Android Application Programming with OpenCV 3

Android Application Programming with OpenCV 3 is a practical, hands-on guide to computer vision and mobile app development. It shows how to capture, manipulate, and analyze images while building an application that combines photography and augmented reality. To help the reader become a well-rounded developer, the book covers OpenCV (a computer vision library), Android SDK (a mobile app framework), OpenGL ES (a 3D graphics framework), and even JNI (a Java/C++ interoperability layer).

Now in its second edition, the book offers thoroughly reviewed code, instructions, and explanations. It is fully updated to support OpenCV 3 and Android 5, as well as earlier versions. Although it focuses on OpenCV’s Java bindings, this edition adds an extensive chapter on JNI and C++, so that the reader is well primed to use OpenCV in other environments.

Who this book is written for
If you are a Java developer who is new to computer vision and would like to learn through application development, then this book is for you. You are expected to have a mobile device running Android 2.3 (Froyo) or greater, including a camera. Experience in Java is a must.

What you will learn from this book
- Install OpenCV and an Android development environment on Windows, Mac, or Linux
- Control a camera and use its perspective in augmented reality
- Share photos with other apps via Android’s MediaStore and Intent classes
- Create GUIs and handle events using Android activities and OpenCV
- Train an image recognizer that can locate famous paintings in a scene
- Apply “curves” and other color transformations to simulate the look of old photos
- Apply convolution filters that sharpen, blur, emboss, or darken the details of an image

Build Android apps to capture, manipulate, and track objects in 2D and 3D
In this package, you will find:

- The author biography
- A preview chapter from the book, Chapter 1 'Setting Up OpenCV'
- A synopsis of the book’s content
- More information on *Android Application Programming with OpenCV 3*
Joseph Howse lives in Canada. During the cold winters, he grows a beard and his four cats grow thick coats of fur. He combs the cats every day. Sometimes, the cats pull his beard.

Joseph has authored *OpenCV for Secret Agents*, *OpenCV Android Application Programming*, and *OpenCV Computer Vision with Python*. When he is not writing books or grooming cats, Joseph provides consulting, training, and software development services. His company is Nummist Media (http://nummist.com).
Preface

This book will show you how to use OpenCV in an Android app that displays a camera feed, saves and shares photos, manipulates colors and edges, and tracks real-world objects in 2D or 3D. Integration with OpenGL is also introduced so that you can start building augmented reality (AR) apps that superimpose virtual 3D scenes onto tracked objects in the camera feed.

OpenCV is an open-source, cross-platform library that provides building blocks for computer vision experiments and applications. It offers high-level interfaces to capture, process, and present image data. For example, it abstracts away details about camera hardware and array allocation. OpenCV is widely used in both academia and industry.

Android is a mobile operating system that is mostly open source. For Java developers, it offers a high-level application framework called Android SDK. Android apps are modular insofar as they have standard, high-level interfaces to launch each other and share data. Mobility, a high level of abstraction, and data sharing are great starting points for a photo sharing app, similar to the one we will build.

Although OpenCV and Android provide a lot of high-level abstractions (and a lot of open source code for curious users to browse), they are not necessarily easy to use for newcomers. Setting up an appropriate development environment and translating the libraries' broad functionality into application features are both daunting tasks. This concise book helps us by placing an emphasis on a clean setup, clean application design, and a simple understanding of each function's purpose.

The need for a book on this subject is particularly great because OpenCV's Java and Android bindings are quite new and their documentation is not yet mature. Little has been written about the steps for integrating OpenCV with Android's standard camera, media, and graphics APIs. Surely, integration is a major part of an app developer's work, so it is a major focus of this book.
Preface

By the end of our journey together, you will have a taste of the breadth of application features that are made possible by integrating OpenCV with other Android libraries. You will have your own small library of reusable classes that you can extend or modify for your future computer vision projects. You will have a development environment and the knowledge to use it, and you will be able to make more apps!

What this book covers

Chapter 1, Setting Up OpenCV, covers the steps to set up OpenCV and an Android development environment, including Eclipse and Android SDK.

Chapter 2, Working with Camera Frames, shows how to integrate OpenCV into an Android Java app that can preview, capture, save, and share photos.

Chapter 3, Applying Image Effects, explores OpenCV’s functionality to manipulate color channels and neighborhoods of pixels. The Apache Commons Math library is also introduced. We expand our app to include channel-mixing filters, "curve" filters, and a filter that darkens edges.

Chapter 4, Recognizing and Tracking Images, demonstrates the steps to recognize and track a known target (such as a painting) when it appears in a video feed. We expand our app so that it draws an outline around any tracked target.

Chapter 5, Combining Image Tracking with 3D Rendering, improves upon our previous tracking technique by determining the target’s position and rotation in real 3D space. We expand our app so that it sets up an OpenGL 3D scene with the same perspective as the Android device’s real camera. Then, we draw a 3D cube atop any tracked target.

Chapter 6, Mixing Java and C++ via JNI, demonstrates the use of Java Native Interface (JNI) to call C++ functions from Java. We convert some of our application’s filters to C++ in order to learn about writing efficient, cross-platform code with OpenCV’s C++ interface.
Setting Up OpenCV

This chapter is a quick guide to setting up a development environment for Android and OpenCV. We will also look at the OpenCV sample applications, documentation, and community.

By the end of this chapter, our development environment will include the following components:

- **Java Development Kit (JDK) 7**: This includes tools for Java programming. JDK 7 is the exact version that we require. The more recent version, JDK 8, is not yet supported for Android development.
- **Cygwin 1.7 or greater (Windows only)**: This is a compatibility layer that provides Unix-like programming tools on Windows. We need it in order to develop in C++ on Android.
- **Android Software Development Kit (Android SDK) r24.0.2 or greater**: This includes tools for programming Android apps in Java.
- **Android Native Development Kit (Android NDK) r10d or greater**: This includes tools for programming Android apps in C++.
- **Eclipse 4.4.2 (Luna) or greater**: This is an integrated development environment (IDE). Although Google has started to recommend Android Studio as an IDE for Android development, Eclipse is still supported too. The OpenCV library and official samples are preconfigured as Eclipse projects, so for our purposes, Eclipse is a bit more convenient than Android Studio.
- **Java Development Tools (JDT)**: This is an Eclipse plugin for Java programming (already included in most Eclipse distributions).
- **C/C++ Development Tooling (CDT) 8.2.0 or greater**: This is an Eclipse plugin for C/C++ programming.
Setting Up OpenCV

- **Android Development Tools (ADT) 24.0.2 or greater**: This is an Eclipse plugin for Android programming.
- **OpenCV4Android 3.0 or greater**: This is OpenCV's Android version, including Java and C++ libraries.

At the time of writing, OpenCV4Android's latest version is 3.0. This book targets version 3.0, but it also includes comprehensive notes on the differences between OpenCV 3.x and OpenCV 2.x. The author's website, [http://nummist.com/opencv](http://nummist.com/opencv), offers two sets of code bundles: one for OpenCV 3.x (tested with 3.0) and another for OpenCV 2.x (tested with 2.4.9).

There are many possible ways to install and configure these components. We will cover several common setup scenarios, but if you are interested in further options, see OpenCV's official documentation at [http://docs.opencv.org/doc/tutorials/introduction/android_binary_package/O4A_SDK.html](http://docs.opencv.org/doc/tutorials/introduction/android_binary_package/O4A_SDK.html).

**System requirements**

All the development tools for Android and OpenCV are cross-platform. The following operating systems are supported with almost identical setup procedures:

- Windows XP or a later version
- Mac OS 10.6 (Snow Leopard) or a later version
- Debian Wheezy or a later version, including derivatives such as Ubuntu 12.04 (Pangolin) or a later version
- Many other Unix-like systems (though not specifically covered in this book)

To run the OpenCV samples and, later on, our own application, we should have an Android device with the following specifications:

- Android 2.2 (Froyo) or greater (required)
- Camera (required): front and rear cameras (recommended)
- Autofocus (recommended)

**Android Virtual Devices (AVDs)** are not recommended. Some parts of OpenCV rely on low-level camera access and might fail with virtualized cameras.
Chapter 1

Setting up a development environment

We are going to install various components of a development environment separately and configure them to work together. Broadly, this task has the following two stages:

1. Set up a general-purpose Android development environment.
2. Set up OpenCV for use in this environment. We may use a prepackaged, preconfigured version of OpenCV, or alternatively, we may configure and build OpenCV from source.

Let's start by looking at the setup steps for a general-purpose Android development environment. We will not delve into very much detail here because good instructions are available at the given links and, as an Android or Java developer, you have probably been through similar steps before.

If you already have an Android development environment or another Java development environment and you just want to add components to it, some of the following steps might not apply to you.

Here are the steps:

1. Download and install Oracle JDK 7 from http://www.oracle.com/technetwork/java/javase/downloads/jdk7-downloads-1880260.html. Alternatively, on Debian or Ubuntu, install Oracle JDK 7 from the WebUpd8 PPA, as described at https://launchpad.net/~webupd8team/+archive/ubuntu/java. Although most Linux distributions have OpenJDK in their standard repositories, Oracle JDK is recommended instead for Android development.

2. Download Eclipse and unzip it to any destination, which we will refer to as <eclipse>. Many up-to-date Eclipse distributions are available at http://www.eclipse.org/downloads/ Of these, Eclipse IDE for Java Developers is a good choice as a foundation for an Android development environment.

3. We now need to set up Android SDK and the ADT plugin for Eclipse. Go to http://developer.android.com/sdk/index.html#Other and download SDK Tools Only. Install or unzip it to any destination, which we will refer to as <android_sdk>. Open Eclipse and install the ADT plugin according to the official instructions at http://developer.android.com/sdk/installing/installing-adt.html. Restart Eclipse. A window, Welcome to Android Development, should appear. Click on Use Existing SDKs, browse to <android_sdk>, and click on Next. Close Eclipse.
4. From the Eclipse menu system, navigate to Windows | Android SDK Manager. Select and install additional SDK packages according to the official instructions at https://developer.android.com/sdk/installing/adding-packages.html. In particular, we will need the most recent versions of the following packages: the latest Android API, such as Android 5.1.1 (API 22), Android SDK Tools, Android SDK Platform-tools, Android SDK Build Tools, and Android Support Library. After the packages are installed, close Eclipse.

5. If we are using Windows, download and install Cygwin from http://cygwin.com/install.html.


7. Edit your system's Path (in Windows) or PATH (in Mac, Linux, or other Unix-like systems) to include <android_sdk>/platform-tools, <android_sdk>/tools, and <android_ndk>. Also, create an environment variable named NDKROOT and set its value to <android_ndk>. (If you are unsure how to edit Path, PATH or other environment variables, see the tips in the boxes on this page and next page.)

Editing environment variables on Windows

The system's Path variable and other environment variables can be edited in the Environment Variables window of the Control Panel.

On Windows Vista/7/8, open the Start menu and launch the Control Panel. Now, go to System and Security | System | Advanced system settings. Click on the Environment Variables button.

On Windows XP, open the Start menu and go to Control Panel | System. Click on the Advanced tab. Click on the Environment Variables button.

Now, under System variables, select an existing environment variable, such as Path, and click on the Edit button. Alternatively, make a new environment variable by clicking on the New button. Edit the variable's name and value as needed. For example, if we want to add C:\android-sdk\platform-tools and C:\android-sdk\tools to Path, we should append ;C:\android-sdk\platform-tools;C:\android-sdk\tools to the existing value of Path. Note the use of semicolons as separators.

To apply the changes, click on all the OK buttons until we are back in the main window of the Control Panel. Now, log out and log in again.
Editing environment variables on Mac

Edit ~/.profile.

To append to an existing environment variable in ~/.profile, add a line such as export PATH=$PATH:-/android-sdk/platform-tools:-/android-sdk/tools. This example appends -/android-sdk/platform-tools and -/android-sdk/tools to PATH. Note the use of colons as separators.

To create a new environment variable in ~/.profile, add a line such as export NDKROOT=-/android-ndk.

Save your changes, log out, and log in again.

Editing environment variables on Linux

Edit either ~/.profile (as described previously for Mac) or ~/.pam_environment (as described next). Note that ~/.profile and ~/.pam_environment use slightly different formats for variables.

To append to an existing environment variable in ~/.pam_environment, add a line such as PATH DEFAULT=${PATH}:-/android-sdk/platform-tools:-/android-sdk/tools. This example appends -/android-sdk/platform-tools and -/android-sdk/tools to PATH. Note the use of colons as separators.

To create a new environment variable in ~/.pam_environment, add a line such as NDKROOT DEFAULT=-/android-ndk.

Save your changes, log out, and log in again.

Now we have an Android development environment, but we still need OpenCV. We may choose to download a prebuilt version of OpenCV, or we may build it from source. These options are discussed in the following two subsections.

Getting prebuilt OpenCV4Android

The prebuilt versions of OpenCV4Android can be downloaded from http://sourceforge.net/projects/opencvlibrary/files/opencv-android/. Look for files that have opencv-android in the name, such as OpenCV-3.0.0-android-sdk.zip (the latest version at the time of writing). Download the latest version and unzip it to any destination, which we will refer to as <opencv>. 
Building OpenCV4Android from source

Alternatively, the process for building OpenCV4Android from trunk (the latest, unstable source code) is documented at http://code.opencv.org/projects/opencv/wiki/Building_OpenCV4Android_from_trunk. For a summary of the process, continue reading this section. Otherwise, skip ahead to Building the OpenCV samples with Eclipse, later in this chapter.

Since trunk contains the latest, unstable source code, there is no guarantee that the build process will succeed. You may need to do your own troubleshooting if you want to build from trunk.

To build OpenCV from source, we need the following additional software:

- **Git**: This is a Source Control Management (SCM) tool, which we will use to obtain OpenCV's source code. On Windows or Mac, download and install Git from http://git-scm.com/. On Linux, install it using your package manager. For example, on Debian or Ubuntu, open Terminal and run `$ sudo apt-get install git-core`.

- **CMake**: This is a set of build tools. On Windows or Mac, download and install CMake from http://www.cmake.org/cmake/resources/software.html. On Linux, install it using your package manager. For example, on Debian or Ubuntu, open Terminal and run `$ sudo apt-get install cmake`.

- **Apache Ant 1.8.0 or greater**: This is a set of build tools for Java. On Linux, just install Ant using your package manager. For example, on Debian or Ubuntu, open Terminal and run `$ sudo apt-get install ant`. On Windows or Mac, download Ant from http://ant.apache.org/bindownload.cgi and unzip it to any destination, which we will refer to as `<ant>`. Make the following changes to your environment variables:
  - Add `<ant>/bin` to Path (Windows) or PATH (Unix).
  - Create a variable ANT_HOME with the value `<ant>`.

- **Python 2.6 or greater (but not 3.0 or greater)**: This is a scripting language that is used by some of the OpenCV build scripts. An appropriate version of Python comes preinstalled on Mac and most Linux systems, including Debian and Ubuntu. On Windows, download and install Python from http://www.python.org/getit/. If you have installed multiple versions of Python on your system, ensure that an installation of Python 2.6 or greater (but not 3.0 or greater) is the only one in Path (Windows) or PATH (Unix). The OpenCV build scripts do not run properly with Python 3.0 or greater.
Once we have these prerequisites, we can download the OpenCV source code to any location, which we will refer to as `<opencv_source>`. Then, we can build it using an included script. Specifically, we should take the following steps:

On Windows, open Git Bash (Git's command prompt). On Mac, Debian, Ubuntu, or other Unix-like systems, open Terminal (or another command-line shell).

Run these commands:

```bash
$ git clone git://code.opencv.org/opencv.git <opencv_source>
$ cd <opencv_source>/platforms
$ sh ./scripts/cmake_android_arm.sh
$ cd build_android_arm
$ make -j8
```

The `-j8` flag specifies that the `make` command will use 8 threads, which is typically a good number for a quad-core processor. For a dual-core processor, a better choice might be the `-j4` flag (4 threads).

If all goes well, we should get a build of OpenCV4Android in `<opencv_source>/platforms/build_android_arm`. We can move it elsewhere if we wish. We will refer to its final location as `<opencv>`.

You might wonder what the `cmake_android_arm.sh` build script is doing. Actually, it just creates a build directory and runs a `CMake` command to populate the directory with a particular configuration of OpenCV. Here are the entire contents of the script file:

```bash
#!/bin/sh

cd `dirname $0`/..

mkdir -p build_android_arm

cd build_android_arm


cmake -DCMAKE_BUILD_WITH_INSTALL_RPATH=ON -

DCMAKE_TOOLCHAIN_FILE=../android/android.toolchain.cmake $@ ../../
```

Advanced users, who are familiar with CMake, might want to copy and modify this script to create a custom configuration of OpenCV. Refer to the code in `<opencv_source>/CMakeLists.txt` for definitions of OpenCV's CMake options.
The preceding steps use the cmake_android_arm.sh script to produce an OpenCV4Android build for ARM, which is the architecture of most Android phones and tablets. Alternatively, you can use the cmake_android_x86.sh script for x86 or the cmake_android_mips.sh script for MIPS. Note that the name of the build directory also changes according to the architecture.

Building the OpenCV samples with Eclipse

Building and running a few sample applications is a good way to test that OpenCV is correctly set up. At the same time, we can practice using Eclipse.

Let's start by launching Eclipse. The Eclipse launcher should be located at <eclipse>/eclipse.exe (Windows), <eclipse>/Eclipse.app (Mac), or <eclipse>/eclipse (Linux). Run it.

We should see a window called Workspace Launcher, which asks us to select a workspace. A workspace is the root directory for a set of related Eclipse projects. Enter any location you choose.

We can return to Workspace Launcher anytime via the menu: File | Switch Workspace | Other....
If the **Welcome to Eclipse** screen appears, click on the **Workbench** button:

![Welcome to Eclipse](image)

Now, we should see a window with several panels, including **Package Explorer**. If we are not using TAPD, we need to import the OpenCV sample projects into our new workspace. Right-click on **Package Explorer** and select **Import...** from the context menu:

![Package Explorer](image)
Setting Up OpenCV

The **Import** window should appear. Navigate to **General | Existing Projects into Workspace**, and then click on **Next >**:

On the second page of the **Import** window enter `<opencv>` in the **Select root directory:** field. Under the **Projects:** label, a list of detected projects should appear. (If not, click on **Refresh**.) The list should include the OpenCV library, samples, and tutorials. They should all be selected by default.

**Downloading the example code**

You can download the example code files from your account at `http://www.packtpub.com` for all the Packt Publishing books you purchased. If you purchased this book elsewhere, you can visit `http://www.packtpub.com/support` and register to have the files e-mailed directly to you.
This means that Eclipse has found the OpenCV library, samples, and tutorials and has recognized them as Eclipse projects. Do not select **Copy projects into workspace** because the OpenCV sample and tutorial projects rely on a relative path to the library project, and this relative path will not be preserved if the projects are copied into the workspace. Click on **Finish** to import the projects:
Once the projects are imported, we might need to fix some configuration issues. Our development environment might have different paths and different versions of Android SDK than the ones in the samples' default configuration:

Any resulting errors will be reported in the Problems tab. For likely solutions, see the section Troubleshooting Eclipse projects, later in this chapter.

We should first resolve any errors in the OpenCV Library project as the samples and tutorials depend on the library.

Once the OpenCV projects no longer show any errors, we can prepare to test them on an Android device. Recall that the device must have Android 2.2 (Froyo) or greater and a camera. To let Eclipse communicate with the device, we must enable the device's USB debugging option. On the Android device, perform the following steps:
1. Open the **Settings** app.
2. On Android 4.2 or greater, go to the **About phone** or **About tablet** section and tap **Build number** seven times. This step enables the **Developer options** section.
3. Go to the **Developer options** section (on Android 4.0 or greater) or the **Applications | Development** section (on Android 3.2 or less). Enable the **USB debugging** option.

Now, we need to install an Android app called OpenCV Manager 3, which takes care of checking for OpenCV library updates when we run any OpenCV application. At the time of writing, OpenCV Manager 3 is not yet available from the Play Store. However, in the `<opencv>/apk` folder of our development environment, we can find prebuilt application bundles (.apk files) for various architectures. Choose an .apk file whose name matches your Android device's architecture. At the time of writing, ARMv7-A is a popular architecture for Android devices. For this architecture, OpenCV 3.0 offers the `OpenCV_3.0.0_Manager_3.00_armeabi-v7a.apk` file. Open a command prompt and enter a command, such as the following, to install the appropriate .apk to your Android device via USB:

```
$ adb install <opencv>/apk/OpenCV_3.0.0_Manager_3.00_armeabi-v7a.apk
```

If the installation succeeds, the terminal should print `Success`.

---

**Supporting OpenCV 2.x applications**

At the time of writing, the Play Store contains an older version of OpenCV Manager that supports OpenCV 2.x only. If you want to run both OpenCV 2.x and OpenCV 3.x applications, you can install this older version from the Play Store alongside OpenCV Manager 3. They do not conflict.
Plug the Android device into your computer’s USB port. In Eclipse, select one of the OpenCV sample projects in Package Explorer. Then, from the menu system, navigate to Run | Run as… | Android Application:

An Android Device Chooser window should appear. Your Android device should be listed under Choose a running Android device. If the device is not listed, refer to the section Troubleshooting the USB connection, later in this chapter.
Select the device and click on OK:

If the Auto Monitor Logcat window appears, select the Yes radio button and the verbose drop-down option, and click on OK. This option ensures that all the log output from the application will be visible in Eclipse:
On the Android device, you might get a message: **OpenCV library package was not found! Try to install it?** Make sure that the device is connected to the Internet and then touch the Yes button on your device. The Play Store will open to show an OpenCV package. Install the package and then press the hardware back button to return to the sample application, which should be ready for use.

For OpenCV 3.0, the samples and tutorials have the following functionality:

- **Sample–15 puzzle**: This splits up a camera feed to make a sliding-block puzzle. The user can swipe blocks to move them.
- **Sample–color-blob-detection**: This detects color regions in a camera feed. The user can touch anywhere to see the outline of a color region.
- **Sample–face-detection**: This draws green rectangles around faces in a camera feed.
- **Sample–image-manipulations**: This applies filters to a camera feed. The user can press the Android menu button to select from a list of filters. For example, one filter draws a color histogram (a bar chart of colors that are present in the image), as seen at the bottom of the following screenshot:

  ![Color Histogram](image)

- **Sample – native-activity**: This displays a camera feed using native (C++) code.
• **Tutorial 1 – Camera Preview**: This displays a camera feed. The user can press the ... menu to select a different camera feed implementation (Java or native C++).

• **Tutorial 2 – Mixed Processing**: This applies filters to a camera feed using native (C++) code. The user can press the ... menu to select from a list of filters. One of the filters draws red circles around interest points or features in a camera feed. Generally speaking, interest points or features lie along the high-contrast edges in an image. They are potentially useful in image recognition and tracking applications, as we will see later in this book.

• **Tutorial 3 – Camera Control**: This applies filters to a camera feed, which has a customizable resolution. The user can press the ... menu to select from a list of filters and a list of resolutions.

Try these applications on your Android device! While an application is running, its log output should appear in the LogCat tab in Eclipse:
Feel free to browse the projects' source code via Package Explorer to see how they were made. Alternatively, you might want to return to the official samples and tutorials later, once we have built our own application over the course of this book.

**Troubleshooting Eclipse projects**

This section is not about troubleshooting Java code. Rather, it addresses a few common problems with the configuration and build process of Eclipse projects. You might encounter these problems when working with the OpenCV library, OpenCV sample projects, other imported projects, or even your own new projects.

Sometimes, Eclipse fails to recognize that a project needs to be rebuilt after the project or one of its dependencies has changed (or after a dependency has been imported). When in doubt, try cleaning all the projects by navigating to Project | Clean... | Clean all projects | OK in the menu system. This will force Eclipse to rebuild everything, thus ensuring that all errors, warnings, and successes are up-to-date.

If a set of cleaned projects still has mysterious errors, then a configuration problem might be the cause.

The target Android version might not be properly specified. The symptoms are that imports from the java and android packages fail, and there are error messages such as The project was not built since its build path is incomplete. The solution is to right-click on the project in Package Explorer, select Properties from the context menu, select the Android section, and checkmark one of the available Android versions. These steps should be repeated for all the projects.
At compile time, OpenCV and its samples must target Android 3.0 (API level 11) or greater, though at runtime they also support Android 2.2 (API level 8) or greater:
Setting Up OpenCV

If imported on Mac or Linux, the OpenCV C++ samples might be misconfigured to use the Windows build executable. The symptom is an error message such as

**Program "/ndk-build.cmd" not found in PATH.**

The solution is to right-click on the project in Package Explorer, select Properties from the context menu, select the C/C++ Build section, and edit the Build command: field to remove the .cmd extension. These steps should be repeated for all the native (C++) projects, which include OpenCV Sample – face-detection and OpenCV Tutorial 2 - Mixed Processing:

If we are still getting an error message such as **Program "/ndk-build.cmd" not found in PATH,** we can conclude that Eclipse is not recognizing the NDKROOT environment variable. As an alternative to relying on an environment variable, we can add NDKROOT as an Eclipse build variable to Eclipse | Preferences | C/C++ | Build | Build Variables. (These preferences are shared across Eclipse projects.)

As the variable’s type, select String, and as its value, enter your NDK path (to which we previously referred as <android_ndk>):

[20]
Troubleshooting the USB connection

If your Android device does not appear in Eclipse's Android Device Chooser window or if your adb commands fail in the command prompt, the USB connection might be at fault. Specifically, USB communication with the Android device is controlled via a tool called Android Debug Bridge (ADB), and this tool (or some other component of the connection) might not be working as expected. Try the possible solutions in this section.

```
To verify whether a USB connection is working, run the following command in a command prompt:

$ adb devices

If the connection is working, the terminal should print the serial number and name of your connected Android device, such as 019d86b921300c7c device
```
Setting Up OpenCV

Many connection problems are intermittent and can be resolved by restoring the USB connection to an initial state. Try the following steps and, after each step, test whether the problem is resolved:

1. Unplug the Android device from the host computer's USB port. Then, plug it back in.
2. Disable and re-enable the device's USB debugging option, as described earlier in the section Building the OpenCV samples with Eclipse.
3. On Mac or Linux, run the following command in Terminal (or another command prompt):
   
   ```sh
   sudo sh -c "adb kill-server && start-server"
   ```

Less commonly, connection problems might relate to drivers or permissions. A one-time setup process, as described next, should resolve such problems.

On Windows, we might need to manually install the USB drivers for the Android device. Different vendors and devices have different drivers. The official Android documentation provides links to the various vendors' driver download sites at http://developer.android.com/tools/extras/oem-usb.html#Drivers.

On Linux, before connecting an Android device via USB, we might need to specify the device's vendor in a permissions file. Each vendor has a unique ID number, as listed in the official Android documentation at http://developer.android.com/tools/device.html#VendorIds. We will refer to this ID number as `<vendor_id>`. To create the permissions file, open a command prompt application (such as Terminal) and run the following commands:

```bash
$ cd /etc/udev/rules.d/
$ sudo touch 51-android.rules
$ sudo chmod a+r 51-android.rules
$ sudo gedit 51-android-rules
```

Note that the permissions file needs to have root ownership, so we use `sudo` while creating or modifying it. Now, open the file in an editor such as gedit:

```bash
$ sudo gedit 51-android-rules
```

For each vendor, append a new line to the file. Each of these lines should have the following format:

```plaintext
SUBSYSTEM=="usb", ATTR{idVendor}="<vendor_id>", MODE="0666",
GROUP="plugdev"
```

Save the permissions file and quit the editor. Reboot.

On Mac, no special drivers or permissions are required.
Finding the documentation and help

The OpenCV Java API and C++ API are both relevant to Android. The Java API documentation is online at http://docs.opencv.org/java/, and an index of OpenCV4Android resources is online at http://opencv.org/platforms/android.html. The C++ API documentation is online at http://docs.opencv.org/. The following documents, which mostly use C++ code, are also available as downloadable PDF files:

- Tutorials: http://docs.opencv.org/opencv_tutorials.pdf
- User guide (incomplete): http://docs.opencv.org/opencv_user.pdf

If the documentation does not seem to answer your question, try talking to the OpenCV community. Here are some sites where you will find helpful people:

- Official OpenCV forum: http://www.answers.opencv.org/questions/
- Jay Rambhia's blog: http://jayrambhia.wordpress.com/
- The support site for my OpenCV books: http://nummist.com/opencv/

Also, you can read or submit bug reports at http://code.opencv.org/projects/opencv/issues?query_id=4. Finally, if you need to take your issue to the highest authority, you can e-mail the OpenCV4Android developers at android@opencv.org.

Summary

By now, we should have an Android and OpenCV development environment that can do everything we need for the application described in this book’s remaining chapters. Depending on the approach we took, we might also have a set of tools that we can use to reconfigure and rebuild OpenCV for our future needs.

We know how to build the OpenCV Android samples in Eclipse. These samples cover a different range of functionality to this book’s project, but they are useful as additional learning aids. We also know where to find the documentation and help.

Now that we have the necessary tools and reference materials to hand, our first ambition as application developers is to control a camera! Throughout the next chapter, we will use Android SDK and OpenCV to preview, capture, and share photographs.
Where to buy this book

You can buy Android Application Programming with OpenCV 3 from the Packt Publishing website.

Alternatively, you can buy the book from Amazon, BN.com, Computer Manuals and most internet book retailers.

Click here for ordering and shipping details.