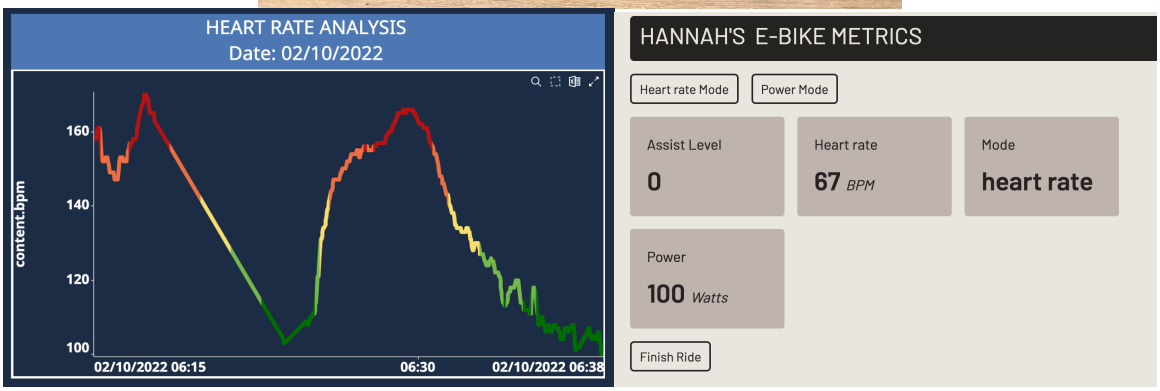
E-Bike Team



Submitted to Prof. Gatchell, Prof. Beltran & Jesse Brown

Submitted By: E-bike Team
Jake Cvetas
Madeline Durmowicz
Gillian Finnegan
Bryan Horn
Hannah Huang
Matthew Kim
Aaron Pulvermacher

Sponsored by Altair: Darius Fadanelli & Armin Veitl
Faculty Supervisors: Prof. Michael Beltran and Prof. David Gatchell

March 17th, 2022

Executive Summary

Over the course of the last twenty weeks, our team has worked to develop a smart[1] electric bicycle that will respond to the rider's biometric feedback in real time. Over the last several years there has been a push to add more devices to the Internet of Things[2] (IoT), which has allowed consumers to have access to more data, real-time data, all through cloud-based connectivity. Altair, a software company, aims to use their IoT with built in visualization software to add connectivity to a smart electric bicycle to allow riders to learn more about their rides before, during, and after a cycling session.

We developed specifications for the bicycle in order to better target the needs of our client. The majority of our specifications pertain to the technical capabilities of the mechatronics and software. In-depth research into several different components was performed to complete our final prototype including research of microcontrollers, heart rate sensors, and Bluetooth modules. We purchased and acquired components with the purpose of meeting the technical e-bike specifications such as the time between data transmission and motor adjustments.

This quarter, our primary goal was to integrate *Altair's* SmartWorks IoT software into the local-control of an e-bike motor that was established using Adafruit's ESP32 Feather early on during the second half of the project. We have used SmartWorks to send and receive data (heart rate, power, cadence, motor assist level, etc.), plot real-time visualizations, and push data to an external website. The process of integrating SmartWorks into the bike and how others could do so is documented in the following report.

We have used SmartWorks to create two different live feedback modes on the bike: heart rate and power meter. Both consist of feedback loops in which biometric data (either heart rate in beats per minute or user input power in Watts) is analyzed to scale the motor output power according to pre-programmed algorithms. The secondary feedback loop that has been added to the bike is one that sets the motor power with an inverse relationship to the user's input power in order to keep the total power output of the system constant. The collected biometric data is plotted and displayed using Panopticon.

This report will walk through the team's design process, prototype iterations, final alpha prototype, and plans for future work and development.

---

[1] Smart products: A smart, or connected, product is a device that is linked to the Internet so it can share information about itself, its environment and its users.

[2] IoT describes physical objects that are embedded with sensors, processing ability, software, and other technologies, and that connect and exchange data with other devices and systems over the Internet or other communications networks.

Table of Contents

List of Figures

List of Tables

Mission Statement

Our team's mission is to take a holistic approach in the design and creation of an e-bike that showcases *Altair's* software integration capabilities. Our client, *Altair*, has provided access to their software, SmartWorks IoT, a platform that can receive data from sensors, create visual dashboarding, and provide detailed diagnostics during data collection. With SmartWorks IoT, our team aims to utilize sensors to collect health data from the electric bicycle rider, in parallel with the design of an e-bike, and integrate the two for a final product and demonstrate the full capabilities of *Altair's* software.

Problem Background

Client Background

Our client, Altair, is a global technology company that provides software and cloud solutions in the areas of product development, high-performance computing (HPC), and data analytics. One of Altair's products is SmartWorks, which is an IoT platform that allows its user to quickly build applications that connect with smart products. Altair aims to showcase the power and versatility of SmartWorks through the creation of an e-bike.

Altair has two divisions of software that the team has access to. The first software platform is called Altair One. Altair One is a catalog of pre-packaged software solutions that the company has created to address business needs that have been previously identified. These solutions include a job scheduler that maximizes throughput, a resource management system, a cloud-native AI-driven design app, as well as a platform to perform real-time dashboarding. Some of these apps are cloud-native but many require the download of the Windows executable. The second software platform is called Altair SmartWorks IoT. This SmartWorks platform enables users to create customizable cloud applications without the need for much coding. The team will focus on using the SmartWorks platform (depicted in Figure 1) to store and display relevant data as well as communicate with the e-bike.

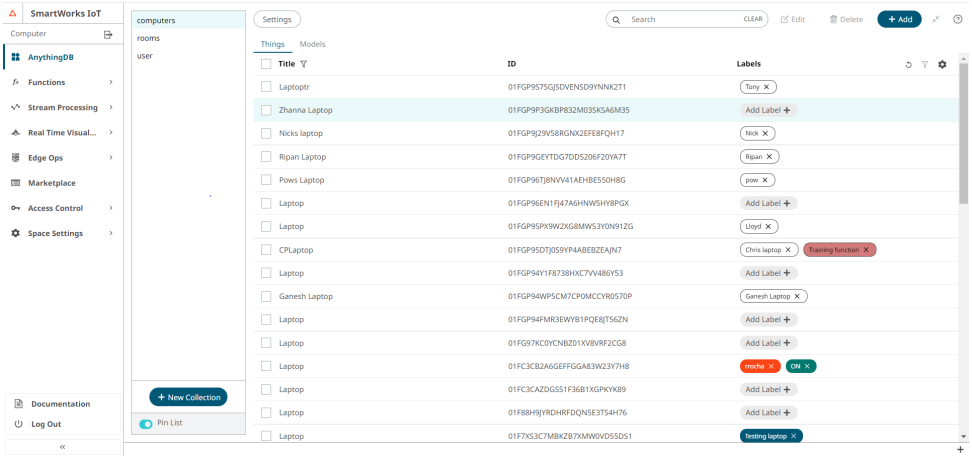


Figure 1: SmartWorks IoT landing page displaying each smart device in the cloud

E-Bikes Today

E-Bikes, or electric bicycles, have become increasingly popular over the past few decades. E-Bikes now account for a large percentage of bike sales in Europe and are beginning to dominate the market in the US as well. The e-bike industry is growing by an average of 23% each year and is showing no signs of slowing down. In fact, the current COVID-19 pandemic has increased sales further. E-bike sales in Europe are rapidly matching the sales of traditional mechanical bikes, as seen in Figure 2.

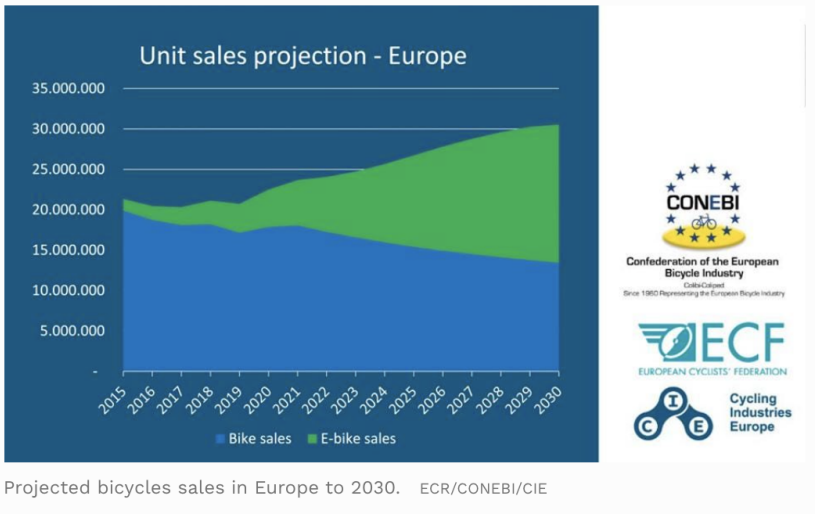


Figure 2: E-Bike Sales To Grow From 3.7 Million To 17 Million Per Year By 2030[3]

Combining the exciting technologies of electric bicycles and a cloud-based Internet of Things allows for the team and subsequent Altair customers to create a bicycle with new and useful features that make for a more comfortable ride.

Primary Users

The primary users of this product are those that are using or could be using Altair's SmartWorks IoT platform for prototyping or developing their own software product. We aimed to utilize several different facets of the SmartWorks IoT platform including MQTT and HTTP connection, Panopticon Visualizations, and live, multi-source data uploads and plotting. The users will be able to view the documentation created from this product and the existing SmartWorks coded functions, which will enable them to develop and problem shoot their own designs and products. These target users would not necessarily require, or desire, the functionality of the e-bike; however, the data collection, storage, processing, and transmittance protocols used in this project could be used across multiple different applications.

[3] C. Reid, "E-bike sales to grow from 3.7 million to 17 million per year by 2030, forecast industry experts," *Forbes*, 02-Dec-2020.

Other target users may include future SmartWorks engineers for continuing to develop the software for more physical implementation, and the Altair marketing team to use as a proof of concept for attracting customers to buy the software.

Secondary User Background Research

When initially beginning the project, the team investigated different populations or users for a biometric, cloud-controlled e-bike. After conducting interviews with members of Northwestern cycling and triathlon teams, e-bike commuters, e-bike owners, and recreational cyclists at Altair, we decided to focus our design on creating a more comfortable riding experience for a group of riders that "cannot bike like they used to, but want to." (Appendix A). We aimed to integrate biometric and environmental sensors with the e-bike and the SmartWorks software with the goal of achieving this, or some other group achieving this, in the future.

In the beginning stages of this project, research was done to determine which feedback metrics would be most valuable to use and viable to collect. Specifically, biometrics were considered to be an area of high potential. However, after looking into biometrics such as pulse oximetry, glucose sensors, sweat sensors, muscle sensors (using EMGs)[4], and more, we decided to create feedback loops surrounding rider input power and heart rate. These would enable the bike to try to mitigate both cardiovascular and musculoskeletal overexertion.

This secondary user of our project, the main user of an e-bike on the market, must be considered when continuing to develop the product design.

Stakeholders

The primary stakeholders in the development of our project is our client, *Altair*, as well as the prototypers and developers that will be using *Altair's* SmartWorks IoT platforms. The product we are developing, a smart biometric e-bike, has other, secondary stakeholders including the user/rider, community members, medical caregivers and personnels, and manufacturers.

Primary Stakeholders:

Altair

Our project goal was to integrate Altair's SmartWorks IoT software into the e-bike as much as possible in order to design the bike ride of the future. Our goal is to develop the bike through exploration of the SmartWorks software, while documenting the process of implementation for

[4]EMG stands for electromyography sensor is one that measures small electrical signals generated by your muscles when they're moved

Shawn, "What is EMG sensor, Myoware and how to use with Arduino?," *Latest Open Tech From Seeed*, 2019. [Online]. Available: https://www.seeedstudio.com/blog/2019/12/29/what-is-emg-sensor-myoware-and-how-to-use-with-arduino/. [Accessed: 09-Dec-2021].

the benefit of Altair's future use. This will help Altair's developers and engineers identify the key functions of their software and provide documentation for these functions.

Software Developers/Prototypers

As a result of our efforts we hope that software developers and prototypers could benefit from following the processes and steps that we took in the development of the prototype through the use of SmartWorks IoT software. Ideally, this will enable developers to integrate SmartWorks into their own market products by following the team's design process.

Secondary Stakeholders:

User: Recreational E-Bike Riders

We aim to create an e-bike for recreational bikers that want to bike comfortably and are unable to do so on a traditional, mechanical bike. Our target user or persona may have physical limitations that we hope to overcome with technology incorporated into the bike, and will be able to track their own personal health data through the software portion of the project.

Family Members of Users

Considering the age of our target user group, the users may want to use the product in order to keep up with or recreate with more athletic or healthy family members. The product must be usable in this situation amongst other riders to allow the users to keep pace with all fellow riders. Considering the customizable software capabilities of our product, the family members of users should also be able to create their own profiles to use the bike at different levels of exertion if they choose to.

Other Bicyclists, Pedestrians, Commuters, Road Sharers

People sharing bike paths, roads, and trails with our bike users must be considered during design as introducing e-bikes, especially those with some automatic control features, impact their own commutes, walks, rides, and drives. The product must be safe for the rider and the individuals around the rider at all times. Also, the product should not be a nuisance to these stakeholders, in that the sound and heat generated by the bike should be negligible to non-riders.

Local Governments

Local governments would be interested in this project as they are trying to keep bike paths & trails safe for pedestrians. E-Bikes can reach higher speeds much faster and more easily than traditional bicycles which poses a greater danger to others on local paths. The determination of regulations for parks and roads, as well as the integration of bike lanes are steps that local governments need to take in order for e-bikes to gain accessibility before popularity.

Bicycle Retailer/Distributor

Bicycle retailers can be consulted and considered with a pivot from mechanical bikes. These are the experts that would best be suited for the determination of bike manufacturability, market interest, as well as using them and their storefront to market e-bike as a viable recreational bike.

Physical Therapist/Medical Caregivers

Physical therapists may be able to provide more insight into the pain points of riders who struggle physically. They would be experts and be able to provide a direction in what biometric data is relevant, give background information on health, and the correlation between data and health. Once the product is completed, physical therapists and medical caregivers of our users can use the data from the user profiles to help track the patient's health.

Manufacturers (of E-Bike Components & Other Existing Technology)

While we do not anticipate manufacturers playing a major role in our design decisions at this stage, manufacturers will need to be consulted to determine feasibility of the design. Manufacturers also interface with retailers which play an important role in getting the bikes to the user. In our design, we may plan to incorporate other existing technology our users may purchase separately and/or already own. We may choose to use cellular phones as control mechanisms and/or existing heart rate monitors in watches or other wearable devices. This may result in a partnership between manufacturers of different devices and/or result in an increase in demand for wearable sensors or smart phones which would require greater rates of manufacturing. Manufacturers of products used to develop our bike will also have more use cases for their own products from our integration into the cloud.

Competitive Analysis, IP, & Benchmarking

Competition & Intellectual Property

Our team looked into several different kinds of competitive products and other intellectual property patents to learn more about the different developments in electronic bicycle technology and the different sensors being incorporated into the product. We looked into e-bikes on the market and different sensors in development. Also, some current solutions for incorporating biometric technology into bike rides were examined for comparison to the intended product.

Relevant Commercial Products

From the benchmarking and alternate solutions research compiled in Tables 1 and 2, eight popular e-bikes have been placed in a matrix with many important parameters, including price, weight, and battery capacity, recorded for each. The full benchmarking matrix is displayed below in Table 1. A more qualitative summary of each alternate solution is displayed in Table 2. This includes features that are most important in a "good" e-bike, as well as readouts and functions

that bikers may want. Note that this data is skewed based on the location of the market it is for—US vs. European regulations.

Table 1: Benchmarking Matrix

												Control/Readout							
Product	American (US) or European (EU)	Price ($)	Weight (kg)	Battery Capacity (Wh)	Motor Torque (Nm)	Max Range (kM)	*Max Speed (km/hr)	Peak Power (W)	Hub/Mid Motor	App	Bike Display	Speed/Assist	Distance/Duration	GPS	Lock Bike	Personalization	Power/Torque	Biometric data	Comments
Mercedez-Benz Ebike	Both	4500	20	504	75	100	***32	???	Mid	???	???	???	???	???	???	???	???	???	Lacking information
FlyKly Smart Bike	EU	1971	**??	160	20	30	25	500	Hub	X		X	X	X	X				Discontinued?
Econic One (Smart Urban)	EU	2656	22	500	45	142	25	500	Hub	X	X	X	X	X	X				
Fazua (Urban, Canyon)	EU	3811	17.6	252	55	120	25	450	Mid	X	X	X	X	X		X	X		Garmin & phone app
Canyon (Grail: ON)	US	5799	16.7	500	85	105	45	600	Mid		X	X	X						
RadMission 1	US	999	22.7	504	50	72	32	500	Hub		X	X							Has optional throttle
Trek (Verve+ 3)	US	3250	24.69	500	50	120	32	415	Mid		X	X	X						
Aventon Pace 500	US	1399	22.58	556.8	50	97	45	750	Hub		X	X	X						Has optional throttle

*Max speeds depend on country limits.

**Cannot find actual weight. Wheels weigh around 3 kg and an average bike is 9 kg.

***This is the top speed in the US. Top speed in the EU is still 25 km/hr.

Table 2: Table of Alternate Solutions

Product	Price ($)	Motor Assist	Materials	Other features
Mercedes Benz eBike	4,500	-User selects amount of assist by pressing buttons on the handle; -4 levels of assist -Mid motor	-Frame, fork: Aluminum 6061	-Lithium-ion battery, 423 Wh—last up to 100 km
Fly Kly	1,971	-Keep pedaling until you have accelerated to your ideal speed (motor will kick in as soon as you pedal). Pedal backwards three times when you have reached your speed and the motor will maintain speed -Hub motor		-100 km battery life—assists for up to 25 km/hr; recharges in 2 hours -has an app—can program desired speed and distance, and monitor travel time and battery -Also has a smart wheel you can buy
Econic One	2,656	-Motor: Bafang, Rear Hub—5 levels of pedal assist	-Frame: Aluminum 6061 -Rims: Aluminum	-app that lets you lock your bike from your phone, track your bike's location -Removable battery (with one click)

Fazua (urban, Canyon)	3,811	-Energy system is integrated in the frame and easily removed; -3 modes—Breeze: constant 100W tailwind; River: the harder you pedal, the more assistance you get; Powerful assistance for steep climbs -Mid motor	-Frame: Aluminum	-Phone app and Garmin app with different profiles per rider -Real-time overview of your performance and bike performance -Removable battery
Canyon (gravel bike, Grail:ON)	5,799	-4 assist modes—300% boost in pedal power; 200% boost, "adapts to effort level"; 120%, for long distance biking; 55%, for multiple day biking -senses terrain and will adjust modes based on that -Mid motor	-Frame: Carbon	-Up to 55.9 (90 km) miles per charge in eco mode -Have a Bosch webapp to help you plan your journey ahead of time (e.g. ensure battery doesn't die during journey). App does not directly connect with your bike. Removable battery
Radmission 1	999	-Single-speed drivetrain—increases power output as pedal power increases -Hub motor -Ability to activate throttle when assist isn't enough (e.g. hills)	-Rims: Aluminum	-Around 45 miles (72 km) max on one battery life. -No app connectivity, but has an LED control panel -Built-in brake light
Trek (Verve+ 3)	3,250	-Bosch active line plus motor—mid motor -4 Levels of assist—40%, 100%, 180%, 270%	-Frame: Aluminum Fork: Steel	-Removable battery -Ability to add "Range Boost" attachment—calculate range with Bosch webapp (see Canyon "Other Features") -Built-in fenders, rear rack, front and rear lights
Aventon (Pace 500)	1,399	-Hub motor -Ability to activate throttle -5 levels of pedal assist -Continuous power of 500W, peak 700W	-Frame, fork: Aluminum 6061 -Rim: Aluminum 36H	-40 miles average range -Bike speed limit can be adjusted based on the area

After conducting research on e-bike alternatives specifically, a deeper dive into biometric feedback devices and other devices more similar to the intended final product was conducted, as shown in Table 3.

Table 3: Table of Competitive Products

Picture	Product	Price	Purpose	Features	Notes
	Garmin Edge Bike Computer	$714.99	Displays bike stats while riding	GPS Heart rate sensor Cadence sensing	Does not control the bike
	CAROL Exercise Bike	$2395	AI controlled exercise bike	AI controlled resistance	Stationary bike
	Calamus Ultrabike One 25	$4213	Advanced e-bike, uses sensors for user experience	Cadence sensor GPS Bike alarm	No biometric sensors

From this research, the gap in the current market was found to be a rideable e-bike that is controlled by biometric feedback, since we found examples of each of the components, but not necessarily integrated. This is because with added functionality of SmartWorks in our product, more technologies and functions are able to be integrated into the bike system.

Relevant Patents and Other Competitive Products

- US8562488B2 - “Systems and Methods for Improving Motor Function with Assisted Exercise”
    - Exercise bike that alters the motor contribution based on sensor data (see *Figure 3)*
    - Sensors measure speed/cadence, torque from the patient, and torque/power from the motor.
    - Meant to improve motor function in a patient exhibiting abnormal motor function.
    - Computes a patient score based on the intensity of the exercise and patient contribution (see *Figure 4)*

Takeaway: From our patent research, this is the patent that most closely resembles our project. This patent’s control system and functional flow diagram were used as resources as we developed our own control system.

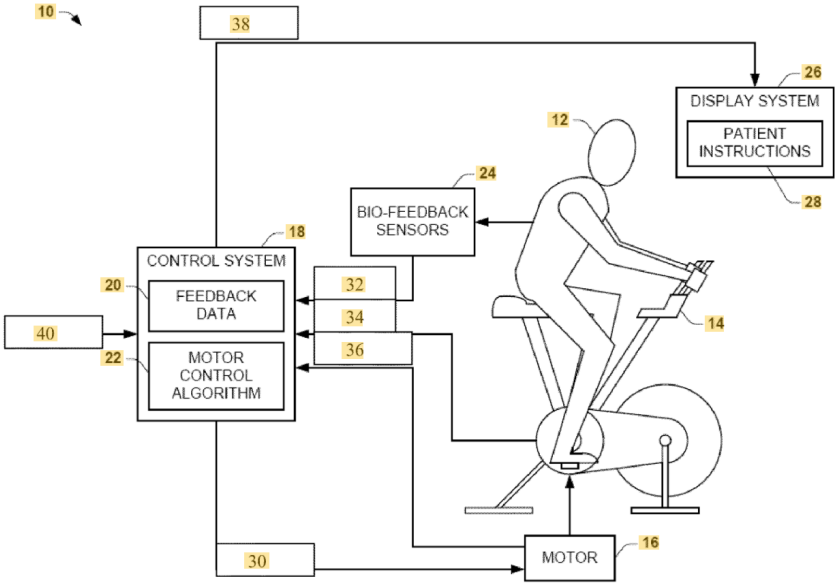


Figure 3: Basic systems of patent US8562488B2

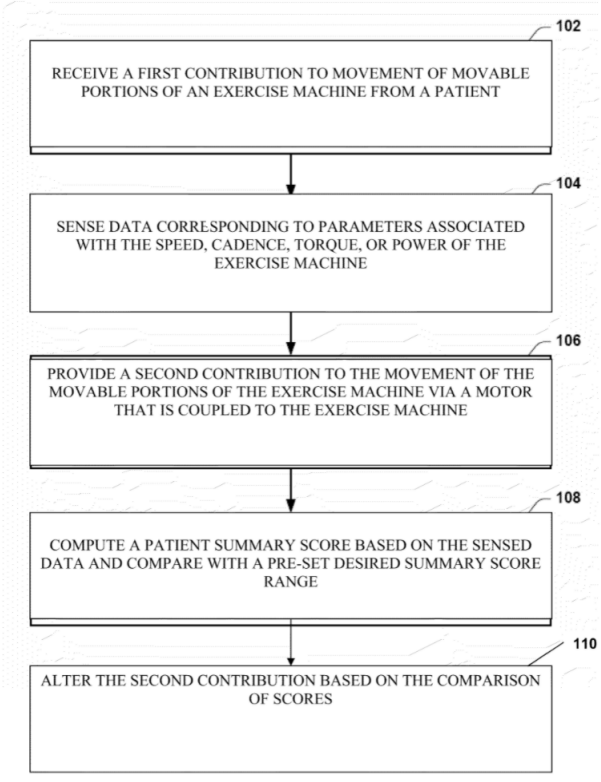


Figure 4: Section of function flow diagram for patent US8562488B2

- Wearable Heart Rate Monitor: US8998815B2
    - Uses heartbeat waveform detector and motion detector to analyze and determine heart rate by comparing outputs from both detectors
    - Device is placed around the wrist with a band containing detectors (see *Figure 5*).
    - Presents output data on the device itself

Takeaway: This product is very helpful for the consideration and implementation of a heart rate monitor on the bike, even though a chest strap heart rate monitor was ultimately selected. This patent has also been crucial in understanding how heart rate monitors work.

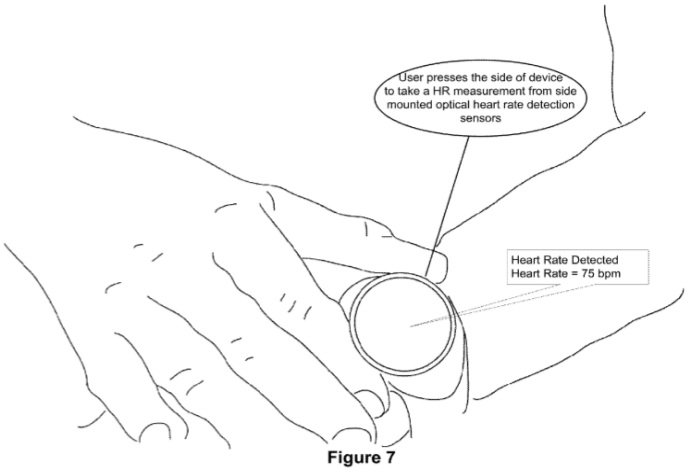


Figure 5: Illustrates how the monitor would be used and example output

- Electric bike with personal digital assistant: US20090181826A1
    - Electrical bicycle that collects user's biometric data to adjust the following mechanisms: gear shift, RPM, speed, acceleration
    - Allows the user to determine heart rate level before ride and bicycle will adjust to achieve and maintain desired heart rate
    - Has GPS tracking systems as an anti-theft measure
    - Exchanges data with bike and information processing unit to adjust the electrical bicycle settings (see *Figure 6*)

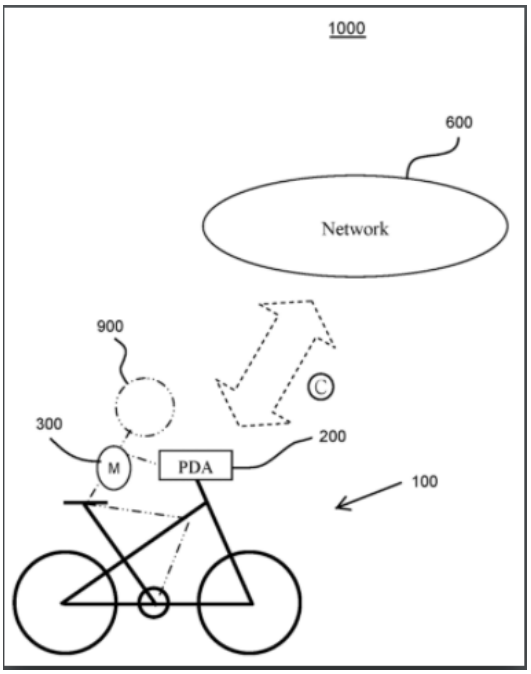


Figure 6: smart e-bike network data exchange

Takeaway: This product is similar to our end-goal product, and allows us to see how e-bikes currently interface with the cloud.

- Wearable device for the arm with close-fitting biometric sensors: US10627861B2
    - utilizes spectroscopic sensors to get the following biometric data: glucose levels, oxygen levels, hydration, or heart rate
        - uses 2 spectroscopic sensors: first sensor and second sensor are 10 degrees apart and both project light into the user's arm tissue (see *Figure 7*)
        - by doing so, it analyzes the data to make conclusions regarding biometric data
    - aims to create a sensor that can conform and adapt to varying body types and shapes
    - spectroscopic sensors: uses light and analysis of electromagnetic radiation to determine molecular composition and more

Takeaway: This product not only identified important types of biometric data to monitor, but helped in determining alternative ways of mounting a sensor, which will be an important consideration should the team not place the sensors directly on the bike.

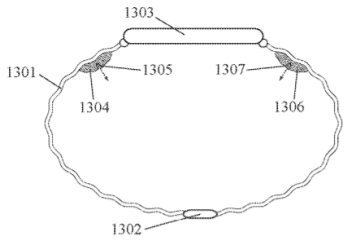


Figure 7: Location of spectroscopy sensors in the wristband sensor

- "Connected Bike-smart IoT-based Cycling Training Solution" - Department of Automation at the Technical University of Cluj-Napoca
    - No research focusing on IoT and smart bike fusion has been published (article is from February 2020)
    - The system is capable of changing gears, adapting the motor's speed and monitoring the torque, all while gathering environmental data and sending it to a server through a mobile phone connected to the internet.
    - Three layers: hardware, application, and network (see *Figure 8*).
    - Hardware side consists of two modules that both have an Arduino microcontroller with separate power banks; they have wireless communication between each other (see *Figure 9*).
        - The Pedal Module reads data from a load cell and sends it to the Bike Frame Module using infrared communication.

- The Bike Frame Module uses sensors to determine speed and cadence. Its microcontroller performs data manipulation, converts it to the proper data type, and sends it to the Raspberry Pi via USB.

Takeaway: This extremely thorough and well-documented paper was a valuable resource as we built the hardware side of our control system. For example, it contains both high-level diagrams of how the hardware can be separated into different "modules" and low-level microcontroller diagrams that can be used as a reference. Beyond this, it was utilized to determine how to connect the hardware side to SmartWorks IoT.

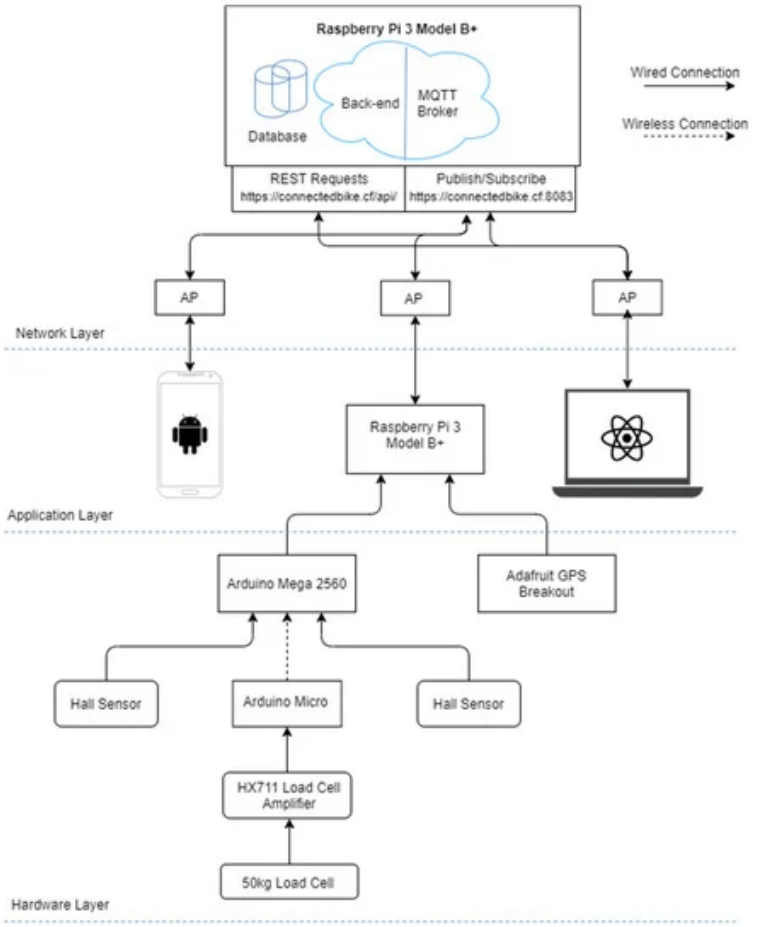


Figure 8: Architecture of connected bike system

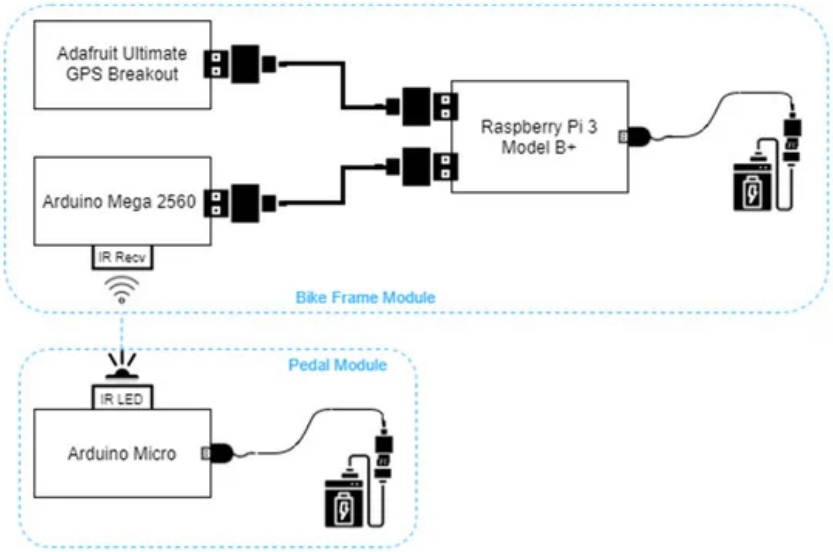


Figure 9: Hardware scheme of Connected bike smart

Final Needs, Requirements, and Specifications

Our team decided on the following needs for our design in *Table 4*. These needs were then utilized to describe what metrics and specifications were needed in order to test if these needs were being satisfied. These metrics and specifications can be found in *Table 5*.

Table 4: Table of Needs

#	Need		Imp. *(1-5, 5 being most important)*
1	The sensors	Process data accurately	3
2	The display	Is legible/understandable	3
3	The bike systems	Are easy to set-up	3
4	The user interface (website)	Displays customizable data	5
5	The motor	Makes quick auto-adjustments	4
6	The software setup	Can interface, collect, and send data from multiple sensors	5
7	The software & sensors	are compatible with a microcontroller	5
8	The bike	Operates safely	5
9	The bike	Has a long product life cycle	1

10	The bike	Is easy to store and transport	1
11	The bike assembly	Is durable	1
12	The bike system	Can be controlled from the website	5

Table 5: Table of Metrics and Specifications

Metric #	Need #(s)	Metric	Units	Marginal Spec	Target Spec	Final Spec	Rationale
1	1	Accuracy of HR sensor relative to Apple Watch	Boolean	Yes	n/a	Yes	The most efficient way of verifying the HR sensor accuracy is by comparing it to proven products on the market.
2	2	Minimum Font Size on the Display	Pixels (1/96 inch)	16	16	32	Font size should be legible to most older riders. This will also be customizable to accommodate those with even worse vision. While there is no minimum font size recommended by the ADA, 16px is the minimum recommended.[5]
3	3	Total labor time	Hours (Hr)	72	24	55	Should Altair choose to implement this as a design challenge in the future, development time should not take longer than a weekend.
4	3	Time spent preparing to ride	Seconds (s)	5	0	0	The only step to start should be to start pedaling — based on experiential research (Appendix B).
5	3,13	Time spent setting up user preferences	Seconds (s)	30	10	8	Based on Divvy/Lyft e-bike experiential research
6	4	The website displays data (assist level, heart rate, power, and mode)	Boolean	Yes	n/a	Yes	This was the data that users most wanted to display according to our interviews. All four data types are also easy to track.
7	4,6,13	Internet Connectivity	Boolean	Yes	n/a	Yes	Necessary for SmartWorks
8	5	Time between	seconds	1	0.5	0.4	Since this time only involves electrically

[5] "Typography: Visual design: Accessibility for teams," Visual design | Accessibility for Teams. [Online]. Available: https://accessibility.digital.gov/visual-design/typography/. [Accessed: 09-Dec-2021].

		data collection and motor adjustments					signaling to the motor to change its power, it should take less than one second.
9	6	Microcontroller to microcontroller communication	Boolean	Yes	n/a	Yes	For multiple sensors to be used, communication between multiple microcontrollers is required
10	6,7	nRF Connect App Compatibility	Boolean	Yes	n/a	Yes	nRF Connect is an app that extracts Bluetooth data. If the app is able to extract data from a sensor, a microcontroller will be able to, too.
11	7	Microcontroller - Motor controller compatibility	Boolean	Yes	n/a	Yes	Needed so that motor control can be achieved with most DIY e-bike kits, not just the one the team ordered.
12	8	Maximum Power	Watts (W)	750	750	750	Based on U.S. regulations
13	8	Maximum Speed	Miles per hour (mph)	28	28	28	Based on U.S. regulations
14	9	Distance before design failure (product lifetime)	Miles (mi)	3,000	10,000	TBD	Based on interview with Ernst, suggesting the lifetime of the last part of the original bicycle as design distance before failure.
15	9	# of times a battery can be charged before replacement needed	Battery charges	1000	1100	2000	This range is the typical battery lifetime for an e-bike battery.
16	10	Weight	Pounds (lbs)	46.3	38.4	TBD	Based on values that are derived from benchmarking. The marginal spec is the median and the target is the 25th percentile.
17	10	Dimensions	Cubic feet (ft^3)	44	40	33.5	71x25x43in is the average bike length/width/height.
18	11	Weight Capacity	Pounds (lbs)	275	300	TBD	The marginal spec is the typical lower limit for a bike and the target spec is the typical upper limit.
19	12	Ability to make a new ride from the display	Boolean	Yes	n/a	Yes	This will allow different users to view their own unique data
20	12	Ability to	Boolean	Yes	n/a	Yes	This will allow the user to switch modes

		switch between different modes from the display					quickly
21	4	Mobile operating system compatibility	Boolean Sum	1	3	1	Android, IOS, and Windows are the three relevant phone operating systems. Each compatible operating system yields a 1, leading to a sum of 3.

Project Architecture

The diagrams below organize the prototype functional requirements into categories and show clusters of functions that are related. In the clustered functional diagram (Figure 10), the functions that are tied to physical components (batteries, motor) are on the left side of the diagram (in green), and the cloud-based functions are on the right side of the diagram (in blue). The center clusters involve both hardware (microcontrollers, sensors) and involve interacting with the cloud or other devices via some wireless connection (Bluetooth, Wifi, etc.).

We have identified six main "clusters" of functions and their corresponding solution:

- Batteries - The batteries serve to provide power and convert stored chemical power into electrical power that may be used by the motor, microcontroller(s), and sensors. This energy flow is indicated in the diagram.
- Microcontroller (ESP32s)- The microcontroller serves to compile and analyze data, and control motor assist according to data. The controller accepts data from sensors via Bluetooth signals, sends wired analog signals to the motor controller to control the motor, and uses a WiFi connection to send data to the cloud.
- Motor - The electric motor is used to provide mechanical assistance to the rider and address the propulsion function.
- Sensors - The sensors collect and gather biometric data from the user and then send the data to a microcontroller via Bluetooth so that the data may be processed and displayed.
- Smartworks - This software manages IoT devices, stores data, creates visualizations for it, and performs functions as needed.
- Website - The Web App provides a user interface for the user and displays data from the ride

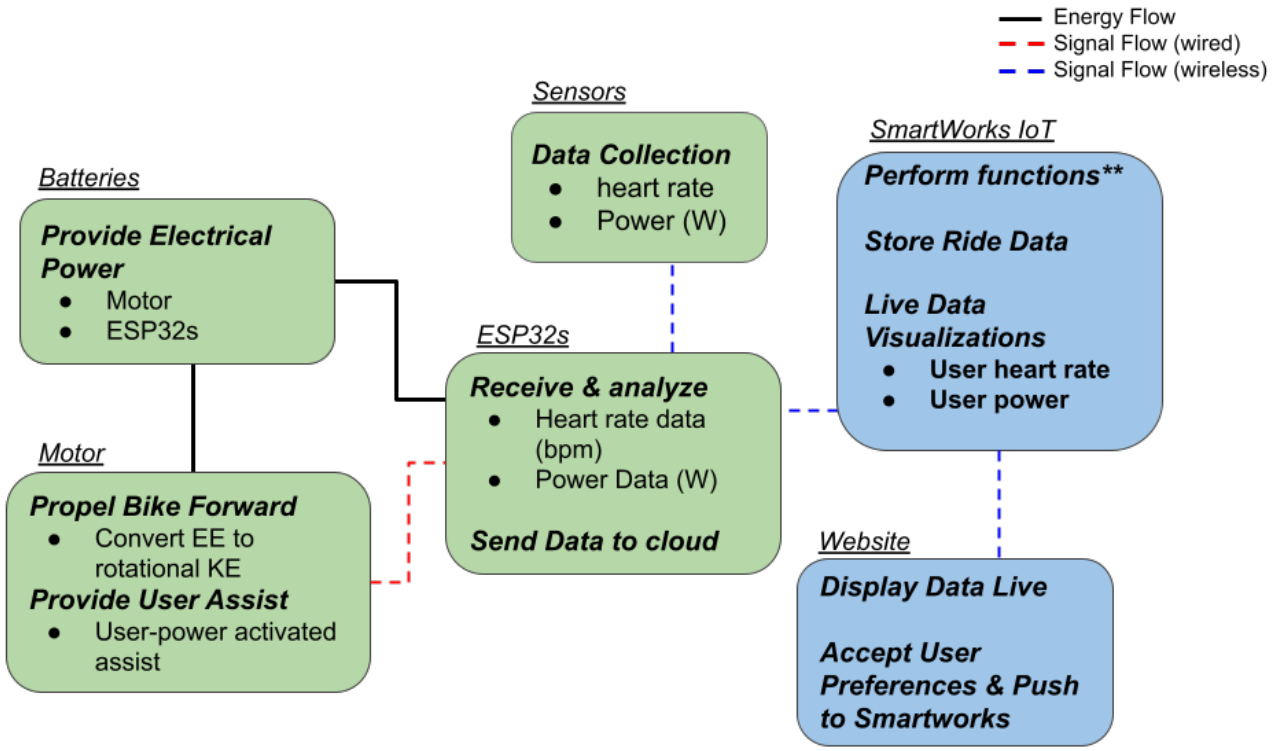


Figure 10: Product Architecture Diagram

The system uses a lithium battery to power the motor and other batteries to power the sensors and microcontroller on the bicycle. The microcontroller serves to compile and analyze data from sensors. The microcontroller receives wireless signals and then transmits wired signals to the motor to specify the motor assist levels that are deemed appropriate according to the collected data.

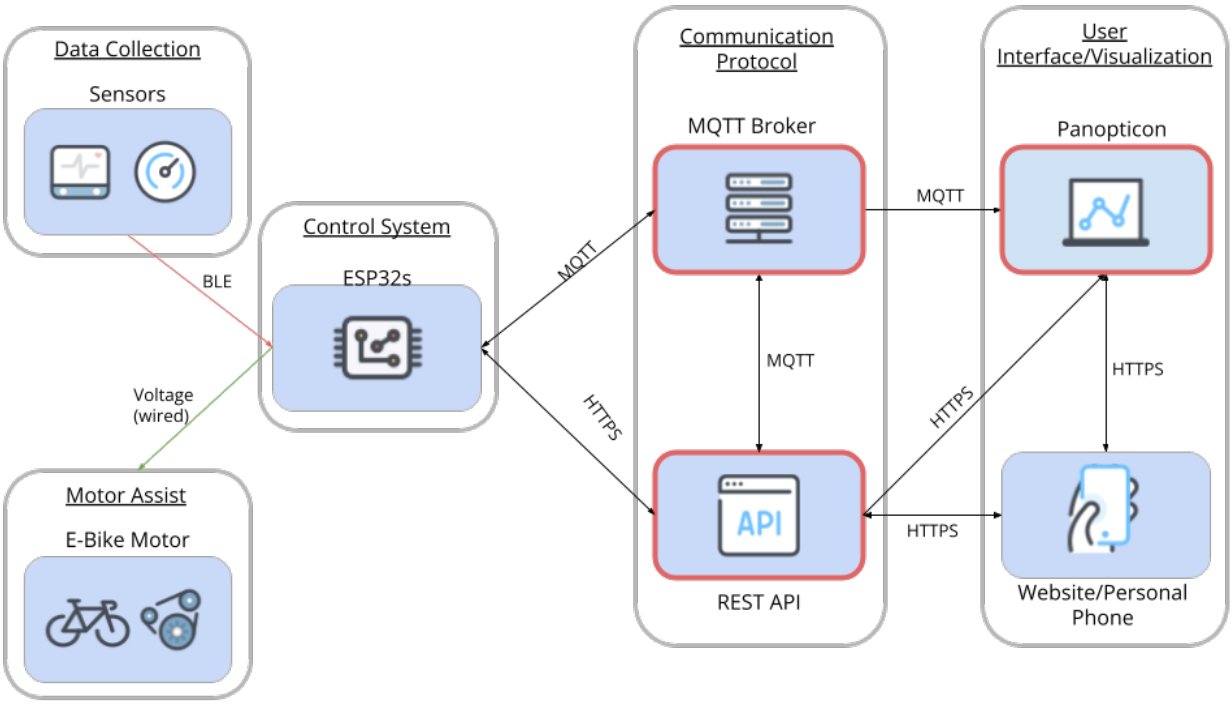


Figure 11: Software & Electronic Architecture

Figure 11 depicts the software and electronic architecture in greater detail. The left side of the diagram is the hardware involved in the project (sensors, motor, and microcontroller). The right side of the diagram displays the components of the software and the communication used to send and receive signals to and from the hardware. The heart rate sensor and power meter both send user biometrics to the ESP32 Feather using Low-Energy Bluetooth signals. The motor adjusts its output based on inputs from the motor controller which are sent by the ESP32 via a wired voltage connection.

In our design we are using a phone to act as a user interface. A website loaded onto the phone is capable of accepting user preferences which it then sends to SmartWorks to adjust motor support. The phone also functions as a display and provides the user with information on their motor assist level, heart rate, and input power.

Overall Product Design

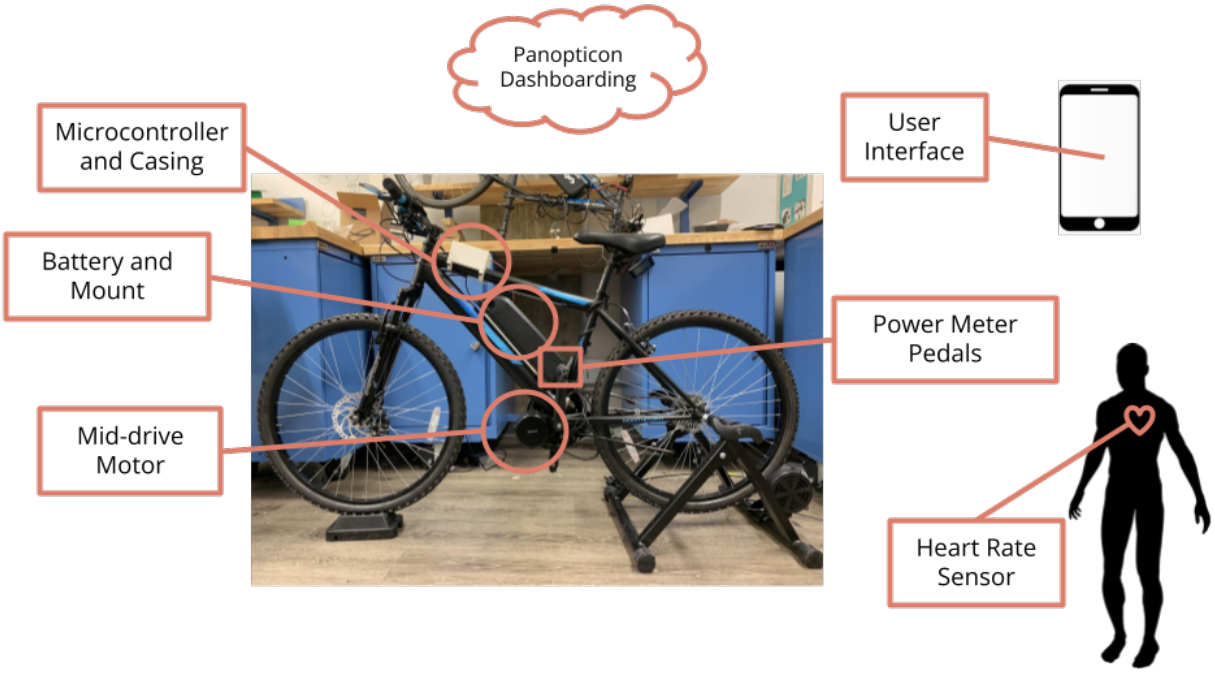


Figure 12: E-bike Design Overview

This e-bike system utilizes a mid-drive motor, a lithium ion battery, two ESP32 Feather Huzzahs, a CooSpo heart rate sensor, and Favero Assioma power meter pedals. Initially, we made many design decisions to reflect the initial human-centered aspects of the rehabilitatory e-bike project. The battery is mounted on the down tube to maintain a low center of gravity on the bike, which is important for the balance and the "ease-of-use" need for the prototype. The mid-drive motor is positioned adjacent to the crank shafts, which allows the bike to maintain a lower center of gravity, and as a result, makes the bike easier to handle. The power meter pedals can quickly replace normal pedals on the bike, and the rider will wear the CooSpo heart rate sensor chest strap as they ride.

The smart e-bike is a framework surrounding three different feedback-motor control loops (see Figure 12) . These loops function by collecting data from the rider and analyzing this data according to predetermined algorithms, resulting in a customized motor support level (see Figure 13). This live, continuous feedback loop is highlighted in the orange-dashed box. This is primarily an electronic and software based system that coordinates the data collection and transmission and the resultant mechanical response of the motor. This critical function is implemented continuously while the user is riding the bicycle in real-time. Beyond this, data is continuously uploaded and displayed using SmartWorks IoT's Panopticon Data Visualization packages that are being exported to a third-party website of our creation.

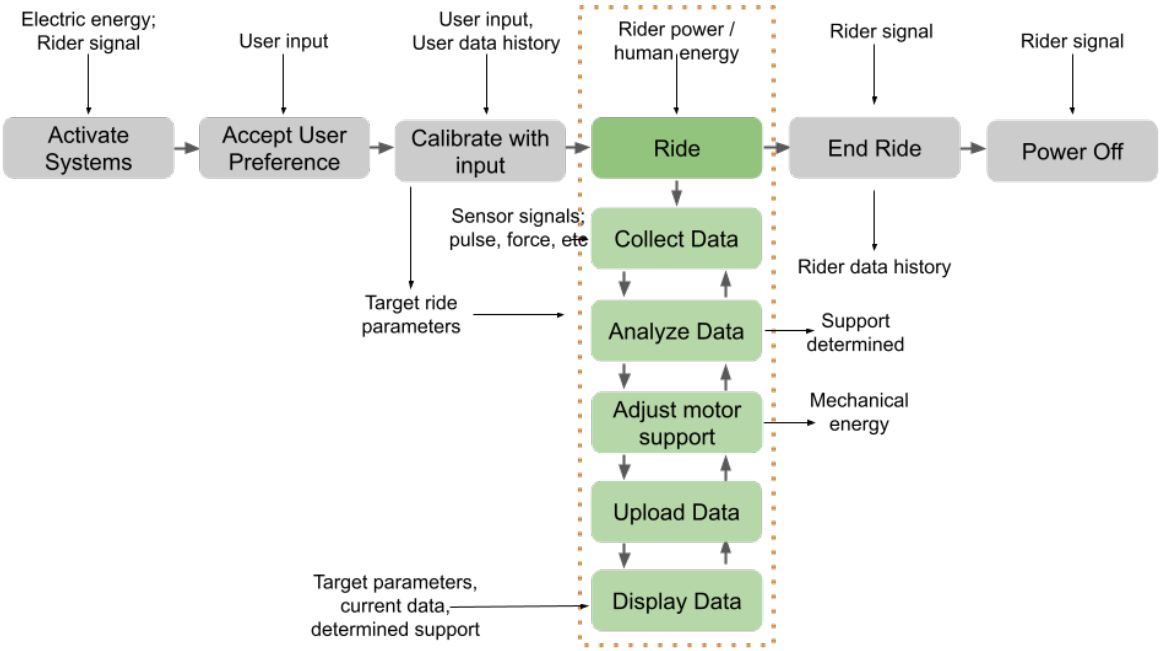


Figure 13: Functional Flow Diagram

The architecture diagram in Figure 10 details, in part, how the subsystems interact. The diagram below (Figure 14) showcases the signals of the electronics and software components which enable cloud connection and motor control.

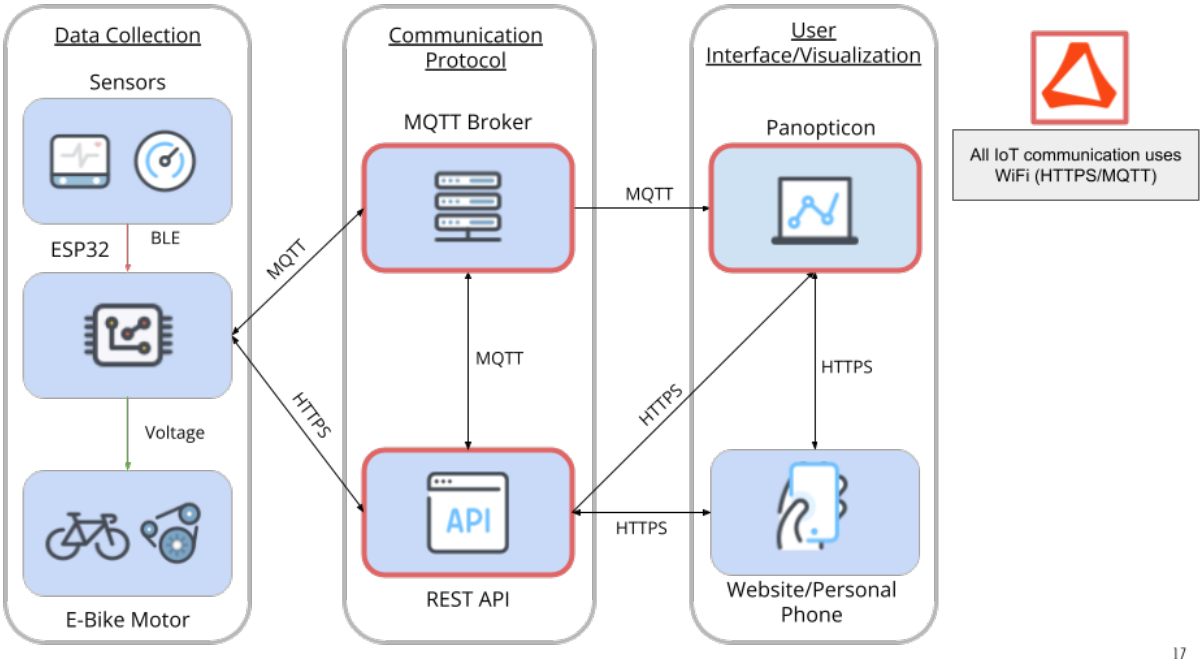


Figure 14: Data Transfer journey map and key components

We have developed two different "modes" which a user can select: heart rate and constant power. The heart rate loop functions such that as the rider's heart rate increases and reaches certain thresholds, the motor responds by increasing the motor assist in order to avoid cardiovascular overexertion. The second feedback loop driven by the user's power input is implemented via a power-meter sensor in the bike pedals. In this second feedback loop, the total output power of the bike remains constant so that as the user's power input increases, the motor's assist level linearly decreases and as the user inputs less power the motor compensates by increasing its power output.

The team also developed and analyzed other bike subsystems including the bike frame, electronics casings and mountings, and the cloud connection to send and receive data from Panopticon. Design decisions, engineering analysis, testing, and future work pertaining to each of these subsystems will be discussed in this report.

GSEE Considerations

As a team we considered the Global, Societal, Environmental, and Economic implications of taking the smart e-bike product to market.

Introduction

Each engineer brings their own viewpoints, experiences, and biases to a design project. Each engineer has their own ideas about what the final product should look like and who the final product should be designed for. In order to eliminate these biases and design ethically for society, one must consider global, societal, environmental, and economic factors to evaluate public perception that the product may receive.

There are countless examples of products that have failed to consider the state of the world throughout the product development lifespan. One classic example of a global factor leading to poor product performance is the Chevy Nova. The claim is that the car sold poorly in Spanish speaking countries because the name translates to "doesn't go" in Spanish. This is a plausible explanation, until you realize that the word "nova" is already defined in the Spanish language, meaning the same thing in English. This assumption is the equivalent of thinking that English speakers might interpret the word "notable" as "no table." In fact, General Motors executives were aware of the translation and decided to retain the name Nova since they believed it to be unimportant.[6] Intricacies like this one need to be considered early on in the product development process in order to ensure launch success.

[6] Snopes

Global Considerations

The team's e-bike will appeal to many demographics across the world, especially since the client, Altair, is a global software company based in Germany and the United States. E-bikes are becoming increasingly popular in Europe, especially in cities that have smaller roads than are typically found in the US. This could lead to our e-bike being more frequently used in Europe due to a lack of cycling infrastructure in US cities. It should be noted that US cities are beginning to take an interest in investing in their cycling infrastructure with Evanston, IL adding green, curbside bike lanes along various routes.

Should the e-bike be marketed in countries other than the US, differences in local regulations that apply to e-bikes will need to be considered. For example, European regulations on e-bikes mandate that they not exceed 250W power, and 25 km/hour (15.5 mph) speed, which is significantly lower than the allowances of 750W and 28 mph in the US. In Australia, the maximum allowable power output is 600W[7]. In cities like Copenhagen, Denmark, the number of electric bikes and scooters on the streets is limited to 3000 at any given time, which is another global factor to consider. As e-bikes continue to gain popularity, there will inevitably be more legislation that may restrict who and where the e-bike may be used. Recently, in Copenhagen, 28mph e-bikes were reclassified as bikes from the small motorbike classification, which allows them the use of bike lanes.[8]

Societal Considerations

Societal considerations for this smart e-bike stem from the location and demographics of our target user base. In certain states, e-bikes require licenses, including in New Jersey and West Virginia. Also, in New York the maximum speed of an e-bike is 25mph, rather than 28mph. Due to differences in regulations across states, this product will not be legal in certain states unless the e-bike includes user customization to limit the power output.

Another societal consideration is the regulation and general view of e-bike use on bike paths and trails. Depending on the region, e-bikes of various classes are not permitted on trails in parks for pedestrian safety, and in other regions they are not allowed on bike paths for cycler safety. In these regions, this could be a huge drawback to making an e-bike purchase, considering the use of bike paths is making biking much more accessible.

Our target user group, individuals in rehabilitation and older adults, is a large factor that contributes to how the product will be received. E-bikes are more expensive than the average mechanical bike, and older individuals usually have more of an ability to spend over a younger population. Currently, the population of various countries, including the US, is aging. This

[7] e-Bike Laws in Europe
[8] Danish E-Bike Laws

makes a larger percentage of the population part of the target demographic for this product. In 2019, people over the age of 65 made up 16% of the US population[9], and this number is expected to continue to grow in the coming years. For older adults and people with health issues that prevent them from traditional biking, an e-bike assist will allow more people worldwide to live a healthier lifestyle; one without additional health issues or pain, through the exacerbation of current health issues.

Environmental Considerations

There are both positive and negative environmental considerations surrounding this smart e-bike design. Due to the nature of electric bicycles, they can be used as an alternative to cars or other "non-green" forms of transportation. Our e-bike is not specifically designed for commuters, however, our target user group could indeed use it to commute to work or travel to other places. Since the e-bike can travel faster than a mechanical bicycle with less energy provided by the user, our e-bike allows a larger group of people to consider removing cars from their daily commute process. This could reduce car traffic in an area, as well as reduce carbon emissions from the reduction of the number of cars on the street.

Currently, there are a number of recyclable materials that will be included in the prototype, allowing for the bike to be recycled at the end of product life. For example, the steel bike frame and aluminum controller casing are able to be recycled easily. The other electronics casings may be 3D printed out of ABS plastic, which as a thermoplastic, can be melted down and reused when it is no longer needed as a component of this product.

It should be understood that the electric bicycle is not a perfect solution to environmental problems. From the materials used in the design currently, there are a number of negative environmental impacts around their production. For example, the e-bike currently uses a lithium battery. Lithium mining causes many environmental issues such as chemical leaks from mines, wasting water to process the ore, and the inability to recycle the batteries at the end of the product's life[10]. These issues are also evident from the use of other electronics on the prototype, since the circuit boards and other components contain rare earth metals that pose similar problems to lithium. Since the battery for the first prototype was reused from another e-bike, this has a slightly smaller impact, due to not needing to purchase a brand new battery for each prototype.

Although electric power is generally considered cleaner than using fossil fuels, this may not be true depending on the location the product is being used in. In the US, fossil fuels currently

[9] Administration for Community Living
[10] Institute for Energy Research

account for 60.6% of all electricity generated[11], which means that the repeated use of electricity to charge the bike could increase use of unclean energy until the US pivots to more clean energy as a whole.

Economic Considerations

As with many electronic products, there will be some economic considerations related to current supply chain issues. With the growing popularity of electric vehicles (including e-bikes), the cost of lithium batteries has increased, and will likely increase even further. Also, things like shipping costs and material costs will increase from the supply chain issues stemming from the COVID-19 pandemic, especially for the microcontroller parts.

From a product sales and marketing perspective, this product could generate a large amount of revenue, considering the current state of the market. According to John Surico of the NYTimes, the pandemic bike boom boosted e-bike sales 145 percent from 2019 to 2020, more than double the rate of classic bikes[12]. As a result, the demand for this product is projected to be high. Also, the incorporation of IoT (Internet of Things) technology will increase the interest in the product, as IoT is a multibillion dollar industry that is rapidly growing. The release of this product could introduce IoT to consumers since the target market may be unaware of how IoT technology can be beneficial.

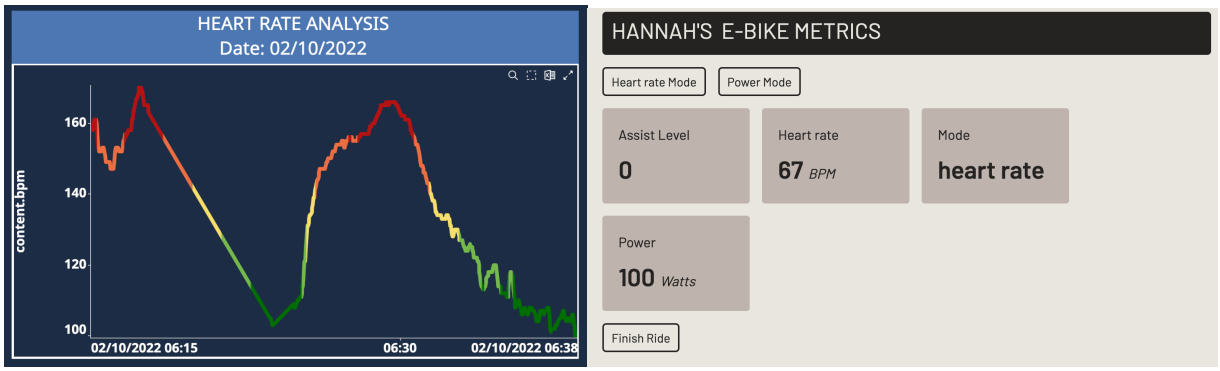
Another economic consideration is the time of year the product will be released. Due to cycling being an outdoor activity, and the bike having electronic parts, it will be best suited for use in the warmer seasons, especially in colder climates. This will allow the manufacturer to more accurately predict the demand, and therefore the sales, of the product throughout the year.

Design Approach & Solutions

The final prototype consists of two microcontrollers that gather heart rate and user power input data. Based on these biometrics, the microcontroller calculates an assist level, which controls the power output of the motor. The biometric data and assist level are sent to SmartWorks for Panopticon visualizations. From SmartWorks, data is also sent to a web app through HTTP protocol, providing a phone display for the user during their ride, and a method for the user to choose the method of assist and create a new ride for a given user.

[11] US Energy Information Administration
[12] New York Times

(a)



(b) (c)

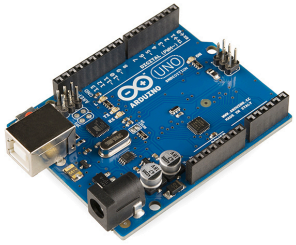
Figure 15: Final (a) bike prototype with (b) SmartWorks and (c) web app.

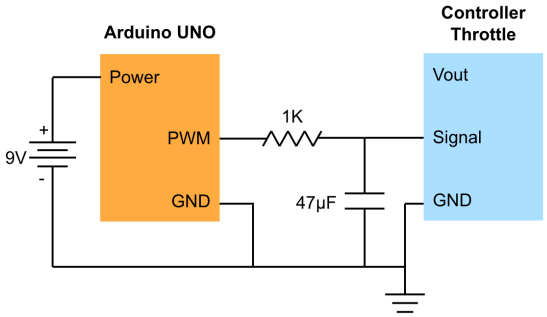
Motor Control

Many members of the team had previous experience with using the NU32 PIC32 development board to control a DC brushed motor to rotate at a certain speed or to a certain angular position. This consisted of sending a PWM wave to a brushed motor, in which adjusting the pulse width of the wave would change the speed of the motor. With that basis, the team looked at other microcontrollers that were more compact and possessed greater processing power. Through research and recommendations from Professor Marchuk, we considered the Raspberry Pi Pico and the Arduino Uno. For our critical systems prototype, we chose the Arduino because of the available documentation and coding language (see Figure 16).

The first bike motor that was controlled was a front hub motor because we were given a used bike in which the motor was already pre-installed. Unlike our test motors, the hub motor uses a brushless DC motor, which takes in a significantly more complicated signal than the brushed motor. It takes in three, precisely out of phase, PWM waves. However, the ebike had a throttle input, which acts as a potentiometer that sends a signal between 0V and 5V, which is processed

in the ebike's motor controller and sent to the hub motor. Thus, we changed our approach of connecting the microcontroller directly to the motor's input signals and instead connected it to the throttle input of the bike. Because the Arduino Uno does not have the ability for an analog output, we had to smooth out a PWM from the Arduino with a low-pass filter, which effectively acts as an analog output with some delay.

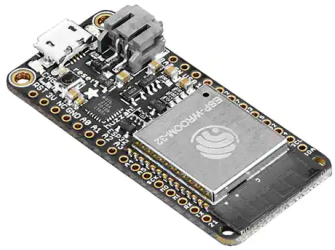


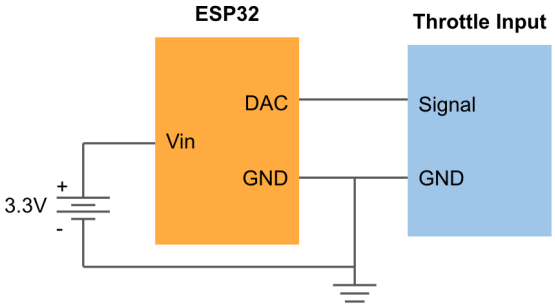


(a) (b)

Figure 16: Critical systems prototype (a) microcontroller and (b) wiring schematic.

For the final design, the team installed a mid-drive motor because of its overall benefits on weight and center of gravity. We switched the microcontroller to an ESP32 because of its compactness and ability to produce analog signals, eliminating our need for a low pass filter (see Figure 17). Our overall approach of sending the signal through the throttle input was the same.





(a) (b)

Figure 17: Final motor controller (a) physical and (b) schematic.

Sensor Incorporation

Research into different biometric sensors led the team to conclude that the only biometric sensor in the system would be a heart rate sensor. Since one of the driving factors behind this project is showcasing SmartWorks, it was important that at least one additional sensor was added, since this would increase the complexity of the data stream flowing through SmartWorks. However, it was clear that this additional sensor would not be biometric. Multiple interviews with experienced bikers indicated that power is an extremely important metric to them; therefore, research was conducted to find the power meter sensor that best fit our needs. This research, which is explained further in the "Engineering Analysis" section, led to the purchasing of pedals that act as power meters.

Both sensors communicate with the bike via Bluetooth Low Energy (BLE). Specifically, each sensor connects to a microcontroller via BLE, and the microcontroller system is then responsible for using that data to adjust the motor's power level and sending the data to SmartWorks. Additional details on how the microcontrollers were able to extract BLE data from the sensors is located below in the "Engineering Analysis" section of this report.

SmartWorks Communication

SmartWorks IoT is a cloud platform capable of communicating data between connected devices through MQTT/HTTP, conducting powerful data analysis on Panopticon, a visual dashboarding program, and hosting and executing automatic and adjustable functions. MQTT stands for MQ Telemetry Transport and is a method of communication between devices using a publish/subscribe model. A device can be subscribed to another device to retrieve data and publish/send data to another location. HTTPS is HyperText Transfer Protocol Secure and is another method of sending data and data transfer is conducted over a secure connection.

Due to the nature of the SmartWorks being developed by Altair, the team was not responsible for any direct changes of the functionalities of the software; however, the team has decided on how it will be integrated to the final product of the capstone project.

SmartWorks' general layout is that there is a "Space" where it contains a number of "Collections" and multiple "Things" within each Collection. In the starting code files that SmartWorks provided for establishing MQTT/Wifi communication between a microcontroller and SmartWorks, the code requires that the data stream's final location is manually determined by the Things' information. The issue with this method is that repeated sensor readings cannot be conducted without manually creating another Thing and changing the code again with the new information. Without creating new things with each iteration, the data is stored in the same location making it difficult to conduct data analysis when multiple sources of data are mixed in one place.

Initially, the motor control determined by heart rate was calculated by an algorithm inside the program of the microcontroller; however, the algorithm's location has shifted from being executed in the microcontroller's program and instead is hosted in SmartWorks where it can be invoked by an HTTP request via WiFi. By doing so, it allows for calculations to be completed in the cloud and allows the microcontroller to save memory by allocating the functions program in the cloud.

The heart rate sensor and the power meter pedals both have BLE (Bluetooth Low Energy) capabilities, meaning that both sensors can communicate to specific microcontrollers like the ESP32. And the ESP32 has WiFi and Bluetooth functionalities giving it the ability to

communicate with both the sensors and send data to SmartWorks. By utilizing both, the ESP32 can retrieve data from both sensors via BLE and send it to SmartWorks via MQTT where it can be stored and used for further data analysis and visualization.

The team used SmartWorks' Panopticon feature to visualize heart rate data with a basic time-series line graph. With more experience and a deeper understanding of the software's functions, the team was able to design more customizable and visually informative dashboards to convey sensor data and any relevant statistics.

Web App Client

To allow interaction between the user, SmartWorks, and the motor control, we created a web app that can be opened in the browser of any phone. Other options we considered included physical buttons with an LCD screen and programming a touch screen. However, the team decided that the web app would give us more flexibility over customization and would be the most straightforward to implement. The web app displays in-ride data to the user, and allows the user to set their name and change the assist mode between "heart rate" and "power." It was created with the React Javascript library, which is optimized for web app development, and it uses HTTP protocol to receive ride metrics and send user input. We chose to take a simple design approach to the website because initial interviewees told us that complex and crowded displays are more distracting than helpful. More description of the website can be found in Appendix E.

Design & Engineering Analysis

Microcontroller Selection

For the critical system prototype, the Arduino Uno was used. However, the addition of Bluetooth sensors to the system meant that either Bluetooth functionality would have to be added to the Arduino Uno via an external chip, or a new microcontroller with inherent Bluetooth capabilities would have to be used. After consulting with Professor Marchuk, the Adafruit Huzzah - ESP32 Feather Board, which is shown below in Figure 18, was investigated. Specifically, a comparison between the ESP32 Feather and Arduino Uno was made to determine if the ESP32 Feather had any shortcomings relative to the Arduino Uno. This alternatives matrix is shown below in Table 6.

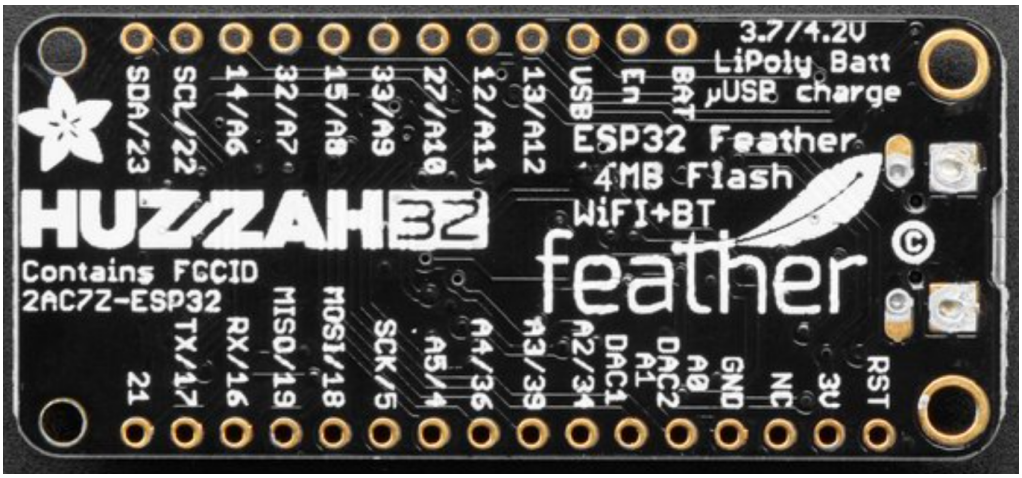


Figure 18: The AdaFruit Huzzah - ESP32 Feather Board

Table 6: Alternatives matrix of the Arduino Uno and ESP32 Feather.

Feature (1=worst, 5=best)	ESP32 Feather	Arduino Uno
Bluetooth (multi-connection)	5	2
WiFi	5	2
Programming language	4	4
SmartWorks Compatibility	5	2
DAC (Digital to Analog Converter)	5	2

- Bluetooth
    - The ESP32 Feather has Bluetooth capabilities, meaning it fully meets our Bluetooth needs.
    - The Arduino Uno does not have Bluetooth capabilities. While an external chip can be used to add these capabilities, doing so would increase the space needed to store the Arduino, add extra wiring, and require additional work to set up.
- WiFi
    - This has the same exact rationale as the Bluetooth section: the ESP32 Feather already possesses Wifi capabilities, while it would need to be added to the Arduino Uno
- Programming Language
    - Both microcontrollers use languages derived from “C++.” This is a language that the team is comfortable with.

    - While this is not captured in the alternatives matrix, it is also important to note that both microcontrollers use the same language. This will save time when converting our existing code from the Arduino Uno to the ESP32.
- SmartWorks Compatibility
    - SmartWorks has a tutorial on how to connect an ESP chip to it. This means that using the ESP32 will allow for a straightforward connection process with SmartWorks.
    - The ease of connecting the Arduino Uno to SmartWorks would depend on the WiFi chip that is used. However, it is undesirable to pursue this unknown quantity when the alternative has an existing tutorial.
- Digital to Analog Converter
    - The Arduino Uno used a low-pass filter to convert its PWM signal to analog signal. This requires extra wiring and the use of a breadboard. While the breadboard could ultimately be replaced by a PCB, this would still result in extra wiring.
    - The ESP32 features two DAC pins. This means that the DAC pin of the ESP32 can be directly wired to the motor controller.

Heart Rate Sensor

For our critical system, a heart rate sensor was selected in order to collect real-time data from the user. Our entire critical system revolves around being able to import real-time data into the ESP32 Feather and then adjust the motor assist based on the signal received. Research was completed to determine if the motor assist should be set on a continuous scale or in discrete intervals. It was found that there are five heart rate zones that are commonly used in industry (Figure 19)[13]. This can be seen in the graphic below. The team will utilize these five zones to correspond to five different motor assists. It should be noted that the user's maximum heart rate is defined for the sake of this project as 220 minus their age[14]. The team conducted brief research to determine the effect that resting heart rate has on maximum heart rate. Although it is agreed upon that a lower resting heart rate is a good indicator of cardiovascular health, there is no established relationship between resting heart rate and maximum heart rate.

[13] A. Mateo Ashley Mateo is a writer, "Heart rate training can be a smart technique to guide your intensity during training," *Runner's World*, 02-Nov-202

[14] Target heart rate and estimated maximum heart rate," *Centers for Disease Control and Prevention*, 14-Oct-2020.

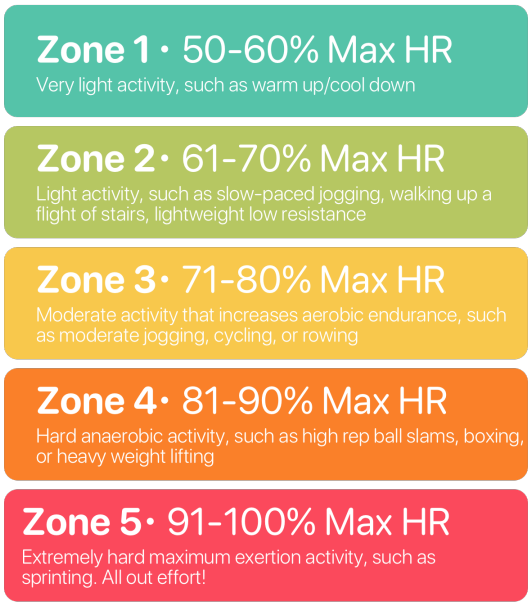


Figure 19: Heart Rate Zones[2]

The CooSpo heart rate sensor (see Figure 20) that the team purchased satisfies the requirements of the critical system of being accurate, customizable, and durable. It is usable in the environments that will be encountered on an e-bike. The manufacturer has tested the sensor to be accurate up to one beat per minute which is more than sufficient for identifying which heart rate zone the user is in. The sensor is suitable for many sizes of users and has the added ability to connect to multiple devices at once. The heart rate sensor is quite durable, boasting a battery life of 300 hours. This is more than sufficient for a user to complete dozens of rides on a single charge. In addition, the heart rate sensor is rated at IP67. IP67 equipment is the most common IPX rating found in the connectivity market. The rating means that it is 100% protected against solid objects like dust and sand, and it has been tested to work for at least 30 minutes while under 15 cm to 1 m of water.

The CooSpo Heart Rate Monitor collects data by recording the heart's electrical activity which is called an electrocardiogram (ECG). Electrodes detect the small electrical changes that are a consequence of cardiac muscle depolarization followed by repolarization during each cardiac cycle (heartbeat). The CooSpo chest strap includes two plastic electrode areas that must be moistened and held firmly against the skin in order to collect accurate data. The manufacturer lists a few failure modes as human error, chest strap damage, static electricity, and friction.

Figure 20: CooSpo Heart Rate Monitor

Because the method of extracting data from the CooSpo was unconventional, the team felt it was necessary to test the accuracy of the sensor's data output. As a result, the team investigated its accuracy by comparing the heart rate data that is available post-workout against the post-workout data from the Apple Watch 7. If the CooSpo HRM could produce accurate data that is comparable to the industry standard, then the HRM would be well suited for use in the critical system.

Both heart rate monitors allowed for post-workout data to be viewed and downloaded in an extensible markup language (XML) format. It should be noted that the Apple XML file was much larger since it included various other health information about the user. Two Google Colab notebooks were created to parse the data and display it. The Apple Watch notebook can be found here[15] and the CooSpo notebook can be found here[16].

After comparing the results, it was concluded that the CooSpo HRM was just as accurate as the Apple Watch. In addition, the CooSpo provided a heart rate reading every second, compared to the Apple Watch which only provided a heart rate reading every five seconds. Figure 21 below shows two cycling workouts: one measured by the CooSpo HRM and the other by the Apple Watch 7. Many sources report that it is common for heart rate data to be averaged over three to four data points. For this reason, the team will calculate the user's heart rate as the average of the four most recent samples from the CooSpo. This will allow for accurate motor assist within every five seconds.

[15] https://colab.research.google.com/drive/1JvHPPmKfM2f64ujKCzgCRCzSOcA2h4e_?usp=sharing

[16] https://colab.research.google.com/drive/1WwAFghIjDrWSw8cYEUNbZKrRsj3R2xoF?usp=sharing

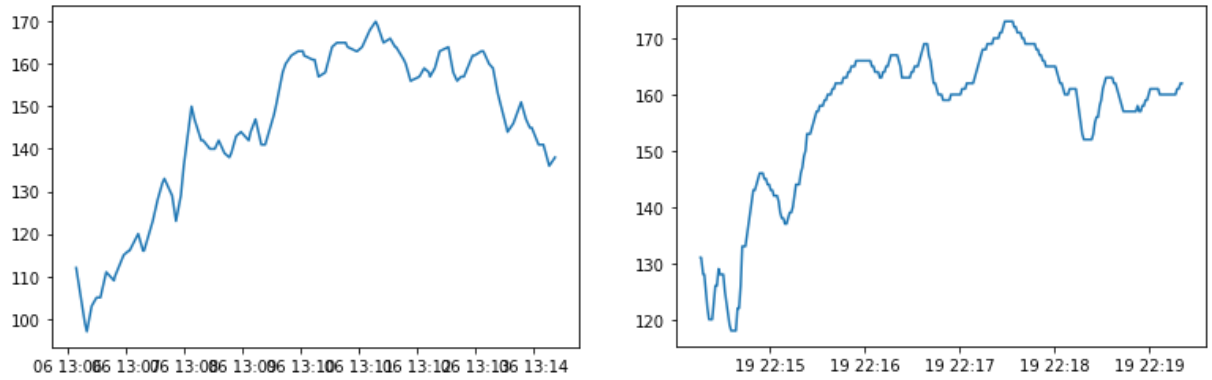


Figure 21: Cycling workouts measured by the Apple Watch 7 (left) and the CooSpo Heart Rate Monitor (right)

Further progress with the CooSpo Heart Rate Monitor included establishing a connection with a microcontroller in order to send data to SmartWorks. This progress is detailed further on in the Engineering Analysis section.

Favero Assioma Power Meter Pedals

Early interviews with cyclists indicated that power is one of the statistics that users tend to be most interested in as they ride (Appendix A). In addition, measuring power and extracting the rider's input force/torque is critical towards the function of the joint protection feedback loop, which was a major priority of the team at the time of researching different types of sensors. Different means of determining user power were explored and compared to determine which type of power meter sensor best met our needs.

Power meters can be placed on five areas of the bike: wheel hub, bottom bracket, chainring, crank arm, and pedal. Upon further research, it was found that the only manufacturer of hub-based power meters, PowerTap, recently discontinued their project.

One of the most important considerations in power meter selection is compatibility between most bikes on the market and the power meter. Power-meter research (see Appendix D) indicated that both the bottom bracket and chainring power meters have a lack of industry standardization. This makes it difficult to guarantee that the components would integrate into the existing bike frame making bottom bracket and chainring power meters unsuitable for this use.

After examining the locations of the mid-drive motor and crank arm-based power meter on the bike, the team was doubtful that the two components would be able to fit together on the bike. A pedal-based power meter, on the other hand, does not interfere with any other components on the bike frame. This conclusion led the team to focus on power meter pedals and research further.

To meet our specifications, the power meter pedal must generate accurate data, have Bluetooth communication to transmit data to the ESP32 Feather, and be compatible with a variety of devices and applications. Some of the most popular power meter pedals are only compatible with certain products. For example, the Garmin Rally RS200 Power Meter Pedals were only compatible with Garmin's applications, and as a result, would not be viable for our project.

The only power meter that met the requirements was the Favero Assioma (see Figure 25)**.** It claims to "work with your favorite cycling computer, GPS watch, smart phone, or tablet", meaning that we should be able to extract data from it on the device of our choice. The Favero Assioma has a guaranteed accuracy of +/- 1% and has Bluetooth capabilities. Overall, the product's average rating of 5 stars across 117 customer reviews leads to a high level of trust in the manufacturer's accuracy claims. While it would have been ideal to test the claims of the manufacturer, the team was unable to find a reasonable method of testing the accuracy of the pedals.

Figure 22: Favero Assioma pedal-based power meter

Exploratory Testing with the Favero Assioma Pedals:
By following the manual and using the manufacturer's app, the pedals were registered, calibrated, and made ready for use. The power meter pedals were easily installed by unscrewing the old pedals and attaching the new ones; this could easily be done by a hackathon team (even one with a limited mechanical background).

To verify the Bluetooth connection and ability to extract data from the pedals, a test ride was conducted. The power meter pedals were connected via Bluetooth to a bike computer phone app "Jepster." A summary of the ride is shown below in Figure 23. The power meter automatically transmitted the statistics of power and cadence; all other data, such as average speed, was irrelevant for the purposes of this test.

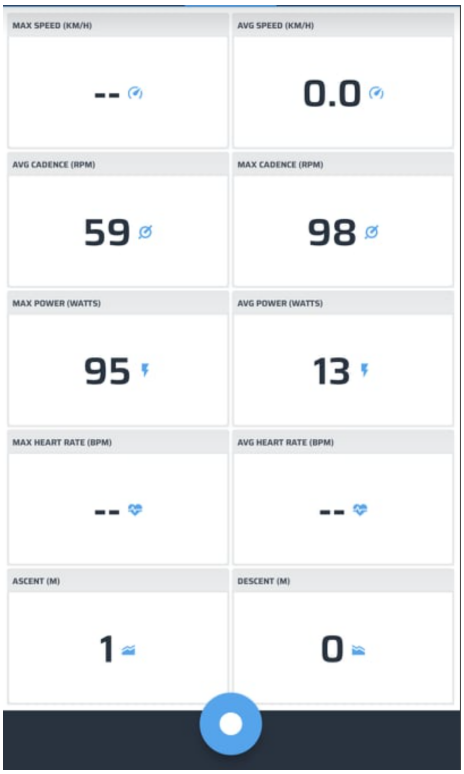


Figure 23: Summary statistics from the power meter test ride. The data includes power in Watts and cadence in RPM.

The pedals were able to export total power (in Watts) as well as the rider's cadence (in RPM). Means of extracting both of these statistics is described later on in the report.

Mid-drive Motor & Motor Controller

Figure 24: Fully installed DIY mid-drive motor kit on selected bike frame

For our final prototype, the team decided to transition to a mid-drive motor which is mounted near the crankshaft of the bicycle (see Figure 24). This change was made due to the lighter weight offered by the mid-drive motor and the change in the center of gravity when compared to a Xhub-drive motor. This makes the bike easier to ride for a user. Specifically, the mid-drive motor sits closer to being in line to the center of gravity of a mechanical bike, allowing the bike to be easier to ride for riders who are familiar with mechanical bikes.

When choosing a mid-drive motor, one essential consideration was that it would be easy to replace the motor control system that has been used for the critical system prototype. The characteristic of the motor controller used in the critical system prototype which allowed the team to control it was that it had a throttle mode. By rerouting the throttle wiring from a potentiometer within the bike handlebars to the analog output of a microcontroller, the motor output can be controlled. As long as this throttle is present, this setup can be used. Therefore, it was essential that the DIY kit featured a throttle mode.

A Bafang motor was purchased from E-Bike Essentials (Figure 25) due to greater reliability of the vendor and manufacturer which relates back to our safety specification. This motor and vendor were selected due to the online support available for installation which will allow design challenge participants to have background and instructions.

For the Bafang motor, the motor controller is incorporated within the mid-drive motor itself. This allows for a more compact and protected electronics set-up. The motor controller maintains throttle outputs and battery connections that are compatible with the selected battery.

Additionally, the pedal assist of this motor is also able to be easily disabled via cutting a wire within the motor, allowing for the team to have complete control over the level of assistance being applied to the rider via the motor. This technique was found via a Reddit post[17] by user MyEyeToTheSky and corroborated by a forum post[18] on the Endless Sphere Forums by user neptronix.

Figure 25: Bafang Mid-drive Motor

Lithium Battery

The team opted to purchase the lithium ion battery as part of the bike kit in order to ensure the compatibility between the battery, the motor controller, and motor pictured in Figure 26. The 750W motor requires either a 48 or 52V battery to be powered. When purchasing our e-bike kit there were several compatible options. The team opted for the 48V battery to better meet our cost spec because it was cheaper and there is no noticeable decrease in performance.

We tested powering the bicycle with the battery. The power and ground connections of the motor and battery were easily and safely connected through Anderson clips, and the battery was able to effectively and safely power the motor.

---

17 https://www.reddit.com/r/ebikes/comments/ogizdk/bafang_bbshd_setup_and_pedal_assist_disable_notes/

18 How to: disable PAS on Bafang BBS02 - Endless Sphere (endless-sphere.com)

Figure 26: E-bike Essential 48V Lithium Ion Battery

Battery Mount

Most bikes, including the frame selected, have built-in threaded holes on the downtube for mounting a water bottle holder. The battery has mounting locations on the bottom that align with these threaded holes. Unfortunately, the location of the water bottle mount is too low on the bike frame, and the battery cannot fit in the space. The team had the possibility to either make an adapter mount between the battery and frame or drill new holes in the current battery mount. The former solution would be more structurally sound, while the latter would be easier to perform.

In order to mount the battery to the bike frame securely, an adapter plate needed to be designed and manufactured, since the holes on the existing battery did not match up with the holes on the bike frame. Rather than use zip ties or other securing methods to mount the battery, the adapter plate was designed to screw into the existing threaded holes on the bike, in order to keep the structural integrity of the bike frame.

The adapter was manufactured out of ABS plastic, rather than PLA, in order to have more durability at the connection points, but keeping it lightweight enough that it doesn't add too much to the battery assembly.

The shape of the adapter plate (Figure 27) was iterated between a curved model and the flat plate model, and after both were installed, the flat plate model held up the most. Due to the irregular shape of the bike down tube, and the slight protrusion of the threaded holes on the frame, the curved mount hovered over the bike frame, flush only at the connection points, and the battery was tangent to the top part of the curve. When using the flat plate adapter, the battery is fully flush with the adapter plate, and the plate is mostly flush with the bike frame.

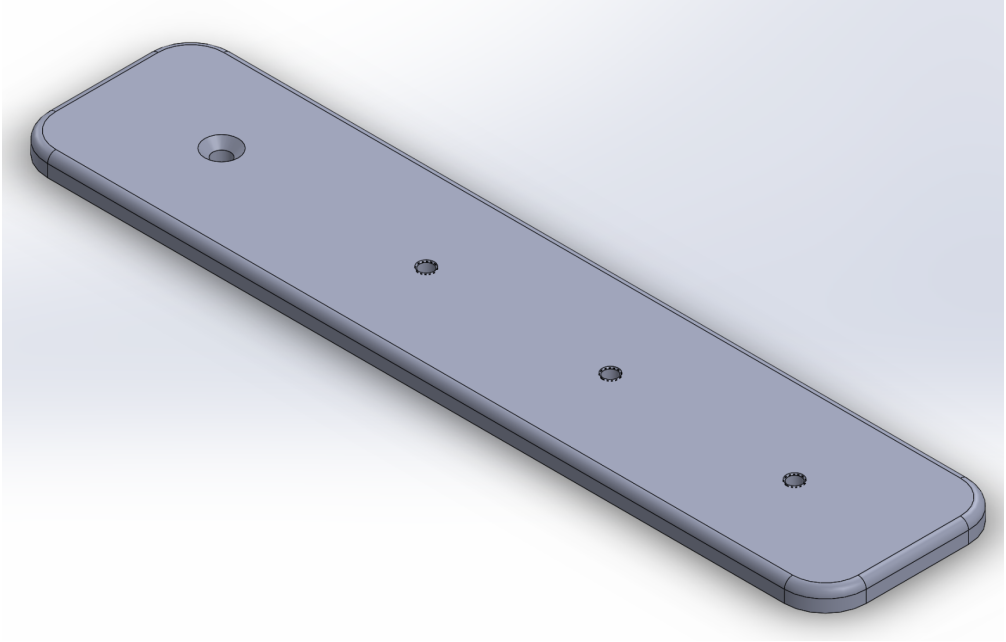


Figure 27: Top View of Battery Mount Adapter

To achieve a more lightweight part without compromising the structural integrity, the mount was designed with ribs on the bottom (Figure 28). This also allowed for a thinner part to be printed.

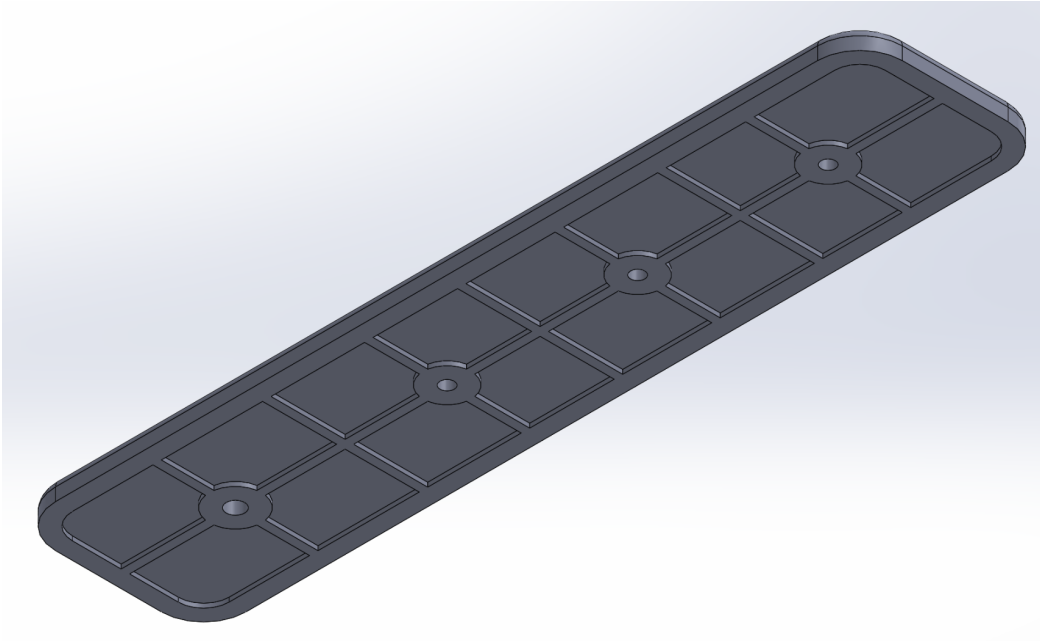


Figure 28: Bottom View of Battery Mount Adapter

In order to connect the adapter plate to the bike and the battery, four M5 bolts were used. The plate has four threaded holes to fit the M5 bolts, and the bike frame has 2. This size was chosen to allow the existing threaded holes to be utilized.

Underneath the two threaded holes that don't thread into the bike, a piece of neoprene was attached to the mount, to achieve a better fit between the adapter and the bike frame, and to reduce vibration of the battery during a ride (Figure 29).

Figure 29: Vibration Reduction on Battery Mount

Casing

To house the electronic components that are utilized in this design, the team decided on creating a casing and mounting assembly. This assembly must allow for the electronic components to maintain a close proximity to the wiring of the mid-drive motor so that they may easily access throttle connection while also allowing for protection to the electronic components stored within.

Casing Material

ABS was chosen as the material utilized in the construction of the casing and mountings due to its ease of manufacturing and light weight.

Vibration Isolation

Vibration isolation is an important consideration when designing for the need of durability, especially when considering the use of microcontrollers. For vibration, three candidate rubbers were selected: natural rubber, neoprene, and high-temperature silicone rubber. For vibration performance, neoprene and silicone rubber were compared to natural rubber. In the case of silicone rubber, silicone has poorer performance in vibration isolation[19]. In the case of neoprene, its performance is comparable to that of natural rubber[20]. Another factor that was considered was

[19] Antonios Argoudelis, "Comparison of Vibration Isolators with Silicone to Anti-Vibration Products Made of Natural Rubber Compounds," The Blog of Vibration Control Experts, May 28, 2018, https://antivibration-systems.com/vibration-isolators-silicon-vibration-control-rubber/.

[20] "The Lord Corporation Free PDF Ebooks," The Lord Corporation - Free PDF eBook, 2011, https://looksbysharon.com/the-lord-corporation.html, 17.

the weather resistance of each rubber. In terms of weather resistance, neoprene performed much better than natural rubber[21]. This led to the selection of neoprene as the vibration isolating material chosen for our design, as shown in Table 7.

Table 7: Vibration Isolating Material Alternatives Matrix

Material	**Price per 12" x 12", 3/16" thick piece**	**Vibration Isolation**	**Weather Resistance**
High-Temperature Silicone Rubber	$47.89	Worse than natural rubber	Good resistance to ozone and salt water
Neoprene	$23.13	As good as natural rubber	Higher resistance to weather, ozone, UV radiation, freons, and mild acids
Natural Rubber	$23.16	Natural rubber	Poor resistance to sunlight, weather, ozone, and high temperatures.

Heat Dissipation

Another important consideration for the need of durability is heat dissipation. For protection from overheating, standoffs were utilized to elevate the microcontrollers so that they have minimal contact with the casing, allowing for greater air flow and heat dissipation.

Fasteners

Another need that was considered in the design of the casing was the ease of setting up the design's systems. Because the motor assembly that was purchased utilizes metric bolts, with a majority of the bolts being size M5, M5 bolts were chosen to fasten the mounting to the casing on the bike frame.

Lid

To allow for easy access to the electronic components within the casing, a sliding lid was needed. This lid both protects the components inside the casing and allows for easy accesses when adjustments are needed.

[21] Brandon Tran, "Neoprene vs Natural Rubber in Applications," Coi Rubber Products, October 26, 2018, https://www.coirubber.com/neoprene-vs-natural-rubber/

With each of these design considerations in mind, the following casing design shown in Figure 30 was created:

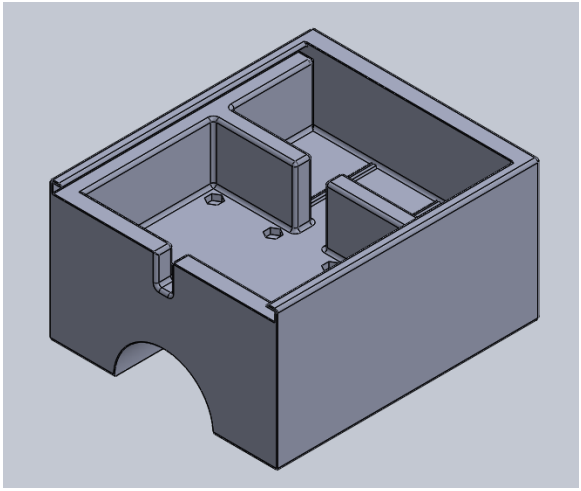


Figure 30: Electronics Casing CAD

This casing allows for the storage of up to three ESP32 microcontrollers and has a curved cut out on the bottom surface to, when covered with a layer of neoprene, snugly fit onto the top tube of the bike frame. The inside of the casing contains hexagonal cutouts to glue in the plastic standoffs that will keep the microcontrollers elevated from the casing floor in order to improve heat dissipation. On the bottom of the casing, four bolt holes were added to be tapped to the proper M5 size and threading. This allows the mounting, shown in Figure 31, to be fastened to the bottom.

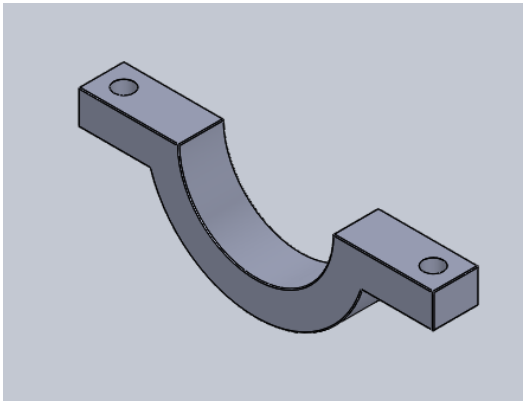


Figure 31: Mounting CAD

Like the casing, the curved surface of the mounting, when covered with a layer of neoprene, fits snuggly underneath the top tube of the bike frame.

Additionally, the lid shown in Figure 32 was constructed in order to both protect the microcontrollers and allow easy access to the inside of the casing. To make the lid easier to open and close, a grip was added at the end.

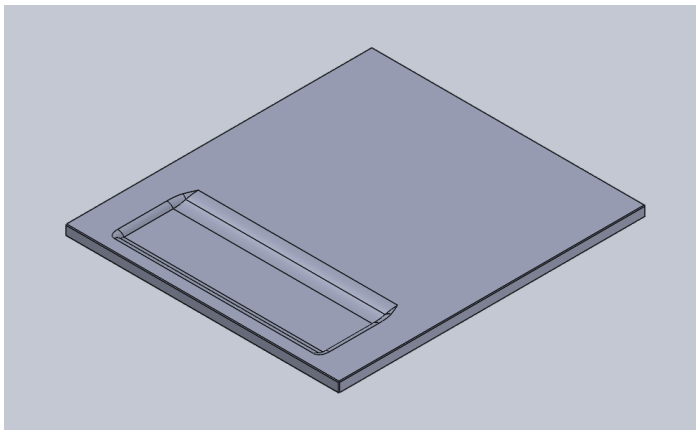
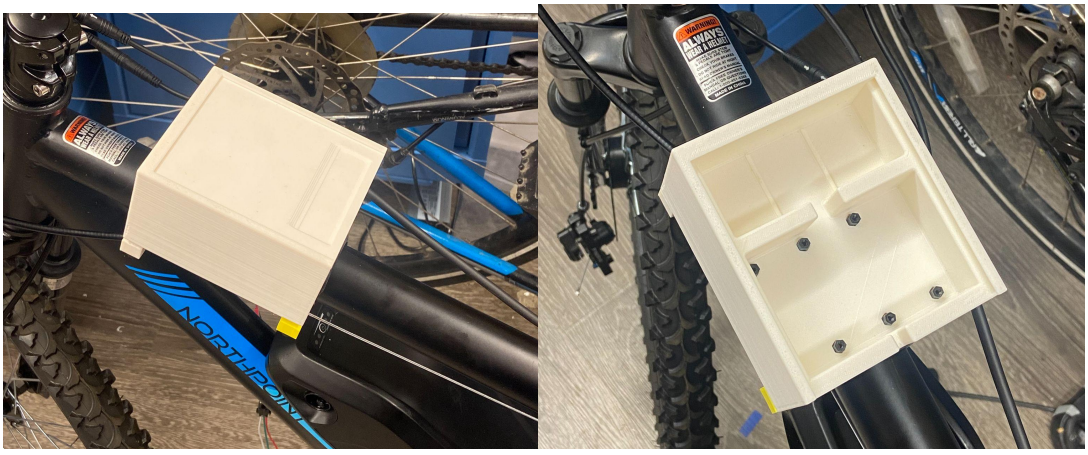


Figure 32: Casing Lid CAD

Once each of these parts are manufactured, with one lid and casing manufactured along with two mounts, the parts were assembled into the configurations shown in Figure 33.



(a) *(b)*

Figure 33: (a) Casing Assembly with Lid (b) Assembly without lid

As is shown, the casing is placed on the bike frame in a location that allows for easy access to the throttle input. Not shown clearly in this picture, a layer of neoprene is also placed between both the casing and the mounts and the bike frame to prevent vibrations.

Connecting the ESP32 to the Motor Controller

For the critical system prototype, the Arduino Uno sent a PWM signal, which was converted into an analog signal by a low-pass filter, to the motor controller. This circuit diagram is shown below in Figure 34.

The final prototype has replaced the Arduino Uno with the ESP32 Feather and has a new motor controller, since the motor itself has been changed. Since the ESP32 Feather features a DAC (Digital to Analog Converter), a low-pass filter is no longer needed. Furthermore, the ESP32 Feather is powered with a 3.7 Volt LiPo battery; this battery has replaced the 9 Volt battery used to power the Arduino Uno. Comparing the circuit diagrams presented in Figure 34 and Figure 35, it is evident that using the ESP32 greatly simplifies the hardware aspect of the motor control. The code is also simplified, as sending a PWM through a low-pass filter makes an indirect voltage control to the throttle and requires the math to convert the PWM signal to the voltage out value. The DAC, on the other hand, allows direct control of the voltage into the throttle.

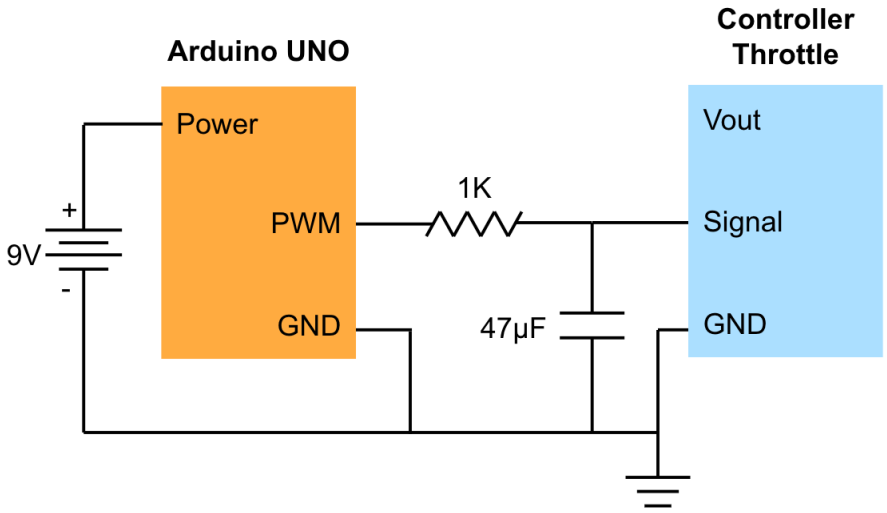


Figure 34: The connection between the Arduino Uno and motor controller used for the critical system prototype

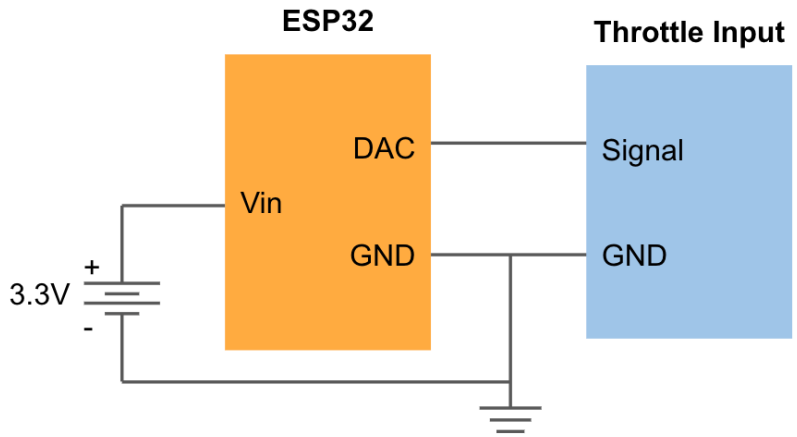


Figure 35: The connection between the ESP32 and motor throttle input used for the final prototype

Bike Frame Selection

The bike frame for the prototype was selected based on a variety of factors based on research into which bikes are best suited for e-bike conversion. The bike frame was selected based on material, brake types, cost, wheel size, and suspension type, and options were sorted into an alternatives matrix pictured in Table 8. Additional information regarding bike frame research and selection is in Appendix H.

Table 8: Bike Frame Alternatives Matrix

Name/Photo	Material	Brakes	Suspension	Cost	Wheel Size	Link
	Steel	Disc	Yes	$199.97	26”	link
	Aluminum	Disc	Yes	$425.99	26”	link
	Aluminum	Disc	Yes	$268	27.5”	link
	Aluminum	Rim	No??	$499	???	link
	Aluminum	Rim	Yes	$248	26”	link
	Steel	Rim	Yes	$148	26”	link

	Steel	Disc in front Rim in rear	Yes	$198	26”	link

According to the original team specifications, the bike was to be as light as possible within the allotted budget, leading the frame materials to be narrowed down to steel and aluminum. The frame needed 26” wheels to fit the bike trainer, and needed front suspension for a more comfortable ride. A frame with rim brakes rather than disc brakes on the rear wheel was also selected in the event that a hub motor was selected, but this ended up being an unnecessary consideration.

The bike that was selected, the Kent Northpoint Men’s Mountain Bike (Figure 36), has 26 inch wheels, so it will fit into the 26” bike trainer purchased. Though the frame does have suspension for a smoother ride, it does not have a rear suspension, which allowed for the battery to be installed on the frame. Finally, the Kent bike frame’s shape is simple enough to allow easy access to the down tube, in order to complete the e-bike conversion, which made it a better choice than some of the other options that had curved or more complex frames.

Figure 36. Kent Northpoint Men’s Mountain Bike

Though aluminum is lighter than steel, this bike frame was still selected even though it is fabricated out of steel due to the other factors being more important to meeting the product specifications, since they directly relate to the ability to convert the frame to an e-bike.

Power Meter Motor Assist Algorithm

The team decided to make the power meter motor assist algorithm a continuous function that is dependent on the anticipated maximum power input by the user. This continuous function is in contrast to the heart rate function, which categories the heart rate input values into five discrete "zones." The power meter motor assist algorithm is:

$Assist = Max\ Power - Power$, where $Assist$ is proportional to the amount of power assistance provided by the motor, $Max\ Power$ is the anticipated maximum power input by the user, and $Power$ is the amount of power input by the user at that specific moment in time. Note that $Max\ Power$ is a constant that must be input prior to the start of the ride.

As the equation shows, there is a negative linear relationship between the amount of power input by the user and the amount of power assist provided by the motor. In effect, this is meant to cause the total power of the system to remain approximately constant, where the total power of the system is the sum of the rider's power contribution and the motor's power contribution.

However, it also accounts for two specific cases:

1. If $Power = 0, then\ Assist = 0$. This would then cause the motor assist equation to become $Assist = Max\ Power$. However, this would not be ideal since this would mean that the assist would start kicking in before the user even starts pedaling. Instead, the user needs to provide at least 1 Watt for the motor assist to start.
2. If $Power > Max\ Power, then\ Assist = 0$. Based on the motor assist equation, this case would result in a negative assist level. Instead of this occurring, the assist will simply hold steady at zero once the maximum power is reached or exceeded.

The above assist equation, however, was still in Watts. For it to be sent to the ESP32, it needed to be converted to a unitless analog value, which ranges from 0 to 255. The formula for this calculation is shown below:

$$Analog\ Assist = Assist \times (255/(Max\ Power - 1))$$

For example, if $Max\ Power = 200W$, then the maximum $Assist$ value would be 199. For all $Max\ Power$ values, the minimum $Assist$ value would be 0. Since analog values go from 0 to

255, the minimum and maximum $Assist$ values should scale to the minimum and maximum $Analog\ Assist$ values.

$$Maximum\ Analog\ Assist\ =\ 199\ \times (255/(200\ -\ 1)) = 255$$
$$Minimum\ Analog\ Assist\ =\ 0\ \times (255/(200\ -\ 1)) = 0$$

As anticipated, the extreme values of $Assist$ scale correctly to the extreme values of $Analog\ Assist$. Combining these two equations yields the final equation to convert between $Power$ and $Analog\ Assist$:

$$Analog\ Assist\ =\ (Max\ Power\ -\ Power)\ \times (255/(Max\ Power\ -\ 1))$$

Initial BLE Connection to Power Meter Pedals

To establish a Bluetooth connection between the ESP32 and the power meter pedals, the computer application “Bluetooth LE Explorer” was first utilized. This was needed to determine important Bluetooth identification properties of the Favero Assimoa pedals, such as their Service UUIDs (Universally Unique Identifier). Figure 37 shows the data that was displayed after connecting to the power meter pedals. Note that there is only one device because the pedals were configured so that all data is sent from the left pedal.

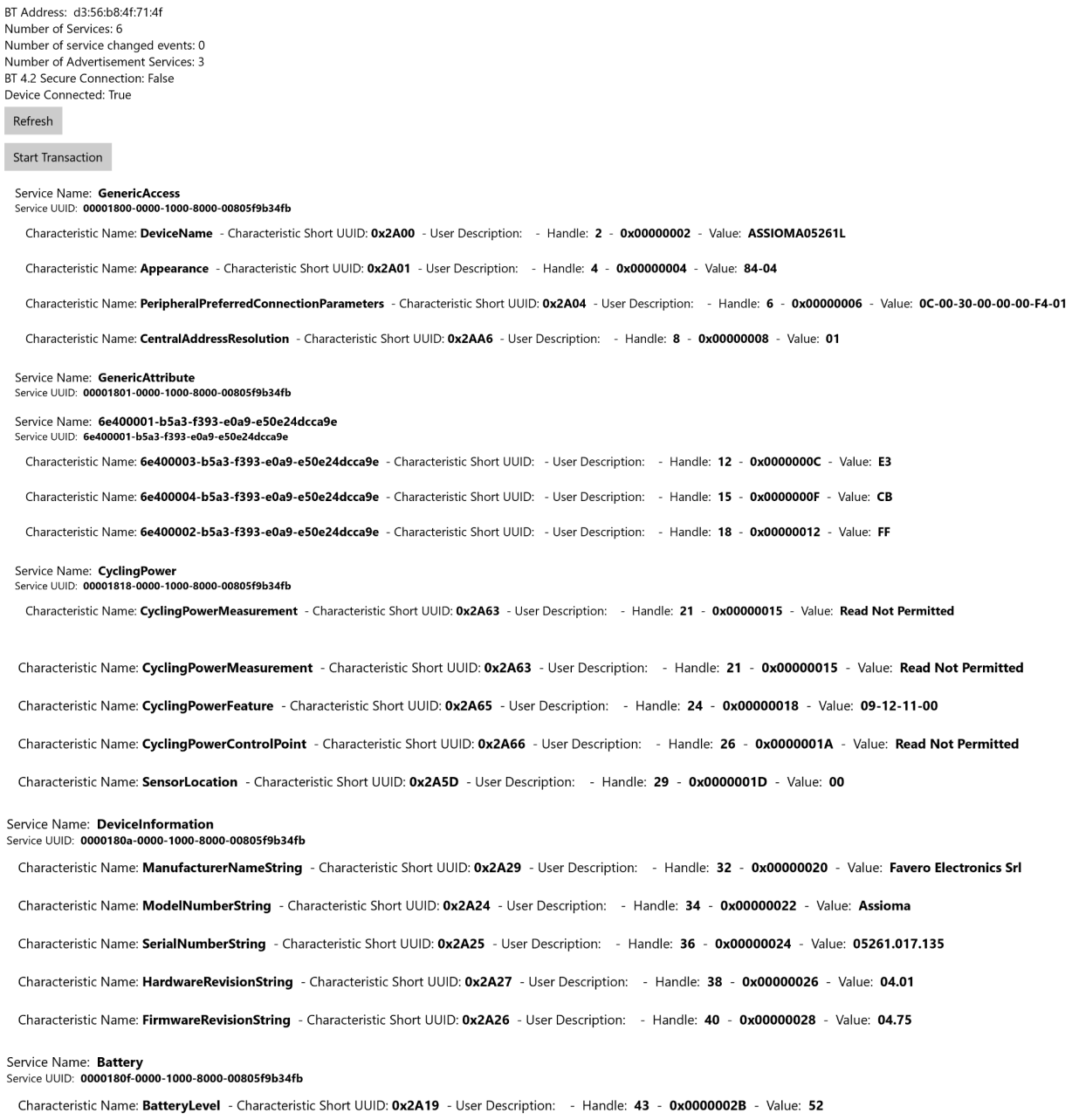


Figure 37: Bluetooth properties of the power meter pedals according to Bluetooth LE Explorer

As Figure 37 shows, the power meter pedals contain many different service UUIDs: “GenericAccess”, “GenericAttribute”, “DeviceInformation”, and “Battery”. Within each service UUID, there are different characteristic short UUIDs. Since our goal is to read the power measurement, the relevant Characteristic Name is “CyclingPowerMeasurement,” which has the Service Name of “CyclingPower” and the Characteristic Short UUID of “0x2A63.” This Short UUID makes sense, as bluetooth.com’s 16-bit UUID Numbers Document states that 0x2A63 is

reserved for “Cycling Power Measurement.” A section of this document is shown below in Figure 38.

GATT Characteristic and Object Type	0x2A63	Cycling Power Measurement
GATT Characteristic and Object Type	0x2A64	Cycling Power Vector
GATT Characteristic and Object Type	0x2A65	Cycling Power Feature
GATT Characteristic and Object Type	0x2A66	Cycling Power Control Point
GATT Characteristic and Object Type	0x2A67	Location and Speed
GATT Characteristic and Object Type	0x2A68	Navigation
GATT Characteristic and Object Type	0x2A69	Position Quality

Figure 38: bluetooth.com’s 16-bit UUID Numbers Document

With the service UUID and characteristic short UUID, the ESP32 has enough information to directly connect to the power meter pedals. However, even though the ESP32 was connected to the power meter pedals, it was still not capable of displaying any information. The root cause of this error was hypothesized to be an incompatibility between the data types being sent to the ESP32 and the data types of the ESP32’s variables that are storing and displaying the information. Because of this, it was important to observe the type of data the power meter pedals were sending. To view this data, the phone app “nRF Connect” was used. A screenshot of the raw data being sent from the power meter pedals to the phone is shown below in Figure 39. The data is in green text, with its corresponding timestamp directly to the left.

Figure 39: Raw data being sent from the power meter pedals to the "nRF Connect" app

As the green text in Figure 39 shows, the power meter pedals send 8 bytes of data in hexadecimal form. As the timestamps show, data is sent at an approximate frequency of once per second. With this information, the code was altered so that the data was properly read in by the ESP32.

The final step was to adjust this data so that it's in its proper form. To do so, it was necessary to understand the formatting of the data. This information was available on an XML file provided by bluetooth.com. A screenshot of the section of the XML file that includes information on "Instantaneous Power" is shown below in Figure 40.

```
▼<Value>
  ▼<Field name="Flags">
      <Requirement>Mandatory</Requirement>
      <Format>16bit</Format>
    ▶<BitField>
      ...
      </BitField>
      <b>C1:These Fields are dependent upon the Flags field</b>
      <p/>
    </Field>
  ▼<Field name="Instantaneous Power">
      <InformativeText> Unit is in watts with a resolution of 1. </InformativeText>
      <Requirement>Mandatory</Requirement>
      <Format>sint16</Format>
      <Unit>org.bluetooth.unit.power.watt</Unit>
      <DecimalExponent>0</DecimalExponent>
    </Field>
  ▼<Field name="Pedal Power Balance">
```

Figure 40: bluetooth.com's document for "Cycling Power Measurement"

In Figure 40, "Flags" is the first Field Name. Since it is shown to be 16 bits, which is equivalent to 2 bytes, it is clear that the first two bytes of data sent represent "Flag" values. The next field name is "Instantaneous Power", which is the field that is relevant for our purposes. Its format, sint16, simply means that it is a signed (can be positive or negative) integer of 16 bits. This means that bytes 2 and 3 of the data correspond to power.

With all of the above information, the final code was implemented. A screenshot of the data output is shown below in Figure 41.

```
Watts: 0
Notify callback for characteristic 00002a63-0000-1000-8000-00805f9b34fb of data length 8
data:
Watts: 0
Notify callback for characteristic 00002a63-0000-1000-8000-00805f9b34fb of data length 8
data:
Watts: 0
Notify callback for characteristic 00002a63-0000-1000-8000-00805f9b34fb of data length 8
data:
Watts: 0
Notify callback for characteristic 00002a63-0000-1000-8000-00805f9b34fb of data length 8
data:
Watts: 44
Notify callback for characteristic 00002a63-0000-1000-8000-00805f9b34fb of data length 8
data:
Watts: 162
Notify callback for characteristic 00002a63-0000-1000-8000-00805f9b34fb of data length 8
data:
Watts: 162
Notify callback for characteristic 00002a63-0000-1000-8000-00805f9b34fb of data length 8
data:
Watts: 174
Notify callback for characteristic 00002a63-0000-1000-8000-00805f9b34fb of data length 8
data:
Watts: 124
Notify callback for characteristic 00002a63-0000-1000-8000-00805f9b34fb of data length 8
data:
Watts: 236
Notify callback for characteristic 00002a63-0000-1000-8000-00805f9b34fb of data length 8
data:
```

Figure 41: Power meter pedals output shown in the ESP32 's serial monitor

BLE Properties of the Power Meter Pedals

While the prior section, “Initial BLE Connection to Power Meter Pedals”, resulted in the capability to extract power values from the power meter pedals, further investigation was needed to understand if any other data was capable of being transmitted by the power meter pedals.

Information regarding all of the data sent by the power meter pedals is stored in bluetooth.com’s document for “Cycling Power Measurement.” However, since this site is a raw XML file, and is therefore extremely hard to read, its relevant information had to be transposed.

First, Table 9 displays the 13 flags associated with Cycling Power Measurement. Each flag is assigned one bit; these bits are stored in the first two bytes that the power meter pedals send. As an example, a sample packet of data that was transmitted by the power meter pedals and will be used throughout this section is shown below:

(0x) 20-00-DE-0A-54-01-FB-3F

For this data, the zeroth byte is (0x)20 and the first byte is (0x)00. In binary form, this translates to 00100000 for the zeroth byte and 00000000 for the first byte. Noting that the first byte should come before the zeroth byte, this becomes 0000000000100000. Therefore, the only bit that has a value of 1 is the fifth bit. As Table 9 shows, this corresponds to “Crank Revolution Data Present.” This means that Crank Revolution Data can be sent.

Table 9: The flag bits, names, and values for the power meter pedals

Bit	Name	Value
0	Pedal Power Balance Present	0 (False)
1	Pedal Power Balance Reference	0 (Unknown)
2	Accumulated Torque Present	0 (True)
3	Accumulated Torque Source	0 (Wheel Based)
4	Wheel Revolution Data Present	0 (False)
5	Crank Revolution Data Present	1 (True)
6	Extreme Force Magnitudes Present	0 (False)

7	Extreme Torque Magnitudes Present	0 (False)
8	Extreme Angles Present	0 (False)
9	Top Dead Spot Angle Present	0 (False)
10	Bottom Dead Spot Angle Present	0 (False)
11	Accumulated Energy Present	0 (False)
12	Offset Compensation Indicator	0 (False)

Beyond displaying the different flags associated with Cycling Power Measurement, the XML file also shows the different types of data and their data format. This information has been included below in Table 10. The information in Table 9 is essential to filling out Table 10, since the value of each flag determines whether or not that specific type of data will be sent by the power meter pedals. For example, if bit 2, “Accumulated Torque Present”, had a value of 1 in Table 9, it would occupy bytes 4 and 5 of the data being sent by the power meter pedals. But since it instead has a value of zero, it can be ignored.

Table 10: Data names, Byte locations, and Data Format for the Power Meter Pedals

Name	**Bytes**	**Data Format**
Flags	0,1	16bit
Instantaneous Power	2,3	sint16
Pedal Power Balance	N/A	N/A
Accumulated Torque (N/m)	N/A	N/A
Wheel Revolution Data - Cumulative Wheel Rotations	N/A	N/A
Wheel Revolutions Data - Last Wheel Event Time	N/A	N/A
Crank Revolution Data - Cumulative Crank Revolutions	4,5	uint16
Crank Revolution Data - Last Crank Event Time	6,7	uint16

With Table 10 filled out, the data packet which was displayed previously,
(0x) 20-00-DE-0A-54-01-FB-3F, can now be interpreted.

- (0x) 20-00 is associated with “Flags”
- (0x) DE-0A is associated with “Instantaneous Power”

- (0x) 54-01 is associated with “Crank Revolution Data - Cumulative Crank Revolutions”
- (0x) FB-3F is associated with “Crank Revolution Data - Last Crank Event Time”

With this knowledge, the next task was to determine a way to extract cadence from the crank revolution data. Unfortunately, there is little to no official Bluetooth documentation on the subject; instead, Internet forums were consulted. After doing so, the following information was discovered.

- “Cumulative Crank Revolutions” simply increments by one each time that the crank completes one full revolution.
- “Last Crank Event Time” is a counter that increments at a frequency of 1024 Hz. \
    - Since it is two bytes long, it rolls over every 64 seconds ($2^{16}/1024Hz = 64s$)
- By taking the difference between the most recent “last crank event time” and the prior one, the time between them can be calculated
    - This is assuming that the counter did not roll over.
- Similarly, taking the difference between their two corresponding “cumulative crank revolutions” will yield how many revolutions occurred during that time period
- By dividing the difference in “cumulative crank revolutions” by the difference in “last crank event time”, the cadence can be calculated.

With this knowledge of the BLE properties of the power meter pedals, code to simultaneously extract and print power and cadence values was written. A screenshot of the result is shown below in Figure 42.

```
Watts: 176
Motor assist analog value (0 to 255):30
Notify callback for characteristic 00002a63-0000-1000-8000-00805f9b34fb of data length 8
data:
Watts: 206
Motor assist analog value (0 to 255):1
Notify callback for characteristic 00002a63-0000-1000-8000-00805f9b34fb of data length 8
data:
Cadence (RPM): 59
Watts: 206
Motor assist analog value (0 to 255):1
Notify callback for characteristic 00002a63-0000-1000-8000-00805f9b34fb of data length 8
data:
Watts: 152
Motor assist analog value (0 to 255):61
Notify callback for characteristic 00002a63-0000-1000-8000-00805f9b34fb of data length 8
data:
Watts: 164
```

Figure 42: Power and cadence values from the power meter pedals

The reason why cadence is only shown once in Figure 42 is because it is only calculated once every 10 seconds, as opposed to power, which is displayed every second. This is due to the fact that a cadence calculation every second would not provide meaningful data. For example, if a

rider is pedaling at slower than 60 RPM, there will be many calculations where the change in “cumulative crank revolutions" is zero; this would then make the cadence value equal to zero.

BLE Properties of the Heart Rate Sensors

Heart Rate Measurement Sensor sends multiple bytes of information and the first bit field, byte, is called the Flag Bit field as graphically represented in Figure 41. Depending on the data sent, each flag bit will either be a 0 or 1 to indicate if specific data is being retrieved from the heart rate sensor.

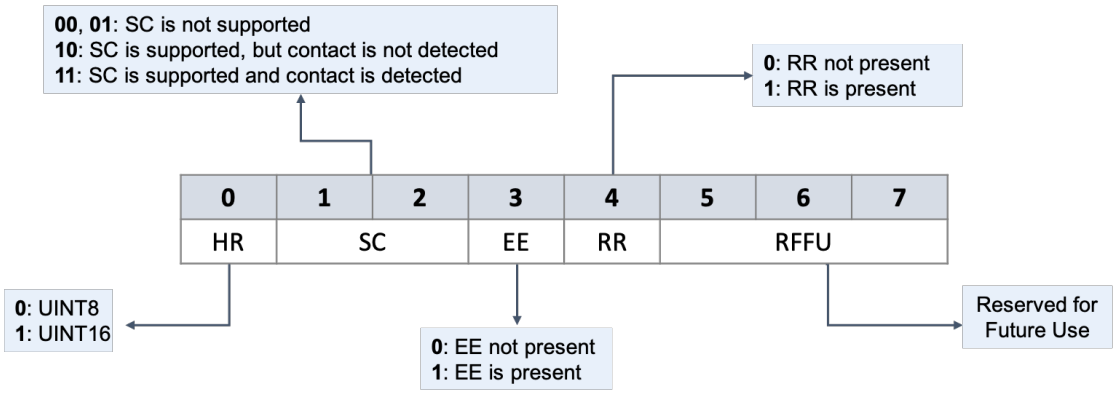


Figure 43: Heart Measurement Service Data - Flag Byte Schema

In this bit field, there are the following bits:

- HR (Heart Rate) - how heart rate data is being represented
    - 0 = uint8
    - 1 = uint16
- SC
    - sensor connection feature
- EE (Energy Expended) in Kilojoule - uint16
    - 0 indicates EE not present
    - 1 indicated EE is present
- RR intervals
    - indicates if RR intervals is present
- RRFU - RR Future Use

In Bluetooth heart rate monitors, the data is organized and structured in the following XML format:

https://github.com/oesmith/gatt-xml/blob/master/org.bluetooth.characteristic.heart_rate_measurement.xml

In this XML code, there are 4 main bytes (flags, HR in uint8, HR in uint16, EE, and RRs). The number of RR bytes present can vary due to heart rate sensor conditions (i.e. increased heart rate, more movement of the sensor, location, etc). Heart Rate data, for the heart rate sensor used in this project, can be accessed by accessing the byte 1 as shown by pData[1] in the code below (Figure 44).

```
static void notifyCallback(
  BLERemoteCharacteristic* pBLERemoteCharacteristic,
  uint8_t* pData,
  size_t length,
  bool isNotify) {
    Serial.print("Notify callback for characteristic ");
    Serial.print(pBLERemoteCharacteristic->getUUID().toString().c_str());
    Serial.print(" of data length ");
    Serial.println(length);
    Serial.print("data: ");
    Serial.println((char*)pData);
    char json_heartrate_data[] = "{ \"bpm\":\"%i\" }";

    if (length < 20){ // ==4 does not include 6, 8
      Serial.print(pData[1], DEC);
      Serial.println("bpm");
      int bpm = ((int)pData[1]);
```

Figure 44: Accessing 'bpm' data in pData[1] or byte 1

Establishing Wireless Communication Between Two ESP32s

The team discovered that simultaneously sending data over Bluetooth from the power meter pedals and the heart rate sensor to a single ESP32 was not feasible. However, it was important to have a "mode" that utilized multiple sensors, since this would demonstrate that SmartWorks could handle a stream of multiple categories of data at the same time. Additionally, it would serve as a proof of concept that could be theoretically expanded upon in the future, as, up to a certain point, there's little difference between using two sensors, three sensors, or more. This proof of concept is also the reason why the casing was built to fit three ESP32s, despite the fact that our final prototype only uses two.

Since it was found that there needed to be one ESP32 for each sensor that was to be used, it was essential to establish a communication protocol between multiple ESP32s. After research was conducted, it was decided that ESP-NOW would be the wireless communication protocol that is used. This is because it was developed by Espressif, the manufacturer of the ESP32, and because it is meant for a use case where fast communication is needed but message sizes are small (up to 250 bytes).

First, connection was established between the two devices without using power meter pedal values from a Bluetooth connection. Instead, a single float value (23.00) was sent from the Sender to the Receiver. The result of this test is shown below in Figure 45.

```
Last Packet Send Status:        Delivery Success
Bytes received: 4
The received temperature is:23.00
Sent with success

Last Packet Send Status:        Delivery Success
Bytes received: 4
The received temperature is:23.00
Sent with success

Last Packet Send Status:        Delivery Success
Bytes received: 4
The received temperature is:23.00
```

Figure 45: Data being sent from the Sender ESP32 to the Receiver ESP32 using ESP-NOW

Next, the existing code which extracts power data from the power meter pedals via Bluetooth was copied and added to this ESP-NOW code. Figure 46 shows power data being sent from the Sender ESP32.

```
Last Packet Send Status:        Delivery Success
Notify callback for characteristic 00002a63-0000-1000-8000-00805f9b34fb of data length 8
data:
154
Sent with success

Last Packet Send Status:        Delivery Success
Notify callback for characteristic 00002a63-0000-1000-8000-00805f9b34fb of data length 8
data:
132
Sent with success

Last Packet Send Status:        Delivery Success
Notify callback for characteristic 00002a63-0000-1000-8000-00805f9b34fb of data length 8
data:
164
Sent with success
```

Figure 46: Power data being sent from the Sender ESP32 to the Receiver ESP32 using ESP-NOW

To verify that this data was being received, the Receiver ESP32 was connected to the Arduino Serial Terminal. This is shown below in Figure 47. Note that the sampling frequency for power is about 1Hz and that riding was only conducted during lines 3, 4, and 5 of Figure 47. With this

ride providing about 25 data points and the sampling frequency being 1Hz, this test ride lasted about 25 seconds.

```
The received power is:0, 0, 0, 0, 0, 0, 0, 0, 0, 0, Bytes received: 40
The received power is:0, 0, 0, 0, 0, 0, 0, 0, 0, 0, Bytes received: 40
The received power is:0, 56, 226, 226, 186, 190, 210, 218, 184, 212, Bytes received: 40
The received power is:218, 208, 206, 214, 188, 208, 182, 134, 134, 134, Bytes received: 40
The received power is:134, 134, 134, 134, 0, 0, 0, 0, 0, 0, Bytes received: 40
The received power is:0, 0, 0, 0, 0, 0, 0, 0, 0, 0, Bytes received: 40
The received power is:0, 0, 0, 0, 0, 0, 0, 0, 0, 0, Bytes received: 40
The received power is:0, 0, 0, 0, 0, 0, 0, 0, 0,
```

Figure 47: Power data being received by the Receiver ESP32

Figure 48 shows the flow of data that was established using this ESP-NOW communication. The Sender ESP32 receives data via Bluetooth from one sensor, while the Receiver ESP32 receives data from the other sensor. Then, the Sender ESP32 sends the data it received from the sensor to the Receiver ESP32, allowing the Receiver ESP32 to simultaneously possess both types of sensor data. The Receiver ESP32 then uses the sensor data to control the motor and send data to SmartWorks.

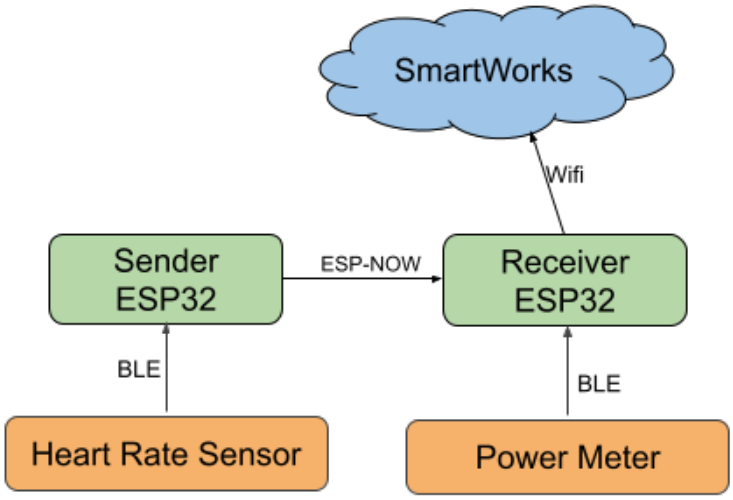


Figure 48: The flow of data for ESP to ESP communication

It is worth noting that the connection of the heart rate sensor to the Sender ESP32 and the power meter to the Receiver ESP32 in Figure 48 is completely arbitrary. It's important that each sensor is connected to its own ESP32, but it does not matter which sensor is connected to which ESP32.

To connect a third sensor to the system shown in Figure 46, a second Sender ESP32 would be used. Just like the Sender ESP32 in Figure 46, this second sender would receive data from a sensor and send its data to the Receiver ESP32 via ESP-NOW communication.

Sending Sensor Data to ESP32 and SmartWorks IoT Platform

Retrieving Sensor Data using BLE

To establish an IoT system, the team utilized sensors capable of communicating with the ESP32 through Bluetooth Low Energy or BLE. The Coospo heart rate sensor and the Favero Assimo power meter pedals both have BLE capabilities and can communicate with the ESP32 using an Arduino library called "BLE Client". The BLE Client library provides the tools to retrieve data by identifying the specified device service UUID, a unique identifier of a bluetooth device. Once connected, the code can be modified to parse critical data points, as each device may send a unique structure of data bytes that indicate different metrics (Figure 49). The parsed data points will then be sent to the ESP32 microcontroller.

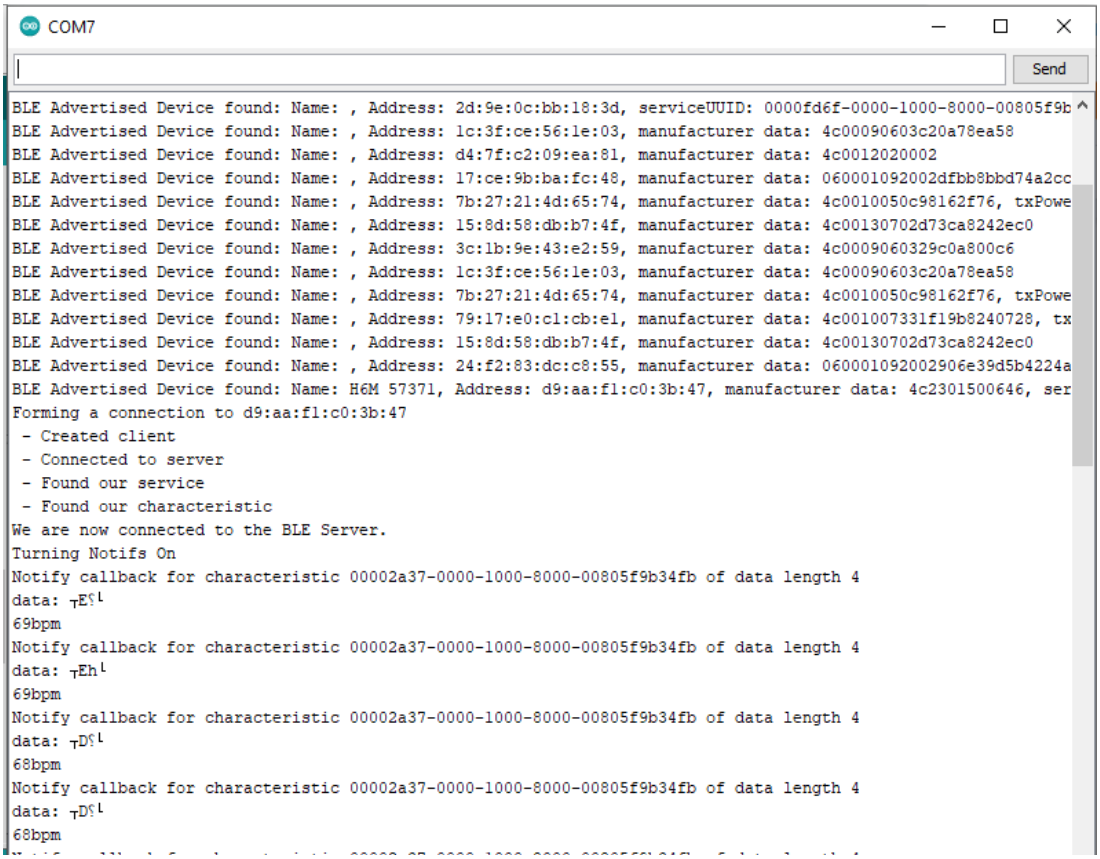


Figure 49: Heart Rate Sensor Data Retrieval displayed on Serial Monitor

Sending from ESP32 to SmartWorks

SmartWorks' development team released code with the goal of connecting a ESP8266 to SmartWorks by utilizing MQTT communication protocols and a coding development platform called PlatformIO. PlatformIO is a coding platform that allows for coding from different platforms on a singular platform; for example, code from the ArduinoIDE can be moved to PlatformIO. Inside the tutorial code were 6 files as listed:

Files

- credentials.h - stores credential information (needs to be manually filled)
- main.cpp - executable code
- topics_publish.h - stores link to send data to (needs to be manually filled)
- topics_subscribe.h - store link to retrieve data from (needs to be manually filled)
- wifi-mqtt.cpp - contains functions for establishing Wifi and MQTT communication
- wifi-mqtt.h - header file for wifi-mqtt.cpp

The team was able to utilize this tutorial to send simple data from the ESP32 to the cloud; however, there were a lot of functionalities that would need to be added in order to create new workout instances automatically and have data sent to that instance/"Thing". The reason being that credentials and data links need to be written in manually.

SmartWorks: Functions, Panopticon, IoT Management

SmartWorks IoT is a cloud platform capable of IoT management, sending and storing data from multiple devices, creating data analytical dashboards, and automating custom functions. One of the main goals of the project was to integrate this software with the critical systems of the e-bike by having the sensors send data and retrieve an appropriate assist level. SmartWorks IoT has several tools to manipulate data and to create an IoT system and the team would utilize

AnythingDB & Things

"AnythingDB", "Real-time Visualization", and "Functions" (Figure 50) to control communication between the e-bike, sensors, and the platform itself.

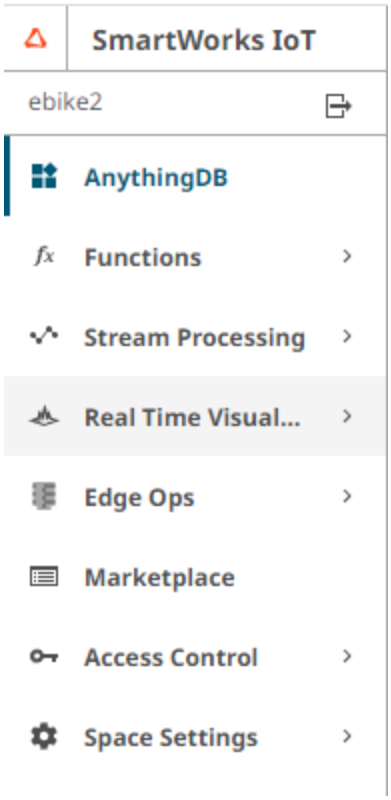


Figure 50: SmartWorks IoT Tools and Functions

"AnythingDB" allows the user to create a digital representation of real world devices and storage of any data that a certain device might receive. In this feature, the team created a "Thing", a representation of a certain device or device activity, and it was used to represent a user's individual workout (Figure 51) Inside a "Thing", there are "Properties" that represent important entities of a given "Thing". For example, in a workout "Thing", the following "Properties" can important such as: "age", "sex", "weight", "state", "bpm", "power", "cadence", "calories_burned", and more. When a data point is sent to the "Thing", it can populate these fields and give a place for SmartWorks to read how the "Thing" is being updated and potentially run functions depending on the readings.

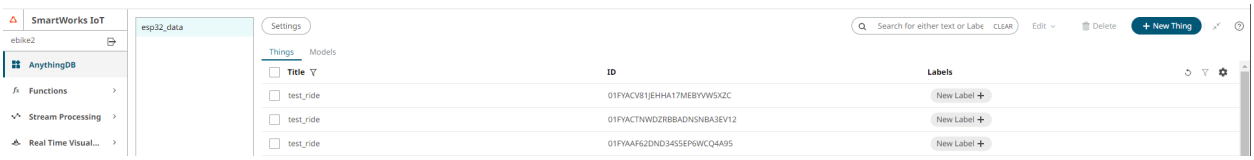


Figure 51: AnythingDB and "Things" as shown as test_ride

Within this "Thing", the data from the heart rate sensors and power meter pedals were sent and the data it received will be uniquely tied to this "Thing". By doing so, the user can have a clearer view of the workout data. However, SmartWorks does not automatically create a new "Thing" with each new workout session and has to be manually created. Without directing new data into different "Things", the data is all stored in one location making it difficult to differentiate

between workout timings and can provide a point of confusion for the user trying to analyze the data. The team will address this issue later in this section.

Each “Thing” contains a unique UID number, “Thing” password, MQTT username and password, and a “Thing” client username and password. These credentials are used as an anchor point to direct data sending devices to the “Thing” and also as a method for SmartWorks to identify what “Things” need to be operated on for any type of functions or visualization.

Functions

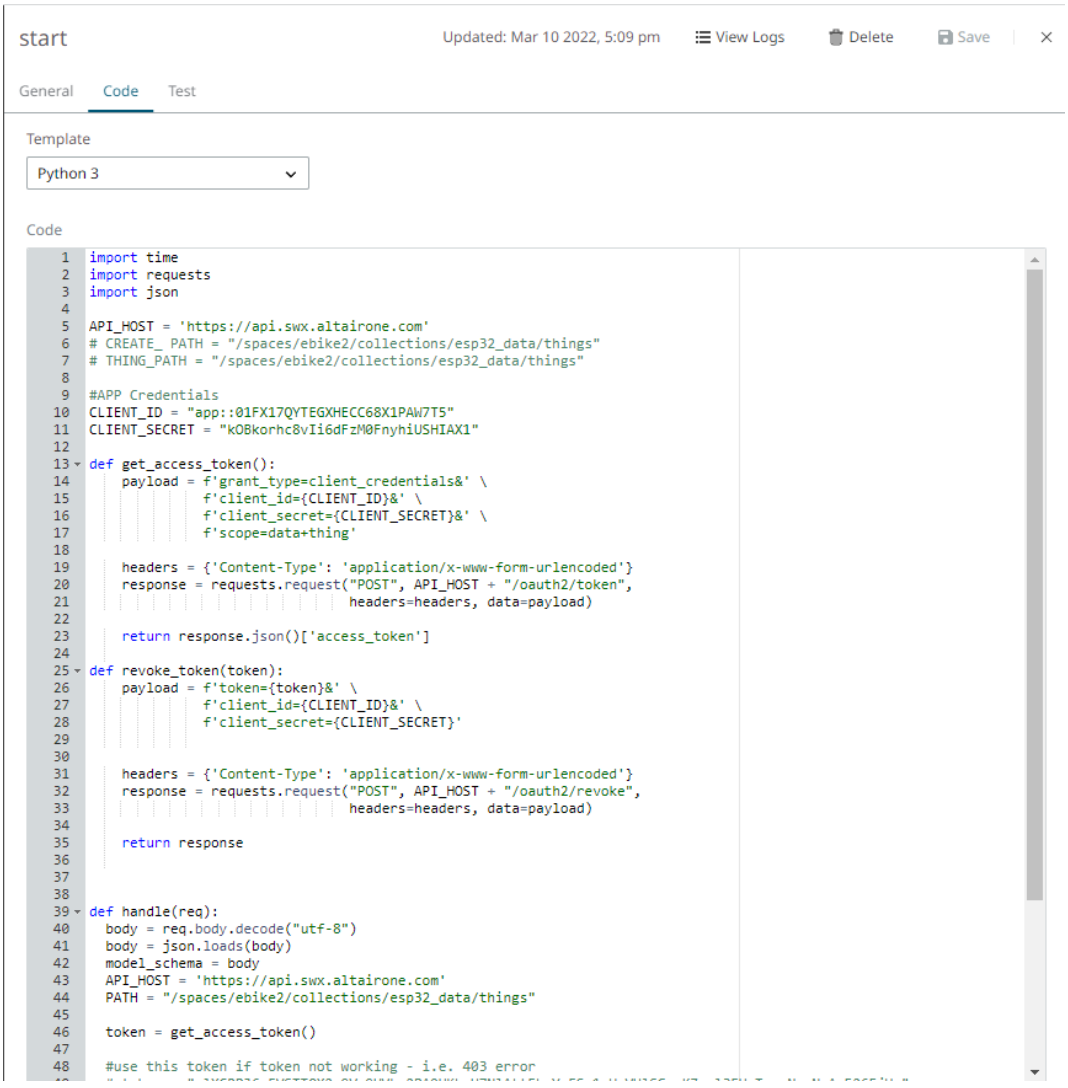


Figure 52: Function Code on SmartWorks

SmartWorks is capable of hosting multiple functions to carry out custom tasks on entities within SmartWorks or communicate with other devices or services through HTTP or MQTT. For instance, in Figure 52 (above), the function named “start” is finding all “Things” with the “state” property set to “opened”. If the workout ride is opened, then the function will return all data

regarding the “Thing” except for passwords. This is one illustration of how functions can be used within SmartWorks to streamline IoT management and a list of functions created and their documentation is placed in Appendix F. Another important aspect of these functions is that it can be called anywhere such as a website, a separate coding IDE, etc. The web app takes advantage of this capability and runs several functions from SmartWorks to retrieve and send necessary data.

Panopticon

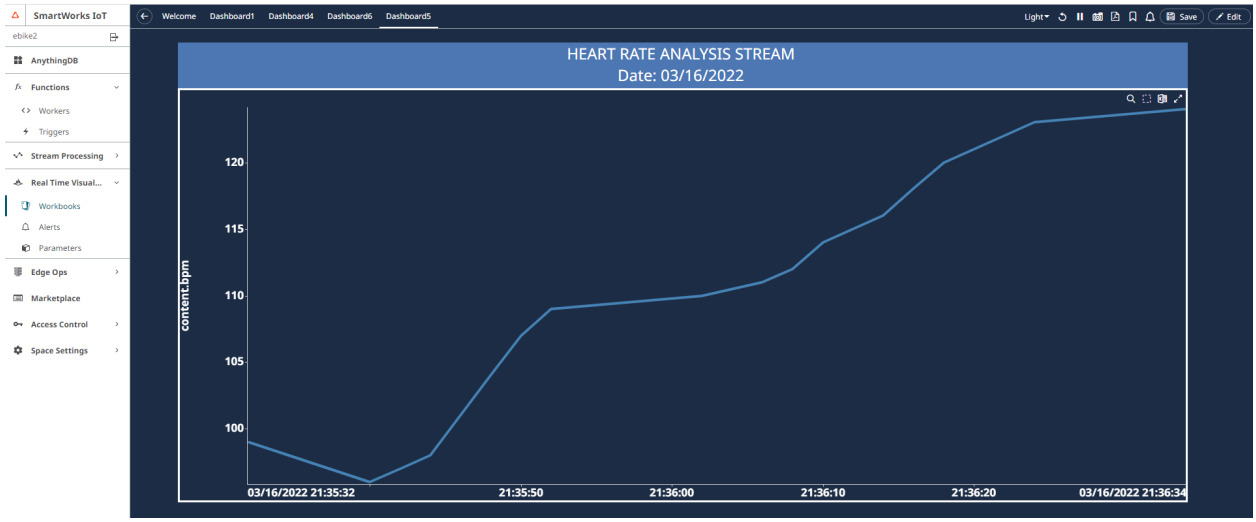


Figure 53: Panopticon Visualization of Heart Rate Stream

The data stored in “Things” can be extracted to be plotted and visualized to provide the user feedback on the data that is being collected. Within the context of this project, heart rate data and assist level was plotted to show the user how long they were in each workout zone and how their biometric data was changing throughout the ride. Panopticon has features that allow the data to be streamed in real-time using MQTT, but also can simulate data streams given an Excel file by plotting data points in a time interval. Panopticon has a plethora of features for data visualization and dashboarding.

Creating New Things & Workout Instances Overview

SmartWorks created a tutorial in the beginning of Winter Quarter that demonstrated the connection between a microcontroller and SmartWorks’ “Thing” through MQTT and released the code used to achieve that goal. In the code, the “Thing” UID and path had to be manually written indicating that every new instance that is created requires the credentials to be coded in, which is inefficient and places a burden on the user. To circumvent this obstacle, the team coded a microcontroller program that runs multiple HTTP and MQTT commands to retrieve credentials and automatically send data to the new “Thing” without the need of user input. The general workflow of the program is highlighted below:

1. Create a new “Thing” from web app
    a. set MQTT password of “Thing”
2. Using SmartWorks functions find opened “Thing”
    a. retrieve credentials such UID, MQTT username
        i. Note: each call to SmartWorks requires an authorization token to allow the calls to be successfully executed and has its own process
        ii. Process
            1. retrieve Oauth token
            2. send the token data with each HTTP call
            3. revoke the token to prevent any lockouts from the server
3. Populate all credential information in the microcontroller program using the data extracted in steps 1 and 2
4. Send data to SmartWorks

The full code is in this Github Repo: https://github.com/mattkim17/ESP32_IoT

Notes Regarding SmartWorks

Throughout the capstone project, SmartWorks IoT has been going through multiple updates, revisions, and changes, as the software is still in its development stage. As a result, new features and capabilities were added to the software, but also the team had to pivot and continually learn with the new changes being integrated to the platform.

Testing & Performance Evaluation

The majority of the testing that was conducted for this project was done to verify that the individual components function as expected. These tests were previously documented in the Engineering Analysis section of this report, since the components’ tests were integrated within their overall analysis. One test that was not covered in this section, however, is the current sensor test. Instead, it is described in full below.

Current Sensor Test

To gain insight on the response of the motor, a current sensor was used. The relationship between the current drawn by the motor and the power output of the motor is given by the following two equations:

1. $\tau = k_t I$, where k_t is the torque constant and I is the current
2. $P = \tau\omega$, where τ is the torque and the ω is the rotational velocity

As the two equations show, there is a proportional relationship between the motor’s current and power. However, documentation for the motor’s torque constant was unavailable and finding the torque constant directly by measuring the amount of torque required to stall the motor was

deemed unsafe for such a powerful motor. Because this meant the team was unable to calculate the power output of the motor, it was decided that insights into the motor's response would be gained by monitoring the current output of the motor.

To monitor the current output of the motor, the ACS724 current sensor was selected. The ACS724 met both essential needs for a current sensor: it was able to withstand a high amount of current, as it can tolerate up to 50 Amps, and it was able to have its output monitored by a microcontroller. The circuit diagram for the ACS724 is shown below in Figure 55.

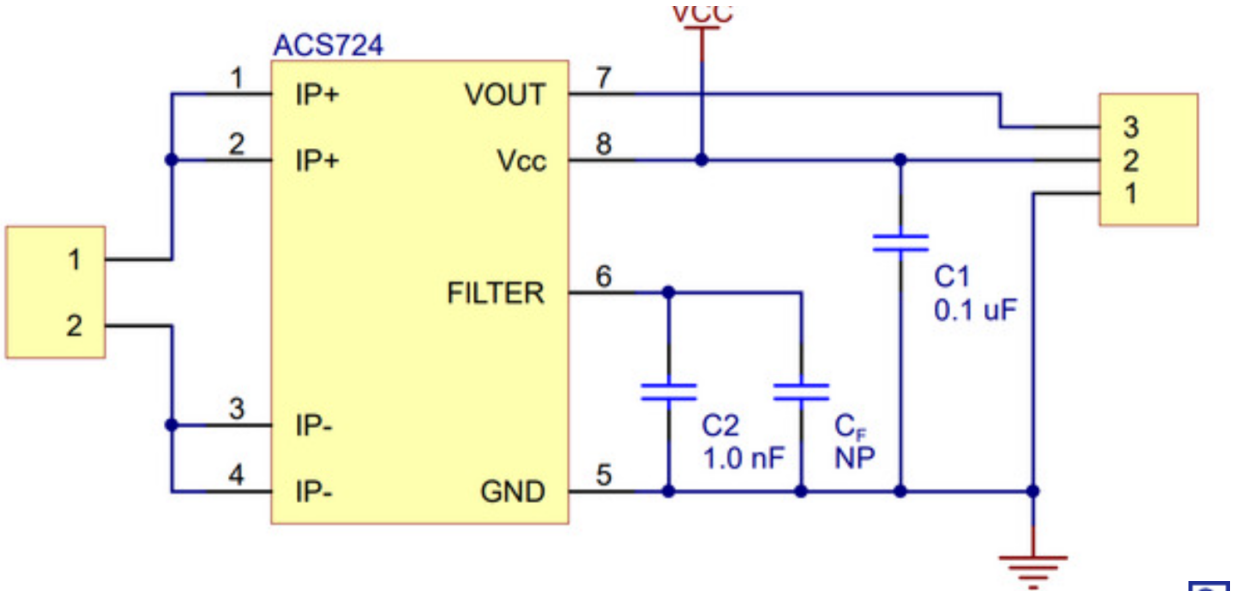


Figure 55: The suggested wiring diagram for the ACS724 current sensor

Descriptions of the pinouts in Figure 55 are shown below. Note that the center component is the ACS724 current sensor and the right-most component represents the Arduino Uno.

- IP+: This is the positive side of the circuit that is being monitored. In our case, IP+ will be connected to the battery that is powering the motor, sinee current flows from the battery to the motor.
- IP-: This is the negative side of the circuit that is being monitored. In our case, IP- will be connected to the motor, since current flows towards the motor.
- IP+ and IP-: In general, these pins are in series with the circuit whose current is being monitored. Since the current going from the battery to the motor is being monitored, IP+ and IP- connect to these two components.
- VOUT: This pin sends the output signal of the current sensor. Therefore, in our setup, it is connected to an analog input pin which will monitor its values over time. According to the manufacturer, this pin should output 2.5V when there is no current flowing through it and change its output at a rate of 40mV/A. For example, 10A should induce an output of

2.9V and -10A should induce an output of 2.1V. In Arduino, analog values go from 0 to 1023, meaning that 2.5V is approximately 511 and 5V is 1023, since 5V is the maximum Arduino voltage.

- Vcc: This is the pin that powers the current sensor. In our setup, the Arduino Uno sends its maximum output voltage, 5V, to this pin.
- Filter: This pin allows the output of the sensor to be filtered by capacitors, resulting in an improved resolution at lower bandwidth.
- GND: This pin allows the current sensor to share a common group with the Arduino Uno.

A picture of the completed setup is shown below in Figure 56. With this setup, multiple tests were run. The first test was done by having a rider increase and decrease their power input. The second test replaced a rider with a square wave at a frequency of 0.1Hz. Finally, the third test monitored how the current drawn by the motor changed as a function of the motor assist level used. The results of these tests, as well as analysis of them, are shown below.

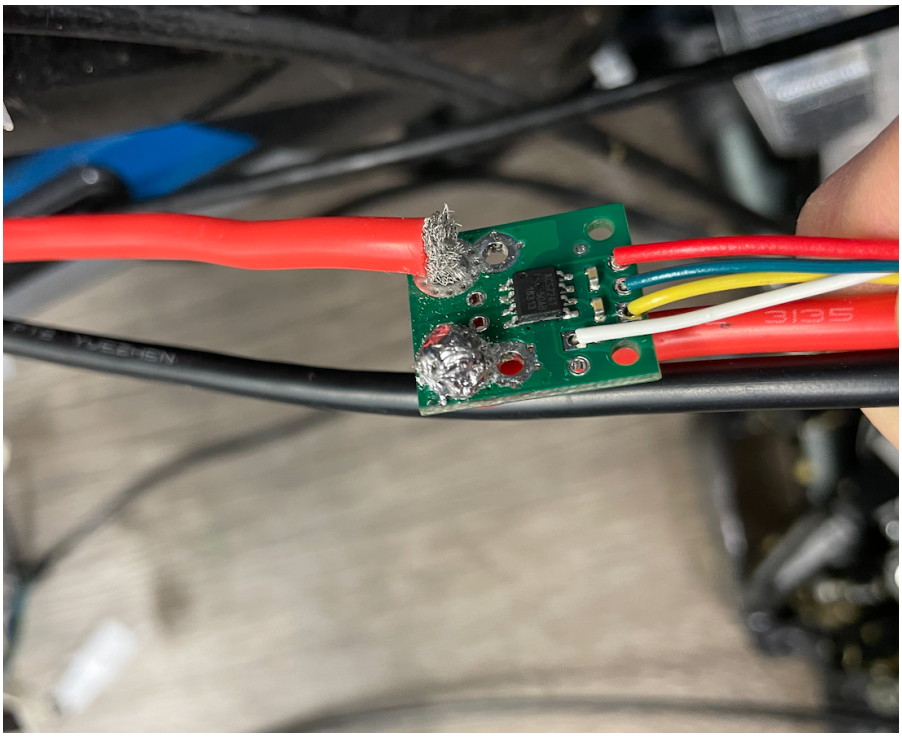


Figure 56: Current sensor installed onto battery

Test One: Monitoring the current output of the motor while a user is riding the bike

The setup for this test is as follows:

Microcontrollers Used

- Sender ESP32
    - Connects via Bluetooth to the power meter pedals and reads in power values
    - Adjusts the motor's assist level based on the power values

        - A description of the algorithm to calculate motor assist is described above in the "Power Meter Motor Assist Algorithm" section of Engineering Analysis
    - Sends the power values via ESP-NOW to the Receiver ESP32
- Receiver ESP32
    - Reads in power values from the Sener ESP32
    - Prints these values into the Arduino IDE Serial Monitor so they can be stored
- Arduino Uno
    - Connects to the VOUT pin of the current sensor
    - Prints the current sensor values into the Arduino IDE Serial Monitor so they can be stored

Acquisition of Data Points

- Current values were read in by the Arduino Uno at a frequency of 10Hz. While it was known that averaging at such a high frequency would result in some noise, it was rationalized by the fact that averaging methods could always be applied during the post-processing stage. If instead, values were sampled at a frequency of 1Hz, it would be impossible to recover all of the data points that occurred in between each 1Hz sample.
- Power values were read in by the Sender ESP32 at a rate of 1Hz. This was done because this is the rate that the power meter pedals sent their data out at, meaning that this was the fastest possible frequency.
- Since the current data and power data were being printed on two different serial ports, some methodology of "lining up" the timing of the data needed to be implemented. This was done using timestamps — next to each data point that was printed, its time of printing was also documented. Then, when the two sets of data were being plotted, the sets were lined up based on their starting times.

Conversion Between Analog Values and Current Values

Since the VOUT pin of the current sensor is read in by an analog input pin, it will read in analog values between 0 and 1023. However, these values need to be converted to current values with units of milliAmps. An explanation of how to do so is shown below.

- For Arduino, 5V corresponds to an analog value of 1023, 0V to an analog value of 510, and -5V to an analog value of 0.
    - Therefore, each analog value is equal to: $\frac{5V}{512} = 0.00977V = 9.77mV$
- The voltage to current ratio of the ACS724 current chip is $40\frac{mV}{A}$
    - Therefore, each analog value is equal to $(9.77mV)/(40\frac{mV}{A}) = 0.244A$
- The formula to translate between analog value and current would then be:

$$current = (analog - 510)(0.244A)$$

- Since, in practice tests that were run , the minimum analog is 510 and the maximum analog is about 527, the range of current for this test is:
    - $current = (510 - 510)(0.244A) = 0$
    - $current = (527 - 510)(0.244A) = 3.9A$
- With a range of only 3.9A, better resolution can be achieved by instead using milliAmps. This changes the formula to: $current = (analog - 510)(244mA)$

Figure 57, which is included below, shows the results of test one. The user power, which is shown in blue and displayed on the secondary (right) axis, was measured by the power meter pedals in Watts. Current, which is shown in orange and displayed on the primary (left) axis, was measured by the current sensor and is measured in mA. "User power" refers to how much power the user is inputting to the bike through the pedals, and "Current" refers to the amount of current that is drawn by the motor as it increases its assistance to the user; this current draw is proportional to the power output of the motor.

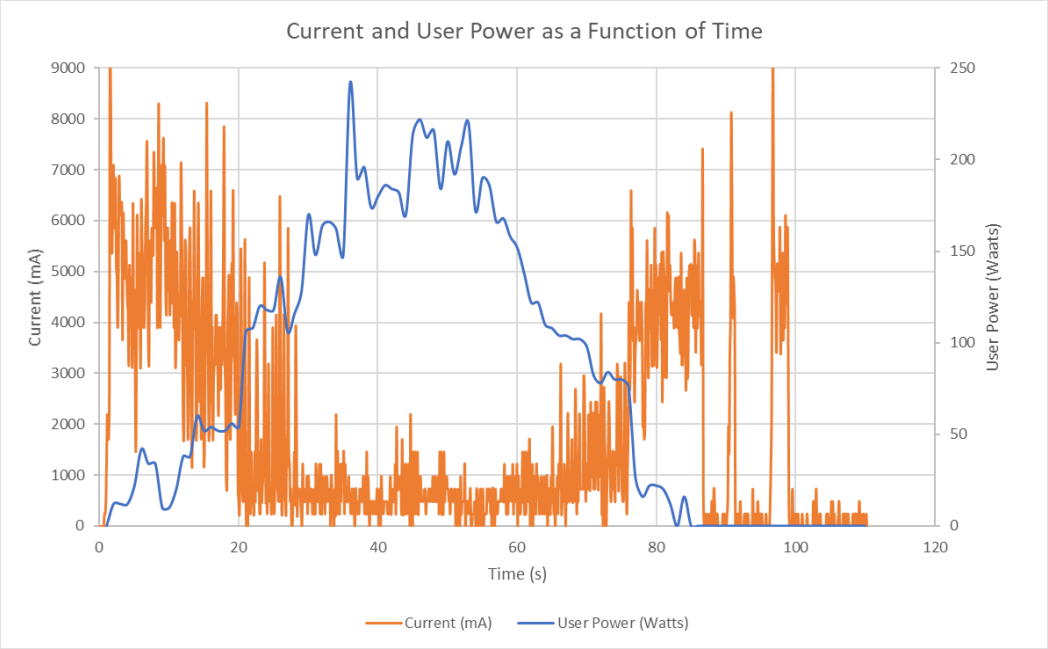


Figure 57: Current and User Power as a Function of Time

Figure 57 shows the sampling of the current at a frequency of 10Hz resulted in a significant amount of noise, making the data extremely difficult to interpret. Because of this, each set of five current values were averaged; essentially, this adjusted the current sampling frequency from 10Hz to 2Hz. The results of this plot are shown below in Figure 58.

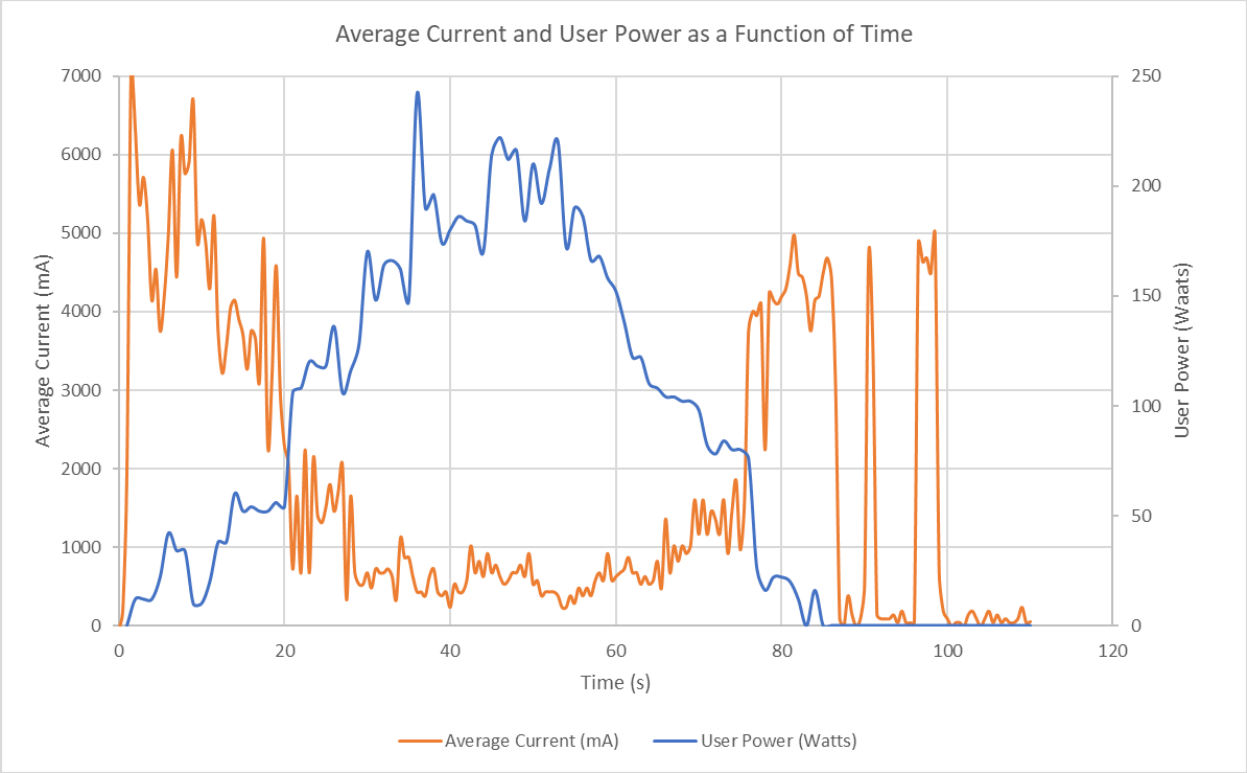


Figure 58: Average Current and User Power as a Function of Time

Figure 58 shows the averaging the data results in a graph that is far easier to understand. Because the relationship between the user's power input and motor assist is given by the equation, $Analog\ Assist\ =\ (Max\ Power\ -\ Power)\ \times (255/(Max\ Power\ -\ 1))$, it makes sense that as the user's power decreases, the current increases. It was also anticipated that when the user's power input returns at zero — which can be seen at the end of the graph — the current also returns to zero.

However, it's often easier to envision two functions that have a similar trend than two functions that have an indirect relationship. Because of this, Figure 59 was created. Figure 59 keeps the 2Hz sampling frequency of Figure 58, but uses the defined relationship between $Analog\ Assist$ and $Power$ to convert from $Power$ to $Analog\ Assist$

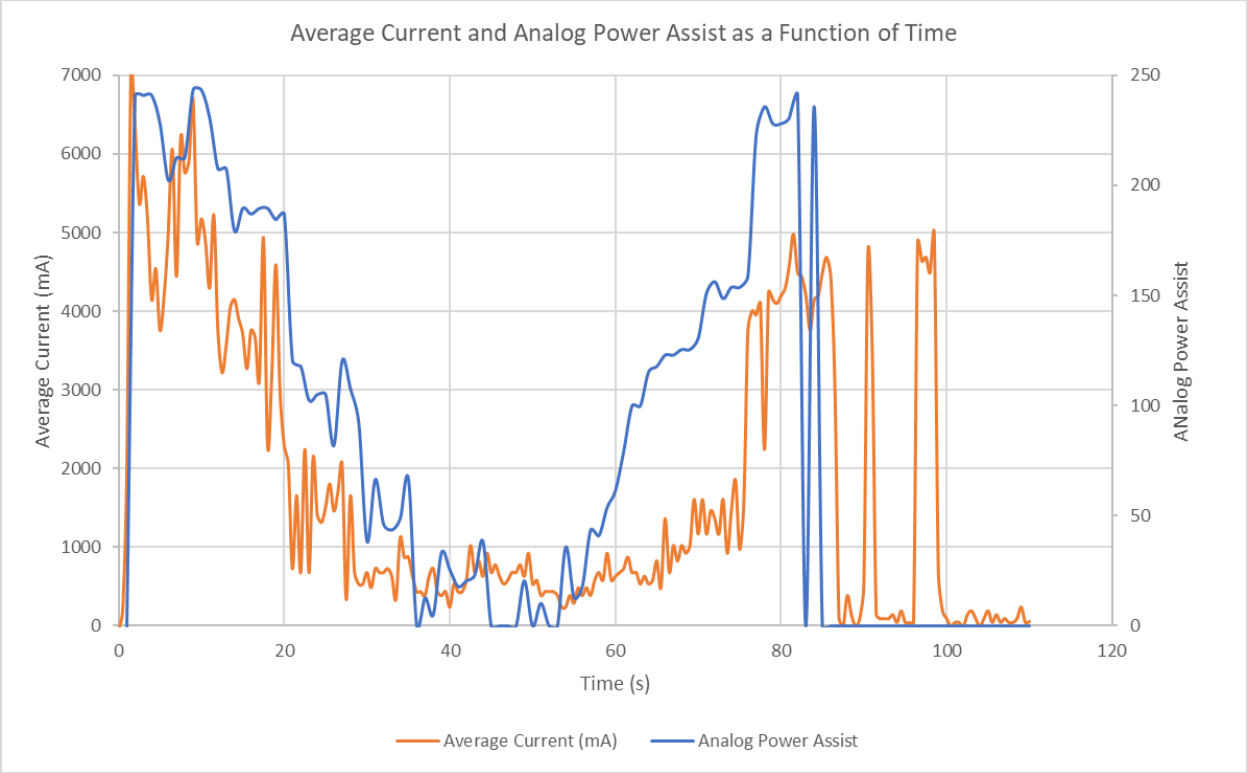


Figure 59: Average Current and Analog (programmed) Power Assist as a Function of Time

Figure 59 shows that there is a clear cause-effect relationship between the ESP32 calculating a new power assist value for the motor and the motor adjusting the amount of current its drawing as a response. However, the decision to use real and unfiltered data power a rider resulted in a graph that, while representative of real-world results, makes it difficult to quantitatively analyze the motor's response. Because of this, Test 2, which replaced the rider power data with a square wave, was conducted.

Test Two: Monitoring the current output of the motor with a square wave input

Compared to the test setup for Test One, the test setup for Test Two was extremely simple. This is because, instead of the Sender ESP32 having to communicate with the power meter pedals via Bluetooth to retrieve the user's power data, the data was preset into the ESP32 Feather. Similarly, there was no need to use ESP-NOW communication to send the power data to the Receiver ESP32 Feather since this data was already known.

A frequency of 0.1Hz was chosen for the square wave to ensure that the wheel would come to a complete stop between each cycle. This way, the response of the motor from rest would be evaluated. Similarly, this would ensure that the wheel was at its maximum velocity before the motor assist was set to zero.

The two values given to the square wave correspond to user power values of 200 Watts and 1 Watt, for a rider that is assumed to have a maximum expected power of 200 Watts. As explained previously, when the user inputs 200 Watts of power, a motor assist value of 0 will be given. Conversely, when the user inputs 1 Watt, the maximum assist value possible will be given; on an analog scale, this maximum value is 255.

With this information decided, Test 2 was conducted. Just like in Test 1, sampling the current at a frequency of 10Hz yielded extremely noisy data; as a result, the same method of averaging every five values was used. Additionally, the similar tactic of displaying the calculated analog power assist of the motor, as opposed to explicitly showing the user's power input, is employed here. The results of this test are shown below in Figure 60.

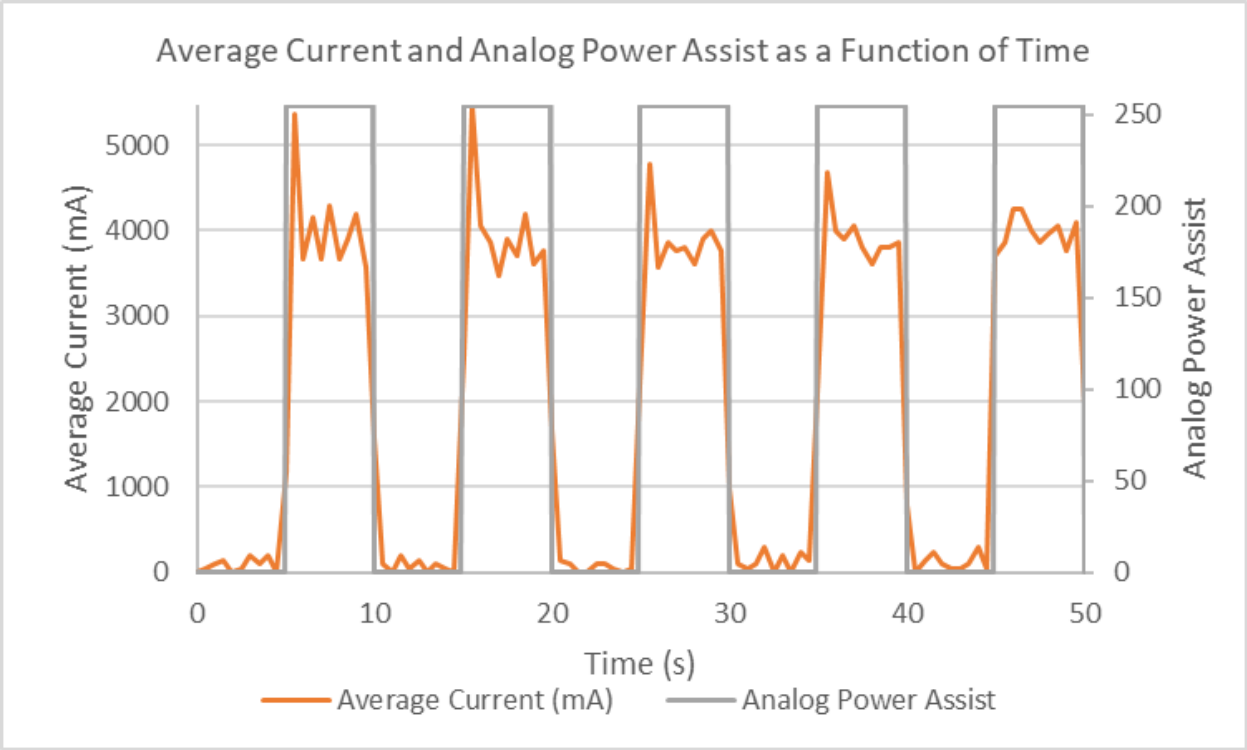


Figure 60: Average Current and Square Wave Analog Power Assist as a Function of Time

Much like the results of Test One, Figure 60 shows that a change in the analog power assist calculated by the ESP32 does induce a clear change in the current drawn by the motor. And by turning the user's power input into a clean square wave, this relationship is even clearer than it was in Test One. It should also be noted that there is no problem with the average current signal appearing below the analog power assist signal on the "high" points on the figure, as the two signals are in completely different units.

Though not shown in Figure 60, this test allowed us to determine the time it takes for the motor to respond to a change in the motor assist value. This time constant is extremely important, since

it represents the important need that “the motor makes quick auto-adjustments”. This time constant was approximately determined by going through the data and observing how much time passed between the motor assist value changing from high to low (or vice versa) and the motor current changing as a result. For example, the first change from low motor assist to high motor assist occurred 5.0 seconds into the test. At 5.0 seconds, the measured current was still 0A. At 5.4 seconds, it was still only 0.976. Finally, at 5.5 seconds, it reached 4.392A, which, as Figure 60 shows, is approximately the steady value for current. Therefore, this sample yielded a time constant of 0.5 seconds.

We conducted multiple test trials to verify these results (shown in Tables 11 and 12). Table 11 is for each time that the motor assist signal goes from low to high while Table 12 is for each time that the signal goes from high to low. Each table contains two different tests: Test 1 and Test 2. This was done because the current measurements were aligned with the power input square wave by resetting the Arduino Uno, which measures the current values, and the ESP32, which sends the power assist values to the motor, at the same time. However, it is possible that there was some human error involved in resetting them at a slightly different time. Therefore, this test was replicated, although with a lower number of samples.

Table 11: Low-to-high time constants for the motor

Overall Number	Test	Time that low-to-high occurs (s)	Time Constant (s)
1	1	0.5	0.5
2	1	1.5	0.4
3	1	2.5	0.4
4	1	3.5	0.4
5	1	4.5	0.3
6	2	0.5	0.4
7	2	1.5	0.4
8	2	2.5	0.3

Table 12: High-to-low time constants for the motor

Overall Number	Test	Time that high-to-low occurs (s)	Time Constant (s)

1	1	1.0	0.3
2	1	2.0	0.3
3	1	3.0	0.2
4	1	4.0	0.2
5	1	5.0	0.2
6	2	1.0	0.2
7	2	2.0	0.2
8	2	3.0	0.2

Based on the results in Table 11 and Table 12, the mean time constants for the low-to-high case and high-to-low case can be calculated.

Low-to-high time constant mean: $\frac{\sum_{i=1}^{i=n} x_i}{n} = \frac{\sum_{i=1}^{i=8} x_i}{8} = 0.3875s \approx 0.4s$

High-to-low time constant mean: $\frac{\sum_{i=1}^{i=n} x_i}{n} = \frac{\sum_{i=1}^{i=8} x_i}{8} = 0.225s \approx 0.2s$

Additionally, the minimum differences between the two tests reinforce the likelihood that both the Arduino Uno and ESP32 Feather were reset at the same time. Because of this, both tests were able to be combined to calculate the overall mean. Regardless, when considering the needs and specifications that are documented earlier, error on the order of 100 milliseconds is not particularly relevant. Overall, both of these time constants satisfy the needs described earlier.

Test Three: Monitoring the current output as a function of the pedal assist level used

The team manipulated the wiring of the bike so that pedal assist would be disabled. However, it was found that the level of pedal assist chosen was still a multiplier on the power associated with the potentiometer, which is what the ESP32 Feather replaced to control the motor. For example, if the pedal assist is set to zero, all outputs of the ESP32 Feather would result in zero motor power, since all values would simply be multiplied by zero. For the tests shown above, the pedal assist level was set to three (out of nine total levels) so that it wouldn't be a variable that could influence the results of the test. However, it is still important to observe how manually altering the pedal assist affects the current drawn by the motor.

In this test, the 0.1Hz square wave that was used in Test 2 is still being used. The only difference was that each time the motor assist level went back down to low, the pedal assist level was increased by one. The current readings associated with each motor assist level have been averaged and are shown below in Table 13.

Table 13: Average Current Drawn by the Motor as a Function of Pedal Assist Level

Pedal Assist Level	**Current (mA)**
0	0
1	1337
2	2464
3	3811
4	5368
5	7096
6	9335
7	10858
8	13210
9	15148

Table 13 shows, the pedal assist level chosen has a significant impact on the amount of current drawn from the motor. While this project did not investigate any benefits of this, it is nonetheless important to note. By not using this feature in our project, the number of bikes that can be used to replicate this system drastically increases, as the bike does not need to possess the pedal assist system that the bike used in this project possesses.

Cost & Manufacturing Analysis

Bill of Materials

Table 14: Bill of Materials

System	*Subsystem*	*No.*	*Part Name*	*Manufacturer*	*Part Number*	*Qty*	*$*	*Material*
Mechanical	Bike Frame	1	26in Mountain Bike	Kent (dist. Walmart)	2648	1	$198.00	Steel
	Electric Motor	2	750W Brushless Geared Mid-Drive Motor	BaFang	BBS02	1	$899.99	
Sensing	Biometric Mode	3	Heart Rate Monitor	Coospo	43238-69716	1	$29.99	
	Power Mode	4	Power Meter Pedals	FAVERO Assioma	772-02	1	$699.99	
Electronics	ESP32	5	ESP32 Feather Huzzah	Adafruit	3405	2	$41.89	
		6	Adafruit Batteries	Adafruit	1578	2	$15.90	Lithium ion
Hardware	Battery	7	Battery Mount	Manufactured		1	TBD	ABS
	Phone	8	VUP Universal Phone Mount	VUP	69493123.22332	1	$13.99	Silicone
	Electronics	9	Neoprene Rubber			2	$31.87	Neoprene
		10	Electronics Mount	Manufactured		1	TBD	ABS
		11	Electronics Mount Lid	Manufactured		1	TBD	ABS
		12	Mount Brackets	Manufactured		2	TBD	ABS
Other	Phone	13						
					Total		$1,931.62	

The cost of the manufactured parts will be added based on invoices from the rapid prototyping lab.

Cost of Product Operations

For this product, the cost of product operations has been analyzed, rather than the cost of scaled product manufacturing. Since the e-bike is being used to demonstrate proof of concept for Altair's SmartWorks software, there is no current intention to scale this product into something mass produced. As a proof of concept, the prototype development time in many areas is assumed to be equivalent to the time of building the part .

To develop this product to the level of the prototype, there are certain steps and labor hours that can be predicted, especially since the team is working with some off-the-shelf products.

The inputs for this product are as follows:

- Development cost and timing
- Labor cost and timing
- Cost of rapid prototyping
- Cost of materials (shown in BOM above)

Table 15: Cost of Product Operations Analysis

Part	**Labor Time**	**# of People**	**Cost (hours*people* cost/hr)**	**Description**
Bafang Mid Drive Motor	8 hours	2	$320	Installing the mid drive motor kit to the selected bike frame.
Motor Control Circuit	1 hour	2	$40	Wiring and soldering of each microcontroller, sensor, and chip used in the product.
Rapid Prototyping	TBD	–	TBD	Both the electronics casing and the battery mount adapter are manufactured and rapidly prototyped out of ABS.
Software Programming	20 hours	4	$1600	All of the microcontrollers need to be programmed in order to control the motor and send/receive data.
Website Development	26 hours	1	$520	In order to display the proof of concept data, a website needed to be developed.

The labor costs were calculated with an assumption of a $20/hour pay rate per person, but this may not be consistent if the product were to be developed elsewhere.

When considering the costs in the context of a design-a-thon situation, the labor costs would no longer be a factor, but the time taken to develop the product software-wise will no longer be in scope, considering the programming time is at approximately 46 hours.

Patent Claims Documentation

Utility Patent – Method for Control of an Electric Bicycle Using Cloud Software Feedback Loop

This patent covers a control system for an electric bicycle (Figure 61) based on biometric user feedback. The electric bicycle physical control system consists of the bicycle, the motor, the batteries, the microcontrollers, and the biometric sensors. The digital control system includes the programmed microcontrollers, the Internet of Things cloud software, the Bluetooth and Wifi connections, and live data visualization. Both of these systems in whole work to process data from a user through the sensors in order to control the e-bike while simultaneously sending data to the cloud to be processed and visualized.

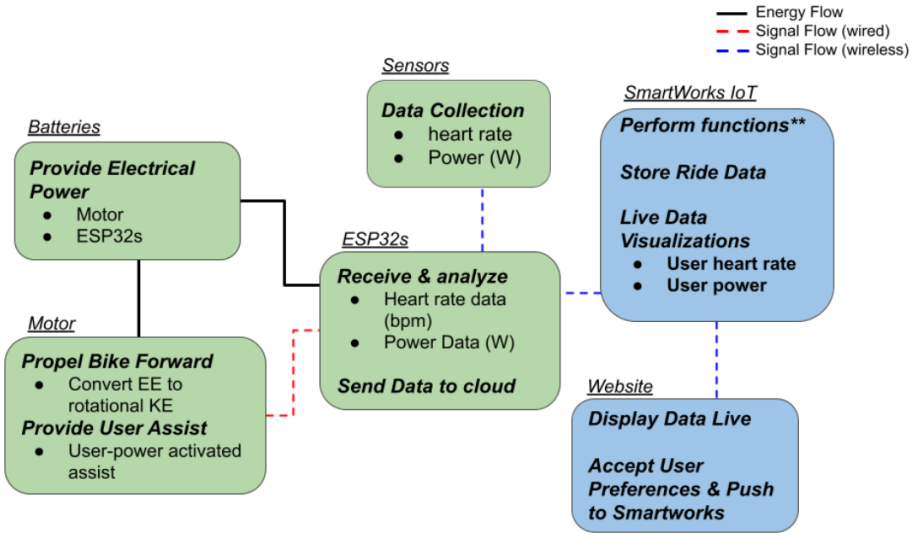


Figure 61: Patentable E-Bike Control System

Claims

1. A control system for controlling an electric bicycle comprising:

a. a bicycle comprising a do-it-yourself (DIY) motor kit assembly wherein the bicycle is controlled by an electric bicycle motor

b. one or more microcontrollers comprising a digital feedback system

c. one or more biometric feedback sensors for receiving biometric data from a rider and providing the data to the microcontroller

2. A software control loop for controlling the DIY motor kit of claim 1 comprising:

    a. the digital feedback system of (b)

    b. a Bluetooth and a wifi connection

    c. IoT cloud software for visualizing the data

    d. an HTTP protocol for pulling the data from the IoT software to a data visualization website

3. Heart rate and power meter sensors for data collection and monitoring for biometric feedback, wherein the rider's biometric data is fed into the aforementioned software control loop to create varying levels of motor assist.

4. The control loop of claim 2, wherein a microcontroller connected either physically or digitally controls all flow of data to the bike, and from the biometric sensors

5. The control loop of claim 4, wherein the motor assist level is selected based on preset algorithms associated with heart rate levels based on a rider's age (BPM) or pedal power based on a rider's expected maximum power input (Watts).

6. A digital feedback system for processing of data from biometric sensors wherein the microcontroller control loop of claim 5 is programmed to determine the levels of motor assist given to the ride wherein the biometric data from claim 3 is processed to associate data levels with motor assist levels.

7. The control loop of claim 5, wherein the data from the feedback system of claim 6 is fed to the Cloud via Wi-Fi connection. The data is processed using IoT software to create visualizations.

Future Work & Recommendations for Client

After twenty-two weeks of development and the conclusion of this project there remain some areas of the project and product that could be improved and iterated upon. Below we will go into the different areas where improvements could be made to create a more full, marketable product. Most of these improvements revolve around creating a more user-centered product or making

improvements surrounding the incorporation of Altair's SmartWorks technology into the final product.

Human Centered Design Considerations

Deciding exactly how and why an e-bike such as the one we created would be used is essential in the development of a more user-centered bike. As a team we determined that an older adult population that may not be able to bike like they used to, but wants to, may be a group that would benefit from a bike that would regulate motor assist according to biometrics. Examining this persona centered around Uncle Mark (Appendix A) and their needs and wants out of an assisted ride experience would help to develop a more successful user-centered product. We have listed a few different considerations below that may improve the rider's experience.

Biometric Sensors

On the part of the design team or other future design teams, the prototype can be developed to include other biometric sensors, such as a breath rate sensor, to add other modes to the system to aid the rider. The addition of sensors could allow for a more cohesive and complete prevention from over exertion. Including other metrics gathered from the user including energy expenditure (calorie-burn), could also allow the user to self-regulate their assist and effort.

Motor Support Algorithm

Improvements could be made to the way that heart-rate and user power data is used to gauge motor assist. As mechanical engineering students, we have a limited understanding of the relationship between heart rate and cardiovascular exertion. Our algorithm seeks to scale motor assist according to one of the five heart rate zones. This could be improved in a variety of ways, and rehabilitation experts would be most knowledgeable on how this should be done whether that be making assist continuously scale, rather than having discrete levels. The heart rate motor assist algorithm is also only a function of the rider's age. There are a number of factors that affect a users maximum heart rate so this should be investigated further to make sure that levels are set and scaled appropriately.

The power meter motor assist algorithm is only a function of the rider's expected maximum power input. This is currently preprogrammed into the code and should be able to be changed by the user from day to day. A better algorithm may also be able to create suggestions based on the time and topography of the ride.

In the future, algorithms that incorporate multiple rider biometrics, such as heart rate, input power, respiration rate, etc. could be more effective in developing a safer, more helpful algorithm.

Finally, machine learning could be incorporated into the system so that the e-bike would "learn" how to best support the rider. The network could be trained based on the user's previous ride data so that the system could become more effective with use. This would most likely require more feedback from the user such as how tired they feel upon completion of the ride and how hard they want to exert themselves. The success of this model is largely dependent on the amount of data that is able to be collected from a variety of users. Additionally, SmartWorks does not allow for additional Python libraries to be imported, so in the future, if added, machine learning algorithms can be performed in the cloud.

User Testing

To bring the bike to market, user testing would need to be conducted on a number of different systems including the user interface, comfort of the bike itself, and perhaps a sort of clinical trial to investigate the reduction of over exertion. In these tests we would want to investigate user comfort riding the bike generally and specifically how comfortable and natural the "kick-in" of the motor assist feels. Testing should be conducted to determine how intuitive setting up a ride and selecting a mode is for our target user group.

Bike Hardware

There are a number of improvements that could be made to the bike hardware itself to allow for this product to be brought to market and satisfy more user considerations. Currently, the electronics casing and motor are not water tight. Traditional bikes are designed to be durable and resilient in a number of different environments and weather conditions. Water, because of rain or puddles, would damage the electronics and motor (rendering the bike useless and extremely heavy.

The wires and other large electronic components could also be tucked into or incorporated into the frame. This would give the bike a cleaner look overall and improve the aesthetics of the final product on top of providing extra protection for the critical components of the system.

Other hardware to make the user more comfortable as they ride, such as a water bottle holder, would make the product more attractive to users as well.

Software & the Cloud

Cloud Control of Motor & Calculations in the Cloud

A more long-term development would be achieving cloud control of the motor, rather than controlling the motor locally. This will allow for a further display of the capabilities

of SmartWorks in both pushing and pulling data in order to create the feedback loop necessary to achieve motor control. By utilizing the functions hosted on SmartWorks, it can read in data relevant to motor control and return a motor assist level output calculated by the data it received. The Function's programming is in Python, it can be useful for writing programs to perform any customizable calculations in the cloud. As of now, the Functions Python library is limited to a few, so any machine learning algorithms cannot be completed there, but is a feature that is being developed according to Altair engineers.

Use cellular or 5G connection (rather than WiFi)

Using WiFi to connect to the internet is not very practical, as this product would be used outside, where there is no WiFi. It is much safer to assume that the user will maintain a cellular connection throughout their ride. Even better, implementing a 5G connection would be more reliable. Additionally, if we decided to replace the screen with a physical screen (instead of a web app viewed through a phone), then the rider will not need to bring their phone with them on their ride.

Client Recommendations

As far as recommendations for our client:

- Allow more functions to be hosted in a Space (currently 4-5 functions)
- Address the steep learning curve for Panopticon
- Improve Panopticon UI/UX
    - Add features for better adjustment/sizing of entities in the dashboard (i.e. graphs, images, text boxes, etc.)
    - could benefit from an auto alignment feature
    - the discrepancy in how the dashboard looks from "view" and "edit" mode can be an obstacle
- Add more time-series analysis graph and charts
- Allow for libraries to be imported in the Functions Python3 code
    - machine learning algorithms, pandas, numpy
- Provide additional documentation for users
    - Some tutorials teach how to do a specific project but do not show how to go beyond the tutorial as effectively

See Appendix G for more detailed documentation on the issues we ran into.

Acknowledgments

The team would like to thank Prof. Gatchell, Prof. Beltran, and Jesse Brown for their mentorship and guidance throughout the project process. Thank you to Professors Peshkin and Marchuk for sharing their mechatronic expertise and components, and for their assistance and advice in the creation of our final prototype.

We would like to acknowledge Darius, Armin, Kruti, Alvaro, and Paloma of Altair for all of the help on SmartWorks and Panopticon. Thank you for your encouragement and technical support throughout the entirety of the project.

We would like to thank all of the people who have assisted in the design, research and creation of our smart e-bike including, but not limited to, Scott Simpson, Uncle Mark, Prof. Geiger, Joey Galindo, MyEyeToTheSky on Reddit, and neptronix on the Endless Sphere Forums.

Appendix A: Expert Interview & Persona Identification Research

Summary of Findings:
We conducted several different interviews, found below to learn more about e-bikes, the e-bike market, and the possible users and uses for the smart e-bike technology. We have interviewed five members of the Northwestern triathlon and cycling teams, e-bike rider and bike expert Mark Ernst, e-bike commuter Prof. Franz Geiger, and a cyclist at *Altair,* Thomas Kitzler, to learn more about e-bike technology, the e-bike market and the target e-bike user group.

Our interviews with Northwestern cycling and triathlon team members helped us to better understand that the original intended market for the bike, young athletic individuals that cycle or want to cycle, may not be the target group for using active feedback loop technology. We found that these cyclists had no intention of investing in an electric bicycle as long as they remained healthy and instead would rather invest in a bike that they could better use for training and/or races. However, these interviews did point us in the direction of our current target user and the basis of our persona when one triathlete, Kate Stumpf, mentioned that one e-bike user she knows primarily does so due to a decline in health with age and injury. This led us to interview e-bike expert and enthusiast, Mark Ernst, pictured in Figure 62.

Figure 62: Mark Ernst, Target User Persona

Through interviews with e-bike expert and enthusiast, Mark Ernst, we were able to determine our target user group: individuals that "cannot bike like they used to, but want to" or older e-bike riders that are no longer able to exercise in the way that they used to for health related reasons. Ernst is a 64-year-old ex-bike shop owner who has had a double knee replacement, hip replacement, and survived cancer. Ernst is an endurance sport athlete and has previously competed in running, cycling, triathlons, and cross-country ski races. In his interview, Ernst emphasized that before purchasing his first e-bike, he was uncomfortable riding traditional mechanical bicycles and he was unable to keep up with his sons, nieces, and nephews. Ernst currently "bikes for grins" more so than to get an intense workout, and enjoys having extra motor assist as he pedals uphill or when he gets tired. This persona was crucial to the solidification of our problem statement, as we were able to pick out that our typical user would need the biometric and environmental feedback to be able to keep up with family members on bike rides without injuring themselves or overexerting themselves.

In interviews with Prof. Franz Geiger and *Altair's* Thomas Kitzler (Appendix A) we've determined that for both recreational and commuter e-bike riders, comfort and "the feel" of the bike is a major focus. Geiger repeatedly mentioned that he wanted his e-bike to look and feel like a standard mechanical bicycle as much as possible. Kitzler mentioned that the weight and handling of the bike were important factors. Riders want their ride to be smooth (no motor "kick") and for the bike to not be excessively heavy. The noise disturbance of the bike to the rider and others should also be minimized. Many e-bike users, including everyone we interviewed, have mentioned comfort and aesthetics of the bike are highly important factors which we have incorporated into our needs statements.

Appendix B: Experiential E-Bike Research

Research Summary

The goal of the experiential research is to gain first-hand experience riding an electric bike.

Team members Jake and Maddie rented Divy/Lyft electric pedal-assist e-bikes for approximately 40-45 minutes. Riders navigated walking/cycling paths, bike lanes, roads, rougher trails and grassy areas in Evanston, IL. Riders qualitatively noted their previous experience with ebikes before detailing their general riding experience. The rides were mapped by the lyft app & Maddie's watch.

Bike Background:

Divy/lyft ebikes are pedal assist electric bicycles (no throttle is incorporated). Each stroke of the pedal is supported by the motor. The bikes have a maximum speed of 20mph. The Divy/lyft bike is considered a Class 1 ebike.

Maybe we add a pic of the bike gears

There are multiple resistance settings on the ebike, broken down into simple drawings to get meaning across to the user quickly. By turning the resistance setting knob away from the user (towards the biker on a hill), the ebike had lower resistance, enabling the user to go off trail or up a hill. By turning the resistance knob towards the user (towards the biker on a road), the ebike had greater resistance, bringing in the pedal support much more quickly and enabling the user to reach greater speeds.

https://www.divvybikes.com/how-it-works/ebike
https://www.lyft.com/blog/posts/meet-lyfts-new-ebike
https://www.lyft.com/bikes/chicago-il/meet-our-bikes

Qualitative Experience

Maddie:

- This was my first time riding an ebike. I've seen them in shops, but have never been on a ride with anyone on an ebike.
- Bike was heavy, difficult to begin pedaling until assist kicked in
    - Estimated to be 40lbs when attempted to pick up
    - Very top heavy - felt wobbly/precarious
        - Battery pack was located super high on the bike
- Felt like you dont know your own power and it could be easy to get out of control (even at the end of the ride)
- There was always assist - the bike would be almost impossible to pedal without it
    - Due to weight, aerodynamic

- Bike was difficult to stop (right brake malfunctioning possibly?)
    - Felt like there was a lot of momentum
- Bike was helpful going up the biggest hill we could find
    - very little exertion and no extreme pressure/ “front loading” of knees (uncomfortable due to previous knee injury)
- E-bike pedal assist helpful on the “off-roading” parts of ride
    - Bike lacked suspension - very bumpy/vibrational, uncomfortable ride
- Ebike was fast - we were able to cruise at 20mph (the bike’s max speed) -
    - which is probably about the fastest I’ve ever been going on a bike
- Very little exertion required. We did not aim for this to be a workout, but I do not know if it could have been? Pedal assist was there with every stroke.
    - Average heart rate = 101bpm

Jake:

- “If you hit someone, it would hurt”
    - These bikes had about the weight that I was expecting, but were harder to maneuver than I anticipated
- It took time to get used to starting the bike
    - First time riding an ebike, had seen plenty of people use them from divvy
    - Pedal assist kicks in quickly
    - Weight requires a lot of force to convert into momentum
- ebike got to top speed pretty quickly and was able to hold speed
    - Large weight probably leads to lots of momentum
    - Didn’t break a sweat during the ride
    - Lots of power, damage would be done with collisions
- Difficult to gauge assist
    - I felt like I was going the same speed the whole time, however, the pedal assist would come and go, almost in a wave-type function
    - Visual feedback of how much help you are getting could be incredibly helpful
- Challenging terrain
    - Absolutely no difficulty riding and maneuvering the bike off road and up the largest hill we could find (norris and path by library)
    - Off roading showed that the bike had very little to no suspension
- App experience
    - I thought that the lyft app was easy to use, provided nice real time data, and looked visually appealing
    - Maybe if it displayed on the bike the stats (distance, charge, cost) could be really useful
    - Ability to purchase yearly pass enables scan and go with ease

Figure 1: Jake on Ebike

Geopositional Information:

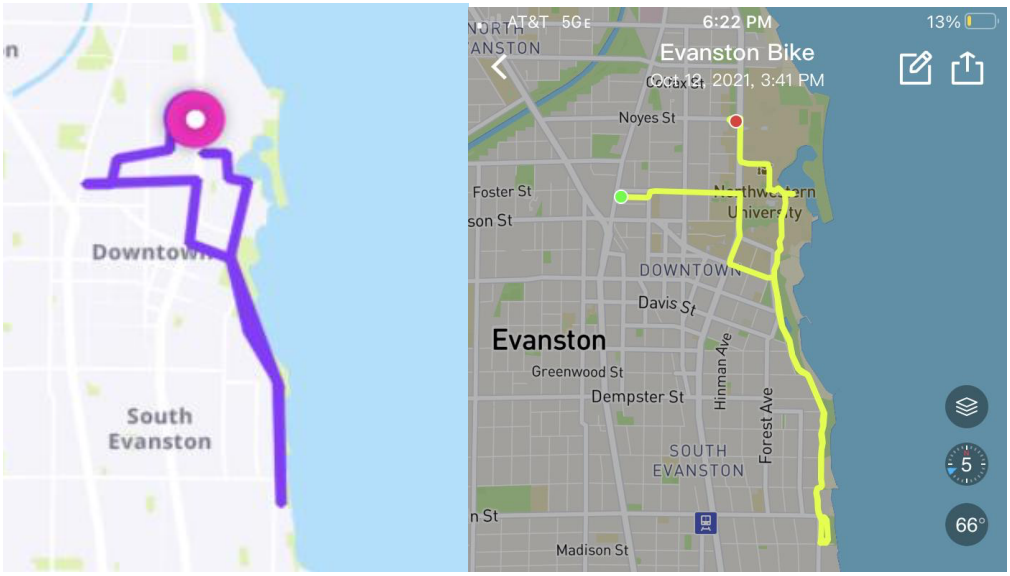


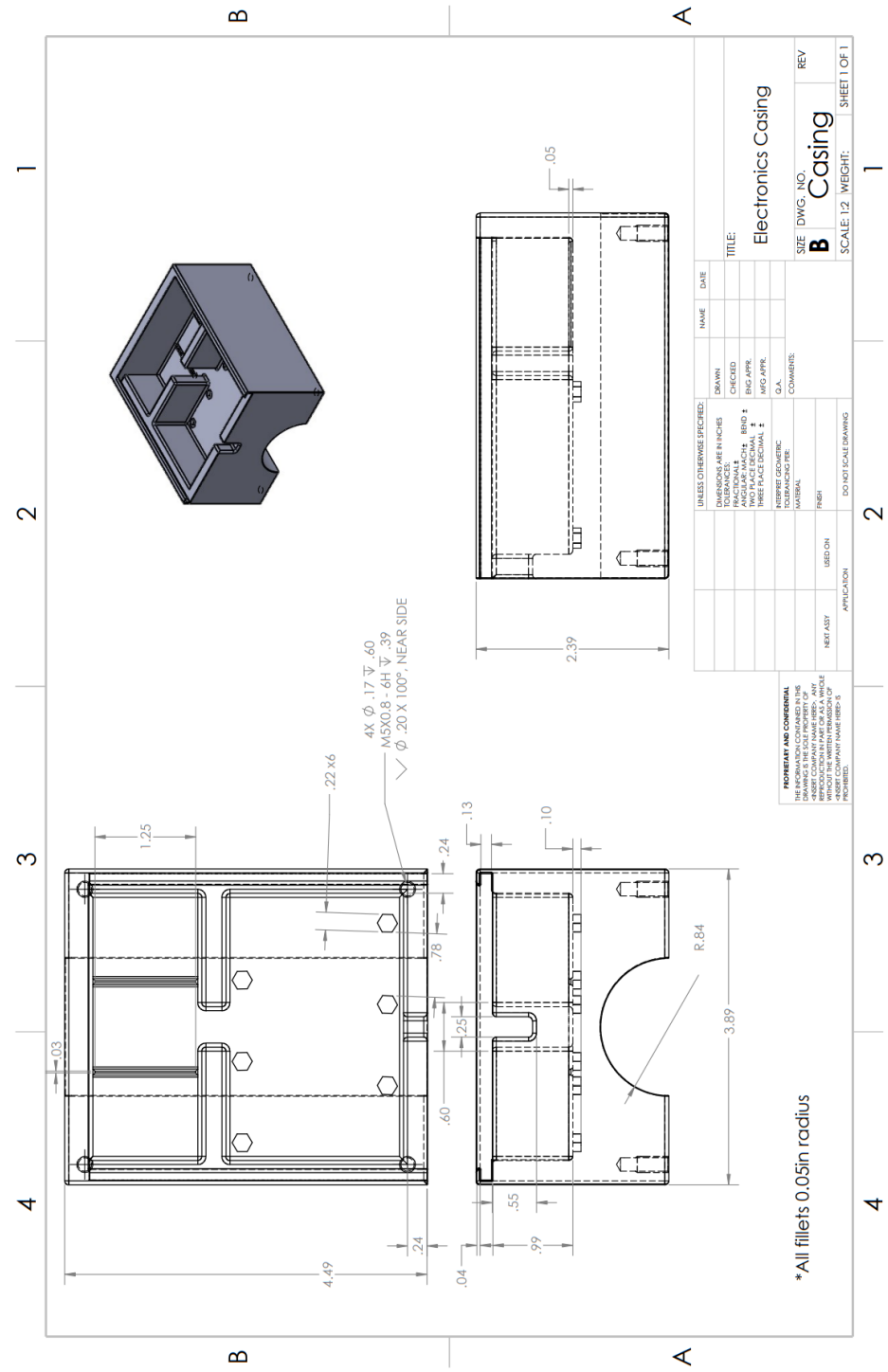
Figure 2: Map of ride

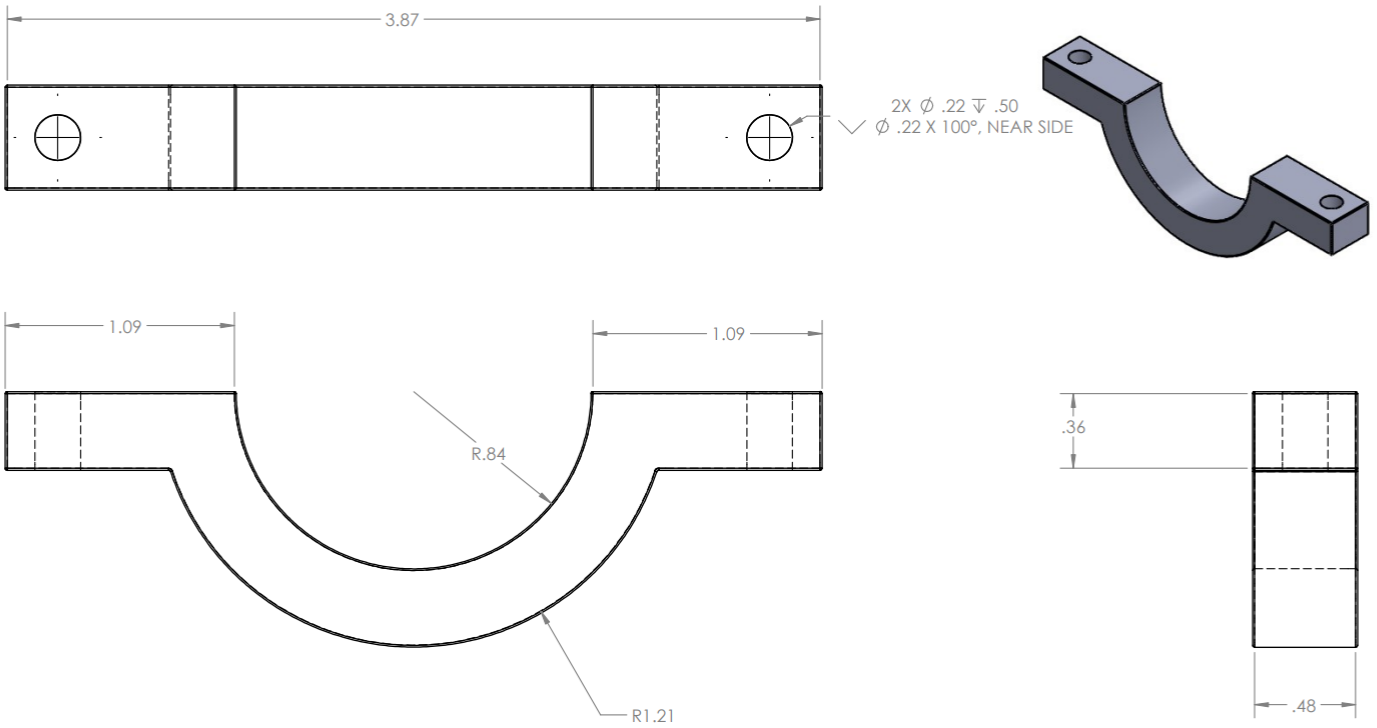
Geopositional Data:

Data was sourced from lyft phone application & watch (Coros)

- Ride time: 3:33pm - 4:16pm (43 minutes)
- Average pace = 7.97 mph
- Max pace = 19.5 mph
- Distance ~ 5miles

Appendix C: Manufactured Parts CAD Drawings



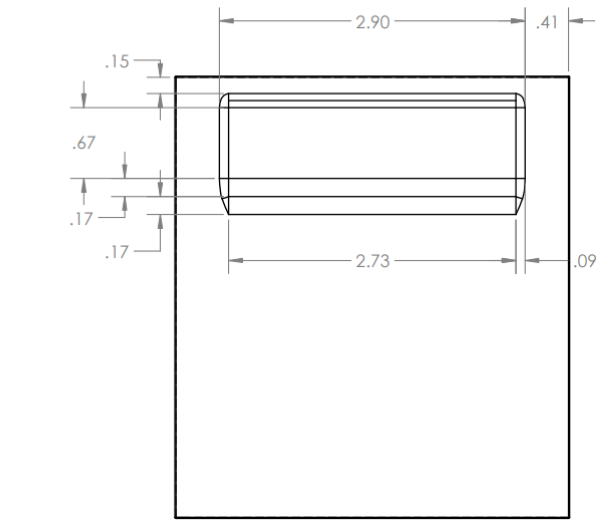


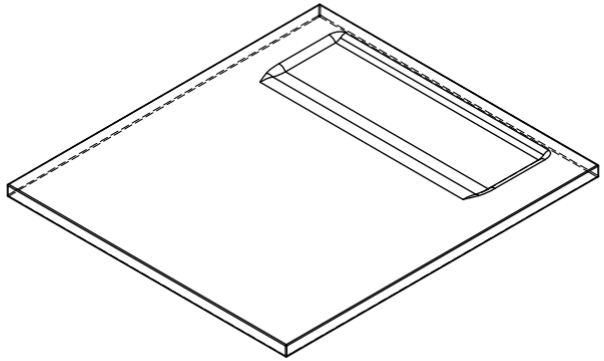
		UNLESS OTHERWISE SPECIFIED:		NAME	DATE
		DIMENSIONS ARE IN INCHES TOLERANCES: FRACTIONAL ± ANGULAR: MACH ± BEND ± TWO PLACE DECIMAL ± THREE PLACE DECIMAL ±	DRAWN		
			CHECKED		
			ENG APPR.		
			MFG APPR.		
		INTERPRET GEOMETRIC TOLERANCING PER:	Q.A.		
		MATERIAL ABS	COMMENTS:		
NEXT ASSY	USED ON	FINISH			
APPLICATION		DO NOT SCALE DRAWING			

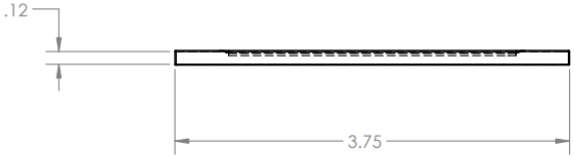
TITLE: Casing Mount

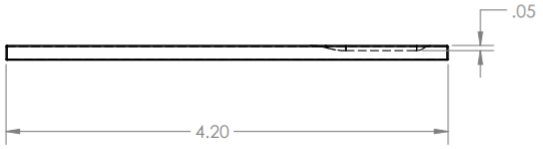
SIZE B | DWG. NO. Bottom_Mount | REV

SCALE: 2:1 | WEIGHT: | SHEET 1 OF 1





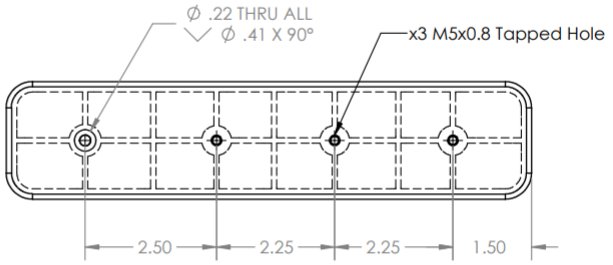


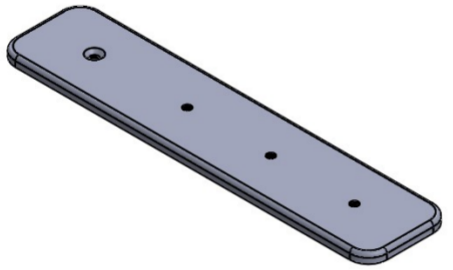


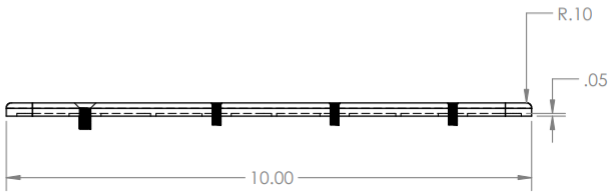
		UNLESS OTHERWISE SPECIFIED:		NAME	DATE
		DIMENSIONS ARE IN INCHES TOLERANCES: FRACTIONAL ± ANGULAR: MACH ± BEND ± TWO PLACE DECIMAL ± THREE PLACE DECIMAL ±	DRAWN		
			CHECKED		
			ENG APPR.		
			MFG APPR.		
		INTERPRET GEOMETRIC TOLERANCING PER:	Q.A.		
		MATERIAL ABS	COMMENTS:		
NEXT ASSY	USED ON	FINISH			
APPLICATION		DO NOT SCALE DRAWING			

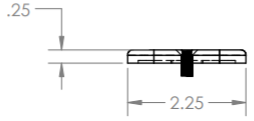
TITLE:
Casing Lid

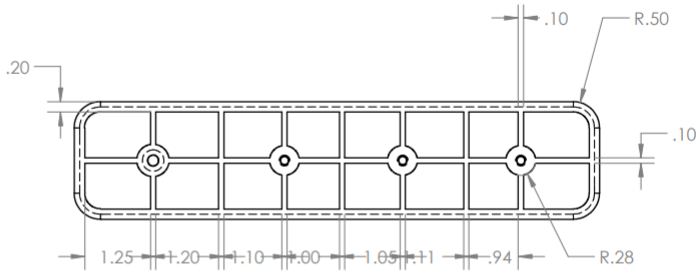
SIZE	DWG. NO.	REV
B	Updated_Lid	
SCALE: 1:1	WEIGHT:	SHEET 1 OF 1











		UNLESS OTHERWISE SPECIFIED:		NAME	DATE
		DIMENSIONS ARE IN INCHES TOLERANCES: FRACTIONAL ± ANGULAR: MACH ± BEND ± TWO PLACE DECIMAL ± THREE PLACE DECIMAL ±	DRAWN		
			CHECKED		
			ENG APPR.		
			MFG APPR.		
		INTERPRET GEOMETRIC TOLERANCING PER:	Q.A.		
		MATERIAL	COMMENTS:		
NEXT ASSY	USED ON	FINISH			
APPLICATION		DO NOT SCALE DRAWING			

TITLE: Battery Adapter

SIZE	DWG. NO.	REV
B	adapter	

SCALE: 1:2 WEIGHT: SHEET 1 OF 1

Appendix D: Power Meter Pedal Research & Pedal Details

Source: https://www.cyclingweekly.com/group-tests/best-power-meters-everything-you-need-to-know-35563

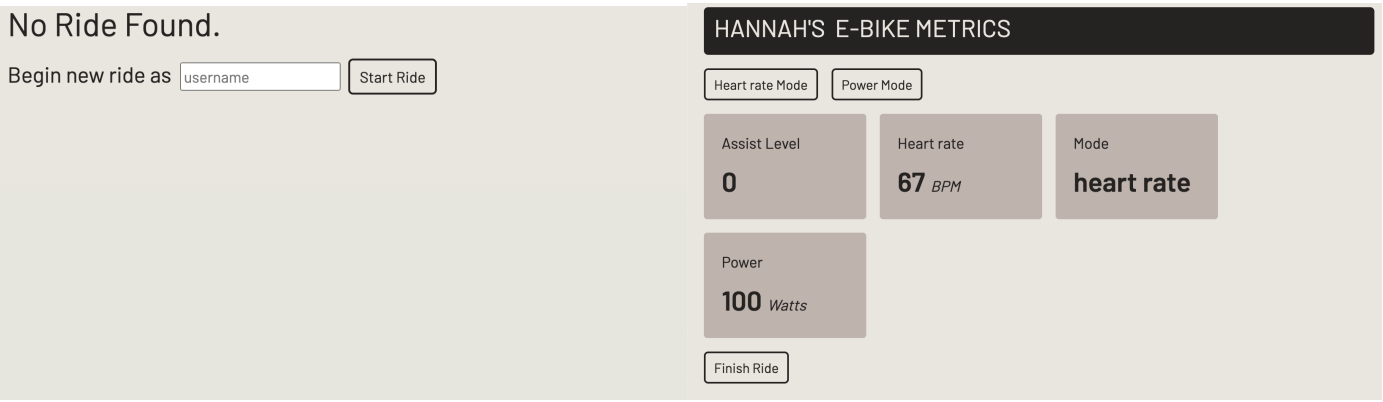
- Power meters can be placed on five areas of the bike
    - Hub-based
        - Generally regarded as the most accurate location to measure power. This is because there are fewer forces acting on the strain gauge here.
        - Power measurement is slightly lower than on pedal or crank since there is energy loss at the drivetrain.
    - Bottom Bracket
        - Accurate and low maintenance
        - Installation is more difficult.
        - Lack of standardization of bottom brackets makes it hard to find one that correctly fits the intended bike
    - Chainring
        - Generally very accurate
        - Similar compatibility issues to the bottom bracket
    - Crank arm-based
        - Easy to swap between bikes; no compatibility issues
        - Can either be single or double-sided
    - Pedal-based
        - Easy to swap between bikes
        - Considered less accurate due 'to the complexity of the force measurement."

- Details on the Favero Assimo
    - Average rating of 5 stars from 117 reviews
    - Installs just like a normal pedal
    - Communicates with ANT+ or Bluetooth SMART
    - Rechargeable battery; 50 hours of battery life
    - Accuracy of plus/minus 1.0%
    - "Will work with your favorite cycling computer, GPS watch, smart phone, or tablet"
    - Weight of each pedal is only 149.5 grams.
        - "This makes it the lightest power meter pedal available"
    - Undergoes strict control tests
        - "Load, shock, fatigue, wear, water resistance, etc"
    - Automatically calibrates each time

Appendix E: Web App Documentation

Introduction

This website was developed with React using Node JS. Its most prominent features include displaying real-time data from SmartWorks and allowing the user to send input to SmartWorks (change mode and start/end rides). The website communicates to SmartWorks through HTTP requests, which get data from SmartWorks things, change properties of SmartWorks Things, and invoke SmartWorks functions. The website may be viewed at https://ebike-app-e8677.web.app/, and the GitHub Repo can be viewed at https://github.com/hannahhuang00/ebike-app.



Website (left) start ride page and (right) data display page.

A tutorial by Northwestern's CS 394 class was used to begin the website (https://courses.cs.northwestern.edu/394/guides/intro-react.php#intro). By completing everything from the beginning through "Creating trackable state," we learned how to set up a React app, use the fetch API, and create states. However, this guide is still rather brief, and a lot of additional online help was needed (see Useful Documentation below). Additionally, I had to familiarize myself with Javascript and JSX, which are used in React.

The bulk of our implementation resides in the "App.js" file. In this file, we create the `bike` object, which is a dictionary of the properties we will pull from SmartWorks. The `App` function is responsible for fetching and displaying data, while the other functions are mainly responsible for the page layout.

HTTP Requests `fetch()`

`fetch()` Function Call

The `fetch()` function is a type of "Promise," which is only fulfilled when it receives a response. It takes the form

```
const response = fetch(url, method, mode, headers, body)
```

where the "url" is the endpoint of a specific API call. The arguments of fetch will be further clarified by explaining the various HTTP requests that the website utilizes.

Constants used in following parts

These constants will show up in the following code

```
const thing_url = 'spaces/ebike2/collections/esp32_data/things/';
const API_PATH = 'https://api.swx.altairone.com/';
const function_path = 'spaces/ebike2/functions/';
const app_id = 'app::01FXGQS619SXE3AWYBPY3XZBP0';
const client_secret = 'rLAwUUofwjfHYEfeOCL9gWmLnTuH4g';
```

Request Access Token

```
const result = await fetch(API_PATH + 'oauth2/token',{
      method: 'POST',
      body: 'client_id=' + app_id + '&client_secret=' + client_secret
            + '&grant_type=client_credentials&scope=function+data+thing+mqtt',
      headers: {
        'Content-Type': 'application/x-www-form-urlencoded',
      }})
```

Our first fetch must request a token from SmartWorks through the use of OAuth. To do this, a SmartWorks application must already be set up and linked to the Thing we are pulling data from. The application must be given the scope of function, data, thing, and mqtt because these are what we need access to. To setup an application, view this tutorial https://2021.help.altair.com/2021/smartworks/topics/access_control/apps_intro.htm#reference_a44_21c_p4b, making sure to set "Type" to "Client Credentials." We first request from SmartWorks OAuth (https://api.swx.altairone.com/oauth2/token) through the "POST" method. POST is similar to "writing" data, but specifically for creating new data. The body of our message is as follows:

```
body: 'client_id=' + app_id + '&client_secret=' + client_secret
```

This information should all be found in the app that has been created. The response to this fetch should be the access type and access token used to retrieve the desired data. This token is stored for the rest of the session to be used by all the functions, instead of generating a new token for every request.

Search for "opened" Rides

```
const result = await fetch(API_PATH + thing_url + '?property:state=opened', {
        method: 'GET',
        headers: {
          'Content-Type': 'application/x-www-form-urlencoded',
          'Authorization': token.token_type + ' ' + token.access_token,
          'Cache-control': 'no-cache',
        }}
      )
```

This is a “GET” request, which will read specific data from SmartWorks. In this case, we are returning all the rides that have a state property that is “opened.” Our SmartWorks has been set up so that each Thing is a separate ride. Each ride has a state that is either “opened” or “closed,” and only one ride can be “opened” at a time. This request constantly looks for opened rides so that it can retrieve its ThingID and begin displaying that specific ride data. Notice that we have to include the “Authorization” header, which includes the credentials from the token generated earlier.

Get Ride Data

```
const result = await fetch(API_PATH + thing_url + uid + '/properties' , {
        headers: {
          'Content-Type': 'application/x-www-form-urlencoded',
          'Authorization': token.token_type + ' ' + token.access_token,
          'Cache-control': 'no-cache',
        }
      })
```

This is another GET request, which returns the properties of a given thing. These properties include the real-time metrics of the user, as well as their assist level.

Starting a Ride

```
await fetch(API_PATH + function_path + 'start/invoke', {
    method: 'POST',
    body: JSON.stringify({"model": {"name": "user_ride","version": 5}}),
    headers: {
      'Content-Type': 'application/x-www-form-urlencoded',
      'Authorization': token.token_type + ' ' + token.access_token,
    },
  })
  .catch( err => console.log('startRide(user)', err))
  .then(data => data.json())
  .then(async data => {
    // SET DEFAULT PROPERTIES
    console.log(data)
    const uid_temp = data['uid'];
    console.log('uid_temp: ',uid_temp)
    fetch(API_PATH + function_path + 'updateproperties2/invoke', {
      method: "POST",
      body: JSON.stringify({"uid": uid_temp, "properties": {
        "assist_level": '0' ,
        "bpm": '...',
        // "calories_burned": 0,
        "power": '...',
```

```
        "mode": "heart rate",
        "state": "opened",
        "username": user
      }}),
      headers: {
        'Content-Type': 'application/x-www-form-urlencoded',
        'Authorization': token.token_type + ' ' + token.access_token,
      }
    })
    console.log(API_PATH + thing_url + uid_temp + '/mqtt-credentials')
    const credentials = await fetch(API_PATH + thing_url + uid_temp + '/mqtt-
credentials', {
      method: "GET",
      headers: {
        'Content-Type': 'application/json',
        'Authorization': token.token_type + ' ' + token.access_token,
      }
    }).then( resp => resp.json() )
    console.log('MQTT', credentials['data'][0]['id']);

    fetch( API_PATH + thing_url + uid_temp + '/mqtt-credentials/'+
credentials['data'][0]['id'], {
      method: "PUT",
      body: JSON.stringify({'password': 'ebike'}),
      headers: {
        'Content-Type': 'application/json',
        'Authorization': token.token_type + ' ' + token.access_token,
      }
    })
    .then( resp => resp.json())
    .then( resp => console.log('MQTT password response: ', resp))
    .then( () => setUID(uid_temp))
    .catch( err => console.log('updateproperties POST: something went wrong', err) )
  })
  .catch( err => console.log('POST: something went wrong', err) );
};
```

Starting a ride consists of multiple HTTP requests that are chained together. First, we invoke a SmartWorks function that will create a new Thing based on a “ride” model. Then, we immediately update the properties of the new Thing to default values, such that the user will know whether or not real-time data has begun sending, but more importantly that we open a ride that can be found by the API call mentioned earlier. We must also change the MQTT password

of the new ride to “ebike,” such that the ESP32, which is also attempting to connect to this newly created Thing, can always provide a constant password.

Ending Ride and Changing Modes

Both of these update individual properties in SmartWorks through a simple API call. Ending a ride updates “state” to “closed,” while changing the mode updates the “mode” to “heart rate” or “power.”

Other: `await()`, and `async()`

They are used for Promise functions, such that we can execute Promises in the background without needing to refresh the page or keeping a timer. `async()` allows running in the background without waiting for other functions to finish, while `await()` allows us to halt a function until a Promise has been fulfilled. This behavior saves us processing power, while also driving the updates through data.

Useful Documentation

JSX: https://reactjs.org/docs/jsx-in-depth.html
Fetch API: https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API/Using_Fetch
useEffect() Hook: https://reactjs.org/docs/hooks-effect.html
setState() Hook: https://reactjs.org/docs/state-and-lifecycle.html

Appendix F: SmartWorks Functions Documentation

SmartWorks Functions

Functions in SmartWorks adhere to the general structure of code shown below:

The function requires a “req” argument which is a Request Body.

A Request body contains the following; however, it does not require all components of this request body to run the function:

1. Method - HTTP method (PUT, GET, POST, DELETE, etc)
2. Body - body of request/the data that is being sent usually sent as JSON format
3. Headers - request headers such as content-type, authorization, etc
4. Query
5. Path - path to the endpoint of the function

Functions return a Response body consisting of two main components:

1. body - the data in a dictionary or JSON format
2. Status_code - an integer/number that represents the server response to the HTTP request (200, 201, 400, 500)

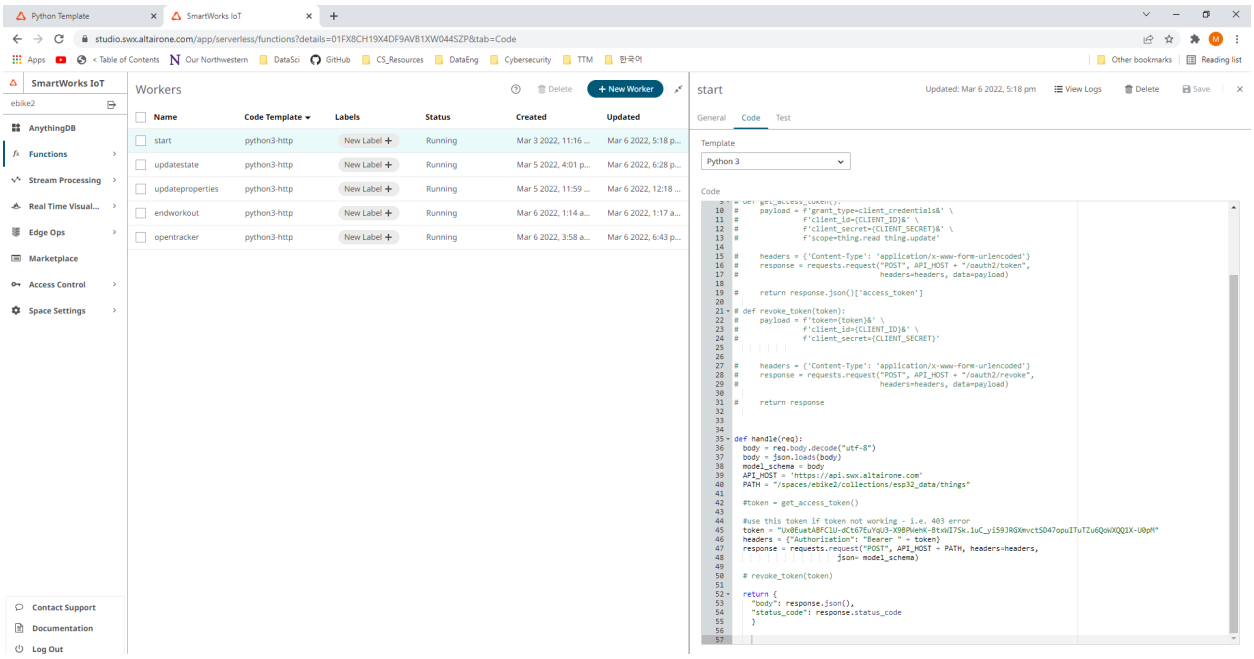


Figure 1: Function Workers page on SmartWorks IoT platform

Functions/Workers on SmartWorks

```
start(req)
```
The start function creates a new ‘thing’ according to the model schema it was assigned. Properties are defaulted to have no values

Args:
req (JSON object): it requires a model schema for the new ‘thing’ being created

Example Schema:
```
model_schema = {"model": {"name": "user_ride","version": 1}}
```

Returns:
response (JSON object): returns a JSON response object with details of the ‘thing’ that was created (includes: model schema, credentials, uid, and more). Also returns a status code
```
```

```
updatestate(req)
```
The updatestate function updates the property, “state”, stored in the ‘thing’ based on the “uid” and “state” inputs given. For this project, the “state” input should be either “opened” or “closed” to indicate if the workout is in progress

Args:
req (JSON object): takes in JSON object with the following format: {“uid”: “thing_id”, “state”: “opened or closed”}

Returns:
response (JSON object): returns a JSON response object with the updated state and a status code

Example output: {"state": “opened” }

```
updateproperties(req)
The updateproperties function updates multiple properties for the provided
“uid”, ‘thing’ identifier. Can update as many properties as given into the
function


Args:
req (JSON object): takes in JSON object with the following format:

Example input:
{"uid": “thing_uid”,"properties": {"username": "name", "age": 24,
"bpm":150, "state":"opened" }}

Returns:
response (JSON object): returns a JSON response object with the updated
properties and a status code

Example output: {"state": “opened” }
```

```
endworkout(req)
The endworkout function updates the “state” property in the provided “uid”
identifier to “closed”.

Args:
req (JSON object): takes in JSON object with the following format:

Example input:
{"uid": “thing_uid”}

Returns:
response (JSON object): returns a JSON response object with the updated
state and a status code

Example output: {"state": “closed” }
```

```
opentracker(req)
The opentracker function sniffs out “things” that have the property,
“state” marked as “opened”. The function is used to identify which
rides/workouts are active and can be used to extract information of
“opened” “things”

Args:
req (JSON object): takes in JSON object with the following format:

Example input:
{} -> empty JSON object

This function technically does not require an input; however, the “handler”
library used to run functions on SmartWorks requires a JSON object


Returns:
response (JSON object): returns a JSON response object with the updated
properties

Example output: {"state": “opened” }
```

Appendix G: SmartWorks Documentation

SmartWorks Documentation

Template

Topic:
Name:
Date:
Goal:
Process & Notes:
Issues:

Topic: Working on Actions/Events/Functions
Name: Matthew Kim
Date: 2/28/2022
Goal: To link properties with actions/events/functions to carry out any calculations with raw history data (heart rate, breath rate, energy expended, etc)

Process & Notes:
Redoing the Altair Collection creation/HTTP/MQTT data sending tutorial

HTTP - a method in which web browsers and web servers communicate information and data
Devices can be connected to SmartWorks to send data through HTTP, as demonstrated in their tutorial.

SmartWorks utilizes an API, an Application Programming Interface, to send data to a server where the data is interpreted, and then a specific function/service is provided. The Altair API is hosted in the link below:

host = 'https://api.swx.altairone.com'

Topic: Alvaro's Tutorial on Creating New Instances/Visualization

Name: Matthew Kim

Date: 2/28/2022

Goal: To automate creating new instances for each ride/workout session to prevent storing all data in one source

Issues with storing in one source:

1. Visualizing data becomes cluttered and the scales of data is hard to understand (large periods of time gap in between sessions)
2. Makes it difficult to perform data analysis on individual rides → calculate calorie burned, pattern recognition (bpm behavior), and more

Process & Notes:

- Create an App
  - App is a method of linking external information from external sources to SmartWorks IoT platform
  - Doc page: https://help.altair.com/2021/smartworks/topics/access_control/apps_intro.htm#task_n42_yyb_p4b

- Create Real-time visualization notebook
  - Set up data source → use SmartWorks IoT as data connector *(was previously JSON)
  - Retrieve data from Raw History stored in "Things"
    - `https://api.swx.altairone.com/spaces/YourSpaceID/data/?source[]=thing_id&limit=1000`
    - to read raw history from multiple things the following segment can be modified as: …/?source[]=thing_id1&source[]=thing_id2?&limit=1000
- Utilizing API and HTTP requests to create new Things
  - APIs can send and receive data by several HTTP request commands such as:
    - POST - send data
    - GET - receive data
    - PUT, PATCH, DELETE

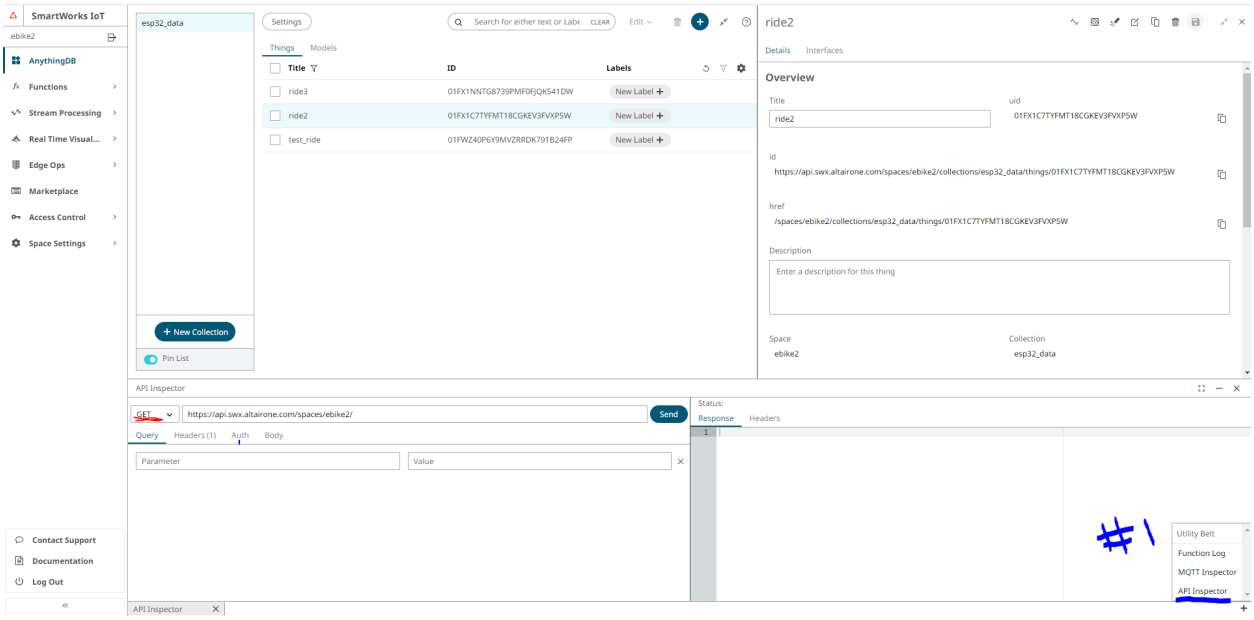


Figure 1: SmartWorks' Space Interface - API Inspector Shown at Bottom with red highlight over Request commands

- The API inspector can be used to create new "things" by POSTing a data body of the "model" that is to be created
    - A "model" is a saved format of a thing which includes properties, actions, and events and this model can be reused to clone or create new things with the same structure

Using Arduino IDE to compile and write code to execute the following workflow:

Workflow of Creating New Instances with Each Workout

1. Retrieve Authorization Token by using PUT to upload SmartWorks App credentials
2. Parse and retrieve the Authorization token
3. Using the token, use another PUT command to upload a new thing with an assigned model in the collection workspace
4. GET the UID of the new thing
5. With the UID, update MQTT publishing route so that the workout/heart rate data is sent to the new thing/workout instance instead of an older version

Install ArduinoJSON.h and Arduino_JSON.h libraries to deserialize JSON data that is PUT/GET through HTTP and HTTPS

- Arduino IDE is not capable of parsing through JSON data without an external library and these libraries will aid in accomplishing that goal

Due to security measures on Altair SmartWorks, HTTP protocols will not work and HTTPS protocols will have to be used instead. To access the API behind HTTPS, the website certificate needs to be extracted and sent with each Requests' command executed.

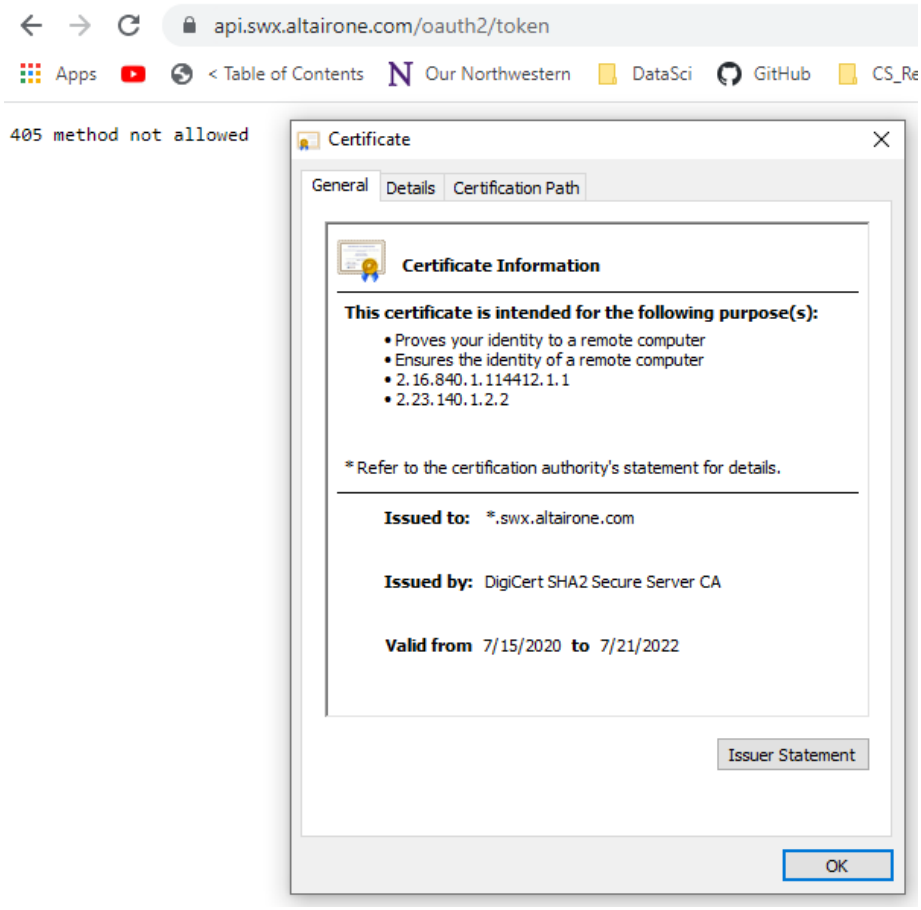


Figure 2: Certificate issued by DigiCert for security on SmartWorks API

By accessing the Certification Path as shown in Figure 2, the parent certificate can be located and from there, the certificate can be downloaded as a text file. The certification then can be sent with each Request command to communicate through HTTPS.

```
const char* ca_cert = \
"-----BEGIN CERTIFICATE-----\n" \
"MIIDrzCCApegAwIBAgIQCDvgVpBCRrGhdWrJWZHHSjANBgkqhkiG9w0BAQUFADBh\n"\
"MQswCQYDVQQGEwJVUzEVMBMGA1UEChMMRGlnaUNlcnQgSW5jMRkwFwYDVQQLExB3\n"\
"d3cuZGlnaWNlcnQuY29tMSAwHgYDVQQDExdEaWdpQ2VydCBHbG9iYWwgUm9vdCBD\n"\
"QTAeFw0wNjExMTAwMDAwMDBaFw0zMTExMTAwMDAwMDBaMGExCzAJBgNVBAYTAlVT\n"\
"MRUwEwYDVQQKEwxEaWdpQ2VydCBJbmMxGTAXBgNVBAsTEHd3dy5kaWdpY2VydC5j\n"\
"b20xIDAeBgNVBAMTF0RpZ2lDZXJ0IEdsb2JhbCBSb290IENBMIIBIjANBgkqhkiG\n"\
"9w0BAQEFAAOCAQ8AMIIBCgKCAQEA4jvhEXLeqKTTo1eqUKKPC3eQyaKl7hLOllsB\n"\
"CSDMAZOnTjC3U/dDxGkAV53ijSLdhwZAAIEJzs4bg7/fzTtxRuLWZscFs3YnFo97\n"\
"nh6Vfe63SKMI2tavegw5BmV/Sl0fvBf4q77uKNd0f3p4mVmFaG5cIzJLv07A6Fpt\n"\
"43C/dxC//AH2hdmoRBBYMql1GNXRor5H4idq9Joz+EkIYIvUX7Q6hL+hqkpMfT7P\n"\
"T19sdl6gSzeRntwi5m3OFBqOasv+zbMUZBfHWymeMr/y7vrTC0LUq7dBMtoM1O/4\n"\
"gdW7jVg/tRvoSSiicNoxBN33shbyTApOB6jtSj1etX+jkMOvJwIDAQABo2MwYTAO\n"\
"BgNVHQ8BAf8EBAMCAYYwDwYDVR0TAQH/BAUwAwEB/zAdBgNVHQ4EFgQUA95QNVbR\n"\
"TLtm8KPiGxvDl7I90VUwHwYDVR0jBBgwFoAUA95QNVbRTLtm8KPiGxvDl7I90VUw\n"\
"DQYJKoZIhvcNAQEFBQADggEBAMucN6pIExIK+t1EnE9SsPTfrgT1eXkIoyQY/Esr\n"\
"hMAtudXH/vTBH1jLuG2cenTnmCmrEbXjcKChzUyImZOMkXDiqw8cvpOp/2PV5Adg\n"\
"06O/nVsJ8dWO41P0jmP6P6fbtGbfYmbW0W5BjfIttep3Sp+dWOIrWcBAI+0tKIJF\n"\
"PnlUkiaY4IBIqDfv8NZ5YBberOgOzW6sRBc4L0na4UU+Krk2U886UAb3LujEV0ls\n"\
"YSEY1QSteDwsOoBrp+uvFRTp2InBuThs4pFsiv9kuXclVzDAGySj4dzp30d8tbQk\n"\
"CAUw7C29C79Fv1C5qfPrmAESrciIxpg0X40KPMbp1ZWVbd4=\n"\
"-----END CERTIFICATE-----\n";


http.begin(spaceServer, ca_cert); //OAuth Token URL & Certificate to access HTTPS
```

Figure 3 - Code Snippets from Arduino IDE showing Certificate and HTTPS connection set up

HTTPS connection is established and Requests commands are coded in the Arduino IDE and uses the ESP32's wifi functionality to carry out all data transfers and communications. As shown in Figure 4, the code is able to use the ESP32 to send a request to create a new thing when the code is activated. Future tasks that still need to be accomplished are the retrieval of the newly created thing's UID and reformatting the Arduino IDE code onto PlatformIO. By consolidating all code on PlatformIO, the ESP32 can perform MQTT and HTTP communication through one program without needing additional ESP32s; however, this could be a possible alternative if it cannot be completed.

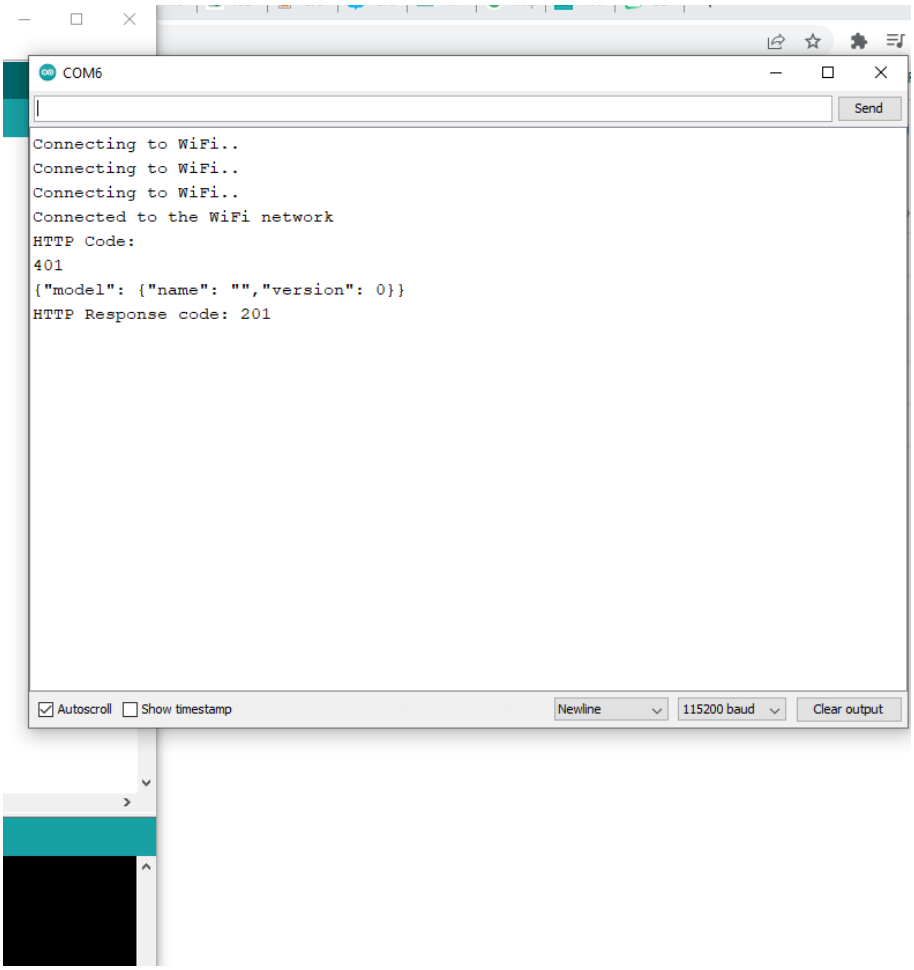


Figure 4: HTTPS Connection Established and Code 201 indicating Creation of New Thing

---

Topic: Creating Functions on SmartWorks

Name: Matthew Kim

Date: 3/2 & 3/10

Goal: Understand how functions work and create functions to create new "things" automatically to emulate the creation of new ride/workout instances

Process & Notes:

SmartWorks supports functions to be written in Python3 or GO. For this project, the team will be using Python. Functions are executed by a function called 'handle' which accepts a request object as its input to retrieve a response object.

```
def handle(req):
    return {
        "status_code": 200,
        "body": req.body.decode('utf-8')
    }
```

Topic: Creating PUT Functions & Potential Bug/Issue

3/10/2022

Goal: Create a function that utilizes HTTP 'PUT' command

Process & Notes:

Previously, a PUT function was created to update the properties of a given thing based on the UID and Properties (in JSON format) it receives. Within this "updateproperties" function, it runs two other functions which retrieves and revokes access tokens in order to gain authorization to run functions on SmartWorks. Currently, the access tokens can be retrieved but they no longer grant access to complete the function and returns:

{"error":{"details":{"reason":"Access token is not active","status":"Unauthorized"},"message":"Access credentials are invalid","status":401}}"

Alvaro was notified; however, no solution was found. To work around this, the authorization token that can be found in the API inspector will need to be hard coded into the function in order to run. The reason why this is not an effective solution is that it expires every day and cannot be programmatically updated on it's own (as far as I know).

Issues:

1. ~~Cannot create new "string" type properties when making a new thing~~ (RESOLVED 3/7)
    - other data types like integer properties can be created without much issue
    - SOLN: edit model schema manually on SmartWorks

Topic: Server Issue

Name: Matthew Kim

Date: 3/12/2022

Goal: Address the server issue and document what has happened

Process & Notes:

I tried to login to SmartWorks studio Saturday morning of March 12th and ran into an error screen stating that my login has expired or is invalid. To fix the issue, I erased any previous website data/cached login info to see if the issue was more specific to my account; however, that did not prevent the error page from opening up again, as shown in figure 1.

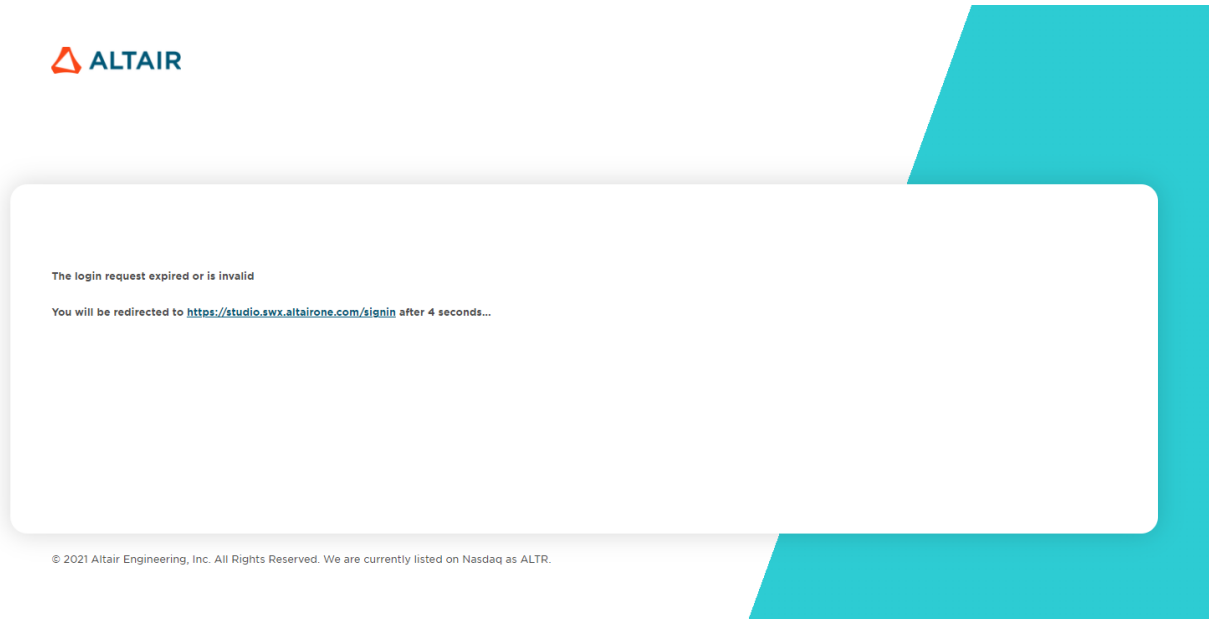


Figure 1: Altair Login Error Screen

The link that is shown in the image above leads to the same page and this error screen is constantly looped. I reached out to Alvaro and Altair contacts to see if the issue can be resolved, but as of 3/13/2022, 10PM, there have been no updates.

Topic: Panopticon XY visualization & Customizability

Name: Matthew Kim, Maddie Durmowicz

Date: 3/12/2022

Goal: Create a graph that plots Y data points across X data points. Issue is that all data points are congregated to one point and the data is not being spread across to create the visualization as desired as shown in Figure 1 below. Another goal is to learn more about the customizability options that Panopticon has to offer and how to integrate to capstone project dashboard.

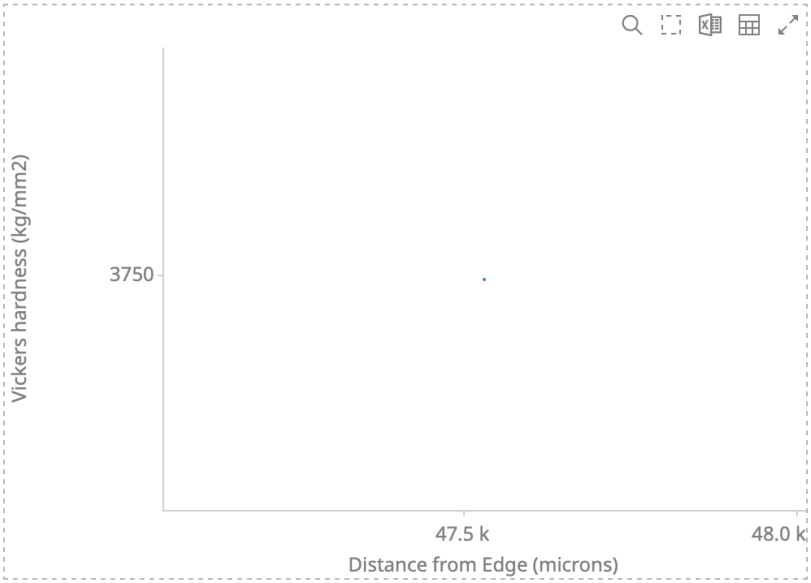


Figure 1: Visualization of a single point across multiple X and Y data points

Data was imported into a workbook in an existing space. A visualization box was then created and a scatter plot & line graph were attempted.

Plotting

Data was *easily* imported and previewed from an Excel file. Columns were labeled as "Numeric" so that it could be plotted. The uploaded columns can be dragged and dropped into the "X←→ " section and the visualization section. In this attempt the "Distance from the edge [micron's]" was inputted as the x-axis variable and the "Vicker's hardness [kg/mm2]" was imported as the visual (see Figure 1).

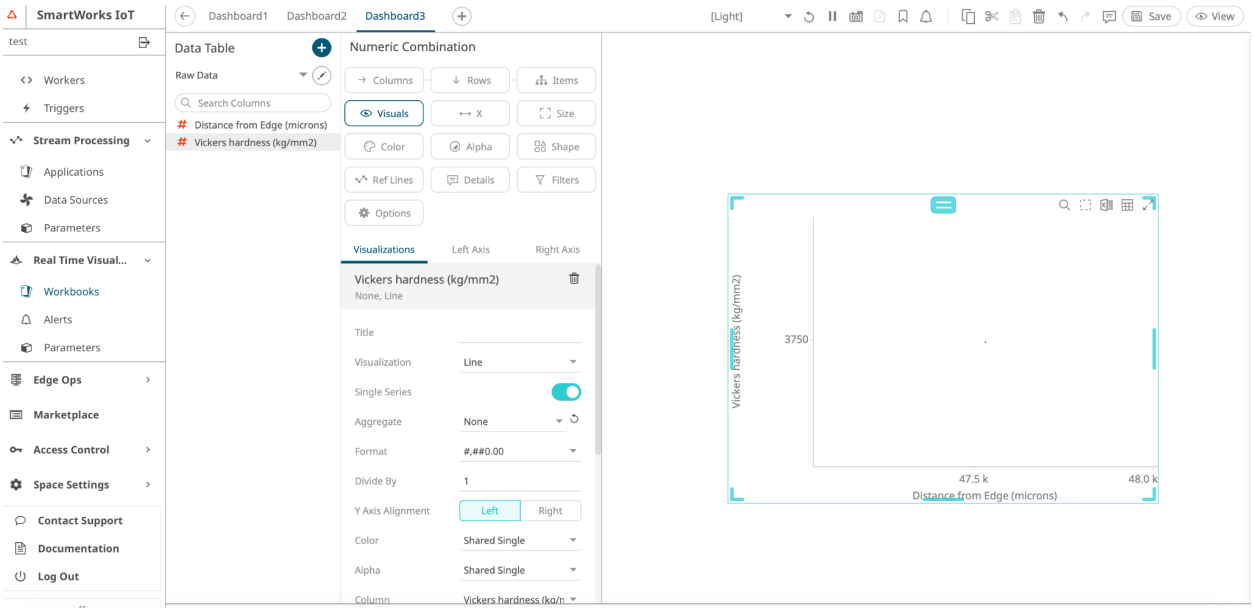


Figure 1: Smartworks Workspace: Visualization edits

Despite having the Aggregate drop down set at “None”, all of the data points were summed and displayed as a singular point on the graph (as seen above in Figure 1). Several “Aggregate” options have been selected and attempted, however, either no or one data point is plotted. Figure 2 below depicts all of the different “Aggregate” options.

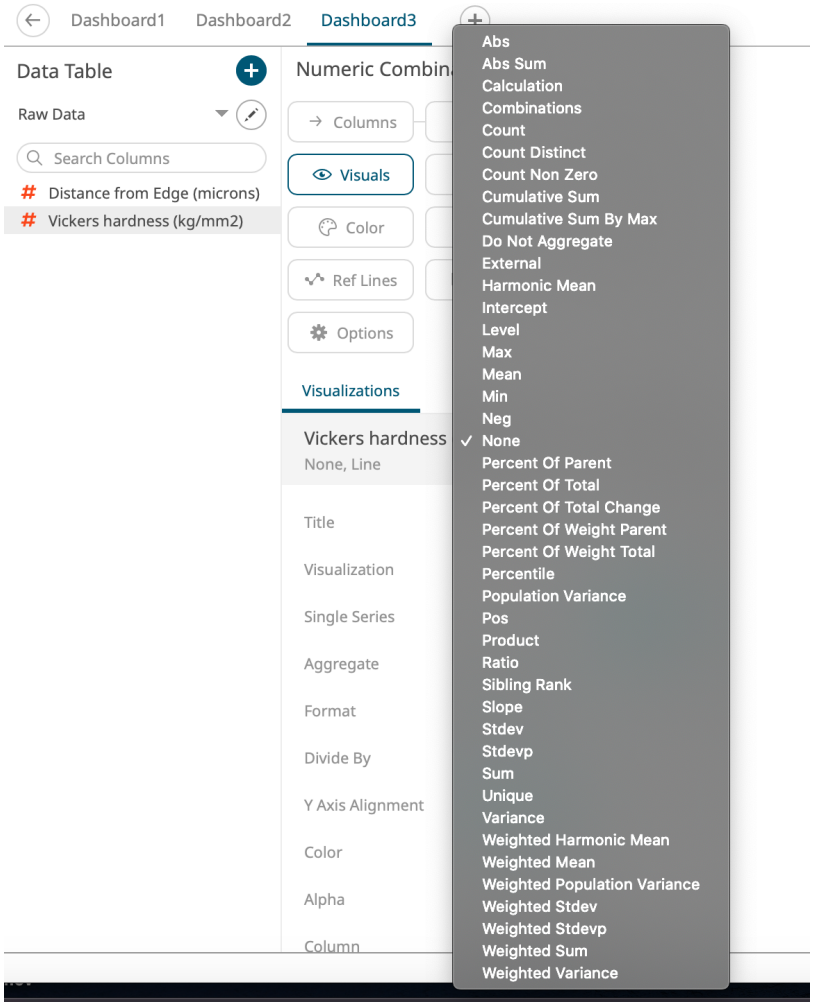


Figure 2: Aggregate Options

Figure 3 below depicts the current Visualization of Data with the Aggregate drop down set to "None".

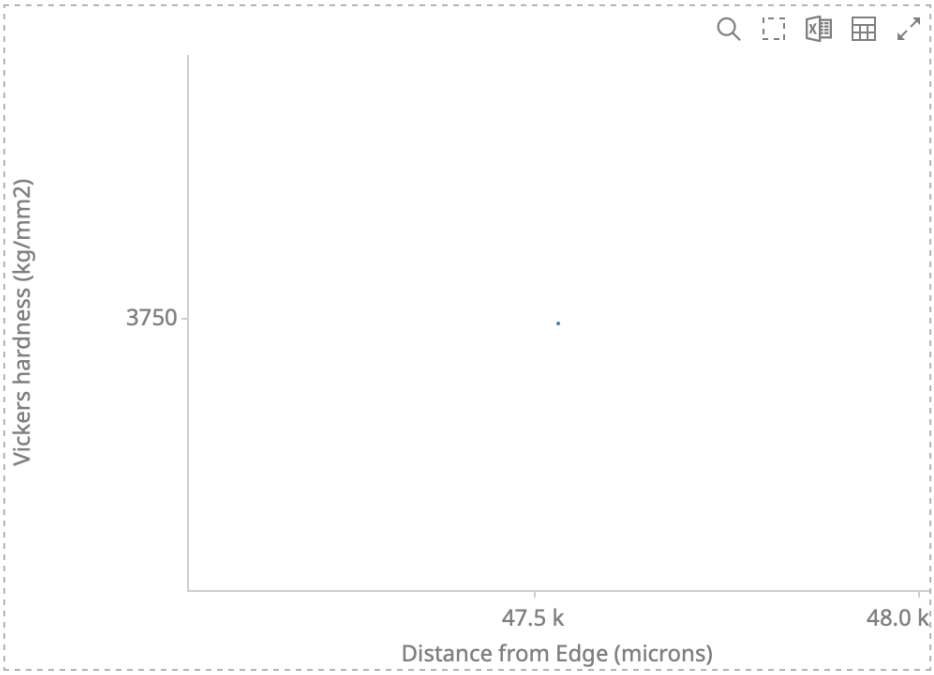


Figure 3: Current Visualization

Color

Some research has been done to identify how to upload color palettes. Little was found, so the team plans to ask Altair Engineers in the Client meeting on Tuesday, March 1st.

Columns that want to have color scaling can be dragged-and-dropped into the "Colors" section of the Visualization Creation page. There are several existing color palette options. The "Color" page of the visualization editing and these default color options can be seen in Figure 4 below.

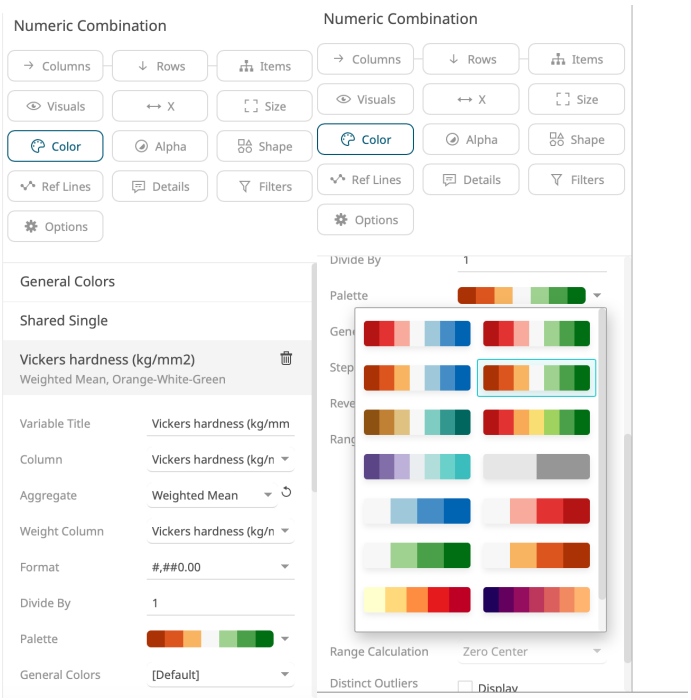


Figure 4: Default Color Page & Palette Options

There is also a place to create a new palette. These colors can be set to be continuous and gradually change with values, or may be discreetly selected for a certain range of values (perhaps heart rate levels or levels of motor assist). These colors can also be specifically selected from color-code. Figure 5 depicts the pop-up DIY color palette page.

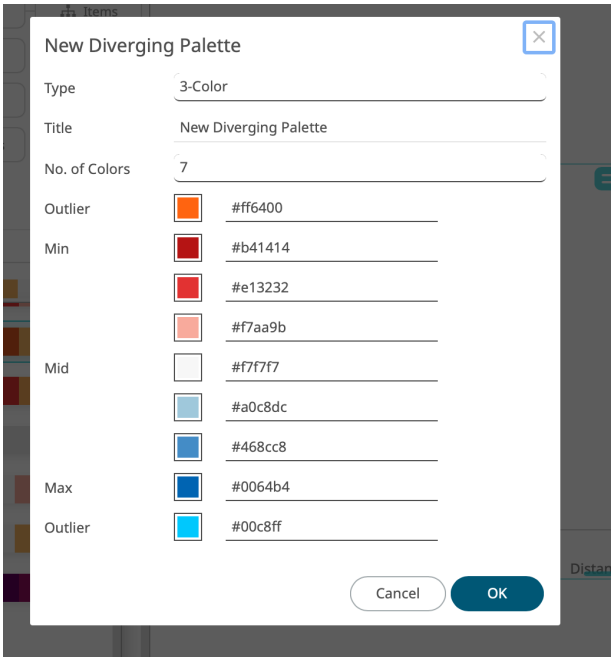


Figure 5: Create new palette page

Next week, we plan to explore the colors and apply them to existing heart rate data and plots.

Solutions:
After consulting with Alvaro and Paloma, X and Y data points can be plotted on a line graph visualization if there is a qualitative variable like source_id, id, etc to split the data points on. The X and Y data columns are dragged to their respective buttons; however, this is not enough for Panopticon to split the data across the axes. In addition to the X and Y columns, a qualitative data column which is marked with a small icon of "abc" needs to be dragged into the "items" button in the visualization bar (Figure 6). This will tell the data how to split the data and plot it.

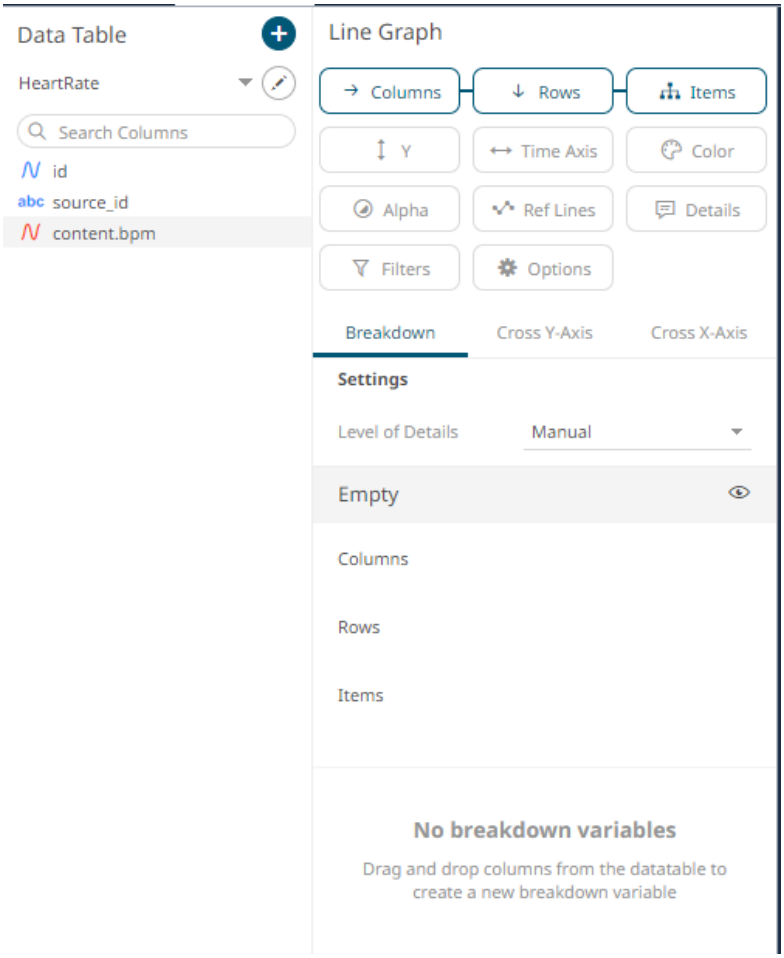


Figure 6: Graph Editor with "items" button in the top right corner

As for data that do not have a qualitative variable like source_id, id, etc., a parameter key needs to be generated using the X-variable. This parameter key will indicate how the data needs to be plotted and needs to be dragged into the "items" button as mentioned previously after being generated in the Data table settings page (Figure 7).

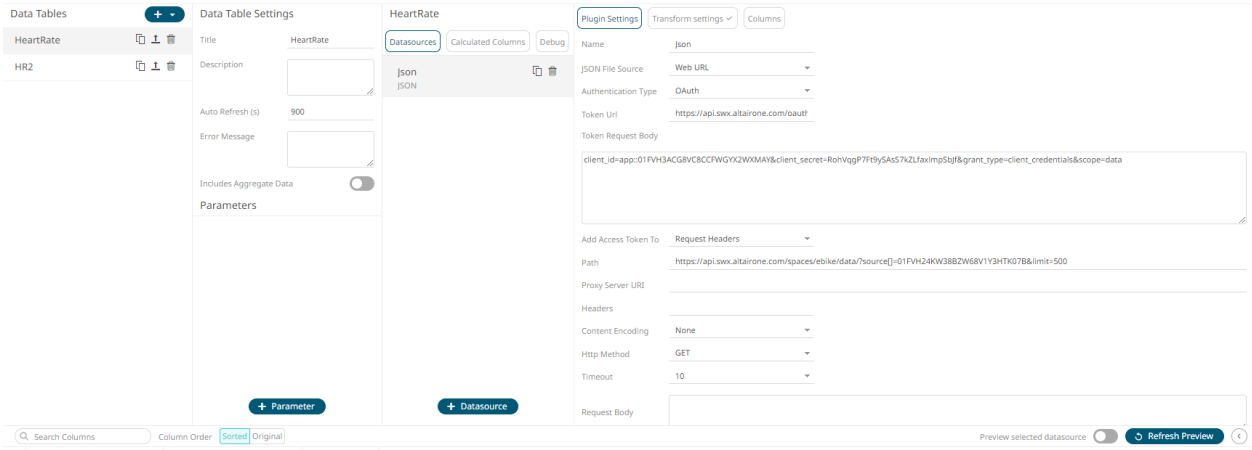


Figure 7: Data Table Editor with "+ Parameter" option in the bottom left

After implementing the solutions, the team was able to have deeper mastery of the Panopticon software and create the following dashboard with customized background and colored lines that correspond to heart rate zones (Figure 8).

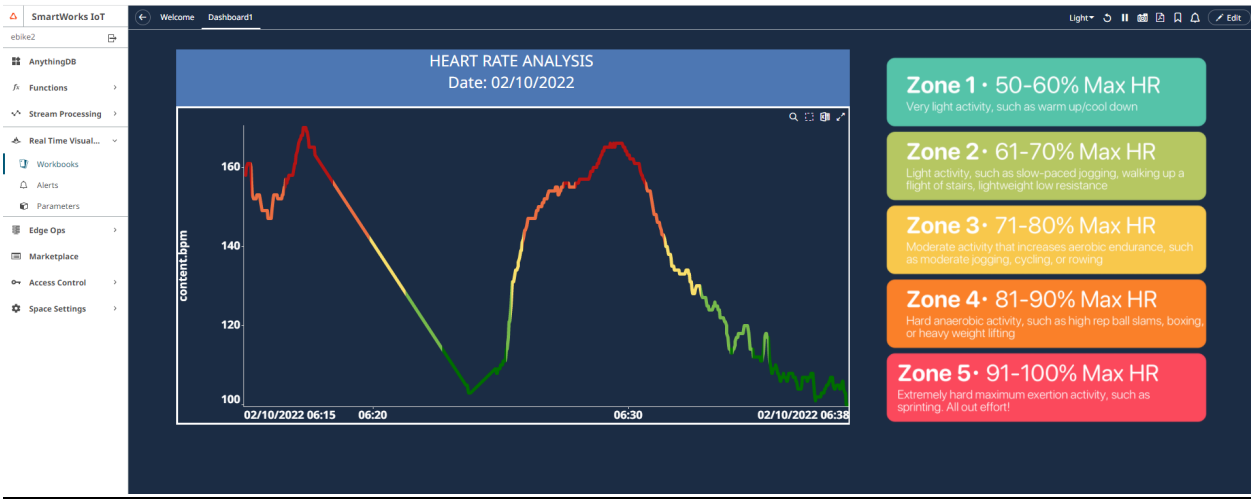


Figure 8: Heart Rate Analysis Dashboard

Topic: Server Issue Update

Name: Matthew Kim

Date: 3/14/2022

Goal: Revisit the server issue and document additional issues

Process & Notes:

On 3/14/2022, around 3AM, the Altair team was able to get the servers up and running again. Despite the servers running, the functions that were on SmartWorks started to return error 500 with non-descript details on why the error is occurring. The team messaged Alvaro and Altair team regarding the issue;

however, the issue is still persistent as of 3/14/200, 4PM. The number of functions have been reduced to 4 to see if it can fix the issue as recommended, but it did not work.

An alternative to using functions would be to directly make calls to the REST API from SmartWorks and code the functions on the microcontroller and website directly.

Appendix H: Bike Frame Research

Search Term: “best bike for diy electric bike”
Search Engine: Google
From: http://www.ebikeschool.com/choosing-right-bicycle-electric-bicycle-conversion/

- “The best bikes for electric conversion are steel bikes with steel dropouts”
- Aluminum bikes are ok
- Considerations
    - Material
        - Steel
        - Aluminum - both are light but cheap
        - Never use carbon fiber, you don’t want to ruin an expensive bike frame
    - Brakes
        - Disc vs rim
        - Disc gears can complicate installation of a hub motor, but should be ok if only the front wheel has disc gears
    - Suspension
        - Nice but not a requirement
        - Rear suspension limits battery
        - Simplifies conversion you don’t have suspension

Bike Options
Source: Walmart

- Criteria for searching
    - Adult
    - Unisex
    - Steel, alloy, aluminum, or stainless
    - 26” wheel (to fit trainer)
    - Above 3 stars

Name/Photo	Material	Brakes	Suspension	Cost	Wheel Size	Link
	Steel	Disc	Yes	$199.97	26”	https://www.walmart.com/ip/Hosim-Oture-Mountain-Bike-26-In-Wheel-21-Speeds-High-Carbon-Steel-Frame-for-Men-and-Women-Black-

						and-Blue/876802907
	Aluminum	Disc	Yes	$425.99	26”	https://www.walmart.com/ip/Piscis-Adults-26-In-Mountain-Hybrid-Bicycle-24-Speed-Suspension-with-Dual-Disc-Brake-Aluminum-Frame-City-Bikes-for-Men-and-Women/775488668?athbdg=L1400
	Aluminum	Disc	Yes	$268	27.5”	https://www.walmart.com/ip/Schwinn-AL-Comp-mountain-bike-21-speeds-27-5-inch-wheels-grey/388726134?athcpid=388726134&athpgid=AthenaItempage&athcgid=null&athznid=si&athieid=v0&athstid=CS055&athguid=ER0VisAY5e--h75VSmgaI0-L_8kl8_KIvaR4&athancid=null&athposb=0&athena=true
	Aluminum	Rim	No??	$499	???	https://www.walmart.com/ip/Decathlon-Triban-Abyss-RC100-Aluminum-Road-Bike-700c-Silver-M/529793109
	Aluminum	Rim	Yes	$248	26”	https://www.walmart.com/ip/Decathlon-Rockrider-ST50-Aluminum-Mountain-Bike-26-Red-Medium/559843315
	Steel	Rim	Yes	$148	26”	https://www.walmart.com/ip/Hyper-Bicycles-Mens-26-Shocker-Mountain-Bike-Black/54169167?athbdg=L1600

	Steel	Disc in front Rim in rear	Yes	$198	26”	https://www.walmart.com/ip/Kent-26-In-Northpoint-Men-s-Mountain-Bike-Black-Blue/992264560