Enabling low-complexity devices for interaction with 3D media content via Android API

Ricardo Santos[1]
[1]Polytechnic Institute of Leiria / ESTG
Leiria, Portugal
ricardo.a.santos@ipleiria.pt

Hugo Costelha[1,2], Luis Bento[1,3] and Pedro Assunção[1,4]
[2]*INESC TEC*, [3]*ISR-UC*,
[4]*Instituto de Telecomunicacoes*
Leiria, Portugal
{hugo.costelha, luis.conde, pedro.assuncao}@ipleiria.pt

Márcio Barata[5]
[5]Tech4Home
São João da Madeira, Portugal
marcio.barata@tech4home.pt

Abstract

This paper deals with an interactive multimedia system based on Android OS, where several functional modules were developed to enable the use of low-complexity remote control devices. The system architecture comprises the remote control device with Magnetic, Angular Rate, Gravity (MARG) sensors for 3D motion tracking and an Android set-top-box, integrating a novel Application Programming Interface (API), specifically developed for this purpose. A proper decision whether the most complex functions should run on the remote control device, or on the Android set-top-box, is an open issue and depends on the specific application and the desired portability. Therefore taking into account energy consumption when balancing the computational burden is paramount. Given that the set-top-box has no limitations on energy consumption and has a superior computational power, the propose API can perform all the processing of sensors data, allowing the implementation of complex fusion algorithms with higher precision. The analysis of energy consumption on the remote control device shows that transmitting the raw sensors data, to be processed in the API, results in lower energy consumption in the remote control device, and consequently higher autonomy with good accuracy.

Keywords

Android, API, Set-Top-Box, USB, HID

1. INTRODUCTION

In the past years there has been a strong investment in technology development for television and multimedia consumer market in general. Besides the evolution of screen resolutions, there has been an evolution that is bringing new types of multimedia content. Before this evolution, the user had a limited interaction with the available content in the television, but the trend is to have more interactive multimedia content and applications. However the devices used for interaction did not follow this evolution, leading to a poor Quality of Experience (QoE). The Remote Control Device (RCD) of current Set-Top-Box (STB) or smart TV is used for interaction with multimedia content, mainly based on two dimensions (2D) [Ohnishi 12]. The evolution to 3D content and operation with enhanced interactive functionalities, requires mapping of the RCD movements into motion on the 2D screen [Zidek 13]. To reduce integration barriers, the main manufacturers are moving towards Android-based systems. This operating system has increasingly been adopted for multimedia services both on television[1] and STB[1] [Song 10]. Since Android is an open system, it provides easy access to its internal architecture, allowing faster development, implementation and testing of Application Programming Interfaces (API).

In the scope of this research, a system for 3D interaction with multimedia content was developed and tested. This system, comprised of an RCD transmitting sensors data to an STB, includes three functional modules: communication, processing and application layer. One of the challenges addressed in this work is related to compute orientation estimates in the RCD. This is because running complex algorithms in the RCD, results in higher energy consumption for processing but lower energy consumption for communications, due to less data being transmitted. Therefore, a proper computational balance between the RCD and the STB is important, *i.e.* one has to decide whether the most complex functions (in terms of computational complexity) should run in the RCD or in the STB, taking into account energy consumption. Transmitting RAW data to the STB increases the energy required for communications, but allows the implementation of more complex algorithms on the STB, thus leading to more accurate estimates.

[1]https://www.android.com/tv/

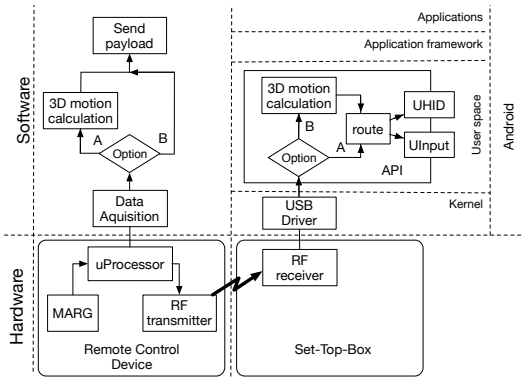

Figure 1. System architecture.

The Application Programming Interface (API) was designed to have the least possible impact on the Operating System (OS), i.e., the API does not require any change in the Android OS kernel and frameworks, therefore simplifying its use. The API receives data from RCD and makes it available to the OS after the computation process. The implementation on the STB side, allows access to information about the user system (e.g., available resources).

This article addresses the analysis of the computational balance between the RCD and the STB. The sensors data processing – implementing an Air mouse that behaves like a pointer – can be done in the RCD or in the STB, depending on the evaluation of both the energy consumption profile and QoE [Rasteiro 15].

2. SYSTEM ARCHITECTURE

The system architecture is comprised of two functional entities: hardware and software. Figure 1 presents the overall system architecture. The system hardware consists of an RCD with 6 Degrees of Freedom (DoF) and a STB. The RCD is able to track 3D motion using a set of sensors commonly known as Magnetic, Angular Rate and Gravity (MARG). MARG Sensors are composed by an accelerometer, gyroscope and magnetometer, each with 3 orthogonal axes. In this research we have used a MARG that also includes an Application-Specific Integrated Circuit (ASIC) embedded processor, designated as Digital Motion Processor (DMP), which computes the orientation of the device using the information retrieved by the accelerometer and gyroscope sensors. The Android-based STB runs the OS 5.0. The STB has no limitations on energy consumption and a far superior computational power, ROM and RAM memory, when compared to the RCD.

To evaluate the problem of computational load balance, two solutions, identified as option "A" and "B" (see figure 1), were implemented:

Option "A" RCD movement is acquired by the MARG unit, the raw data is used by the RCD processing unit, and the processed data is transmitted to the STB;

Option "B" All raw data is transmitted from the RCD to the STB where it is processed.

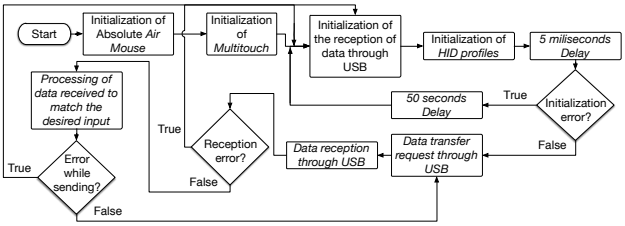

Figure 2. API flowchart.

The processed data in Option "A" results in far less data to be transmitted, as described later.

An external module (USB dongle) was developed to implement the communication between the RCD and the STB via Radio Frequency (RF). The dongle receives data from the RCD through ZigBee® Radio Frequency for Consumer Electronics (RF4CE) protocol and forwards them to the STB through Universal Serial Bus (USB) 2.0 Human Interface Device (HID) custom [USB-IF 00], [USB-IF 01].

3 SET-TOP-BOX

Although the STB is based on Android OS and the communication is made through an USB dongle, it is not possible to use the HID software stack to implement all the interfaces (e.g., absolute mouse HID) because the data must be pre-processed before making it available to the OS. Another limitation imposed by the OS, is that it does not allow to get the mouse pointer coordinate values when absolute coordinates are used. The system was implemented through an API running in STB for both options ("A" and "B"), that was designed to have the least possible impact in the STB, to avoid the need for recompiling the kernel or the Android OS.

The API was implemented in the Hardware Abstraction Layer (HAL) user space, because this layer is not hardware-dependent and allows receiving data from any communication interface (*e.g.* Bluetooth, I2C, SPI, etc.), as shown in figure 1.

The main purpose of the API is:

Option "A" to integrate the processed sensors data from RCD with screen information from the user's setup.

Option "B" to perform all the heavy processing that requires a great deal of power consumption in the RCD and integrate the result data with screen information from the user's setup.

The API was developed in the C programming language as a native application. For the development of such applications there is a Native Development Kit (NDK) available for Android devices. However this was not used, since it requires an application in Java for Android to run and start the native application. Thus, only the GNU Compiler Collection (GCC), available with NDK, was used to build the native application developed for the Android device.

The application flowchart can be observed in figure 2. The native application must start when the device is powered, therefore one had to make changes to the bootloader, to

```
...
service hermes /data/local/API/REMOTE6DOF
   class main
   user root
...
```

Code 1. init.< *device* >.rc

load the API as a service. Those changes consisted in editing the "init.< *device* >.rc" file (code shown in code 1), followed by the build and flash of the boot image into the device [Yaghmour 13].

Given that the API is in the HAL level, it is possible to declare HID profiles and inputs that make the data available for the entire OS and respective applications at higher layers. Figure 3 shows the interaction between the blocks of the API in the Android software stack. After starting the API, two input profiles on the Android OS are declared and initialised: the pen and multitouch inputs. These profiles were created using a module in user space to create and handle the input devices, *i.e.* an "uinput" kernel module. To create a new virtual device the following sequential actions have to be taken:

1 Open the user interface ("/dev/uinput") and create a temporary device;

2 Publish which input events the device will generate;

3 Create a structure with the basic information of device, namely the maximum and minimum values for the input events;

4 Send the command to the interface to create the device.

The pen input device was implemented to overcome the Android OS limitation of not making the pointer visually available for the absolute mouse. Two conditions have to be fulfilled in order to make the pointer visible on the screen: (i) explicit configuration of the requirement for a pointer and (ii) claim that the pen is in the range of the screen.

These virtual input devices are initialized taking into account the screen size of the device where it is running. This is done by reading the resolution field "FBIOGET_VSCREENINFO" of the framebuffer "/dev/graphics/fb0". The relative mouse, joystick, gamepad and consumer eletronic virtual USB HID devices, are created through a similar procedure but this time, using an USB interface in user space "/dev/uhid".

Using the "libusb"[1] the USB is started as host in order to receive data from the dongle. If there is an error during any of the initializations, the API waits 50 seconds and tries again, repeating the process until there is no error. This ensures that the API only continues after establishing a proper connection with the dongle. It should be noted that the device may not be connected when the API is started, so through this cycle, it can be ensured that the device is detected with a maximum delay of 50 seconds from the connection.

[1] http://libusb.info

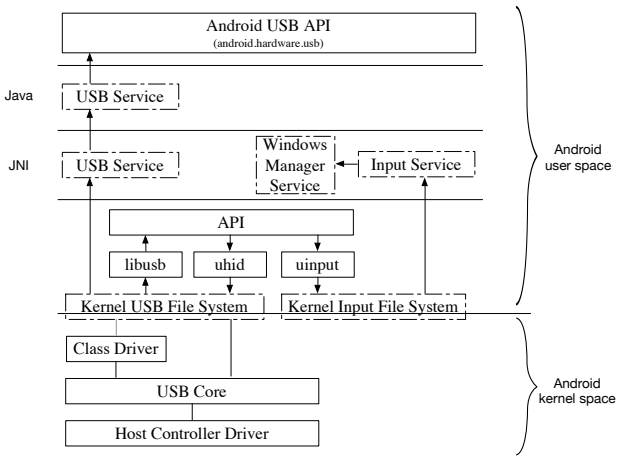

Figure 3. Android API Stack [Regupathy 14].

The initialisation request for information is sent to the dongle and the data is read from the USB buffer. The received data passes through an error checking routine and, if there are no data errors, the data is handled and sent to the corresponding USB HID profile. If an error occurs while receiving or sending data, a soft reset is performed by software, leading to an API reinitialisation which ensures that there is no accumulation of errors.

4 ENERGY CONSUMPTION ANALYSIS ON THE REMOTE CONTROL

The RCD modules consists of a Microcontroller Unit (MCU), MARG sensors and a RF module. In order to chose the best option (Option "A" or "B") in terms of greater battery life, an analysis on the energy consumption of each module was made. Consumption measurements were taken with the Analog Discovery board that provides a 100 kHz sampling rate and 14 bit resolution.

Five setups were implemented and tested, comprising representative sensors data acquisition and software algorithm implementation. Setups 1, 2 and 3 correspond to the processing in the RCD (Option "A"). Setups 4 and 5 have the computational load in the STB (Option "B"). All setups are presented in table 1.

4.1 Tests Characterization

TheQoE was taken into account in the power consumption tests, since it is affected by the pointer position refresh rate on the screen, which depends on the sensors data processing and transmission frequency. For all tested setups, the data acquisition is done at 100Hz and transmission at 50Hz. In the first setup, the data from the sensors is acquired in RAW and, using the MhF [Rasteiro 15], is computed the device orientation to determine the HID relative mouse position. Results are fitted in 2 bytes (X and Y) and sent via RF, with 1 byte representing the header. The second setup also uses the MhF to determine the HID absolute mouse position, in this setup the results are fitted in 4 bytes (X and Y). The two added bytes arise by matching the range of values with the size of the screen, which requires at least 2 bytes for each dimension. In the setup 3 the device orientation is computed by the DMP, so it

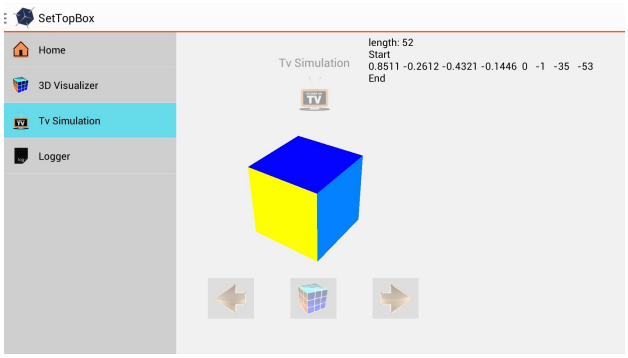

Figure 4. Android application.

is not necessary to apply further data processing. The results are also fitted in 4 bytes (X and Y) has in the previous setup. Setup 4 consists in the acquisition of raw data and its respective transmission, i.e., 3 bytes for each sensor (Accelerometer, Gyroscope and Magnetometer), plus 1 byte for the header. In the setup 5 both DMP and RAW data are acquired, but only the orientation computed by the DMP and the Magnetometer data are transmitted. It consist on 3 bytes for each component of the DMP, 2 bytes for the Magnetometer data and 1 byte for the header.

4.2 Analysis of Results

Setups 1 and 2 were used to test the sensors data processing in the RCD using the RAW data, which results in the relative and absolute mouse, respectively. As expected, the absolute mouse implementation requires more energy, as it needs more processing and more data to be sent. Setup 3 implements part of the sensors data processing in the sensors module through the DMP. Results show that it requires more energy than Setup 2, which performs data processing from the RAW sensors data in the microcontroller. In Setup, 4 RAW data is obtained from the sensors without any processing being done in the RCD. Although this results in more data to be transmitted, this Setup revealed to consume less energy than all the Setups presented above. Setup 5 uses the DMP processing in sensors module in order to transmit less data, however it consumes more energy than the Setup 4. The results listed in table 1 show that there is less energy consumption on the acquisition of RAW data. Setup 4 has the lowest consumption, since no processing is done in the remote, as it consists in reading RAW data and its respective RF transmission. When the processing is performed in the RCD, acquire raw data and processing Mahony filter evidence a lower power consumption compared to acquire DMP plus RAW data from sensors in order to avoid Mahony filter processing.

The QoE was tested using an Android application, shown in Figure 4, specifically developed to simulate and test a potential usage environment. The results of the subjective tests, that were carried out to evaluate how friendly is the RCD to non-expert users, revealed that absolute orientation computed in the STB presents smoother motion tracking and good user experience.

Setup	Data acquisition	Computational processing	Payload [Bytes]	Energy consumption [mJ]	Peak duration [ms]
1	Gyro + Acc + Mag	MhF + *Air Mouse* Rel.	3	1.1612726	16.69
2	Gyro + Acc + Mag	MhF + *Air Mouse* Abs.	5	1.2275447	16.92
3	DMPquat + Gyro + Acc + Mag	*Air Mouse* Abs.	5	1.5198403	17.92
4	Gyro + Acc + Mag	-	19	1.1252908	16.98
5	DMPquat + Gyro + Acc + Mag	-	15	1.4749114	18.18

Table 1. Remote control setups and results.

5 CONCLUSIONS

In this research, the energy consumption analysis revealed that the computational processing of RCD sensors data should be made in the STB. Although more data needs to be transmitted when using this option, less energy is consumed in comparison with the case where the computed orientation is read from the DMP or estimated in the RCD. Our solution was to build the RCD interface through an API implemented in the Android-based STB. The kernel source code of the STB was not modified, in order to maximize compatibility with different vendors and to allow easier deployment without any OS changes. The overall system was successfully tested with good user QoE

Acknowledgment

This work is co-financed by European Union, COMPETE, QREN and Fundo Europeu de Desenvolvimento Regional (FEDER), Project HERMES, Co-promotion N° 34149.

References

[Ohnishi 12] T. Ohnishi, N. Katzakis, K. Kiyokawa, and H. Takemura. Virtual interaction surface: Decoupling of interaction and view dimensions for flexible indirect 3D interaction. In *3D User Interfaces (3DUI), 2012 IEEE Symposium on*, pages 113–116, March 2012.

[Rasteiro 15] M. Rasteiro, H. Costelha, L. Bento, and P. Assuncao. Accuracy versus complexity of MARG-based filters for remote control pointing devices. In *Consumer Electronics - Taiwan (ICCE-TW), 2015 IEEE International Conference on*, pages 51–52, June 2015.

[Regupathy 14] R. Regupathy. *Unboxing Android USB: A hands on approach with real world examples*. Apress, May 2014.

[Song 10] M. Song, W. Xiong, and X. Fu. Research on Architecture of Multimedia and Its Design Based on Android. In *Internet Technology and Applications, 2010 International Conference on*, pages 1–4, Aug 2010.

[USB-IF 00] USB-IF. Universal Serial Bus Specification, April 2000.

[USB-IF 01] USB-IF. Device Class Definition for Human Interface Devices (HID), June 2001.

[Yaghmour 13] K. Yaghmour. *Embedded Android*. O'Reilly Media, Inc., 2013.

[Zidek 13] K. Zidek and J. Pitel. Smart 3D pointing device based on MEMS sensor and bluetooth low energy. In *Computational Intelligence in Control and Automation (CICA), 2013 IEEE Symposium on*, pages 207–211, April 2013.