Designing principles for mobile application data of body sensors on physical activities

Amin Amini Marvast
Garsivaz Rafieian
Garsivaz Rafieian
Amin Amini Marvast

Designing principles for mobile application data of body sensors on physical activities

Master Thesis

School of Mathematics and Systems Engineering

2009
Abstract

This thesis has been divided into two essential parts, the purpose of the first part is to investigate and explore a three-tier architecture for remote health monitoring system capable to collect, store and forward the physiological data, which has been collected by a mobile device via a Bluetooth connection from body sensors, to an internet data base server.

During the second part, we have tried to take a deep look into a heart beat modeling method. We have studied and investigated on extended integral pulse frequency modulation (IPFM) model which is used for the presence of ectopic beats and heart rate turbulence (HRT). Also it can be used for detection and characterization of HRT.
Acknowledgements

We started this thesis in June of 2008 and the seminar was held in February of 2009. We hope that the reader of this report will conduct further investigation on this subject.

We wish to thank master student Flavio Ruvalcaba, our friend, for his assistance during the first part of the thesis, and our special thanks to our supervisor Marcelo Milrad and our examiner professor Sven Nordebo for introducing and suggesting the topic of this thesis and their cooperation throughout this master thesis. Finally, we would like to thank our families for their supports along the process of this thesis.
## Contents

1 **Introduction** .......................................................... 7

2 **Mobile Devices Descriptions** ........................................... 9
   2.1 Hardware ........................................................................ 9
       2.1.1 Mobile Device .......................................................... 9
       2.1.2 Wireless Sensor ....................................................... 10
       2.1.3 GPS Device ............................................................. 11
   2.2 Software Description ..................................................... 11
       2.2.1 .NET Compact Framework ......................................... 12
       2.2.2 Software Development (Introduction to C#) ...................... 12
       2.2.3 Getting Started ....................................................... 13
       2.2.4 EMEI Number Application ........................................... 15
       2.2.5 GPS Handler Application ............................................ 15
       2.2.6 .NET Compact Framework HTTP Communication .................. 20

3 **Extended Integral Pulse Frequency Modulation (IPFM) Model** ........ 25
   3.1 Turbulence Onset (TO) and Turbulence Slope (TS) .................. 26
3.2 HRT Detection Based on Extended IPFM Model................................. 26

4 Conclusions ................................................................. 29

References ................................................................. 31

A C# Code ................................................................. 35

A.1 Phone Number and EMEI Number in C# ........................................... 35

A.2 Mapping Serial Port and Com Ports (GPS Device Bluetooth).............. 41
Chapter 1

Introduction

The availability of communication systems such as the public switched telephone network, the Global System for Mobile communication (GSM) network, the Internet, and proprietary wide area networks have enabled some parts of the healthcare industry to transition from in-person visits to remote consultation. Fitness and Sport fields were also between those which have been affected and facilitated in many ways by wireless telecommunication.

Wireless sensor networks (WSNs) will enable point-of-care-diagnostics and provide the data input to e-health management systems that will allow significant enhancement of healthcare programs. They will also support the evolution of wearable medical devices aimed at improving the fitness and general well-being of the individual.

In the first part of this thesis, chapter 2, it is explored and designed a three-tier architecture for remote condition capable to collect, store and forward the navigation and identification data, which has been collected by a windows mobile phone from GPS device (via a Bluetooth connection), also including mobile device identification, to a database server over the internet. We have tried to create an independent communication environment. This chapter is divided into two main sections (hardware and software). In the hardware section, devices that has been used and related to this thesis have been described in detail. In the software section, programming language and software that have been developed and used by, either the hardware devices and thesis members, are discussed.

So, as a scenario in this project, imagine a running competition in a Nordic forest with several runners who attend to this event. The area has been divided to several territories on Google map at the data base server. On the map each competitor is shown by a flag and the position of the flag is being updated every few seconds by the GPS data sending to server via internet connection.
Chapter 1. Introduction

As a future project, each person can hold an Alive heart monitor and a GPS Windows mobile phone with a digital camera (Alive heart monitor and phone are connected via Bluetooth) in the competition. So, one task is to make a both way connection via internet over HTTP to the server (long term monitoring ECG project), first two tasks (navigation and identification data detection) has been done in this thesis. The goal in second-tier could be to observe ECG signals via Bluetooth connection in real time and later for advance analyzes on windows mobile phone. By installing the AliveECG on the phone or pc you can observe the heart beat signals in real time, see figure 1.1. Some related projects that can be useful to develop this thesis for future (through health monitoring field) can be named as cardioWiz and TripleBeat.

One of the most powerful predictor of mortality after acute myocardial infarction is Heart Rate Turbulence (HRT). In addition, the considering of HRT can be useful on some other areas such as, congestive heart rate and hypotension in hemodialysis patients.

In the second part of this thesis, chapter 3, the extended integral pulse frequency modulation model for which has been used the detection and characterization of HRT, has been studied.

Finally, the conclusions have been given in chapter 4, and the C# code can be found in appendix A.
Chapter 2

Mobile Device Descriptions

In this chapter we will look inside both hardware and software parts of this thesis. First part about the hardware covers the whole equipments used along the project such as Mobile devices, Wireless sensors and GPS devices. Description contains mostly the general functionality concept of the hardware devices but also in some cases it was critical for the reader to have some additional information about the manufacturer standards and device versions. In the second part about the software, we have explained the software that either was used by the equipments such as Wireless sensors and ECG device or those ones being used as software development tools like C#. And also we tried to explain some of the facilities that has been provided for application developments with these programming languages like LIBRARY, SDK, etc.

2.1 Hardware

2.1.1 Mobile Device

PDA: A personal data assistant (PDA) is a handheld computer also known as palmtop computers. Newer PDAs also have both color screens and audio capabilities, enabling them to be used as mobile phones, (Smartphone), web browsers, or portable media player. Many PDAs can access the internet, intranet or extranets via WIFI, or wireless Wide Area Networks (WWANs). Many PDAs employ touch screen technology [1].

Smartphone: Although there is no industrial standard definition of a Smartphone but maybe the broadest definition of a Smartphone is as Rick Roesler, vice president of handhelds for HP, said: "Smartphone are computers you talk to" [2].
Chapter 2. Mobile Device Descriptions

Smartphone is normally understood as a mobile phone that runs a complete operating system software providing a standardize interface and platform for application developers.

Operating systems:

A. Symbian Os from Symbian Ltd. It is used by many major handset manufactures, including BenQ, LG, Motorola, Samsung, and Sony Ericsson.

B. iPhone Os from Apple Inc.

C. RIM OS.

D. Windows Mobile (2003) from Microsoft. This Os is used in this project as the Os of mobile device. This Os as it is mentioned below is one of the platforms that AliveECG software supports and used in this thesis as the Os of the mobile device.

E. BlackBerry

Windows Mobile(2003) is released in four editions:

2. Windows Mobile 2003 for Pocket PC Professional Edition
3. Windows Mobile 2003 for Smartphone
4. Windows Mobile 2003 for Pocket PC Phone Edition

In the following some of the basic key concepts about application development are going to be introduced.

2.1.2 Wireless sensors

Wireless sensors that have been used in this thesis are Alive Heart Monitor. A monitoring health system for consumer’s health and fitness. Designed for use in the doctor’s office, home or gym, the monitor uses wireless Bluetooth and mobile phone networks to immediately transmit ECG and accelerometer data to a computer, PDA, or central monitoring centre. It features the latest generation of Bluetooth wireless technology and new electrode technology for optimum signal quality.
The Alive Heart Monitor can be used for remote real time monitoring of exercise programs via the internet. Using the Alive Heart Monitor, a GPS and mobile phone carried by the person exercising, the program supervisor or coach can remotely monitor the exact position, speed, ECG and heart rate in real time.

For cardiac rehabilitation programs, and for training programs for elite athletes this provides real-time feedback on the intensity and performance, allowing optimization of the programs for maximum benefit and safety.

Wireless sensor software: The AliveECG software connects to the Alive Heart Monitor over a wireless Bluetooth connection. It can be used to monitor a patient's ECG, heart rate, activity, and device status, and to record this data for later review.

The AliveECG software is provided with the Alive Heart Monitor and supports the following devices and platforms:

1. Windows Mobile Pocket PC 2003 and 5.0
2. Windows Mobile Smartphone 2003 and 5.0
3. Windows XP

Wireless sensors are assumed to be attached to the participant body while he/she is holding the mobile device and receiving ECG, GPS data from the wireless sensors via Bluetooth connection.

2.1.3 GPS Device

GPS Bluetooth navigator together with a pocket pc and Smartphone is a quite popular and fruitful method for navigation. To navigate the route in this scenario we used a Bluetooth GPS device beside our pocket pc. As it is explained in the following chapters, we have tried to establish a way for communication between mobile and GPS devices over HTTP and via serial port of the mobile device (in this case pocket pc) and also we made an effort to parse the GPS data (as NMEA number) in the mobile device to show the result simply as latitude and longitude.

2.2 Software Description
2.2.1 .NET Compact Frame Work

.NET Compact Framework is a version of the .NET Framework and its goal is to run the Windows CE (Windows Embedded Compact) based mobile/embedded devices such as PDAs, mobile phones, etc on it. The .NET Compact Framework uses some of the same class libraries as the full .NET Framework and also a few libraries designed specifically for mobile devices such as Windows CE InputPanel. However, the libraries are not exact copies of the .NET Framework. The ones in the .NET Compact Framework are scaled down to take up less space [8].

2.2.2 Software Development (Introduction to C#)

In this section, some of the most important concepts about Software Development in general and Visual C# programming language, in particular, have been defined and discussed. Nowadays some of these concepts such as Libraries, SDK, and API are essential tools in the hands of software developers.

Libraries:

Library usually refers to a set of classes using for software development including code and data to provide services to independent programs. Libraries are mostly not executable. Executables and libraries make references known as links to each other through the process known as linking, which is typically done by a linker.

Most modern operating systems (OS) provide libraries that implement the majority of system services. Such libraries have commoditized the services a modern application expects an OS to provide. As such, most code used by modern applications is provided in these libraries [3].

API:

An application programming interface (API) is a set of routines, data structures, object classes and/or protocols provided by libraries and/or operating system services in order to support the building of applications.

An API may be:

- Language-dependent, that is, only available in a particular programming language, utilizing the particular syntax and elements of the programming language to make the API convenient to use in this particular context.
• Language-independent, that is, written in a way that means they can be called from several programming languages (typically an assembly/C-level interface). This is a desired feature for a service-style API which is not bound to a particular process or system and is available as a remote procedure call.

**SDK:**

SDK stands for Software Development Kit. An SDK includes tools, headers, libraries, sample code, and documentation to help you develop and write to Microsoft technologies and products. They actually expose much of their code in the form of samples that will allow you to write against their providers. The Platform SDK encompasses the SDKs for most Microsoft products. It is available as a single download or in smaller components (individual product SDK’s)[4].

**Microsoft Developer Network:**

The goal of the MSDN is to hold a tight relationship between Microsoft and even developers standing on the other Os platforms such as: hardware developers interested in Os, developers leveraging API, etc. The relationship management is situated in assorted media: web sites, newsletters, developer conferences, trade media, blogs and DVD distribution. The life cycle of the relationships ranges from legacy support through evangelizing potential offerings [5].

**Visual C# Introduction:**

It is an extensive tool using C# development language to create Microsoft .Net-connected application for the web and the windows. Plus, get information on how to obtain this robust development package offering beginning and intermediate developers with C++ or Java experience a modern language and environment for creating next-generation software [6].

**Visual Studio introduction:**

Visual Studio is a complete set of development tools for building ASP.NET Web applications, XML Web Services, desktop applications, and mobile applications. Visual Basic, Visual C#, and Visual C++ all use the same integrated development environment (IDE), which enables tool sharing and eases the creation of mixed-language solutions. In addition, these languages use the functionality of the .NET Framework, which provides access to key technologies that simplify the development of ASP Web applications and XML Web Services [7].

2.2.3 Getting Started
Chapter 2. Mobile Device Descriptions

To start the connection between our mobile device, in this case smart phone or pocket pc, the first task would be writing a simple application just to initiate the very beginning interconnection.

After installation of visual studio.net 2005 and ActiveSync, to play the role of intermediate software between PC and mobile device, it is time to install Windows Mobile 5.0 SDK. After selecting a new project from the visual studio 2005 under C# programming and smart device division, windows mobile 5.0 pocket pc should be the option.

In the “Templates” panel device application must be selected. On the PDA skin in the design view by clicking on the surface (and F4 button) properties panel will be appeared. Changing the text property to “First APP”.

Next task is to drag a Label control and a Button control to the form. Then we must change the button text to “Click Me” and text property of the label to an empty string.

By double clicking on the button, Code view will be appeared, see figure 2.1. So the only remaining task is just to type the following line in the button1_Click method:

```c#
label1.Text = "Hello World!";
```

Figure 2.1: Hello World application on a pocket pc
2.2.4 EMEI Number Application

a) *EMEI number description*, The International Mobile Equipment Identity or IMEI is a unique number used by GSM network to validate devices. This number is used for all cell phones that operate under the global system for mobile communications (GSM) protocol. This is the standard system for cell phones not only in Europe, but also in over 100 countries worldwide [9].

b) *Structure of the IMEISV number*, It is a 16 digit number containing information about model, origin, and serial number of the device. It’s new format since 2004 includes of five parts such as Reporting Body Identifier(A), Remainder of the Type Allocation Code(B), Serial sequence of the model(C), “Luhn check digit” of the entire number(D), Software Version Number (E), in the form of AA-BBBBBB-CCCCC-D-EE.

Type Allocation Code is a portion of IMEISV to uniquely identify wireless devices. And “Luhn check digit” is simply an IBM made checksum formula for numbers identification validation. SVN or Software Version Number was added to IMEI number after 2004 to verify the Software Version of the mobile phones.

From GSM network point of view, Equipment Identity Register (EIR) works as a data base of all valid equipment on the network, see figure 2.2. Due to unique EMEI number for each MS, EIR has its own data list of stolen MS thus it is not easy anymore for the thieves to hide the mobile identity by simply changing the SIM.

c) *IMEI number application*, To track which device, with running application on it, is in charge of sending the data toward the server, we need to distinguish its EMEI number so Reading this number in C# will help us attaching the data to our mobile device, see figure 2.3.

2.2.5 GPS Handler Application

a) *An introduction to NMEA*, The National Marine Electronics Association is a specification defined by the U.S.-based National Marine Electronics Association which helps the communication between various marine electronic devices such as echo sounder, autopilot, GPS receivers (purpose of discussion) and many other types of instruments. NMEA 0183 sentences are all in ASCII and serial communication protocols and they describe how data is being transmitted from a “talker” to a “listener” in a so called “sentence”.

*NMEA Sentence Structures*: NMEA is 8 bit ASCII with MSB (most significant bit) = 0.
Figure 2.2: General GSM Architecture, Including the Three Main Separations in the Network [10].

Figure 2.3: Getting an EMEI number from a pocket pc [11].
There are some reserved characters to specify the formatting of the NMEA data and of course valid characters (usable) must be mentioned in a table.

Maximum number of characters in The NMEA 0183 specification version 3.01 shall be 82, consisting of a maximum of 79 characters between start of message “$” or “!” and terminating delimiter <CR><LF> (HEX 0D and 0A). The minimum number of fields is one (1).

In following we bring some NMEA Basic sentence formats [12] and some popular NMEA sentences respectively,

$$\text{Start of sentence}$$

$$\text{aaccc} \quad \text{Address field/Command}$$

$$\text{“,”} \quad \text{Field delimiter (Hex 2C)}$$

$$\text{c--c} \quad \text{Data sentence block}$$

$$\text{*} \quad \text{Checksum delimiter (HEX 2A)}$$

$$\text{hh} \quad \text{Checksum field (the hexadecimal value represented in ASCII)}$$

$$\text{<CR><LF>} \quad \text{End of sentence (HEX OD OA)}$$

Some typical NMEA sentences that are supported by Bluetooth GPS units are as follows: GGA, GSA, GSV, RMC, VTG, GLL

$$\text{GPGSA} - \text{GPS DOP and active satellites}$$

$$\text{GPGGA} - \text{Global Positioning System Fix Data}$$

$$\text{GPGSV} - \text{GPS Satellites in view}$$

$$\text{GPRMC} - \text{Recommended minimum specific GPS/Transit data}$$

$$\text{GPVTG} - \text{Track made good and ground speed}$$

$$\text{GPGLL} - \text{Geographic position, latitude / longitude}$$

The hardware interfaces in GPS devices are able to deal with NMEA sentences and also with RS232 protocol (a standard for serial binary data signals used in computer serial ports), see figure 2.4.

After this brief description to NMEA the task would be to read the GPS data from the serial port of mobile device (Smartphone or Pocket pc) and parse NMEA 0183 data (which is as mentioned above the standard GPS protocol) using .Net compact framework® and then present position as latitude and longitude (or even later other parameters such as speed and satellite information).

Receiving GPS data[13]: Microsoft® .NET framework class library (FCL), which provides an interesting range of the underlying Win32®API, contributing to the sophistication of the C# and visual basic ®.NET languages, does not cover RS232 serial communications. With the Platform
Invocation Services (P/invoke), .NET technology, one can manage code in the common language runtime (CLR) to make calls into unmanaged DLLs, including those that implements the Win32 API. So the task is to make the API functions provided for RS232 communications in CLR managed classes using C#.

After using the base classes and doing the com settings it is time to start receiving the data, and to do that we must first call the API ReadFile function. To encounter the difficulties of receiving times and to avoid forcing the application continually sampling the data we need a call back mechanism. So we call the CommeBase and CommLine methods on this way, first to invoke a virtual method on reception of each byte and then to override in CommLine to buffer up the bytes and when a line terminator is received, it is time to call a different virtual method.

```csharp
rxThread = new Thread(new ThreadStart(this.ReceiveThread));
rxThread.Name = "ComBaseRx";
rxThread.Priority = ThreadPriority.AboveNormal;
rxThread.Start;
Thread.Sleep(1);
rxThread.Abort;
```
As it can be seen now, here a new thread is running the code in the private method called ReceiveThread. Last line terminates the Thread.

NMEA Data Parsing:

The first step, as the most common NMEA sentence, here we can introduce the “Recommended Minimum” and below is an example which includes everything such as latitude, longitude, bearing, satellite-derived time, fix status and magnetic variation.

$GPRMC,025401.563,A,3939.7,N,10706.6,W,0.57,459.95,100903,,*1A

NMEA Data Parsing Interpreting:

1) Writing a method to separate the sentence into its words and try to understand which information’s are available to extract from the first sentence.

2) Beginning of data extraction. From left to right, fourth word of the sentence showing the latitude in form of “DDD°MM’S’SS.S” such as hours (D), minutes (M), Seconds(S). Longitudes also can be shown the same. Fifth and seventh words are showing the “hemisphere”, where “N” means north “w”, means west etc.

One thing to consider is to check the checksum control. Checksum is obtained via XOR of the bytes between the dollar sign and asterisk and comparing this XOR with the checksum of the sentence and unmatched answers are discarded. So the system can throw out any sentence with invalid checksum, see figure 2.5.

Figure 2.5: A general overview of Flow Control for GPS [15].
2.2.6 .Net Compact Framework HTTP Communication

In the following, the process of establishing a new connection with .Net compact Framework is going to be discussed. The final task is to be succeeded to upload data over the server via .Net compact Framework HTTP Communication.

a) start HTTP communication with building an HTTP client [16]

As a web application framework, ASP.NET is being used to construct dynamic websites, web applications and web services. It is developed by Microsoft and is the built on the Common Language Runtime (CLR) and therefore it allows the users to write its code using any family members of .NET languages.

Hypertext Transfer Protocol (HTTP) is a robust tool that creates a strong but also simple communication mechanism for mobile applications. Especially when there is large amounts of data to transfer and it cost quite much with billing systems of mobile applications, its better to deal with binary format and also more efficient to take the direct control of HTTP communication process. So in these cases we try to transfer the only core application data.

As the task here is to understand the relation of an ASP.NET custom handler and an HTTP client, thus it can be a good start to build a simple .NET compact Framework program as a download and upload file manager with HTTP.

HttpWebResponse and HttpWebRequest are two communication classes of HTTP which provide respectively send and receive requests to and from a web server.

Next task is to create the Upload Functions and to perform it from “New Project” Wizard we must create a Smart Device Microsoft Windows® application targeting the Smartphone. UploadFile is a method that accepts two string arguments. These two strings are to identify the name of the local file to read and the HTTP URL of destination server.

public void UploadFile(string localFile, string uploadUrl)

Uploads require an instance of HttpWebRequest and like all other webrequest derived classes, it needs to be created with the new static WebRequest Create method. URLs of the form http://... Create an instance of HttpWebRequest. The URL here specifies the server, virtual directory, and name of the file.
HttpWebRequest req =
(HttpWebRequest)WebRequest.Create(uploadUrl);
As soon as we are finished with creating the HttpWebRequest object it is the Method properties job to identify which HTTP operation to perform. Here the put verb is used to show, uploading a file requires that data be sent as part of the body request.

req.Method = "PUT";

Submitting the data to the server must be done after buffering the the data in the request buffer of the HttpWebRequest object.

req.AllowWriteStreamBuffering = true;

GetRequestStream method prepares the accessibility to data stream in request buffer. Request stream will be wrapped in ad StreamWriter. The following code accesses the request stream and wraps it in a StreamWriter:

Stream reqStream = req.GetRequestStream();
StreamWriter wrtr = new StreamWriter(reqStream);

Now, it is time to submit the stream to server using Req.GetResponse method.

Req.GetResponse();

GetResponse also retrieves returned data from the server.

public void UploadFile(string localFile, string uploadUrl)
{
    HttpWebRequest req =
(HttpWebRequest)WebRequest.Create(uploadUrl);
    req.Method = "PUT";
    req.AllowWriteStreamBuffering = true;

    // Retrieve request stream and wrap in StreamWriter
    Stream reqStream = req.GetRequestStream();
    StreamWriter wrtr = new StreamWriter(reqStream);

    // Open the local file
    StreamReader rdr = new StreamReader(localFile);

    // loop through the local file reading each line
Chapter 2. Mobile Device Descriptions

The StreamWriter and The StreamReader classes are perfect tools for uploading text files. And as they have the best efficiency when the smart device application needs to process the local data before sending it to the server, they are quite convenient for our case which we need to process data such as GPS, ECG, ...

```csharp
public void UploadFileBinary(string localFile, string uploadUrl)
{
    HttpWebRequest req = (HttpWebRequest)WebRequest.Create(uploadUrl);
    req.Method = "PUT";
    req.AllowWriteStreamBuffering = true;

    // Retrieve request stream
    Stream reqStream = req.GetRequestStream();

    // Open the local file
    FileStream rdr = new FileStream(localFile, FileMode.Open);

    // Allocate byte buffer to hold file contents
    byte[] inData = new byte[4096];

    // loop through the local file reading each data block
    // and writing to the request stream buffer
    int bytesRead = rdr.Read(inData, 0, inData.Length);
    while (bytesRead > 0)
    {
        wrtr.WriteLine(inLine);
        inLine = rdr.ReadLine();
    }

    rdr.Close();
    wrtr.Close();

    req.GetResponse();
}
```

string inLine = rdr.ReadLine();
while (inLine != null)
{
    wrtr.WriteLine(inLine);
    inLine = rdr.ReadLine();
}

rdr.Close();
wrtr.Close();
req.GetResponse();
reqStream.Write(inData, 0, bytesRead);
bytesRead = rdr.Read(inData, 0, inData.Length);
}

rdr.Close();
reqStream.Close();
req.GetResponse();

After writing two upload functions its time to complete the application by adding two textboxes and two buttons to the application form. One of them specifying an HTTP URL and the other one a local smart device file name. by clicking the buttons UploadFile and UploadFileBinary methods will be called respectively, see figure 2.6.

Now we can add a click event handler for each button. Each button click event handler calls the appropriate upload method.

private void uploadButton_Click (object sender, System.EventArgs e)
{
    UploadFile(localFileTextBox.Text, urlTextBox.Text);
}

private void uploadBinaryButton_Click (object sender, System.EventArgs e)
{
    UploadFileBinary(localFileTextBox.Text, urlTextBox.Text);
}

For testing process we need a virtual directory on the running server and as the application is using HTTP to upload a file, a so called “write” privilege should be allowed by the virtual directory. So it is needed to create a virtual directory with so called “write” privilege.

Now exactly after the HttpWebRequest, we must assign an instance of the NetworkCredential class to the HttpWebRequest Credentials property.

HttpWebRequest req =
(HttpWebRequest)WebRequest.Create(uploadUrl);
req.Credentials = new NetworkCredential("username", "password");
Now the respond of `HttpWebRequest` object will be automatic when its asked by the server for a log on but of course the Credentials must be set. For the last part we need to execute the upload functionality to test it so we Run the program and Enter file URL and in the second textbox we should enter the full path of the smart device file to upload.

**Note:** file must already exist on the smart device.

To check the whole program we can see if each of the “upload” or “uploadbinary” button is clicked, the file identified by the URL should appear on the server populated with the contents of the smart device file, see figure 2.6.
Chapter 3

Extended Integral Pulse Frequency Modulation (IPFM) Model

In the previous chapter, we discussed about collecting the vital and geographical data of each competitor, and monitoring the heart beat behavior of each runner. In this chapter, the unnatural heart behavior of runners is considered by HRT detection and characterization of HRT based on a new extended integral pulse frequency modulation.

The extended integral pulse frequency modulation (IPFM) model is used to account for the presence of ectopic beats and heart rate turbulence (HRT) [17]. This is a new model to detect and characterize the HRT. By using a set of Karhunen-Loève (KL) basis functions and a generalized likelihood ratio test (GLRT) statistic, the HRT can be detected and characterized.

Ectopic heartbeats are small variations in an otherwise normal heartbeat that cause an irregular heartbeat. They may occur without an obvious cause and are usually harmless [18]. Ventricular Ectopic Beat (VEB) is an extra heartbeat originating in lower chamber of the heart, and occurs before the beat triggered by the heart’s normal function [27].

HRT is the short-term fluctuation in heart rate which follows a VEB, and it is a powerful predictor of mortality and unusual behavior of heart rate [19, 20]. The HRT creates after the VEB, and it causes an acceleration in heart rate which followed by a deceleration in heart rate to return to normal heart rate. The compensation of sudden drop in blood pressure is the cause of the HRT phenomenon [21, 22, 23, 24, 25].

There are some parameters for HRT characterization which turbulence onset (TO) and turbulence slope (TS) are the most commonly parameters.
3.1 Turbulence Onset (TO) and Turbulence Slope (TS)

*Turbulence Onset (TO)* is the difference between the averages of RR intervals before and after the VEB. The negative value of TO means heart rate acceleration. *Turbulence Slope (TS)* is the steepest slope observed over 5 consecutive RR intervals in the first 15 RR intervals following the VEB [17].

3.2 HRT Detection Based on Extended IPFM Model

The *original IPFM model* transforms a continuous signal into a series of occurrence times for sinus beats. The IPFM model consists of an integrator and a comparator. The input signal into the IPFM model will be integrated at the first step, and when the integrated signal reaches the fixed reference value, the mean time interval between successive sinus beats, the integrator will be reset and a pulse or delta function as occurrence time of the first sinus beat will be appeared at output of the system.

A normal heart beat will generate electrical wave propagation from the atria to the ventricles, while a VEB will generate electrical wave propagation, in opposite direction of normal heart beat, from ventricles to atria [28]. As a result, the wave produced by normal heart beat will not arrive to the ventricles when a VEB appears. This phenomenon is considered by a switch in IPFM model. The Switch is normally close. When the VEB appears, the switch will open, and when the ventricular refractory period is over, the switch will return to normally close status.

In order to consider the HRT phenomenon, a feedback is added to original IPFM model. To incorporate the heart rate turbulence, a linear system with impulse response $h(t)$ is included in the feedback, where $h(t)$ is the turbulence related to the VEB. When the VEB appears, the linear system will be fed by an impulse, and the output of the linear system will be added to the input of the integrator.

Heart rate turbulence (HRT) is modeled as a linear combination of basis function,

$$h (t) = b^T (t) \cdot \theta ,$$

where $b(t)$ consists $r$ basis functions, and $\theta$ is a $r \times 1$ weight vector. The $b(t)$ shows dynamics of heart rate turbulence, and the $\theta$ shows turbulence shape in the VEB.
In extended IPFM model, data-dependent basis functions, in particular, the KL basis functions are considered. The KL basis functions are obtained as the most significant eigenvectors of the mean correlation matrix that results from subjects with HRT. For each subject, the sample correlation matrix $\mathcal{R}_x$ is obtained by

$$
\mathcal{R}_x = \frac{1}{N_e} \sum_{i=1}^{N_e} x_i x_i^T,
$$

where $x_i$ is the input of the integrator in the extended IPFM model when the $i^{th}$ VEB appears. While correlation matrix $\mathcal{R}_x$ is found for each of the subjects, the mean correlation matrix for all subjects is determined. The $r$ most significant eigenvectors of the mean correlation matrix are selected as the discrete representation of the basis functions contained in $b(t)$, and $h(t)$ in (1) is a $N \times 1$ vector which is contained by discrete representation of the heart rate turbulence associated with the VEB.

Detection and characterization of heart rate turbulence is based on the extended IPFM model and Neyman-Pearson theorem [29]. For HRT detection, two hypotheses are considered, HRT is absent (hypothesis $H_0$) or HRT is present (hypothesis $H_1$). The Neyman-Pearson theorem maximizes the probability of detection for a given probability of false alarm $P_{FA} = \alpha$. Heart rate turbulence will be detected if

$$
L(x) = \frac{p(x; H_1)}{p(x; H_0)} > \gamma,
$$

where $p(x; H_1)$ is the probability density function (PDF) of $x$ under $H_1$ [29].
Chapter 4

Conclusions

As the final discussion for this thesis, it is essential to point out some important concepts that we faced through this project. Purpose of this thesis was to simulate a specific competition scenario and observe the effect of divers side effects, from geographic and health point of view, on the competitors. Our result together with further research in health monitoring systems such as Alive ECG could be used in a real situation and among several competitors to investigate each competitor’s geographic position and health situation.

As we investigate in this thesis, by attaching wireless sensors to each competitor, collecting the vital and geographical data, and sending it via the mobile device over the internet to the server, we are able to monitor the heart beat behavior of each runner in this competition. Furthermore, by HRT detection and characterization of HRT, it is possible to predict unnatural heart behavior of runners.
Chapter 4. Conclusions
References


References


References


Appendix A

C# Code

A.1 Phone Number and EMEI Number in C#

```csharp
public class DeviceInfo
{
    private string _manufacture;
    private string _model;
    private string _revision;
    private string _serialNumber; // IMEI
    private string _subscriberID; // IMSI

    public string Manufacture
    {
        get { return _manufacture; }
    }

    public string Model
    {
        get { return _model; }
    }

    public string Revision
    {
        get { return _revision; }
    }

    public string SerialNumber
    {
        get { return _serialNumber; }
    }

    public string SubscriberID
```
Appendix A. C# Code

```csharp
{
    get { return _subscriberID; }
}

public DeviceInfo()
{
    GetDeviceInfo();
}

private void GetDeviceInfo()
{
    IntPtr hLine;
    int dwNumDev;
    int num1 = 0x20000;

    LINEINITIALIZEEXPARAMS lineInitializeParams = new LINEINITIALIZEEXPARAMS();
    lineInitializeParams.dwTotalSize = (uint)Marshal.SizeOf(lineInitializeParams);
    lineInitializeParams.dwNeededSize = lineInitializeParams.dwTotalSize;
    lineInitializeParams.dwUsedSize = lineInitializeParams.dwUsedSize;
    lineInitializeParams.dwOptions = 2;
    lineInitializeParams.hEvent = IntPtr.Zero;
    lineInitializeParams.hCompletionPort = IntPtr.Zero;

    // lineInitializeEx
    int result = NativeTapi.lineInitializeEx(out hLine, IntPtr.Zero, IntPtr.Zero, null, out dwNumDev, ref num1, ref lineInitializeParams);
    if (result != 0)
    {
        LogManager.GetLogger("DeviceInfo").Error(string.Format("lineInitializeEx failed!\n\nError Code: {0}", result.ToString()));
        return;
    }

    // lineNegotiateAPIVerison
    int version;
    int dwAPIVersionLow = 0x10004;
```
int dwAPIVersionHigh = 0x20000;
LINEEXTENSIONID lineExtensionID;
result = NativeTapi.lineNegotiateAPIVersion(hLine, 0,
dwAPIVersionLow, dwAPIVersionHigh, out version, out
lineExtensionID);

if (result != 0)
{
LogManager.GetLogger("DeviceInfo").Error(string.Format("lineNegotiateAPIVersion
failed!

Error Code: {0}", result.ToString()));
return;
}

// lineOpen
IntPtr hLine2 = IntPtr.Zero;
result = NativeTapi.lineOpen(hLine, 0, out hLine2,
version, 0, IntPtr.Zero, 0x00000002, 0x00000004, IntPtr.Zero);

if (result != 0)
{
LogManager.GetLogger("DeviceInfo").Error(string.Format("lineNegotiateAPIVersion
failed!

Error Code: {0}", result.ToString()));
return;
}

// lineGetGeneralInfo
int structSize = Marshal.SizeOf(new LINEGENERALINFO());
byte[] bytes = new byte[structSize];
byte[] tmpBytes = BitConverter.GetBytes(structSize);

for (int index = 0; index < tmpBytes.Length; index++)
{
bytes[index] = tmpBytes[index];
}

// make initial query to retrieve necessary size
result = NativeTapi.lineGetGeneralInfo(hLine2, bytes);

// get the needed size
int neededSize = BitConverter.ToInt32(bytes, 4);
// resize the array
bytes = new byte[neededSize];

// write out the new allocated size to the byte stream
tmpBytes = BitConverter.GetBytes(neededSize);
for (int index = 0; index < tmpBytes.Length; index++)
{
    bytes[index] = tmpBytes[index];
}

// fetch the information with properly size buffer
result = NativeTapi.lineGetGeneralInfo(hLine2, bytes);

if (result != 0)
{
    LogManager.GetLogger("DeviceInfo").Error(Marshal.GetLastWin32Error().ToString());
    return;
}

int size;
int offset;
size = BitConverter.ToInt32(bytes, 12);
offset = BitConverter.ToInt32(bytes, 16);

// manufacture
if (size > 0 && offset > 0)
{
    _manufacture = Encoding.Unicode.GetString(bytes, offset, size);
    _manufacture = _manufacture.Substring(0, _manufacture.IndexOf('\0'));
}

size = BitConverter.ToInt32(bytes, 20);
offset = BitConverter.ToInt32(bytes, 24);

// model
if (size > 0 && offset > 0)
{
    _model = Encoding.Unicode.GetString(bytes, offset, size);
size);
  _model = _model.Substring(0, _model.IndexOf('\0'));
}

size = BitConverter.ToInt32(bytes, 28);
offset = BitConverter.ToInt32(bytes, 32);

// revision
if (size > 0 && offset > 0)
{
  _revision = Encoding.Unicode.GetString(bytes, offset, size);
  _revision = _revision.Substring(0, 
  _revision.IndexOf('\0'));
}

size = BitConverter.ToInt32(bytes, 36);
offset = BitConverter.ToInt32(bytes, 40);

// serial number (IMEI)
if (size > 0 && offset > 0)
{
  _serialNumber = Encoding.Unicode.GetString(bytes, offset, size);
  _serialNumber = _serialNumber.Substring(0, 
  _serialNumber.IndexOf('\0'));
}

size = BitConverter.ToInt32(bytes, 44);
offset = BitConverter.ToInt32(bytes, 48);

// subscriber id (IMSI)
if (size > 0 && offset > 0)
{
  _subscriberID = Encoding.Unicode.GetString(bytes, offset, size);
  _subscriberID = _subscriberID.Substring(0, 
  _subscriberID.IndexOf('\0'));
}

// lineClose for hLine2
NativeTapi.lineClose(hLine2);

// lineShutdown for hLine
Appendix A. C# Code

```csharp
NativeTapi.lineShutdown(hLine);
}

[StructLayout(LayoutKind.Sequential)]
private struct LINEEXTENSIONID
{
    public IntPtr dwExtensionID0;
    public IntPtr dwExtensionID1;
    public IntPtr dwExtensionID2;
    public IntPtr dwExtensionID3;
}

[StructLayout(LayoutKind.Sequential)]
private struct LINEINITIALIZEEXPARAMS
{
    public uint dwTotalSize;
    public uint dwNeededSize;
    public uint dwUsedSize;
    public uint dwOptions;
    public System.IntPtr hEvent;
    public System.IntPtr hCompletionPort;
    public uint dwCompletionKey;
}

[StructLayout(LayoutKind.Sequential)]
private struct LINEGENERALINFO
{
    public int dwTotalSize;
    public int dwNeededSize;
    public int dwUsedSize;
    public int dwManufacturerSize;
    public int dwManufacturerOffset;
    public int dwModelSize;
    public int dwModelOffset;
    public int dwRevisionSize;
    public int dwRevisionOffset;
    public int dwSerialNumberSize;
    public int dwSerialNumberOffset;
    public int dwSubscriberNumberSize;
    public int dwSubscriberNumberOffset;
}

private class NativeTapi
{
```
A.2 Mapping Serial Port and Com Ports (GPS Device Bluetooth)

gpsport = New System.IO.Ports.SerialPort("COM" + i.ToString)
gpsport.Open()
Appendix A. C# Code

gpsport.ReadTimeout = 1000 * 60 * 2
Catch ex As Exception
TextScan.Text += "COM" + i.ToString() + " port unable to open." + ControlChars.NewLine ' RaiseEvent gpsdatared(gpsdata)
Continue For
End Try
Thread.Sleep(10)
Try
gpsdata = gpsport.ReadLine
gpsport.Close()
Catch ex As Exception
TextScan.Text += "COM" + i.ToString() + " port not working." + ControlChars.NewLine ' RaiseEvent gpsdatared(gpsdata)
If gpsport.IsOpen Then gpsport.Close()
Continue For
End Try
'If gpsdata Is Nothing Then
' TextScan.Text = "COM" + i.ToString() + " port not working."
' RaiseEvent gpsdatared(gpsdata)
' Continue For
'End If
If gpsdata.Length > 0 Then
TextScan.Text += "COM" + i.ToString() + " port Working." + ControlChars.NewLine ' RaiseEvent gpsdatared(gpsdata)
ComboBox1.Items.Add("COM" + i.ToString())
End If
Next