SmartHub BLE: Validation of a Data Acquisition, Visualization, and Analysis Tool for Manual Wheelchair Usage Characterization

Thesis

Presented in Partial Fulfillment of the Requirements for the Degree Master of Science in the Graduate School of The Ohio State University

By

Nathan Waltz, B.S.

Graduate Program in Mechanical Engineering

The Ohio State University

2025

Thesis Committee

Dr. Sandra Metzler, Advisor

Dr. Carmen DiGiovine

Abstract

Many manual wheelchair users experience upper extremity injuries, often caused by either sub-optimal biomechanical techniques during wheelchair propulsion or by improperly fit wheelchairs. The forces required to propel a wheelchair are very repetitive and the associated stress in the muscles and tendons need to be minimized to avoid these injuries. By monitoring key propulsion metrics such as stroke frequency, push force, stroke length, and net velocity, both users and clinicians can gain information that may help identify and address underlying issues. A prior device, the SmartWheel, was developed to collect such data; however, its high cost, limited availability, and outdated software led to a desire for the development of a successor, the SmartHub, by Ohio State capstone students and researchers. The work presented in this thesis introduces a number of validated modifications built on this original design to overcome previous functional limitations. This new model utilizes a microcontroller with an onboard inertial measurement unit and Bluetooth Low Energy (BLE) capability, allowing for real-time streaming of gyroscope data to a graphical user interface running on a host computer for visualization. The device utilizes two individual units, each mounted to one wheel hub on the wheelchair to collect gyroscope data. The system is calibrated via a prescribed data collection process before user data is collected. To protect user privacy, wheelchair user data is de-identified and stored in a cloud database. User operation data and derived

wheelchair metrics can be paired with clinical analysis of patient propulsion as well as patient feedback on the ease of use and comfortability in order to characterize and better understand the use of manual wheelchairs through these wheelchair propulsion metrics. Finally, all code and designs are made open-source under the GPL-3 license to facilitate future work in this area by other researchers and limit the loss of knowledge associated with the turnover rate in academia.

Acknowledgments

I would like to express my deepest gratitude to my colleagues, advisors, instructors, and friends for their support throughout my graduate studies.

To Dr. Sandra Metzler, thank you for your guidance and encouragement. Your advice has helped shape both this research and my academic growth. The research I've been able to perform with your leadership and direction has been an invaluable experience, and I appreciate your continued support.

To Dr. Carmen DiGiovine and Liz Gauen, I am grateful for your input and the clinical perspective you brought to this project. Your feedback and knowledge has been essential to its success.

Finally, I want to thank my friends and family for their support and encouragement throughout this journey. I couldn't have done it without them.

Vita

May 2020……Troy Christian High School

May 2024……B.S. Mechanical Engineering

The Ohio State University

August 2024 – May 2025……Graduate Teaching Assistant

Department of Mechanical Engineering

The Ohio State University

Fields of Study

Major Field: Mechanical Engineering

Table of Contents

List of Tables

List of Figures

Chapter 1: Introduction

1.1 Background

The National Household Travel Survey performed in 2022 by the United States Department of Transportation demonstrated that over 18.6 million Americans self-report a travel-limiting disability. Of these individuals, 9.4% — approximately 1.74 million people — rely on manual wheelchairs or scooters to maintain their mobility and access to transportation. [1] However, even highly skilled manual wheelchair users still face mobility issues. The typical manual wheelchair user only travels 1.7 km per day on average, with a tendency not to engage in long and fast bouts (defined by exceeding speeds of 1 m/s^2 or lasting longer 2 minutes). [2] One commonly reported factor for these continued mobility issues is the excessive strain placed upon the upper body during the propulsion of a manual wheelchair. Studies have shown that approximately 50 percent of manual wheelchair users develop upper body injuries, with shoulder pain being one of the most prevalent issues. Frequent diagnoses include rotator cuff impingement, bicep tendonitis, and carpal tunnel syndrome (CTS). [3], [4]

The increased stress on the upper body among manual wheelchair users is not unexpected, as wheelchair propulsion relies almost exclusively on the muscles of the upper body. Over the course of an hour of operation, a manual wheelchair user performs

approximately 1800 pushes, with each push requiring a peak force of between 800 and 1400 Newtons. [5] Additionally, a manual wheelchair user will typically perform approximately 15 weight-relief lifts or transfers per day, defined as a movement consisting of lifting the entirety of their body weight. [6] These repetitive, high-stress activities contribute significantly to the risk of upper body pain and injury. To mitigate or reduce these risks, optimizing various aspects of wheelchair usage is critical. [7]

The two primary methods of correcting wheelchair usage to improve propulsion include adjusting propulsion technique and modifying wheelchair parameters, such as the user's center of gravity relative to the wheel, the size of the wheels, and the weight of the wheelchair. [8], [9] However, because each user will have different physical capabilities and expectations for daily activities, it is important to address the outcomes of wheelchair fitting and propulsion technique correction. While users can express qualitative and subjective satisfaction with the wheelchair setup as a means of validation, there are quantitative wheelchair metrics that have demonstrated correlations with specific upper body injuries. A graphic showing various subsystems of a wheelchair is shown below, and each can be modified to suit a user's needs.

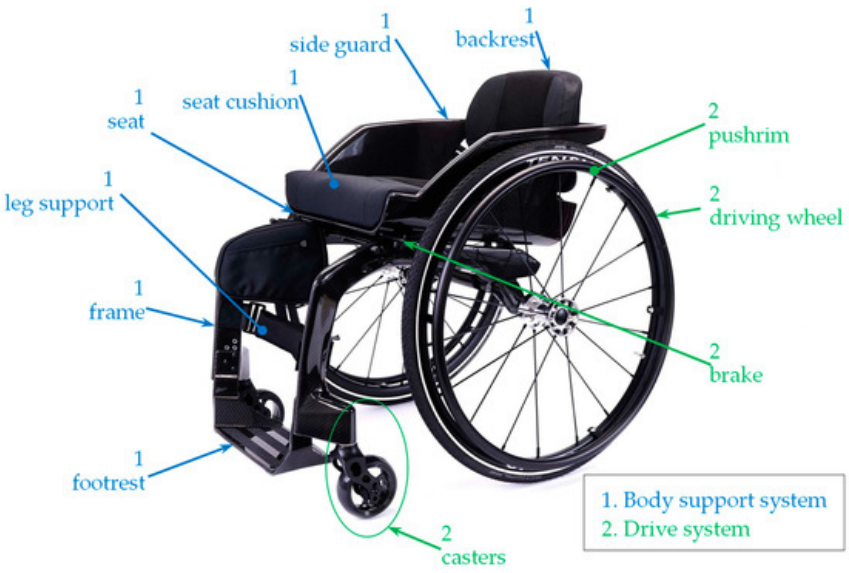

Figure 1: Wheelchair Subsystems [10]

As an example, one of the main contributing factors to CTS specifically is the regular cadence that the wheel is propelled at, referred to as the stroke frequency. Each push is required to have a similar force, regardless of the time taken in between strokes. [11] It has been proven that the repetitiveness of a task is a higher risk factor than the actual force applied in the task, and as a result, individuals with a higher stroke frequency have a higher risk for median nerve injury. [12], [13] Other key metrics of potential interest to clinicians, researchers, or users attempting to characterize manual wheelchair usage include velocity, stroke length, push force, and rolling resistance. [14]

To address the need for these quantitative metrics, several devices and means of data collection have been designed over the years. While there exist optical systems to track stroke patterns on stationary wheelchairs mounted on rollers and odometers to track distance over longer periods of time, there is a notable lack of devices to track these more

advanced metrics in the context of normal everyday wheelchair use. [11], [15] The most effective solution for an individual seeking to track kinetic and kinematic wheelchair statistics is the SmartWheel, which was later improved in a new device called the OptiPush Biofeedback System. [16], [17] Both of these systems function as a fully independent wheel with strain gauges to monitor the forces applied to the wheelchair, alongside other sensors such as rotary encoders to measure kinematic attributes of wheel propulsion. OptiPush demonstrated that by tracking push rim forces and stroke frequency and showing these values in real time to the user, these metrics could be optimized to levels that have been proven to decrease the likelihood of upper extremity injury. [18] However, while OptiPush remained confined to the research setting, SmartWheel, although once commercially available, was priced prohibitively high and has become outdated for modern manual wheelchair research.

A device was also developed by researchers at the University of Pittsburgh in an effort to track gyroscopic wheelchair data to obtain net distance and velocity statistics. [19] This device functioned by transmitting gyroscope data from one wheel over Bluetooth to a mobile phone, which processed all data and gave activity reports to a user. While this device was very accurate for wheelchair activity tracking, it is not available for purchase, and the software and specifications used to handle the data transmission and construction of such a device are not published.

Research has been conducted recently into tracking of wheelchair propulsion metrics by means of an Apple Watch or other fitness tracker worn on the wrist. [20], [21] While

this has been shown to have poor accuracy in determining overall strokes, leading to inaccurate stroke frequencies (20.61% error for Apple Watch Series 1 and 9.20% error for Apple Watch Series 4), it does appear to have the ability to track stroke types by detecting the angle of the wrist and position of the hand during a stroke. This has the potential to be used in conjunction with a more accurate device like the one presented in this paper for stroke categorization but does not appear to be able to be used as a standalone device. Even if improvements are made in the detection of stroke count, such a device would still be unable to calculate metrics such as rolling resistance.

As a result, there are currently no suitable and commercially available devices capable of providing quantitative analysis of manual wheelchair usage in an everyday context. While such devices have been developed and validated previously, the work that went into them cannot be easily built off of, since it has not been made publicly available.

1.2 SmartHub History

Beginning in 2015, a group of Ohio State capstone students began work on a new device to address the need for comprehensive wheelchair analytics in both the clinical and everyday environments. This device, called SmartHub, was significantly lower cost than SmartWheel and was designed as a standalone device mounted to a wheel, allowing for more portable applications. [22] The original SmartHub prototype captured general statistics such as distance travelled by using a reed switch, which tracked number of rotations and calculated stroke frequency by means of an accelerometer. This data was fed into an Arduino Pro Mini and stored on an SD card, which could be transferred to a

computer and processed in Matlab. This design was improved with work performed by Ryan Letcher in fulfillment of a master's thesis. [23] This new design, referred to as SmartHub I, tracked gyroscope values from an inertial measurement unit (IMU) linked with an Arduino UNO and relied on an SD card to transfer data to a central computer for processing, just like the initial prototype. Due to the use of only a single gyroscope, the unit also relied on the assumption that the user was travelling forward in a straight line.

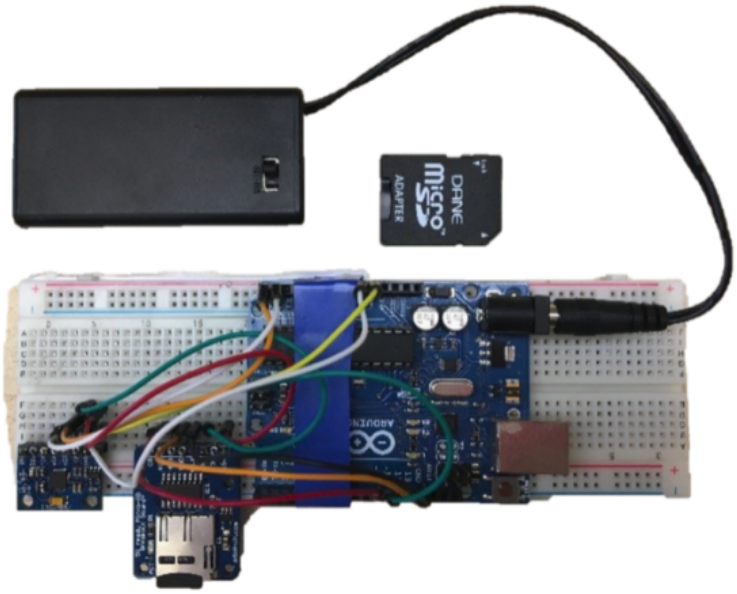

Figure 2: SmartHub I, developed by Ryan Letcher

This iteration of the SmartHub contained validated metrics including distance travelled, stroke length, stroke frequency, stroke angle, and tangential peak force. These values were compared against the metrics given by the SmartWheel and evaluated as a percentage difference. While the stroke angle and tangential force calculations were off by a significant factor, the stroke frequency was found to be very accurate with this methodology, with an error of about 1%.

To improve upon this design, master's student Noah Einstein upgraded the hardware to utilize a Raspberry Pi Zero W (RPi), a Linux-based microprocessor. This new device is referred to as SmartHub II. [24] There were several major improvements over SmartHub I in this new design. The 9-axis IMU used for this device also captured quaternion values, which provides more insight than gyroscope data. Quaternions allow for the capture of heading data through the magnetometer, which can be used to generate plots of the trajectory. The RPi has wireless capability, so it was programmed to broadcast a secure Wi-Fi network that allows a computer or phone to connect to it. Complex test runs were required to have a computer use Secure Shell (SSH) to access the RPi's command line, but a simple user interface allowing the user to start and stop test runs from a browser was designed, allowing for the collection of data from just a phone.

Figure 3: SmartHub II, developed by Noah Einstein

Through more complex test runs including 20-meter runs, 180 degree turns and figure-8 maneuvers, the error of important metrics including stroke frequency, distance, and tangential force were calculated. Stroke frequency was accurate to about 5% error, distance was accurate to approximately 3.3% error, while force was accurate to 16% error on a tile surface and 22% error on carpet. While the frequency and distance metrics are sufficiently accurate, the significant error in the force means that a standalone force value should not be used to draw a conclusion. However, it does generally characterize force, with greater applied forces resulting in higher calculated forces.

To improve the form factor of the SmartHub, undergraduate researcher Noah Kaplan redesigned the device to fit within the spokes of the wheels, while keeping the same software and sensors. [25]

Figure 4: SmartHub III, developed by Noah Kaplan

Following the development of SmartHub III, master's students Kai Vogeler and Morgan Strauss intended to use the device to further validate the device in a clinical setting and implement more complicated research protocol. [26] However, the project stalled due to several reasons. First, the code for SmartHub II had incomplete documentation, creating a knowledge gap between the original and subsequent teams. This exacerbated a second issue, which was a failure to accurately read the IMU quaternion values since the original design process was not fully understood. An in-depth code review and hardware evaluation was conducted, and it was determined that a substantial software and hardware redesign would need to be conducted for the SmartHub to continue in its current format. As a result, a comprehensive design review of the SmartHub was conducted, and new requirements were established to determine the most suitable form for the SmartHub to take moving forward. No devices previously produced offer a complete set of features desired by researchers and clinicians today, as summarized below in Table 1.

Table 1: Wheelchair Activity Monitor Comparison

Name	**Cost**	**Year**	**Availability**	**Other Notes**
SmartWheel	$12000	2009	No longer sold	No current support for device, operates on outdated software, all hardware/ software is proprietary
OptiPush	Unknown	2006	No longer sold	No design specifications released

G-WRM	N/A	2013	Not released	Only used at Pittsburgh University, no specifications released
SmartHub (capstone)	$140	2015	Available to Ohio State researchers	Manual transfer from SD card needed to computer
SmartHub I	$69.47	2017	Available to Ohio State researchers	Manual transfer from SD card needed to computer
SmartHub II	$100	2019	Available to Ohio State researchers	Quaternion inaccuracies determined, SSH and UNIX familiarity required
SmartHub III	$100	2020	Available to Ohio State researchers	Quaternion inaccuracies determined, SSH and UNIX familiarity required
Apple Watch Series 1, 4	$70	2018	Available	Stroke propulsion metric inaccuracies determined, but useful in stroke categorization

Chapter 2: SmartHub Initial Development

2.1 SmartHub III Design Review and Limitations

The SmartHub III relied on the use of a Raspberry Pi Zero W alongside a BNO055 IMU. The IMU fed raw quaternion data into the RPi, which then extracted the desired metrics.

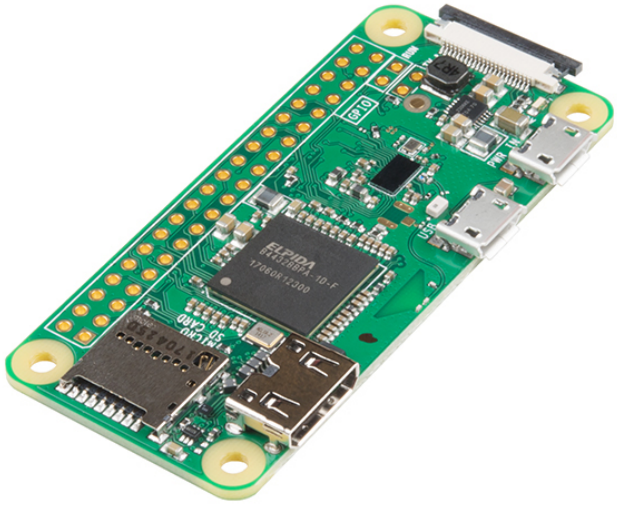

Figure 5: Raspberry Pi Zero W [27]

The original design change in SmartHub II from using gyroscope data to quaternion data was spurred by the ability of quaternions to encode the absolute rotation of the SmartHub III in space, allowing for the capture of trajectory, heading, distances

and velocities derived from the center of the wheelchair rather than the wheel that the SmartHub III is affixed to. The absolute rotation of the SmartHub III is derived from the use of an accelerometer, magnetometer, and gyroscope within the IMU, and the quaternion format encodes it through a unit vector, indicating the axis of rotation, and a rotation angle about this axis. A diagram of the quaternion coordinate system is shown below.

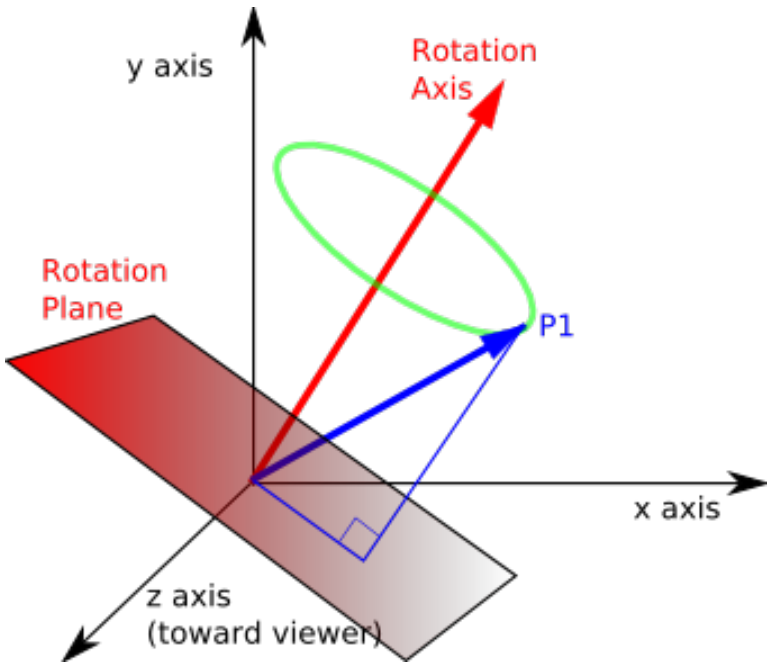

Figure 6: Quaternion Coordinate System [28]

The RPi and IMU were accompanied by a battery, a PowerBoost 1000 for battery charging and voltage conversion, and an SD card for storing files and the operating system.

To perform a test run, a user would need to be able to access the command line of the Raspberry Pi. The most effective way to accomplish this is to connect via SSH over

an access point, which is a Wi-Fi signal emitted by the RPi. SSH is not natively provided on Windows, so if the user does not have a MacOS or Linux distribution, the software PuTTY needs to be installed to access the RPi command line. Once connected to the RPi, Python scripts can be run to perform a test run. Basic familiarity with a UNIX terminal is beneficial to understanding how to run these scripts. The screen capture below demonstrates the required commands to perform a test run.

Figure 7: Data Logging Script through SSH

After the completion of a test run, an analysis and visualization script must be run in order to turn the data captured into usable metrics. Graphs such as a 2D trajectory path, heading (the angle of the SmartHub III over time), and displacement over time would be created and stored as HTML files. In order to view and use this data, it would need to be transferred to the host computer, which required the use of a software called

WinSCP. WinSCP is a free file manager and transfer application built for Windows, and it supports numerous file transfer protocols. There are alternatives available for MacOS systems, but since all testing of SmartHub II and III was performed on Windows, none were explored.

Overall, there were several identified shortcomings of the SmartHub in its current state. Namely, the learning curve needed in order to input commands to the Raspberry Pi was high, there were several additional software applications that needed to be installed, and data was only able to be viewed at the end of test runs, so no validation could be performed during a test run. Furthermore, there were several issues when obtaining the quaternion values from the IMU. Between the development of SmartHub III and the work performed by master's students Kai Vogeler and Morgan Strauss, significant errors were identified in the quaternion data, to the point where all data returned from all IMUs in the lab was essentially worthless. Tests were performed to ensure that it was not the handling of the data returned, and the issue was isolated to the IMUs. It was unclear what may have caused this issue, but one potential factor that was not explored at length was the proximity of the magnets to the IMUs, which could have potentially damaged the magnetometers. Magnetic fields may be encountered in the course of wheelchair propulsion, which may alter quaternion readings and mandate recalibration. Through the troubleshooting of this issue, several other issues in the way the data was handled in the code were identified, indicating that even if the issue with the IMUs was resolved, the SmartHub still would require a significant troubleshooting effort.

2.2 SmartHub Redesign Proposal

Since it was evident to the SmartHub III researchers that significant work still needed to be performed to make the device functional again, and some amount of redesign to the device would need to be performed, features that would be useful in a successor to the SmartHub III were identified and integrated. [29]

One new desired feature is the ability to view data in real time. This requires transferring data to the host computer at sufficiently high rates and providing an interface for the user to view this data. Alongside this, this new proposed SmartHub would avoid the use of a UNIX terminal to run commands. From the same interface described to view the data, the user would be able to control the data collection through the use of simple and intuitive buttons. Furthermore, this interface could be linked to a cloud database to avoid any tedious file transfer and management issues. Through this cloud database, test runs could be available across devices, assuming the user was properly authenticated.

In terms of specific improvements to features on the older design, the new design would ideally have a smaller form factor and be able to fit within the spokes of the wheels. No specific desired data acquisition rates were documented by the previous researchers, but their acquired data shows that data was read in at a rate of approximately 40 Hz. The goal for the new proposed SmartHub was to read data in at a rate of 50 Hz and focus on improving data acquisition speeds as much as reasonably possible to eliminate error caused by too low of a read rate. Finally, this new system would not use

magnetometers, as this can introduce unwanted error when traversing changing magnetic fields, and as demonstrated by the previous device, irreparable damage could be caused by unknown factors. A more stable and resilient method of capturing data would be needed for this new proposed SmartHub, which necessitated a transition away from using quaternions to establish heading and trajectory. However, feedback from clinicians indicated that providing heading and trajectory data, while not directly applied during a wheelchair fitting, would still be useful as an informal data validation tool in the sense that a user could visually identify if the path shown aligned with the performed test run. Also, when presented in graph format, heading and trajectory would be data points of interest not provided by any other tool, which indicates potential to drive user adoption and engagement. Widespread adoption of the proposed SmartHub would allow for more general datasets of wheelchair usage in an everyday environment, so retaining the capability of providing heading and trajectory graphs became a requirement for the new iteration.

2.3 SmartHub BLE Hardware Design

The initial feature designed for the new SmartHub was its wireless connectivity. While Wi-Fi compatibility could be kept in the new design, it was much too bulky for the application of the SmartHub. If data was to be immediately transferred to the host computer without any preprocessing, the throughput of the wireless connection would be very small comparatively to what a Wi-Fi is capable of handling. If data were to be taken at a rate of 100 Hz with 4 bytes of information per data point, only 400 B/s needed to be

transferred. The amount of power used by various wireless connection protocols varies widely by application and implementation, but similar use cases to the SmartHub indicate that the usage of Bluetooth Low Energy (BLE) can reduce the energy consumed above the baseline energy by approximately 50% versus the usage of Wi-Fi. [30] Furthermore, modern computers only allow for a connection with one access point, requiring the user lose Internet connectivity if a host computer was to connect to the SmartHub instead. For these reasons, the wireless connection method was chosen to be BLE. With the fundamental change in the connection process and the changes made in the data acquisition and analysis of the SmartHub, the new name for this device was chosen to be SmartHub BLE.

While the initial SmartHub BLE setup was performed with an Adafruit Feather nRF52840 Express with an Adafruit BNO085 IMU, this microcontroller was later switched to an Arduino Nano 33 BLE, since the Arduino has an onboard IMU. The same method of power regulation was used as the SmartHub III, with a PowerBoost 1000 C and a lithium-ion battery. A 3.7V 1200mAh battery was chosen, which used the PowerBoost to convert the voltage to 5 volts for the Arduino.

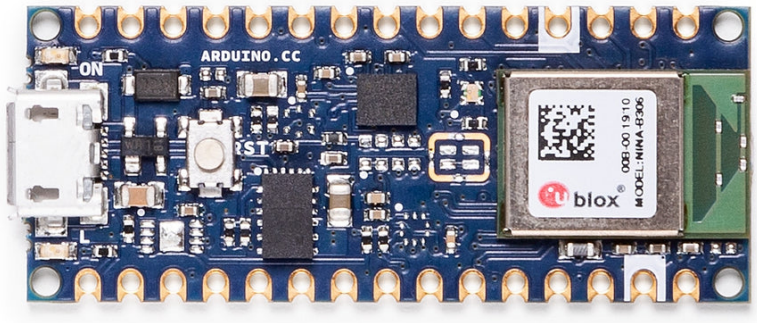

Figure 8: Arduino Nano 33 BLE [31]

The next feature incorporated into the SmartHub was the ability to gather metrics and accurately position the wheelchair in absolute space, which required the collection of raw sensor data for processing. Since a magnetometer was no longer an option for data collection, a gyroscope was used for the collection of rotational data. A gyroscope on a wheel can only collect rotational data about that wheel, but tracking rotational information about both wheels simultaneously allows for the positioning of the wheelchair on top of distance and velocity measurements for each wheel. As a result, the new design would use one Arduino, PowerBoost, and battery for each wheel and relay the axial gyroscope data to the computer over BLE. Acceleration in the radial direction is also gathered, as this data can in theory be used in the future for validation purposes.

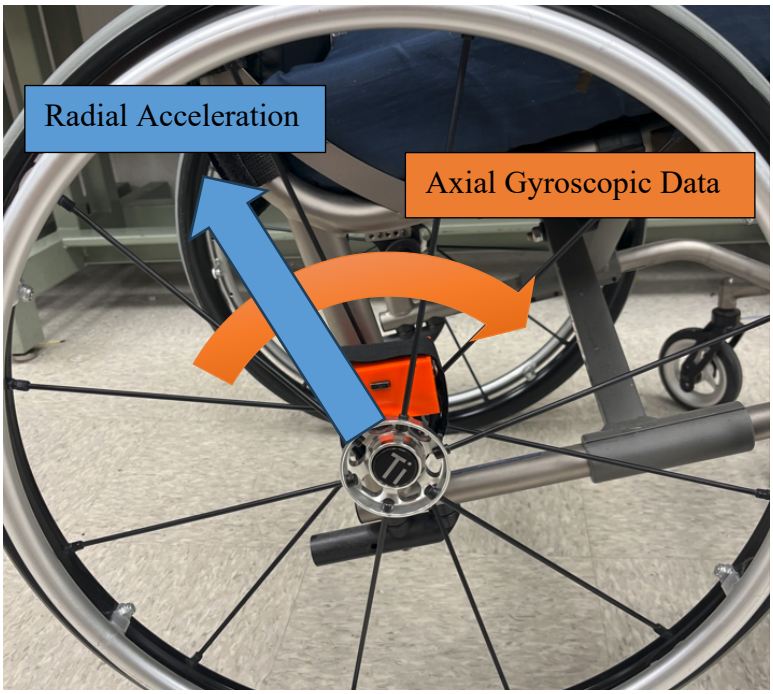

Figure 9: Gyroscope and Acceleration Directions

As stated earlier, one of the design goals of this new device was to maximize the data acquisition rate of the gyroscopes. The gyroscopes return data as an angular velocity, or radians per second. To convert these values into angles, which can then be converted to distances, the angular velocity must be integrated with respect to time.

$$\int_0^t \omega * dt = \theta$$

Equation 1: Rotation Calculation for Wheel

The error in the angle is directly proportional to the size of the dt value in the rotation equation, so dt must be minimized. The initial implementation and subsequent tests of BLE on the Arduino indicated that the maximum packet size was 20 bytes with a

maximum transfer rate of 17 packets per second. To maximize the amount of information in a packet to send to the central computer for processing, a custom packet structure was created. All accelerometer values were multiplied by 1000, and gyroscope values were multiplied by 100, with the sign bit handled separately in a byte at the start of the packet. As a result, acceleration and rotational velocity could be stored in 2 bytes each with an accuracy of 0.001 m/s^2 and 0.01 rad/s, respectively. In each packet, the four most recent values for acceleration and rotational velocity are gathered and grouped. The resulting custom packet structure is as shown below:

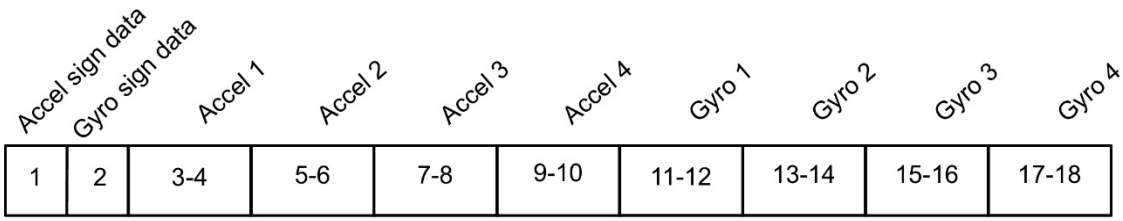

Figure 10: Custom Packet Structure for BLE Data Transfer

After the packet structure was generated, it was posted to the specific BLE characteristic used for data transfer and received by the host computer using the *.read_gatt_char()* method in the *bleak* library. [32] This method polls the characteristic as frequently as possible to retrieve the most recent data, and since the maximum polling rate was shown to be a consistent 17 Hz, the Arduino code was modified to retrieve values in $17 * 4 = 68\ Hz$ intervals. The code on the central computer took each packet and appended it to the dataset with the most recent data assumed to be at the timestamp of when the data was read in, and each prior datapoint at 1/68 second increments before.

2.4 SmartHub BLE User Interface

One of the primary features of the SmartHub BLE was an intuitive user interface that allowed for the collection and viewing of data without a significant learning curve or training. While JavaScript has libraries supporting the use of BLE and has more tools available for the creation of a user interface, the lab researchers were not as familiar with it as other languages. To make the transfer of knowledge easier, Python was chosen as the language for the user interface. Within Python, there were two choices for building the user interface. The first was *PyQT*, which is a wrapper for the low-level *QT5* framework, written in C++. The second is *tkinter*, which interfaces with the Tcl/Tk toolkit. While *PyQT* is a more efficient abstraction and can run more quickly and smoothly than *tkinter*, it has several drawbacks. First, *PyQT* requires significantly more knowledge to create customizable user applications, while *tkinter* has a low learning curve and can be used for extremely rapid prototyping of user-generated interfaces. Also, *PyQT* is distributed under the GNU General Public License (GPL), a license requiring that all derivative works and deployments of *PyQT* remain free and open-source software. [33], [34] At this time, the scope and future implementation of the SmartHub BLE was still uncertain, so *tkinter* was chosen to be the library to generate the user interface. *Tkinter* is a standard Python package, meaning that it is distributed under the Python license, which is a relatively unrestrictive license and allows for many styles of deployment. [35]

One advantage that *PyQT* has over the base *tkinter* library is the general appearance of the interface, which looks more modern. However, with the addition of the *ttk* module, which stands for themed tkinter, more customized interfaces can be created. [36] For the SmartHub BLE user interface, the theme *forest-dark*, generated by GitHub user *rdbende*, was integrated into the interface. [37]

For deployment of the interface, the Python library *pyinstaller* was used alongside the *onefile* tag. [38] *Pyinstaller* combines the Python interpreter, all dependencies, and the original scripts used into a single package, which can be deployed on any user interface. When the *onefile* tag is used during compilation, this single package is condensed to a single executable file. Using the *onefile* tag means all the dependencies get extracted to a temporary directory each time the executable is run, which slightly sacrifices performance but increases portability of the package.

2.4.1 Cloud Database

Another feature of SmartHub BLE was the addition of a cloud database for data storage. Rather than having files stored locally on each machine, any test runs captured through the interface would be stored in the selected database for retrieval by any clinician or researcher interested in this information. In the interest of data security and privacy, all patient data is de-identified with a 6-digit user ID. The database chosen for this purpose was MongoDB, a NoSQL database which offers free plans for <512 MB usage and has extensive documentation. A SQL database could have been chosen for this purpose and would offer higher lookup speeds when filtering by values internal to a

single test run, but the data up to this point had all been stored in Python in “dictionary” format. A “document” in a NoSQL database is an object intended to be retrieved and stored as a single unit, and in this case, it would refer to a single test run and all the data associated with it. NoSQL documents follow key-value format, which is the same type as a Python “dictionary,” so a NoSQL database was chosen since no data conversion between formats would need to be done.

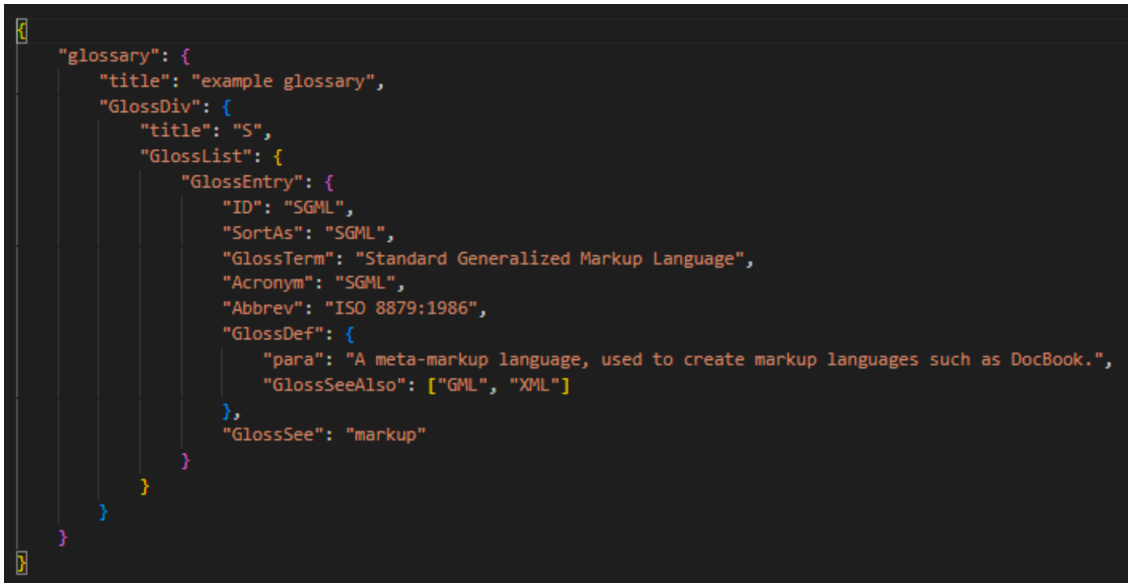

Figure 11: Key-Value Example [39]

To access the database securely, the executable contains the username and password, which means that possession of the executable gave any user full access to the database. This is inherently insecure but was permissible for this early stage of the interface, since the executable was not distributed beyond the researchers.

2.4.2 SmartHub BLE Data Acquisition

The initial screen presented upon loading of the executable showed two buttons, one titled “record data” and another called “view data.” These represented the two primary functionalities of the user interface, with the intended goal of achieving an intuitive and streamlined process for data acquisition and analysis. When the “record data” button was pushed, the user would be taken to a page prompting them to input the 6-digit user ID, and a button titled “Connect BLE Devices” would be clicked to initiate the connection protocol with the BLE devices, and after connection, a button titled “Start Logging” could be pushed to start the test run. Once the test run began, the screen would update in real time with the plots of displacement versus time, heading versus time, and trajectory. A button also appeared in the lower right side of the screen allowing the user to stop the test run. When the button to stop the recording is pushed, the user is returned to the home screen, at which point they can press “view data” to view any test run or “record data” to record another run.

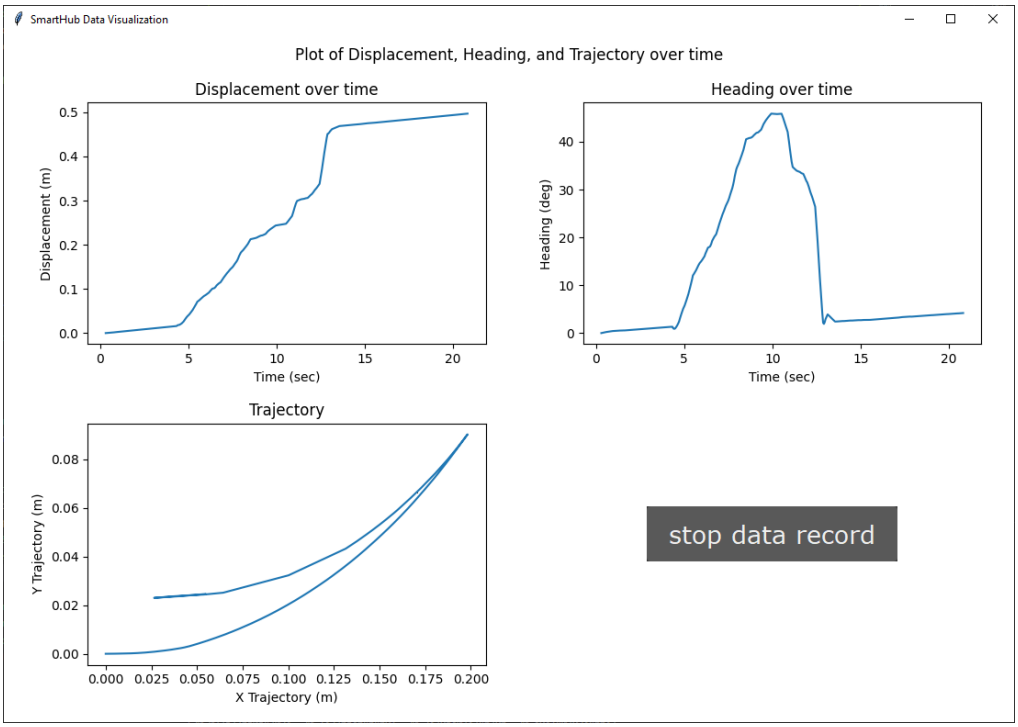

Figure 12: SmartHub BLE Record Data GUI

The graphs were generated by using the *FigureCanvasTkAgg* method within the *matplotlib* library. [40] To ensure the computing required to generate and update graphs did not interfere with the data read in through the *bleak* library, the *multiprocessing* library was used to create 2 separate CPU processes, one for the raw data acquisition and one for the data processing, analysis, and visualization. [41] The actual time to read the registers through the *.read_gatt_char()* method was established to be 1/17 seconds, but any other slowdowns in the code would add to the amount of time between data points being read in. To avoid this, the code for reading in the data was sent to its own CPU process, and all other tasks would remain in the original process. Multiprocessing in Python can use shared memory, but this requires careful memory management in order to

avoid a race condition, which is when two processes are attempting to access and modify memory at the same time.

To achieve the transfer of data between processes and allow for communication between the two, two "queues" were created. One informed the data acquisition process when to connect to the SmartHub BLE units and when to send data, based on the inputs to the user interface. The other was responsible for putting all data into the queue to be retrieved by the user interface process. Upon completion of the test run, the first queue was used to end the data acquisition process, and all data was stored in the database.

2.4.3 SmartHub BLE Data Retrieval

The second feature of the user interface is the ability to retrieve test runs from the database and view the data. After selecting the "view data" button from the home screen, the user is prompted to enter the 6-digit user ID submitted when the test run was recorded, and then a drop-down menu containing all the test runs associated with that ID allows the user to select the desired test run. The screen then changes to show graphs of displacement versus time, heading versus time, and trajectory, with the specific graph shown able to be modified by a drop-down menu at the title of the graph. The operator ID is shown at the top of the screen, and an arrow key is available to change between screens. In the bottom right side of the interface, metadata associated with the test run can be entered or modified including the test run name, the clinician ID, the location ID, and a text box for additional notes. This data is all saved in the database whenever the save button is pushed.

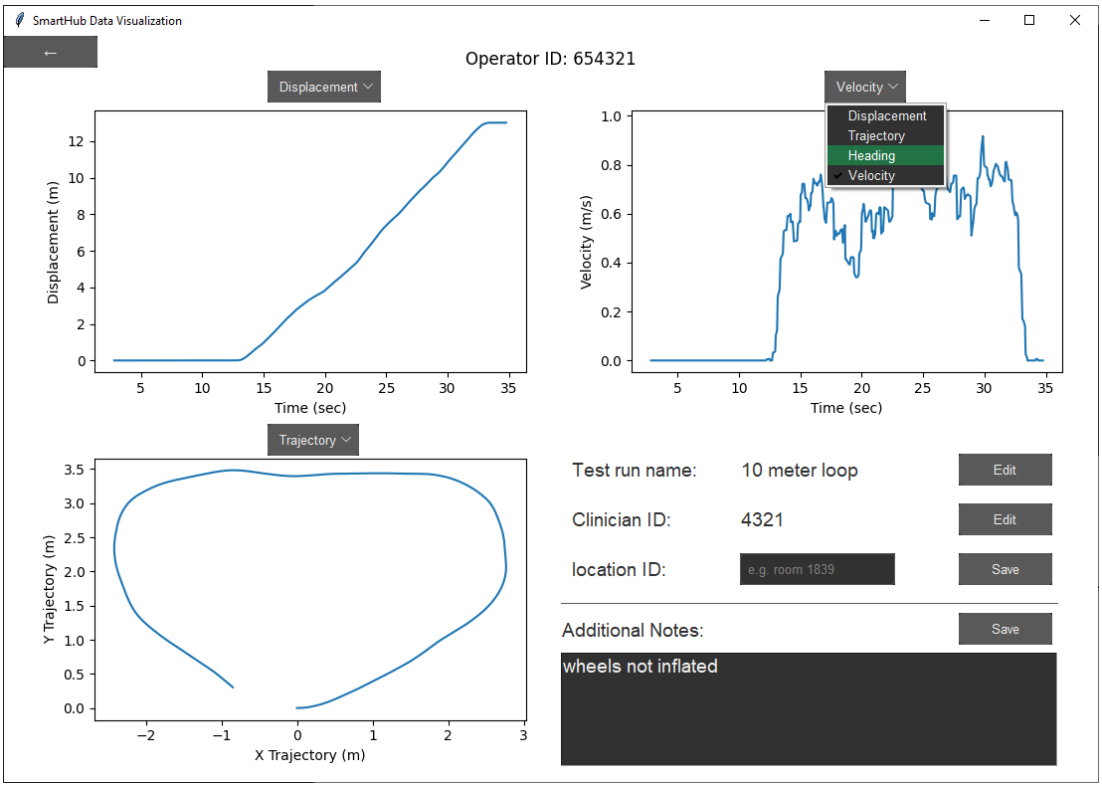

Figure 13: SmartHub BLE Data Visualization

2.5 SmartHub BLE Results and Feedback

Feedback from all clinicians interviewed and all researchers was overwhelmingly positive as an improvement from the SmartHub III. This user interface was used by team member Strauss to aid his collection of data and wheelchair metrics validation in completion of a master's thesis, establishing a strong foundation for taking data from current wheelchair users to more effectively understand wheelchair usage patterns. [42] Feedback from clinicians and discussions with other researchers highlighted areas of improvement to the interface, so the newest iteration of the SmartHub incorporated this feedback to be more useful in both the clinical and lab environments.

Chapter 3: Methodology

3.1 SmartHub BLE GUI Redesign

Based on the feedback from the clinicians interviewed and further research into intelligent user experience practices, one of the main objectives when redesigning the GUI was to minimize the number of user actions relative to the amount of information available and the amount of data input, while still maintaining intuitive navigation routes and eliminating screen clutter. This would serve to create a more natural interaction experience with less guidance from a more knowledgeable user. To achieve this, buttons should be clearly and concisely labeled, and data must be presented in a structured and organized format to reduce reliance on a user manual. The interface should ideally allow users with only a general idea of its capabilities the ability to collect and view data with little to no learning curve.

The initial change to the user interface for ease of use and simplicity was a transition to a "notebook" style of navigation resembling the functionality of a web browser, which is an application many users should be familiar with. Rather than having a home screen with buttons to access different capabilities of the interface, a notebook style instead uses tabs at the top of the screen to access the various features. Any actions the user takes in any tab persist as they switch between tabs, a feature that is given by the

tkinter handler, and a tab can be changed simply by clicking the desired tab at the top of the screen. Furthermore, with this style, visual resources and widgets (a term describing any GUI object) can be loaded in and rendered upon startup, reducing load times when attempting to change screens.

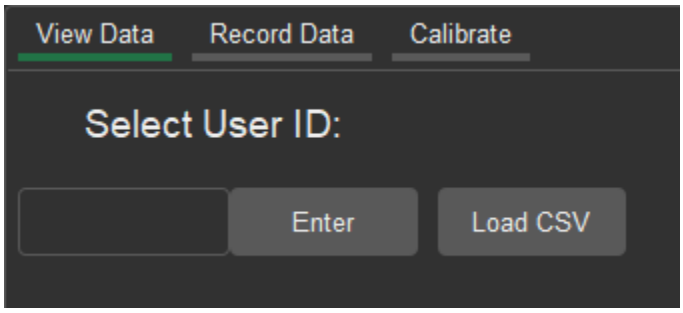

Figure 14: Tab Display in GUI

Each tab is intended to be able to fit all necessary buttons and data presentation on the same window, to eliminate navigation between screens. Navigating through screens will take information off of the screen, and the user will have to understand the interface navigation routes to know how to access or input this information again.

3.2 Database Authentication

Since the previous version of the SmartHub was not used in a clinical environment, the credentials to access the database were hard coded into the source code for the executable. This obfuscated all credentials, but any individual with access to the executable had access to the database. While patient details were still de-identified, it was desirable to have more robust security around the data to prevent bad actors from interfering with the research. All data was previously end-to-end encrypted through the

initiation of a TLS session with the database. TLS, which stands for Transport Layer Security, is the current standard way to authenticate users in network applications, and allows either the client or the server the ability to verify that their data was not tampered with. [43] However, leaving the credentials locally on the computer allows anyone who could access them to impersonate an administrator of the system. This could be avoided by only whitelisting specific IP addresses, but that requires manual input of new IP addresses which is a time-consuming process. To address this, the method of authentication was changed to require manual input of a username and password. As a result, the only way to have a data breach is by unauthorized sharing of the username and password combination.

To avoid needing to manually input the username and password for each instance of the executable, which can be rather time consuming when the application is used frequently, a X.509 certificate can be issued which can automatically authenticate the user and maintain a TLS connection. To authenticate with this, the executable was coded so that the file (with extension .pem) has to be placed in the same folder as the executable. As before with hard coding the username and password into the executable, the primary issue is the ability to transfer this certificate off of the host computer. As a result, this method is only used when it is determined that the computer is secure and not subject to unauthorized access. Otherwise, a user is responsible for inputting a username and password. This certificate can be burned (meaning removed from the list of valid users) and reissued at regular intervals to the users of the system, allowing for added confidence that only certain users have access to the database. Burning and reissuing the

previous version of the SmartHub's credentials is significantly more tedious, since it requires manually updating the source code, recompilation of the executable, and redistribution and redownload of a >80-megabyte file. Redistribution of the certificate is much easier, since it is 5 kilobytes and can easily be generated from the MongoDB website.

3.3 View Data Tab

The purpose of the view data tab is to retrieve data from the MongoDB database for viewing graphical data, recording or viewing notes associated with the test run, and downloading data and metrics into a .csv file for local access. While the user can upload csv files locally for viewing through this tab, the interface must still have internet access so that the database can be accessed. A majority of the features in the GUI rely on database access, and all data recorded for the purpose of this study needs to be stored in the database so that researchers have direct access. As such, the application does not load the main functionalities of the user interface if a connection to the database is not made.

To view a specific test run, the user must know the 6-digit code given when deidentifying the patient's data. After pressing the "Enter" button on the screen or the Enter key on the keyboard, a dropdown menu containing the names of the test runs will appear below the input box. Selecting a test run will cause four graphs to populate in the right side of the screen.

Table 2: Graph Labels and Units

Graph Position:	Y-Axis	Y-Axis Unit	X-Axis	X-Axis Unit
Top Left	Displacement	Meters	Time	Seconds
Top Right	Trajectory (Y)	Meters	Trajectory (X)	Meters
Bottom Left	Heading	Degrees	Time	Seconds
Bottom Right	Velocity	Meters/Second	Time	Seconds

Metadata associated with the test run will appear along the right side of the screen. Pressing either the Enter key on the keyboard or the "Save Metadata" button will cause the values input into the metadata entry boxes to be saved to the database. Underneath the metadata entry boxes, there are two buttons for either downloading a report file containing the metrics calculated by team member Strauss or to just download the raw gyroscope data. Both download buttons will attempt to download a .csv file and will prompt the user for what file location it should be downloaded to. Another button exists titled "Set Record Background." As data is recorded in the Record Data tab, it will be updated in real time with the same graphs shown in this tab on the screen. Selecting this button will use the test run currently being displayed in the View Data tab as a background to the graphs being updated in real time so data can be compared. Velocity is often compared between test runs, so this allows for the visual comparison of velocity data in real time. Multiple test runs can be set as the background in the Record Data tab for comparison if the user clicks the "Set Record Background" button on each of the desired test runs.

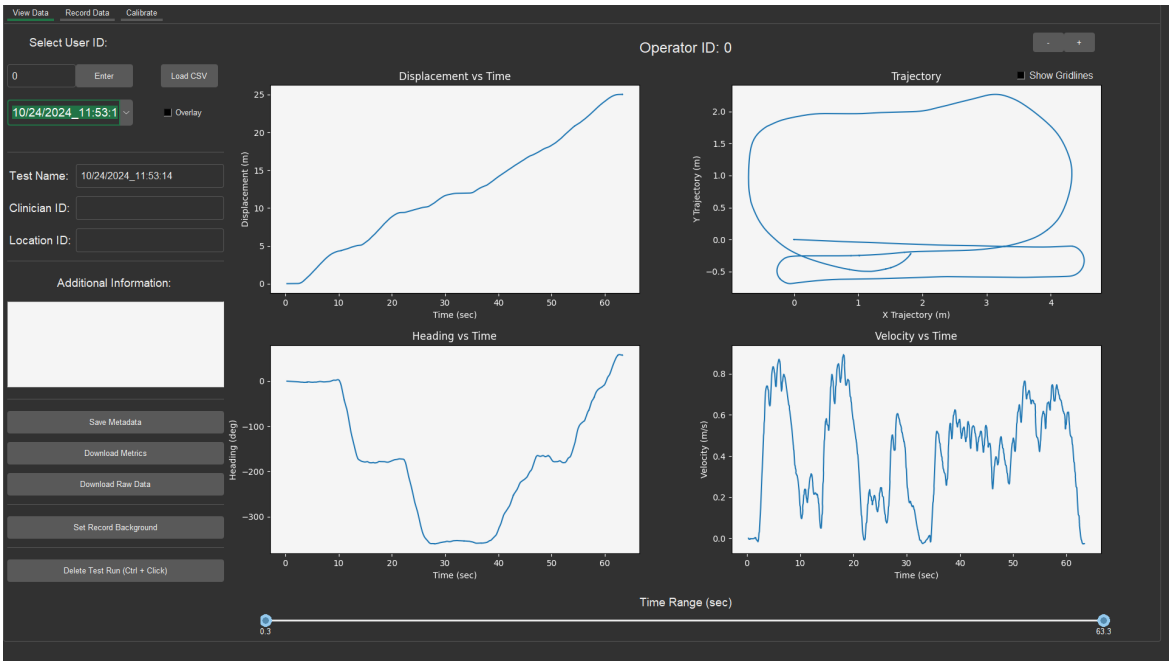

Figure 15: Screen Capture of Example Test Run in View Data Tab

A checkbox titled “overlay” is placed next to the drop-down menu, and selecting this box will cause any new test runs selected to be overlaid on the graphs. The purpose of the overlay is to also compare data visually on the screen, rather than needing to compare each test one at a time. A checkbox placed directly above the trajectory graph allows for gridlines to be shown on the trajectory graph. All graphs are automatically scaled to fit with the maximum amount of coverage on each axis; however, the trajectory graph is normalized so that the graph tick spacing for each axis is equal which provides a true representation of what a bird’s eye view of the trajectory would present. Finally, a sliding scale on the bottom of the graphs allows for a custom data range to be shown. Both the left and right sliders can be dragged, which will change the data range shown on the graphs in real time as they are dragged. The data is also be scaled on each axis, so it

will appear to zoom in on the part of the test run the user is looking to analyze. The primary purpose of this sliding scale is so that when longer test runs have been performed, a specific portion of the test run can be visually analyzed. The metrics download script already sorts longer test runs by "bout," which is considered a confined continuous segment of wheelchair movement.

Many of the features included in the View Data tab portion of the user interface were either developed by request of clinicians after presentation of the previous iteration of the SmartHub BLE or through analysis of user needs through discussion of the proposed applied environment of this device. As this device continues to be implemented in the clinical setting, care must be taken to ensure the data presented and the features available align with the needs of clinicians.

3.4 Record Data Tab

The purpose of the Record Data Tab in the user interface is to allow the user to perform test runs and to save these test runs to the database. While this capability was developed in the previous design of the SmartHub BLE, it had several key issues. There was minimal error handling in the previous design, meaning that if any actions taken by the user were incorrect or if there were any hardware malfunctions, the application would crash. Feedback would also not be given to the user about what went wrong, rendering it difficult to troubleshoot the error. If the Bluetooth Low Energy connection was not established properly on the first attempt or if the SmartHub units disconnected at any point for any reason, the screen would freeze and need to be restarted. To address these

issues, the recording capability of the user interface was redesigned in the notebook format. One major consideration when designing this new tab was that if any errors occurred that could reasonably be foreseen with the expected use of the tab, the application should inform the user of the error presented and give an opportunity for corrective action. For instance, if the SmartHub units failed to connect on the first attempt, the user should be able to retry that action instead of the application crashing.

3.4.1 Recording Via Threading

One of the key changes in order to achieve this goal of error handling in the new interface for recording data was the switch away from multiprocessing and into threading. The advantage of multiprocessing, and the reason multiprocessing was used in the original design, was because separating the computing power for the rendering of the graphs in real time and the data calculations would guarantee that data was not lost in transfer. Also, having the metrics calculations being performed in a different process than the graph rendering meant that the graphs could be updated more frequently. However, as the user interface gained more features and more information needed to be communicated between the processes, the logic to coordinate the processes became excessively complex to maintain. Variables between processes cannot be shared, so any information that needed to be communicated had to travel through a "pipe," as explained in Section X.X. The pipe method was successful when the only information that needed to be transferred was the raw data and "start" and "end" commands. Unfortunately, to effectively maintain the BLE connection, cleanly kill processes when unexpected errors

arise, and let user interface actions be performed in unintended orders, the conditional logic and the maintenance of variables on both sides of the pipes became difficult to account for when features were being added to the GUI.

To allow variables to be shared across processes and significantly reduce the burden of software logic associated with updating features of the Record Data tab, the choice was made to switch to a threading approach. Threading in Python is limited by the Global Interpreter Lock (GIL), which mandates thread-safe behavior. [44] Any resources or variables can only be manipulated by a thread which has control of the GIL in order to prevent multiple threads trying to change resources at the same time, which would lead to undefined behavior. Since the GIL can only be held by one thread at a time, parallel thread execution cannot happen. Thus, the performance advantages demonstrated with multiprocessing and the separation of computing power no longer exist with this approach. Threading will instead simulate actions being run concurrently.

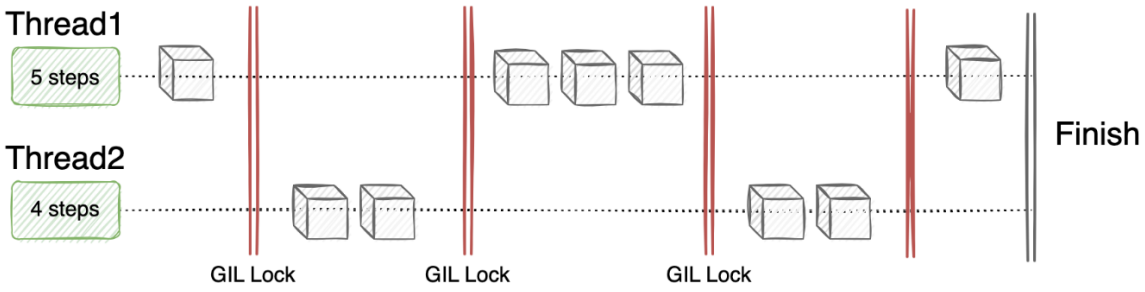

Figure 16: Threading Diagram [45]

Each small action will be run when the GIL is available and will give up control of the GIL when completed. Despite not having fully split the rendering and the BLE

read processes into their own cores, each one will have access to the single CPU core many times each second, letting each perform the actions necessary for reading data in and updating the graphs at a sufficiently high rate.

One of the central concepts with both the *tkinter* library and the *bleak* library is the use of *asyncio* event loops. An event loop is a handler that schedules tasks to be run, which is similar to how the GIL works but can be controlled by the individual libraries present or by the developer. When these two libraries were being run in their own process, there was no issue with the handler since each process had its own event loop. Each process implements the event loop internally, meaning that the developer does not need to directly interface with the *asyncio* library. However, both libraries seek to have full control of the main event loop created when the program is run and cannot have control at the same time, since both are now in the same process. To resolve this issue, the *tkinter* library was given primary access to the event loop, and a secondary event loop was created that the *bleak* library could run in. It is not possible to create two event loops within the same scope, so a *Thread* from the *threading* library is instantiated with a function to contain the event loop whenever the user attempts to connect to the SmartHub units over BLE. As shown in the code snippet below, the *self.find_smarthubs()* function is the one containing the BLE code and must be run in its own event loop. The *Thread* object calls the *ble_task()* function, which creates a new event loop. This event loop can then call the desired function, so now there are two pieces of code running in parallel.

```
def ble_task():
    loop = asyncio.new_event_loop()
    asyncio.set_event_loop(loop)
    loop.run_until_complete(self._find_smarthubs(smarthub_id))
    loop.close()

ble_thread = threading.Thread(target=ble_task, daemon=True)
ble_thread.start()
```

Figure 17: Secondary Event Loop Initialization

3.4.2 SmartHub Connection

To connect to the SmartHub, the four digit identification code associated with the specific unit must be known. The previous design would attempt to any SmartHub units available for pairing, but in a clinical setting there is a possibility that multiple devices could be turned on and looking for a connection. To avoid this, each SmartHub unit is assigned a four-digit code that is broadcast when looking to connect to a central device. This ensures that the correct SmartHub is connected to when the "connect" button in the interface is pressed. After it is pressed (or the user presses the keyboard Enter key while highlighting the entry box for the SmartHub ID), the new event loop is created, and the *bleak* library will see if both a left and a right SmartHub with the correct ID is available to connect to. If this not the case, an error message will appear stating the SmartHub that is missing. Once both SmartHub units are found and connected, the screen will update the statuses of each from "Not Connected" to "Connected."

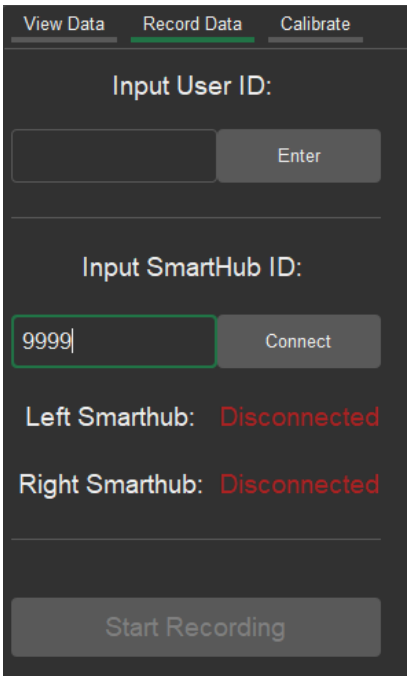

Figure 18: Display of SmartHub Connection

The two primary errors that can occur when connecting to the SmartHub are that the SmartHub does not identify both units and that the SmartHub cannot connect to the units once identified. In either case, a pop-up message will show the SmartHub unit that failed to connect, and the visible SmartHub unit will show "Connected" as a status.

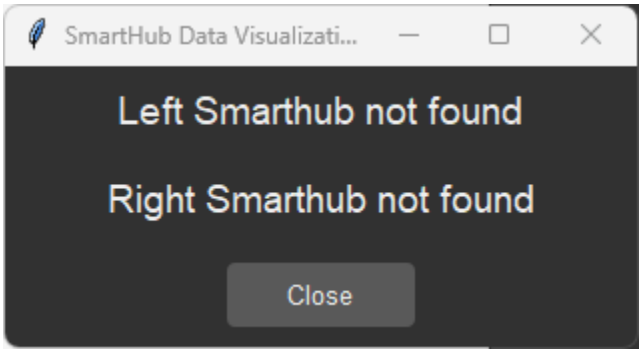

Figure 19: SmartHub Failure to Connect Pop-up

While the visible unit will show that it is connected, it will not actually be paired. Once a device is paired, it will begin attempting to stream data, which consumes significantly more battery than it would otherwise. Since data from just one SmartHub is essentially useless, the connection to the lone SmartHub is terminated, and can be reconnected through pressing the "Connect" button again.

If either SmartHub is not connected or a user ID is not input into the user ID entry box, the "Start Recording" button will be grayed out and unclickable, since a test run is unable to be taken. Test runs are automatically saved to the database after the test run is complete, so to find the test run later a user ID must be associated with it. Once both a user ID is input and both SmartHub units show as connected, the "Start Recording" button shows on the screen as clickable, and a dropdown menu appears below this button allowing for the selection of a calibration file associated with the specific SmartHub unit ID. Calibration files are discussed at length in Chapter 3.5, but contain information regarding the diameter of the wheel, the width between the SmartHubs mounted on either wheel, and gain values for the gyroscope values being streamed. The calibration file selected can be modified during the test run if desired, but once the run is complete, this option is no longer available since the data will already have been saved to the database. This drop-down menu can be seen in the screen capture below, which also shows the full layout of the Record Data Tab.

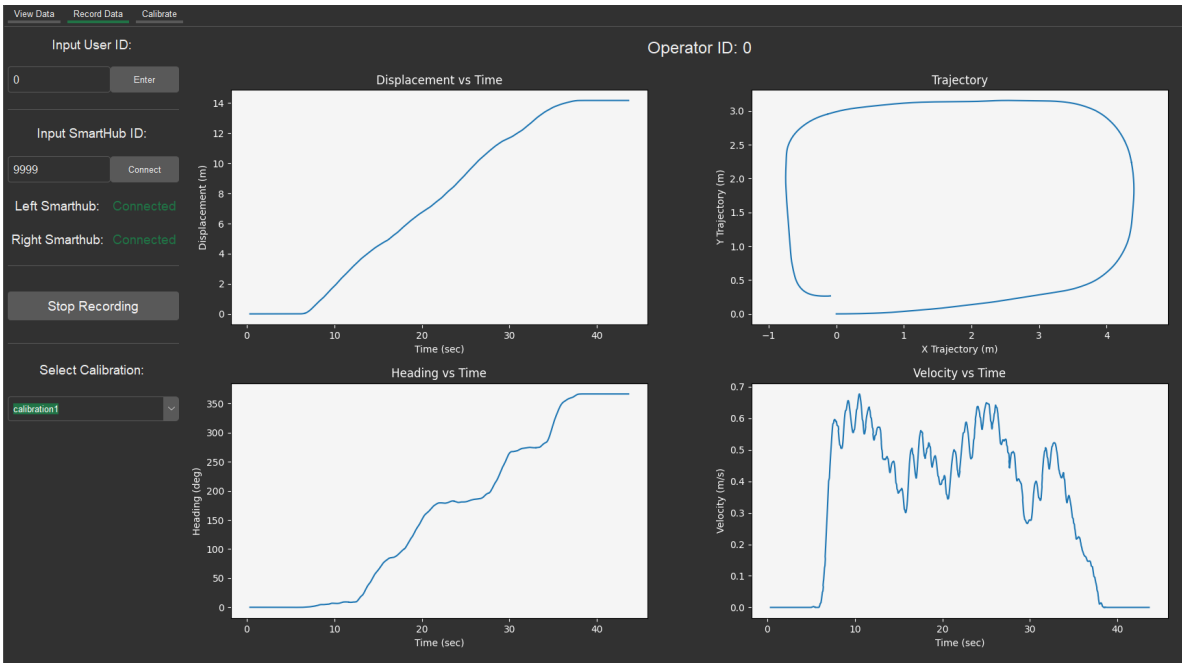

Figure 20: Record Data Tab During Test Run

3.4.3 Data Intake and Handling

Once the "Start Recording" button is clicked, *.start_notify()* methods are called for each BLE client instance, with a callback function to handle the data that was received. The previous SmartHub version used the *.read_gatt_char()* method, which was more straightforward to use since calling that function would return the values at the characteristic on the BLE peripheral device at that moment in time. However, the *.start_notify()* method instead utilizes a callback function, so whenever the peripheral device updates the values at a characteristic, it will send a notification to the central device. The central device will then run a callback function the instant this notification is received. This method is significantly faster than the *.read_gatt_char()* method, since it essentially streams the data, rather than polling the characteristics at regular intervals. It

was identified when using *.read_gatt_char()* alongside the rendering of graphs in the same thread that the computational power required to analyze metrics as well as show graphs on the screen was enough to slow the intake of data, while *.start_notify()* could still update in real time due to automatically giving runtime priority when new data was read in.

In this situation, however, unique considerations need to be made for the effective pre-processing of data before it is handled by the metric calculation scripts. When team member Strauss was originally developing all of the metric calculations, the methodology surrounding the reading of data was to read the left characteristic, read the right characteristic, and sent all of this data as one packet to the other process. Each timestamp in the data frame as a result had two datapoints associated with it, one for the left and one for the right. When using callback functions, the data will not be handled at the same time, but the calculation scripts need to see both left and right readings for each timestamp. There are several methods to handle this, including interpolation of intermediate data points and recoding of the metric scripts, but for the sake of simplicity logic was developed to only call the function when new information for both the left and right SmartHubs was seen. A flowchart with this logic can be seen below, but future iterations of the SmartHub should rewrite the calculation scripts to be able to handle reading in each SmartHubs values at different times.

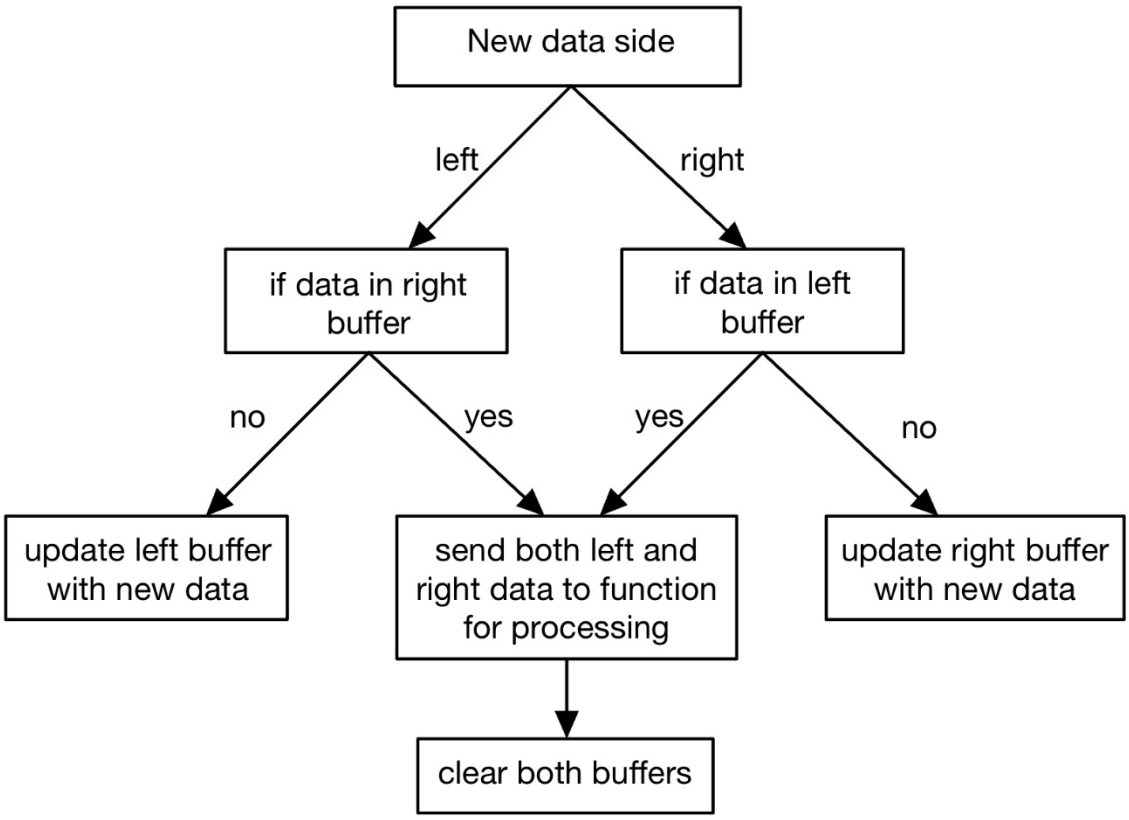

Figure 21: Flowchart for Data Intake Handling

3.4.4 Data Visualization

Alongside the callback functions implemented to handle incoming data, a new thread is created to allow for the visualization of graphs in real time. Once this thread is initiated, it uses the *tkinter.after()* method to restart the graph updating function asynchronously after a certain interval of time. This method schedules the function to be called at a certain point in time in the future. This function will thus be constantly run at regular intervals until the "Stop Recording" button is pressed by the user. The benefit of the implementation of threading is that the variables can be directly accessed within the

context of the graph updating function without extra logic for intake through a queue, but issues arise with the introduction of race conditions.

A race condition is when the value of a variable, and thus the flow of code, is influenced by the thread scheduler and not intentionally handled by the code itself. [4] In this application, all metrics calculation functions and graphing functions rely on all arrays to have the same length. For example, if the user is attempting to plot displacement versus time, and the time array is currently storing 4 more values than the displacement array, the plotting function will throw an error because it was not designed to handle the 4 extra values in the time array. On a case-by-case basis, this is not typically an issue since the function for adding new values to the arrays attempts to perform these actions all at the same time, keeping all arrays the same length. However, occasionally the values will be in the process of being appended to the arrays at the exact same time that the graph updating thread is attempting to perform calculations with them, so the arrays end up appearing to be different lengths.

This problem is resolved in two ways. First, at the start of the function before any actions are performed with the arrays, the arrays are checked to confirm that they are all the same length. If they are not, the *tkinter.after()* method is called to restart the function and the function is exited, meaning that no improper actions can be performed on the arrays. If the arrays are all the same length, copies are made of each variable to ensure that they are not changed at some point during the execution of the function. The *copy.deepcopy()* function is used to ensure this new copy is generated by value, not

reference. If the copy variable is simply set equal to the original variable, the copy will be generated by reference, meaning that it acts as a pointer to the original variable. Thus, changes to the new variable will affect the original one and vice versa.

```
# set value of time and gyros so we don't get a race condition, other data is being updated in real time
data = {'time_from_start': copy.deepcopy(self.data['time_from_start']),
        'gyro_left': copy.deepcopy(self.data['gyro_left']),
        'gyro_right': copy.deepcopy(self.data['gyro_right'])}
```

Figure 22: Copies of Arrays for Race Condition Prevention

After this point in the graph updating function, the metrics calculations are performed. These were largely designed by team member Strauss during Spring 2024 and were reformatted to fit the new interface structure. As an overview, the raw gyroscope data is passed through a Fourier transformation to return a frequency analysis. All frequencies higher than 6 Hz are filtered out and the signal is converted back into the time domain, which results in a smoothed signal. A smoothed signal is necessary for the peak analysis performed to find the starts and ends of wheelchair strokes. A clear peak indicates a clear end of a stroke, and extraneous noise in the data can lead to false peaks being identified. After this pre-processing is performed, the trajectory, displacement, heading, and velocity arrays are all calculated. These values are then displayed on the graphs. Rather than updating the entire graph again and re-rendering the *tkinter* widget, only the data is reset with the *.set_data()* command used to update the data being shown on the previous graph iteration. This does not target any other lines shown on the graph, so when the "Set Record Background" button is clicked on the View Data Tab, these graphs are not modified. Multiple graphs can be added through this button, even while

the test is running, and a variable tracking the order that these were added is included to ensure that the correct data is being updated.

After each graph update, the graphs are rescaled, so all the data is able to be seen on the screen, including any data that was used as part of the "Set Record Background" command. The *tk.after()* command is invoked to call the function again after a specified amount of time, and the function is exited.

After the test run is complete, all data will be saved to the MongoDB database. This is done automatically and requires no input from the user. The primary pieces of information used to uniquely identify the data are the user ID that must be input to begin a test run and the timestamp associated with the run. These should be used to find the test run in the View Data Tab and customize any metadata.

The SmartHub units are not automatically disconnected at the end of a test run. Due to the fact that it usually takes approximately 10 seconds to connect to the SmartHub units, it was desirable to reduce the number of times this connection needs to occur. At the end of a test run, all variables are reset, since the data has already been saved to the database. The thread to update the graphs is killed, and while the graphs are left on the screen, they are reset when the user begins a new test run. The *.stop_notify()* command is called to disable updating the data as it's seen by the host computer, but as stated earlier, the data is streamed constantly while the peripheral SmartHub devices are connected, so if reduced power consumption is desired at this point (if no test runs will be performed for a considerable amount of time) the user must disconnect the devices.

3.5 Calibration Tab

The purpose of the calibration tab is to calculate values to scale the SmartHub by in order to obtain the most accurate data for both the graphs shown on the screen and the metrics that can be downloaded. Several initial requirements for this methodology which were not incorporated into the last version of the SmartHub include the ability to input all values in the user interface, requiring no interaction with external configuration files or pieces of code, since this interface is intended to be used by a user with minimal software development experience. Also, the setup must be able to be performed with only tools that would be easily accessible in a clinic to avoid the need to purchase expensive items for accurate data collection.

3.5.1 Previous Calibration Methodology

In the previous version of the SmartHub, the methodology used to calibrate the gyroscopes relied on comparing a known rotational velocity to the rotational velocity output of the gyroscope. To conduct this, the gyroscope was placed on a turntable and measured at speeds of 33.3 and 45 RPM, and a linear fit was applied to the output. These speeds were then verified with an optical tachometer. Each gyroscope value read then in the future would have a linear transformation applied to it in the format $m*(rot/s) + b$.

There were several notable drawbacks with this setup, primarily with the tools used for the calibration. Turntables and optical tachometers are not easily accessible components in a clinical setting, so any recalibration that needed to be performed would

require sending the SmartHub units to another facility. Furthermore, the values that were calculated through this calibration needed to be input manually into a file listed as *params.py*. This would also require recompilation of the executable for each specific gyroscope used. Finally, this prior methodology allowed for a great deal of accuracy in the gyroscope readings, but inconsistencies in how the SmartHub was attached to the wheel as well as inexact measurements for the diameter and the distance between the wheels contributed to a significant amount of error.

```
1  import os
2  from datetime import datetime
3
4  left_gain = 1.13
5  left_offset = -0.049
6  right_gain = 1.12
7  right_offset = -0.0357
8
9  D_EULER_THRESH = 25
10 WHEEL_DIAM_IN = 24
11 IN_TO_M = 0.0254
12 DIST_WHEELS_IN = 26
13
```

Figure 23: *params.py* File for Manual Parameter Input

3.5.2 Proposed Calibration Methodology

It was evident that any new methodology for calibration implemented must be able to include frequent recalibration in response to changing environmental conditions as well as aging of components. Key parameters for calculating wheelchair propulsion metrics include diameter and distance between wheels and should be able to be fine-tuned more precisely than simply with a tape measure, which was previously used to determine the diameter of the wheels and the distance between the wheels.

By having the wheelchair with SmartHubs attached traverse a known trajectory and turn around known angles, the expected distances and headings at specific points in a test run can be found. The equations used to calculate the distance and heading can be compared to the expected values and the difference between these two numbers can be used as a loss value. Through use of a numerical solver, the expected gain for each gyroscope can be calculated as well as the diameter of the wheels and the distance between the wheels so as to minimize this loss value. These values should be able to be stored in the MongoDB database for retrieval alongside a specific test run and linked to the SmartHub ID used to perform the calibration.

It should be noted that while this method of calibration may be able to reach a certain degree of accuracy, it is not technically considered a calibration under the definition provided by the International Bureau of Weights and Measures, which states that a calibration is "[a]n operation that, under specified conditions, in a first step, establishes a relation between the quantity values with measurement uncertainties provided by measurement standards and corresponding indications with associated measurement and, in a second step, uses this information to establish a relation for obtaining a measurement result from an indication." [46] Since a user's ability to traverse a designated path does not correlate with a documented uncertainty, it cannot be considered a calibration in terms of the formal definition provided. The accuracy of this "calibration" method is almost entirely reliant on the ability of the user to traverse the known path provided and may lead to large errors if the user performing this calibration is not sufficiently skilled in the procedure. However, this method of determining the

wheelchair parameters is still useful for the provision of wheelchair velocities and trajectories. True calibration requires equipment not accessible to many clinics and users, and careful design of a calibration procedure can minimize the error introduced by the user. Due to these limitations, this type of test run consisting of a comparison between a known path and traversal will continue to be referred as a calibration, since it accomplishes the same goal as a traditional calibration, despite only loosely fitting the definition of the word.

When possible, the calibration of the units should be performed by an experienced user, such as a clinician, to minimize these losses. Re-calibration may be necessary in response to a changing environment or changing wheelchair specifications, but a successful design of a SmartHub case would ensure that the units reside on the wheel hub in the same way before and after removal and replacement of the devices to reduce the frequency of recalibration.

3.5.3 Original Calibration Implementation

A large portion of the logic for BLE data acquisition was reused from the Record Data Tab, including the connection process and the buffers for time synchronization between each SmartHub side. However, because there was no need for real time graphs for metrics, since this tab was not intended to be used for analysis, the only threading implementation was the initial connection to the SmartHub units. To start a calibration test run, the user must input the four-digit number for the SmartHub ID, and press connect in order for a button titled “Start Calibration” to be clickable.

The original setup for the calibration of the units consisted of 13 discrete smaller test runs. Users would travel with one wheel along a 5-meter straight path, make a turn around the end of the path while keeping the wheel on the path, then travel back with the same wheel along the path, and turn around the end of the path to end up back in the starting position. This would be repeated for the other wheel as well. Between each smaller run, the user would need to press a button titled "Next Step." At the end of the entire calibration, the distances and wheelchair headings would be gathered for each phase and compared against the known headings and distances at each of these points.

There were four equations utilized, consisting of the left and right wheel distances and the heading change while turning both left and right. The error in these values need to be minimized via the adjusting of the left and right wheel gains, the diameter, and the distance between the wheels. The equations for the left turns and distances are shown below, with the equations for the right wheel being identical.

$$Loss(D, m_L) = 20 - \sum_{i=0}^{n-1} \left\{ \left(R_{L_i} * m_L \right) * (t_{i+1} - t_i) * \frac{D}{2} \right\}$$

Equation 1: Loss in Left Wheel Distance

$$Loss(D, W, m_L, m_R) = 360 - \sum_{i=0}^{n-1} \left\{ \left(R_{R_i} * m_R - R_{L_i} * m_L \right) * (t_{i+1} - t_i) * \frac{D}{W} * \frac{180}{\pi} \right\}$$

Equation 3: Loss in Left Turn Heading

The numerical solver chosen to perform the error minimization was the *fsolve* function in the *scipy* Python library, which is built as a wrapper around the MINPACK

hybrd and *hybrj* routines. These routines serve to minimize the function provided by means of a hybrid algorithm, which coordinates the Newton method and gradient descent. [47], [48], [49]

While this method of setting up a calibration test is theoretically accurate, the primary issue lies in the user's inability to precisely position the wheelchair to face a specific direction. The model relies on the user being able to exactly position the wheelchair at a given angle, and determines the net error based on the difference between the expected angle and the given angle. However, it was quickly determined that without an external standard of reference, users were unable to position the wheelchair to face a direction within a few degrees during this test. Thus, the resulting calibrated values for the wheelchair parameters were often not as accurate as desired.

Another shortcoming of this prior calibration method is the way that the variables are implemented. With each metric derived from the wheelchair, the velocity of each wheel is first determined, then other mathematical operations can be performed. The equation for the velocity of a specific wheel at a specific time is shown below:

$$Velocity = \left(R_{L_i} * m_{L,R}\right) * \frac{D}{2}$$

Equation 3: Velocity Calculation

Since the rotational speed of the wheel is never used, only the translational velocity of the wheel itself, and because both the gain value and the diameter are terms

being solved for, the equation can be modified to use only one unknown per wheel, where Dm_L and Dm_R represent the diameter multiplied by the gain value:

$$Velocity = \frac{\left(R_{L_i} * Dm_{L,R}\right)}{2}$$

Equation 4: Modified Velocity Calculation

As a result, the only variables that need to be solved by the *fsolve* function are Dm_L, Dm_R, and W (the distance between the wheels).

3.5.4 Modified Calibration Implementation

The goal of the calibration redesign was to ensure that the calibration was representative of the entirety of the test run, not as determined by individual points taken during it. Also, while the original calibration run took several minutes to complete, a test run of less than a minute could also be achieved in theory, since only 3 variables need to be found. It was determined that the shortest methodology for this new calibration while retaining the ability to capture all necessary data required consisted of traveling in a straight line for 5 meters, turning around 180 degrees in place, and traveling back to the original starting point. The equations associated with this, in order to minimize the error in the 3 unknowns, tracked distance, the straightness of each of the 5-meter lines, and the proximity of the first 5-meter line to the second 5-meter line. The distance ensured that the magnitude of the gain values was correct, the straightness verified that the difference between the gain values was correct, and the trajectory alignment ensured that the distance between wheels was correct.

To achieve the straightness and trajectory alignment loss equations, a nearest neighbors (NN, not to be confused with neural network) algorithm was designed and implemented. Given two time series sets of data, the algorithm would iterate through each point in the first time series and find the net distance to the nearest point in the second time series. The results of all these distances were squared, added together, then the square root was taken, as given by the root mean squared (RMS) equation. The resulting value is an effective indicator of how well two paths align, since higher discrepancies between trajectory paths for the first and second time series will result in higher loss values. To calculate straightness loss, a straight line was drawn between the first point and the final point, and the loss was determined as a NN calculation of this line and the trajectory. To calculate the trajectory alignment loss, the paths for the initial 5-meter run and the second 5-meter run were compared with the NN algorithm. Finally, to calculate distance loss, the difference between a 10-meter total displacement and the calibration run displacement was found.

3.5.5 Calibration Considerations

An issue that presented itself during initial calibration testing was misaligned SmartHub units relative to the wheel plane. If a unit reading a gyroscope value is not vertical, it can still accurately record distance, as long as it has been calibrated with the procedure described above, since the gain value will compensate for this misalignment. However, if the gyroscopes are tilted partially into the horizontal plane, any change in the wheelchair's heading will introduce unwanted rotational velocities, with higher turning

rates leading to greater trajectory errors. To mitigate this, the gyroscopes must be positioned in the same plane as the wheel, meaning they should be precisely aligned in the radial direction. On cambered wheels, a gyroscope mounted at the top of the wheel may be angled slightly to the right or left, but as the wheel rotates, this orientation will reverse when it reaches the bottom.

While the current design of the SmartHub does not force the gyroscopes to remain on plane, future designs with this knowledge will be built so the gyroscopes remain radial and are unable to be mounted in an incorrect orientation. This will eliminate some user error associated with inexperience with mounting the SmartHub units.

Chapter 4: Results

4.1 Distance and Trajectory Validation

To validate the accuracy of the SmartHub gyroscope data and the calibration procedure designed in Section 3.5, the distance and the trajectory after calibration were measured against known values, and the error of the distance and trajectory were determined. Since the test runs taken were not necessarily traveling along a straight line, the final trajectory value at the end was taken as a straight-line distance and compared with the measured distance. The measured distance was determined with a laser rangefinder with <0.01 m accuracy. Trial distance runs were taken at distances of 5 meters, 20 meters, and 50 meters. Trajectory error was determined by traveling in a loop of a specified distance and returning to the original point. Trajectory loops were approximately 15 meters and 140 meters in length. Each trajectory loop was equally split between clockwise and counter-clockwise loops, to avoid bias in any given loop traversal orientation.

$$Distance\ Error = \frac{|traj[end]| - distance_{measured}}{distance_{measured}} * 100$$

Equation 5: Distance Error

$$Distance\ Error = \frac{|traj[end]|}{distance_{traveled}} * 100$$

Equation 6: Trajectory Error

Before each set of trials, 3 different calibrations were performed. Each run was applied to each calibration to achieve a larger dataset of trial runs. There were 3 total trials performed, which meant that 9 different calibration runs were tested for these trials. Each trial consisted of six 5-meter, six 20-meter, and four 50-meter straight line displacement calculations, as well as eight 15-meter and four 140-meter loop trajectory calculations. The first two sets of trials were performed by an experienced researcher, and the final trial set was performed by a researcher with minimal calibration experience. The full results of these trials are shown in Appendix A, and a summary is shown below.

Table 3: SmartHub Distance and Velocity Errors

Run Type:	**Trial 1-3 Average Error**	**Trial 4-6 Average Error**	**Trial 7-9 Average Error**
5-meter distance	0.656%	0.767%	1.106%
20-meter distance	0.920%	1.128%	1.800%
50-meter distance	0.655%	0.982%	1.571%
15-meter trajectory	2.728%	3.255%	2.134%
140-meter trajectory	7.527%	8.468%	9.650%

By taking the average of these errors, it is shown that the average error in distance is 1.065%, the average error for the 15-meter trajectory is 2.706%, and the error for the

140-meter trajectory is 8.727%. The error in distance is exceptionally accurate, providing a 72% reduction in error versus SmartHub II and III, which had an average error of 3.365%. [24] It can be inferred that the error in velocity is smaller than the error in the distance, since the velocity values are integrated to obtain distance, which introduces error to the system.

The error in the trajectory appears to scale with the time spent recording a run, with a run taking 10 times longer giving 4 times more net error relative to the distance. While the longer test runs of 140 meters still have small enough error to visually identify and characterize the type of run, it can be assumed that with significantly longer runs on the order of several hours, the trajectory plots would become unrecognizably poor. However, for standard clinical testing, which involves tests which last shorter than a few minutes (as shown by the Wheelchair Skills Test, a commonly used clinical analysis tool), the trajectory plots presented should be more than sufficient. [50] An example of a 15-meter and 140-meter trajectory plot with typical error is shown below. The accuracy of this trajectory is clearly able to indicate the type of the test run to a viewer for both runs, but the 15-meter run, which would be more similar to the types of runs performed in the Wheelchair Skills Test, is more accurate.

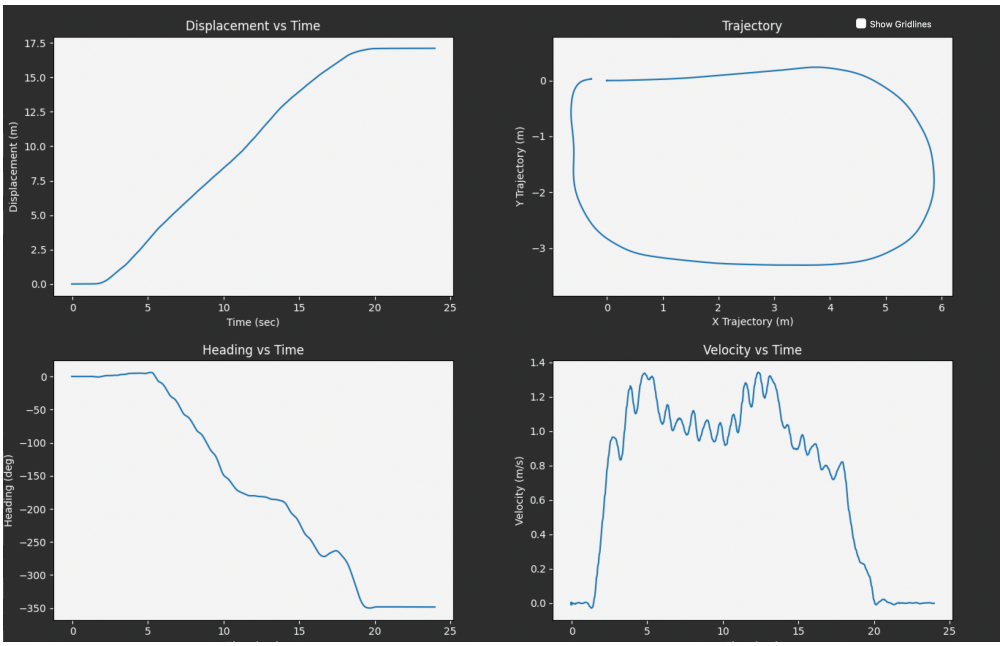

Figure 24: Example Trajectory, 15-meter loop

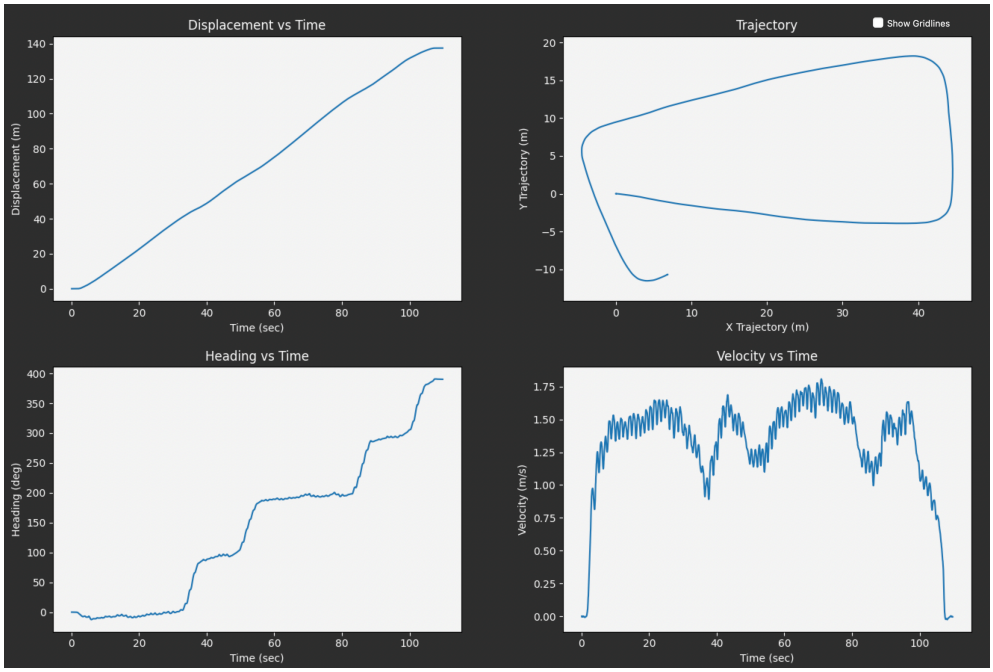

Figure 25: Example Trajectory: 140-meter loop

4.2 Propulsion Metrics

The primary benefit of the SmartHub over a typical wheelchair distance monitor is the ability to gather wheelchair propulsion data, as developed by team member Strauss. [42] To ensure accuracy, the calculated propulsion metrics should be validated against a source with empirically correct values. While previous SmartHub models were able to test against the SmartWheel, that device is no longer available and as a result, there are no other devices available to prove the validity of the metrics. The primary metric that has been linked to upper extremity strain is stroke frequency, or the number of strokes taken per unit time. [12], [13] This is a metric that can be validated visually, so video can be taken at the level of the wheelchair alongside the user propelling the wheelchair and the visually identified number of strokes can be compared to the calculated number of strokes from the SmartHub algorithm.

Since one of the main uses of the SmartHub is for analysis of novice wheelchair users who may not understand how to effectively propel a wheelchair, it is important that the device is also validated with these types of users. 9 novice users were chosen, each taking multiple test runs with the wheelchair. The age of all users was between 20-25, with 7 male and 2 female subjects. All users were in self-reported good physical condition but had never operated a wheelchair. The data gathered served to both validate the SmartHub's ability to identify stroke numbers and its ability to track improvements in wheelchair propulsion.

For each user, there were several types of tests that they were instructed to perform, as shown below:

Table 4: SmartHub Validation Test Types

	Distance	Pace	Trained or Untrained
1	20-meter	Comfortable	Untrained
2	20-meter	Fast	Untrained
3	20-meter	Comfortable	Trained
4	20-meter	Fast	Trained
5	45-meter	Comfortable	Trained

The distances given are approximate distances, since the users were not always able to effectively stop on a given point. By comfortable pace, users were instructed to travel at a rate that feels natural to them. A fast pace is the fastest pace the user could go while retaining full control of the wheelchair. Runs were split between trained and untrained tests so that the SmartHub could detect changes in propulsion when users were given instruction as to how to effectively propel the wheelchair. These instructions were based off prior studies which indicated the need to minimize stroke frequency and maximize stroke angle (given as the angle around the rim that the user's hand travels during the push phase), as well as discussions with clinicians. The instructions given to users to consider them "trained" are as follows:

1. Attempt to reach further back on the wheel rim as you propel the wheelchair and attempt to keep pushing the wheel as far forward as you can before starting the next stroke.
2. Focus on long and smooth strokes on the wheelchair rather than short and fast strokes.
3. Make sure that your hands do not drag on the rim as you bring them back to the starting position.

The expected outcome with this instruction was to have both smaller stroke frequency values calculated and to have a higher peak velocity after the wheelchair user was trained.

Figure 26: Frame of Stroke Frequency Validation Video

The summarized results of the stroke frequency analysis are shown below. The average number of strokes per trial is 22.67 with a standard deviation of 8.34.

Table 5: Stroke Frequency Visual Error Analysis

Test Subject Number	Number of Trials	Avg. Stroke Error (# strokes wrong/trial)
1	4	0.5
2	4	7.5
3	7	2.0
4	10	1.7
5	7	0.71
6	5	1.2
7	10	0.3
8	3	0
9	8	0.75

Table 6: Stroke Frequency Error - Before/After Training

	Number of Trials	Average Stroke Error (# strokes/trial)
Before Training	23	2.26
After Training	35	0.60

The stroke frequency and maximum velocity analysis performed to highlight the difference between a trained and untrained user is shown below.

Table 7: Stroke Frequency and Maximum Velocity Analysis

Subject Number	Untrained Avg Stroke Freq. (str/min)	Untrained Max Velocity (m/s)	Trained Avg Stroke Freq. (str/min)	Trained Max Velocity (m/s)
1	50.00	1.18	51.92	1.40
2	28.11	0.96	31.14	0.92
3	55.42	1.42	48.97	1.69
4	78.09	2.25	49.91	2.33
5	38.53	1.25	30.65	1.73
6	49.73	1.28	48.35	1.84
7	88.71	2.38	34.83	2.11
8	60.29	2.23	43.47	1.61
9	50.26	1.57	56.78	1.83

The results above show that there was an average increase in maximum velocity achieved of approximately 10.7%, and there was an average stroke frequency reduction of approximately 14.7%. The highest maximum velocity increase of all the participants was with subject 6, who achieved a velocity increase of 43.8%, and the greatest stroke

frequency reduction was with subject 7, who achieved a stroke frequency reduction of 60.7%. The results shown above thus clearly show that the SmartHub can effectively track improvements in various usage statistics with different types of users.

Chapter 5: Discussion

5.1 Comparison to SmartWheel

As shown above, the SmartHub BLE is effectively able to identify the number of strokes taken over a certain distance. However, there are small inconsistencies between the number of strokes identified in video analysis versus SmartHub statistical analysis. This is an issue that has also been identified in the SmartWheel and appears to result largely from conflicting definitions of what constitutes a stroke. [51] When novice wheelchair users in the test group propelled the wheelchair, there were instances identified where extremely short and jerky movements were taken which did not impart a significant amount of force into the wheelchair. This is identified in the video analysis by an extremely small stroke angle and is confirmed by analysis of the velocity plot. Since no increase in velocity is seen at these points, it can be assumed that no force was applied tangentially forward to the wheels, which is not something a purely gyroscope-based system can account for. This type of movement is something that has been described as non-propulsive pushrim contact, and it has been proposed as a third phase of a stroke alongside the push phase and recovery phase. [52] The SmartWheel is able to identify hand contact time separate from forces applied, so it is in theory able to identify these types of strokes.

Figure 27: Example of Unidentified Stroke

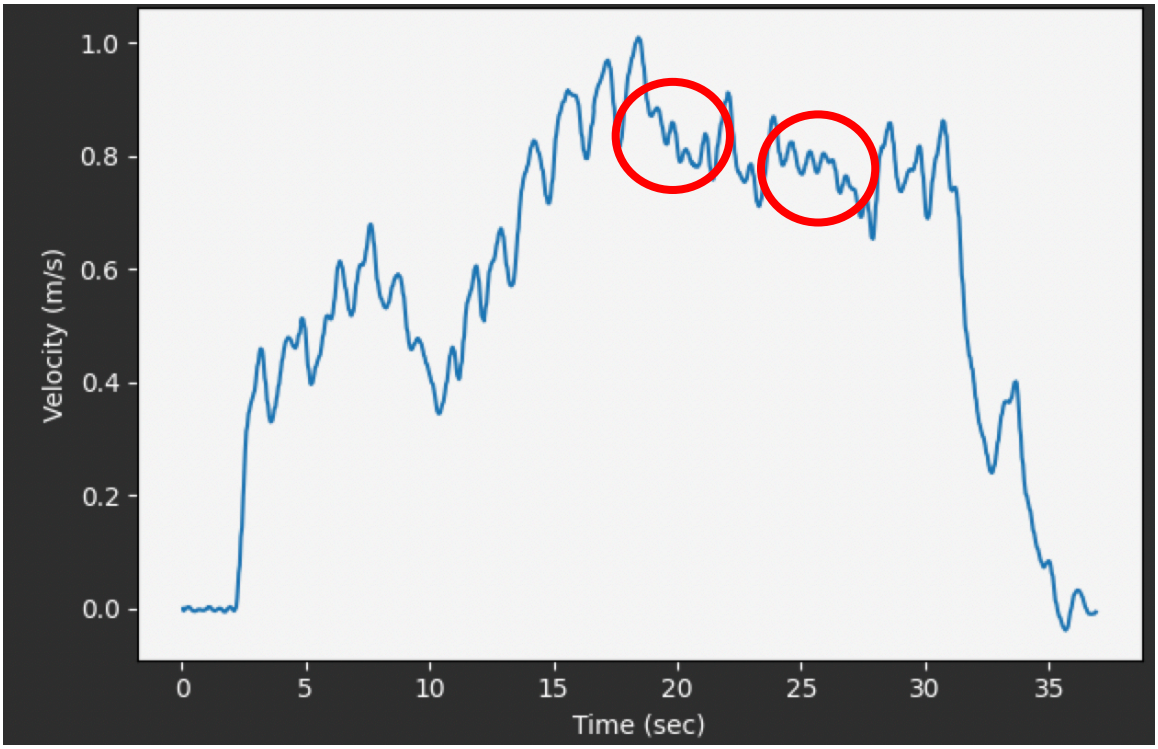

Figure 28: Velocity Plot with Unidentified Strokes

The graph above shows the velocity changes that are associated with this stroke. Because the net change in velocity during a stroke is so small and short relative to the other strokes, it is misclassified. As a result, the stroke frequency for this run is lower than it should be. The accuracy of the SmartHub in detecting stroke numbers did improve after the users were trained, however, indicating that in straight-line tests, a more capable and experienced individual will have fewer non-propulsive strokes. The plot below shows an example of a run in which the user has no non-propulsive strokes, which shows clear push and recovery phases.

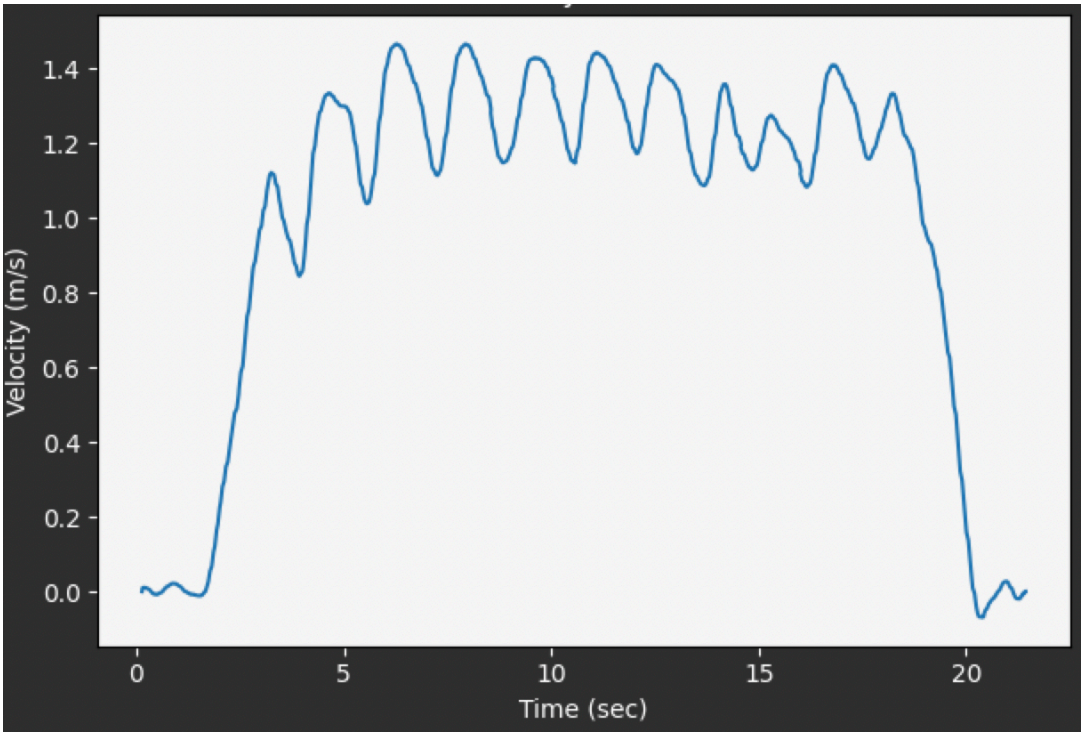

Figure 29: Velocity Plot with Zero Unidentified Strokes

The test runs taken for this were all straight-line runs, which is not always typical for a wheelchair user. Wheelchair users will need to navigate obstacles and make turns, which is not something that is captured in this data. Turning in a wheelchair has a higher likelihood of non-propulsive pushrim contact, since the focus of the user will be more on controlling the wheelchair rather than propelling it. Future work would continue to validate the SmartHub in environments requiring more turning and complex wheelchair usage.

These findings give insight into the future intended usage of the SmartHub. Since the SmartHub can only capture changes in velocity (which is directly correlated with force applied to the wheelchair), ineffective hand placements and usages will still need to be analyzed in a clinical setting. One benefit the SmartHub has over other data acquisition equipment such as the SmartWheel is the ability to test outside of the clinic, but the type of data gathered is a key factor in how to analyze it. It has been demonstrated that stroke frequency changes with the type of turn being performed, indicating that stroke frequency data cannot be effectively analyzed without knowledge of the type of motion being captured, which as a result requires manual analysis of the trajectory and velocity data for characterization. [53]

The SmartWheel has been shown to have discrepancies at detecting number of strokes, so the accuracy of the SmartWheel versus the SmartHub can be compared. [51] The table below shows the number of strokes miscalculated for the SmartWheel, the SmartHub before training, and the SmartHub after training, and an Apple Watch. [54]

Table 8: Stroke Count Error Comparison

Measurement Device	Number of Strokes Lost/100 Strokes
SmartWheel	4.79
SmartHub (untrained)	9.97
SmartHub (trained)	2.65
Apple Watch	13.9

It is unclear from the study analyzing the SmartWheel the exact wheelchair propulsion experience level of the individuals participating. Furthermore, the exact test procedure varied between the Apple Watch, the SmartWheel, and the SmartHubs. While extremely novice wheelchair users may introduce discrepancies into the system as shown before with non-propulsive strokes, even minimal training appears to allow for very accurate results. A better test of the accuracies of these 3 devices would require coordinating test procedures and method of determining what constitutes a stroke, as well as the use of similar participants. Further work should be performed on all 3 devices to determine the accuracy variance during turning. However, the results do indicate that the SmartHub generates similar accuracies in stroke recognition as the SmartWheel. It can be concluded that the SmartHub is among the most accurate stroke monitors available, and accessible for only a fraction of the price of the SmartWheel.

5.2 Rolling Resistance Relevance

The SmartHub offers researchers and clinicians alike the chance to study the effects of various data points outside of the clinic. Rolling resistance has been identified as a major influence in the likelihood of injury, due to the amount of force applied to a wheelchair. [55] This can be impacted by several outside factors, including caster wear, surface material, and the angle of the surface. [56], [57] Since the exact rolling resistance of the type of surface the patient is travelling on during their day-to-day activities is not able to be determined directly in a clinic, the SmartHub can be used to provide this insight to a clinician remotely. The rolling resistance of a wheelchair is determined by identifying the recovery phase of a stroke and calculating the average change in velocity, yielding a negative acceleration. To demonstrate the ability of the SmartHub to identify surfaces with a higher rolling resistance, several tests were taken on both a level surface and an angled surface. The average rolling resistance of the level surface was found to be -0.466 m/s^2, and the rolling resistance on the ramp was found to be -1.040 m/s^2. The SmartHub is able to effectively identify changes in rolling resistance, and an individual tracking wheelchair data in the home could use the SmartHub to diagnose increases in rolling resistance before they explicitly notice it themselves, especially in the context of a gradual change such as bearing degradation. On top of this, the SmartHub could be used to inform future studies testing the effects of various propulsion patterns and biomechanics on rolling resistance.

5.3 SmartHub Live Feedback Use

As demonstrated by the OptiPush feedback system, presenting a wheelchair user with visual real-time feedback of their stroke frequency can improve their stroke frequency as tested in a clinical setting. [18] While real-time stroke frequency is not currently available on the SmartHub, the protocol set up to transfer data in real time over BLE and process it allows for future work to add options to view other metrics in real time. As of now, the main benefit of the real time analysis is to view velocity. In a clinical setting currently, a therapist can only track approximate time and distance of a user's run, but the SmartHub enables clinicians to view velocity plots and gather conclusions in real time.

The current understanding of real-time stroke feedback is entirely based on studies in a laboratory and clinical setting. The development of the SmartHub as a portable and easy-to-use device allows for the implementation of this real-time feedback outside of the clinic as well and should be used to inform future studies on this matter. Live observation by a therapist may influence patient behaviors, while including this device during everyday activities could yield different results.

5.4 SmartHub Open-Sourcing

While there have been several research projects through the years seeking to capture wheelchair analytical data with many different purposes, there is a notable lack of published specifications for the construction of such a device. Devices range from

accelerometer-based systems, gyroscope-based systems with Bluetooth such as the G-WRM, and variants of the SmartWheel such as the OptiPush BioFeedback System. [18], [19], [58] None of these devices give information on their construction, custom data transfer protocols, or interfaces, so their usage is limited solely to the university or research organization it was developed at. Furthermore, research students may leave after graduating, which can lead to a significant loss of knowledge about the construction and design of a device. The drawbacks and issues with software development in academia are documented at length by team member Vogeler. [26] The design and construction of such a device for use in a clinical or research setting takes months or years, which is time that could be spent working with wheelchair users and gathering valuable data. The SmartHub intends to not only validate the use of a gyroscope-based system for velocity, trajectory, and wheel propulsion statistics, but also to make the software and hardware design specifications freely available by making it open-source to allow other researchers to produce and use this device with minimal effort. By reducing the time spent in the initial device design phase, researchers will be able to build off the SmartHub's design directly, which may lead to more robust, cheap, and widely available versions of the SmartHub. Instructions for the setup of the code for both the central and Arduino device are found in Appendix B. The code is released under a GPL-3 license, which ensures that future users can modify, distribute, and use the code freely, provided that any derivative work remains open-source under the same license. [59]

Chapter 6: Conclusion

6.1 SmartHub Future Design

Based on the knowledge gained through the development of SmartHub BLE, a new, updated version of the SmartHub has entered the development phase. While all of the core protocols and methods remain the same due to the clear success and accuracy of the current design, the hardware and software are being modified to more applicably match user needs and cut costs.

One primary concern with the SmartHub BLE is the need to keep the SmartHub perfectly on the plane of the wheel to avoid induced rotations when turning, which significantly affects trajectory accuracy. The current design does not ensure that it is mounted as such, and care must be taken to align it both directly vertically and not angled to the left or the right. A more effective design would guarantee that regardless of the wheel hub type used, the intuitive method of attaching the SmartHub would align it as intended. Future intended research outside of the clinic relies on inexperienced and untrained users affixing the SmartHub to a wheel hub, so the design should be modified to accommodate this use case.

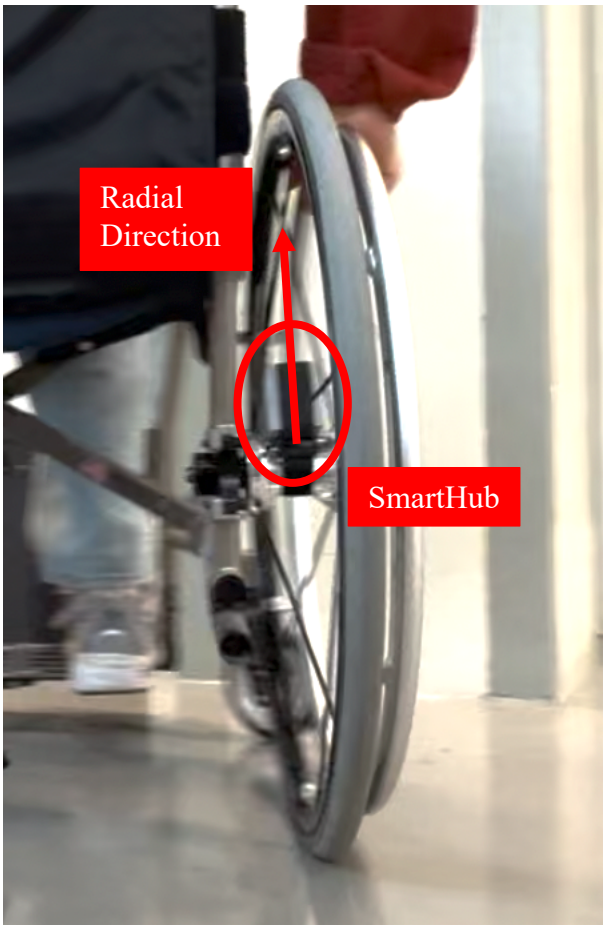

Figure 30: SmartHub Aligned Radially on Wheel Plane

One other change in design revolves around the hardware used. While the Arduino Nano 33 BLE alongside the PowerBoost 1000C contains many features beneficial for testing, there are significantly more components on these boards than are necessary to achieve the SmartHub's core functionality. The only components needed are the nRF52840 system on a chip (SoC) for data processing and BLE capability, a gyroscope to capture rotational velocities, and a chip with an external connector (e.g. USB-C, Micro-USB) to handle charging. Since the testing portion of the SmartHub has

largely been completed, fewer extraneous testing capabilities such as GPIO pins are needed. With these requirements, a new board was chosen to succeed the Arduino Nano 33 BLE. The Seeed Studio XIAO nRF52840 Sense board contains these core functionalities with a significantly smaller package size of only 21x17.8mm. [60] Since the Arduino Nano 33 BLE also uses the nRF52840 SoC, the majority of the code on the Arduino can directly transferred to the XIAO board without modification. The combined price of an Arduino Nano 33 BLE with an Adafruit PowerBoost 1000C is 45 dollars when purchased from their respective manufacturers, while a Seeed Studio XIAO nRF52840 Sense is only 16 dollars. [31], [60], [61] With the inclusion of a Li-ion battery (available for less than $10), this new design is less than half the price of the old design, is significantly smaller, and retains entirely the same functionality. The core components are thus only $52 dollars for a pair of units.

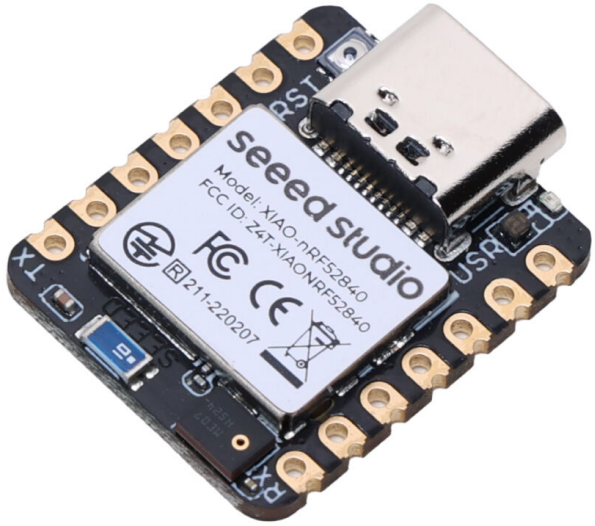

Figure 31: Seeed Studio XIAO nRF52840 Sense Development Board

One proposed design improvement to the SmartHub revolves around passive distance monitoring when inactive. This type of design would have 3 modes: On, which would constantly relay gyroscope data at 68 Hz to the central device for processing; Low Power, which would gather gyroscope data at a lower frequency, process it into a distance calculation, and relay that to a central device at longer intervals; and Off. While measurements still need to be taken with this new device to determine exactly how much power is used, it is estimated that approximately 20-30mA would be consumed during active usage, giving approximately 40 hours of battery life on a 1200 mAh battery. A Low Power mode would use significantly less battery power since it would only make a BLE connection once every several minutes or hours to provide a distance status update, and as a result could track long-term wheelchair usage statistics with minimal charging.

The other improvement that has been initiated for a newer version of the SmartHub is an updated user interface. While the current version achieves all of the intended functionality of the SmartHub in terms of live data recording and visualization, data retrieval from the cloud, and built-in calibration, there are several drawbacks of this design. Many design choices were selected with speed of development in mind to decrease the time taken to achieve a functional final interface. However, some issues with the current design include large file sizes, excessive RAM usage, lag during extended test runs, and non-responsive visuals. Non-responsive in this context refers to the ability of the interface to modify its appearance and functionality on various screen shapes and operating systems. Mobile phone usage with the current design of the user interface in Python and tkinter is not currently possible, and many of the issues with the

lag and RAM usage of the user interface can be resolved by moving to a language with better runtime performance. [62] However, shifting to a more applicable language for user interface development and cross-platform compatibility would resolve many of these issues. Due to the significant amount of tooling for web development for JavaScript, a preliminary successor to the user interface has been planned which would use React as a framework to construct the user interface with modular components, Electron for bundling the interface into a Windows, MacOS, and Linux compatible design with minimal code rewrites, and Capacitor for cross-compatibility between mobile phones and computers. [63], [64], [65]

6.2 SmartHub Future Studies

The work performed on the SmartHub allows for more in-depth studies to be performed regarding manual wheelchair usage in a variety of environments. The stroke frequencies and jerk values of wheelchair users with various skill levels can be determined in a variety of environments, whether that be in the home, in the clinic, or travelling in public. Further studies could also investigate correlations between strength of an individual's upper body and various wheelchair metrics. In order to examine the SmartHub's role as a tool to facilitate wheelchair fittings, novice wheelchair users could be placed into a variety of styles of wheelchairs with different diameter wheels, wheel widths, and cushion shapes, and their effectiveness at propelling themselves could be analyzed. Another study with significant potential implications would track novice wheelchair users over an extended period of time, such as months or years, to track how

their wheelchair propulsion metrics change with experience and time in the wheelchair. The SmartHub could also be used to track asymmetries in wheelchair propulsion, whether that be related to wheelchair inconsistencies or user inconsistencies. By monitoring subtle biases in the wheelchair trajectory to turn left or right when a straight line path is traversed, resulting in explicit user corrections, a clinician could devote more attention to the remedy of this problem.

On top of the work previously discussed for validating the SmartHub, more work can still be performed to accurately assess its ability to provide useful information in the clinic. Clinicians and patients should receive copies of the SmartHub, trained on its usage, and interviewed to assess the applicability of it in a clinical and at-home setting. Both user groups will be able to provide valuable feedback on the user interface of the current design, the new proposed JavaScript based design, as well as the ease of use of attaching the hardware of the SmartHub to the wheel of the wheelchair.

Finally, work was previously done on an IMU-based motion capture suit integrated into OpenSim to track the upper body biomechanics of wheelchair usage by team member Strauss. This suit could correlate force and muscle activation data through the OpenSim model, derived from the movements of the IMUs, to the kinematic data presented by the SmartHub. This could in turn give more insight into various propulsion metrics and how they correspond to forces exerted by the upper body during wheelchair propulsion. While an IMU suit can provide extensive information on the biomechanics of wheelchair usage, a wearable activity monitor such as an Apple Watch attached to a

user's wrist may be able to provide similar feedback, especially with regards to the stroke path taken by a user. Such devices are already commonly used widely, but custom software would need to be developed to accurately capture this information. The various stroke patterns commonly used by users are shown below.

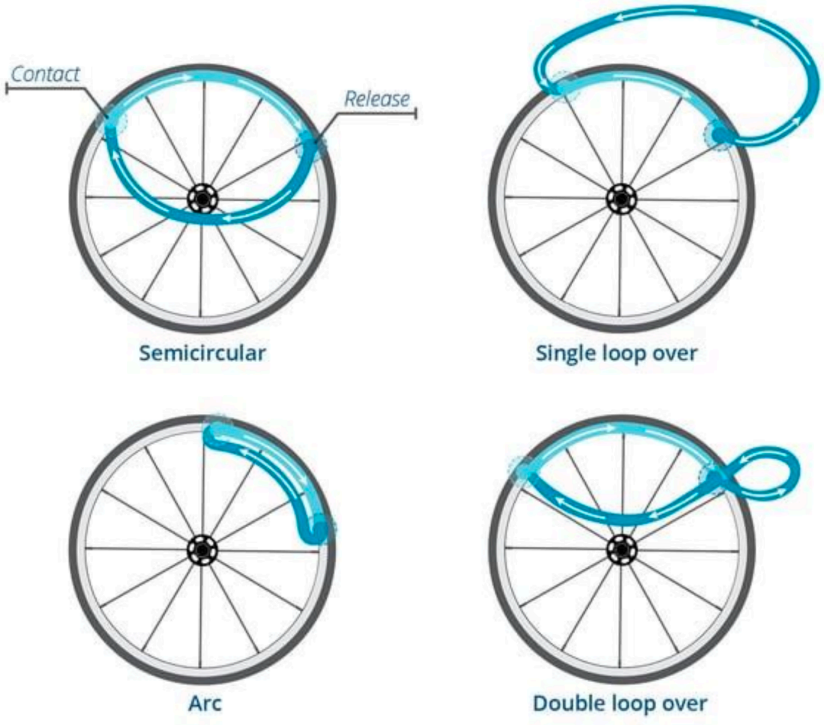

Figure 32: Stroke Patterns by Wheelchair Users [66]

Prior research has indicated that the semicircular or double loop stroke patterns have the lowest muscle demands, leading to the smallest risk for injury. [67] However, these studies often happen in a research facility and in a controlled environment. By using a device like an Apple Watch to monitor stroke patterns, it may be found that users do not always perform one stroke motion when traversing different paths than that found in a clinical setting. If evidence is found supporting variance in stroke technique in

various scenarios, the SmartHub could be used to identify information surrounding why a user would change stroke pattern, such as increased rolling resistance, sharp turns, or higher than normal acceleration.

There are several categories of manual wheelchair research in which the SmartHub could prove to be uniquely useful, and its portability, low cost, ease of use, and availability make it an ideal tool for both broad research studies and clinical individualized assessments. By providing detailed propulsion metrics across various environments, skill levels, and wheelchair configurations, the SmartHub enables clinicians and researchers alike the ability to analyze stroke propulsion efficiency and injury risk with greater accuracy than before. From a wheelchair users' perspective, the SmartHub offers real-time feedback and metrics for use as user training, allowing for more optimized propulsion techniques. As manual wheelchair research continues to evolve, the SmartHub allows for widespread data acquisition opportunities to more effectively identify room for growth in the field.

Bibliography

[1] "Travel Patterns of American Adults with Disabilities | Bureau of Transportation Statistics." Accessed: Dec. 18, 2024. [Online]. Available: https://www.bts.gov/travel-patterns-with-disabilities

[2] S. E. Sonenblum and S. Sprigle, "Wheelchair use in ultra-lightweight wheelchair users," *Disabil. Rehabil. Assist. Technol.*, vol. 12, no. 4, pp. 396–401, May 2017, doi: 10.1080/17483107.2016.1178819.

[3] M. A. Finley and M. M. Rodgers, "Prevalence and identification of shoulder pathology in athletic and nonathletic wheelchair users with shoulder pain: A pilot study," *J. Rehabil. Res. Dev.*, vol. 41, no. 3B, pp. 395–402, May 2004, doi: 10.1682/jrrd.2003.02.0022.

[4] A. Liampas *et al.*, "Musculoskeletal Pain Due to Wheelchair Use: A Systematic Review and Meta-Analysis," *Pain Ther.*, vol. 10, no. 2, pp. 973–984, Dec. 2021, doi: 10.1007/s40122-021-00294-5.

[5] H. E. J. Veeger, L. A. Rozendaal, and F. C. T. van der Helm, "Load on the shoulder in low intensity wheelchair propulsion," *Clin. Biomech.*, vol. 17, no. 3, pp. 211–218, Mar. 2002, doi: 10.1016/S0268-0033(02)00008-6.

[6] S. van Drongelen, L. H. van der Woude, T. W. Janssen, E. L. Angenot, E. K. Chadwick, and D. H. Veeger, "Mechanical Load on the Upper Extremity During Wheelchair Activities," *Arch. Phys. Med. Rehabil.*, vol. 86, no. 6, pp. 1214–1220, Jun. 2005, doi: 10.1016/j.apmr.2004.09.023.

[7] C. Hughes, W. Weimar, P. Sheth, and C. Brubaker, "Biomechanics of wheelchair propulsion as a function of seat position and user-to-chair interface," *Arch. Phys. Med. Rehabil.*, vol. 73, pp. 263–9, Apr. 1992.

[8] L. H. V. van der Woude, S. de Groot, and T. W. J. Janssen, "Manual wheelchairs: Research and innovation in rehabilitation, sports, daily life and health," *Med. Eng. Phys.*, vol. 28, no. 9, pp. 905–915, Nov. 2006, doi: 10.1016/j.medengphy.2005.12.001.

[9] L. C. Mâsse, M. Lamontagne, and M. D. O'Riain, "Biomechanical analysis of wheelchair propulsion for various seating positions," *J. Rehabil. Res. Dev.*, vol. 29, no. 3, pp. 12–28, 1992, doi: 10.1682/jrrd.1992.07.0012.

[10] B. Wieczorek and M. Sydor, "Laboratory Assessment of Manual Wheelchair Propulsion," *Appl. Sci.*, vol. 14, no. 22, Art. no. 22, Jan. 2024, doi: 10.3390/app142210737.

[11] M. L. Boninger, A. L. Souza, R. A. Cooper, S. G. Fitzgerald, A. M. Koontz, and B. T. Fay, "Propulsion patterns and pushrim biomechanics in manual wheelchair propulsion," *Arch. Phys. Med. Rehabil.*, vol. 83, no. 5, pp. 718–723, May 2002, doi: 10.1053/apmr.2002.32455.

[12] M. L. Boninger, R. A. Cooper, M. A. Baldwin, S. D. Shimada, and A. Koontz, "Wheelchair pushrim kinetics: Body weight and median nerve function," *Arch. Phys.*

Med. Rehabil., vol. 80, no. 8, pp. 910–915, Aug. 1999, doi: 10.1016/S0003-9993(99)90082-5.

[13] B. A. Silverstein, L. J. Fine, and T. J. Armstrong, "Occupational factors and carpal tunnel syndrome," *Am. J. Ind. Med.*, vol. 11, no. 3, pp. 343–358, 1987, doi: 10.1002/ajim.4700110310.

[14] R. E. Hoxie and L. Z. Rubenstein, "Are Older Pedestrians Allowed Enough Time to Cross Intersections Safely?," *J. Am. Geriatr. Soc.*, vol. 42, no. 3, pp. 241–244, Mar. 1994, doi: 10.1111/j.1532-5415.1994.tb01745.x.

[15] S. E. Sonenblum, S. Sprigle, J. Caspall, and R. Lopez, "Validation of an accelerometer-based method to measure the use of manual wheelchairs," *Med. Eng. Phys.*, vol. 34, no. 6, pp. 781–786, Jul. 2012, doi: 10.1016/j.medengphy.2012.05.009.

[16] L. Guo, A. M. Kwarciak, R. Rodriguez, N. Sarkar, and W. M. Richter, "Validation of a Biofeedback System for Wheelchair Propulsion Training," *Rehabil. Res. Pract.*, vol. 2011, p. 590780, 2011, doi: 10.1155/2011/590780.

[17] R. A. Cooper, "SMARTWheel: From Concept to Clinical Practice," *Prosthet. Orthot. Int.*, vol. 33, no. 3, pp. 198–209, Sep. 2009, doi: 10.1080/03093640903082126.

[18] L. Guo, A. M. Kwarciak, R. Rodriguez, N. Sarkar, and W. M. Richter, "Validation of a Biofeedback System for Wheelchair Propulsion Training," *Rehabil. Res. Pract.*, vol. 2011, p. 590780, 2011, doi: 10.1155/2011/590780.

[19] S. V. Hiremath, D. Ding, and R. A. Cooper, "Development and evaluation of a gyroscope-based wheel rotation monitor for manual wheelchair users," *J. Spinal Cord Med.*, vol. 36, no. 4, pp. 347–356, Jul. 2013, doi: 10.1179/2045772313Y.0000000113.

[20] E. Glasheen, A. Domingo, and J. Kressler, "Accuracy of Apple Watch fitness tracker for wheelchair use varies according to movement frequency and task," *Ann. Phys. Rehabil. Med.*, vol. 64, no. 1, p. 101382, Jan. 2021, doi: 10.1016/j.rehab.2020.03.007.

[21] N.-H. Benning, P. Knaup, and R. Rupp, "Measurement Performance of Activity Measurements with Newer Generation of Apple Watch in Wheelchair Users with Spinal Cord Injury," *Methods Inf. Med.*, vol. 60, no. Suppl 2, pp. e103–e110, Dec. 2021, doi: 10.1055/s-0041-1740236.

[22] "SmartHub: A personal fitness tracking device for manual wheelchair users (The Ohio State University)." Accessed: Dec. 21, 2024. [Online]. Available: https://www.resna.org/sites/default/files/conference/2015/sdc2015/shaffer.html

[23] R. Letcher, "SmartHub: A Low Cost Manual Wheelchair Fitness Metrics Tool for Clinicians, Researchers, and Wheelchair Users," The Ohio State University, 2017. Accessed: Dec. 22, 2024. [Online]. Available: https://etd.ohiolink.edu/acprod/odb_etd/etd/r/1501/10?clear=10&p10_accession_num=osu149270449042227

[24] N. Einstein, "SmartHub: Manual Wheelchair Data Extraction and Processing Device," The Ohio State University, 2019. Accessed: Dec. 29, 2024. [Online]. Available:

https://etd.ohiolink.edu/acprod/odb_etd/etd/r/1501/10?clear=10&p10_accession_num=osu1555352793977171
[25] N. Kaplan, “SmartHub: A Manual Wheelchair Activity Tracking Device for Patients and Clinicians,” The Ohio State University, 2020.
[26] K. Vogeler, “SmartHub: Transitioning Mixed Software and Hardware Projects Across Student Generations – Lessons Learned,” The Ohio State University, 2023.
[27] “Raspberry Pi Zero W - DEV-14277 - SparkFun Electronics.” Accessed: Jan. 06, 2025. [Online]. Available: https://www.sparkfun.com/products/14277
[28] “Quaternions.” Accessed: Jan. 06, 2025. [Online]. Available: https://ece.montana.edu/seniordesign/archive/SP14/UnderwaterNavigation/Quaternions.html
[29] N. Waltz, “SmartHub: A Manual Wheelchair Data Acquisition and Analysis Tool for Patients and Clinicians,” The Ohio State University, 2024. Accessed: Jan. 13, 2025. [Online]. Available: https://kb.osu.edu/handle/1811/104104
[30] G. D. Putra, A. R. Pratama, A. Lazovik, and M. Aiello, “Comparison of energy consumption in Wi-Fi and bluetooth communication in a Smart Building,” in *2017 IEEE 7th Annual Computing and Communication Workshop and Conference (CCWC)*, Las Vegas, NV, USA: IEEE, Jan. 2017, pp. 1–6. doi: 10.1109/CCWC.2017.7868425.
[31] “Arduino Nano 33 BLE,” Arduino Official Store. Accessed: Jan. 08, 2025. [Online]. Available: https://store.arduino.cc/products/arduino-nano-33-ble
[32] H. Blidh, *hbldh/bleak*. (Jan. 12, 2025). Python. Accessed: Jan. 13, 2025. [Online]. Available: https://github.com/hbldh/bleak
[33] “The GNU General Public License v3.0 - GNU Project - Free Software Foundation.” Accessed: Jan. 12, 2025. [Online]. Available: https://www.gnu.org/licenses/gpl-3.0.en.html
[34] “Riverbank Computing | PyQt Commercial Version.” Accessed: Jan. 12, 2025. [Online]. Available: https://riverbankcomputing.com/commercial/pyqt
[35] “History and License,” Python documentation. Accessed: Jan. 12, 2025. [Online]. Available: https://docs.python.org/3/license.html
[36] “tkinter.ttk — Tk themed widgets,” Python documentation. Accessed: Jan. 12, 2025. [Online]. Available: https://docs.python.org/3/library/tkinter.ttk.html
[37] B. Dévényi, *rdbende/Forest-ttk-theme*. (Jan. 11, 2025). Tcl. Accessed: Jan. 12, 2025. [Online]. Available: https://github.com/rdbende/Forest-ttk-theme
[38] “PyInstaller Manual — PyInstaller 6.11.1 documentation.” Accessed: Jan. 12, 2025. [Online]. Available: https://pyinstaller.org/en/stable/
[39] “JSON Example.” Accessed: Mar. 31, 2025. [Online]. Available: https://json.org/example.html
[40] “Matplotlib — Visualization with Python.” Accessed: Jan. 13, 2025. [Online]. Available: https://matplotlib.org/
[41] “multiprocessing — Process-based parallelism,” Python documentation. Accessed: Jan. 13, 2025. [Online]. Available: https://docs.python.org/3/library/multiprocessing.html

[42] M. Strauss, “SmartHub: Device for Quantitative Analysis of Manual Wheelchair Propulsion,” The Ohio State University, 2024. Accessed: Jan. 13, 2025. [Online]. Available: https://etd.ohiolink.edu/acprod/odb_etd/r/etd/search/10?p10_accession_num=osu1713372214169183&clear=10&session=104775387599063
[43] “What is Transport Layer Security (TLS)?” Accessed: Jan. 06, 2025. [Online]. Available: https://www.cloudflare.com/learning/ssl/transport-layer-security-tls/
[44] “GlobalInterpreterLock - Python Wiki.” Accessed: Jan. 06, 2025. [Online]. Available: https://wiki.python.org/moin/GlobalInterpreterLock
[45] C. Tao, “Probably the Easiest Tutorial for Python Threads, Processes and GIL,” Medium. Accessed: Jan. 06, 2025. [Online]. Available: https://towardsdatascience.com/dont-know-what-is-python-gil-this-may-be-the-easiest-tutorial-3b99805d2225
[46] “BIPM Glossary of Terms.” Accessed: Mar. 25, 2025. [Online]. Available: https://www.bipm.org/documents/20126/46590067/WIP-GVv3.4-Glossary.pdf/4a855e8f-344b-5c8f-eafb-0bb913089bf9
[47] “minpack/hybrd.html.” Accessed: Jan. 06, 2025. [Online]. Available: https://www.math.utah.edu/software/minpack/minpack/hybrd.html
[48] “minpack/hybrj.html.” Accessed: Jan. 06, 2025. [Online]. Available: https://www.math.utah.edu/software/minpack/minpack/hybrj.html
[49] “fsolve — SciPy v1.15.0 Manual.” Accessed: Jan. 06, 2025. [Online]. Available: https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.fsolve.html
[50] T. Chu, “Wheelchair Skills Test (WST, WST-Q),” SCIRE Professional. Accessed: Mar. 27, 2025. [Online]. Available: https://scireproject.com/outcome/wheelchair-skills-test/
[51] H. Soleymani, B. Jeng, B. Abdelmessih, R. Cowan, and R. W. Motl, “Accuracy and Precision of Actigraphy and SMARTwheels for Measuring Push Counts Across a Series of Wheelchair Propulsion Trials in Non-disabled Young Adults,” *Int. J. Med. Stud.*, vol. 11, no. 1, Art. no. 1, Mar. 2023, doi: 10.5195/ijms.2023.1950.
[52] A. M. Kwarciak, S. A. Sisto, M. Yarossi, R. Price, E. Komaroff, and M. L. Boninger, “Redefining the Manual Wheelchair Stroke Cycle: Identification and Impact of Nonpropulsive Pushrim Contact,” *Arch. Phys. Med. Rehabil.*, vol. 90, no. 1, pp. 20–26, Jan. 2009, doi: 10.1016/j.apmr.2008.07.013.
[53] D. Chaikhot, M. J. D. Taylor, W. H. K. de Vries, and F. J. Hettinga, “Biomechanics of wheelchair turning manoeuvres: novel insights into wheelchair propulsion,” *Front. Sports Act. Living*, vol. 5, p. 1127514, Jun. 2023, doi: 10.3389/fspor.2023.1127514.
[54] N. Benning, P. Knaup, and R. Rupp, “Comparison of accuracy of activity measurements with wearable activity trackers in wheelchair users: a preliminary evaluation,” vol. 16, p. Doc05, Aug. 2020, doi: 10.3205/mibe000208.
[55] “Preservation of Upper Limb Function Following Spinal Cord Injury,” *J. Spinal Cord Med.*, vol. 28, no. 5, pp. 434–470, 2005.
[56] “Exploring the Relationship of Rolling Resistance, Tire Type, and Surface in Wheelchair Rear Wheels.” Accessed: Mar. 29, 2025. [Online]. Available:

https://www.resna.org/sites/default/files/conference/2018/wheelchair_seating/Dickey.html
[57] J. Bascou, C. Sauret, F. Lavaste, and H. Pillet, “Is bearing resistance negligible during wheelchair locomotion? Design and validation of a testing device,” *Acta Bioeng. Biomech.*, vol. 19, no. 3, pp. 165–176, 2017.
[58] S. E. Sonenblum, S. Sprigle, J. Caspall, and R. Lopez, “Validation of an accelerometer-based method to measure the use of manual wheelchairs,” *Med. Eng. Phys.*, vol. 34, no. 6, pp. 781–786, Jul. 2012, doi: 10.1016/j.medengphy.2012.05.009.
[59] “The GNU General Public License v3.0 - GNU Project - Free Software Foundation.” Accessed: Mar. 31, 2025. [Online]. Available: https://www.gnu.org/licenses/gpl-3.0.en.html
[60] “Seeed Studio XIAO nRF52840 Sense - TinyML/TensorFlow Lite- IMU / Microphone - Bluetooth5.0.” Accessed: Mar. 30, 2025. [Online]. Available: https://www.seeedstudio.com/Seeed-XIAO-BLE-Sense-nRF52840-p-5253.html
[61] A. Industries, “PowerBoost 1000 Charger - Rechargeable 5V Lipo USB Boost @ 1A.” Accessed: Mar. 30, 2025. [Online]. Available: https://www.adafruit.com/product/2465
[62] D. Lion, A. Chiu, M. Stumm, and D. Yuan, “Investigating Managed Language Runtime Performance: Why {JavaScript} and Python are 8x and 29x slower than C++, yet Java and Go can be Faster?,” presented at the 2022 USENIX Annual Technical Conference (USENIX ATC 22), 2022, pp. 835–852. Accessed: Mar. 30, 2025. [Online]. Available: https://www.usenix.org/conference/atc22/presentation/lion
[63] “React.” Accessed: Mar. 30, 2025. [Online]. Available: https://react.dev/
[64] “Build cross-platform desktop apps with JavaScript, HTML, and CSS | Electron.” Accessed: Mar. 30, 2025. [Online]. Available: https://electronjs.org/
[65] “Capacitor by Ionic - Cross-platform apps with web technology,” Capacitor. Accessed: Mar. 30, 2025. [Online]. Available: https://capacitorjs.com/
[66] S. M. OTR/ATP, “Manual Wheelchair Propulsion Patterns and Efficiency.” Accessed: Mar. 30, 2025. [Online]. Available: https://hub.permobil.com/blog/manual-wheelchair-propulsion-patterns-efficiency
[67] J. S. Slowik, P. S. Requejo, S. J. Mulroy, and R. R. Neptune, “The Influence of Wheelchair Propulsion Hand Pattern on Upper Extremity Muscle Power and Stress,” *J. Biomech.*, vol. 49, no. 9, pp. 1554–1561, Jun. 2016, doi: 10.1016/j.jbiomech.2016.03.031.

Appendix A: Full SmartHub Validation Data

A.1 SmartHub Distance Validation

The following data contains the distances for all 5 meter, 20 meter, and 50 meter trial runs for SmartHub distance validation. The units are in meters. Each number in the tables represents the calculated distance, and each run was measured to follow the distance in the left-hand column.

Intended Distance	Trial 1	Trial 2	Trial 3
5 meters	5.0071	4.9575	4.9468
5 meters	5.0555	5.0054	4.9946
5 meters	5.0339	4.9839	4.9732
5 meters	5.0157	4.9660	4.9553
5 meters	5.0202	4.9704	4.9597
5 meters	4.9833	4.9338	4.9232

Intended Distance	Trial 4	Trial 5	Trial 6
5 meters	5.0490	5.0158	4.9508
5 meters	5.0522	5.0202	4.9544
5 meters	5.0291	4.9965	4.9315
5 meters	5.0520	5.0200	4.9543
5 meters	5.0578	5.0250	4.9596
5 meters	5.0507	5.0185	4.9529

Intended Distance	Trial 7	Trial 8	Trial 9
5 meters	5.0279	5.1057	5.0789
5 meters	4.9708	5.0477	5.0209
5 meters	4.9918	5.0691	5.0429
5 meters	5.0362	5.1141	5.0868
5 meters	5.0228	5.1005	5.0732
5 meters	5.0012	5.0786	5.0516

Intended Distance	Trial 1	Trial 2	Trial 3
20 meters	20.2221	20.0513	20.0513
20 meters	20.3549	20.1714	20.1714
20 meters	20.2284	20.0525	20.0525
20 meters	20.3218	20.1419	20.1419
20 meters	20.3628	20.1799	20.1799
20 meters	20.3224	20.1520	20.1520

Intended Distance	Trial 4	Trial 5	Trial 6
20 meters	20.3696	20.2285	19.9775
20 meters	20.4197	20.3065	20.0374
20 meters	20.3873	20.2431	19.9938
20 meters	20.4053	20.3012	20.0267
20 meters	20.3789	20.2285	19.9831
20 meters	20.4015	20.2693	20.0123

Intended Distance	Trial 7	Trial 8	Trial 9
20 meters	20.1740	20.4886	20.3829
20 meters	20.1731	20.4866	20.3664
20 meters	20.2000	20.5138	20.3919
20 meters	20.2115	20.5248	20.3930
20 meters	20.1886	20.5029	20.3909
20 meters	20.1981	20.5113	20.3816

Intended Distance	Trial 1	Trial 2	Trial 3
50 meters	50.5298	50.2519	50.1650
50 meters	50.4690	50.2897	50.2391
50 meters	50.4315	50.2909	50.2550
50 meters	50.4544	50.2982	50.2564

Intended Distance	Trial 4	Trial 5	Trial 6
50 meters	50.8820	50.4673	49.9746
50 meters	50.9340	50.5662	50.0445
50 meters	50.8114	50.5933	49.9801
50 meters	50.8948	50.6149	50.0393

Intended Distance	Trial 7	Trial 8	Trial 9
50 meters	50.3223	51.1336	50.8107
50 meters	50.3280	51.1392	50.8138
50 meters	50.4193	51.2220	50.7652
50 meters	50.4139	51.2217	50.8368

A.2: SmartHub Trajectory Validation

The following data contains the trajectories for the 15 meter loops and 140 meter loop runs for SmartHub trajectory validation. The units are in meters. Each number in the tables represents the error in meters from the original point, and each run was measured to follow the distance in the left hand column.

Intended Distance	Trial 1	Trial 2	Trial 3
15 meter loop (counter-clockwise)	0.2317	0.4698	0.7976
15 meter loop (counter-clockwise)	0.1470	0.3627	0.6840
15 meter loop (counter-clockwise)	0.1798	0.4067	0.7265
15 meter loop (counter-clockwise)	0.0575	0.2956	0.6314
15 meter loop (clockwise)	0.4172	0.0252	0.6484
15 meter loop (clockwise)	0.4923	0.1037	0.7243
15 meter loop (clockwise)	0.3233	0.1283	0.5533
15 meter loop (clockwise)	0.5228	0.1388	0.7543

Intended Distance	Trial 4	Trial 5	Trial 6
15 meter loop (counter-clockwise)	0.5310	0.5324	0.3021
15 meter loop (counter-clockwise)	0.5873	0.6002	0.3688
15 meter loop (counter-clockwise)	0.5279	0.5352	0.3069
15 meter loop (counter-clockwise)	0.6375	0.6253	0.3964
15 meter loop (clockwise)	0.8090	0.7870	0.5510
15 meter loop (clockwise)	0.7198	0.7146	0.4958
15 meter loop (clockwise)	0.3800	0.3790	0.2027
15 meter loop (clockwise)	0.3080	0.3146	0.1072

Intended Distance	Trial 7	Trial 8	Trial 9
15 meter loop (counter-clockwise)	0.1983	0.4703	0.1895
15 meter loop (counter-clockwise)	0.1946	0.3105	0.2019
15 meter loop (counter-clockwise)	0.2089	0.2254	0.2238
15 meter loop (counter-clockwise)	0.1725	0.3596	0.1739
15 meter loop (clockwise)	0.2206	0.2277	0.6634
15 meter loop (clockwise)	0.2401	0.2203	0.6463
15 meter loop (clockwise)	0.1651	0.3528	0.7606
15 meter loop (clockwise)	0.0693	0.3817	0.8062

Intended Distance	Trial 1	Trial 2	Trial 3
140 meter loop (counter-clockwise)	18.2408	2.8627	5.1265
140 meter loop (counter-clockwise)	17.7613	3.3455	4.1868
140 meter loop (clockwise)	25.3841	2.1950	15.1622
140 meter loop (clockwise)	24.9979	1.4699	14.7504

Intended Distance	Trial 4	Trial 5	Trial 6
140 meter loop (counter-clockwise)	24.3844	4.0938	10.7444
140 meter loop (counter-clockwise)	13.4021	12.6892	0.7179
140 meter loop (clockwise)	15.7289	16.9596	6.3485
140 meter loop (clockwise)	13.8084	19.2464	4.1307

Intended Distance	Trial 7	Trial 8	Trial 9
140 meter loop (counter-clockwise)	18.5683	14.4582	14.8439
140 meter loop (counter-clockwise)	18.4989	14.5324	14.1006
140 meter loop (clockwise)	8.7709	10.3715	17.8005
140 meter loop (clockwise)	3.4890	5.0506	21.6382

A.3 SmartHub Stroke Count Validation

The following data demonstrates the SmartHub's ability to accurately measure strokes during wheelchair propulsion. The following table (a copy of Table 4) shows the 5 different test types patients were asked to complete. Detailed information about the test procedure is found in Chapter 4.2.

	Distance	Pace	Trained or Untrained
1	20-meter	Comfortable	Untrained
2	20-meter	Fast	Untrained
3	20-meter	Comfortable	Trained
4	20-meter	Fast	Trained
5	45-meter	Comfortable	Trained

The data contains the subject number, the test type, the total number strokes that the SmartHub determined took place, the strokes visually identified by an observer, and the total stroke difference between the two. Several runs had invalid video evidence, and for several runs the SmartHub was unable to determine the number of strokes. These instances are left blank.

Subject Number	Test Type (from Table 4)	Strokes Calculated by SmartHub	Strokes Visually Identified	Count Difference
1	1	21	22	1
1	1	17	17	0
1	3	18	19	1
1	3	13	13	0

Subject Number	Test Type (from Table 4)	Strokes Calculated by SmartHub	Strokes Visually Identified	Count Difference
2	1	16	29	13
2	1	15	26	11
2	3	14	18	4
2	3	17	19	2

Subject Number	Test Type (from Table 4)	Strokes Calculated by SmartHub	Strokes Visually Identified	Count Difference
3	1	35	40	5
3	1	36	39	3
3	2	23	26	3
3	3	17	17	0
3	4	17	17	0
3	5	36	39	3
3	5	34	34	0

Subject Number	Test Type (from Table 4)	Strokes Calculated by SmartHub	Strokes Visually Identified	Count Difference
4	1	19	25	6
4	1	22	24	2
4	2	17	23	6
4	2	14	14	0
4	3	14	14	0
4	3	12	12	0
4	5	35	35	0
4	5	34	33	1
4	5	35	36	1
4	5	31	32	1

Subject Number	Test Type (from Table 4)	Strokes Calculated by SmartHub	Strokes Visually Identified	Count Difference
5	1	21	24	3
5	1	18	18	0
5	2	12	14	2
5	2	14	14	0
5	4	7	7	0
5	4		8	
5	5	24	24	0
5	5	25	25	0

Subject Number	Test Type (from Table 4)	Strokes Calculated by SmartHub	Strokes Visually Identified	Count Difference
6	1	25		
6	1	26		
6	2	19		
6	2	18		
6	3	13		
6	3	13	13	0
6	4	17	13	-4
6	4	16	15	-1
6	5	31	32	1
6	5	34	34	0

Subject Number	Test Type (from Table 4)	Strokes Calculated by SmartHub	Strokes Visually Identified	Count Difference
7	1	21	22	1
7	1	19	20	1
7	2	19	19	0
7	2	19	19	0
7	3	9	9	0
7	3	10	9	1
7	3	9	9	0
7	3	8	8	0
7	5	20	20	0
7	5	19	19	0

Subject Number	Test Type (from Table 4)	Strokes Calculated by SmartHub	Strokes Visually Identified	Count Difference
8	1	24		
8	1	21		
8	2	15		
8	2	11		
8	3	12		
8	3	11		
8	5	30		
8	5	24	24	0
8	5	28	28	0
8	5	22	22	0

Subject Number	Test Type (from Table 4)	Strokes Calculated by SmartHub	Strokes Visually Identified	Count Difference
9	1	20	22	2
9	1	19	21	2
9	2	16	16	0
9	2	15	16	1
9	4	14	14	0
9	4	15	15	0
9	5	35	35	0
9	5	33	34	1

A.4 Other SmartHub Metrics

While the following data was not explicitly validated, it is all directly derived from velocity and stroke data, which was validated. The following metrics were calculated by the SmartHub and are used for analysis of user performance. The metrics included are maximum velocity achieved, steady state stroke frequency, and rolling resistance.

Subject Number	Test Type (from Table 4)	Maximum Velocity (m/s)	Steady-State Stroke Frequency (str/min)	Rolling Resistance (m/s^2)
1	1	1.15	53.33	-0.470
1	1	1.18	46.67	-0.358
1	3	1.40	56.54	-0.497
1	3	1.39	47.29	-0.459

Subject Number	Test Type (from Table 4)	Maximum Velocity (m/s)	Steady-State Stroke Frequency (str/min)	Rolling Resistance (m/s^2)
2	1	0.96	28.20	-0.275
2	1	0.91	28.01	-0.219
2	3	0.92	29.91	-0.197
2	3	0.84	32.37	-0.293

Subject Number	Test Type (from Table 4)	Maximum Velocity (m/s)	Steady-State Stroke Frequency (str/min)	Rolling Resistance (m/s^2)
3	1	0.70	49.63	-0.369
3	1	0.67	49.40	-0.318
3	2	1.42	67.22	4.929
3	3	1.42	51.76	-0.437
3	4	1.69	56.85	-0.447
3	5	1.24	42.97	-0.288
3	5	1.21	44.30	-0.360

Subject Number	Test Type (from Table 4)	Maximum Velocity (m/s)	Steady-State Stroke Frequency (str/min)	Rolling Resistance (m/s^2)
4	1	1.43	73.23	-0.495
4	1	1.12	60.32	-0.023
4	2	2.25	100.72	-0.704
4	2	2.33	77.71	-1.542
4	3	1.56	40.20	-0.544
4	3	1.34	34.73	-0.459
4	5	1.41	52.21	-0.473
4	5	1.35	47.73	-0.469
4	5	1.33	48.64	-0.459
4	5	1.41	48.12	-0.456

Subject Number	Test Type (from Table 4)	Maximum Velocity (m/s)	Steady-State Stroke Frequency (str/min)	Rolling Resistance (m/s^2)
5	1	0.82	37.13	-0.261
5	1	0.88	37.51	-0.282
5	2	1.25	38.02	-0.281
5	2	1.25	41.45	-0.352
5	4	1.73	25.56	-0.474
5	4	1.65		
5	5	1.28	32.58	-0.246
5	5	1.31	33.80	-0.400

Subject Number	Test Type (from Table 4)	Maximum Velocity (m/s)	Steady-State Stroke Frequency (str/min)	Rolling Resistance (m/s^2)
6	1	0.71	38.57	-0.230
6	1	0.77	43.73	-0.294
6	2	1.28	58.29	-0.449
6	2	1.25	58.31	-0.449
6	3	1.31	39.87	-0.512
6	3	1.26	38.55	-0.509
6	4	1.63	51.71	-0.711
6	4	1.84	62.05	-0.910
6	5	1.45	48.42	-0.470
6	5	1.33	49.51	-0.476

Subject Number	Test Type (from Table 4)	Maximum Velocity (m/s)	Steady-State Stroke Frequency (str/min)	Rolling Resistance (m/s^2)
7	1	1.35	66.17	-2.503
7	1	1.37	59.76	-0.428
7	2	2.24	106.64	-1.075
7	2	2.38	122.26	-1.448
7	3	1.72	33.11	-0.580
7	3	1.63	33.81	-0.589
7	3	2.11	40.75	-1.063
7	3	1.85	30.69	-0.653
7	5	1.56	34.39	-0.579
7	5	1.81	36.23	-0.637

Subject Number	Test Type (from Table 4)	Maximum Velocity (m/s)	Steady-State Stroke Frequency (str/min)	Rolling Resistance (m/s^2)
8	1	1.33	66.29	-0.590
8	1	1.11	51.23	-0.441
8	2	2.06	73.97	-0.641
8	2	2.23	49.67	-0.692
8	3	1.61	44.01	-0.462
8	3	1.39	37.46	-0.266
8	5	1.46	44.25	-0.411
8	5	1.56	38.56	-0.378
8	5	1.35	39.55	-0.363
8	5	2.41	57.00	-0.417

Subject Number	Test Type (from Table 4)	Maximum Velocity (m/s)	Steady-State Stroke Frequency (str/min)	Rolling Resistance (m/s^2)
9	1	1.04	45.06	-0.388
9	1	1.01	44.17	-0.409
9	2	1.57	57.26	-0.499
9	2	1.57	54.55	-0.564
9	4	1.74	57.59	-0.567
9	4	1.83	59.11	-0.680
9	5	1.49	55.78	-0.507
9	5	1.47	54.63	-0.446

Appendix B: SmartHub Code Instructions

The code associated with the SmartHub can be accessed from https://github.com/nwaltz/smarthub_ble. To run the interface, only the executable at dist/smarthub_executable.exe needs to be downloaded. To run the code, Git and Python 3.11 should be installed locally, and familiarity with these tools is expected. Any lines preceded with a "$" sign should be run in the terminal. After downloading the code locally (either as a .zip file or by cloning the repository with the command below).

```
$ git clone https://github.com/nwaltz/smarthub_ble
```

A virtual environment will need to be created to install all the packages associated with the SmartHub.

```
$ python -m venv .smarthub
$ source .smarthub/bin/activate (if on MacOS or Linux)
$ .smarthub/Scripts/activate (if on Windows)
$ pip install -r requirements.txt (if this yields errors to compatibility issues, libraries may need to be installed manually)
```

If this all works correctly, running "$ python main.py" should initiate the user interface. The code in the GitHub page, however, relies on interfacing with a MongoDB database to save and retrieve data. An account, database, and collections can be created at https://cloud.mongodb.com. The code in the SmarthubApp class will need to be modified to point to the new URL generated with the creation of the database. The database will need to be titled Smarthub and the collections in the database should be titled test_collection and test_config to align with the code in the other files. Finally, a .pem certificate can be generated to avoid needing to login with a username and password. By clicking the Database Access tab and pressing "add new database user," a custom certificate can be created with a specific role. Placing this certificate in the top level of the local repository will automatically authenticate the user on startup of the app.

For building the hardware of the SmartHub units, an Arduino Nano 33 BLE Rev 2 unit can be obtained from https://store-usa.arduino.cc/products/nano-33-ble-rev2, and a PowerBoost 1000C can be obtained from https://www.adafruit.com/product/2465. A 3.7V lithium-ion battery is also required to power the device. By soldering the 5V and GND of the PowerBoost to the VIN and GND of the Arduino, the Arduino will be turned on while the battery is connected to the PowerBoost. A switch can be attached to the EN pin of the PowerBoost, and by connecting the other side of the switch to GND it can turn the unit off.

To upload the code to the Arduino, the Arduino code in arduino/arduino_code/arduino_code.ino will need to be uploaded. The unit ID and unit

side can be set in the preprocessor definitions at the top of the code. If the Nano Rev 2 board is used, the code does not need to be modified otherwise. If the Nano Rev 1 or other gyroscope system is used, the appropriate gyroscope library will need to be included. For ease of use in flashing the device, a custom script has been developed which installs all libraries, compiles the code, detects the associated port, and uploads it to the Arduino. This code can be run with the following command:

```
$ python arduino/flash_nano.py
```

Many dependencies used in this project are consistently being updated, so the information presented here is subject to change. While many core algorithms used here will remain the same, issues with library versions, APIs, and deprecated functions may arise over time. To ensure compatibility and maintain functionality, it's important to regularly check for updates, review changelogs, and test new versions before fully integrating them into the project.