Pokemoneter: A Beaglebone Pedometer For Your Pokemon!

Team Name: Pokemoneter

Edwin Sun (edsun@bu.edu)

Mari Tsunoda (tsunodam@bu.edu)

GitHub Repository: https://github.com/edsun123/ec535_FinalProject

Abstract

Many current pedometer applications on the market recommend a daily step count of 10,000 steps per day. However, only having a large number to work towards lacks an incentive for people to stay healthy. The Pokemoneter allows people to enjoy the stepping experience by hatching and finding cute pokemon eggs. This project uses the LSM9DS0 accelerometer to detect a user's step and relays the information to the Beaglebone black. The Beaglebone is connected to an LCD display cape which shows the interactive QT Pokemoneter application. There are three types of Pokemon egg, which each have different step requirements to hatch. The user can find a new egg to capture 10 steps after hatching a pokemon egg. The application also has a storage and shop feature. The storage can hold up to 6 pokemon, and the user can select a pokemon to walk with or release. In the shop, the user can purchase new pokemon eggs if space permits in the storage.

Introduction

The goal of this project is to develop a pedometer to train your pokemon. Being active is important to be healthy, but many people are sedentary and that can lead to weight and stress issues. Moreover, video games are becoming more popular but they can encourage players to become more inactive. However, Pokemon GO is an excellent application that gets inactive individuals to go out and walk. This motivates us to create a similar application but for an QT application in a system consisting of the beaglebone and an accelerometer. Also, this application will also gamify the experience and motivate users using nice displays and goals of raising a pokemon egg. An overview of our system architecture consists of two sides: hardware and software. In the hardware side of the system, a sensor is hooked up to the beaglebone with which it conducts transactions of data that contains reading of acceleration, magnetic field strength, and angular momentum. In the software side of the system, a driver will open the device file for the sensor and communicate the sensor device using the i2c protocol. Given specific i2c device addresses and register addresses, we can copy data concerning acceleration readings from which program can determine if a user has made a step or not. As for the front end portion of the software, the QT application will read from a text file which the driver module will write the step counter. The Qt application consists of a start screen, a store page, a pokemon storage screen, and the main screen where the step counter is displayed. Once the number of steps is accomplished, playful art and gifs will be played to show the egg hatching and congratulating the player. Overall, we were extremely successful in fully implementing the complete system: connecting the hardware and programming the driver and QT application. The system is able to successfully display the number of user's steps and play gifs. Although we were able to finish our project goal, it is far from functionally perfect which we discuss more in the project details.

Design Flow

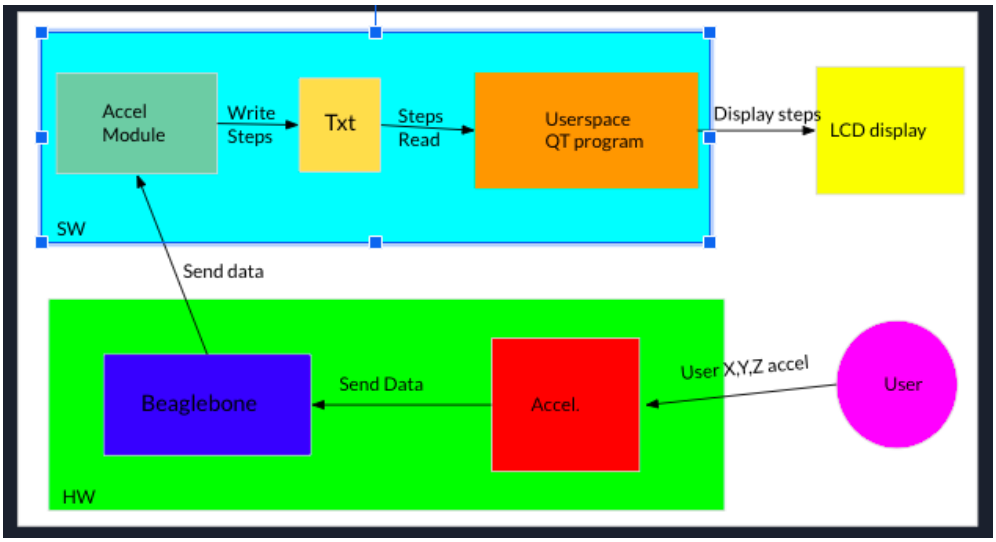

Figure 1. System Architecture Diagram

The following table shows the responsibilities and project split for each team member:

Table 1. Team Member Responsibilities

Contributors	% of Project	Responsibilities
Edwin Sun	50	• Connecting hardware between Beaglebone, touch screen, and accelerometer • Writing the driver module to communicate between Beagleboard and sensor • Data process the data • Write step detection • Handled GitHub and handled integrating both partner's code
Mari Tsunoda	50	• Developed QT Application • Added 4 windows in-app • Found and created graphics (gifs/images) • Handled logic for hatching eggs in different stages • Read from and write to a text file to show the number of steps done

Project Details

Hardware Component List:

1) 9-DOF Accelerometer/Magnetometer/Gyroscope (LSM9DSO)
2) Beaglebone Black
3) LCD Display Cape
4) USB to TTL Serial Cable
5) Small Breadboard
6) Jumper Wire Kit
7) AC/DC Power Supply

Accelerometer Module

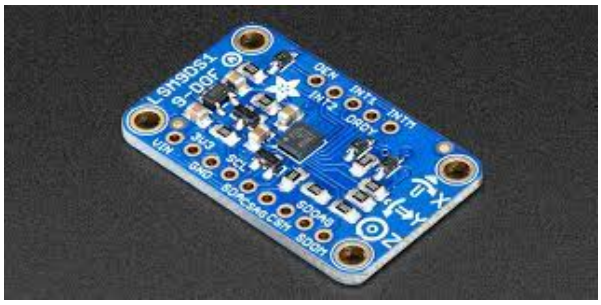

Figure 2. LSM9DS0 Sensor

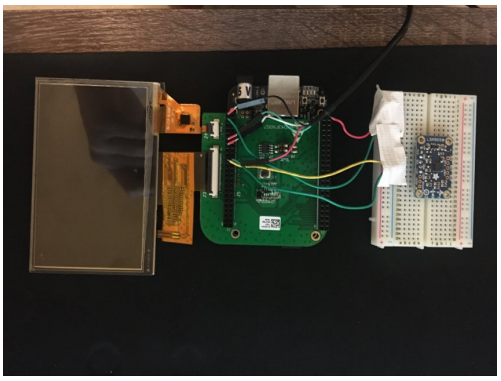

Figure 3: Hardware Setup

On the hardware side of the system, three components are needed: Beaglebone, the sensor, and the LCD cape interface. The LCD cape interface is connected to the Beaglebone with ribbon wires onto the connectors on the cape interface. The USB-to-TTL serial cable can be attached to the UART4_RXD, UART4_TXD, and DGND pins. Finally, the sensor's power, ground, SCL, and SDL can be connected to their respective pins on top of the interface. (Pins 19 and 20 are used for bus 2 I2C devices for SCL and SDL connection).

Inside our driver program, the device file "/dev/i2c-2" is open, and the file descriptor is retrieved. The device needs to be communicated using ioctl which requires a file descriptor of the device file and a specific i2c address. There are two specific i2c addresses: x6b for the gyroscope and x1d for the magnetometer and accelerometer. Once communication is established between Beaglebone and i2c devices, reading and writing data becomes possible. However, specific registers' addresses are needed to know where to read and write the data. Using the datasheet, there is a table of all the addresses for all registers. For example, x28 is a registered address that holds the most significant byte for the acceleration data. First, setting configs must be written to enable the sensor to begin taking measurements and allow reading/writing into the device file. Once configs are set, the device file is written the specific

register addresses we wish to read from. The bytes from those registers are read and copied into a buffer.

With the data in our buffer, that data needs to be processed. The data stored in the buffer is split into most significant and least significant bytes of 2's complement. Those bytes need to be rearranged in the proper order and converted from 2's complement to a regular integer. These measurement readings are stored in a struct.

However, given these readings, whether a step is taken or not needs to be determined. Out of all the measurements, the accelerations readings are most useful; and for simplicity, only the acceleration in the z-axis is used. A step detection algorithm is written to read a stream of z acceleration readings to determine if a step is taken. It determines if a step is taken if it detects if the user is raising their and lowering their foot. This can be determined using thresholds of which there are four: top, bottom, high, and low. The algorithm can detect if the user is raising their foot if readings enter the region between the bottom and low threshold and if the user is lowering their foot if readings enter the region between top and high threshold. Once it establishes that the user has already entered the state of raising and lowering their foot, it can determine a step is taken and increments a global step counter. Additionally, as time goes on, the threshold changes and adjusts to changes in the user's stride.

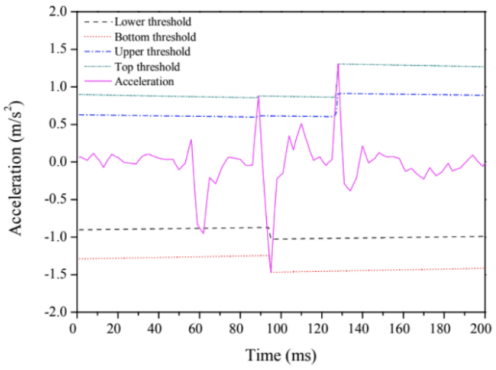

Fig 3. Example graph of accel. readings with thresholds

Finally, inside the main function, there is a while loop to keep taking measurements from the accelerometer every 0.1 seconds, and the readings are passed into the step_detection function. Then a text file will be opened and written with the global step counter. That text file will be read by the Qt application to know how many steps to display.

QT Application

The QT application is split into the following four windows: main menu, step count, storage, and shop. The main menu is the landing page and has three buttons: play, reset and quit. The play button connects to the step counter page and the quit button closes the application. The reset button stores a green egg as default, and the user walks with the egg from step 0.

Figure 4. Main Menu Page

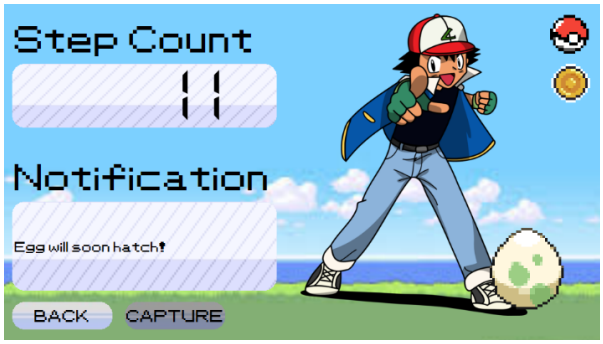

Figure 5. Step Count Page

Figure 6. Storage Page

Figure 7. Shop Page

The step count page has the step count indicator, the notification box, and storage, shop, capture, and back buttons. The step count display shows the step since the last time the user pressed the reset button on the main menu page. The notification box shows different messages at different stages of the pokemon egg being walked. The egg itself is split into four stages: pre-hatched egg, about to hatch egg, hatched pokemon, and found the new egg. The notifications differ as shown in the images below.

(Note: The “about to hatch” egg stage shows a jumping egg gif and cannot be inserted into PDF)

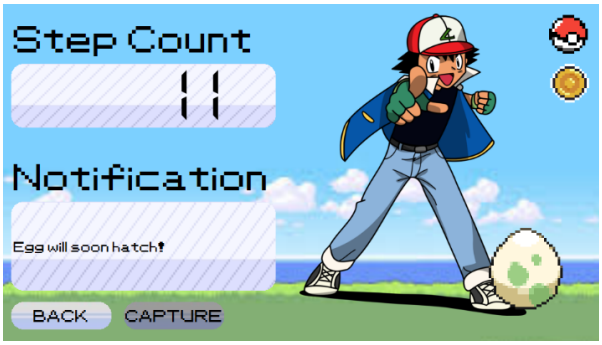

Figure 8. Pre-hatched Egg

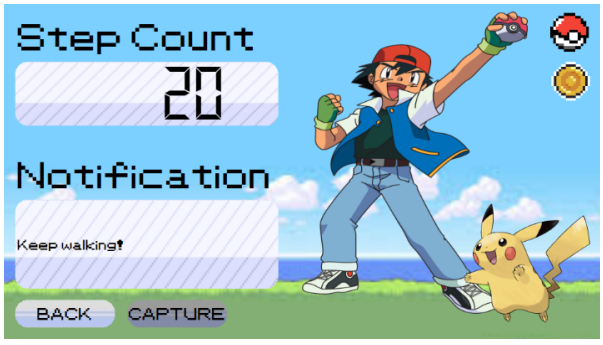

Figure 9. Hatched Pokemon

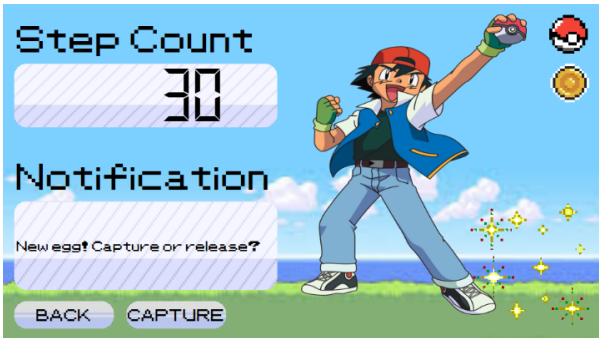

Figure 10. Hatched Egg

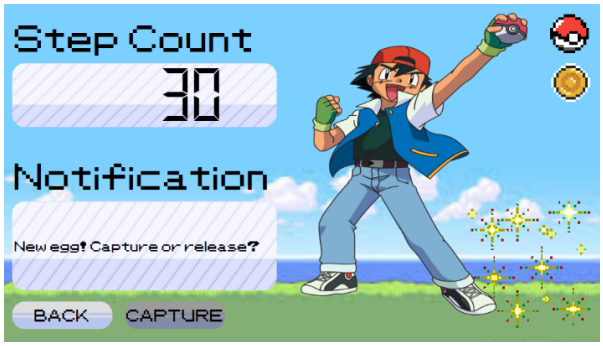

Figure 11. Captured Egg

When prompted to catch the new egg, the user can decide to capture it or not. The user can only store a new egg if space is available in the storage, which holds a maximum of 6 pokemon. If the user decides to not capture, the user can keep stepping and the notification will change. If the storage is full or the user presses the capture button, the button will be shaded.

Figure 12. Bulbasaur Selected

Figure 13. Captured Egg

On the storage page, the user can tap on an egg or pokemon to select to walk with or release. If the pokemon highlighted is currently selected for walking, the select button is shaded. In the example above, the Bulbasaur is selected, and when release is pressed, the Bulbasaur disappears from the slot and the first slot is highlighted and selected as default. When a pokemon is released, the icon background and name disappear.

Figure 14. Buying Green Egg with 20 Coins

Figure 15. Green Egg Bought, Eggs Not Available

In the shop, there are three types of eggs available for purchase. The green egg hatches Pikachu, the blue egg hatches Squirtle, and the purple egg hatches Bulbasaur. The user has 20 coins to use in the example above. When the green egg is pressed, the user can select the buy option, but when the user presses options that cost more than the user's coin amount, the buy button is shaded. If the user has no space in the storage, the buy button is also shaded and the user cannot purchase any eggs.

The main challenge for this QT application was to figure out how to use QT and finding and collecting the images to match a theme. A lot of time was spent creating variations of the same images. For example, when a button is pressed, the background image is changed to a shaded version. Another example is when an egg is selected in the shop, the egg is highlighted. Unfortunately, there are still some glitches with reading and writing files that are yet to be fixed on the step counter page. Another issue we still have is that the step counter skips over certain steps at times. However, the initial goal during the project proposal was to just have the steps displayed. We were able to add the two bonus features of displaying a message in the notification box and adding images of eggs and pokemon for each stage. We surpassed the bonus features and were able to add the shop and storage for extra fun!

Summary

This project creates a fun way to work towards the daily step count goal of 10,000 steps. By utilizing an accelerometer, we were able to understand and implement the I2C bus. We also learned how to create a widget application in QT Creator for the first time. Initially, we expected to classify a step from the accelerometer data and display the step count as our baseline goal for this project. The bonus features we thought to implement during our project proposal were the notification boxes and the pokemon hatching animations. However, we were able to surpass this goal and even implement a shop and storage feature. Some remaining challenges on the QT application side are the small bugs in the storage functionality. This issue seems to be caused by the file reads and writes. This causes a small delay in showing the step counter. The new egg finding system only finds the same colored egg as the pokemon being walked with. The next step would be to randomize the new egg generated. With regards to the pokemon available, we only have three pokemon (one per egg-type) so the next step would be to add more. In the storage system, it would be nice to have a progress bar for the steps required until hatching for an egg. As for the driver application, the algorithm for detecting human steps can be improved by handling more measurements from other axes of acceleration and gyroscope to better predict when a step is made. Moreover, the application can become more responsive if both driver and Qt applications can be integrated into a singular kernel module. (Unfortunately, the 12 hour time difference makes collaboration difficult)

Overall, we really enjoyed creating our Pokemoneter pedometer and learned a lot about the I2C bus and how to create widget applications in QT Creator.

References

[1] QT Documentation. “https://doc.qt.io/qt-5/”

[2] LSM9DS0 Data Sheet. “https://cdn-shop.adafruit.com/datasheets/LSM9DS0.pdf”

[3] Adafruit. “Adafruit LSM9DS0 Accelerometer + Gyro + Magnetometer 9-DOF Breakouts” https://learn.adafruit.com

[4] Derek Molloy I2C tutorial. “http://derekmolloy.ie/beaglebone/beaglebone-an-i2c-tutorial-interfacing-to-a-bma180-accelerometer/”

[5] Adafruit I2C addresses list. “https://learn.adafruit.com/i2c-addresses/the-list”

[6] Adafruit LSM9DS0 Library. “https://github.com/adafruit/Adafruit_LSM9DS0_Library”

[7] BeagleBone Black I2C References. “https://datko.net/2013/11/03/bbb_i2c/”

[8] ioctl linux manual. “https://man7.org/linux/man-pages/man2/ioctl.2.html”

[9] Sparkfun LSM9DS0 Hookup Guide. “https://learn.sparkfun.com/tutorials/lsm9ds0-hookup-guide”

[10] Naqvib, N.Z., Kumar, A., Chauhan, A. and Sahni, K., 2012. Step counting using smartphone-based accelerometer. International Journal on Computer Science and Engineering, 4(5), p.675.

[11] Beaglebone Black Pinout. “https://beagleboard.org/Support/bone101”

[12] i2cdetect linux man page. “https://linux.die.net/man/8/i2cdetect”