Basic Android Setup for Machine Vision

6.870 Fall 2015

Introduction

Here we will learn how to set up the Android software development environment and how to implement machine vision operations on an Android device[†]. Android is an open-source platform developed by Google and the Open Handset Alliance on which interesting and powerful new applications can be quickly developed and distributed to many mobile devices. There is a large, growing community of Android developers and a vast selection of Android devices, which includes smartphones, tablets, watches and TV setup boxes. Android also comes with an extension library of useful functions, including functions for user interfaces, image/bitmap manipulation, and camera control.

The note is split into two parts. In the first part, we will explain how to download and install the Android software tools onto your computer. Then, in the second part, we will explain how to develop image processing programs that can run on an Android device.

Part I: Creating the Integrated Development Environment (IDE)

We will use Android Studio to design, implement, and debug android-compatible programs. It is possible, instead, to use the Eclipse IDE with the Google Android SDK for development. However, Android Studio is now the official Android IDE for Android application development[‡]. Instructions here are aimed at installation in Windows, instructions for Mac OS and Linux can be found on the Android Studio website.

A. Downloading and installing Java SE Development Kit

Android devices are programmed using Java[‡]. Java is used so that the programs can run on several different platforms without recompilation. (The *same* byte code works on four different processors used in Android

---

[†] Example adapted from Stanford EE368/CS232 Digitial Image Processing Notes
http://web.stanford.edu/class/ee368/Android/Tutorial-1-Basic-Android-Setup-Windows.pdf

[‡] The NDK (native development kit — needed for generating native code), supported under Google Android SDK and the Eclipse IDE, is not supported in Android Studio 1.2

products – 32-bit and 64-bit ARM, x86, and MIPS CPU architectures). Android Studio will need the Java SE Development Kit (JDK) from Oracle.

Check whether you have the Java SE Development Kit (JDK) version 7. JDK version 7 is required when developing for Android 5.0 and higher. The Java Runtime Environment, (JRE), alone is *not* sufficient. To check that you have the JDK installed, and which version you have, in "Control Panel" select "Programs and Features," scan down to "Java SE Development Kit" and check the number after that. Additional detail will be displayed if you click on that entry. (Note that browser-based tests for Java version number typically no longer work because of security settings in browsers).

If the JDK is not found, or the version number is too low, download the JDK from Oracle.

1. Download the JDK from this website
   `http://www.oracle.com/technetwork/java/javase/downloads/index.html`
   or, specifically for version 7,
   `http://www.oracle.com/technetwork/java/javase/downloads/jdk7-downloads-1880260.html`
   (This is a large file (140-180 MB) so its best to do this when on a high speed connection.)

2. Execute the downloaded installer. Accept the defaults. It is a good idea to make a note of which folder java is installed into.

B. Downloading and installing Android Studio itself

1. Next, download Android Studio from
   `https://developer.android.com/sdk/index.html`.
   (This is a very large file (800-900 MB) so its best to do this when on a high speed connection.)

2. To install, just follow the directions at
   `https://developer.android.com/sdk/installing/index.html?pkg=studio`

You can install in the default location, or use a separate folder with an easy-to-remember name, like `AndroidStudio`. Make this choice now, since it not possible to change later.

When you start Android Studio, it may notice that a more up to date version is available and ask to download and install the incremental update files for that. Later, you can "Check for Updates" manually — but it is easier if Android Studio is set up to check for updates automatically. If needed, change this using the "Updates" dialog of the IDE's "Settings."

C. Installing the Android System Development Kit (SDK)

In addition to Android Studio itself, you'll need the SDK containing various packages before you can develop apps for an Android device.

1. Launch Android Studio. Select "Configure," then "SDK Manager." Make sure that at least the following boxes are checked (several will already be checked automatically for you, leave those checked):

Under "Tools" check:
"Android SDK Tools",
"Android SDK Platform-tools",
"Android SDK Build-tools" (highest version).

Under "Android 5.1.1 (API 22)" check:
"SDK Platform"

Finally, under "Extras," check
"Android Support Repository"
"Android Support Library"
"Google USB Driver"

2. Click "Install <number> packages," choose "Accept License" for all items listed, and click "Install." The selected packages will now be downloaded and copied to your Android SDK installation folder. This may take a while. You can monitor the download/installation progress at the bottom of the Android SDK Manager window (Do not exit the SDK Manager or it will cancel the download).

If "SDK Manager" does not show up as an option in Android Studio under "Configure," then try running Android Studio "As Administrator" (right click on the Android Studio icon and select "Run as administrator").

If Java cannot be found, then you may need to set up an environment variable named `JAVA_HOME` that has as its value the folder that java was installed into. To do this, from Control Panel, select "System" and then "Advanced Settings" and then "Environmental Variables." Add a new system variable named `JAVA_HOME`. (For further details on setting environment variables see next subsection below).

Later, it may be a good idea to periodically launch the SDK Manager to check whether new versions of packages in the SDK are available (the SDK is not automatically updated).

If you encountered problems in this section, please take a look at the tips on these sites:

http://developer.android.com/sdk/index.html
http://developer.android.com/sdk/installing/index.html?pkg=studio
http://developer.android.com/sdk/installing/adding-packages.html

D. Optional: Environment Variables and Batch Files

For some purposes it may be helpful to add the location of the `tools` and `platform-tools` subfolders for the Android SDK to your system `PATH`. How to add folders to the `PATH` depends on which version of Windows you are running. For help on editing the `PATH`, please follow the tips here:

http://www.computerhope.com/issues/ch000549.htm

Alternatively, you may find it useful to create suitable desktop shortcuts or batch files. For example, if you wanted to run the Android Debug Bridge (ADB) (see below for more details) from the command line, then you can set up a batch file calling

```
c:\AndroidStudio\SDK\platform-tools\adb
```

or

```
c:\Users\<name>\android-sdks\platform-tools\adb
```

(or wherever your `adb.exe` is installed — use "Search" to find it).

E. Linking Your Android Device/Smartphone to Your Computer

1. Turn on your Android device, if it is not on already. Screen-unlock it.
2. Go to the home screen on the Android device. Find "Settings" (where it is found depends on the version of Android your device is running).
3. If your device has not been used for development before, then you need to enable the development settings. How to do this depends on the version of Android you are running. Try the following. Go to "Settings > System > About Phone > Build Number." Tap it seven times! This will create a new menu entry for "Developer Options."
4. Select "Settings > System > Developer Options" and enable "Debugging > USB debugging." This allows loading and controlling of apps over USB cord from your laptop (via Android Debug Bridge, ADB).
5. Connect the device to your computer via USB cable. The USB driver is included in the updates you downloaded for the Android SDK above. So Windows should install the driver for you automatically when you first plug in an Android device. If needed, follow the tips on:
   http://developer.android.com/tools/device.html

`http://developer.android.com/sdk/win-usb.html`

`http://developer.android.com/tools/extras/oem-usb.html`

You may need to restart your computer after installing the USB driver.

5. For some purposes, such as downloading an app file (extension `.apk`), you may also want to enable "Settings > Personal > Security > Unknown Sources" on the device.

Part II: Developing Image Processing Programs for Android

Now that the Java JDK, Android Studio, and Android SDK are all set up on your computer, we are ready to start writing image processing programs that can run on Android-compatible devices. But first — the obligatory

A. Hello World Example

We will build a simple Android program in Android Studio. This simple example will help you become familiar with how to create an Android project, how to (auto) compile source code, and how to run the generated executable on the Android device. Please follow the instructions to develop your "First App" on the page:

`http://developer.android.com/training/basics/firstapp/index.html`

You may let Android Studio put your project files into a default workspace or a folder with an easy-to-remember name that you pick, like `AndroidStudioProjects`. You make that choice when you create your first project, and it will remember (Note that it is hard to move a project once it has been created). Ignore instructions for creating the project using command line tools.

If you get "Invalid Gradle JDK configuration found." then under Android Studio, select "File > Project Structure > JDK location." Specify the folder of your JAVA SDK.

Next, run the app on your Android device, as described on

`http://developer.android.com/training/basics/firstapp/running-app.html`

Make sure your device is properly linked to your computer (see above). Use "Run > Run app" (or the "Run" button — the green right-pointing triangle icon). The app will be compiled and built automatically if needed. (Again, ignore the instructions for running the app from a command line).

If your device does not show up in the list of devices, then (i) it may not be connected by USB cable, (ii) it may be screen-locked, (iii) it may be

powered off, (iv) the USB driver may not be installed, or, (v) you may not have enabled "USB Debugging," as described above).

It is also possible to run the app on an emulator, as described on the above referenced web page. This is done by first creating an Android Virtual Device (AVD) and using that instead of a physical device attached via USB cord. This can be useful for checking whether the app will run on a variety of devices with different screen sizes and different versions of Android, without actually having access to those devices.

Of course, some sensors, such as accelerometers and gyroscopes, can not (or just are not) emulated. Similarly, the image processing applications we will be building depend on real time processing of input from a camera and so will not run as expected on an emulator. As a result, we will not pay much attention to emulation and AVDs here.

It may be beneficial to continue further with the tutorial on

`http://developer.android.com/training/basics/firstapp/building-ui.html`

and

`http://developer.android.com/training/basics/firstapp/starting-activity.html`

By the way, in case you haven't noticed it yet, the developer site

`http://developer.android.com`

has an overabundance of useful information, incuding details on the API! Finally, note that, if you wish to start on a new project, you have to first close the currently open project using "File > Close Project."

B. Viewfinder Example

Now, having grasped the fundamentals of building and running an Android application, we will create a more interesting project involving the onboard camera and real-time image processing. You may be able to use this as a template for your own machine vision project.

1. Start a new Android Studio project with the following parameters:
   Application name: Viewfinder
   Company Domain: example.com
   Minimum SDK: API 15: Android 4.0.3
   Added Activity: Blank Activity
   Accept the default values on the next page for "Activity Name" etc.

2. While in Android Studio, make changes in the file
   `ViewFinder/app/src/main/AndroidManifest.xml`

(which defines the main activities and permissions for this program). Add the lines

```
<uses-permission android:name="android.permission.CAMERA"/>
```

and

```
<uses-feature android:name="android.hardware.camera" />
```

before the line starting with `<application>`. Also add the line

```
android:screenOrientation="landscape"
```

before the line starting with `android:label=...`

3. Again, while in Android Study, copy and paste the text from
   `http://people.csail.mit.edu/bkph/courses/6870/ViewFinder/MainActivity.java`
   to replace the text in the file
   `ViewFinder/app/src/main/java/com/example/viewfinder/MainActivity.java`
   This file defines the classes in the application.

4. Check to make sure everything is copied correctly into the project. If there are compilation errors, these will be marked in the source files.

5. Select "Run" and then, in the Device Chooser dialog, select your device. On your Android device, you should see the preview image from the camera displayed with some graphics and text overlaid. What do you see? Point the camera at different objects around you to see how the information displayed changes. Note that the information is updated in real time! Take a screen shot (see below).

For more information about using the `camera` class see*:

`https://developer.android.com/reference/android/hardware/Camera.html`

For more information about using the camera in general see*:

`http://developer.android.com/guide/topics/media/camera.html`

To learn more about the structure of the "ViewFinder" project, (i) change the text displayed on the screen, (ii) change the font size, and (iii) change where the text appears on the page. Take a screen shot (see below).

C. Real-time Device Debugging in Android Studio

It is possible to view real-time log messages from the device in Android Studio, which can be very helpful for debugging and code development.

*Note that the `Camera` class is "deprecated" as of Android API 21. The new `Camera2` API (not a class) is more flexible, powerful and complex. However few devices support the new API so far (Motorola Nexus 5 & 6 and just possibly Samsung Galaxy S6 & S6 Edge), so it makes sense to continue using the `Camera` class for now, as we did in the "ViewFinder" sample app.

Tagged messages can have different "severity" ratings (Verbose, Debugging, Information, Warning, Error, Assertion).

After you press "Run" a window opens at the bottom of Android Studio which can display a sequential list of real-time "logcat" output from the device (If for some reason this window is not visible, use "View > Tool Windows > Android"). The various severity levels are color coded (red for the most severe) and log entries can be filtered to show only the more serious log entries, or the ones coming from your specific application.

Logcat tagged messages provide a very useful debugging technique but can also be used just for displaying parameters of interest.

You can add your own debugging output using `Log.e(...)` etc. from the `android.util.Log` package. For more details, see

`http://developer.android.com/reference/android/util/Log.html`

D. Taking a Screenshot of the Android Device Display

At some point, it may be useful to take a screenshot of the device, e.g., to use as a figure in a paper or project report. This is easy in Android Studio. Click on the camera icon with the blue lens in the left edge of the "Android" Window (if for some reason this window is not visible, use "View > Tool Windows > Android"). The screen shot will appear in a separate window, in which it can be manipulated and saved on your laptop.

You can record a video (of limited duration) of screen activity on your device using the white triangle in a green square button right below the above mentioned camera icon.

You can also take a screen shot directly on many devices by some combination of button pushes. For example, on the Samsung Galaxy Nexus and Motorola Nexus 5, simultaneously push the "volume down" and the "power" buttons. On Samsung Galaxy Note 3, push the "home" and "power" buttons. This creates a file on the Android device, which you can then transfer to your computer, perhaps by email, or via the Android Debugging Bridge (ADB — see below), or file transfer (the device's memory appears as a "media device" in "My Computer" on your laptop).

E. Showing the Android Device Display on your Laptop

It is also possible to show the device screen on your laptop using a program called "Droid@Screen" found at

`http://droid-at-screen.org/download.html`

The downloaded compiled Java `.jar` file can be run using your existing java implementation. If java was properly installed, just doubling clicking on

the downloaded file (e.g. `droidAtScreen-1.2.jar`) will launch java. You will see an image on your laptop of what the screen of your device is showing (you may as well set up a new "Shortcut" to this file on your desktop if you plan on using it frequently). This is very handy when giving presentations.

Droid@Screen will search for the ADB (Android Debug Bridge) executable (`adb.exe`) and if it cannot find it, it will prompt you for the full path (It also checks the environment variables `ANDROID_HOME` and `ANDROID_SDK_HOME` which may be pointing the the Android SDK installation directory).

The screen update rate is limited by the USB connection (which in almost all devices is still USB 2.0) to about one frame every second or two. You can easily customize the size of the image on your screen using "Options > Preferred Scale." Droid@Screen is also able to save screenshot image files, and to record a sequence of screen shots (see icons on the window with the device screen display).

F. Listeners and Receivers

Much programming on Android is done using "callbacks." For example, when dealing with the touch screen one can set up a "listener" that is called whenever there is a change in the state of the screen. This "interrupt driven" process is to be preferred to a "polling" mode where the application explicitly asks for the information whenever it needs it. See

`http://developer.android.com/guide/topics/ui/ui-events.html`

Similarly, one can register a "receiver" for general "broadcasts." See e.g.

`http://developer.android.com/reference/android/content/BroadcastReceiver.html`

Dealing with video similarly involves callbacks for each frame. To see how this might be done, take a look at the sample "ViewFinder" java code.

G. The Android Life Cycle

Your application lives in the Android world and has a "life cycle." It should provide "callback" functions that deal with starting, pausing, resuming and stopping. Your "Main Activity" which extends "Activity," should override `onCreate`, which is called when the app is started. This is the place to do all the initialization that is needed. Symmetrically, there is an `onDestroy` that can be overriden to deal with last minute cleanup before the app if flushed from memory (although there is actually no guarantee that this will always be called).

More typically, your app is simply put in the background rather than destroyed. In this case `onPause` is called (e.g. when the user switches to another app, or when the "Back" button is pressed while your app is active).

Correspondingly there is `onResume` which is called when focus comes back to your app. `onPause` and `onResume` are good places to save and restore your app's state. See also

`http://developer.android.com/training/basics/activity-lifecycle/index.html`

Since the cameras are shared resources, apps should be polite about always releasing them in `onPause`, and (re-)opening them in `onResume` (as opposed to opening them in `onCreate` for example). To see how any of this might be done, take a look at the sample "ViewFinder" java code.

H. Some Low Level Nitty Gritty — Android Debug Bridge (ADB)

The interaction between Android Studio and your device goes over the Android Debug Bridge. You can actually use ADB directly in Windows, from the command line ("Start > Command Prompt"). For this purpose it is handy to have a batch file (e.g. `adb.bat`) that calls `adb.exe`, which may, for example, be in

`c:\AndroidStudio\SDK\platform-tools\adb`

or

`c:\Users\<name>\android-sdks\platform-tools\adb`

(or wherever your `adb.exe` is installed — use "Search" to find it.

Calling ADB without any command arguments will give a listing of its command line arguments. You can "push" and "pull" files from and to your device and "install" apps (file extension `.apk`). You can also receive the debugging "logcat" output on your computer (e.g. to receive low level details about the radio telephony use `adb logcat -b radio`).

You can even open a "shell" on the device and talk directly to the operating system, listing files and directories etc. (It's Linux after all)!

H. Adding Information Output to ViewFinder

As an exercise, download the java class file

`http://people.csail.mit.edu/bkph/courses/6870/ViewFinder/ExtraInfo.java`

and insert it into your ViewFinder project. The "ExtraInfo" class provides methods for dumping intersting information about the display screen and the cameras into the log file. Where in "MainActivity" would be a good place to call these methods? How would you call them? What is the focal length of the rearward looking camera in your Android device?