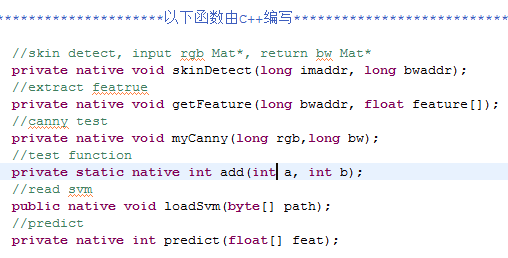
# 手势识别程序移植总结

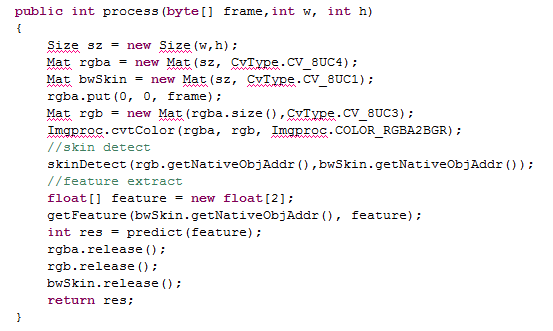
在本项目中，手势识别程序主要是用C++编写的，然后使用Jni技术打包成so文件，在JAVA中调用，打包过程在“NDK编译C++文件步骤”一文中已详细说明，简单描述过程如下：

## 在JAVA中写好接口类文件

在本项目中，该文件使用Java实现了主要处理逻辑，对其中需要调用的主要方法编写相应的函数接口：



主要处理逻辑：



将整个程序分解成主程序逻辑和多个子函数，将将子函数用C++实现的形式，比将整个程序用C++实现然后打包的形式优点在于，便于调试检查。注意这些导出的子函数需要在返回类型前加native关键字。

## 用C++实现导出函数

对导出函数用C++实现，需要先将函数名改写，可以使用JAVAC和JAVAH命令生成，也可以自己改写。改写的规则是：

JNIEXPORT **void** JNICALL Java\_包名\_类名 \_函数名

(JNIEnv \* jenv, jclass, 参数1，参数2...);

在JAVA中的参数基本类型需要改写，对应如下：

|  |  |  |
| --- | --- | --- |
| **Java类型** | **C++类型** | **字节** |
| boolean | jboolean | 8 |
| byte | jbyte | 8 |
| char | jchar | 16 |
| short | jshort | 16 |
| int | jint | 32 |
| long | jlong | 64 |
| float | jfloat | 32 |
| double | jdouble | 64 |
| void | void | -- |

例如在包名为com.ermu.sixshot.colorful.detector.jni下的类名为JniGestureDetector下的add函数，改写前：

private static native int add(int a, int b);

改写后：

JNIEXPORT jint JNICALL

Java\_com\_ermu\_sixshot\_colorful\_detector\_jni\_JniGestureDetector\_add

(JNIEnv \* jenv, jclass, jint a, jint b);

另外，对于数组的传递，Java中float[] arr不对简单的改写成C++的jfloat arr[]， float数组类型对应 jfloatArray类型，其它类型的数组也有类似相应的对应类型。在改写函数中，由于要使用我们常用的C++数组，因此还应将jfloatArry类型转换为C++的float数组，可以使用下面的函数：

env->GetFloatArrayElements

其中evn为传入的JNIEnv对象，例如，

**int** l = env->GetArrayLength(arr);

**float** \*featarr = (**float**\*)env->GetFloatArrayElements(arr, 0);

这时就可以将featarr当作一般的float数组处理了，但是如果要改变数组中的数据，并返回，最后需要调用以下函数：

env->SetFloatArrayRegion(arr, 0, l, featarr);

例如，以上在com.ermu.sixshot.colorful.detector.jni下的类名为JniGestureDetector下的getFeature函数，改写前：

private native void getFeature(long bwaddr, float feature[]);

改写后：

JNIEXPORT void JNICALL

Java\_com\_ermu\_sixshot\_colorful\_detector\_jni\_JniGestureDetector\_getFeature(JNIEnv \* env, jclass, jlong bw, jfloatArray arr);

具体实现：

JNIEXPORT void JNICALL

Java\_com\_ermu\_sixshot\_colorful\_detector\_jni\_JniGestureDetector\_getFeature(JNIEnv \* env, jclass, jlong bw, jfloatArray arr)

{

Mat& bwMat = \*(Mat\*)bw;

int l = env->GetArrayLength(arr);

float \*featarr = (float\*)env->GetFloatArrayElements(arr, 0);

vector<float> feat = hgr.getFeatrue(bwMat);

for (int i = 0; i < l; i++)

{

featarr[i] = feat[i];

}

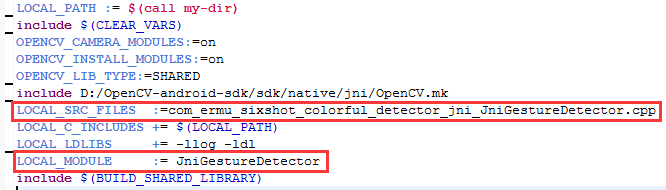
env->SetFloatArrayRegion(arr, 0, l, featarr);

}

以上有一点需要说明Android版的OpenCV的Mat类型可以调用getNativeObjAddr()来获得其内存地址，相当于C++的指针，是一个long类型，因此，可以使用指针来传递参数，如上面的函数那样。

## 编写mk文件

内容如下：



红框标识了需要打包的CPP文件和生成的模块名，编译成功后会生成libJniGestureDetector.so文件，在libs目录下，名字形式为”lib”+模块名(JniGestureDetector)。

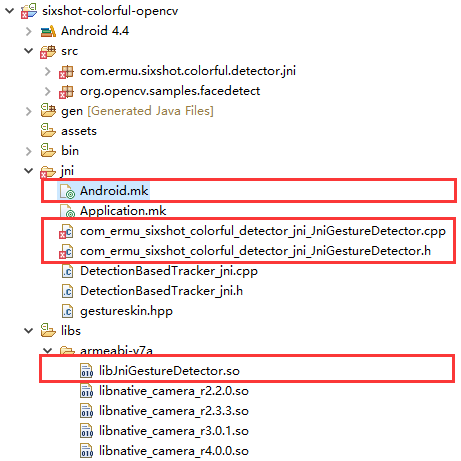
如果工程需要包含多个cpp文件，可以这样：

LOCAL\_SRC\_FILES := test1.cpp \

test2.cpp \

test3.cpp

mk文件和CPP文件、H 文件放在jni目录下。



## 在Java文件中导入生成的库文件

在第1步编写的接口类中，导出函数在第2步中被实现，而第2步的cpp文件已在第3步中生成so文件，因此，应在接口类中加载so文件，加载方法，在类中加入以下语句：

**static**{

System.*loadLibrary*("JniGestureDetector");

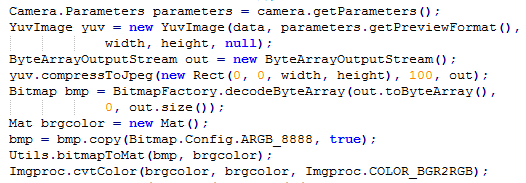
}

虽然第3步中生成的so文件是有lib前缀的，但是在这里加载不需要该前缀。

## 需要注意的问题

### 关于图像的转换与保存

由于Android系统上的Camera对象预览时，在public void onPreviewFrame(byte[] data, Camera camera)函数中，data数据是一个YUV格式的数据缓存，大小为1.5Height×Width字节，要转换为我们一般常见的RGB或BGR格式进行处理。转换方法如下：



为了查看中间图像处理结果，可以将图像保存到手机中进行查看，方法是调用OpenCV的有关函数：



### 支持向量机训练结果的读取

经反复实验Android版的OpenCV库的CvSVM的load函数，加载XML文件失败，可以采用以下方案解决：将CvSVM对象封闭在C++代码中，使用Jni技术，将读取训练结果的过程在C++代码中完成，例如：

JNIEXPORT **void** JNICALL Java\_com\_ermu\_sixshot\_colorful\_detector\_jni\_JniGestureDetector\_loadSvm

(JNIEnv \*env, jclass obj, jbyteArray data)

{

**int** l = env->GetArrayLength(data);

**char**\* pData = (**char**\*)env->GetByteArrayElements(data, 0);

hgr.read\_svms(pData);

}

对象hgr的read\_svms的实现如下：

**void** **HandGestureRecognizer::read\_svms**(**char**\* path)

{

main\_svm.load(path);

}

在Java类文件中申明：

public native void loadSvm(byte[] path);

因为训练文件的加载是用C++完成的，因此需要注意，传入路径中的斜杠”\”必须写成双斜杠”\\”，如：

String svmPath = [\\storage\\emulated\\0\\main\_svm.xml](file:///\\storage\\emulated\\0\\main_svm.xml)

或使用以下形式：

String strFace = "/storage/emulated/0/genderrecognizer\_lbph.xml";

另外，由于训练结果XML文件放置在res/raw目录下，在程序开始时，把XML文件拷贝到指定手机目录下，例如手机的根目录，这样可以避免人为的把文件从PC上拷贝到手机目录。实现的代码如下：

**public** **boolean** loadXMLFile()

{

**boolean** bRes = **true**;

**try**{

// main\_svm.xml in res/raw dir

InputStream is = getResources().openRawResource(R.raw.*main\_svm*);

String filePath=Environment.*getExternalStorageDirectory*()+ File.*separator* + "main\_svm.xml";

File mSvmFile = **new** File(filePath);

**if** (!mSvmFile.exists()) {

mSvmFile.createNewFile();

}

FileOutputStream os = **new** FileOutputStream(mSvmFile);

**byte**[] buffer = **new** **byte**[8092];

**int** bytesRead = 0;

**while**((bytesRead = is.read(buffer))!=-1)

{

os.write(buffer,0,bytesRead);

}

is.close();

os.close();

String svmPath = "//storage//emulated//0//main\_svm.xml";

**byte**[] path=svmPath.getBytes();

gestureDetector.loadSvm(path);

}

**catch**(Exception e)

{

Log.*i*("GESTURE","Not found xml file!");

bRes=**false**;

}

**return** bRes;

}

### 在so文件中打印日志

为便于调试，需要在Android程序中打印so文件中的运行中间结果，可以采取以下方法：

在C++头文件中添加以下代码：

**#include** <android/log.h>

**#define** LOG\_TAG "show infomation"

**#define** LOGW(a ) \_\_android\_log\_write(ANDROID\_LOG\_WARN,LOG\_TAG,a)

在需要打印日志的地方：

LOGW("This is a Log test!");