A Low-Power Wearable Device for Smart Step Counting

Zachary Murtishi, University of Rhode Island

Abstract

A prior effort resulted in a smart step counting system combining an IMU-equipped hardware platform with Bluetooth communication capabilities to communicate with a machine learning model in order to perform motion classification. [1] However, the resulting hardware system required a power-hungry computing platform that was far more complex than required for this specific application. A new hardware solution was developed to replace the original system with significantly reduced power consumption and design complexity while retaining much of the original functionality. The resulting hardware system is able to successfully interface with a specially-trained motion classifier over a Bluetooth Low Energy interface to determine user motion patterns.

Introduction

Smart motion classification refers to a technique in which the motion patterns of a user are captured and identified using machine learning in lieu of a traditional deterministic algorithm. A classifier constructed using machine learning techniques is trained to recognize specific combinations of time series data as certain motion types, including but not limited to an idle user, a user in a walking motion, and motion unrelated to user activity (a pattern that will be referred to as “shaking”). A classifier trained in this manner is fed input data collected through a wearable device containing a gyroscope or accelerometer to identify the wearer’s motion patterns.

A prior effort succeeded in developing a smart motion classification solution using a battery-powered Raspberry Pi and integrated sensor suite to implement the required communication and processing elements. [1] The processing power of the Raspberry Pi allowed the system to perform client-side execution of a motion classifier in addition to data collection functions and wireless communications. The result was a wearable device with the ability to be used as a standalone system running both data classification in addition to a tethered mode in which the device was reduced to a data streaming system and the CPU-intensive classifier was offloaded to a system with greater processing power. However, the high power consumption of the Raspberry Pi coupled with the inefficiency of its software algorithms did not result in a wearable device with useful battery life and so a solution that implemented the core functionality of the system in a low-power package is required.

Due to the high computational power and capabilities required to run the Keras-implemented classifier, it was determined that a low-power implementation of the step counter should not be required to perform client-side classification. Instead, the new step counter was defined to have two core functions: 1) the device should be able to communicate with an inertial measurement unit (IMU) to collect the relevant data required for motion classification and 2) the device should be capable of wirelessly communicating with a host platform on which the classifier is run. By simplifying the requirements and assuming the user

is always using it as a tethered device, the step counter can be made much less complex and power-efficient to maintain the same functionality as a tethered Raspberry Pi-based step counter with significant improvements in battery life.

Design and implementation

Changes from previous design

The prior step counter design was designed around a Raspberry Pi 4 board. A Pi Sense Hat from element14 was connected to the Raspberry Pi to provide the sensor suite required for motion classification and a 10000 mAh USB power bank was used to allow this system to be used as a mobile device. As the Raspberry Pi 4 is designed to be used with an operating system, all software required for its implementation was developed using Python 3.8, including the on-board classifier, Bluetooth Classic interface (BlueZ), and Sense Hat API. A host computer is capable of communicating with this system over a Bluetooth Classic interface to receive motion data and return motion classification results. The system as assembled is shown in Figure 1.

Figure 1: The Raspberry Pi-based system is shown, with the Sense Hat, Raspberry Pi 4 board, and portable battery pack visible.

The prior design was not optimized for mobility, as the Raspberry Pi at idle demands a hefty 540 mA, which can increase to over 1 A as CPU load increases. While this power consumption is low for a general-purpose computer, a constant 540 mA current draw will deplete a small battery in little time. The Raspberry Pi is not particularly well-suited for mobile applications, as the high-performance processor and CPU load of the operating system lead to significant power consumption that is not required for the core functionalities including sensor polling and wireless communication. Forcing the use of power input through USB is also a drawback, as it requires the use of manufactured USB power banks to make the device portable instead of generic, rechargeable batteries.

The new step counter was designed to be a wearable device from the start and several design decisions were made to support mobility and battery life. The Raspberry Pi has been removed from the design and completely replaced with a custom printed circuit board (“main board”) with a PIC18LF2420 microcontroller performing all processing functions. The Pi Sense Hat is once again included in the design to allow the PIC18 microcontroller to interface with the on-board IMU and interfaces with the main board through an I2C interface. An HM-10 Bluetooth Low Energy (Bluetooth LE) module is connected to the main board microcontroller over a serial interface to add wireless communications to the system. Additionally, the main board is equipped with an on-board 18650 lithium-ion battery with an included battery charging circuit and simple circuit to monitor battery life. The main board is then equipped with its two peripheral boards (HM-10 and Sense Hat) and affixed to a wearable pouch to be mounted on the user’s calf. The complete assembled device is shown in Figure 2.

Figure 2: The new assembly is shown, with the HM-10 (blue board, right), main board (center), and Sense Hat (green board with LED matrix) visible.

The current board is a Revision 1 example; a Revision 0 exists, but is unusable due to an incorrectly-spaced 28-DIP footprint for the PIC18 microcontroller.

Processor and peripherals

The PIC18LF2420 was selected as the centerpiece of the system due to its low cost, low power consumption, low-voltage operation, and easy access to development software and tools. The PIC18F2420 is equipped with several peripherals, including hardware timers, analog-to-digital converters, hardware USART and high-speed serial modules, and external interrupt pins to allow it to interface with various other hardware modules and devices both on-board and off-board. Several operating modes are available to the programmer that allow it to be used in either active or low-power modes. As configured in this system, the PIC18 is run from a 4 MHz quartz oscillator, which allows for precise timing for the

on-chip timer modules and asynchronous serial interface compared to the internal RC oscillator. 4 MHz was determined to be an ideal clock speed as it is fast enough to ensure low-latency communication with the host while remaining far below the maximum clock speed (32 MHz) to cut power consumption. A schematic of the PIC18 microcontroller is shown in Figure 3.

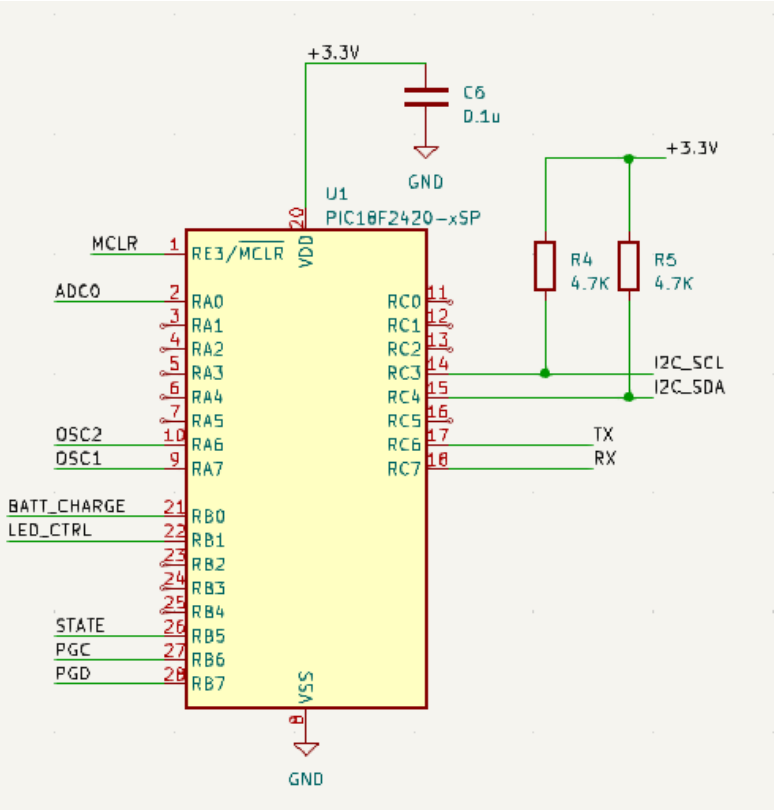

Figure 3: A schematic of the PIC18LF2420 used in the step counter design, with peripherals, oscillator, digital input and output pins, I2C interface, serial port, programming interface, 3.3V power input with bypass capacitor, and analog-to-digital interface shown.

Software architecture

The PIC18LF2420 microcontroller was programmed entirely in C, with the exception of an assembly instruction required to place the microcontroller into sleep mode. As it does not run an operating system and is unable to do so due to its low processing power and lack of a memory management unit, the microcontroller is programmed as a bare-metal device. Instead of the multi-threaded software architecture used on the Raspberry Pi-based device, the new software architecture makes use of an interrupt-driven program to speed up responses to host inquiries and remove the blocking I/O functions present in the Python implementation that would otherwise hold up response time. The microcontroller runs in an infinite loop in between servicing interrupts. This interrupt-driven program architecture allows the microcontroller to perform multiple functions upon request and makes up for the lack of a true multitasking solution.

As designed, the system is intended to initialize all of its hardware modules and enter a sleep mode upon power-on. This sleep mode shuts off both the microcontroller's processor and peripherals to achieve

maximum power savings and the only way to exit this sleep mode is for an interrupt to be generated. The PIC18 is to remain in sleep mode until the HM-10 module has established a connection to a host machine; at that point, a digital I/O pin on the HM-10 changes state and activates an external pin interrupt to wake the PIC18 microcontroller from sleep mode and execute its program. This is in order to save power while the board is in an idle state, as there is no need for any hardware other than the HM-10 module to remain active until a connection has been established. This process is described in Figure 4 below.

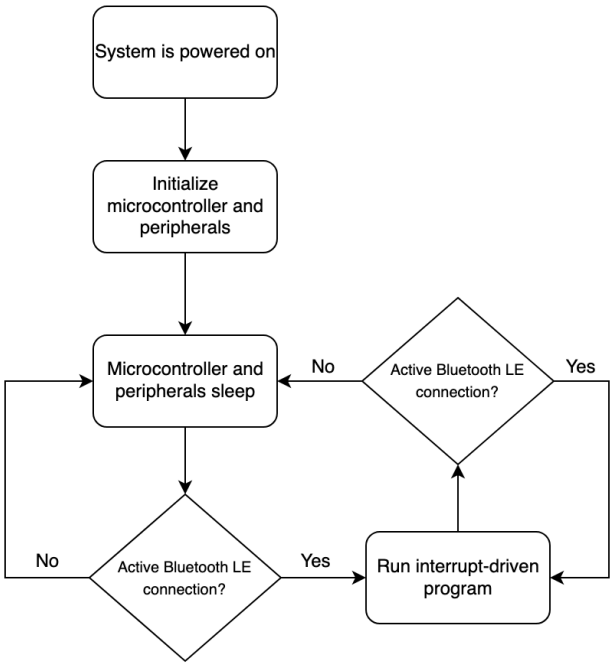

Figure 4: The main state machine of the PIC18LF2420 microcontroller in the step counter application is shown

The system is intended to be polled for information from the host over a Bluetooth LE connection. When requested, the system will poll the Sense Hat's LSM9DS1 IMU and send the current contents of the gyroscope or accelerometer output registers to the host, with each axis value represented as a 16-bit value in two's complement format. These requests are limited to a single byte, reducing the amount of data to be read. The values of the X, Y, and Z axises will all be included for either a gyroscope or accelerometer request, resulting in 48-bit (or 6 byte) responses. Configuration options, such as an EEPROM refresh or DC offset calculation may also be performed over the wireless interface.

Wireless interface

As the PIC18 microcontroller does not have an included wireless interface, this functionality is included through the use of an external hardware module. The HM-10 module is a complete circuit board that makes use of the Texas Instruments CC2541 system-on-chip to serve as a full Bluetooth LE transceiver. The HM-10 is interfaced to the PIC18 microcontroller through a TTL-level asynchronous serial interface in which the HM-10 interacts with the PIC18's hardware UART module to exchange information over the wireless interface. This serial interface is a standard 8-bit, no parity, 1 stop bit serial (8-N-1) scheme run

at 19200 baud, which requires reconfiguration of the HM-10 from its default setting of a 9600 baud interface. This configuration of the HM-10 was performed by the developer prior to installation through the use of a USB-to-Serial converter implemented with an Arduino Uno Rev3 board. It is recommended that this configuration is performed prior to runtime to ensure that the proper AT commands are used to configure the device.

Bluetooth Low Energy was selected over Bluetooth Classic for this specific application as it better suits the operating requirements. Bluetooth LE offers significant power savings over its classic counterpart at the expense of transmission speed and range. While Bluetooth Classic is well-suited for applications that require constant, high-speed data streams, Bluetooth LE is better suited for this specific application with power constraints and short distance, infrequent, and low-throughput data exchange. It is also the case that many software libraries supporting the Bluetooth Classic protocol are becoming deprecated over time as more devices adopt the newer Bluetooth LE standard, as seen with iOS dropping support for Bluetooth Classic and the Python RFCOMM library Pybluez becoming unmaintained.

As the host sends data to the HM-10, it is converted to serial bytes that are sent to the PIC18's serial port. While the HM-10 is active, the PIC18 microcontroller waits for a command byte at its asynchronous serial port. Upon arrival, a received byte will trigger an interrupt in which the PIC18 will exit its wait state and enter an interrupt service routine. In this interrupt service routine, the byte is then decoded by the PIC18, which then performs a command corresponding to the command set. This command can be a request for data in which data is sent in response to the original command, or a command that results in an on-board action in which there is no expected response. After executing the command, the PIC18 returns to a waiting state in which it awaits more data to execute a command. This process is described by the flow chart in Figure 5.

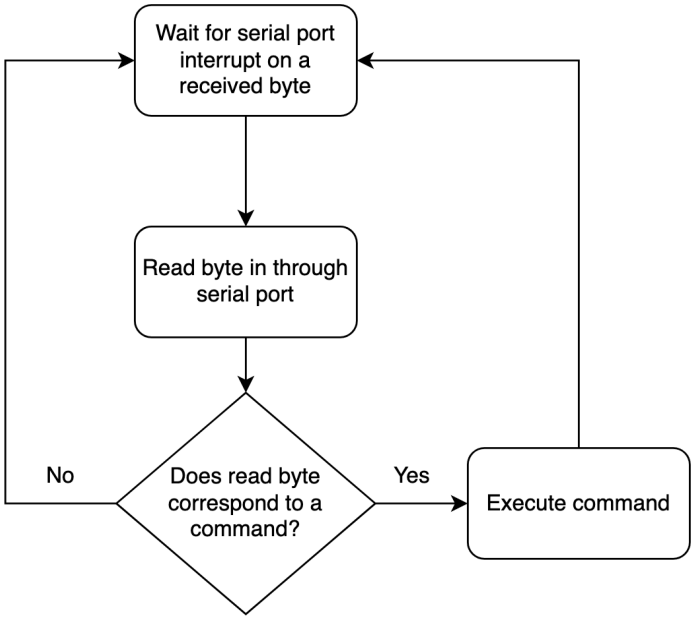

Figure 5: The serial port state machine used by the PIC18LF2420 to process data received through the HM-10 Bluetooth LE *module*

A host program running on a laptop computer is used to poll the HM-10 for data. This program is written in Python 3.10 and uses the Bleak library to implement Bluetooth LE communications on the host side. Communication between the host and step counter is performed through modification of Generic Attribute Profiles, or GATT. While some devices will contain specific GATT fields for different measurements, the HM-10 contains only a single GATT. For example, the host will write a character to the HM-10's general-purpose GATT, which will then send that character to the PIC18 microcontroller over a serial interface. The microcontroller will then respond with serial communications to modify that GATT and the response will be sent to the host. A list of valid commands that can be sent to the step counter is shown in Figure 6.

Command byte (decimal)	Action
0	Send 10-bit value of current battery level
10	Send 6 bytes of gyroscope data (X, Y, Z axises) with DC offset subtracted
11	Calculate and set gyroscope DC offsets
12	Send 6 bytes of gyroscope data (X, Y, Z axises) without DC offset subtracted
20	Send 6 bytes of accelerometer data (X, Y, Z axises) with DC offset subtracted
21	Calculate and set accelerometer DC offsets
22	Send 6 bytes of accelerometer data (X, Y, Z axises) without DC offset subtracted
30	Refresh EEPROM contents
40	Turn on low battery indicator LED (test)
41	Turn off low battery indicator LED (test)

Figure 6: A list of valid command bytes recognized by the step counter.

Although the Bluetooth LE specifications define the link layer bit rate to be 1 Mbps, a quick test was performed to determine the throughput of the Bluetooth LE connection between the HM-10 and the host; in this case, a normal connection in which data was continually exchanged between the host and HM-10 was captured by Wireshark; every 100ms, the HM-10 would receive a single byte and the HM-10 would send six bytes. This behavior was used as it is typical of normal operation. Per Figure 7, throughput was determined to be about 2,656 bps.

Capture

Hardware:	Intel(R) Core(TM) i5-6300U CPU @ 2.40GHz (with SSE4.2)
OS:	Linux 5.15.0-52-generic
Application:	Dumpcap (Wireshark) 3.6.2 (Git v3.6.2 packaged as 3.6.2-2)

Interfaces

Interface	Dropped packets	Capture filter	Link type	Packet size limit (snaplen)
bluetooth0	0 (0.0%)	none	Bluetooth H4 with linux header	262144 bytes

Statistics

Measurement	Captured	Displayed	Marked
Packets	6093	6093 (100.0%)	—
Time span, s	204.904	204.904	—
Average pps	29.7	29.7	—
Average packet size, B	11	11	—
Bytes	68035	68035 (100.0%)	0
Average bytes/s	332	332	—
Average bits/s	2,656	2,656	—

Figure 7: A Wireshark printout with statistics describing a Bluetooth Low Energy connection between the step counter and host.

Sense Hat interface and IMU configuration

In order to preserve the functionality of the Raspberry Pi-based system, the new system makes use of the same Sense Hat as its sensor suite. The Sense Hat is an expansion board intended for use with a Raspberry Pi in order to provide developers access to a set of sensors for use in embedded projects. It is intended to communicate with a microprocessor over an I2C interface. As in the prior system, an I2C interface implemented by the PIC18's hardware synchronous serial port is used to communicate with the Sense Hat and replace the Raspberry Pi's I2C interface in the loop. The LSM9DS1 inertial measurement unit remains the only part of the board polled during runtime, although the ATTiny85 microcontroller controlling the LED matrix receives a set of commands to disable the LED matrix on startup to cut power consumption.

The PIC18 microcontroller communicates with the Sense Hat at the full speed I2C clock rate of 100 kHz, which is compliant with both the I2C routine running on the ATTiny85 microcontroller and the LSM9DS1 IMU. Communication is byte-wise, with the microcontroller requesting data from the LSM9DS1 data registers one byte at a time. Before each data request, the microcontroller confirms that new data is available through reading the LSM9DS1's status register. Once this is confirmed, the microcontroller performs double I2C reads (a read ended by a positive acknowledgement followed by a read with a negative acknowledgement) for the register requested in order to retrieve both the lower and upper bytes.

On startup, the PIC18LF2420 uses the I2C interface to initialize and configure the LSM9DS1. Some of these settings are shown in Figure 8 below. Note that the settings below only show a small subset of the device's configurable settings.

Gyroscope output data rate	119 Hz
Accelerometer output data rate	119 Hz

Auto increment address on read	Enabled
FIFO memory	Disabled
Gyroscope high pass filter cutoff	0.01 Hz
Low power mode	Disabled
Gyroscope scale	500 dps
Accelerometer scale	2 g

Figure 8: Some configuration settings sent to the LSM9DS1 upon startup of the PIC18LF2420 microcontroller.

As configured in this system, only the accelerometer and gyroscope may be polled for data. No function on the microcontroller or host program exists to poll data from the magnetometer and this specific feature of the LSM9DS1 is left powered down in order to reduce power consumption.

Power consumption

The board's components are largely designed to run off of 3.3VDC in order to reduce power consumption and achieve compatibility with the Sense Hat interface and HM-10, both of which communicate at 3.3V voltage levels. It is also much easier to generate a 3.3V power rail from lithium-ion cells, which typically feature a nominal voltage of 3.6V. The battery's output is reduced to 3.3 V through the use of a Microchip Technology MIC5504 low-dropout voltage regulator which accepts the 3.6-4.2V output from the battery and outputs a constant 3.3V. However, the HM-10 module's power rail receives the raw battery voltage as an input to its power rail due to it containing an onboard DC-DC converter.

Per Figure 9, the system consumes 26.3 mA of current while the PIC18 microcontroller is in sleep mode; while the PIC18 is in active mode, the total power consumption climbs to 27.3 mA. This data was collected through the use of a multimeter on the battery's positive terminal to measure current flow. Note that the Sense Hat and HM-10 both consume power even while the system is in sleep mode. About 9mA of this current is from the green LED, which is connected to the 3.3V rail through a 150 ohm resistor. This results in a runtime of 128.2 hours in active mode and 133.1 hours in sleep mode, using the full rated capacity of 3500 mAh for the 18650 cell.

System current consumption, idle	26.3 mA
System current consumption, active	27.3 mA

Figure 9: Current consumption of the hardware system for both sleep and active mode.

Battery information

The system is powered from a single 18650 lithium-ion cell; the specific cell is a Molicel M35A cell with a maximum capacity of 3500 mAh and integrated protection circuitry to prevent both overcharge and over-discharge conditions from damaging the battery. In lieu of a traditional battery management system or gas gauge IC, a simple power-saving circuit is used to poll the voltage of the battery. As the board's

circuitry places a light load on the cell, the measured voltage is very similar to the battery's true open-circuit voltage and the state of charge may be inferred from this value.

Battery monitoring

The circuit used to measure the battery terminal voltage is implemented through the use of a pair of transistors that control the current draw of a voltage divider connected to the battery terminal. An NMOS transistor is driven by the microcontroller, which in turn drives a PMOS transistor to energize a voltage divider connected to the analog-to-digital converter on the microcontroller. This is done in order to save power, as the voltage divider intended to normalize the voltage to a safe value for the microcontroller input pins draws a small current; the transistor pair makes it so that this voltage divider is only active when a measurement is to be taken. The voltage value is measured by the on-board analog-to-digital converter with a 10-bit resolution. This value is transmitted in its raw form to the host platform, which uses the voltage divider equation and ADC reference of 3.3V to convert it to the true battery voltage. It is understood that this is not the most reliable method for determining a battery's state of charge, but it is a simple one that does not require complex battery management hardware or algorithms to implement. The schematic for this circuit is shown in Figure 10.

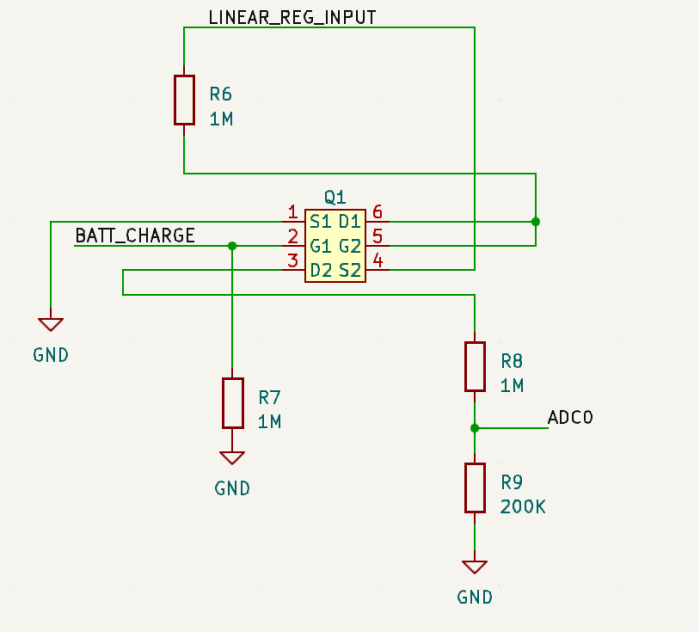

Figure 10: Schematic of the battery voltage monitoring circuit used by the step counter to determine the charge level of the onboard 18650 lithium-ion battery.

This circuit is used to sample the battery's voltage every one second and this is implemented through the use of a hardware timer incrementing an overflow interrupt-serviced counter until one second has passed. If this voltage drops below 3.4V, a yellow LED is lit to inform the user that the battery level is low. This threshold was selected due to the fact the MIC5504 linear regulator has a dropout voltage of about 100mV at a 50 mA current output and its 3.3V output rail becomes unstable after this point, which then causes errors in the ADC using it as a reference to read battery voltage. Although the system's total current draw is below 50 mA, this dropout voltage was selected for safety purposes.

Battery charging

The board contains an external interface in the form of a barrel jack which is intended to be connected to an off-board AC-DC converter in order to charge the on-board battery. A cheap AC-DC power supply with a 9VDC output was used for this purpose. This 9V input is reduced to 5V through a DC-DC converter, which is then used for two purposes: 1) as an input to the MCP73831 charge controller and 2) to remove the load on the lithium-ion cell by activating a single-pole double throw (SPDT) relay to divert all current draw from the battery to the off-board power converter to improve charge efficiency. The full battery and power circuit is shown in Figure 11 below.

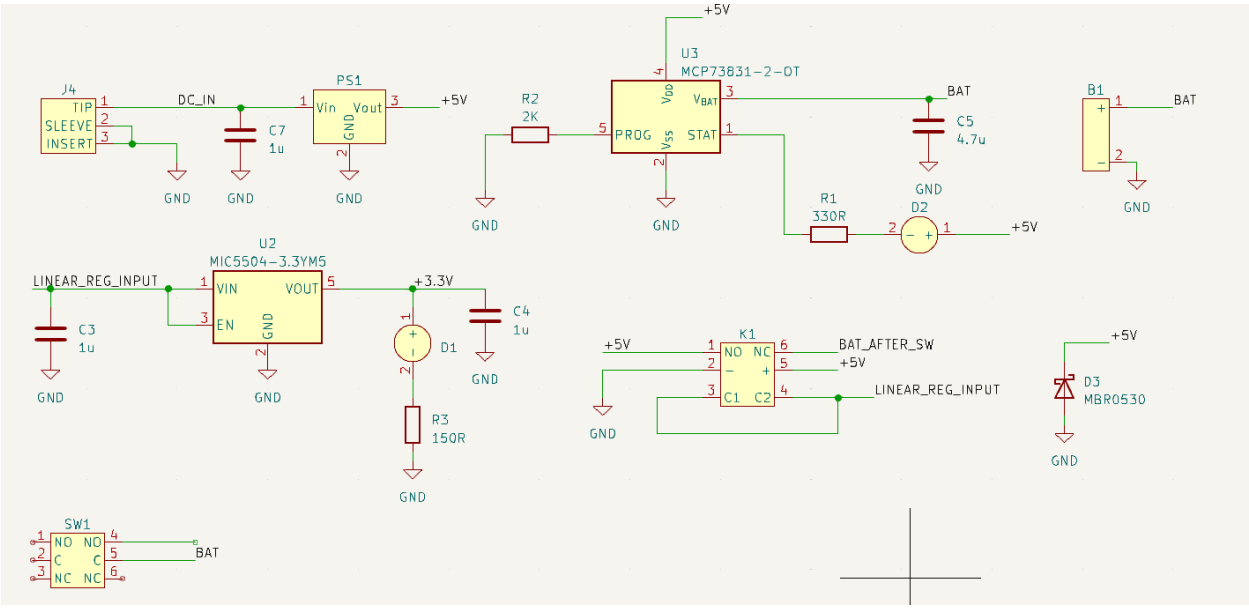

Figure 11: A full schematic of the step counter's power and battery-related circuitry.

The MCP73831 charge controller uses a constant-voltage, constant-current (CCCV) algorithm to charge the battery, which is standard practice in the safe charging of lithium-ion cells. This algorithm starts off by charging the battery with a constant charging current; once this current begins to decline to a set level, the battery is then charged with a constant voltage. Once the charge current declines to a specific value determined by the charger in question, the battery is considered to be at full charge. Charging voltage is pulled from the battery and the charge cycle is then complete.

A 2K ohm programming resistor is placed in order to charge the battery at the maximum charge current of 500 mA. This is a considerably lower charge current than some other commercially-available lithium-ion chargers, but this is due to the small form factor and lower complexity of the MCP73831. During the charge process, a red LED will activate as the STAT pin will go low and sink the LED current. The battery will charge until the battery voltage rises sufficiently that the charge current declines to a fraction of the programmed charge current. At the end of the battery's charge, this pin will go high and the LED will turn off to signal that the battery is charged.

DC offset calculation and corrections

There exists a small DC offset on the readings of the IMU despite the use of digital filtering on the LSM9DS1. The PIC18 microcontroller is programmed with a routine that attempts to subtract this offset from the data sent to the host. A routine exists for calculating the offsets in both the gyroscope and accelerometer; the offsets calculated during this routine are then used prior to transmission of the IMU data to the host to subtract this offset from the raw data.

The DC offset calculation takes 128 samples of each IMU axis to generate a mean value for each. Note that the system must remain at rest to calculate an accurate set of offsets. It was decided to use 128 sums for this purpose as 128 is a power of 2 and division may be performed by shifting the sum to the right by 7 bits in lieu of complex software division. The mean value calculated represents the average DC offset value in an idle state. This mean value requires careful calculation through a function created specifically to add 16-bit integers while accounting for two's complement. This final mean value is then written to nonvolatile memory in the form of the microcontroller's internal EEPROM module so that it persists after the system is powered down. On power-up, the microcontroller will retrieve the offsets for both the gyroscope and accelerometer from EEPROM and read them into its data memory for use during runtime. Figure 12 describes this DC offset calculation algorithm in a flow chart.

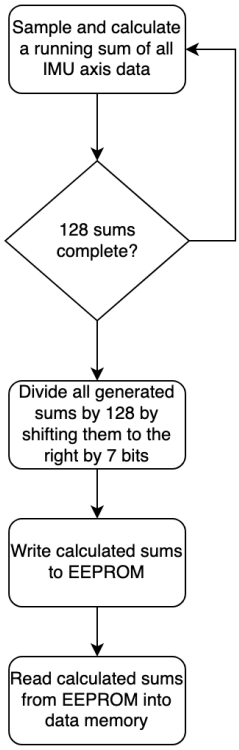

Figure 12: A flow chart of the algorithm used by the PIC18LF2420 microcontroller to determine the DC offsets of the LSM9DS1 output registers.

The microcontroller is also equipped with an EEPROM refresh function to allow a user to periodically re-write data so that it is preserved for a long period of time. A refresh function is required for values that are infrequently written to the EEPROM, as leakage currents and environmental effects may degrade the data in seldom-written memory cells. This refresh function simply reads all EEPROM data bytes in the

module from addresses 0 to 256 and re-writes them immediately to restore their value. This function can only be called by the host application.

Programming interface

A programming interface for the PIC18LF2420 microcontroller is included in the form of a 5-pin header with a 2.54 mm pitch. This programming header is connected directly to the microcontroller's programming interface intended to allow for the direct connection of an MPLAB X-compatible debugger to perform programming and debugging functions. During development, a PICkit 3 debugger was used to both program and perform debugging on the onboard microcontroller. As configured, the PICkit 2, PICkit 3, and PICkit 4 programmers should be supported by this interface.

Host interface and motion classifier

Host interface

The step counter is designed to interface with a host system running a software application to capture IMU data over Bluetooth Low Energy. The host system is an M2 Macbook Air running macOS 13.0, though the software has been tested and is compatible with Linux-based Intel systems with Bluetooth LE interfaces. All software on the host side is developed in Python 3.10 in order to make use of the Bleak library for Bluetooth LE communications and to leverage the existing classifier trained using Keras. The result is an extremely simple command-line interface that is largely automated and requires no user input during runtime; the program is designed to automatically connect to a Bluetooth LE device with a set name, in this case "STEPCOUNTER".

The host application is designed to read raw IMU data from the step counter over the Bluetooth LE interface and process it for entry into the neural network-based motion classifier. The processing begins when the data is first read from the Bluetooth LE interface; the data is first normalized to a floating-point normalized to 1 by dividing the captured data by 2^15. A scaling factor is then applied to the data through a dot product; this scaling factor corresponds to the scale values described in Figure 8. This data is then stored in a Python list.

Requests to the step counter for IMU data are made by the host program every 100ms and they are timed such that a request and reply will take exactly 100 milliseconds. The classifier is designed to accept a 1-D timeseries array consisting of 1 second of motion. In order to satisfy this requirement, the classifier performs ten of these 100 millisecond requests and inputs the data to the classifier upon receipt of the tenth data request. After this result is obtained, the counter is cleared and the process repeats for another second. This algorithm is described in Figure 13. A more simple zero-counting algorithm is used to count steps performed during this time by running the z-axis array through it to determine the number of steps. After each classifier result is obtained, it is displayed to the user, along with the total number of steps counted during the application's runtime.

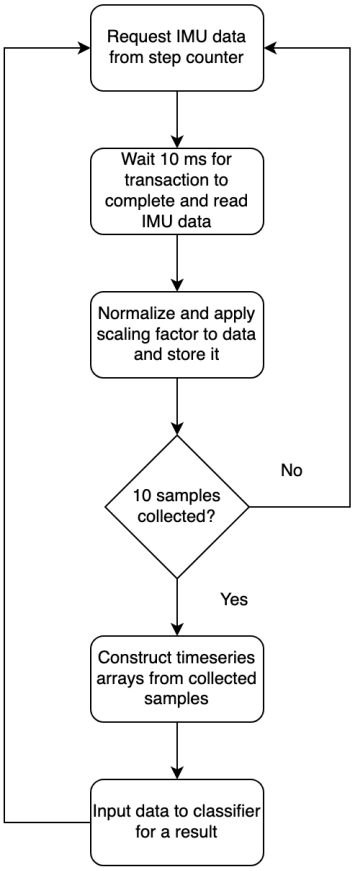

Figure 13: A flowchart describing the host application's step counter polling algorithm.

Motion classifier

The motion classifier is at the center of the host application as it is responsible for making sense of the recorded IMU data. The motion classifier is implemented as a set of 1-D convolutional neural networks trained using data recorded from user motion, as well as data generated from psuedorandom sources. As previously discussed, it is trained to operate on one-second windows of timeseries data to determine user motion. Figure 14 shows an example of the timeseries data input to the classifier in the form of a training data snippet. It is able to inform the user of three different types of motion that may have occurred in the last one second of time: an idle user, active walking motion, or shaking motion that is inconsistent with the other options. This model was trained using the Keras interface for the TensorFlow software library.

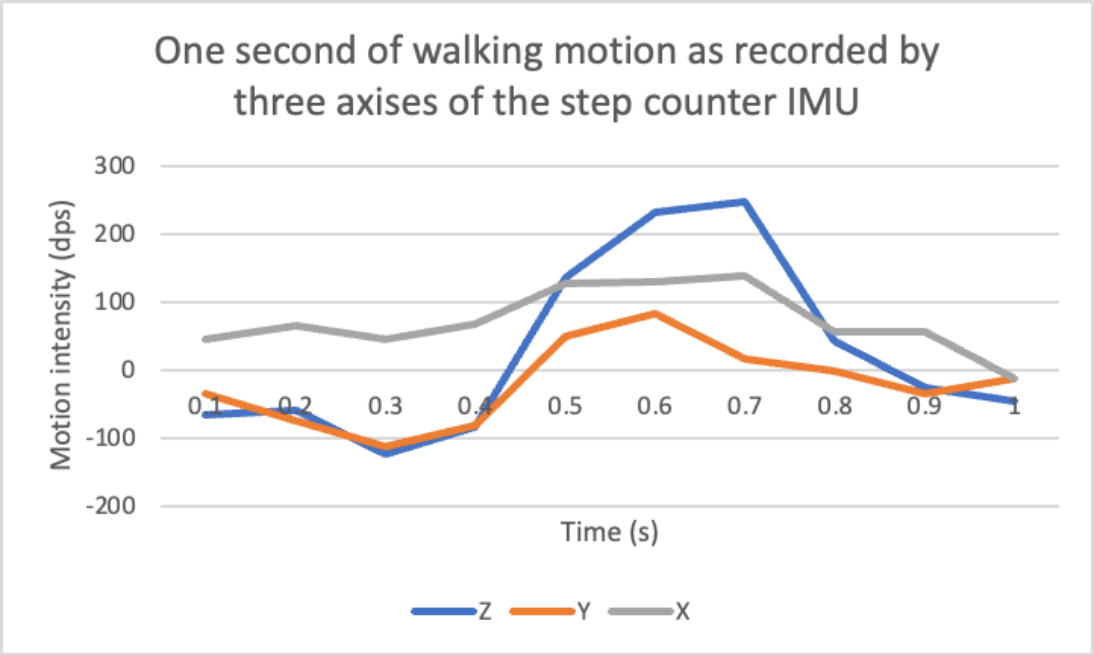

Figure 14: A one-second window of user motion as recorded by the step counter's IMU, which approximates the input data from the step counter to the classifier. This data is a snippet of the training data used to train the motion classifier model.

The motion classifier was trained using three data sets: a dataset recorded while a user wore the step counter while not in motion, a dataset recorded while a user was actively walking, and a dataset generated using the Python psuedorandom number generator module. The first dataset was intended to train the classifier's ability to detect an idle user, the second was to train the classifier's ability to track walking motion, and the last was intended to train the classifier to detect the user shaking or engaging in pseudo-motion that should not be considered true motion. Each axis was used to train a distinct classifier model, all of which are identical in structure: two 1-D convolutional layers, followed by three hidden layers using the ReLu activation function, with a softmax-activated output layer.

The motion classifier performs ensembling in that results from its multiple convolutional neural networks are combined and used to achieve a final result. In this case, only the gyroscope's X and Z axises are used for the classifier and are assigned a weight of 0.3 and 0.7 respectively (the Y axis is therefore assigned a weight of 0). These weights are applied to the softmax outputs of each classifier before they are summed for a final result. Figure 15 is a screenshot of the host application that shows the outputs of each model during the host application's execution, along with the final output after ensembling. As determined during the prior effort, the gyroscope's data was determined to be the most useful for determining walking motion and therefore the accelerometer data is not used for the classifier; however, the capability to read accelerometer data was added in order to assist with future efforts.

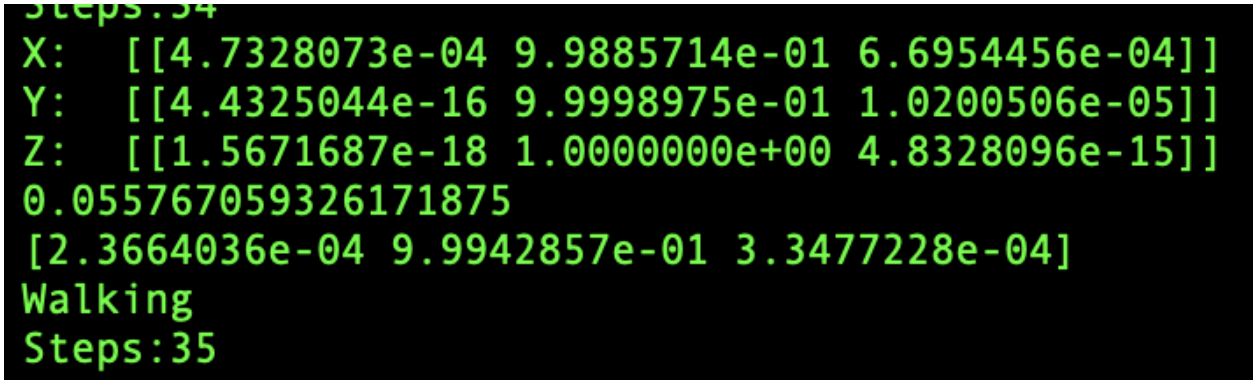

Figure 15: The host application prints the output of each classifier's output, the total execution time of the classifiers in seconds, and the final output after ensembling. The motion type and total step count are printed afterward.

Future improvements

Hardware

Changes in the hardware design may unlock additional benefits, including a further reduced PCB footprint, increased power efficiency, and additional functionality. Much of this functionality may be obtained through the use of an even-lower power processor, such as those with an ARM Cortex-M0 core. A smaller battery than the 18650 cell currently in use may be used in such a system and allow for further miniaturization of the system; this battery may be a small lithium-ion pouch cell, or even a lithium coin cell. The Sense Hat itself would not be required in its entirety and a small form factor MEMS-based sensor may be used as the IMU solution. An on-board Bluetooth Low Energy solution may also be used to further reduce power consumption on side; the CC2541 IC used by the HM-10 may be used for this purpose. Other solutions include a microcontroller solution with built-in Bluetooth connectivity such as the ESP32 series. The resulting board with all these changes implemented would be small enough to be a fraction of its current size while retaining a similar battery life.

More power-efficient circuitry may be used in addition to these changes, including a proper battery management system IC and more power-efficient LEDs. Algorithm changes to the microcontroller code may be made to use less transmissions and therefore consume less power. It is likely that such a change can be made with the use of filters and thresholds to prevent data indicating idle motion from ever leaving the microcontroller.

Host system and application

The host and its application software remain a proof-of-concept system at this point in time, although there exists significant potential for improvement here. Today, most people carry around a powerful smartphone with integrated Bluetooth interfaces and are capable of running machine learning models using dedicated hardware. It is natural that the clunky laptop system and command-line interface Python application can be replaced with a smartphone application, as it is expected that most users will carry their smartphones out and about. Additional motion data may be used to refine the model to make it more accurate. The Keras model may even be converted in its entirety to support this purpose, although it is

expected that such an adaptation would be followed by extensive re-training of the model with new motion data.

Other additional features may be added to the classifier, such as the ability to determine specific types of walking motion (running, stairs, walking, small steps) and the user's current walking speed through analysis of the IMU data. It is also possible for the classifier to be trained to operate in the frequency domain instead of using timeseries data as an input.

Acknowledgements

[1] Z. Murtishi, "Bluetooth-enabled Smart Pedometer System," ELE 547 Final Report, University of Rhode Island, 2022.