p1.x / 1.35;
        }
    }
}
else
{
    // no eyes detected but we have a face
    px = 25;
    py = 25;
    rec_w = recFaceSize.width-50;
    rec_h = recFaceSize.height-30;
}

rec_w = (faces[0].width - px) * 0.75;
cout << "px :": " << px << endl;
cout << "py :": " << py << endl;
cout << "rec_w:": " << rec_w << endl;
cout << "rec_h:": " << rec_h << endl;
cropImage = imageInRectangle(Rect(px, py, rec_w, rec_h));

Size dstImgSize(70,70); // same image size of db
Mat finalSizeImg;
resize(cropImage, finalSizeImg, dstImgSize);

// for debug
imwrite("onlyface.jpg", finalSizeImg);

cvtColor( finalSizeImg, finalSizeImg, CV_BGR2GRAY );

return finalSizeImg;

Reviewing opencv_emotion_classification.cpp

In the beginning of the code, there is an enumerator created to define the emotional state. Note the value of each element on this enum matches the emotion index in the CSV file.

enum EmotionState_t {
    SMILE     =0,   // 0
    SURPRISED,      // 1
    SERIOUS,        // 2
};

In the main() function, a variable of type EmotionState_t is created and it is expected to receive the name of the CSV file as an argument.

int main(int argc, const char *argv[]) {

    EmotionState_t emotion;

    // Check for valid command line arguments, print usage
    // if no arguments were given.
    if (argc < 2) {
        cout << "usage: " << argv[0] << " <csv.ext> <output_folder> " << endl;
        exit(1);
    }

    When the webcam is opened, the picture is collected as before. The faceDetect() method changes, compared to the faceDetect() method shown earlier:

    Mat testSample = faceDetect(frame);
This new object stored in testSample contains the cropped face. This cropped image is the same size as the images in the database. This image returned is in grayscale and is cropped like the images shown in Figure 7-12.

The frame contains an 432x240 image and the testSample image is 70x70. For now, let's continue with the main() function. faceDetect() will be discussed in more detail later.

With the image prepared to be analyzed, new components are used to predict the emotional state:

```cpp
Ptr<FaceRecognizer> model = createFisherFaceRecognizer();
model->train(images, labels);

// The following line predicts the label of a given test image:
int predictedLabel = model->predict(testSample);
```

class FaceRecognizer : public Algorithm

At a glance, FaceRecognizer looks very simple, but in fact it's very powerful and complex. This class allows you to set different algorithms, including your own, to perform different kinds of image recognitions.

The model used in the code is fisherface and it's created by the line:

```cpp
Ptr<FaceRecognizer> model = createFisherFaceRecognizer();
```

void FaceRecognizer::train(InputArrayOfArrays src, InputArray labels)

This method trains the model based on your database. The code passes the images and index (or labels):

```cpp
model->train(images, labels);
```

int FaceRecognizer::predict(InputArray src) const = 0

This method predicts the classification index (label) based on the image casted as the input array src.

For example, if the emotion "happy" is labeled as 0 in the CSV file and the FaceRecognizer was trained, the prediction will return 0 if the image src is a picture of you smiling.

This is represented by the following snippet:

```cpp
int predictedLabel = model->predict(testSample);
```
// giving the result
switch (predictedLabel) {
    case SMILE:
        cout << "You are happy!" << endl; break;
    case SURPRISED:
        cout << "You are surprised!" << endl; break;
    case SERIOUS:
        cout << "You are serious!" << endl; break;
}

If the image returned by faceDetect() is cropped properly and your expression is similar to the expression in the database, the algorithm will predict accurately.

The faceDetect() method basically does what was done before, as explained in the flowchart in Figure 7-10. In other words, it detects the face and eyes.

After this example detects the face and eyes, an algorithm containing a few simple concepts about human anatomy was introduced that crops the image to the face area.

It tries to crop the image captured by the webcam dynamically and do the same thing that was done manually by the script in Listing 7-7, but instead crop the image exclusively in the area the face is detected.

To understand how the logic works, see Figure 7-13.
Take a look at Figure 7-13 and follow this logic. When a face is recognized the whole face is typically captured, including the ears, head or hat, part of the neck, and part of the background image (imageRectangle). However, these elements are not interesting to the emotion classifier and must be removed (the red arrows area) and only the portion containing eyes, nose, and mouth are cropped (cropImage).

The cropped image has the initial px and py coordinates with the extensions rec_w and rec_h, which forms a triangle with perfect dimensions for cropping the area. Such a rectangle corresponds to the ROI (Region of Interest) area.

To reach the ROI area, the eyes are detected and then the human proportions are found using the px, py, rec_w, and rec_h values in the image and crop the image.

When the eyes are detected, it is possible to define a point object p1 that corresponds to the center of the eye. The point object p1 has two members x and y that represent the distance in pixels to the original image. There are a couple of problems, however. Sometimes only one eye is detected and the algorithm must determine if it’s the right or the left. Other times, no eye is detected.
//detecting faces
face_cascade.detectMultiScale( img, faces, 1.1, 2, 0|CV_HAAR_SCALE_IMAGE, Size(30, 30) );
Point p1 = Point(0,0);
for( size_t i = 0; i < faces.size(); i++ )
{
  ...
  ...
  ...

  //In each face, detect eyes
  eyes_cascade.detectMultiScale( faceROI, eyes, 1.1, 3, 0 |CV_HAAR_SCALE_IMAGE, Size(30, 30) );

  for( size_t j = 0; j < eyes.size(); j++ )
  {
    Point center( faces[i].x + eyes[j].x + eyes[j].width*0.5,
                  faces[i].y + eyes[j].y + eyes[j].height*0.5 );
    ...
    ...
    ...

    if (j==0)
    {
      p1 = center;
      any_eye_detected = true;
    }
    else
    {
      two_eyes = true;
    }
  }
}

At the moment, you might have the center of one of the eyes and it is known if one, two, or none eyes were detected. Now it is necessary to find the px and py coordinates, as well as the ROI dimensions, rec_w and rec_h.

In human anatomy, the eyes are located at the top of the horizontal red line that splits the human face in half. If you split the middle horizontal line equally in four parts, the eyes are separated by each other by half of the largest horizontal proportion and one-fourth from the laterals.

The nose and mouth are centralized in the middle of the face, with the inferior proportional divided in five equal parts. The eyebrows are 12.5% above the lines of the eyes because they are 50%/4 of the superior part of the face.
If no eye is detected then it is not possible to crop using a simple algorithm. With these proportions in mind, the following lines were created:

```cpp
int rec_w = 0;
int rec_h = faces[0].height * 0.64;

// checking the (x,y) for cropped rectangle
// based in human anatomy
int px = 0;
int py = 2 * 0.125 * faces[0].height;

Mat cropImage;

cout << "faces[0].x:" << faces[0].x << endl;
p1.x = p1.x - faces[0].x;
cout << "p1.x:" << p1.x << endl;
if (any_eye_detected)
{
    if (two_eyes)
    {
        cout << "two eyes detected" << endl;
        // we have detected two eyes
        // we have p1 and p2
        // left eye
        px = p1.x / 1.35;
    }
    else
    {
        // only one eye was found.. need to check if the
        // left or right eye
        // we have only p1
        if (p1.x > recFaceSize.width/2)
        {
            // right eye
            cout << "only right eye detected" << endl;
            px = p1.x / 1.75;
        }
        else
        {
            // left eye
            cout << "only left eye detected" << endl;
            px = p1.x / 1.35;
        }
    }
}
```
else
{
    // no eyes detected but we have a face
    px = 25;
    py = 25;
    rec_w = recFaceSize.width-50;
    rec_h = recFaceSize.height-30;
}

rec_w = (faces[0].width - px) * 0.75;
cout << "px   :" << px << endl;
cout << "py   :" << py << endl;
cout << "rec_w:" << rec_w << endl;
cout << "rec_h:" << rec_h << endl;

cropImage = imageInRectangle(Rect(px, py, rec_w, rec_h));

For debugging purposes, the faceDetect() method saves two images in the file system every time the software runs. One is called onlyface.jpg and it contains the cropped image. The other is called imageInRectangle.jpg and it contains the detected image.

Mat imageInRectangle;
imageInRectangle = img(faces[0]);
...
...
...
// for debug
imwrite("imageInRectangle.jpg", imageInRectangle);
cropImage = imageInRectangle(Rect(px, py, rec_w, rec_h));
...
...
Size dstImgSize(70,70); // same image size of db
Mat finalSizeImg;
resize(cropImage, finalSizeImg, dstImgSize);

Running opencv_emotion_classification.cpp

Compile the code and transfer the file to Intel Galileo. Make sure the uvcvideo driver is loaded and the webcam is connected to the USB port (read the "Connecting the Webcam" section in this chapter), and transfer the program to the same location of your CSV file. Stay in front your camera, preferably two feet away, make some emotional expressions, and then run the following command:

root@clanton:~/emotion# ./opencv_emotion_classification my_csv.csv 2> /dev/null

camera is ok.. Stay 2 ft away from your camera
processing the image....
SOME DEBUG
-------------------------
faces detected:1
x: 172
y: 25
w: 132
h: 132

[132 x 132]
faces[0].x:172
p1.x:-172
px: 25
py: 25
rec_w:80
rec_h:102
Predicted class = 0
You are happy!

The software classifies the image as happy. Extracting the debug images onlyface.jpg and imageInRectangle.jpg from the file system, it is possible to observe my expression in the cropped image, shown in Figure 7-14.

Figure 7-14. The happy face is detected and the image is cropped automatically

Note in Figure 7-14 the areas that are cropped out, including the background, the hair, and the ears.

root@clanton:~/emotion# ./opencv_emotion_classification my_csv.csv 2> /dev/null
camera is ok.. Stay 2 ft away from your camera

processing the image....
SOME DEBUG
-------------------------
faces detected:1
The software classifies this image as surprised. Extracting the debug images onlyface.jpg and imageInRectangle.jpg from the file system, it observes my expression and crops the image, as shown in Figure 7-15.

Figure 7-15. The surprised face is detected and the image is cropped automatically

Keep varying the expression and checking the efficiency of the captured images.

Ideas for Improving the Project

At this point you should see the potential that OpenCV and Intel Galileo have. However, all projects have room for improvement. Let’s discuss some of the ways you can improve this project.
Integrating Your Emotions with a Robotic Head

Chapter 13 demonstrated a robotic head that expressed emotion. The idea here is to integrate the emotion classification in this chapter and make the robot to imitate emotions. Thus, if you smile the robot will smile; if you are sad, the robot will be sad, and so on.

Expanding the Classifications

The fisherface model was used to classify emotions but the same technique can be used to classify gender or recognize your family and friends.

You need to create databases for gender and for the faces of people you know.

An example of gender classification can be found at this link http://docs.opencv.org/trunk/modules/contrib/doc/facerec/tutorial/facerec_gender_classification.html.

Improving the Emotion Classification Using Large Databases

There are several databases on the Internet containing thousands of images showing emotions (some with more than 4,000 faces). To learn about these different database and download the images, visit http://face-rec.org/databases/.

Improving the Emotion Classification for Several Faces

On the example of Listing 7-8, I created a database using my own pictures and the software is prepared to classify only me using the first face detected in the code, in other words, the object faces[0]. Add more people in the database and improve the code to classify all faces detected instead only one.

Summary

This chapter explained many principles needed to explore OpenCV: The creation of SD releases based on eGlibc, the suppression of GPU support in OpenCV development packages, the generation of the right toolchain to support V4L and OpenCV, the study of UVC devices, and the exploration of the webcam capabilities using Video4Linux. These principles serve as a study of OpenCV.

You learned how to capture and process images in OpenCV, including complex tasks like edge detection with the Canny algorithm, face and eyes detection based on Haar techniques, and the emotion classification based on the fisherface model.

This is only the beginning in terms of OpenCV and its possibilities. There are several articles on the Internet and specialized books if you want to explore more features.