



西安石油大学
Xi'an Shiyou University

本科生毕业设计（论文）

题 目： 基于树莓派远程计步监控系统

的设计与实现

学院（系）： 计算机学院

专业班级： 计 1601

学生姓名： 李永辉




指导教师： 陈磊

评 阅 人： 李杰

完成时间： 2020 年 6 月 4 日



毕业设计(论文)任务书

题 目	基于树莓派远程计步监控系统的设计与实现				
学生姓名	李永辉	学 号	201611090218	专业班级	计 1601
设计(论文)内容及基本要求	<p>设计(论文)内容:</p> <ol style="list-style-type: none">1. 熟悉树莓派开发板的硬件构成和软件开发流程;2. 学习加速度传感器的基本工作原理和典型应用;3. 选择一种三轴加速度传感器, 掌握其接口和数据通信原理;4. 选用合适的软硬件模块搭建一个能够实现远程计步器监控的软硬件系统, 其中计步终端采集三轴运动信息, 并定时将该信息发送给树莓派主机; 树莓派主机进行数据分析, 计算并显示步数。5. 分析加速度传感器的三轴数据, 实现计步算法。 <p>基本要求:</p> <ol style="list-style-type: none">1. 进行广泛的资料查询和文献检索等前期准备工作, 并完成不少于 15000 印刷符号与本专业相关的英文资料翻译。2. 论联系实际, 对系统进行需求分析及课题调研, 查阅相关文献资料, 落实研究内容, 制定研究方法、步骤和措施, 撰写开题报告, 字数不少于 1000 字。3. 严格执行工作计划, 认真填写《西安石油大学本科毕业设计(论文)工作记录》, 按照进度安排在规定时间内完成设计内容, 提交成果。4. 论文写作应符合《西安石油大学本科毕业设计(论文)撰写规范》, 条理清晰, 语言流畅, 论点明确, 论据充分。论文字数不低于 20000 字。按时、按质、按量完成论文撰写, 按标准格式装订成册。				
设计(论文)起止时间	2020 年 1 月 7 日 至 2020 年 6 月 12 日				
设计(论文)地点	西安石油大学				
指导教师签名					2020 年 1 月 3 日
系(教研室)主任签名					2020 年 1 月 7 日
学生签名					2020 年 1 月 9 日

基于树莓派远程计步监控系统的设计与实现

摘要

计步器是人们常见的随身设备之一，一般融合在计步手环手表或者手机等智能设备之中，用于统计人们每天的步行运动量从而给予使用者健康运动的一些建议数据。本文通过对传统计步器的硬件设备架构分离分治作为基础，同时利用时间条件和空间条件的判别方案对步伐判定算法的迭代优化。

在系统硬件设计上，整套硬件系统包括三部分，计步数据采集节点和物联网主机以及远程监控主机，数据采集节点由 MPU6050 运动传感器件和具有无线传输功能的单片机主控 ESP8266 组成，通过 I2C 总线连接。物联网主机由树莓派 3B 模块和 Nokia 5110 液晶显示屏组成并使用 SPI 总线连接。将数据采集和数据处理两部分设备分离并通过无线传输数据的优越性在于可以避免受到穿戴设备功耗和体积的限制，为数据采集提供了更大的空间自由度，降低功耗。第三部分就是远程监控主机，用于监控计步数据采集节点数据变化。

软件设计上，为确保精度，将数据采集节点的原始计步数据，不经过处理直接上传到设备物联网主机后，物联网主机经过解包原始数据，首先进行平均值滤波处理，使得数据更加稳定平滑，再记录更新峰值，同时记录更新动态阈值，这两者将是之后判定步伐空间条件的重要依据。通过记录和对比两个步伐时间间隔和人类正常的步频，实现步伐时间条件的判定。经过时间条件和空间条件的双重限定，步伐统计的准确度显著提升。

测试表明，经过软硬件两方面的优化，相比于同类的计步手环，本计步监控系统计步的准确度更高。

关键词：计步器；加速度计；计步算法；物联网

Design and Implementation of Remote Pedometer Monitoring System Based on Raspberry Pi

ABSTRACT

The pedometer is one of the common portable devices of people, It is generally integrated in smart devices such as pedometer bracelet watches or mobile phones, It is used to count people's daily walking exercise and give users some recommended data for healthy exercise. In this paper, the separation of the traditional pedometer hardware device architecture is used as the basis, and the time and space condition discrimination scheme is used to iteratively optimize the step determination algorithm.

In terms of system hardware design, the entire hardware system includes three parts, a pedometer data acquisition node and an Internet of Things host and a remote monitoring host. The data acquisition node is composed of MPU6050 motion sensor device and single-chip microcomputer main control ESP8266 with wireless transmission function, which is via I2C bus connection. The IoT host consists of a Raspberry Pi 3B module and a Nokia 5110 LCD display and is connected using an SPI bus. The advantage of separating the two parts of data collection and data processing and transmitting data wirelessly is that it can avoid the limitations of power consumption and volume of the wearable device, providing greater space freedom for data collection and reducing power consumption. The third part is the remote monitoring host, which is used to monitor the data changes of the pedometer data collection node.

In terms of software design, in order to ensure accuracy, the original pedometer data of the data collection node is directly uploaded to the IoT host of the device without processing. After the IoT host unpacks the original data, it first performs average filter processing to make the data more stable Smooth, and then record and update the peak value, and record and update the dynamic threshold at the same time, both of which will be an important basis for determining the pace space conditions later. By recording and comparing the two pace time intervals and the normal human stride frequency, the determination of the pace time condition is achieved. After the double limitation of time and space conditions, the accuracy of step statistics has been significantly improved.

Tests show that, after optimization in both hardware and software, the step counting monitoring system has higher step counting accuracy than similar Bracelet.

ABSTRACT

Keywords: Pedometer; Accelerometer; Pedometer algorithm; Internet of Things

目录

摘 要	I
ABSTRACT	I
1 绪论	1
1.1 课题的研究背景和意义	1
1.2 国内外研究现状和发展趋势	1
1.3 文章的组织结构	2
2 系统整体结构概述	4
2.1 系统结构框图	4
2.2 主要功能模块硬件选型	5
2.2.1 计步数据采集节点	5
2.2.2 物联网主机	5
2.2.3 无线路由器	6
2.2.4 远程监控主机	6
2.3 相关软件系统	6
2.3.1 I2C 设备库和 MPU6050 驱动库	7
2.3.2 Arduino 框架和 ESP 相关库	7
2.3.3 Raspbian 操作系统和 Linux API	8
2.3.4 Python 可视化绘图库 Matplotlib	8
3 相关技术简介	9
3.1 Linux 开发环境及常用工具简介	9
3.1.1 Linux	9
3.1.2 远程计步监控系统用到的 Linux	9
3.1.3 Linux 常用工具	9
3.2 交叉编译环境和 Makefile 技术	11
3.3 Visual Studio Code 简介	13
3.4 VS Code 插件 PlatformIO 简介	14
4 计步数据采集节点设计	19
4.1 采集节点简介	19
4.2 运动传感器工作原理	20
4.2.1 陀螺仪	20
4.2.2 加速计	21
4.3 ESP8266 模块介绍与采样实现	21
4.3.1 ESP8266 开发流程和方法	22
4.3.2 I2C 协议和 I2C 库	22
4.3.3 采样率的选择	27

4.3.4 设备初始化和采样实现	27
4.4 无线传输设计与实现	29
4.4.1 网络协议栈结构和应用层协议设计	29
4.4.2 无线网络传输范围和发送广播域	29
4.4.3 WiFi 配置和异步 UDP 广播发送流程	30
5 物联网主机设计	32
5.1 物联网主机简介	32
5.1.1 硬件模块组成	32
5.1.2 软件模块组成	33
5.2 网络通信模块	35
5.3 计步算法模块概述	36
5.4 数据显示模块	37
5.4.1 Nokia5110 与主机连接方式和 SPI 协议简介	37
连接后如图 5.9 所示:	40
5.4.2 内核模块技术和 Frambuffer 驱动加载	41
5.4.3 con2fbmap 控制台映射显示技术实现	41
5.5 守护进程模块	44
5.6 Makefile 编译规则描述	44
6 数据处理和计步算法	46
6.1 算法流水线初始化	46
6.2 样本均值过滤处理	46
6.3 峰值的检测和更新	47
6.4 动态阈值检测和更新	48
6.5 计步的时间条件和时间更新	49
6.6 计步的空间条件和运动轴检测	50
7 远程监控主机程序设计与实现	52
7.1 远程监控主机程序介绍	52
7.2 网络数据的接收和绘图	52
7.3 运动时三轴数据变化分析	55
8 功能性能的调试测试	57
8.1 单独计步数据采集终端调试	57
8.2 运算主机端数据接收联调、显示设备测试	60
8.3 PC 端数据接收和绘图联调	61
8.4 实际步行测试	62
9 结 论	63
参考文献	64

致 谢	65
-----------	----

1 绪论

1.1 课题的研究背景和意义

随着人们对运动与健康的关注度逐渐上升，运动最简单的方式就是步行或者跑步，合理的运动量有利于身体健康，增强免疫力，但过量运动或运动量不足则不一定能够起到强身健体的效果，因此，手环、手机等随身计步设备逐渐进入人们的视野。

计步器通常使用三轴加速度计，也有使用包括陀螺仪的六轴采集设备，传感器通过采集运动数据，并通过校准和滤波等一些算法，最终实时输出当前运动的步数。

现在已经存在的计步方案有配合手机加速度传感器并依赖手机软件进行计步的方案，如微信运动，或手机操作系统包括的计步系统。也有将计步硬件和手机主机分离，手环通过与手机进行通信进行计步。无论那种方式，在软件算法或许不够先进，若提高硬件精度则会提高产品成本。

目前市面上的兼容在手环或者手机中的计步系统，无论是在运动数据采集设备的精度上，算法优良性上导致计步的准确性参差不齐。本文尝试将运动数据采集设备和计步算法设备分离，同时进行算法优化，加入计步的时间条件和空间条件，提高计步准确率。

本文将描述一种在提升计步软件算法准确度的同时，使用一种硬件系统将采用数据采集硬件和计步软件分离的计步方式，采集的数据使用无线网络进行传输，这种方式增大了计步硬件设备的空间可扩展性，间接的降低了成本，并且更能充分发挥空间优势去提升精度。在分离采集设备和计算设备的同时，也优化了计步算法，加入了时间判别条件和空间判别条件，当经过处理的数据同时符合时间条件和空间条件则累加计步，这将大大提高计步精度。

1.2 国内外研究现状和发展趋势

智能穿戴设备的普及，计步器一直是国内外的研究的热门话题。主要的研究方向是提高计步精度，同时尽量降低硬件的成本。在软件上大多数的研究方向是尽量优化算法，以弥补硬件上精度的不足。

国外更多的是通过解析原始数据，将其处理为四元组的形式，并在一些加速度设备上进行测试，很大程度上解决了在慢走上的精度问题。也有很多文章另辟蹊径，通过步伐搜索和识别步行状态的功能，从而使其可以利用变化不太稳定的手臂加速度来实现精确计步，既通过加速度变化计算步长和步态来计步，这种计步算法更加接近计步测量的本质——对个人每日运动量的统计。

国内目前发展出来较为优化的算法一般步骤是采样、卡尔曼滤波、均值滤波、动态峰值、动态阈值等一些数学方法进行数据处理，在步伐判断上采用步间时间间隔或

步伐长度等作为阈值进行判定的计步算法。所以在计步算法上，除了进行必要的数据处理滤波外，计步判别较为准确的方法就是通过步行的峰峰值检测和时间间隔检测值与一般人步行实际测量的步伐进行对比从而判别计步。

M. Oner, J. A. Pulcifer-Stump 等人^[1]，利用陀螺仪设备能够较为准确检测步幅大小的特性，将步伐分类为快步和慢步来用不同的计步算法进行计步。主要解决了在慢走条件下，计步算法精度不够的缺点，另外峰值检测和阈值检测算法值得参考。

Jim Scarlett^[2]主要通过构建步行时的运动模型，使用单个 AN-602 加速度的简单计步器进行测试，并获得良好的实验结果。文章代表了一组尝试从使用单个加速度的简单计步器获得良好性能的实验结果。最终结果达到了规定的精度目标，并通过校准提高了精度，尽管更多加速度计能够实现更高精度，但文章的出发点在于低成本。

Neil Zhao^[3]从人体行走各个步行状态和其加速模型，将人体的动作解析为类似飞机上“横滚、偏航、俯仰”等四元组数据，并据此开发了计步算法。算法中主要利用一些数学方法如数字滤波、动态峰值、动态阈值等方式进行计步，将算法用在 ADXL345 模块上测试得到了较好的测试效果。

叶继超^[4]使用 MPU6050 传感器采集的加速度和角速度数据实现了计步器，传感器内部对加速度和角速度的原始数据进行了滤波处理，经过卡尔曼滤波器筛选最佳角速度，最终能够得到较平稳的加速度和角速度值。用高等数学相关知识如积分等方法可以将速度—时间曲线分解为无限个个长方形，再利用物理方法，初速度和其他时段的加速度，可根据牛顿定律求出运动距离，从而求出步长和步数。

魏芬^[5]同样利用了动态阈值、滤波算法，其中滑动滤波算法是采用循环缓冲区的形式进行，文章参考 Jim 部分算法^[2]，文章设计了整个软硬件架构包括蓝牙无线传输和安卓端应用程序。

本文主要解决更加精确计步算法的精度和数据采集设备与运算设备分离这两方面的问题。提高算法精度方面，主要分为两层，第一层是数据处理，经过各种滤波算法和峰值检测等算法，处理原始数据，第二层判断是否为行走一步将使用时间条件和空间条件共同决定。在运动数据采集和计步运算设备分离中，通过结合加速度传感器和无线传输模块构成了运动数据采集终端，此终端将采集数据上传到计步运算设备，从而实现了采集和运算分离。实现算法代码和硬件连接后，软硬件联合调试测试效果。

1.3 文章的组织结构

本文主要结构安排是通过本系统的各个模块进行分割，正文共 9 章。前面 3 章是一些技术说明或者概述性的章节：系统整体概述、相关技术简介部分。中间部分 4-7 章系统模块的主要组成部分和软件算法等构成了文章的核心支撑：计步数据采集节点、物联网主机、远程监控主机、计步算法相关章节。最后部分 8-9 章是一些测试调试结果和结论的叙述部分，构成文章结论。

首先要明确题目的应用范围和技术栈，明确任务，概述部分主要解释题目，题目

的来源和意义，国内外目前的发展现状，以及整篇文章的结构。明确题目后要做的是介绍整个系统的架构组成以及写入系统的相关技术。概述部分最后是介绍用于开发相关的技术，更多的是介绍开发环境和开发工具。

论文的核心部分是计步系统各个模块的设计和实现方案，包括四，五，七章的系统硬件架构和于算法无关的软件设计，以及第六章的数据处理和算法相关内容。

最后的测试和结论部分为 8，9 章，第八章介绍所有的调试通道和单元测试，以及最后与市场计步手环的对比测试。结论章叙述了本文的最终研究的结果解释。

2 系统整体结构概述

整个系统通过加速度传感器对计步数据的采集，无线传输模块对计步数据的校正和上传，上传到物联网主机后将由主机对数据进行解包和数据处理，并通过计步算法计步，计步结果实时显示在液晶屏上。

2.1 系统结构框图

如图 2.1 所示，系统主要由四部分组成，自下而上分别是计步数据采集节点、物联网主机、远程监控主机和提供局域网的无线路由器。

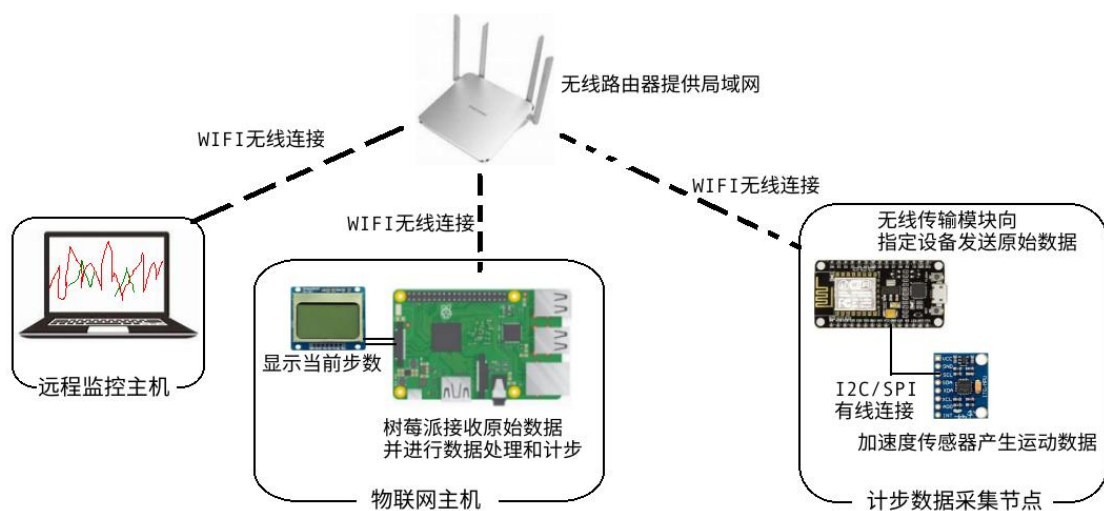


图 2.1 系统结构图

无线路由器是家中常见的网络设备，将为整个系统提供局域网络，方便各个设备交换数据。

计步数据采集节点是计步所用到的运动数据来源，主要功能是对运动数据进行采集并发送，对整个系统起数据源头作用。节点包括两部分：加速度传感器 MPU6050，用于加速度的原始数据采集，无线传输模块 ESP8266，用于原始数据的采样率控制，数据校准，同时将连接无线路由器进行数据发送。

物联网主机是一个以家庭或小型组织为单位的物联网处理中心，负责对各物联网终端节点的数据收集和处理。由主机 raspberry PI 3b 和 Nokia 5110 液晶屏幕组成，树莓派在本系统中需要接入无线路由器，接收计步数据采集节点发送来的运动数据，并对原始数据进行滤波处理，通过计步算法计步，将当前步数和当前加速度状态显示到 LCD 液晶屏幕上。

远程监控主机用来监控整个片区的所有物联网主机设备运行状态，主要运行的一些监控程序和绘图程序将一些数据可视化展示出来，如用折线图展示计步数据采集节点的原始数据，为访问到局域网设备数据，也将接入无线路由器。

2.2 主要功能模块硬件选型

一个从无到有的系统，从思路的设计，功能模块的确定，到硬件的选型都至关重要。硬件的选型关系到产品在市场上的竞争力，这就需要在设备的精度质量和设备造价之间达到一个平衡。

2.2.1 计步数据采集节点

数据采集节点包括两部分，一是采集人体运动数据的传感器，能准确采集人体三个方向的加速度值，使用的 MPU6050 加速度传感器，二是一个进行传感器原始数据无线发送的并且具有 I2C 接口的单片机，本系统选择乐鑫的 ESP8266。

加速度计 (accelerometer) 又称加速规、加速针、加速度感测器等，是测量加速度的装置。加速度计种类多样，按照工作原理分类，常见的有压电效应、压阻效应、电容式感应等。市面上常用的有 Pmod ACL2、ADXL345、MPU-6050, Pmod ACL2 具有 12 位的分辨率，也是基于 MEMS 的加速度器件，包含了 ADXL362，特点是具有 512 个大容量的样本 FIFO 缓冲区，当然造价也偏高，不适用于本系统。ADXL345 主要特点是低功耗，正常工作的功耗只有 100uA 左右，待机模式甚至只有 0.1uA，但只具有三轴的加速度数据，因此一般用来作为仅存在重力的平面倾角测量，不能用来作为全方位的运动数据采集传感器。MPU6050 是一个包括了加速度传感器和角速度传感器的 6 轴运动处理传感器，甚至可以外接一个 I2C 三轴磁力计从设备，组成 9 轴运动传感器，而且内置了数字运动处理 (DMP) 引擎，减少了滤波等复杂算法，精度达到 16 位并具有 1024 字节的 FIFO 缓冲区，而且造价低廉，是非常合适的计步器运动数据采集器件。

物联网的井喷式发展，家庭物联网 SoC 等方案也顺势而上，如乐鑫的 ESP8266 和 ESP32 系列芯片具有较高性价比并且应用非常广泛，大到大型机械的继电器控制，小到物联网空调物联网插座等。这是将一定范围内（一般以家庭为单位）的无网络设备接入互联网作为终端设备，实现万物互联的最佳方法。也有使用移动运营商网络如 2G/4G 移动卡模块，目前常见的有 NB-IoT、Lora 方案，这两种方案的优势是没有距离限制，但同时带来的缺陷是需要支付昂贵的流量费用，因此更多用于工业物联网。对于家庭物联网，ESP8266 拥有超低功耗 Tensilica L106 32 位处理器，CPU 主频最高 160 MHz，支持实时操作系统和适用于嵌入式设备的轻量级 lwIP 协议栈，作为控制型 SoC，ESP8266 更是具有主机 SPI 和从机 SPI 接口，I2C 接口，UART 接口以及通用的 GPIO 等，这将是更加成熟的物联网方案，是本系统进行数据采集和无线传输设备的不二之选。

2.2.2 物联网主机

本系统的物联网主机将选用树莓派 3B，3B 具有无线局域网和蓝牙连接功能的卡片计算机。相比与刚出来的树莓派 4B，3B 的购买成本更低，并且几乎具有和 4B 相

同的所有接口，只是在计算和存储性能上稍逊与 4B，而更廉价树莓派 zero，并没有充足的硬件接口和计算性能和无线装置，树莓派 3B 甚至可以作为简单上网的桌面电脑，3B 拥有以下资源和接口：

- 四核 1.2GHz Broadcom BCM2837 64 位 CPU
- 1GB 内存
- 板载 BCM43438 无线局域网和蓝牙低功耗（BLE）
- 100 Base 以太网
- 40 引脚扩展 GPIO
- 4 个 USB 2 端口
- 4 极立体声输出和复合视频端口
- 全尺寸 HDMI
- CSI 相机端口，用于连接 Raspberry Pi 相机
- DSI 显示端口，用于连接 Raspberry Pi 触摸屏显示器
- Micro SD 端口，用于加载操作系统和存储数据
- Micro USB 电源可负载高达 2.5A

使用他作为物联网主机支撑以家庭为单位的轻量级物联网运算最合适。

物联网主机上携带的液晶显示屏模块选用 Nokia 5110 液晶 LCD 屏幕，使用 Linux 自带的 Frambuffer 驱动将映射终端标准输出重定向到/dev/fb1 Frambuffer 设备，这样可以将程序的标准输出 printf 内容打印到 LCD 屏幕上。

2.2.3 无线路由器

无线路由器为整个系统提供局域网，常用的家用无线路由器即可。本计步系统选用的 PHICOMM K2P，此路由器拥有 MIPS 架构双核 880Mhz 的 MT7621A 处理器，并安装开源的 Linux OpenWrt 路由器系统，使用局域网 C 类地址池，192.168.1.0/24 网段。因为 OpenWrt 系统具有更多的可操作性，实际计步系统不依赖于 OpenWrt，仅仅能够提供局域网的无线路由器均可。

2.2.4 远程监控主机

远程监控主机用于监控物联网主机、传感器等设备的工作状态，普通的电脑即可。一般安装 Linux 操作系统，并安装 Python 和相应的可视化绘图库，将监视软件上传安装至监控主机，运行监视软件后能看到传感器传输的实时数据曲线。

也可用监控主机对物联网主机进行软件的更新等操作，如更新计步系统可以使用 SCP 命令进行远程文件拷贝更新，使用 SSH 命令可以对物联网主机进行任意操作。

2.3 相关软件系统

计步数据采集节点上，底层针对传感器的通信和控制使用到 I2C 库和 MPU6050 驱动库，上层针对无线发送实现上用到了 Arduino 框架和 ESPWiFi、异步 UDP 库，

串口调试用到了 Serial 串口库。物联网主机上,安装官方支持的树莓派系统基于 Linux 的 Raspbian, Linux 所有的发行版本都支持 POSIX, 并开放了简单易用 Linux API。远程监控主机上为展示可视化传感器数据, 需要用到 Python 的可视化绘图库 Matplotlib。

2.3.1 I2C 设备库和 MPU6050 驱动库

由于设备驱动具有较强的面向对象特点, I2C 和 MPU6050 驱动均使用 C++编写, 他们均运行在计步数据采集节点的 ESP8266 模块上。MPU6050 传感器则使用 ESP8266 驱动, 并使用 I2C 端口通信, 因此, MPU6050 驱动中使用到了 I2C 设备驱动, 对于 I2C 总线, 顾名思义, 总线需要用来传输数据, 因此库中大多数实现的是类似于 read/writeBit(), read/writeByte(), read/writeWord()等接口函数, 最底层则实现了 I2C 两线的时序。

MPU6050 驱动将利用上述 I2C 驱动实现对传感器的操作, 接口中有大量的 set/get() 方法, 包括最基本的原始 FIFO 数据读取、各个寄存器的设置, 获取各轴加速度陀螺仪等数据, 以及之前介绍过的 DMP 库相关的配置接口, 滤波寄存器配置等, 甚至 DMP 库中包含的计步算法等。但本系统将对计步算法进行优化, 因此用不到 DMP 库, 仅使用它获取原始数据。

2.3.2 Arduino 框架和 ESP 相关库

Arduino 是一款简单灵活、容易上手的开源电子平台。包含 Arduino 开发板和平台 Arduino IDE 或者称为 Arduino 框架。在本系统没有用到 Arduino 的硬件, 因为 ESP 系列无线模块对 Arduino 框架、ArduinoIDE 的支持性较好, 本系统的 ESP8266 将使用 Arduino 的框架, 此框架主要包含了一些 C++库, 比如本次用到的 Serial 库, 并且还可以使用 ESP 的相关库, 比如 ESPWiFi 库, 异步 UDP 库等。

Arduino 框架的快速上手还体现在他的代码实现, 此框架不像其他开发板需要从 main 函数开始编写, Arduino 框架充分考虑硬件编码的特点, 将代码分为初始化部分和循环部分, 一些设备和接口的初始化部分写在 setup()函数中, 循环执行部分写在 loop()函数中, 这样逻辑层次清楚, 代替了原始的单片机固定代码 main {while (1);}的形式。

Serial 库是 ESP 中自带的串口库, 在本系统中仅仅用于打印调试信息, 通过 TTL 转 USB 连接电脑输出调试信息, 烧写 ESP8266 的二进制代码也需要使用串口烧写。

ESPWiFi 库是 ESP 系列无线模块的 WiFi 库, 他包含 TCP/IPv4 协议栈, WiFi 协议栈, 在 setup()的初始化中需要调用 WiFi 配置以及连接相关接口, 使 ESP 设备接入局域网。

异步 UDP (AsyncUDP), 主要为开发人员提供的 send/recv/read/write/listen/broadcast()等一些常见的网络接口, 也实现了一些相关子类作为实现支撑, 如,

AsyncUDPPacket, AsyncUDPMessage 类。

2.3.3 Raspbian 操作系统和 Linux API

Raspbian 是树莓派官方支持的操作系统，他是基于 Debian 系统经过硬件定制后运行在树莓派上的 Linux 操作系统，本计步系统将选用 Raspbian Buster Lite 版本，这个版本是官方支持的最小镜像。在 Linux 开发环境下安装本系统的方法是下载镜像并解压，得到 img 后缀的镜像，然后使用 dd 命令，将 xxx.img 镜像烧写进 SD 卡，SD 卡设备一般在/dev/sdc。

Linux API 是 Linux 操作系统为开发者提供的应用程序开发接口，包括本计步器用到的一些常见系统调用 open/close/stat/fork/setuid()，包括一些网络 API 如 socket/bind/recv()等，还用到了针对设备文件的操作如/dev/fb0 设备的 frambuffer、/dev/tty0 设备的 console 重定向等操作。

2.3.4 Python 可视化绘图库 Matplotlib

Python 是一种脚本型的面向对象的高级程序设计语言，尤其是在数据分析方面，Python 可以更高效更快速的完成工作。本系统的远程监控主机程序将使用 Python 语言设计。

Matplotlib 是一个 Python 的数据分析绘图库，它以各种硬拷贝格式和跨平台的交互式环境生成较高质量级别的图形。远程监控主机上运行的程序所绘制的动态图形是 Python 创建的 UDP 客户端接受到的原始数据，使用 FuncAnimation 类动态的绘制到画布上。

3 相关技术简介

3.1 Linux 开发环境及常用工具简介

3.1.1 Linux

Linux 一般是指一类免费使用和自由复制的‘类 Unix’操作系统的 Linux Kernel, 有时也可泛指所有的基于 Linux Kernel 的操作系统。这类操作系统具有上百种不同的发行版, 发行版本可以大体分为两类, 一类是商业公司维护的发行版本, 比如 Redhat, 一类是自由软件社区维护的发行版本, 比如 Debian。本计步系统所有用到的操作系统均基于 Linux。

3.1.2 远程计步监控系统用到的 Linux

远程计步监控系统的开发环境使用国内自主的发行版本深度系统 deepin, deepin 是基于 Debian 稳定版本的一个 Linux 发行版。deepin 可以运行在个人计算机和服务器的上, 并免费提供给个人用户使用。deepin 因其美观和易用性而广受赞誉, 据 DistroWatch 的数据, 截至 2017 年, deepin 是最受欢迎的源自中国的 Linux 发行版。2019 年, 华为开始销售预装有深度操作系统的笔记本电脑。

物联网主机的树莓派上所安装的操作系统 Raspbian 是基于 Debian 的操作系统, 其针对树莓派的硬件进行了优化。2.2.3 节已介绍, 此处不在赘述。

为计步系统提供局域网的无线路由器安装基于 Linux 内核的 OpenWrt 的操作系统, OpenWrt 是更适合经裁剪的资源有限的嵌入式设备的操作系统, 其高度模块化、高度自动化, 且拥有强大的网络组件和扩展性而被广泛应用于工控设备、机器人、智能家居、路由器以及智能音箱中。

远程监控主机可以使用任何装有 Python 和相关库的 Linux 发行版, 本次开发将使用开发环境 deepin 测试。

3.1.3 Linux 常用工具

SSH (SSH 客户端) 是用于登录到远程主机并在远程主机上执行命令的程序。SSH 是在两个互不信任的主机之间提供较为安全的通信。它要求目标主机运行 sshd 服务 (SSH 服务端), 可通过密钥文件或者 Unix 密码输入进行认证, 如登录树莓派 PI 用户的命令是 `ssh pi@192.168.1.1`, 之后输入密码即可登录, 登录后即可进入 Unix 终端, 可以执行命令。

SCP 是一个远程文件拷贝程序, 在网络上的主机之间复制文件。SCP 使用 SSH 进行数据传输, 并使用与上述 ssh 相同的安全协议。它支持将本地文件拷贝到远程主机, 也支持把远程主机文件拷贝到本地, 将本地 pedometer 文件拷贝到树莓派使用 `scp /home/lee/pedometer pi@192.168.1.1:/home/pi/pedometer`。

GCC 是指 GNU 编译器集合（GNU Compiler Collection），指一套 C 语言编译器，GCC 最简单的使用方法不带参数是执行 `gcc main.c` 命令，它将编译后生成一个默认名称为 `a.out` 的可执行程序。GCC 常用的参数有 `-o` 指定输出的可执行文件名，`-g` 生成的可执行程序包含调试信息，使用 `gdb` 等调试工具之后可以看见代码或者变量信息，因而文件也较大，`-I` 指示除了系统头文件路径外需要另外包含的头文件路径，`-l<库名称>` 可以包含用到的动态库，这样在程序运行时候将加载对应的动态库，如 `-lpthread` 多线程库，`-Wall` 选项是 **Warning All**，也就是打开所有级别的 **Warning** 开关，这样将按照最严格的编译标准报告 **Warning**，`-O0/1/2/3` 是对代码优化，O 后的数字是优化级别，默认 `O1`，`O0` 为不优化，`O3` 优化级别最大，也是最危险的，该选项除了执行 `-O2` 所有的优化选项之外，一般都是采取很多向量化算法，在不影响代码逻辑的情况下，调整指令先后顺序，利用处理器的流水线机制，**Cache** 等提高代码的并行成都。

GDB 是一个终端调试程序，即只需在终端执行相应的命令即可完成调试，与常见的 windows 平台编译器 VC++ 软件的调试方式对比如下图 3.1 所示

功能	VC++图标	VC++名称	GDB 命令	GDB 命令缩写
新增，删除断点		Insert/Remove breakpoint	break	b
开始调试		go	run	r
单步执行，如有函数调用，则进入函数		Step into	step	s
单步执行，如有函数调用，则跳过函数		Step over	next	n
完成函数调用，并返回		Step out	finish	f
运行到光标所在行，GDB 无对应命令，但 continue 类似，		Run to cursor	continue (表示运行到程序结束或下一断点处)	c
查看程序调用堆栈		Call Stack	backtrace	bt
查看内存		Memory	examine	x
反汇编代码		dissassembly	disassemble	
查看寄存器		Registers	info registers	i r
查看变量		Variables	info local	i lo
退出调试		Stop debugging	Quit	q

图 3.1 VC++和 GDB 用法对比

GDB 加上 `-tui` 参数可以同 VC++ 一样可以显示当前正在运行的代码，但使用 GDB 的插件 `gef` 将是最佳方案，如图 3.2 被调试程序运行后默认显示图上内容，自上而下依次是当前处理器的寄存器值、压入的栈地址、正在执行在 `x86_64` 平台的汇编代码（即将执行的是黄色）、正在执行的 C 语言代码（代码上方自动显示当前变量的值）、

当前线程和函数的执行路径 trace 等。

```
[ Legend: Modified register | Code | Heap | Stack | String ]

registers
$rax : 0x0
$rbx : 0x0
$rcx : 0x00007ffff7ffcca0 → 0x0004095d00000000
$rdx : 0x0
$rsp : 0x00007ffffffffffe530 → 0x0000000000000000
$rbp : 0x00007ffffffffffe560 → 0x000000000004007f0 → <__libc_csu_init+0> push r15
$rsi : 0x00007ffff7dd1b78 → 0x00000000000602000 → 0x0000000000000000
$rdi : 0x20000
$rip : 0x00000000000400799 → <main+64> mov QWORD PTR [rbp-0x28], rax
$r8 : 0x00007ffff7fec700 → 0x00007ffff7fec700 → [loop detected]
$r9 : 0x1
$r10 : 0x0
$r11 : 0x246
$r12 : 0x00000000000400580 → <_start+0> xor ebp, ebp
$r13 : 0x00007ffffffffffe640 → 0x0000000000000001
$r14 : 0x0
$r15 : 0x0
$eflags: [carry PARITY adjust ZERO sign trap INTERRUPT direction overflow resume virtualx86 identification]
$ss: 0x002b $cs: 0x0033 $ds: 0x0000 $gs: 0x0000 $es: 0x0000 $fs: 0x0000

stack
0x00007ffffffffffe530 +0x0000: 0x0000000000000000 ← $rsp
0x00007ffffffffffe538 +0x0008: 0x0000000000000000
0x00007ffffffffffe540 +0x0010: "myfile.txt"
0x00007ffffffffffe548 +0x0018: 0x00000000000007478 ("xt"? )
0x00007ffffffffffe550 +0x0020: 0x00007ffffffffffe640 → 0x0000000000000001
0x00007ffffffffffe558 +0x0028: 0xd7c3f14d3cddb000
0x00007ffffffffffe560 +0x0030: 0x000000000004007f0 → <__libc_csu_init+0> push r15 ← $rbp
0x00007ffffffffffe568 +0x0038: 0x00007ffff7a2d830 → <__libc_start_main+240> mov edi, eax

code:i386:x86-64
0x40078c <main+51> mov esi, 0x400874
0x400791 <main+56> mov rdi, rax
0x400794 <main+59> call 0x400550 <fopen@plt>
→ 0x400799 <main+64> mov QWORD PTR [rbp-0x28], rax
0x40079d <main+68> cmp QWORD PTR [rbp-0x28], 0x0
0x4007a2 <main+73> jne 0x4007bc <main+99>
0x4007a4 <main+75> lea rax, [rbp-0x20]
0x4007a8 <main+79> mov rsi, rax
0x4007ab <main+82> mov edi, 0x400876

source:vsprintf.c+20
15 int main ()
16 {
17     FILE * pFile;
18     char szFileName[]="myfile.txt";
19     // pFile=0x00007ffffffffffe538 → 0x0000000000000000, szFileName=0x00007ffffffffffe540 → "myfile.txt"
→ 20     pFile = fopen (szFileName,"r");
21     if (pFile == NULL)
22         PrintFError ("Error opening '%s'",szFileName);
23     else
24     {
25         // file successfully open

threads
[#0] Id 1, Name: "vsprintf", stopped, reason: SINGLE STEP

trace
[#0] 0x400799 → Name: main()

gef>
```

图 3.2 GDB 的 gef 插件界面

3.2 交叉编译环境和 Makefile 技术

交叉编译器（Cross compiler）是指一个在某个指令集架构的主机下可以产生另一个指令集架构主机的可执行文件的编译器。交叉编译器在目标系统平台上难以或不容易编译时非常有用。

一般的本地编译器，编译时候使用的是本地处理器架构的指令集进行对照编译，因此在一般的个人电脑 x86 平台编译的可执行程序由于指令集不同而不能运行在 ARM 平台。尽管可以在 ARM 平台编译后可在 ARM 平台运行，但大多数嵌入式设备因为需要控制成本，板载的资源 and 性能不足以用来编译代码甚至不足以安装操作系统，而个人电脑又具有充足的资源，因此需要一款编译器需要运行在个人电脑上并且编译出的可执行程序是目标平台（开发板）的二进制指令集，这种编译器就是交叉编译器。那么一般的嵌入式开发模式就是在 PC x86 平台编写代码并使用交叉编译工具

编译，之后使用 USB-TTL 串口或者网口 tftp 等工具上传烧写到目标平台的存储器，之后重启目标平台则程序开始运行。

计步数据采集节点主控模块的 ESP8266 使用的是 Tensilica 公司的 Xtensa 处理器，运行在 ESP8266 上的代码也将使用 Xtensa 交叉编译工具编译，即在 x86->Xtensa 的跨平台编译，Xtensa 编译器将被安装集成在后续章节提到的 PlatformIO 插件中。

物联网主机树莓派使用的处理器是基于 ARM 架构博通处理器 BCM2837，虽然树莓派安装了 Linux 和 gcc 可以直接编译，但使用资源充足的个人电脑 x86 平台进行交叉编译速度会更快，使用交叉编译工具 arm-linux-gcc 即 x86->ARM 的跨平台编译。

make 是一个指定编译过程的编译脚本解释器，执行 make 命令后会读取当前目录的 Makefile 文件，依照 Makefile 脚本自动化建构软件。几乎每个 IDE 平台都会有自己的 make 或者类似于 make 的东西，Qt 使用的是 qmake，Visual Studio 使用的是 cmake，Visual C++ 的 nmake，Linux 则使用的是 GNU make。

GNU 上大部分用 C/C++ 编写的软件都使 Makefile 构建，当然它是没有限制语言，也可以将 Java 或 Python 编写的程序使用它构建。由于大型工程编译依赖树的复杂度较高，各种库依赖和头文件依赖等，编译一个工程一次性敲完这些命令是一个很麻烦的事情，Makefile 可以完成这一切，它只是定义一系列规则，哪些文件先编译，哪些文件后编译，哪些和哪些文件链接或者和库链接等，通常也在 Makefile 脚本中指定依赖的源码目录、头文件目录、库名称、编译参数等。它将会在依赖头下执行的是 shell 命令比如 arm-linux-gcc 或 gcc 等。

Makefile 往往通过依赖规则来寻找编译路径：脚本将自动寻找并组建依赖关系树，它首先要寻找的是最终目标 all：后面的内容，最终目标将成为树的根（通常是一个也可以是多个），在进行最终目标编译的时候发现依赖其他目标，那么继续寻找其他目标，以此类推最终达到树的叶节点，它通常是一个源文件或者一个库文件，最后执行叶节点之后的命令，即编译为目标文件（windows 平台位 obj 文件，Linux 平台位 elf 文件），各个叶节点编译后再向树的根部合并，使用链接器链接各个目标文件链接。

另外 Makefile 也有很强的时效性，它并不会去编译已经编译过的而且没有被修改的源文件，这样可以加快编译速度，规则如下：

- 1) 如果这个工程没有编译过，那么我们的所有.c 文件都要编译并被链接。
- 2) 如果这个工程的有些.c 文件被修改，那么我们只编译被修改的.c 文件为.o 文件，并链接为最终的目标程序。
- 3) 如果这个工程的某些.h 文件被改变了，那么我们需要编译引用了这个头文件的所有.c 文件，并链接为最终目标程序。

那么上述规则也就是 Shell 脚本不能代替 Makefile 的原因。

使用时候只需要编写一个 Makefile 脚本文件来定义编译规则，放在源码根目录，执行 make 命令即可完成复杂软件的构建，执行 make clean 可清理编译。

3.3 Visual Studio Code 简介

Visual Studio Code(以下简称 VS Code)是由微软开发,同时支持 Windows、Linux 和 macOS 等操作系统且开原的代码编辑器。VS Code 支持自动补全,测试调试,编译运行功能,同时也具有集成开发环境的功能,例如代码片段和代码重构等,支持多种语言如 Shell, C/C++,Java,Python, Vala, Clojure, CoffeeScript, DockerFile, F#, Go, Jade, HandleBars, Ini, Lua, Makefile, Markdown, Objective-C, Perl, PHP, PowerShell, R, Razor, Ruby, Rust, SQL, Visual Basic, XML, Tex 等常见语言的语法高亮、智能补全提示、编译调试等常用工具,这些功能实现需要它的插件和系统软件(如 gcc gdb)配合使用。该编辑器可配置度高,大多数配置在配置文件中完成,符合程序员开发习惯,同时还在编辑器中内置了扩展程序管理以及他人开发的各种意想不到的功能的插件。

VS Code 是基于 Electron (原来叫 Atom Shell) 进行开发的。Electron 基于 Node.js(作为后端运行时)和 Chromium(作为前端渲染),使得开发者可以使用 HTML, CSS 和 JavaScript 等前端技术来开发跨平台桌面 GUI 应用程序。VS Code 的主要技术栈是使用 Web 技术来编写 UI,用 Chrome 浏览器内核来运行,使用 Node.js 来操作文件系统和发起网络请求,使用 Node.js/C++/Addon 去调用操作系统的 Native API。

2016 年 4 月 14 日,VS Code 正式版发布,版本号为 1.0.0。不到四年时间 VS Code 迅速成为最受欢迎的代码编辑器,VS Code 支持多种编程语言,集成终端,可以在编辑器中运行脚本、编译软件、调试脚本、设置断点、做版本管理。而且它还支持远程开发,只需要设置可访问的远程主机 IP 地址和路径以及认证信息,即可访问到任何一个计算机的任何目录进行远程开发,这样实现了无需本地写完代码后上传编译或编译上传,直接可在远程主机上编译调试运行等。VS Code 支持直接将全部服务部署到服务器,这样即不失去完美编辑器的功能,同时也能让所有的服务均处于离线或内网状态,保证企业信息不泄露。

VS Code 对有限的电脑平面进行充分的利用,的如图 3.3 所示,它去掉了标题栏,取而代之的是置顶的菜单栏,最左侧是主要的功能标签页,如文件列表、全局搜索、git 版本管理、调试、插件管理等页面。往左较窄的是文件列表标签页的文件列表,在其他标签页则显示其他内容。中间就是代码区域了,最右侧是代码缩略图,可以迅速找到想找的代码。底下蓝色条是状态栏,同时也包括提交 git、文件编码设置等很多操作,上拉状态栏可以拉出终端界面。

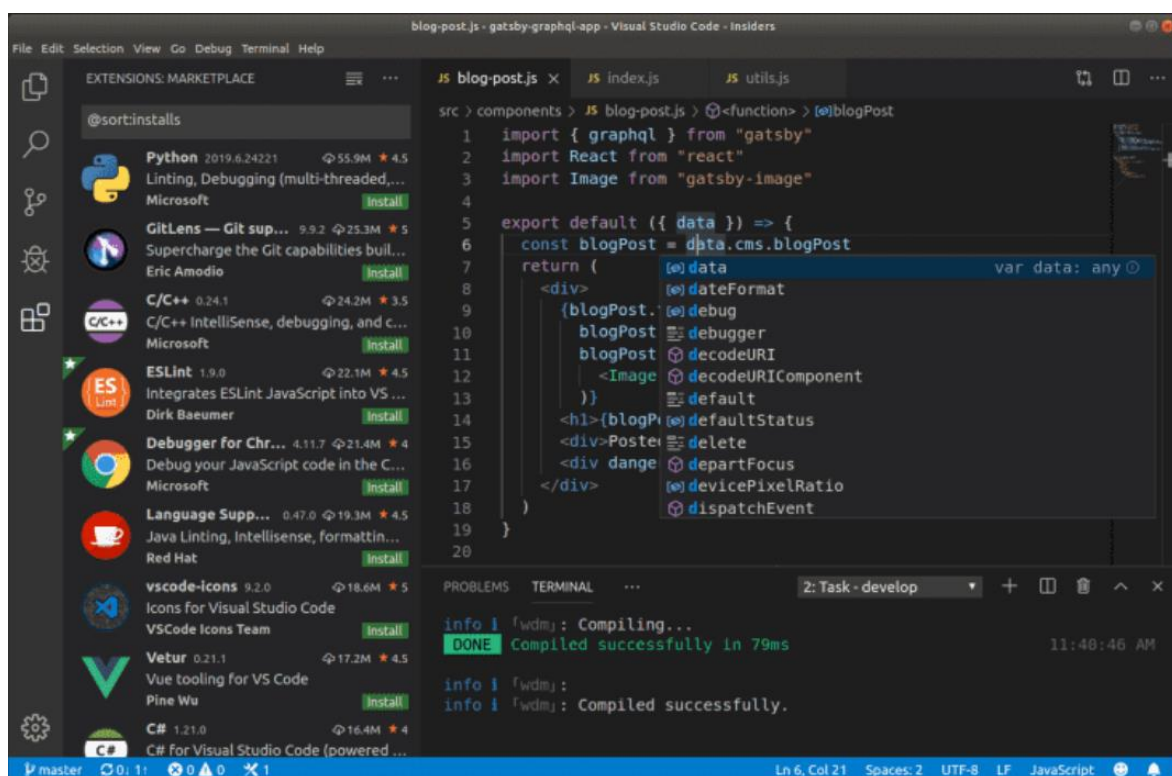


图 3.3 deepin 系统中 Visual Studio Code 运行截图

3.4 VS Code 插件 PlatformIO 简介

PlatformIO 是面向嵌入式系统工程师和为嵌入式产品编写应用程序的软件开发人员的跨平台，跨体系结构，多框架的专业工具，它支持图 3.4 列出的平台。



图 3.4 PlatformIO 支持的平台

PlatformIO 在嵌入式市场中的独特理念为开发人员提供了一个现代的集成开发环境（Cloud&Desktop IDE），该环境可以跨平台工作，支持许多不同的软件开发工具包（SDK）或框架，并包括复杂的调试、单元测试、代码静态动态分析和远程管理。它作为嵌入式开发平台旨在最大程度地提高开发人员的灵活性和选择范围，开发人员可以选择使用图形编辑器或命令行编辑器。

它可以在任何现代操作系统（macOS，Windows，Linux，FreeBSD）上运行，并可以集成在 VS Code、Atom、Clion、Eclipse、Emacs、Qt、Sublime、Vim 等 14 个软件平台。如图 3.5 所示，同 VS Code 一样，也可以在浏览器端运行这个开发环境，web

端体验同桌面端，具有简单易读的在线手册（<https://docs.platformio.org/>）

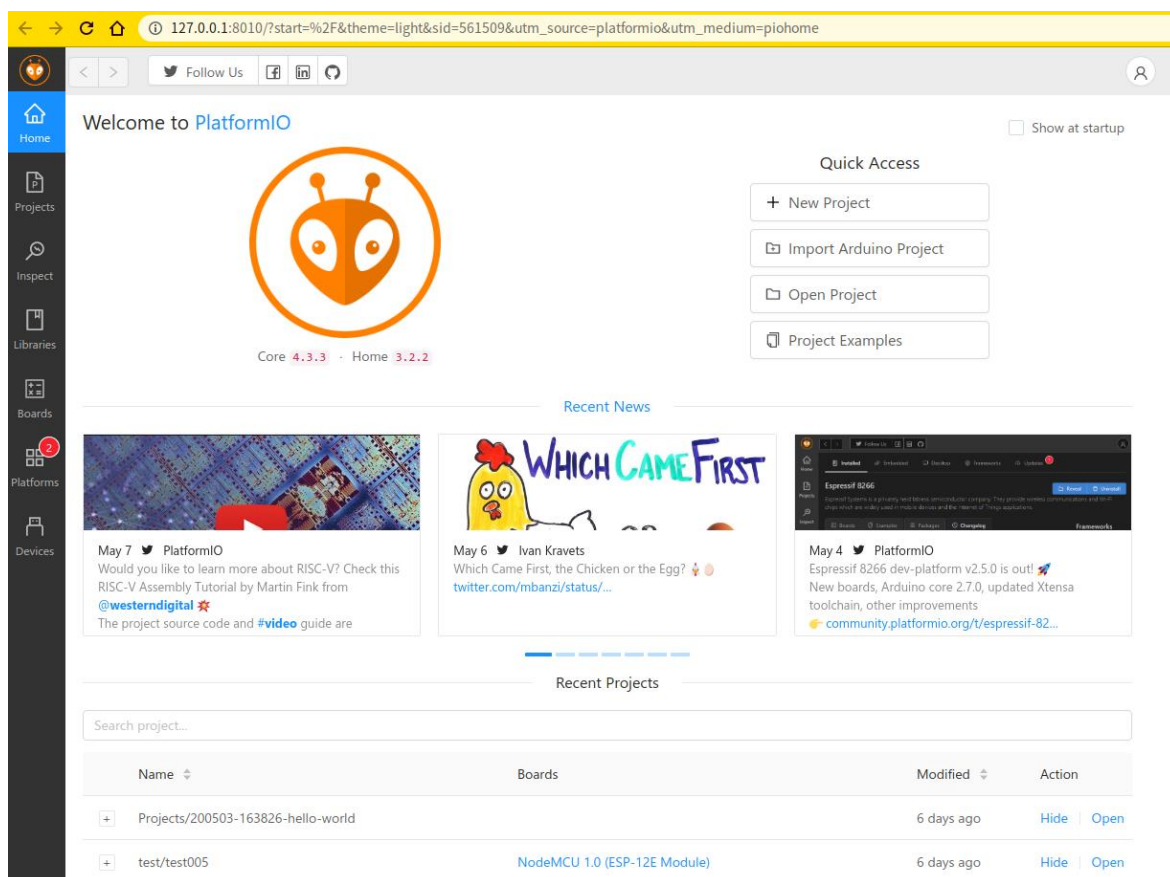


图 3.5 PlatformIO 运行在 Chrome 浏览器

对于一般的简单代码如 helloworld，一般的开发流程是先选择开发平台如 ST 系列的 STM32，ESP 系列等，在 Platforms 页中搜索或者选择嵌入式、桌面、框架等，点击安装对应的平台后会自动集成其编译环境和依赖包，如 ESP 系列依赖 Xtensa 交叉编译环境。安装好平台后，可以在 Home 页选择 New Project 新建项目，需要选择刚刚安装的平台、Board 和 Framework，选择后即可按照框架生成代码依赖，按照开发板生成配置文件。

之后在代码文件中写入简单代码后可以编译上传和测试/监视，编译使用 Build，没有错误则插入开发板，检查 Devices 选择设备，即可上传程序 Upload，上传后程序在目标板开始运行，假如你写的是串口循环输出语句，点击 Monitor 即可监视串口输出内容。

如图 3.6，PlatformIO 插件在 VS Code 上各个功能分布，先点击最左侧的蚂蚁图标进入插件，左上部分 Build(图上框中三种使用方式均可)、Upload 等按钮是对代码的编译或上传检查测试等常用操作，左下部分 PIO Home、Debug 等是对当前插入的设备或安装更新的库管理，设备管理等一些快速访问。

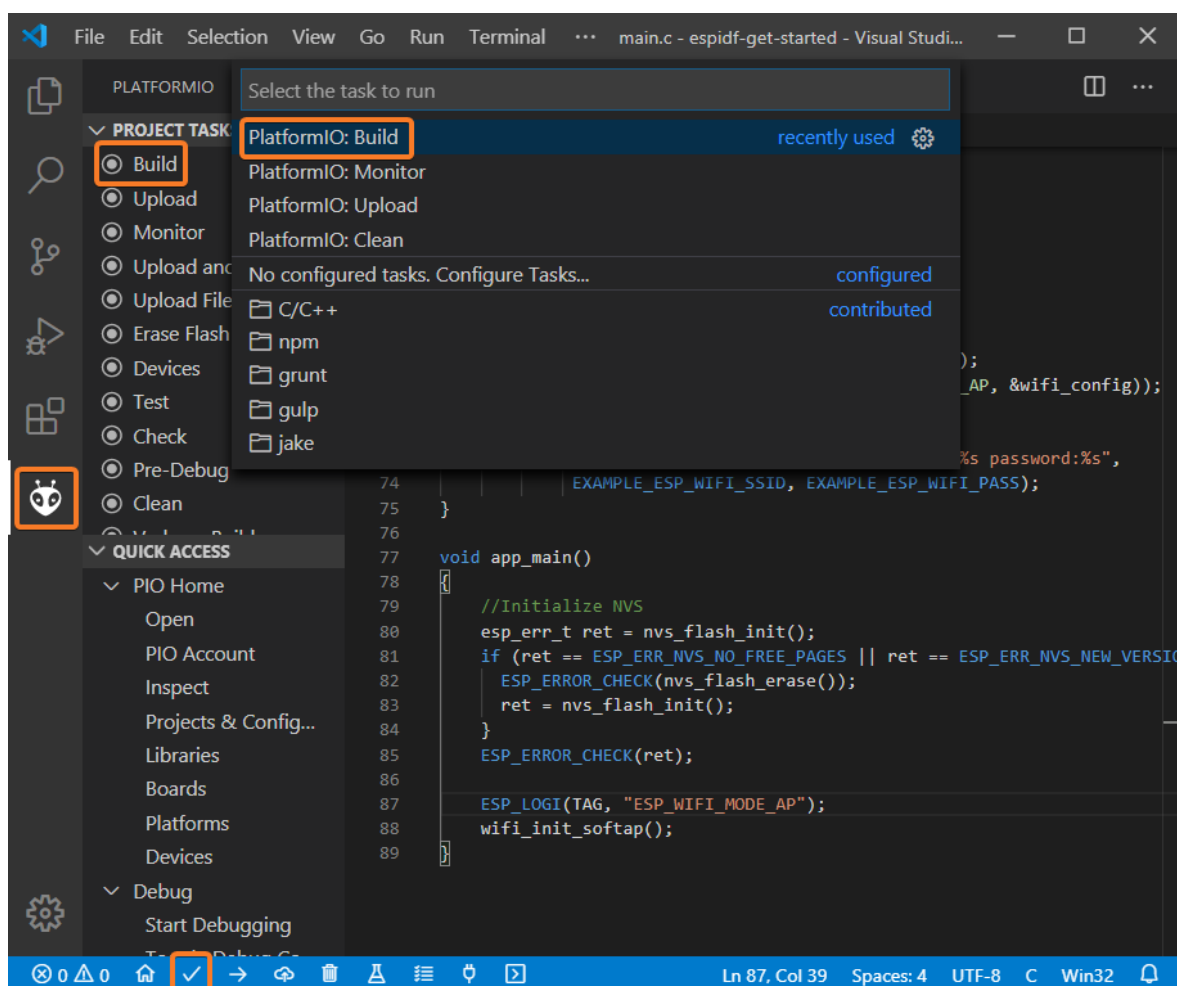


图 3.6 PlatformIO 插件运行在 VS Code

VS Code 和 PlatformIO 让硬件代码调试变得非常简单，如图 3.7，在代码行号附近打上断点，图上 588 行，然后点击调试，呈现在眼前的最左侧是各个作用域的变量值、函数调用堆栈、断点、源码文件，甚至处理器上各个寄存器的值，中间的是正在执行的 CPP 代码，最右侧是运行在当前平台的处理器汇编代码。下方是终端监视器输出的内容。总之，VS Code 和 PlatformIO 的结合在感受上如同使用编辑器一样轻便，而在功能上如同使用集成化 IDE 一样强大。

调试过程中的各种操作如图 3.8，启动、终止调试，跳入跳过跳出等。

— 17 —

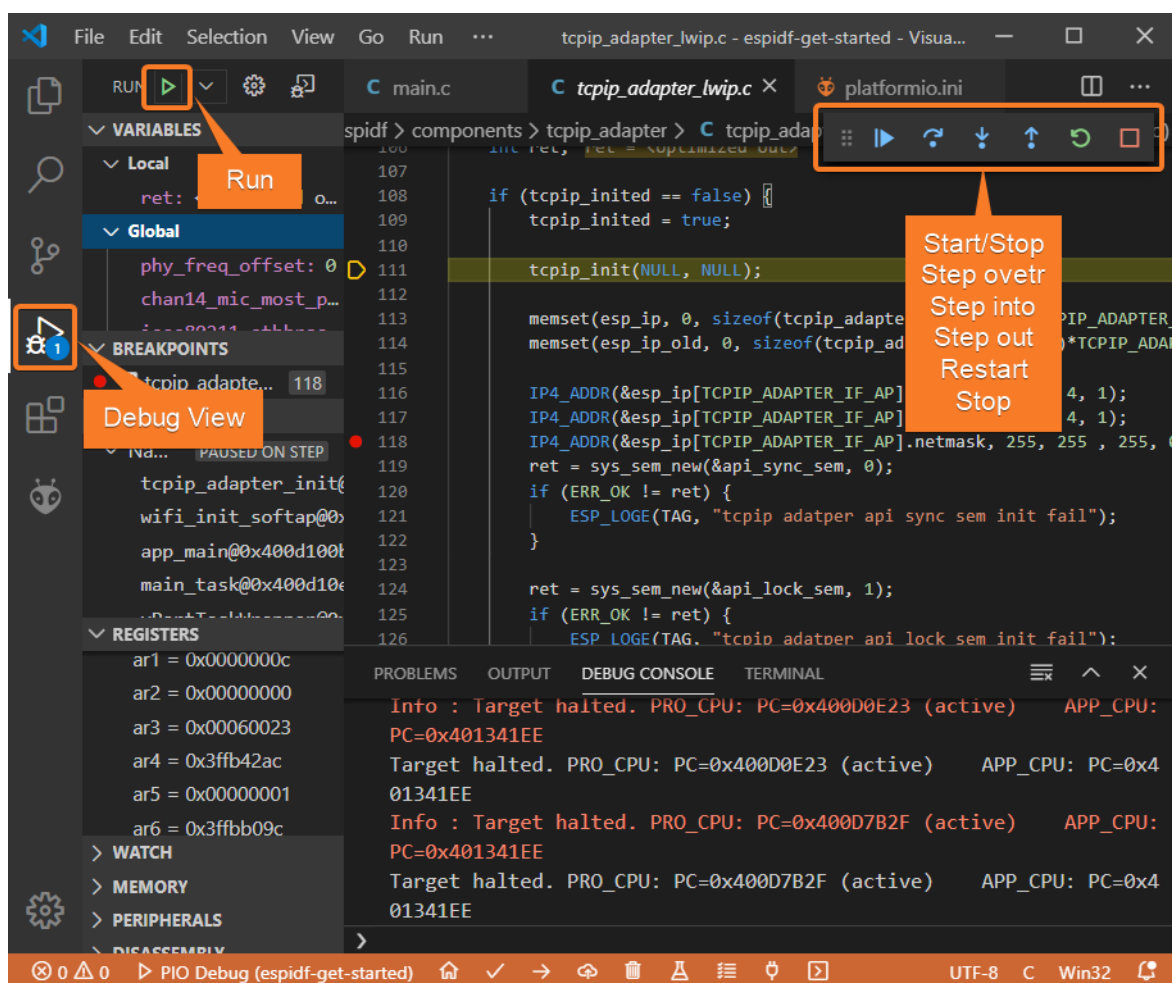


图 3.8 VS Code 在目标平台调试代码

4 计步数据采集节点设计

计步数据采集节点处于整个系统的头部，作为运动数据的输入端，精准采集并即时同步的发送传感器原始数据是精准计步的基础。

4.1 采集节点简介

采集节点包括两部分，ESP8266 和 MPU6050，ESP8266 作为主控单片机，MPU6050 作为运动数据采集传感器，它们之间通过 I2C 连接通信。

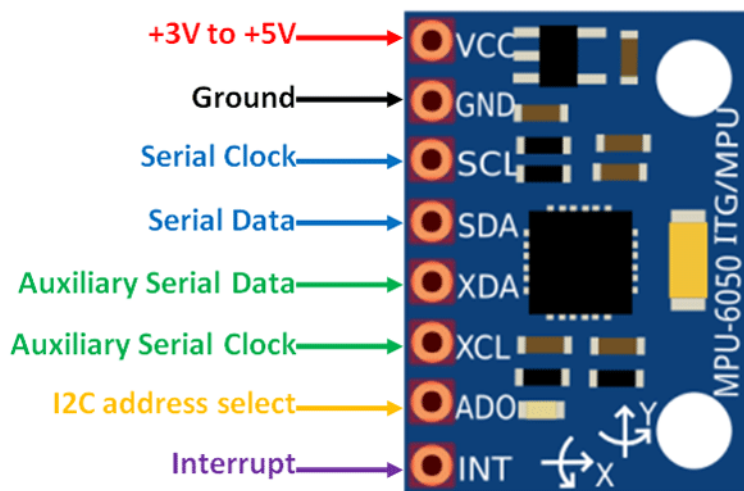


图 4.1 MPU6050 模块引脚

MPU6050 是一个整合了三轴加速度和三轴角速度并内置硬件 DMP 的六轴运动数据处理器件。如图 4.1 所示，它的供电引脚是 VCC GND，内置了稳压芯片可以接受 3.3V-5V 的供电电压。SCL 和 SDA 是它作为从设备时候进行 I2C 传输的线路，将接入主控单片机 ESP8266 的 I2C 从设备接口，XCL 和 XDA 是当传感器作为主设备，一般是外接一个磁场传感器，通过测量地磁场矢量的大小和方向可以确定当前传感器的空间方向，这样就组成了九轴运动数据处理器件。地址引脚 AD0 和中断引脚 INT 除非有特殊要求一般不接。除了可以采集运动数据，还可以采集传感器所处环境的温度，加速度和陀螺仪每个轴每个样本将占用 2Bytes 空间，温度也占用 2Bytes 空间，因此在不外接磁场传感器的情况下要存储一个样本的所有原始数据要占用 14Bytes 的存储空间。

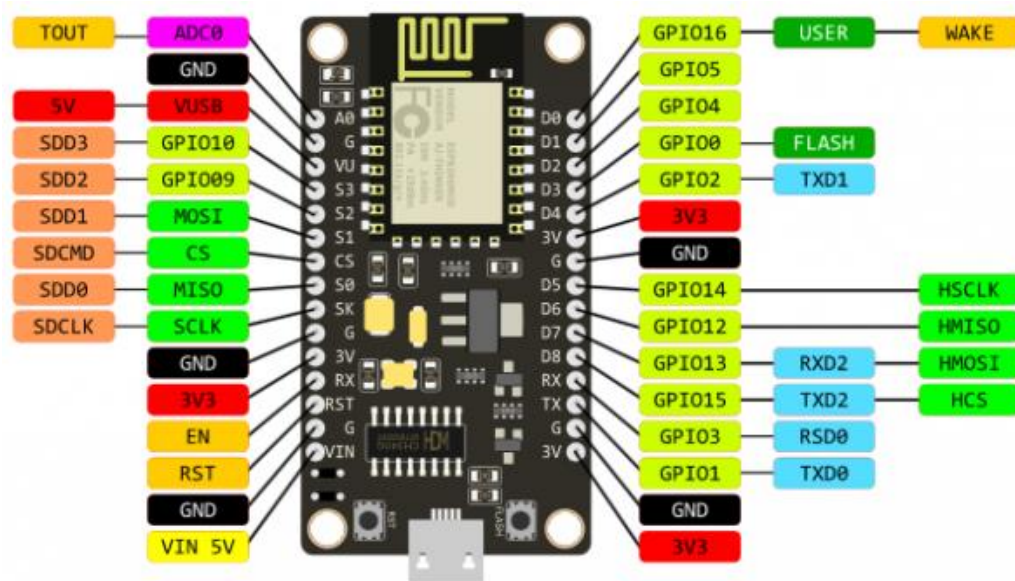


图 4.2 NodeMCU 模块引脚

ESP8266 的评估板 NodeMCU 作为一个控制型无线单片机具有丰富的接口 SDIO 2.0、主机 SPI 和从机 SPI、UART、I2C、I2S、IR Remote Control、PWM、GPIO 等，MPU6050 的 I2C 接口将接入 ESP8266 默认的 I2C 接口 D0 和 D1。ESP8266 具有无线网络的 WiFi 协议栈，可以工作在三个模式下，Station（客户端模式）是将 ESP 模块当做终端设备连接 WiFi，相当于普通手机连接 WiFi 的模式，WiFi 接入 K2P 路由器，AP（Access Point 接入点模式）是将 ESP 模块当做无线路由器，Station+AP（两种模式共存）相当于中继 WiFi，连接别的 WiFi 并且自己发射 WiFi 信号。本系统将使用 Station 模式，接入家用无线路由器，在这个局域网内，只要没有路由器的防火墙隔离，任何设备之间都可互相通信，因此 ESP8266 在从 I2C 接口获得数据之后，指定要发送的设备 IP 地址和进程对应端口号即可发送数据，也可仅指定端口，通过广播的方式对局域网所有的在线设备指定程序（端口号）发送传感器数据。

4.2 运动传感器工作原理

4.2.1 陀螺仪

陀螺仪，是用于测量物体旋转的角速度矢量的运动器件，也就是用于检测角度变化快慢的元件。其工作原理如图 4.3，陀螺仪的传统结构是内部有个三轴陀螺 X，Y，Z，图中 R_{xy} ， R_{yz} ， R_{xz} 各自夹角在一段时间内的变化除以这段时间就是每个轴对应的角速度。陀螺仪的工作原理是，物体在旋转时候轴的指向不容易变化，人们根据旋转轴不改变的特性，并通过很多方法去读取旋转轴所指示的方向（按右手螺旋定则，大拇指方向为矢量方向）和旋转的快慢，并将旋转轴的选择大小和方向数据信号传给控制系统。

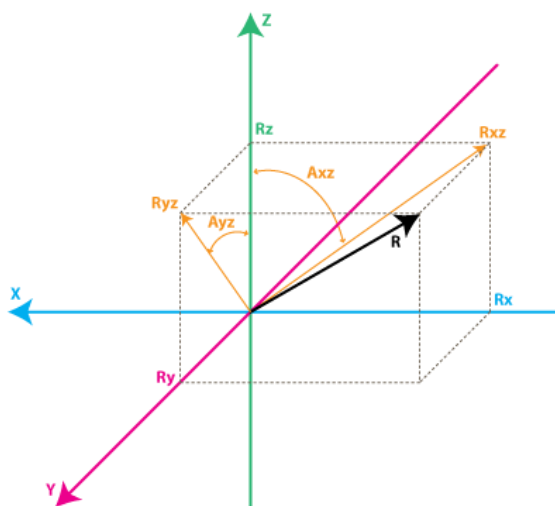


图 4.3 陀螺仪模型

4.2.2 加速计

加速计，也叫加速度传感器，是用来测量加速度矢量的一种运动器件。加速计通常由质量块、阻尼器、弹性元件、敏感元件和适调电路等部分组成。传感器在加速运动过程中，根据牛顿第一定律，物体保持原有运动状态不变，理论变成实际就是图 4.4 中质量球保持运动状态不变而和装质量块的外壳有了速度差和相对位置移动，从而导致质量块对外壳有了压力，外壳内壁有敏感元件，敏感元件测出压力（由电压值通过 ADC 转化为具体压力值）大小，再根据质量块的质量利用牛顿定律计算出具体加速度值。根据传感器敏感元件的不同分为电容式、电感式、应变式、压阻式、压电式这几类。

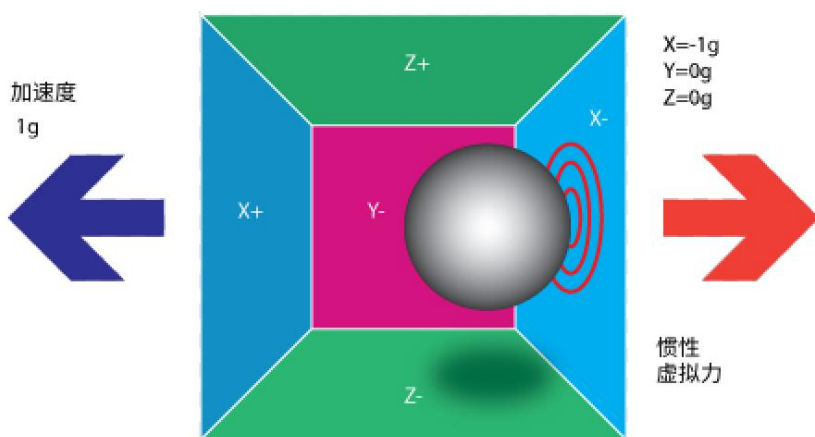


图 4.4 加速度模型

4.3 ESP8266 模块介绍与采样实现

ESP8266 是整个计步数据采集节点的核心部分，作为物联网中的一个节点以及一

个控制型 MCU，需要起到承上启下的作用，对下读取采集 I2C 接口的运动传感器原始数据，对上向主机发送传感器数据。

4.3.1 ESP8266 开发流程和方法

ESP8266 是一个可以集成在很多物联网开发板上，比如智能音响智能插座等，但在开发的时候需要使用串口连接电脑与电脑通信，用于烧录可执行程序，并输出一些串口调试信息，因此需要一个串口 TTL 转 USB 的模块或芯片，能直接连接到电脑 USB 口，一般使用的是 TTL 转 USB 模块，但 NodeMCU 是 ESP8266 的一个评估开发板，它自带了 TTL 转 USB 上的 CH340 芯片，而对电脑的接口就是 USB，因此可以省去 TTL 转 USB 接线，插入电脑即可识别。

在 Linux 中，将串行设备插入 USB 后会在/dev 目录下新出现一个 ttyUSB0 设备，这个设备由 Linux 文件系统虚拟出来的 USB 终端设备，基于 Linux 一切皆文件的 VFS 的优点，对此文件进行读写系统调用即可对串口设备发送或读取数据，即使用 read 对 /dev/ttyUSB0 文件进行读取即可读取串口设备数据，使用 write 将内容写入 /dev/ttyUSB0 文件即可向串口发送数据，VFS 隐藏了硬件设备细节，使得应用编程更容易。

当串口设备 ESP8266 没有产生任何数据时候你将读不到任何内容，因此 ESP8266 上需要运行一个程序来完成对 ESP8266 上 I2C 从传感器 MPU6050 的数据采集，并在采集的单个周期内将原始数据组包发送到局域网的指定主机的指定进程（端口）。

由于嵌入式设备资源有限，这个程序需要在 PC 端编写、交叉编译，并使用 USB 线上传至 ESP8266 的 Flash 存储芯片，之后重启模块程序可自动运行。ESP8266 的程序编写和交叉编译和串口调试均使用上述章节提到的 VS Code 以及 PlatformIO 插件完成。

编写 ESP8266 程序的第一步是 New Project 输入 Project name 并选择 Board 为 NodeMCU 1.0，Framework 选择 Arduino 框架。新建后将产生 5 个目录 include、lib、src、test 和 .pio 目录，以及配置文件 platformio.ini 和 README 文件，其中除 .pio 目录是新建项目一些构建和依赖库如 Arduino 框架的依赖，其他目录则是留给开发人员使用的。

对于本次 ESP8266 程序设主要目录有驱动相关库文件包括 I2C 设备驱动、MPU6050 设备驱动的头文件放入 include 目录，CPP 源码文件包括 Arduinio 主程序文件放入 src 目录。

4.3.2 I2C 协议和 I2C 库

I2C 总线是是飞利浦半导体（现为 NXP Semiconductors）于 1982 年发明的一种同步，多主机，多从机，分组交换，单端，串行 计算机总线。I2C 总线只需要串行时钟线 SCL 和串行数据线 SDA 组成，两线都是双向 I/O 线。I2C 在标准模式下可达

100Kbit/s，超高速模式 5Mbit/s。

该总线是一种多主控总线，即可以在总线上放置任意多主节点。此外，在停止信号发出后，一个主节点也可以成为从节点，反之亦然。

如时序图 4.5 所示：

(1) 在 SCL 保持高电平的同时，通过将 SDA 拉低作为启动信号（S）来启动数据传输。

(2) SCL 被拉低，并且 SDA 设置第一个数据位电平，同时保持 SCL 为低（在蓝色条形时间内）。

(3) 当 SCL 上升到第一位（B1）时，对数据进行采样（接收）。为了使位有效，SDA 不得在 SCL 的上升沿和随后的下降沿（整个绿条时间）之间改变。

(4) 重复此过程，在 SCL 为低电平时 SDA 转换，而在 SCL 为高电平（B2, ... Bn）时读取数据。

(5) 最后一位之后是时钟脉冲，在此期间，SDA 被拉低以准备停止位。

(6) 当 SCL 上升作为停止信号（P）的条件，随后 SDA 上升。

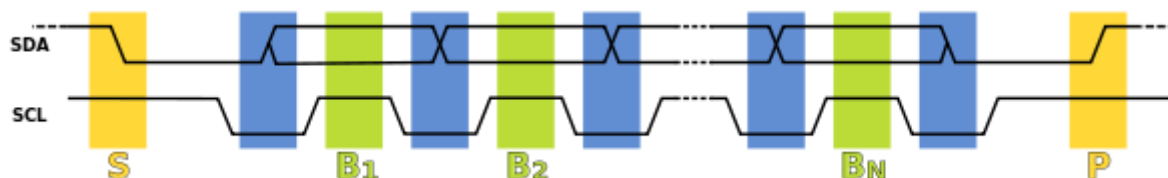


图 4.5 I2C 通信时序图

通过查看 I2C 库的代码可知，库中主要包含两个类，如图 4.6，I2Cdev 类是上层的软件类，I2Cdev 类只包含一些读写函数，几乎没有属性信息，主要实现上层读写的软件业务逻辑，比如 read/writeBit/Bits/Byte/Bytes/Words()等。如图 4.7，TwoWire 类是底层的对 I2C 进行时序控制的物理层类，主要实现一些 beginTransmission()、endTransmission()、和数据传输 Receive()、Send()等一些函数用于操作电平信号，并有一些缓冲区。

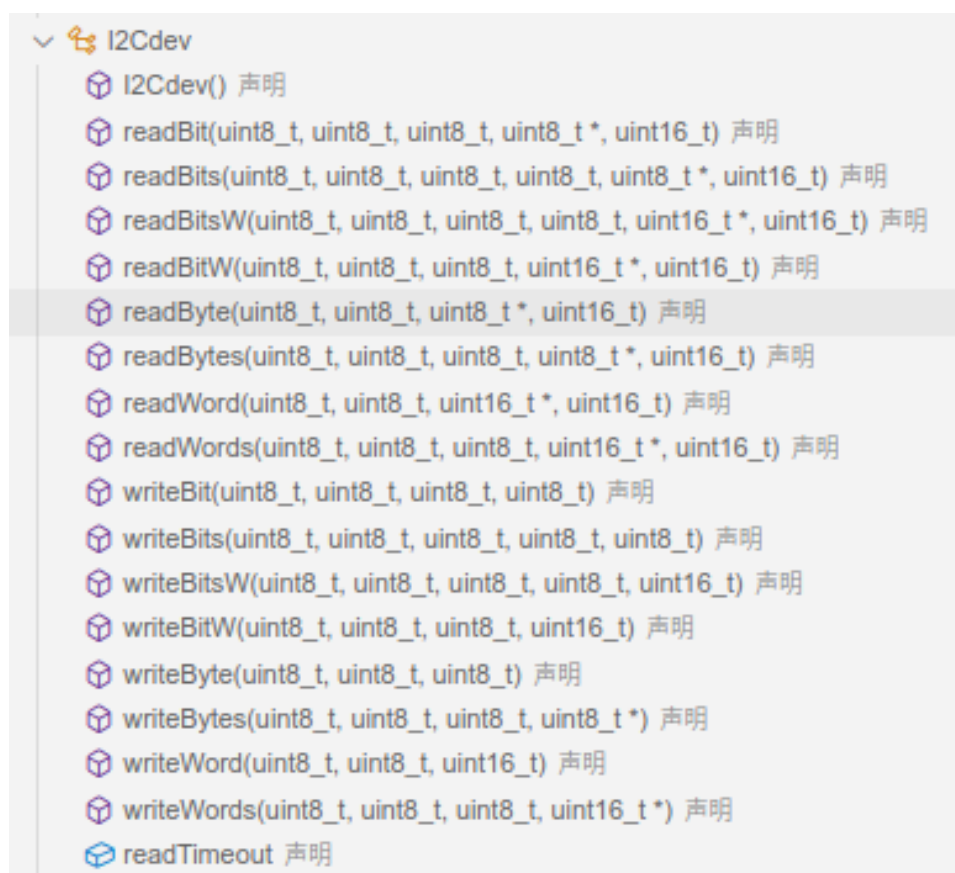


图 4.6 I2Cdev 类

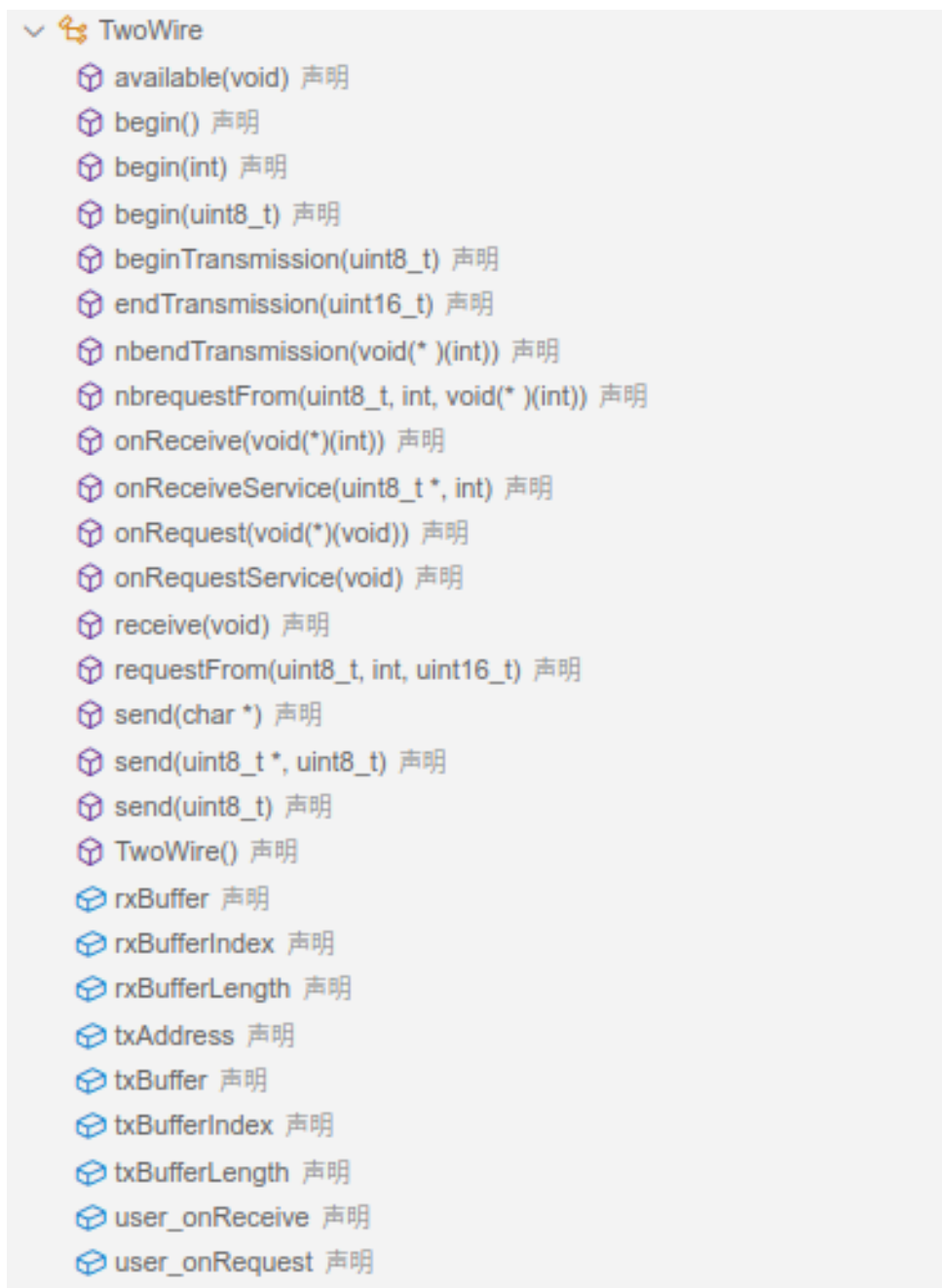


图 4.7 TwoWire 类

I2C 的使命就是读写传输数据，那么最重要的接口函数当然是 read/write，接口函数中有如此多的 read/write() 只是为了方便读取一个或者多个数据而已，最终实现的是 I2Cdev 类的 readBytes() 和 writeBytes()，从函数字面意思就是读取和写入若干字节数据。图 4.8 中，左边主要表述了 readBytes() 函数的整个 I2C 数据读取流程，函数需要传入设备地址、寄存器地址、数据长度、数据指针和超时时间。读取数据的整个流程是，先通过 TwoWire() 对象的 beginTransmission() 操作并传入总线上的从设备地址启动总线，再通过 TwoWire() 对象的 send 操作发送寄存器地址，使用 endTransmission() 结束寄存器地址传输，总线暂停。再次用 beginTransmission() 启动总线，requestFrom() 传入设备地址和数据长度，通过 for 循环中的 Wire.read 依次读取数

据，读取完成用 `endTransmission()` 结束总线。

图 4.8 中，右边边主要表述了 `writeBytes()` 函数的整个 I2C 数据写入流程，函数需要传入从设备地址、寄存器地址、数据长度和数据指针。写寄存器的过程稍简单于读寄存器，因为写地址和写数据都是主机到从机的单方向通信，只需要 `beginTransaction()` 开始传输，发送寄存器地址后可以直接在 `for` 循环中依次发送数据，发送完成，结束传输关闭总线。

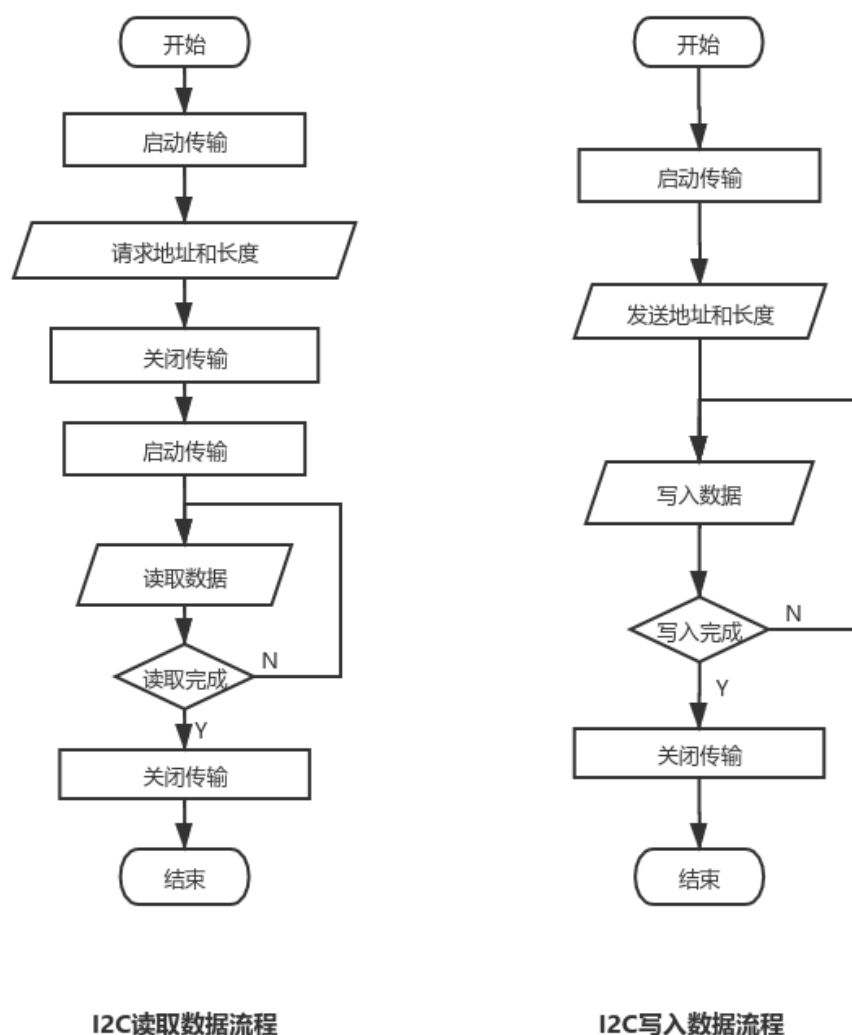


图 4.8 I2C 读写流程

从以上两核心函数来看，实现 `I2Cdev` 类就是通过调用物理层类 `TwoWire` 相应的接口完成整个设备驱动的实现。

计步监控系统用到了 MPU6050 的驱动库，这个驱动库包含了对 MPU6050 的一系列操作，设置所有的寄存器、配置 DMP 库等等，这些操作都依赖于主控芯片对 MPU6050 在 I2C 总线的操作，通俗来讲，读取寄存器值就是主控芯片通过向 I2C 发

送给定的寄存器地址，可以读取寄存器值，设置寄存器就是通过 I2C 向 MPU6050 发送地址和新值后，将更改寄存器原值，因此，MPU6050 驱动库依赖于 I2C 库。

I2C 被广泛的应用于控制芯片间通信，在计步监控系统中，应用在运动传感器 MPU6050 和单片机主控 ESP8266 之间的通信，采集前对 MPU6050 做简单的初始化，开始采集后主要的通信方向是数据从 MPU6050 到 ESP8266，I2C 总线将作为本系统的重要传输通道。

4.3.3 采样率的选择

主控单片机 ESP8266 要发送原始数据必须先获得原始数据，获得一次原始数据的操作称为采样，即采集一个样本，但人类步行活动是一个持续性的动作，因此需要持续采样，那么选择一个合理的采样率将是一个很关键的步骤，这将关系到计步的准确性。若采样率过高会导致采样处理器负载较高，同时功耗增大，但采样率过低则会影响样本密度下降从而导致计步的数据可信度下降。

众所周知，采样频率要高于步行频率，这样才能用算法识别到步数，不过每个人的步行频率又是不同的，有研究^[3]表明步行频率和人的身高有关，具体关系如表 4.1。

表 4.1 步行频率和身高的关系

每 2 秒步数 (m/s)	跨步 (m/s)
0~2	身高/5
2~3	身高/4
3~4	身高/3
4~5	身高/2
5~6	身高/1.2
6~8	身高
>=8	$1.2 \times \text{身高}$

正常情况下，人们的跑步最快速度最快每秒 5 步，最慢每秒半步，也就是说最小的步行时间间隔为 200ms，最大间隔 2000ms，步行频率为 0.5Hz-5Hz 之间，那么最大的采样率不得低于 5Hz，也就是每次采样间隔不得高于 200ms。不过在算法上，必然需要用到均值滤波器，一般以 4 个样本作为均值滤波样本较为合理，那么 4 个样本合为一个样本，因而 $200\text{ms}/4=50\text{ms}$ ，采样间隔不得高于 50ms，当然采样率越高数据越精确，这是必然的，但是考虑到 ESP8266 属于嵌入式设备、穿戴设备，并使用有限电源供电，在合理考虑 CPU 功耗和并适当提高采样率的情况下，并对采样准确性做了多次测试（后续章节将提到），采样间隔 40ms，采样率为 25Hz 是最恰当的。

4.3.4 设备初始化和采样实现

如 4.9 流程图所示，采样方法使用了之前章节提到的 Arduion 框架和 MPU6050

驱动库在全局作用域定义 MPU6050 对象，因为使用面向对象的方法，本系统中只有一个 MPU6050 设备，因此只声明这一个对象。在 `setup()` 函数中对 MPU6050 对象进行初始化 `MPU6050.initialize()`，并进行 I2C 的连接测试 `MPU6050.testConnection()`，这样，MPU6050 设备（对象）初始化完成。进行循环采集的代码则需要放入 Arduino 框架的 `loop()` 函数中，之前介绍过的 MPU6050 驱动库有众多方便的 API，此处获取 MPU6050 设备的 6 轴原始数据使用 `MPU6050.getMotion6()`，根据 MPU6050 模块和驱动库手册配合驱动代码来看，每个轴原始数据需要用 2 Bytes 来存储，每个轴需要 2 Bytes 空间，`getMotion6` 需要传入 6 个这样的 2 Bytes 内存大小的指针来保存 6 轴数据。使用 `MPU6050.getTemperature()` 过的当前模块的温度，同样用 2 Bytes 存储，通过函数返回值接收。

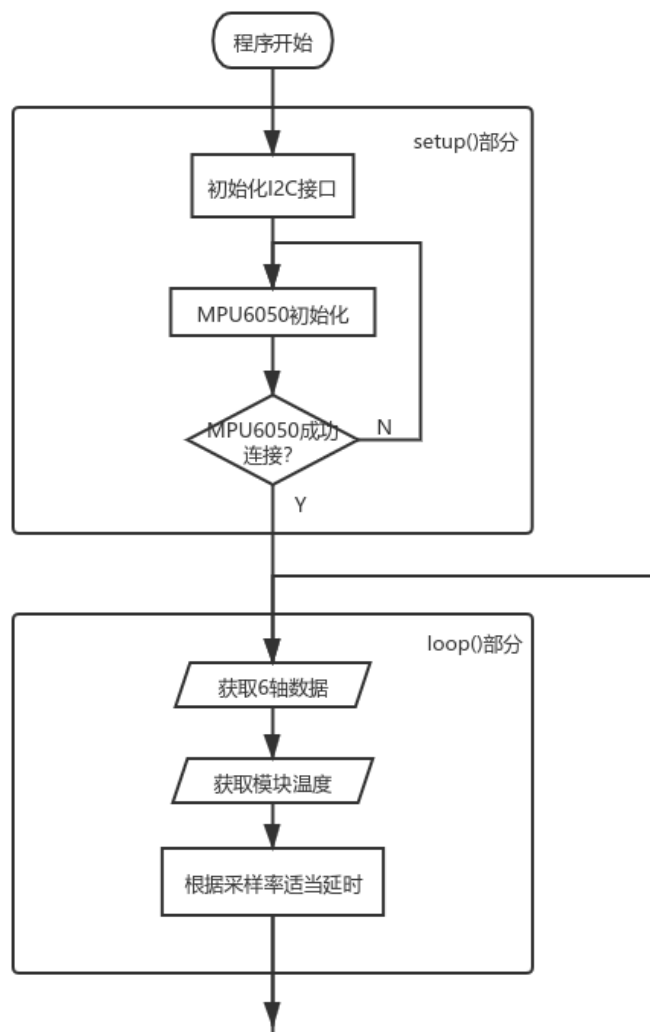


图 4.9 Arduino 框架传感器数据采集过程

之前叙述的采样率使用 25 Hz，如果忽略采样代码运行时间，那么需要在采样结束后，`loop` 循环结束之前延迟 40ms。至此，一轮采样结束，可通过网络相关库发送本轮样本并继续 `loop` 循环采样。

4.4 无线传输设计与实现

4.4.1 网络协议栈结构和应用层协议设计

lwIP 是一个轻量级 IP 协议栈，在没有操作系统的情况下也可以运行。轻量级的 TCP/IP 协议栈实现的重点是减少内存资源使用。这使得 lwIP 在具有 20KB 的 RAM 和大约 40KB 的 ROM 的嵌入式系统，这使 lwIP 协议栈更适合在低端的嵌入式系统中使用，支持常见的协议 IP、IPv6、ICMP、ND、MLD、UDP、TCP、IGMP、ARP、PPPoS、PPPoE 等。

ESP8266 通过调用 lwIP 的网络 API，实现了 ESPAsyncUDP、ESP8266WiFi 等网络相关的类，同时在 PlatformIO 配置文件中需要声明 lib_deps = ESPAsyncUDP，这样即可使用 ESPAsyncUDP 作为传输层协议，使用 ESP8266WiFi 作为传输介质，实现计步数据采集节点的无线信息传输。

当然传输链路已经准备好了可以传输数据，但是传输的数据内容需要自己定义，这就需要自己规定应用层的传输协议了。根据上述提到的采集数据的获取和储存，加速度和陀螺仪共 6 个 2 Bytes 数据域，和一个 2 Bytes 温度数据，应用层协议组包至少需要 14 Bytes，如图 4.10 就是最节约网络带宽的数据格式。

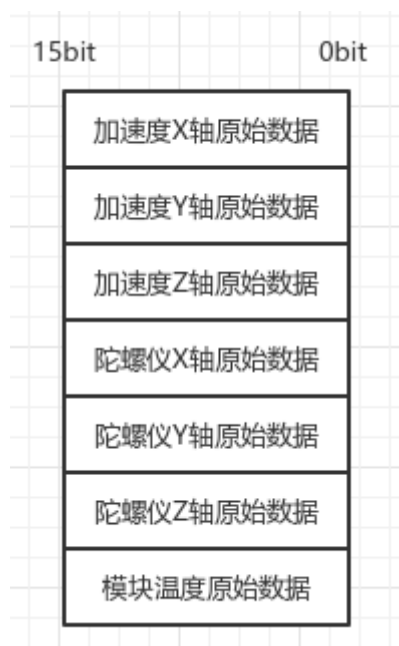


图 4.10 数据包格式

4.4.2 无线网络传输范围和发送广播域

无线 WiFi 信号的连接质量由发射无线网的路由器和连接无线网的终端设备通过决定，经 ESP 代理商测试^[6]，使用 K2P 路由器和 ESP8266 模块连接通信，在 300 米内可以保持可靠连接，丢包率小于 1.3%，超过 400 米时信号会丢失。在家庭物联网中，这个距离已经足够了。

只要连接路由器的设备,都在 192.168.1.* IP 的局域网内,如 ESP8266 连接 WiFi 后使用 send 或 sendTo 函数可以发送到已连接或者指定主机 IP 和端口,使用 broadcast 或 broadcastTo 函数可以发送到当前广播域,也可指定端口。

4.4.3 WiFi 配置和异步 UDP 广播发送流程

根据面向对象的理论基础,ESP8266 只能连接一个 WiFi,因此需要初始化一个 WiFi 对象,如图 4.11,先使用 mode 函数来设置 WiFi 工作模式为 STA 模式,用 begin 函数传入 WiFi 的名称和 WiFi 的密码,最后用 waitForConnectResult 函数等待连接结果即可(为了增强程序的高可用性,这里可使用循环连接,连接成功则出循环),连接成功返回 WL_CONNECTED。

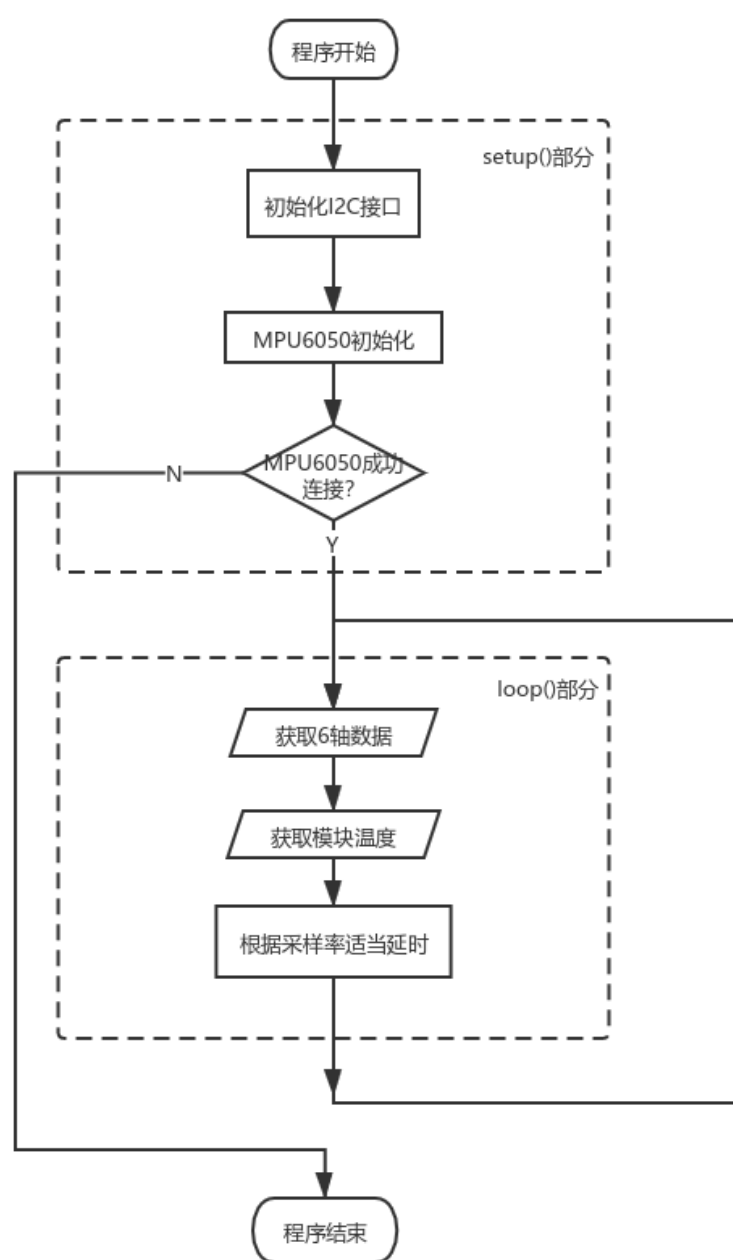


图 4.11 Arduino 框架传感器数据无线发送过程

连接 WiFi 后进入 `loop` 循环采样，如同上述应用协议所描述，循环内创建一个存储传感器原始数据的对象，当前对象存储的是当前 `loop` 轮采样的数据，通过库中的 `get*()` 函数族进行采样，并将数据保存至数据包对象中，采样后直接通过 `broadcastTo()` 广播发送并指定数据包指针和大小，以及服务端口号。

5 物联网主机设计

5.1 物联网主机简介

5.1.1 硬件模块组成

计步监控系统的其中一个应用场景是家庭物联网系统，家庭物联网中需要有一个具有较高并行运算能力的主机作为物联网中央主控机，物联网主机选用树莓派 3B，物联网主机用于服务每个物联网终端节点，如计步数据采集节点。

物联网主机由树莓派 3B 主机和 Nokia5110 液晶屏组成，主机作为核心运算设备，处理来自各个家庭中各个物联网终端节点的信息，屏幕作为主机状态或者某些节点的一些简单信息显示，本计步监控系统中将用来显示当前计步数据采集节点当前的加速度值、当前步数。

3B 模块实物如图 5.1，由于物联网主机需要连接多个物联网终端节点并为之服务，因此需要具有并行运算能力的多核多线程的处理器，3B 拥有四核 1.2GHz Broadcom BCM2837 64 位 CPU，可以满足一般的家庭物联网服务，其他更多关于 3B 的资源接口在之前章节已经叙述。

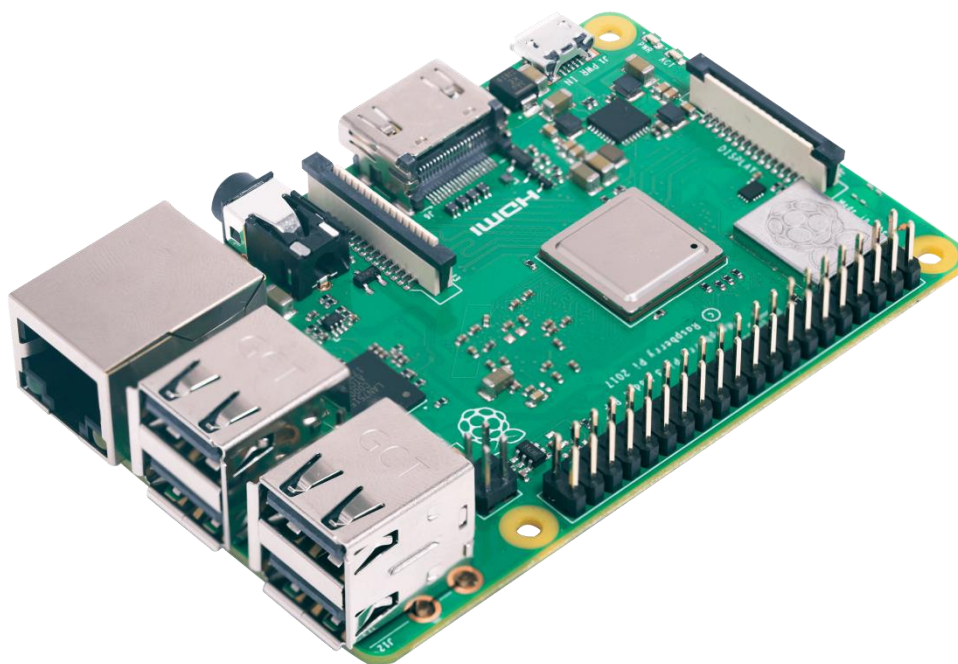


图 5.1 树莓派 3B 模块实物图

Nokia5110 LCD 屏幕（图 5.2）最早被用于 1998 年生产的诺基亚手机屏幕，分辨率为 80*48 像素，液晶模块使用 LPH7366，主控芯片使用 PCD8544。Nokia5110 具有较高性价比，速度比 LCD1602 和 LCD12864 都快，接线方便，使用简单，在不适用

背光的情况下只需要四根线。树莓派 3B 和 Nokia5110 使用 SPI 作为通信协议进行通信，使用 Raspbian 系统自带的 fbtf_device 驱动作为 Nokia5110 的驱动。



图 5.2 Nokia5110 LCD 模块实物图

5.1.2 软件模块组成

物联网主机安装的是 Linux 的基于 Debian 发布版 Raspbian 系统，在已安装操作系统的硬件上可以直接执行可执行程序，因此物联网主机上的最终发布的是一个 Linux 可执行程序。物联网主机上的计步程序在 Linux 下拥有可执行权限后即可运行，由于设置为后台守护进程，因此需要 root 权限运行。

计步程序除了用到的 C 语言自身的一些函数和 Linux 自带的 glibc 库外，没有用到其他的第三方静态库或动态库。

计步程序根据业务逻辑将被设计为 5 个模块，出主模块 main.c 外，每个模块均有一个同名的*.c 和*.h 文件，在*.c 中实现相关任务函数，在*.h 中声明函数和一些全局变量、宏和一些数据结构，除此之外，单独一个 common.h 头文件用于包含常用的系统头文件，和一些与各个模块业务无关的全局宏和全局数据结构。Makefile 文件用于定义编译这些源码的编译规则。

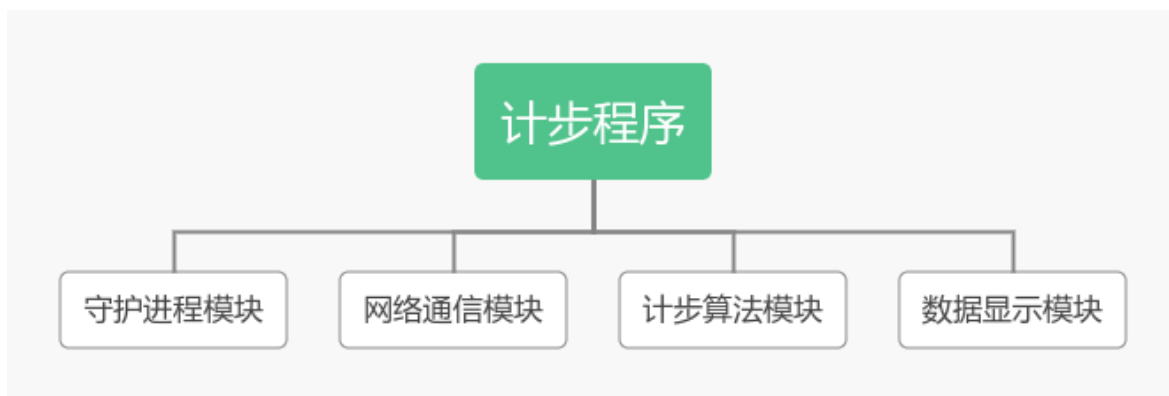


图 5.3 软件模块结构图

根据计步程序的需求来看，需要与计步数据采集节点进行通信的网络模块 network 模块，各种数据处理和计步算法 pace 模块，在 LCD 液晶屏显示相关的 display 模块，要求切换为守护进程守护运行的 daemon 模块，可以设计出图 5.3 所示的模块结构。这些文件将由 arm-linux-gcc 编译器编译为对应的*.o 目标文件，并由 arm-linux-ld 链接器链接为可执行程序，这个可执行程序就是物联网主机上的计步程序。

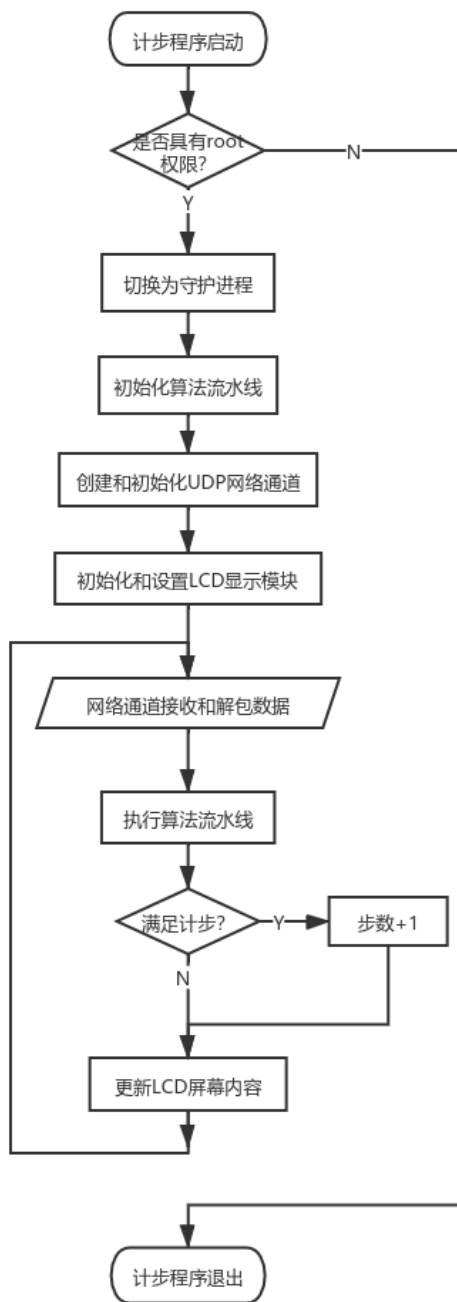


图 5.4 计步程序流程图

如图 5.4 计步程序的基本流程是先将当前进程设置为守护进程，使用 `daemon` 模块的 `init_daemon` 函数实现，然后是各种算法流水线的初始化，初始化函数均在 `pace` 模块中实现，最后初始化网络通信部分，在 `network` 模块实现，最后设置并初始化屏幕驱动，至此计步程序的相关初始化完成。之后进入算法的主循环，那么网络接收和解包一定在算法的主循环内，之后经过算法流程，判定计步的时间条件和空间条件是否满足，都满足则记为一步。

5.2 网络通信模块

计步数据采集节点和物联网主机的通信在传输层采用 UDP 方式，UDP 是面向无

连接的传输，计步程序的服务端一直处于监听或通信状态，计步数据采集节点也一直处于数据发送状态，无论哪一端断开或关机，都不会影响系统的稳定运行，只是断开过程中用户走的步数无法统计到，当开机后则继续正常计步，这是 UDP 的优点，只要明确目标主机，可以直接进行数据传输。

有关网络的部分都在 `network` 模块中实现，包括网络初始化，网络接收和解包，网络调试。

网络初始化函数主要完成 UDP Socket 服务器的创建，使用系统 API `socket()` 创建一个 `AF_INET`（IP 协议族）的 `SOCK_DGRAM`（UDP 数据报）套接字，并返回套接字文件描述符。之后设定允许接收的地址（本次设置为 `INADDR_ANY`，即任意 IP 可发送）和端口，这个端口就是这个进程的端口标志，在数据采集节点发送数据就依靠本端口来识别进程。最后使用 `bind()` 系统调用绑定设置的地址端口对象，完成 UDP 服务器初始化，初始化函数返回 `socket` 文件描述符。

UDP 接收函数接口封装输入参数为 `socket` 文件描述符，输出参数为三轴加速度结构体指针，返回值是这次接收数据字节数，函数内使用 `recv()` 系统调用，通过给定的 `socket` 文件描述符和原始数据包结构体指针，接收数据，并判断是否成功接收，将原始数据指针的三轴加速度部分作为出参返回，将接收的原始数据包总大小作为返回值返回。

通过 `recv()` 系统调用接受数据的参数是 `void*` 类型，也就是没有声明指针类型，网络接受到的原始数据是二进制数据，通过指针的强制类型转换可以依照对应指针类型排列的内存空间来存储，但本系统调用没有规定指针类型，那么传入上述文中应用层协议规定的协议的结构体指针即可，`recv()` 系统调用收到的数据将默认按照实参（应用协议结构体指针）类型依次填充结构体指针指向的内存，这样依据 C 语言指针具有类型并且指针可运算的特性，完成网络原始数据包的自动解包。

网络调试功能是在开发时期使用的调试函数，正常使用计步监控系统不会用到此函数。网络调试函数主要功能是在标准控制台输出接收到的数据源 IP、源端口和原始数据，只输出数据，不存储。和 UDP 接收函数不同，调试函数只需要拿到 `socket` 文件描述符，通过调用系统 API `recvfrom()` 函数即可拿到网络上的原始数据包和数据的来源 IP 地址和端口，这样就可以知道当前收到的数据来源于哪一个物联网终端节点。

5.3 计步算法模块概述

全部的算法部分内容较多，本节只做算法简介，下一章详细叙述记步算法。

整个算法流水线分为 5 步，均值滤波、更新峰值、更新精度阈值、判断时间和空间条件满足则记一步、更新时间间隔。

第一步，使用平均值滤波器将多个样本进行均值化处理，滤波器采用循环队列的方式进行数据循环。

第二步，使用平均值滤波器的滤波结果进行峰值更新。

第三步，对比当前样本和最新样本的差值，若差值超过精度阈值则更新，同时返回本轮是否已更新。

第四步，若第三步精度值已经更新，则判断计步是否满足时间条件和空间条件，都满足则记一步。

第五步，若第四步确认记一步，则更新时间间隔，以便于统计下一步是否满足时间条件。

算法流水线结束，进行下一轮算法。

5.4 数据显示模块

数据显示模块不使用任何第三方库，仅使用 Linux 自带的 fbtf_device 驱动即可，加载驱动，并使用 con2fbmap 命令即可将终端 console 映射到显示屏上，即可实现标准输出显示在屏幕上。

5.4.1 Nokia5110 与主机连接方式和 SPI 协议简介

SPI 总线（Serial Peripheral interface），就是串行外围设备接口。最初用在摩托罗拉的一些处理器上，现在经常用于小型的存储器例如 EEPROM 和 FLASH，和一些帧缓冲设备如显示屏。SPI 总线具有高速、全双工、同步通信的特点，具有主从和多从通信的特点，如图 5.5 是 SPI 单主机到单从机连接示例。

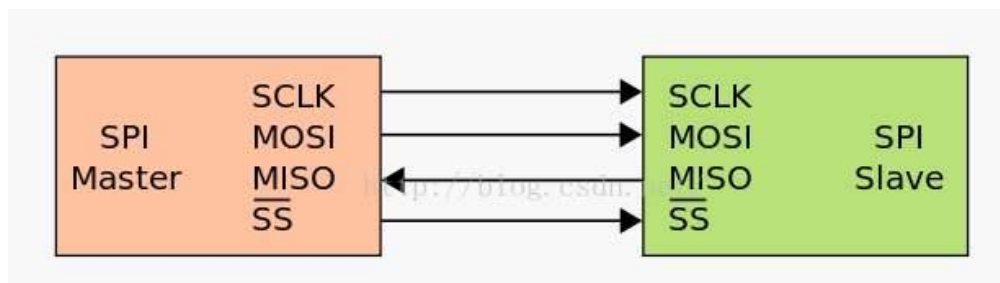


图 5.5 SPI 单主单从模式连接

如图 5.5，SPI 只需要 4 根线即可完成通信，作为一个短距离通信总线，这将大大节约存储器、显示器的 PCB 接线空间。

SCLK: 串行时钟

MOSI: 主机输出从机输入

MISO: 主机输入从机输出

SS: 片选信号，低电平有效

从通信线片选 SS 信号可以看出，SPI 总线支持多个从设备，但在多机中至少需要一个主设备。

上述通信总线的名称是目前大多数人接受的名词，SPI 总线较为古老，在旧的设备上名称可能有所变化，但通信线的功能不变，如图 5.6 是 Nokia5110 的引脚图，

Nokia5110 LCD 也是比较古老的一款屏幕，因此其通信线路与标准名称有所差异，与标准名称对比如下：

SCLK: CLK

MOSI: 空

MISO: Din

SS: CE

上述 MOSI 之所以为空，是因为 LCD 一直被当做从设备来使用，因此不需要 MOSI（LCD 主设备输出，线另一端作为从设备输入）。

其他引脚和 SPI 通信无关，但在 LCD 的使用逻辑上依然很重要。

Vcc: 3.3V 或 5V 供电

Gnd: 接底线

BL: 背光灯信号

Reset: 重置 LCD 屏幕

DC: 指令/数据选择信号，当低电平时表示传输指令状态，高电平时表示传输数据状态

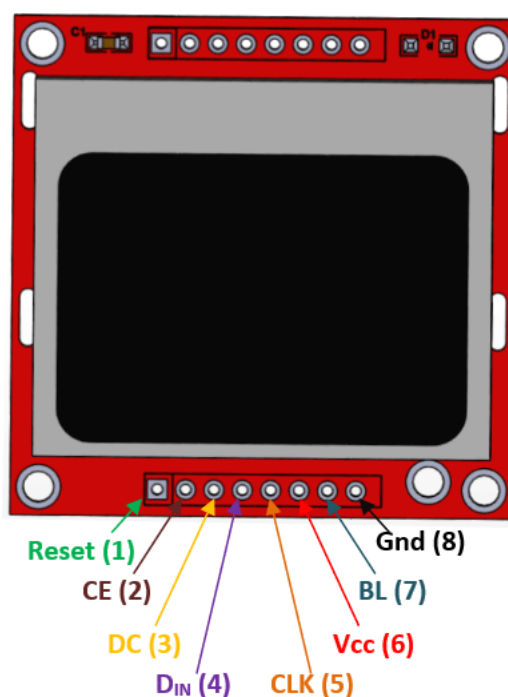


图 5.6 Nokia5110 引脚图

树莓派上有丰富的 GPIO 和 SPI 引脚，可以通过 `gpio readall` 命令查看引脚对应表如图 5.7，在开发板上的实物对照如图 5.8

Pi 3											
BCM	wPi	Name	Mode	V	Physical	V	Mode	Name	wPi	BCM	
		3.3v			1	2		5v			
2	8	SDA.1	IN	1	3	4		5V			
3	9	SCL.1	IN	1	5	6		0v			
4	7	GPIO. 7	IN	1	7	8	1	ALT5	TxD	15	14
		0v			9	10	1	ALT5	RxD	16	15
17	0	GPIO. 0	IN	0	11	12	0	IN	GPIO. 1	1	18
27	2	GPIO. 2	IN	0	13	14		0v			
22	3	GPIO. 3	IN	0	15	16	1	OUT	GPIO. 4	4	23
		3.3v			17	18	1	OUT	GPIO. 5	5	24
10	12	MOSI	ALT0	0	19	20		0v			
9	13	MISO	ALT0	0	21	22	1	OUT	GPIO. 6	6	25
11	14	SCLK	ALT0	0	23	24	1	OUT	CE0	10	8
		0v			25	26	1	OUT	CE1	11	7
0	30	SDA.0	IN	1	27	28	1	IN	SCL.0	31	1
5	21	GPIO.21	IN	1	29	30		0v			
6	22	GPIO.22	IN	1	31	32	0	IN	GPIO.26	26	12
13	23	GPIO.23	IN	0	33	34		0v			
19	24	GPIO.24	IN	0	35	36	0	IN	GPIO.27	27	16
26	25	GPIO.25	IN	0	37	38	0	IN	GPIO.28	28	20
		0v			39	40	0	IN	GPIO.29	29	21
BCM	wPi	Name	Mode	V	Physical	V	Mode	Name	wPi	BCM	
Pi 3											

图 5.7 Raspberry PI 3B 引脚图

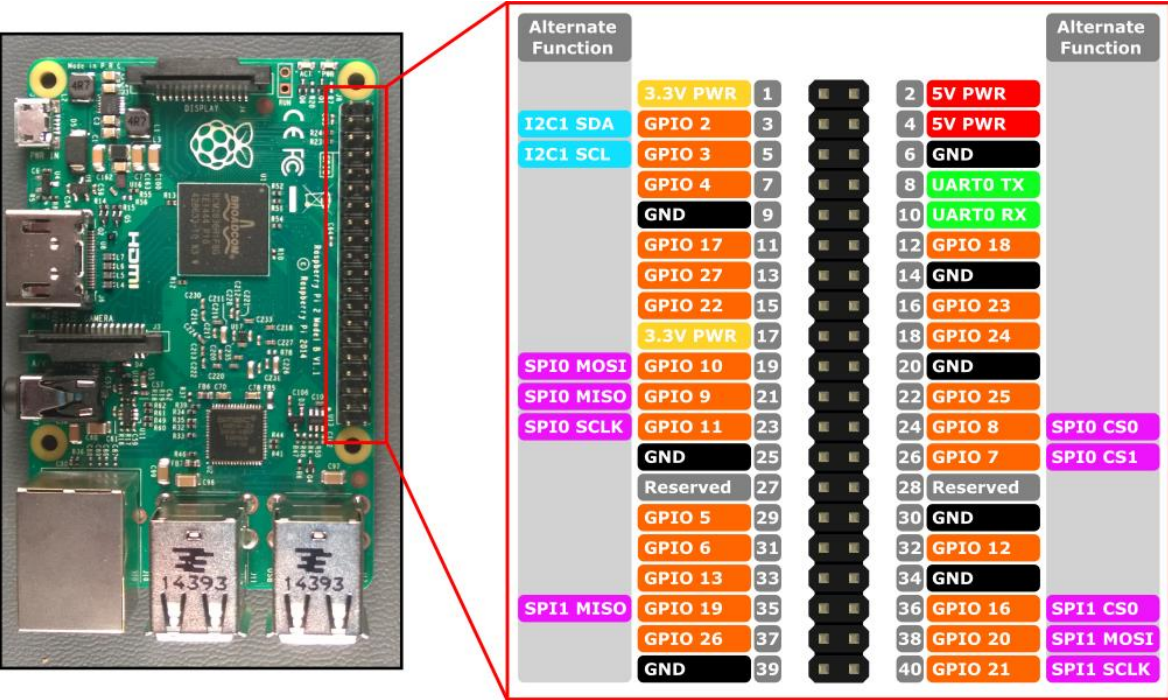


图 5.8 Raspberry PI 3B 引脚实物对照

图 5.7 最中间两列 Physical 是树莓派开发板上的两列物理引脚对应编号。再根据 fbttft_device 驱动文档^[7]，最终可以确定连接在树莓派的引脚对应关系如下表 5.1。

表 5.1 Nokia5110 与树莓派 3B 引脚对应关系

LCD 引脚	Raspberry PI	Raspberry PI Physical	备注
Gnd	GND	9	接地线
Vcc	3.3V	16	电源
CLK	SCLK	1	SPI 时钟
Din	MOSI	23	SPI 的 MOSI
DC	GPIO24	19	指令数据选择
CE	CE0	18	SPI 从机片选
Reset	GPIO25	24	重置 LCD
BL	GPIO23	22	背光

连接后如图 5.9 所示：

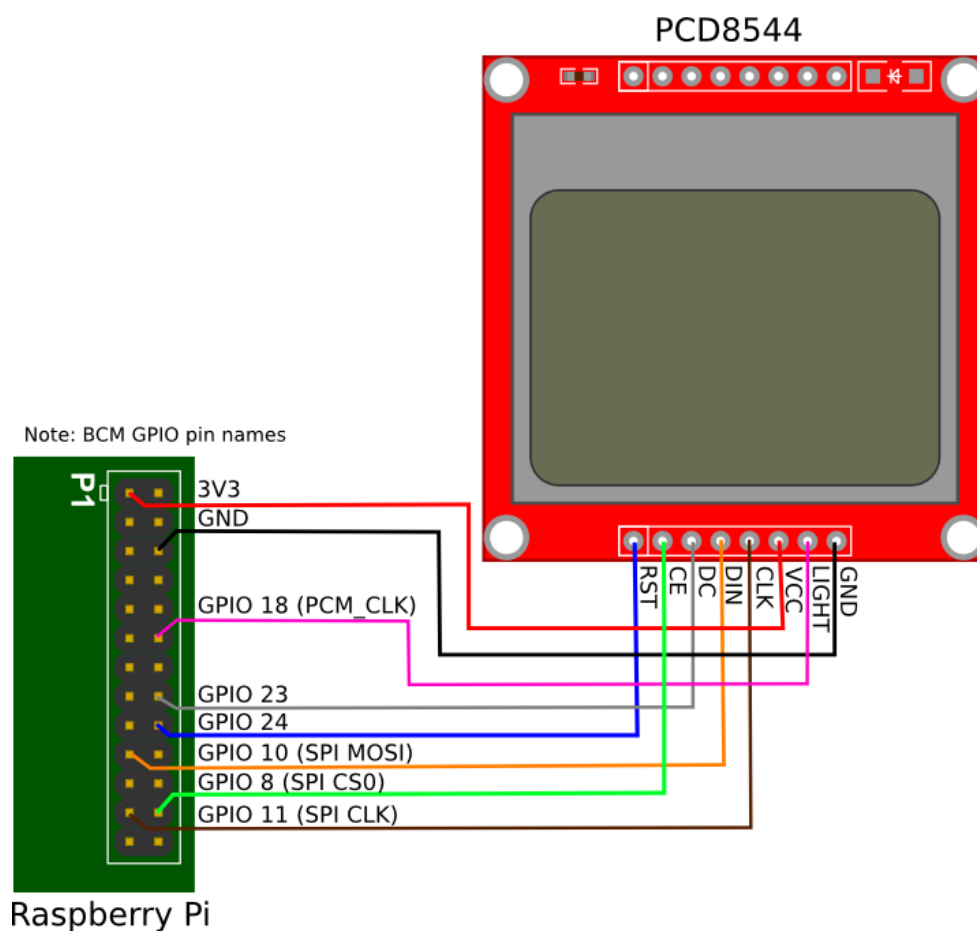


图 5.9 树莓派与 LCD 屏幕连接图

5.4.2 内核模块技术和 Frambuffer 驱动加载

Linux 是一个宏内核操作系统，可以在不重启系统的情况下加载插入或卸载删除内核代码，这样在无需重启操作系统就可以扩展内核的功能。

内核模块代码进入内核分为两种形式，一种是在构建内核的时候 `make menuconfig` 将指定内核模块设置为 Y，这个模块将集成进内核，最终生成的内核镜像包含此代码，将随着内核镜像启动一起启动。另一种方式是在在内核代码中只构建当前指定的内核模块，最终生成二进制文件.ko（kernel object）文件，通过 `insmod` 或者 `modprobe` 命令来插入内核模块。

在单独构建内核模块的时候需要有与模块将要运行的内核镜像为同一内核版本，否则 `insmod` 时候会出错，因为编译内核模块将会用到完整的内核代码。

几乎所有的 Linux 驱动都是通过动态插入内核模块或者编译进内核的模块来驱动硬件的，小型的 TFT 屏幕 Nokia5110 也不例外。

在 2015 年之前，Linux 下的 TFT 屏幕通过 Noralf Trønnes 先生在 github 开源社区开发适用于 TFT 小屏幕的 Framebuffer 驱动，自从 2015-01-19 之后，该项目被合并进 Linux 主线，也就是目前在内核网站（kernel.org）下载的源码已包含此代码，并默认编译为内核模块。

现在，在编译好的内核模块中我们可以很方便的找到 `fbtft_device` 内核模块，默认的内核模块放在 `/lib/modules/<内核版本>/` 目录下，当执行 `modprobe <内核模块名>` 命令进行加载内核模块时候自动搜寻此目录，进行模块加载。

`fbtft_device` 模块完整的加载命令是：

```
sudo modprobe fbtft_device name=nokia3310
```

`fbtft_device` 模块包含一个必要参数 `name`，显示屏的名称，这里使用 `nokia3310`，`3310` 屏幕和 `5110` 屏幕使用的是同一个 PCD8544 芯片，因此 `nokia3310` 和 `nokia5110` 可以公用一个驱动。

加载驱动后屏幕背光亮起，使用 `lsmod` 命令可以看到 `fbtft_device` 模块已经加载。查看 `/dev/fb*` 设备多了 `fb1`，这个 `fb1` 就是 Frambuffer 设备，这个设备可以存储 `48*84bit` 的数据，也就是屏幕分辨率的大小，这也符合 Linux 一切皆为文件的特点，这样在 `/dev/fb1` 写某个 bit 上写数据就可以了写到屏幕上对应的某个 bit。

5.4.3 con2fbmap 控制台映射显示技术实现

在没有字模的情况下，虽然有了 Frambuffer 驱动，依然无法将字符描述成点阵方式写入 Frambuffer 设备，不过可以通过 `con2fbmap` 命令将 console 映射到 Frambuffer 设备中，也就是相当于屏幕能显示终端命令行内容，标准输出将映射在 Frambuffer 上。命令 `con2fbmap 1 1`，第一个参数 1 是终端设备的次设备号 `tty1`，第二个参数 1 是 Frambuffer 的次设备号 `fb1`，执行后可以看到屏幕输出的终端内容，之后程序输出的内容在屏幕上都看得见。

为了程序的通用性，不能输入命令来设置设备，如果能写入代码来映射，开机后启动程序即可立即映射，无需多输入一条命令。

通过查看内核代码，全局搜索 `con2fbmap`，可以获得 `fbmem.c` 和 `fb.h` 中包含 `con2fbmap` 字段，`fbmem.c` 中，实现了一个 `fbmem` 的内核模块，实现了几乎应用层的所有关于 `Frambuffer` 设备的系统调用，如 `fb_open/close/read/write/ioctl` 等，总体实现的是一个 `fb` 字符设备的模块，当实现这些内容，在应用程序使用 `open/read/write` 等系统调用对 `/dev/fb` 设备进行操作时候即可调用本模块相应的函数，这就是在 `Linux` 用 `C` 语言进行面向对象分层设计中的多态思想。

其中 `fb_ioctl` 函数是传统 `ioctl` 的系统调用，实际实现为 `fb_ioctl` 函数，在这个函数中，通过 `C` 语言的 `switch` 语句分流 `ioctl` 函数中的 `cmd` 参数，针对此设备去执行不同的 `cmd` 任务。

图 5.10 为 `Linux` 内核中 `Frambuffer` 驱动的部分代码，`con2fbmap` 命令的实现依赖于此代码，并且在计步程序中设置屏幕也需要调用到此驱动中的函数，这是 `fb_ioctl` 函数部分代码截图，是 `ioctl` 系统调用的具体实现，由图得，有两个和 `con2fbmap` 有关的 `cmd`，`F BIOGET_CON2FBMAP`，`F BIOPUT_CON2FBMAP`，从 `cmd` 名称可以看出，一个是获得当前 `con2fbmap` 的设置，一个是设置当前 `con2fbmap`，下面的代码流程也印证了这一点，那么应该使用 `ioctl` 的 `F BIOPUT_CON2FBMAP` 命令来设置 `con2fbmap`。

```

1156         break;
1157     case FBIOGET_CON2FBMAP:
1158         if (copy_from_user(&con2fb, argp, sizeof(con2fb)))
1159             return -EFAULT;
1160         if (con2fb.console < 1 || con2fb.console > MAX_NR_CONSOLES)
1161             return -EINVAL;
1162         con2fb.framebuffer = -1;
1163         event.data = &con2fb;
1164         if (!lock_fb_info(info))
1165             return -ENODEV;
1166         event.info = info;
1167         fb_notifier_call_chain(FB_EVENT_GET_CONSOLE_MAP, &event);
1168         unlock_fb_info(info);
1169         ret = copy_to_user(argp, &con2fb, sizeof(con2fb)) ? -EFAULT : 0;
1170         break;
1171     case FBIOPUT_CON2FBMAP:
1172         if (copy_from_user(&con2fb, argp, sizeof(con2fb)))
1173             return -EFAULT;
1174         if (con2fb.console < 1 || con2fb.console > MAX_NR_CONSOLES)
1175             return -EINVAL;
1176         if (con2fb.framebuffer >= FB_MAX)
1177             return -EINVAL;
1178         if (!registered_fb[con2fb.framebuffer])
1179             request_module("fb%d", con2fb.framebuffer);
1180         if (!registered_fb[con2fb.framebuffer]) {
1181             ret = -EINVAL;
1182             break;
1183         }
1184         event.data = &con2fb;
1185         console_lock();
1186         if (!lock_fb_info(info)) {
1187             console_unlock();
1188             return -ENODEV;
1189         }
1190         event.info = info;
1191         ret = fb_notifier_call_chain(FB_EVENT_SET_CONSOLE_MAP, &event);
1192         unlock_fb_info(info);
1193         console_unlock();
1194         break;
1195     case FBIOBLANK:
1196         console_lock();

```

图 5.10 内核 fbmem.c 部分代码

由于 ioctl 最后的 arg 参数是不定类型的 void* 参数，那么就需要知道 FBIOPUT_CON2FBMAP 执行中所需要的参数，来确定 ioctl 的参数。

从图 5.10 的 1172 行可以看出，从应用层获取了一个 argp 参数，也就是应用层传入内核层的一个指针，将这个指针的内容赋值给 con2fb，那么应用层需要传入的参数应该是一个 con2fb 的结构体指针，通过查看定义得知，con2fb 结构体包含了一个 unsigned int 类型的 console 和 Frambuffer，那么这个 con2fb 结构体就是通过设备的次设备号指定了执行控制台映射到帧缓冲设备两个最重要的设备，Console 和 Frambuffer 设备，接下来 ioctl 的参数就非常明确了，应用层在 ioctl 中，只需给定要设置的 fb 设

备文件描述符和 `con2fb` 结构体指针，即可对指定的 `fb` 设备映射指定的 `Console` 内容。

回到计步系统的 `display` 模块，在这个模块中，显示设备进行初始化完成的任务就是完成将 `Console` 映射到 `Frambuffer`。经过上述代 `fb` 驱动代码的分析，应用层实现起来很简单，通过系统调用 `stat` 获得设备文件 `/dev/tty1` 和 `/dev/fb1` 的次设备号，组装成 `con2fb` 结构体，通过 `fb` 设备的系统调用 `ioctl`，传入此 `con2fb` 指针即可实现映射，即内核中会执行图 5.10 中 1171-1194 行的内容。

现在，在应用程序中输出到标准控制台的任何内容在屏幕中都可看到，如此即可显示传感器的加速度值和计步结果。

5.5 守护进程模块

守护进程是在一个计算机系统后台执行的计算机进程，并且不受任何终端的控制。守护进程的父进程是 `init` 进程，也就是其 `PPID=1`。一般情况下，对一个子进程执行 `fork()` 系统调用，然后使其父进程终止并退出，使得这个进程在 `init` 进程下

创建守护进程的完整步骤是：

1. 创建子进程，终止父进程。
2. 脱离控制终端，在子进程中创建新会话组。
3. 禁止进程打开终端的权限。
4. 关闭父进程的文件描述符。
5. 改变当前进程的工作目录。
6. 重置文件创建掩码。
7. 忽略 `SIGCHLD` 信号。

物联网主机在启动计步程序时第一步先判断 `root` 权限，再初始化为守护程序，这是程序初始化的第一步，初始化守护进程写在 `daemon` 模块的 `daemon.c` 文件中。

5.6 Makefile 编译规则描述

`Makefile` 在工程编码中主要完成编译链接的整个过程，之前章节介绍了 `make` 和 `Makefile` 的作用。物联网主机端的代码相对一般项目较小，但也有多个源码文件多个模块，使用 `Makefile` 能够更快速的构建。

首先要设定好 `Makefile` 的最终目标，还要设定好生成最终目标的所有依赖，最终目标显然就是计步程序，命名为 `pedometer`，所有依赖就是所有的源码文件，通过 `Makefile` 的内置函数 `wildcard` 获取所有的 `*.c` 文件列表，并通过 `patsubst` 函数将任意一个 `%.c` 文件命令为同名的 `%.o` 文件，这样就拥有了所有的 `*.o` 文件列表。

如此，所有的依赖关系列表都有了，这个编译链接顺序是：`%.c-->%.o-->pedometer`，即所有的 `.c` 文件编译为 `.o` 文件，最后将所有的 `.o` 链接为 `elf` 可执行文件。

`Pedometer` 程序依赖所链接的所有由 `patsubst` 生成的 `.o` 文件列表，`.o` 文件列表又依赖对应的 `.c` 列表，也就是所有的源码文件，这样整个编译结构就很清晰了。

除了确定依赖关系外，还需要设定一些编译参数，编译工具使用 `gcc`，编译标志 `-g` 在可执行文件中保留调试信息，`-Wall` 显示出所有警告。

`clean` 指令下需要写下要清理整个工程的过程文件和最终目标文件，执行 `make` 将依据依赖树进行编译生成和链接，执行 `make clean` 即可清除所有的过程文件和目标，仅保留源码文件以待重新编译。

6 数据处理和计步算法

6.1 算法流水线初始化

计步算法主要分为原始数据的滤波处理，峰值的检测和更新，动态阈值的检测和更新，最后根据时间条件、空间条件均满足步伐要求，即可记为一步。最后更新计步的时刻，以便于下一轮对时间条件的检查。

计步算法的各个步骤需要对相应的全局参数进行初始化，以便于以后滤波器和计步算法的使用。在正式的数据轮询之前，先对各个算法的容器进行初始化，也就是对上述均值滤波器、峰值检测存储器、步伐时间间隔存储器、步伐幅度（峰值）存储器、步伐精度阈值存储器依次进行初始化。

对于均值滤波器，采用循环队列的方式存储样本，样本容器容量大小为 4，初始化需要将容器置零，容器样本计数置零。

对于峰值检测存储器，分别存储了最大的三个轴的最大峰值和最小峰值两个样本，由于需要持续性更新峰值，因此至少需要两个存储空间，分别来保存之前的最大最小峰值样本和新的最大最小峰值样本，需要将最大值初始化为对应类型的最小值，最小值初始化为对应类型的最大值，方便后期的更新。

对于动态精度阈值的存储器，需要存储新旧两个样本，并通过给定样本差的预定义差值，对比后决定是否保存并更新精度阈值，在初始化时候将其清空。

在步伐检测的决定性因素之一时间条件上，需要保存上一步的时刻和当前轮的时刻，通过差值来对比预定义的人的正常步频时间，在初始化期间均赋值为当前时间。

6.2 样本均值过滤处理

通过上个章节网络部分的介绍，数据从网络上来，在主机端一直轮询读取数据，这些都需要在一个主循环中进行，每一轮循环都会接收一个数据包，也就是一个样本。拿到的数据将作为本轮的新数据进入算法流水线。

如图 6.1 均值滤波器是第一个算法节点，填充 4 个样本之后，每插入一个样本都要计算一次均值，也就是每次计算只有一个样本和前一轮不同，新进入的样本覆盖掉最早的样本，并进行求和计算均值，4 个样本的平均值作为一个样本并存储在参数的指针中返回。

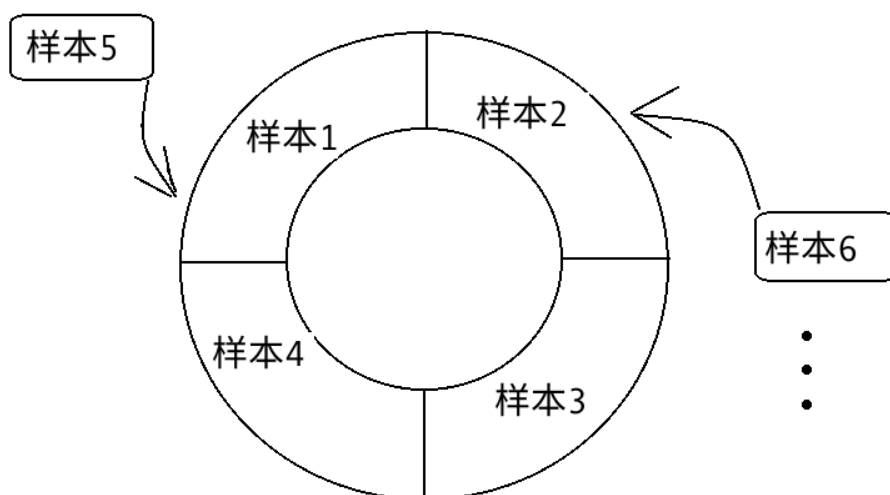


图 6.1 均值滤波器循环队列

均值滤波器的索引通过循环计数变量自增对容器大小的取余可以得到当前样本存入循环队列的索引，覆盖的这个样本就是最旧的样本。

滤波函数在填充新样本后输入滤波器容器指针和滤波结果的样本指针，通过公式 6.1 计算均值后结果保存到刚刚传入的指针。

$$\text{filter_result} = \frac{s1+s2+s3+s4}{4} \quad (6.1)$$

其中，s1-s4 为容器中的 4 个样本，filter_result 为均值滤波的结果。传出参数将用于之后的算法节点，并将其又作为基本样本，相当于处理过的原始数据来使用。

6.3 峰值的检测和更新

记录加速度值的峰值是用于之后检测步伐的空间条件的重要依据，峰值的检测更新较为简单，顾名思义，当前样本大于最大值或者小于最小值，则更新峰值。

不过为了避免极端情况，比如人跳跃后，峰值极高，之后步伐中再也无法达到这个峰值，因此峰值也设定了周期性重置，以 50 个样本一个周期，通过采样频率来计算也就是 2 秒一个周期。通过静态变量，将函数的调用次数记录下来，超过设定周期就重置峰值。正常情况下，峰值的新值更新为新值和当前值较大或者最小值中较小的一个。

峰值更新算法流程如下图，对样本进行计数统计+1，样本计数器为静态变量，不会因为算法的重新调用而被重置，样本以 50 为周期重置，最后判断最大最小峰值是否需要更新。

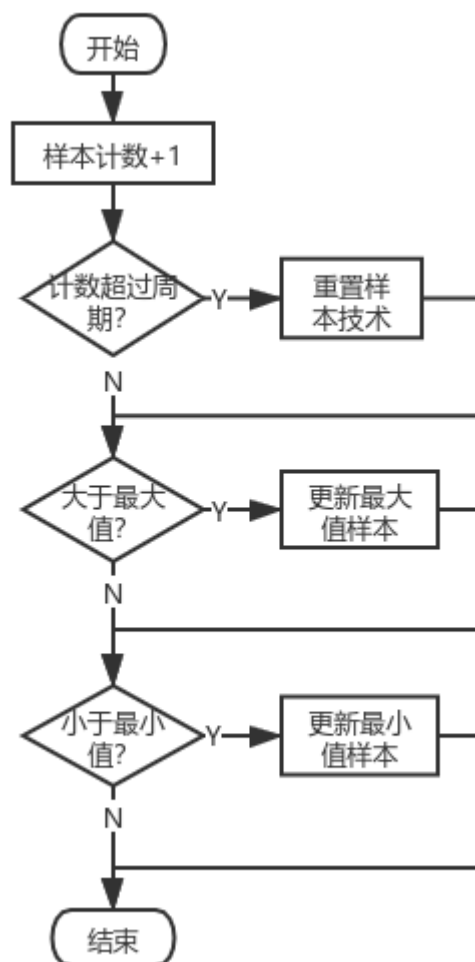


图 6.2 峰值更新流程

6.4 动态阈值检测和更新

之前已经介绍过，动态阈值有新旧两个样本组成，在动态阈值的更新过程中，一共涉及到三个样本，其中两个是动态阈值容器中存储的较旧的阈值样本和较新的阈值样本，还有一个是更新函数传入当前样本，第一步将容器中旧样本无条件的覆盖为容器中较新的样本，而函数新传入的样本是否覆盖较新的样本，需要通过对比函数当前传入的样本的每个轴和容器内新样本每个轴的差值来决定，如果差值超过预定义的敏感度，则更新阈值容器的新样本。

步伐检测的敏感度就是通过上述预定义的值设定，如果在测试过程中，计步检测稍偏敏感，可通过调高此值，反之亦然。

动态阈值的结果将通过指针传回，同时返回是否已经更新动态阈值，阈值容器中的样本将是步伐检测中空间条件的又一个重要依据。

6.5 计步的时间条件和时间更新

上一节谈到的动态阈值更新与否将通过返回值带回，若已更新，则检测当前轮的更新是否同时满足步伐的时间条件和空间条件。时间条件就是人行走的步伐频率在一定范围内，超出这个范围将不计为步行运动，通常称为时域分析。

时间存储器保存了最晚一次计入步伐的时间和在时间进行更新时候记录的时间，这里分别记作 `last`（最后一步的时刻）和 `this`（更新函数调用的时刻），注意，这里仅仅在确认记为一步的时候进行时间更新，这为了保证每次的 `last` 和 `this` 时刻是相对当前时刻来讲最后一步计步时间，另外，时间条件的检测时机也是在确认更新了动态阈值后进行，没有更新阈值则不检查时间条件和空间条件满足与否。

如图 6.3，步伐检测的第一步就是判断阈值是否更新，如果没有更新则直接结束本轮检测，因为和原来阈值一样，空间条件一定不满足，不需要进行判别步伐所满足的两个条件，如果更新了阈值则判定这两个条件是否满足，同时满足则计步并更新 `last` 和 `this` 时刻。

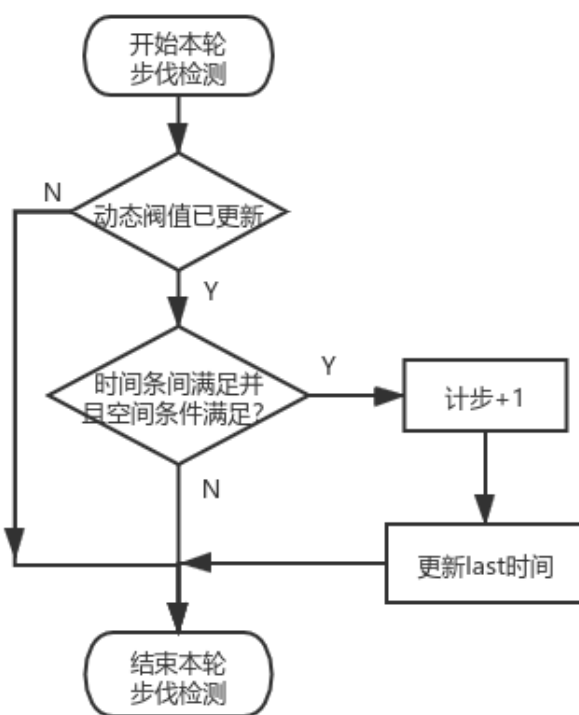


图 6.3 步伐检测流程

上述计步要求满足的空间条件将在下一节详细论述。时间条件上完成的方式是通过对 `last` 时刻和 `this` 时刻做差，求得时间差后与人们通常的步伐频率对应的时间间隔进行比较，两步在合理的时间间隔内，一般认为人类步伐间隔在 200ms-2000ms 之间，这与之前章节已经论述过的选取的采样率有一定关联。

时间的记录方法是 Linux 中已经定义的 `timeval` 结构体，`timeval` 结构包含了秒和

毫秒，也就是最多精确到 1ms，计算时间差需要将秒转为毫秒后做差即可。获取当前时刻，对于精度要求不高的精确到 1ms 时间，使用 Linux API `gettimeofday()`。

时间差和满足的时间范围可表示为下式（6.1-6.4）：

$$\text{last_ms} = \text{last.s} * 1000 + \text{last.ms} \quad (6.1)$$

$$\text{this_ms} = \text{this.s} * 1000 + \text{this.ms} \quad (6.2)$$

$$\text{interval} = \text{this_ms} - \text{last_ms} \quad (6.3)$$

$$200\text{ms} < \text{interval} < 2000\text{ms} \quad (6.4)$$

上式中，`last_ms` 和 `this_ms` 分别是两个时刻换算为 ms 后的值，`interval` 是做差后的结果，要求 `interval` 处于 200ms 到 2000ms 之间。

步判定的时间条件函数中，如果 `interval` 满足范围，则返回 `step` 为 1，否则重新初始化储存两个时刻的存储器。

6.6 计步的空间条件和运动轴检测

步伐检测的另一个必要条件是空间条件，也就是加速度变化的幅度大小，更准确点说就是本次加速度的改变幅度是否能够满足一定的运动量从而判定为一步，空间条件使用的是频域分析法分析运动的幅度大小。

人步行是处于一个三维空间运动的物体，因此在三个轴上都有加速度，由于运动方向固定，因而无论加速度传感器如何放置在身上，总有一个运动轴变化强烈，这是由于人步行时候腿的上下运动而产生的大幅度震动，引发加速度剧烈变化。计步检测的空间条件就指腿在步行时候的抖动。

根据上述的三轴仅仅获取一个最活跃的轴的运动幅度即可，那么需要先找到这个运动轴。之前记录的峰值在这里就可以派上用场，通过分别对峰值样本的每个轴的最大和最小样本做差，求得三个轴的最大变化量，再求三轴最大变化量中最大的轴，并将其返回。

对每个轴来说，首先计算峰值的中间值，再对比阈值的新样本是否处于极值的中位之上，并且阈值的旧样本处于极值的中位之下，如果是，则在计步的空间条件上满足了，返回一个步伐。这个过程可以通过公式(6.5-6.6)表示：

$$\text{middle} = \frac{(\text{peak_max} + \text{peak_min})}{2} \quad (6.5)$$

$$\text{slid_old} > \text{middle} > \text{slid_new} \quad (6.6)$$

上式中，`middle` 表示峰值极点的中间值，`peak_max` 和 `peak_min` 分别表示两个极值点，`slid_old` 和 `slid_new` 分别表示新旧两个阈值，只要满足式 6.6，就判定此次更新记一步。

步伐检测中的空间条件判定的流程图如图 6.4，在空间条件检测中首先要做的就是寻找最活跃的轴，即可拿到活跃轴，之后利用前面获得的动态阈值和峰值来检测对于这个轴是否满足一个步伐的运动量。

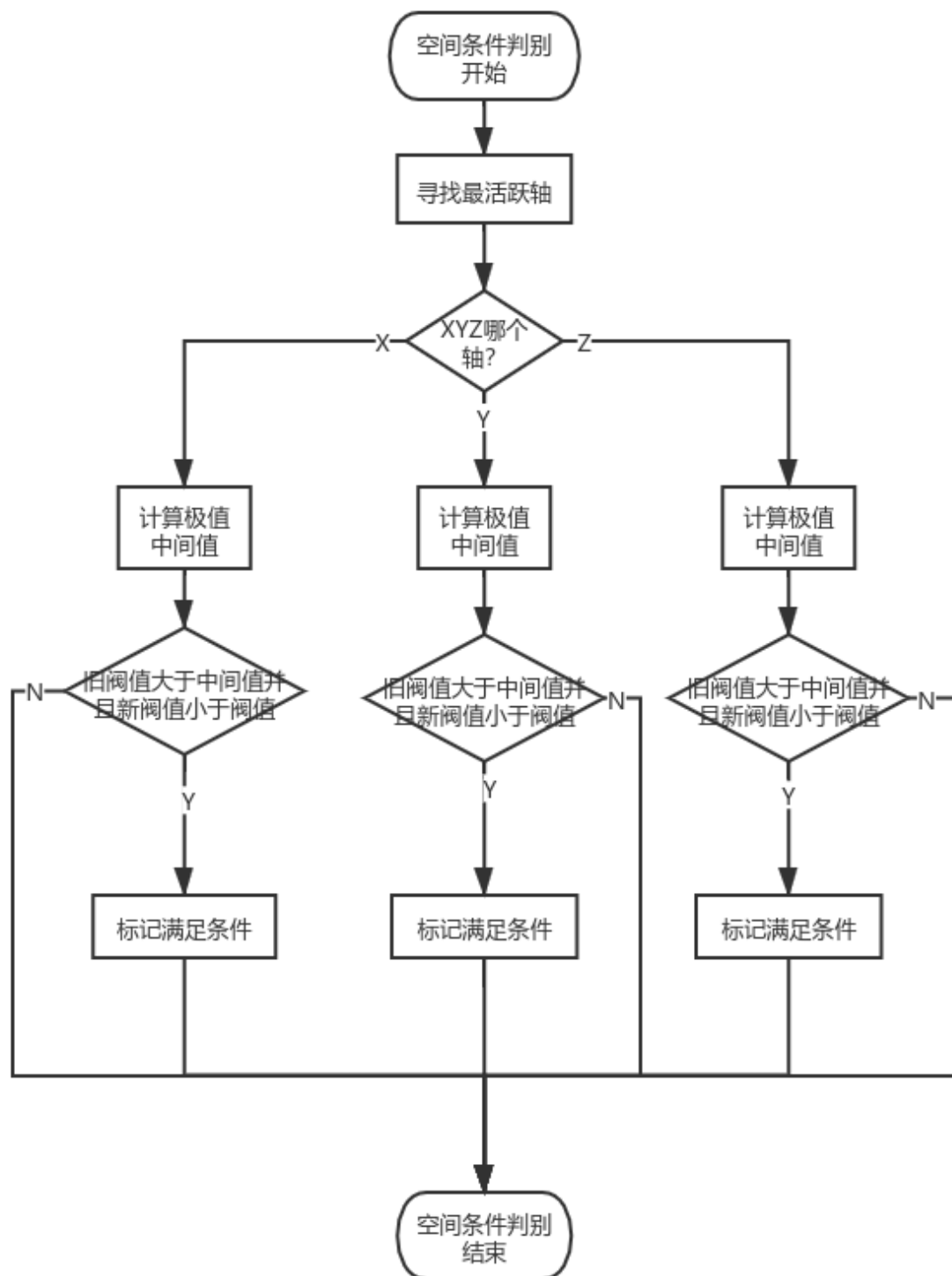


图 6.4 步伐检测空间条件流程

上述的空间条件和上节的时间条件共同决定了是否记为一个步伐，但前提是已经更新了动态阈值的情况下。确定记一步后计步变量自增 1，之后更新计步时间，完整的一轮计步结束。

7 远程监控主机程序设计与实现

7.1 远程监控主机程序介绍

远程监控主机常用于对整套设备的监控，可以通过 SSH 或者远程桌面连接到树莓派控制和操作树莓派，也可以将计步数据采集节点的数据发送到监控主机，在监控主机上做数据分析。

远程监控主机程序是一个使用 Python3 写的一个简单的原始数据接收和绘图监视程序，主要使用 Python 的数据分析和可视化 matplotlib 库。也用到其他库，如对列表操作的 numpy 模块，网络通信上使用的 socket 模块，进行数据解包的 struct 模块。

监控程序不是作为计步监控系统的核心关键程序，但是依然有很大的作用：其中一个作用是在开发过程中，开发人员可以根据绘制的折线图来直观的看到数据的变化，从而找到数据的规律，以便于编写计步算法。另外，在这个系统中，智能通过一个小的 LCD 显示屏显示简单信息，显示不了所有的（物联网主机作为家庭物联网运算核心设备以后可能接入更多的物联网节点）节点信息，因此第二个作用就是通过远程监控主机来监视不限于计步节点的所有物联网节点。需要清楚的是，本计步监控系统不依赖远程监控主机以至于整个家庭物联网系统的运行也不依赖于远程监控主机。

7.2 网络数据的接收和绘图

这个程序要完成的是数据监控和可视化绘制，那么首先应该获得数据，从网络上获得数据，计步数据采集节点上是一个 UDP 客户端，并且一直在采集传感器数据并通过 WiFi 上传到指定的 UDP 服务端，那么本程序首先要创建一个 UDP 服务器，使用 socket 模块的 socket 函数，传入 IPv4 协议族 AF_INET，套接字类型使用数据包套接字。由于计步数据采集节点是向指定端口发送数据，但是服务端不知道数据的来源 IP，因此需要绑定端口，不指定 IP，也就是所有的 IP 都可通过。

初始化一个画布，初始化范围为 0-200，精度为 1 的时间横轴，出事范围为 -2g-2g，精度为 1 的加速度纵轴，初始化三个画线器分别为红、绿、蓝色，将使用它们绘制折线图，同时初始化三个列表用来保存当前在图上的点的所有加速度值，每个轴 200 个点。

matplotlib 模块包含一个 animation 包，其作为一个动态画图的核心包，使用起来较为简单，在本程序中只有两部分，曲线的调用和更新。animation 的核心函数是 FuncAnimation，函数需要指定画布，将曲线画在哪个画布上，还要指定更新的时间间隔，刷新率缓冲区等渲染相关的参数，最重要的一个参数还是动态的更新函数，因为曲线是动态变化的，就必须使用一个函数来更新每条曲线对应的数据，因此需将这

个函数将函数名称作为参数传入 `FuncAnimation`，曲线的每一步如何变化是由你自己决定的，这个更新函数也需要完成定义。

`animate` 函数完成了整个从数据的接受、解包、以及完成对三条曲线纵坐标值的更新等，最后函数返回三条曲线对象，`FuncAnimation` 将按照返回的曲线对象绘图。

网络接收数据包使用最开始初始化的 `socket` 对象的 `recvfrom` 方法进行，该函数只需要传入接受信息的大小，返回二进制数据和源地址信息。

二进制数据使用 `struct` 模块的 `unpack()` 函数解包。根据之前计步数据采集节点定义的数据发送格式，数据分为 7 个 2Bytes 大小的 `short` 类型，`unpack()` 函数需要拿到该数据格式，对比图 7.1 的格式对照表，其中“h”表示 `short` 类型，用格式化字符串“hhhhhhh”可以解包这个一个二进制数据包，并通过列表的方式返回。

Format	C Type	Python type	Standard size	Notes
x	pad byte	no value		
c	char	string of length 1	1	
b	signed char	integer	1	(3)
B	unsigned char	integer	1	(3)
?	_Bool	bool	1	(1)
h	short	integer	2	(3)
H	unsigned short	integer	2	(3)
i	int	integer	4	(3)
I	unsigned int	integer	4	(3)
l	long	integer	4	(3)
L	unsigned long	integer	4	(3)
q	long long	integer	8	(2), (3)
Q	unsigned long long	integer	8	(2), (3)
f	float	float	4	(4)
d	double	float	8	(4)
s	char[]	string		
p	char[]	string		
P	void *	integer		(5), (3)

图 7.1 C 类型和 Python 类型解包格式对照

执行 `Python3 display.py` 可以运行监视程序，如图 7.2 所示，每行为一个数据包，解包后的每行 7 组数据输出到连接的到树莓派的 SSH 虚拟终端。

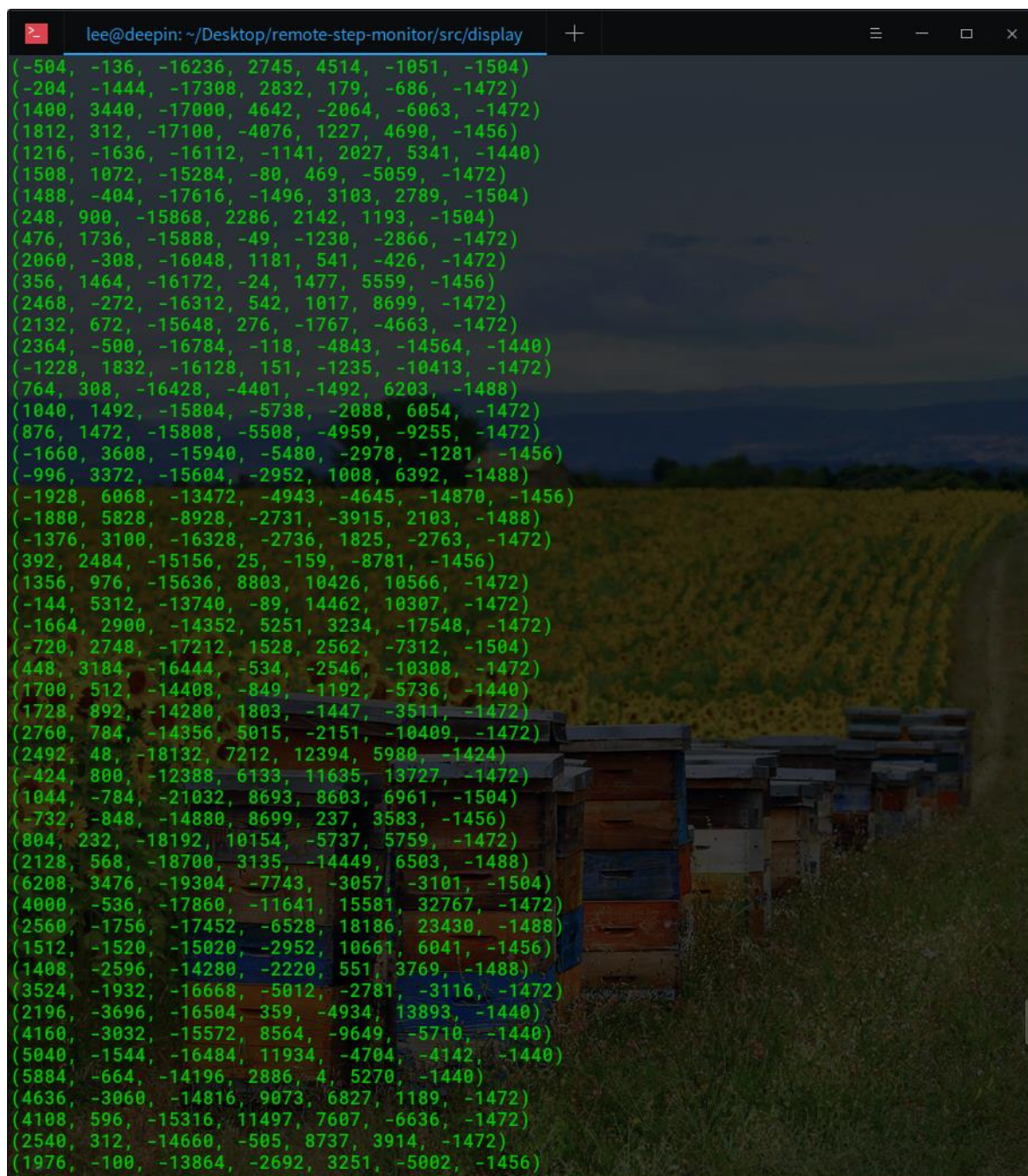


图 7.2 监控程序输出解包后的原始数据

通过返回的数据能够拿到每个轴的加速度数据，因此可以更细每个轴对应列表存储的纵轴值，动态绘图中，每个轴的数据就像一个管道一样进出数据，显然这里数据的更新应该使用队列的方式先进先出，使用 `pop` 能够弹出最早进入列表的数据点，使用 `insert` 并指定插入位置为 0 则能够在列表的入口出插入纵轴数据，这样即可实现数据更新，实现数据动态变化，再通过曲线的 `set_ydata` 方法可以将列表数据更新到曲线上，最终将三条曲线返回即可。

图 7.2 的数据看不出任何特点，将其用 Python 的可视化绘图模块绘制出来的图像如图 7.3 所示，X、Y、Z 轴加速度数据分别为红、绿、蓝三色，纵坐标轴是加速度的

具体值，单位是重力 g ，横轴是时间。

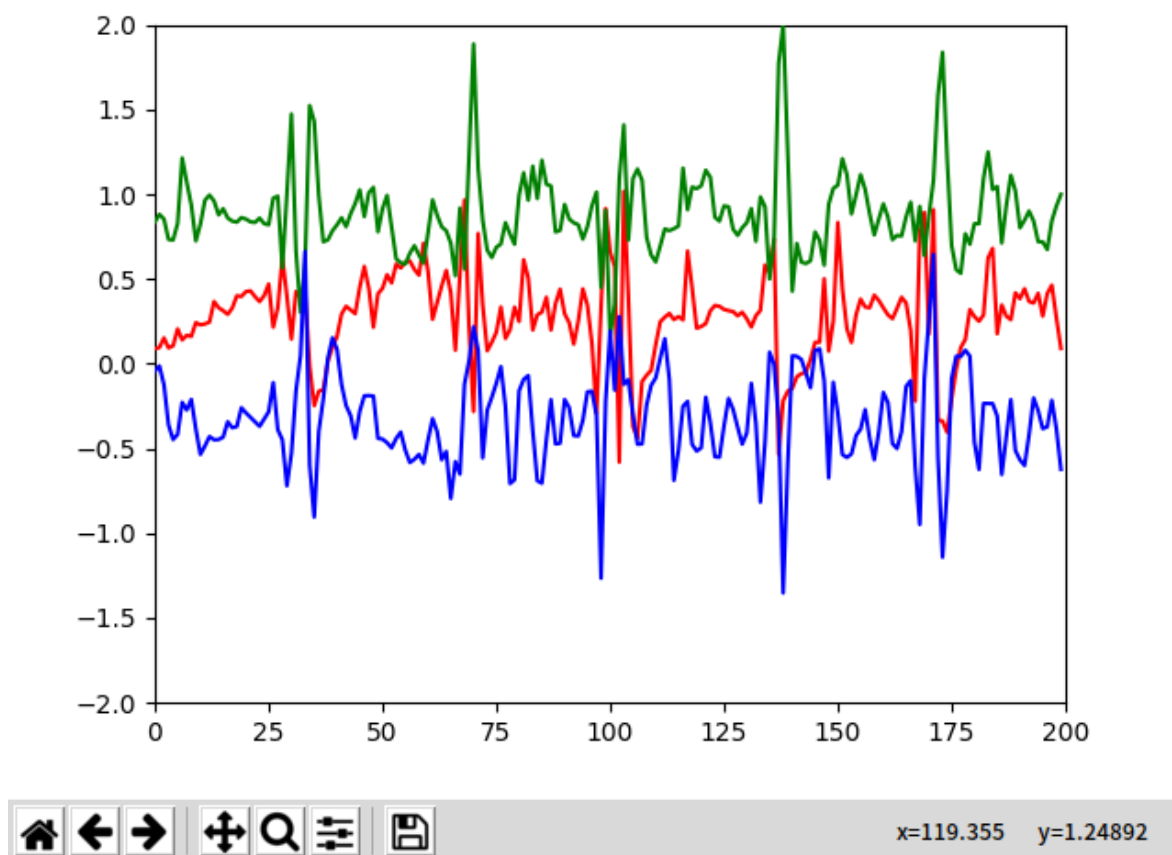


图 7.3 加速度值的可视化绘图结果

7.3 运动时三轴数据变化分析

正常步行状态的加速度数据绘制出来如图 7.4 所示，横轴的 50 之前加速度变化是起身变化，50 之后加速度变化在步伐的影响下显示出相应的特点，红色和绿色在 0 附近波动，蓝色曲线在 1 附近波动，1 是指一个重力加速度，这说明运动时候传感器的摆放位置是 Z 轴正方向向上，0 附近的红色和绿色没有受到重力的影响仅仅在运动时候产生的波动。而且也很明显看出，步行过程中，最活跃（波动最大）的轴，一定是和重力在一个方向上的，也就是步行中运动幅度最大的是垂直于地面的上下运动，那么通过最活跃轴的监视是最有利于统计步伐的。

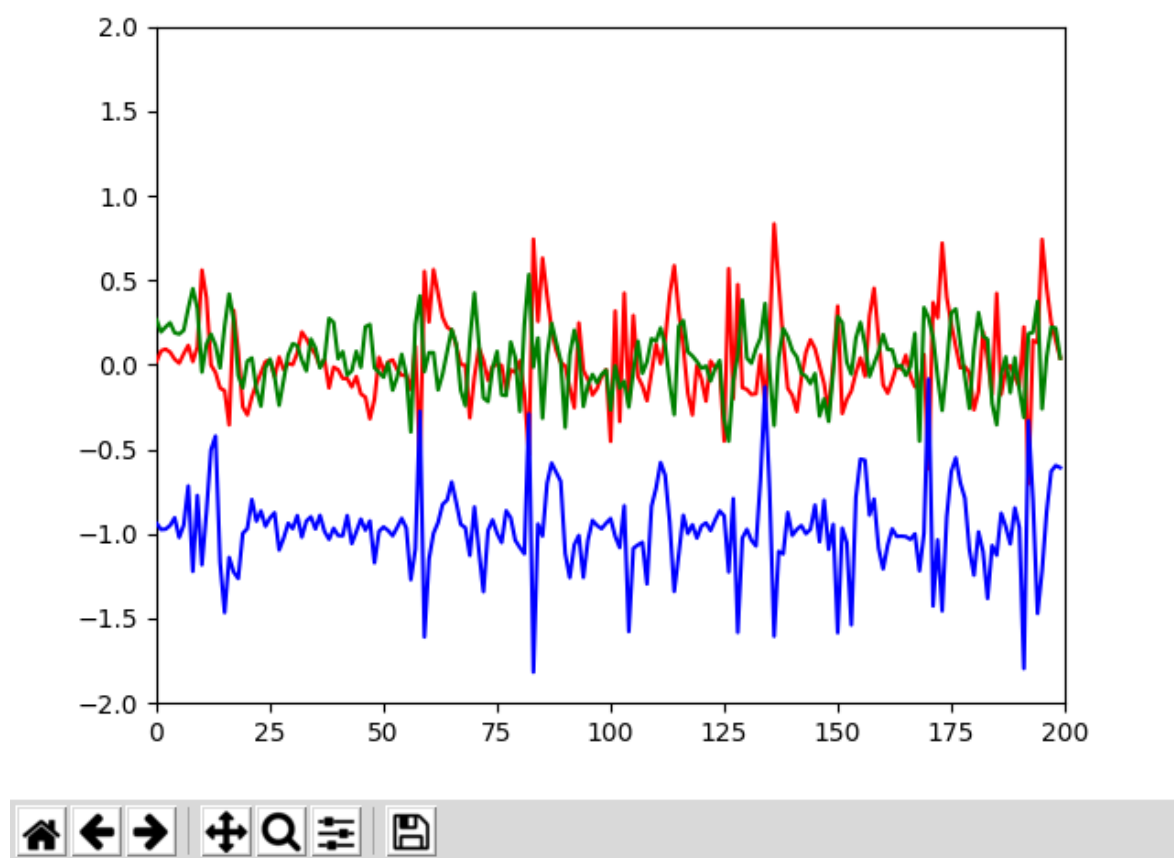


图 7.4 步行中加速度的变化

8 功能性能的调试测试

8.1 单独计步数据采集终端调试

计步数据采集终端使用 ESP8266 和 MPU6050 两个模块，通过 I2C 连接，根据之前章节对各个模块引脚的论述，连接方式应如图 8.1 所示。

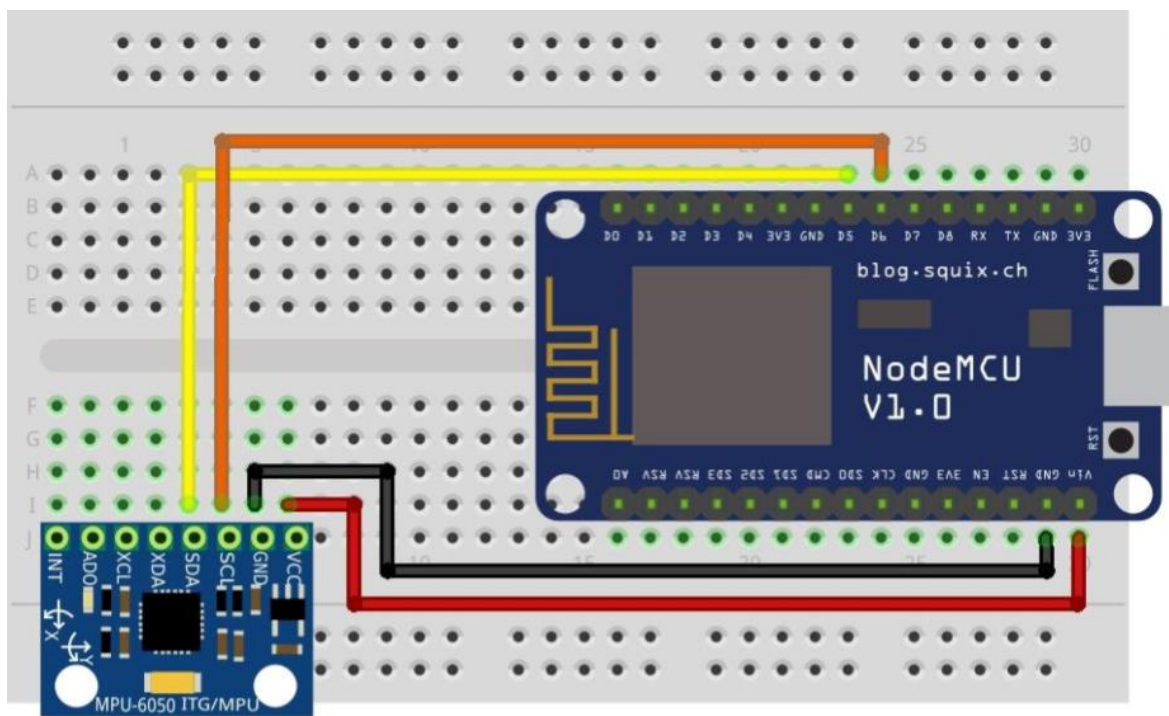


图 8.1 ESP8266 与 MPU6050 连接

之前论述过，调试信息需要通过串口打印到电脑屏幕上，需要用到串口转 USB 工具，ESP8266 的 NodeMCU 评估板具有串口转 USB 的功能，板载 Micro-B 接口，因此连接电脑的方式很简单，如下图 8.2 直接插入 USB 线即可。

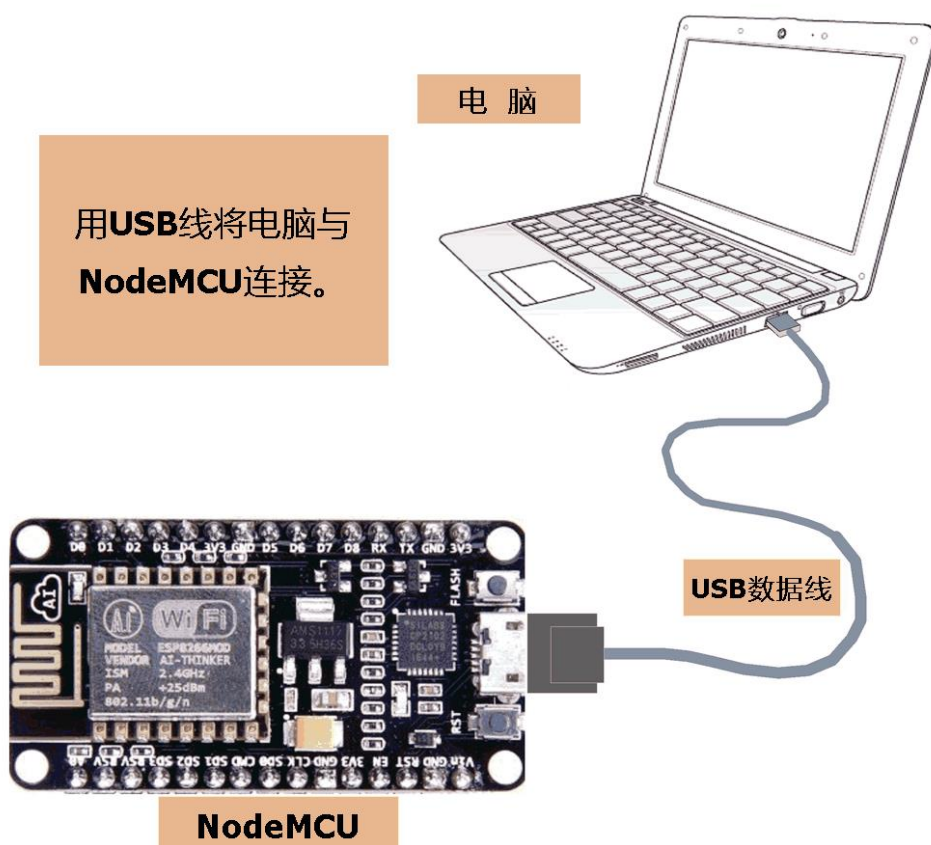


图 8.2 NodeMCU（ESP8266）与电脑连接

编写设备初始化，采集和输出打印的代码后即可进行编译调试。

在 VS Code 的 PlatformIO 插件中需要在 platformio.ini 配置文件中输入点击 monitor 即可连接 ESP8266 并输出程序打印的串口数据。如图 8.3 所示，上方为 ESP8266 的评估板 NodeMCU 的相关配置，其中 monitor_port 是连接串口的设备，monitor_speed 串口的波特率，需要和程序中的初始化一致，否则打印会乱码。下面部分是串口输出的传感器原始数据，同样是 7 组数据。

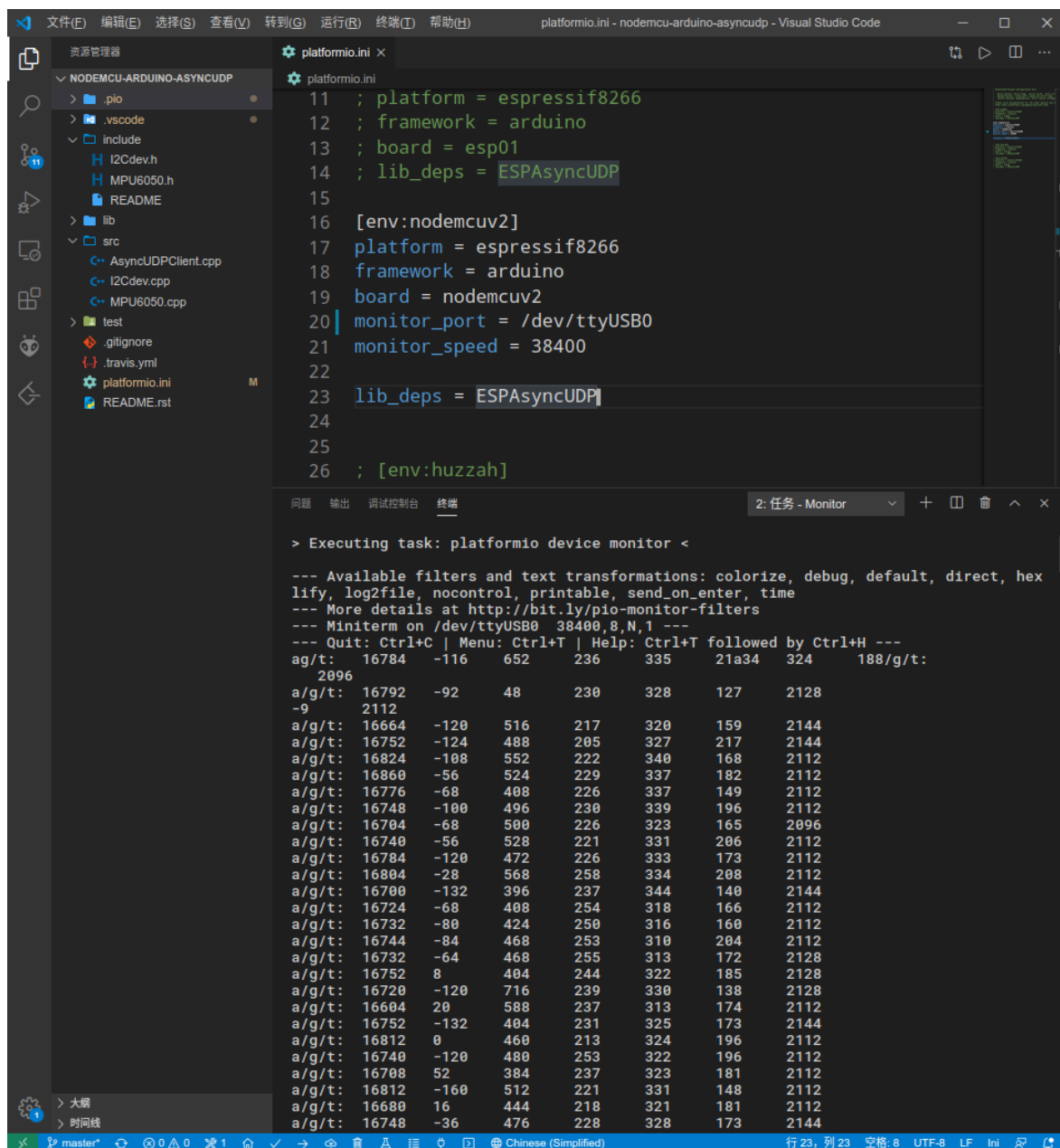


图 8.3 ESP8266 在 VS Code 中的配置和串口输出

计步数据采集节点实物图如图 8.4 所示。

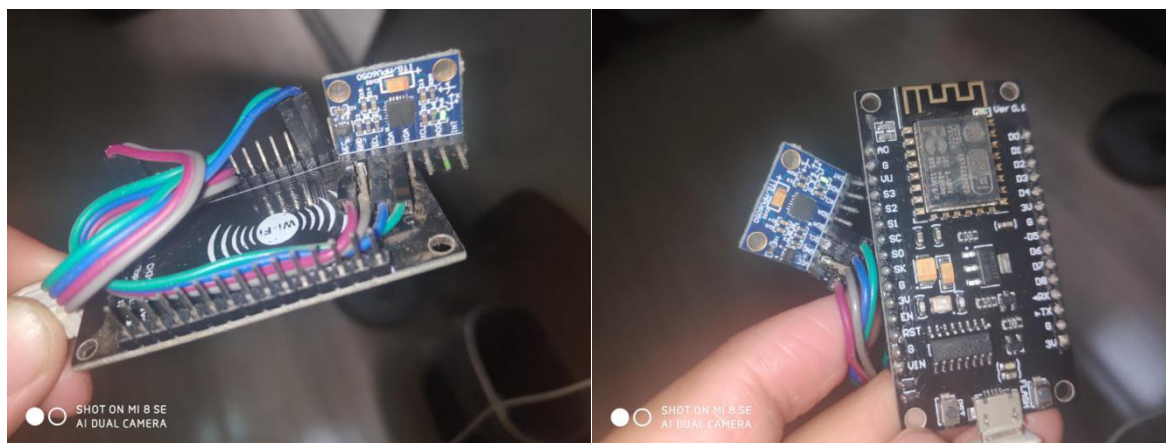


图 8.4 计步数据采集节点实物图

在程序中需要以波特率为 38400 打开串口，使用 `HardwareSerial` 类的对象 `Serial.begin(38400)` 即可对串口进行初始化，使用 `Serial.print()` 即可输出要打印的内容。以上内容说明传感器和 ESP8266 数据采集和串口调试输出工作正常。

8.2 运算主机端数据接收联调、显示设备测试

现在将测试除远程监控主机外，整个计步监控系统的功能，包括从计步数据采集节点的采集和发送数据，到物联网主机进行数据接收和处理并通过计步算法计步，最后显示到 LCD 屏幕上。

如之前章节介绍的 LCD 与树莓派的连接方式如图 8.5

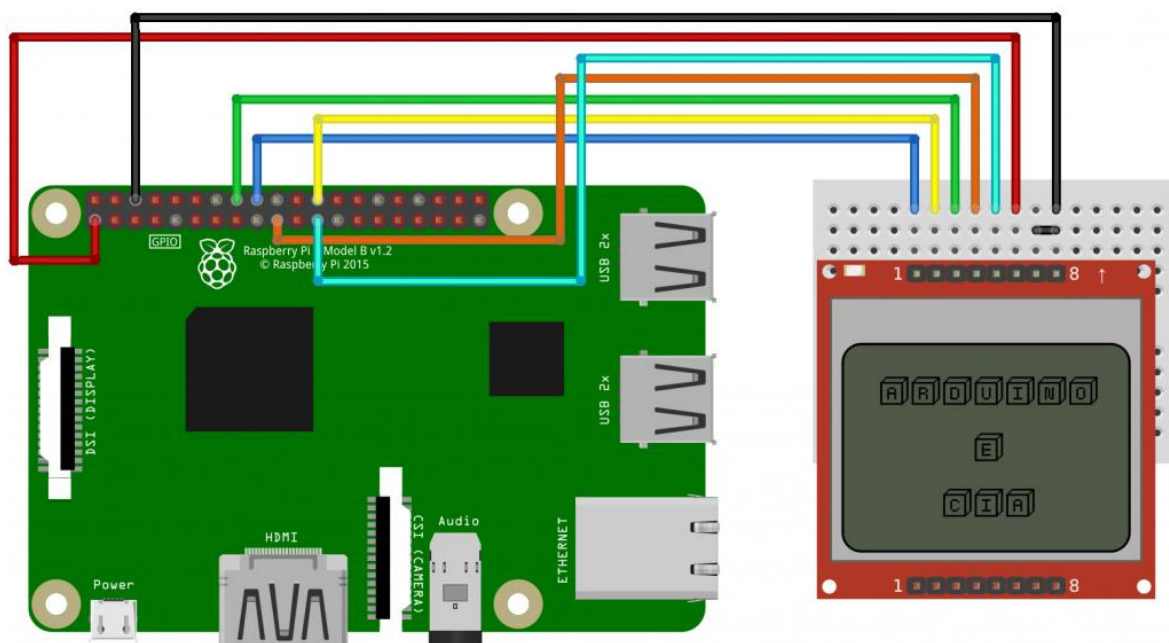


图 8.5 树莓派与 LCD 连接方式

整体的连接实物图为 8.6 所示：



图 8.6 树莓派与 LCD 整体实物图

将物联网主机树莓派的程序在电脑上经过交叉编译后，上传到树莓派，加载屏幕驱动，连接 WiFi 并启动计步程序后，如图 8.7 屏幕显示三轴加速度值和计步的步数 step，前三个数字分别为 X，Y，Z 轴的加速度值，最后一行为步数。

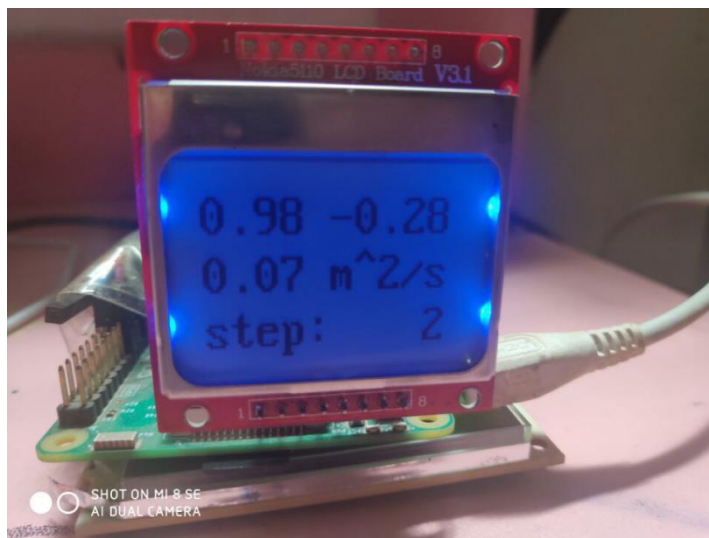


图 8.7 LCD 显示的加速度数据和当前步数

8.3 PC 端数据接收和绘图联调

由 Python 编写的可视化监控程序在电脑上运行，执行 Python3 display.py 可以看到图 8.8 计步数据采集节点的实物和其发送至监控主机 PC 的原始数据和绘制的曲线图像。

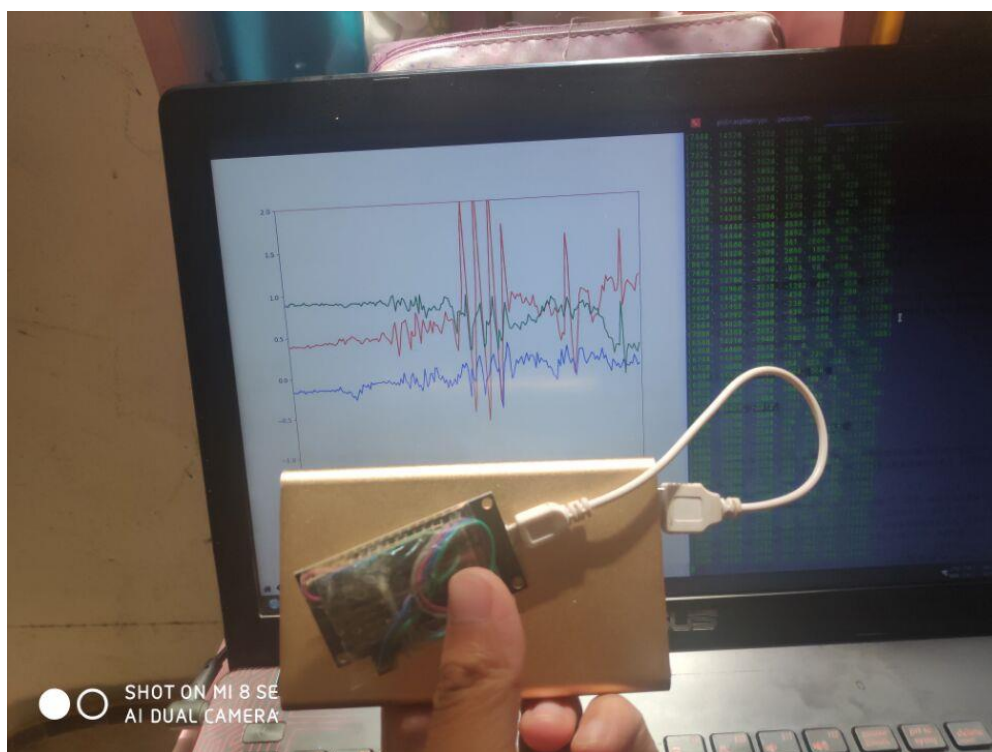


图 8.8 计步数据采集节点和可视化数据

图 8.8 中前方实物为计步数据采集节点和给其供电的充电宝，后方监控主机上执行的 Python 程序是计步数据采集节点发给它的数据，左侧为数据的绘图，右侧为直接打印的直接数据值，两者都处于 40ms 为周期的更新速度滚动变化。

8.4 实际步行测试

将路由器上电，计步数据采集节点上电，树莓派上电，两者通过无线网络连接到路由器，在物联网主机端执行计步程序，进行走步测试。

本次测试同时对比包含计步功能的手环，走路的实际步数通过口头数数计得，因此本次由三组数据：实际步数，手环步数，本计步系统步数。

将手环戴在左手上，计步数据采集节点拿在右手上。经过 5 次测试得出下表 8.1 数据。可以看出，本计步系统的测试结果较为中肯，准确度稍高于市场同类计步手环。

表 8.1 实际走路多设备同步测试数据

编号	测试用时	实际步数	其他手环步数	本计步器步数
1	2:08	120	125	123
2	2:42	260	179	234
3	1:54	240	282	238
4	2:22	280	281	264
5	1:55	190	200	209

9 结 论

本计步监控系统整体上具有较好的表现，这体现在计步算法的优化和采集设备和计算设备分离的优势上。

在计步算法上，采用了更为准确更深入本质的计步的时间条件和计步的空间条件判别方法，在步伐判定时，只有这两个因素能决定是否为一个步伐，从空间条件判定运动量运动幅度是否足够，从时间条件来判定满足了运动量的步伐频率是否足够接近人类正常的步伐频率。

在设备分离上，实现了数据采集设备和计步计算设备分离，将计步所需的运动数据经过无线模块采集并通过无线 WiFi 传输到计步算法的计算设备，前者在文章中称为“计步数据采集节点”，后者在文章中称为“物联网主机”。这样做优势非常明显，首先可以更大的扩展数据采集的物理空间，只采集数据不进行大功率的计算则大大节约了穿戴设备的功耗，正所谓各司其职物尽其用，进行计步算法运算的物联网主机具有更多的硬件资源供使用，可以利用采集到的大量数据进行详细的数据分析和处理。

从最后的测试数据得知，经过对算法的优化，对设备的优化这两方面的努力，本计步监控系统能达到比同类手环稍高的准确度。

不过需要回归初心，做计步器的目的是统计运动量，将运动量化，而不是检查每一步是否统计到位，如果有一步的运动量很大，可以达到平均一步运动的两倍，那么就应该记为两步，反之亦然。因此计步器的发展需要回归初心，更加注重对运动量运动幅度和能量消耗的统计，而不是一共踩了多少脚。

参考文献

- [1] Oner, PulciferStump, Seeling, et al. Towards the run and walk activity classification through step detection - An android application[J]. conf proc ieee eng med biol soc, 2012, 2012(4):1980-1983.
- [2] Jim Scarlett.Enhancing the Performance of Pedometers Using a SingleAccelerometer[EB/OL].
<http://notes-application.abcelectronique.com/013/13-14983.pdf>,2003-08-16/2020-2-15.
- [3] Neil Zhao. Full-Featured Pedometer Design Realized with 3-Axis DigitalAccelerometer[EB/OL].
<https://www.analog.com/media/en/analog-dialogue/volume-44/number-2/articles/pedometer-design-3-axis-digital-acceler.pdf>,2010-06-16/2020-2-15.
- [4] 叶继超.基于 MEMS 的精确计步算法的设计与实现[J].传感器与微系统, (2018)05-0080-04
- [5] 魏芬, 邓海琴. 基于加速度传感器的自适应采样计步器设计[J]. 自动化技术与应用, 2019(5): 121-124.
- [6] 雪人, 乐鑫代理商 ESP8266 Wi-Fi 到底传多远[EB/OL].
http://www.fst-tech.com/newsinfo_513_2_5.html. 2016-07-14 01:00:31
- [7] Noralf Trønes. Linux Framebuffer drivers for small TFT LCD display modules[EB/OL].
<https://github.com/notro/fbtf/wiki/LCD-Modules#nokia-51103310>
- [8] [10]王文杰, 李军. 基于手机加速度传感器的计步算法设计[J]. 工业控制计算机, 2016, v.29(01):80-81+84.
- [9] 蔚利娜. 基于加速度的计步算法和步长计算研究与实现[D]. 东北大学, 2013.
- [10] 王文杰, 李军. 基于手机加速度传感器的计步算法设计[J]. 工业控制计算机, v.29(1):80-81+84.
- [11] 王革超, 梁久祯, 陈璟. 加速度差分有限状态机计步算法[J]. 计算机科学与探索, 2016, 010(008):1133-1142.
- [12] 陈国良, 杨洲. 基于加速度量测幅值零速检测的计步算法研究[J]. 武汉大学学报（信息科学版）2017(6).
- [13] 朱军, 王文举, 陈敬良. 基于蓝牙和计步器的融合定位算法[J]. 包装工程, 2018, 039(005):77-81.
- [14] 徐斌, 裴晓芳, 李太云. 穿戴式智能计步器设计[J]. 电子科技, 2016, 29(3):178-182.
- [15] 陈善武. 基于惯性传感器 MPU6050 的计步器设计[D]. 大连海事大学, 2016.
- [16] 李博戈, 许晓飞. 智能加速度计在电子计步器中的应用[J]. 电子技术(7):55-57.
- [17] 陆兆峰, 秦旻, 陈禾. 压电式加速度传感器在振动测量系统的应用研究[J]. 仪表技术与传感器, 2007, 000(007):3-4,9.
- [18] 王欣蕾, 刘念. 基于 ESP8266 模块的数据采集与上传系统的设计与实现[J]. 科技风, 2018, No.359(27):120.
- [19] Zmjames. MPU6050 和加速度陀螺仪原理[EB/OL].
<https://blog.csdn.net/zmjames2000/article/details/88379640>
- [20] He yuan. MPU6050 工作原理及 STM32 控制 MPU6050[EB/OL].
https://blog.csdn.net/he__yuan/article/details/76559569

致 谢

在本文的撰写过程中，康磊老师作为我的指导老师，非常感激康老师对我毕业设计作品的建议，百忙之中熬夜审批论文并对论文字斟句酌的审阅建议，康老师对工作和教学认真负责的态度，以及对工作的热情，这使我难以忘记的榜样力量。同时也非常感谢康老师将自己个人设备赠送于我，助我完成毕设，这将是我以后学习和创造的动力。最后，请允许我向尊敬的康磊老师表示真挚的谢意。

同时也感谢我的好朋友谢宝伟同学，在我创作时候给予我的鼓励，也协助我绘制了数个类图，功能图等。