Journal of Knowledge Information Technology and Systems(JKITS), Vol. 10, No. 3, pp. 355~365, June 2015

Journal of Knowledge Information Technology and Systems

ISSN 1975-7700

http://www.kkits.or.kr

The Method to use Native Code(C/C++) in Android OS and Its Applications

Sangmin Ahn, Keonyong Lee, Woonhyung Jung, Nakju Lett Doh*

School of Electrical Engineering, Korea University

ABSTRACT

In general an Android application based on Java operates on DALVIK virtual machine. It is safe that an application operates on a virtual machine which means it is platform-independent. However, to be operated on a virtual machine, it takes some time for the program codes to be converted. This is slower than an operation of native codes like C because the codes are translated to machine language, which means it is platform-dependent. Also, it is restricted to access a hardware directly on a virtual machine. Many programs have been developed based on C/C++. We might save time and cost for development if we can reuse these programs without re-developing for other programming languages. On this paper, we explain how to operate a code based on C/C++ using JNI (Java Native Interface). Starting with how to call C/C++ native functions from Java classes, we introduce how codes developed with two different programming languages communicate each other. Also, we introduce a specific case using native codes on Android as an example.

KEYWORDS: Native code, Java, Android, JNI, SLAM

ARTICLE INFO: Received 6 May 2015, Revised 12 June 2015, Accepted 12 June 2015.

1. 서 론

최근 스마트폰이 널리 보급되면서 많은 작업들이 손바닥 안에서 이루어지는 이른바 모바일 시대

*Corresponding author is with the School of Electrical Engineering, Korea University, Anam dong, Seong-buk Gu Seoul, 136-713, KOREA.
E-mail address: nakju@korea.ac.kr

가 시작되었다. 단순히 전화의 기능을 넘어서 인터넷에 연결되어 무한한 정보의 바다에 접근할 수 있게 되었고, 카메라, 적외선 센서, 자이로 센서와 같은 다양한 센서들이 이미 스마트폰에 탑재되어 있으며, 최근 출시된 아이폰에는 기압계도 탑재되어 있다. 기술이 발전함에 따라 점점 더 다양한 센서가 스마트폰에 탑재되어 다양한 기능을 하나의 기기로 이용할 수 있게 될 전망이다. 이에 따라 최근의 스마트폰은 단순히 사용자의 편의를 위해 어플리케이션을 구동하기 위한 용도를 넘어 연구용으로 사용되고 있다[1-6]. 그 중에서 로보틱스의 한 연구 분야인 로봇의 위치 인식 알고리즘(SLAM)[7]의 경우, 지금까지는 주로 네이티브 코드를 이용하여 구현된 후, 고사양의 컴퓨터에서 그 알고리즘의 성능이 분석 되어져 왔다. 하지만 최근에는, 스마트폰의 성능이 급격히 향상되면서 노트북이 아닌 스마트폰을 이용한 SLAM알고리즘 연구도 늘어나고 있다. 스마트폰에 부착된 카메라에서 얻어진 영상 정보를 이용한 SLAM알고리즘의 성능이 좋아지고 있고[1], 스마트폰을 이용하여 3차원 RGBD정보도 얻을 수 있게 됨으로써[2] 이러한 연구는 더 활발해 질 것으로 보인다.

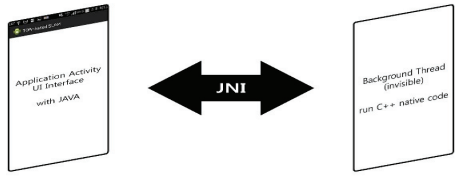


그림 1. JNI를 이용한 자바-네이티브 코드간 연동 구조
Figure 1. Linked structure between Java and Native codes using JNI

스마트폰, 특히 안드로이드 기반 스마트폰에서의 어플리케이션 개발은 자바 언어에 기반한다. 자바 언어는 객체 지향 프로그래밍을 지원하며 가상 머신(안드로이드의 달빅[8] 가상 머신)을 통해 코드를 구동하는 점이 특징이다. 가상 머신을 이용하여 코드를 구동하는 방식은 플랫폼에 독립적이라는 장점이 있고, 또한 하드웨어를 직접적으로 제어하지 않기 때문에 안정성이 뛰어나다. 하지만 플랫폼에 종속적인 C/C++에 비해 속도가 느리고[9-12] 하드웨어 수준의 접근과 제어가 용이하지 않다는 단점이 있다. 포인터를 이용하여 직접적으로 메모리에 접근하여 쉽게 변수를 다룰 수 있는 C/C++와는 달리 자바에서는 메모리 참조가 제한된다는 점도 또 하나의 단점이다. 스마트폰에 많은 센서들이 내장되고 기능이 다양해짐에 따라 C/C++와 같이 하드웨어를 자유자재로 다룰 수 있는 언어의 필요성이 증대되었고 구글에서도 <그림 1>과 같이 자바 네이티브 인터페이스(이하 JNI)를 선보이며 자바와 C/C++를 연동할 수 있도록 발판을 마련했는데, 이는 자바의 구조적 한계성을 인식한 것으로 보인다.

안드로이드 NDK(Native Development Kit)에서 JNI를 이용한 네이티브 코드의 활용을 권장하지 않고 사용상의 제약이 존재할지라도[13] 더 많은 기능을 효율적으로 활용할 수 있는 어플리케이션 개발을 위해서는 C/C++ 네이티브 코드의 사용이 필수 불가결하다. 많은 프로그램들이 이미 C/C++기반으로 설계되어 있고, 이러한 프로그램들을 안드로이드 어플리케이션에서 재사용할 수 있다면 같은 프로그램을 자바로 재개발하기 위해 소요되는 비용을 크게 절감할 수 있을 것이다[6,10,14]. 그러나 안드로이드에서 C/C++ 코드를 활용하는 방법을 체계적으로 정리해 놓은 문서는 없었다. 본 논문에서는 그 변환 과정을 체계적으로 정리하였고, 실제의 활용을 보임으로써 이 변환 과정을 검증하였다 또한 이미 구현되어 있는 다양한 C/C++ 코드들이 실제로 스마트폰에서 활용될 수 있음을 실험 결과를 통해서 증명한다.

2. 네이티브 코드 연동을 위해 필요한 프로그램

2.1 JDK(Java Development Kit)

Java Development Kit(이하 JDK)은 Oracle Corporation에서 개발 및 배포하는 소프트웨어 개발 도구이다. 안드로이드에서 사용하는 달빅 가상 머신과 자바에서 사용하는 자바 가상 머신은 구동 방식에서 차이가 있고[15] 따라서 안드로이드 API와 자바 API 사이에도 다소 차이가 있지만 기본적으로 안드로이드 어플리케이션은 자바 클래스를 이용하여 개발된다. Windows XP의 지원이 공식적으로 중단됨에 따라 JDK도 Windows XP에서의 지원을 공식 중단했고, 이후 배포되는 JDK는 일반적인 방법으로 Windows XP에서 설치가 되지 않으며 수동으로 패키지를 추출하는 방식으로 설치할 수 있다. JDK를 설치한 후에 편리한 통합 개발 환경 구성을 위하여 Windows의 환경 변수에 JDK의 설치 경로를 등록해준다. 본 논문은 JDK 8버전을 기준으로 한다.

2.2 안드로이드 SDK (Software Development Kit)

안드로이드 소프트웨어 개발 도구(이하 SDK)는 안드로이드 어플리케이션을 만들고 디버깅하기 위해 필요한 라이브러리와 도구들을 모아둔 것이다. 각 버전의 안드로이드 플랫폼을 위한 도구들과 가상 단말기 환경을 구축할 수 있는 에뮬레이터를 제공하며 통합 개발 환경인 Eclipse에 포함되어 배포되기도 한다. 마찬가지로 설치 후 환경 변수에 설치 경로를 등록해준다. 본 논문은 안드로이드 SDK r20.0.3버전을 기준으로 한다.

2.3 안드로이드 NDK (Native Development Kit)

안드로이드 네이티브 개발 도구(이하 NDK)는 C/C++와 같은 네이티브 코드 언어를 크로스 컴파일하여 안드로이드에 적용하기 위해 필요한 도구들을 모아둔 것이다. 네이티브 코드의 사용은 어플리케이션의 복잡도를 증가시키므로 필요한 경우에만 사용할 것을 권장한다. 마찬가지로 설치 후 환경 변수에 설치 경로를 등록해준다. 본 논문은 안드로이드 NDK r9d버전을 기준으로 한다.

2.4 통합 개발 환경: Eclipse

Eclipse(이하 이클립스)는 이클립스 재단에서 개발하여 배포하는 프로그래밍 통합 개발 환경으로 자바를 비롯하여 다양한 프로그래밍 언어를 지원한다. 이클립스는 자바를 기반으로 제작되어 있으며 설치 과정에서 JDK를 설치한 경로를 입력하여 연동한다. 구글에서 제공하는 안드로이드 개발 패키지 플러그인을 차례대로 설치하고 이클립스를 재시작하여 SDK를 설치한 경로를 입력해주면 기본적인 안드로이드 개발 환경을 구축할 수 있다. 본 논문은 이클립스 Kepler버전을 기준으로 한다.

3. 안드로이드 C/C++ 하이브리드 프로젝트 설정

다음 순서를 따라 기존의 안드로이드 프로젝트를 C/C++ 하이브리드 프로젝트로 전환한다.

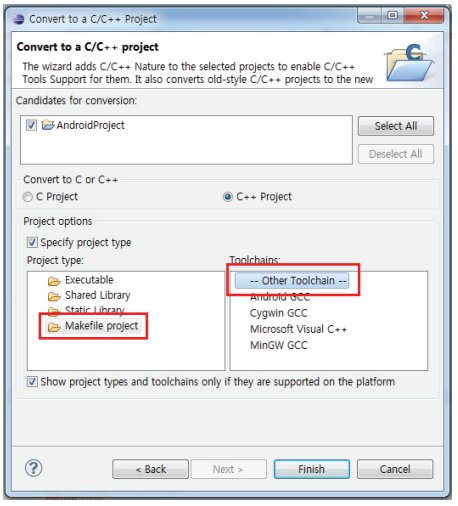


그림 3. C/C++ 프로젝트 속성 설정
Figure 2. C/C++ Project property setting

1. 패키지 탐색창에서 프로젝트를 우클릭하여 새로 만들기 창을 띄우고 C/C++ 프로젝트로 전환한다. <그림 2>처럼 전환 설정 창에서 메이크파일 프로젝트와 다른 툴체인을 선택하고 설정을 완료한다.

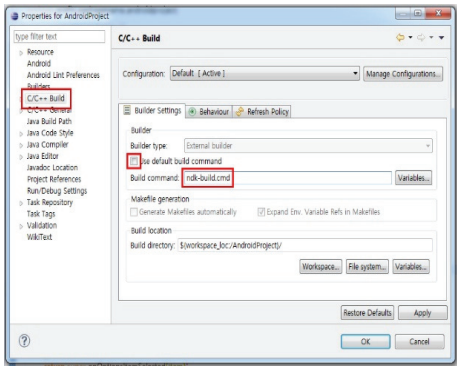


그림 4. 프로젝트 빌드 설정
Figure 3. Project build setting

2. <그림 3>처럼 패키지 탐색창에서 프로젝트를 우클릭하여 설정 창을 띄워 C/C++ 빌드 항목에서 기본 빌드 명령의 체크를 해제하고 ndk-build.cmd를 입력한다.

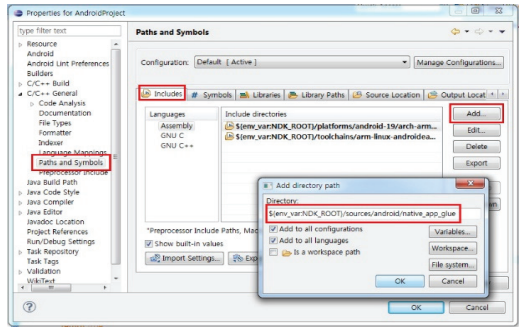


그림 5. 빌드 시 필요한 라이브러리 폴더를 연동
Figure 4. Linking library folders which are necessary for build

3. <그림 4>에서와 같이 C/C++ 일반 항목 하위의 경로와 기호 항목에서 인클루드 탭을 선택한다. 추가 버튼을 누르고 모든 설정과 모든 언어에 적용할 수 있도록 체크한다. 윈도우의 환경 변수에 등록해 두었던 NDK의 경로는 ${env_var: 환경 변수명}과 같은 방법으로 사용할 수 있다. 물론 환경 변수를 이용한 경로 참조가 아닌 폴더의 절대 경로를 직접 입력해도 무방하다. 여기에서는 NDK를 설치한 폴더 경로에 대한 환경 변수를 NDK_ROOT로 등록해 두었으므로 폴더 경로는 다음과 같이 입력할 수 있다.

${env_var:NDK_ROOT}/platforms/android-19/arch-arm/usr/include

${env_var:NDK_ROOT}/toolchains/arm-linux-androideabi-4.8/prebuilt/windows-x86_64/lib/gcc/arm-linux-androideabi/4.8/include

${env_var:NDK_ROOT}/sources/android/native_app_glue

여기에서는 안드로이드 API 19 버전(킷캣)과 64비트 윈도우즈 7을 사용했다. 이 때, 안드로이드 패키지의 버전과 윈도우즈에 따라 중간 파일 경로가 달라질 수 있음에 유의한다. 위의 예시 경로에서 밑줄 부분은 개발 환경에 따라 달라질 가능성이 있는 폴더 이름을 나타낸다.

4. 패키지 탐색창에서 프로젝트를 우클릭하여 jni라는 이름으로 새로운 폴더를 생성한다. 패키지 탐색창에서 생성된 jni 폴더를 우클릭하여 Android.mk 파일과 Application.mk 파일을 생성한다.

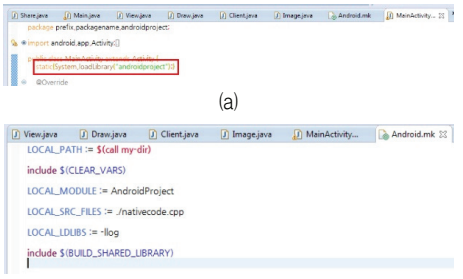


그림 5. 네이티브 코드를 동적 라이브러리로 불러오기 위한 과정
Figure 5. Procedure for loading native codes to dynamic library

5. <그림 5 (a)>와 같이 메인 액티비티의 자바 클래스 내에 static{System.loadLibrary("프로젝트 이름");}과 같이 입력한다.

f. <그림 5 (b)>의 Android.mk 파일의 내용은 다음과 같다.

```
LOCAL_PATH := $(call my-dir)
include $(CLEAR_VARS)
LOCAL_MODULE := AndroidProject
LOCAL_SRC_FILES := ./nativecode.cpp
LOCAL_LDLIBS := -llog
include $(BUILD_SHARED_LIBRARY)
```

LOCAL_MODULE은 프로젝트 이름과 같도록 입력하고, LOCAL_SRC_FILES는 사용할 네이티브 코드 소스 파일들의 경로를 모두 입력한다.

Application.mk 파일의 내용은 다음과 같다.

```
APP_STL := gnustl_static
```

4. 네이티브 코드 연동 어플리케이션의 구조

4.1 UI 스레드와 백그라운드 스레드

안드로이드의 사용자 인터페이스(이하 UI)는 일반적으로 어플리케이션의 액티비티를 통하여 이루어진다. 액티비티는 화면 출력과 사용자 입력 등을 처리해야 하며, 이러한 연산이 이루어지는 스레드를 UI 스레드라고 한다. 액티비티는 언제든 사용자의 입력을 처리하고 변경된 결과를 화면상에 출력할 수 있도록 빠른 반응 속도가 요구되는데, 이러한 이유 때문에 UI 스레드에서 시간이 오래 걸리는 알고리즘을 수행하는 경우 어플리케이션의 응답이 없는 것으로 간주하여 달빅 가상 머신에 의해 강제로 스레드를 종료하는 경우가 발생한다. 또한, UI 스레드가 아닌 스레드에서 입출력을 처리하려고 하면 달빅 가상 머신에서 에러를 발생시키므로 입출력만을 담당하는 UI 스레드와 연산을 담당하는 백그라운드 스레드를 분리할 필요가 있다.

4.2 자바에서 C/C++ 함수를 호출

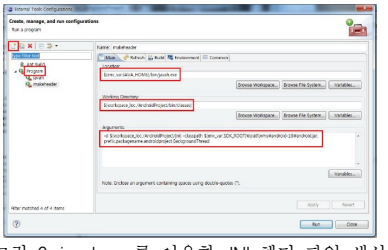


그림 6. javah.exe를 이용한 JNI 헤더 파일 생성
Figure 6. Making JNI header file with javah.exe

자바에서 네이티브 코드에 정의된 함수를 사용하기 위해 네이티브 함수를 호출하고자 하는 자바 클래스 내부에 native 명령문으로 선언한다. <그림 6>처럼 실행 메뉴의 외부 도구 항목에서 외부 도구 설정을 선택한다. 새로운 프로그램을 추가하여 다음과 같이 입력한다.

Location: ${env_var:JAVA_HOME}/bin/javah.exe

Working Directory: ${workspace_loc:/프로젝트 이름/bin/classes}

Arguments: -d ${workspace_loc:/프로젝트 이름/jni} -classpath ${env_var:SDK_ROOT}\platforms\android-19\android.jar; 접두어.패키지 이름.프로젝트 이름.클래스 이름

여기에서도 사용하는 안드로이드 패키지 API 버전에 따라 중간 파일 경로(예시 경로의 밑줄 부분)가 달라질 수 있음에 유의한다. 전체 패키지 이름과 자바 클래스 이름을 정확하게 입력하고 적용 버튼을 누른 후 실행한다. 패키지 탐색창을 새로고침하면 jni 폴더 하위에 새롭게 헤더 파일이 생성된 것을 확인할 수 있다. 네이티브 코드 소스 파일에서 이 헤더 파일을 포함시키고 함수를 정의한다.

4.3 C/C++에서 자바 함수를 호출

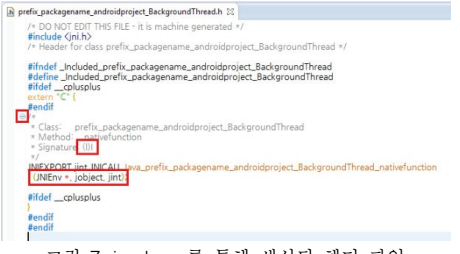


그림 7. javah.exe를 통해 생성된 헤더 파일
Figure 7. Header file made from javah.exe

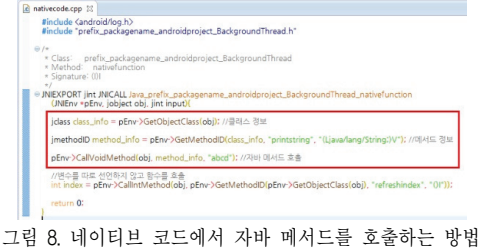


그림 8. 네이티브 코드에서 자바 메서드를 호출하는 방법
Figure 8. Way to call java method in native code

<그림 7>에서 보이듯 자바 클래스에서 네이티브 함수를 호출할 때 자동적으로 JNI 포인터와 객체 인자를 포함하여 함수를 호출하며, 이는 이전 단계에서 생성된 헤더 파일을 살펴보면 확인할 수 있다. 자동 생성되는 두 개의 인자는 네이티브 함수를 호출한 자바 클래스에 대한 정보를 가지고 있다. JNI 포인터 ->GetObjectClass(객체);와 같이 자바 클래스의 정보를 얻을 수 있고, JNI 포인터 ->GetMethodID(클래스, 메서드 이름, 메서드 서명);과 같이 메서드(자바 함수) 정보를 얻을 수 있다. 메서드 서명은 인자 형식과 리턴 타입에 따라 정해져 있으며, 올바른 메서드를 찾아 호출하기 위해 필요하다. 네이티브 메서드의 서명은 헤더 파일의 주석 부분에서 확인할 수 있다. 네이티브 메서드가 아닌 일반적인 자바 메서드도 서명 형식은 동일하므로 이를 참고하여 호출하려는 자바 메서드의 서명을 알 수 있다. <그림 8>처럼 최종적으로 JNI 포인터->Call리턴 타입Method(객체, 메서드, 입력 인자);와 같이 자바 클래스에 정의된 메서드를 호출할 수 있다.

4.4 변수 처리와 전달

```
public class setDoubleArray extends Thread{
   setDoubleArray(byte[] receivedData){
      byte[] bTemp = new byte[Share.sizeDataType];
      for(int i = 0; i < numberofData; i++){
         for(int j = 0; j < Share.sizeDataType; j++)
            bTemp[j] = receivedData[j + i * Share.sizeDataType];

         tempDoubleArray[i] = rtnDoubleFrom8bitReverse(bTemp);
      }
   }

   private double rtnDoubleFrom8bitReverse(byte[] bANumber){
      long tempL = 0;
      for(int j = 0; j < 8; j++){
         tempL += (long)((bANumber[j] & 0xFFL) << (j * 8));
      }
      return Double.longBitsToDouble(tempL);
   }

   @Override
   public void run(){
      hdlDraw.sendEmptyMessage(5);
   }
}
```

그림 9. 통신으로 전송 받은 바이트 순서 정렬
Figure 9. Reordering transmitted bytes

C/C++와 자바에서는 모두 하위 바이트부터 기록하는 리틀 엔디언 방식을 사용하지만, 패킷을 이용한 네트워크 통신에서는 상위 바이트부터 기록하는

빅 엔디언 방식을 사용한다. memset등의 C 함수를 사용하여 일반적인 소켓 통신으로 바이트 배열을 송수신하는 경우 <그림 9>처럼 수신한 바이트 배열을 자료형 크기에 맞춰 역순으로 변환해야 한다.

그림 10. 네이티브 코드에서 자바 배열을 불러오는 방법
Figure 10. Way to call java array in native code

자바 배열을 네이티브 코드에서 인자로 받아 사용하기 위해서는 자바 배열 인자를 C/C++ 배열 형식으로 바꾸어주는 JNI의 클래스 함수인 Get자료형ArrayElements(자바 배열, 사본 여부) 함수를 사용해야 한다. 이 함수는 동적 할당 방식으로 배열을 선언하고 자바 배열의 값으로 배열을 초기화하여 포인터를 리턴한다. 동적 할당을 사용하므로 사용을 완료한 후에 반드시 <그림 10>과 같이 클래스 함수 Release자료형ArrayElements(자바 배열, 배열 포인터, 삭제 방식)을 호출하여 메모리 할당을 해제해야 한다.

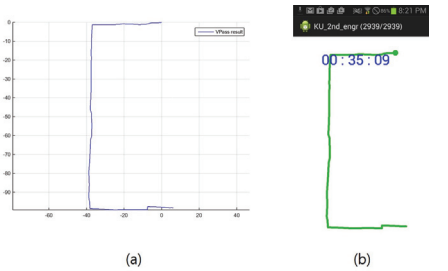
C/C++ 네이티브 코드에서 자바 클래스의 함수를 호출할 때 가장 중요한 것은 호출받은 클래스 내부의 함수만 호출할 수 있다는 것이다. 예를 들면 백그라운드 스레드에서 호출되어 실행되는 네이티브 코드는 UI 스레드의 액티비티 클래스에 선언되어 있는 자바 메서드를 호출할 수 없다. 따라서 백그라운드 스레드의 네이티브 코드로부터 직접적으로 UI 스레드에 접근하여 정보를 주고 받을 수 없기 때문에, 네이티브 코드의 연산 결과를 UI 스레드를 통하여 출력하기 위해 핸들러를 사용한다. 네이티브 코드에서 백그라운드 스레드의 자바 메서드를 호출하고, 호출 시 전달받은 인자를 다시 메시지에 저장하여 액티비티 클래스의 핸들러로 보내는 방식이다.

4.5 로그캣을 이용한 디버깅

그림 11. 네이티브 코드에서 로그캣으로 로그 메시지를 보내는 방법
Figure 11. Way to send Log message to LogCat in native code

이클립스의 로그캣을 이용하면 마치 윈도우즈의 콘솔 창처럼 액티비티에서 별도의 출력 UI를 구현하지 않고도 어플리케이션에서 발생한 이벤트에 대한 로그 메시지를 확인할 수 있다. 자바 클래스에서는 android.util.Log 패키지에 로그 출력 메서드가 정의되어 있으며, <그림 11>처럼 네이티브 코드에서는 log.h 헤더 파일에 정의되어 있는 __android_log_print(알림 수준, 확인용 태그, 문자열 내용) 함수를 이용하여 로그 메시지를 로그캣에 출력할 수 있다. 이 함수에서는 C/C++의 printf 함수처럼 %d, %f와 같은 변수 참조를 통해 변수의 값을 확인하는 것이 가능하다.

5. 실험



(a) (b)

그림 12 (a) 리눅스 환경에서 얻어진 결과 (b) 안드로이드 환경에서 얻어진 결과
Figre 12. (a) Result from Linux (b) Result from android

5.1 실제 코드를 이용한 검증.

본 논문에서 설명한 변환 과정을 이용해서 실제

로 C++ 코드를 사용하는 안드로이드 어플리케이션을 만든 후, 스마트폰에서 얻은 결과가 리눅스 환경에서 얻은 결과와 동일한 지를 확인하는 작업을 수행하였다. 실험에 사용한 스마트폰은 삼성의 갤럭시 S2 LTE이고, 안드로이드의 버전은 '4.1.2 젤리빈'이다. 검증 과정에서 사용한 C++ 코드는 SLAM 알고리즘[16]이다. 이 알고리즘은 미리 얻어진 이동로봇의 오도메트리 데이터와 레이저 센서 데이터가 저장되어 있는 파일을 불러 들인 후, 특징점 정보를 추출하여 에러가 보정된 지도를 만드는 것을 목적으로 한다. 검증을 위해 만들어진 안드로이드 어플리케이션은 C++ 코드를 연동하고, 자바와 네이티브 코드간의 입출력 부분을 추가하여 만들어졌다.

알고리즘에 의해 보정된 로봇의 경로 데이터를 분석한 결과, 서로 다른 두 운영체제에서 완전히 동일한 결과가 나왔음을 확인하였다. 는 각각의 운영체제에서 로봇의 경로를 좌표에 표시한 결과이다.

5.2 활용 예시

그림 13. PowerTutor 어플리케이션을 활용한 전력 소모량 측정
Figure 13. Power consumption measurement Using PowerTutor application

네이티브 코드로 작성된 알고리즘을 스마트폰에서 동작시켰을 때, 다양한 어플을 이용하여 그 알고리즘의 성능 분석이 가능하다. 그 하나의 예로, 안드로이드 마켓에 상용화되어 있는 PowerTutor 어플리케이션[17]을 사용하여 알고리즘의 복잡도(complexity)을 직관적으로 확인할 수 있다. <그림 13>처럼 이 어플리케이션은 실행되고 있는 모든 어플리케이션에 대한 전력 소모량을 실시간으로 보여준다. 페이스북이나 트위터와 같은 일반적으로 많이 사용되는 어플리케이션과의 전력 소모량을 비교함으로써[18] 특정 알고리즘의 복잡도 또는 모바일 실용 가능성을 쉽게 확인할 수 있다. 즉, 네이티브 코드를 안드로이드 운영체제에서 동작시키는 과정은 임의의 알고리즘의 복잡도를 직관적으로 보여주기 위한 하나의 방법으로 사용될 수 있다.

6. 결 론

본 논문에서는 C/C++ 네이티브 코드를 안드로이드 운영체제에서 구동하는 방법을 설명하였다. 네이티브 코드를 쉽고 빠르게 안드로이드 어플리케이션과 연동할 수 있도록 그 방법을 체계적, 순차적으로 정리했으며, 이를 적용하여 구현한 어플리케이션을 제시하였다.

안드로이드 NDK는 네이티브 코드를 안드로이드 환경에서 사용할 수 있도록 해 주는 유용한 도구이지만, 모든 네이티브 코드를 사용할 수 있게 해 주지는 못한다. 따라서 향후 연구를 통해 변환 가능한 코드와 그렇지 않은 코드를 구분하는 작업을 수행할 예정이다.

Reference

[1] L. Mingyang, B.H. Kim, and A. I. Mourikis, *Real-time motion tracking on a cellphone using inertial sensing and a rolling-shutter camera*, IEEE International Conference on Robotics and Automation, pp. 4712-4719, 2013.

[2] Google Project Tango website, https://www.google.com/atap/projecttango/#project

[3] W. Liu, J. Liu, Q. Wu, and B. Qin, *Android PBC: A pairing based cryptography toolkit for android platform*, Communications Security Conference, pp. 1-6, 2014.

[4] L.S. Andrés, M.A. López, J.P.G. Emillio, and V.R. Carlos, *An analysis of android smartphones as a platform for augmented reality games*, International Conference on Mobile Ubiquitous Computing, Systems, Services and Technologies, pp. 71-76, 2011.

[5] P.H. Truong, C.-W. Park, M.S. Lee, S.-I. Choi, S.-H. Ji, and G.-M Jeong, *Rapid implementation of 3D facial reconstruction from a single image on an android mobile device*, KSII Transactions on Internet and Information Systems, Vol. 8, No. 5, pp. 1690-1710, 2014.

[6] W. Wu, Q. He, Y. Wang, and Y. Hu, *An improved method of vibe for motion detection based on android system*, IEEE International Conference on Robotics and Biomimetics, pp. 2436-2440, 2013.

[7] G. Grisetti, R. Kummerle, C. Stachniss, and W. Burgard, *A tutorial on graph-based SLAM*, IEEE Intelligent Transportation Systems Magazine, pp. 31-43, 2010.

[8] H.-S. Oh, B.-J. Kim, H.-K. Choi, and S.-M Moon, *Evaluation of android dalvik virtual machine*, Proceedings of the 10th International Workshop on Java Technologies for Real-time and Embedded Systems, pp. 115-124, 2012.

[9] S. Lee, and J.W. Jeon, *Evaluating performance of android platform using native C for embedded systems*, International Conference on Control, Automation and Systems, pp. 1160-1163, 2010.

[10] J.-H. Lee, G.Y. Lee, and C.K. Jeong, *Performance comparison of implementation technologies for image quality enhancement operations on android platforms*, Journal of Digital Contents Society, Vol .14, No. 1, pp. 7-14, 2013.

[11] C.-M. Lin, J.-H. Lin, C.-R. Dow, and C.-M. Wen, *Benchmark dalvik and native code for android system*, IEEE International Conference on Innovations in Bio-inspired Computing and Applications, pp. 320-323, 2011.

[12] L. Batyuk, A.-D. Schmidt, H.-G. Schmidt, A. Camtepe, and S. Albayrak, *Developing and benchmarking native linux applications on android*, International Conference on Mobile Wireless Middleware, Operating Systems, and Applications,, pp. 381-390, 2009.

[13] Android Developer website https://developer.android.com/tools/sdk/ndk/index.html

[14] Y. Wu, J. Luo, and L. Luo, *Porting mobile web application engine to the android platform*, IEEE 10th International Conference on Computer and Information Technology, pp. 2157-2161, 2010.

[15] C.-W. Chang, C.-Y. Lin, C.-T. King, Y.-F. Chung, and S.-Y. Tseng, *Implementation of JVM too linterface on dalvik virtual machine*, International Symposium on VLSI Design Automation and Test, pp. 143-146, 2010.

[16] Y.H. Lee, C. Nam, K.Y. Lee, Y.S. Li, S.Y. Yeon, and N.L. Doh, *VPass: Algorithmic compass using vanishing points in indoor environments*, IEEE/RSJ International Conference on Intelligent Robots and Systems, pp. 936-941, 2009.

[17] Power Tutor Application, http://powertutor.org

[18] J.M. Lee, H.W. Joe, and H.S Kim, *Power*

consumption analysis of smart phone applications, Proceedings of the Korea Information Science Society FALL Conference, Vol. 38, No. 2(D), pp. 39-42, 2011.

네이티브 코드(C/C++)를 안드로이드 운영 체제에서 사용하는 방법과 그 활용법

안상민, 이건용, 정운형, 도낙주
고려대학교 전기전자공학부

요 약

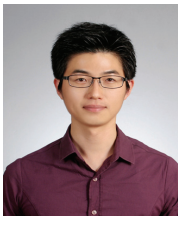
일반적으로 자바에 기반하는 안드로이드 어플리케이션은 달빅(Dalvik) 가상 머신상에서 구동된다. 플랫폼에 독립적인 가상 머신을 이용하여 어플리케이션을 구동하는 방식은 하드웨어에 손상을 주지 않고 어플리케이션을 안정적으로 구동할 수 있다는 장점이 있다. 그러나 가상 머신에서 구동하기 위해 코드를 변환하는 과정이 필수적으로 포함되면서 플랫폼에 의존하지만 기계어로 변환되는 네이티브 코드에 비하여 상대적으로 속도가 느리다. 또한 하드웨어에 직접적으로 접근하는 것이 제한된다는 단점 또한 존재한다. 수많은 프로그램들이 이미 C/C++를 기반으로 개발되어 있으며 이러한 프로그램들을 다른 언어로 다시 개발하는 일 없이 재사용할 수 있다면 개발에 소요되는 시간과 비용이 크게 절감될 것이다. 본 논문에서는 JNI(Java Native Interface)를 통해 C/C++로 짜여진 코드를 안드로이드에서 구동하는 방법을 제시한다. 먼저 자바 클래스에서 C/C++ 네이티브 함수를 호출하는 방법과 C/C++ 네이티브 코드에서 자바 클래스 내부에 선언된 자바 함수를 호출하는 방법을 알아보고, 다른 언어로 짜여진 코드 간에 정보를 주고 받으며 상호 연동하는 과정에 대하여 설명한다. 또한 안드로이드에서 네이티브 코드를 활용한 실제 예를 구체적으로 설명한다.

감사의 글

이 논문은 2015 한인도 협력 사업으로 수행된 연구임.

Sangmin Ahn has been a undergraduate student in the School of Electrical Engineering from the Korea University since 2008. His current research interests include Android application Using JNI.

E-mail address: snoopy8945@korea.ac.kr



Keonyong Lee received the bachelor's degree in the School of Electrical Engineering from the Korea University in 2009. He has been in M.S. and Ph.D. Integration course in the School of Electrical Engineering at Korea University since 2009. His current research interests include SLAM(Simultaneous localization and mapping), Mobile Robot

E-mail address: keonyongpaul@gmail.com

Woonhyung Jung received the bachelor's degree in the School of Electrical Engineering from the Korea University in 2015. He has been in M.S. and Ph.D. Integration course in the School of Electrical Engineering at Korea University since 2015. His current research interests include Motion Planning, SLAM(Simultaneous localization and mapping).

E-mail address: jungjwh@gmail.com

Nakju Lett Doh received the bachelor's degree in the Department of Mechanical Engineering from the Pohang University of Science and Technology in 1998. He received the M.S. degree and the Ph.D. degree in the Department of Mechanical Engineering from Pohang University of Science and Technology in 2000 and 2005, respectively. From 2005 to 2006, he was a Senior researcher at Intelligent Robotics Research Division, ETRI. He has been a professor in the School of Electrical Engineering at Korea University since 2006. His current research interests include SLAM(Simultaneous localization and mapping), Mobile Robot localization and navigation

E-mail address: nakju@korea.ac.kr