**项目概述**

本项目是一个基于PyQt5和Matplotlib的串口数据可视化工具，用于实时显示关节角度数据，并进行步态分析。通过串口接收数据，程序能够实时绘制关节角度曲线，并计算步态周期、摆动相时间、支撑相时间等步态参数。

**项目结构**

**主要文件**

* serial\_plotter.py：主程序文件，包含GUI界面、串口通信、数据解析和步态分析逻辑。

**依赖库**

* PyQt5：用于构建GUI界面。
* matplotlib：用于绘制实时数据曲线。
* pyserial：用于串口通信。
* numpy：用于数据处理和计算。

**功能模块**

**1. GUI界面**

* **主窗口**：包含开始、停止、重置步态数据按钮，串口选择下拉菜单，实时数据显示区域和绘图区域。
* **按钮功能**：
  + **Start**：开始读取串口数据。
  + **Stop**：停止读取串口数据。
  + **Reset Gait Data**：重置步态分析数据。

**2. 串口通信**

* **串口选择**：通过下拉菜单选择可用的串口。
* **数据读取**：从串口读取数据，解析关节角度、ID1、ID2等信息。
* **数据解析**：解析串口数据，提取关节角度、角速度等信息。

**3. 数据可视化**

* **实时绘图**：实时绘制关节角度曲线。
* **动态更新**：通过QTimer定时更新UI，确保曲线和数据实时刷新。

**4. 步态分析**

* **步态周期计算**：计算步态周期的总时间、摆动相时间和支撑相时间。
* **相位检测**：根据角速度数据检测当前步态相位（摆动相或支撑相）。
* **动态阈值**：根据角速度的标准差动态调整阈值，用于检测步态变化。

**关键代码说明**

**1. GUI初始化**

class SerialPlotter(QWidget):  
 def \_\_init\_\_(self):  
 super().\_\_init\_\_()  
 self.setWindowTitle("Serial Plotter with Gait Analysis")  
 self.setGeometry(100, 100, 800, 600)  
 self.layout = QVBoxLayout()  
 # 初始化按钮、绘图区域、数据显示区域等

**2. 串口数据读取**

def read\_serial\_data(self):  
 while not self.stop\_event.is\_set():  
 try:  
 x1 = self.serial\_port.read(2)  
 if x1.hex() == '5aa5':  
 x2 = self.serial\_port.read(4)  
 x3 = self.serial\_port.read(8)  
 if x3[2:3].hex() == '02':  
 x4 = self.serial\_port.read(152)  
 ID1 = struct.unpack('B', x4[1:2])[0]  
 ID2 = struct.unpack('B', x4[77:78])[0]  
 Roll = x4[51:52] + x4[50:51] + x4[49:50] + x4[48:49]  
 Roll = struct.unpack('!f', Roll)[0]  
 Roll1 = x4[127:128] + x4[126:127] + x4[125:126] + x4[124:125]  
 Roll1 = struct.unpack('!f', Roll1)[0]  
 roll\_value = 180 - abs(Roll - Roll1)  
 self.data\_buffer\_roll.append(roll\_value)  
 # 解析角速度数据并进行步态分析

**3. 步态分析**

def low\_pass\_filter(self, new\_value, alpha=0.2):  
 self.filtered\_angular\_velocity = alpha \* new\_value + (1 - alpha) \* self.filtered\_angular\_velocity  
 return self.filtered\_angular\_velocity

def read\_serial\_data(self):  
 # 解析角速度数据  
 angular\_velocity = x4[31:32] + x4[30:31] + x4[29:30] + x4[28:29]  
 angular\_velocity = struct.unpack('!f', angular\_velocity)[0]  
 filtered\_angular\_velocity = self.low\_pass\_filter(angular\_velocity)  
 self.angular\_velocity\_buffer.append(filtered\_angular\_velocity)  
 # 动态调整阈值并检测步态相位

# Calculate dynamic threshold (mean + standard deviation)

if len(self.angular\_velocity\_buffer) > 10:

mean\_angular\_velocity = np.mean(self.angular\_velocity\_buffer)

std\_angular\_velocity = np.std(self.angular\_velocity\_buffer)

self.angular\_velocity\_threshold = mean\_angular\_velocity + std\_angular\_velocity

# Detect walking based on angular velocity variation

if std\_angular\_velocity > 10.0: # If standard deviation is high, assume walking

self.is\_walking = True

else:

self.is\_walking = False

# Update gait phase based on filtered angular velocity

current\_time = datetime.datetime.now()

time\_diff = (current\_time - self.last\_phase\_change\_time).total\_seconds()

if self.is\_walking:

if abs(filtered\_angular\_velocity) > self.angular\_velocity\_threshold:

if self.current\_phase != "Swing":

self.current\_phase = "Swing"

self.last\_phase\_change\_time = current\_time

self.stance\_phase\_time += time\_diff # Update Stance phase time

if self.current\_gait\_start\_time is None:

self.current\_gait\_start\_time = current\_time # Start of a new gait cycle

else:

if self.current\_phase != "Stance":

self.current\_phase = "Stance"

self.last\_phase\_change\_time = current\_time

self.swing\_phase\_time += time\_diff # Update Swing phase time

# Update total gait cycle time

if self.current\_gait\_start\_time is not None:

self.gait\_cycle\_time = (current\_time - self.current\_gait\_start\_time).total\_seconds()

else:

self.current\_gait\_start\_time = None

self.gait\_cycle\_time = 0.0

self.swing\_phase\_time = 0.0

self.stance\_phase\_time = 0.0

**4. 数据可视化**

def update\_ui(self):  
 if self.data\_buffer\_roll and self.data\_buffer\_ID1 and self.data\_buffer\_ID2:  
 x\_data = list(range(len(self.data\_buffer\_roll)))  
 y\_roll = list(self.data\_buffer\_roll)  
 self.ax\_roll.clear()  
 self.ax\_roll.plot(x\_data, y\_roll, label='Roll')  
 self.ax\_roll.set\_ylim([min(y\_roll) - 90, max(y\_roll) + 90])  
 self.ax\_roll.legend()  
 self.canvas.draw()  
 # 显示步态分析数据

若要详细了解该串口协议，参考以下文档：

