Lab Report: Face Detection Recorder

Name: Bingbing Huang
Instructor: Bilge Soran
Steele Lab
University of Washington
Spring 2016

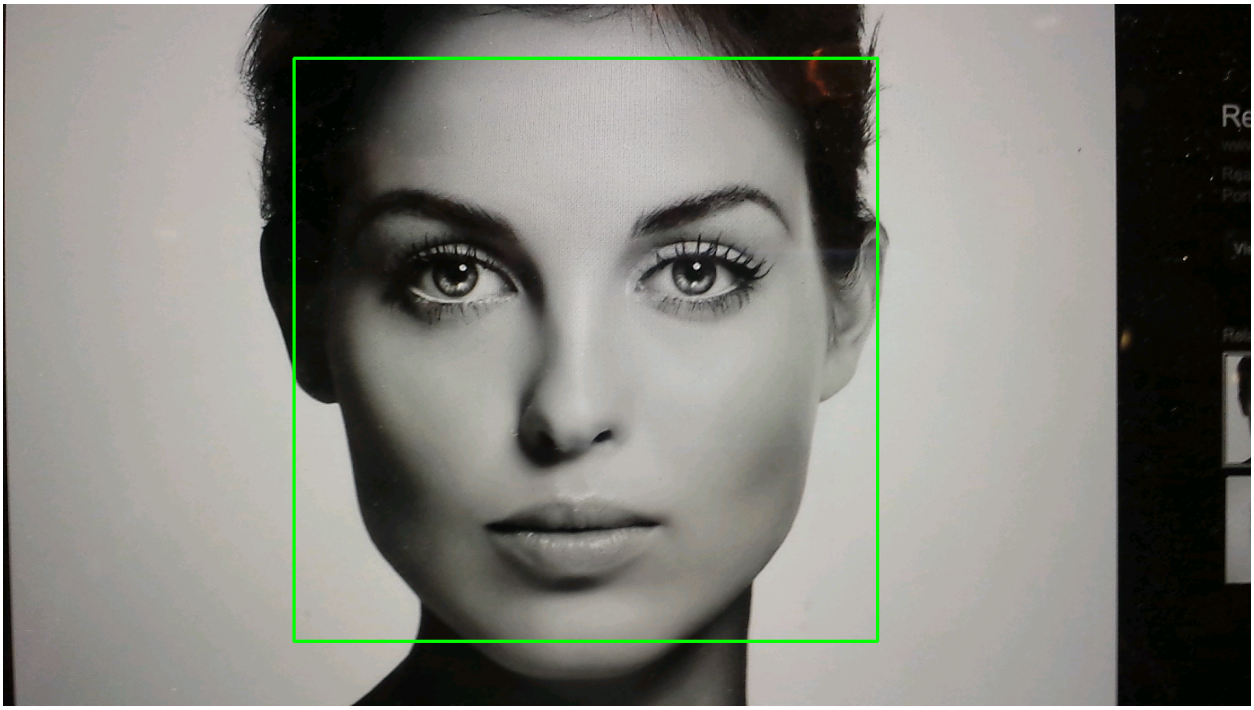
Table of Contents

Introduction

This lab focus on developing a mobile application with face detection function in order to get a better perspective for the movement of babies. This mobile app is based on Android platform with Android Studio IDE, OpenCV and JavaCV libraries.

- Android Studio is the official IDE for Android platform development.
- OpenCV is a library of programming functions mainly aimed at real-time computer vision,
- JavaCV uses wrappers from the JavaCPP Presets of commonly used libraries by researchers in the field of computer vision (OpenCV and FFmpeg will be the mainly two libraries we used in this app), and provides utility classes to make their functionality easier to use on the Java platform, including Android.

This mobile app is able to record a video with native camera, and save the video file to the device. After that, the video will be processed, and the faces will be detected and blurred. The processed video will be saved back to device along with the input video.

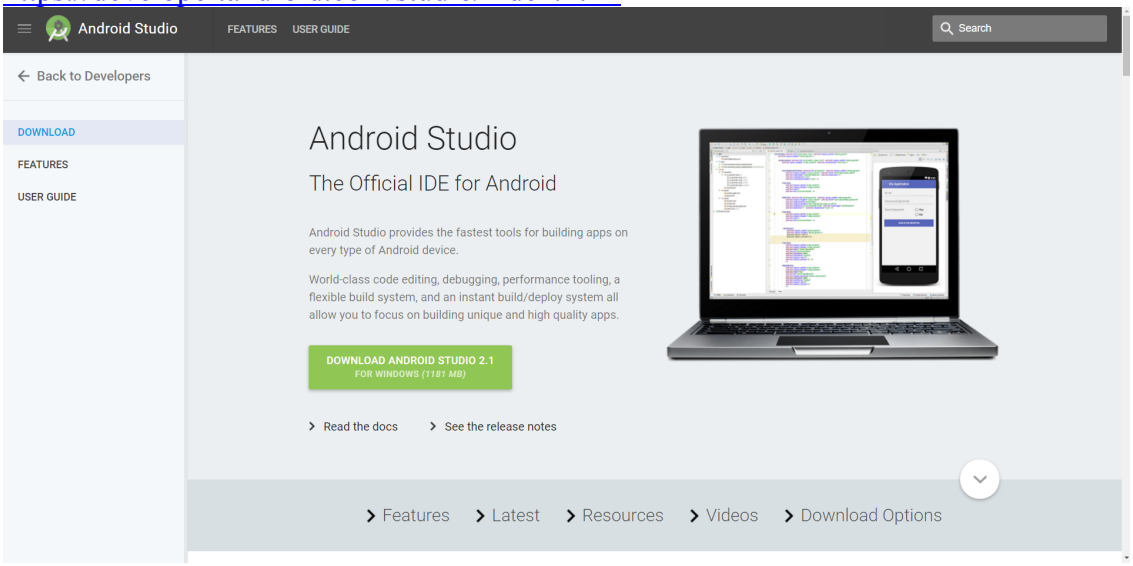


Preparation

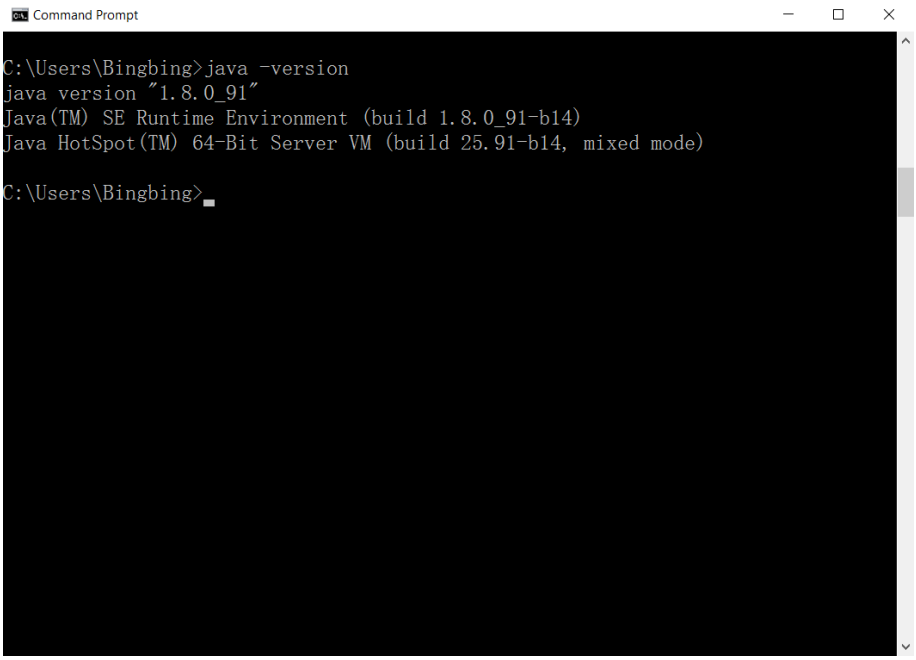
Install Android Studio

Windows Version:

a) Go to the official website of Android Studio, and download the latest version of the IDE https://developer.android.com/studio/index.html



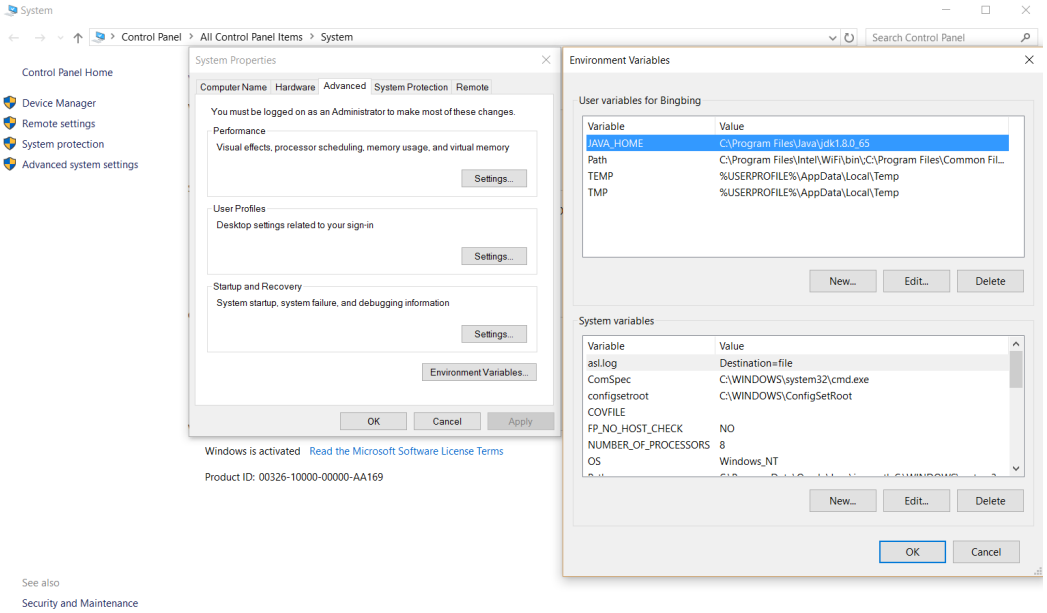
b) While the Android Studio download completes, verify which version of the JDK you have: open a command line and type **java -version**. If the JDK is not available or the version is lower than 1.8, download the Java SE Development Kit 8 with your computer version(http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html)



c) Launch the .exe file you downloaded.

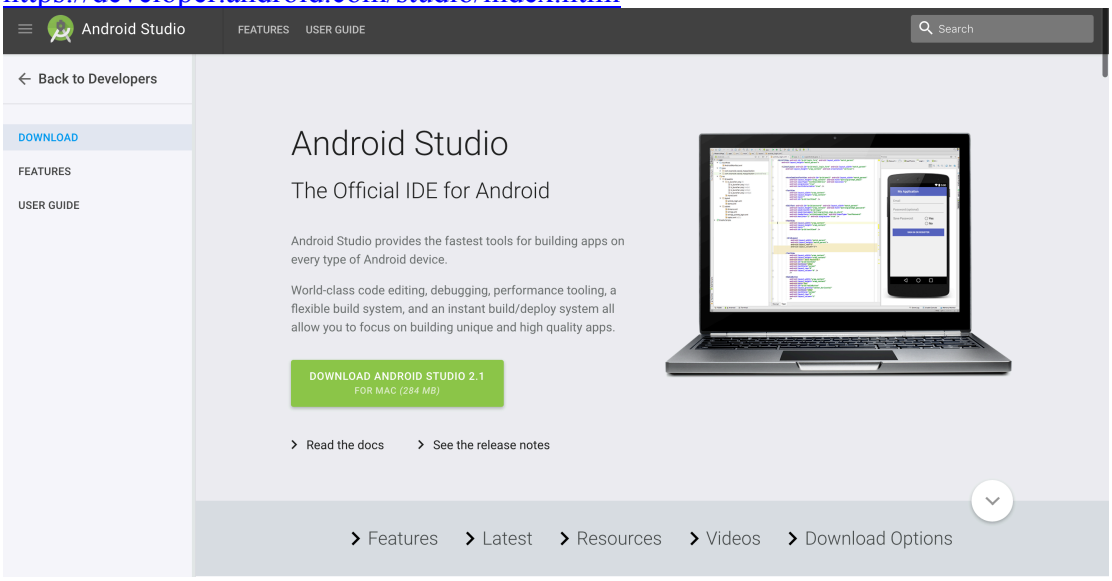
d) Follow the setup wizard to install Android Studio and any necessary SDK tools.

- On some Windows systems, the launcher script does not find where the JDK is installed. If you encounter this problem, you need to set an environment variable indicating the correct location.
- Select **Start menu > Computer > System Properties > Advanced System Properties**. Then open **Advanced tab > Environment Variables** and add a new system variable JAVA_HOME that points to your JDK folder, for example C:\Program Files\Java\jdk1.8.0_77 (If you installed JDK with default path)

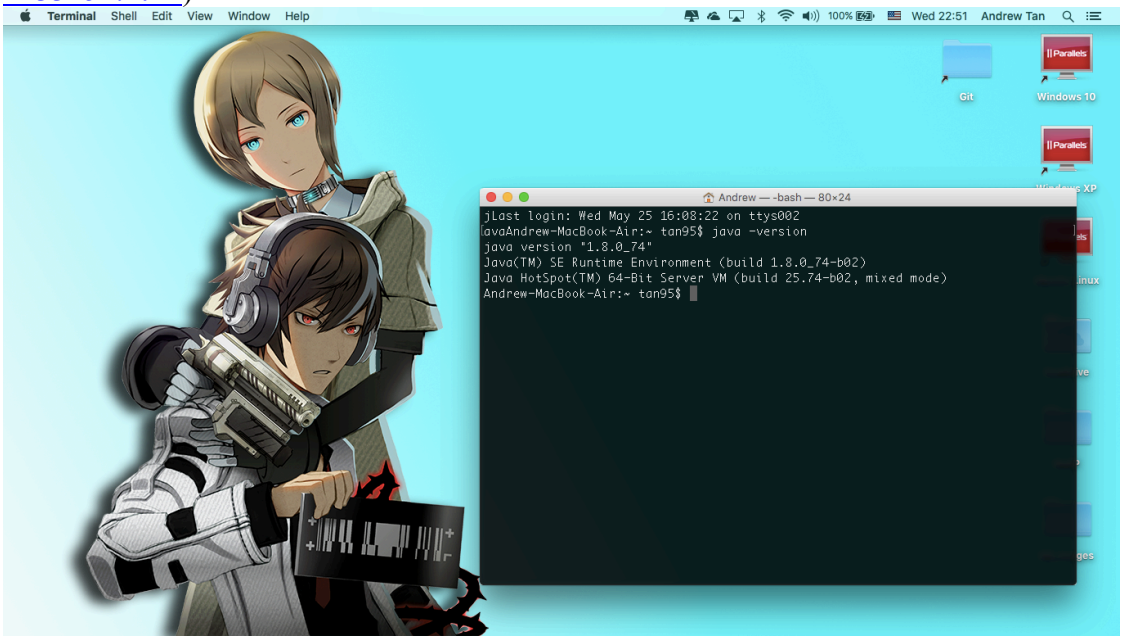


Mac Version:

a) Go to the official website of Android Studio, and download the latest version of the IDE https://developer.android.com/studio/index.html



b) While the Android Studio download completes, verify which version of the JDK you have: open a command line and type **java -version**. If the JDK is not available or the version is lower than 1.8, download the Java SE Development Kit 8 with your computer version(http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html)



c) Launch the Android Studio DMG file.

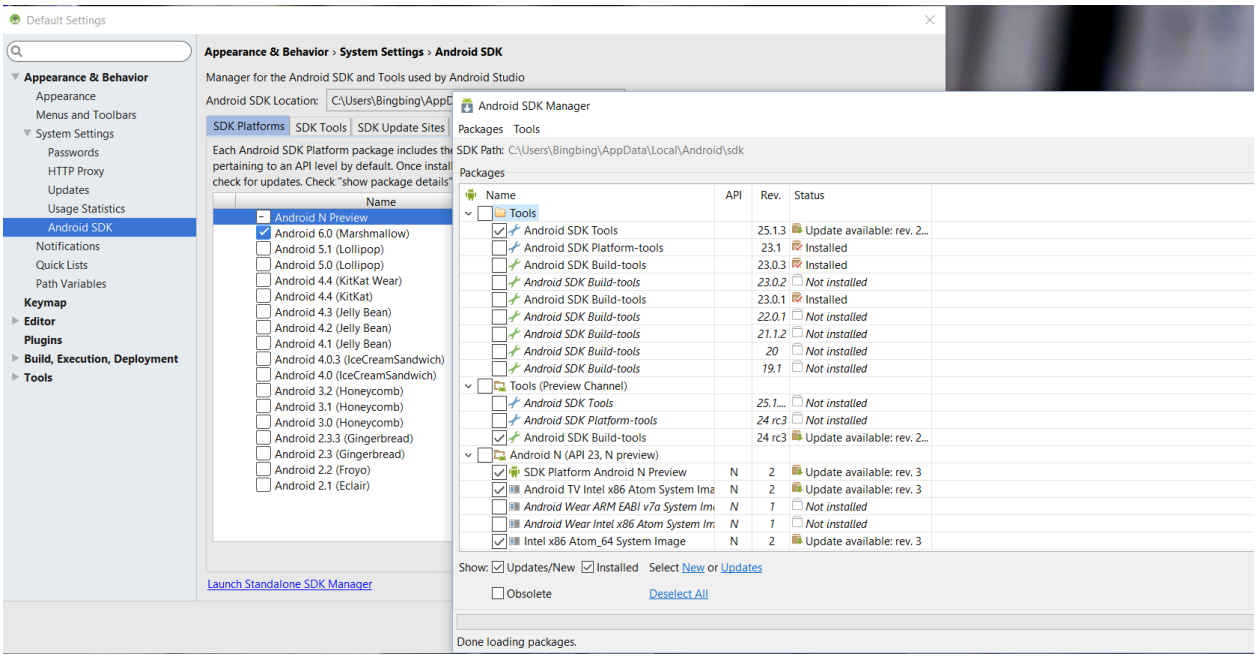
d) Drag and drop Android Studio into the Applications folder, then launch Android Studio.

e) Select whether you want to import previous Android Studio settings, then click OK.

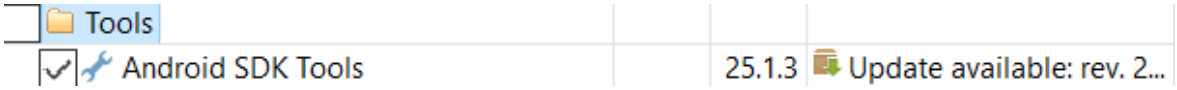
f) The Android Studio Setup Wizard guides you though the rest of the setup, which includes downloading Android SDK components that are required for development.

Additional Tool Install

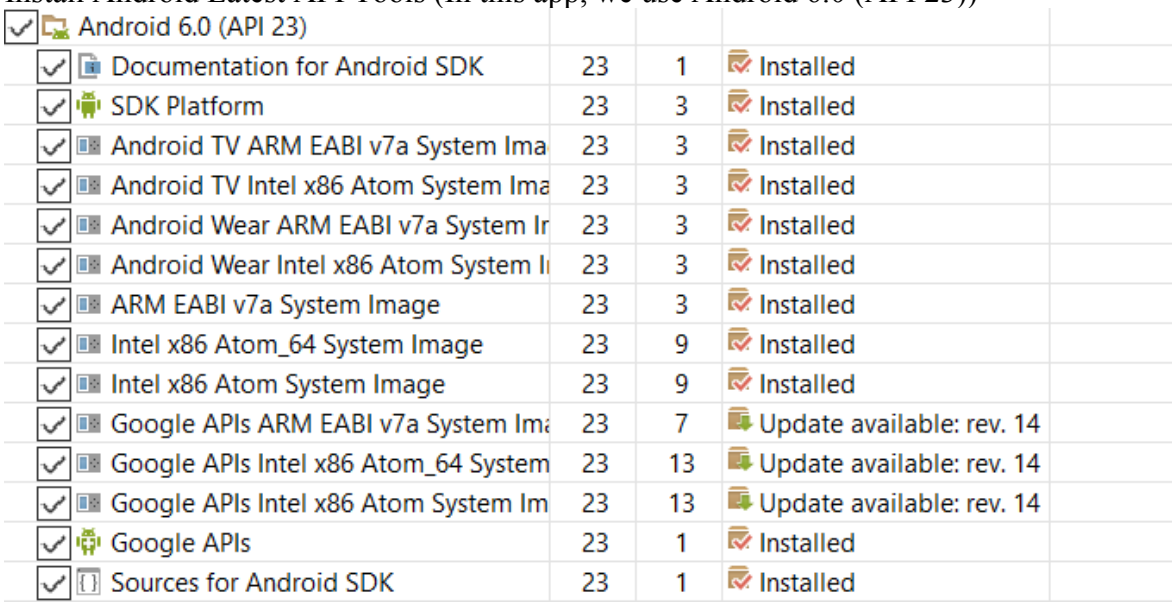
a) Open Android Studio, and go to **Configure > SDK Manager > Launch Standalone SDK Manager**



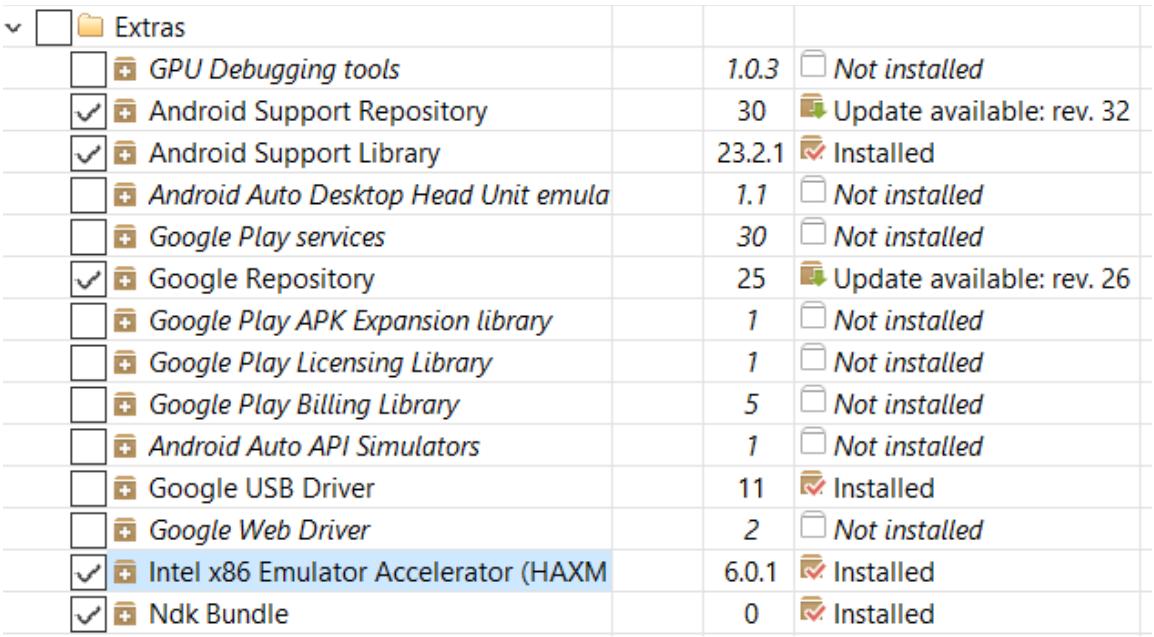
b) Install Tools: Android SDK Tools



Install Android Latest API Tools (In this app, we use Android 6.0 (API 23))



Install Extra: Android Support Repository, Android Support Library, Google Repository, Intel x86 Emulator Accelerator, and Ndk bundle.



Extras		
GPU Debugging tools	1.0.3	Not installed
Android Support Repository	30	Update available: rev. 32
Android Support Library	23.2.1	Installed
Android Auto Desktop Head Unit emula	1.1	Not installed
Google Play services	30	Not installed
Google Repository	25	Update available: rev. 26
Google Play APK Expansion library	1	Not installed
Google Play Licensing Library	1	Not installed
Google Play Billing Library	5	Not installed
Android Auto API Simulators	1	Not installed
Google USB Driver	11	Installed
Google Web Driver	2	Not installed
Intel x86 Emulator Accelerator (HAXM	6.0.1	Installed
Ndk Bundle	0	Installed

Create First Test Application (Hello World)

a) In Android Studio, create a new project:
- If you don't have a project opened, in the Welcome screen, click New Project.
- If you have a project opened, from the File menu, select New Project. The Create New Project screen appears.

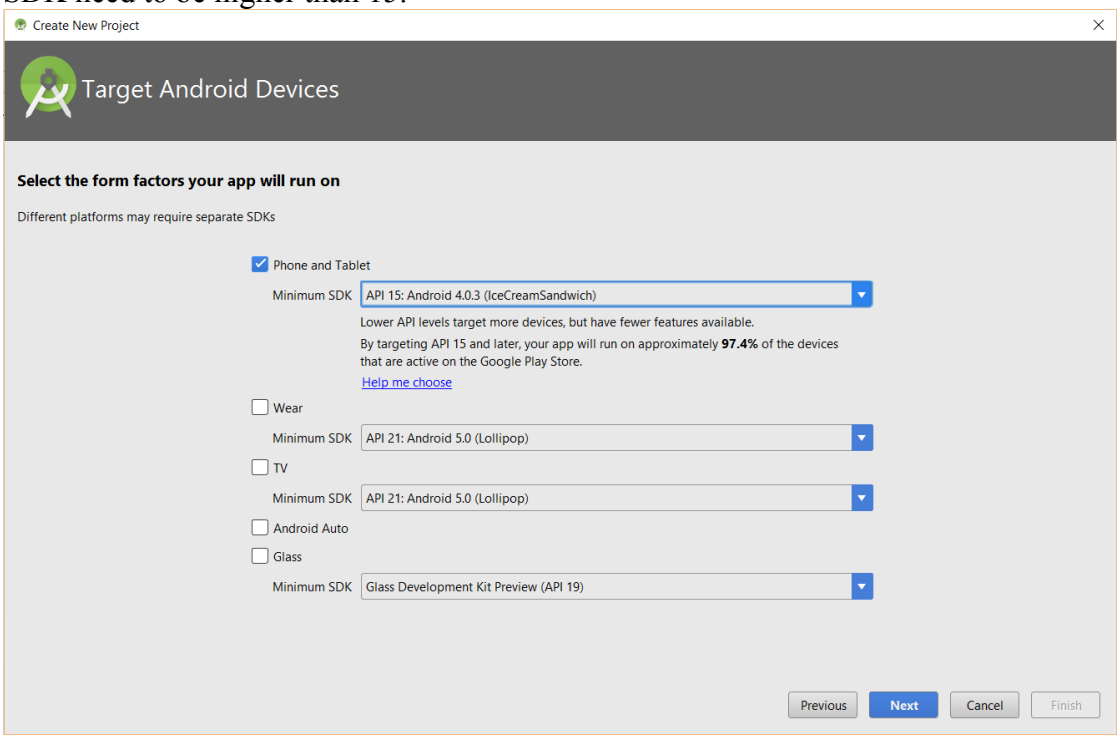
b) Fill out the fields on the screen, and click **Next.**
It is easier to follow these lessons if you use the same values as shown.
- **Application Name** is the app name that appears to users. For this project, use "My First App."
- **Company domain** provides a qualifier that will be appended to the package name; Android Studio will remember this qualifier for each new project you create.
- **Package name** is the fully qualified name for the project (following the same rules as those for naming packages in the Java programming language). Your package name must be unique across all packages installed on the Android system. You can Edit this value independently from the application name or the company domain.
- **Project location** is the directory on your system that holds the project files.

c) Under Select the form factors your app will run on, check the box for Phone and Tablet.
For **Minimum SDK**, select API 15:

- The Minimum Required SDK is the earliest version of Android that your app supports, indicated using the API level. To support as many devices as possible, you should set this to the lowest version available that allows your app to provide its core feature set. If

any feature of your app is possible only on newer versions of Android and it's not critical to the app's core feature set, you can enable the feature only when running on the versions that support it (as discussed in Supporting Different Platform Versions).

- In order to achieve some necessaries functionalities in this app, the Minimum Required SDK need to be higher than 15.



d) Leave all of the other options (TV, Wear, and Glass) unchecked and click **Next**.

Activities (terminology in Android Development)
An activity is one of the distinguishing features of the Android framework. Activities provide the user with access to your app, and there may be many activities. An application will usually have a main activity for when the user launches the application, another activity for when she selects some content to view, for example, and other activities for when she performs other tasks within the app.

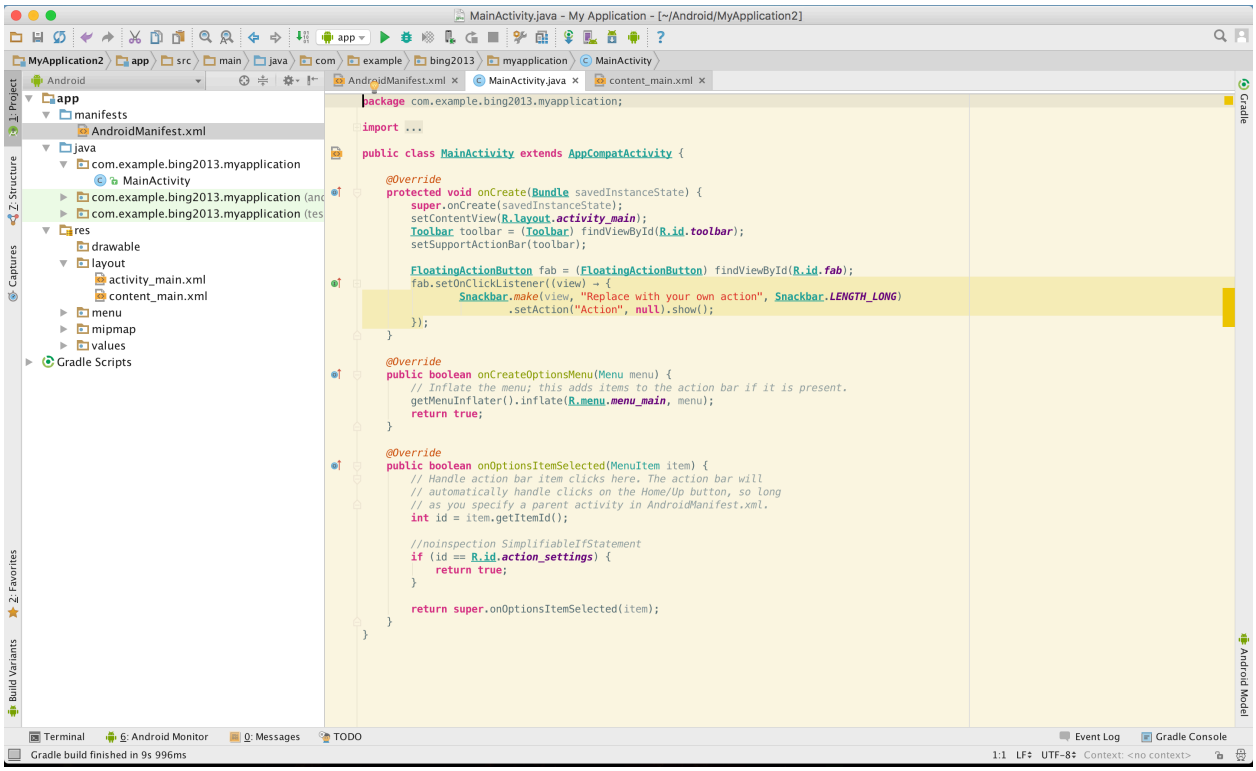
More information about activities:
https://developer.android.com/guide/components/activities.html

e) Under **Add an activity to <template>**, select **Blank Activity** and click **Next**.

f) Under **Customize the Activity**, change the **Activity Name** to *MyActivity*. The **Layout Name** changes to *activity_my*, and the **Title** to *MyActivity*. The **Menu Resource Name** is *menu_my*.

g) Click the **Finish** button to create the project.

Explanations for some important files in Android App Development



a. app/src/main/res/layout/activity_my.xml

This XML layout file is for the activity you added when you created the project with Android Studio. Following the New Project workflow, Android Studio presents this file with both a text view and a preview of the screen UI. The file contains some default interface elements from the material design library, including the app bar and a floating action button. It also includes a separate layout file with the main content.

b. app/src/main/res/layout/content_my.xml

This XML layout file resides in activity_my.xml, and contains some settings and a TextView element that displays the message, "Hello world!".

c. app/src/main/java/com.mycompany.myfirstapp/MyActivity.java

A tab for this file appears in Android Studio when the New Project workflow finishes. When you select the file you see the class definition for the activity you created. When you build and run the app, the Activity class starts the activity and loads the layout file that says "Hello World!"

d. app/src/main/AndroidManifest.xml

The manifest file describes the fundamental characteristics of the app and defines each of its components. You'll revisit this file as you follow these lessons and add more components to your app.

e. app/build.gradle

Android Studio uses Gradle to compile and build your app. There is a build.gradle file for each module of your project, as well as a build.gradle file for the entire project. Usually, you're only interested in the build.gradle file for the module, in this case the app or application module. This is where your app's build dependencies are set, including the defaultConfig settings:

- compiledSdkVersion is the platform version against which you will compile your app. By default, this is set to the latest version of Android available in your SDK. You can still build your app to support older versions, but setting this to the latest version allows you to enable new features and optimize your app for a great user experience on the latest devices.
- applicationId is the fully qualified package name for your application that you specified during the New Project workflow.

- minSdkVersion is the Minimum SDK version you specified during the New Project workflow. This is the earliest version of the Android SDK that your app supports.

- targetSdkVersion indicates the highest version of Android with which you have tested your application. As new versions of Android become available, you should test your app on the new version and update this value to match the latest API level and thereby take advantage of new platform features.

f. The /res subdirectories (contain the resources for your application)

- drawable-<density>/

Directories for drawable resources, other than launcher icons, designed for various densities.

- layout/

Directory for files that define your app's user interface like activity_my.xml, discussed above, which describes a basic layout for the MyActivity class.

- menu/

Directory for files that define your app's menu items.

- mipmap/

Launcher icons reside in the mipmap/ folder rather than the drawable/ folders. This folder contains the ic_launcher.png image that appears when you run the default app.

- values/

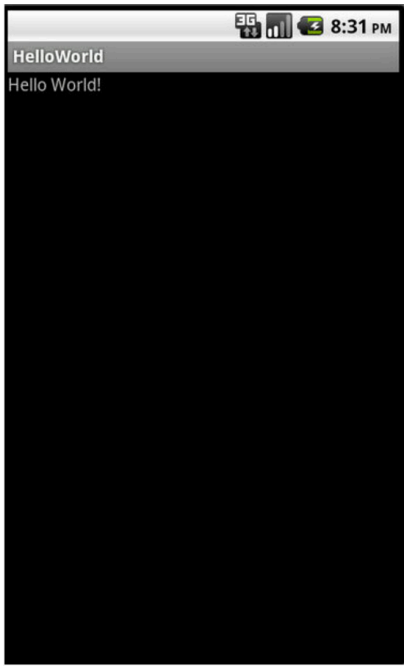
Directory for other XML files that contain a collection of resources, such as string and color definitions.

Running Your App (Test if HelloWorld is successfully created)

Run with Hardware Device (Real Android Phone)

- Plug in your device to your development machine with a USB cable.
- Enable **USB debugging** on your device. On Android 4.0 and newer, go to **Settings > Developer options**.

  Attention: On Android 4.2 and newer (Most our Android Phone has higher API than 4.2), **Developer options** is hidden by default. To make it available, go to **Settings > About phone** and tap **Build number** seven times. Return to the previous screen to find **Developer options**.
- Intsall OpenCV Manager in your device in order to use OpenCV Library
  Go to **Googe Play Store** > Search **"OpenCV Manager"** and install it
- Set up your system to detect your device.
  - If you're developing on Windows, you need to install a USB driver for adb.
  - If you're developing on Mac OS X, it just works. Skip this step.
- Open an Android Studio project and select Run icon.
- In **Select Deployment Target** > **Connected Devices**, select your Android Phone, and click **OK.**
- The App will run in your device. The picture below is running HelloWorld App in an Android Device.

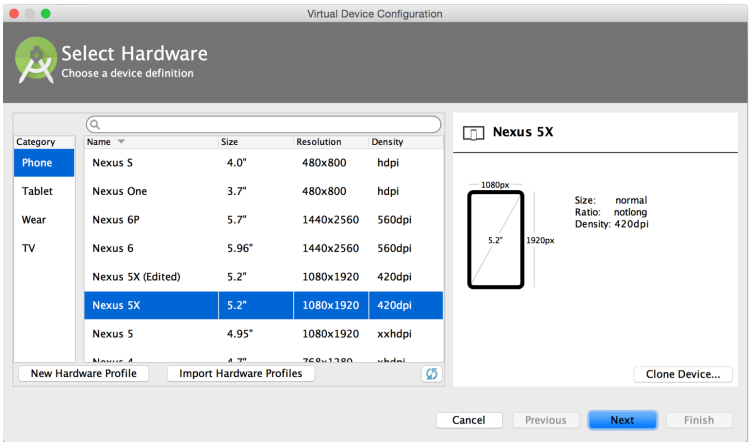


Run with AVD (Android Virtual Device)

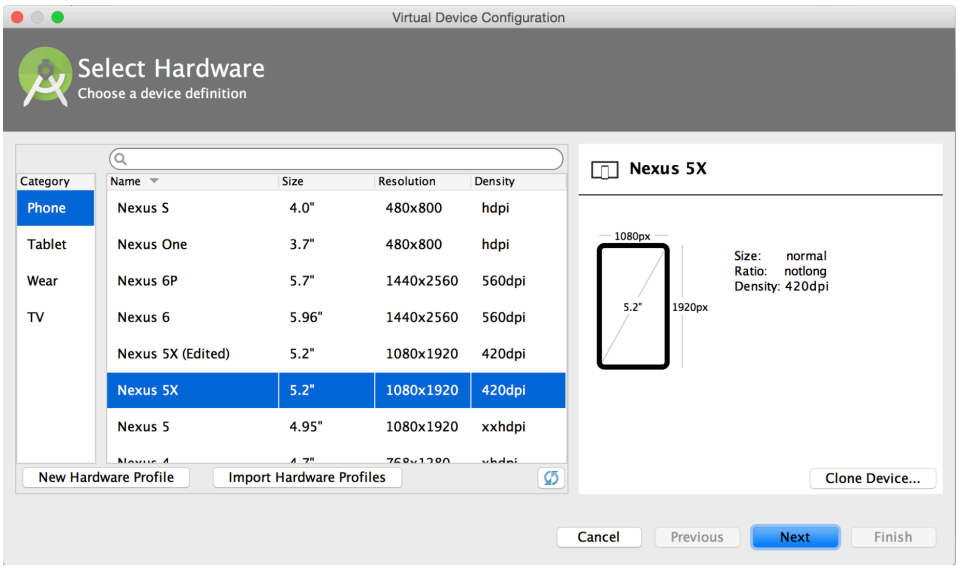
An **Android Virtual Device (AVD)** definition lets you define the characteristics of an Android phone, tablet, Android Wear, or Android TV device that you want to simulate in the Android Emulator.

Note: Even if you can use AVD to test some simple functions for the app, however, we need to have real Android Phone to test Native Camera API.

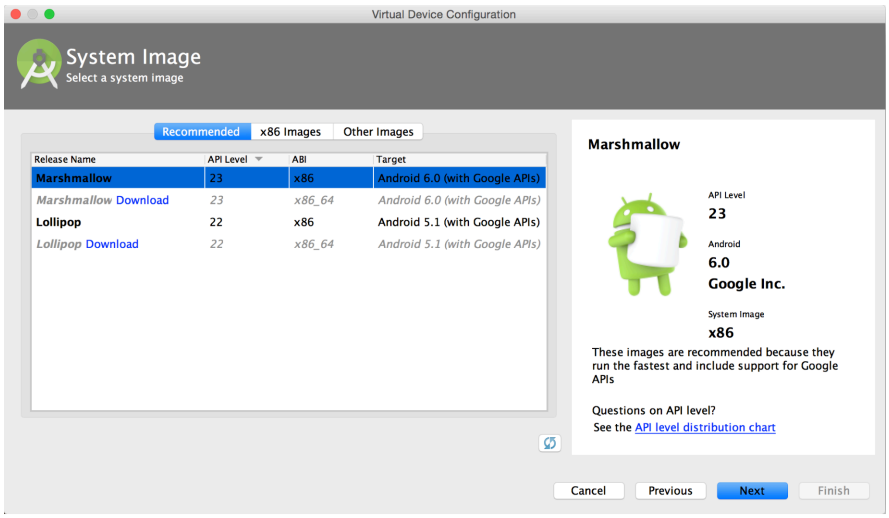
- Create AVD
  - In Android Studio, select **Tools** > **Android** > **AVD Manager**.
  - **Quick Access:** Click AVD Manager  in the toolbar.



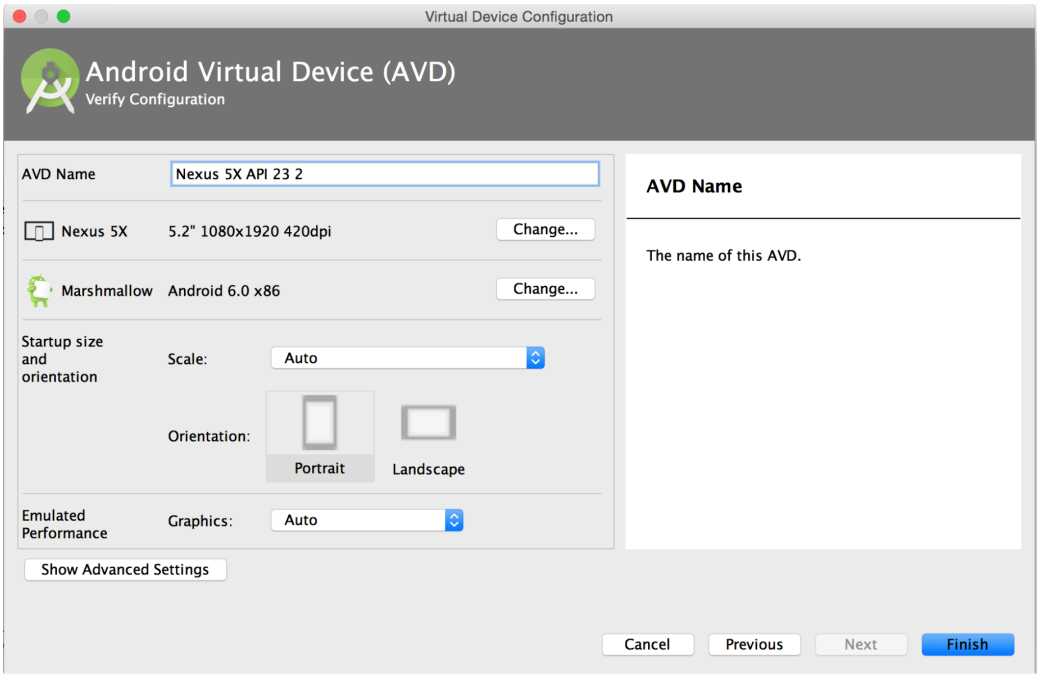
  - From the Your Virtual Device page of the AVD Manager, click **Create Virtual Device**
  - Select a hardware profile, and then click **Next**. For this app, **Phone > Nexus 5X** (Nexus 5X is good enough for our testing).



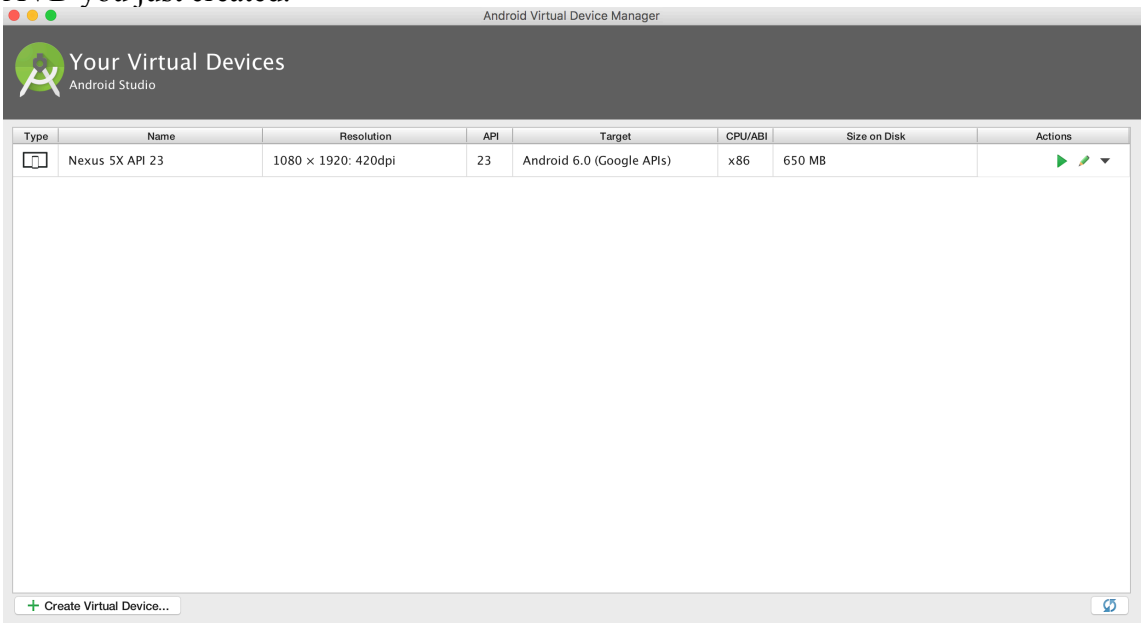
- Select the system image for a particular API level, and then click **Next**. (Basically, you will want to choose the highest API ensuring the test functionalities.)



- Select the system image for a particular API level, and then click **Next**.
  - The **Recommended tab** lists recommended system images. The other tabs include a more complete list. The right pane describes the selected system image. x86 images run the fastest in the emulator.
  - If you see **Download** next to the system image, you need to click it to download the system image. You must be connected to the internet to download it.

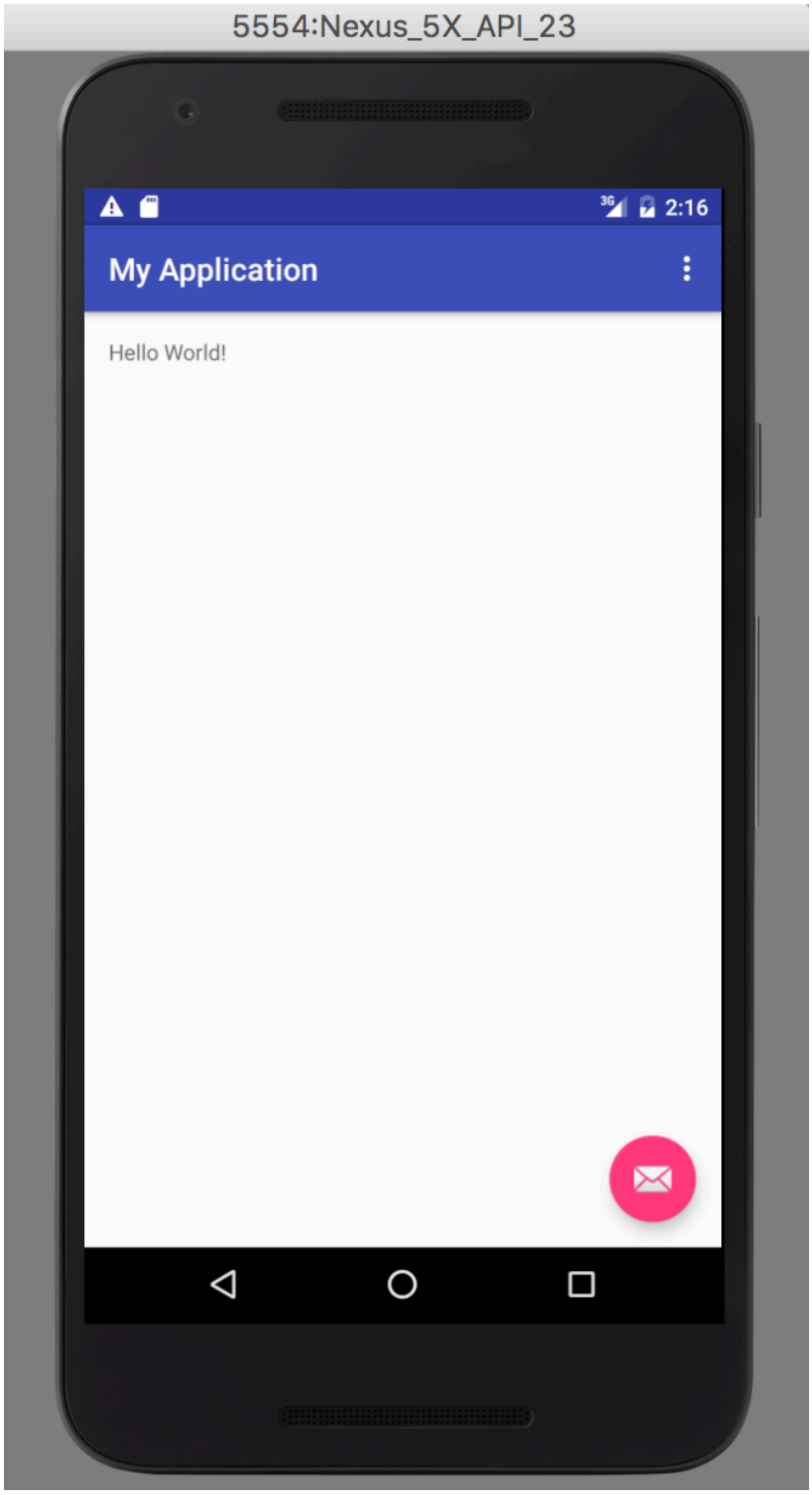


- Change AVD properties as needed, and then click **Finish**.
  - Click **Show Advanced Settings** to show more settings, such as the skin.
  - The new AVD appears in the *Your Virtual Devices page* or *the Select Deployment Target dialog*.
- Click  to open **Android Virtual Device Management**, and click ▶ to start the AVD you just created.



Note: You may want to keep your AVD open because it is more convenient for you to test your app step by step

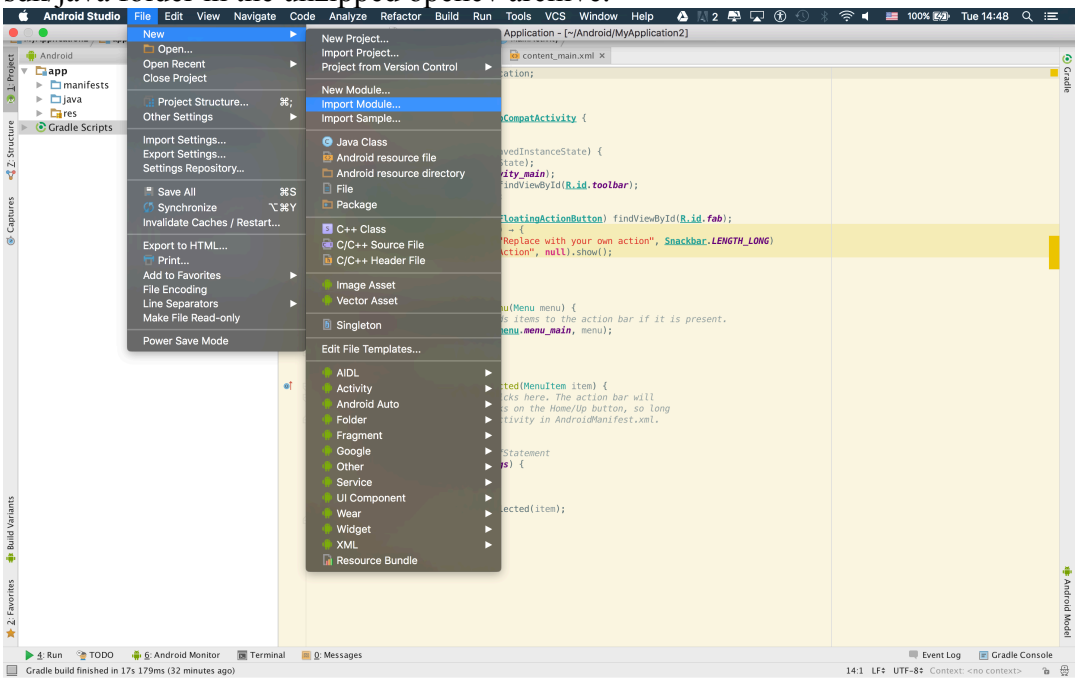
- Open an Android Studio project and select Run icon.
- In **Select Deployment Target** > **Connected Devices**, select your AVD, and click **OK.**
- The App will run in your AVD. The picture below is running HelloWorld App in an AVD.



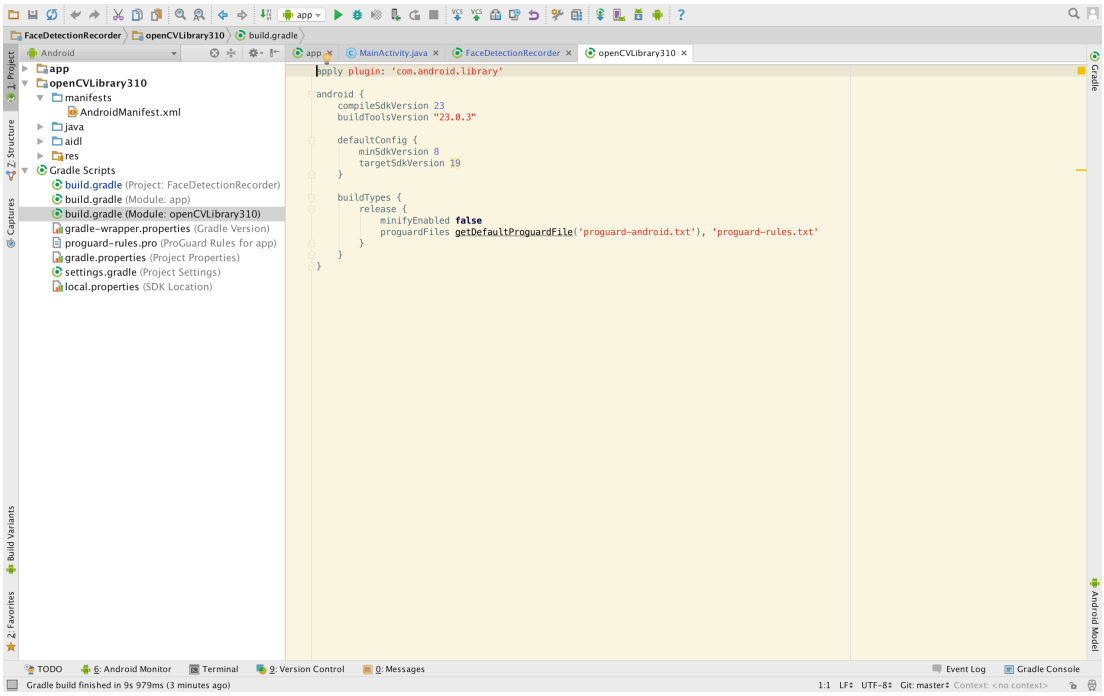
Import Necessary Libraries

Import OpenCV Library

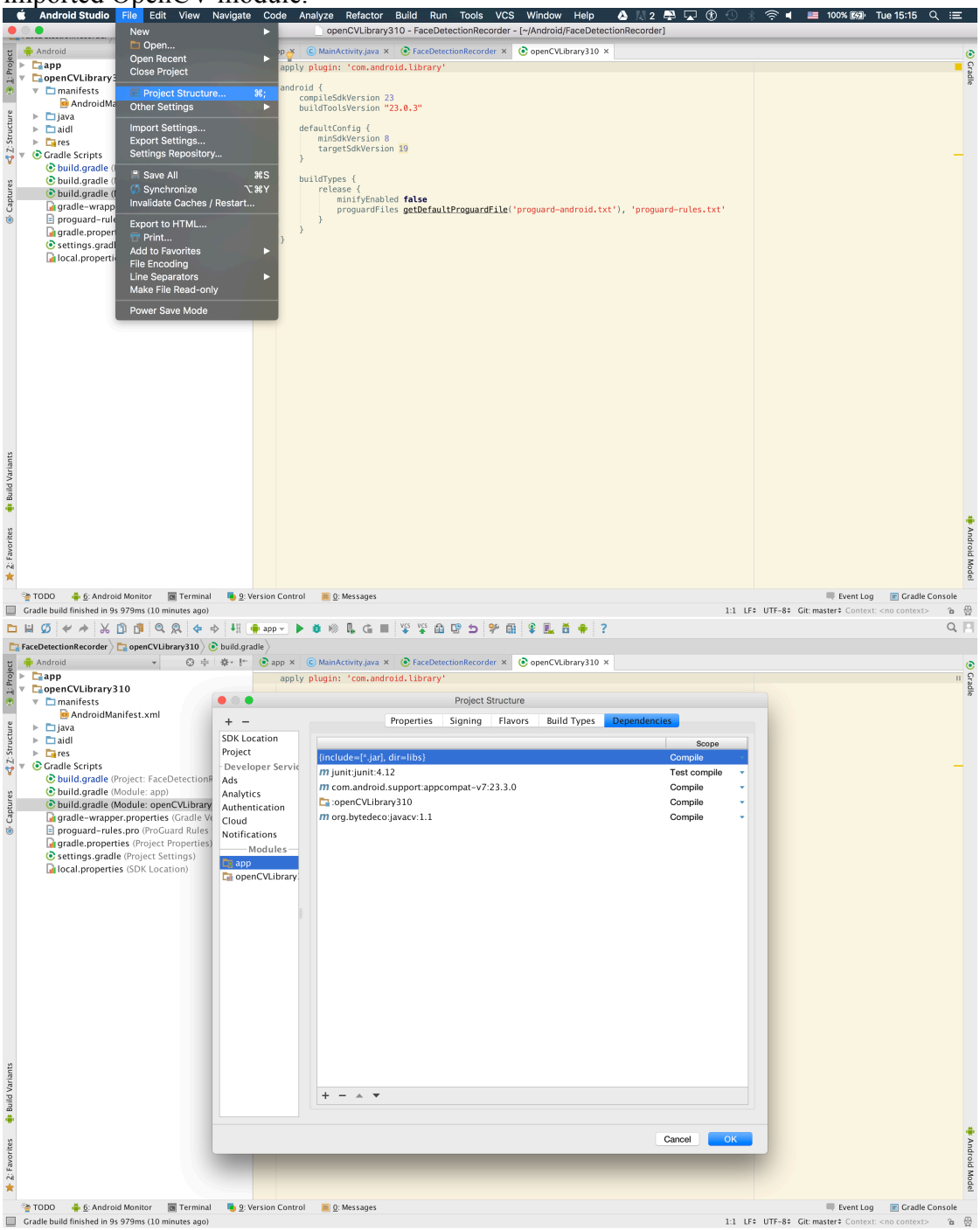
- Download latest OpenCV SDK for Android from OpenCV.org and decompress the zip file. (Download Website: [http://opencv.org/downloads.html)](http://opencv.org/downloads.html)
- **Import OpenCV to Android Studio**, From File -> New -> Import Module, choose sdk/java folder in the unzipped opencv archive.



- **Update build.gradle** under imported OpenCV module to update 4 fields to match your project build.gradle a) compileSdkVersion b) buildToolsVersion c) minSdkVersion and d) targetSdkVersion.



- **Add module dependency** by *File -> Project Structure -> Module: app*, and select the Dependencies tab. Click + icon at bottom, choose *Module Dependency* and select the imported OpenCV module.



- **Copy libs** folder under *sdk/native* to Android Studio under *app/src/main*.
- In Android Studio, **rename *the copied libs* directory to *jniLibs*** and we are done.

Import JavaCV Library

- We can have everything downloaded and installed automatically with:
- Gradle (inside the `build.gradle` file)

```
repositories {
  mavenCentral()
}
dependencies {
  compile group: 'org.bytedeco', name: 'javacv', version: '1.2'
}
```

Additionally, we need to either set the javacpp.platform system property (via the -D command line option) to something like android-arm, or set the javacpp.platform.dependencies one to true to get all the binaries for Android, Linux, Mac OS X, and Windows. **On build systems where this does not work, we need to add the platform-specific artifacts manually.** I did not install the JavaCV Library manually, so if the automatic installation does not work, please infer the official website to do manual installation (https://github.com/bytedeco/javacv).

Application Design & Coding

Simple User Interface Design

Change the layout's background color

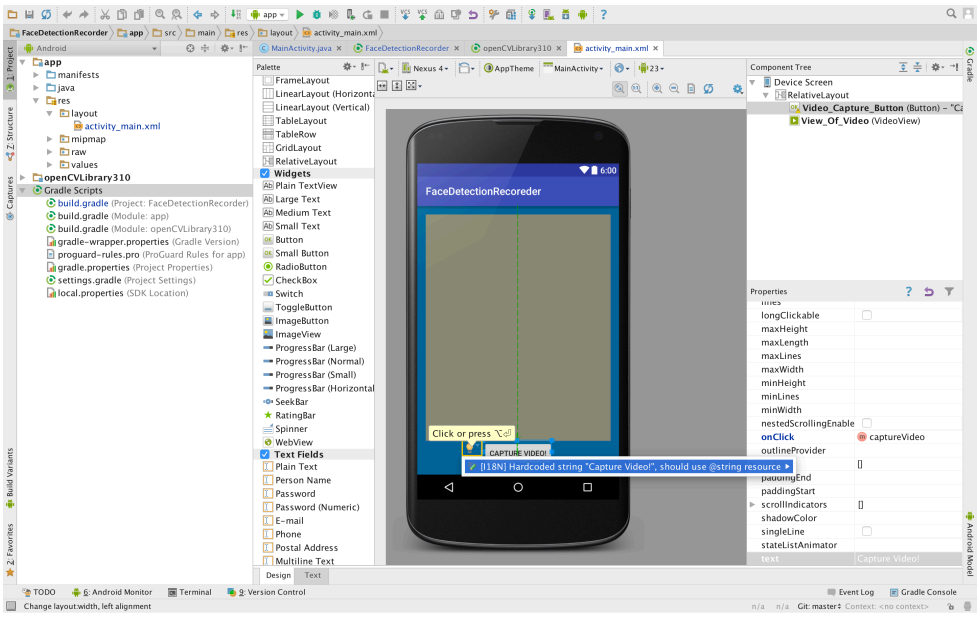
In **app** > **res** > **layout** > **activity_main.xml**, Choose **RelativeLayout** and change **background** color in **properties**. In this project, I changed color to dark blue The color code for dark blue is #006699

Insert a button for recording videos

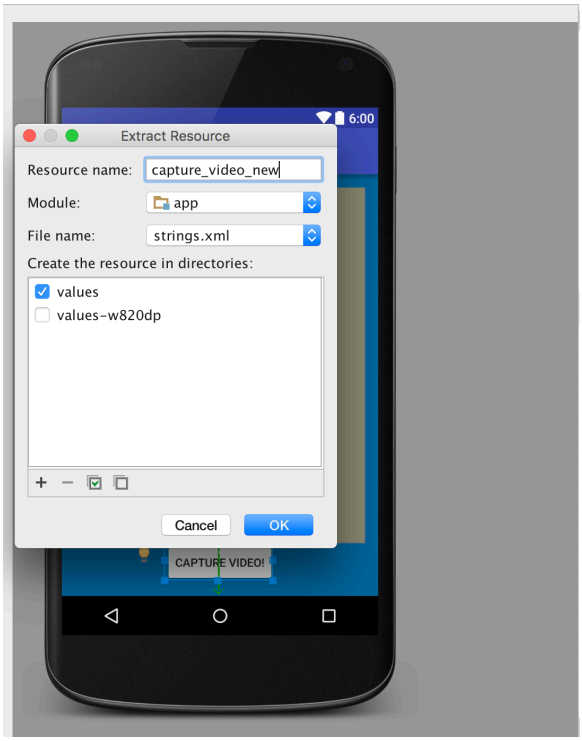
In **app** > **res** > **layout** > **activity_main.xml**, select **Palette** > **Widgets** > **Button**, and place it in the downside of the **RelativeLayout**.

1. In **properties** > **id**, set the ID for this button as **Video_Capture_Button**.
2. In **properties** > **onClick**, set it to **captureVideo**, which means it will call captureVideo method when you click the button.
3. In **properties** > **text**, change the text into **Capture Video!**, which will be the text showing in the button

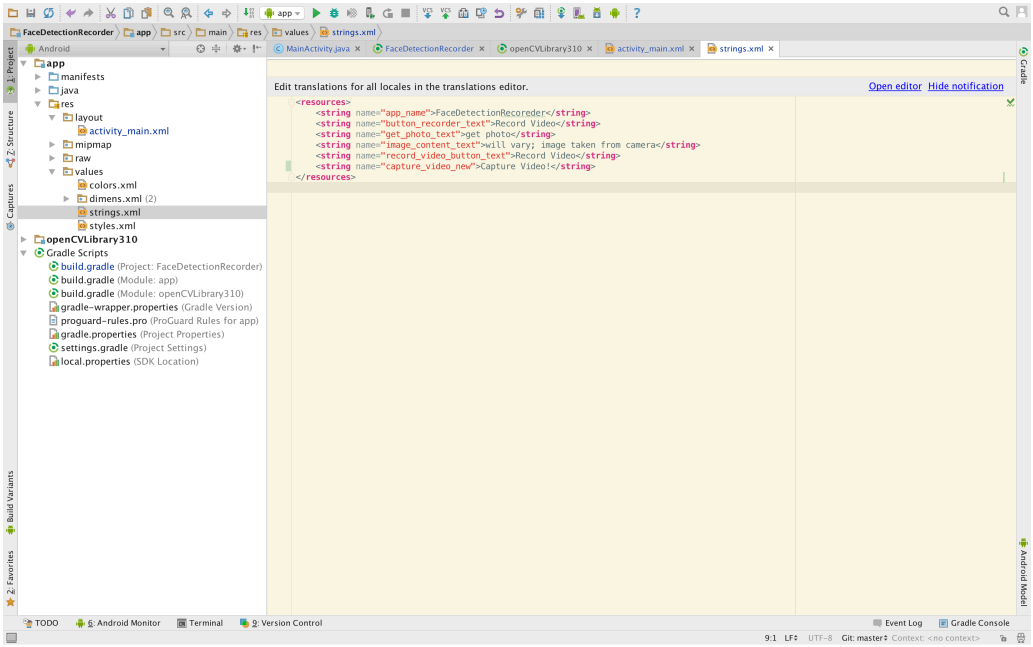
Note: in Android Application Development, you will want to set up @string resource. When you set the text to the button, and move the mouse to the button, you will see a small light ball to ask you to set the text into string variable.



i. Follow the instruction, set the resource name into same or similar name as your text in the button



ii. The text variables will be stored at **app** > **res** > **values** > **strings.xml**.

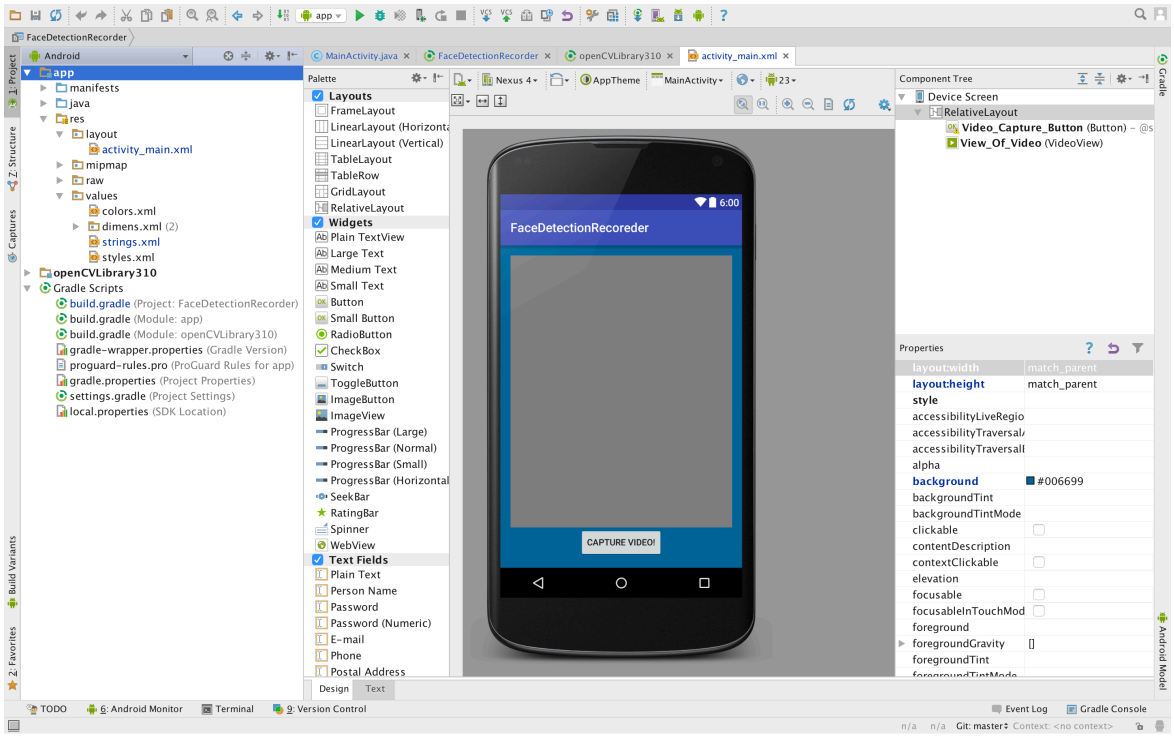


The reason why we wanted to do this is that Android Development System tending to stores all strings in one file, which is more convenient for both developers and system to manage strings.

Insert a VideoView for displaying videos

a. In **app** > **res** > **layout** > **activity_main.xml**, select **Palette** > **Containers** > **VideoView**, and place it in the upside of the **RelativeLayout**.
b. In **properties** > **id**, set the ID for this button as **Video_Capture_Button**.
c.

After the set out, the layout should look like the blew picture, which will also be the interface when app started.



Main Java Codes

The major coding part for this project is in **app** > **java** > **MainActivity.java,** and enabling the Native Camera API, and External Storage in **app** > **manifests** > **AndroidManifest.xml**. In the report, I will explain the **MainActivity** file method by method.

Imports for this Java file:

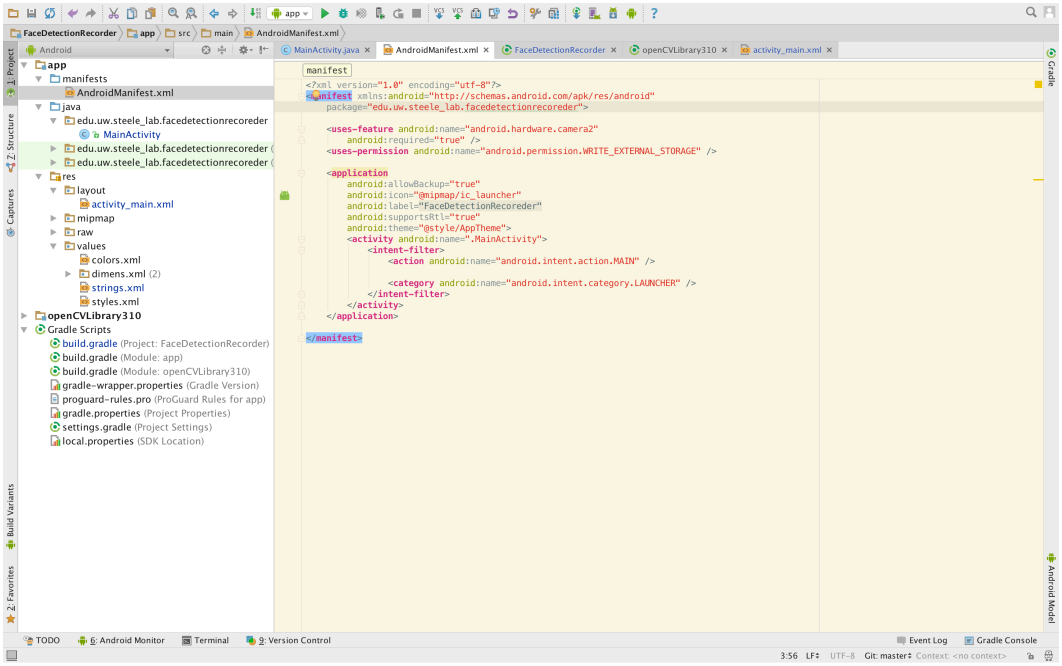
```
package edu.uw.steele_lab.facedetectionrecoreder;

import android.app.Activity;
import android.content.Intent;
import android.graphics.Bitmap;
import android.net.Uri;
import android.os.Environment;
import android.provider.MediaStore;
import android.os.Bundle;
import android.util.Log;
import android.view.View;
import android.widget.MediaController;
import android.widget.Toast;
import android.widget.VideoView;

import org.bytedeco.javacpp.opencv_objdetect;
import org.bytedeco.javacv.AndroidFrameConverter;
import org.bytedeco.javacv.FFmpegFrameGrabber;
import org.bytedeco.javacv.Frame;
import org.bytedeco.javacv.FrameGrabber;
import org.bytedeco.javacv.OpenCVFrameConverter;
import org.opencv.core.CvType;
import org.opencv.core.Mat;

import org.opencv.core.MatOfRect;
import org.opencv.imgproc.Imgproc;
import org.opencv.objdetect.CascadeClassifier;

import java.io.File;
```

Enable Android Native Camera API, and External Storage

a) In **app** > **manifests** > **AndroidManifest.xml,** add these codes between manifest and application. Attention: the position is important for the codes to work.

```
<uses-feature android:name="android.hardware.camera2"
    android:required="true" />
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
```

The first one is allow the app to use native camera, and the second line gives app the permission to save recorded video back to device.



MainActivity > VIDEO_REQUEST_CODE

```
private final int VIDEO_REQUEST_CODE = 100;
```

This field is a private, and immutable int field, which a self-defined value 100. We use this int field to check whether the video is successfully recorded, and whether the video is the correct version we want.

MainActivity > onCreate

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
}
```

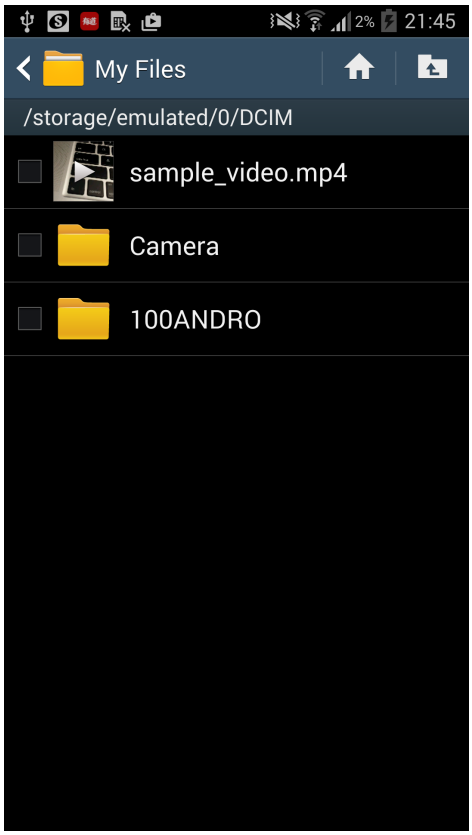
This method is a default method, which created with MainActivity.java. It will be automatically called after the app started.
You can ignore the first line of codes, which just a routine call for all onCreate method. The second line is to set up the interface as we designed in **activity_main.xml**

MainActivity > getFilePath

```
public File getFilePath() {
    // Get the path of the traditional location for pictures and videos when mounting the device as a camera.
    File folder = Environment.getExternalStoragePublicDirectory(Environment.DIRECTORY_DCIM);
    // Check if the folder already existed in the device
    if (folder.exists()) {
        folder.mkdir();
    }
    File videoFile = new File(folder,"sample_video.mp4");
    return videoFile;
}
```

This method provides **File-Formatted** address, where the video is stored into.

- The first line of codes gets the address of **DCIM folder**, and stores it into a **File** variable. **DCIM** is the traditional location for pictures and videos when mounting the device as a camera.
- The if statement here is checking if there is a **DCIM** folder in the device. If not, it will create a **DCIM** folder.
- The rest of codes firstly indicates the name and type for the video file (sample_video.mp4), and the return **File-Formatted** address.



MainActivity > captureVideo

```
public void captureVideo(View view){
    // In this project we use intent to capture the videos
    Intent camera_intent = new Intent(MediaStore.ACTION_VIDEO_CAPTURE);
    // Get the path, and store into video_file
    File video_file = getFilePath();
    // Convert it to uri address
    Uri video_uri = Uri.fromFile(video_file);
    // Add the address into the Intent object
    camera_intent.putExtra(MediaStore.EXTRA_OUTPUT,video_uri);
    // Specify the quality of the video (1 is the maximum)
    camera_intent.putExtra(MediaStore.EXTRA_VIDEO_QUALITY,1);
    // Start the activity, and specify the intent
    startActivityForResult(camera_intent,VIDEO_REQUEST_CODE);
}
```

This method is called we used click **"Capture Video!" button**, and main function of it is to **capture by Native Camera API**, and **save it back to device**.

- In this project, we use Intent as the major tool to capture the video. The parameter, *MediaStore.ACTION_VIDEO_CAPTURE,* indicates the use of video capture for the intent.

i)

> **Intents**
>
> An `Intent` is an object that provides runtime binding between separate components (such as two activities). The `Intent` represents an app's "intent to do something." You can use intents for a wide variety of tasks, but most often they're used to start another activity. For more information, see Intents and Intent Filters.

- The second and third lines of codes is to specify the location we are going to store the video. We use **getFilePath** method, whose details will be explained later, to get the path of the location. Next, we convert it into Uri format, which can be accepted with Intent.

- The forth line adds the address into Intent, and the fifth line specifies the quality of video for the Intent

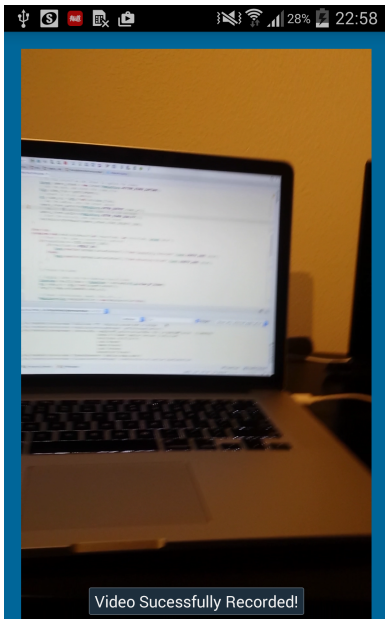
- The last line starts the capture video activities, and passes **VIDEO_REQUEST_CODE** for later confirmation

MainActivity > onActivityResult

```
@Override
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
    // Check if the video is successfully captured
    if(requestCode == VIDEO_REQUEST_CODE){
        if(resultCode == RESULT_OK){
            Toast.makeText(getApplicationContext(),"Video Sucessfully Recorded!",Toast.LENGTH_LONG).show();
        }else{
            Toast.makeText(getApplicationContext(),"Video Recording Failed!",Toast.LENGTH_LONG).show();
        }
    }

    // Process the video

    // Dispaly videos into the VideoView View_Of_Video
    VideoView View_Of_Video = (VideoView) findViewById(R.id.View_Of_Video);
    Uri video_uri = Uri.fromFile(getFilePath());
    View_Of_Video.setVideoURI(video_uri);

    // Player controls(play, pause, stop, etc....)
    MediaController mediaController = new MediaController(this);
    mediaController.setAnchorView(View_Of_Video);
    View_Of_Video.setMediaController(mediaController);

    View_Of_Video.start();
}
```

This method will be called automatically after **startActivityForResult**

- The **if statement** checks if the video is successfully captured, and if the video is captured as required with the **VIDEO_REQUEST_CODE**.
- The video will be processed with the method **ProcessVideo,** which is not yet finised for this project, and the video will be saved back into the device.
- In the third part of the codes, first of all, it creates a reference for VideoView in order to display the processed video. Then, it get the video address by calling **getFilePath** method. Next, display the video into VideoView, which users can watch the video through the app
- The forth part of the codes, it sets up **media controller**. These three lines of codes are the standard routine in Android Development to set up media controller.
- It starts playing video by call **start** method with **the VideoView object**.



MainActivity > ProcessVideo (not yet finished)

```
public File ProcessVideo(File videoFile) {

    // Load the video to videoGrabber
    FrameGrabber videoGrabber = new FFmpegFrameGrabber(videoFile);

    // Set convertToBitmap, which can transfer Fram to Bitmap (also Bitmap to Frame)
    AndroidFrameConverter converterToBitmap = new AndroidFrameConverter();

    // Set convertToMat, which can transfer Frame to Mat (also Mat to Frame)
    OpenCVFrameConverter.ToMat converterToMat = new OpenCVFrameConverter.ToMat();

    opencv_objdetect.CascadeClassifier face_detect;

    try
    {
        videoGrabber.setFormat("mp4");
        videoGrabber.start();
    }catch (org.bytedeco.javacv.FrameGrabber.Exception e){
        Log.e("javacv", "Failed to start grabber" + e);
        return new File("null");
    }

    Frame vFrame = null;

   do{
       try{
           // Grab an image Frame from the video file
          vFrame = videoGrabber.grabFrame();
           if(vFrame != null){
               //Necessary for making the native detector happy
               MatOfRect output = new MatOfRect();

               // Perform a shallow copy to represent Frame as a Bitmap
               Bitmap input = converterToBitmap.convert(vFrame);
               // Convert our bitmap to a Mat so the detector can use it
               Mat inputMat = new Mat(input.getWidth(), input.getHeight(), CvType.CV_8UC1);
               org.opencv.android.Utils.bitmapToMat(input, inputMat);
               Imgproc.cvtColor(inputMat, inputMat, Imgproc.COLOR_RGB2GRAY);

               // Actually do the detection

           }
       } catch (org.bytedeco.javacv.FrameGrabber.Exception e){
           Log.e("javacv", "video grabFrame failed: "+ e);
       }
   } while(vFrame != null);

   try
   {
       videoGrabber.stop();
   }catch (org.bytedeco.javacv.FrameGrabber.Exception e)
   {
       Log.e("javacv", "failed to stop video grabber", e);
       return new File("null");
   }
   return null;
}
```

In the beginning of the method, we create several object references:

- FrameGrabber: it contains methods that can convert **mp4 video** file to **Frame**
- AndroidFrameConverter: it contains methods that can convert **Bitmap** to **Frame**, or **Frame** to **Bitmap**. (Bitmap is a standard format of image in Android Stuido)

- OpenCVFrameConverter.ToMat: it contains methods that can convert **Frame** to **Mat** or **Mat** to **Frame**. Note: if you are not familiar with Mat structure, please read through the following website: http://docs.opencv.org/2.4/doc/tutorials/core/mat_the_basic_image_container/mat_the_basic_image_container.html

  This method processes the video as following steps:

  1) Grab three channel IplImage frame from video with the FrameGrabber
  2) Convert grabbed three channel frame to four channel IplImage frame
  3) Convert four channel frame to Android Bitmap image
  4) Convert Android Bitmap image to OpenCV Mat frame
  5) Process OpenCV Mat frame with face detection functions
  6) Convert Processed Mat frame to Android Bitmap image
  7) Convert Android Bitmap image to four channel IplImage frame
  8) Convert four channel IplImage frame to three channel IplImage
  9) Store three channel IplImage as frame in the video with the FrameRecorder

  The first **try and catch statement** is to check if the video is **mp4 format**, and can be processed **frame by frame**.

  a) Thanks to **FrameGrabber** we can directly convert **mp4 file** to **four channel frame**, **grabFrame** method does the job in the **do while loop**, which finishes the first two steps.
  b) Use **converterToBitmap.convert** method converting **Frame** we got from previous steps to **Bitmap** for step 3.
  c) Create a **Mat** structure object, and use **bitmapToMat** method to convert **Bitmap** we got from previous steps to **Mat** for step 4.

Spring 2016 by Bingbing Huang

For fellow students:

1) Get familiar with OpenCV Mat structure (http://docs.opencv.org/2.4/doc/tutorials/core/mat_the_basic_image_container/mat_the_basic_image_container.html)
2) Get familiar with Haar feature-based cascade classifiers (http://docs.opencv.org/3.1.0/d7/d8b/tutorial_py_face_detection.html#gsc.tab=0 http://docs.opencv.org/2.4/doc/tutorials/objdetect/cascade_classifier/cascade_classifier.html)
3) Go through the codes in OpenCV Face Dectection Samples (http://opencvexamples.blogspot.com/2013/10/face-detection-using-haar-cascade.html)
4) Do research in how to apply face detection functions in OpenCV Mat Structure