fAndroid软件

一．实现功能

1.蓝牙设备扫描和连接

2.加速度数字信号接收和显示

3.在坐标图上即时显示加速度

4. 计步

5. 对加速度信号进行FIR滤波，还可以在FIR基础上进行平滑滤波

6. 即时显示滤波后的信号图和步数

二．主要特点

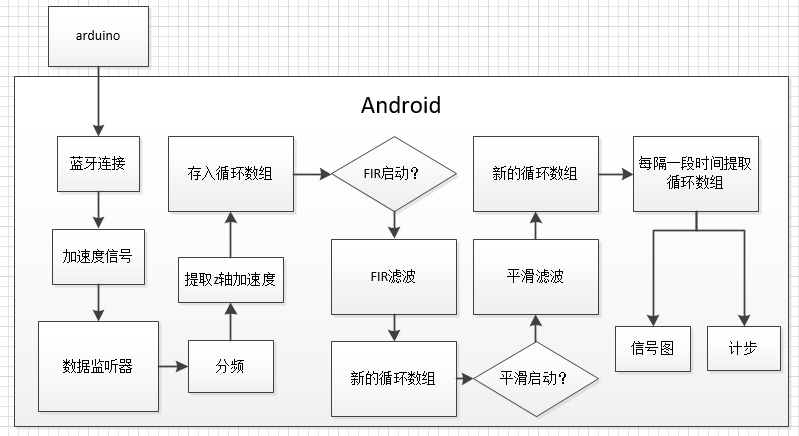
1.可实现对FIR和平滑滤波器参数的更改和调整

2.可随时关闭和开启滤波器

3.可实现加速度信号值、步数和信号图的即时显示

4.可随时重新计步和重绘信号图

三．系统架构



四 关键功能实现

1. 循环数组

由于加速度测量频率为100HZ,经过分频后的频率也达到25HZ，即时处理数据并显示是一个难题，既要考虑到手机处理器的性能及时清除数据，又要考虑到处理的需要缓存下一定量的数据。

循环数组由两个等长度的数组0和数组1组成，是本软件中实现即时处理并显示的关键部分。循环数组的规则非常简单：

1. 设置数组的大小，存入的数据都有序号，当序号等于数组大小时重新计数；
2. 序号小于数组的一半时，存入数组0的前半段和数组1的后半段；
3. 序号大于数组的一半时，存入数组0的后半段和数组1的前半段；
4. 读取数组时，序号小于数组一半在数组0读取；
5. 读取数组时，序号大于数组一半在数组1读取；

以上规则已用在本软件中用代码实现，相关代码如下：

1. 存入数据

|  |
| --- |
| **private void** save(Model model) {  **if** (**counter** == **size**)  **counter** = 0;  **if** (**counter** < **half**) {  **Array0**[**counter**] = model.getaZ();  **Array1**[**counter** + **half**] = model.getaZ();  } **else if** (**counter** >= **half**) {  **Array0**[**counter**] = model.getaZ();  **Array1**[**counter** - **half**] = model.getaZ();  } } |

1. 读出数据

|  |
| --- |
| **private int**[] readForUpdate(**int** num){  **int**[] array=**new int**[num];  **if** (**counterLag**<**half**){  **for** (**int** i=0;i<num;i++){  **if** (**FilterFlag**){  **if** (**AverageFlag**) {  array[i]=**AverageArray0**[**counterLag**+i];  }**else** {  array[i]=**FilterArray0**[**counterLag**+i];  }  }**else** {  array[i]=**Array0**[**counterLag**+i];  }   }  }**else if** (**counterLag**>**half**){  **for** (**int** i=0;i<num;i++){  **if** (**FilterFlag**){  **if** (**AverageFlag**) {  array[i]=**AverageArray1**[**counterLag**-**half**+i];  }**else** {  array[i]=**FilterArray1**[**counterLag**-**half**+i];  }  }**else** {  array[i]=**Array1**[**counterLag**-**half**+i];  }  }  }  **return** array; } |

这种循环数组的优点在于用两个简单的数组实现了数据的缓存和动态更新，节省了系统资源；同时由于读取和存入时有简单可靠的规则，在遵守规则的情况下可以把循环数组当作一个无限长的数组来使用，大大降低了编写程序的复杂性和程序的稳定性。

2.FIR滤波器

由于信号可能存在高频干扰，需要对得到的加速度信号进行FIR滤波以增加计步的准确度。本软件中FIR滤波器实现的原理为：原始循环数组每增加一个值，FIR滤波函数就会提取循环数组中该数值和其前6个数进行FIR滤波算法，得到滤波后的数值存入新的循环数组。

由于循环数组可以当作无限长数组使用，在滤波过程中无需考虑前几个数是否存在的问题，使算法得到大大简化。核心实现代码如下：

|  |
| --- |
| **private void** FIRfilter(){  **int** filterCounterLag=**counter**-**filterOrder**;  **if** (filterCounterLag<0)  filterCounterLag=filterCounterLag+**size**-1;  **if** (**counter** < **half**) {  **double** sum=0;  **for** (**int** i=0;i<**filterOrder**+1;i++) {  **if** (filterCounterLag<**half**)  sum=sum+**FilterParams**[i]\***Array0**[filterCounterLag+i];  **else if** (filterCounterLag>=**half**)  sum=sum+**FilterParams**[i]\***Array1**[filterCounterLag+i-**half**];  }  **FilterArray0**[**counter**]=(**int**)sum;  **FilterArray1**[**counter**+**half**]=(**int**)sum;  } **else if** (**counter** >= **half**) {  **double** sum=0;  **for** (**int** i=0;i<**filterOrder**+1;i++) {  **if** (filterCounterLag<**half**)  sum=sum+**FilterParams**[i]\***Array0**[filterCounterLag+i];  **else if** (filterCounterLag>=**half**)  sum=sum+**FilterParams**[i]\***Array1**[filterCounterLag+i-**half**];  }  **FilterArray0**[**counter**]=(**int**)sum;  **FilterArray1**[**counter**-**half**]=(**int**)sum;  }  } |

3.平滑滤波器

平滑滤波器是对FIR滤波后的信号的进一步处理，由于平滑滤波器在低频段信号处理的优越性，加速度信号的波形会得到进一步增强。平滑滤波器的代码与FIR滤波器类似。核心代码如下：

|  |
| --- |
| **private void** AverageFilter(){  **int** filterCounterLag=**counter**-**averageOrder**;  **if** (filterCounterLag<0)  filterCounterLag=filterCounterLag+**size**-1;  **if** (**counter** < **half**) {  **float** sum=0;  **int** underSum=0;  **for** (**int** i=0;i<**averageOrder**+1;i++) {  **if** (filterCounterLag<**half**)  sum=sum+(**averageOrder**+1-i)\***FilterArray0**[filterCounterLag+i];  **else if** (filterCounterLag>=**half**)  sum=sum+(**averageOrder**+1-i)\***FilterArray1**[filterCounterLag+i-**half**];  underSum=underSum+**averageOrder**+1-i;  }  sum=sum/underSum;  **AverageArray0**[**counter**]=(**int**)sum;  **AverageArray1**[**counter**+**half**]=(**int**)sum;  } **else if** (**counter** >= **half**) {  **float** sum=0;  **int** underSum=0;  **for** (**int** i=0;i<**averageOrder**+1;i++) {  **if** (filterCounterLag<**half**)  sum=sum+(**averageOrder**+1-i)\***FilterArray0**[filterCounterLag+i];  **else if** (filterCounterLag>=**half**)  sum=sum+(**averageOrder**+1-i)\***FilterArray1**[filterCounterLag+i-**half**];  underSum=underSum+**averageOrder**+1-i;  }  sum=sum/underSum;  **AverageArray0**[**counter**]=(**int**)sum;  **AverageArray1**[**counter**-**half**]=(**int**)sum;  } } |

4．信号图实时更新

信号图实时更新的难点有两个：数据的读入和开启新线程。由于使用了循环数组，数据读入大大简化；而由于android编程规范中规定非UI线程无法直接更新UI中的视图，开启新线程的时候需要一定的技巧。

本软件参考了许多种解决方案后最终决定采用android API提供的异步任务类(AsyncTask)来解决这个问题，异步任务简单来说就是整合了子线程和更新UI功能的一个类。

信号图更新方式为：每隔150毫秒判断一次新产生了多少个数据，将新产生的数据全部加到信号图中。由于循环数组的使用，判断新产生的数据过程简化了许多。以下是实现信号图更新的核心代码：

|  |
| --- |
| **private class** ChartTask **extends** AsyncTask<Void, String, Void> {  *// Generates data in a non-ui thread* @Override  **protected** Void doInBackground(Void... params) {  **for** (;;){  **int** i = 0;  **int** updateNum=**counter**-**counterLag**;  **if** (updateNum<0)  updateNum=updateNum+**size**;  **int**[] Array=readForUpdate(updateNum);  **try** {  **do**{  String [] values = **new** String[1];  values[0]=Array[i]+**""**;  publishProgress(values);  i++;  **updateFlag**=**true**;  } **while**(i<updateNum);  **counterLag**=**counter**;  Thread.*sleep*(150);  } **catch** (Exception e) {   }  }  *// return null;* }  *// Plotting generated data in the graph* @Override  **protected void** onProgressUpdate(String... values) {  **if**(**visitsSeries**!=**null**)  {  *// visitsSeries.remove();* **visitsSeries**.add(**visitsSeries**.getItemCount(), Integer.*parseInt*(values[0]));  }  **if** (**updateFlag**)  **mChart**.repaint();  } } |

5.计步

计步是本软件需要实现的最终功能。加速度信号经过滤波处理后，信号已经可以完全反映出走动过程中的加速度变化规律，只需要设定阀值并判断最大值的个数即可。

计步算法中最重要的是如何不重复计步，也不少计步。本软件的计算方式为：每产生10个加速度信号开启一次计步算法，保证步数更新的即时性；从上一个最大值的位置开始，到最新的加速度值对应的序号结束，寻找新增加的超过阀值的最大值。

本软件的算法同样考虑到了边界条件。由于边界上的点可能刚好是极大值点，在算法中特意在最后的结束数据后多增加了一个数据，这样就可以保证步数计算的不重不漏了。计步算法的核心代码如下：

|  |
| --- |
| **private void** Pedometer(){  TextView pedoView=(TextView)findViewById(R.id.***pedometer***);  String stepString=**"step: "**;  **int** stepCounter=**PedometerCounterLag**;  **int** step=0;  **int** length=0;  **int** start=**PedometerCounterLag**;  **int** end=**counter**;   **if** (**counter**%**refreshTime**==0){ *// FindStepsBetween(PedometerCounterLag,counter,TempArray0,TempArray1);* **int**[] TempArray0 = **Array0**;  **int**[] TempArray1 = **Array1**;  **if** (**FilterFlag**){  **if** (**AverageFlag**){  TempArray0 = **AverageArray0**;  TempArray1 = **AverageArray1**;  }**else** {  TempArray0 = **FilterArray0**;  TempArray1 = **FilterArray1**;  }  }   **if**(end>start){  length=end-start+2;  }**else if** (end<start){  length=end-start+2+**size**;  }  **if** (length>0){  **int**[] array=**new int**[length];  **for** (**int** i=0;i<length;i++){   **if** (start+i<**half**){  array[i]=TempArray0[start+i];  }**else if** (start+i>=**half**){  **if** (start+i>**size**-1){  array[i]=TempArray1[start+i-**size**];  }**else** array[i]=TempArray1[start+i-**half**];  }  }  **for** (**int** i=1;i<length-1;i++){  **if** ((array[i]-array[i-1])>0&&(array[i]-array[i+1])>0){  **if** (array[i]>**amplitude**){  step++;  stepCounter=start+i;  Log.*i*(**"step"**, **"Pedometer: "**+start+**" "**+end);  }  }  }  Log.*i*(**"step"**, **"Pedometer-stepCounter: "**+stepCounter);  }   **if** (stepCounter>**size**-1)  stepCounter=stepCounter-**size**;   **pedometer**=**pedometer**+step;  **PedometerCounterLag**=stepCounter;  }   stepString=stepString+**pedometer**;  pedoView.setText(stepString);  } |

所有代码都已开源 可在https://github.com/meiqua/pedometer 上看到