

**信息与软件工程学院**

**综合设计II中期报告**

课程名称： 综合设计II

课题名称： 基于音频的安全D2D通信系统

指导教师： 陈大江

所在系别： 互联网安全

执行学期： 4

学生信息：

|  |  |  |
| --- | --- | --- |
| 序号 | 学号 | 姓名 |
| 1（组长） | 2019091616012 | 冉江皓 |
| 2 | 2019091616018 | 邓亚辉 |
| 3 | 2019091616007 | 官宏 |
| 4 | 2019091603029 | 冯一凡 |
| 5 | 2019091603009 | 罗驰宇 |

目 录

第一章 综合设计的进展情况 1

1.1 针对工程问题的方案设计 1

1.2 针对工程问题的推理分析 1

1.3 针对工程问题的具体实现 1

1.4 知识技能学习情况 1

第二章 存在问题与解决方案 2

2.1 存在的主要问题 2

2.2 解决方案 2

第三章 前期任务完成度与后续实施计划 3

参考文献 4

# 第一章 综合设计的进展情况

针对工程问题的方案设计

本期的复杂软件工程问题是开发用于检测移动的软件。具体涉及到的问题有对物体移动的检测、对检测数据的汇报以及服务器通信的加密等等。总体结构设计上，我们采用客户端-服务器模式，分别开发用于检测的客户端和用于监控、接收检测动态的客户端，由服务器存储数据并负责和各客户端之间的通信。

具体详细设计上，检测客户端利用智能手机内置的各种传感器实现运动检测；接收检测动态的客户端和服务器建立长连接接收推送；服务端使用Socket进行和各客户端之间的通信和数据交换。

针对工程问题的推理分析

检测物体移动方面，需要实现对于物体移动的检测有多种方法，既包括使用硬件的UWB技术、陀螺仪、红外、磁力等传感器，也可以使用基于图像的识别等方法。考虑到不同手机配备的传感器存在差异，而图像成像素质也各有不同，需要找到一个能适应绝大部分情况并且拥有良好和稳定检测率的检测方法。同时，可以考虑在耗电、算法效率等多方面进行优化。

与服务器通信方面，监测用的客户端不必一直和服务器保持连接，而只需要在检测到相应状态时向服务器发起通信。同时，为了保护相应动态免受可能的网络攻击，需要对通信进行一定的加密保护。选用低延时的通信协议和高性能的加密方法也是需求的一部分。

服务器架构设计方面，考虑到客户端的数量不固定，需要为服务器的可扩展性进行相应的设计。同时，由于需要并发处理需求，期间的数据库操作、死锁预防以及并发处理的性能都需要各种优化。对此，根据实际规模考虑是否要对服务端进一步进行分布式设计、负载均衡等以优化实际性能。后端开发和检测软件既可以采用相同的语言，也可以采用有更加适合满足解决实际问题需要的后端设计特性的语言。

监控客户端方面，涉及到对数据的定期更新甚至实时刷新，所以应该考虑与服务器长期保持一个活跃连接的形式，并且服务器与客户端之间通过管道进行通信和状态推送。有关如何实现这种长连接的技术也有很多，以及推送相关机制的实现和改进。应该考虑在长时间使用时，如何保持消息的及时性的同时尽可能降低能耗，防止监控客户端耗电过快从而导致监控失效等情况。

针对工程问题的具体实现

检测物体移动方面：

时至今日，手机已经成为我们生活中必不可少的一种物品。人们使用手机来进行信息交流、娱乐休闲、购物出行等等，这是因为手机的功能不断在扩展，手机的性能在不断的进步，才使得我们获得如此强大的手机功能。现在的手机，一般都会配备有各种各样的传感器。

​手机传感器是手机内部设置的感应器，一般专门指的是安卓手机中的可用感应器。能够通过感应光照度、热量、距离等，来调节手机的工作状态，从而使人们能够更加方便地使用手机。一些比较高档的手机，常常会配备有光纤传感器、热力传感器、温度传感器等等，这些传感器可以感应到手机使用者的使用状态的变化，从而使手机做出自动的调整，不需要人们去进行手动的、人为的调整，这是非常方便的，而且这些调整一般都是在极短的时间之内完成的，不会浪费过多的时间，使人们第一时间感受到方便，这对于使用者来说，是非常有利的。

​用安卓手机接口获取加速度传感器和方向传感器数据，用运动算法得到运动的类型、方向、距离。

具体实现如下：

1. 首先获取传感器的实例并且设置监听：

private SensorManager sensorManager;

private Sensor acc\_sensor;

private Sensor mag\_sensor;

sensorManager = (SensorManager) getSystemService(SENSOR\_SERVICE);

acc\_sensor = sensorManager.getDefaultSensor(Sensor.TYPE\_ORIENTATION);

mag\_sensor = sensorManager.getDefaultSensor(Sensor.TYPE\_ACCELEROMETER);

// 给传感器注册监听：

sensorManager.registerListener(this, acc\_sensor, SensorManager.SENSOR\_DELAY\_GAME);

sensorManager.registerListener(this, mag\_sensor,SensorManager.SENSOR\_DELAY\_GAME);

// 在activity中添加接口

implements SensorEventListener //实现接口函数：

@Override

public void onSensorChanged(SensorEvent event) {

if(event.sensor.getType()==Sensor.TYPE\_ACCELEROMETER){

v\_tmp1 = (float)(Math.round(event.values[0]\*1000)/1000);

v\_tmp2 = (float)(Math.round(event.values[1]\*1000)/1000);

v\_tmp3 = (float)(Math.round(event.values[2]\*1000)/1000);

tv\_axu2.setText("ACCELEROMETER:"+v\_tmp1+","+v\_tmp2+","+v\_tmp3);

}

else if (event.sensor.getType()==Sensor.TYPE\_ORIENTATION){

yaw = (float)(Math.round(event.values[0]\*1000)/1000);

pitch =(~Math.round(event.values[1]\*1000))/1000 ;

mRoll =(~Math.round(event.values[2]\*1000))/1000 ;

yaw = (float) (yaw/57.3);

pitch = (float) (pitch/57.3);

mRoll = (float) (mRoll/57.3);

}

double cosx,sinx,cosy,siny,cosz,sinz;

cosx= Math.cos(mRoll);

sinx = Math.sin(mRoll);

cosy = Math.cos(pitch);

siny = Math.sin(pitch);

cosz = Math.cos(yaw);

sinz = Math.sin(yaw);

double mat0,mat1,mat2,mat3,mat4,mat5,mat6,mat7,mat8;

mat0 = cosz \* cosy;

mat1 = -cosy \* sinz;

mat2 = siny;

mat3 = sinz\*cosx + (cosz\*sinx \* siny);

mat4 = cosz\*cosx - (sinz\*sinx \* siny);

mat5 = -sinx \* cosy;

mat6 = (sinz\*sinx) - (cosz\*cosx \* siny);

mat7 = (cosz\*sinx) + (sinz\*cosx \* siny);

mat8 = cosy \* cosx;

double X,Y,Z;

X = v\_tmp1 \* mat0 + v\_tmp2 \* mat3 + v\_tmp3 \* mat6;

Y = v\_tmp1 \* mat1 + v\_tmp2 \* mat4 + v\_tmp3 \* mat7;

Z = v\_tmp1 \* mat2 + v\_tmp2 \* mat5 + v\_tmp3 \* mat8;

tv\_axu1.setText("v->X:"+ X +" v->Y:"+Y +" v->Z:"+Z);

// 代码中的转换跟变量名比较多，都是为了方便区分跟获取小数点后三位而做的一些处理。大致的意思是：

if(event.sensor.getType()==Sensor.TYPE\_ACCELEROMETER){

v\_tmp1 = (float)(Math.round(event.values[0]\*1000)/1000);

v\_tmp2 = (float)(Math.round(event.values[1]\*1000)/1000);

v\_tmp3 = (float)(Math.round(event.values[2]\*1000)/1000);

tv\_axu2.setText("ACCELEROMETER:"+v\_tmp1+","+v\_tmp2+","+v\_tmp3);

}

// 判断获取的是手机加速度传感器的值，取值小数点后三位并且转换为float类型。并且在界面上显示出来。

else if (event.sensor.getType()==Sensor.TYPE\_ORIENTATION){

yaw = (float)(Math.round(event.values[0]\*1000)/1000);

pitch =(~Math.round(event.values[1]\*1000))/1000 ;

mRoll =(~Math.round(event.values[2]\*1000))/1000 ;

yaw = (float) (yaw/57.3);

pitch = (float) (pitch/57.3);

mRoll = (float) (mRoll/57.3);

}

判断获取方向传感器，同样取值小数点后三位并且转化为float类型，pitch与mRoll在这里需要取反，因为运动算法中需要用到，并且将得到的数值除以57.3，得到弧度制，同样是为了方便算法的运算。

与服务器通信方面：采用socket通信

这是为了实现以上的通信过程而建立成来的通信管道，其真实的代表是客户端和服务器端的一个通信进程，双方进程通过socket进行通信，而通信的规则采用指定的协议。socket只是一种连接模式，不是协议，socket是对TCP/IP协议的封装，Socket本身并不是协议，而是一个调用接口（API），

​通过Socket，我们才能使用TCP/IP协议。tcp、udp，简单的说（虽然不准确）是两个最基本的协议,很多其它协议都是基于这两个协议如，http就是基于tcp的，.用socket可以创建tcp连接，也可以创建udp连接，这意味着，用socket可以创建任何协议的连接，因为其它协议都是基于此的。

Socket 传输的特点：

优点

1) 传输数据为字节级，传输数据可自定义，数据量小（对于手机应用讲：费用低）

2）传输数据时间短，性能高

3）适合于客户端和服务器端之间信息实时交互

4）可以加密，数据安全性强

服务器架构设计方面：采用Java编写

​MyServerSocket:

public class MyServerSocket {

public static void main(String[] args) throws IOException {

new ServerListener().start();

}

}

​ServerListener:

public class ServerListener extends Thread {

@Override

public void run(){

try {

ServerSocket serverSocket=new ServerSocket(12345);

while (true){

// bliock

Socket socket = serverSocket.accept();

// 建立连接

JOptionPane.showMessageDialog(null,"有客户端链接到了本地12345端口");

// 传递socket

ChatSocket cs = new ChatSocket(socket);

cs.start();

ChatManager.getChatManager().add(cs);

}

} catch (IOException e) {

e.printStackTrace();

}

}

}

​ChatManager:

public class ChatManager {

private ChatManager(){}

private static final ChatManager cm = new ChatManager();

public static ChatManager getChatManager(){

return cm;

}

Vector<ChatSocket> vector = new Vector<ChatSocket>();

public void add(ChatSocket cs){

vector.add(cs);

}

// 发送给其他客户

public void publish(ChatSocket cs,String out) throws IOException {

for(int i=0;i< vector.size();i++){

ChatSocket csChatSocket =vector.get(i);

if(!cs.equals(csChatSocket)){

csChatSocket.out(out);

}

}

}

}

​ChatSocket:

public class ChatSocket extends Thread{

Socket socket;

public ChatSocket(Socket s){

this.socket=s;

}

public void out(String out) throws IOException {

socket.getOutputStream().write(out.getBytes("UTF-8"));

}

@Override

public void run(){

try {

BufferedReader br = new BufferedReader(

new InputStreamReader(

socket.getInputStream(),"UTF-8"));

String Line =null;

while ((Line = br.readLine())!=null){

ChatManager.getChatManager().publish(this,Line);

}

br.close();

} catch (IOException e) {

e.printStackTrace();

}

}

}

监控客户端方面，采用与客户端类似的方法，使用socket完成和服务器之间的通信。

知识技能学习情况

要完成本期的复杂工程问题，需要对于Java编程相关的知识有较好的学习和使用。目前我们已经学习了Java中对于传感器的使用，以及服务端建立Socket连接的方法。Java技术栈能较好满足我们的需求，当然我们也可以根据实际需求考虑转向Java的衍生语言如Kotlin、Dart等。

# 第二章 存在问题与解决方案

2.1 存在的主要问题

目前，对于基于Socket的通讯方案细节仍未敲定，主要是有关服务端和客户端的推送相关具体细节，如通讯频率、通讯内容等。

对于移动检测，如果检测效果不佳，可能需要进行算法调优或者更换算法，甚至是更换检测原理的措施。

2.2 解决方案

尽快确定Socket通讯中各方互相通讯时需要的信息内容，设计规范的回报，确保覆盖内容的同时留有一定扩展性。

对移动检测进行测试，如果出现效果不佳的情况就进行相应的修改措施。

# 第三章 前期任务完成度与后续实施计划

根据对问题的分解和各方向的分工开发，前期团队中按照服务端、监控客户端、检测客户端的方向分别进行了相应的研究和讨论，同时进行了功能的开发。目前各个方向的开发已经具备雏形，主要功能有了大致的实现。接下来的计划是按照之前对问题的分析，进行完整开发或者进行一定取舍，根据实际情况对预期功能进行一定调整。

# 参考文献