具身只能引论-fall 2024

忻思阳 22307130465

1 实验目的

步态分析通过对人体行走、跑步等活动时的运动模式进行分析，可以提供关于运动效率、身体健康状态和行为特征的重要信息。

本实验利用手机内置的加速度传感器可以捕捉到与人体运动相关的三维加速度数据，通过对这些数据的分析，不仅能够了解步态特征，还可以进一步估算步速、步数等参数。

2 实验流程

2.1 数据采集

使用手机内置的三轴加速度传感器采集数据，利用 MATLAB APP 实时记录数据，勾选手机应用中的加速度传感器采集功能，仅采集加速度数据。

记录的数据范式如下：

```
Timestamp X Y Z
15-Dec-2024 07:14:21.487 -0.744766 2.882155 9.31466
```

2.2 相关参数

采样频率： 设置采样频率为 100 Hz，这样的采样频率较为适中，能够覆盖人体运动时加速度的主要变化情况，避免采样频率过低数据失真。

采样时长： 每组实验的采样时长设定为 30 秒，保证采集的数据足够用于分析运动特征。每种步态重复 5 次实验，便于计算平均值并分析误差，避免因单次错误实验导致的严重误差，破坏实验的准确性。

数据保存： 数据保存为.csv 格式文件，便于后续处理和分析。每个实验的文件命名格式为步态类型_ 实验编号.csv

手机摆放位置： 为了减少数据波动的随机性，将手机放置于贴身口袋，位置大概在腰部中央，与人体重心位置接近。这样不仅有利于捕捉人体整体运动的加速度特征，还能避免因手臂摆动或脚步震动引入的噪声。

实验步态设置：

正常匀速步行：

即匀速行走，保持自然的步幅和步频。记录正常步态下的加速度信号特征，用于基准比较，输出为文件：walk_n.csv

正常匀速跑步：

以自定义配速慢跑，记录跑步步态特征，尽量保持匀速，避免信号波动过大，输出为文件：run_n.csv

正常步行 - 跑步切换
以正常匀速步行开始，随后切换为跑步，最后再切换回步行，用于分析步态切换过程中的加速度信号变化特征，输出为文件：walkrun_n.csv
以上 n 为实验数据的组号

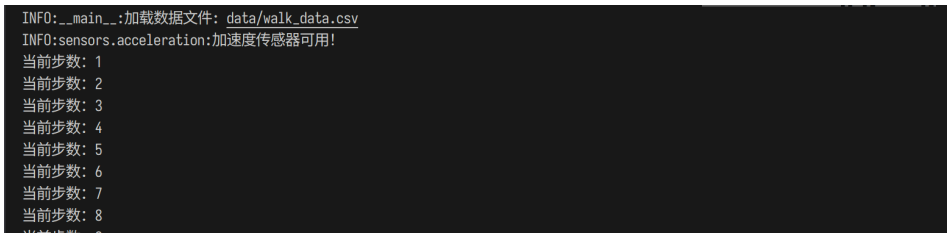

Figure 1: output

Figure 2: output

输出示例：

3 数据处理

数据处理是步态分析的核心环节，旨在对原始采集的加速度数据进行预处理、特征提取和计算分析。

3.1 预处理

预处理的目标是去除原始加速度数据中的噪声和无用信息，为后续分析提供清晰的信号。

时间的归一化： 原始数据通常包含时间记录（例如 15-Dec-2024 07:14:21.487）。将时间转换为相对时间（以采集开始时间为 0，单位为毫秒）以便于分析连续的时间序列信号，处理后时间列为 rel_time_ms。

加速度的合成 由于人体运动通常三维的，因此提取三轴的加速度 x，y，z
合成加速度计算公式为：

$$A_{\text{total}} = \sqrt{X^2 + Y^2 + Z^2}$$

后续可以用于检测运动特征。

去均值： 数据中可能存在重力分量，因此加速度传感器可能在静止状态下存在偏移，所以采用去均值方法通过减去数据的平均值，使信号中心化：

$$A_{\text{centered}} = A - \text{mean}(A)$$

3.2 滤波

滤波用于去除信号中的高频噪声或低频干扰，保留步态运动的有效频率特征。

低通滤波：
步态的主要频率通常低于 $3H_z$，我通过快速傅里叶变换，将高于设定截止频率的频率成分置零，仅保留低频部分。
采用的公式如下：

傅里叶变换：

$$F(f) = \mathrm{FFT}(A)$$

高频去除：

$$F(f) = 0, \quad \forall f > f_{\mathrm{cutoff}}$$

逆傅里叶变换：

$$A_{\mathrm{filtered}} = \mathrm{IFFT}(F(f))$$

高通滤波： 用于去除信号中的低频分量（例如缓慢的传感器漂移或重力影响），并设置低频截止点，仅保留高于某频率的部分。

移动平均滤波： 通过滑动窗口对信号平滑处理，消除短期波动：

$$A_{\mathrm{smooth}}[i] = \frac{1}{N} \sum_{j=i-N+1}^{i} A[j]$$

其中 N 是窗口大小，实验中选择 N=5（可根据实际情况提高 N，更精确）

3.3 步态特征提取:

步态特征提取是分析加速度信号中每一步的特征。

峰谷检测： 步态信号中的每一步通常表现为加速度的一个波动周期（一个峰值和一个谷值）。检测波峰（步态的最大值）和波谷（步态的最小值）的方法如下：
波峰条件：加速度从上升切换为下降的转折点（局部最大值）

$$A[i] > A[i-1] \quad \text{且} \quad A[i] > A[i+1]$$

波谷条件：加速度从下降切换为上升的转折点（局部最小值）

$$A[i] < A[i-1] \quad \text{且} \quad A[i] < A[i+1]$$

由于步态信号的周期性波动，波谷和波峰的出现是相辅相成的，不能忽略任何一方

步数和速度计算： 每次检测到波峰就认为完成了一步。我们认为：波峰之间的时间间隔在合理范围内、波峰与波谷之间的振幅超过设定阈值即完成一步。
假设每一步的平均步长为 L（实验中取 0.7 米），根据步数 N 和时间 T 计算速度：

$$v = \frac{T}{L \cdot N}$$

数据标准化： 为便于不同步态数据的对比分析，对加速度数据进行标准化处理：

$$A_{\mathrm{standardized}} = \frac{A - \mathrm{mean}(A)}{\mathrm{std}(A)}$$

其中 mean(A) 是信号的均值，std(A) 是信号的标准差。

4 实验方法——各代码分析

4.1 各功能分析

4.1.1 utils/signal_processing.py

该模块主要用于信号处理，为步态分析的预处理部分提供功能。核心内容包括：滤波（移动平均滤波、FFT 低通滤波、FFT 高通滤波）、数据去均值、数据标准化、综合的多通道数据预处理函数

移动平均滤波函数：moving_average_filter 该函数对 1D 数据应用滑动窗口平均滤波，平滑信号。

快速傅里叶变换低通滤波：fft_lowpass_filter 利用快速傅里叶变换（FFT）实现低通滤波。

快速傅里叶变换高通滤波：fft_highpass_filter 利用快速傅里叶变换（FFT）实现高通滤波。

数据去均值：remove_mean 从信号中减去均值，消除直流分量。

数据标准化：standardize 对数据进行标准化，使其均值为 0，标准差为 1。

数据预处理函数：preprocess_data 对三轴数据（x, y, z）进行去均值、滤波和平滑处理。

各函数伪代码如下：

Algorithm 1 Moving Average Filter

Require: data: array of numbers, window_size: integer
Ensure: filtered: array of filtered values
if window_size < 1 **then**
  raise "Window size must be at least 1" // Window size check
end if
cumsum $\leftarrow$ cumulative_sum($[0] + \text{data}$) // Compute cumulative sum for efficiency
filtered $\leftarrow \frac{\text{cumsum[window\_size:]} - \text{cumsum[:-window\_size]}}{\text{window\_size}}$ // Compute moving average
front_fill $\leftarrow$ [filtered[0]] $\times$ (window_size $- 1$) // Fill initial values
return front_fill + filtered // Concatenate and return result

Algorithm 2 FFT Lowpass Filter

Require: data: input signal, cutoff_freq: cutoff frequency, sampling_rate: sampling rate
Ensure: filtered_data: signal after lowpass filtering
$n \leftarrow$ length(data) // Length of the data
freqs $\leftarrow$ calculate_frequencies(n, sampling_rate) // Compute frequency range
fft_data $\leftarrow$ fft(data) // Perform FFT on the signal
for each frequency **in** freqs **do**
  if frequency $>$ cutoff_freq **then**
    fft_data[frequency] $\leftarrow 0$ // Zero out high frequencies
  end if
end for
filtered_data $\leftarrow$ ifft(fft_data) // Transform back to time domain
return filtered_data

4.1.2 utils/file_io.py

这部分代码主要用来读取文件中的数据

4.1.3 sensors/acceleration.py

StepSensorAcceleration 是基于加速度传感器实现步数检测的类，核心功能包括：波峰与波谷检测、步数更新、动态阈值调整、数据监听。

Algorithm 3 FFT Highpass Filter

Require: data: input signal, cutoff_freq: cutoff frequency, sampling_rate: sampling rate
Ensure: filtered_data: signal after highpass filtering

$n \leftarrow$ length(data) // Length of the data
freqs $\leftarrow$ calculate_frequencies(n, sampling_rate) // Compute frequency range
fft_data $\leftarrow$ fft(data) // Perform FFT on the signal
for each frequency **in** freqs **do**
  if frequency $<$ cutoff_freq **then**
    fft_data[frequency] $\leftarrow 0$ // Zero out low frequencies
  end if
end for
filtered_data $\leftarrow$ ifft(fft_data) // Transform back to time domain
return filtered_data

Algorithm 4 Remove Mean

Require: data: input signal
Ensure: centered_data: mean-centered signal

mean_value $\leftarrow$ calculate_mean(data) // Compute the mean of the signal
centered_data $\leftarrow$ data $-$ mean_value // Subtract mean to center the data
return centered_data

Algorithm 5 Standardize

Require: data: input signal
Ensure: standardized_data: standardized signal

mean_value $\leftarrow$ calculate_mean(data) // Compute the mean
std_value $\leftarrow$ calculate_std(data) // Compute the standard deviation
standardized_data $\leftarrow \frac{\text{data}-\text{mean\_value}}{\text{std\_value}}$ // Standardize the data
return standardized_data

Algorithm 6 Preprocess Data

Require: x, y, z: input signals, sampling_rate: sampling rate, lowpass_cutoff: lowpass cutoff frequency, highpass_cutoff: highpass cutoff frequency
Ensure: x_proc, y_proc, z_proc: preprocessed signals

// Step 1: Remove mean
$x\_proc \leftarrow$ remove_mean(x)
$y\_proc \leftarrow$ remove_mean(y)
$z\_proc \leftarrow$ remove_mean(z)
// Step 2: Lowpass filtering (if applicable)
if lowpass_cutoff $\neq$ None **then**
  $x\_proc \leftarrow$ fft_lowpass_filter(x_proc, lowpass_cutoff, sampling_rate)
  $y\_proc \leftarrow$ fft_lowpass_filter(y_proc, lowpass_cutoff, sampling_rate)
  $z\_proc \leftarrow$ fft_lowpass_filter(z_proc, lowpass_cutoff, sampling_rate)
end if
// Step 3: Highpass filtering (if applicable)
if highpass_cutoff $\neq$ None **then**
  $x\_proc \leftarrow$ fft_highpass_filter(x_proc, highpass_cutoff, sampling_rate)
  $y\_proc \leftarrow$ fft_highpass_filter(y_proc, highpass_cutoff, sampling_rate)
  $z\_proc \leftarrow$ fft_highpass_filter(z_proc, highpass_cutoff, sampling_rate)
end if
// Step 4: Apply moving average smoothing
$x\_proc \leftarrow$ moving_average_filter(x_proc, window_size $= 5$)
$y\_proc \leftarrow$ moving_average_filter(y_proc, window_size $= 5$)
$z\_proc \leftarrow$ moving_average_filter(z_proc, window_size $= 5$)
return x_proc, y_proc, z_proc

register_step_listener　激活加速度传感器的步数检测功能。

unregister_step_listener　停止监听加速度传感器的数据（未实现具体功能，保留接口）。

on_sensor_data　接收传感器的实时数据，并计算加速度向量模长，传递给步数检测函数。

detectorNewStep　功能：通过比较当前值和前一个值，判断是否出现波峰，更新步数。

preStep　检测到有效波峰时，步数加 1

DetectorPeak　通过信号方向的变化，判断是否存在波峰或波谷。

Peak_Valley_Thread　动态调整波峰波谷差值的阈值

averageValue　计算一个滑动窗口中的平均值，并返回动态调整的阈值。
伪代码如下：

Algorithm 7 Register Step Listener

1: Set sensor as available
2: Log "Sensor is available"

Algorithm 8 Unregister Step Listener

1: Stop listening to sensor data

Algorithm 9 On Sensor Data

Require: x, y, z, timestamp: sensor data values
1: magnitude $\leftarrow \sqrt{x^2 + y^2 + z^2}$
2: Call **detectorNewStep**(magnitude, timestamp)

Algorithm 10 Detector New Step

Require: current_value, timestamp

```
if no previous value then
    Set current_value as initial gravity
else
    if is_peak_detected(current_value, previous_value, timestamp) then
        Update peak timestamps
        if peak is valid by time and amplitude then
            Count a step
            Adjust detection threshold
        end if
    end if
    Update previous_value to current_value
end if
```

Algorithm 11 Pre Step

```
Increment step count
Notify step detected
```

Algorithm 12 Detector Peak

Require: current_value, previous_value, timestamp

```
Update direction based on current and previous values
if direction switches and peak conditions are met then
    Store as peak or valley
    return True
else
    return False
end if
```

Algorithm 13 Peak Valley Thread

Require: difference

```
if threshold history is not full then
    Add difference to history
else
    Calculate average threshold from history
    Update history with new difference
end if
return Updated threshold
```

Algorithm 14 Average Value

Require: history, size

```
average ← calculate mean of history
return Adjusted threshold based on average range
```

4.2 sensors/base.py

这段代码定义了一个 StepSensorBase，为计步器相关的功能提供了统一的接口和基础功能。

定义了 CURRENT_STEP, 作为全局步数计数，所有继承类共享此计数器; 定义了 step_callback，用来存储外部回调函数，触发步数更新时调用；定义了 is_available, 标识当前传感器是否可用。

4.3 sensors/pedometer.py

这段代码实现了一个基于系统内置计步传感器的具体子类 StepSensorPedometer，继承自 StepSensorBase。它模拟了两种计步器工作模式的处理逻辑：detector（每次传感器检测到步伐时触发）和 counter（传感器直接返回累计步数），并通过回调机制实时更新步数。

5 不同步态可视化

不同的步态可以通过加速度来体现，由于我们测的是三轴的加速度，需要对其用公式

$$A_{\text{total}} = \sqrt{X^2 + Y^2 + Z^2}$$

来合成。

处理后的数据将其以时间为横轴，加速度的值为纵轴画图，可以得到如下的图（这里仅三种状态各画一副作为参考）：

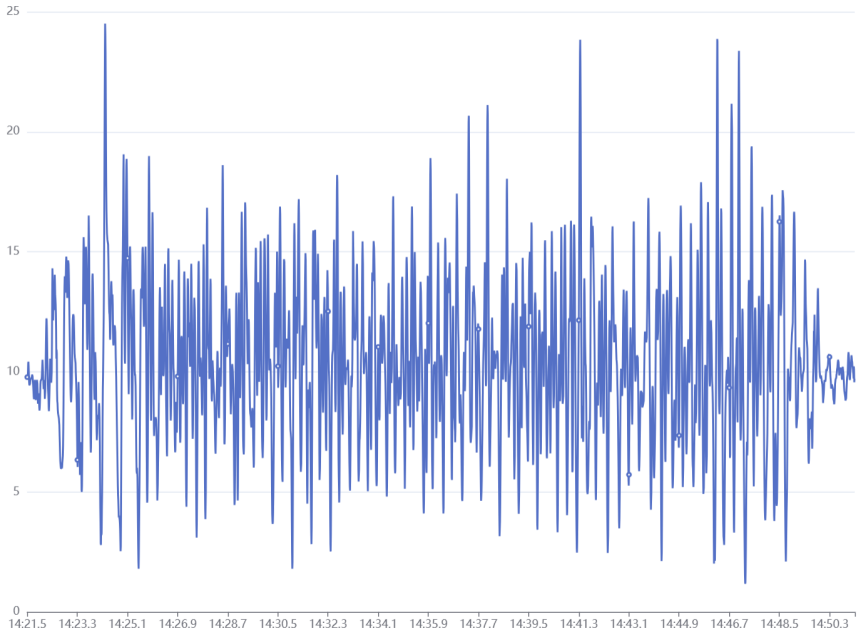

Figure 3: the acceleration of running

Figure 4: the acceleration of walking

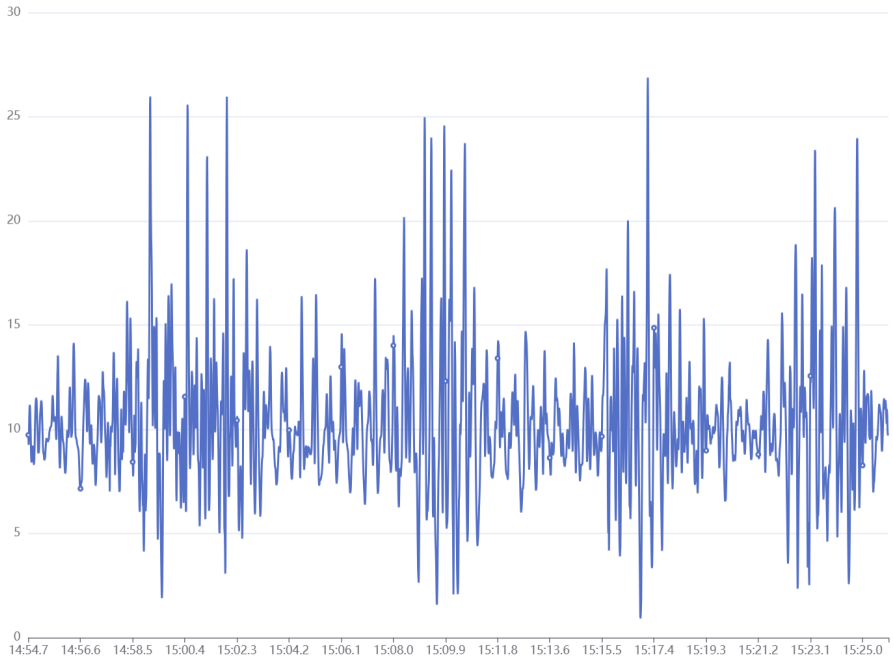

Figure 5: the acceleration of running and walking

6 行走速度可视化

以时间为横轴，速度的值为纵轴画图，可以得到如下的图（这里仅三种状态各画一副作为参考）

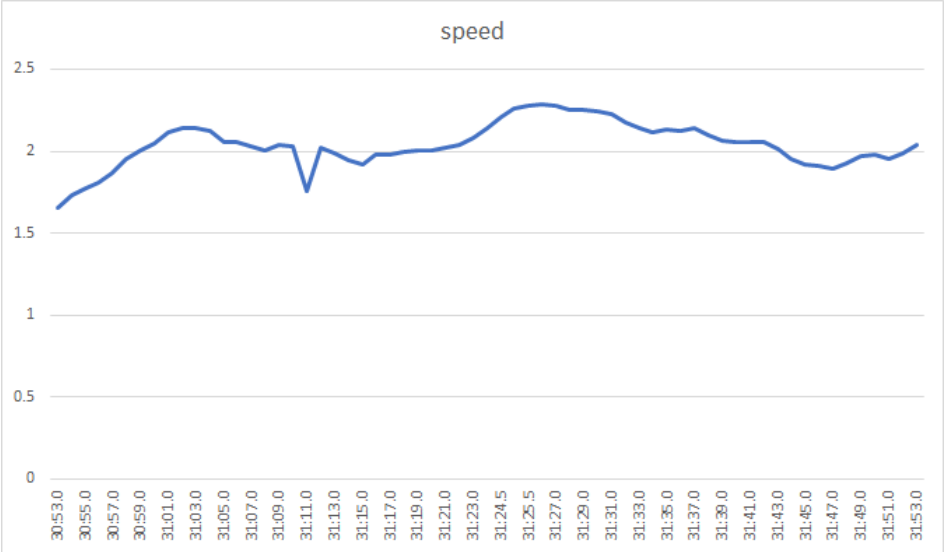

Figure 6: the speed of running

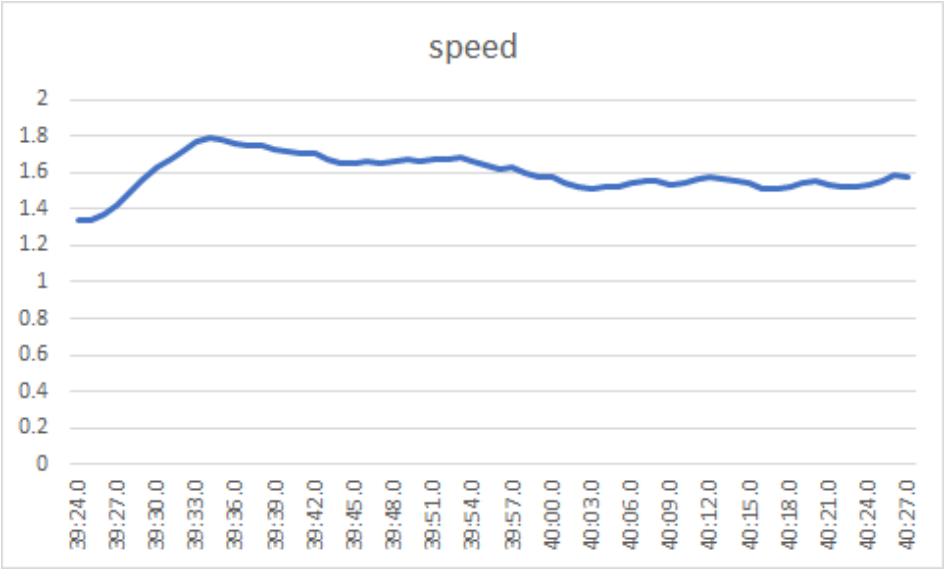

Figure 7: the speed of walking

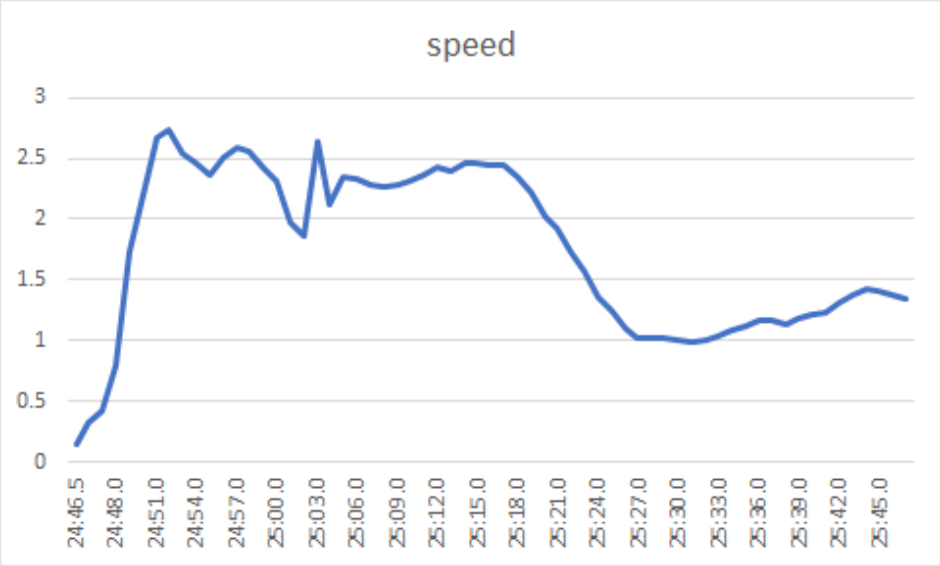

Figure 8: the speed of running and walking