嵌入式系统开发期末考试设计报告

小组成员：

刘洋 202231060901

王麒围 202231061001

刘传友 202231061430

刘兴旺 202231061031

分工情况：

刘洋：数据获取及数据显示，代码编码量40%

王麒围：数据存储，数据库的使用，代码编码量20%

刘传友：页面设计，采集数据，代码编码量20%

刘兴旺：选用模型对数据进行训练，代码编码量20%

最终实现的功能：

本项目旨在开发一个基于 PyQt 框架的跨平台嵌入式应用程序，功能主要包括数据采集、显示、存储和基于机器学习的动作识别。以下是详细的功能模块和要求：

#### 1. 数据采集与接口：

* **目标**：通过串口、以太网、WiFi 等接口，实时采集人体动作过程中来自 RTOS 实验箱的传感器数据，包括三轴重力加速度、温湿度、大气压力、光照等传感器信息。
* **实现**：通过与传感器硬件的连接，实时获取数据并进行处理。

#### 2. 实时数据显示：

* **目标**：实时显示采集到的各类数据。
* **实现**：
  + 以数值的方式显示三轴重力加速度、温湿度、大气压力、光照等数据。
  + 以曲线的方式实时显示三轴重力加速度数据，便于观察动作变化。

#### 3. 实时计算与显示采样率：

* **目标**：计算并实时显示数据的采样率，确保数据采集过程稳定。
* **实现**：在采集数据的同时计算采样间隔，并动态显示当前采样率。

#### 4. 数据存储与管理：

* **目标**：选择合适的数据库（如 SQLite3）存储传感器数据，并以表格形式进行展示。
* **实现**：使用 QTableView + QSqlTableModel 结合 SQLite3 实现数据的持久化存储，并支持分页显示。
  + 包括功能：分页浏览（首页、尾页、上一页、下一页、手动输入页数）以及数据导出。

#### 5. 带标签的样本采集：

* **目标**：用户可以选择当前动作标签（如上楼、跑步、散步、汽车等），采集标注数据并存储。
* **实现**：为每个数据采集任务添加标签，标签与数据一起存储在数据库中，便于后期的模型训练和分析。

#### 6. 数据导出功能：

* **目标**：支持将采集的数据导出为合适的格式（如 CSV、Excel 等），方便后期数据处理和模型训练。
* **实现**：在数据列表中提供导出功能，导出选定的数据到文件。

#### 7. 动作识别模型训练与推理：

* **目标**：使用采集到的数据进行机器学习模型训练，识别人体动作。
* **实现**：
  + **模型训练**：选择合适的机器学习模型（如决策树、支持向量机、深度学习等），在标注数据上训练模型。
  + **模型推理**：训练后的模型在应用程序中进行实时推理，识别当前人体动作并显示在界面中。

#### 8. 项目结构与界面设计：

* **目标**：采用结构化、模块化的开发方式，确保代码的可维护性与扩展性；界面设计简洁、美观。
* **实现**：
  + 按照功能模块进行代码组织，使用 Qt 的 MVC 设计模式（Model-View-Controller）。
  + 界面简洁、易用，具有良好的用户体验。

设计过程：

### 1. 需求分析

首先，需要对项目的需求进行详细分析，确保每个功能模块的设计目标清晰明确。关键需求包括：

* 数据采集：通过不同接口（串口、WiFi、以太网等）获取传感器数据。
* 数据展示：实时显示数据，以数值和曲线形式展示。
* 数据存储：选择合适的数据库存储数据，并支持数据分页浏览。
* 标签采集：支持标签化数据采集，便于后期模型训练。
* 数据导出：支持将采集的数据导出为文件。
* 动作识别：基于数据训练模型，并在实时应用中进行动作识别。

### 2. 系统设计

#### 2.1 整体架构设计

系统的架构可以分为三个主要层次：

**数据采集层**：从传感器设备（RTOS 实验箱）获取数据，主要包括三轴重力加速度、温湿度、大气压力、光照等传感器数据。采用串口、WiFi 或以太网等方式与传感器进行通信。

**数据处理与展示层**：将采集的数据进行实时计算、展示和存储。该层包括：

* + 实时数据显示模块（数值显示与曲线展示）
  + 数据采样率计算模块
  + 数据存储模块（使用 SQLite3 数据库）
  + 数据列表管理与分页显示

**模型训练与推理层**：使用采集到的数据进行机器学习模型的训练，训练完的模型用于动作识别。该层包括：

* + 数据标注与样本采集
  + 机器学习模型训练与评估
  + 动作推理模块

#### 2.2 技术选型

* **前端**：采用 **PyQt5/6** 构建用户界面，利用 Qt 的事件驱动和信号槽机制来管理 UI 组件。
* **后端**：选择 **SQLite3** 作为本地数据库存储数据。
* **数据分析与机器学习**：可以使用 **scikit-learn**、**TensorFlow** 或 **PyTorch** 进行数据建模与推理。

### 3. 模块设计与开发

#### 3.1 数据采集模块

* **串口通信**：利用 PyQt 的串口通信类（如 QSerialPort）来接收传感器数据。
* **网络通信**：可以使用 socket 模块通过以太网或 WiFi 获取数据，支持实时数据流的接收。
* **数据解析**：根据传感器数据格式进行解析，将原始数据转换为三轴加速度、温湿度、大气压力等数值。

#### 3.2 数据展示模块

* **数值显示**：通过 QLabel、QTextEdit 或 QLineEdit 实现数据的实时数值展示。
* **曲线显示**：使用 **Matplotlib** 或 **PyQtGraph** 库进行数据的曲线绘制，实时展示三轴重力加速度的数据变化。
* **采样率显示**：通过计算数据采集间隔并实时更新显示。

#### 3.3 数据存储与管理

* **SQLite3 数据库**：用于存储传感器数据以及标签化的样本数据。表格设计包括：数据 ID、时间戳、传感器类型（加速度、温湿度等）、数据值、标签等字段。
* **QTableView + QSqlTableModel**：通过 QTableView 展示数据表，并利用 QSqlTableModel 进行数据管理。
* **分页功能**：使用 QSqlTableModel 提供的分页机制，或者通过自定义分页查询来实现数据的分批显示。

#### 3.4 标签采集与数据导出模块

* **标签采集**：通过界面让用户选择当前活动的标签（例如：跑步、上楼、散步等），并开始采集数据，数据保存时附带标签信息。
* **数据导出**：实现数据导出功能，支持导出为 CSV、Excel 等格式，便于后续处理与模型训练。

#### 3.5 动作识别模型

* **模型选择与训练**：
  + **数据预处理**：数据会进行归一化、滤波等预处理。
  + **特征提取**：可以从重力加速度数据中提取特征，如均值、方差、峰值等统计特征。
  + **训练**：使用标注数据训练机器学习模型，可能的模型包括：
    - 决策树
    - 随机森林
    - 支持向量机（SVM）
    - 深度神经网络（如 LSTM）
* **推理与识别**：训练完成的模型将嵌入到应用中，实时识别采集数据对应的动作标签。

### 4. 数据库设计

#### 4.1 数据库表设计

**SensorData** 表：

* + id INTEGER PRIMARY KEY
  + timestamp DATETIME
  + sensor\_type TEXT (例如：加速度、温湿度等)
  + value\_x FLOAT
  + value\_y FLOAT
  + value\_z FLOAT
  + temperature FLOAT
  + humidity FLOAT
  + pressure FLOAT
  + light\_intensity FLOAT

**LabelData** 表：

* + id INTEGER PRIMARY KEY
  + timestamp DATETIME
  + sensor\_data\_id INTEGER (外键关联 SensorData)
  + label TEXT (例如：跑步、上楼等)

#### 4.2 数据管理

* **数据分页**：通过 QSqlTableModel 或自定义分页查询来实现分页浏览。
* **数据查询**：提供数据查询、排序和筛选功能。

### 5. 界面设计

* **主界面**：包括数据展示面板、实时数据曲线、采样率显示、当前标签选择、数据列表等。
* **标签采集面板**：用于选择当前的活动标签（上楼、跑步、散步等）。
* **数据列表**：展示数据库中的数据，并提供分页、查询、导出等功能。
* **设置与参数配置**：允许用户调整数据采集频率、接口设置等参数。

6. 测试与优化

* **单元测试**：对各个模块（数据采集、数据展示、模型训练等）进行单元测试，确保各功能正常运行。
* **性能优化**：针对实时数据采集与显示，优化数据处理速度，减少延迟，提高数据采集的稳定性。
* **UI 测试**：确保界面简洁、易用，操作流畅。

遇到的问题及解决办法：

### 1. ****数据采集延迟与丢失****

* **问题描述**：数据从传感器采集时可能会出现延迟或者数据丢失，尤其是在采集频率较高时。
* **解决办法**：
  + **优化通信协议**：如果通过串口通信，确保波特率与数据包大小设置合理，避免超时。使用适当的缓冲区大小，减少丢包。
  + **多线程处理**：将数据采集与数据处理分开，使用多线程（如 QThread）来并行处理数据的接收和展示，减少主线程的负担。
  + **增大采样缓冲区**：增加数据缓冲区大小，确保数据不会因为短时间内无法处理而丢失。

### 2. ****实时数据展示卡顿****

* **问题描述**：当数据量大或者更新频繁时，界面更新可能会出现卡顿，影响用户体验。
* **解决办法**：
  + **优化 UI 更新**：通过减少不必要的 UI 更新频率，避免每次数据更新时都重绘整个界面。可以使用 QTimer 设置固定的更新频率，而不是每次数据变化时都立即更新。
  + **采用高效的图形库**：使用 PyQtGraph 或者 Matplotlib 等高效绘图库绘制数据曲线，避免 PyQt 自带的图形绘制控件造成性能瓶颈。
  + **分离数据处理和 UI 更新**：通过多线程将数据处理和 UI 更新分离，确保界面更新不会影响数据采集与处理。

3. **数据库性能问题**

* **问题描述**：随着数据量增大，SQLite3 数据库可能出现性能下降，尤其是在查询、分页和数据插入时。
* **解决办法**：
  + **索引优化**：确保数据库表上有合适的索引（如时间戳、传感器类型、数据 ID），提高查询效率。
  + **批量插入**：在插入大量数据时，避免每次插入都提交事务，采用批量插入的方式，将数据一次性写入数据库。
  + **分页查询优化**：对于分页查询，避免一次性加载所有数据，使用适当的 SQL 查询语句限制返回的数据量，并动态加载数据。

### 4. ****模型训练时间过长****

* **问题描述**：在进行模型训练时，尤其是当数据集较大或者采用复杂模型（如深度学习）时，训练时间可能非常长。
* **解决办法**：
  + **特征选择与降维**：通过提取重要的特征，去除冗余数据，减少输入数据的维度，从而加速训练过程。
  + **模型简化**：如果使用深度学习模型，可以考虑简化模型结构，减少参数数量，或者采用更快的算法（如 SVM 或决策树）。
  + **使用 GPU 加速**：如果采用深度学习模型，确保使用 GPU 来加速训练过程，使用 TensorFlow 或 PyTorch 等框架支持 GPU 训练。

### 5. ****动作识别准确性不高****

* **问题描述**：训练出的模型可能在动作识别时出现误判，准确率不高。
* **解决办法**：
  + **数据标注质量**：确保采集的数据具有高质量的标注，标签准确无误。考虑数据清洗与去噪处理，消除无关数据的影响。
  + **特征工程**：在重力加速度数据中提取更多的统计特征（如均值、标准差、频域特征等），以提高模型的区分能力。
  + **数据扩增**：增加数据的多样性，通过对现有数据进行增广（例如，加入噪声、数据平移等）来提升模型的泛化能力。
  + **模型调优**：使用交叉验证来调整模型的超参数，选用适合数据特征的算法，例如使用随机森林、支持向量机（SVM）等。

### 6. ****标签采集与数据导出问题****

* **问题描述**：标签采集时，可能存在用户误操作，或者数据导出时格式不规范，导致后续模型训练失败。
* **解决办法**：
  + **标签确认机制**：在采集标签时，设计一个确认步骤，用户需确认标签选择无误后，方可开始数据采集。
  + **数据验证**：在数据导出前，进行数据格式和完整性验证，确保导出的文件能够正确读取和使用。
  + **导出多格式支持**：支持多种文件格式的导出（如 CSV、Excel），并提供简单的错误提示机制，防止用户在导出时遇到格式问题。

### 7. ****UI 界面布局问题****

* **问题描述**：界面布局不合理，可能导致用户操作不便，或者在不同屏幕尺寸上显示不适应。
* **解决办法**：
  + **响应式布局**：使用 Qt 的布局管理器（如 QVBoxLayout、QHBoxLayout）来确保界面在不同尺寸的屏幕上自适应。
  + **统一设计风格**：遵循简洁、直观的设计原则，确保界面功能清晰，并通过用户测试进一步优化。
  + **界面测试**：进行跨平台测试，确保界面在不同操作系统和设备上显示正常，避免出现界面错乱的问题。

### 设计总结：

本项目旨在实现一个基于 PyQt 框架的跨平台嵌入式应用程序，用于采集、存储、展示和分析人体动作行为数据。通过结合惯性测量单元（IMU）传感器的数据、机器学习模型的训练与推理，实现人体动作识别功能。项目的设计流程经历了需求分析、系统设计、模块开发、界面设计、数据库设计、数据处理与模型训练、测试优化等多个阶段，最终成功实现了预定目标。

#### 1. ****项目目标的明确与实现****

* **数据采集**：通过串口、WiFi、以太网等多种接口采集传感器数据，确保系统能够与 RTOS 实验箱无缝对接，实时获取三轴重力加速度、温湿度、大气压力等数据。
* **数据展示**：采用实时数值显示和曲线图展示的方式，帮助用户直观地查看传感器数据，尤其是加速度数据的动态变化。
* **数据存储与管理**：使用 SQLite3 数据库存储传感器数据和标签信息，利用 QTableView 和 QSqlTableModel 实现数据的展示、查询和分页浏览。
* **标签采集与数据导出**：用户可以为数据添加标签（如上楼、跑步、散步等），并能够方便地导出数据进行后续分析或模型训练。
* **机器学习与动作识别**：通过对采集到的标注数据进行机器学习训练，生成模型并用于实时动作识别，展示当前人体动作。

#### 2. ****关键技术与实现****

* **PyQt 框架**：作为 GUI 开发框架，PyQt 提供了强大的控件、布局管理和信号槽机制，帮助实现跨平台的界面设计。
* **SQLite3**：作为本地数据库，SQLite3 轻量级且便于嵌入应用程序，适合存储小型项目的数据。
* **Matplotlib 和 PyQtGraph**：这两个库被用于实时曲线绘制，Matplotlib 提供了丰富的绘图功能，而 PyQtGraph 则在实时数据更新和性能上表现更优。
* **机器学习**：使用 scikit-learn 或 TensorFlow 等工具进行数据预处理、特征提取、模型训练及推理，识别人体动作。

#### 3. ****模块化与结构化开发****

项目采用模块化开发，确保各个功能模块之间的独立性和可维护性：

* **数据采集模块**：专注于与硬件设备的通信和数据获取。
* **数据处理与展示模块**：负责处理原始数据并展示给用户，同时计算采样率等实时参数。
* **数据库模块**：负责数据存储、分页查询和数据管理。
* **机器学习模块**：处理数据标注、训练机器学习模型、执行实时推理。

采用结构化开发使得每个模块都能独立测试、调试和优化，同时在后期进行扩展和升级时也更加灵活。

#### 4. ****用户界面设计****

* **简洁直观**：界面设计强调简洁与直观，避免了过多复杂的操作，确保用户能够快速上手。
* **响应式布局**：通过 Qt 的布局管理器，使得界面能够适应不同尺寸的屏幕，确保在各种设备上都能正常显示。
* **实时反馈**：数据采集、显示和推理结果均及时反馈给用户，增加了应用的互动性和可操作性。

#### 5. ****挑战与问题****

在项目的开发过程中，面临了多种技术挑战：

* **数据丢失与延迟问题**：由于采集频率较高，出现了数据丢失和延迟的情况。通过多线程处理和优化串口通信协议，解决了这一问题。
* **实时数据显示卡顿**：数据量较大时，界面更新会出现卡顿，最终通过优化 UI 更新频率和使用高效的绘图库（如 PyQtGraph）来解决。
* **数据库性能问题**：随着数据量增大，数据库查询变得缓慢。通过优化 SQL 查询、索引设置和批量插入方法，显著提升了数据库性能。
* **模型训练时间过长**：机器学习模型的训练过程可能过长，通过特征降维和简化模型结构，减少了训练时间。

#### 6. ****优化与扩展****

* **性能优化**：通过多线程和数据缓存的方式减少了延迟，提高了系统的响应速度；同时，优化了数据库操作，确保大数据量下系统的高效运行。
* **用户体验改进**：增强了界面的交互性与可操作性，提供了数据导出、标签选择、页面切换等功能，提升了系统的实用性。
* **模型扩展**：未来可以引入更复杂的深度学习模型（如 LSTM 或 CNN），提升动作识别的准确性和泛化能力。

#### 7. ****项目总结与未来发展****

本项目完成了从数据采集、存储、展示到机器学习模型训练与推理的全过程。通过模块化设计与结构化开发，使得系统具备良好的可扩展性和可维护性。同时，项目的实现展示了 PyQt 在跨平台应用中的优势，并为后期的动作识别研究和应用奠定了基础。

未来，项目可以进一步优化动作识别模型的准确性，增强系统的鲁棒性，并扩展到更多类型的传感器数据采集与分析。同时，考虑将应用推向云端，进行大规模的数据分析和实时监控，进一步提升系统的功能和应用场景。