618194126

TK
51059
SS3
2008

FULLY MOBILE FUNCTIONAL BRAIN SPECTROSCOPY · USING NEAR-INFRARED LIGHT AND WIRELESS NETWORKS

by

Salah Sharieh

B.Sc., Al-Balqa' Applied University, Jordan, 1993

A thesis

presented to Ryerson University

in partial fulfillment of the

requirements for the degree of

Master of Science

in the Program of

Computer Science

Toronto, Ontario, Canada, 2008

© Salah Sharieh 2008

**PROPERTY OF
RYERSON UNIVERSITY LIBRARY**

I hereby declare that I am the sole author of this thesis.

I authorize Ryerson University to lend this thesis to other institutions or individuals for the purpose of scholarly research.

I further authorize Ryerson University to reproduce this thesis by photocopying or by other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research.

ABSTRACT

FULLY MOBILE FUNCTIONAL BRAIN SPECTROSCOPY USING NEAR-INFRARED LIGHT AND WIRELESS NETWORKS

Salah Sharieh

MSc, Computer Science, Ryerson University, 2008

This thesis presents research that creates a mobile solution to monitor human brain functions during real-life activities. The solution utilizes the Internet, the Global Standard for Mobile Communication (GSM) wireless networks, Bluetooth technology, and a number of data protocols. It consists of three main parts:

1. a Bluetooth portable near-infrared light sensor,
2. a personal digital assistant (PDA), and
3. a personal computer (PC).

The near real time data acquisition is performed by the sensor while mobility is provided by the GSM PDA. The data is sent over a multi-protocol stack until it reaches the final destination PC. The system provides a light-weight, tissue hemoglobin monitoring system for real-life situations not restricted to a lab.

ACKNOWLEDGEMENTS

I never thought that writing the acknowledgment would be harder than writing the thesis itself until the moment I tried to search for words or statements that may fully express my gratitude to Professor Alex Ferworn. The completion of this thesis would not have been possible without his kind encouragement and intellectual instruction. Professor Alex Ferworn made me acknowledge that hard journeys can always end successfully and be memorable when the captain is not only a professional but also a humane.

Also, I would like to thank Professor Vlad Toronov for introducing me to the biomedical physics field as well as sharing with me his knowledge and time throughout the research. Professor Vlad Toronov has been always there to ask me the right questions and to comment on my answers thereto.

TABLE OF CONTENTS

1	CHAPTER 1 - INTRODUCTION.....	1
1.1	OVERVIEW	1
1.2	NETWORKS.....	2
1.3	BRAIN SPECTROSCOPY	3
1.4	SUMMARY	4
2	CHAPTER 2 – LITERATURE REVIEW AND BACKGROUND MATERIAL.....	5
2.1	SENSOR NETWORKS	5
2.2	SOFTWARE AND PROTOCOLS	7
2.3	KREIOS PROTOCOL	9
2.3.1	<i>Protocol Overview</i>	9
2.4	NEAR-INFRARED SPECTROSCOPY	9
2.5	TISSUES OPTICAL PROPERTIES	11
2.6	MODIFIED BEER LAMBERT LAW	12
2.7	ANATOMY AND PHYSIOLOGY	17
3	CHAPTER 3 - MATERIALS AND METHODS.....	19
3.1	HARDWARE ARCHITECTURE	19
3.1.1	<i>The Sensor</i>	20
3.1.2	<i>The Mobile Phone (PDA)</i>	23
3.1.3	<i>The Host PC (Server)</i>	24
3.2	LAYERPRO PROTOCOL	24
3.2.1	<i>Protocol Overview</i>	24
3.2.2	<i>Packet Format</i>	25
3.2.3	<i>Transactions and Packet Flow</i>	25
3.3	SOFTWARE ARCHITECTURE.....	26
3.3.1	<i>Overall System Component Interaction</i>	28
3.3.2	<i>Sensor Software Architecture</i>	30
3.3.2.1	Sensor Software Component.....	30
3.3.2.2	Sensor Software Component Interaction.....	31
3.3.3	<i>PDA Software Architecture</i>	33
3.3.3.1	PDA Software Components	33
3.3.3.2	PDA Software Components Interaction	37
3.3.4	<i>Server Software Architecture</i>	39
3.3.4.1	Server Components	39
3.3.4.2	Server Component Interactions.....	40
3.4	SOFTWARE DESIGN AND IMPLEMENTATION	41
3.4.1	<i>PDA Software Detailed Design</i>	41
3.4.2	<i>PDA Software Detailed Implementation</i>	44
3.4.2.1	PDA Main Controller.....	44
3.4.2.2	PDA Sensor Communication Controller	47
3.4.2.3	PDA Server Controller.....	51
3.4.2.4	PDA Socket Client	52
3.4.2.5	LayerPro Protocol	55
3.4.3	<i>Server Software Detailed Design</i>	56
3.4.4	<i>Server Software Detailed Implementation</i>	57
3.4.4.1	LayerPro Socket Server	57
4	CHAPTER 4 - RESULTS AND DISCUSSION.....	59
4.1	INFORMATION SYSTEMS RESULTS.....	59
4.1.1	<i>Summary</i>	59

4.1.2	<i>System Data Transmission Performance</i>	60
4.1.3	<i>System Data Integrity</i>	61
4.1.3.1	Packet Sequence Number.....	61
4.1.4	<i>System Availability</i>	63
4.1.5	<i>LayerPro Protocol</i>	63
4.2	BIOMEDICAL APPLICATION RESULTS.....	64
4.2.1	<i>Summary</i>	64
4.2.2	<i>Breath Holding Experiment Results</i>	65
4.2.3	<i>Smoking Experiment Results</i>	67
4.2.4	<i>Canine Experiment Result</i>	71
4.3	DATA TRANSMISSION DISCUSSION.....	74
5	CHAPTER 5- SUMMARY, CONCLUSION, AND FUTURE WORK.....	76
5.1	CONCLUSION AND CONTRIBUTIONS OF THIS WORK	76
5.2	FUTURE WORK.....	77
6	REFERENCES.....	79
7	APPENDICES.....	91

LIST OF TABLES

Table 4.1 Average Delay Using Layer Pro Protocol per day /Location	61
Table 4.2 LayerPro Protocol versus HTTP Protocol	64
Table A.7.1: Packet Description	99
Table A.7.2: Device Type.....	100
Table A.7.3: Packet Type.....	100
Table A.7.4: Control Commands.....	101
Table A.7.5: Error Codes	101
Table A.7.6: OPC_EMPTY Packet	102
Table A.7.7: OPC_ERROR Packet.....	102
Table A.7.8: OPC_ACK Packet.....	103
Table A.7.9: OPC_NACK Packet.....	103
Table A.7.10: OPC_NACK_ERROR Packet	104
Table A.7.11: OPC_NOT_SPECIFIED Packet.....	104
Table A.7.12: OPC_CTRL_COMMAND Packet	105
Table A.7.13: OPC_ACK_PAYLOAD Packet	105
Table A.7.14: OPC_NIRS_DATA Packet.....	106
Table A.7.15: OPC_WRAPPED Packet.....	106
Table A.7.16: OPC_STATUS_INFO Packet.....	107
Table C.7.17: Test Subject Information.....	131

LIST OF FIGURES

Figure 2.1: Modified Beer-Lambert Law, Light Photon Propagation in Tissues Between Source and Detectors	14
Figure 2.2: The Brain's Protective Layers[18]	18
Figure 3.1: System Architecture	20
Figure 3.2: Sensor Components	21
Figure 3.3: Sensor Front (left) &	22
Figure 3.4: Sensor Back (right).....	22
Figure 3.5: Sensor Light Source and Light Detector Distribution	22
Figure 3.6: Mobile Phone (PDA).....	23
Figure 3.7: Protocol Stack.....	25
Figure 3.8: LayerPro Packet Format.....	25
Figure 3.9: Layer Pro Transactions.....	26
Figure 3.10: The System Software Architecture.....	28
Figure 3.11: The Overall System Sequence Diagram.....	29
Figure 3.12: Sensor Components Diagram.....	30
Figure 3.13: Open Connection and Start Measurements Use Cases.....	32
Figure 3.14: PDA Components Diagram.....	34
Figure 3.15: Communication Manager Design.....	37
Figure 3.16: PDA Main Use Cases Sequence Diagram.....	38
Figure 3.17: Server Components Diagram	39
Figure 3.18: Server Components Sequence Diagram	41
Figure 3.19: PDA Packages	42
Figure 3.20: LayerPro Protocol Packet Class Diagram	44
Figure 3.21: Measurements Algorithm	46
Figure 3.22: Controller Class Outlines	47
Figure 3.23: Sensor Class Outlines.....	50
Figure 3.24: Buffer Flash Algorithm	50
Figure 3.25: PDA Socket Client Outlines.....	53
Figure 3.26: Client Server Communication Algorithm	54
Figure 3.27: LayerPro Protocol Packet Class Outline View.....	55
Figure 3.28: Server Packages.....	57
Figure 3.29: LayerPro Server Main Class Outlines.....	58
Figure 3.30: LayerPro Main Algorithm	58
Figure 4.1: Test Subject Wearing the Sensor during the Breath Holding Experiments ...	66
Figure 4.2: Breath Holding Trial # 1	67
Figure 4.3: Breath Holding Trial # 2	67
Figure 4.4: Test Subject Wearing the Sensor during the Smoking Experiments.....	69
Figure 4.5: Smoking.....	70
Figure 4.6: Sample Smoking Trial.....	70
Figure 4.7: This is a picture of Kathy from CAN-TF3 (Toronto Heavy Urban Search and Rescue) and her dog while performing the experiment in the first phase	72

Figure 4.8: Constable Kevin Barnum hold sensor on USAR canine Dare's head while Constable Denis Harkness presents the toy while Salah collects the data in the background.....	73
Figure 4.9: Canine Trial # 1	74
Figure A.7.1: Control Activities Transactions.....	96
Figure 7.2: Measurement Activities Transaction.....	99
Figure B.7.3: Server Status	108
Figure B.7.4: PDA Main Screen.....	109
Figure B.7.5: Menu Screen	109
Figure B.7.6: Splash Screen.....	110
Figure B.7.7: Application Main Screen	111
Figure B.7.8: Sensor Front.....	111
Figure B.7.9: Application Menu	112
Figure B.7.10: File Access Request.....	113
Figure B.7.11: File Access Request.....	113
Figure B.7.12: File Access Request.....	114
Figure B.7.13: File Access Request.....	114
Figure B.7.14: Network Access Request.....	115
FigureB.7.15: Network Selection	115
Figure B.7.16: Bluetooth Password Screen	116
Figure B.7.17: System Status Screen.....	116
Figure B.7.18: Sensor Bottom	117
Figure B.7.19: Pooling Status	117
Figure B.7.20: Stop Measurements Screen.....	118
Figure B.7.21: File Access Request.....	118
Figure B.7.22: File Access Request.....	119
Figure B.7.23: File Access Request.....	119
FigureB .7.24: File Access Request.....	120
Figure B.7.25: Bluetooth Password Screen	120
Figure 7 B.7.26: System Status Screen.....	121
Figure B.7.27: Pooling Status	122
Figure B.7.28: Stop Measurements Screen.....	122
Figure B.7.29: Setup Server IP menu	123
Figure B.7.30: Server IP Configuration.....	124
Figure B.7.31: Import Menu	125
Figure B.7.32: Server Screen.....	126
Figure B.7.33: Exit Menu	127
Figure B.7.34: Server Main Screen.....	128

LIST OF ILLUSTRATIONS

Equation 2.1: Light Attenuation between Sources Light and Detected Light	14
Equation 2.2: Optical Density for Wavelength λ	14
Equation 2.3: Absorption of Light in Blood	15
Equation 2.4: Path Length	15
Equation 2.5: Differential Path Length	16
Equation 2.6: Differential Value	16
Equation 2.7: Transformation from Light Output Change to Change in Blood Chromophore Concentrations	17
Equation 2.8: Blood Chromophore Concentration Parameters	17
Equation 4.1: Total Packet Number Calculator	62
Code Snippet D.1: Main Controller Class	134
Code Snippet D.2: Sensor Class	138
Code Snippet D.3: Server Class	141
Code Snippet D.4: LayerPro Packet Class	146
Code Snippet D.5: LayerPro Server Class	147

NOMENCLATURE

C_i :	Blood Chromophores Concentration
d	Distance Between the Light Source and the Light Detector
DPF_λ	Differential Path Length Factor.
$\epsilon_{i,\lambda}$	Extinction Coefficient of Blood Chromophore for Wavelength λ
I_{out}	Detected Light.
I_{in}	Source Light.
L_λ	Path Length of Light at λ .
OD_λ	Optical Density for Wavelength λ
μ_a, λ	Absorption Coefficient at Wavelength λ .
$\mu_{s',\lambda}$	Reduced Scattering Coefficient at Wavelength λ .

ABBREVIATIONS

BOLD	Blood Oxygen Level Dependent
CSF	Cerebrospinal Fluid
CT	Computed Tomography
CWS	Continuous Wave Spectroscopy
DOT	Diffuse Optical Tomography
EEG	Electroencephalography
fMRI	Functional Magnetic Resonance Imaging
fNIRS	Functional Optical Brain Spectroscopy Using Near-infrared Light
FDS	Frequency Domain Spectroscopy
GSM	Global Standard for Mobile Communications
GPRS	General Packet Radio Services
Hb	deoxyhemoglobin
HbO₂	oxyhemoglobin
J2EE	Java Enterprise Edition
J2ME	Java Micro Edition
J2SE	Java Standard Edition
MRI	Magnetic Resonance Imaging
NIRS	Near-infrared Spectroscopy
NIRI	Near-infrared Imaging
PET	Positron Imaging Tomography
RFCOMM	Radio Frequency Communications

SDK	Software Development Kit
SPECT	Single Photon Emission Computed Tomography
SPP	Serial Port Profile
TRS	Time-Resolved Spectroscopy
URL	Unified Resource Locator
TCP	Transmission Control Protocol
UDP	User Datagram Protocol

1 CHAPTER 1 - INTRODUCTION

1.1 Overview

The purpose of this research is to determine the feasibility of providing full mobility in everyday environments using a near-infrared light sensor designed to monitor brain function in humans. Multiple wireless networks employing several different protocols are used for data carriage and provide new freedom to conduct tests in real environments outside a lab. The measurements of the changes of the concentration of oxyhemoglobin (HbO_2) and deoxyhemoglobin (Hb) in the real-life environment, might lead to better understanding of tissue pathologies.

A fully mobile functional brain Spectroscopy system was developed in this research to allow the possibility of testing subjects to be monitored in their real environments. To test this hypothesis, communication software was developed to allow for the collection of physiological data from a mobile near-infrared sensor via a mobile telephone. The developed system is used to track the changes in the concentrations of HbO_2 and Hb during various activities and send the data to a computer at a monitoring site.

The specific aims of this research are to build a fully mobile system to monitor the concentrations of HbO_2 and Hb in near real time, to monitor the concentrations of HbO_2 and Hb during smoking, as well as to analyze the gathered data, to try to understand the correlation between HbO_2 and Hb during smoking.

1.2 Networks

Networks – whether infrastructure based or non-infrastructure based – play an important role in our lives [1, 2, 3]. Wired networks, such as the Internet, provide us with global data access, while wireless networks such as the Global Standard for Mobile Communications (GSM) give us mobility. Non-infrastructure-based networks (ad-hoc networks) such as Bluetooth networks give us the freedom to communicate at no cost over short ranges [4].

Bluetooth devices utilize the unlicensed frequency 2.4 GHZ that offers a 10 to 100 meter range and a data transfer rate of up to 1 Mbps [5]. Bluetooth technology offers point-to-point and point-to-multiple-points communication [6]. It performs communication through a protocol stack divided into hardware and software layers [7]. Bluetooth standards were created to provide guidelines to device manufacturers to facilitate interoperability between devices from different vendors. Moreover, Bluetooth standards specify profiles that determine the usage of the device and the services offered by it [8]. Standardization, low cost, minimum hardware, low power requirements, and the free use of the unlicensed band all contributed to the wide spread use of Bluetooth devices [9].

GSM is widely used in more than 200 countries around the world, having an estimated subscriber base of over two billion users [10]. Roaming is one of the value-added features introduced by the GSM standard. This capability allows mobile users to travel the world and still be able to use their phones to connect with local operators. The introduction of data communication has also helped GSM standards to become more and

more popular. GSM networks currently offer wide varieties of services, ranging from basic voice services to more advanced capabilities such as allowing Internet access.

GSM's many features make it possible to use this type of network to assist in the monitoring of people's physiological parameters in everyday life regardless of their location [11, 12]. GSM networks use different frequencies for upload and download links, which offer various data transfer rates between the network and the device. The data transfer rates can reach up to 9.6 kbps, which allows the networks to offer basic data services to their users [13]. The introduction of General Packet Radio Services (GPRS) – data services to GSM networks – has made it possible to run more varieties of applications than before at a lower cost and faster speed [14]. GPRS was added on top of the traditional GSM network to allow network operators to offer better data communications. GPRS is a packet-switched communication method where the communication channel can be employed by other users, unlike other data communication methods such as circuit switched data. With GPRS download rates reaching 236 Kbps and upload transfers reaching up to 118 Kbps, GPRS offers enhanced speed over the traditional GSM network [15].

1.3 Brain Spectroscopy

Functional brain imaging using fMRI and Positron Imaging Tomography (PET) have increased our understanding of the neural circuits that support cognitive and emotional processes [16, 17]. However, these methods are expensive, uncomfortable, and might have side effects such as exposure to radioactive materials (with PET) or loud noises (with fMRI) [18, 19]. Such disadvantages make these imaging methods

inappropriate for many uses that require the monitoring of brain activities under daily, real-life conditions.

Functional Optical Brain Spectroscopy Using Near-infrared Light (fNIRS) is another method to conduct functional brain analysis. fNIRS is a non-invasive method that uses infrared light reflection to gather changes in the concentration of HbO₂ and Hb in the blood [20]. The main advantages of fNIRS are the ability to measure concentration of chemical substance, the device's low cost, low power requirements, non-invasiveness, and portability. Low cost and portability have made it possible to use fNIRS to monitor patients in their homes for an extended period of time, allowing health care providers to monitor slowly developing diseases in patients. The non-invasive nature of fNIRS has also made it possible to perform as many tests as needed without worrying about side effects [21].

1.4 Summary

Combining mobile network technologies with fNIRS introduces a new way to monitor human brain activities under real-life conditions. This research contributes a system that combines all of the above discussed technologies to provide a novel, cost effective and mobile means to monitor human brain activities.

2 CHAPTER 2 – Literature Review and Background Material

2.1 Sensor Networks

Several types of sensors have been created and used to monitor automobile functions [22]. These sensors can often communicate with each other to form sensor networks [23], moreover such networks have been used in manufacturing, telecommunication, security [24, 25, 26, 27] and natural environment [28]. Currently there are wide varieties of sensor networks used in factories to perform different tasks ranging from monitoring machines' performance and product quality to ensuring worker safety [29, 30, 31]. Monitoring machines or products on factory floors does not require mobility or wireless connectivity since everything is in close proximity. On the other hand, monitoring habitat [32] using multiple sensors without disturbing the surrounding environment for an extend periods of time is not practical with wired networks. Wireless sensor networks are used in this case [33, 23, 34,35].

For example, several sensors can be dropped from the air in different locations, where they can perform data collection about the environment, and then send the data back to a central location using wireless communication. Wireless sensor networks have limited mobility; they are stationary. Mobile wireless sensor networks, however, allow mobile monitoring of conditions and situations that are at least somewhat independent of location and, use wireless connections to transmit their collected data [36]. This type of network is used in many fields. Health monitoring [37, 38] and telemedicine [39] utilize

this type of network to monitor patients remotely [40]. Other applications of mobile wireless sensor networks include [41], but are not limited to, entertainment [42], automobile [43, 44], social studies [45], and home appliances [46].

Sensor networks differ in their technical capabilities and implementation (hardware, software, communication), for example researchers have managed to create a very small operating system that can be loaded into 178 bytes of memory [47, 33,48]. One of the most advanced sensor network platforms available on the market today is Intel Mote [36, 49]. It features an ARM 7 processor, Bluetooth transceiver [50], RAM, flash memory, and multiple IO options. It runs TinyOS, which is an open source embedded operating system that provides enhanced data acquisition functionality and control. Network layers were created on top of TinyOS to support the multi-hop functionality to allow the use of Bluetooth [36,51].

Currently, available sensors and sensor networks support long range and short range communication (433 MHz – 5.9 GHz) [52, 53]. Some health care sensor networks combine short- and long-range communications to monitor patients [54, 55, 40,56]. They utilize mobile phone networks and Bluetooth networks together to achieve better coverage [41,57]. Sensor networks have wide variations in data transfer rates. Some have a low transfer rate (Spike, 35 kbps) and others have a high transfer rate (WLAN, IEE 802.11a, 54 000 kbps). Transfer rate requirements depend highly on the type of sensor application. Some sensor applications require a high transfer rate [58] because they generate large amounts of data that must be dealt with quickly. Examples are sensors that stream video data from remote locations [59].

In summary sensor networks are an active research area, with various groups focusing on different elements of sensor networks; networks, sensors, data acquisition, protocols, performance. [60, 61, 62, 63]

2.2 Software and Protocols

Java portability allows it to run on a wide range of operating systems and devices. Sun Microsystems realized that one size does not fit all; it grouped Java into three main editions, each targeted at a specific range of devices. Java Enterprise Edition (J2EE) is targeted for enterprise servers to create large scalable applications. Java Standard Edition (J2SE) is targeted for desktop applications. Java Micro Edition (J2ME) is targeted for small devices with limited hardware capabilities [64].

Eclipse [65] and Netbeans [66] are among the most popular integrated development environments (IDE) used to build and debug Java applications. Both IDE's are used in this research.

In order to make Java a portable language, Sun created a virtual machine for each specific platform where Java programs can run. The virtual machine concept made it easy to port network Java programs from one edition to another. Java has native network support. It is possible to create applications to support different kinds of networks and protocols. Java has native libraries that support wired and wireless communications. It supports Bluetooth, WiFi, and more. Several popular network protocols and standards are also supported. By default, Java libraries support Transmission Control Protocol

(TCP) protocol, User Datagram Protocol (UDP) protocols, and binary stream communications.

TCP protocol is a reliable protocol used in communication when a reliable connection is required [67]. It allows two hosts to communicate and exchange data streams and guarantees the data delivery [68]. Data packets are delivered in the same order they were sent. In contrast, UDP does not provide guaranteed delivery and does not guarantee packet ordering [69]. Selecting which protocol to choose for a particular application mainly depends on the application requirements. These protocols have proven their value and made their way into Bluetooth and GSM networks.

Bluetooth networks support both TCP and UDP communications [70]. Applications running on the Bluetooth networks can use any of these protocols to send and receive data. The most common way to send TCP and UDP packets over Bluetooth is using Bluetooth Radio Frequency Communications (RFCOMM) [71]. RFCOMM is a transport protocol that provides RS-232 serial port emulation. Bluetooth Serial Port Profile (SPP) is based on this protocol [72,73].

GSM networks are similar to Bluetooth networks and wired local area networks. They support TCP and UDP communication protocols [74-78]. Since wireless networks support the same communication protocol as wired local area networks, applications running on wireless networks can communicate and exchange data with the applications running on wired local area networks.

Application level protocols are created to support specific applications. These protocols can run on top of either TCP or UDP protocols. KREIOS protocol and LayerPro protocol in this research are examples of such protocols. KREIOS is a packet-

oriented protocol created to support data exchange between the sensor used in this research and any other application running in another device [78-81]. LayerPro is a protocol created in this research based on KERIOS protocol to allow global communication between the sensor, the PDA, and the server over Bluetooth and GSM networks.

2.3 KERIOS Protocol

2.3.1 Protocol Overview

The wireless sensor used in this research implemented KERIOS protocol, which was created by [79] and implemented by Arquatis GmbH, Rieden Switzerland [78]. The KERIOS is a packet-oriented protocol between two devices: one acts as a master and the second one acts as a slave; both communicate through a request and response transaction. In this research, the master is the PDA and the slave is the sensor. The detailed description of the protocol is provided in appendix A.

2.4 Near-infrared Spectroscopy

Several methods have been devised for imaging the human brain, in particular Electroencephalography (EEG), Computed Tomography (CT), Magnetic Resonance Imaging (MRI), Functional Magnetic Resonance Imaging (fMRI), Positron Emission Tomography (PET), Single Photon Emission Computed Tomography (SPECT), Near-infrared Spectroscopy (NIRS), and Diffuse Optical Tomography (DOT). These methods vary in their strengths and weaknesses [82]. In recent years, researchers have started using NIRS and DOT, either alone or in combination with other methods, to image brain

functions. The non-invasive nature of the NIRS is appealing to researchers to measure changes in HbO₂ and Hb during brain function activities [83].

Functional Optical Brain Spectroscopy using Near-infrared Light (fNIRS) has been introduced as a new method to conduct functional brain analysis. fNIRS is a method that uses the reflection of infrared light to observe changes in the concentration of HbO₂ and Hb in the blood, and can provide a similar result to fMRI [84]. fNIRS takes advantage of the absorption and scattering of near-infrared light to provide information about brain activities [85]. For a long time, it was thought that it was only possible to collect information from the superficial layers of tissue (e.g., microscopy) due to light scattering. However, about 25 years ago, it was discovered that functional information could be obtained from brain tissue using light shone at the scalp and detected from the scalp [20]. This discovery motivated the development of diffuse optics as a method for brain monitoring. This method has different names: Near-infrared Spectroscopy (NIRS), Diffuse Optical Tomography, and/or Near-infrared Imaging (NIRI). Today, several types of NIRS devices have been built to image brain functions. These devices differ in their capabilities, designs, and costs [82, 86].

The NIRS devices can be classified into three main types: Continuous Wave Spectroscopy (CWS), Time-resolved Spectroscopy (TRS), and Frequency Domain Spectroscopy (FDS). The CWS device consists of a continuous light source, which transmits light waves with constant amplitude, and a detector that locates the attenuated incident light after it passes through the tissues. The TRS device transmits short incidents of light pulses into tissues and measures the light after it passes through the tissues. On

the other hand, the FDS device transmits a sinusoidally modulated light wave into the tissue [82].

Each of these types of NIRS devices has limitations and strengths [87]. CWS has the advantage of low cost; however, with CWS it is difficult to distinguish contributions of absorption and scattering to the light attenuation. FDS, on the other hand, is known for its good spatial resolution, penetration depth, and accurate separation of absorption and scattering effects. Nevertheless, FDS is significantly more expensive than CWS. As for TRS, although theoretically, it can provide a better spatial resolution than FDS, it has a lower signal-to-noise ratio. Since TRS requires short pulsed lasers and photon counting detection, it is the most expensive type of the NIRS instrumentation. Despite the advancements in NIRS technology, NIRS still has limitations, such as the short path length and the artifacts' movements during measurements.

2.5 Tissues Optical Properties

Absorption and scattering are the main physical processes affecting the transmission of light photons in tissues. Light photon absorption and scattering causes the light intensity to decrease. Both absorption and scattering are wavelength dependent. The amount of absorbed light photons is also impacted by the concentration of blood HbO_2 and Hb in tissues which vary in time, reflecting physiological changes in tissues' optical properties [84].

When light photons travel through tissues, they are scattered several times before finally reaching the receiver. Scattering increases light optical path length, causing

photons to spend more time in tissues which in turn affects the tissues' absorption characteristics.

Despite the fact that both absorption and scattering play a major role in light transmission, scattering is more dominant than absorption. When light travels through tissues and blood, photon absorption leads to a loss of energy to tissues and blood chromophores, or induces either fluorescence (or delayed fluorescence), or phosphorescence. The main substances of biological tissues that contribute to light photon absorption in the near-infrared light are water, fat, and hemoglobin. While water and fat remain fairly constant over a short period of time, the concentrations of oxygenated and deoxygenated hemoglobin change according to the function and metabolism of the tissues. Thus, the corresponding changes in absorption can provide clinically useful physiological information.

2.6 Modified Beer Lambert Law

Near-infrared light, in the range of 700-900 nm, can travel relatively deep into body tissues. It is also worth mentioning that such light can easily travel through soft tissues and bones, such as those of neonates and infants. Therefore, it is suitable to use near-infrared devices to monitor brain activities or other oxygen-dependent organs in this category of humans [88].

NIRS relies on a simple principle: light in the range of near-infrared light emitted on the organ of interest passes through the different layers above the organ. When it passes through the tissues, light photons go through physical interactions, such as

scattering and absorption that lead to a loss of energy in the emitted light. When the remaining light exits the organ, it is measured by a detector.

In neuroimaging applications, the light is injected through the scalp, so the photons pass through several layers of tissue surrounding the brain, such as the scalp, skull, Cerebrospinal Fluid (CSF) and meninges. Then, the NIR light reaches the brain and the blood vessels, and backscattered light gets detected by a set of detectors. The light in this case follows the so-called banana-shaped path due to scattering effects caused by the tissues. Due to the fact that water and lipids are relatively transparent to near-infrared light and the optical properties of the layers surrounding the brain and blood are fixed within a given period of time, it was found that light is mainly absorbed by oxygenated and deoxygenated hemoglobin. Here, it must be noted that the scattering of the near-infrared light in the human tissues is much larger than its absorption, while absorption of this kind of light is much larger in the blood. This leads to the belief that the optical properties of the blood, which in fact change based on the amount of oxygen in the blood, can play a vital role in determining the amount of backscattered light from the brain. The amount of blood volume and blood oxygen concentration can be an indicator of hemodynamic activities that are related to brain functions. Analyzing the amount of backscattered light during the oxygenation and deoxygenation process of the blood flow in the brain can lead to a better understanding of the brain function [89].

NIRS measures the optical properties of HbO_2 and Hb in near-infrared light. The effects of the changes in concentration levels of HbO_2 and Hb in the blood stream on light absorption can be described by the Beer-Lambert's Law. A Modified Beer-Lambert

Law can be used to predict the amount of blood chromophores (HbO₂ and Hb) in tissues [86].

Figure 2.1: Modified Beer-Lambert Law, Light Photon Propagation in Tissues Between Source and Detectors

$$I_{out} = I_{in} \cdot 10^{-OD_{\lambda}} \quad (1) \quad [86,90,91,92,93]$$

I_{out} : detected light.

I_{in} : source light.

Equation 2.1: Light Attenuation between Sources Light and Detected Light

$$OD_{\lambda} = -\log_{10} \frac{I_{out}}{I_{in}} = \text{attenuation} = A_{\lambda} + S_{\lambda} \quad (2) \quad [86,90,91,92,93]$$

OD_{λ} : optical density for wavelength λ

Equation 2.2: Optical Density for Wavelength λ

$$A_{\lambda} = \sum_{i=Hb, HbO_2} \epsilon_{i,\lambda} C_i L_{\lambda} \quad (3)$$

[86,90,91,92,93]

A_{λ} : light absorption in blood HbO₂ and Hb [42]

$\epsilon_{i,\lambda}$: extinction coefficient of blood chromophore for wavelength λ

C_i : blood chromophores concentration

L_{λ} : path length of light at λ

Equation 2.3: Absorption of Light in Blood

$$L_{\lambda} = d \cdot DPF_{\lambda} \quad (4)$$

[86,90,91,92,93]

L_{λ} : path length

d : distance between the light source and the light detector

DPF_{λ} : differential path length factor

Equation 2.4: Path Length [86]

$$DPF_{\lambda} = \frac{1}{2} \left(\frac{3\mu'_{s,\lambda}}{\mu_{a,\lambda}} \right)^{1/2} \left[1 - \frac{1}{1 + d \cdot (3\mu'_{s,\lambda} \mu_{a,\lambda})^{1/2}} \right] \quad (5)$$

[86]

μ_a, λ : absorption coefficient at wavelength λ

μ_s', λ : reduced scattering coefficient at wavelength λ

Equation 2.5: Differential Path Length
[86]

$$\Delta OD_\lambda = OD_{\lambda, final} - OD_{\lambda, initial} = \sum_{i=Hb, HbO_2} \epsilon_{i, \lambda} \Delta C_i d DPF_\lambda \quad (6)$$

[42,123,124,125,126]

Equation 2.6: Differential Value

$$\overline{\Delta OD} = \overline{M} \times \overline{\Delta C} \quad (7)$$

where

$$\overline{\Delta OD} = \begin{bmatrix} \Delta OD_{\lambda_1} \\ \Delta OD_{\lambda_2} \end{bmatrix}, \overline{\Delta C} = \begin{bmatrix} \Delta C_{HbO_2} \\ \Delta C_{Hb} \end{bmatrix} \text{ and}$$

$$\overline{M} = d \cdot (\overline{\epsilon} \times \overline{DPF})^T. \quad (8)$$

[42,123,124,125,126]

$$\overline{\epsilon} \text{ and } \overline{DPF} \text{ are defined as } \overline{\epsilon} = \begin{bmatrix} \epsilon_{HbO_2, \lambda_1} & \epsilon_{HbO_2, \lambda_2} \\ \epsilon_{Hb, \lambda_1} & \epsilon_{Hb, \lambda_2} \end{bmatrix} \text{ and}$$

$$\overline{DPF} = \begin{bmatrix} DPF_{\lambda_1} & 0 \\ 0 & DPF_{\lambda_2} \end{bmatrix}.$$

Therefore,

$$\overline{\Delta C} = \overline{M}^{-1} \times \overline{\Delta OD} \quad (9)$$

[86]

Equation 2.7: Transformation from Light Output Change to Change in Blood Chromophore Concentrations

Equation 9 “provides a transformation from light output change to a change in blood chromophore concentrations. By using blood chromophore concentrations, we define two parameters” [42]:

$$OXY = \Delta C_{HbO_2} - \Delta C_{Hb} \quad (10)$$

and

$$BV = \Delta C_{HbO_2} + \Delta C_{Hb} \quad (11)$$

Equation 2.8: Blood Chromophore Concentration Parameters

“OXY and BV are estimates proportional to, respectively, oxygenation and blood volume changes in the tissue due to hemodynamic activation”. [86]

2.7 Anatomy and Physiology

Blood carries oxygen and nutrients to tissues. Also, it carries carbon dioxide and other products of metabolism away from tissues, so the body can eliminate them. Red blood cells contain hemoglobin, which is the main oxygen transporter. When the red blood cells pass through the lungs, they collect oxygen where it becomes bound with the hemoglobin. Furthermore, red blood cells release carbon dioxide to the lungs. Blood vessels form a comprehensive network inside the body where they deliver blood to different tissues and organs. Arteries, arterioles, and capillaries deliver oxygenated blood to tissues whereas veins and venules collect deoxygenated blood from them.

The human brain is protected by several layers. These layers provide a safe and secure environment for the brain. Near-infrared light, used to measure changes in the blood oxygenation, has to pass through all the protective layers: scalp, periosteum, skull, and the meninges (Figure 2.2). The meninges contain three layers: dura mater, arachnoid mater, and pia mater. [94]

Figure 2.2: The Brain's Protective Layers[18]

3 CHAPTER 3 - MATERIALS AND METHODS

3.1 Hardware Architecture

This system, developed for this research, consists of three main hardware components. The first component is a Bluetooth wireless sensor (built by Arquatis GmbH, Rieden, Switzerland), which is the data acquisition device [79,80]. The second component is a PDA and is the main controller for the measurement process and the data communication bridge between the sensor and the central computer. The third component is a central computer (Server, or Host Computer, or PC) that stores the data for later analysis. See Figure 3.1 for a full display of the system's architecture.

Two different ranges of communication are used in the developed system. First, the communication between the sensor and the PDA is carried over a Bluetooth network. The signal range between the PDA and the sensor is approximately 10 m (short range). This short range is enough to perform bedside monitoring without the need to carry the PDA. The other range of communication occurs between the PDA and the central computer and is carried over the GSM network (wide range). The range of the GSM is very wide indeed since the system employs the mobile phone network with roaming feature. Technically, it is possible to monitor a test subject wearing the sensor in any part of the world as long as they are within the range of a GSM network with roaming capabilities.

The PDA and the sensor are light weight devices that make it possible to carry them easily. The sensor has a set of programs developed in the C language required to

enable the data acquisition and data transmission. The PDA runs the Java ME program that performs the data transmission between the sensor and the host PC. The host PC works as a server and a database server as well. Additionally, the PC is configured with a public IP address to make it accessible through the Internet and to the GSM network. The communication between the PDA and the sensor is bidirectional and the communication between the PDA and the PC is unidirectional – from the PDA to the server.

The combination of these communication technologies allowed the creation of a fully mobile system for Functional Optical Brain Spectroscopy using Near-infrared Light (fNIRS) technology extending the range and the mobility of an existing solution [78-81].

Figure 3.1: System Architecture

3.1.1 The Sensor

The sensor is a wireless near-infrared imaging sensor developed by Arquatis GmbH, Rieden Switzerland [78] (see Figures 3.2-3.5). The sensor quantifies changes in the concentration of HbO_2 and Hb in tissues and sends information to a computer using

wireless Bluetooth technology. The sensor has the following components: four light sources (Figure 3.5); each light source has two LEDs (Light Emitting Diodes) emitting at 730 and 830 nm. It has four light detectors (PIN silicon photodiodes) with a sampling rate of 100 HZ. An analog LED controller controls the emitted light. A microcontroller has an analog/digital (A/D) controller to manage the light intensity signal detection and the conversion of the signal into data. The accuracy of the A/D is 12 bits. The Bluetooth transceiver sends and receives data between the sensor and the Bluetooth networks within a range of 3 m. The sensor requires a power supply and uses a 3.7 volt rechargeable lithium-ion battery that can last up to 3 hours. The sensor's total weight is about (40 grams); its dimensions are (90 x 34 x 20 mm). To achieve better measurements, the sensor components are mounted on a rigid-flexible printed circuit board (PCB). The sensor employs programs written by T.L. Muhlemann from the Swiss Federal Institute of Technology Zurich. The programs in the sensor are responsible for data acquisition, LEDs control, and performing data exchange and communication.

Figure 3.2: Sensor Components
[79,80]

**Figure 3.3: Sensor Front (left) &
Figure 3.4: Sensor Back (right)**

**Figure 3.5: Sensor Light Source and Light Detector Distribution
[79,80]**

3.1.2 The Mobile Phone (PDA)

The personal digital assistance (PDA) device is a commercially available cell phone with data access capabilities, Bluetooth communication support, and Java support. The mobile phone used in this project is a Nokia E62 smart phone (Figure 3.6). It has extensive features and capabilities, however not all are necessary to run the developed system. It supports the following Bluetooth profiles SPP. The developed system uses only the SSP profile to carry out the communication between the sensor and the mobile phone. The developed system uses mobile information device profile (MIDP) v2.0, connected limited device configuration (CLDC) 1.1, and the optional Java package for Bluetooth (JSR 82).

Figure 3.6: Mobile Phone (PDA)
[95]

The phone has single ARM 9 CPU with clock rate of 233 MHZ. The available memory allows the system to acquire and save data for at least 8 continuous hours; it has 80 MB by default. The battery can last up to 5.5 hours when the system is in full use.

3.1.3 The Host PC (Server)

The host PC is a regular desktop personal computer with 2 GB RAM and dual core Intel processor.

3.2 LayerPro Protocol

Initially the system used HTTP protocol as a data encapsulation protocol. HTTP protocol is designed to be a request-response protocol to transmit text based data. Therefore makes it not suitable for binary transmission without adding performance overhead.

This research required continuous fast binary data upload. After reviewing existing upload protocols and approaches, we came to conclusion that a new protocol is needed to be created. Performance and native binary upload were key requirements for the protocol. Based on the requirements the protocol was design and extended KERIOS protocol. The protocol achieved the requirement through minimizing the control data and the number of the overall transactions. Moreover the protocol packet was designed to hold binary data which reduced the data representation overhead.

3.2.1 Protocol Overview

This protocol (see Figure 3.7) for this research was created to encapsulate only the acquired data and send it to the server; it is based on KERIOS protocol. The extension was necessary to ensure data integrity and improve KERIOS protocol parsing. LayerPro carries only the KERIOS data packet and adds 3 extra bytes as a sequence number. The sequence number ensures that the packets are continuous and no packet loss will occur during transmission. Moreover LayerPro has a fixed length; it has 35 bytes, while KERIOS has a variable length. These modifications made LayerPro packet parsing easier

and faster on the server. This protocol is stateless and supports limited transactions, of which it allows three: open, close, and send.

Figure 3.7: Protocol Stack

3.2.2 Packet Format

This protocol has two parts: head and tail (see Figure 3.8). The head contains 3 bytes representing the transmission sequence number. The tail contains the KERIOS data packet. In this protocol the fixed length is used to determine the end of the packet.

Figure 3.8: LayerPro Packet Format

3.2.3 Transactions and Packet Flow

The first packet sent in this protocol is an open transaction packet. This transaction packet indicates to the server the beginning of the transmission. The sequence number value in the packet head is “00 00 00”, and no data is in the packet tail. The open

transaction packet is followed by a send transaction packet that contains the acquired data from the sensor. The close transaction packet indicates to the server the end of transmission. The sequence number value in the packet head in this transaction is the last data sequence number with no data in the packet tail. Figure 3.9 demonstrates these transactions and flow.

Figure 3.9: Layer Pro Transactions

3.3 Software Architecture

The system software architecture (Figure 3.10) has three major layers: a data acquisition layer (DAL), a control layer (CL), and a data storage layer (DSL). The DAL software component in the sensor controls data acquisition and packet transmission. It is

composed of a set of programs that implement the data communication protocol, the RFCOMM Bluetooth protocol, and the sensor's low-level controls. The second layer (CL) resides on the PDA and acts as the central control unit for the application. The majority of the system components reside in this layer. The third layer (DSL) is mainly used to accept connections from the PDA and store the received data packets in the server for later analysis. The PDA creates a persistent connection with the sensor and with the PC during the duration of the measurements. The system is designed to support a wide range of measurements and acquisition activities. Several types of tests can be performed using the system without the need to modify the programs. Most of the components are designed to be configuration-driven. The system architecture provides high interoperability between heterogeneous hardware and software.

Figure 3.10: The System Software Architecture

3.3.1 Overall System Component Interaction

The overall system interaction is depicted in the sequence diagram in Figure 3.11. The sequence diagram shows the steps and the sequences of the steps in relation to time.

The system administrator starts the server. The sensor is carried by a test subject. The test subject fixes the sensor on a test area and starts the measurement process from the PDA software. The sensor emits light directly in the test area and starts the data collection. The sensor transmits the data back to the PDA. The PDA receives the data packet and forwards it to the server and at the same time the PDA will save a local copy

to the file system. The server receives the data packet and displays the data on the screen at the same time it saves the packet in the server's local file system. Once the test is completed, the data is extracted and analyzed.

Figure 3.11: The Overall System Sequence Diagram

3.3.2 Sensor Software Architecture

3.3.2.1 Sensor Software Component

The sensor software components were developed previously by another team as part of a Master's thesis [79] and are presented here for completeness. The sensor software components are a set of C programs. Those software components will be used as is without any modifications. The sensor has three main software components (Figure 3.12): Data Acquisition and Control, KREIOS, and Bluetooth.

Figure 3.12: Sensor Components Diagram

- **Data Acquisition and Control:** This component is generally responsible for data acquisition and the overall control of the sensor. The main functionality of this component is to enable the light-source power supply,

trigger the measurements, request device configuration, request current measurements, reset the device, turn the light-source power supply off, stop measurements, and turn the power device off.

- ***KREIOS:*** This component is responsible for creating a KREIOS data packet. It assembles and disassembles the commands and the data from and to the sensor in the form of a KREIOS packet which was designed by [79] and implemented by Arquatis GmbH, Rieden Switzerland [78]
- ***Bluetooth:*** This component is responsible for the Bluetooth communication between the sensor and any other Bluetooth devices that have Bluetooth protocol implemented.

3.3.2.2 Sensor Software Component Interaction

The sensor component interactions follow well defined steps depicted in Figure 3.13. The PDA opens a Bluetooth connection with the sensor and sends a command packet. The Bluetooth component in the sensor accepts the connection and receives the command packet. The command packet gets parsed by the KREIOS component. The parsed command gets dispatched to the data acquisition and control component. The component executes the command and the sensor starts sending data. During the data acquisition step, the data acquisition and control component capture the light intensity and convert it to bytes. The KREIOS component creates packets from the acquired bytes

and sends it over to the PDA using the Bluetooth connection. Figure 3.13 shows the first two main use cases in the system, which are open connection and start measurements.

Figure 3.13: Open Connection and Start Measurements Use Cases

3.3.3 PDA Software Architecture

3.3.3.1 PDA Software Components

The software in the PDA is modular and was designed for this project to be reusable. For example another sensor can be used instead of the near-infrared sensor to collect data. In this case most software components can be used without modifications. Only one component will be needed to be added to interface with the new sensor.

The PDA software consists of seven software components (Figure 3.14). Each component provides well-defined system functionality and interfaces. Components in the system are represented by Java packages: PDAProgCtrl; PDAUI; PDAUtil; PDADA; BluetoothCom; LayerPro and ComManager.

Figure 3.14: PDA Components Diagram

- **PDAProgCtrl:** This component controls the program flow and the sequence between all the components; it receives requests from the user interface and dispatches the commands to the different software components. This component acts as a traffic controller between all the software components.
- **PDAUI:** This component contains the user interface; there are multiple screens that the end-user and the application administrator can use. The

end-user screens have all the necessary fields and forms to allow the end-user to control the measurement process. The end-user can select the monitoring type, start the monitoring, and stop the monitoring. The administrator screens allow the administrator to control the sensor low-level functions. Using the screens, the administrator can enable the sensor's light source power supply, turn the sensor's light source power supply off, request the sensor configurations, and reset the sensor.

- ***PDAUtil:*** This is a utility component; it provides helper functionality to the other components. It has logging, tracing, timers, and validation functions. Moreover it has a secure file naming schema and linkage between the test and the test subject. The secure file name schema is used to maintain the confidentiality of the test subject.
- ***PDADA:*** This component acts as a data access layer between the other software components and the PDA file system. This component provides data persistence functionality. Data persistence is very important for this system because GSM data connection is not always available. When the connection drops during the measurement process, the system will switch to offline mode and continue the operation. The acquired data will be saved to the PDA local file system using the functionality available in this component. The system-user can afterward download the saved data to the server and import the data into the system using the administrator

console. In general, this component is responsible for all data persistence operations.

- **BluetoothCom:** BluetoothCom was created to manage the Bluetooth connection between the PDA and the sensor. The communication between the PDA and the sensor uses RFCOMM Bluetooth protocol; this component plays two roles: Bluetooth client and Bluetooth server. In the role of the Bluetooth client, the PDA searches for the sensor. Once it finds it, it opens a serial connection. Then it retrieves the input stream and the output stream of the sensor and makes them available to the application. Through the connection and the input/output streams, the application sends command packets and receives data packets from the sensor.
- **LayerPro:** This component provides an implementation for the data communication protocol between the PDA, the sensor, and the server. All data exchanged between the sensor, the PDA, and the server must be encapsulated in a LayerPro packet. LayerPro is a modified protocol based on KREIOS protocol [80]. In order to allow the data exchange to happen between the sensor and the PDA, the PDA has to have the LayerPro protocol implemented. This component assembles the LayerPro packet that needs to be sent to the sensor; it also disassembles the LayerPro packet that is being received from the sensor. Moreover, when the PDA is in offline mode, the data gets saved in a file in LayerPro packet format.
- **ComManager:** This component establishes and manages the connection between the PDA and the PC. It forms an abstraction layer between the

application layer and the communication functionality where it provides a flexible physical connection configuration. Furthermore, this component makes communication transparent to the system if the connection gets lost during the transmission. In the case where the GSM data connection gets lost, this component switches the system to offline mode and starts saving the data to the PDA local file system. The system will continue functioning as normal without any interruption. Also, this component allows the system to work with other wireless connections, such as WiFi or any other available wireless connection or protocols, without any change to the other software components (Figure 3.15).

Figure 3.15: Communication Manager Design

3.3.3.2 PDA Software Components Interaction

All user interactions (Figure 3.16) in the system are initiated by the User Interface Component (PDAUI) that is controlled by the program control component (PDAProgCtrl). Program control calls the LayerPro component to create command and

data packets. All commands are encapsulated by a LayerPro packet before they are sent/received to/from the sensor; this is performed by the LayerPro component. A LayerPro packet is sent and received over the air using the Bluetooth communication component. When data is collected from the sensor it is sent to the server using the communication manager component (ComManager); then a local copy of the packet will be saved to the mobile local file system using the Mobile Database Access Component (PDADA).

Figure 3.16: PDA Main Use Cases Sequence Diagram

3.3.4 Server Software Architecture

3.3.4.1 Server Components

The server has four major components (Figure 3.17): MainCtrl, LayerPro, SRVDA, and SRVUI.

Figure 3.17: Server Components Diagram

- **MainCtrl:** In the server side all requests from the PDA to the server go through a central controller that is responsible for accepting requests from the PDA.
- **SRVDA:** This component is responsible for saving the received data from the PDA.
- **LayerPro:** This component is LayerPro protocol implementations.

- **SRVUI:** It contains the graphical user interface. It has the administration user interface that allows the administrator to start and stop the server. Moreover, it provides the user with the ability to see the content of the received packets in near real time. Additionally, it has the user interface to import offline data to the system.

3.3.4.2 Server Component Interactions

The system administrator starts the server using the user interface, which calls the SRVUI component. The server in the background starts a TCP/IP socket server that listens at a specific port and waits for any incoming requests from the PDA. The PDA opens a connection with the server and starts sending data packets in form of LayerPro packets. The server's main controller component (MainCtrl) dispatches the received packet to the LayerPro (LayerPro) component. LayerPro parses the packets and extracts the data. The server data access component (SRVDA) saves the data to the local file system. At the same time, the server user interface component (SRVUI) plots the extracted data. Also SRVUI provides the researcher with the ability to import data that has been collected during the offline measurements. Figure 3.18 shows the server components interaction.

Figure 3.18: Server Components Sequence Diagram

3.4 Software Design and Implementation

3.4.1 PDA Software Detailed Design

This system has been built using Java language mobile edition [64]. A typical Java program consists of a set of packages, classes, methods, and properties. This system has eight packages based on the system's main functionality. Each package has a set of classes that perform together the intended functionality for the package. Figure 3.19 shows all the PDA packages.

Figure 3.19: PDA Packages

The Bluetooth package (`com.salah.mnirs.pda.bluetoothCom`) encompasses all the classes that implement the Bluetooth system and sensor-related functionality – Sensor class (`Sensor.java`), Bluetooth serial port reader class (`BluetoothSPPReader.java`), sensor packet buffer class (`SensorPacketBuffer`), and Bluetooth communication manager class (`BluetoothComManager.java`).

The communication manager package, `com.salah.mnirs.pda.comManager`, contains the classes that deal with wide-range network communication. This package has one major class: `Server.java`. The server class has the common functionality for all the communication aspects of the server.

The LayerPro package, `com.salah.mnirs.pda.layerpro`, represents the main communication protocol in the system and includes – Bluetooth sensor constants class (`BTSensorConstants.java`), control commands (`ControlCommands.java`), device types class (`DeviceTypes.java`), error code class (`ErrorCodes.java`), packet type class (`PacketType.java`), and position offset constant class (`PosOffsetConstants.java`).

The LayerPro packet package, `com.salah.mnirs.pda.layerpro.packet` contains all the necessary classes to represent the protocol packet (Figure 3.20) – Bluetooth sensor packet

class (BTSensorPacket.java), operation acknowledgement class (OPCAck.java), operation acknowledgement payload class (OPCAckPayload.java), operation control command class (OPCCtrlCommand.java), operation empty class (OPCEmpty.java), operation Error class (OPCError.java), operation no acknowledgement class (OPCNack.java), operation no acknowledgement error class (OPCNackError.java), operation near infrared data class (OPCNirsData.java", operation not specified class "OPCNotSpecified.java), operation status information class (OPCStatusInfo.java), operation wrapped class (OPCWrapped.java), packet class (Packet.java), and packet codes class (PacketCodec). All packet classes are extended from the main packet class, "Packet", which contains the common functionality shared by all packet types. Figure 3.20 shows the class diagram that represents the relationship between the protocol classes.

The package PDAUI (com.salah.mnirs.pda.pdaui) encapsulates the system's graphical user interface functionality. The system requires some helper classes to perform functionalities not specific to any component in the system, such as logging, writing files, etc. These functionalities are provided by the com.salah.mnirs.pda.pdautil package. The last important package is the PDA data access package (com.salah.mnirs.pda.pdda). It has the classes that persistent the to the PDA local file system. The entry point for the system is located in the com.salah.mnirs.pda.pdaprogctrl package, which has the main control over the system.

Figure 3.20: LayerPro Protocol Packet Class Diagram

3.4.2 PDA Software Detailed Implementation

3.4.2.1 PDA Main Controller

The measurement process is completely controlled by a multi-threaded class, Controller.java (Figure 3.22). The class follows a well-defined algorithm (Figure 3.21).

The class (Code Snippet Appendix D.1) runs in its own thread to achieve better performance. The run method starts with checking if the user selection is to perform the measurements online or offline. The graphical user interface offers the user two modes of operations: online mode and offline mode. Online mode allows the system to send the acquired data to the server. Offline mode saves the data to the PDA local file system for later download. The controller's class main method (run) starts with checking the user selection. In the case of the offline mode, the system will not open any connection. However, in the online mode, the system will open a connection with the server using the available network connection (GSM, WiFi, etc.) based on the user selection. When the system is in the online mode, it is important to open a connection with the server before any other operation to ensure that the server is ready to receive the data when the sensor starts sending it. Once the connection to the server is ready, the system creates a sensor object and establishes a Bluetooth connection with the sensor using the sensor open Bluetooth method. After the connection is established with the sensor, the system uses the sensor object to start the measurement, using the sensor start measurements method. All incoming data from the sensor is buffered in a temporary buffer in the PDA program. In order for the main controller to be able to get the data, the main controller object needs to connect to the sensor buffer using the sensor's connect to sensor buffer method after successfully issuing the start measurements command. The main controller starts an infinite loop to read from the sensor buffer. The controller reads one packet at a time from the buffer using the read packet method. The packet sequence gets displayed to the user and then the complete packet gets sent to the server. The system keeps sending data to the server until the user issues a stop measurement command from the graphical user

interface. The infinite loop will stop and the system will start executing the necessary command to close the connection with the Bluetooth device and the connection with the server. At this stage, the sensor will power off and the system exists. It is worth mentioning that the sensor continuously sends data to the PDA buffer till the stop measurement command is issued.

Figure 3.21: Measurements Algorithm

Figure 3.22: Controller Class Outlines

3.4.2.2 PDA Sensor Communication Controller

To fully control the sensor, a set of Java classes has been created. The most important classes are `Sensor`, `BluetoothComManager`, `BluetoothSPPReader`, and `SensorPacketBuffer`. Each of these classes plays a vital role in achieving the functionality. Each one of these classes solves a challenge in this research. In this section, I will describe these major classes; I will start with the main class, the `Sensor` class. Code Snippet Appendix D.2 shows the complete sensor class.

The sensor class (Figure 3.23) has several methods where some of these methods encapsulate the other major classes and the main protocol. `Open Bluetooth` method opens a serial port connection with the sensor. It forms a Unified Resource Locator (URL that specifies the IP address of the sensor and the Bluetooth protocol that will be used to connect to it. The `Open Bluetooth` method calls another method to establish the physical connection using the connector class method (`Open`) through passing the sensor URL. If the connection with the sensor is created successfully, an object that holds the connection will be returned. The returned connection object is used to retrieve the input and output stream of the sensor. The input and output streams are used to read and write packets from and to the sensor. `Close Bluetooth` method is used to close the input and output

streams. Further, this method closes the physical connection with the sensor. The start measurements method creates a command packet based on the protocol to instruct the sensor to start the data acquisition process and send data. Once the packet is created, it gets written to the output stream of the sensor where it gets sent to the sensor which then parses the packet and performs the necessary steps to start the measurements process. Once the sensor acknowledges the completion of the command, the data starts arriving in the input stream. The stop measurements command performs the same step as the start measurement command. It creates a command packet and writes it to the sensor output stream. The sensor reads the packet, performs the operation, and acknowledges the completion. At this point, the data stops coming from the sensor. By issuing a start measurement command, the data can start to flow again. The Power off Bluetooth method also sends a command packet that causes the sensor to power off. After this command gets issued, a manual power on is required before starting the measurement again.

The sensor transmits data at 100 HZ, which means 100 packets per second. In order to read every single packet from the sensor, the PDA has to read the data at that speed and then sends it to the server. Due to the limitation of the GSM network, the system cannot send the data at the same rate as it receives it from the sensor. Therefore, a buffering mechanism is required to ensure no packet is lost. A special buffer class was created to handle this challenge through implementing a buffer flush algorithm (Figure 3.24). The ConnectToSensorBuffer method was created to connect to the buffer class. Another method was created to read from the buffer class, readPacket; this method waits for the data in the buffer class and reads it when it is available.

The buffer class implements a growable array of objects. It is just like an array and can be accessed using an index. It acts as a stack where the first pushed object gets pulled first. The read method reads the object from the array and removes it once it is read. The size of the array is set to 250 000 objects. The size is pre-set for performance considerations and to allow the system to acquire data for 60 minutes. The size of the array poses a limitation on the system. If the system cannot read the data and send it to the server fast enough, this array will get full and the system will flush it, which means data will be lost. In the original design, the size of the array was set for 5 000 objects. This size causes a lost data packet every 5 minutes because the system cannot read and send the data to the server fast enough.

One of the major classes is a class that wraps the serial communication with the sensor. BluetoothSPPReader class reads the data stream from the sensor and stores the data in the buffer. The sensor class creates a BluetoothComManager class to abstract the communication between the PDA and the sensor. One of the major functions of this class is to parse and sort the received packet from the sensor by reading the buffer and creating a LayerPro packet that the PDA can understand. This class also encapsulates the protocol to be able to parse all the incoming packets from the sensor. The class has a method called sortPacket where it sorts packets based on their type. The command packets get handled through command interaction between the sensor and the PDA. Data packets are saved in another buffer to hold the data when it gets sent to the server or saved to the PDA local file system later.

Figure 3.23: Sensor Class Outlines

Figure 3.24: Buffer Flash Algorithm

3.4.2.3 PDA Server Controller

There are several approaches to send data to the server. Initially the system used HTTP as the main protocol to encapsulate the acquired data. Shortly after implementing the HTTP approach. The system was so slow that it could not transmit the data in a timely fashion. Also it experienced data loss due to the fact that the transmission was slow; therefore, the data would stay in the PDA data buffer for a longer time than anticipated causing the PDA to become full and eventually flushes it. Data transmission between the PDA and the server goes through multiple layers over which the developer has no control. The PDA system sends the data over GPRS protocol and then this data is passed over to the GSM connection. Finally, it gets delivered to the server. The GSM network gives data transmission lower priority than voice transmission, which makes the data transmission rate lower when the network is busy with voice transmission. On analysis, several factors caused the implementation of the HTTP protocol to have poor performance. The acquired data is binary data. Therefore, to transmit it over HTTP the data requires further manipulation in the PDA to make it ready to be transmitted over the HTTP protocol. Moreover the approach encapsulated each packet (32 bytes) with an HTTP request and then waited for the reply. The system generates a large number of packets, 100 packets per second. On the server side, an off-the-shelf web server (Tomcat web server) was used to accept the requests from the PDA which was slow.

Performance for the developed system is very important due to the fact that the sensor transmits data in near real time, making the overall system time-dependant. Every millisecond in the implementation counts. Also, on the server side the acquired data is

plotted on the screen during the measurement process and monitored by the researcher. If the data is not transmitted fast enough, the plot will be far behind the actual data and will not reflect the real status.

To solve the problem several steps were taken: The HTTP implementation was totally discarded and was replaced with LayerPro implementation; The off-shelve server was replaced with a custom socket server; On the PDA side, the HTTP client was replaced with a custom socket client that runs in its own thread; the HTTP protocol was replaced with LayerPro protocol; the data is sent from the PDA to the server in binary format as it arrives from the sensor without further manipulation; and sent continuous stream of data without waiting for a reply from the server until the measurements is completed.

3.4.2.4 PDA Socket Client

The socket client (Figure 3.25) is a TCP/IP socket client. It opens a socket connection with the server. The OpenServer method in the server class prepares the connection parameters, opens the connection with the server, returns the server output stream, and makes it available to the system in its own thread. The class (Code Snippet Appendix D.3) keeps the connection open during the entire measurement process. The CloseServer method closes the connection when the measurements process is complete. The SendToServe method sends a binary LayerPro packet to the server. In the case of a communication interruption, the system will switch to the offline mode and save the data to the local file system to prevent data loss. During the switch, there is a chance that the transmitted packet at that time will be lost. A trace files is used to track the lost packet.

The offline data can later be imported to the server and merged with the rest of the measurement data. The client has two methods for checking the connection status and ensuring that the system is aware of the connection state. The client can take the necessary steps to switch the system mode when it is required. Figure 3.26 shows the complete communication algorithm.

Figure 3.25: PDA Socket Client Outlines

Figure 3.26: Client Server Communication Algorithm

3.4.2.5 LayerPro Protocol

LayerPro is the protocol created in this research. To implement the protocol, several classes have been developed. Packet class (Code Snippet Appendix D.4) is the core class (Figure 3.27). It is the base class for most classes that implement the protocol.

Figure 3.27: LayerPro Protocol Packet Class Outline View

3.4.3 Server Software Detailed Design

The system was built using the Java language standard edition [64]. This system has four packages (Figure 3.28). Each package has a set of classes that perform together the intended functionality for the package. The main controller (`com.salah.mainctrl`) encompasses the classes that implement the main socket server, which is used to accept the request from the PDA. The main classes in this package are LayerPro server (`LayerProServer.java`) and sensor raw packet `SensorRawPacket.java`. The LayerPro package (`com.salah.layerpro`) and LayPropacket package (`com.salah.layerpro.packet`) are the layerPro Protocol implementation. The user interface package (`com.salah.srvui`) includes all the classes that are required to provide the user interface in the server, such as plotting the data, monitoring the measurements, and importing offline line data from the PDA. The last package in the data access layer package is ("`com.salah.srvda`"); it provides all low level data persistence functionality for the server.

In the initial phases of the research, the system was designed to use tomcat as a web server in combination with Java servlet technologies and MySql database to implement the HTTP approach mentioned earlier. The tomcat and Mysql implementation was replaced with a custom build server.

Figure 3.28: Server Packages

3.4.4 Server Software Detailed Implementation

3.4.4.1 LayerPro Socket Server

LayerPro Server (Figure 3.29) is implemented as a Java socket server. The server listens at a preconfigured port and accepts connections from the master (the PDA). Once the connection is established between the PDA and the server, the server starts reading data from the input stream. The received data is encapsulated in LayerPro protocol. LayerPro packages parse the received packets and pass it to the graphical user interfaces to be displayed in a near real time plot. At the same time, the parsed packet gets passed to the persistence layer components to save it for further analysis. Figure 3.30 shows the complete algorithm and Code Snippet Appendix D.5 shows the server code.

Figure 3.29: LayerPro Server Main Class Outlines

Figure 3.30: LayerPro Main Algorithm

4 CHAPTER 4 - RESULTS AND DISCUSSION

We have shown that it is feasible to provide full mobility monitoring of brain functions in humans in everyday environment using a near-infrared light sensor by employing a set of heterogeneous networks accessed using a methodology we have outlined. To validate the system's functionality, we used it to monitor brain functions during smoking outside the lab environment. Again, it was used to collect changes in oxygenation concentration levels in the brain during breath holding, and finally was used to measure the changes in oxygenation concentration levels in dogs' brains when presented with their favorite toys.

The research contributes a fully mobile system to monitor human brain activities outside the lab environment. Additionally, our work contributes a protocol to stream binary data in near-real time. To validate the system's basic data functionality, more than 100 tests were performed.

This chapter presents the results in two sections; the information technology-related results and the biomedical application-related results.

4.1 Information Systems Results

4.1.1 Summary

The tests were focused on performance, data integrity, availability and the effectiveness of the developed protocol and algorithm. Our results indicate that the system has achieved its goals.

4.1.2 System Data Transmission Performance

The system worked in all cases, but different amounts of delay were experienced in the data transmission. In this section, we discuss the factors affecting our system's transmission performance and explain what we mean by "near real time".

A real time system can be defined as a system that can perform operations within a predefined time and constraints regardless of the system's load [96]. In this research, we define our system to be a near-real time system, because the system transmits the data with a delay that varies according to the load of the involved networks, which we do not have control over. While performing the experiments, we encountered delays in data transmission. Typically, the delays vary between 1 to 5 seconds as illustrated in Table 4.1. The delay is impacted by the networks' speed during the time of day the experiments were performed. Because GSM networks give higher priority to the voice channel and consequently lower priority to data channel these delays are unavoidable. The Internet bandwidth and usage also impact the end-to-end transmission.

To minimize the delay, the protocol was designed to allow the sending of one packet at a time. This approach reduced the overall packet size which makes it possible to send the data with a very short delay (1 second) most of the time. Whereas the packet size is very small (32 bytes) due to the protocol design, the network bandwidth requirements became very small. Therefore, the system required only a very few resources to transmit the data to the server which thus makes it possible to transmit the data in near-real time despite unpredictable changes in the networks load.

The system can still send data with a very short delay utilizing the available resource even when load on the network increase from other applications. Table 4.1 shows the average encountered delay during 5 days for two separate test locations. We ran two location-based tests for 12 hours at different times. One test was performed in the city of Toronto; the second test was done in the city of Vaughan. In both tests, the PDA was connected to the server in the lab (Ryerson University) which is connected to the Internet using the university's network. Time stamps were inserted into the PDA log file while sending the data, and another time stamp was inserted into the server log file while receiving the data. The time difference between the two time stamps is used to calculate the delay.

Day	Location	Average Delay	Location	Average Delay
17/3/2008	Toronto	5 second	Vaughan	2 second
18/3/2008	Toronto	1 second	Vaughan	1 second
19/3/2008	Toronto	1 second	Vaughan	1 second
20/3/2008	Toronto	3 second	Vaughan	2 second
21/3/2008	Toronto	1 second	Vaughan	1 second

Table 4.1 Average Delay Using Layer Pro Protocol per day /Location

4.1.3 System Data Integrity

4.1.3.1 Packet Sequence Number

Packet sequence numbers are used to validate the data transmission. The system generates 100 packets per second. Each packet has a unique sequence number. The PDA tags the packet with a unique number and on the server it is used to check the integrity of the data transmission.

More than 100 data integrity tests were performed. The system was used to generate continuous packets (90,000 packets) for duration of 15 minutes. In each test the packet numbers were checked to ensure that the server received the complete number and the correct sequence. The result shows that the system successfully sent all acquired packets with the correct sequence. It is worth mentioning that the number of packets is proportional to the test duration. The formula (Equation 4.1) provides a way to calculate the number of packets. The sequence number is just a sequence number starting from 0.

$$P_n = P_f \cdot T$$

P_n = Packet Numbers

P_f = Packet Frequency (The packet frequency for the used sensor is 100 HZ)

T = time in milliseconds

Equation 4.1: Total Packet Number Calculator

On the server side we detected packets without a sequence number in some cases. We tracked down the cause to the sensor. Occasionally at random, the sensor sends packets without a sequence number. The total number of packets sent from the sensor without a sequence number did not exceed 100 packets out of 90,000. In most of the cases, packets without sequence numbers are random in time.. In the case when the packets without sequence numbers are continuous, the total number did not exceed 10 packets. The packets that do not have a sequence number can be ignored and this does not impact the overall result. Simple manipulation can be used to eliminate these packets or fix them and add them to the overall acquired packets.

4.1.4 System Availability

The system was designed to provide full mobility. The system can function with or without GSM connectivity. Where there was no GSM connectivity or the GSM connection was lost during the measurement process, the system was able to save all acquired packets in the PDA's local file system as expected.

4.1.5 LayerPro Protocol

The LayerPro protocol was designed to support near-real time binary data streams over heterogenic networks. We consider our protocol one of the most important contributors to achieving minimum data transmission delay. To compare LayerPro performance versus HTTP protocol performance, two version of the system were implemented. The first version implemented LayerPro protocol and the second version implemented HTTP protocol. Performance tests were performed using time stamps. The first time stamp was added on the PDA code just before the code starts sending the data. The second time stamp was added in the server code just where the code starts receiving the data. The results demonstrate that LayerPro protocol performs better than the HTTP protocol for near-real time binary data streaming. Table 4.2 shows a comparison between the LayerPro protocol and the HTTP protocol.

Location	Layer Pro Average Delay	HTTP Average Delay
Vaughan	1 second	15 second
Vaughan	1 second	10 second
Vaughan	1 second	15 second

Vaughan	1 second	20 second
Vaughan	1 second	10 second

Table 4.2 LayerPro Protocol versus HTTP Protocol

4.2 Biomedical Application results

4.2.1 Summary

Our system was designed to support a wide range of measurement activities. We wished to ensure that a variety of data be available for testing. In order to achieve this goal we performed tests on both humans and trained dogs with tests being conducted both inside and outside a lab environment. HbO₂ and Hb changes in brain and tissue were collected for both species in different circumstances. In total, three major types of biomedical experiments were conducted using our system.

The first experiment was a breath holding experiment. The test was used as a validation experiment in order to ensure that our system worked correctly and could collect biological data. The second experiment was related to smoking and it was conducted entirely outside the lab. This experiment was performed to understand the effect of smoking on the brain in a real environment away from the distractions and unrealities of a rigid laboratory environment—an environment where smoking actually takes place. The third experiment was conducted to attempt to monitor a trained canine's brain activities. The experiment was conducted in order to determine if it was possible to monitor the brain activity of animals. We found the second and third test to be particularly compelling. Smoking is an addictive behavior that occurs in the real world. In order to more accurately understand the factors that cause this behavior we believe that

any measurement must occur in the true circumstances of the activity. The third involving trained canines was motivated by both the need for data outside the human realm but also because we believe it may be possible to determine elements of mental activity within working animals—specifically canines—that directly relate to the activity that the animal is about to engage in. This is significant because it implies a certain level of predictability. Whether this is actually feasible is beyond the scope of this thesis however Helton Et. Al. have run a similar test in a lab environment without the benefit of our system [97] However, if testing is ever to be done in a real world setting, there must be a mechanism for allowing it.

4.2.2 Breath Holding Experiment Results

To validate that the system was functioning as expected, a breath holding experiment was performed on humans. The result was compared with a lab method [98]. Test subjects were asked to rest for 20 seconds, then to hold their breath for 20 seconds, and thereafter exhale and breathe normally for 20 seconds. The trial for each test subject lasted for 120 seconds. The rest duration between trials for each test subject was approximately 2 days.

We performed 15 breath holding trials. We asked three different test subjects (two males and one female) to hold their breath. The first test subject was a 23-year-old healthy female, non-smoker; the second test subject was a 46-year-old healthy male, non-smoker; and the third test subject was a 36-year-old healthy male smoker. During the lab trials, the test subjects were asked to wear the sensor on their forehead near the hair line and lay down on their backs on the test bed; they were asked not to move and not to speak. Instructions to inhale and exhale were communicated to them by the person

running the trials. In the outside trials, the test subjects were asked to wear the sensor on their foreheads and sit on a chair in the open and they were asked not to move or speak while performing the breath holding trial.

Figure 4.1 shows the test subject wearing the sensor during the breath holding experiment.

Figure 4.1: Test Subject Wearing the Sensor during the Breath Holding Experiments

After analyzing the collected data using our system, we can see that each breath holding trial had a measured impact on the HbO_2 and Hb concentration. The result was compared to a result obtained from a similar experiment using fMRI [98]. This experiment proved that the system can provide results similar to the ones previously obtained by other test methods, such as [98]. Clearly the system worked as expected.

Figures 4.2 and 4.3 show two examples of data obtained during the breath holding trials. The graphs show that HbO_2 increases during the breath holding. The arrow indicates when the increase happens due to breath holding. The brain compensates for the lack of oxygen by increasing the blood flow [98]. Then the HbO_2 level goes down after breathing was resumed.

Figure 4.2: Breath Holding Trial # 1

Figure 4.3: Breath Holding Trial # 2

4.2.3 Smoking Experiment Results

There is an agreement among scientists that cigarette smoking causes lung cancer, heart diseases, and other serious illnesses [99, 100]. Almost five million Canadians smoke 15 times or more per day [101, 102]. The chemical substances, including nicotine,

found in cigarettes [103, 104, 105, 106] entering the human body during smoking can cause several physiological changes. Few studies have applied fMRI to detect the oxygen level changes in the human brain under the effect of direct nicotine administration. The results have proven that nicotine can impact the level of oxygen in the hemoglobin in the brain [100, 107]. It is important to emphasize, however, that all these studies have tested the impact of the nicotine on the oxygen level in the brain using direct nicotine administration rather than actual smoking.

To understand the real effect cigarettes (nicotine and other chemicals) have on the brain, as opposed to direct administration of nicotine, smoke testing must be performed in a natural way rather than in a controlled environment. One contribution of the developed system in this research is to address this need. In fact, there are pragmatic health and safety reasons why our method is superior to in-lab testing. Because test subjects can be tested independently of the environment, no collateral damage from smoking need be accidentally inflicted on auxiliary participants in the test. Thus our method is safer, does not require special lab modifications and is as effective as other methods.

In total, six smoking trials were conducted. The experiment's purpose was to examine the relationship between smoking and HbO₂ and Hb changes in the brain. Five healthy human males and one healthy human female participated in the experiment. The test subjects' ages ranged from 30 to 40 years old. All the test subjects were active smokers for a period of more than 2 years. During the trials, the test subjects were asked to wear the sensor on their foreheads and sit on a chair in the open where they were asked not to move more than they had to in order to smoke and not to speak. The sensor was fixed with a bandage on the test subject's head to improve the sensor's stability on the

head and minimize the effect of the test subject's movement during the smoking process. Instructions as to when to smoke were communicated to the subjects by the person running the trials. Each trial lasted for 15 minutes, which included a five-minute baseline, five minutes of smoking, and a five-minute recovery after smoking. Figure 4.4 shows the test subject wearing the sensor during the smoking experiment.

Figure 4.4: Test Subject Wearing the Sensor during the Smoking Experiments

Baseline data was recorded for 5 minutes before the test subject started smoking. The test subject was asked to smoke for 5 minutes. The test subject inhaled every 20 seconds for the duration of the test. After the 5-minute smoking period, the test subject was asked to keep wearing the sensor for another 5 minutes. The data collection continued during the 5-minute waiting period after the smoking was complete. The recovery period allowed us to capture any delayed after-effect changes that occurred due to smoking.

When we analyzed the data, we observed HbO_2 and Hb changes during the baseline, the smoking, and the recovery periods. Figures 4.5 to 4.6 illustrate the results from the smoking experiment. The graphs show that during the baseline duration, changes in HbO_2 and HHb reflected normal physiological states. Sharper changes in

HbO₂ and Hb were appeared during smoking. These changes were similar to changes that occur during functional brain activities. Usually, such changes occurred due to the increase in the blood flow [108].

Figure 4.5: Smoking

Figure 4.6: Sample Smoking Trial

4.2.4 Canine Experiment Result

The purpose of canine experiments was to determine the possibility of measuring changes in the blood oxygenation in the canine brain in vivo. The experiment was conducted employing trained Urban Search and Rescue (USAR) canine teams in two phases. The first phase was to determine an appropriate place to position the sensor (intended for humans) on the canine head. We received the assistance of USAR canine teams from 3 Canadian national USAR Task Forces. We determined a feasible location to collect data through trial and error resulting in a general understanding of where the sensor should be held. No attempt was made to hold the sensor in place except through the assistance of the canine handler who held either the sensor in place or their dogs during the tests.

We found that we were able to measure a signal from a canine head however; the signal was rather noisy if the dog was very active as the handlers had difficulty in keeping the sensors in place.

Figure 4.7: This is a picture of Kathy from CAN-TF3 (Toronto Heavy Urban Search and Rescue) and her dog while performing the experiment in the first phase

In the second phase we used what we learned from the first phase to run a controlled test similar in nature to those employed by Pavlov [109]. The sensor was placed and held on the dog's head and the handler was asked to keep the dog as motionless as possible for 120 seconds. While no dog remained motionless throughout this entire period, the time allowed the dogs to calm themselves and, by the time the period had elapsed, the dogs were still. After the 120 second-period, an assistant was asked to show the dog its favorite chew toy. The data collection process continued for another 60 seconds after the dog saw the toy. The sensor was then removed and the dog was given its toy to play with.

Figure 4.8: Constable Kevin Barnum hold sensor on USAR canine Dare's head while Constable Denis Harkness presents the toy while Salah collects the data in the background

Figure 4.9 shows the collected data in the second phase of our experiment—indicating that it was possible to obtain data from dogs with our sensor. However, the system did not detect any HB and HbO₂ changes from some dogs. Some factors that might have caused the failure to detect Hb and HbO₂ changes in all the canines' brains were the canines' movements, their head anatomy, and/or the hair on their head.

Figure 4.9 shows the obtained results from the canine experiment while showing the canine its favorite toy. The arrows indicate the time when the canines saw the toy. The time when the canine was shows the toy starts at second 560. Hb and HbO₂ start increasing at that moment. The changes indicated between the arrows may indicate brain activities response to the toy.

Figure 4.9: Canine Trial # 1

4.3 Data Transmission Discussion

The results showed that the system can detect HbO₂ and Hb changes in the brains of humans and animals in their every day environments. This is a major indicator that the contributed protocol and algorithm have also achieved their goals. The protocol and the algorithm make it possible for the system to transmit accurate data in near-real time over heterogenic networks.

The collected data during smoking trials was different in its quality. Some trials were able to provide good results while other trials showed no effect for smoking. We noticed that test subjects' movements and positions could impact the acquired data. In Figure 4.6, we conducted the experiment on the test subject while he was standing. We noticed a big difference in HbO₂ and Hb changes resulting from the test subject while sitting. The changes are different between subjects; therefore, further research will be required to understand the differences.

Finally, the collected data from the dogs showed some changes. It was a challenge to keep the dogs motionless for a period of time long enough to complete the data acquisition. However, even with the sharp movements of the canines, the system was able to detect HbO₂ and Hb changes in their brains.

The result gives us the confidence that the system is functional as expected and ready to be used to monitor tissues hemodynamics changes during other behaviors. In short, the system achieved the specified goals and, as such, supports our claims.

5 CHAPTER 5- SUMMARY, CONCLUSION, AND FUTURE WORK

5.1 Conclusion and Contributions of this Work

This project developed a fully mobile system to monitor human brain functions in real-life, not in the lab.

The project contributes to the body of knowledge in the biomedical engineering field an inexpensive solution to monitor human tissues in near-real time.

The research also contributed to the body of knowledge in computer science a framework for integrating wired networks, wireless networks, and sensor networks to detect tissues oxygenation in near-real time. Further, a new architecture has been created to address the challenges and the issues that arise from integrating heterogenic networks. The algorithm and the protocol ensure the acquired data's integrity and accuracy during near-real time online transmission. Moreover, the algorithm handles the case when the transmission is interrupted through a store-forward mechanism. The LayerPro protocol performed better than HTTP protocol to stream binary data form the near-infrared sensor. We have demonstrated that without employing our LayerPro protocol it would not be possible to stream data in near-real time over the heterogenic networks using the HTTP protocol and achieve timely results.

The results have shown that it is possible to utilize mobile networks and near-infrared technologies to create a system to monitor HbO₂ and Hb in the human brain and tissues in near-real time. Further work will be required to ensure that the system provides more reliable results for animals.

The major difficulty we faced was a biomedical engineering challenge arising from the canine experiments. Canine brain anatomy is different from human brain anatomy. The sensor was designed to be used on the human brain and, therefore, it needs to be modified to fit canine brain anatomy (and by extension, any other animal physiology) to provide better readings.

5.2 Future Work

In order to address the challenges we faced during the research, further work will be required. There are several improvements that can be completed to make the system more universal.

Data transmission delay can reach up to 5 seconds (Table 4.1). Future work will be required to improve the system performance and to reduce the delay to less than 1 second. Moreover, it would be interesting to test the protocol and the algorithm with a bigger data packet (more than 32 bytes), and test if the system still can achieve its timely functionality.

Moreover it would be interesting to try different types of sensors that have higher frequency (more than 100 HZ) than the sensor we used and see if the system will perform the same. Also more work can be done to try LayerPro protocol to stream text based data

and compare it with HTTP protocol to see if it will perform the same way it did with the binary data.

Canines are useful animals in search and rescue and other domains where their sense of smell can be employed. Understanding what goes on inside their brains when they are far away from their handlers can provide important information to the handlers, allowing them to better understand the canine's signals and, in turn, what they found during the search operation. Modifying the sensor to better fit canines' anatomy would be another area where our initial work can be fruitfully employed. It may be possible to make improvements to diverse areas such as law enforcement (drug and bomb detection), public safety (fire accelerant detection) and search and rescue work (disaster victim cadaver search).

6 REFERENCES

- [1] Dhakan, P. and Menezes, R. (2005). The Role of Social Structures in Mobile Ad-hoc Networks. In Proceedings of the 43rd Annual Southeast Regional Conference, vol. 2 (Kennesaw, Georgia, March 18-20). ACM, New York, NY, USA, 59-64.
- [2] Sammon, M. J., Brotman Karmin, L. S., Peebles, E., and Seligmann, D. D. (2007). MACCS: An Industrial Study of Hands-free Wireless Communications for Mobile Healthcare Workers. In Proceedings of the 1st ACM SIGMOBILE International Workshop on Systems and Networking Support for Healthcare and Assisted Living Environments, HealthNet '07 (San Juan, Puerto Rico, June 11). ACM, New York, NY, USA, 55-60.
- [3] Chu, Y. and Ganz, A. (2007). WISTA: A Wireless Telemedicine System for Disaster Patient Care. *Mob. Netw. Appl.* 12, no. 2-3, (March), 201-14.
- [4] Ali, H. M., Naimi, A. M., Busson, A., and Vèque, V. (2007). An efficient link management algorithm for high mobility mesh networks. In Proceedings of the 5th ACM International Workshop on Mobility Management and Wireless Access, MobiWac '07 (Chania, Crete Island, Greece, October 22-22).. ACM, New York, NY, USA, 42-49.
- [5] Bluetooth Core Specifications Version 2.1. 2007. available at http://www.bluetooth.com/NR/rdonlyres/F8E8276A-3898-4EC6-B7DA-E5535258B056/6545/Core_V21__EDR.zip.
- [6] Bray, J. (2002). *Bluetooth: Connect without Cables*, 2nd ed. Prentice Hall, Upper Saddle River, NJ.
- [7] Ganguli, M. (2002). *Getting Started with Bluetooth*. Premier Press, Cincinnati, Ohio.
- [8] Muller, N.J. (2001). *Bluetooth Demystified*. McGraw-Hill, New York.
- [9] Barnes S.J. (2002). Under the Skin: Short-range Embedded Wireless Technology. *International Journal of Information Management*, 22, no. 3, (June), 165-79.

- [10] Keshav, S. (2005). Why Cell Phones will Dominate the Future Internet. SIGCOMM Comput. Commun. Rev., 35, no. 2 (April), 83-86.
- [11] Varshney, U. (2007). Pervasive Healthcare and Wireless Health Monitoring. Mob. Netw. Appl. 12, no. 2-3 (March), 113-27.
- [12] Hulkko, S., Mattelmäki, T., Virtanen, K., and Keinonen, T. (2004). Mobile probes. In Proceedings of the Third Nordic Conference on Human-Computer Interaction, NordiCHI '04, (Tampere, Finland, October 23 - 27). vol. 82. ACM, New York, NY, USA, 43-51.
- [13] Delord, X., Perret, S., and Duda, A. (1998). Efficient Mobile Access to the WWW over GSM. In Proceedings of the 8th ACM SIGOPS European Workshop on Support For Composing Distributed Applications, EW 8. (Sintra, Portugal, September). ACM, New York, NY, USA, 1-6.
- [14] Eberspächer, J., Vögel, H.J, and Bettstetter, C. (2001). GSM: Switching, Services and Protocols, 2nd ed. John Wiley & Sons, Toronto.
- [15] Chakravorty, R., Clark, A., and Pratt, I. (2003). GPRSWeb: Optimizing the Web for GPRS Links. In Proceedings of the 1st International Conference on Mobile Systems, Applications and Services, (San Francisco, California, May). ACM, New York, NY, USA, 317-30
- [16] Cabeza, R. and Nyberg, L. (2000). Imaging Cognition II: An Empirical Review of 275 PET and fMRI Studies. Journal of Cognitive Neuroscience, 12, no.1 (January), 1-47.
- [17] Davidson, R. J. and Sutton, S. K. (1998). Affective Neuroscience: The Emergence of A Discipline. Curr. Current Opinion in Neurobiology, 5, no.2 (April), 217-224.
- [18] Chance, B., Anday, E., Nioka, S., Zhou, S., Hong, L., Worden, K., Li, C., Murray, T., Ovetsky, Y., Pidikiti, D., and Thomas, R. (1998). A Novel Method for Fast Imaging of Brain Function, Non-invasively, with Light. Optics Express, 2, no. 10, (May), 411-23.
- [19] Chance, B., Zhuang, Z., UnAh, C., Alter, C., and Lipton, L. (1993). Cognition-activated Low-frequency Modulation of Light Absorption in Human Brain. Proceedings of the National Academy of Sciences, 90, no. 8 (April), 3770-74.

- [20] Jobsis, F.F. (1977). Noninvasive Infrared Monitoring of Cerebral and Myocardial Sufficiency and Circulatory Parameters. *Science*, 198, no.4323, (December),1264-67.
- [21] Boas, D.A., Franceschini, M.A., Dunn, A.K. and Strangman, G. (2002). Non-Invasive Imaging of Cerebral Activation with Diffuse Optical Tomography, in *Vivo Optical Imaging of Brain Function*, R. Frostig, ed. CRC Press, Boca Raton, 193-221.
- [22] Abhyankar, S., Toshiwal, R., Cordeiro, C., and Agrawal, D. (2003). On the application of traffic engineering over bluetooth ad hoc networks. In *Proceedings of the 6th ACM international Workshop on Modeling Analysis and Simulation of Wireless and Mobile Systems* (San Diego, CA, USA, September 19, 2003). MSWIM '03. ACM, New York, NY, 116-23.
- [23] Kumar, V. (2003). Sensor: the atomic computing particle. *SIGMOD Rec.* 32, 4 (Dec. 2003), 16-21.
- [24] Lorincz, K., Kuris, B., Ayer, S. M., Patel, S., Bonato, P., and Welsh, M. (2007). Wearable wireless sensor network to assess clinical status in patients with neurological disorders. In *Proceedings of the 6th international Conference on information Processing in Sensor Networks* (Cambridge, Massachusetts, USA, April 25 - 27, 2007). IPSN '07. ACM, New York, NY, 563-64.
- [25] Krishnamurthy, L., Adler, R., Buonadonna, P., Chhabra, J., Flanigan, M., Kushalnagar, N., Nachman, L., and Yarvis, M. (2005). Design and deployment of industrial sensor networks: experiences from a semiconductor plant and the north sea. In *Proceedings of the 3rd international Conference on Embedded Networked Sensor Systems* (San Diego, California, USA, November 02 - 04, 2005). SenSys '05. ACM, New York, NY, 64-75.
- [26] Schluessler, T., Goglin, S., and Johnson, E. (2007). Is a bot at the controls?: Detecting input data attacks. In *Proceedings of the 6th ACM SIGCOMM Workshop on Network and System Support For Games* (Melbourne, Australia, September 19 - 20, 2007). NetGames '07. ACM, New York, NY, 1-6.
- [27] Soo-Hwan Choi, Byung-Kug Kim, Jinwoo Park, Chul-Hee Kang, and Doo-Seop Eom (2004). An implementation of wireless sensor network. *Consumer Electronics, IEEE Transactions on* Volume 50, Issue 1, Feb 2004, 236-44.

- [28] Tolle, G., Polastre, J., Szewczyk, R., Culler, D., Turner, N., Tu, K., Burgess, S., Dawson, T., Buonadonna, P., Gay, D., and Hong, W. (2005). A macroscope in the redwoods. In Proceedings of the 3rd international Conference on Embedded Networked Sensor Systems (San Diego, California, USA, November 02 - 04, 2005). SenSys '05. ACM, New York, NY, 51-63.
- [29] Kansal, A., Goraczko, M., and Zhao, F. (2007). Building a sensor network of mobile phones. In Proceedings of the 6th international Conference on information Processing in Sensor Networks (Cambridge, Massachusetts, USA, April 25 - 27, 2007). IPSN '07. ACM, New York, NY, 547-48.
- [30] Kargl, F., Ribhegge, S., Schlott, S., and Weber, M. (2003). Bluetooth-based Ad-Hoc Networks for Voice Transmission. In Proceedings of the 36th Annual Hawaii international Conference on System Sciences (Hicss'03) - Track 9 - Volume 9 (January 06 - 09, 2003). HICSS. IEEE Computer Society, Washington, DC, 314.1.
- [31] Chu, Y. and Ganz, A. (2007). WISTA: a wireless telemedicine system for disaster patient care. Mob. Netw. Appl. 12, 2-3 (Mar. 2007), 201-14.
- [32] Mainwaring, A., Culler, D., Polastre, J., Szewczyk, R., and Anderson, J. (2002). Wireless sensor networks for habitat monitoring, Proceedings of the 1st ACM international workshop on Wireless sensor networks and applications, September 28-28, 2002, Atlanta, Georgia, USA.
- [33] Brettlecker, G. and Schuldt, H. (2007). The OSIRIS-SE (stream-enabled) infrastructure for reliable data stream management on mobile devices. In Proceedings of the 2007 ACM SIGMOD international Conference on Management of Data (Beijing, China, June 11 - 14, 2007). SIGMOD '07. ACM, New York, NY, 1097-99.
- [34] Lo BPL, Yang GZ (2005). Key technical challenges and current implementations of body sensor networks. In Proceedings of the 2nd international workshop on wearable and implantable body sensor networks (BSN 2005), London, 1.
- [35] Yu-Chee Tseng, Senior Member, Ting-Yu Lin, Yen-Ku Liu, and Bing-Rong Lin, (2005). Event-Driven Messaging Services Over Integrated Cellular and Wireless Sensor Networks: Prototyping Experiences of a Visitor System. IEEE JSAC, Vol. 23, No. 6, 1133-45, June.

- [36] Kling, R., Adler, R., Huang, J., Hummel, V., and Nachman, L. (2004). Intel Mote: using Bluetooth in sensor networks. In Proceedings of the 2nd international Conference on Embedded Networked Sensor Systems (Baltimore, MD, USA, November 03 - 05, 2004). SenSys '04. ACM, New York, NY, 318-18.
- [37] Dagtas, S., Natchetoi, Y., and Wu, H. (2007). An integrated wireless sensing and mobile processing architecture for assisted living and healthcare applications. In Proceedings of the 1st ACM SIGMOBILE international Workshop on Systems and Networking Support For Healthcare and Assisted Living Environments (San Juan, Puerto Rico, June 11, 2007). HealthNet '07. ACM, New York, NY, 70-72.
- [38] Tsai, C. C., Lee, G., Raab, F., Norman, G. J., Sohn, T., Griswold, W. G., and Patrick, K. (2007). Usability and feasibility of PmEB: a mobile phone application for monitoring real time caloric balance. Mob. Netw. Appl. 12, 2-3 (Mar. 2007), 173-84.
- [39] Chu, Y. and Ganz, A. (2007). WISTA: a wireless telemedicine system for disaster patient care. Mob. Netw. Appl. 12, 2-3 (Mar. 2007), 201-14.
- [40] Varshney, U. (2007). Pervasive healthcare and wireless health monitoring. Mob. Netw. Appl. 12, 2-3 (Mar. 2007), 113-27.
- [41] Edlinger, G., Guger, C. (2006). Laboratory PC and Mobile Pocket PC Brain-Computer Interface Architectures. Engineering in Medicine and Biology Society, 2005. IEEE-EMBS 2005. 27th Annual International Conference of the 17-18 Jan. 2006, 534750.
- [42] Cheok, A., Goh, K. H., Liu, W., Farbiz, F., Fong, S. W., Teo, S. L., Li, Y., and Yang, X. (2004). Human Pacman: a mobile, wide-area entertainment system based on physical, social, and ubiquitous computing. Personal Ubiquitous Comput. 8, 2 (May. 2004), 71-81.
- [43] Bychkovsky, V., Chen, K., Goraczko, M., Hu, H., Hull, B., Miu, A., Shih, E., Zhang, Y., Balakrishnan, H., and Madden, S. (2006). The CarTel mobile sensor computing system. In Proceedings of the 4th international Conference on Embedded Networked Sensor Systems (Boulder, Colorado, USA, October 31 - November 03, 2006). SenSys '06. ACM, New York, NY, 383-84.

[44] Sawant, H., Jindong Tan, Qingyan Yang, and Qizhi Wang. (2004). Using Bluetooth and sensor networks for intelligent transportation systems. *Intelligent Transportation Systems*, 2004. In *Proceedings of the 7th International IEEE Conference on* 3-6 Oct. 2004, 767-72.

[45] Eagle, N. and (Sandy) Pentland, A. (2006). Reality mining: sensing complex social systems. *Personal Ubiquitous Comput.* 10, 4 (Mar. 2006), 255-68.

[46] Croft, C., Efron, A., and Purver, E. (2007). Future perfect. In *ACM SIGGRAPH 2007 Art Gallery* (San Diego, California, August 05 - 09, 2007). *SIGGRAPH '07*. ACM, New York, NY, 195.

[47] Jason Hill, Robert Szewczyk, Alec Woo, Seth Hollar, David Culler, Kristofer Pister, (2000). System architecture directions for networked sensors. *Proceedings of the ninth international conference on Architectural support for programming languages and operating systems*, 93-104, November, Cambridge, Massachusetts, United States.

[48] Treese, W. (2004). The coming of sensor networks. *netWorker* 8, 2 (Jun. 2004), 15-17.

[49] Hill, J., Horton, M., Kling, R., and Krishnamurthy, L. (2004). The platforms enabling wireless sensor networks. *Commun. ACM* 47, 6 (Jun. 2004), 41-46.

[50] Garner, P.D. (2003). Mobile Bluetooth networking: technical considerations and applications. *3G Mobile Communication Technologies, 2003. 4th International Conference on* (Conf. Publ. No. 494) 25-27 June 2003, 274-76.

[51] Sairam, K.V.S.S.S.S., Gunasekaran, N., and Redd, S.R. (2002). Bluetooth in wireless communication. *Communications Magazine, IEEE*, Volume 40, Issue 6, June 2002, 90-96.

[52] Shih-Yen Chiu, Hsung-Pin Chang, and Ruei-Chuan Chang (2002). Providing mobile LAN access capability for Bluetooth devices. *Parallel and Distributed Systems*, 2002. In *Proceedings of the Ninth International Conference on* 17-20 Dec. 2002, 631-36.

- [53] Junnala, S. and Niittylahti, J. (2003). Wireless technologies for data acquisition systems. In Proceedings of the 1st international Symposium on information and Communication Technologies (Dublin, Ireland, September 24 - 26, 2003). ACM International Conference Proceeding Series, vol. 49. Trinity College Dublin, 132-37.
- [54] Bartelt, C., Fischer, T., Niebuhr, D., Rausch, A., Seidl, F., and Trapp, M. (2005). Dynamic integration of heterogeneous mobile devices. SIGSOFT Softw. Eng. Notes 30, 4 (Jul. 2005), 1-7.
- [55] Gil-Castiñeira, F. J., González-Castaño, F. J., and Pousada-Carballo, J. M. (2005). Personal microdevices for wide-area location via mobile gateways. In Proceedings of the 7th international Conference on Human Computer interaction with Mobile Devices & Services (Salzburg, Austria, September 19 - 22, 2005). MobileHCI '05, vol. 111. ACM, New York, NY, 381-82.
- [56] DeRenzi, B., Anokwa, Y., Parikh, T., and Borriello, G. (2007). Reliable data collection in highly disconnected environments using mobile phones. In Proceedings of the 2007 Workshop on Networked Systems For Developing Regions (Kyoto, Japan, August 27 - 27, 2007). NSDR '07. ACM, New York, NY, 1-5.
- [57] Ni, Y., Kremer, U., Stere, A., and Iftode, L. (2005). Programming ad-hoc networks of mobile and resource-constrained devices. In Proceedings of the 2005 ACM SIGPLAN Conference on Programming Language Design and Implementation (Chicago, IL, USA, June 12 - 15, 2005). PLDI '05. ACM, New York, NY, 249-60.
- [58] S. Zeadally and A. Kumar (2004). Protocol support for audio streaming between bluetooth devices. IEEE Radio and Wireless Conference, 303-306.
- [59] Shen, G., Li, Y., and Zhang, Y. (2007). MobiUS: enable together-viewing video experience across two mobile devices. In Proceedings of the 5th international Conference on Mobile Systems, Applications and Services (San Juan, Puerto Rico, June 11 - 13, 2007). MobiSys '07. ACM, New York, NY, 30-42.
- [60] Auletta, V., Blundo, C., De Cristofaro, E., and Raimato, G. (2006). Performance evaluation of web services invocation over Bluetooth. In Proceedings of the ACM international Workshop on Performance Monitoring, Measurement, and Evaluation of Heterogeneous Wireless and Wired Networks (Terromolinos, Spain, October 02 - 02, 2006). PM2HW2N '06. ACM, New York, NY, 1-8.

- [61] Mundt, T. (2004). How much is a byte?: a survey of costs for mobile data transmission. In Proceedings of the Winter international Symposium on information and Communication Technologies (Cancun, Mexico, January 05 - 08, 2004). ACM International Conference Proceeding Series, vol. 58. Trinity College Dublin, 1-6.
- [62] Rouhana, N. and Horlait, E. (2002). BWIG: Bluetooth Web Internet Gateway. In Proceedings of the Seventh international Symposium on Computers and Communications (Iscc'02) (July 01 - 04, 2002). ISCC. IEEE Computer Society, Washington, DC, 679.
- [63] Dutta, A., Zhang, T., Madhani, S., Taniuchi, K., Fujimoto, K., Katsube, Y., Ohba, Y., and Schulzrinne, H. (2004). Secure universal mobility for wireless internet. In Proceedings of the 2nd ACM international Workshop on Wireless Mobile Applications and Services on WLAN Hotspots (Philadelphia, PA, USA, October 1, 2004). WMASH '04. ACM, New York, NY, 71-80.
- [64] Sun Microsystems Java <<http://www.java.sun.com>>.
- [65] Eclipse 3.3 <<http://www.eclipse.org/>>.
- [66] Netbeans 6.0 <<http://www.netbeans.org/>>.
- [67] Comer, D. (1997). In Stevens D. L. (Ed.), Internetworking with TCP (Windows sockets version. ed.). Prentice Hall, Upper Saddle River, N.J..
- [68] Stevens, W. R. (1994-). Addison-Wesley Pub. Co., TCP. Reading, Mass.
- [69] Comer, D. (2007). The internet book: Everything you need to know about computer networking and how the internet works, 4th ed. Pearson Prentice Hall, Upper Saddle River, NJ..
- [70] Bray, J. (2002). Bluetooth: connect without cables, 2nd ed., Prentice Hall, Upper Saddle River, NJ.
- [71] Ganguli, M. (2002), Getting started with Bluetooth. Premier Press, Cincinnati, Ohio.

[72] Huang, A. S. (2007). In Rudolph L. (Ed.), Bluetooth essentials for programmers. New York, NY: Cambridge University Press. Muller, N.J. (2001).

[73] Bluetooth Core Specifications Version 2.1. 2007. Available at
<http://www.bluetooth.com/NR/rdonlyres/F8E8276A-3898-4EC6-B7DA-E5535258B056/6545/Core_V21__EDR.zip>.

[74] Delord, X., Perret, S., and Duda, A. (1998). Efficient Mobile Access to the WWW over GSM. In Proceedings of the 8th ACM SIGOPS European Workshop on Support For Composing Distributed Applications, EW 8. (Sintra, Portugal, September). ACM, New York, NY, USA, 1-6.

[75]. Eberspächer, J., Vögel, H.J, and Bettstetter, C. (2001). GSM: Switching, Services and Protocols, 2nd ed. John Wiley & Sons, Toronto.

[76] Chakravorty, R., Clark, A., and Pratt, I. (2003). GPRSWeb: Optimizing the Web for GPRS Links. In Proceedings of the 1st International Conference on Mobile Systems, Applications and Services, (San Francisco, California, May). ACM, New York, NY, USA, 317-30.

[77] Eberspaecher, J. (2001). GSM: switching, services and protocols, 2nd ed. Wiley, Toronto.

[78] Arquatis GmbH, Rieden, Switzerland. Developed in cooperation with the Biomedical Optics Research Laboratory at the Clinic of Neonatology
<<http://www.arquatis.com>>: product name nScan W1200.

[79] Muhlemann, T.L. (2006). New Wireless Probes for Near Infrared Spectroscopy. Master Thesis, Swiss Federal Institute of Technology, Zurich.

[80] Muhlemann, T.L., Haensse, D. and Wolf, M. (2006). A Wireless Near-Infrared Imaging Device. The 34th Annual Meeting of the International Society on Oxygen Transfer to Tissue. (Louisville, Kentucky, August 12-17), available at
<<http://louisville.edu/conference/isott06/WebProgram.pdf>>.

[81] Trajkovic, I. (2006). Examination of the Brain with Light: Integration of a Wireless Sensor into a Graphical User Interface based on Java. Semester Thesis, Swiss Federal Institute of Technology, Zurich.

[82] Strangman, G., Boas, D.A., and Sutton, J.P. (2002). Non-invasive neuroimaging using near-infrared light. *Biological psychiatry*, vol. 52, no. 7, pp. 679-93.

[83] Izzetoglu, K., Yurtsever, G., Bozkurt, A., and Bunce, S. (2003). Functional brain monitoring via NIR based optical spectroscopy. In *Proceedings of the 29th IEEE Annual Conference-Bioengineering Conference* (Newark, N.J., USA, March 22-23, 2003) 335-36.

[84] Villringer, A. and Chance, B. (1997). Non-invasive optical spectroscopy and imaging of human brain function. *Trends in Neuroscience*, 20(10), 435-42.

[85] Gratton, E., Fantini, S., Franceschini, M.A., Gratton, G., and Fabiani, M. (1997). Measurements of scattering and absorption changes in muscle and brain. *Philosophical Transactions: Biological Sciences*, 352(1354), 727-35.

[86] Bozkurt, A., Rosen, A., Rosen, H., and Onaral, B. (2005). A portable near infrared spectroscopy system for bedside monitoring of newborn brain. *Biomedical engineering online*, vol. 4, no. 1, 29.

[87] Hong, L., Worden, K., Li, C., Murray, T., Ovetsky, Y., Pidikiti, D., and Thomas, R. (1998). A novel method for fast imaging of brain function, non-invasively, with light. *Optics Express*, vol. 2, 411.

[88] Germon, T.J., Evans, P.D., Manara, A.R., Barnett, N.J., Wall, P., and Nelson, R.J. (1998) Sensitivity of near infrared spectroscopy to cerebral and extra-cerebral oxygenation changes is determined by emitter-detector separation. *Journal of clinical monitoring and computing*, vol. 14, no. 5, 353-60.

[89] Benni, P.B., Bo, C., Amory, D., and Li, J.K. (1995). A novel near-infrared spectroscopy (NIRS) system for measuring regional oxygen saturation. *Bioengineering Conference, 1995, Proceedings of the 1995 IEEE 21st Annual Northeast*, 105-107.

- [90] David A. Boas, Tom Gaudette, Gary Strangman, Xuefeng Cheng, John J. A. Marota, and Joseph B. Mandeville "The Accuracy of Near Infrared Spectroscopy and Imaging during Focal Changes in Cerebral Hemodynamics" NMR Center, Massachusetts General Hospital, Harvard Medical School, Charlestown, Massachusetts 02129; and Electro-Optic Technology Center, Tufts University, Medford, Massachusetts 02155
Received January 21, 2000; published online November 7, 2000.
- [91] Tuchin, V. V. (2000). *Tissue optics: Light scattering methods and instruments for medical diagnosis*. Bellingham, Wash.: SPIE Optical Engineering Press.
- [92] Wang, L. V. (2007). In Wu H. (Ed.), *Biomedical optics: Principles and imaging*. Hoboken, N.J.: Wiley-Interscience.
- [93] Splinter, R. (2007). In Hooper B. A. (Ed.), *An introduction to biomedical optics*. New York: Taylor & Francis.
- [94] Porth, C. (2005). *Pathophysiology: concepts of altered health states*, 7th ed. Lippincott Williams & Wilkins, Philadelphia.
- [95] <<http://www.forum.nokia.com/devices/E62>>.
- [96] Real-time vision for human-computer interaction (2005). In KisaUcanin B., Pavlovi V., Huang T. S. and MyiLibrary. (Eds.), p.16 . New York: Springer Science.
- [97] W.S. Helton, T.D. Hollander, J.S. Warm, L.D. Tripp, K. Parsons, G. Matthews, W.N. Dember, R. Parasuraman, and P.A. Hancock (2007), "The abbreviated Vigilance Task and Cerebral Hemodynamics", *J. of Clinical and Experimental Neuropsychology*, 29(5), 545-552.
- [98] Zhang, X., Toronov, V., Webb, A. (2005). Methodology development for simultaneous diffuse optical tomography and magnetic resonance imaging in functional human brain mapping. In *Proceedings of SPIE*, Vol 5686, (April), 453-63.
- [99] Carmines, E.L. (2002). Evaluation of the potential effects of ingredients added to cigarettes. Part 1: Cigarette design, testing approach, and review of results. *Food and Chemical Toxicology* 40(1), 77-91.
- [100] Giessing, C., et al. (2006). The modulatory effects of nicotine on parietal cortex activity in a cued target detection task depend on cue reliability. *Neuroscience* 137(3), 853-64.

- [101] Flight, J. (2007). Canadian Addiction Survey (CAS): A national survey of Canadians' use of alcohol and other drugs: substance use by youths. Health Canada, Ottawa.
- [102] Health Canada (2007). Canadian Tobacco Use Monitoring Survey. Health Canada, Ottawa.
- [103] Hoffmann, D., Hoffmann, I., El-Bayoumy, K. (2001). The less harmful cigarette: A controversial issue. A tribute to Ernst L. Wynder. *Chemical Research in Toxicology* 14(7), 767- 90.
- [104] Baker, R.R., Massey, E.D., Smith, G. (2004). An overview of the effects of tobacco ingredients on smoke chemistry and toxicity. *Food and Chemical Toxicology* 42S, S53-S83.
- [105] Rodgman, A., Smith, C.J., Perfetti, T.A. (2000). The composition of cigarette smoke: a retrospective, with emphasis on polycyclic components. *Human and Experimental Toxicology* 19(10), 573-95.
- [106] Frederick, B., et al. (2007). An MR-compatible device for delivering smoked marijuana during functional imaging. *Pharmacology, Biochemistry, and Behavior* 87(1), 81-89.
- [107] Siafaka, A., et al. (2007). Acute effects of smoking on skeletal muscle microcirculation monitored by near-infrared spectroscopy. *Chest Journal* 131(5), 1479-85.
- [108] Toronov, V., et al. (2001). Investigation of human brain hemodynamics by simultaneous near-infrared spectroscopy and functional magnetic resonance imaging. *Medical Physics* 28(4), 521-27.
- [109] Pavlov, Ivan Petrovitch (1928). In Gantt W. H. (Ed.), *Lectures on conditioned reflexes*. New York: Liveright.
- [110] Basler, M., Bolliger, J., and Wacha, C. (2005). Examination of the Brain with Light: A Redesign of a Wireless Sensor Based on Bluetooth and Linux

7 APPENDICES

Appendix A: KERIOS Protocol Packet Format

Packet Format

Commands and data are passed using a well-defined data structure called a packet. The packet has three sections: a header, optional payload, and stop byte [110].

Header

The header section is four-bytes long. The first byte contains the device type. Since the protocol can be used to support communication between different devices, the device type is used to identify the sender device and the receiver device. The following are the allowed device types (Device Types): WL_ID_CODE (wireless device), MCP_ID_CODE (Multi Channel Photometer), and ST_ID_CODE (Stimbox device). In order to support other devices, a new device type needs to be defined [79]. The second byte in the header section contains the packet type OpCodes. Packet types specify the type of packet that is under way, such as the packet purpose. The third byte contains the overall packet total length, which includes header, optional payload, and the stop byte. The fourth byte is used to store a check-sum used to ensure data integrity. Fletcher's checksum algorithm is used to generate the checksum value for the overall packet [79].

There are 11 packet types defined in the protocol; however, more packet types can be added. The most significant packet type used is OPC_CTRL_COMMAND, which is issued from the PDA to the sensor to pass the command over to the sensor.

Payload

Payload, the second part of the packet, is optional and contains either data or a control command. The first three bytes of the payload have a unique sequence number

that identifies the sequence of the sent and received data packets and command packets. The sequence number is useful for assembling the data and the command at the receiving end. The value of the sequence number is between 1 to $(2^{24}-1)$. When the packet does not include NIRS data or operation acknowledgment, the sequence number will not exist.

All commands are issued from the master device to the slave device (the sensor). NIRS Data is the actual data that has been captured from the sensor. There is also another data type sent from the slave device to the master device called configuration data. The master device asks the slave device for its configurations and the slave device responds with its configurations data in the payload. Additionally, when a packet contains NIRS data, it also includes the number of channels that acquired the data. The packet assigns 1 byte to store the number of channels. Each channel needs 2 bytes to hold the data acquired per channel. When an NIRS_DATA operation code is issued, the data (e.g., sequence number, NIRS data) is encapsulated in two bytes and the most significant byte is transmitted first.

Stop Byte

Stop Byte is the last part of the packet and it identifies the end of the packet and always has the value 0xFF.

Transactions and Packet Flow

A KREIOS protocol transaction consists of a request and a response. They must always be paired. That is, for each request there is a response. The transaction request and response are in the form of a packet.

The transactions can be placed into two categories: start up activities and measurement transactions. The following is an example of a transaction used to acquire data from the sensor.

- 1- User powers on the slave device (the sensor) by pushing the power on button on the sensor.
- 2- Master sends the `ENABLE_LEDS` command to enable the slave device LEDs. If the command is executed successfully by the slave device, the slave responds with `OPC_ACK`.
- 3- Master issues `START_MEASURE` command. If the command is executed successfully by the slave device, the slave device will respond with `OPC_ACK` and start the measurement activities. Once the slave device completes the measurement activities, the sensor will send `OPC_NIRS_DATA` packet, which contains the measurement data.
- 4- When the measurements are completed by the slave device, the master device sends the `OPC_CTRL_COMMAND` packet containing a `DISABLE_LEDS` command to the slave device and waits for the response from the slave device.
- 5- Once the previous steps are completed successfully, the master device sends the `OPC_CTRL_COMMAND` packet containing a `POWER_OFF` command to switch off the device.

Control Activities Transactions

Below is a sequence of activities to enable start up activities. See Figure 2.3 for the same explanation in a diagram.

- 1- **ENABLE_LEDS:** This transaction enables the power supply for the light source. The master device will issue the command and send a packet of type **OPC_CTRL_COMMAND** and command **ENABLED_LEDS**. If the request is performed successfully in the slave device, the slave device will respond with the **OPC_ACK** packet. In contrast, if the request is not executed correctly in the slave device, the slave device will send the **OPC_NACK_ERROR** packet.
- 2- **DISABLE_LEDS:** This transaction disables the power supply source for the light source. The master device will issue the command and send a packet type **OPC_CTRL_COMMAND** which contains the command **DISABLE_LEDS**. If the request is performed successfully in the slave device, the slave device will respond with the **OPC_ACK** packet. If the request is not executed correctly in the slave device, the slave device will send the **OPC_NACK_ERROR** packet.
- 3- **STLED_ON:** This transaction enables the status led for debugging purposes. The master device will issue the command and send the packet type **OPC_CTRL_COMMAND** and the command **STLED_ON** in the sent packet. If the request is performed successfully in the slave device, the slave device will respond with the **OPC_ACK** packet. If the request is not executed correctly in the slave device, the slave device will send the **OPC_NACK_ERROR** packet.
- 4- **STLED_OFF:** This transaction disables the status led for debugging purposes. The master device will issue the command by sending a packet type **OPC_CTRL_COMMAND** which contains **STLED_OFF** command. If the request is performed successfully in the slave devices, the slave device will respond with

the OPC_ACK packet. If the request is not executed successfully in the slave device, the slave device will respond with the OPC_NACK_ERROR packet.

- 5- **DEVICE_RESET**: This transaction resets the slave device to its initial status. The master device issues the command and sends the packet to the slave device. The sent packet will contain the packet type OPC_CTRL_COMMAND and the command DEVICE_REST. If the request is performed successfully in the slave device, the slave device responds with the OPC_ACK packet. If the request is not executed successfully in the slave device, the slave device will respond with the OPC_NACK_ERROR packet.
- 6- **POWER_OFF**: This command switches off the slave device power supply. The master device issues the command and sends the packet to the slave device. The sent packet contains a packet type of the OPC_CTRK_COMMAND and a command POWER_OFF. If the request is performed successfully in the slave device, the slave device responds with OPC_ACK packet. If the request is not executed successfully in the slave device, the slave device responds with the OPC_NACK_ERROR packet.

Figure A.7.1: Control Activities Transactions

Measurement Activities Transactions

Below is an example of a measurement activities transaction. Figure 2.4 provides the same explanation in a diagram.

- 1- **START_MEASURE:** This transaction starts the measurement activities. The master device issues the command and sends the packet to the slave device. The sent packet contains a packet type `OPC_CTRL_COMMAND` and a command `START_MEASURE`. If the request is performed correctly in the slave device, the slave device will respond with the `OPC_ACK` packet. If the request is not executed successfully in the slave device, the slave device responds with the

OPC_NACK_ERROR packet. The measurement process starts in the slave device (the sensor). Once the measurement process is completed from all the available channels, the slave device sends the OPC_NIRS_DATA packet to the master device.

- 2- REQUEST_CURRENT: This transaction requests the slave device to send the currently available acquired data. The master device issues the command and sends the packet to the slave device. The sent packet contains a packet type OPC_CTRL_COMMAND and a command REQUEST_CURRENT. If the request is performed successfully in the slave device, the slave device responds with the OPC_ACK_PAYLOAD packet. This request returns a packet containing the NIRS data.
- 3- REQUEST_CONFIGURATION: This transaction requests the slave device configurations, such as number of channels, number of light source, number of light detectors, and channel configurations. The master device issues the command and sends the packet to the slave device. The sent packet contains a packet type OPC_ACK_COMMAND and a command REQUEST_CONFIGURATION. If the request is performed successfully in the slave device, the slave device responds with the OPC_ACK_PAYLOAD packet.
- 4- CURRENT_RESET: This transaction sets the current configurations as the default configurations. The master device issues the command and sends the packet to the slave device. The sent packet contains a packet type OPC_ACK_COMMAND and a command CURRENT_RESET. If the request is

performed successfully in the slave device, the slave device responds with the OPC_ACK packet.

- 5- REQUEST_MAX_CURRENT: This transaction requests the maximum measurements. The master device issues the command and sends the packet to the slave device. The sent packet contains a packet type OPC_ACK_COMMAND and a command REQUEST_MAX_CURRENT. If the request is performed successfully in the slave device, the slave device responds with the OPC_ACK_PAYLOAD packet.
- 6- STOP_MEASURE: This transaction requests the slave device stop the measurement activities. The master device issues the command and sends the packet to the slave device. The sent packet contains a packet type OPC_ACK_COMMAND and a command STOP_MEASURE. If the request is performed successfully in the slave device the slave device responds with the OPC_ACK packet.

Figure 7.2: Measurement Activities Transaction

Packets

	Packet									
	Header				Optional Payload					Stop Byte
Packet Fields Byte Location	0	1	2	3	4	...	...	...	...	n-1
<i>Packet fields Name & Sequences</i>	<i>Device Type</i>	<i>Packet Type</i>	Length	Checksum						Stop Byte
Packet Fields Value	Table 1	Table 2	n							0xFF

Table A.7.1: Packet Description

n: number of bytes

Device Types: Byte 0 in the packet		
Packet Type (ID)	Value	Description
WL_ID_CODE	0xA0	The device type is a wireless device
MCP_ID_CODE	0xB0	The device type is a MCP (Multi Channel photometer) device

ST_ID_CODE	0xC0	The device type is a Stimbox device
------------	------	-------------------------------------

Table A.7.2: Device Type

Packet type : Byte 1 in the packet			
<i>Operation Code (ID)</i>	<i>Value</i>	<i>Description</i>	<i>Direction</i>
OPC_EMPTY	0x00	Empty packet (no payload)	Device→PC
OPC_CTRL_COMMAND	0x02	Packet contains a control command for the device	PC→Devices
OPC_WRAPPED	0x01	Packet contains another complete packet	Device→PC
OPC_NIRS_DATA	0x04	Packet contains NIRS measurement data	Device→PC
OPC_STATUS_INFO	0x06	Packet contains information about the status of a device	Device→PC
OPC_ACK	0x0A	ACK to a control command	Device→PC
OPC_ACK_PAYLOAD	0x0B	ACK to a control command and additional payload	Device→PC
OPC_NACK	0x0C	NACK to a control command	Device→PC
OPC_NACK_ERROR	0x0D	NACK to a control command with error message	Device→PC
OPC_NOT_SPECIFIED	0x0E	Unspecified packet	Device→PC
OPC_ERROR	0x0F	Error message containing an error code	Device→PC

Table A.7.3: Packet Type

Control Commands			
<i>Control Command (ID)</i>	<i>Value</i>	<i>Description</i>	<i>Direction</i>
STLED_ON	0x01	Enable status LED (for debugging purpose)	PC→devices
STLED_OFF	0x02	Disable status LED (for debugging purpose)	
START_MEASURE	0x03	Start the measurement	PC→devices
STOP_MEASURE	0x04	Stop the measurement	PC→devices
CURRENT_RESET	0x05	Reset currents to default	PC→devices
REQUEST_CURRENT	0x06	Ask for transmission of actual current settings (The reply to this command is an ACK_PL (i.e., an ACK with payload). The payload contains the sequence number (3 bytes), the number of channels (1 byte) and the current values (numOfChannels * 2 bytes)).	PC→devices
REQUEST_MAX_CURRENT	0x07	Ask for transmission of maximum currents	PC→devices
REQUEST_CONFIGURATION	0x08	Ask for sensor/device configuration info	PC→devices
ENABLE_LEDSUPPLY / ENABLE_LEDS	0x09	Enable power supply for light sources	PC→devices
DISABLE_LEDSUPPLY / DISABLE_LEDS	0x0A	Disable power supply for light sources	PC→devices
DEVICE_RESET	0x0E	Reset the device	PC→devices
POWER_OFF	0x0F	Disable device power supply	PC→devices

Table A.7.4: Control Commands

Error Codes		
<i>Error Code (ID)</i>	<i>Value</i>	<i>Description</i>
NO_ERROR	0xC0	No error has occurred
GENERAL_ERROR	0xC1	An unspecified error has occurred
NOT_IMPLEMENTED	0xC2	The functionality requested was not implemented
MAX_CURRENT_EXC	0xD1	LED currents exceed the allowed maximum rating

Table A.7.5: Error Codes

Detailed Packet Types Description

OPC_EMPTY Packets

This packet has no useful information. The request packet does not require a response packet. The total packet length is 5 bytes.

OPC_EMPTY	Packet				
	Header				Stop Byte
Packet Fields Byte Location	0	1	2	3	4
Packet fields Name & Sequences	Device Type	Packet Type	Length	Checksum	Stop Byte
Packet Fields Value	0xA0	0x00	0x05	0x05	0xFF

Table A.7.6: OPC_EMPTY Packet

OPC_ERROR Packet

This packet represents a packet containing an error message. The error code will be one byte at position byte 4 of the packet. The possible values are listed in the error codes table. Only one value is allowed. The total packet length is 6 bytes

OPC_ERROR	Packet					
	Header				Optional Payload	Stop Byte
Packet Fields Byte Location	0	1	2	3	4	5
Packet fields Name & Sequences	Device Type	Packet Type	Length	Checksum	Error code	Stop Byte
Packet Fields Value	0xA0	0x0F	0x06	0x06	Table 5	0xFF

Table A.7.7: OPC_ERROR Packet

OPC_ACK Packet

This packet represents an acknowledgement to a command code packet. The master sends a command packet and the receiver replies back with this packet to acknowledge the reception of the packet if no error occurs.

OPC_ACK	Packet							
	Header				Optional Payload			Stop Byte
Packet Fields Byte Location	0	1	2	3	4	5	6	7
Packet fields Name & Sequences	Device Type	Packet Type	Length	Checksum	Sequence Number			Stop Byte
					seqN Hi	seqN Mid	seqN Low	
Packet Fields Value	0xA0	0x0A	0x08	0x08	0x00	0x00	0x00	0xFF

Table A.7.8: OPC_ACK Packet

OPC_NACK Packet

This packet contains a non-acknowledgement of the sent command packet. This packet contains the original sequence number for the sent command that was not successfully processed by the slave device. The sequence number does not represent an acknowledgement.

OPC_NACK	Packet							
	Header				Optional Payload			Stop Byte
Packet Fields Byte Location	0	1	2	3	4	5	6	7
Packet fields Name & Sequences	Device Type	Packet Type	Length	Checksum	Sequence Number			Stop Byte
					seqN Hi	seqN Mid	seqN Low	
Packet Fields Value	0xA0	0x0C	0x08	0x08	0x00	0x00	0x00	0xFF

Table A.7.9: OPC_NACK Packet

OPC_NACK_ERROR Packet

This packet contains an acknowledgement of an error; the packet contains the error code that describes the error. The sequence number indicates to which operation command this error belongs. The error code is 1 byte long.

OPC_NACK_ERROR	Packet								
	Header				Optional Payload			Stop Byte	
Packet Fields Byte Location	0	1	2	3	4	5	6	7	8
Packet fields Name & Sequences	Device Type	Packet Type	Length	Checksum	Sequence Number			Error Code	Stop Byte
					seqN Hi	seqN Mid	seqN Low		
Packet Fields Value	0xA0	0x0D	0x09	0x09	0x00	0x00	0x00	Table 3	0xFF

Table A.7.10: OPC_NACK_ERROR Packet

OPC_NOT_SPECIFIED Packet

This packet is a data transport packet. It is used to transport data without belonging to any command between the master device and the slave device or *vice versa*.

OPC_NOT_SPECIFIED	Packet							
	Header				Optional Payload			Stop Byte
Packet Fields Byte Location	0	1	2	3	4	..	..	8
Packet fields Name & Sequences	Device Type	Packet Type	Length	Checksum	Payload			Stop Byte
Packet Fields Value	0xA0	0x0D	0x05+ payload length	0x09	..	..	..	0xFF

Table A.7.11: OPC_NOT_SPECIFIED Packet

OPC_CTRL_COMMAND Packet

This packet contains the control command. This control command field is used to carry the issued commands to the slave. The control command is 1 byte long. This packet type is issued only from the master device. The sequence number identifies the current command packet to which this command belongs.

OPC_CTRL_COMMAND	Packet								
	Header				Optional Payload				Stop Byte
Packet Fields Byte Location	0	1	2	3	4	5	6	7	8
Packet fields Name & Sequences	Device Type	Packet Type	Length	Checksum	Sequence Number			Control Command	Stop Byte
					seqN Hi	seqN Mid	seqN Low		
Packet Fields Value	0xA0	0x0D	0x09	0x09	0x00	0x00	0x00	Table 3	0xFF

Table A.7.12: OPC_CTRL_COMMAND Packet

OPC_ACK_PAYLOAD Packet

This packet contains payload as an acknowledgement for a control command. The sequence number identifies to which control command this packet belongs. The total length of the packet will depend on the length of the payload. This packet confirms a proper reception of the command by the slave device and contains the desired data in the payload.

OPC_ACK_PAYLOAD	Packet									
	Header				Optional Payload					Stop Byte
Packet Fields Byte Location	0	1	2	3	4	5	6	7	..	...
Packet fields Name & Sequences	Device Type	Packet Type	Length	Checksum	Sequence Number			Additional payload		Stop Byte
					seqN Hi	seqN Mid	seqN Low			
Packet Fields Value	0xA0	0x0B	n+ payload length	0x00	0x00	0x00	0x00			0xFF

Table A.7.13: OPC_ACK_PAYLOAD Packet

OPC_NIRS_DATA Packet

This packet contains NIRS data that has been captured by the sensor. The number of channels is 1 byte long and contains the number of channels for the device.

OPC_NIRS_DATA	Packet									
	Header				Optional Payload					Stop Byte
Packet Fields Byte Location	0	1	2	3	4	5	6	7	..	n-1
Packet fields Name & Sequences	Device Type	Packet Type	Length	Check sum	Sequence Number			Num Of channels	Pay load	Stop Byte
					seqN Hi	seqN Mid	seqN Low			
Packet Fields Value	0xA0	0x04	n+ payload length	0x00	0x00	0x00	0x00			0xFF
					CH0	CH0	CH1	CH1		

Table A.7.14: OPC_NIRS_DATA Packet

OPC_WRAPPED Packet:

This packet contains another complete packet in the payload.

OPC_WRAPPED	Packet									
	Header				Optional Payload					Stop Byte
Packet Fields Byte Location	0	1	2	3	4	5	6	7	..	n-1
Packet fields Name & Sequences	Device Type	Packet Type	Length	Check sum	Sequence Number			Payload		Stop Byte
					seqN Hi	seqN Mid	seqN Low			
Packet Fields Value	0xA0	0x0B	n+ payload length	0x00	0x00	0x00	0x00			0xFF
Another packet	Packet									
	Header				Optional Payload					Stop Byte
Packet Fields Byte Location	0	1	2	3	4	5	6	7		n-1
Packet fields Name & Sequences	Device Type	Packet Type	Length	Check sum	Sequence Number			payload		Stop Byte
					seqN Hi	seqN Mid	seqN Low			
Packet Fields Value	0xA0	0x01	n+ payload length	0x00	0x00	0x00	0x00			0xFF

Table A.7.15: OPC_WRAPPED Packet

OPC_STATUS_INFO Packet

This packet contains information about the device status.

OPC_STATUS_INFO	Packet									
	Header				Optional Payload					
Packet Fields Byte Location	0	1	2	3	4	5	6	7	..	
Packet fields Name & Sequences	Device Type	Packet Type	Length	Check sum	Sequence Number			Payload		
					seqN Hi	seqN Mid	seqN Low			
Packet Fields Value	0xA0	0x06	n+ payload length	0x00	0x00	0x00	0x00			

Table A.7.16: OPC_STATUS_INFO Packet

User

Mobile Client

Starting the System:

- Make sure the server is up and running as described in the server section.

The system must be in “Waiting Status”.

Figure B.7.3: Server Status

- Start the PDA: Press the power button. When the PDA starts it will bring up the screen below.

Figure B.7.4: PDA Main Screen
The main screen when the PDA is started

- Click on the menu key on the PDA . The screen below will appear.

Figure B.7.5: Menu Screen

- In the PDA move the mouse on top of the MNIRS icon and click the mouse to start the application. Alternatively move the mouse over the icon and click on options>open
- The splash screen will appear. Click OK with the mouse.

Figure B.7.6: Splash Screen

- The application main screen will appear. See below.

Figure B.7.7: Application Main Screen

- Start the sensor: push the power button and make sure the green light is on and the light source is off in the bottom of the sensor.

Figure B.7.8: Sensor Front

- Click "Options": Application main menu will appear.

Figure B.7.9: Application Menu

- 1- ***Start Measurements Online***: Allow the user to start measurements and send the data to the server in real time.
- 2- ***Start Measurements Offline***: Allow the user to start measurements offline and save the data to a file in the PDA. In this case the system will not send the data to the server.
- 3- ***Stop Measurements***: After each start measurements, a stop operation must be performed.
- 4- ***Setup Server IP***: This option allows the user to point the PDA to a specific server.
- 5- ***Exit***: Exit the system.

Note: after each measurement it is better to switch off the sensor, wait 5 seconds and switch it on again.

Online Measurements

- When you click on online measurements the system will prompt the user with several screens to confirm the operations. The user should accept all the screens and answer “yes”.

File Access Request Screens: All requests should be answered with “Yes”.

- Answer Yes

Figure B.7.10: File Access Request

- Answer Yes

Figure B.7.11: File Access Request

- Answer Yes

Figure B.7.12: File Access Request

- Answer Yes

Figure B.7.13: File Access Request

Internet Network Access Request Screens:

- Answer Yes

Figure B.7.14: Network Access Request

- Internet Option should be selected.

FigureB.7.15: Network Selection

Bluetooth Network Access Request Screen:

- Enter the pass code “0000”.

Figure B.7.16: Bluetooth Password Screen

System Status Screens:

- The screens below show the status of the systems. It is divided into three sections.
 - 1- **Sensor Status:** Display the Bluetooth connection status, any errors that might occur and display the send operation status.
 - 2- **Server Status:** Display status and information related to the Internet server.
 - 3- **File Server Status:** Display status related to the file server.

Figure B.7.17: System Status Screen

- Check the sensor light to make sure it is red.

Figure B.7.18: Sensor Bottom

- When the system displays Polling Status for more than 30 seconds with the same number, please rest the sensor and the system. In this case the sensor needs to be charged.

Figure B.7.19: Pooling Status

- To finish the measurements selection options, select Stop measurements and then select Exit.

Figure B.7.20: Stop Measurements Screen

Offline Measurements

- When you click on Offline measurements the system will prompt the user with several screens to confirm the operations. The user should accept all the screens and answer “yes”.

File Access Request Screens: All requests should be answered with “Yes”.

- Answer Yes

Figure B.7.21: File Access Request

- Answer Yes

Figure B.7.22: File Access Request

- Answer Yes

Figure B.7.23: File Access Request

- Answer Yes

FigureB .7.24: File Access Request

Bluetooth Network Access Request Screen:

- Enter the pass code “0000”.

Figure B.7.25: Bluetooth Password Screen

System Status Screens:

- The screens below show the status of the systems. It is divided into three sections.

1- **Sensor Status:** Display the Bluetooth connection status, any errors that might occur, and display the send operation status.

2- **Server Status:** When performing the measurements offline, the server log will display “Server Started Offline” to indicate that the system is working offline.

3- **File Server Status:** Display status related to the file server.

Figure 7 B.7.26: System Status Screen

- When the system displays Polling Status for more than 30 seconds with the same number, please rest the sensor and the system. In this case the sensor needs to be charged.

Figure B.7.27: Pooling Status

- To finish the measurements, select Stop measurements and then select Exit.

Figure B.7.28: Stop Measurements Screen

Setup Server IP

- The PDA needs to be configured to point to a web server. This menu option allows the user to setup the server IP address and the server Port/Socket.

Figure B.7.29: Setup Server IP menu

The system will save the server configuration in a file in the PDA. This setup is required to be done once when the server is set up for the first time. The data will be stored in a file in the PDA, Phone memory\Data\Images\serverIP.dat.

- **Server IP Address:** The server address that has the server.
- **Server Socket:** The socket that the server is listening to.
- **Status:** Display the save operation result.

Figure B.7.30: Server IP Configuration

Measurement Data

Offline Measurement Data

The data is saved in the phone's memory and follows the pattern below. The ID is the combination of date and time in an integer format.

"Phone memory\Data\Images\traceFile_***_.dat"

The data file is downloaded from the phone and imported using the server. The server will convert the data into a comma delimited file ready for analysis by Matt Lab. The output file from the import operation will have the following file name pattern:

offLinepayload_99_****_.dat.

Figure B.7.31: Import Menu

Online Measurement Data

The measurement data is saved in the server under \data\payload*****.dat.

The ID is the combination of date and time in an integer format. The output file is a comma delimited file ready to be imported into Matt Lab.

Server

Server Setup

The server needs to be running on a specific port. The port number is setup in a file called serverport.txt. The port number needs to be accessible from the Internet. Open the file with a text editor and insert the port number.

Start Operations

Start Server

To start the server select go.cmd and wait for the system to start. Ensure that you get the screen below and the server status is waiting. When the PDA connects to the server the server status will change to running.

Figure B.7.32: Server Screen

Stop Server

To stop the server, click on Administration menu and select exit.

Figure B.7.33: Exit Menu

Server Screen Description

Figure B.7.34: Server Main Screen

- **Display:** The graph displays the received data from the PDA.
- **Duration:** The elapsed time since the PDA connected to the server and started sending data to the server.
- **Current Packet Seq Number:** The packet sequence number is just received from the PDA.
- **Received Packet Value for Current Display Channel:** The graph can display any channel data. This field shows the current display channel.

- ***Channels (0,1,2):*** Channels values.
- ***Current Display Channel:*** The user can change the current graphed channel.
- ***Value Upper Limit:*** The upper limit for the graph.
- ***Value Lower Limit:*** The lower limit for the graph.
- ***Frequency:*** Time scale.

Refresh Button is used to update the information and refresh the screen to display based on the user input.

Administrator

Mobile Client

Installation

To install the system, copy the following files to the PDA and run the installation wizard.

- MNIRS.jar
- MNIRS.jad

Configurations

To configure the server, select from the application menu, Setup Server IP Address.

Server

Installation

Copy the files to a server that has Internet access and has Java 5.0 installed.

Configurations

Set up the server port number in the file, serverport.txt.

Appendix C: Smoking Experiment Steps

Test Subject Information

Test Subject Information		
Date		DD:MM:YYYY
Time		HH:MM:SS AM/PM
Age		N Years
Gender		M/F
Number of Years of Smoking		N Cig
Number of Cigarettes Smoked Every Day		N Cig
Regular Smoker		Y/N
Age Started Smoking		N Years
Location		
Picture/Video Filename		

Table C.7.17: Test Subject Information

Notes:

Test Procedures

Note: The total duration of the test is 60 minutes.

Step1: Baseline

1. Start the sensor.
2. Fix the sensor on the head of the test subject. Make sure it has good contact.
3. Ask the test subject to be relaxed and breath normally.
4. Start the measurements.
5. Make sure the test subject is relaxed for 5 minutes so you can get the baseline data.
6. After 5 minutes, light the cigarette for the test subject.
7. Ask the test subject to smoke every 20 seconds.
8. Keep the subject smoking for a duration of 5 minutes.
9. After 5 minuets of smoking have passed, ask the test subject to stop.
10. Ask the test subject to be relaxed and breath normally again.
11. After 5 minutes from when the test subject stopped smoking, stop the measurements and the sensor.
12. Remove the sensor.
13. Record the file name

Repeat the above test procedures three times.

Appendix D: Code Snippets

```
//-----
// Main Controller
//-----
public class Controller extends Thread {
    public Sensor sensor = null;
    public boolean stopFlag = false;
    public boolean aliveFlag = false;
    public int sentToServer_seq_nr = -1;
    public BTSensorPacket packet;
    // -----
    // Controller
    // -----
    public Controller() {
        Util.sensorLog("Sensor Thread started");
        aliveFlag = true;
    }
    // -----
    // run
    // -----
    public void run() {
        if (Server.mode == Server.onlineMode)
            Server.openServer();

        Util.sensorLog("Starting Sensor....");
        sensor = new Sensor();
        if (sensor.openBluetooth() == true) {
            if (sensor.startMeasuerments() == true) {
                sensor.connectToSensorBuffer();
                while (!stopFlag) {
                    // Util.sensorInfo("Polling... " +
                    sentToServer_seq_nr);
                    packet = sensor.readPacket();
                    sentToServer_seq_nr = packet.GetSeqNr();

                    Util.sensorInfo("Sending to Server ... "
                        + sentToServer_seq_nr);
                    LogFile.write("sentToServer_seq_nr.txt",
                        sentToServer_seq_nr + "");
                    Server.sendToServre(packet.content);
                }
            } // if start bluetooth=true
        } // if bluetooth ==true
        sensor.stopMeasuerments();
        sensor.powerOffBluetooth();
        sensor.closeBluetooth();
        if (Server.mode == Server.onlineMode)
            Server.closeServer(); // Close
        Server.stopFlag = true;
        Util.sensorInfo("Sensor Stoped On : " +
            Util.getTimeString());
    }
}
```


Code Snippet D.1: Main Controller Class

```

public class Sensor {
    //-----
    // Sensor Variables
    //-----
    public static StreamConnection sensorConn = null;
    public static InputStream SensorIn =null;
    public static OutputStream SensorOut=null;
    BluetoothComManager commManger =null;
    public int cmd_seq_nr = 0;

    //-----
    // openBluetooth
    //-----
    public boolean openBluetooth(){

        Util.sensorLog("Openning Bluetooth Connection ...");

        try {

            String url =
                "btspp://00025b00a6a5:1;authenticate=false;
                encrypt=false;master=false";
            sensorConn = (StreamConnection) Connector.open(url);
            SensorIn = sensorConn.openInputStream();
            SensorOut = sensorConn.openOutputStream();

        } catch (IOException e) {

            Util.sensorError("Can not Open Bluetooth Connection "+
                e.getMessage());

            return false;

        }

        Util.sensorLog("Opened Bluetooth Connection Successfully");

        return true;

    }

    //-----
    // closeBluetooth
    //-----
    public boolean closeBluetooth() {
        try {

            Util.sensorLog("Closing Bluetooth Connections...");

            if(SensorIn!=null) {
                SensorIn.close();
                SensorOut.close();
                sensorConn.close();
            }

        } catch (IOException e) {
            Util.sensorLog("Can not close Bluetooth " + e.getMessage());
        };

        return false;
    }
}

```

```

    }
    Util.sensorLog("Bluetooth Closed Connections Successfully");
    return true;
}

//-----
//  startMeasuerments
//-----
public boolean startMeasuerments() {

    Util.sensorLog("Starting Measurements ");

    byte[] raw_packet = new
        byte[BTSensorConstants.OPC_CTRL_COMMAND_LENGTH];

    raw_packet[BTSensorConstants.TYPE_ID_POS] =
        BTSensorConstants.UP_ID_CODE;
    raw_packet[BTSensorConstants.OPCODE_POS] =
        BTSensorConstants.OPC_CTRL_COMMAND;
    raw_packet[BTSensorConstants.LENGTH_POS] =
        BTSensorConstants.OPC_CTRL_COMMAND_LENGTH;
    raw_packet[BTSensorConstants.PAYLOAD_POS] =
        (byte) (cmd_seq_nr >> 16);
    raw_packet[BTSensorConstants.PAYLOAD_POS + 1] = (byte)
        (cmd_seq_nr >> 8);
    raw_packet[BTSensorConstants.PAYLOAD_POS + 2] = (byte)
        (cmd_seq_nr);
    raw_packet[BTSensorConstants.CC_CTRLCODE_POS] =
        BTSensorConstants.START_MEASURE;
    raw_packet[raw_packet.length - 1] =
        BTSensorConstants.STOP_BYTE;
    cmd_seq_nr++;

    try {
        if (SensorOut!=null) {
            SensorOut.write(raw_packet);
            SensorOut.flush();
        }
    } catch (IOException e) {
        Util.sensorError("Can not Start Measuerments "
            +e.getMessage());

        return false;
    }
    Util.sensorLog("Measurements Started Successfully ");
    return true;
}

//-----
//  stopMeasuerments
//-----
public boolean stopMeasuerments() {

    Util.sensorLog("Stoping Measurements ");

    byte[] raw_packet = new
        byte[BTSensorConstants.OPC_CTRL_COMMAND_LENGTH];
    raw_packet[BTSensorConstants.TYPE_ID_POS] =

```

```

        BTSensorConstants.UP_ID_CODE;
raw_packet[BTSensorConstants.OPCODE_POS] =
        BTSensorConstants.OPC_CTRL_COMMAND;
raw_packet[BTSensorConstants.LENGTH_POS] =
        BTSensorConstants.OPC_CTRL_COMMAND_LENGTH;
raw_packet[BTSensorConstants.PAYLOAD_POS] =
        (byte) (cmd_seq_nr >> 16);
raw_packet[BTSensorConstants.PAYLOAD_POS + 1] =
        (byte) (cmd_seq_nr >> 8);
raw_packet[BTSensorConstants.PAYLOAD_POS + 2] =
        (byte) (cmd_seq_nr);
raw_packet[BTSensorConstants.CC_CTRLCODE_POS] =
        TSensorConstants.STOP_MEASURE;
raw_packet[raw_packet.length - 1] =
        TSensorConstants.STOP_BYTE;

cmd_seq_nr++;

try {
    if (SensorOut!=null) {
        SensorOut.write(raw_packet);
        SensorOut.flush();
    }
} catch (IOException e) {
    Util.sensorError("Can not Stop measurements "+
        e.getMessage());
    return false;
}
Util.sensorLog("Measurements Stopped Successfully ");
return true;
}

//-----
//  powerOffBluetooth
//-----
public boolean powerOffBluetooth(){

    Util.sensorLog("Powering Off Bluetooth ");

    byte[] raw_packet = new
        byte[BTSensorConstants.OPC_CTRL_COMMAND_LENGTH];
raw_packet[BTSensorConstants.TYPE_ID_POS] =
        BTSensorConstants.UP_ID_CODE;
raw_packet[BTSensorConstants.OPCODE_POS] =
        BTSensorConstants.OPC_CTRL_COMMAND;
raw_packet[BTSensorConstants.LENGTH_POS] =
        BTSensorConstants.OPC_CTRL_COMMAND_LENGTH;
raw_packet[BTSensorConstants.PAYLOAD_POS] =
        (byte) (cmd_seq_nr >> 16);
raw_packet[BTSensorConstants.PAYLOAD_POS + 1] =
        (byte) (cmd_seq_nr >> 8);
raw_packet[BTSensorConstants.PAYLOAD_POS + 2] =
        (byte) (cmd_seq_nr);
        raw_packet[BTSensorConstants.CC_CTRLCODE_POS] =
        BTSensorConstants.POWER_OFF;

```

```

        raw_packet[raw_packet.length - 1] =
            BTSensorConstants.STOP_BYTE;
        cmd_seq_nr++;

        try {
            if (SensorOut!=null) {
                SensorOut.write(raw_packet);
                SensorOut.flush();
            }
        } catch (IOException e) {
            Util.sensorError("Can not Power Off Bluetooth "
                "+e.getMessage());
            return false;
        }
        Util.sensorLog("Bluetooth PoweredOff Successfully ");
        return true;
    }

    //-----
    // connectToSensorBuffer()
    //-----
    public void connectToSensorBuffer() {
        commManger = new BluetoothComManager();
        try {
            commManger.Connect(SensorIn);
        } catch (Exception e) {

            Util.sensorError("Can not Pass Input Stradm to
                sensor"
                    + e.getMessage());

        }

        Util.sensorLog("Sensor Started On : Successfully"
            + Util.getTimeString());
    }

    //-----
    // connectToSensorBuffer()
    //-----

    public BTSensorPacket readPacket() {
        BTSensorPacket packet;
        packet = (BTSensorPacket) commManger.nirs_data_q
            .WaitThenPoll();
        return packet;
    }
}

```

Code Snippet D.2: Sensor Class

```

//-----
// Server Manager
//
//-----
public class Server extends Thread {

    //-----
    static SocketConnection serverConn;
    static OutputStream serverOut;

    //-----
    public static boolean stopFlag = false;
    public static boolean aliveFlag=false;

    public static final boolean online = true;
    public static final boolean onlineMode = true;
    public static final boolean offline=false;
    private static boolean serverConnectionFlag=false;
    public static boolean mode=true;

    //-----
    // Server
    //-----
    public Server(MNIRS mdl) {

        Util.serverLog("Servre Thread started");
        aliveFlag=true;

    }

    //-----
    // Server
    //-----
    public Server() {

        Util.serverLog("Servre Started Offline ");
        aliveFlag=true;
        //openServer();

    }

    //-----
    // getOutputStream
    //-----
    public OutputStream getOutputStream() {
        return serverOut;
    }

    //-----
    // run
    //-----
    public void run() {

        while (!stopFlag) {
            try {

```

```

        Util.zzz(1000000000);
    }catch (Exception e){

    }
}
Util.serverLog("Server Stopped Successfully");

aliveFlag=false;
}
//-----
// openServer
//-----
public static boolean openServer() {
    try {

        Util.serverLog("Opening Server Connection Please Wait...");
        String ips=IPWriter.readFromFile();
        if(!ips.startsWith("[")
            ips="192.168.0.115:8090";
        else {
            int end=ips.indexOf("]");
            ips=ips.substring(1,end);
        }
        System.out.println("Server IP = "+ips);
        serverConn = (SocketConnection)
        Connector.open("socket://" +ips);

        serverOut = serverConn.openOutputStream();

        serverConnectionFlag=online;
    } catch (IOException e) {

        serverOut = null;
        serverConn = null;

        Util.serverLog("The System will Work Offline");
        Util.serverError("" +e.getMessage());

        serverConnectionFlag=offline;
        return false;

    }

    Util.serverLog("Server Started Successfully on :
        "+Util.getTimeString());

    return true;
}
//-----
// closeServer
//-----
public static boolean closeServer() {
    try {

        Util.serverLog("Closing Server Connection ...");

```



```

        if(serverOut!=null ){
            serverOut.close();
            serverConn.close();
            serverConnectionFlag=offline;
        }

    } catch (IOException e) {
        Util.serverError("Can not Close Servre Connection
            "+e.getMessage());
        serverConnectionFlag=offline;
        return false;
    }
    Util.serverLog("Server Connection Closed Sucessfully" );

    return true;
}
//-----
//  sendToServre
//-----
public static boolean sendToServre(byte [] data) {

    try {
        if (serverOut != null) {
            serverOut.write(data);
            serverOut.flush();
        } else {
            throw new IOException();
        }

    } catch (IOException e) {
        serverConnectionFlag=offline;
        Util.serverError("Sending Data  to File Server
            "+e.getMessage());
        LogFile.writeArrayToFile(data,"dump_bin.txt");
        FileManager.writeToFile(data);
        return false;
    }
    return true;
}
//-----
//  setServerStatus
//-----
public static void setServerStatus(boolean flag){
    serverConnectionFlag=flag;
}

//-----
//  getServerStatus
//-----
public static boolean getServerStatus(){
    return serverConnectionFlag;
}
}

```

Code Snippet D.3: Server Class


```

public class Packet implements PacketCodec {
    private static final byte STOP_BYTE = (byte) 0xFF;
    public int length;
    public byte[] content = null;

    protected byte mbyteDeviceType = 0;
    protected byte mbytePacketType = 0;

    protected byte mbytChecksum = 99;
    protected int mintCmdSeqNum = 0;

    protected byte mbytCtrlCode = 0;
    protected byte mbytErrorCode = 0;

    protected int mintMeasSeqNum = 0;
    protected int mintMeasResolution = 0;
    protected int[] mintChannelData; //SAL
    protected int channelNum = 0;
    /**
     * Represents the Sequence number of the packet. If the
     * packet does not have a sequence number, then it is 0.
     */

    public byte getDeviceType()
    {
        return mbyteDeviceType;
    }

    public void setDeviceType(byte byteDeviceType)
    {
        mbyteDeviceType = byteDeviceType;
    }

    public byte getPacketType()
    {
        return mbytePacketType;
    }

    public void setPacketType(byte bytePacketType)
    {
        mbytePacketType = bytePacketType;
    }

    public byte getChecksum() {
        return mbytChecksum;
    }

    public void setChecksum(byte bytChecksum) {
        mbytChecksum = bytChecksum;
    }

    public int getCmdSeqNum() {
        return mintCmdSeqNum;
    }

    public void setCmdSeqNum(int intCmdSeqNum) {

```

```

        mintCmdSeqNum = intCmdSeqNum;
    }

    public byte getCtrlCode() {
        mbytCtrlCode =
            content[PosOffsetConstants.CC_CTRLCODE_POS+1];
        return mbytCtrlCode;
    }

    public void setCtrlCode(byte bytCtrlCode) {
        mbytCtrlCode = bytCtrlCode;
    }

    public byte getErrorCode() {
        return mbytErrorCode;
    }

    public void setErrorCode(byte bytErrorCode) {
        mbytErrorCode = bytErrorCode;
    }

    public int getMeasSeqNum() {
        return mintMeasSeqNum;
    }

    public void setMeasSeqNum(int intMeasSeqNum) {
        mintMeasSeqNum = intMeasSeqNum;
    }

    public int[] getChannelData() {
        return mintChannelData;
    }

    public void setChannelData(int[] intChannelData) {
        mintChannelData = intChannelData;
    }

    public int getMeasResolution() {
        return mintMeasResolution;
    }

    public void setMeasResolution(int intMeasResolution) {
        mintMeasResolution = intMeasResolution;
    }

    public byte[] encode()
    {
        byte bytLength = CalcPacketLength();

        byte[] bytPacket = new byte[bytLength];

        bytPacket[PosOffsetConstants.TYPE_ID_POS] =
            mbyteDeviceType;
        bytPacket[PosOffsetConstants.OPCODE_POS] = mbytePacketType;
        bytPacket[PosOffsetConstants.LENGTH_POS] = bytLength;
    }

```

```

        bytPacket[PosOffsetConstants.CHECKSUM_POS] = mbytCheckSum;

        encodePayload(bytPacket);

        bytPacket[bytLength-1] = STOP_BYTE;

        return bytPacket;
    }

    public byte getStopByte(){
        return content[length-1];
    }

    public void decode(byte[] arrPacket)
    {
        mbyteDeviceType =
            arrPacket[PosOffsetConstants.TYPE_ID_POS];
        mbytePacketType = arrPacket[PosOffsetConstants.OPCODE_POS];
        mbytCheckSum = arrPacket[PosOffsetConstants.CHECKSUM_POS];

        decodePayload(arrPacket);
    }

    protected byte CalcPacketLength()
    {
        return 0;
    }

    protected void encodePayload(byte[] arrPacket)
    {
    }

    protected void decodePayload(byte[] arrPacket)
    {
    }

    public int getChannelNum() {
        return channelNum;
    }

    public void setChannelNum(int channelNum) {
        this.channelNum = channelNum;
    }

    public void extractSensorPacket(byte[] raw_packet) {
        mbyteDeviceType =
            raw_packet[BTSensorConstants.TYPE_ID_POS];
        mbytePacketType = raw_packet[BTSensorConstants.OPCODE_POS];
        length = raw_packet[BTSensorConstants.LENGTH_POS];
        content = raw_packet;

        switch(mbytePacketType) {
            case BTSensorConstants.OPC_EMPTY:
                break;
            case BTSensorConstants.OPC_UP_PACKET:
                break;
            case BTSensorConstants.OPC_CTRL_COMMAND:
                getSeqNr();
        }
    }

```

```

        getSeq();
        break;
    case BTSensorConstants.OPC_NIRS_DATA:
        getSeqNr();
        break;
    case BTSensorConstants.OPC_CURRENT_COMMAND:
        getSeqNr();

        break;
    case BTSensorConstants.OPC_CURRENT_REPLY:
        break;
    case BTSensorConstants.OPC_ACK:
        getSeqNr();
        break;
    case BTSensorConstants.OPC_ACK_PL:
        getSeqNr();
        break;
    case BTSensorConstants.OPC_NACK:
        getSeqNr();
        break;
    case BTSensorConstants.OPC_NACK_ERROR:
        getSeqNr();
        break;
    case BTSensorConstants.OPC_NOT_SPECIFIED:
        break;
    case BTSensorConstants.OPC_ERROR:
        break;
}
}

private void getSeqNr() {
    int lo = content[BTSensorConstants.PAYLOAD_POS +
        BTSensorConstants.SEQ_NR_INSIDE_PL_POS + 2];
    int mid = content[BTSensorConstants.PAYLOAD_POS +
        BTSensorConstants.SEQ_NR_INSIDE_PL_POS + 1];
    int hi = content[BTSensorConstants.PAYLOAD_POS +
        BTSensorConstants.SEQ_NR_INSIDE_PL_POS];

    if (lo < 0) lo += 256;
    if (mid < 0) mid += 256;
    if (hi < 0) hi += 256;

    mintCmdSeqNum = (hi << 16) + (mid << 8) + lo;
}

public void setSeq () {
    content[PosOffsetConstants.PAYLOAD_POS] = new
        Integer(mintCmdSeqNum >> 16).byteValue();
    content[PosOffsetConstants.PAYLOAD_POS + 1] = new
        Integer(mintCmdSeqNum >> 8).byteValue();
    content[PosOffsetConstants.PAYLOAD_POS + 2] = new
        Integer(mintCmdSeqNum).byteValue();
}

public int getSeq() {

```

```

        mintCmdSeqNum = (content[PosOffsetConstants.PAYLOAD_POS] &
0xFF) << 16;
        mintCmdSeqNum += (content[PosOffsetConstants.PAYLOAD_POS +
1] & 0xFF) << 8;
        mintCmdSeqNum += (content[PosOffsetConstants.PAYLOAD_POS +
2] & 0xFF);
        return mintCmdSeqNum;
    }
}

```

Code Snippet D.4: LayerPro Packet Class

```

while (true) {

    try {

        System.out.println("Waiting.....");
        check = 0;
        clientSocket = serverSocket.accept();
        sw.start();
        fileID++;
        outPutFileName = "payload" + fileID +
            Util.getTM() + ".dat";
        System.out.println("Running.....");
    } catch (IOException e) {
        System.out.println("Accept failed: 8090");
        System.exit(-1);
    }

    try {
        datain = clientSocket.getInputStream();

    } catch (IOException e) {

        System.out.println("datain : Could not get
            getInputStream");
    }

    try {
        while (true) {

            value = datain.read(byteArray);

            if (value == -1) {
                ii = 0;
                System.out.println("No More data-----
                    ----");
                throw new IOException();
            } else {

                packet = new
                    BTSensorPacket(byteArray);

                sample_seq_nr = packet.GetSeqNr();
                check++;
                saveWithUpdateGUI(Util.byteArrayToS
                    tring(packet.content),
                    outPutFileName, check);
                kk++;
            }
        }
    } catch (IOException e) {

```

Code Snippet D.5: LayerPro Server Class

Appendix E : Publications

This thesis work has generated the following publications:

- **Salah Sharieh**, Alexander Ferworn, Vladislav Toronov, (2008), "A GSM Mobile System to Monitor Brain Function Using a Near-infrared Light Sensor", in Proceedings of the 21st Canadian Conference on Electrical and Computer Engineering, Computer Applications, Niagara Falls, Ontario, Canada.
- **Salah Sharieh**, Alexander Ferworn, Vladislav Toronov, Abdolreza Abhari. "An Ad-hoc Network Based Framework for Monitoring Brain Function", the 11th Communications and Networking Simulation Symposium, Ottawa, Canada, April 14-17 2008.
- Olivia Pucci, **Salah Sharieh**, Vladislav Toronov. "Spectral and Spatial Characteristics of the Differential Pathlengths in Non-homogeneous Tissues", Proceedings of SPIE – Volume 6855, 1, 685506, Complex Dynamics and Fluctuations in Biomedical Photonics (Feb. 6, 2008).
- Olivia Pucci, **Salah Sharieh**, Vladislav Toronov, Alex Ferwon. "Examining the Effects Naturally Administered Smoke Has on Cerebral Hemodynamics Using a Fully Mobile Near-infrared Sensor", Canadian Graduate Student Conference on Biomedical Computing (March. 13, 2008).

**PROPERTY OF
RYERSON UNIVERSITY LIBRARY**