# Android计步器算法

采用重力加速计来实现计步功能，本文在此介绍的就是这种实现方式。

1.我们使用如下代码来检测手机是否支持系统自带的计步传感器：

// 获取传感器管理器的实例

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

PackageManager pm=getPackageManager();

//检查手机是否支持计步传感器

if(pm.hasSystemFeature(PackageManager.FEATURE\_SENSOR\_STEP\_COUNTER)){

stepCounterListener();

}else {

stepAccelerometerListener();

}

1.使用这种方式来判断比较安全，如果是用系统版本号来判断不保险，因为我试过在有些安卓5.0的机子上也不存在计步传感器。

2.下面开始介绍我们重力加速计的实现方式：

import android.hardware.Sensor;import android.hardware.SensorEvent;import android.hardware.SensorEventListener;

public class StepDetector implements SensorEventListener {//重力感应器接口

private StepCountListener mStepListeners;

//存放三轴数据

float[] oriValues = new float[3];

final int valueNum = 4;

//用于存放计算阈值的波峰波谷差值

float[] tempValue = new float[valueNum];

int tempCount = 0;

//是否上升的标志位

boolean isDirectionUp = false;

//持续上升次数

int continueUpCount = 0;

//上一点的持续上升的次数，为了记录波峰的上升次数

int continueUpFormerCount = 0;

//上一点的状态，上升还是下降

boolean lastStatus = false;

//波峰值

float peakOfWave = 0;

//波谷值

float valleyOfWave = 0;

//此次波峰的时间

long timeOfThisPeak = 0;

//上次波峰的时间

long timeOfLastPeak = 0;

//当前的时间

long timeOfNow = 0;

//当前传感器的值

float gravityNew = 0;

//上次传感器的值

float gravityOld = 0;

//动态阈值需要动态的数据，这个值用于这些动态数据的阈值

final float initialValue = (float) 1.3;

//初始阈值

float ThreadValue = (float) 1.8;

public StepDetector() {

} public void setStepListeners(StepCountListener sl) {

mStepListeners = sl;

}

@Override

public void onSensorChanged(SensorEvent event) {

for(int i = 0; i < 3; i++) {

oriValues[i] = event.values[i];//重力感应器传回的数据

}

gravityNew = (float) Math.sqrt(oriValues[0] \* oriValues[0]

+ oriValues[1] \* oriValues[1] + oriValues[2] \* oriValues[2]);

DetectorNewStep(gravityNew);

}/\*

\* 检测步子，并开始计步

\* 1.传入sersor中的数据

\* 2.如果检测到了波峰，并且符合时间差以及阈值的条件，则判定为1步

\* 3.符合时间差条件，波峰波谷差值大于initialValue，则将该差值纳入阈值的计算中

\* \*/

public void DetectorNewStep(float values) {

if (gravityOld == 0) {//上次传感器的值为0

gravityOld = values;//上次传感器的值等于本次传感器的值

} else {

if (DetectorPeak(values, gravityOld)) {//如果到达波峰

timeOfLastPeak = timeOfThisPeak;//更新上次到达波峰的时间

timeOfNow = System.currentTimeMillis();//获取当前时间

if (timeOfNow - timeOfLastPeak >= 250时间差

&& (peakOfWave - valleyOfWave >= ThreadValue)) {//波峰波谷高度差

timeOfThisPeak = timeOfNow;更新本次波峰时间

/\*

\* 更新界面的处理，不涉及到算法

\* \*/

mStepListeners.onStep();

}

if (timeOfNow - timeOfLastPeak >= 250//如果波峰减去波谷的值大于初始阀值则重新计算阀值

&& (peakOfWave - valleyOfWave >= initialValue)) {

timeOfThisPeak = timeOfNow;

ThreadValue = Peak\_Valley\_Thread(peakOfWave - valleyOfWave);

}

}

}

gravityOld = values;

}

/\*

\* 检测波峰

\* 以下四个条件判断为波峰：

\* 1.目前点为下降的趋势：isDirectionUp为false

\* 2.之前的点为上升的趋势：lastStatus为true

\* 3.到波峰为止，持续上升大于等于2次

\* 4.波峰值大于20

\* 记录波谷值

\* 1.观察波形图，可以发现在出现步子的地方，波谷的下一个就是波峰，有比较明显的特征以及差值

\* 2.所以要记录每次的波谷值，为了和下次的波峰做对比

\* \*/

public boolean DetectorPeak(float newValue, float oldValue) {

lastStatus = isDirectionUp;//记录上一次状态是上升还是下降

if (newValue >= oldValue) {

//如果新的值大于旧的值，则表示上升状态

isDirectionUp = true;

continueUpCount++;//记录连续上升次数

} else {

//如果新的值小于旧的值，表示是下降状态

continueUpFormerCount = continueUpCount;//记录了上一次上升的点数

continueUpCount = 0;//初始化上升状态值

isDirectionUp = false;//更改为下降状态

}

//如果是下降状态状太且上一次状态是上升状态并且连续上升次数大于等于2

//表示到了下降状态了 可以算是一次峰谷返回true

if (!isDirectionUp && lastStatus

&& (continueUpFormerCount >= 2 || oldValue >= 20)) {

peakOfWave = oldValue;//记录波峰值

return true;

//如果上一次状态是下降状态且当前状态是上升的则记录波谷值

} else if (!lastStatus && isDirectionUp) {

valleyOfWave = oldValue;

}

return false;

}

/\*

\* 阈值的计算

\* 1.通过波峰波谷的差值计算阈值

\* 2.记录4个值，存入tempValue[]数组中

\* 3.在将数组传入函数averageValue中计算阈值

\* \*/

public float Peak\_Valley\_Thread(float value) {

float tempThread = ThreadValue;

if (tempCount < valueNum) {

tempValue[tempCount] = value;

tempCount++;

} else {

tempThread = averageValue(tempValue, valueNum);

for (int i = 1; i < valueNum; i++) {

tempValue[i - 1] = tempValue[i];

}

tempValue[valueNum - 1] = value;

}

return tempThread;

}

/\*

\* 梯度化阈值

\* 1.计算数组的均值

\* 2.通过均值将阈值梯度化在一个范围里

\* \*/

public float averageValue(float value[], int n) {

float ave = 0;

for (int i = 0; i < n; i++) {

ave += value[i];

}

ave = ave / valueNum;//将数组中的值求一个平均数

if (ave >= 8)

ave = (float) 4.3;

else if (ave >= 7 && ave < 8)

ave = (float) 3.3;

else if (ave >= 4 && ave < 7)

ave = (float) 2.3;

else if (ave >= 3 && ave < 4)

ave = (float) 2.0;

else{

ave = (float) 1.3;

}

return ave;

}

@Override

public void onAccuracyChanged(Sensor arg0, int arg1) {

}

}

3.以上是计步器的核心算法，该算法可以更具实际需求去调整一些参数，相应的注视都写在上面了相信不难理解.该算法 最终得到的是一个步数的变化，具体怎么处理这个变化可以自己释放的去处理。比如下面我们判断连续运动5步才算是开始运动。两次步数间隔超过3秒就作废。

public class StepCount implements StepCountListener {

private int count;

private int mCount;

long timeOfLastPeak;

long timeOfThisPeak;

private StepPaseValueListener mListeners;

@Override

public void onStep() {

if(timeOfLastPeak==0){

timeOfLastPeak = System.currentTimeMillis();

}else{

timeOfLastPeak = timeOfThisPeak;

}

timeOfThisPeak = System.currentTimeMillis();

if ((timeOfThisPeak - timeOfLastPeak) >= 3000) {

count = 1;

} else {

count++;

}

if (count == 5) {

mCount = mCount + count;

if (mListeners != null) {

mListeners.stepsChanged(mCount);

}

return;

}

if (count > 5) {

mCount++;

if (mListeners != null) {

mListeners.stepsChanged(mCount);

}

}

}

public void setmListeners(StepPaseValueListener mListeners) {

this.mListeners = mListeners;

}

}