People Flow Density Monitoring & Warning System

基于卡尔曼滤波和 SARIMA 的人流密度监测预警系统

系统概述

本系统是一套综合性人流密度监测预警解决方案，利用多种传感器数据采集、卡尔曼滤波算法实时处理以及 SARIMA 模型进行时序预测，为各类公共场所（如图书馆、商场、地铁站等）提供实时人流密度监测和预警服务。

核心功能

- **多传感器数据采集**：支持红外计数器、WiFi 探针等多种传感器
- **实时数据处理**：基于卡尔曼滤波算法的噪声过滤和数据融合
- **历史数据存储与分析**：自动记录并分析历史人流数据
- **密度预测**：基于 SARIMA 时间序列模型预测未来人流密度变化
- **可视化监控界面**：直观展示实时人流密度热图和趋势图
- **预警系统**：当人流密度超过阈值时发出预警

系统架构

系统由三个主要组件构成：

1. **Raspberry Pi 传感器采集层** (Raspb/)
    - 从各类传感器采集原始数据
    - 初步处理和过滤数据
    - 通过 HTTP 客户端将数据发送至服务器
2. **数据处理服务器** (Server/)
    - 接收并存储传感器数据
    - 应用卡尔曼滤波算法处理数据
    - 提供 RESTful API 供前端访问
    - 结合历史数据应用 SARIMA 模型进行预测
3. **Web 用户界面** (WebUI/)
    - 实时显示各区域人流密度热图
    - 展示历史数据统计和趋势图
    - 提供预测功能和预警显示

📋 目录结构

```
.
├── README.md
├── start.bat
├── Raspb/
|   ├── data_mock.py
|   ├── density_monitor.log
|   ├── http_client.py
|   ├── kalman_people_flow_density.py
|   ├── sensor_simulator.py
|   └── simplified_sensor_simulator.py
```

```
├── Server/
|   ├── mock_data.py
|   ├── server.log
|   ├── server.py
|   └── density_data/
|       ├── density_data_20250406.json
|       └── ...
|   └── Model/
|       ├── density_predictor.py
|       ├── density_data/
|       |   ├── density_data_20250309.json
|       |   └── ...
|       └── models/
|           ├── density_sarima_model_library.pkl
|           └── ...
└── WebUI/
    ├── index.html
    ├── script.js
    └── styles.css
```

🚀 安装与部署

系统要求

- Python 3.10+
- Raspberry Pi 4+ (传感器节点)
- 服务器/PC (中央服务器)
- Web 浏览器 (访问前端)

1. **启动系统**

可以使用提供的`start.bat`脚本一键启动系统，或手动启动各组件:

```
# 启动服务器
cd Server
python server.py

# 另一个终端启动传感器模拟器
cd Raspb
python http_client.py # 模拟传感器程序

# 传感器程序在sensor_simulator.py中，为方便调用默认使用模拟传感器程序
```

2. **访问 Web 界面**

打开浏览器，访问: `http://localhost:5000`

📈 系统使用

Web 界面功能

- **实时监控**：首页展示各区域实时人流密度热图
- **位置选择**：可通过位置选择器切换不同监控区域
- **时间范围**：支持不同时间范围的数据查看（1 小时、3 小时、24 小时）
- **预测分析**：点击"预测"按钮可查看未来人流密度预测

预警级别

系统根据人流密度设置了三级预警:

- **低密度** (< 0.5 人/㎡): 绿色，正常状态
- **中密度** (0.5-1.0 人/㎡): 黄色，注意状态
- **高密度** (> 1.0 人/㎡): 红色，警告状态

传感器接口

支持的传感器类型

1. **红外传感器** (InfraredSensor)
   - 通过串口通信 (Serial)
   - 计数人流进出次数
2. **WiFi 探针** (WiFiProbe)
   - 通过 HTTP API 获取数据
   - 识别区域内活跃设备数量

添加新的传感器

要添加新的传感器类型，可以继承`HardwareSensor`基类:

```
class NewSensorType(HardwareSensor):
    def __init__(self, ...):
        super().__init__(sensor_id, location, interval)
        # 初始化特定参数

    def connect(self):
        # 实现连接逻辑

    def disconnect(self):
        # 实现断开连接逻辑

    def read_data(self):
        # 实现数据读取逻辑
        return data_dict
```

💻 API 参考

服务器提供以下 REST API:

GET /api/density/recent

获取最近的人流密度数据

参数:

- `limit`: 返回记录数量，默认 50

返回:

```
{
  "status": "success",
  "data": [
```

```
    {
      "location": "图书馆入口",
      "timestamp": "2025-04-15T14:23:45",
      "raw_density": 0.75,
      "filtered_density": 0.72,
      "estimated_people": 36,
      "area_size": 50
    },
    ...
  ]
}
```

GET /api/density/statistics

获取人流密度统计数据

参数:

- `hours`: 时间范围（小时），默认 24
- `location`: 可选，指定位置

返回:

```
{
  "status": "success",
  "statistics": {
    "density": {
      "current": 0.72,
      "average": 0.65,
      "max": 1.2,
      "min": 0.1
    },
    "people": {
      "current": 36,
      "average": 32.5,
      "max": 60,
      "min": 5
    }
  }
}
```

POST /api/density/predict

预测未来人流密度

请求体:

```
{
  "location": "图书馆入口",
  "hours": 12
}
```

返回:

```
{
  "status": "success",
  "prediction": {
    "location": "图书馆入口",
    "predictions": [
      {
        "timestamp": "2025-04-15T15:00:00",
        "predicted_density": 0.85
      },
      ...
    ]
  }
}
```

算法说明

卡尔曼滤波 (Kalman Filter)

系统使用卡尔曼滤波器来处理来自传感器的噪声数据，并融合多个传感器数据源：

- **状态估计**：估计真实人流密度
- **误差校正**：自适应调整测量和过程噪声
- **数据融合**：结合多种传感器输入提高精度

在`Raspb/kalman_people_flow_density.py`中实现。

SARIMA 模型 (Seasonal ARIMA)

系统使用 SARIMA (Seasonal AutoRegressive Integrated Moving Average) 时间序列模型进行人流密度预测：

- **季节性考量**：模型考虑一天中不同时段、一周中不同日期的规律变化
- **定期训练**：使用历史积累数据定期重新训练模型
- **场景适应**：为不同场景（图书馆、地铁站等）分别训练不同模型

在`Server/Model/density_predictor.py`中实现。

高级配置

调整卡尔曼滤波参数

在`Raspb/kalman_people_flow_density.py`中，可以调整以下参数：

```
# 过程噪声协方差
self.process_noise_cov = np.eye(2) * 0.01

# 测量噪声协方差
self.measurement_noise_cov = np.eye(1) * 0.1
```

修改 SARIMA 模型参数

在`Server/Model/density_predictor.py`中，可以调整模型参数：

```
# SARIMA模型参数(p,d,q)x(P,D,Q,s)
order = (1, 1, 1)
seasonal_order = (1, 1, 1, 24)  # 24小时周期
```