ARL-TN-0796 • SEP 2016

US Army Research Laboratory

Development and Application of a Wireless, Networked Raspberry Pi-Controlled Head-Mounted Tactile Display (HMTD)

by David Chhan, Joel T Kalb, and Kimberly Myles

NOTICES

Disclaimers

ARL-TN-0796 ● SEP 2016

US Army Research Laboratory

Development and Application of a Wireless, Networked Raspberry Pi-Controlled Head-Mounted Tactile Display (HMTD)

by David Chhan, Joel T Kalb, and Kimberly Myles
Human Research and Engineering Directorate, ARL

REPORT DOCUMENTATION PAGE			*Form Approved* *OMB No. 0704-0188*

1. REPORT DATE *(DD-MM-YYYY)*	**2. REPORT TYPE**	**3. DATES COVERED (From - To)**
September 2016	Technical Note	October 2014–July 2016

4. TITLE AND SUBTITLE Development and Application of a Wireless, Networked Raspberry Pi-Controlled Head-Mounted Tactile Display (HMTD)	**5a. CONTRACT NUMBER**
	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S) David Chhan, Joel T Kalb, and Kimberly Myles	**5d. PROJECT NUMBER** H70
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) US Army Research Laboratory Human Research and Engineering Directorate ATTN: RDRL- HRF-C Aberdeen Proving Ground, MD 21005-5425	**8. PERFORMING ORGANIZATION REPORT NUMBER** ARL-TN-0796
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	**10. SPONSOR/MONITOR'S ACRONYM(S)**
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT

13. SUPPLEMENTARY NOTES

14. ABSTRACT

As the head-mounted tactile display's (HMTD's) efficacy in augmenting Warfighter performance was studied and transitioned from lab-based to field experimentation, the need for a portable and robust system emerged. Previously, a Windows-based netbook computer was used as a tactor controller but its size, weight, and power consumption limited its use as a wearable, outdoor device. Raspberry Pi (RPi), part of the "wearable computer" trend, became an ideal replacement. The RPi's size and weight support HMTD portability; the ad hoc wireless-networking mode allows a network of them to move freely while communicating with one another without a centralized infrastructure. This is critical to field studies where team tactile communication, on the move or in a highly dynamic setting, is a priority. This report details the development of RPi as a tactor controller and fills informational gaps during development of the RPi-controlled HMTD. It lists procedural steps in setting up the RPi and dealing with its functions and operations—a guiding manual for the RPi's use as a low-cost controller to power prototypes for field studies. While this report applies specifically to the RPi's development as a tactor controller, we believe the procedures are of general interest and applicable for mobile experimentations.

15. SUBJECT TERMS

Raspberry Pi, wireless ad hoc networking, head-mounted tactile display, tactile communication, HMTD

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON**
					David Chhan
a. REPORT	**b. ABSTRACT**	**c. THIS PAGE**			**19b. TELEPHONE NUMBER (Include area code)**
Unclassified	Unclassified	Unclassified	UU	45	410-278-5985

Standard Form 298 (Rev. 8/98)
Prescribed by ANSI Std. Z39.18

Contents

List of Figures

List of Tables

INTENTIONALLY LEFT BLANK.

1. Introduction

The development of a wireless, networked tactor controller using a Raspberry Pi (RPi) was motivated by the need to deploy a head-mounted tactile display (HMTD) in field studies. These studies evaluated the efficacy of the display in augmenting Warfighter performance. The previous version of the HMTD used a Windows-based netbook computer that was suitable only for lab-based experiments.[1] Its limitations for the field were not only its 3-lb weight and 4-h battery life but, more severely, the heat it generated within the confinement of a backpack. Limited wireless range was also an issue. As our experiments transitioned from lab to field, a more mobile and robust system was needed. When evaluating a Warfighter's ability to perceive directional information via the HMTD while running and jogging, a lightweight, portable, low-power, heat- and shock-resistant, rugged prototype tactor controller was required to support our data-collection effort. Future field studies will involve Warfighters engaged in intense activities and maneuvers like crawling and jumping on an obstacle course while wearing the system. These activities could potentially damage the hard drive and screen display of a netbook; therefore, the replacement of the netbook computer with a credit-card-sized Raspberry Pi Model B+ (made available July 2014) was required. Since future applications will also include the support of small-team and squad communications, we implemented a peer-to-peer, ad hoc mode that permits multiple RPi's to be wirelessly connected. This application will be critical to the development of a bidirectional HMTD to support up to squad-level communications field tests. This technical note serves as a guiding manual for those who wish to use RPi as a low-cost controller to power portable electronic prototypes. While this manual applies specifically to the development of RPi as a tactor controller, we believe the procedures are of general interest and applicable for mobile experimentations with audio and video signals.

2. The RPi

2.1 Raspberry Pi Model B+ Specifications

We used an RPi Model B+ to replace a netbook computer as a wireless tactor controller. Figure 1 is a picture of the RPi Model B+.

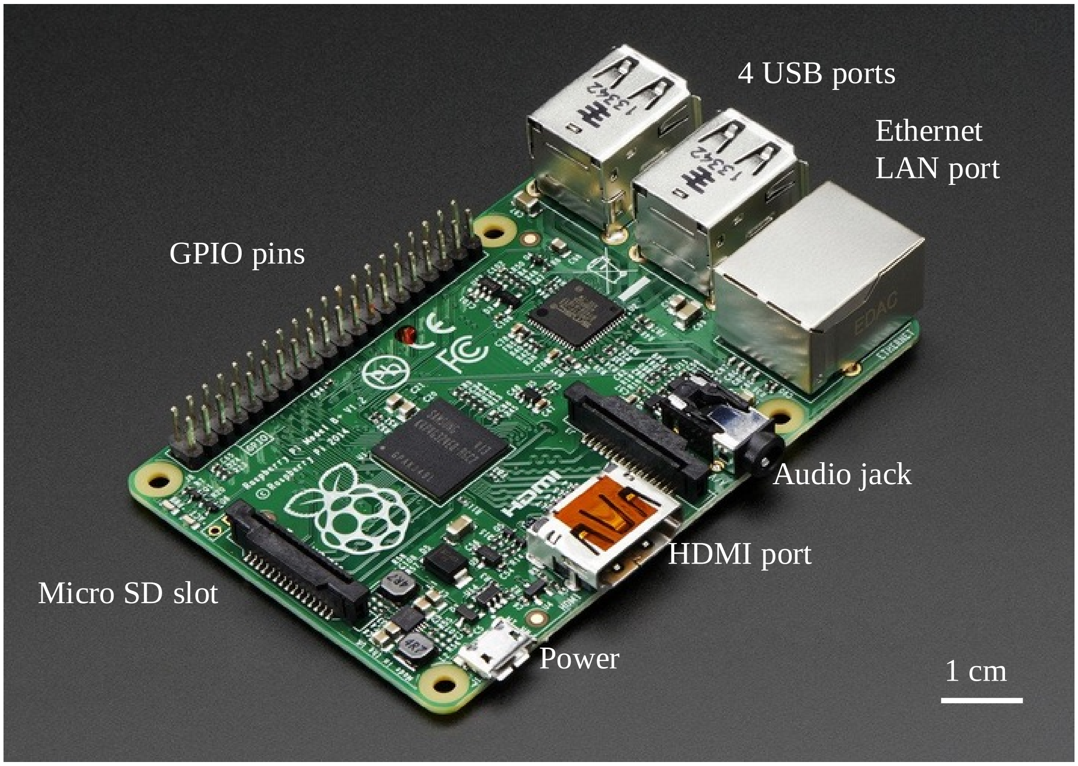

Fig. 1 Components of the RPi Model B+ [2]

Table 1 lists the specifications of the RPi Model B+.

Table 1 RPi Model B+ specifications

Processor system on chip (SoC)	BCM2835 SoC
Central processing unit (CPU)	700 MHz single-core ARM1176JZF-S
Memory (SDRAM)	512 MB
Storage	Micro storage device (SD), 4 GB or 8 GB
Expansion header	40
General Purpose Input/Output (GPIO)	26
USB 2.0 ports	4
Video input	15-pin MIPI camera interface (CSI) connector
Video output	HDMI port
Audio output	3.5-mm jack
Network	10/100 M bit/s Ethernet port
Liquid crystal display (LCD) interface port (DSI)	1
Power	650 mA, 3 W
Size	85 × 56 ×17 mm
Weight	45 g

2.2 Raspbian Operating System

The "officially recommended" operating system (OS) for RPi is a Linux-based Raspbian OS. The OS was developed and optimized for RPi hardware, though there are other third-party operating systems (Ubuntu, Windows, etc.) available for the RPi. For convenience and general acceptance, we used the recommended Raspbian OS. The OS is stored and installed on a micro-SD card. One can purchase a micro-SD card with a preinstalled Raspbian OS. For self-installation, visit http://raspberrypi.org/downloads and follow the instructions on the page. The website provides good resources on how to install Raspbian OS and other third-party OSs.

2.3 Setting up the RPi

Assuming the Raspbian OS is already installed on the micro-SD card, the RPi can be set up for running with a display monitor (connected through HDMI port) and a keyboard (connected through USB port). Once a display and a keyboard are connected, power the RPi. A terminal-like window appears. If login is required, the default username is `pi` and password is `raspberry`. (The password can be changed in the configuration.) Run the configuration tool using the following command. A menu-type window (Fig. 2) will appear. Use the arrow keys to navigate and return key to select-menu options.

```
pi@raspberrypi ~ $ sudo raspi-config  %Open
configuration tool setting
```

`pi@raspberrypi ~ $` is the command prompt; `sudo raspi-config` is the command; `%Open configuration tool setting` is the description of the command. That format will be used throughout this technical note.

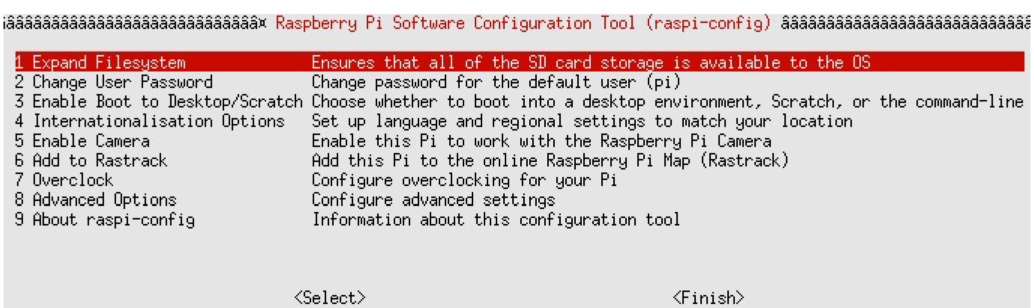

Fig. 2 Raspbian configuration menu

2.4 Network–Internet Connection

Once the RPi is set up and running, the next step is to connect it to the Internet. Here, we describe a general way of how it is done using a Dynamic Host Configuration Protocol (DHCP). In a later section, we will go into details of how to use an ad hoc or peer-to-peer mode to form a cluster of networked RPi's that allows us to communicate between multiple Pi's without the need for a centralized network such as a router. The DHCP is the common service available on the network equipment (i.e., the router) that hands out unique IP addresses to all computers that want to join the network. The network connection can be made through a wired (Ethernet local area network [LAN]) or wireless (Wi-Fi USB adapter) setup. For wired setup (Ethernet cable needed), connect the RPi to the router through the Ethernet LAN port. Wi-Fi setup can be completed in the terminal mode through a modification of the network interfaces or in the graphical user interface's (GUI's) desktop mode through Wi-Fi Config application. A Wi-Fi USB adapter is needed. For the Wi-Fi adapter, the RealTek RT5370 chipset is recommended because we found it was the only one that worked and had consistent network connectivity. Use the `lsusb` command to see a list of connected USB devices and details.

2.4.1 Terminal Mode

In the terminal, type the following command to edit the interfaces file and edit the file as follows:

```
pi@raspberrypi ~ $ sudo nano /etc/network/interfaces
     %open and edit the interfaces file
>
auto wlan0
iface wlan0 inet dhcp
wpa-ssid "SSID"                %your router ESSID
wpa-psk "password"             %your router password
```

Press `Ctrl+x` to exit the nano text editor and enter `y` to save the document. Also edit wpa_supplicant.conf file as the following:

```
pi@raspberrypi ~ $ sudo nano
/etc/wpa_supplicant/wpa_supplicant.conf
>
ctrl_interface=DIR=/var/run/wpa_supplicant
GROUP=netdev
update_config=1
network={
     ssid="SSID"
     psk="password"
     proto=RSN
     pairwise=CCMP
}
```

Press `Ctrl+x` to exit the editor and enter `y` to save the document. Restart the RPi with a command '`sudo reboot`'. After the reboot, the RPi should be connected to the Wi-Fi network.

2.4.2 GUI Desktop Mode

Make sure the `/etc/network/interfaces` file includes the following line:

`wpa-conf /etc/wpa_supplicant/wpa_supplicant.conf`.

In the terminal, type the following command to open the GUI desktop mode:

```
pi@raspberrypi ~ $ startx    %start a GUI desktop mode
```

Once the terminal switches to GUI desktop mode, open the "Wi-Fi Config" application. A "wpa_gui" window will appear. You should be able to see the Service Set Identifier (SSID) of your router. Use "scan" to see a list of available Wi-Fi networks. Select the one you want to connect to and enter the password.

Once set up and connected, the RPi's IP address and network Extended Service Set Identifier (ESSID) can be checked using the following commands:

```
pi@raspberrypi ~ $ ifconfig        %display the
network configuration
pi@raspberrypi ~ $ iwconfig        %display
information about the access point
pi@raspberrypi ~ $ ip addr show eth0    %eth0 is the
Ethernet port.
> inet 192.168.1.20/24 brd 192.168.1.255 scope global
eth0
pi@raspberrypi ~ $ ip addr show wlan0   %wlan0 is the
Wi-Fi adapter.
> inet 192.168.1.15/24 brd 192.168.1.255 scope global
wlan0
```

The digits between `inet` and the `/` character are the RPi's IP address. If the IP address does not show up, RPi is not connected to the network. Once connected to the Internet, we can update the system with the following commands[3]:

```
pi@raspberrypi ~ $ sudo apt-get update  %check what
packages have been updated.
pi@raspberrypi ~ $ sudo apt-get upgrade %upgrade and
install new up-to-date packages.
```

The RPi can also be accessed headless (no monitor, screen, or keyboard connected to the RPi) using a laptop computer with SSH (secure shell), assuming SSH is enabled in the Raspberry Pi Software Configuration Tool (raspi-config) and given that the RPi's IP address is known. Access to the RPi using SSH can be achieved through either wired or Wi-Fi as described previously with the Internet connection. For Windows computer, use PuTTY (free online download) as an SSH client to connect to the RPi. Provide a host name or IP address and log in as “pi” with a password (the default password is “raspberry” if it is not changed). For Mac OS computers, use a Terminal or X11 (free online download). Type `ssh pi@[ip address]` and enter the password to connect. If having a problem connecting to the RPi, make sure your computer is connected to the same Wi-Fi network as the RPi. If the RPi is assigned a static IP address, make sure to configure your computer’s IP address to be in the range of the same private network class as the RPi; that is, if the RPi’s IP address is 192.168.2.1 with the subnet mask of 255.255.255.0, your computer’s IP address should match the first 3 numbers with the unique 4th as 192.168.2.5 with the same subnet mask. This can be done by manually entering the numbers under the Transmission Control Protocol (TCP)/IP tab in the network configuration advanced setting.

2.5 Useful Commands

Since the RPi uses a Linux-kernel-based OS, commands used in its terminal are basically Linux commands. Here are some useful commands[4]:

```
ls                      % list the content in current directory
lsusb                   % list attached USB devices
cd                      % change current directory to a specified
one
pwd                     % print (display) working directory
mkdir                   % make a new directory
rmdir                   % remove a specified directory
nano example.txt        % open example.txt using nano, the Linux
                        text editor
cat example.txt         % list the content of the file
example.txt
startx                  % open the graphic user interface (GUI)
rm                      % remove a specified file
cp                      % copy a file and place it in a specified
                        location
mv                      % move a file to a specified location
chmod                   % change permission of a file
df / -h                 % display disk space
ping [ip address]       % check if communication can be made with
                        another host
ifconfig                % display the network configuration
iwconfig                % display information about the access
                        point and signal quality
iwlist wlan0 scan       % print a list of the currently available
                        wireless networks
sudo su                 % become the root user
sudo reboot             % reboot
sudo shutdown –h now    % power off your Pi before pulling out
the power plug
exit                    % logout
```

3. Development of RPi as a Tactor Controller

In our specific application of an HMTD, the RPi is used to control an array of Engineering Acoustics, Inc. (EAI) C-2 tactors through a number of Class D audio amplifiers. To drive the tactors, a generated tactile-signal waveform stored in the micro-SD card is played through the audio port of the RPi using a system function called "aplay". Tactors are selectively enabled for activation using RPi GPIO pins. Figure 3 shows a working prototype of our wireless RPi tactor controller with some of its hardware components. In the following sections, we discuss a step-by-step "how to" for each hardware and software component required to successfully activate the tactors.

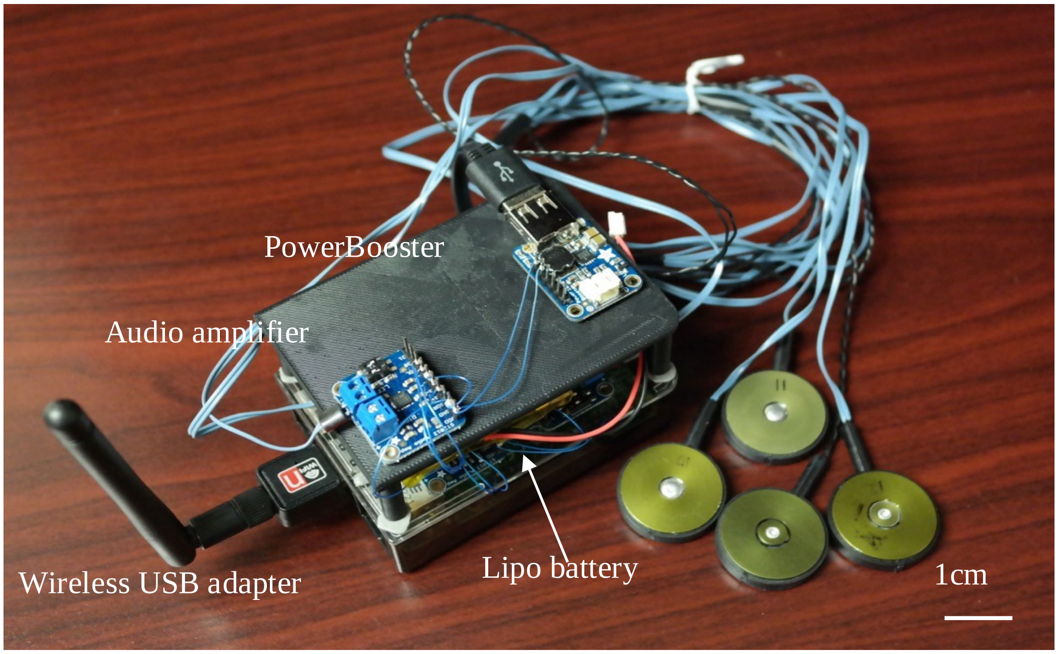

Fig. 3 A working prototype of the wireless RPi tactor controller

3.1 Hardware

A list of hardware items needed to build a wireless tactile controller is listed in Table 2.

Table 2 Hardware for building a wireless tactile controller using an RPi

Hardware	Quantity
Tactors	4
RPi Model B+	1
4-GB micro-SD card	1
Class D audio amplifier	2 (left and right channels can be used separately to power 2 tactors)
Rechargeable lipo[a] battery	1
PowerBoost 5V boost	1
Micro USB to USB adapter	1
3.5-mm audio connector	1 (not needed if connected wires are soldered onto the audio port directly)
RealTek RT5370 Wi-Fi USB adapter	1
Wire-wrapping wires	. . .
Wire-wrap hand tool	. . .
Soldering kit	. . .

[a]lipo: lithium-ion polymer.

3.1.1 EAI C-2 Tactor

Similar to a vibrator in a cellphone, the EAI C-2 tactor (shown in Fig. 4) is a miniature vibrotactile transducer that has been optimized to create a strong localized sensation on the body. It is designed with a primary resonance in the 200–300-Hz range that coincides with peak sensitivity of the Pacinian corpuscles, the skin's mechanoreceptors that sense vibration. Table 3 lists the specifications of the C-2 tactor from EAI.

Table 3 EAI C-2 tactor's specifications[5]

Physical dimension	1.2-inch diameter × 0.3 inch high
Weight	17 g
Exposed material	Anodized aluminum polyurethane
Electrical wiring	Flexible, insulated #24 AWG
Skin contactor	0.3-inch diameter, preloaded on skin
Electrical characteristics	7.0 Ω nominal
Insulation resistance	50 MΩ minimum at 25Vdc, leads to housing
Response time	33 ms max
Transducer linearity	+/– 1 dB from sensory threshold to 0.04-inch peak displacement
Recommended drive	Sine-wave tone bursts 250 Hz at 0.25A rms nominal, 0.5 A rms max for short durations
Recommended driver	Bipolar, linear or switching amplifier, 1 W max, 0.5 W typical

Fig. 4 EAI C2 tactor[5]

3.1.2 Class-D Audio Amplifier, Its Wiring, and RPi GPIO Pins Layout

Figure 5 shows the TS2012 Class-D stereo amplifier, which is capable of delivering 2 × 2.8 W channels into 4-ohm impedance speakers. It is available at online electronic retailers (such as Adafruit) for less than $10. Inside the miniature chip is a Class-D controller, able to run from 2.7 V-5.5 V DC. Since the amplifier is Class D, it is highly efficient (89% efficient when driving an 8Ω speaker at 1.5 W)—perfect for portable and battery-powered projects. It has built-in thermal and over-current protection.

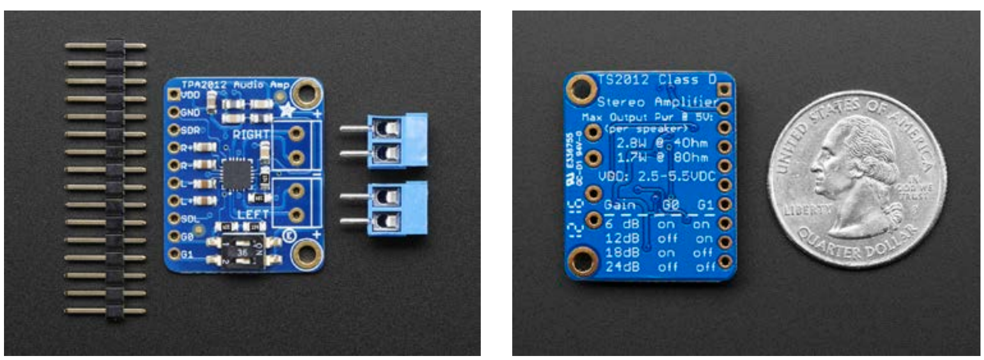

Fig. 5 Class-D audio amplifier[6]

The inputs of the amplifier go through 1.0 µF capacitors, so they are fully "differential". In our case, we simply tied the Right and Left to ground (see Fig. 6). The outputs are "bridge tied", meaning they connect directly to the outputs, not to ground. They cannot be connected to another amplifier and must drive the speakers directly. The enable pins SDL and SDR are enabled by either 3.3 V or 5 V so they can be controlled by either the 3.3 V RPi or the 5 V Arduino. (Arduino is another common and popular microcontroller.) Figure 6 also shows input and output wiring of the amplifier. At the inputs of the amplifier, both VDD and GND can be connected to either the battery or the RPi. Enable pins SDR and SDL are connected to the RPi GPIO pins. The RPi GPIO layout is shown in Fig. 7. GPIO pins allow RPi to interact with the physical world; thus, we used them as a switch to control and enable the tactor through amplifier enabled pins. The R+ and L+ are connected to the audio output of the RPi (3.5-mm audio connector). At the output end, 2 tactors are connected to the left and right channels.

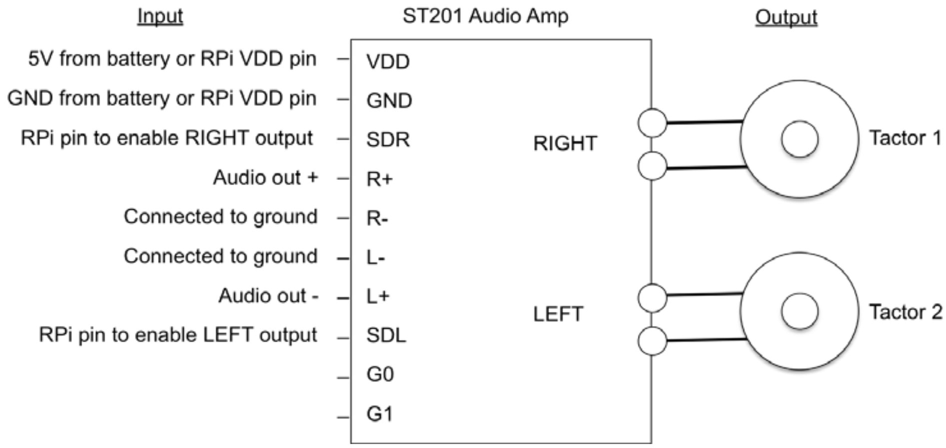

Fig. 6 A schematic of how the audio amp is wired

Function	RPi B+ J8 Pin		Function
3.3v	1	2	5v
GPIO2	3	4	5v
GPIO3	5	6	Ground
GPIO4	7	8	GPIO14
Ground	9	10	GPIO15
GPIO17	11	12	GPIO18
GPIO27	13	14	Ground
GPIO22	15	16	GPIO23
3.3v	17	18	GPIO24
GPIO10	19	20	Ground
GPIO9	21	22	GPIO25
GPIO11	23	24	GPIO8
Ground	25	26	GPIO7
ID_SD	27	28	ID_SC
GPIO5	29	30	Ground
GPIO6	31	32	GPIO12
GPIO13	33	34	Ground
GPIO19	35	36	GPIO16
GPIO26	37	38	GPIO20
Ground	39	40	GPIO21

Fig. 7 RPi B+ GPIO pins layout

3.1.3 Lipo Rechargeable Battery

We used a lipo rechargeable battery, Model LP785060 (Fig. 8), to power both the RPi and audio amplifiers. The battery is thin, light, and powerful. The output ranges from 4.2 V when completely charged to 3.7 V. It has a capacity of 2500 mAh for a total of about 10 Wh. It also is available at online electronic retailers, for less than $15.

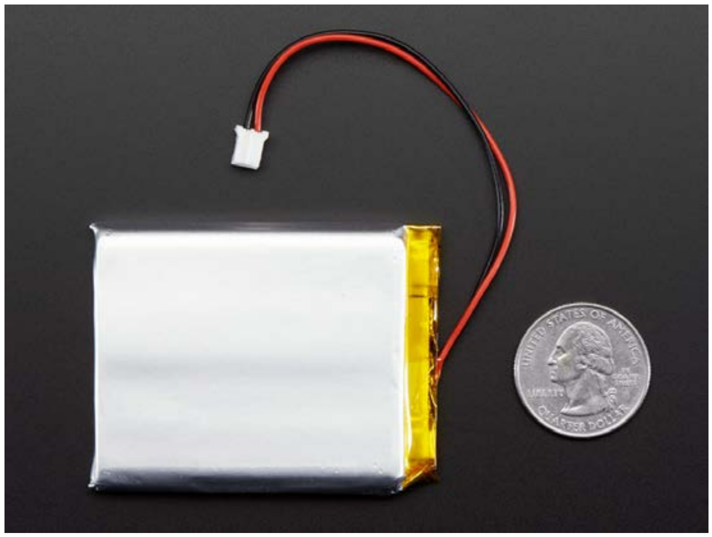

Fig. 8 Lithium ion polymer battery[7]

Since the RPi is powered by a 5 V micro-USB supply, we used a PowerBoost 1000C rechargeable 5 V lipo USB Boost to step up the 3.7 V lipo battery to 5 V. It is available online for less than $20. The lipo battery can be connected to the PowerBoost directly while the connection from the PowerBoost to the RPi needs a USB-to-Micro USB adapter. In the left picture of Fig. 9 is the PowerBoost with a detached USB port (soldering is needed to mount the USB port to the PowerBoost).

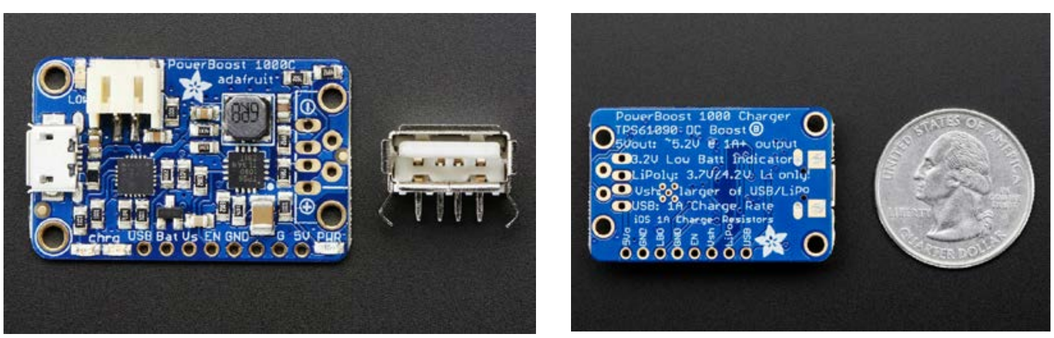

Fig. 9 PowerBoost 1000C (right) and with USB port detached (left)[8]

3.2 Software

The code was written in C programming language using the nano text editor. There are a number of subroutines that were used to run the tactile display. These include subroutines for generating waveforms and a Waveform Audio (WAV) file, reading keyboard entry, enabling/selecting RPi GPIO pins, and sending characters among multiple RPi units using the TCP/IP wireless–ad hoc network. Details of the programming codes are attached in Appendixes A through E. In the following subsections is an overview of the functionality of each subroutine.

3.2.1 How to Make Waveforms and Generate a WAV File

WAV files are a standardized format for acoustic signals. The format used in this project are mono, 16-bit samples with a sampling rate of 48 kHz. These can be recorded from a microphone or generated using computer calculations. The structure of a WAV file begins with a header chunk containing the file information (e.g., file type and size) followed by a format chunk containing information such as number of channels and sampling rate; this, in turn, is followed by a data chunk containing the memory allocation for the total number of samples. The data stored in this chunk are either mono or stereo with the left and right channels interleaved. The finished file is written to the SD card for storage using the block-write binary C command. The file can then be played out of the RPi audio stereo port using the shell command "aplay (WAV ffile)". The example code (Appendix A) shows how a WAV file using Morse code was generated from the dot–dash script. This requires a precalculation of the total number of samples needed in order to allocate memory. The SD card can hold a large number of prerecorded WAV files that can be accessed by either a basic–intermediate shell (also known as BASH) script or a C program.

3.2.2 How to Read Keyboard Input

We have an array of tactors and a number of different WAV files to play, which required a mechanism to control them using an input interface. For proof of concept and prototyping demonstration, we chose a simple keyboard entry as our input interface. An example C code to detect keyboard press and read keyboard input is shown in Appendix B.[9]

3.2.3 How to Use RPi GPIO for Tactor Selection and Activation

General-purpose input/output can be programmed to select and connect to the peripheral interfaces (in our case, the connected interface is the tactor). An example C code of how to access and manipulate GPIO registers is shown in Appendix C.

3.2.4 How to Wirelessly Connect 2 or More RPi's via Ad Hoc Network

In an effort to transition our research from the lab to the field (i.e., outdoor environment), we extended the capability of the RPi using a wireless connection. In addition, in an outdoor environment where a router or access point is not available, we needed to implement RPi in a wireless–ad hoc mode. The advantage of an ad hoc network is that it is quick and easy to set up. An ad hoc mode or peer-to-peer network does not require a centralized infrastructure like an access-point- or router-type network. Computers on an ad hoc network can form their own network and communicate among themselves. One disadvantage of such implementation is that the computers need to be within range of their wireless adapters. Our RPi unit, with the Wi-Fi adapter RT5370, has a range of about 100 ft within direct line of sight. If needed, our RPi units can be programmed to switch connection to the centralized Wi-Fi when an access point is available to get better and wider coverage.

3.2.4.1 Wireless Ad Hoc Mode Setup

In an ad hoc-mode network, each individual RPi unit is assigned its own static IP address, whereas in a centralized access-point network each RPi is assigned an IP address from the router through DHCP (described in Subsection 2.4). We set up a static IP address and an SSID in a shell script[10] shown below. In this example, SSID is pi_ala_mode and the static IP address is 192.168.2.1.

```
echo `pwd`
echo `ifconfig wlan0 down`
echo wlan0 down
echo `iwconfig wlan0 channel 1 essid pi_ala_mode mode
ad-hoc`
echo setting essid
echo `ifconfig wlan0 up`
echo wlan0 up
echo `ifconfig wlan0 192.168.2.2 netmask
255.255.255.0`
echo setting ip and netmask
```

Different RPi units must have different IP addresses with the same SSID; otherwise, they will not be capable of communicating with each other.

3.2.4.2 Network Communication

We used a TCP/IP client–server protocol over wireless ad hoc mode for network communication. Two Wi-Fi capable RPi units are needed for this example. One serves as a client unit sending out commands and the other is a server unit waiting and listening to receive commands. Check to make sure the SSID and IP address are set up correctly on both units. Use commands `iwconfig` and `ip addr show wlan0` to display the SSID and IP address. Try pinging with the command `ping [ip address]` to see if the packets are transmitted and received without any losses. If pinging is successful, you may proceed to execute server–client programs for wireless networking. The sequences for the server and client implementation under TCP/IP network protocol are illustrated in Fig. 10.

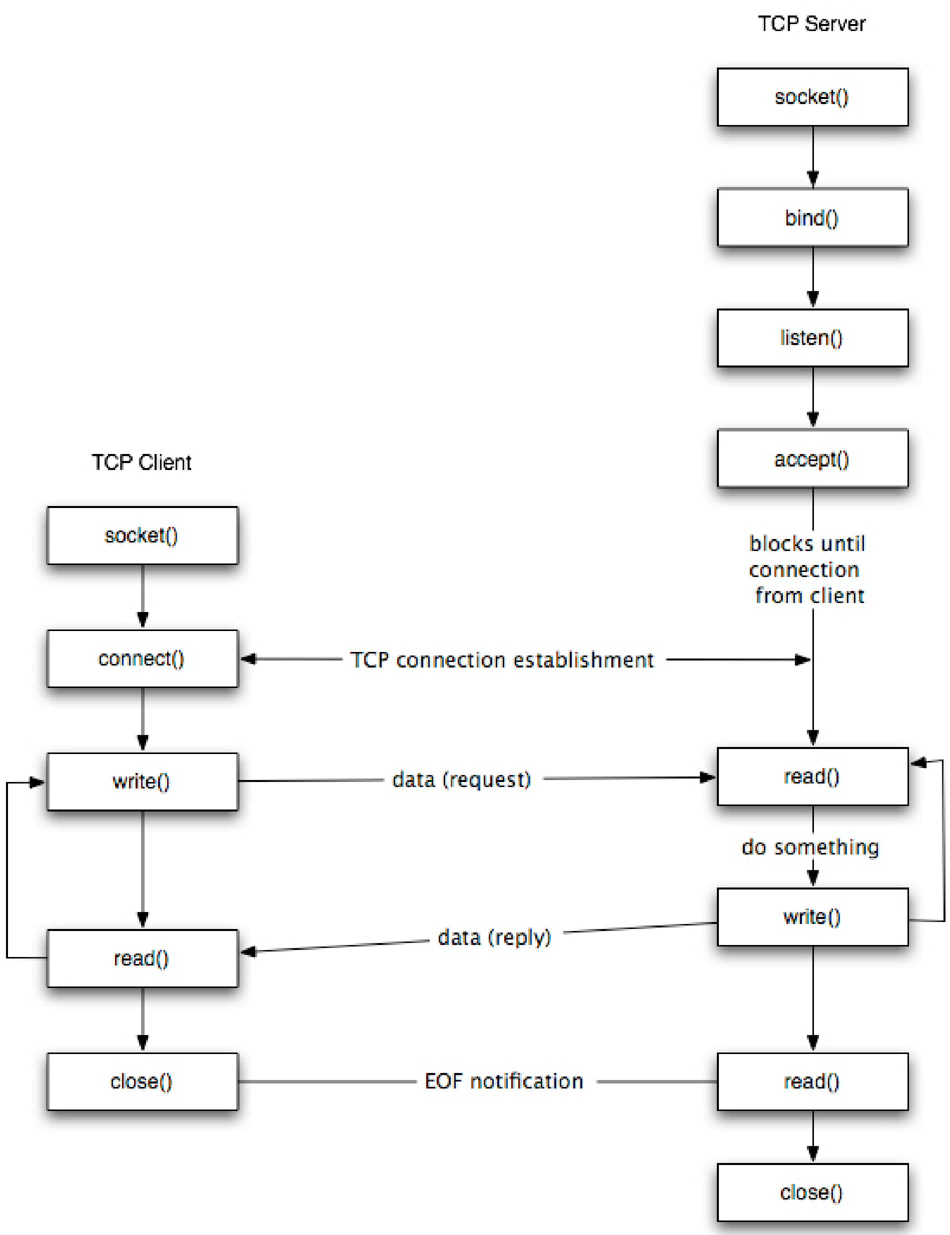

Fig. 10 TCP client–server flowchart illustrating network-protocol sequences, concluding after the client closes the socket (used with permission from Dartmouth College)[11]

The steps involved in establishing a TCP socket on the server side are as follows:

- Create a socket with the socket() function;
- Bind the socket to an address using the bind() function;
- Listen for connections with the listen() function;
- Accept a connection with the accept() function system call. This call typically blocks until a client connects with the server.
- Send and receive data by means of send() and receive().
- Close the connection by means of the close() function.

The steps for establishing a TCP socket on the client side are as follows:

- Create a socket using the socket() function;
- Connect the socket to the address of the server using the connect() function;
- Send and receive data by means of the read() and write() functions.
- Close the connection by means of the close() function.

As show in Fig. 10, the server must run first to initiate the socket and binding procedure with its specified port number. This allows the server to start listening for the client connection and communication. After the server executes its server program, the client can start its client program. At this time, the socket and binding handshake between the 2 takes place and connection is initiated. Once connected, the client can send binary characters (such as the examples in Appendixes D and E[12]). There is example code written in C for running server (server.c) and client (client.c) mode, respectively. The programs need compilation with commands `gcc server.c –o server` or `gcc client.c –o client`. To run, type `./server` on one RPi unit and `./client` on the other.

The TCP/IP network protocol is not restricted to one-to-one communication; it can be easily extended to multiple connections. For example, one client can connect and talk to a selective server or multiple servers at the same time as long as their IP addresses are distinctively assigned and known. To implement a seamless bidirectional communication between multiple units using TCP/IP, a switching capability between server (listening) and client (talking) would have to be integrated.

4. Application of the Wireless, Networked RPi HMTD

The development of the wireless, networked RPi HMTD system described in this technical note enables us to study head-mounted tactile displays as an alternative communication modality to maintain a high level of situation awareness while unburdening cognitive load. We have completed 2 studies using Wi-Fi RPi HMTD: 1) comparison of computer-simulated city navigation via tactile stimulation and visual guide, and 2) evaluation of the effects of head-tactile stimulation on shooting performance. In the first study, the goal was to use tactile stimulation on the head as a navigational tool to replace a visual guided display in a simulated environment. We calculated the angle and distance between the avatar and the target, then communicated that information to the RPi HMTD system via Wi-Fi network. The RPi HMTD responded and stimulated a tactor on the head in the direction of the target. The second study evaluated the effects of the head-tactile display on shooting performance. The head tactor was stimulated just a few seconds after the target popped up and before the shooter fired his weapon. We were able to use one RPi to pick up the firing range's target-up signal and wirelessly send the signal to stimulate a tactor on another RPi-controlled HMTD worn by the shooter.

5. Summary

In this technical note, the development and application of a wireless and portable Raspberry Pi-controlled HMTD were discussed. A how-to guide for each hardware and software component needed to implement the HMTD was also provided. Though the system is a working prototype, it is a capable tool that enables various research studies in using the skin as a novel sensory modality for communication. The RPi can do more than controlling tactors and can be extended to include a number of peripheral interfaces such as audio recording and playback with a USB headset (recommended: Plantronics Audio 478 USB Stereo Headset), video camera recording and screen display, and Global Positioning System. Such features will allow a more versatile wearable technology.

6. References

1. Kalb JT, Amrein BE, Myles K. Instrumentation and tactor considerations for a head-mounted tactile display. Aberdeen Proving Ground (MD): Army Research Laboratory (US); 2008 Sep. Report No.: ARL-MR-705.

2. Raspberry Pi Model B 512MB RAM. [accessed 2016 Jul 7]. https://adafruit.com /products /1914.

3. Sjogelid S. Raspberry Pi for secret agents. Birmingham (UK): Packt Publishing; 2013.

4. Unix Useful Commands. [accessed 2016 Jul 7]. http://www.tutorialspoint.com/unix/unix-useful-commands.htm.

5. C-2 Tactor. [accessed 2016 Jul 7]. http://bdml.stanford.edu.

6. Stereo 2.1W Class D Audio Amplifier–TPA2012. [accessed 2016 Jul 7]. http://www.adafruit.com/products/1552.

7. Lithium Ion Polymer Battery–3.7v 2500mAh. [accessed Jul 7]. https://www.adafruit.com/products/328.

8. PowerBoost 1000 Charger–Rechargeable 5V Lipo USB Boost @ 1A. [accessed 2016 Jul 7]. https://www.adafruit.com/products/2465.

9. Ubuntu Forums. Detect arrow keys using termios.h. [accessed 2016 Sep 23]. https://ubuntuforums.org/showthread.php?t=2276177.

10. Stack Exchange. Raspberry Pi ad-hoc networking. [accessed 2016 Jul 7]. http://stackoverflow.com/questions/15423325/raspberry-pi-ad-hoc-networking.

11. CS 60 computer networks, socket programming. Hanover (NH): Dartmouth College, Department of Computer Science. [accessed 2016 July 7]. http://cs.dartmouth.edu/~campbell/cs60/socketprogramming.html.

12. Binary Tides. Server and client example with C sockets on Linux. [accessed 2016 Jul 7]. http://www.binarytides.com/server-client-example-c-sockets-linux/.

INTENTIONALLY LEFT BLANK.

Appendix A. C Code to Generate Morse-Code Modulated Carrier Tones in WAV Format

This appendix appears in its original form, without editorial change.

```
/* compile with gcc makwav9.c -o makwav9 -lm  */
/* the lm flag will link the math library  */
#include <stdio.h>
#include <libusb-1.0/libusb.h>
#include <stdlib.h>
#include <math.h>
#include <string.h>
#include <inttypes.h>

void dot(void);
void dash(void);
void space(void);
void setup_tactors(void);
void shutdown_tactors(void);
void send_cmd(char *d, int n);
void set_gain(char gain);   // 0: 0x00  1: 0x40  2: 0x80  3: 0xc0, gain 0..3
uses most significant two bits
void set_tactors(char tbm); // tbm: tactor bit map 1: 0x1 2:0x2 3:0x4 4:0x8
5:0x10 6:0x20 7:0x40 8:0x80

/* define global variables before function main, local variables
   are defined within main function */
struct wavfile_header {
  char      ChunkID[4];
  int ChunkSize;
  char      Format[4];
  char      Subchunk1ID[4];
  int Subchunk1Size;
  short     AudioFormat;
  short     NumChannels;
  int SampleRate;
  int ByteRate;
  short     BlockAlign;
  short     BitsPerSample;
  char      Subchunk2ID[4];
  int Subchunk2Size;
};
int i,j,i1,i2,i3,i4,i5,amp;
int sample_rate;
double ph,ph1,frequency,c,s,c1,s1,c2,s2,c3,s3,t;
short *waveform;
char str1[80],str2[80];  //allocate space to hold combined strings in system
call
libusb_device *dev;
struct libusb_device_handle *devh = NULL;
int configuration = 1;
int interface = 1;
int r,rr,rw,n,num_written,num_read;
char chk;
char e[63];

int main(int argc, char *argv[]){
  if (argc !=3) {
    printf("\nUsage: %s  1/3_oct_band_no  pulse_type \n",argv[0]);
    printf("for example: %s 15 6.\n",argv[0]);
    printf("for example: %s 24 5.\n",argv[0]);
```

```
  }
  setup_tactors();
  set_gain(0xc0);
  set_tactors(0x1);
  FILE *fp;  /* declare pointer to type FILE  */
  int band_no = atoi(argv[1]);
  int pulse_no = atoi(argv[2]);
  frequency=exp(log(10)*band_no/10); //round to nearest 0.1 Hz
  printf("frequency = %f .\n",frequency);
  char *filename;
  printf("pulse_no = %d.\n",pulse_no);
  strcpy(str1,"temp_");
  if (pulse_no==1)
    strcat(str1,"CQ.wav");
  else if (pulse_no==2)
    strcat(str1,"HI.wav");
  else if (pulse_no==3)
    strcat(str1,"SOS.wav");
  else if (pulse_no==4)
    strcat(str1,"ESEEE.wav");
  else if (pulse_no==5)
    strcat(str1,"short.wav");
  else if (pulse_no==6)
    strcat(str1,"long.wav");
  printf("str1 = %s, sizeof(str1) = %d.\n",str1,sizeof(str1));
  filename = str1;  //  filename = argv[3]; /* sound.wav */
  sample_rate = 22050;
  amp = 32000;
  float dot_on_time = 0.12; /*0.06  0.24 = 5wpm */
  float dash_on_time = 3 * dot_on_time;
  float rise_fall_time = 0.1 * dot_on_time;
  float dot_sustain_time = dot_on_time  - 2 * rise_fall_time;
  float dash_sustain_time = dash_on_time -2 * rise_fall_time;
  float off_time = dot_on_time;
  float dot_time = dot_on_time + off_time;
  float dash_time = dash_on_time + off_time;
  float off_time2 = 2 * off_time;        /* adds to off_time to give 3 *
off_time between characters */
  int i6,i7,jj,kk;

  i1=floor(0.0+rise_fall_time * sample_rate);
  i2=floor(0.0+dot_sustain_time * sample_rate);
  i3=floor(0.0+dash_sustain_time * sample_rate);
  i4=floor(0.0+off_time * sample_rate);
  i5=floor(0.0 + off_time2 * sample_rate);
  i6=floor(0.0 + dot_time * sample_rate);
  i7=floor(0.0 + dash_time * sample_rate);

  ph = 2 * M_PI * frequency / sample_rate; c1=cos(ph); s1=sin(ph);
  ph1 = M_PI / (2 * i1); c3=cos(ph1); s3=sin(ph1);
  j=0;

  int num_samples;
//  printf("Please select the value you want\n");
//  scanf("%d", &number);
//  number=argv[2];
  if(pulse_no==1) {
```

```
    num_samples = 3*i6+5*i7+1*i5; // CQ
    waveform = (short *) malloc(num_samples * sizeof(short));
    dash(); dot(); dash(); dot(); space(); dash(); dash(); dot(); dash();
  } else if (pulse_no==2) {
    num_samples = 6*i6+0*i7+1*i5; // HI
    waveform = (short *) malloc(num_samples * sizeof(short));
    dot(); dot(); dot(); dot(); space(); dot(); dot();
  } else if (pulse_no==3) {
    num_samples = 6*i6+3*i7;    // SOS
    waveform = (short *) malloc(num_samples * sizeof(short));
    dot(); dot(); dot(); dash(); dash(); dash(); dot(); dot(); dot();
  } if(pulse_no==4) {
    num_samples = 7*i6+0*i7+5*i5; // ESEEE
    waveform = (short *) malloc(num_samples * sizeof(short));
    dot(); space(); dot(); dot(); dot(); space(); dot(); space(); space();
dot(); space(); dot();
  } else if (pulse_no==5) {
    num_samples = 3*i6+0*i7;    // Short Tap Tap Tap
    waveform = (short *) malloc(num_samples * sizeof(short));
    dot(); dot(); dot();
  } else if (pulse_no==6) {
    num_samples = 0*i6+3*i7+2*i5;    // Long Tap Tap Tap
    waveform = (short *) malloc(num_samples * sizeof(short));
    dash(); space(); dash(); space(); dash();
  }

  short num_channels = 1;    /* 1: mono, 2: stereo */
  short bits_per_sample = 16; /* make a mono 16-bit WAV file */
  int data_bytes = num_samples * num_channels * bits_per_sample / 8; /* bytes
of data */
  int chunk_size = 36 + data_bytes;  /* size of rest of chunk following this
number */
                                     /* also size of entire file - 8 bytes
*/
  struct wavfile_header header;
  strncpy(header.ChunkID,"RIFF",4);     /* at 0 */
  header.ChunkSize = chunk_size;        /* at 4 */
  strncpy(header.Format,"WAVE",4);      /* at 8 */
  strncpy(header.Subchunk1ID,"fmt ",4);/* at 12 */
  header.Subchunk1Size = 16;            /* at 16, rest of subchunk follows
this number */
  header.AudioFormat = 1;               /* at 20, PCM mode, linear
quantization */
  header.NumChannels = num_channels;    /* at 22 */
  header.SampleRate = sample_rate;      /* at 24 */
  header.ByteRate = sample_rate * num_channels * bits_per_sample / 8; /* at
28 */
  header.BlockAlign = num_channels * bits_per_sample / 8; /* at 32  */
  header.BitsPerSample = bits_per_sample; /* at 34  */
  strncpy(header.Subchunk2ID,"data",4);  /* at 36  */
  header.Subchunk2Size = data_bytes;     /* at 40, number bytes in data, size
of read  */
                                         /* of the subchunk following this
number  */
                                         /* at 44 start of sound data
(left,right order stereo) */
```

```
  /* create instance of the FILE structure and returns a pointer to that
structure */
  fp = fopen(filename,"wb");  /* opens file in binary mode for writing to new
or over old file */
  fwrite(&header,sizeof(header),1,fp); /* writes block of data from memory to
binary-mode file */
  fwrite(waveform,sizeof(short),num_samples,fp); /*  writes waveform array as
a single "element" */
  fclose(fp); /* close file, flush buffer  */
  free(waveform);
  strcpy(str2,"aplay ");
  strcat(str2,filename);
  printf("str = %s, sizeof(str2) = %d.\n",str2,sizeof(str2));
  system(str2);
  shutdown_tactors();
  return 0;
}

void dot(void) {
  c=1; s=0;
  c2=1; s2=0;
  for(i=0;i<i1;i++,j++) { /* dot rise */
    waveform[j]=amp*s*s2*s2;
    t=c*c1-s*s1;
    s=c*s1+s*c1;
    c=t;
    t=c2*c3-s2*s3;
    s2=c2*s3+s2*c3;
    c2=t;
  }
  for(i=0;i<i2;i++,j++) { /* dot sustain */
    waveform[j]=amp*s;
    t=c*c1-s*s1;
    s=c*s1+s*c1;
    c=t;
  }
  c2=1; s2=0;
  for(i=0;i<i1;i++,j++) { /* dot fall */
    waveform[j]=amp*s*c2*c2;
    t=c*c1-s*s1;
    s=c*s1+s*c1;
    c=t;
    t=c2*c3-s2*s3;
    s2=c2*s3+s2*c3;
    c2=t;
  }
  for(i=0;i<i4;i++,j++) { /* off after dot */
    waveform[j]=0;
  }
}

void dash(void) {
  c=1; s=0;
  c2=1; s2=0;
  for(i=0;i<i1;i++,j++) { /* dot rise */
    waveform[j]=amp*s*s2*s2;
    t=c*c1-s*s1;
```

```
    s=c*s1+s*c1;
    c=t;
    t=c2*c3-s2*s3;
    s2=c2*s3+s2*c3;
    c2=t;
  }
  for(i=0;i<i3;i++,j++) { /* dot sustain */
    waveform[j]=amp*s;
    t=c*c1-s*s1;
    s=c*s1+s*c1;
    c=t;
  }
  c2=1; s2=0;
  for(i=0;i<i1;i++,j++) { /* dot fall */
    waveform[j]=amp*s*c2*c2;
    t=c*c1-s*s1;
    s=c*s1+s*c1;
    c=t;
    t=c2*c3-s2*s3;
    s2=c2*s3+s2*c3;
    c2=t;
  }
  for(i=0;i<i4;i++,j++) { /* off after dot */
    waveform[j]=0;
  }
}

void space(void) {
  for(i=0;i<i5;i++,j++) { /* off after character */
    waveform[j]=0;
  }
}
```

Appendix B. C Code to Read Keyboard Input

This appendix appears in its original form, without editorial change.

```
#include <termios.h>
#include <unistd.h>
#include <stdio.h>

Int getch(int ms);
Int main(void){
Int x;
Do {
If ((x = getch(500))){
If (48<=x && x<=57)
X=x-48;
Else if (65<=x && x<=90)
X=x-55;
Else if (97<=x && x<=122)
X=x-87;
Else
X=0;
Print("Got it: '%d', '%c'\n",x,x);
} else {
Printf("Not yet!\n");
}
While (x != 'q');
Return 0;
}

Int getch(int ms) {
  Int ret;
  Struct termio oldt, newt;
  Struct pollfd pfds[1];

Tcgetattr(STDIN_FILENO,&oldt);
Newt=oldt;
Newt.c_lflag &=~(ICANON | ECHO);
Tcsetattr(STDIN_FILENO, TCSANOW, &newt);
Pfds[0].fd=STDIN_FILENO;
Pfds[0].events=POLLIN;
Poll(pfds,1,ms);
If (pfds[0].revents&POLLIN){
  Char ch;
  Read(STDIN_FILENO,&ch,1);
  Ret=ch;
} else {
Ret=0;
}
tcsetattr(STDIN_FILENO,TCSANOW,&oldt);
return ret;
}
```

Appendix C. C Code to Access and Enable GPIO Pins Written by Gert van Loo and Dom (elinux.org/RPi_GPIO_Code _Samples#pigpio)

This appendix appears in its original form, without editorial change.

```
#define BCM2708_PERI_BASE        0x20000000
#define GPIO_BASE                (BCM2708_PERI_BASE + 0x200000) /* GPIO
controller */

#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <sys/mman.h>
#include <unistd.h>

#define PAGE_SIZE (4*1024)
#define BLOCK_SIZE (4*1024)

int  mem_fd;
void *gpio_map;

// I/O access
volatile unsigned *gpio;


// GPIO setup macros. Always use INP_GPIO(x) before using OUT_GPIO(x) or
SET_GPIO_ALT(x,y)
#define INP_GPIO(g) *(gpio+((g)/10)) &= ~(7<<(((g)%10)*3))
#define OUT_GPIO(g) *(gpio+((g)/10)) |=  (1<<(((g)%10)*3))
#define SET_GPIO_ALT(g,a) *(gpio+(((g)/10))) |=
(((a)<=3?(a)+4:(a)==4?3:2)<<(((g)%10)*3))

#define GPIO_SET *(gpio+7)  // sets   bits which are 1 ignores bits which are
0
#define GPIO_CLR *(gpio+10) // clears bits which are 1 ignores bits which are
0

#define GET_GPIO(g) (*(gpio+13)&(1<<g)) // 0 if LOW, (1<<g) if HIGH

#define GPIO_PULL *(gpio+37) // Pull up/pull down
#define GPIO_PULLCLK0 *(gpio+38) // Pull up/pull down clock
```

Appendix D. C Code to Implement a Server Mode (from BinaryTides.com)

This appendix appears in its original form, without editorial change.

```
#include<stdio.h>
#include<string.h>    //strlen
#include<sys/socket.h>
#include<arpa/inet.h> //inet_addr
#include<unistd.h>    //write

int main(int argc , char *argv[])
{
    int socket_desc , client_sock , c , read_size;
    struct sockaddr_in server , client;
    char client_message[2000];

    //Create socket
    socket_desc = socket(AF_INET , SOCK_STREAM , 0);
    if (socket_desc == -1)
    {
        printf("Could not create socket");
    }
    puts("Socket created");

    //Prepare the sockaddr_in structure
    server.sin_family = AF_INET;
    server.sin_addr.s_addr = INADDR_ANY;
    server.sin_port = htons( 8888 );

    //Bind
    if( bind(socket_desc,(struct sockaddr *)&server , sizeof(server)) < 0)
    {
        //print the error message
        perror("bind failed. Error");
        return 1;
    }
    puts("bind done");

    //Listen
    listen(socket_desc , 3);

    //Accept and incoming connection
    puts("Waiting for incoming connections...");
    c = sizeof(struct sockaddr_in);

    //accept connection from an incoming client
    client_sock = accept(socket_desc, (struct sockaddr *)&client,
(socklen_t*)&c);
    if (client_sock < 0)
    {
        perror("accept failed");
        return 1;
    }
    puts("Connection accepted");

    //Receive a message from client
    while( (read_size = recv(client_sock , client_message , 2000 , 0)) > 0 )
    {
        //Send the message back to client
```

```
        write(client_sock , client_message , strlen(client_message));
    }

    if(read_size == 0)
    {
        puts("Client disconnected");
        fflush(stdout);
    }
    else if(read_size == -1)
    {
        perror("recv failed");
    }

    return 0;
}
```

INTENTIONALLY LEFT BLANK.

Appendix E. C Code to Implement a Client Mode (from BinaryTides.com)

This appendix appears in its original form, without editorial change.

```
#include<stdio.h> //printf
#include<string.h>    //strlen
#include<sys/socket.h>    //socket
#include<arpa/inet.h> //inet_addr

int main(int argc , char *argv[])
{
    int sock;
    struct sockaddr_in server;
    char message[1000] , server_reply[2000];

    //Create socket
    sock = socket(AF_INET , SOCK_STREAM , 0);
    if (sock == -1)
    {
        printf("Could not create socket");
    }
    puts("Socket created");

       //IP address of the server
    server.sin_addr.s_addr = inet_addr("192.168.2.1 ");
    server.sin_family = AF_INET;
    server.sin_port = htons( 8888 );

    //Connect to remote server
    if (connect(sock , (struct sockaddr *)&server , sizeof(server)) < 0)
    {
        perror("connect failed. Error");
        return 1;
    }

    puts("Connected\n");

    //keep communicating with server
    while(1)
    {
        printf("Enter message : ");
        scanf("%s" , message);

        //Send some data
        if( send(sock , message , strlen(message) , 0) < 0)
        {
            puts("Send failed");
            return 1;
        }

        //Receive a reply from the server
        if( recv(sock , server_reply , 2000 , 0) < 0)
        {
            puts("recv failed");
            break;
        }

        puts("Server reply :");
        puts(server_reply);
```

```
    }

    close(sock);
    return 0;
}
```

List of Symbols, Abbreviations, and Acronyms

CPU	central processing unit
DHCP	Dynamic Host Configuration Protocol
EAI	Engineering Acoustics, Inc.
ESSID	Extended Service Set Identifier
GPIO	General Purpose Input/Output
GUI	graphical user interface
HMTD	head-mounted tactile display
IP	Internet Protocol
LAN	local area network
LCD	liquid crystal display
lipo	lithium-ion polymer
OS	operating system
RPi	Raspberry Pi
SD	storage device
SoC	system on chip
SSH	secure shell
SSID	Service Set Identifier
TCP	Transmission Control Protocol
USB	universal serial bus
WAV	Waveform Audio

1 DEFENSE TECHNICAL
(PDF) INFORMATION CTR
DTIC OCA

2 DIRECTOR
(PDF) US ARMY RESEARCH LAB
RDRL CIO L
IMAL HRA MAIL & RECORDS
MGMT

1 GOVT PRINTG OFC
(PDF) A MALHOTRA

1 DIR USARL
(PDF) RDRL HRF C
D CHHAN