Pipboy 3000

Team: Vault 101
Team Members: David Klaus, Kevin LeShane, Alexander Welles
Class/Semester: EC 535 - Spring 2013
Professor: Dr. Ayse Coskun

BOSTON UNIVERSITY
Boston University College of Engineering
Department of Electrical & Computer Engineering

ABSTRACT

Team Vault 101 designed an intuitive, visually appealing, wrist mounted display capable of reporting environmental data from an accelerometer, thermistor, and magnetometer. A Gumstix Verdex waysmall computer was used to communicate with each sensor via the i2c bus as well as run a Fallout 3 themed Pipboy graphical user interface. The Pipboy is battery powered, mounted on the left wrist, contained within a plastic case, and uses accelerometry data to activate the LCD screen when the user makes a "watch gazing" gesture. User accelerations, ambient temperature, compass heading, and ambient magnetic measurements are displayed on three different screens which can be selected via the 4.3" touch screen. The project required writing various user level c and c++ programs and the development of a QT based GUI. Although the project was a success and exceeded initial expectations, there is room for improvement in terms of aesthetic appeal, functionality, and marketability.

INTRODUCTION

Topic:

The goal of Team Vault 101's EC535 final project was to create an intuitive, visually appealing, wrist mounted display capable of reporting environmental data when triggered by a gesture. The sensors chosen for integration were a thermistor, for sampling ambient temperature; an accelerometer, for gesture control; and a magnetometer, for navigation (degree heading) and ambient magnetic field measurements. Our intent was to emulate the form and function of the Pipboy 3000, a device from the video game series Fallout. The gesture we chose to control our Pipboy was the "watch gazing" motion that the average user makes when checking the time.

Motivation:

The motivation for this project drew fundamentally from our team's interest in video games (the aesthetic of the Fallout franchise in particular) and the potential for implementing a feature-full worn device.

From an academic standpoint, designing, organizing, and implementing the Pipboy provided a number of useful lessons that expanded upon skills we had already learned through EC535 coursework. Aside from reminding us of the value of persistence (above all) -- see project details -- this project required multiple levels of embedded system development including the development of kernel modules (even if eventually abandoned), algorithms (gesture recognition), i2c bus communication (sensors), and GUI aesthetics. Furthermore, implementation of our project's goals required working with outdated hardware, outdated software, and under time constraints exasperated by personal lives and rapacious academic schedules.

Personally, each group member wanted to realize a physical object from a familiar video game. We all wanted to build something we could be proud of that was not only "cool" but had the potential to scale into a useful device. The Pipboy's utilitarian functionality and interesting design aesthetic made it an obvious choice particularly considering the form factor allowed us to explore several interesting product design and prototyping problems.

The Big Picture:

After our initial research into available sensors, and considering the group's skill level, time limitations, and the requirements of the EC535 final project, Team Vault 101 created the following design overview for the Pipboy project:

The Pipboy design has three elements: One, a visually appealing and intuitive graphical and physical user interface; two, the ability to collect and display environmental data from a variety of sensors (initially an accelerometer and thermistor); and three, the ability to recognize a user's gesture as a control input (gesture turns the screen on and off).

Accomplishments:

We were able to accomplish all of our stated goals and added an additional sensor and functionality into the Pipboy. An accelerometer was successfully integrated along with the development of a robust algorithm capable of detecting a specific gesture. The "watch gazing" gesture was used to trigger the screen when the user desires and to return it to a sleep state when no longer being viewed. The graphical user interface (GUI) provides multiple screens of data from all three of our sensors in an intuitive format with aesthetically pleasing graphics and touchscreen interaction. The device is battery powered, comfortably mounts on the left wrist, and boots the Pipboy GUI when powered on. The addition of a magnetometer provides a compass heading and ambient magnetic field data for the user.

Design Flow:

In its final form the Pipboy is mounted on the wrist using a velcro armband connected to a plastic case displaying an LCD screen.The device has five components, a Gumstix board with arm processor running linux 2.6.21, an ADXL345 accelerometer, a HMC5883L magnetometer, a TMP102 thermistor, and a Samsung 4.3" LCD touch screen. The sensors are housed in a separate compartment from the battery pack, LCD screen, and Gumstix board and interface via the I2C communication bus. A GUI built in QT displays sensor data on the LCD screen and allows for touchscreen interaction.

The QT program displays three different frames accessed through user touch control; one frame for accelerometry data, one frame for magnetometer data, and one frame for thermistor data. The GUI emulates the look of the Pipboy 3000 from the Fallout series, including the background, fonts, and the use of the Vault boy character. The Pipboy boots its GUI at startup and is battery powered for portability. If the accelerometer reads that the Pipboy is not being held up for viewing, the LCD screen output of the Pipboy is disabled to save batteries.

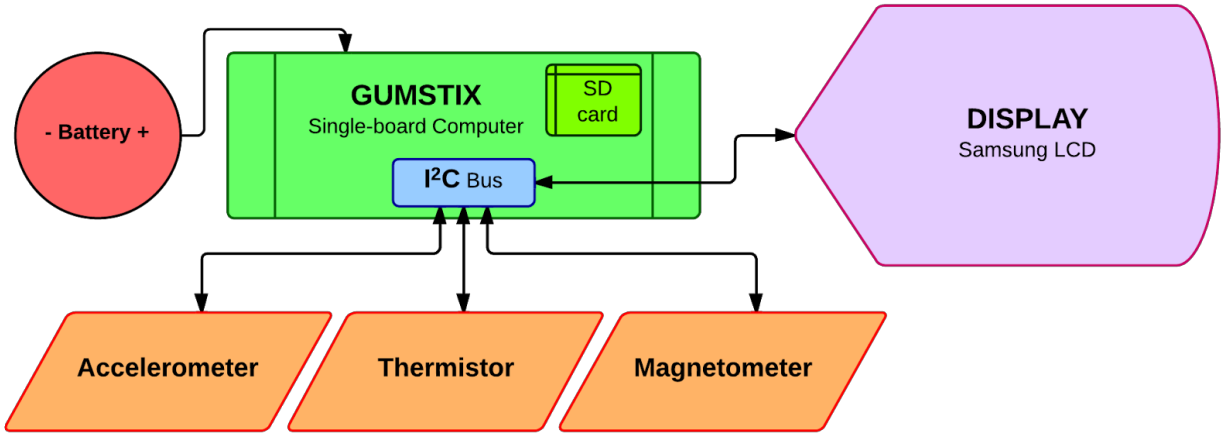

Figure 1: Hardware flow chart (Gumstix, peripherals, battery, memory and data lines)

Work By Team Member:

Alex Welles wrote several C programs to interface with each of the sensors over the I2C bus, detect the "watch gazing" gesture, and disable/enable the LCD screen. These programs were eventually combined and inserted into the QT GUI. Alex wired the sensors and came up with the method used for powering the gumstix board using batteries. Finally, Alex created the mounting and case used to house the Pipboy device.

David Klaus developed a set of steps used to compile QT programs created with QT creator and developed a basic template GUI used to debug the implementation of Alex's c program's functionality in a QT program named pipboy_1. David ported the QT libraries to an SD card and linked these libraries from the Gumstix board. David created script S98pip_boy_start script used by the Gumstix bootup scripts to start the final QT program -- Pipboy -- during boot. David assisted in debugging Kevin's QT program.

Kevin LeShane coded the Pipboy's QT GUI and designed its graphical elements. His efforts combined all of the functionality of Alex's mainwindow.cpp program in a clean, easy to use, attractive GUI. Kevin created the presentation slides for Vault 101's initial and final presentation and most of the graphics that Vault 101 used.

All group members contributed to this final project report.

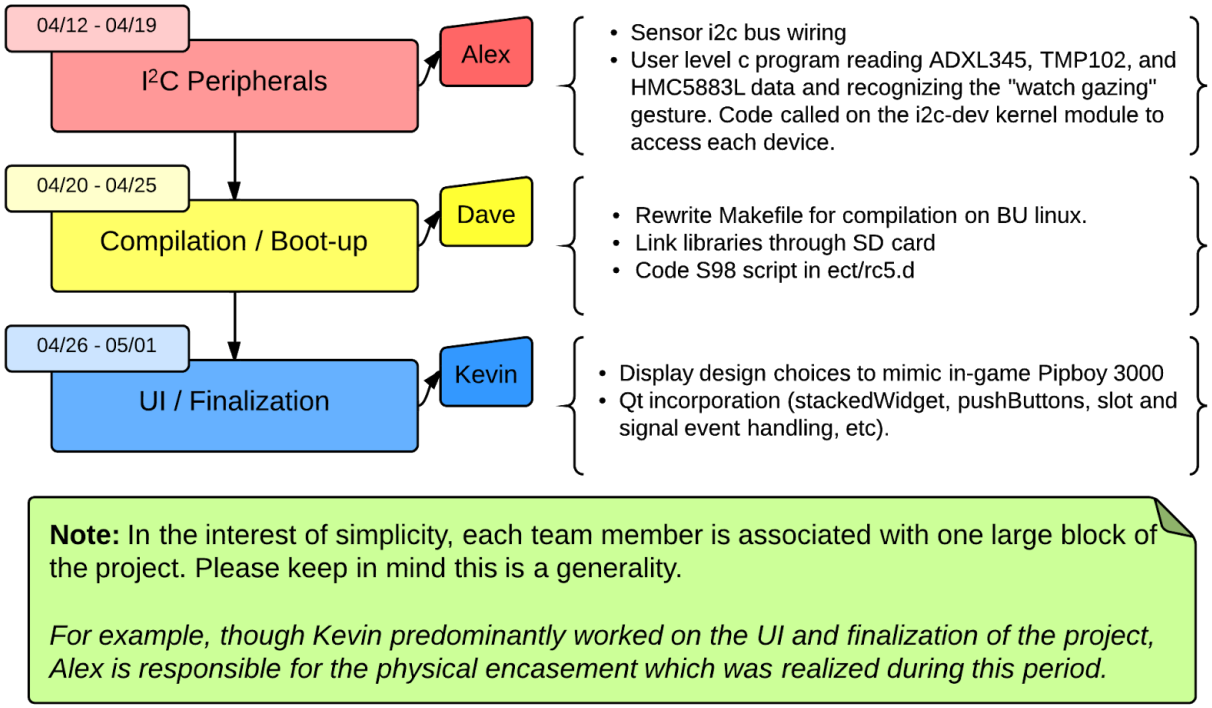

Figure 2: Workflow and task distribution flow chart.

PROJECT DETAILS

Preliminary research:

Most of the preliminary research for this project revolved around porting device drivers for the three sensors used to linux 2.6.21. All of the devices for this project have device drivers available online however all of these drivers require the use of several functions not included in linux 2.6.21 -- mostly relating to functions clear_work(), and flush_work() as well as certain structures within linux workqueue.h. Unfortunately, backporting these functions to linux proved time prohibitive and the premade device drivers for the three sensors were not used.

After deciding to move in a different direction research shifted to requesting data from the three sensors used in this project using existing I2C device drivers and manually reading to and writing from registers on the sensors through I2C. This research was fruitful and was used to read and write from the sensors using code developed by Alex Welles's.

Finally, research moved onto developing QT programs. Initially, the group's research focused on searching for a premade GUI program similar enough to our requirements for a Pipboy GUI such that minor modifications could produce a good quality Pipboy GUI. Simultaneously, the group searched for a method of creating a QT program from scratch. Eventually David developed an executable using QT Creator capable of visualizing on the LCD. The group decided to use QT Creator to create an original GUI rather than modify existing code.

Hardware:

Triple Axis Accelerometer Breakout - ADXL345

Initial Inter-Integrated Circuit (I2C) efforts focused on reading data from the ADXL345 accelerometer chip. The retailer's website (1) provided links to existing drivers (2), sample code, and the chip data sheet. However, the existing drivers were written for a more recent linux version (Linux 2.6.30 or above) and the sample code was written for use with the Serial Peripheral Interface (SPI) Bus, a protocol we were unfamiliar with. Attempts at backporting the ADXL345 driver provided by Analog Devices stalled due to the complexity of writing new code to register I2C devices with the Gumstix and the large volume of code (> 1000 lines) contained within the drivers and their peripheral files. Successful communication with the ADXL345 chip was accomplished with assistance from Juan See (also in our class) who provided a link to sample c code reading from the ADXL345 on a Raspberry Pi (3). This code utilized the i2c-dev driver which was already included as a module in the default Gumstix open embedded kernel we were provided with.

Exploration of the i2c-dev.h/.c files and probing using the i2c shell command (i2c [options] [i2c-address] [cmd] [cmd options]) allowed us to successfully read data from the ADXL345 periodically. However, we encountered a surprisingly high rate of "failure to read/i2c bus busy" error messages. These failures to read data from the ADXL345 were initially thought to be due to a lack of pick up resistors; however, testing revealed that the ADXL345 chip was not correctly configured/wired. Although the GND, VCC, SCL, and SDA pins had been set correctly, the chip was not operating as an i2c slave device until the CS pin was set high (VCC) and the SD0 pin was set low (GND) (Figure 3 and 4) per datasheet instructions (4).

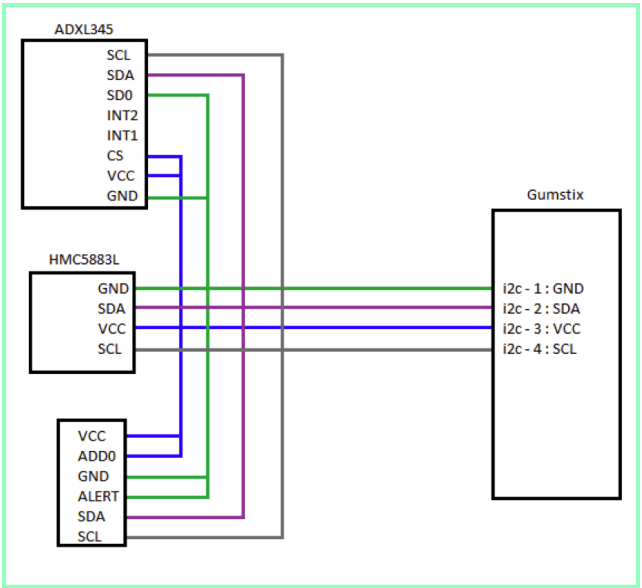

Figure 3 : Wiring diagram of gumstix and peripherals. Note: the unlabeled chip on the bottom

left is the TMP102.

Once configured as a slave device, reading to and writing from the ADXL345 chip was trivial and accomplished with a brief c user level program containing an initialization block (setting the chip's power mode) and a while loop reading six data buffers at a defined frequency. Data for the accelerations measured by each axis was stored in two registers per axis in a twos-complement format with the lower address register containing the least significant bit (LSB). The portion of the attached mainwindow.cpp code labeled "/* ADXL345 Initialization */" shows the i2c read and write calls made to initialize the ADXL345 power, data acquisition, and data reporting modes into a suitable state for our Pipboy. Almost all values were set to default except the POWER_CTL register (0x2D) which was set to measure/autosleep mode (0x11) and we did not utilize any of the chip's other power saving options. Although the chip was set for automatic sleep when data is not being read, it never enters into the sleep mode as Pipboy GUI reads data from the ADXL345 at a rate of 10Hz or greater. The GUI's calls to the ADXL345 occur in the timerTimeout() function in mainwindow.cpp under the comment /* ADXL345 read */.

Digital Temperature Sensor Breakout – TMP102

The TMP102 was the second sensor to be successfully integrated via the I2C bus but, required its bus address to be set by connecting the ADD0 pin to VCC, GND, SCL, or SDA (Figure 3). We connected ADD0 to VCC giving the chip an address of 0x49. Sample code was provided from the retailer's website (5) but it proved simpler to adapt the code previously written for the ADXL345. The TMP102 was left in its default temperature sensing range and data was read from two registers storing temperature data in two's complement. See the portions of mainwindow.cpp commented as /* TMP102 Initialization */ and /* TMP102 Read*/ for details. Note: although we used the calibration coefficient (0.0625) suggested in the previously mentioned sample code, we found that the temperature values reported by our program using this calibration seemed high.

Triple Axis Magnetometer Breakout – HMC5883L

The HMC5883L Magnetometer was connected to the I2C bus but required no wiring modifications beyond connecting the GND, VCC, SDA, and SCL pins (Figure 3). A modified version of the i2c-dev code used for the previous two sensors was used to initialize and read from the HMC5883L (see /* Initialize HMC5883L */ and / * HMC5883L Read */ respectively in mainwindow.cpp). The magnetometer was initialized to its default gain/sampling rate and continuous measurement mode (6). Note: our compass degree heading was calculated by taking the arctangent of the scaled x and y magnetic magnitudes (in LSb/Gauss). However, our heading readings behaved erratically and often seemed incorrect. This may in part be due to interference from interference created by nearby electronics but also suggests that a more accurate heading might be determined by summing the magnitudes of the 3 axes and adding them. The direction in which the sum of these magnitudes is greatest is most likely to be magnetic north and could provide a modified/additional heading output.

PROGRAMS/SOFTWARE:

User level C program for reading sensor data and "watch gazing" gesture recognition

Code was initially written for debugging and testing each sensor separately in order to minimize the likelihood of errors. Once each device was working satisfactorily on its own, the code for each chip was combined into a user level C code that reported the data collected from all three chips to the terminal via the printf function. At this point our research efforts forked with David Klaus focusing on integrating the C level code into a basic QT graphical user interface (GUI) template which Kevin Leshane later modified into the final polished GUI. Alex Welles developed the algorithm for detecting the "watch gazing" gesture based on empirical research and implemented the screen sleep/wake modes.

The gesture recognition algorithm was developed by having 10 individuals wear the ADXL345 on both their right and left wrists (connected by a 3' cable) and repeatedly raise their arms as if reading a wrist watch. The ADXL345 was mounted on the right with the axes in the orientation shown in Figure 4. By using a ratio between the x/y, x/z, and y/z we were able to develop gesture recognition algorithm that worked on both hands. Our prototype however was designed to be worn on the left hand as most right handed and left handed people wear their watches on their left wrist (according to a sample size of 10).

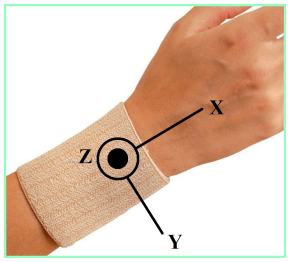

Figure 4 : ADXL345 X, Y, and Z axes orientation. The Z axis extends out of the page.

Repeated tests revealed that ratios of x/y, x/z, and y/z accelerations were generally close to 0, 0, 1 when rounded. Interestingly, this ratio is preserved during the "watch gazing" gesture when the accelerometer is worn in a mirrored position on the left wrist. This is most likely due to the expected accelerations of the X and Y axes being inverted which does not have an effect on the overall sign of the ratio (as -X/Y == X/-Y). The code beneath the comment "/* Gesture Recognition */" in mainwindow.cpp contains the algorithm's implementation and the following snippet contains the "if" statement identifying the "watch gazing" gesture.

```
if( (adxl345_xa_ya > -1.0) && (adxl345_xa_ya < 1.0) && (adxl345_xa_za > -1.0)
&& (adxl345_xa_za < 1.0) && (adxl345_ya_za > 0) && (adxl345_ya_za < 5.0) ){
```

Sampling windows were selected so that 10 consecutive reports of the desired gesture triggered a screen on event. The user level program sampled at a rate of 10Hz which translates to 1 second in the desired gesture position. A period of 2.5 seconds or 25 consecutive reports of a position other than the desired “watching gazing” gesture trigger the screen to enter quick sleep mode and disable the LCD screen backlight.

Enabling and disabling the LCD screen was done by using “system("pxaregs LCCR0 [register value]")” and “system("echo 0 > /sys/class/graphics/fb0/blank").” The former command enables or disables the LCD screen by setting or clearing the LCCR0_DIS and LCCR0_ENB bits. When the LCD is disabled by setting the LCCR0_DIS bit it allows the frame buffer to empty without writing any new data. This creates a ghosting effect but is the suggested method for sending the LCD screen to sleep per the pxa27X user manual’s suggestion (7). Writing a 0 or 1 to the /sys/class/graphics/fb0/blank file enables or disables the LCD backlight.

Final Form: Wiring, Case, and Wrist Strap

The case design, wrist strap integration, and final wiring were done the day before the presentation and hinged around the happy accident that the Samsung 4.3” touchscreen fit perfectly in the lid of a Muji pencil case. To reduce the size of the wiring which had previously been done on a breadboard, the GND, VCC, SCL, and SDA lines were split to connect to each sensor using male to female headers. The male pins of each of the four headers were soldered together to the output wire running from the gumstix board (Figure 5) allowing up to 10 individual connections per input (GND, VCC, SCL, and SDA). The pencil case had accessibility ports cut into it to display the screen and access the serial and power ports as well as to pass wires from the main compartment to the sensor compartment (Figure 5). A type A 0.7mm barrel power connector was soldered to a battery pack for 4 AAA batteries in order to provide a modest 45 minutes of battery life. It is worth noting that the batteries do not drain completely but unfortunately only provide > 4.5v (despite being 1.5v each) for roughly 45 minutes. Once the voltage drops beneath 4.5v the Gumstix enters into a perpetual boot cycle where it reboots as it loads the QT program onto the screen. Further modifications were made to the pencil case to allow the insertion of a wrist brace band which served as the arm mounting method for the device (Figure 8).

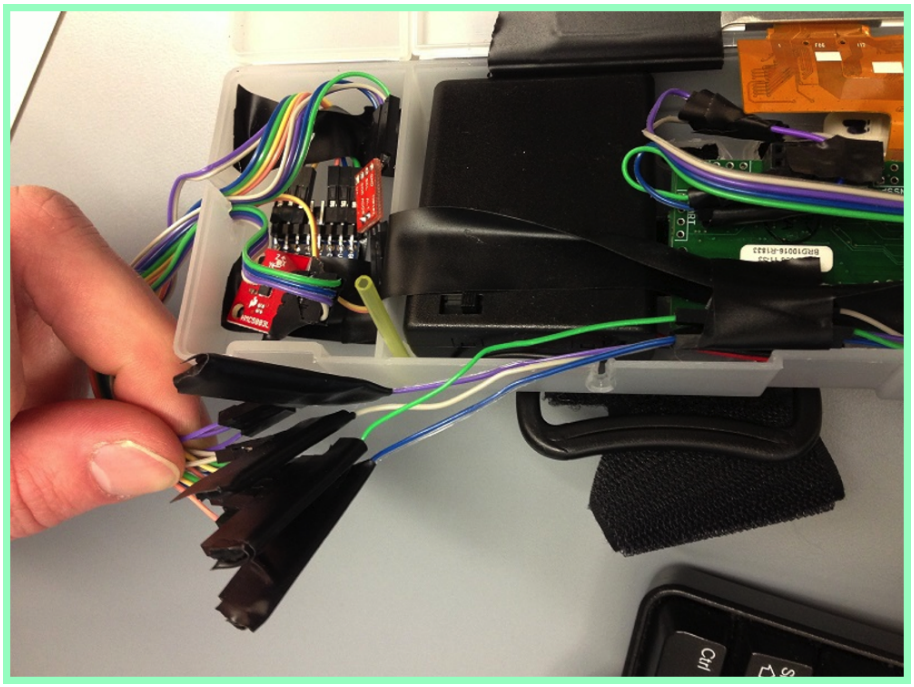

Figure 5 : Wiring from Gumstix to i2c sensors. The green wire is GND, blue is VCC, purple is SDA, and grey is SCL. Also shown: 4xAAA battery pack with power switch and GND, VCC, SDA, SCL splitters for connecting to multiple sensors.

Pipboy Program:

Difficulties during development:

Like all aspects of this project, taking the user interface from concept to completion with an adequate level of polish and functionality was no trivial endeavor. The prototype UI developed by David Klaus was a great launchpad for development, but a new project file and hours of sifting through QT documentation (8) and developer forum threads proved requisite to create a final platform. As an example of the type of challenges faced, finding a suitable Qt-specific container to hold a multi-paged UI demanded a certain familiarity with nuances of the software. After numerous failed attempts Kevin eventually found the stackedWidget object QT employs to provide a paged interface to be the solution for multiple screens. This example demonstrates the lengthy process necessary for implementing only one function of the Pipboy UI using QT - many similar examples can be cited: pushButtons, Slot and Signal event handling, Widgets, QLabels, etc.

Aesthetics and final implementation:

The final design of Vault 101's Pipboy UI was strictly based on graphics from Fallout 3. Our group obtained image sources such as Vault Boy illustrations from the Bethesda website, home of the game studio that developed Fallout 3. Using Adobe Photoshop, Kevin rendered images in order to create a visually cohesive Pipboy UI emulating the look of Fallout 3's Pipboy 3000. After finalizing individual page layout, pushButtons were added and a signal/slot handler was used to decide the next page to display. Meanwhile, Alex Welles' timer method updated sensor input variables regardless of current page display

every 100 milliseconds.

Notes and thanks:

Finally, A notable design challenge encountered during the UI development process was the realization that a Mouse Pointer Event (traditionally used for desktop clickable mouse pointers) was the solution for touch input using our LCD. Due to the dated Linux and QT versions available to us, QTouchEvents were unavailable. A brief acknowledgement is due to Ozan of Team Awesome who provided our team with this tip. Thanks Ozan!

SCRIPTS:

S98pip_boy_start:

The S98pip_boy_start script accomplishes three tasks. First the pipboy script exports several paths local to the S98pip_boy_start script which provide paths to ts libraries and device drivers used to communicate with the Samsung LCD screen. These drivers and libraries enable screen output, touch functionality, mouse support, screen interrupts, as well as other functionality. The final two export paths:

```
export QT_QWS_FONTDIR=/media/card/lib/fonts
export TSLIB_PLUGINDIR=/usr/lib/ts
```

provide paths which direct the -qws flag for QT compilation to the QT fonts/ts_lib folders. Without these path exports the -qws flag directs compilation based on default qmake paths for fonts/ts_lib specified by the BU linux qmake.

Second, the S98pip_boy_start script links provide soft links to the QT libraries located on the SD memory card. Though these soft links should only need to be created once, the gumstix system sometimes deletes the soft links between power down and power up. This could possibly be a problem with our gumstix board, but it is a simple and not terribly wasteful solution to include the soft link commands within the S98pip_boy_start script.

Finally the S98pip_boy_start script boots the “Pipboy” QT program during boot. The S98pip_boy_start script is located in the /etc/rc5.d folder, and linked with a soft link through the /ect/init.d folder. all S files run at boot up based on the number following the S tag, S1 is run first at boot, then S2, then S3, and so on until S99 which is the login script. By saving S98pip_boy_start with the S98 tag the “Pipboy” program is run at boot before the system requests a login password. As an added benefit, because the QT libraries required to run “Pipboy” are stored on an ejectable SD card, if the SD card is ejected before boot, the gumstix linux terminal will boot normally for debug or system modifications. This ensures that broken code, or a missing library does not brick the gumstix board.

MAKEFILE:

QT creator project compilation process:

Despite generating a .pro file which directs qmake when creating a Makefile for a particular

project, .pro files imported from QT creator assume particular paths for linux based QT development. Unfortunately, the BU linux setup places files in paths that the QT creator .pro file does not anticipate. Because of this, when qmake is called on a .pro file generated by QT creator a Makefile is generated which can not compile an arm linux executable. Because of this the compilation methods, include paths, libraries, qmake paths and other fields of a resultant qmake file must be rewritten in order to create a working arm-linux executable file.

Summary and closing remarks:

Although our Pipboy is only a proof of principle device, we feel that the features we have incorporated into it make it extremely marketable. The goal of Team Vault 101's EC535 final project was to create an intuitive, visually appealing, wrist mounted display with functionality and style emulating the Pipboy 3000 from the Fallout video game series. The final "Pipboy" achieved all of these goals. It reads information from multiple sensors into a stylized GUI which takes visual elements and design aesthetics from the Fallout series and displays real time data. It has a simple touch screen interface that allows the user to select a different data display by touching the left or right edge of the current screen. The Pipboy is portable and implements interesting and useful additional components such as the "watch gazing" screen disable/enable feature and a compass heading.

However, there is more that can be done. A custom case more closely resembling the packaging of the Pipboy 3000 from one of the Fallout games could be developed in a program such as solidworks and printed with a 3-D printer or vacuum formed. The GUI could be expanded to include features such as a GPS based map display or additional sensor data. Finally, additional gestures could be used in conjunction with knobs, buttons and lights so as to more closely resemble the functionality of the UI from the Fallout series.

Although these improvements might only appeal to fans of the Fallout series, finding other uses for a portable sensor display capable of multiple forms of input and sensing with gesture recognition is not a difficult task. Specialized sensor suites could be selected for use in particular civilian and military fields and trainable gesture recognition could eventually be implemented allowing each user to customize the motions that control his or her device. The unit could be sold as a pre configured device or a base unit with easy to install sensor pack add-ons allowing users to customize the function of their device. Regardless of the myriad marketing strategies and options available for our Pipboy, Vault 101 is proud of our creation and can't wait for the next project.

Figure 6: Pipboy with Temperature Display Screen.

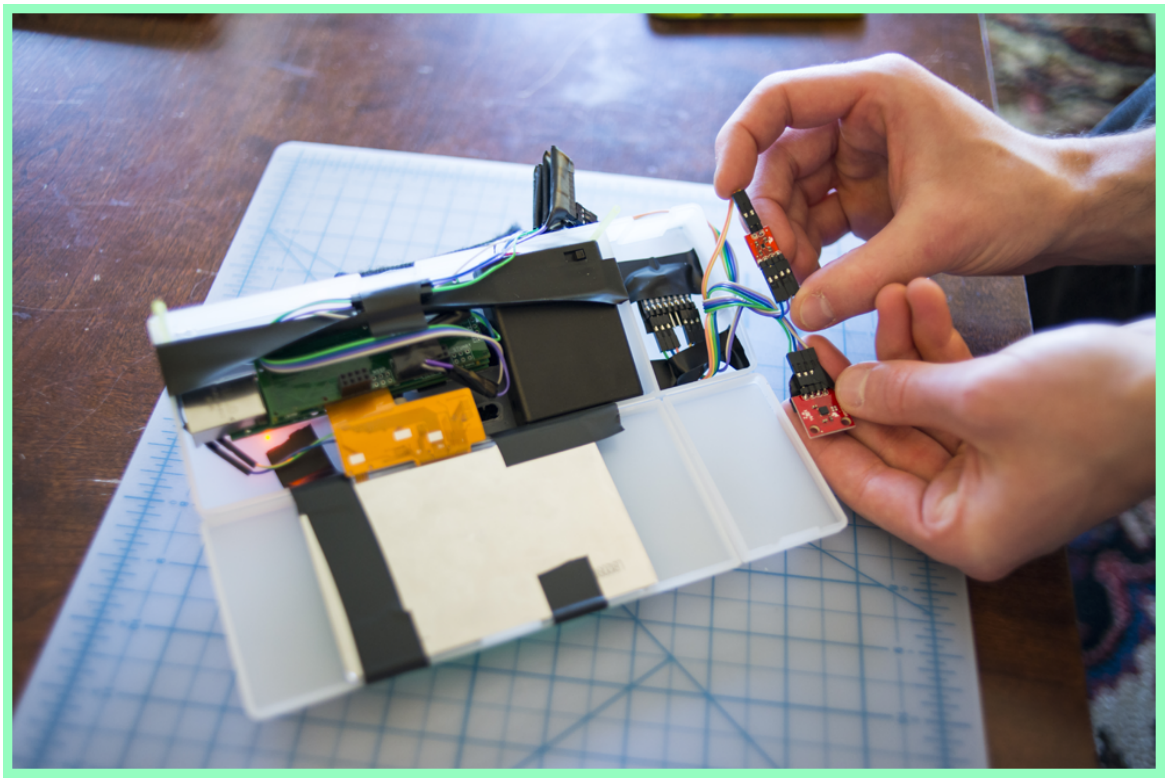

Figure 7: Pipboy casing open with TMP102 Temperature Sensor and HMC5883L Magnetometer out.

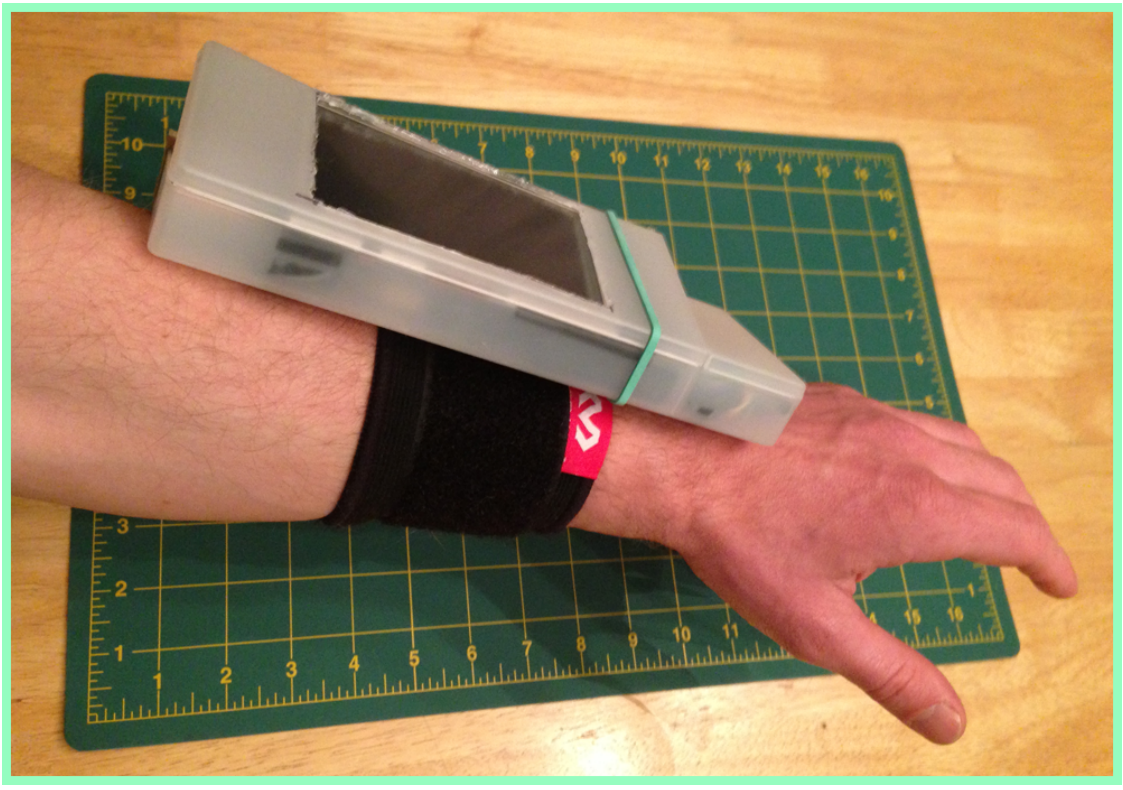

Figure 8: Pipboy with wristband mount.

References:

1. ADXL345 retailer website, datasheet, and sample code: https://www.sparkfun.com/products/9836

2. ADXL345 linux 2.6.3 drivers, http://wiki.analog.com/resources/tools-software/linux-drivers/input-misc/adxl345

3. ADXL345 i2c-dev sample code, http://www.raspberrypi.org/phpBB3/viewtopic.php?t=12503

4. ADXL345 data sheet, https://www.sparkfun.com/datasheets/Sensors/Accelerometer/ADXL345.pdf

5. TMP102 retailer website, datasheet, and sample code: https://www.sparkfun.com/products/9418

6.HMC5883L retailer website, datasheet, and sample code: https://www.sparkfun.com/products/10530

7. pxa27X developers manual: http://int.xscale-freak.com/XSDoc/PXA27X/28000402.pdf

8. QT Developers code repository/website: http://qt-project.org/