Chapter 1

Sensor-enabled smart suit electronic IoT design platform with emergency services application

Migdat Hodzic[1], James M. Brennan[2] and Enis Dzanic[3]

An integrated smart suit sensor and positioning system electronic Internet of Things (IoT) prototype has been developed to address the growing need for personal welfare monitoring of first-line responders, defenders, and workers exposed to industrial or other hazards, as well as other commercial and defense, and new applications in cloud-based IoT. The system provides a global positioning system (GPS) position map with coordinate data, current Greenwich mean time (GMT) readout, subject's heart rate, body temperature, and a long-wave thermal video camera that provides a forward-looking thermal image. Physiologic data and thermal imaging of the subject may be viewed by monitoring personnel using Internet browser connected to the system's static Internet protocol (IP) address. The system is Wi-Fi connected to a local network, which can be extended to enable secure connection to the Internet with incorporation of additional firmware. Details regarding hardware and software configuration are presented along with an appendix containing additional data. Source code for the software modules currently running on the prototype system is also available for interested parties or potential users and customers.

1.1 Introduction

Recent years witnessed very considerable development in the areas of various sensors for many related applications. In this context new IoT (Internet of Things) technologies [1], in particular Cloud-based IoT [2,3], emerged as a response to a growing need to connect a variety of devices in our homes, in the streets, cities, or sensor-enabled devices which attach to our body or uniforms. New low power wireless and wired sensor technologies have been developed and are used more and more in many old and lots of new applications [4]. Sensors range from

[1]Engineering Department, American University in Bosnia and Herzegovina, Sarajevo, Bosnia and Herzegovina
[2]BH Analytics, Santa Clara, California, USA and Sarajevo, Bosnia and Herzegovina
[3]Economics Department, University of Bihac, Bihac, Bosnia and Herzegovina

environmental, physiologic, range measurements, proximity sensors, and all the way to very sophisticated special-purpose sensors for industrial and defense use [5–7]. A variety of embedded and inexpensive platforms exist now for fast design and prototyping such as Raspberry Pi [8]. Besides new sensors and IoT technologies, new "smart" materials are also becoming available more and more, with a variety of functionality built-in, even some basic electronics built into the material fabric [9]. One specific and important area of sensor development is infrared (IR) and thermal sensors and cameras based on them [10]. These sensors now allow for very detailed IR or thermal image sensing and digital framing with a usable number of video frames [10]. In any useful IoT sensor-enabled smart suite, there is typically a need for positioning data and hence GPS sensors are also required. In the smart suite case, it would be required to have physiological data of the person wearing the suit, and this means a need for temperature, heart rate, blood pressure, outside environment pressure, humidity and temperature, and sensor for some dangerous gas presence (such as in mines). It is in this spirit that we developed simple IoT sensor-based smart suit design and testing electronic platform described in this chapter. We opted to incorporate only basic sensor components such as GPS with the antenna, temperature, and heart rate sensors, as well as thermal sensors. The platform is based on popular Raspberry Pi HW and SW computer board, with Wi-Fi and 4G built-in for Internet communications. In order to be able to demonstrate a smart suit design platform, we also incorporated ability to view the suit online using its static IP address. Wi-Fi is used for Internet connectivity. A variety of customizations are possible depending on the interest of final users and customers and their needs for specific sensors.

1.2 System components

A general system block diagram of the smart suit system electronics is shown in Figure 1.1. More specific component choices for our prototype and demonstration design are indicated in Figure 1.2. The thermal camera (FLIR Lepton brand) is connected to the host processor via SPI0 bus. The subject's heartbeat and surface body temperature sensors are connected to the host processor through the analog to digital converter (ADC) via SPI1 bus and processed by an algorithm to extract heart rate and average temperature. Additional inputs to the ADC are available for future expansion capabilities to provide respiration rate and activity monitoring. The host processor for the smart suit system is a Raspberry Pi 3 running a Debian Stretch Linux distribution. When the system powers up, it automatically starts a C++-based thermal imaging application as well as a Python-based Flask web server, which also inputs and formats all incoming data for presentation as a served webpage.

1.2.1 FLIR Lepton IR camera

The FLIR Lepton is an infrared camera system that integrates a fixed-focus lens assembly, an 80×60 long-wave infrared (LWIR) microbolometer sensor array, and incorporates signal processing electronics [10]. Easy to integrate and operate,

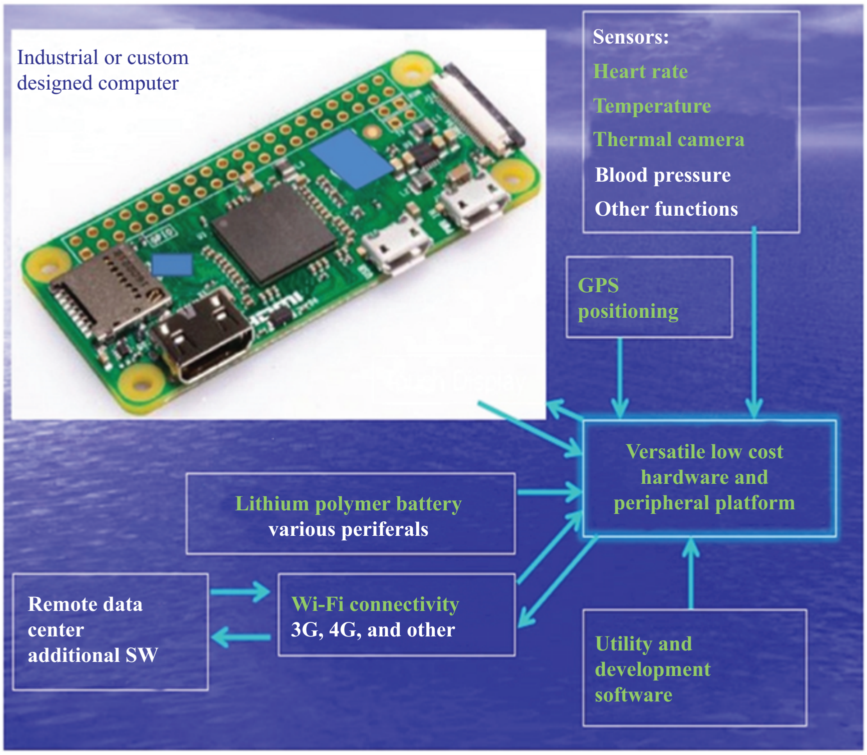

Industrial or custom designed computer
Sensors:
Heart rate
Temperature
Thermal camera
Blood pressure
Other functions
GPS positioning
Versatile low cost hardware and peripheral platform
Lithium polymer battery various periferals
Remote data center additional SW
Wi-Fi connectivity 3G, 4G, and other
Utility and development software

Figure 1.1 Smart suit system general electronic block diagram

Lepton is intended for mobile devices as well as any other application requiring a very small footprint, very low power, and instant-on operation. Lepton can be operated in its default mode or configured into other modes through a command and control interface (CCI). Using its default conditions, the FLIR Lepton camera outputs 60 video packets per frame, each 1,312 bits long, at approximately 25.9 frames per second. The minimum output data clocking rate to the camera is on the order of 2 MHz to allow it to keep up with its real-time image generation. The camera data output provides three repeated identical frames in a row, followed by a new frame making up another series of three frames. This activity will repeat indefinitely if not interrupted or an error has happened. It should be noted that this frame series format means that real-time new frame output is approximately at a 9 Hz rate so that actual frame processing only operates using only one out of three frames. The smart suit system processes thermal frames at about nine per second but, at present, only grabs one still frame per web page update. Software processing of the thermal frames includes false-color map encoding that normalizes the brightest pixels to the most intense coloring. This ensures that the image does not saturate thus masking less bright features of the scene.

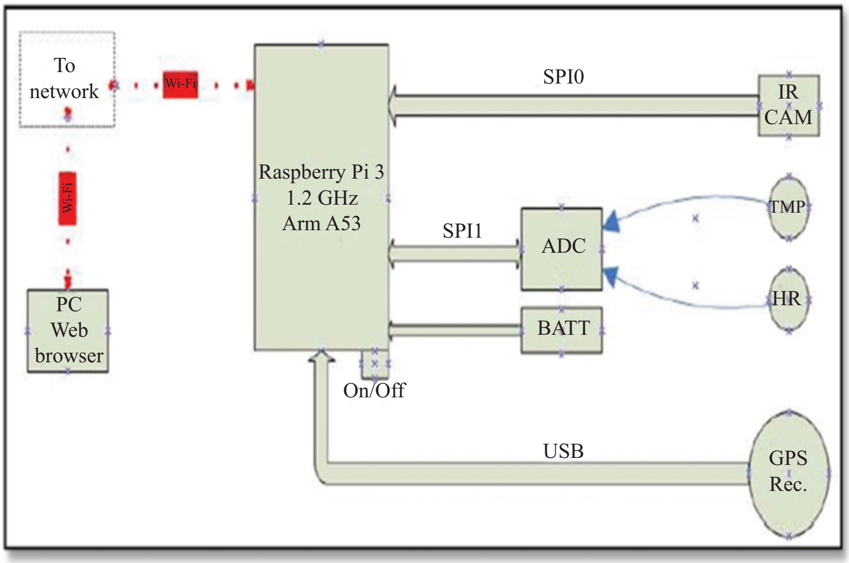


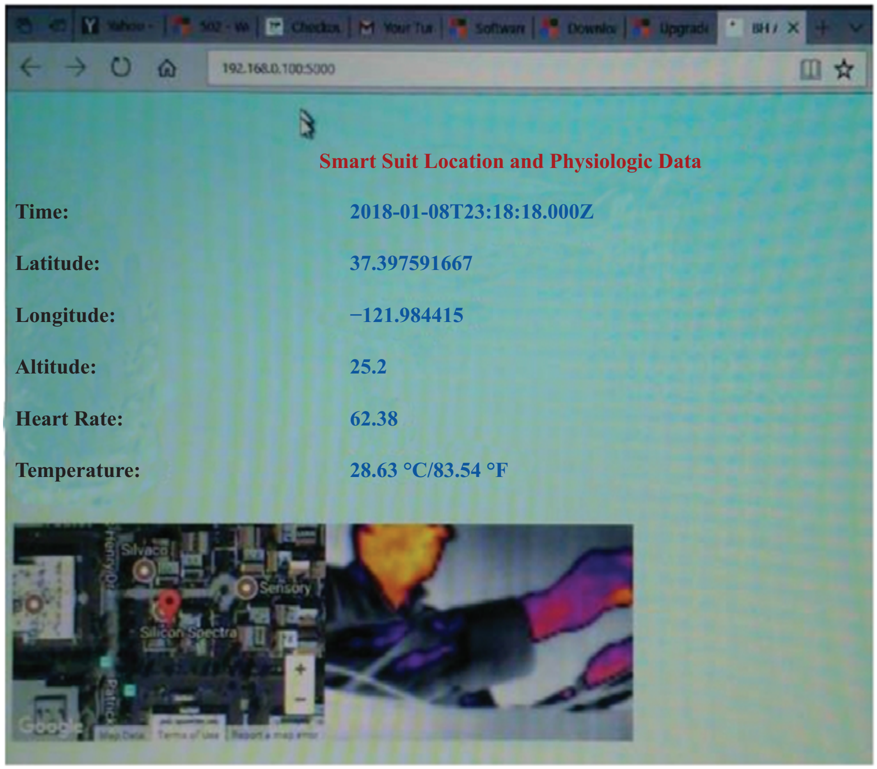
Figure 1.2 Smart suit system-specific block diagram

1.2.2 IR camera software

Image processing is done in the C++ language and includes algorithms to identify and synchronize processing starting with frame packet number one, formatting and processing each FLIR Lepton data line input into a 480 × 800 pixel image frame, color mapping the brightness of pixels, and then converting the frame into a JPG format for storage in memory. The frame stored in memory is accessed by the Python-based web server application approximately once every 2–3 s for output to the client display for the demonstration purposes.

1.2.3 Python-based flask web server

A Flask micro web server has been implemented to serve a web page containing human subject physiologic data, current GPS coordinate data, an up-to-date GPS location map, and the IR thermal image showing a color mapped scene ahead of the subject. This server is implemented in Python, which generates a new web page when a client application asks for an update (currently once every 2–3 s). Processing includes capturing GPS time and coordinates, acquiring subject's heart rate and external body temperature, computing and formatting all of these values into a python dictionary, and then sending the dictionary (data strings) to the client webpage view. The client webpage presently uses a timed refresh period to ask for an update from the server every 2–3 s. The data are collected in Java variables within the page and processed to form result text, a google API map of the location, and to render the thermal JPG image seen in Figure 1.3 (US Silicon Valley



Smart Suit Location and Physiologic Data

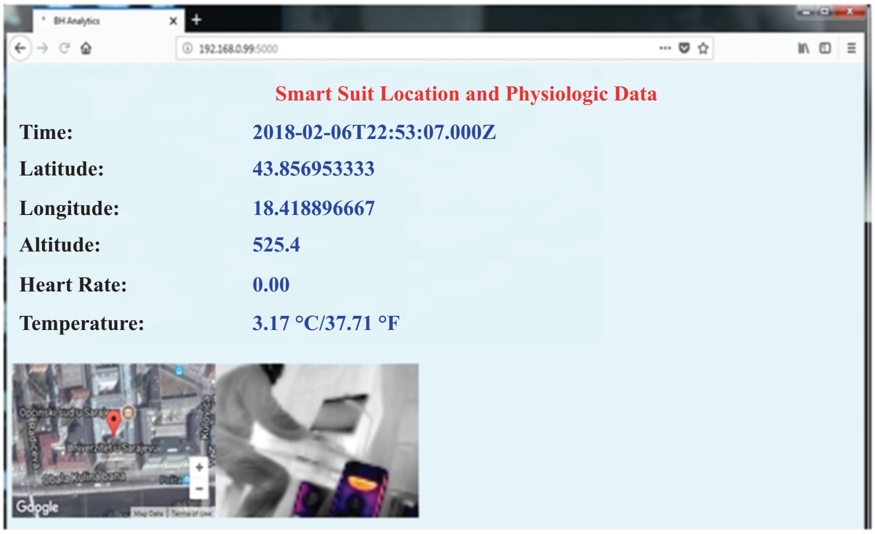
Time:	2018-01-08T23:18:18.000Z
Latitude:	37.397591667
Longitude:	−121.984415
Altitude:	25.2
Heart Rate:	62.38
Temperature:	28.63 °C/83.54 °F

Figure 1.3 Client web page displaying collected data, map, and image located in Silicon Valley, USA

location) and Figure 1.4 (Sarajevo, Bosnia and Herzegovina location). At this point, we did not spend lots of time in making graphical user interface (GUI) more sophisticated. As Figures 1.3–1.5 indicate the GUI is just a basic one for the suit system demonstration and prototyping purposes. Figure 1.6 shows an additional thermal image of a person in front of the Lepton thermal image sensor built into the suit.

1.2.4 Raspberry Pi 3 Debian stretch operating system start

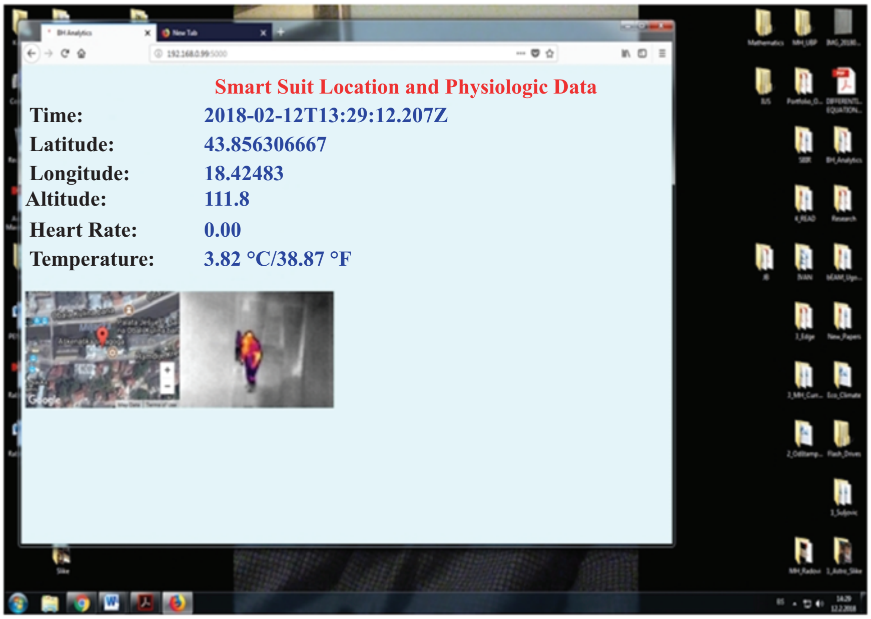
The smart suit system connection diagram shown in Figure 1.7 relies upon the underlying Linux operating system to host the thermal and Flask web server applications. At power-on, the system automatically starts both the Flask server and the Lepton Thermal Imager applications. The Flask web server startup process uses the CHRON daemon to read a script at startup. This is initially setup during development using the CRONTAB command and is done from within the directory **/var/spool/cron/crontabs/pi**. See Appendix A for the general procedure. During startup, the operating system will look for shell scripts in the user’s home directory,



Smart Suit Location and Physiologic Data

Time:	2018-02-06T22:53:07.000Z
Latitude:	43.856953333
Longitude:	18.418896667
Altitude:	525.4
Heart Rate:	0.00
Temperature:	3.17 °C/37.71 °F

Figure 1.4 Client web page displaying collected data, map, and image located in Silicon Valley, USA



Smart Suit Location and Physiologic Data

Time:	2018-02-12T13:29:12.207Z
Latitude:	43.856306667
Longitude:	18.42483
Altitude:	111.8
Heart Rate:	0.00
Temperature:	3.82 °C/38.87 °F

Figure 1.5 Client web page displaying collected data, map, and image located in Ilidza TRZ factory, B&H

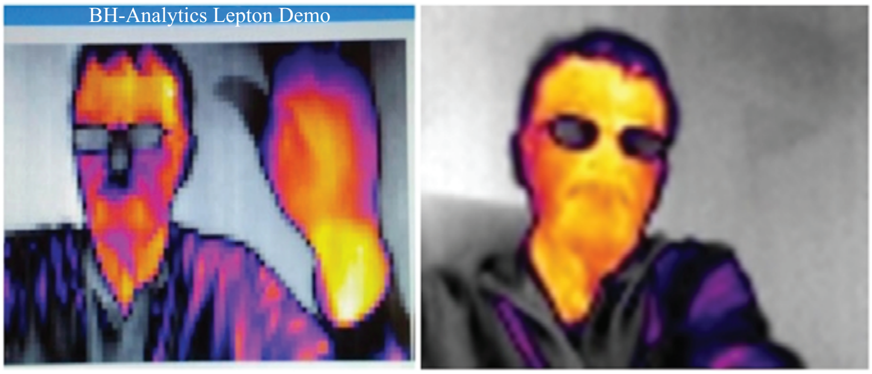


Figure 1.6 Thermal images of a person in front of Lepton thermal camera

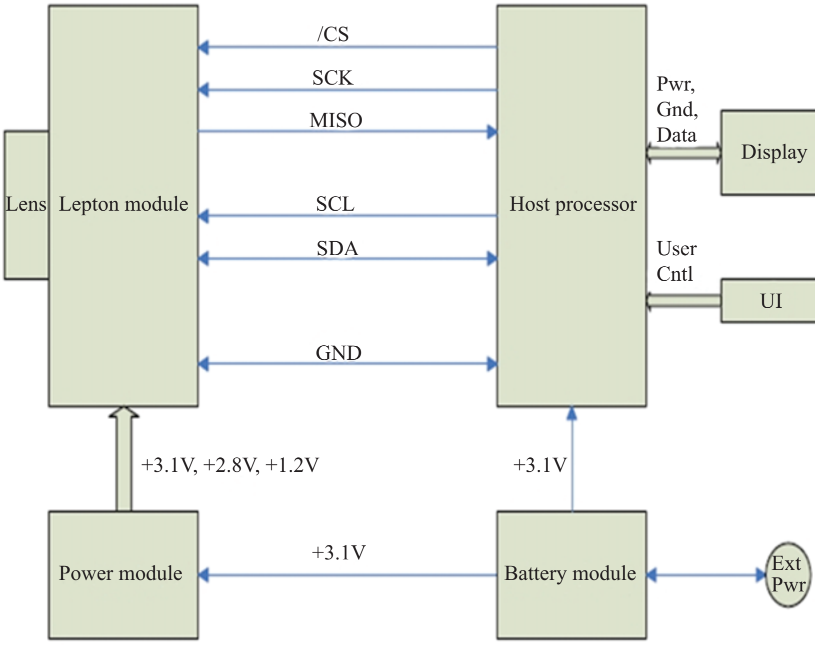


Figure 1.7 Smart suit system connection block diagram

in this case **/home/pi/startup.sh**. The script found at this location is used to start up the Lepton thermal imaging application. Content of startup.sh can be obtained if a customer requires this. It should be noted that the Lepton thermal imaging application named “Demo,” as found in directory **/home/pi/Qt/Demo/build-Demo-kit2-Debug/**, is a stand-alone executable and has been built using the Qt4 development application. The setup of the Qt4 environment is referenced in smart suit system documentation. Another script that is automatically executed during the startup process allows system shutdown when a specific hardware pin is grounded. This script is found at **/etc/init.d/listen-for-shutdown.sh** and runs a Python script found at **/usr/local/bin/listenfor-shutdown.py**, which when invoked runs continuously in the background waiting for a hardware pin instituted shutdown interrupt signal. Once the system starts up and the required applications are each separately running, the Flask web server is ready to respond to a client application request if the Wi-Fi feature is operational.

1.3 System hardware

The connection diagram in Figure 1.6 shows the Raspberry Pi 3 I/O ports and general pins used to interface system hardware components. Numerous connectors are used to route sensor data to the required I/O pins. Depending on the number of required sensors, the I/O pins functionality can be customized for a specific application. The system has enough capacity to accommodate a number of additional sensors if required.

1.3.1 Hardware data collection and transfer

The Raspberry Pi 3 (RP3) uses 2 separate SPI busses to transfer data: (1) SPI0 to clock in data from the MCP3008 ADC and (2) SPI1 to clock in data from the FLIR Lepton thermal imaging module. This is done to provide the different clock speeds required by each device. The MPC3008 ADC uses a 200 KHz transfer clock to provide the lower demand heart rate and temperature measurements, which are collected at ten samples per second.

The FLIR Lepton thermal module, however, requires a minimum transfer clock of 2 MHz to provide its 27 fps image rate. The thermal module operates in an autonomous manner in its present configuration after it is powered on by providing serial data in an open-loop fashion when given a clock. No other programming is required to access its data via SPI serial bus. At present, a momentary power interrupt switch is placed in the thermal module's Vdd input to allow for operator reset of the device due to over temperature or other uncontrolled noisy conditions. A later version of the smart suit system will use a dedicated I/O control bit to periodically reset and resynchronize the module to prevent unexpected loss of sync. The MPC3008 requires a lower clock speed due to its internal analog-to-digital converter electronics conversion of the input signal into a 12-bit digital value. Two analog channels are currently used with the heart rate monitor requiring ten acquisitions per second, with temperature acquisition being converted at the same time for convenience as these parameters do not change rapidly. GPS information is input to the RP3 via a USB connection and provides NMEA standard 0183 output using a simple serial protocol. The GPS module used is an Adafruit Version 3 Ultimate Breakout, with a -165 dBm input sensitivity and capable of receiving 66 channels with 10 Hz updates. It is possible to connect an external active antenna to its input to increase its sensitivity to in excess of −185 dBm when the environment provides weak signal conditions. GPS coordinate information is provided to a Python program running in the background and feeding the Flask web server each time it packages data to send to the client web page. A hardware shutdown button is provided to ensure the orderly shutdown of the Linux operating system. The operation of this switch causes an interrupt that executes the listen-for-shutdown.sh shell script. The shutdown script is available in the smart suit detailed documentation.

1.4 Smart suit system

As indicated earlier two separate software modules and applications are automatically started during the system power-on operation:

- The Lepton Thermal Imager and
- The Flask server

1.4.1 Thermal imager module

When the Lepton Thermal Imager application is started, it creates a "LeptonThread" object to set up the operating parameters, such as SPI clock speed and frame size, then enters a continuous working loop. The loop inputs a line of data from the thermal camera consisting of row number plus 80 pixels while scanning for and then synchronizing to the first line count number of a frame. Once the first row is identified, the thread continues inputting lines of pixel data, storing them in an array, until the line count reaches the maximum (currently set to 60 lines), after which it scans the data array to find its max and min pixel values. Finally, it invokes an update image operation with arguments consisting of the data array, with maximum and minimum values for further processing in the main thread. When the main thread receives a completed frame, it pseudo colors the pixels (using a selected color map) based upon the maximum and minimum values present to form a completed image frame. The completed image frame is written as a high-quality JPG to working memory based upon a periodic timer event, which at present is limited to 1.8 s per frame.

1.4.2 Flask server module

When the Flask server is started, it first initializes the GPS system and senses that it is actually attached to the RP3. If the hardware is not present, it keeps looking for the attached hardware and will not continue until this important component part of the system is connected. When present, the GPS subsystem runs continuously in the background to populate time and coordinate data arrays for use by the web server. When the web server receives a request from the client application, it utilizes the populated GMT time and coordinates data arrays to construct a data structure (python dictionary). Additionally, the collected physiologic data (heart rate and temperature) are appended to the data structure. When all data have been incorporated, the web page template is rendered to contain the newly updated data, along with other java script operations to finalize the page. See Appendix A.1 for more details on system startup detailed scripts and Appendix A.2 on smart suit system application modules.

1.5 Wi-Fi setup and operation

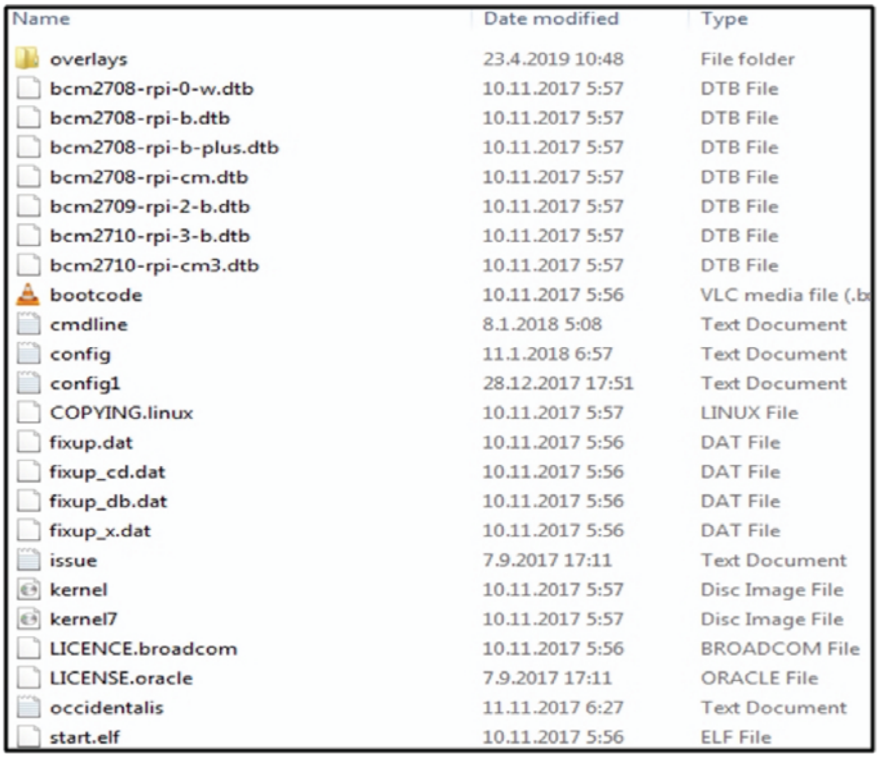
Since the smart suit system uses a local area network to transport and to present its data, it must first be connected to the network. The smart suit system uses a fixed dedicated network IP address (e.g., 192.168.0.99), which will be different for each smart suit system, and must fall within the range of addresses used by the network configuration. Each smart suit system acts as its own web server and is accessed on a fixed IP address at port 5000 (such as 192.168.0.99:5000). Wi-Fi setup of the system requires that a specific file contains the Service Set Identifier "SSID" and passphrases "PSK" for the network. For a "headless" system (one without monitor, keyboard, and mouse), these two items must be known ahead of time and added to the configuration file. This file may be set up prior to connecting to the Wi-Fi network for the first time, which should allow for automatic connection thereafter when the smart suit system boots. In a "headless" system, this configuration file must be positioned in the root directory on the RP3, which then gets automatically transferred to the **/etc/wpa_supplicant** directory upon the first bootup. The easiest way to setup this file is to edit it directly on the micro-SD card using a PC. This may be accomplished by using a micro-SD adapter plugged into the PC. Note that the boot sector of the micro-SD card is readable by the PC (under Windows) because it has the correct FAT32 structure. A text editor (Notepad) may be used to edit this file (**wpa_supplicant.conf**) in the micro-SD cards root directory. One has to make sure that no nontext or other characters are inadvertently entered in the file. The detailed contents of this file are in smart suit system documentation and it is used when setting up the system at some particular location with a specific Wi-Fi Access Point. After booting, Wi-Fi configuration file will be relocated to RP3 directory: **/etc/wpa_supplicant/wpa_supplicant.conf**. If the Wi-Fi does not appear to start up during operation, it could be due to an error in the original wpa_supplicant. conf file (such as tabs, or other wrong characters, or formatting) that had been placed in the micro-SD card's/boot directory. If HDMI display output with keyboard and mouse is available for the RP3, then editing the **/etc/wpa_supplicant.conf** file directly with the correct parameters should restore Wi-Fi hardware operation. This file may contain multiple network entries to allow the RP3 Wi-Fi to automatically connect at a number of locations. Since the Smart Server System web server delivers its web page to a fixed (programmed in) IP address, for example, **http://192.168.0.99:5000**, the router (Wi-Fi Access Point) used must not automatically assign some other device to this fixed address value. It might be necessary to set a reserved static IP address in the router for this purpose, but one needs to check the local router set up first. When finally connected to the network, one should use a newer Edge Microsoft web browser, or an updated Firefox browser to view the web page at the above IP address.

It is not recommended to use Chrome on PC, as it caches the thermal image then only uses the first one received with none of the new updates being viewable. It is possible to use a smart phone or tablet web browser to see the web page at the above IP address. This may later be useful in using a tablet directly as one of the smart suit components, for example, for some control purposes [11]. Figure 1.8 shows a partial list of configuration files.

1.6 Implementation

1.6.1 Components

The smart suit system platform components are shown in Figure 1.9, and their typical cost is summarized in Table 1.1. The total component cost is less than $150 in small quantities. These components were chosen because they were readily available and more choices exist today at smaller prices, so we can assume the total cost to be less than $100 in large quantities. More sensors can be added as well per the customer's requirement. In any case, we did not optimize either the cost or the



Name	Date modified	Type
overlays	23.4.2019 10:48	File folder
bcm2708-rpi-0-w.dtb	10.11.2017 5:57	DTB File
bcm2708-rpi-b.dtb	10.11.2017 5:57	DTB File
bcm2708-rpi-b-plus.dtb	10.11.2017 5:57	DTB File
bcm2708-rpi-cm.dtb	10.11.2017 5:57	DTB File
bcm2709-rpi-2-b.dtb	10.11.2017 5:57	DTB File
bcm2710-rpi-3-b.dtb	10.11.2017 5:57	DTB File
bcm2710-rpi-cm3.dtb	10.11.2017 5:57	DTB File
bootcode	10.11.2017 5:56	VLC media file (.b
cmdline	8.1.2018 5:08	Text Document
config	11.1.2018 6:57	Text Document
config1	28.12.2017 17:51	Text Document
COPYING.linux	10.11.2017 5:57	LINUX File
fixup.dat	10.11.2017 5:56	DAT File
fixup_cd.dat	10.11.2017 5:56	DAT File
fixup_db.dat	10.11.2017 5:56	DAT File
fixup_x.dat	10.11.2017 5:56	DAT File
issue	7.9.2017 17:11	Text Document
kernel	10.11.2017 5:57	Disc Image File
kernel7	10.11.2017 5:57	Disc Image File
LICENCE.broadcom	10.11.2017 5:56	BROADCOM File
LICENSE.oracle	7.9.2017 17:11	ORACLE File
occidentalis	11.11.2017 6:27	Text Document
start.elf	10.11.2017 5:56	ELF File

Figure 1.8 An excerpt from smart suit set of configuration files

Table 1.1 Component cost in small quantities

Description	Unit cost
Raspberry Pi 3—Model B—ARMv8 with 1G RAM	\$35.00
Aluminum heat sink for Raspberry Pi 3 15 × 15 × 15 mm	\$1.95
Pi Model B+ Pi 3 case base—smoke gray	\$5.00
Adafruit ultimate GPS breakout-66 channel W/10 Hz updates—Version 3	\$39.99
GPS antenna—external active antenna 3–5 V 28 dB 5 m	\$12.95
SMA to uFL u.FL/IPEX RF adapter cable	\$3.95
USB to TTL serial cable—debug/console cable for Raspberry Pi	\$9.95
Pulse sensor amped	\$25.00
TMP36—Analog temperature sensor	\$1.50

design configuration. The aim was to have an electronic demonstration and marketing platform as well as a general electronic design platform. The biggest challenge in the project was to design this platform in a very short period of time which was only 3 months. Additional 2 weeks were needed to implement the components into the suit itself and test it. All the components were embedded inside the suit with addition of a couple of pockets to hold the components and make them available for set up and system activation/reset. The GPS sensor was placed up in the shoulder of the suit, and its antenna on the other shoulder. The thermal camera was situated in the upper pocket. The computer module with the battery is in an inner pocket secured with Velcro; heart rate and temperature sensors were located in one of the suit sleeves. Also, cabling was embedded throughout the suit to connect all the components. The computer module has both 4G as well as Wi-Fi modules, and Wi-Fi was used to connect the suit with Internet via local Wi-Fi access point. A Micro SD card had all the drivers and software required to run the suit. The suit was tested in a number of locations around Sarajevo, Bosnia and Herzegovina and Silicon Valley, USA. For example, Figures 1.4 and 1.5 show two streets in Sarajevo area and Figure 1.3 shows a street in Silicon Valley, with Google Map in lower-left corner indicating the street, as well as showing a person inside the building with his temperature and heart rate data and his thermal image obtained from the camera in the suit upper pocket. These data were obtained by logging into the suit "web site" which showed what the suit condition at that point in time. The platform is suitable for a specific sensor(s) extension as it may be required by a specific application at hand.

1.6.2 Application

The smart suit platform as described in this chapter is suitable for many different applications, both for defense and for a variety of commercial applications [11]. The suit in the right upper corner of Figure 1.9 was supplied for demonstrating purposes, and a smaller and lighter suit (jacket) can be also used. The components from Table 1.1 were physically implemented in that demo suit using proper wiring. One could have also used various wireless versions of these components. The suit

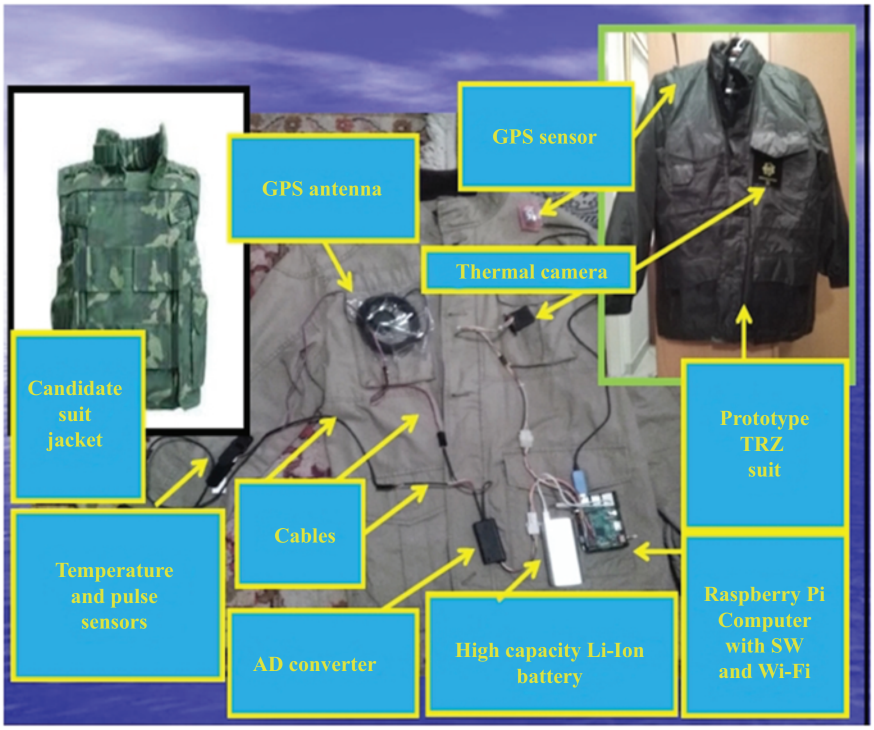

GPS sensor
GPS antenna
Thermal camera
Candidate suit jacket
Prototype TRZ suit
Cables
Temperature and pulse sensors
Raspberry Pi Computer with SW and Wi-Fi
AD converter
High capacity Li-Ion battery

Figure 1.9 Smart suit prototype and components layout

was demonstrated to a number of potential users such as local police, mountain rescue groups, as well as civil protection service and several defense application users. Each of the potential customers indicated their own specific requirements, in particular sensor-related details. Our smart suit design platform can accommodate adding additional sensors and integrating them via embedded computer and software developed for the Platform. Sensors can be either wired or wireless, and each customer may supply their own specific suit or jacket. Figure 1.10 shows some possible applications and a variety of different suits and also a possible remote connectivity arrangements whereas the suit would be connected to some customer control center from where the suit parameters and the movement of the person wearing it could be observed. One of the applications which emerged from potential customers following demonstration of our design was a need to have personnel positions and their health as well as kinetic (moving, not moving, running, and walking) information available at all times within a company or other campus, or a large building. For this, the Wi-Fi would be good enough. For field applications one

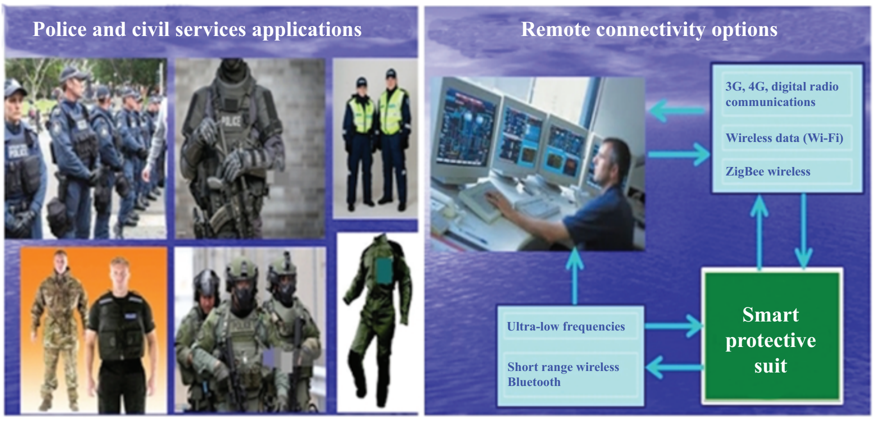

Police and civil services applications
Remote connectivity options
3G, 4G, digital radio communications
Wireless data (Wi-Fi)
ZigBee wireless
Smart protective suit
Ultra-low frequencies
Short range wireless Bluetooth

Figure 1.10 Applications and remote connectivity

would need 3G or 4G for wide-area wireless networking. Our design has both of these options built-in.

1.6.3 Mountain rescue services emergency response application

As described in Section 6.2, our smart suit design has a wide applicability across various commercial, security, and emergency services applications. One specific area of interest has emerged from our application surveys in Sarajevo area in Bosnia and Herzegovina came from a dedicated mountain rescue unit, which is of interest due to a number of surrounding mountains and Olympic skiing tradition, with a sizeable amount of snow and skiing activities during the winter season. Per our discussion with a local mountain rescue group, our basic suit design is an excellent technological base which would require some sensor additions and fine tuning. In particular, due to the remoteness of the applicable terrain where mountain rescue might transpire, Wi-Fi may not be the best choice for audio, physiological, and visual or thermal data communications, hence mobile GSM 3G, 4G, or 5G would be a preferable choice. In addition to GSM communication, a digital VHF radio infrastructure already used by the mountain rescue unit is under consideration. This is important as there are potential blind spots for GSM coverage in some deep ravines and canyons in the area. Our design accommodates that option as well, whereas a suitable data modem would be connected instead of (or on top of) Wi-Fi device, see Figures 1.1 and 1.10. Mountain rescue teams would be able to communicate real time data from the terrain, as they move around in their rescue effort. As this activity transpires a control center nearby (Figure 1.10) would be able to access the team physical condition, their heart rate, and temperature, indicating the condition of the team, whereas additional sensors would be used to evaluate vital functions of the persons being rescued, once they are located. It would be pretty trivial to add additional sensors, such as a standard video camera to have a real time video transmission as the search is progressing. In addition to the above, an additional integrated or a separate voice communication device could be added to the rescue team suits, which can further facilitate the effectiveness of their efforts. As the prototype suit is designed, the rescue team will be used to test its various features in real rescue or simulated conditions on the real terrain, which can assist in fine-tuning the suit design. Figure 1.10 shows a few possible rescue uniforms and suits which can be used to implement our design for mountain rescue application.

1.7 Conclusion

In this chapter, we present a prototype design of a smart suit (jacket) which uses GPS, temperature, heart rate, as well as thermal information sensor embedded into a specific suit for demonstration purposes. The suit has its own Internet address and as such can be accessed remotely and its condition can be observed in real time every second. Real data transfer which the platform can accommodate is much larger. Other sensors can be added as well. Wi-Fi is also viable for the data communications, as well as 3G or 4G mobile communications if required by a specific application. These applications range from a variety of commercial to specific defense areas. The smart suit as implemented can be a part of a larger system such as of a Smart City with a number of city services interconnected and smart suit as a part of these services.

Appendix A

Software startup scripts and modules

A.1 System startup detailed scripts

Crontab

At system powerup, the Flask web server is started using the cron daemon, which is setup within directory **/var/spool/cron/crontabs/pi**. This daemon has already been setup for operation in released code. Development programming uses the terminal crontab command:

```
crontab -u pi -e
```

This will bring up an editor allowing changes to be made to the crontab (cron table). The following is then entered on the last line of the file that opens (Note: press i for insert):

```
@reboot sleep 20; /usr/bin/python3
/home/pi/webpy/servflask.py
```

When finished editing type “esc” to exit the edit mode, then save and exit the editor by typing “wq”. The file will run at startup and delay bootup for 20 s to allow Python to execute the script servflask.py found in directory /home/pi/webpy. After the 20 s delay is over, the python script should have run and the bootup process can continue.

startup.sh When the system starts up, it executes the startup.sh script found in the user's home directory /home/pi/startup.sh. This shell script is used to start up the Lepton thermal imaging application and contains the following:

```
#! /bin/sh
#Start Lepton
/home/pi/Qt/Demo/build-Demo-kit2-Debug/./ Demo
```

listen-for-shutdown.sh

During system startup, a script allowing shutdown when I/O Port pin 3 is grounded is executed. Note that shutting down the Operating System in this manner causes a processor halt. When halted, the system may be re-started again by grounding I/O Port pin 3. The shutdown script contains the following:

```
#! /bin/sh
# BEGIN INIT INFO
# Provides: listen-for-shutdown.py
# Required-Start: $remote_fs $syslog
# Required-Stop: $remote_fs $syslog
# Default-Start: 2 3 4 5
# Default-Stop: 0 1 6
# END INIT INFO
case "$1" in
start)
echo "Starting listen-for-shutdown.py"
/usr/local/bin/listen-for-shutdown.py
;;
stop)
echo "Stopping listen-for-shutdown.py"
pkill -f /usr/local/bin/listen-for-shutdown.py
;;
*)
echo "Usage: /etc/init.d/listen-for-shutdown.sh
start|stop"
exit 1
;;
esac
exit 0
```

listen-for-shutdown.py

Python script invoked from startup shell script listen-for-shutdown.sh. This script sets GPIO3 as an interrupt source that triggers when Raspberry Pi port pin 5 is grounded, causing system shutdown. Contents of Python script listen-for-shutdown.py is: Contents of listen-for-shutdown.py is:

```
#!/usr/bin/env python
import RPi.GPIO as GPIO
import subprocess
GPIO.setmode(GPIO.BCM)
GPIO.setup(3, GPIO.IN, pullq_up_down=GPIO.PUD_UP)
GPIO.wait_for_edge(3, GPIO.FALLING)
subprocess.call([“killall”, “python3”])
subprocess.call([“killall”, “python3”])
subprocess.call([“sudo”, “shutdown”, “now”])
subprocess.call([“echo”, “POWER OFF
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!”], shell=True)
wpa_supplicant.conf
```

A first-time boot-up of the system transfers the wpa_supplicant.conf file (if found) from the root directory on microSD card to the Linux system /etc/wpa_-supplicant directory. After the system is started, look for this file in directory etc/wpa_supplicant/wpa_supplicant.conf. This file is used to configure the Wi-Fi with its operational credentials and has the following format:

```
ctrl_interface=DIR=/var/run/wpa_supplicant GROUP=netdev
update_config=1
country=BA
network=
ssid=“Your SSID name”
psk=“Your password”
key_mgmt=WPA-PSK
```

A.2 Smart suit system application modules

Thermal imager module

The thermal imager executable is invoked during the startup procedure when startup.sh, located at /home/pi/startup.sh, is run. This shell script issues the run operation (./Demo) on this executable file in directory /home/pi/Qt/Demo/build-Demo-kit2-Debug/. Contents of the startup.sh script are:

```
# /bin/sh
# Start Lepton
/home/pi/Qt/Demo/build-Demo-kit2-Debug/./Demo
```

The Demo project was developed using Qt4 and generates the executable file named “Demo.” Qt is a cross-platform development environment but is used natively on the Raspberry Pi 3 platform under Debian Linux distribution. This means that the entire development environment operates on the Pi and its build output is saved directly to a local development directory. Program execution by the startup shell file invokes the executable out of this target directory. Once started, the application initializes the Qt-based MainWindow parameters to setup the image for display, starts a periodic timer, and then starts the Lepton thread reading the Lepton thermal camera image. The Lepton thread runs in the background, taking in and processing pixel data at the thermal camera’s normal data rate of 27 frames per second. New frames are generated at only nine frames per second, however, with frames being saved to an array in memory. The periodic timer is, at present, setup to save the presently available image to a JPG format in memory, which is accessible by the web server when an image is needed. Thermal Imager Module operation is outlined in the following diagram in Figure A.1.

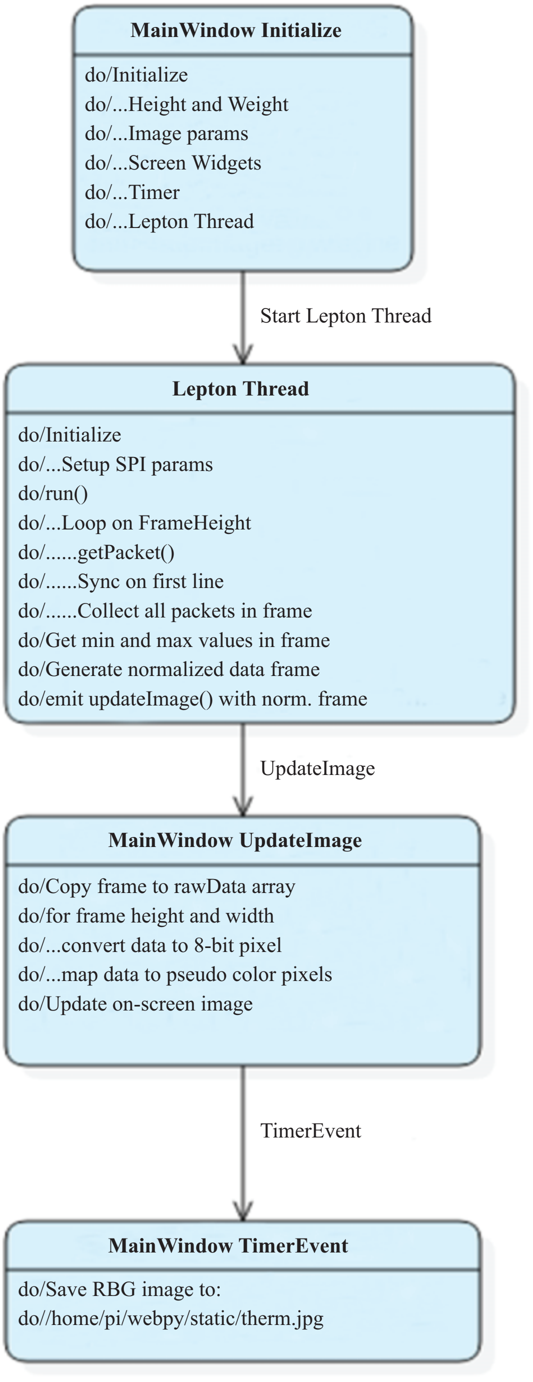

MainWindow Initialize
do/Initialize
do/...Height and Weight
do/...Image params
do/...Screen Widgets
do/...Timer
do/...Lepton Thread
Start Lepton Thread
Lepton Thread
do/Initialize
do/...Setup SPI params
do/run()
do/...Loop on FrameHeight
do/......getPacket()
do/......Sync on first line
do/......Collect all packets in frame
do/Get min and max values in frame
do/Generate normalized data frame
do/emit updateImage() with norm. frame
UpdateImage
MainWindow UpdateImage
do/Copy frame to rawData array
do/for frame height and width
do/...convert data to 8-bit pixel
do/...map data to pseudo color pixels
do/Update on-screen image
TimerEvent
MainWindow TimerEvent
do/Save RBG image to:
do//home/pi/webpy/static/therm.jpg

Figure A.1 Diagram of thermal imager application demo

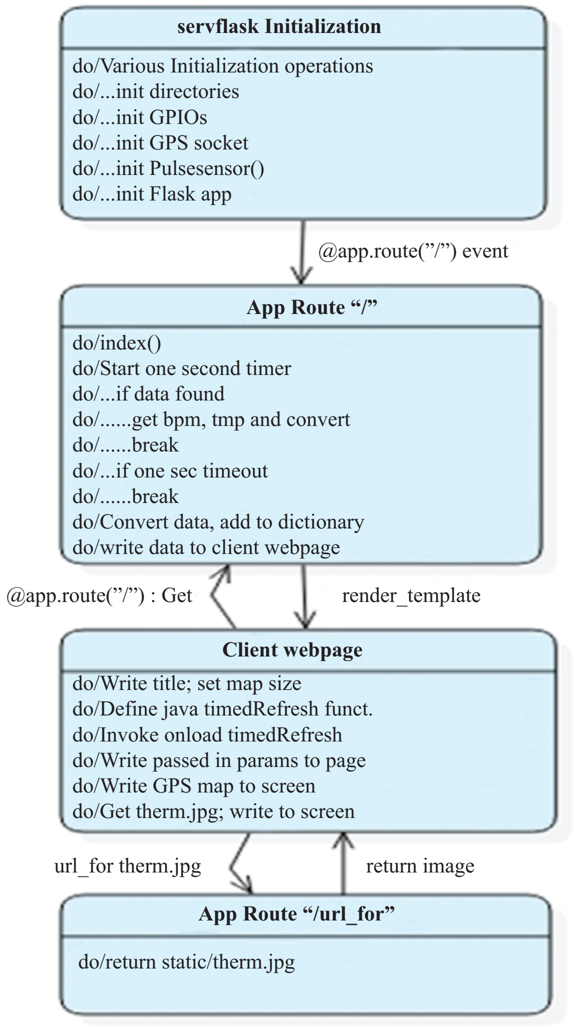

servflask Initialization
do/Various Initialization operations
do/...init directories
do/...init GPIOs
do/...init GPS socket
do/...init Pulsesensor()
do/...init Flask app
@app.route("/") event
App Route "/"
do/index()
do/Start one second timer
do/...if data found
do/......get bpm, tmp and convert
do/......break
do/...if one sec timeout
do/......break
do/Convert data, add to dictionary
do/write data to client webpage
@app.route("/") : Get
render_template
Client webpage
do/Write title; set map size
do/Define java timedRefresh funct.
do/Invoke onload timedRefresh
do/Write passed in params to page
do/Write GPS map to screen
do/Get therm.jpg; write to screen
url_for therm.jpg
return image
App Route "/url_for"
do/return static/therm.jpg

Figure A.2 Diagram of flask server module application

Qt4 creator

The software development framework from open source Qt4 has been used to edit and build the Thermal Imager module to produce executable "Demo." Installation of Qt4 on the Raspberry Pi is done using the following steps from terminal (use LXTerminal to build, compile, and run Qt):

1. Verify that current time and date are set on the Raspberry Pi
2. sudo apt-get update

3. sudo apt-get upgrade
4. sudo apt-get install qtcreator
5. Open Qtcreator and go to: Options ¿ build & run ¿ tab tool chain ¿ button add ¿Choose GCC
6. Set compiler path: /usr/bin/arm-linux-gnueabihf-gcc-4.6
   Debugger: /usr/bin/gdb Mkspec : default
7. Go to menu help ¿ about plugins and Uncheck device support ¿ remote linux

Restart Qt creator

Go to tools ¿ options TAB ¿ build & run ¿ Qt versions ¿ add "/usr/bin/qmake-qt4"

After code has been edited, open LXterminal and make the current directory containing the project files the current directory. To build project use the following:

1. qmake -project
2. qmake <yourprojectname>.pro
3. make (//For compilation of the code)
4. sudo ./< yourprojectname> (//For running the program)

Command qmake only needs to be done the first time to generate the make file, then after code changes are made only command make is necessary to build the project.

Flask server module smart suit web server software development is undertaken using Python version 3 and the Flask server environment. The Raspberry Pi 3 operating system comes with both Python v2.7.13 and v3.5.4 installed. To run programs under Python 3, however, the user must explicitly enter "python3" on the command line (entering just python will invoke version 2.7). Note that the startup script that invokes the Flask server application first calls Python 3 (/usr/bin/python3), before Flask. Flask operation is outlined in the following diagram in Figure A.2.

References

[1] Al-Turjman F., Altrjman C., Din S., and Paul A. "Energy monitoring in IoT-based ad hoc networks: An overview." Elsevier Computers & Electrical Engineering Journal. 2019, vol. 76, pp. 133–142.

[2] Al-Turjman F. "Cognitive routing protocol for disaster-inspired Internet of Things." Elsevier Future Generation Computer Systems. 2019, vol. 92, 1103–1115.

[3] Al-Turjman F. "Cognitive-node architecture and a deployment strategy for the future sensor networks." Springer Mobile Networks and Applications. 2019, vol. 24, no. 5, pp. 1663–1681.

[4] Hodzic M. and Muhic I. "Internet of Things: Current technological review." Periodicals of Engineering and Natural Sciences. 2014, vol. 2, no. 2.

[5] Accenture Corporation. *Delivering public service for the future: Navigating the shifts. Corporate handbook*. Austin, TX, USA; 2012.
[6] European Commission. *Smart wearables: Reflection and orientation paper*. Directorate-General for Communications Networks, Content and Technology. Brussels, Belgium: EU; 2016.
[7] Research and Markets. *Wearable electronics—Market analysis, trends, and forecasts*. Global Industry Analysts, Inc. San Jose, CA, USA; 2016.
[8] Raspberry Pi Foundation. Raspberry Pi 3 Model B (Reduced Schematics). Cambridge, UK; 2015.
[9] Coyle, S., Wu, Y., Lau, K.T., De Rossi, D., Wallace, G. and Diamond, D. "Smart nanotextiles: A review of materials and applications." Mrs Bulletin. 2007, 32(5), pp. 434–442.
[10] FLIR Corporation. Lepton Engineering Datasheet. Document Number: 500-0659-00-09 Rev: 203. Wilsonville, OR, USA; 2017.
[11] Hodzic M. Smart suit presentation. Author's private archive. 2019.