International Journal of Networked and Distributed Computing, Vol. 1, No. 3 (August 2013), 159-166

Device driver development of an external human interaction device for the Android

Naoto Ogawa [1] , Shuichi Oikawa [2]

[1] *Department of Computer Science, University of Tsukuba,*
1-1-1 Tennodai,
Tsukuba, Ibaraki 305-8573, Japan
E-mail: nogawa@cs.tsukuba.ac.jp

[2] *Division of Information Engineering, Faculty of Engineering,*
Information and Systems , University of Tsukuba
E-mail: shui@cs.tsukuba.ac.jp

Received 16 April 2013

Accepted 25 June 2013

Abstract

Recently, many people have used Smartphones. The feature of Smartphones is that they are equipped with a mobile operating system (OS) for the control of sensors and modules. In this situation, we can connect external human interaction devices with the Smartphone. Our objective is to know how the external human interaction device affects to the OS. In this paper, we implement the programs for connecting the Wii Remote to the Android. Furthermore, we evaluate the performance of our programs and benchmark the Android while running our programs. We intend to shift our environment to the virtualized environment.

Keywords: device driver; Android; Linux; Bluetooth; Wii Remote;

1. Introduction

These days, the user of smart phones is increasing in all over the world[1]. Few people used the Smartphone at 2008, while now, 70% of cellular phones are Smartphones. Moerever, the rate of spread is expected to reach 80% by the end of 2016.

One feature of the Smartphones is that they employ a mobile operating system (OS), such as iOS or Android. Another feature of Smartphones is that it is equipped with in number of sensor modules that include GPS, Bluetooth, acceleration sensor, direction sensor, luminance sensor and so on. Because it has many modules and it also can connect the kinds of external devices, a large amount of data come from modules and/or external devices to the smart phones, so smart phones must process these data. These data are too large, so if we don't process these data effectively, the application crash at worst. There is no study how we can process data from the external device effectively and how external devices can be connected with Android.

The objective of this study is to examine how the data from the external device affect the Android and how time the input event complete. For our objective, we connect the external device with the Android and handle the Android by the external device.

We implement programs to get the data from the Wii Remote and use this data to make input event to the Android. We get the each processing time of the

Published by Atlantis Press

input event by using this program.

There is no profiling software on our environment now, so we also prepare profiling software for our environment.

Furthermore, we plan to migrate our execution environment to the virtualized environment.

2. Related Works

Benchmarking for the Android system already exists[23], but these studies are about Dalvik virtual machine. Therefore, these studies are different in the point of, not for sensor devices.

The study using the sensors of the Android platform is present[4]. This study pays attention to the data from the sensor itself and don't present the influence of the Android system.

The studies using the Wii Remote as a human interaction device are present[567], but the platform of these studies are not Android system.

The study using the external device as a human interaction device is present[8]. This study uses Kinect[16]. The Kinect is enable to detect the distance and the picture by using the sensor. In this study, they make input an event with human gesture.

The investigation of input interface using acceleration sensor of the Android is present[9]. In this investigation, the data from the acceleration sensor makes input event. This study is different from our study in the point of that they don't use the external device.

3. Execution Environment

We set up the following environment to get data from the external device and handle these data.

3.1. OS

We choose Android[11] as an embedded operating system on Smartphone because Android is an open source software, and it is Linux based operating system so we can use some software for Linux. The version we selected is 4.0.4 (Ice Cream Sandwich).

3.2. Board

We choose Pandaboard-ES[12] for the target board. Pandaboard-ES is a single-board computer based on the OMAP4460 developed by Texas Instruments. The features of Pandaboard-ES are a dual-core 1.2GHzCPU and 384MHz GPU, 1GB low power DDR2 RAM, on-board 10/100 Ethernet, 802.11 b/g/n, Bluetooth v2.1, general purpose expansion header, and so on. Some Smartphones use the OMAP4460, so this board is similar to the Smartphone. Figure1 is the appearance of Pandaboard-ES.

Fig. 1. Pandaboard-ES

3.3. Wii Remote

We choose the Wii Remote[14] as an external device to connect with the Android. Wii Remote is a standard controller for Nintendo Wii[13]. The Wii is a home video game console released by Nintendo. This controller connects to the console by Bluetooth. We will use this controller to operate Android on Pandaboard-ES. Figure2 is the appearance of Wii Remote.

Fig. 2. Wii Remote

The other device like Wii Remote exists. This product is Zeemote JS1H[15] that is made by BUFFALO INC. This device connect with the Android by Bluetooth, but it uses Serial Port Profile (SPP) for the connection. This method emulates a serial cable to provide a simple substitute. On the other hand, the Wii Remote uses the Human Interface Device Profile (HID). This method is for connecting input device like a mouse or keyboard.

4. Implementation

We implement a program to use Wii Remote for handling the Android. This program is based on Wiiuse. Wiiuse is a C library to connect Wii Remote by Bluetooth. This program enable to get the data from the Wii Remote and Nunchuk if connected, but this library is for Linux and Windows, so we can't use this library on the Android originally. Thus, we must make a change to the library to use on the Android.

The Android NDK is a tool set that enables to make Android applications or executable program from C and/or C++ source files. We enable to use Wiiuse by using this tool and preparing the required libraries. We prepare libraries about Bluetooth[10].

We use Android Debug Bridge (adb) to connect with the board and a PC. After connecting, we can use the shell of the Android from the PC's terminal. We execute the Wiiuse on the shell from the PC. Figure3 is the system configuration diagram.

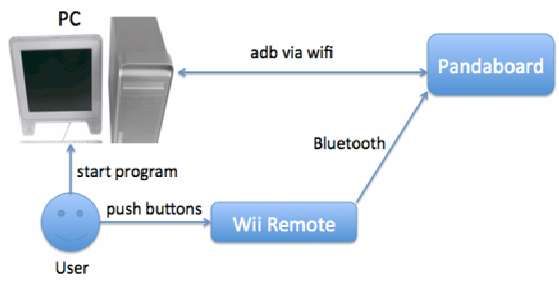

Fig. 3. system configuration diagram

We would like to use the Wii Remote to handle the Android, but Wiiuse can only connect with Wii Remote and get data from the Wii Remote. Thus, we must implement an input function to the Android. Table 1 is the table from the Wii Remote button to the keyboard and mouse. BTN_LEFT corresponds to mouse left click event and BTN_RIGHT corresponds to the mouse right click event.

Table 1. Wii Remote correspond table

Wii Remote	Keyboard
⇑	KEY_UP
⇓	KEY_DOWN
⇐	KEY_LEFT
⇒	KEY_RIGHT
A	KEY_ENTER
①	KEY_SEARCH
②	KEY_ESC
HOME	KEY_HOME
C	BTN_LEFT
Z	BTN_RIGHT
Joystick	mouse cursor

We implement the input method in three ways. First method use shell command, second method uses Linux libraries and third method use device driver.

We can use mouse function if we use the second or third method.

4.1. Shell Command

Android shell has input command. This command makes keyboard input event. This command is used like this

 # input keyevent EVENT_CODE

If you would like to make the ENTER key input event, you should just type like this on the shell

 # input keyevent 66

These EVENT_CODEs are defined in Android's source code.

Because NDK contain C libraries, we can use functions for executing shell commands in the C source. We duplicate the process. Duplicated process executes the input command. This command generates Virtual Machine (VM) and execute the Java program on the VM. This Java program makes key input event and sends key input event to the Android application. The other process waits for the

first process to be completed. One input process is completed now. This program polls till next input comes. Figure4 shows the summary of this program.

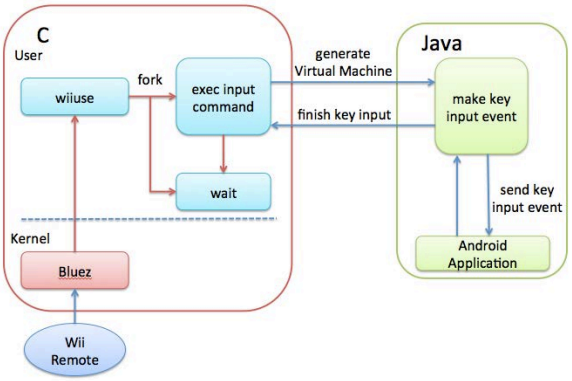

Fig. 4. use shell command

4.2. Linux Libraries

We implement another version. This version makes virtual input device. Android prepares user input module, so we use this module. We open uinput[17] device file to use this module and write device type we would like to generate to this file. In this case, the type is keyboard and mouse. 'ioctl'[18] function manipulates the underlying device parameters of special files. We write some kinds of events that we will generate to this file by using the ioctl function because the Linux standard device driver does not pass the input event to the application if this device file doesn't have information about the events that this device can generate. Now we are completing creating virtual input device.

We can generate input event by writing the relevant value to the virtual input device file. When this program gets data from the Wii Remote, this program makes key input event by writing the virtual input device file. One input process is completed now.

However, we couldn't use HOME button because Android doesn't get KEY_HOME from the keyboard, so we change the configuration file to get KEY_HOME from the keyboard. Figure5 shows the summary of this program.

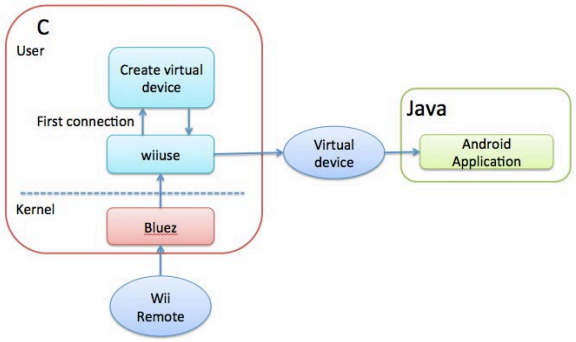

Fig. 5. use ioctl function

4.3. Device Driver

We implement another version. This version makes the device driver for Wii Remote. Our Android prepares insmod and rmmod commands so we can use kernel module on this Android. Device driver for Ubuntu is already existing, but we cannot use this driver on the Android. Thus, we implement a device driver based on it for the Android. When we load this driver, a character device is generated at '/dev/wm'. We generate input event by writing to this character device file. The type of input event is similar to the virtual input device version.

This driver can make the input event but can't get data from the Wii Remote, so we must prepare a program which gets the data from the Wii Remote. We change virtual device program not to write virtual input device but to write '/dev/wm'. Figure6 shows the summary of this program.

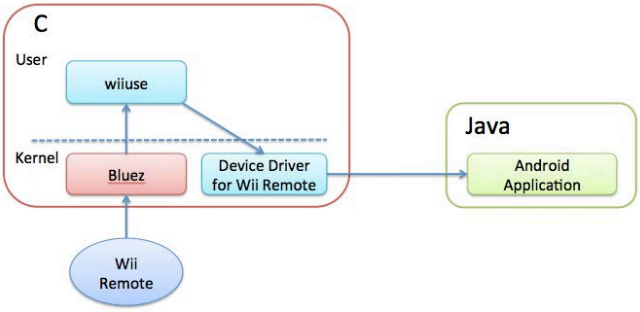

Fig. 6. use Wii Remote device driver

5. Evaluation

5.1. One Input Time

We measure the time to complete one key input event on each version. We use a gettimeofday function to get time. We get time before generating the input event. This is the start time. We get time when the input event is completed. This is the end time. We determine that the value obtained by subtracting the start time from end time is the execution time. Table 2 shows the execution time of each button and Figure7 is the graph to compare the execution time of three versions. We get each execution time ten times and calculate each average execution time. The row of Average means the average execution time of each method. Figure7 is logarithmic graph and time are represented at a microsecond rate.

Table 2. each button input time

Method \ Button	UP	DOWN	LEFT	RIGHT	ONE	TWO	A
Device driver	15.56	16.16	20.76	12.84	21.97	15.88	17.10
Virtual device	17.73	18.00	22.90	20.13	24.09	21.68	21.06
Input command	$65.30*10^3$	$63.96*10^3$	$66.66*10^3$	$64.00*10^3$	$70.22*10^3$	$70.48*10^3$	$66.71*10^3$

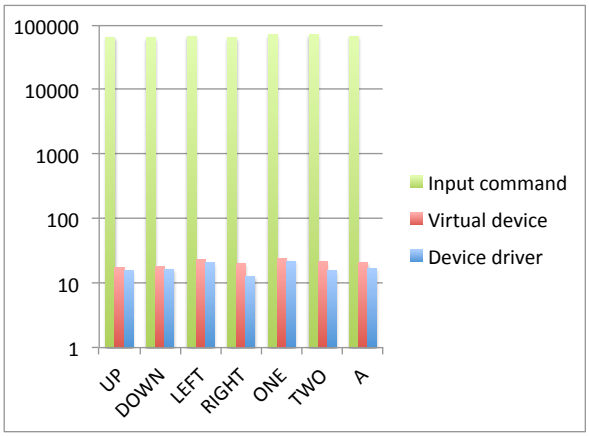

Fig. 7. compare three versions

The execution time of the input version is about ten thousand times the execution time of the other versions of all buttons. The input version is too late so when we use the Wii Remote to handle the Android and hit buttons repeatedly, the application doesn't work as we mentioned immediately. We get stressed in this situation.

On the other hand, the virtual device version and device driver version are so fast. As you see the Table 2, all execution time is under one millisecond and device driver version is a little faster than the virtual device version. We don't feel stress when we use the Wii Remote as an input device.

We examine the input command version in detail to discover where does occupy the majority of the execution time. We measure the time of the process waiting. We get start time before fork function and end time after the waiting process is restarted. We determine the value obtained by subtracting the start time from the end time as the fork time. Table 3 shows the total execution time, the fork time and the others time. Figure8 is the graph comparing the fork time and the others time.

Table 3. each total processing time and fork time of the input command version

	UP	DOWN	LEFT	RIGHT	ONE	TWO	A
total time(us)	625549	637848	656005	672271	772614	789551	786346
fork time(us)	625366	637024	654876	671692	771881	780337	785980
the others(us)	183	824	1129	579	733	214	366

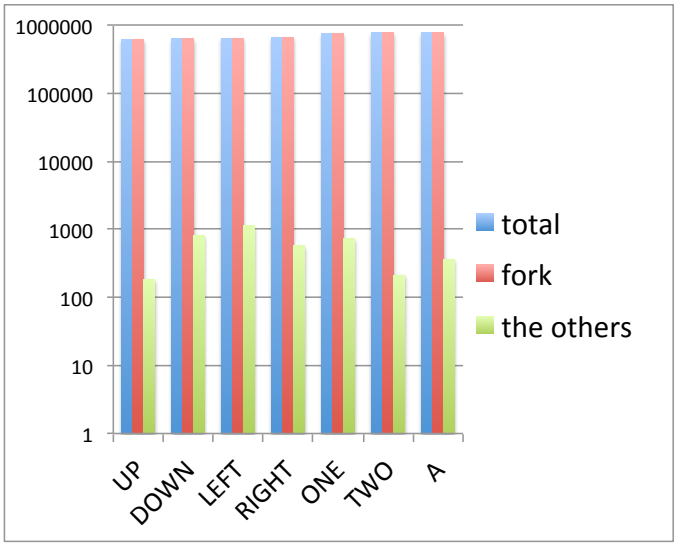

Fig. 8. compare occupied time of the fork and the others

As you see, fork time occupies almost all execution time, but we cannot judge whether the process generated by the fork function or input command occupies almost all the time.

We measure the time of input command itself on the shell. We use the time command to measure the time. This command gets the execution time the command, but this command gets only two decimal places. Table 4 is the result.

Table 4. each processing time of input command on the shell

	UP	DOWN	LEFT	RIGHT	ONE	TWO	A
time(s)	0.64	0.64	0.62	0.63	0.69	0.70	0.72

The each execution time of Table 4 almost corresponds to Table 2. This result suggests that the input command occupies almost all execution time of the input command version.

5.2. Top Command

To examine how these three program occupies the cpu time, we perform 'top' command while running each program. Table 5 is the result.

Table 5. top command results

Input Methd	Device driver	Input command	Virtual input device
Average(%)	0.1460	6.000	24.13

This result suggests that the virtual input device method uses more CPU time than device driver method, so we predict that virtual input device method has a bad influence on the Android system. We think that virtual input device method uses three threads in the program and manage virtual input device file, so this program use much CPU time than other method.

5.3. Java Whetstone

To examine how the sensor and the program affect on the Android system, we perform Java Whet Stone benchmark. In this case, the sensor is Bluetooth module. This Java Whetstone Benchmark is written by Roy Longbottom. This application provides a result of speeds of the eight test loop in the benchmark and MWIPS that means Millions of Whetstone Instructions Per Second. The speed is represented by Millions of Floating Point Instructions Per Second (MFLOPS) and Million Operations Per Second (MOPS). While running this benchmark, we connect the Wii Remote to the Android but don't handle the Wii Remote, so input event isn't generated. Figure9 is the each method result of the Java Whet Stone benchmark.

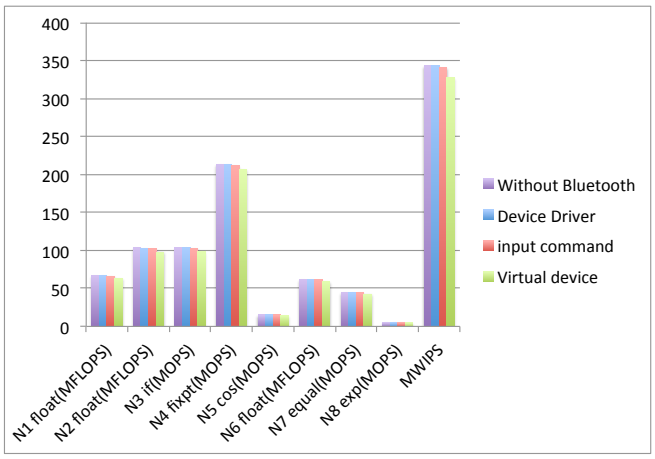

Fig. 9. Java Whet Stone results

The results are almost same. Virtual input device version is a little worse. This result agrees with our prediction.

6. OProfile

We make 'OProfile' available at Android on the Pandaboard-ES. OProfile is a system-wide profiler for Linux system. OMAPpedia [19] provide the way to use OProfile at Android on the Pandaboard-ES, but we can't use OProfile in that way. In our environment, we can't use hardware event counter, so we use timer interruption to profile the programs. We execute 'opcontrol' command to get profiling information on the target board. To see the result of the profiling, we have to pull the result of the profiling of the target system to the host computer running Linux because we have to execute 'opreport' command to see the result. However, we can't execute it on the Android. This command can't execute if the version of the 'opcontrol' command and the 'opreport' command are different. In our environment, these versions are different. We make change the opreport to be available in our environment, and now we can use opreport. Figure10 is the profiling result of the

virtual input device method. 'wiiuse5' is the program name of the virtual device method.

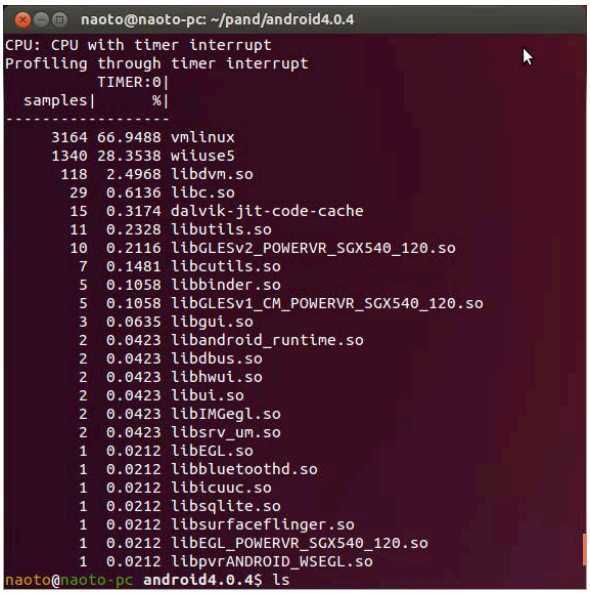

Fig. 10. OProfile result of virtual device method

We can see many libraries in this figure, but we can't find which libraries do correspond with our program.

7. Virtualization Expansion

We shift our execution environment to a virtualized environment. We have evaluated the performance of our programs and the Android on the Pandaboard-ES. Now we can run the Android on the virtual machine monitor (VMM). We choose 'KVM[20]' as a VMM. Figure11 shows the virtualized environment we intend to construct. We run the Linux and the Android on the VMM. The Linux get the data from the Wii Remote through Host OS and VMM. The Linux sends some data to the Android to handle the Android. In this situation, we will evaluate the performance of the Android.

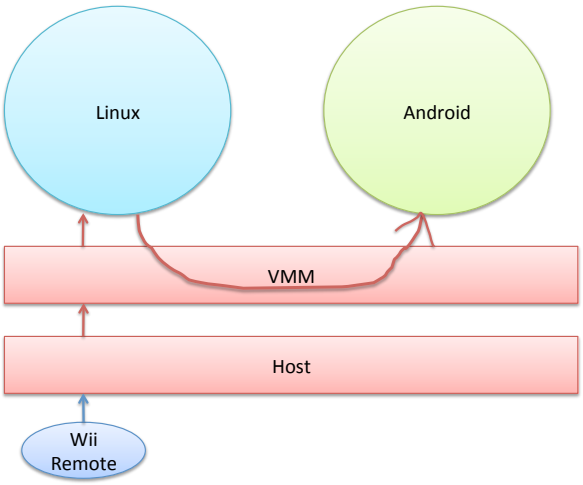

Fig. 11. virtualized environment

8. Discussion

We implement the methods to handle data from an external device and measure the execution time at each method. Significant difference appears in the execution time. The method that use input command is so late that it is not practical. On the other hand, the other methods are so fast that it can use as an input method. This result is as we expected because the methods that use ioctl or device driver are a single process and runs on the real machine, whereas the method that use the input command duplicate the process and generate VM. Therefore, we can say that it is obvious that the methods using ioctl or device driver are faster than the method using input command.

Comparing ioctl and device driver, ioctl use more CPU time than device driver. This cause is that ioctl use three threads to handle the data from the Wii Remote. On the other hand, the device driver version uses two threads. Furthermore, the ioctl must manage virtual input device in its process while device driver version doesn't have to manage Wii Remote device driver. Android manages Wii Remote device driver. We consider using short CPU time is scalable. Thus, we predict we can connect other devices while connecting the Wii Remote using device driver method.

Furthermore, we consider that when you connect an external device to the Android and handle data

from the device, you should implement the program to handle the data with C and/or C++.

We estimate that the Android increasingly spread from now on and the demand for connecting the external device with the Android increase.

9. Conclusion

9.1. Summary

In this paper, We connect the Android with the Wii Remote to get the data from the external device. We implement input methods in three ways. First one makes another process in the program, and use shell function to make input event. Second one uses C library to make input event from the data on the Wii Remote. Third method use device driver for the Wii Remote. We implement this device driver. Comparing three methods, the second and third one use short time to complete a keyboard input event. Comparing the second one and the third one, the third one uses short CPU time than second one. So we think the third one is the best method to use an external device for handling the Android.

We think connecting Wii Remote with the Android hardly have a bad influence by reason of our benchmark.

9.2. Future Works

We think that our implementation has three points that should be improved.

1. Our device driver requires the daemon which runs on the user space for getting the data from the Wii Remote. This still requires context switching, so we do change our device driver to run in the kernel completely.

2. Our Android has no debug symbols, so we can't get detailed information about our programs. We will compile the Android with debug symbols.

3. Virtualized environment has not been completed yet. Thus we will construct a virtualized execution environment.

1. Sangwon Lee and Senmi Lee, *Diffusion of Smartphone in Global Telecommunication Markets*, PTC'12, 2012-01-17
2. Cheng-Min Lin, *Benchmark Dalvik and Native Code for Android System* , IEICE Technical Report Innovations in Bio-inspired Computing and Applications (IBICA), 2011 Second International Conference on, 16-18 Dec,2011
3. Takashi Majima et al., *CPU Load Analysis Using Dalvik Bytecode on Android*, IEICE Technical Report 109(475), 125-132, 2010-03-26
4. Tomonori Arakaki et al.,*Trial Manufacture of the Hadoop-based LifeLog System using the Android Terminal*, IPSJ SIG Technical Report.UBI, 2011-UBI-32(5), 1-4, 2011-11-17
5. OKAWA Takashi et al., *Development of three-dimensional graphic viewer software using game-device Wii®Remote and medical image displaying*, Medical Imaging and Information Sciences, 28(2), 46-50, 2011
6. Tomohisa Ogawa and Jiro Katto, *Percussion Instrument Interface Using Wii Remote Controller*, IPSJ SIG Technical Report.HCI, 2011-HCI-143(1), 1-6, 2011-05-20
7. Sugimoto Masaki et al., *Input Method of Japanese Sentence by Using Wiimote*, ITE Technical Report, 35(16), 59-62, 2011-03-08
8. keiji Sakata and Takao Takahashi, *Development of learning system using noncontact input device*, IPSJ SIG Technical Report.CE, 2011-CE-112(1), 1, 2011-12-10
9. Fukunaga Hibiki et al.,*Investigation of Input Interface using Android Device embedded Acceleration Sensor*, Proceedings of the ... Society Conference of IEICE 2011_Communication(2), 436, 2011-08-30
10. Bluez:http://www.bluez.org/
11. Android: http://www.android.com/
12. Pandaboard-ES: http://pandaboard.org/
13. Nntendo Wii: http://www.nintendo.com/wii
14. Wii Remote: http://android.ccpcreations.com/wiicontroller
15. Zeemote:http://www.zeemote.com/
16. Kinect:http://www.xbox.com/en-US/KINECT
17. uinput:http://thiemonge.org/getting-started-with-uinput
18. ioctl:http://man7.org/linux/man-pages/man2/ioctl.2.html
19. OMAPpedia http://omappedia.org/wiki/Android_Debugging
20. KVM http://www.linux-kvm.org/