Raspberry Pi 5タッチハブ開発記

Behelit
2025年9月20日 00:47

Raspberry Pi 5タッチハブ開発記

開発動機

Raspberry Pi 5（8GB）と7インチタッチスクリーンで実用的なスマートダッシュボードを作った。市販のスマートディスプレイもあるが、自分のニーズに完全に合わせたシステムを直接作りたかった。

システム構成

ハードウェア

- Raspberry Pi 5（8GB RAM）
- 7インチタッチスクリーン（800×480）
- CSIカメラ（赤外線対応）
- 各種センサー（温湿度、ガスなど）
- **3Dプリント製カスタムフレーム**（カーボンファイバー風仕上げ）

ソフトウェア構成

- **バックエンド**: Python FastAPI
- **フロントエンド**: 純粋なHTML/CSS/JavaScript
- **カメラ処理**: OpenCV
- **システム監視**: システム監視ツール連携
- **天気情報**: OpenWeatherMap API

ハードウェア設計とカスタムフレーム

3Dプリント製フレーム設計

実用性と美観を両立させるため、3Dプリンターで専用フレームを設計・製作した。カーボンファイバー風の仕上げで、プロフェッショナルな外観を実現。

設計のポイント：

- **放熱設計**: Raspberry Pi 5の発熱に対応した通気性重視の構造
- **ケーブル管理**: 各種ケーブルをスマートに収納する配線経路
- **拡張性**: 将来的なセンサー追加に対応できるモジュラー設計
- **操作性**: タッチスクリーンへのアクセスを妨げない最適な角度

```
# フレーム設計パラメータ（参考値）
FRAME_DIMENSIONS = {
    "width": 125,       # mm
    "height": 85,       # mm
    "depth": 45,        # mm
    "wall_thickness": 2.0,  # mm
    "ventilation_holes": 1.5,  # mm径
}
```

フレーム前面には以下の要素を配置：

- **カメラレンズ開口部**（中央）
- **ステータスLED表示窓**（上部）
- **センサー通気口**（側面）
- **各種インジケーター**（コーナー部）

主な機能

1. 3ページスワイプUI

小さな画面でも使いやすいよう、3つの画面を横スワイプで切り替える設計にした：

- **ページ1**: CSIカメラライブストリーミング
- **ページ2**: 天気情報 + システム監視状態
- **ページ3**: 音声・テキスト命令コンソール

2. リアルタイムカメラストリーミング

CSIカメラからのMJPEGストリーミングを実装。夜間の赤外線撮影にも対応している。

```python
@app.get("/cam/csi")
async def csi_camera_stream():
    """CSI camera MJPEG stream"""
    return StreamingResponse(
        mjpeg_generator(csi_cap),
        media_type="multipart/x-mixed-replace; boundary=frame"
    )

def mjpeg_generator(camera):
    """Generate MJPEG frames"""
    while True:
        success, frame = camera.read()
        if not success:
            continue

        _, buffer = cv2.imencode('.jpg', frame,
                                 [cv2.IMWRITE_JPEG_QUALITY, 80])
        frame_bytes = buffer.tobytes()

        yield (b'--frame\r\n'
               b'Content-Type: image/jpeg\r\n\r\n' + frame_bytes + b'\r\n')
```

3. インテリジェント音声コマンド処理

自然言語でのコマンドを解析し、適切なアクションを実行する。

```python
@app.post("/assistant/command")
async def process_command(request: CommandRequest):
    """Process voice/text commands"""
    text = request.text.lower().strip()

    # 天気コマンド
    if any(word in text for word in ["天気", "気温", "weather"]):
        weather_data = await get_weather_data()
        return {"type": "weather", "data": weather_data}

    # YouTube検索
    if "youtube" in text or "ユーチューブ" in text:
        query = text.replace("youtube", "").replace("ユーチューブ", "").strip()
        search_url = f"https://www.youtube.com/results?search_query={quote(query)}"
        return {"action": "navigate", "url": search_url}

    # ニュース
    if "ニュース" in text or "news" in text:
        return {"action": "navigate", "url": "https://news.google.com"}

    return {"type": "unknown", "message": "コマンドを理解できませんでした"}
```

4. システム監視連携

システム監視ツールとの連携で、ネットワーク上のサーバーやサービスの死活監視を表示。

```
@app.get("/uptime_summary")
async def get_uptime_summary():
    """Get formatted uptime status"""
    if not MONITORING_URL:
        return {"enabled": False}

    try:
        response = requests.get(f"{MONITORING_URL}/api/status-page/{STATUS_SLUG}",
                                timeout=5)
        data = response.json()

        monitors = []
        for group in data.get("publicGroupList", []):
            for monitor in group.get("monitorList", []):
                monitors.append({
                    "name": monitor.get("name", "Unknown"),
                    "status": monitor.get("status", 0),
                    "ping": monitor.get("ping"),
                    "time": monitor.get("lastReportTime")
                })

        return {"enabled": True, "list": monitors}
    except Exception as e:
        return {"enabled": True, "error": str(e)}
```

5. レスポンシブタッチインターフェース

800×480の小画面に最適化したタッチ操作を実装。

```
// 改良されたタッチスワイプ処理
let startX = 0, startY = 0, moved = false;

function onTouchStart(e) {
    if (isTransitioning) return;
    const touch = e.touches[0];
    startX = touch.clientX;
    startY = touch.clientY;
    moved = false;
}

function onTouchMove(e) {
    if (isTransitioning) return;
```

```
        const touch = e.touches[0];
        const deltaX = touch.clientX - startX;
        const deltaY = touch.clientY - startY;

        // 横方向の動きが優勢な場合、縦スクロールを防ぐ
        if (Math.abs(deltaX) > Math.abs(deltaY) && Math.abs(deltaX) > 10) {
            e.preventDefault();
            moved = true;
        }
    }

    function onTouchEnd(e) {
        if (isTransitioning || !moved) return;
        const touch = e.changedTouches[0];
        const deltaX = touch.clientX - startX;

        if (Math.abs(deltaX) > 80) {
            if (deltaX < 0) {
                goPage(currentPage + 1); // 次のページ
            } else {
                goPage(currentPage - 1); // 前のページ
            }
        }
    }
```

技術的工夫

1. 環境変数による柔軟設定

すべての設定を環境変数で制御できるようにし、コードの再コンパイルなしで動作を調整可能にした。

```
# .env設定例
WIDTH = int(os.getenv("WIDTH", "1280"))
HEIGHT = int(os.getenv("HEIGHT", "720"))
WEATHER_API_KEY = os.getenv("WEATHER_API_KEY", "")
MONITORING_URL = os.getenv("MONITORING_URL", "")
CITY_NAME = os.getenv("CITY_NAME", "Tokyo")
```

2. オフライン優先アーキテクチャ

基本機能はすべてローカルで動作し、外部APIは補助的な役割のみ。ネットワークが切れても基本操作は継続可能。

3. 軽量実装

外部ライブラリへの依存を最小限に抑え、Raspberry Pi 5の性能を最大限活用。

```
// 外部ライブラリなしの純粋なJavaScript実装
const slider = document.getElementById('slider');
let currentPage = 0;

function goPage(pageNum) {
    const maxPage = 2;
    pageNum = Math.max(0, Math.min(maxPage, pageNum));
    currentPage = pageNum;

    slider.style.transform = `translateX(-${currentPage * 100}svw)`;
    updateIndicators();
}
```

MVP開発プロセス

最初から完璧なシステムを目指さず、段階的に機能を追加していった：

フェーズ1: 基本インフラ

- FastAPIサーバー構築
- 単一ページHTML表示
- CSIカメラストリーミング
- **3Dプリントフレーム設計・製作**

フェーズ2: UI改善

- 3ページスワイプインターフェース
- タッチ操作最適化
- レスポンシブデザイン
- **フレーム組み込みとケーブル整理**

フェーズ3: 機能拡張

- 天気API連携

- システム監視連携
- 音声コマンド処理
- オーバーレイナビゲーション

パフォーマンス

- **起動時間**: 約15秒
- **メモリ使用量**: アイドル時800MB、ピーク時1.5GB
- **応答時間**: ローカル処理で平均50ms以下
- **ストリーミング**: 720p @ 15fps安定動作

実際の使用感

良い点

- タッチ操作が直感的で家族も簡単に使える
- ローカル処理でレスポンスが速い
- カスタマイズの自由度が高い
- プライバシーが保護される
- **3Dプリントフレームで完全にカスタマイズされた外観**
- **コンパクトで置き場所を選ばないデザイン**

改善点

- 初期設定に技術知識が必要
- UIデザインをもっと洗練できる
- 音声認識精度の向上余地
- **3Dプリントの積層痕が目立つ部分がある**
- **フレーム重量でやや重くなった**

今後の展望

1. **ローカルLLM統合**: より自然な音声対話
2. **IoTデバイス連携**: スマートホーム制御

3. **データ可視化**: センサーデータのグラフ表示
4. **マルチデバイス同期**: 複数のRaspberry Pi連携

技術仕様

- **プラットフォーム**: Raspberry Pi OS (64-bit)
- **言語**: Python 3.11 + JavaScript ES2020
- **フレームワーク**: FastAPI, OpenCV
- **画面サイズ**: 800×480最適化
- **メモリ要件**: 最低4GB、推奨8GB

まとめ

Raspberry Pi 5の性能向上により、実用的なスマートホームハブを自作することが現実的になった。特に8GBメモリは複数のサービスを同時実行するのに十分な余裕がある。

既製品では実現できない、自分だけの機能やインターフェースを実現できるのが自作の魅力だ。MVPから始めて段階的に機能を追加していくアプローチで、無理なく実用的なシステムを構築できた。

このプロジェクトを通じて、Raspberry Piが単なるホビー用途から実用的な日常ツールに進化したことを実感している。技術的なハードルはあるが、その分満足度の高いシステムを構築できる。

同じようにRaspberry Piで実用的なものを作りたいと考えている方の参考になれば幸いである。