Smart Insole for Real Time Gait Analysis

Alexander Khoury, Craig Ives, Tyler Richard,
Eric Waters, Vincent Tang

ECE 190, Professor Truong Nguyen

University of California, San Diego, La Jolla, CA 92093

June 16, 2017

CONTENTS

1 ABSTRACT

Millions of senior-citizens have a significant fall every year because of uneven surfaces and unhealthy walking patterns. This impacts their quality of life, and those around them as any fall can be life threatening and require extensive, expensive medical care. The smartSole insole was developed to track and help diagnose unhealthy walking habits so that they can be corrected or minimized. It implements sleek integrated data collection, processing, and transmission into a comfortable foam insole for everyday use. This is paired with machine learning algorithms to detect potentially hazardous gaits so that medical professionals can develop personalized treatment plans. smartSole is able to differentiate between healthy and unhealthy walking habits which can be displayed in real time or as a gait summary report. This system could potentially prevent hundreds of thousands of falling injuries and help improve mobility for senior citizens.

2 INTRODUCTION

Any device seeking to solve medical problems for the senior-citizen community must necessarily be tightly integrated with modern medical research. There is extensive evidence in the literature that the risk of falling, as well as many other ailments affecting the senior community, can be linked to gait dysfunction.

For example, studies have found that gait and balance instability are crucial risk factors for falling [1], and that variability of stride length and swing time are risk factors for specifically injurious falls [2]. In one meta-study of 12 other large studies, which assessed the risk of falling in elderly people, gait and balance disorders were the second most common cause of falls behind the "accident and environment" category. The authors note that "gait and balance impairments were a significant risk factor for falls, and were associated with about a threefold increased risk for falling..." [3]. The American Geriatrics Society, the British Geriatrics Society, and the American Academy of Orthopaedic Surgeons convened a panel on falls prevention that published a special article "Guideline for the Prevention of Falls in Older Persons", which is a large meta-analysis of research on the causes of falling, as well as recommendations to prevent falling. The panel specifically recommends assessing older persons with the "Get Up and Go Test" - a functional mobility test - which has been correlated with gait velocity [4]. The panel also recommends, for both community-dwelling and assisted living seniors, the specific use of gait training as a important part of a multifactorial intervention [5].

In addition, specific gait parameters are often associated with brain disorders and other impairments. Alzheimer's patients display increased stride length variability [2], and gait problems such as shuffling and reduced gait velocity are strong symptoms of Parkinson disease [6]. Slower gait speed is also associated with loss of independence (i.e. nursing home placement, inability to perform instrumental activities, etc.) and an overall increased risk of falling [7]. In broader terms, gait dysfunction has been linked with memory decline, mild cognitive impairment, risk of dementia, decline in executive function, institutionalization, and an overall higher risk of death [8] [9] [10].

Currently, the detection of gait problems requires expensive equipment and a controlled environment. For instance, Verghese *et al* used a computerized walkway that automatically outputs certain gait parameters such as velocity, cadence, swing time, and more - parameters that the authors claim are "...widely used in clinical and research settings..." [9]. However this form of testing can become expensive and time consuming, not to mention that it may fail to capture the nuances of the outside world. Moreover, seniors who are aware of their risk of falling may preempt such measurements by walking unrealistically in the clinical setting. Any device that can solve these problems and yet obtain the same gait parameters would be highly useful to doctors and researchers. Such a device would allow doctors and researchers to utilize the entire compendium of past research on gait disorders in their assessments.

SmartSole can collect those exact parameters that have been linked to gait disorders in the literature, and it can do so without forcing seniors into a controlled environment. By using motion sensors and pressure sensors within an insole, smartSole can collect data in the day to day life of seniors. With wireless charging, the device will be completely hidden away so no maintenance will ever be required. Software in the device will be able to process the data and provide the metrics doctors look at, as well as make note of any time period where there poor gait. This solution provides doctors a gait profile which they can then observe and diagnose as they see fit.

3 SYSTEM

The smartSole project is a system capable of recording real-time walking data with advanced hardware, and interpreting mass amounts of data to give valuable and concise measurements. The complex system involves several feature sets working together to accomplish the overall goal of better gait analysis.

3.1 Overview

Two main levels of hardware accomplish practical data collection and simple use. Integrated hardware measures and records critical data, all while fitting inside a thin, flexible insole. External support hardware is contained within a floor mat, providing a set battery charging and data collection point. The external mat also serves as the first point for networking all smartSole devices.

Even with hardware capable of recording mass amounts of data, determining what and when to record is critical for conserving memory, power, and computing resources. Developing ways of focusing data collection will benefit the entire proposed smartSole system and network.

Bluetooth low-energy communications link the integrated hardware to the base station and other existing BLE enabled devices such as smart phones and computers. While critical to the system, efficiently implementing BLE is critical for power conservation and for freeing up integrated hardware memory.

While raw data alone is valuable, the smartSole project includes specific applications of machine learning software to accomplish advanced analysis of raw sensor data. Data

processing software allows the user's recorded actions to be analyzed by medical professionals by providing concise and useful metrics.

3.1.1 *Integrated Hardware*

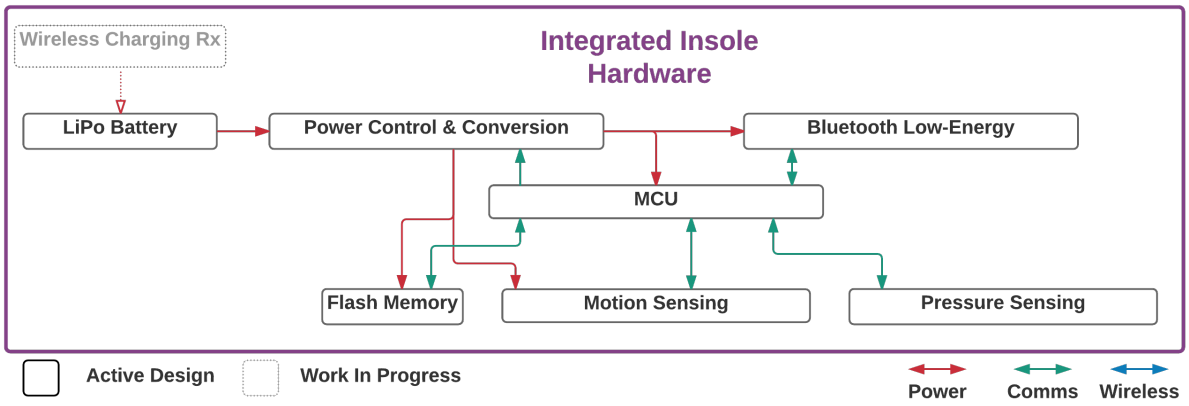


Figure 1: Integrated Hardware Overview

The integrated hardware serves a main purpose of implementing various sensors to record the necessary data. This simple goal faces mechanical, power, and price restrictions that require careful consideration. The core of the integrated hardware is a STM32L4 MCU. The MCU serves as the master controller for all integrated sensors, power control, memory, and communications.

To enable wireless communications, a Silicon Labs EFR32 Blue Gecko SoC is used. Its main purpose is to offload the complex BLE software stack from the MCU. It interfaces with the MCU through LEUART and hardware interrupt lines. The Blue Gecko SoC is placed is placed in a low-power sleep mode whenever BLE communications are unnecessary or unavailable.

Pressure sensors placed give dynamic readings on where the user is shifting their weight. These sensors take advantage the MCU's built-in capacitive sensing unit, and the compressible nature of the foam insole. By using a flexible conductive fabric, three pressure-sensitive capacitors are formed, providing fast and accurate readings of the user's weight distribution and balance.

Motion sensing is achieved with the use of an Invensense MPU6050 3-axis accelerometer and 3-axis gyroscope, and integrated Digital Motion Processor. This allows the raw sensor data to be translated into more useful outputs such as quaternion angles, or acceleration in reference to the world instead of the local chip.

Local data storage is implemented with NOR SPI flash interfaced with the MCU. It has a capacity of 128 Megabytes, making data collection algorithms important for conserving this limited space. Although limited in capacity, SPI flash is more physically compact and power friendly than other memory options.

To power the insole, a low-profile and high capacity battery is required. A 120mAh nominal capacity Lithium Polymer battery gives the integrated hardware a battery life of at least 24 active hours. Battery stability and safety is critical, and depending on testing, a less-power dense, but more stable Lithium-Iron-Phosphate battery may be required.

Integrated into the insole is a resonant inductive charging receiver, allowing the user to charge the smartSole simply by placing it on a floor charging mat overnight. This intuitive and simple charging method requires a custom resonant inductive charging solution to ensure compatibility with any shoe. This system is capable of providing different charge currents in each case, so the MCU will use a specialized control algorithm to charge the battery as quickly as possible.

Power supply regulation for the embedded systems is done with a high-efficiency Texas Instruments TPS62400 dual buck converter. Separating power into two buses allows the lowest required voltage to be used for each device, conserving power.

3.1.2 *External Hardware*

To support the advanced integrated hardware, an external system is required to maximize smartSole's utility. Support hardware takes care of charging the insole battery, pulling stored data, and acts the primary networking point. To minimize user effort, a small floor mat with inductive charging, BLE communications, and WiFi internet access provides an intuitive and simple to use support system. All the complex support processes are hidden from the user in a inconspicuous mat.

To charge the insole's battery, a resonant inductive charging transmitter is used. DC power from a wall adapter is fed into an inverter with a variable output amplitude and frequency. Multiple parallel inductive coils create a strong magnetic field with resonant coupling to the receiving coil in the insole. This resonant coupling allows for the controller to sense the presence of the Rx coil, and tune the inverter for maximum charge current.

To handle BLE and WiFi communications, a Raspberry Pi Zero is used. This platform runs Linux, and has integrated BLE and WiFi capability. When the insole is set on the mat for charging, the Raspberry Pi Zero uses BLE to pull all the data stored on the insole and makes it available to the smartSole network. It also has enough computing power to run initial data processing algorithms on board, prepping data to be sent to servers. While nowhere near as powerful as servers, the base station has the advantage of having a generous amount of time to process information for only one user, making it a viable data processing platform to reduce server dependency.

3.1.3 *Data Collection*

The sensor systems in the integrated hardware are capable of sample rates and resolution much higher that what is required for accurate diagnosis. Also, all times the user is sitting or not wearing the smartSole device are non-critical. This means that choosing exactly when and how fast to measure data is important. Doing so properly conserves memory and power on the insole, as well as processing effort on the base station and servers. This process of carefully and dynamically configuring data collection will be referred to as adaptive sensing.

3.1.4 *Communications*

In order to create a seamless integration of the system for the user, wireless communication is selected to allow the smartSole insole to remain in the shoe during charging and data offload. The golden standard in wireless communication is Bluetooth technology which is used in the smartSole system. A general GATT profile is created to implement the functions of packet creation and efficient data sending with the Blue Gecko SoC.

Data is fed into the Blue Gecko through the use of either a LEUART or normal UART connection depending upon the required data transmission rates which can use the same wired connection. This data is queued up within the Blue gecko's memory so that complete data for all sensors can be sent as a single packet across the Bluetooth connection.The data is then transmitted to a Raspberry Pi Zero to be decoded and processed.

The Communications system is designed to be triggered by connection to the smart charging mat system so that the Blue Gecko only offloads it's payload after a complete session. This is achieved by configuring the STM MCU as the master and the Blue Gecko as slave. This reduces data loss as well as power consumption which is imperative for the smartSole system.

3.1.5 *Data Processing*

Once the data is collected, smartSole begins processing the data. Once it has enough data points, it is preprocessed for ease of use. First the data is filtered through a five point moving average and then smartSole starts looking for peaks and troughs of the pitch. This step is necessary for calculating gait parameters as the pitch provides knowledge of when the foot is stepping down or lifting up. From there it can isolate the step into individual arrays for gait parameter calculation.

The data is also run through different machine learning techniques. This process is useful in determining when someone's walking is healthy or unhealthy. To make this classification, the device makes use of neural networks. To see which was more efficient, the device applied two different versions: logistic regression and feed-forward neural network.

3.2 Integrated Hardware

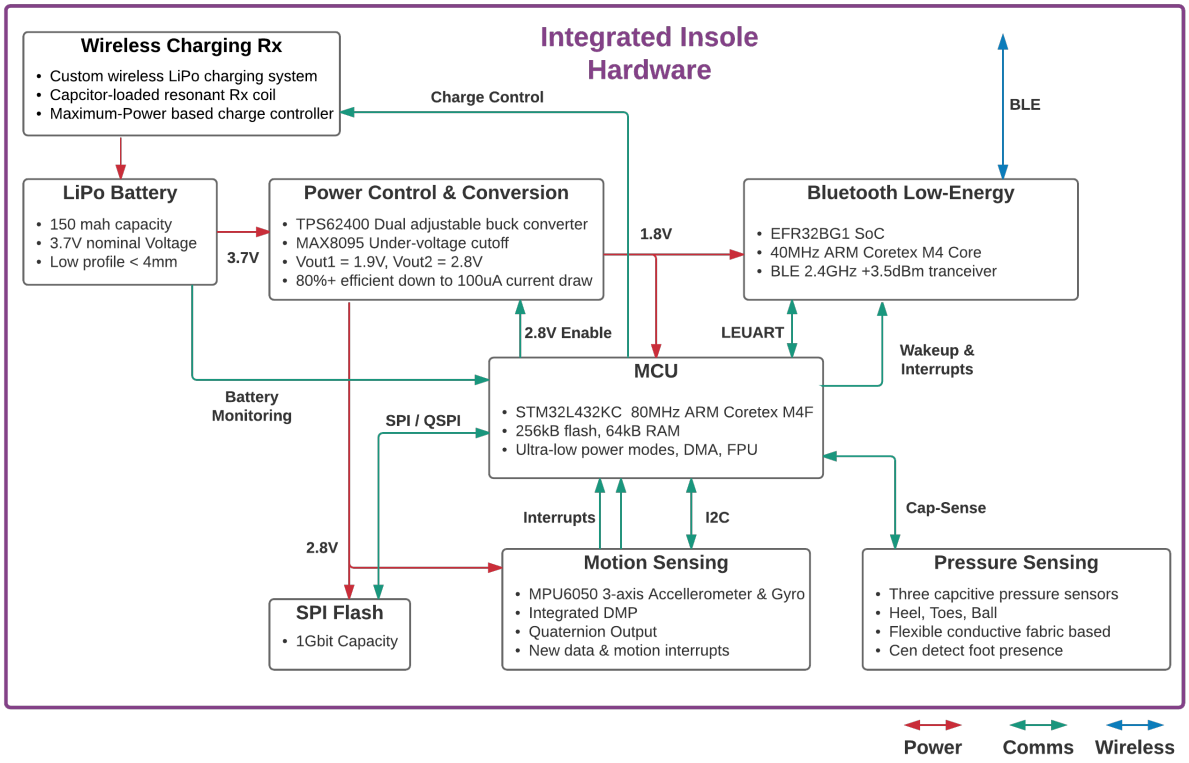


Figure 2: Integrated Hardware Detailed Block Diagram

The integrated hardware design is driven by strict performance, physical, and cost restraints. The integrated hardware requirements are as follows:

- Sense all necessary data required for diagnosis
- Store at least one active day's data
- Have basic data processing ability
- Interface wirelessly with base station and paired insole
- Charge wirelessly overnight
- Battery life of at least one active day
- Able to flex with the user's shoe
- Integrate electronics into a physically acceptable insole thickness
- Be safe and comfortable to use

- Have a competitive manufacturing cost

To achieve all these goals, the design focuses on putting together a system of small, low-power embedded devices. Careful attention is paid to the physical package dimensions, as well as operating voltage. For prototyping concerns, research on available support documentation and example applications is important, greatly speeding up the initial development stages.

3.2.1 *Motion Sensing*

Accurately tracing the user's foot movements is achieved by using an electronic Inertial Measurement Unit. These units typically measure linear acceleration and angular velocity along multiple axis. While this raw data is useful, complicated and error-prone mathematics are required to translate these raw values into more usable data such as quaternion angles, yaw pitch roll, or world-referenced acceleration vectors. The Invensense MPU6050 integrates a 3-axis accelerometer, 3-axis gyroscope, and a Digital Motion Processor. This device allows measurements of of the 6-axis raw data values at varying rates, as well as quaternion angle outputs from the DMP. Using the DMP offloads all of motion computations from the MCU, and implements highly specialized algorithms developed by Invensense, increasing accuracy. Activating the DMP only requires 100uA of additional current compared to raw data sampling alone, and allows the MCU to sleep while waiting for new data.

The DMP communicates to the MCU via 400kHz I^2C. The MPU6050 has one configurable hardware interrupt line, capable of signaling when new data is available in its output FIFO, or when set motion thresholds are exceeded. After configuring the FIFO rate, this allows the MCU to sleep whenever new data is not available, and removed the need for polling over I^2C. Also, motion threshold interrupt are useful for waking up the system from standby modes. A software state flow diagram illustrating this process is shown in Figure 3.

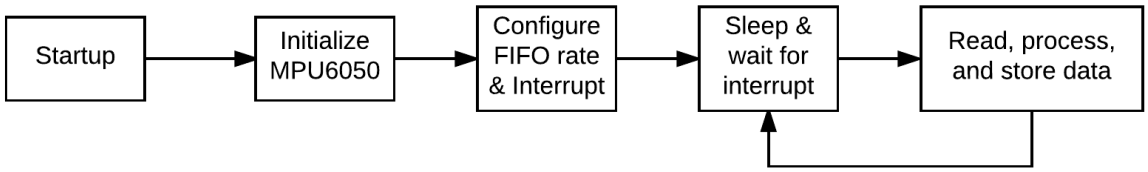


Figure 3: MCU software handling of MPU6050

All IMU integrated circuits are sensitive to physical stresses, as the contain sensitive microscopic physical structures to make inertial measurements. The MPU6050 application notes outline specific PCB layout guidelines for solid PCB boards, but using the MPU6050 on a flexible PCB within a shoe will require special reinforcement.

3.2.2 *Pressure Sensing*

The main goal of the smartSole insole is to obtain accurate and useful data about both the foot and the gait of the user to determine if unhealthy walking conditions are

occurring. A good metric of gait analysis is the pressure map of a users foot because it reveals directional balance and helps to define the user's current state (walking, sitting, standing, etc...). An effectively designed pressure sensor for the insole needs to be both slim and have low power consumption. At the same time a comprehensive mapping of the major foot components must be created to yield useful data.

Originally force-sensing resistors were considered for this purpose because of their simplicity. As the pressure exerted upon the resistor pad is increased the resistance of the pad lowers from a relatively infinite value to a few hundred kiloohms. This can be directly converted into weight by measure the changing current through the resistor at a known voltage and comparing it to a weight to current relationship. However, most of the slim profile force-sensing resistors are only rated up to 100 pounds for accurate measurements which will not work for the typical consumer. The price of these units also makes it economically impracticable to use multiple at a reasonable price point, so a different method is developed to better suit the insole.

It is possible to implement a rudimentary capacitor between a charged surface and a body because humans have a natural capacitance due to charge buildup on the skin. This method is used broadly in touch screen applications and can be arranged to detect an appendage even before contact is made. The parallel plate capacitance equation is given as

$$C = \frac{\epsilon_0 K A}{d} \tag{1}$$

Where ϵ_0 is the constant permittivity of free space, K is the dielectric of the material and equals about 1 in air, A is area of the plates, and d is distance between the plates. When a small conductive surface is applied to the bottom of the insole and charged, a small capacitance is generated between the surface and the foot through the foam. As the area and of the two "plates" and the dielectric of the foam insole remain constant only the distance varies with each step creating a variable capacitor. This capacitance can be measured and gives a general idea of the pressure map of the user's foot.

The conductive surface is implemented with a metal-coated cloth fabric to create a conductive surface capable of bending without fracturing. In bulk quantities this fabric is about one twentieth the cost of pressure resistors and is not limited in weight magnitude making it ideal for this application.

In order to measure this pressure relative capacitance with the STM MCU the capSense system is implemented in Figure 4.

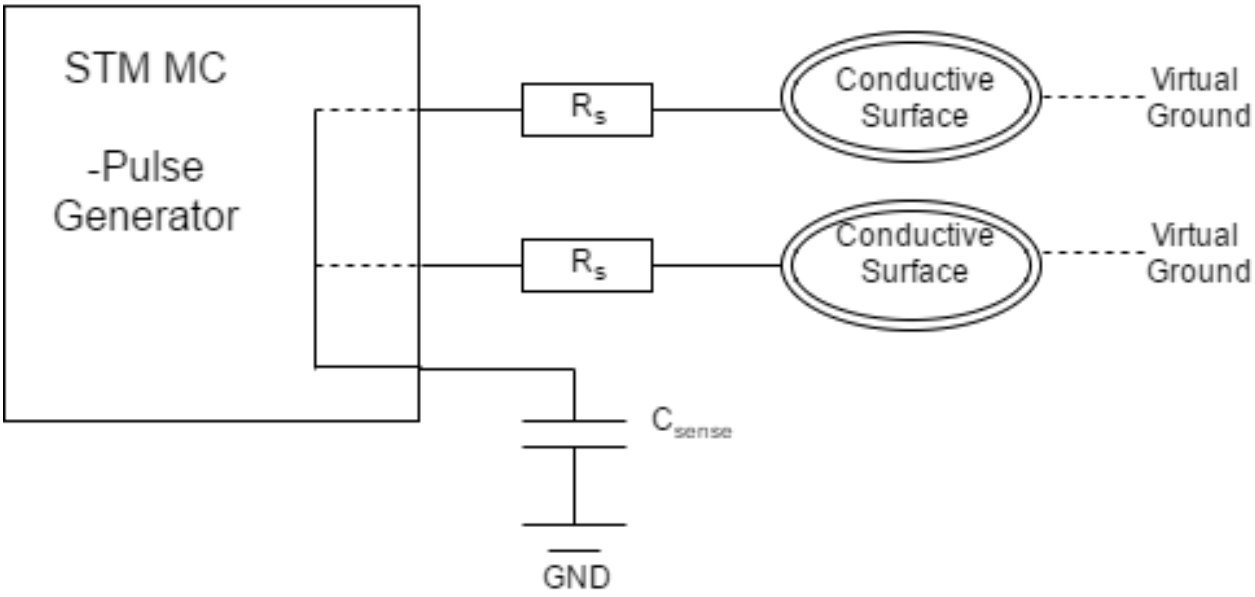


Figure 4: STM CapSense configuration

The system charges up each of the conductive surfaces to maximum through a 1kΩ resistor and slowly pulses a single sensor through an output capacitor. Then a counter is created to track the number of pulses needed to charge the output capacitor to a specific voltage. The duty cycle of each pulse and the value of the output capacitor are known, meaning that a higher conductive surface capacitance will transfer more electrons per pulse and charge the output capacitor in less cycles. This number is tracked and is considered the arbitrary capacitance at a single time. It is inversely proportional to the capacitance. For the smart Sole system only the normalized change in pressure and therefore capacitance is required so this arbitrary value can give a good idea of pressure on the insole.

This system can be run much faster than is required for smartSole's resolution so an averaging of the capacitance over one hundred cycles can be used to reduce the noise of the system without slowing down collection.

Three pressure pads are used and located one at the heel and two at the ball of the foot. This configuration allows for front to back as well left to right isolation to determine balance of a single foot. With one of these units in each shoe the overall balance of each step can be determined to detect uneven walking gaits that can lead to foot dragging or discomfort.

3.2.3 *Bluetooth*

To enable wireless communication capabilities on the integrated smartSole platform, several options are available. Wifi, ZigBee, Bluetooth, Bluetooh Low-Energy, and custom radio communications are all considered. Filtering first by power consumption, Wifi and traditional Bluetooth are removed due to their increased power demand. Comparing ZigBee, BLE, and a custom option, BLE is chosen due to the low energy consumption, ease of development, and compatibility with common electronics.

To implement BLE communication, both BLE modules and SoC's are available. Modules have simpler interfaces, a lighter BOM, and are often pre-approved by the FCC. However, modules have a bulky form factor. The only device found that is comparable to a surface mount SMD was the Silicon Labs BGM12X series of System in Place (SiP) modules. However, slightly higher power consumption, higher cost, and lower transmit power makes that series less desireable than the Silicon Labs EFR32BG1 series of BLE SoC's.

The chosen BLE device is the Silicon Labs EFR32BG1B25632. It has the smallest available footprint, 10.5dBm output power, and can be fully programmed in C/C++ with the GCC compiler. The BLE software stack is run on board its ARM Cortex M4 processor, which is also used to run custom code, with limitations.

3.2.4 *Data Storage*

A compact, low-power flash memory device is required to store all recorded and processed data. To calculate the necessary capacity, all sensor variables are examined in table 1.

Table 1: Recorded data memory totals

Variable	Quantity	Data Type	Size Each (bits)	Total (bits)
Quaternion Angles	4	Float	32	128
Acceleration	3	Signed integer	16	48
Pressure Sensors	3	Unsigned Integer	32	96
Time: ms	1	Unsigned Integer	32	32
Time: Month	1	Unsigned Byte	8	8
Time Year:	1	Unsigned Byte	8	8
			Total:	**330 bits/frame**

Assuming a worst case scenario of 24 hours of constant recording, and a 30Hz sample rate, then 816Mbits of capacity is required. Rounding to the nearest common value, 1Gbit of capacity will leave significant margin for a single day of data collection.

Typical high-capacity memory devices are micro SDHC cards, or other similar NAND flash devices. The downfalls of these types of memory are the power consumption and physical form factor. A micro SD card may be thin enough to fit within an insole, but it requires a fragile connector, and is not tolerant of bending forces. Also, SDHC cards have a current consumption of over 100mA during write cycles depending on capacity, and require a minimum of 2.75 volts to operate.

Higher speed surface mount QSPI NOR Flash is available in capacities up to 1Gbit from Cycpress Semiconductor, and their FS-S series requires only 1.8V to operate, and is available in a small SO-16 package. The QSPI interface runs at speeds above 80MHz allowing the system to spend less time writing data, saving power. Also, current consumption for write operations is 60mA max, further reducing power consumption. Not needing a connector improves mechanical robustness, and saves cost.

3.2.5 *Microcontroller*

Operating as the brain of the integrated hardware, an appropriate MCU is critical for overall device performance and ease of development. The MCU requirements were set by required computing, required flash and SRAM, and power consumption. To determine the processing and peripheral requirements, an operation outline was compiled. Based on what specific actions the MCU needed to perform to control the integrated system, required search parameters, peripherals, and hardware capabilities were determined.
The main operations the MCU performs are:

- Collect data from MPU6050
- Measure capacitance of the pressure sensors
- Send data to be stored to SPI flash memory
- Oversee BLE communications and send/receive required data
- Run adaptive data collection control
- Adaptively set all device power modes

Collecting data from the MPU6050 motion sensor is done over a 400kHz I^2C interface. Initialization of the MPU6050 includes setting several parameters, and loading a large block of DMP code, which must be held in the MCU's flash. Also, a hardware interrupt line from the MPU6050 to the MCU eliminated the need for polling over I^2C for new data. This interrupt-based handling will drive the overall data sample and storage rate. The MCU will use the MPU6050 interrupt to synchronize the pressure sensing, and any other data collection. Then entire frames of data can be compiled and stored in flash. Also, to handle the output data of the MPU6050, 3D mathematic operations must be performed. This requires fast computation with floating point numbers, and of trigonometric functions.

Recording pressure data requires a method of measuring capacitance rapidly. Many MCU's have integrated touch sensing inputs. Different manufactures have different touch sensing methods, but they all are in essence a method of measuring relative capacitance. Even without a hardware touch or capacitance sensing peripheral it is possible to use simple digital IO pins to achieve a rough measurement. So long as each pin's input threshold is consistent, they can be used to measure an RC time constant. A minimum of three touch sensing channels are required.

Recording data requires a SPI interface, a standard MCU peripheral. However, the chosen SPI flash is capable of a Quad-SPI interface, and speeds of 80MHz and above. This favors faster processors, and those with QSPI interfaces. Having a MCU capable of reducing flash memory active write time is critical for conserving power, as writing to flash can draw more than 60mA, significantly more than any other component in the smartSole integrated system. High write speed achievable with QSPI reduce the write duty cycle, reducing overall power consumption.

Communication with the Blue Gecko device can be done with a variety of standard hardware interfaces. However, UART is a favorable choice as it requires only two lines for

full duplex communications, allowing for a smaller I/O count and smaller package. Also, the Blue Gecko has a LEUART connection, capable of running with only the standby 32.768kHz clock active. An MCU with a comparable LEUART would match well.

Adaptive data collection is only achievable with a significant degree of integrated data processing. Because the methods for efficiently implementing adaptive data collection are derived from neural network data processing, running a simplified neural network in the MCU is desired. This favors MCU's with faster clock speeds, hardware FPU's, and significant SRAM.

Power mode control is related to adaptive data collection. Setting every device in its lowest possible power state is always necessary, and driven by the real time data collection, processing, and communication requirements. Besides being able to determine power stated by processing data, extended sleep-mode capabilities are useful. For example, data collection and storage can, in some cases, be done with peripherals and DMA channels only, allowing the power-hungry core to remain halted, or run slowly.

With this list of requirements, and educated MCU part search was made. From this operations list, these parameters were deemed necessary:

- Communication peripherals: I^2C, QSPI, LEUART
- Integrated capacitive sensing
- Hardware floating point unit
- Low voltage operation & flexible power states
- Multiple DMA and interrupt channels available in standby/sleep modes
- Minimum 256kB Flash & 64kB SRAM
- Maximum clock speed of 40MHz or above
- Package dimentions no greater than 8mm x 8mm x 2.5mm (L x W x H), and not BGA

The best-suited MCU that fits these requirements was found to be the STM32L432KC.

Table 2: STM32L4KC Critical Specifications

80MHz ARM Cortex M4F
256kB Flash, 64kB SRAM
Min Supply Voltage: 1.71V
Power Modes: 6
84uA/MHz Run Mode
Hardware FPU
DSP Instruction Support
Quad SPI, LEUART, I^2C
14 channel DMA
Nested Vectored Interrupt Controller
QFN32 package

Additionally, a detailed HAL peripheral code library is supplied by STM, as well as the STM-Cube MCU configuration software. Small form-factor development Nucleo boards with integrated ST-Link debugger are available at a low cost as well. While newer, and therefore less universal than the STM32F4 series of microcontrollers, the STM32L4 series has significant performance advantages, and is a capable device for the smartSole platform.

3.2.6 *Power Supply and Control*

The power supply scheme is designed around the system voltage, current, and ripple requirements, as well maximum available battery size. Four main power modes are defined as Run, Standby, Sleep, and Home. Run mode is any mode where full data is being collected and processed, although the sample rate will be varied based on the minimum requirements. Standby mode is where the insole is still in use, but the user may be sitting or otherwise inactive. This mode requires periodic measurements to choose when to jump back to Run mode. Sleep mode is where the user is not wearing the smartSole device, and no base station BLE or wireless charging is present. This mode will go into the deepest sleep possible, waking infrequently to check if the user has put their shoes on again. Home mode is where wireless charging is available via the base mat, and BLE communications with base mat are available. This mode all sensors are set to be the same as Standby mode, but the MCU and Blue Gecko devices are fully active to send data and receive algorithm updates.

The physical power supply scheme consists of two power buses, a 1.9V line and 2.8V line. These two voltages allow all devices to be run at their minimum required voltages, conserving energy. A Texas Instruments TPS62400 dual buck converter supplies both voltages from a 3.7V LiPo battery. This converter has two modes: PWM, and low-power mode. The PWM mode is the standard fixed-frequency variable duty-cycle mode that most converters operate in. This mode gives a low output ripple, which is important for the accuracy of the pressure sensing. However, this mode is not efficiency when the output current decreases. This is dues to switching losses, and discontinuous mode operation. To regain efficiency in the low current draw states, the TPS62400 automatically switches into a fixed on-time variable off-time mode. This mode has a higher output ripple, but maintains 80% + efficiency down to 100uA of output current, illustrated in Figure 5.

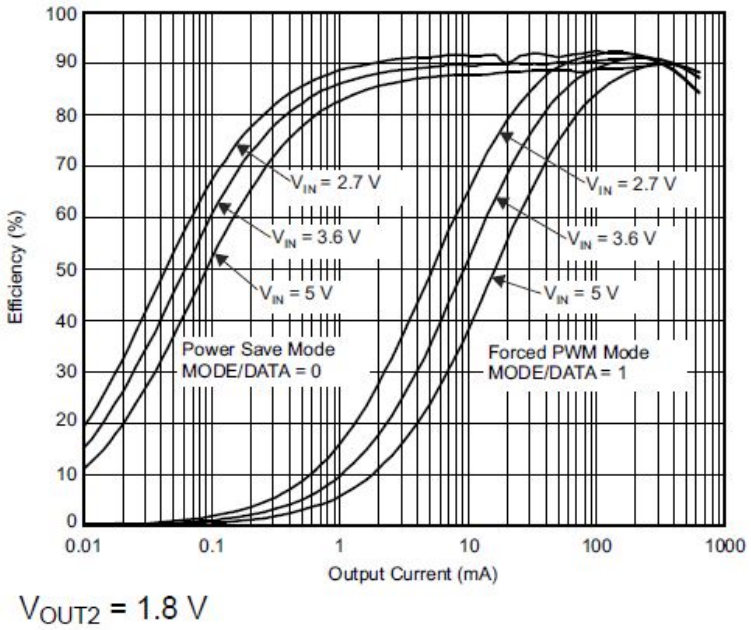


Figure 5: TPS62400 Efficiency vs output current

3.2.7 *Battery*

The smartSole device uses a 120mAh LiPo battery due to its low profile and high energy density. The high cell voltage allows all devices to be run off of the single cell. The charging profile of LiPo batteries simplifies the wireless charger controller. Care must be taken to not drain the battery much below 3.0 volts, or permanent loss of capacity will result. A MAX809 series reset controller automatically disables the TPS62400 outputs when the battery voltage falls below 2.93V. When the TPS62400 is in shutdown, the current draw is reduced to 32uA, giving the device an extended period of time before it must be charged of physically switched off.

3.2.8 *Wireless Charger Receiver*

The wireless charger is a resonant inductive charging solution. The smartSole device uses a custom solution to be able to charge through any common shoe thickness. The high frequency AC signal it outputs when charging is rectified, fed into a smoothing capacitor, and then into a buck-boost converter. The buck-boost converter allows the the battery to be charged within a large range of input voltages, and is controlled by maximum power point tracking algorithms to reduce charge times. The wireless charging receiver can also trigger BLE communications to the base mat to adjust charging parameters, and control power flow.

The charging algorithms and monitoring are facilitated by the MCU. Battery voltage and current sensing allow the MCU to precisely control the buck-boost converter to charge the battery.

The Rx coil itself is paced in the toe area of the insole to be closest to the surface of the charging mat. It's self-resonant frequency is over 5MHz, but the system resonant

frequency is brought down to 300kHz with a parallel loading capacitor. This allows for better transmitter efficiency.

3.3 External Hardware

3.3.1 *Wireless Charging Transmitter*

The base mat includes the transmitting side of the resonant inductive wireless charging system. There are several Tx coils running in parallel to increase power delivered to the receiver. The controller monitors voltage and current waveforms to detect the presence of a receiver, and can tune both output frequency and amplitude to charge the insole battery as quickly as possible.

A dedicated MCU performs all of the voltage and current waveform analysis. Both left and right insoles are charged simultaneously, and each one is tuned individually for overall maximum performance. ’

The actual output stage of the transmitter is a high frequency AC sine inverter. It is supplied from a 20V wall adapter, typical of laptop chargers. A high-power H bridge forms the output stage, and is switched by a PWM modulator. Harmonic distortion is not critical in this application, so the modulation frequency ratio does not need to be high, saving on switching losses. The PWM switching method avoids a square-wave output, reducing high current spike transients from the power supply.

The output of the high frequency inverter is fed into several identical Tx coils. Loading capacitance tunes the Tx system into the same resonant frequency as the receiving coil.

3.3.2 *Bluetooth Base*

In order to offload the collected gait data a smart docking system is proposed. Once the smartSole is placed onto the mat for charging an automatic BLE connection is established with a Raspberry Pi Zero built into the station. It runs a Linux OS which is the best environment for use with low energy Bluetooth and communicates with the Blue Gecko SoC via a python based Bluetooth library called bluepy.

The system scans for the specific UUID associated with the smartSole GATT profile and initiates connection when it is discovered. The data is streamed asynchronously and without confirmation required in order to generate a 1 mega bit per second throughput. The Blue gecko sends the data in 126 Byte packages with the first byte denoting the packet structure. The Raspberry Pi decodes this structure and converts the sent information to the decimal system to be fed into machine learning algorithms. This leads to up to 255 unique packet structures to send other information to the Raspberry Pi and to send information back to the Blue Gecko without needing a physical connection.

3.3.3 *Local Data Storage and Processing*

The Raspberry Pi Zero W used in the base mat runs off of a SDHC card, giving it a much higher memory capacity than the integrated insole. During every charge cycle, the base mat pulls all recorded insole data via BLE, freeing up the limited SPI flash in the integrated system. The Raspberry Pi Zero W runs linux with a 1GHz single core processor,

and 512MB of RAM. This, along with Python compatibility, allows it to pre-process the data. Also, given that charge cycles last overnight while the user sleeps, ample time is available for data pre-processing, and analysis. In the absence of an internet connection, the Raspberry Pi in the base mat can be tasked with performing all data analysis, albeit with limitations.

3.4 Data Collection

To maximize the utility of power, memory, and computing resources, a dynamic data collection control scheme is implemented. By examining the performance of the data analysis algorithms, data collection can shift to focus on the more important timing and variables associated with the users gait.

The first order of adaptive sensing is to determine if the user is wearing the smartSole device. This is accomplished through the capacitive pressure sensors. Even with no applied pressure, the resting measurement value for these sensors changes drastically with the presence of a foot in the user's shoe. Luckily, performing this measurement requires very little active processing time or power, and can be done at any frequency picked by the MCU.

Adaptive sensing determines when it is most critical to record data while the user is wearing the smartSole device. For example, data is not needed when the user is sitting down. This can be determined in a low power state by a combination of pressure monitoring, and motion activated interrupts. The MPU6050 has a low sample rate and low-power mode capable of waking up the smartSole device when movement over a certain threshold is detected. Detection of this movement generates hardware interrupts capable of waking up the MCU from deep sleep states.

Varying the rate at which data is collected allows for the most efficient use of memory and processing resources. This is driven by the development of the data processing algorithms. Through their development, the most useful sensors axis and stride points will become evident. Guided by this information, insole MCU processes are developed for dynamically optimizing all sensor axis sample rates.

3.5 Communications

Successful data transmission is integral for the smartSole system as the bulk of the processing capabilities are contained in off site servers and the Raspberry PI Zero docking station. Bluetooth low energy (BLE) version 4.0 is selected for it's low power consumption and high data transfer rate rate over reasonable distances.

The critical requirement for a Bluetooth device within the insole are low power consumption, sufficient range, and a slim package profile to increase user comfort. The Blue Gecko EFR32BG1 SoC meets these restraints and is comprised of a PCB radio antenna, ARM Cortex M4 processor, and general purpose I/O pins for interfacing. The Blue Gecko development kit comes with a built in serial debugger for simple code flashing and corrections as well as a host of peripheral devices to facilitate easy test environments.

The ARM processor supports both C++ and a proprietary scripting language which can both control the on board Bluetooth stack, however C++ was selected as it gives more control over interrupts and system parameters. The Blue Gecko can support a fast bit rate

with the Raspberry Pi which is useful as the gait monitoring systems generate a significant amount of data over a day's use.

Another important feature of the ERF34BG SoC is the ability to enter a low power consumption "Deep Sleep" mode which reduces current draw down to 3.3uA. This is initiated while Bluetooth capabilities are not required to lengthen battery life. The board also supports Low energy UART wired connections between itself and the STM32L4 MCU which is used to transmit collected data for transmission. This LEUART connection has the added benefit of transmission while the Blue Gecko is in Deep Sleep mode limiting the time needed in high power mode and allowing the Bluetooth connection to be woken up by the main MCU.

When the smartSole is connected to the charging mat a unique value is sent through the LEUART connection which initiates wake up of the Bluetooth GATT profile. This is possible because the STM MCU is the effective master controller with the ERF32BG simply listening to the LEUART connection for instructions.

Once a Bluetooth connection has been established the smartSole low energy GATT profile is initiated which handles the incoming gait data from the main processor. A simple packet structure is implemented that maximizes the data throughput as well as multiple packet profiles for selective data offloading.

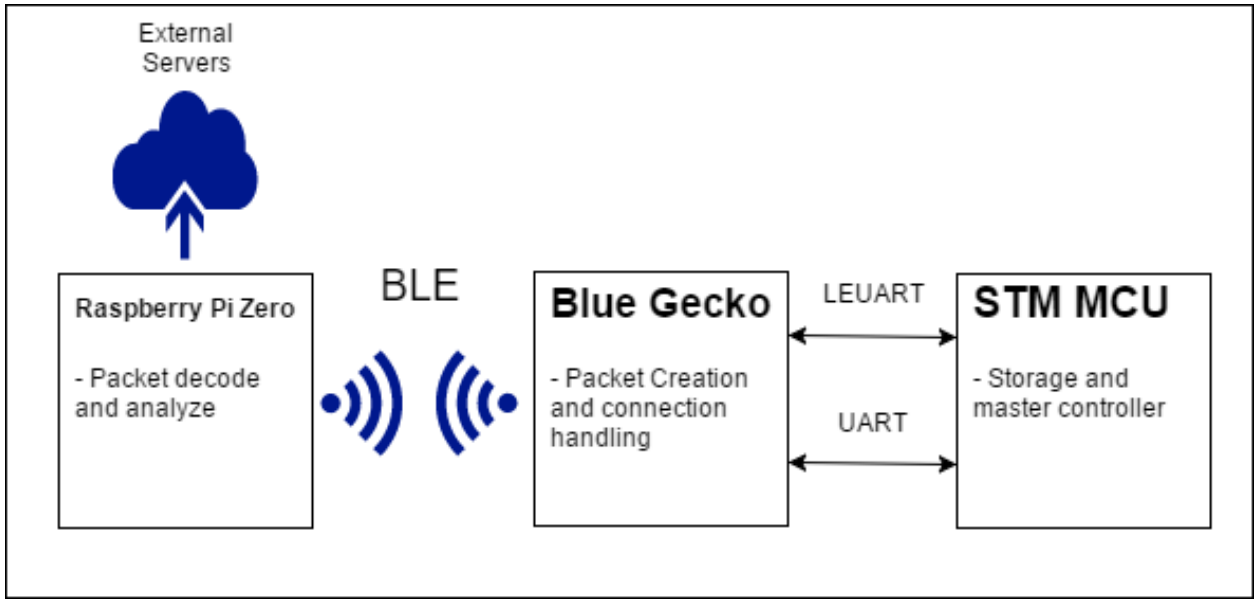


Figure 6: Communications Block Diagram

3.5.1 *smartSole GATT Profile*

The layout of the embedded C++ Code within the ERF32BG was designed to operate efficiently and utilize low power mode to minimize battery drain. While this is less important while offloading the data to the Raspberry Pi zero because device will be charging for a decent length of time, it opens up the possibility for real time data analysis through a smart phone connection. In this case battery consumption should be limited to avoid power loss during the day.

When discussing Bluetooth operation there are two main structures implemented on the chip. The first is the Bluetooth stack, which requires up to 60% of the available memory and does not need to be personally configured. This stack handles all of the Bluetooth communications and implements proper protocol between the sending and receiving devices. This comes standard in blank templates in the silicon labs Simplicity Studios Development tool.

The second are GATT profiles for specific functionalities based on either the send requirements of the Blue Gecko or the preference of the connected device. The GATT consists of Services and Characteristics of the service to organize how data is processed and sent. In the smartSole Bluetooth code a single data carrying GATT was created for simplicity as it can handle all of the necessary data queuing and sleep cycling.

Initially the Blue Gecko defines the necessary components and attributes for LEUART, board configuration, and header files. It also configures a single connection that is password protected to protect the user's data. The number of allowed connections can be increased for a two insole system with one ERF32BG configured as a slave to the main SoC. This eliminates the need to have multiple STM MCU's for a single system and reduces the cost.

It then enters a main function and initializes Bluetooth scanning. When a device is connected to the Blue Gecko the GATT profile is initialized and a signal is sent to the STM MCU to begin serial sending data through the LEUART connection. This is fed into one of four dedicated array locations until the final element is sent and loaded. Then an event is trigged by this filled array which calls the Bluetooth Packet creator and send function within the GATT which sends the elements of the array through a bit-stream to a hexadecimal value and ultimately to the Bluetooth stack. This is repeated to generate a continuous data stream. If there is either a loss of Bluetooth connection or a stop in data received from the main MCU then the Blue Gecko is set to initiate deep sleep mode, lowering current usage by a factor of one thousand.

3.5.2 *Connections and packet structure*

The maximum Packet size for the current ARM SDK version is 125 Bytes of useful space with a one initial Byte to define the packet structure. The current packet structure contains all of the measured parameters which are represented as 16 bit numbers as well as a time stamp which can be up to 32 bits. This does not fill up the entire 125 Bytes so future data collection can be added.

The Bluetooth stack takes about 7.5 ms to complete a single send or receive action. Generally there is a received notification packet sent back to the Blue Gecko to ensure no data is lost which limits the throughput to 66kbps in this configuration. It is possible to increase this up to 1Mbps by eliminating this confirmation packet so that multiple 125 Byte packets are sent in one send action. Each can be sent in 800μ seconds allowing up to nine packets per action as well as during the previous receive action. This fast enough to easily offload the data collected in a single day overnight.

The LEUART connection consists of 10 total data framing bits with one start and one stop bit. The maximum baud rate for a low energy UART connection is only 9600 which may be too slow to offload the data from the STM MCU, however its ability to function while the Blue Gecko is in deep sleep mode makes it invaluable for initial wake up. A secondary

UART connection is considered as a backup for the LEUART to increase the baud rate to a more reasonable range of 115200 to match the blue-tooth connection. Both are implemented as both send and revive for greater flexibility.

3.6 Data Processing

3.6.1 *Statistical Analysis*

The device is meant to be a tool that complements how doctors make a diagnosis on a person's walking habits. As such, it is needed to understand what podiatrists are particularly looking at and providing this data in an easy to process way. Research in this topic led to six particular parameters that are important indications of health walking: velocity, stride length, cadence, stance time, swing time, and double support.

To calculate these parameters there are nineteen different datasets that could be observed. Three are from the pressure sensors, and the next sixteen are either direct motion data, or various mathematical representations of the motion data. The one that showed most useful was pitch. It was very intuitive to understand that relative maximums correspond with when the heel strikes the ground and how relative minimums correspond with when the toes leave the ground. Calculating these parameters then simply became extracting when these points in time occur and finding the difference between them. Most of the testing was done in MatLab as it is much easier to plot and analyze the data, but ultimately this was all done in Python.

Before these parameters are calculated however the data needs to be expressed clearly. To do this the data is smoothed using a five point moving average that takes in account the points around it. This filters noise out of data and expresses clearly where the relative maximum and minimums are. From here the peaks and troughs are detected through a simple three point comparison. Derivation would technically have worked as well, but since the data isn't completely continuous, derivation wouldn't have performed as desired.

4 RESULTS

4.1 Integrated Hardware

Progress in the integrated hardware culminated with a working prototype sensor insole and several development building blocks for future devices. Motion and pressure sensing was successfully integrated into a store-bought foam insole with no adverse affect on the inserted hardware, or the comfort of the insole. The MCU, flash memory, battery, and bluetooth module used on the prototype straps onto the users ankle.

Figure 7: smartSole Prototype V1.0

Additionally, development with the STM32L432KC MCU has started with the Nucleo board. BLE communications have been achieved with a Blue Gecko development kit, and a breakout radio board PCB has been designed and soldered. The TPS62400 dual buck converter has been integrated into a test PCB, and compatibility with the MCU has been confirmed.

4.1.1 *Motion Sensing*

Successfully integrating the MPU6050 into a foam insole has been a significant proof of concept for the project. Active MPU6050 features on prototype V1.0 include:

- Complete 6-axis raw data collection
- DMP activation and quaternion angle outputs
- One-time FIFO rate configuration
- Comfortable integration into the foam insole

- Proof the MPU6050 can withstand typical physical stresses

The current use of the MPU6050 relies on a open source code library developed for the Arduino platform. This library takes demo DMP code from Invensense and supplies it in precompiled form, along with functions for initialization and some configuration. While completely successful in allowing accurate data to be collected, the code is out-dated, and a newer library is available from Invensense. However, it is not directly compatible with the STM32L4.

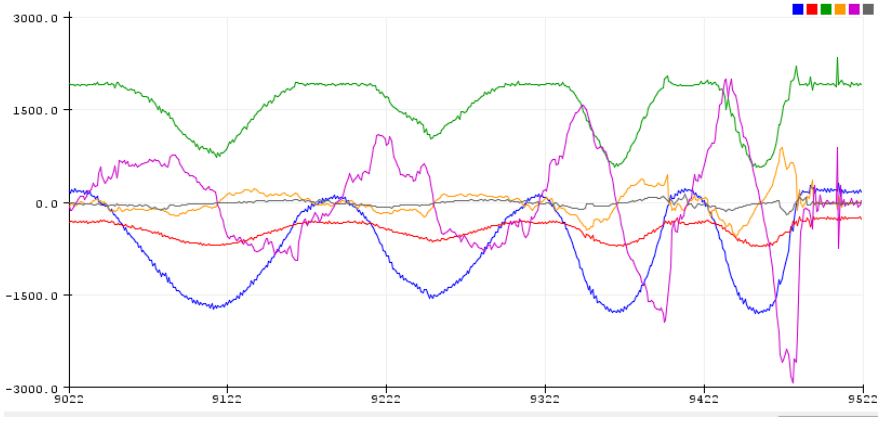


Figure 8: MPU6050 1st raw data
X axis: Time (ms) Y axis: g, rad/s (not scaled)

Efforts to port the Invensense code to the STM32L4 have faced several obstacles, as the closest available example is for a STM32F4 series MCU using completely different peripheral libraries. An understanding of the core library functions has been gained, but no hardware testing has been achieved yet.

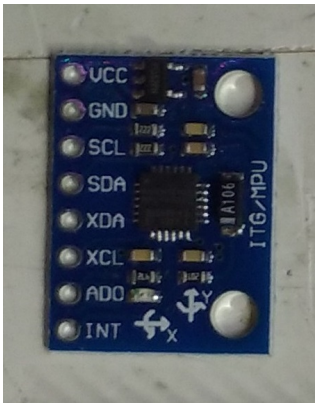


Figure 9: MPU6050 breakout

With physical restrains on integrated hardware being a significant concern for the project as a whole, MPU6050 integration success is important. A small off-the-shelf breakout board is used in the current prototype. The foam insole was carved to accept the board with

some room to spare, and double-sided foam tape secures the MPU6050 in place. The success of the prototype proves that the structural support of a thin but rigid PCB is enough to keep the MPU6050 stable during walking and jogging activities. Because it is comfortably integrated in the insole, even with the rigid PCB, it is a telling proof of concept that the entire integrated system could be fit inside and insole.

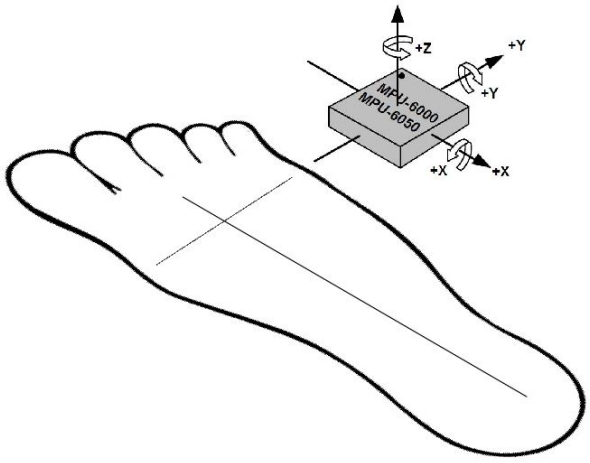


Figure 10: Prototype V1.0 MPU6050 axis alignment with user's foot

The MPU6050 data has proven critical for software development, as the pitch angle (Y axis rotation of the foot, shown in Figure 10 of the user's foot quickly became the most import variable for gait classification. The motion data from the prototype has been proven sufficient for initial gait classification tests.

4.1.2 *Pressure Sensing*

The current prototype V1.0 uses the same capacitive sheet implementation to generate the foots pressure data, however the connection to the board and the way to capacitance is determined is different. The Arduino configuration requires both a send and receive pin for every pressure sensor implemented so the pin usage is higher than the STM MCU configuration.

A similar RC circuit is implemented without the output capacitor. The Arduino quickly censured the sensor is discharge to 0V by setting the receive pin temporarily to an output low. Then the sensor is charged more slowly through R_s until the sensor voltage crosses the digital threshold of the receive pin. The time this voltage change takes is determined by the RC time constant of the system, which varies as the conductive surfaces capacitance changes with pressure. This counter creates an arbitrary capacitance for each sensor for which the ΔC can be tracked, giving pressure data.

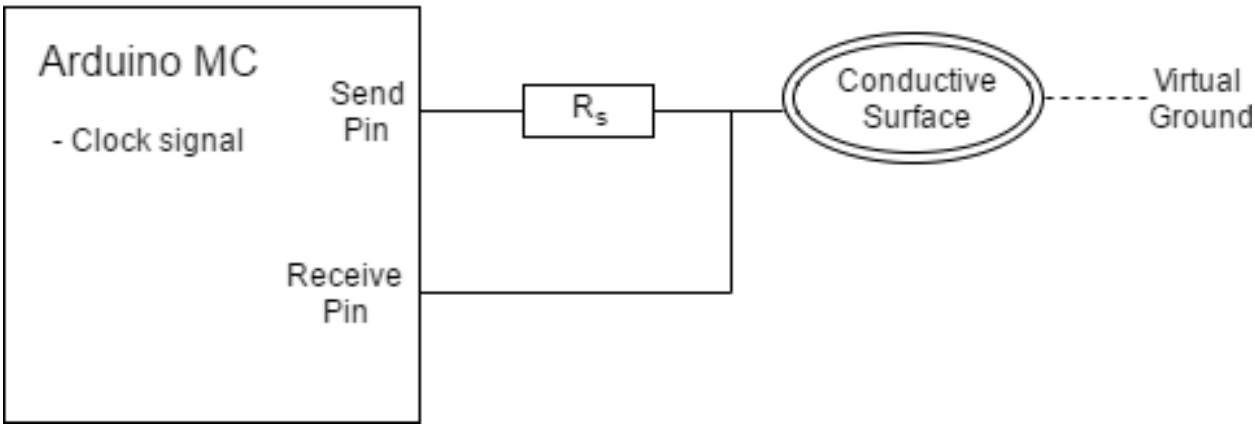


Figure 11: Arduino CapSense configuration

The system is running at about 10ms per sense which can be used to find the general capacitance of the conductive surface through the equations.

$$\tau = RC \tag{2}$$

$$V = V_0(1 - e^{\frac{-t}{\tau}}) \tag{3}$$

When no foot is applied the capacitance is about 5nF, and is increased to up to 11nF with full pressure. This gives a good idea of the pressure mapping of the foot when comparing the three sensors implemented on the insole prototype.

4.1.3 *Bluetooth*

Blue Gecko development is done with a development kit, shown in Figure 12. However, due to the size of the entire development kit, a smaller breakout board was designed and assembled, shown in Figure 13.

Figure 12: Blue gecko development kit

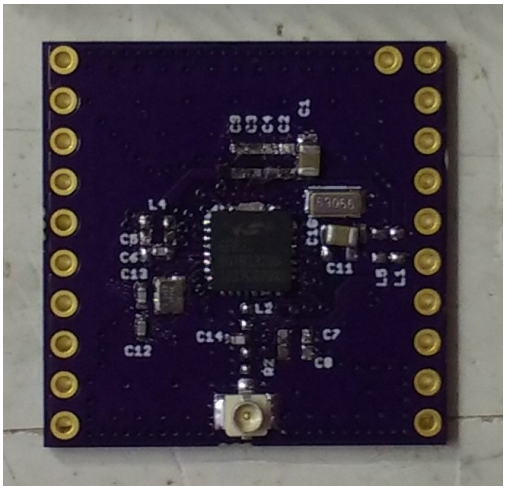


Figure 13: Blue Gecko breakout board

Bluetooth development is focused on software and configuration. Initial prototyping requirements did not require BLE capabilities, so the Blue Gecko breakout board remains untested, but is in line to be integrated into the next prototype.

4.1.4 *Data Storage*

Prototype V1.0 data storage is done with a micro SDHC card running in SPI mode, using a FAT32 file system, and storing data as text. This streamlines the development process, using existing Arduino libraries, and makes data importing easy for software development. Power consumption is not a prototype concern.

For a V2.0 prototype Cypress FS-S QSPI flash will be implemented. However, interface code is not readily available and will need to be built out. with the STM32L432KC.

4.1.5 *Microcontroller*

The prototype V1.0 uses an Arduino Micro board, with a ATMEGA32u4 8-bit AVR MCU. This device is very limited compared to the overall scope of the project, and is incapable of running all prototype hardware features simultaneously due to program flash limitations.

The MPU6050 code alone takes up 46% of the MCU's 32kB of program flash. This is due to the pre-compiled DMP algorithm code needing to be loaded to the MPU6050 every power cycle. Even with the unofficial library that the Arduino is running, functionality in the scope of the prototype V1.0 is not impaired. Moving MPU6050 code to the STM32L4 MCU requires intensive porting of code supplied by Invensense.

Development with the STM32L4 MCU started with adapting the Nucleo development board's power supply scheme to accept voltages below 3.3V. The board comes with an integrated programmer/debugger, but in the standard configuration if the debugger is connected but not powered, the MCU's reset pin will be indefinitely pulled low despite the MCU's internal pullup resistor. Disconnecting the reset line between the debugger and MCU would allow the MCU to run on it's own power, but remove programming and debugging

capability. A small SMD MOSFET was used to create a circuit that allowed the debugger to reset the MCU only when the debugger was powered. This setup allows the Nucleo board to run down to the MCU's minimum voltage of 1.71V, but with a USB programming connection this voltage is safely overridden by a regulated 3.3V powering both the MCU and debugger.

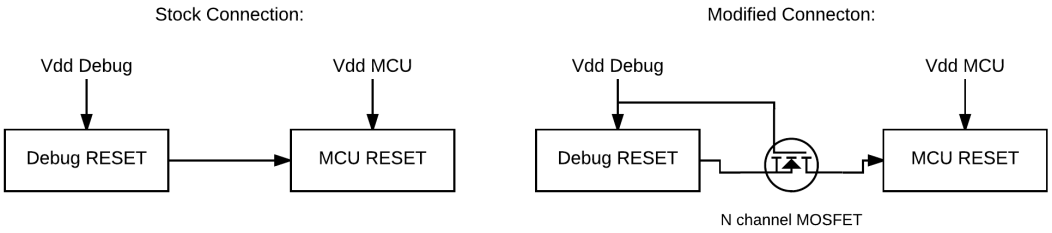


Figure 14: Nucleo board low-voltage reset modification schematic

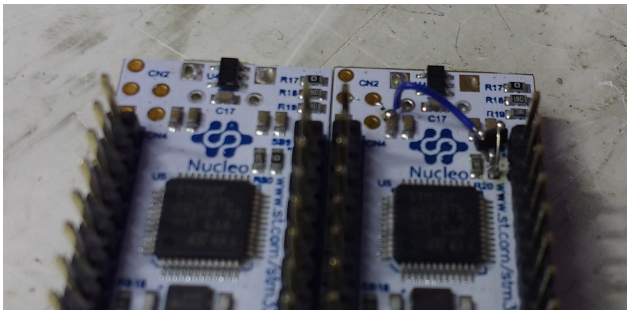


Figure 15: Nucleo board low-voltage reset modification board

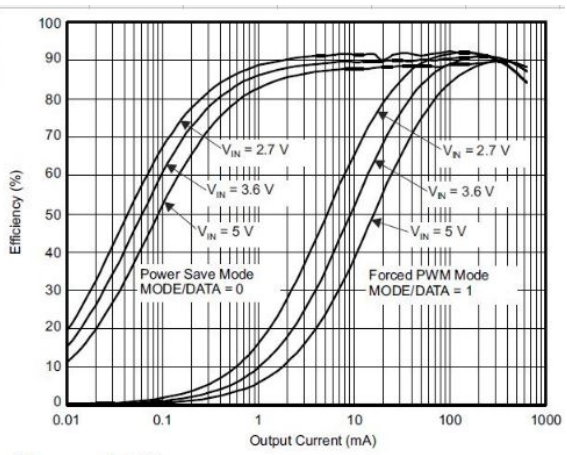
The Nucleo board only required a micro USB connection to a computer to be programmed and debugged. The STM32 Cube software allows for simple device configuration code generation. This tool is used as a plugin for the Atollic TrueStudio IDE. This combined with the provided HAL peripheral libraries has allowed progress to be made developing the MCU software.

The Nucleo board allows for a second prototype to be constructed using all the final parts, but in debugging friendly setup. The final device schematic and embedded software will be developed through incremental prototyping with separate breakout boards. That way errors are easily isolated, and are faster and cheaper to fix.

4.1.6 *Power Supply and Control*

Power supply development started with initial calculations. A compilation of all device power consumptions in three run modes allows the average current draw characteristics to be determined. Combined with the TPS62400 buck converter efficiency charts, an accurate depiction of battery current consumption is created.

Low Voltage Supply	1.90V	Battery Capacity	120.00mAh	put 80% of battery rated cap. here	Total Draw
High Voltage Supply	2.80V	Battery Voltage	3.7V		2.59mA
					Run Time
	Run Mode	Standby	Sleep		46.38 Hours
Duty Cycle	25.00%	40.00%	35.00%	0.00%	Left
Eqiv. 1 day hours	6.0 Hours	9.6 Hours	8.4 Hours		
total battery life hrs	11.60 Hours	18.55 Hours	16.23 Hours		
STM32L432KC	Run Mode	Standby	Sleep		
MCU Pwr Mode	Run	LP Run	Standby		
VREG Range	1	2	2		
Supply Voltage	1.90V	1.90V	1.90V		
Core MHz	40MHz	2000kHz			
uA/MHz	170.0uA/MHz	210.0uA/MHz			
Current	6.8mA	420uA	2uA		
Power	12.9mW	0.798mW	0.004mW		
MPU 6050	Run Mode	Standby	Sleep		
MPU Mode	ACC+GYR+MPU	ACC LP	ACC LP		
Supply Voltage	2.80V	2.80V	2.80V		
Refresh Rate	8000Hz	1.25Hz			
mA	3.9mA	10uA	5uA		
Power	10.920mW	0.028mW	0.014mW		
EFR32BG	Run Mode	Standby	Sleep		
MCU Pwr Mode	EM0	EM2	EM2		
BLE Available?	YES	NO	NO	No BLE means complete device disconnection.	
DC-DC?	NO	NO	NO		
Supply Voltage	1.90V	1.90V	1.90V		
mA	4.2mA	4.0uA	4.0uA		
Power	8.0mW	0.0076mW	0.0076mW		
SPI Flash	Run Mode	Standby	Sleep		
Action	Write	Standby	Deep sleep		
Duty Cycle	0.05%				
Supply Voltage	1.90V	1.90V	1.90V		
ma (avg)	0.030mA	140uA	14uA		
Power	0.057mW	0.266mW	0.027mW		
LV Draw	11.03mA	564uA	20uA		
Efficiency	92.00%	85.00%	10.00%	based on 1.8V chart	
Battery draw	6.16mA	341uA	103uA		
HV Draw	4.2mA	4.0uA	4uA		
Efficiency	92.00%	82.00%	10.00%	based on 3.3V chart	
Battery draw	3.45mA	4uA	30uA		
Mode Totals	9.61mA	344uA	133uA		
Weighted Total		2.59mA		Battery Draw	
Run Time		46.38 Hours			



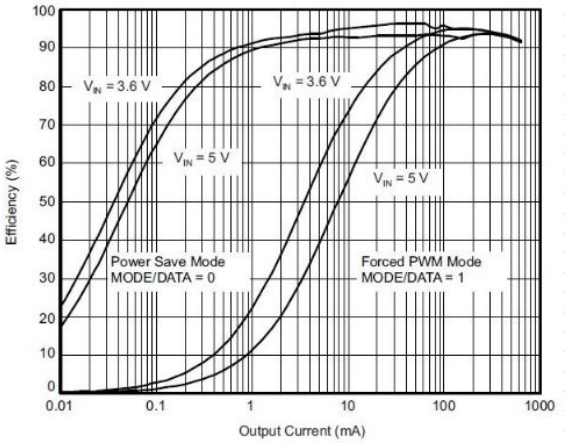


Figure 16: Power consumption calculations

Several variables determine the calculated current draw and battery life. Estimated duty cycles of each non-charging power mode are set. The nominal battery capacity of 120mAh is used, giving a battery life of up to 50 hours of typical use. The variables with the largest effect on battery life are:

1. User active time
2. MCU average clock speed in Run
3. Auxiliary BLE connections

This conclusion is valuable, as it helps guide MCU software development to reduce active MCU time and average clock speed. The other two significant variables require research into what typical application values will be.

A prototype power supply board was designed around the TPS62400 with testing and version 2.0 prototyping in mind. It includes PCB pads for various output filtering to determine how much supply ripple the rest of the system can handle. It has two nominal outputs of 1.9 and 2.8V, with automatic PWM and low-power mode switching enabled.

A high accuracy high-side current sense amplifier (Texas Instruments INA225) allows easy monitoring of the actual battery draw, and has connections for external power so its presence does not affect current draw. A MAX809 reset supervisory IC is re-purposed as a low voltage cutoff for the LiPo. It is set up to disable both TPS62400 outputs as soon as the battery voltage falls below 2.93V. A linear LiPo charging circuit is also integrated to facilitate easy re-charging of future prototypes.

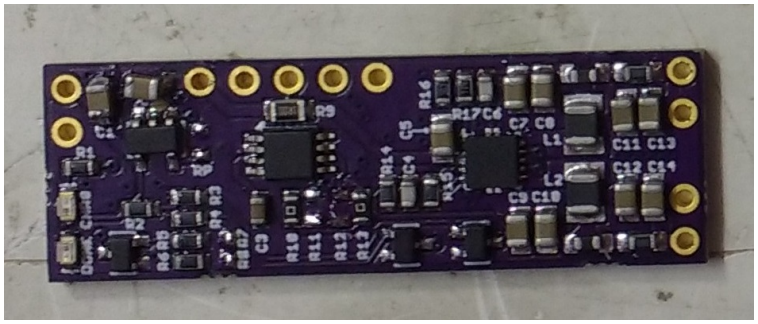


Figure 17: TPS62400 power supply board

The prototype V1.0 uses two standard 14500 LiPo batteries in series and linear regulator to run the Arduino board at 5.0V. This bulky setup has worked because the prototype is power-hungry, and the large battery capacity gives the prototype a long life for extended data collection. Research for future prototypes has found sources for low-profile single-cell LiPo batteries. 120mAh batteries with a thickness of less than 4mm are readily available.

4.1.7 *Wireless Charging*

Initial wireless charging development consists of a set of resonant coils and examination of their behavior with various loading capacitance. Figure 18 shows a test charging coil, with physical specification in table 3

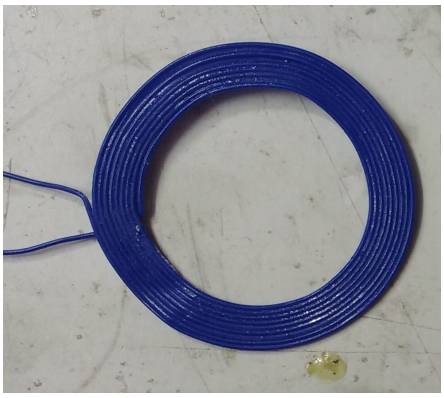


Figure 18: Wireless charging coil

Table 3: Test coil physical specifications

Outer Diameter	40mm
Inner Diameter	27mm
Turns	10
Wire Gauge	26 AWG solid

Measurement of the static coil properties was done using a signal generator and an oscilloscope. The coil was excited with a sinewave voltage, and the coil current and voltage were probed. The coil can be simply modeled as a parallel LC circuit. The resonant frequency of this circuit is given by:

$$\omega_0 = \frac{1}{\sqrt{LC}}$$

The complex impedance of the coil is given by:

$$Z_{coil} = \frac{1}{\frac{1}{j\omega L} + j\omega C}$$

At the resonant frequency, $\omega = \omega_0$, coil impedance simplifies to:

$$Z_{coil} = \frac{1}{0} = \infty$$

So while monitoring coil voltage and current, resonant frequency is signified by a near-zero current, and current and voltage being perfectly in phase. Any other excitation frequency should result in a non-zero current, and some phase shift. This test determined the LC constant of the coil, but not the individual values. The next step was to make the justified assumption that the coil capacitance is small. Therefore, adding a large known capacitor allows an accurate measurement of the coil inductance. Adding 47nF capacitor yielded a f_0 of 309.7kHz. From these measurements coil inductance and capacitance were calculated, shown in Table 4. This measurement and calculation process was repeated with several capacitors, and the results averaged to account for part tolerances.

Table 4: Test coil electrical specifications

DC Resistance	1.5Ω
Measured Inductance	5.6μH
Measured Capacitance	140pF
Measured Self-resonant Frequency	5.684MHz

To test the wireless charging concept, two nearly identical coils were held with a separation of 19mm. The Tx was energized by a lab frequency generator with oscilloscope probes monitoring voltage across the coil, and current through it, as in the single coil test. The Rx coil was hooked to a fixed resistive load, and it's voltage hooked a third oscilloscope probe. Sweeping loading capacitance across several values gave a voltage gain profile based on the loading capacitance and system resonant frequency. Figure 19 shows that the maximum

voltage gain of -24.8dB occurs with a loading capacitance of 47nF, and resonant frequency of 309.769KHz.

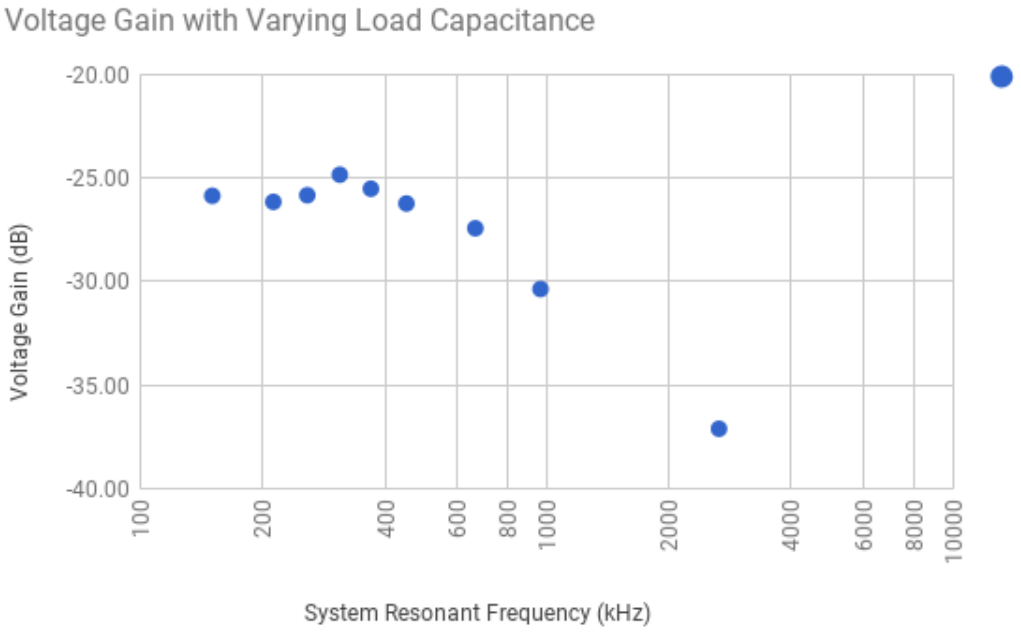


Figure 19: Wireless charging test: 1 Tx coil, 1 Rx coil, 19mm separation, 83Ω Rx load

Based on this test with a 83Ω load, even with a full 20V input signal to the Tx coil and 100% charging efficiency, a 120mAh LiPo battery would take 35 hours to charge. Increasing the voltage gain for future devices can be achieved with multiple parallel Tx coils, as well as experimentation with coil geometries, and using less resistive wire.

4.2 External Hardware

4.2.1 *Bluetooth Base*

The Raspberry Pi Zero Bluetooth base is currently substituted for a Linux based laptop running the same python library that will be used in the final product. It will be easy to port the code over once the station is created. The Bluepy library has enabled simple communication between the laptop and Blue Gecko which is decoded into the correct numbers desired. The python script connects to the Gecko, reads all GATT attributes, and then begins the read function allowing for a data stream.

4.2.2 *Local Data storage and Processing*

As with Bluetooth base functionality, early prototyping has favored the use of linux laptops for development. In this case all data storage, pre-processing, and processing is performed on the powerful laptop with Python. However, due to platform compatibility these features are easily migrated to the Raspberry Pi Zero W platform.

4.3 Adaptive Sensing

Testing with the smartSole V1.0 prototype and initial data analysis has given distinct insight into the valuable variables for classification, as well as critical timing. Initial data processing software development shows that the pitch angle of the user's foot is the most important variable in determining gait metrics, and classifying gait health. Additionally, within one stride the local maximums and minimums are most important in calculating gait metrics. Based on this insight alone, a focus on pitch data, as well as increasing sample rate near pitch maximums and minimums forms an outline for advanced adaptive sensing.

Additionally, sensing of the user's foot presence is achieved with knowledge of empty and occupied shoe pressure data. The capacitive sensors are able to pick up the presence of a foot, even with no applied pressure. This is due to that the user's foot drastically affects the electric fields of the capacitive sensor plates.

4.4 Communications

While the Blue Gecko SoC is close to full operation, it was designed to function with the second simpler prototype. For the current prototype a simple Serial Bluetooth Slave Wireless RF Transceiver Module was used instead. It has many drawbacks compared to the Blue Gecko, but it allowed a simple demonstration.

The transceiver module is running Bluetooth 2.0 which is an older, more power hungry version than BLE. It uses a a fast UART connection with a 115200 baud rate to receive pitch data from the Arduino MCU which is serial ported to the Bluetooth stack. This system sends single bytes across the Bluetooth connection to the Linux based laptop receiver. This doesn't allow for easy packet encoding, but allows a continuous data stream for a single variable to be fed into the machine learning algorithms on-board the laptop.

The next step in communications with the insole will be to polish and fully utilize the ERF32BG C++ program so that the system is more responsive, controllable, and efficient. The current program has initial connective capabilities as well as simple LEUART based connection triggers. It can send readable data to the receiver module which can be decoded. LEUART wakeup triggers will also be implemented so that the system can sleep whenever it is not required to be searching or exporting data.

4.5 Data Processing

4.5.1 *Dataset Collection*

In order to achieve high algorithm accuracy, a large dataset is needed with examples of healthy, and unhealthy walking. Although the ideal datasets would constitute of data collected from various individuals and senior citizens; however, in the timespan of the smartSole project, to achieve acceptable accuracy, a majority of data was collected from team members, where unhealthy walking was imitated with as much accuracy as possible.

Data that is recorded and utilized from the sensors are as follows:

1. Time (ms)

2. Quaternion Angles [x, y, z, w]
3. Tait-Bryan Angles [Yaw, Pitch, Roll]
4. Raw Accelerations [x, y, z]
5. Pressure Sensor Information [Heel, Ball, Toe]

4.5.2 *Dataset Creation*

When the device is power cycled, it writes to a file on the SD card a header containing the information of the data being saved. After the header is printed, the data from the sensors are echoed to the same file in a CSV format. This makes it easy to create different datasets without reconfiguring or uploading the data, as different data trials can be separated by a header line simply by power cycling the device. Thus the contents of the file can be parsed and restructured easily with a simple script.

For our training datasets, various MATLAB scripts were used to parse, clean, and label the data trials from the insole. The resulting structure consists of a T by 2 cell array named *dataset*, where T is the amount of trials recorded, the first column consisting of matrices containing the data, and the second consisting of labels corresponding to the left-adjacent cell. Also included in the file is a cell array, *header*, where labels of the columns of the data matrices are contained.

4.5.3 *Statistical Analysis*

Preprocessing: Before the system is able to begin conducting statistical analysis on the collected data, there are three main preprocessing steps that need to be conducted: smoothing, peak detection, and step detection. By preprocessing the data, the system is able to have increase the accuracy of the data and readily calculate gait parameters for proper diagnosis of healthy walking.

Smoothing: To ensure that the data is able to run with as much accuracy as possible, smartSole implements a five point moving average. This simply takes in account the two values before the point and the two values after the point to create an average value for that location.

$$\begin{aligned} y(1) &= x(1) \\ y(2) &= \frac{x(1)+x(2)+x(3)}{3} \\ y(3) &= \frac{x(1)+x(2)+x(3)+x(4)+x(5)}{5} \\ y(4) &= \frac{x(2)+x(3)+x(4)+x(5)+x(6)}{5} \\ y(i) &= \frac{x(i-2)+x(i-1)+x(i)+x(i+1)+x(i+2)}{5} \end{aligned} \tag{4}$$

Through this smoothing, the system is able to filter out some of the noise and achieve a more continuous dataset. By nature of how the peak detection works, if the system does not smooth the data, the system will believe there are consecutive troughs without any peaks in between or vice versa. Ideally the data should alternate between peak and trough as this represents a foot stepping down and lifting up. This data is critical in the calculation of gait parameters.

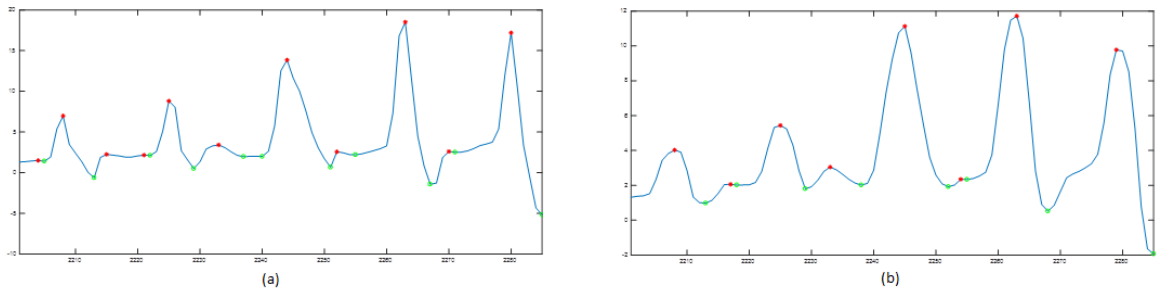


Figure 20: (a) Pre-Smoothing, (b) Post-Smoothing

As seen in Figure /reffig:smoothing the peak (red asterisks) and trough (green circles) detection works more optimally after smoothing. It eliminates consecutive peaks and troughs while still maintaining the general form of the function. Although this does slightly affect the magnitude of the function, the important part of the device is to determine the time that elapses between peaks and troughs, which will be explained in greater detail.

Peak/Step Detection: Calculating gait parameters is heavily dependent on knowing the time at certain points of a step. The two most important times that needs to be recognized is when the heel first strikes the ground and when the toes leave the ground. One way to easily recognize these moments happen is by analyzing the pitch of the foot. The times where the pitch is at a relative maximum are the times where the heel strikes the ground. The times where the pitch is at a relative minimum are the times where the toes leave the ground. If the device is able to find the relative maximum and minimums, many gait parameters can be found from the collected data.

$$x(i-1) > x(i-2) \text{ and } x(i-1) > x(i) \tag{5}$$

$$x(i-1) < x(i-2) \text{ and } x(i-1) < x(i) \tag{6}$$

To find the relative maximums and minimums, the device simply needs to constantly do a three point comparison as it collects data. Equation 3 shows the comparison that smartSole conducts in order to determine if it's a relative maximum As data keeps being added smartSole will consistently check for this condition and keep track of when this occurred. Likewise, smartSole follows equation 4 to keep track of when a relative minimum occurs.

The locations of the detected peaks are then used to cut the datastream into steps. The steps are stored in a list of matrices, and can be overlaid to visual the standard deviation. In Figure 21, one hundred steps for three variables (out of nineteen) are overlaid. In this

case, steps were cut by applying smoothing and peak detection to the pitch variable. The other variables do not necessarily agree with these step demarcations, so they are not as consistent.

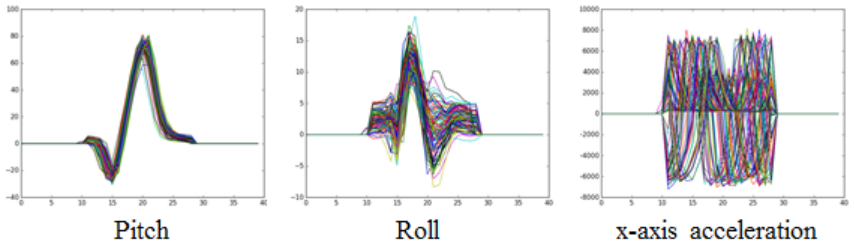


Figure 21: One hundred steps, isolated and overlaid by variable

Python Implementation: Since the algorithm in Python runs on a real-time datastream, it is desirable to improve on the efficiency of the step detection algorithms to ensure that the calculation lag is insignificant. First, note that the smoothing step uses a five-point moving average three times consecutively. For the Python implementation, instead of calling on a smoothing function three times, and then calling on a peak detection function, the data enters a series of buffers. There are three smoothing buffers and one peak/trough detection buffer. In parallel, the unaltered data is placed in a storage buffer.

At the beginning of every cycle of the algorithm, all the buffers are rolled, which advances the last data point from the end of the buffer to the beginning. This preserves the mean of each buffer yet simultaneously prepares the buffer to accept new data. Since the input of each buffer is the mean of all the data in the previous buffer, the buffers must be filled last to first. Consequently, the first position of the peak/trough detection buffer is overwritten with the mean of the third smoothing buffer, the first position of the third smoothing buffer is overwritten with the mean of the second smoothing buffer, and so on. This process is depicted in Figure 22. The storage buffer is only rolled with the other buffers and performs no other operations, allowing the unaltered datastream to be indexed and tracked with the rest of the algorithm.

After every buffer has been filled with new data, the algorithm compares the central value in the peak/trough detection buffer to its neighbors in order to determine if it qualifies as a peak or a trough. Whether a peak or a trough is used to separate steps is an optional parameter. If the central value is indeed a maximum or minimum, the data in the storage buffer will be copied from that point to the previous maximum or minimum, respectively, and that data, a "step", will be stored in a separate global list. The global list is a list of matrices, each matrix being a single step, and it is continually appended for other threads in the software stack to use.

Note that the moving average could be implemented as a simple moving average, taking only previous data points into account, or as a central moving average, which takes past and future data into account (with respect to the current data point). In other words, when the mean of five data points is computed, the type of moving average is determined simply by which data point that mean is indexed to. If the mean is assigned the same position in the datastream as the central data point, then it is a central moving average.

This is the approach taken in the current algorithm, with the overlap of all the averages resulting in a lag of only eight cycles of the algorithm. If a simple moving average had been implemented, the lag would have been fourteen cycles of the algorithm.

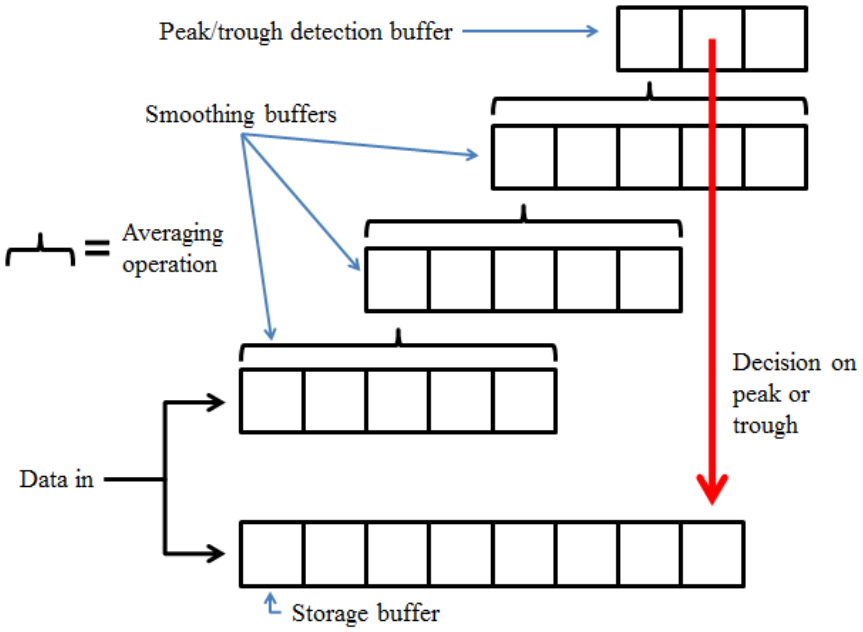


Figure 22: Illustration of the step detection algorithm in Python

Calculating Gait Parameters: Since smartSole is designed to be a complement to podiatrists, it has to be able to implement the parameters that these doctors look at. If the device is able to calculate these metrics, smartSole can prove to be extremely useful in helping determine a proper diagnosis.

Cadence: Cadence is a measurement of the number of steps someone takes per minute. To properly measure this, the device measures the time that elapses between one heel strike to the next heel strike. The amount of time that occurs during this measurement is the amount of time it takes to take a total of two steps. Since this only represents the number of steps taken in this short time interval, the system extrapolates the number of steps taken by the following equation.

$$\frac{\text{steps}}{\text{minute}} = \frac{2 * 60}{time} \tag{7}$$

Due to our peak detection method, the device is able to find the time difference between the two steps and apply it in this equation to find the number of steps taken in a minute. However it could be possible to get a better measurement if the time difference is taken between the first step to the third step, or the first step to the fourth step, or etc.

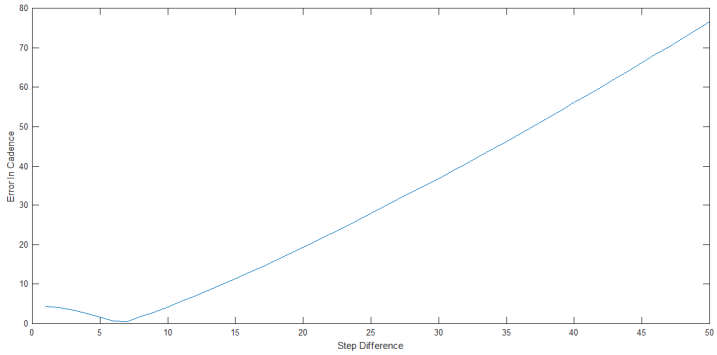


Figure 23: Error in Cadence vs. Different Steps

In order to determine how many step differences was optimal, this calculation was run over multiple iterations and the mean steps per minute were calculated over the dataset. The collected data was compared to the average healthy cadence of 107.5 and observed which was closest to this standard. The result of this told us that the optimal cadence is at a value of seven peak differences.

Stance Time: Stance time is a measurement of how long the foot is on the ground. The way this was calculated is by comparing the time when the heel first strikes the ground to when the toes leave the ground. The device will recognize this as the time difference between a relative maximum and the next relative minimum. The numbers associated with this time resulted in stance time accounting for approximately seventy percent of a step which is in line with research of sixty percent.

Swing Time: Swing time is a measurement of how long the foot is not on the ground. This is calculated by comparing the time when the toes leave the ground to the next time that the heel strikes the ground. So this calculation will be done by comparing the difference in time between a relative minimum to the next relative maximum. The numbers associated with this time resulted in swing time accounting for approximately thirty percent of a step which is in line with research of forty percent.

Velocity: Velocity actually turns out to be one of the most important metrics that podiatrists look at in determining healthy gait. Since the device contains a three-axis accelerometer, a simple integration over the acceleration should result in the velocity. However there are two main concerns observed during initial calculations.

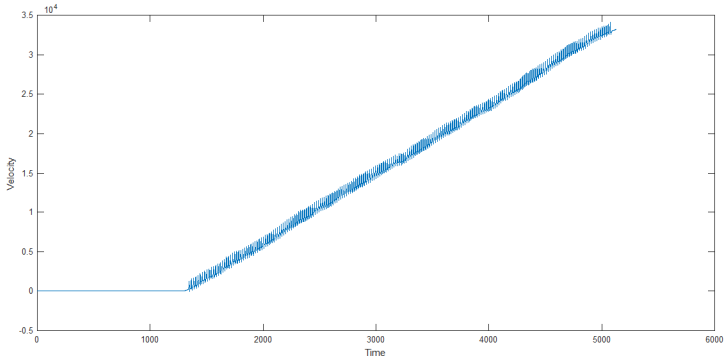


Figure 24: Demonstration of Velocity Error

The first issue is that the accelerometer is located in the foot. When measuring a person's velocity, doctors measure the entire body's velocity, not the foot's velocity. This results in points where the velocity would equal to zero even though the body's velocity is not.

The second issue is that the accelerometer accrues slight errors as it collects the data. This isn't as large of an issue if the device was focused on acceleration, but since the device continuously integrates upon this error, the error in velocity gets larger and larger. As a result, we achieve a velocity that increases without bound which renders the data useless.

In an effort to fix this issue, usage of stride length could help in achieving a calculation for velocity. Since a stride consists of two full steps, this knowledge can be used in accordance with cadence. Multiplying the stride length with the cadence would then result in a length per time, or the velocity. The only concern with this method is that this would require a previous measurement of stride length as we are unable to calculate that with our device.

Another possible method for finding velocity would to be to integrate over single steps. This way the error doesn't accrue from step to step and could possibly give us the velocity of the foot. As pointed out earlier, this is the velocity of the foot and not the body, but by integrating over this velocity, it is possible to get the distance traveled in that one step and generate a velocity from this and cadence. It would just have to be determined where the error in this method is still enough to cause a false reading.

Double Support: Double Support is the time difference between the heel strike of the current step and toe release of the previous step plus the time difference between the toe release of the current step and the heel strike of the next step. This metric can be calculated similarly to stance and swing time, but since the device is currently implemented on just one foot for now, smartSole is unable to get this metric just yet. However this will just simply be a calculation in time differences of relative maximums and minimums, which has been shown to be possible.

DTW Analysis With pressure data and all calculated variations on the motion data, there are 19 variables to consider in the data processing. For the sake of expediency, it is helpful to know which variables change the most when a user shifts from walking to shuffling. This can be found through dynamic time warping (DTW) analysis, in which two vectors of

data in time are compared and "warped" to each other. The DTW algorithm then outputs the abstract distance between the two vectors, which can be used to gauge their similarity.

This initial DTW analysis is performed on three datasets obtained from the first prototype. One dataset consists of normal walking (called 'Walking to RIMAC'), another of walking on a treadmill, and the last of shuffling on a treadmill. First the data is split into steps, and each step from each dataset is warped against steps from the same dataset, and from other datasets. The result for every variable is a matrix of histograms showing the distribution of warping distances, and the mean distances (Fig. 25).

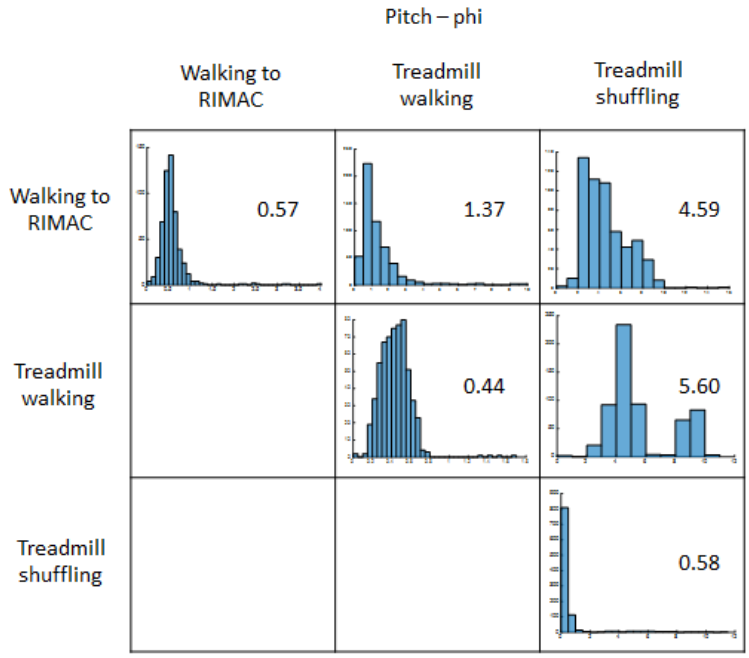


Figure 25: Histograms of warping distances on phi. The mean for each set of warping distances is shown as a number inside each box.

This analysis is done for eight of the variables in order to determine which are the most useful. Ultimately, the pitch angle phi and the x-axis acceleration differ the most between walking and shuffling, while differing the least between walking datasets. The pitch angle phi gives a warping distance of 4.59, while the x-axis acceleration yields a warping distance of 2.59. As a result, for machine learning algorithms that can only run on one or two variables, these are the optimal variables to use.

4.5.4 *Classification*

Two of the biggest workhorses in machine learning are hidden Markov models (HMM's) and neural networks. We did not implement HMM's since they require knowledge of either the discrete hidden states or the transition probabilities between states. Since our data does not have strict classification boundaries (i.e. there is no perfect walk or otherwise model walking impairment), there can be no delineation of discrete states, and so a HMM cannot be easily implemented. In contrast, neural networks do not require discrete states, and due

to the inherent nonlinearities, they can be used to distinguish between states with smooth, continuous transitions.

For ease of demonstration, however, this project only distinguishes between healthy walking and one gait abnormality: shuffling. The classification process began by collecting and labeling tens of thousands of data points, for both healthy walking and shuffling. Then, depending on the model being used, the data is either separated by steps (see Statistical Analysis above) or simply by a predetermined "chunk" size. That data is then fed into one of our two experimental models. The first model is simple logistic regression, used in an attempt to reduce computation time. The second model is a feed-forward multi-layer neural network (FNN), which operates on chunks of data and is more accurate than the logistic regression. In addition, currently only the FNN is integrated into the final software stack.

Logistic Regression: Multinomial logistic regression, also known as *softmax* regression, simply weights and biases the inputs to the model, then normalizes the output to produce a probability distribution. The softmax function is defined as

$$softmax(x)_i = \frac{exp(x_i)}{\sum_j exp(x_j)} \tag{8}$$

so that the inputs are exponentially weighted, and the denominator ensures the output sums to unity and hence is a probability distribution.

The abstract structure of the model, particularly the weights and biases, is shown in Figure 26. It is trained through the backpropagation algorithm using stochastic gradient descent. Despite its simplicity, the model performed well on initial data. The accuracies obtained are shown in Table 5. Current work is focusing on running the model on all variables simultaneously.

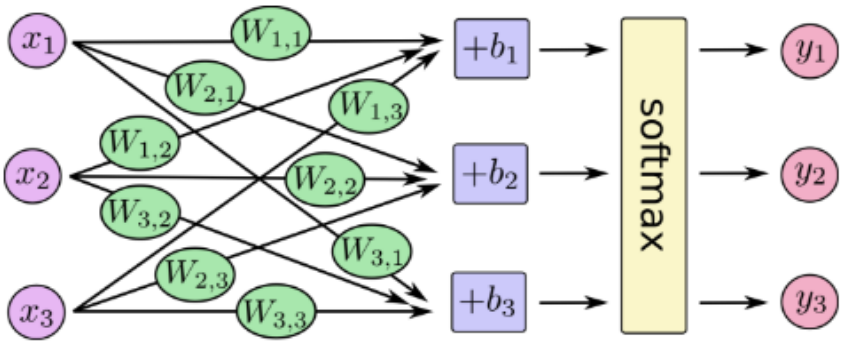


Figure 26: Example softmax network structure

Table 5: Accuracies for logistic regression model

Results:	Pitch	x-acceleration
Model walking	99%	98.5%
Real walking	91%	91.5%

Feed-Forward Neural Network:

Network Structure: The structure that was chosen was a one hidden layer feed-forward neural network model (FNN). The input and hidden layer dimension sizes are of variable size, such that it allows for programmatic tuning of these dimensions to allow searching for an optimum network configuration. With this implementation, the output structure is fixed to the number of classes, such that we are able to train each node to activate with a specific class.

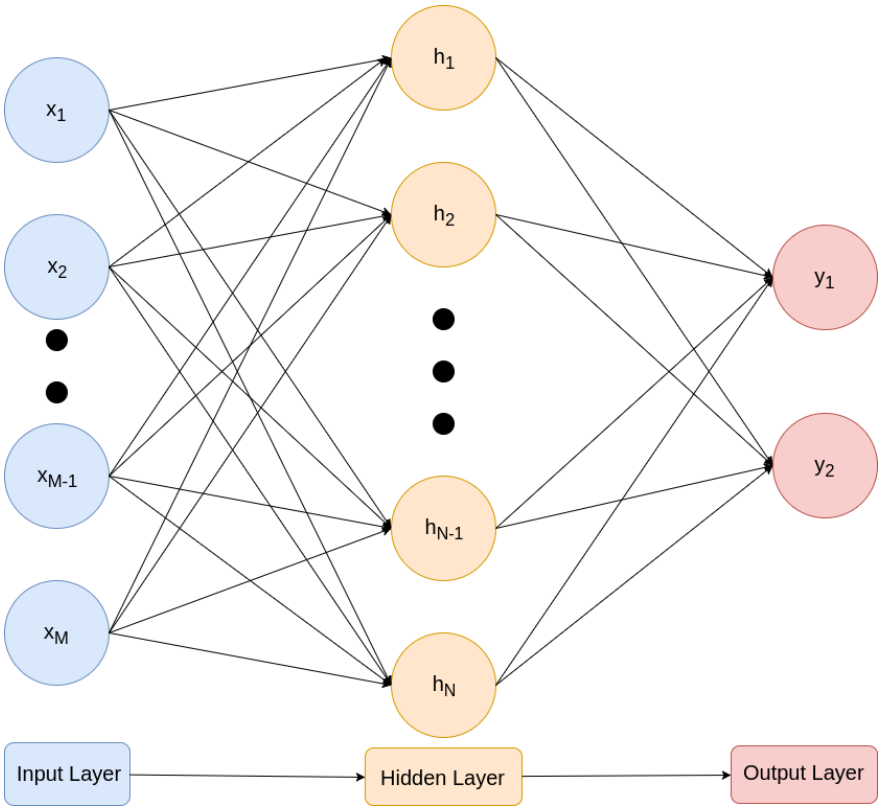


Figure 27: FNN structure with variable input and hidden layer dimension sizes

The equations describing the network from the inputs to the outputs:

$$\begin{aligned} z_1 &= \sum_{i=1}^{M} x_i \cdot W_i + b \\ h &= tanh(z_1) \\ z_2 &= h \cdot W_h + b_h \\ \hat{y} &= softmax(z_2) \end{aligned} \tag{9}$$

The linear estimator to the hidden layer relies on a weighted sum over the inputs. This is fed into a tanh non-linear activation function, which in turn is the input to another linear estimator to the output layer. The softmax function at the output converts the resulting logits into probabilities per class.

Loss Function: This model was trained using a cross-entropy loss:

$$L(y, \hat{y}) = -\frac{1}{T}\sum_{t=1}^{T}\sum_{c=1}^{C} y_{t,c} \cdot log(\hat{y}_{t,c}) \tag{10}$$

where T is the number of training examples, and C is the number of classes, which in our case is 2, walking versus shuffling. The cross-entropy loss punishes on wrong guesses logarithmically proportional to the confidence of the model in the wrong guess itself.

Training: Stochastic Gradient Descent was used to train the model, allowing for flexibility in batch size, learning rate, and max iterations. In order to calculate the gradients for updating each parameter, the back propagation algorithm was used. Each parameter's gradients would be calculated as follows:

$$\begin{aligned}
\frac{dL}{dW_h} &= \frac{dL}{d\hat{y}} \cdot \frac{d\hat{y}}{dz_2} \cdot \frac{dz_2}{dW_h} \\
\frac{dL}{db_h} &= \frac{dL}{d\hat{y}} \cdot \frac{d\hat{y}}{dz_2} \cdot \frac{dz_2}{db_h} \\
\frac{dL}{dW_i} &= \frac{dL}{d\hat{y}} \cdot \frac{d\hat{y}}{dz_2} \cdot \frac{dz_2}{dh} \cdot \frac{dh}{dz_1} \cdot \frac{dz_1}{dW_i} \\
\frac{dL}{db} &= \frac{dL}{d\hat{y}} \cdot \frac{d\hat{y}}{dz_2} \cdot \frac{dz_2}{dh} \cdot \frac{dh}{dz_1} \cdot \frac{dz_1}{db}
\end{aligned} \tag{11}$$

Data Preprocessing: The main preprocessing steps are summarized as follows:

1. Combine datasets
2. Isolate data sources selected as inputs

   When starting the algorithm, the user can specify the data sources to be used to train the model, such as combinations from Pitch, Yaw, Roll, Y-Acceleration, etc.
3. Reshape data into arrays of blocks (chunks)

   Each of these data sources begins as a 1-dimensional vector, it is necessary to reshape them a vector of chunks of specified width, chunk size, to feed into the network for classification. Because the resulting array is 2-dimensional, handling multiple inputs requires a 3-dimensional tensor structure to hold the data.
4. Generate one-hot labels

   Each of these chunks will require a one-hot label, defined as either the vector $[1, 0]$ for walking, or $[0, 1]$ for shuffling.
5. Shuffle data

It is necessary to shuffle the data before the next preprocessing step, such as to include examples from each of the combined datasets in the training/validation sets. The labels are appended to the end of the chunks, such that the shuffling process preserves the labels.

6. Split into training and validation sets

   The data needs to be split in favor of a larger training than validation set, such that the networks accuracy can be tested on data it has not seen beforehand.

Model Implementation For ease of implementation, the model was built and trained using Google's TensorFlow. It was structured such that it could be iterated on over various vectors of parameters, and would save per iteration statistics and models.

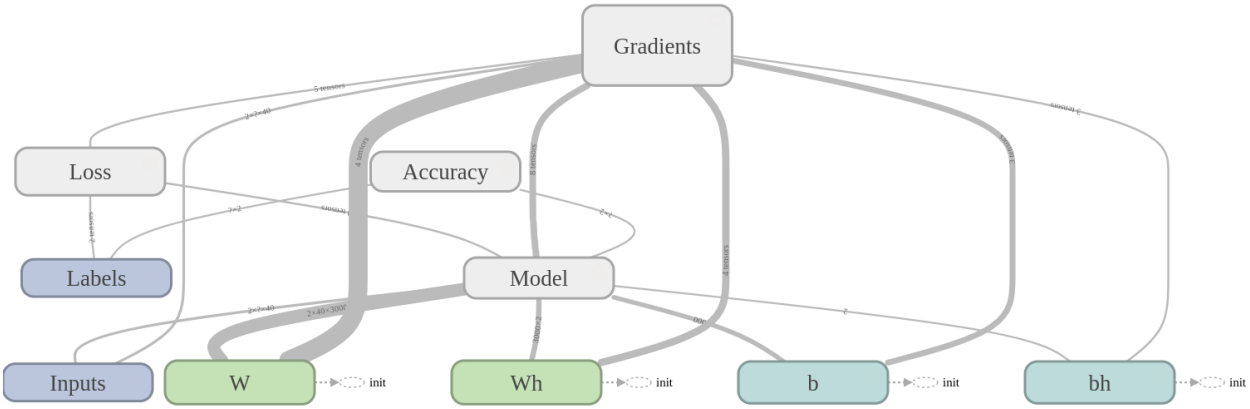


Figure 28: FNN graphical model, using TensorFlow's TensorBoard

Search for an Optimal Configuration With building an adaptable model, parametric grid sweeps can be preformed to find an optimal configuration for the model. These parameters include the inputs selected, hidden layer dimension size, chunk size, batch size, and learning rate.

1. Input Selection

   To avoid exhaustive sweeps over 15+ data sources, intuition was used to narrow them down to a few for analysis. Pitch, all three acceleration axes, as well as the pressure data sources were considered. Among them, there were only two that gave high accuracy, Pitch, and X-Acceleration.

2. Hidden Layer Dimension Size, Chunk Size, Batch Size

With less physical intuition to base off of, to find the optimal values for these parameters, 12 by 12 grid sweeps were preformed, recording accuracy on the validation set.

The first of which was a sweep of hidden layer dimension size versus the chunk size. A higher hidden dimension size increases the complexity of the model, and increases the computation time of the network; however, having this parameter being larger, also dramatically increases the accuracy on our validation set. Also, having a larger chunk size means a longer real-time computation, as the network would have to wait for that chunk of data before processing. Batch size is an important factor in finding a global minimum in the loss function, as using a lower batch size will reduce the chance of the algorithm getting stuck in undesired local minima. A higher batch size will have more stable gradients, but takes more time to optimize.

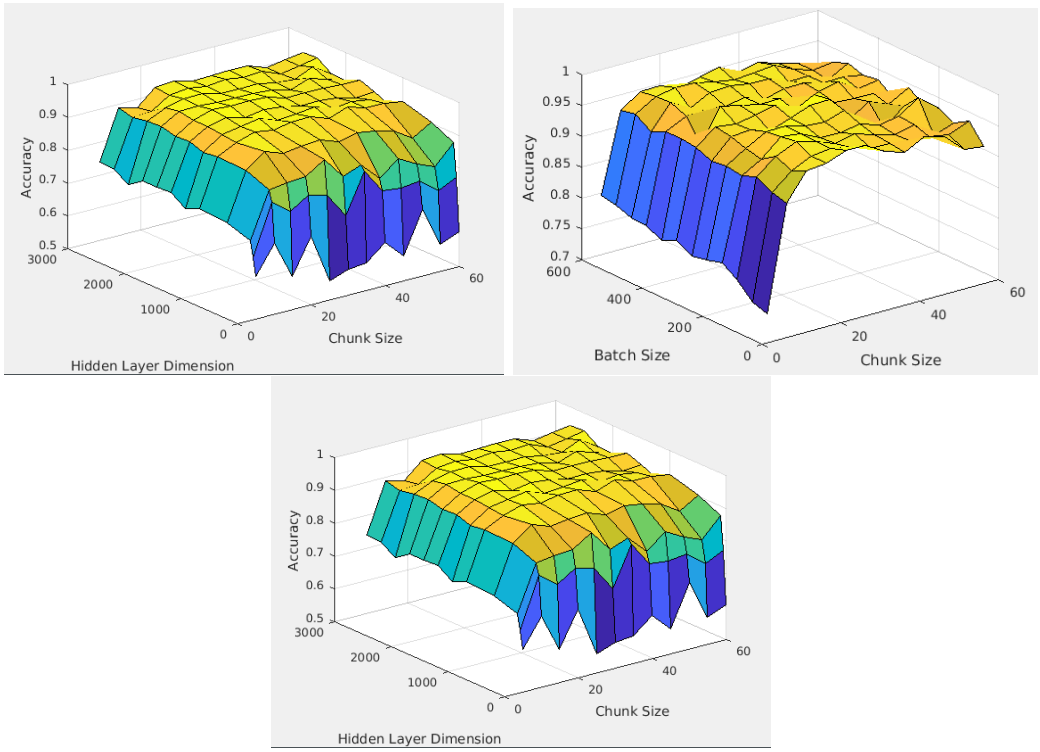


Figure 29: 12x12 Sweeps

3. Learning Rate

   Learning rate is the proportionality of which to adjust the weights by the gradients. Too small of a learning rate will result in a large convergence time, but a large learning rate can put too much trust in a step in the wrong dimension.

4. Max Iterations

   It is important not to overfit the model onto the data, otherwise the performance on new data will be lower.

High accuracies, 90-93 percent, were found with many combinations of these parameters, but the most consistent higher accuracies, 93-96 percent, were found with the following parameters.

Optimal Configuration Summary:

Inputs Selected:	$[Pitch, X - Acceleration]$
Hidden Layer Dimension:	2000
Chunk Size:	20
Batch Size:	200
Learning Rate:	0.001

Results Summary:

Training Examples:	3487
Validation Examples:	1494
Accuracy:	0.96185
Cross Entropy Loss:	0.35213

(12)

5 CONCLUSION

The smartSole project demonstrates that integrated insole hardware and advanced gait analysis software are both feasible and valuable. The largest challenges have been met with well-aimed design, and the fundamental concept has been proven. With future development steps planned out in detail, smartSole data collection and analysis may become another tool for evaluating user gait profiles in detail.

The comfortable prototype proves a more discrete and natural gait sensing device is available in comparison to stationary lab measurement tools. With complete integration and wireless charging on the horizon, smartSole is truly out of sight and out of mind solution.

Communications and data analysis give the smartSole project capabilities beyond simply measuring raw data, but actually interpreting it, offering a level of analysis that does not rely on doctors. Networking and data aggregation turn smartSole from a device to high-level system with a scope far beyond a single user.

Proving valuable in several aspects of gait analysis, the smartSole project outlines an innovation in gait diagnosis technology.

6 APPENDIX

6.1 Abbreviations

BLE	Bluetooth Low-Energy
CSV	Comma separated value
DMP	Digital Motion Processor
FNN	Feed-forward Neural Network
FPU	Floating point unit
GATT	General Attribute
HAL	Hardware Abstraction Layer
IMU	Inertial Measurement Unit
LEUART	Low energy UART
LiPo	Lithium polymer (battery)
MCU	Microcontroller
QSPI	Quad-SPI
Rx	Receiver
SoC	System on a chip
STM	ST Microelectrics
Tx	Transmitter

REFERENCES

[1] Rubenstein *et al.*, "Falls in the nursing home," *Ann Intern Med.*, vol. 121, no. 6, 1994.

[2] Kearney *et al.*, "The relationship between executive function and falls and gait abnormalities in older adults: A systematic review," *Dementia and Geriatric Cognitive Disorders*, vol. 36, no. 1-2, 2013.

[3] L. Rubenstein and K. Josephson, "Falls and their prevention in elderly people: What does the evidence show?" *Med Clin N Am*, vol. 90, 2006.

[4] Montero-Odasso *et al.*, "Gait velocity in senior people: An easy test for detecting mobility impairment in community elderly," *The Journal of Nutrition, Health and Aging*, vol. 8, no. 5, 2004.

[5] A. G. Society, B. G. Society, and A. A. of Orthopaedic Surgeons, "Guideline for the prevention of falls in older persons," *Journal of the American Geriatrics Society*, vol. 49, no. 5, 2001.

[6] Morris *et al.*, "Striding out with parkinson disease: Evidence-based physical therapy for gait disorders," *Physical Therapy*, vol. 90, no. 2, 2010.

[7] Lopopolo *et al.*, "Effect of therapeutic exercise on gait speed in community-dwelling elderly people: A meta-analysis," *Physical Therapy*, vol. 86, no. 4, 2006.

[8] Verghese *et al.*, "Epidemiology of gait disorders in community-residing older adults," *J Am Geriatr Soc*, vol. 54, no. 2, 2006.

[9] ——, "Quantitative gait dysfunction and risk of cognitive decline and dementia," *J Neurol Neurosurg Psychiatry*, vol. 78, no. 9, 2007.

[10] ——, "Gait dysfunction in mild cognitive impairment syndromes," *J Am Geriatr Soc*, vol. 56, no. 7, 2008.