Android 系统智能电视 HAL 层 Sensor 数据传输的一种实现

胡济豪[1]
（上海交通大学软件学院，上海 1130379118）

摘要：随着智能平板电视技术的飞速发展，智能电视的交互设计也在不断改进，其中电视体感交互就是其中的一个方向。当前许多厂商推出的智能电视都不约而同地选择 Android 系统。本文将讨论如何设计并实现 Android 系统下 HAL 层获取外置电视体感遥控器内传感器数据及手机第二屏虚拟传感器的数据。

键词：智能电视，Android 系统，HAL，传感器

THE IMPLEMENTION OF SENSOR HAL IN ANDROID SYSTEM

Hu Jihao
(School of Software, Shanghai Jiaotong University, Shanghai 1130379118)

Abstract: With the rapid development of technology in smart flat-panel TV, smart TV interaction design has been improved steadily. For an Example，The TV somatosensory interaction is one area which is worthy of research. This article will introduces the design and implement of Sensor HAL in Android system to get real sensor data from real devices and virtual sensor data from a mobile phone.

Keywords: Smart TV, Android system, HAL, Sensor

1.引言

智能电视如今已经成为家电市场上的一个焦点，许多厂商推出的智能电视一般选择的都是 Android 操作系统，当然这更多的是因为 Android 的开放特性。在智能电视领域，随着其功能的日渐丰富和强大，人们对电视的使用不仅仅限于接收和观看电视节目。现在，中高端的智能电视一般都支持了体感游戏的功能，比如厂商 TCL、海信、创维，等等都在自己的智能电视平台上集成了体感游戏这些功能模块。还有，现在广电总局推出的 TVOS2.0 系统也把人机交互作为一个卖点推出。市场也已经出了不少的体感游戏，比如 1905 互动出品的赛车类游戏、球类运动游戏、等等，在智能电视上通过外设体感设备进行电视版体感游戏的操控已经成为一种潮流。这些游戏的操控一般是通过集成了传感器芯片的外设遥控器进行完成。Android 系统在 framework 层已经有了比较完备的获取传感器数据的实现，针对各个不同的厂商，他们要做的就是基于自己的硬件设备来完成 HAL 层的实现，这样传感器的数据就能传送给上层应用。本文的目标在于讨论如何设计和实现 Android 系统的 HAL 层，读取遥控器中的传感器数据以及虚拟传感器数据。

2.Android 系统 Sensor 数据传输的框架分析

2.1Android 系统整体架构

Android 的系统架构和其他操作系统类似，采用了分层的架构。从架构图（图 2.1.1）来

[1]作者简介：胡济豪（1985.6-）男，工程硕士，研究方向：软件工程

看，Android 分为四个层，从高层到低层分别是应用程序层、应用程序框架层、系统运行库层和 Linux 核心层[1]。

(1) 应用程序

所有的应用程序都是使用 java 语言编写的，每一个应用程序由一个或者多个活动组成，

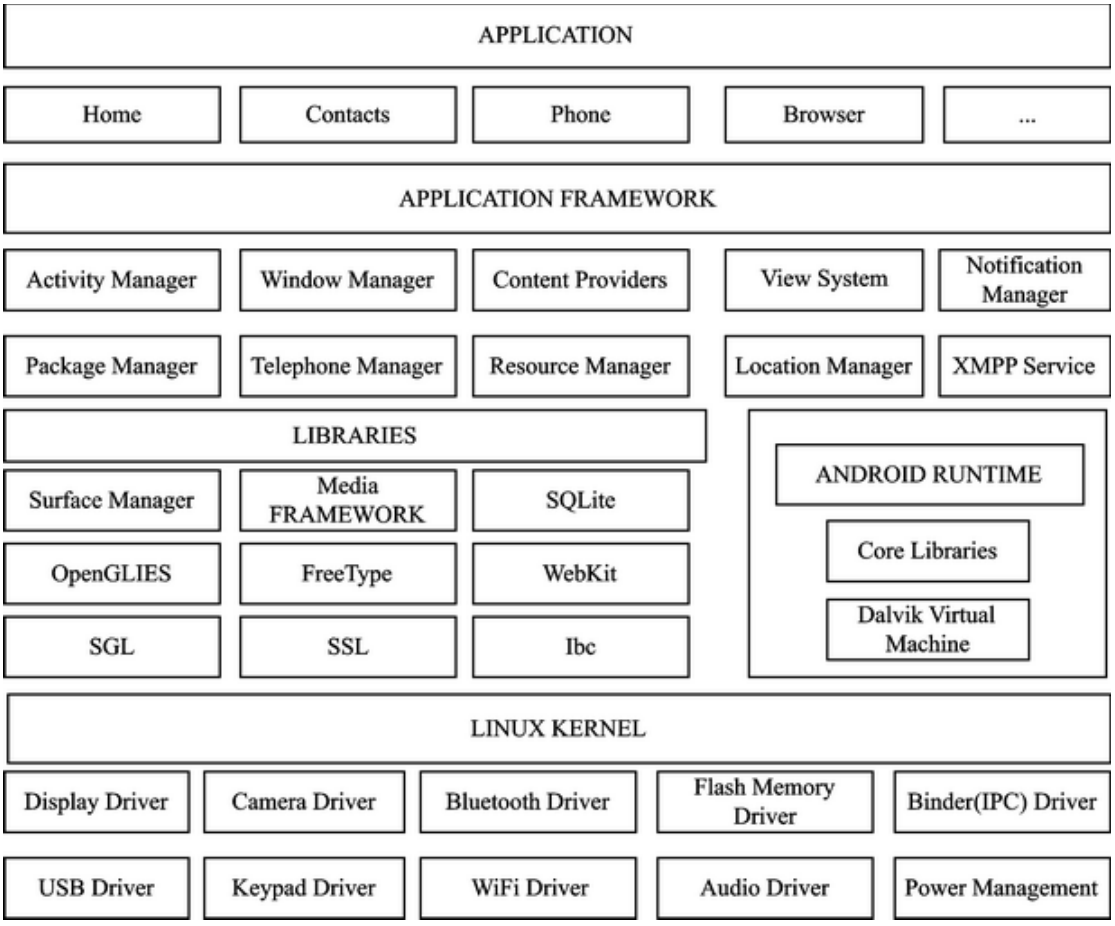

图 2.1.1 Android 系统整体架构图

活动必须以 Activity 类为超类，活动类似于操作系统上的进程，但是活动比操作系统的进程要更为灵活。与进程类似的是，活动在多种状态之间切换。利用 java 的跨平台特性，基于 Android 框架开发的应用程序可以不用编译运行于任何一台安装有 Android 系统的平台，这点正是 Android 的精髓所在。

(2) 应用程序框架

应用程序的架构设计简化了组件的重用；任何一个应用程序都可以发布它的功能块并且其他任何应用程序都可以使用其所发布的功能模块(遵循框架的安全性限制)，帮助程序员快速的开发程序，并且该应用程序重用机制也使用户可以方便的替换程序组件。隐藏在每个应用后面的是一系列的服务和系统，其中包括：丰富而又可扩展的视图（Views），内容提供器（Content Providers），资源管理器（Resource Manager），通知管理器（Notification Manager），活动管理器（Activity Manager），等。

(3) 系统运行库

Android 系统运行库包括程序库（Libraries）和运行时库（Android Runtime）。Android 包含一些 C/C++库，这些库能被 Android 系统中不同的组件使用。他们通过 Android 应用程序框架为开发者提供服务。其核心库主要包括 Bionic 系统 C 库、媒体库、Surface Manager、

基于 WebKit 的浏览器、SGL 2D 图形引擎、3D libraries、FreeType 位图和矢量字体显示、功能强劲的轻型关系型数据库引擎 SQLite，等等。

值得注意的是，这里还有一个硬件抽象层，即所谓的 HAL。其实 Android 并非所有的设备驱动都放在 linux 内核里面，有一部分实现在用户空间，这么做的主要原因是可以避开 Linux 所遵循的 GPL 协议，一般情况下如果要将 Android 系统移植到其他硬件去运行，只需要实现这部分代码即可[2]。包括：显示器驱动，声音，相机，GPS，GSM，Sensor 等等。在后边的章节中要讨论的 Sensor HAL 层的实现变是在这一层。

Android 包括了一个核心库，该核心库提供了 JAVA 编程语言核心库的大多数功能。每一个 Android 应用程序都在它自己的进程中运行，都拥有一个独立的 Dalvik 虚拟机实例。Dalvik 被设计成一个设备可以同时高效地运行多个虚拟系统。 Dalvik 虚拟机执行(.dex)的 Dalvik 可执行文件，该格式文件针对小内存使用做了优化。同时虚拟机是基于寄存器的，所有的类都经由 JAVA 编译器编译，然后通过 SDK 中的工具转化成.dex 格式由虚拟机执行。Dalvik 虚拟机依赖于 linux 内核的一些功能，比如线程机制和底层内存管理机制。

(4) Linux 内核

Android 的核心系统服务依赖于 Linux 内核，如安全性，内存管理，进程管理，网络协议栈和驱动模型。Linux 内核也同时作为硬件和软件栈之间的抽象层。其还对其做了部分修改，主要设计两部分修改：另外实现了一套进程间通讯机制 Binder 和为手持设别的电源能耗做了较好的电源管理[3]。

2.2Android Sensor 架构

现在的 Android 系统内置对传感器的支持已经很多，常见的是：加速度传感器（accelerometer）、磁力传感器（magnetic field）、方向传感器（orientation）、陀螺仪（gyroscope）、环境光照传感器（light）、压力传感器（pressure）、温度传感器（temperature）和距离传感器（proximity）等等[4]。Android 实现传感器系统包括以下几个部分（如表 2.2.1 所示）：

类别	名称	代码
用户空间	Java 应用程序	用户实现
	Java framework 框架层	SensorManager.java SensorListener.java SensorEvent.java …
	JNI 层	android_hardware_SensorManager.cpp com_android_server_SensorService.cpp
	HAL 硬件抽象层	用户实现(sensor.c)
内核空间	设备驱动程序	用户实现
具体硬件	加速度传感器、陀螺仪传感器、压力传感器，等等	用户实现

表 2.2.1 Android Sensor 代码空间划分

Java 应用程序指的是最上层的 app，比如一些体感游戏等等，一般由第三方公司开发，运行于 Android 平台，通过 SDK 提供的 framework 层 Java 接口和系统交互；Java framework 框架层为 app 提供了所需要的系统接口调用，app 可以通过这些接口获取所支持的 Sensor 列表，以及实时的 Sensor 数据，比如获取系统服务 getSystemService、获得传感器对象

getDefaultSensor, 注册监听函数 registerListener；JNI 是 Java 程序调用 C/C++接口的衔接部分；HAL 硬件抽象层屏蔽了 Android 系统对驱动细节的具体的依赖。设备驱动程序和硬件部分由不同的厂商根据自己的产品而定。

各部分之间的层次结构如下图（图 2.2.1）所示。

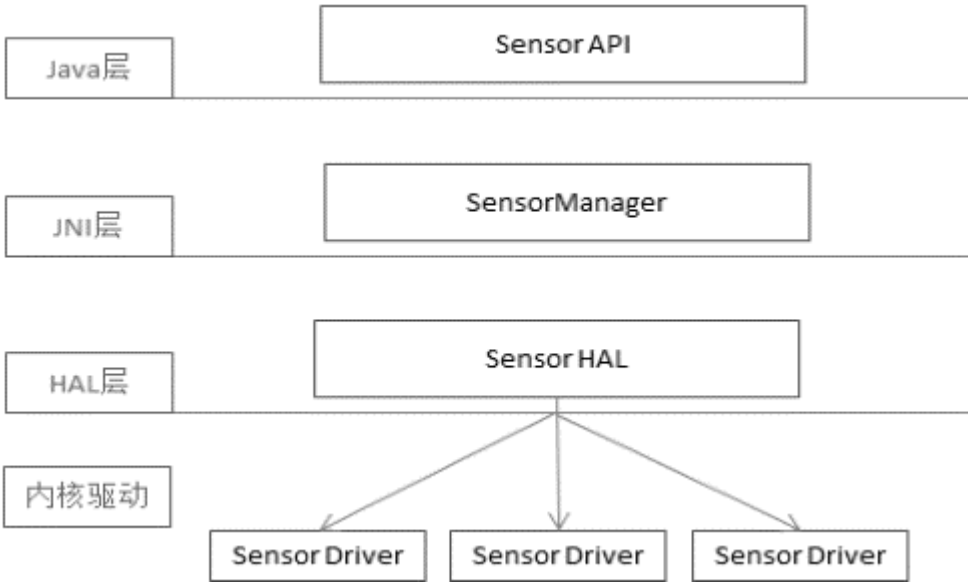

图 2.2.1 Android Sensor 层次结构图

和 Android 系统相关的 Sensor 框架主要分为三个层次：客户端、服务端、HAL 层，这三个层次的总体调用关系图如图 2.2.2 所示。服务端负责从 HAL 读取数据，并将数据写到管道中，客户端通过管道读取服务端数据。

客户端主要类：

SensorManager.java 从 Android4.1 开始，把 SensorManager 定义为一个抽象类，定义了一些主要的方法，该类主要是应用层直接使用的类，提供给应用层的接口；

SystemSensorManager.java 继承于 SensorManager，客户端消息处理的实体，应用程序通过获取其实例，并注册监听接口，获取 Sensor 数据；

SensorEventListener 接口用于注册监听 sensor 事件；

SensorThread 是 SystemSensorManager 的内部类，开启一个新线程负责读取 sensor 数据，当注册了 SensorEventListener 接口的时候才会启动线程；

android_hardware_SensorManager.cpp 是负责与 java 层通信的 JNI 接口；

SensorManager.cpp 是 sensor 在 Native 层的客户端，负责与服务端 SensorService.cpp 的通信；

SensorEventQueue.cpp 是消息队列。

服务端主要类：

SensorService.cpp 服务端数据处理中心；

SensorEventConnection 从 BnSensorEventConnection 继承来，实现接口 ISensorEventConnection 的一些方法，ISensorEventConnection 在 SensorEventQueue 会保存一个指针，指向调用服务接口创建的 SensorEventConnection 对象；

Bittube.cpp 在这个类中创建了管道，用于服务端与客户端读写数据；

SensorDevice 负责与 HAL 读取数据

HAL 层：

Sensor.h 是 google 为 Sensor 定义的 HAL 接口。我们需要探讨的就是 HAL 接口的实现，具体的将在后边进行讨论，下边先关注一下服务端获取 Sensor 数据的调用时序图，如图 2.2.3 所示。在 SystemServer 进程中的 main 函数中，通过 JNI 调到 com_android_server_SystemServer.cpp 的 android_server_SystemServer_init1() 方法，该方法又调用 system_init.cpp 中的 system_init()。SensorService 创建完之后，将会调用 SensorService::onFirstRef()方法，在该方法中完成初始化工作。首先获取 SensorDevice 实例，在其构造函数中，完成了对 Sensor 模块 HAL 初始化。在这里主要做了三个工作：调用 HAL 层的 hw_get_module()方法，加载 Sensor 模块 so 文件；调用 sensor.h 的 sensor_open 方法打开设备；调用 sensor_poll_device_t 的 activate()对 Sensor 模块使能。

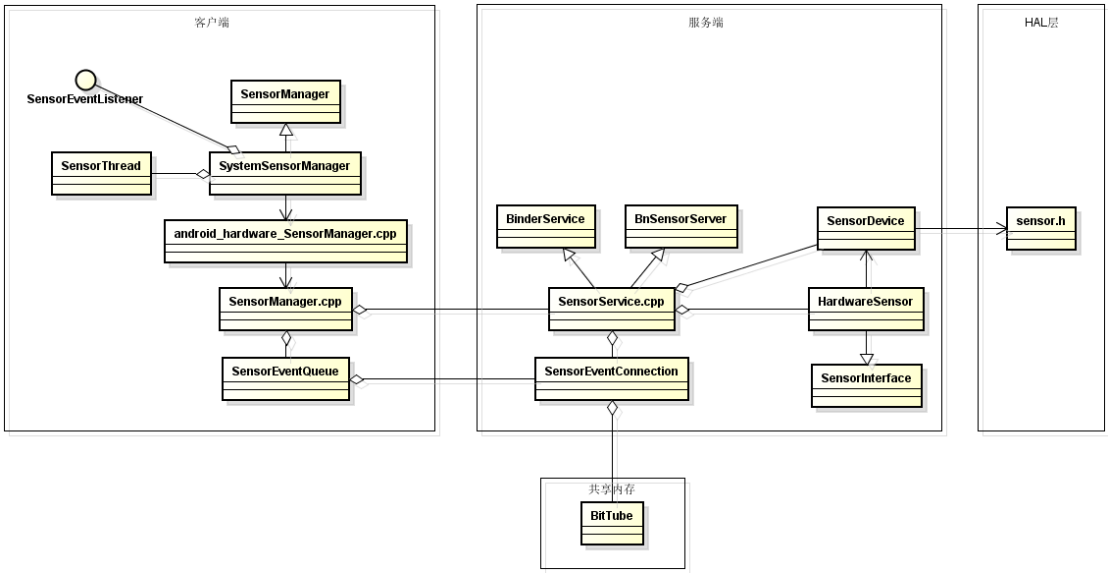

图 2.2.2 Android Sensor 层次类图

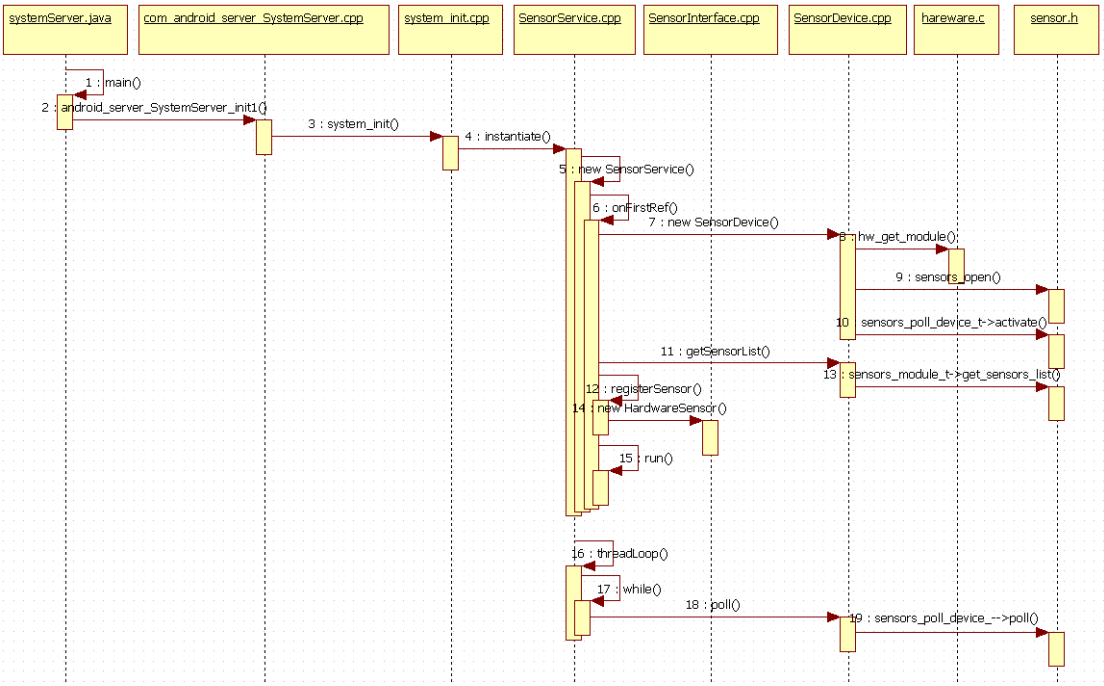

图 2.2.3 Android Sensor 服务端调用 Sequence

2.3 Sensor HAL 分析

HAL 存在两种架构[5]：旧的 HAL 架构（位于 libhardware_legacy 目录）和新的 HAL 架构（位于 libhardware 目录）。

过去的 libhardware_legacy，将*.so 当做 shared library 来使用，在 runtime(JNI)部分以 direct function call 使用 HAL module。通过直接函数调用的方式，来操作驱动程序。当然，应用程序也可以不需要透过 JNI 的方式进行，直接以加载动态库*.so（dlopen）的做法来调用动态库里的符号(symbol)也是一种方式。之而言之是没有经过封装，上层可以直接操作硬件。如下图（图 2.3.1 所示）。

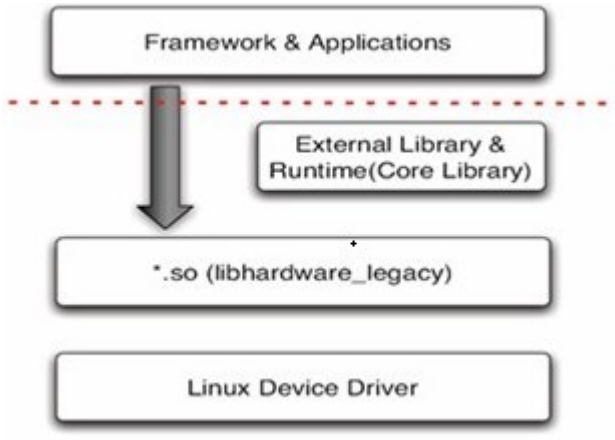

图 2.3.1 旧 HAL 框架

现在的 libhardware 采用 stub 做法，如图 2.3.2 所示。HAL stub 是一种代理人（proxy）的概念，stub 虽然仍是以*.so 的形式存在，但 HAL 已经将*.so 隐藏起来了。Stub 向 HAL 提供操作函数（operations），而 runtime 则是向 HAL 取得特定模块（stub）的 operations，再 callback 这些函数。这种以 indirect function call 的框架，让 HAL Stub 变成是一种包含关系，即 HAL 里包含了许许多多的 stub（代理人）。Runtime 只要说明“类型”，即 module ID，就可以取得操作函数。对于目前的 HAL，可以认为 Android 定义了 HAL 层结构框架，通过几个接口访问硬件从而统一了调用方式。

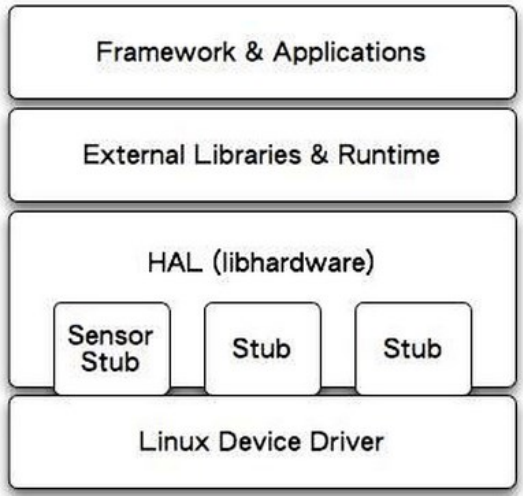

图 2.3.2 新 HAL 框架

HAL_legacy 和 HAL 对比：

HAL_legacy：旧式的 HAL 是一个模块，采用共享库形式，在编译时会调用到。由于采用 function call 形式调用，因此可被多个进程调用，但会被 mapping 到过个进程空间中，造

成浪费，同时需要考虑代码能否安短重入的问题（thread safe）。

HAL：新式的 HAL 采用 HAL module 和 HAL stub 结合形式，HAL stub 不是一个 share library，编译时上层只拥有访问 HAL stub 的函数指针，并不需要 HAL stub。上层通过 HAL module 提供的统一接口获取并操作 HAL stub，so 文件只会被 mapping 到一个进程，也不存在重复 mapping 和重入问题。

3. Sensor HAL 的一种实现

通过上面章节的分析，我们对 Android 系统下的 Sensor HAL 整体架构有了比较系统的了解。下边，我们将讨论和具体需求相关的 HAL 层的一种实现。

从需求来看，第一是需要实现从外部硬件设备中读取真实 Sensor 的数据，第二是需要开辟一个通道将手机第二屏 Sensor 的虚拟数据传输给 Android 系统。

3.1 获取真实 Sensor 数据

现在市场面上有许多的体感遥控器是通过 2.4G 技术进行无线连接与数据传输，在主机接收端插入一个类似 USB 端口的接收器，遥感数据可以从此 USB 接收端读出，图 3.1.1 是此类产品的一个外观图。我们将讨论此类产品 Sensor 数据的读取。

图 3.1.1 2.4G 无线遥控器产品示意图

将此设备的 USB 端子插入 Android 系统的智能电视 USB 接口，在 dev 下可以看到多出多出 hidraw 设备。

我们可以通过调用系统函数 read()读取 hidraw 的原始数据，将其转换成 Android Sensor 说需要的数据结构即可，同时为了更好的兼容外设体感遥控器的 Sensor 数据，我们定义一套标准的数据结构以进行规格适配，如图 3.1.2 所示。

											Accelerometer Data						Gyroscope Data														
											x		y		z		azimuth		pitch		roll								
											L	H	L	H	L	H	L	H	L	H	L	H							
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
x	x	x	x	x	x	x	x	x	x	x	90	0	e0	0	30	fe	80	0	80	ff	80	0	x	x	x	x	x	x	x	x	x
x	x	x	x	x	x	x	x	x	x	x	d0	0	20	1	a0	fe	20	0	0	0	60	0	x	x	x	x	x	x	x	x	x
x	x	x	x	x	x	x	x	x	x	x	c0	0	70	1	10	ff	e0	ff	a0	0	a0	0	x	x	x	x	x	x	x	x	x
x	x	x	x	x	x	x	x	x	x	x	90	0	e0	1	40	ff	e0	fe	a0	0	c0	0	x	x	x	x	x	x	x	x	x
x	x	x	x	x	x	x	x	x	x	x	40	fe	f0	0	70	0	e0	fe	0	ff	20	0	x	x	x	x	x	x	x	x	x
x	x	x	x	x	x	x	x	x	x	x	f0	fd	90	ff	30	ff	80	0	80	ff	40	0	x	x	x	x	x	x	x	x	x
x	x	x	x	x	x	x	x	x	x	x	50	fe	a0	ff	f0	fe	e0	ff	40	ff	80	0	x	x	x	x	x	x	x	x	x
x	x	x	x	x	x	x	x	x	x	x	60	fe	b0	ff	e0	fe	e0	ff	40	ff	a0	0	x	x	x	x	x	x	x	x	x

图 3.1.2 Sensor 数据格式

在图 3.1.2 中，每一行所示的是从 hidraw 读出的原始数据，每次读出数据 32 个字节，data[11~16]表示加速度传感器数据，data[17~22]表示陀螺仪数据。在加速度传感器数据中，

x 轴数据用[11][12]表示；其中[11]表示低字节，[12]表示高字节；y 轴数据用[13][14]表示；其中[13]表示低字节，[14]表示高字节；z 轴数据用[15][16]表示；其中[15]表示低字节，[16]表示高字节。其他类似。对于读出的原始数据，我们将其乘以一个系数，转换成合适的数据类型被 Android 读取。此系数可根据需求自行进项调整。提供参考如下：

acceleration.x = 原始数据 * CONVERT_A_X;

其中 CONVERT_A_X = (((9.80665f) / (64.0f)/4))

现在，还有一个问题没有解决：打开 dev 可以看到 dev 下不止一个 hidraw 设备，我们从哪个 hidraw 读取数据呢？这里就涉及到 USB HID。USB HID （Human Interface Device，人机接口设备）是 USB 设备中常用的设备类型，是直接与人交互的 USB 设备，例如键盘鼠标与游戏杆等。在 USB 设备中，HID 设备的成本较低。另外，HID 设备并不一定要有人机交互功能，只要符合 HID 类别规范的设备都是 HID 设备。使用 HID 设备的一个好处就是，操作系统自带了 HID 类的驱动程序，而用户无需去开发复杂的驱动程序，只要直接使用 API 调用即可完成通信。所以很所简单的 USB 设备喜欢枚举成 HID 设备，这样就可以不用安装驱动直接使用。HID 协议的详解可参看官网，此处不做深入讨论。

为了便于设计上的通用性，我们采用读报告描述符的方法来确定需要的 hidraw，像定义 Sensor 数据格式那样，我们定义一个报告描述符，只要 HID 设备按照该报告描述符发送数据，我们就认为它是我们要读取的 hidraw 设备。报告描述符的定义如图 3.1.3 所示。

```
{
    0x06,0xA0,0xFF,//用法页(vendor defined)
    0x09, 0x01,//用法(vendor defined)
    0xA1, 0x01,//集合(Application)
    0x09, 0x03 ,//用法(vendor defined)
    0x85, 0x04
    0x15, 0x80,//逻辑最小值(0x80 or -128)
    0x25, 0x7F,//逻辑最大值(0x7F or 127)
    0x35, 0x00,//物理最小值(0)
    0x45,0xFF,//物理最大值(255)
    0x75, 0x08,//报告长度Report size (8位)
    0x95, 0x1f,//报告数值(31 fields)
    0x81, 0x02,//输入(data, variable, absolute)
    0x09, 0x05,//用法(vendor defined)
    0x85, 0x05,
    0x15, 0x80,//逻辑最小值(0x80 or -128)
    0x25, 0x7F,//逻辑最大值(0x7F or 127)
    0x35, 0x00,//物理最小值(0)
    0x45,0xFF,//物理最大值(255)
    0x75,0x08,//报告长度(8位)
    0x95, 0x07,//报告数值(7 fields)
    0x91, 0x02,//输出(data, variable, absolute)
    0xC0,//集合结束(Physical)
};
```

图 3.1.3 HID 报告描述符定义

我们解决了 Sensor 数据格式的定义以及读取数据的方法，至此，通过编码实现和调试，就可以实现读取真实 Sensor 数据了。

一般来说，当前智能电视中体感游戏说需要的传感器主要包括加速度传感器和陀螺仪传感器，底层支持了这两个传感器，就满足了当前市场上大多数的体感游戏对传感器数据的要求。

3.2 获取虚拟 Sensor 数据

上边讨论了如何获取真实 Sensor 的数据，下边看第二个需求：获取虚拟 Sensor 的数据。虚拟 Sensor 的数据来源于手机第二屏遥控器，这里解释下“手机第二屏遥控器”的含义，手机第二屏遥控器指手机或 pad 上安装相应的遥控 app 软件，将手机模拟成电视/机顶盒的软遥控器。

我们要做的是提供相应的接口，便于 app 调用此接口写入 Sensor 数据到系统。为此，我们创建一个命名管道，Sensor HAL 从命名管道中读取虚拟 Sensor 数据，app 侧调用接口写入虚拟 Sensor 数据。在读取端，随系统启动的时候 Sensor HAL 调用 mkfifo 在系统目录 data 下创建一个命名管道，然后调用 open 打开将其做为读入端，接着调用 read 和 poll 读取和等待数据。在 app 侧，我们提供数据写入的接口，主要包括三个：openSensor()，writeData(const sensor_event_t *data)，和 closeSensor()。在 openSensor 中我们打开命名管道作为写入端，在 writeData 中进行数据的写入，其数据结构采用 android 原生的结构体 sensor_event_t，最后在数据写入完毕的时候调用 closeSensor 进行关闭。

手机第二屏遥控器在使用的时候将从本机获取的 Sensor 数据或者模拟出的虚拟数据转换成 sensor_event_t，直接调用写入接口即可。

此项设计及实现的方案并不复杂，主要涉及的是命名管道的使用，及注意写入时的同步操作。

4. 测试与验证

测试与验证，可采取操控体感游戏的方式进行，比如赛车游戏及体感运动。本文测试过的游戏包括两款：“真实赛车”和“运动加加”中的乒乓球。下边简单描述了真实赛车的测试流程和结果。

4.1 真实 Sensor 数据

插入体感遥控器 USB 端子到智能电视 USB 端口，安装测试游戏“真实赛车”，打开真实赛车游戏，挥动体感手柄，可以看到赛车可以流畅地进行左转右转，并且游戏操作体验流畅。

4.2 虚拟 Sensor 数据

安装，并打开“真实赛车”游戏，接入手机第二屏遥控器 app，挥动手机进行控制，可以发现赛车已经被操控随手机倾斜而左转右转。

以下（图 4.2.1）是游戏过程中操控不同车型在不同场地比赛的截图。

图 4.2.1 测试画面截图

5. 结束语[2]

本文较为系统的阐述了 Android 系统下 Sensor 数据传输的框架和流程，设计和实现了 Sensor 数据传输的一种方案，解决了真实 Sensor 数据和虚拟 Sensor 数据的传输需求。在文中提供了 Sensor 数据结构和 HID 报告描述符的标准定义，任何厂家的体感设备只要符合该标准，都可以进行良好的兼容与适配。

当然，对于 Android HAL 层的实现方案不止一种，各个厂商可根据自己的需求进行定制实现，本文只是提供了一个方式进行讨论。此方案也存在某些不足，比如同时连接多个遥控器操作，这是本文没有深究的，可作为以后的课题深入探讨。

最后，向此论文完成过程中给予支持和鼓励的学校导师，公司领导、同仁和各位同学表示感谢，谢谢你们耐心的指导和中肯的建议。

参考文献

[1] 汪永松,Android 平台开发之旅(第 2 版)[M]，北京，机械工业出版社，2012，10-13

[2] 王振丽，Android 底层开发技术实战详解：内核、移植和驱动[M]，北京，电子工业出版社，2012，84-98

[3] 苏健，Android 智能手机平台电源管理[J].微机处理，2011(5)：66-69

[4] Google Inc，Sensors Overview[EB/OL]. http://developer.android.com/guide/topics/sensors/sensors_overview.html，2016-3-20

[5] 罗升阳, Android 系统源代码情景分析[D]，北京，电子工业出版社，2016，13-43

[2]作者联系方式：邮箱 hujihaoxinxiang@163.com 电话 13524535860