KneeKarePro Release Candidate

ACL Recovery Brace

KneeCarePro is a brace for ACL recovery patients. The patient's brace will be able to upload their rotary data for a clinician to review.

▼ Table of Contents

Overview

The KneeKarePro is a brace for ACL recovery patients. The brace combines hardware and software to collect a patient's rotary data. The data is then uploaded to a web app for a clinician to review.

Hardware

The hardware consists of a ESP32 microcontroller, a LiPo battery, and a potentimeter to measure the rotary data. The firmware on the microcontroller collects, interpolates, and sends notifications to a connected device over Bluetooth Low Energy (BLE). The firmware frameworks utilized are the Arduino framework and the PlatformIO meta-framework. We decided to use the ESP32 microcontroller because of its built-in BLE capabilities and its low power consumption. The decision to use BLE was made to ensure that the battery life of the device would be long enough to last a full day of use. However, there exists a discussion in shifting the purpose of the device to record and store the data locally on the device. This would allow the device to be used without the need for a connected device. In the event of a user establishing connection with the device the Bluetooth Low Energy communications protocol would no longer be sufficient. In order to improve the bit rate, our connecting device and peripheral device would need to use the Bluetooth Classic protocol.

Currently, we do not have any existing issues or bugs within the hardware or firmware. The hardware and firmware are both in the production stage, and can be easily flashed to a device. Testing firmware and hardware capabilities exist within python scripts which connect and poll data at differing rates. The testing suite includes cases of long wait times in attempt to create timeouts. However, the BLE protocol is resistant to timeout issues due to the nature of request and notifications.

Hardware Repositories

Software

Data Streamer

The **Data Streamer** is the cornerstone of our application and allows the user to establish a connection to the local hardware device and stream data points. The **Data Streamer** simultaneously attempt to reach a user's a backend endpoint, create a user if a user does not exist, and store user data on a persistent database using a user specific table. The currently implementation of the data streamer is a python applications using the `bleak` library to connect to the device and the `requests` library to send data to the backend. The **Data Streamer** is currently in the development stage and due to be migrated to another platform. As mentioned previously, there are existings discussion to reorganize the frequency of communications with the device. If the pivot occurs, the data streamer application will be migrated into a desktop application through the use of Electron. However, a large number of either existing components within the KneeKarePro project make use of the flexibility and speed of development Python offers. The decision to migrate to Electron would be made in the event of a need for a more robust application however, would likely incorporate Python.

Backend

The backend handles all user data traffic directing the request and analytics of user data. The backend is a RESTful API built using the FastAPI framework. The backend is currently in the development stage and is on going a migration from the Flask framework. Currently, our backend endpoints consist of basic CRUD operations for users data in addition to more complex analytical requests handled by `pandas` and `numpy`.

For example, there is a backend endpoint of `/data/stats/{username}` which returns the mean, median, and standard deviation of both angular and rotational speed data.

```
@app.route('/data/stats/<username>', methods=['GET'])
def get_data_stats(username):
    """
    Retrieve and return statistical data for a given user's device data.
    This function fetches the device data associated with the specified
username
    from the database, computes statistical metrics (mean, standard
deviation,
    minimum, and maximum) for the 'angle' and 'rotation' fields, and
returns
    these statistics in a JSON response.
        username (str): The username of the user whose data statistics are
to be fetched.
    Returns:
        Response: A JSON response containing the username and the computed
statistics
                  for 'angle' and 'rotation' fields if the user is found,
otherwise
                  an error message with a 404 status code.
    """
    user = User.query.filter_by(username=username).first()
    if not user:
        return jsonify({'error': 'User not found'}), 404

    data = pd.DataFrame([{
        'angle': d.angle,
        'rotation': d.rotation,
        'timestamp': d.timestamp
    } for d in user.device_data])

    stats = {
        'angle': {
            'mean': data['angle'].mean(),
            'std': data['angle'].std(),
            'min': data['angle'].min(),
            'max': data['angle'].max()
        },
        'rotation': {
            'mean': data['rotation'].mean(),
            'std': data['rotation'].std(),
            'min': data['rotation'].min(),
            'max': data['rotation'].max()
        }
    }

    return jsonify({'username': username, 'stats': stats}), 200
```

Future endpoints will entail more complex analytics such as the rate of change of the data and the rate of change of the rate of change of the data. The numerical manipulations for these endpoints will be handled by `numpy` and `pandas` libraries. Additionally, data persistance is handle through databasing using the `SQLAlchemy` ORM and `SQLite` database. The decision to use `SQLite` was made due to the ease of use and the ability to quickly prototype the backend. However, the decision to use `SQLite` was made with the understanding that the database would be migrated to a more robust database such as `PostgreSQL` or `MySQL` in the future. `SQLite` is not a production grade database and is not intended to be used in a production environment. The decision to migrate to a more robust database was made to ensure that the data would be secure and that the database would be able to handle the large amount of data that would be generated by the device. However, due to the client-server nature of more robust applications, the decision to migrate to a more robust database would be made in the event of a need for a more robust application.

Installation and Usage

To install the backend, clone the repository and navigate to the `KneeKareBackend` directory. The backend and other Python packages are managed using Poetry and can be installed using the following command:

```
poetry install
```

To run the backend, there is a script describe in the `pyproject.toml` file. The backend can be run using the following command:

```
poetry run start
```

To work with the development environment, the backend can be run using the following command:

```
poetry shell
```

Web Application

The web application is the front facing component of the KneeKarePro project. The web application is built using the React framework and is currently in the production stage. The web application is a single page application that allows users to view their data and analytics. The web application utilizese various data visualization libraries such as `chart.js`.

Features

The KneeKarePro system integrates the following key features:

- ✅ **Real-time Data Streaming**: The KneeKarePro system is capable of streaming real-time data from the hardware device to the user's connected device.
- ✅ **Data Analytics**: The KneeKarePro system is capable of performing data analytics on the user's data, such as computing statistical metrics and generating visualizations.
- ✅ **User Interface**: The KneeKarePro system provides a user-friendly interface for clinicians, patients, and other stakeholders to interact with the system.
- ✅ **Data Persistence**: The KneeKarePro system is capable of persisting user data to a database for future reference and analysis.
- ✅ **Adaptable Hardware**: The KneeKarePro system is designed to be adaptable to different hardware configurations and form factors.
- ✅ **Low-Power Consumption**: The KneeKarePro system is designed to be powered by a rechargeable battery for ease of use and portability.

Intended Impact

KneeKarePro addresses the need for accurate, continuous data in ACL rehabilitation, allowing clinicians to make data-driven decisions and helping patients stay engaged in their recovery. By providing immediate feedback and a comprehensive tracking solution, KneeKarePro aims to improve rehabilitation outcomes, shorten recovery times, and reduce re-injury risks. The device’s versatility makes it accessible for a wide range of rehabilitation settings and adaptable to various brace types, potentially expanding its use across different physical therapy applications.

Repositories

The following repositories are associated with the KneeKarePro project and are being actively developed:

- Firmware
- Hardware + STL
- Data Streamer

- Backend
- Web Application

The following repositories are associated with the KneeKarePro project however, are **not being actively developed**:

- Deprecated Backend

Instructions

Milestone Work

The following milestones have been completed for the KneeKarePro release candidate:

1. UI/UX Improvements

- Redesigned data cards and interactive charts for clearer data presentation and improved readability.
- Simplified navigation menus to enhance usability, with core functions more accessible from a centralized location.
- Added status indicators for BLE connectivity and battery level to provide users with real-time device status updates.

2. Backend Enhancements

- Implemented faster and more responsive FastAPI backend to handle user data more efficiently.
- Reorganized endpoint structure through the use of Router to improve code organization and scalability.
- Refactored data analytics functions to optimize performance and reduce processing time for large datasets. This was done through asynchronous processing and parallelization of data computations.

3. Device and Hardware Adjustments

- Enhanced BLE communication protocol to improve data transmission reliability and reduce data loss.
- Optimized power management features to extend battery life and reduce power consumption during data streaming.
- Improved the casing design, smoothing edges and reinforcing the Velcro attachment for secure and comfortable use.

4. Aesthetic Enhancements

- Updated frontend UI to resolve minor cosmetic issues, ensuring uniform icon sizes, color schemes, and font alignment.
- Finalized casing aesthetics to provide a professional and cohesive appearance in line with medical device standards.

Time Worked

Since Beta Presentation

Plan of Action (10/22) 12 Total team hours:

Here we discussed what it was we needed to polish up in time for the Release Candidate Submission. We believed that our systems worked as intended and our functionality was practically where we wanted it to be, therefore it was up to us to make the project better rather than fixing anything.

Begin Polishing (10/24) 10 Total team hours:

We all had midterms which it made it very difficult for us to work as much as we would have liked, however we worked over zoom to touch up some of the parts that needed some polishing. This included perfecting the BLE connection, beginning initial calculations/diagrams for making the device attachable, and making the user experience as seamless as possible.

More Polishing + Casing Development/Printing (10/29) 33 Total team hours:

This began a week of lots of hard work. We now had all of our parts working absolutely perfectly, now it was just a matter of continuing to make it better. This meant a lot of research. Between beginning the development of making the device an attachment and finding new ways to make the experience more seamless through the use of ap possible desktop app, and more features such as password security and could management for data; a lot of work was done.

Adding to Backend + More Casing Development (10/30) 15 Total team hours:

The brace was now beginning to look like an attachment, there was development of a battery pack as well and it should be a 3-step application either next week or the week after. The backend is now fully operational and up to standard at this point.

Meeting with Carsten + Documentation (10/31-11/01) 27 Total team hours:

Our meeting with Carsten allowed us to see where we are as a team and how our project is looking. We then worked on the documentation for the Release Candidate.

Known Bugs and Development Horizon

The KneeKarePro system is currently in the final stages of development, with the following known bugs and future development plans:

Known Bugs

1. **Intermittent Data Display Lag**

- **Description**: Occasional delays in data updating on the frontend when BLE signal strength is weak.
- **Impact**: Minimal; does not affect data accuracy but may cause slight delays in real-time display.
- **Workaround**: Ensuring close proximity between the device and the local device minimizes lag

2. **UI/UX Inconsistencies on Mobile Devices**

- **Description**: Minor layout inconsistencies observed on screens smaller than 5 inches, especially for interactive charts.
- **Impact**: Limited to users accessing the system on small displays; no effect on core functionality.
- **Workaround**: Accessing the UI on larger screens or devices with responsive capabilities resolves this issue.

Development Horizon

1. **Enhanced Data Visualization**

- **Objective**: Implement additional data visualization features, such as 3D motion tracking and interactive heatmaps, to provide more comprehensive insights.
- **Timeline**: 2-3 weeks

2. **User Authentication and Data Security**

- **Objective**: Integrate user authentication and data encryption features to enhance data security and user privacy. This will done through password hashing, JWT tokens, and OAuth2.
- **Timeline**: 1-2 weeks

3. **Device Data Persistence**

- **Objective**: Implement local data storage on the hardware device to allow data collection without a connected device. This will be done through the use of an SD card.
- **Timeline**: 1-2 weeks

4. **Bluetooth Classic Integration**

- **Objective**: Integrate Bluetooth Classic protocol to improve data transmission speed and reliability for connected devices.
- **Timeline**: 1-2 weeks

5. **Device RTC and Data Synchronization**

- **Objective**: Integrate a real-time clock (RTC) module to timestamp data points and synchronize data with the backend server.
- **Timeline**: 1-2 weeks