Trigraph latency as a method to infer fatigue during typing

Bhagirathi Nagaraju

University of New Hampshire, Durham

Follow this and additional works at: https://scholars.unh.edu/thesis

Recommended Citation

This Thesis is brought to you for free and open access by the Student Scholarship at University of New Hampshire Scholars' Repository. It has been accepted for inclusion in Master's Theses and Capstones by an authorized administrator of University of New Hampshire Scholars' Repository. For more information, please contact nicole.hentz@unh.edu.
Trigraph latency as a method to infer fatigue during typing

Abstract
Due to the use of computers for long durations and prolonged typing on keyboards many computer users are suffering from Repetitive Strain Injuries. In this thesis keystroke dynamics, specifically the changes in the trigraph latencies of some words are analyzed to infer fatigue during typing. These dynamics are collected using a software application and are represented as bar-graphs for data analysis. The technology developed is completely seamless and does not require any hardware modifications. The investigation here is one aspect of the integrated effort to infer fatigue from typing dynamics.

The thesis is divided into three phases. In the first phase, a software application is developed to collect the keystroke dynamics under consideration. In the second phase the collected data is displayed as bar-charts for analysis purposes. In the final phase to validate the software, data from three volunteers, who typed on the keyboard for four hours is used to predict their onset fatigue.*.

Keywords
Engineering, Electronics and Electrical
TRIGRAPH LATENCY AS A METHOD TO INFERENCE FATIGUE DURING TYPING

BY

BHAGIRATHI NAGARAJU

B.E, University College of Engineering, Osmania University, India (2001)

THESIS

Submitted to the University of New Hampshire
in Partial Fulfillment of
the Requirements for the Degree of
Master of Science
in
Electrical and Computer Engineering
September, 2006
This thesis has been examined and approved.

Thesis Director, Dr. John R. LaCourse, Professor
of Electrical and Computer Engineering

Dr. Kondagunta U. Sivaprasad, Professor
of Electrical and Computer Engineering

Dr. L. Gordon Kraft, III, Professor
of Electrical and Computer Engineering

7/05/06

Date
DEDICATION

This thesis is dedicated to my family.
ACKNOWLEDGEMENTS

First and foremost, I would like to express my sincere thanks to Dr. John R. LaCourse, for his valuable advices, support and guidance throughout this thesis.

I would like to express my sincere gratitude to Dr. Kondagunta U. Sivaprasad for his unwavering support throughout my course work and for serving on my thesis committee. To Dr. L. Gordon Kraft III, I extend my thanks for serving on my thesis committee.

I would like to thank all my friends, Sridev, Anupama, Leon, Smita, Gopal, Chanchal Shilpa and others for their help and encouragement during this period.

I would like to thank Mike, Bill, Dominik and Stanley from Complex Systems Research Center at UNH for their continuous support and suggestions during my thesis work.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEDICATION</td>
<td>iii</td>
</tr>
<tr>
<td>ACKNOWLEDGEMENTS</td>
<td>iv</td>
</tr>
<tr>
<td>LIST OF FIGURES</td>
<td>vii</td>
</tr>
<tr>
<td>ABSTRACT</td>
<td>ix</td>
</tr>
</tbody>
</table>

## CHAPTER 1

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>INTRODUCTION</td>
<td>1</td>
</tr>
<tr>
<td>1.1 BACKGROUND</td>
<td>1</td>
</tr>
<tr>
<td>1.2 REPETITIVE STRAIN INJURIES</td>
<td>3</td>
</tr>
<tr>
<td>1.3 MOTIVATION AND PURPOSE OF THE THESIS</td>
<td>5</td>
</tr>
<tr>
<td>1.4 SCOPE OF THE THESIS</td>
<td>6</td>
</tr>
<tr>
<td>1.5 THESIS ORGANIZATION</td>
<td>6</td>
</tr>
</tbody>
</table>

## CHAPTER 2

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>SOFTWARE DEVELOPMENT</td>
<td>7</td>
</tr>
<tr>
<td>2.1 MICROSOFT .NET FRAMEWORK</td>
<td>8</td>
</tr>
<tr>
<td>2.2 RECORDER APPLICATION</td>
<td>9</td>
</tr>
<tr>
<td>2.2.1 TECHNICAL DETAILS OF RECORDER APPLICATION</td>
<td>11</td>
</tr>
<tr>
<td>2.3 TRIGRAPH LATENCY ANALYZER APPLICATION</td>
<td>16</td>
</tr>
<tr>
<td>2.3.1 TECHNICAL DETAILS OF THE TRIGRAPH LATENCY ANALYZER</td>
<td>17</td>
</tr>
</tbody>
</table>

## CHAPTER 3

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>HUMAN INTERFACE</td>
<td>19</td>
</tr>
<tr>
<td>3.1 RECORDER APPLICATION GRAPHICAL USER INTERFACE</td>
<td>21</td>
</tr>
<tr>
<td>3.2 TRIGRAPH LATENCY ANALYZER</td>
<td>23</td>
</tr>
</tbody>
</table>

## CHAPTER 4

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>RESULTS</td>
<td>32</td>
</tr>
<tr>
<td>4.1 DATA ANALYSIS – RESULTS</td>
<td>33</td>
</tr>
<tr>
<td>4.2 VOLUNTEER1- RESULTS</td>
<td>34</td>
</tr>
<tr>
<td>4.3 VOLUNTEER2 – RESULTS</td>
<td>39</td>
</tr>
</tbody>
</table>
# LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Figure 1: Recorder Application Initial Window</td>
<td>22</td>
<td></td>
</tr>
<tr>
<td>Figure 2: Recorder Application Initial Window</td>
<td>22</td>
<td></td>
</tr>
<tr>
<td>Figure 3: Trigraph Latency of COULD for USER1</td>
<td>26</td>
<td></td>
</tr>
<tr>
<td>Figure 4: Trigraph Latency of COULD for USER1</td>
<td>27</td>
<td></td>
</tr>
<tr>
<td>Figure 5: Trigraph Latency of COULD for USER1</td>
<td>27</td>
<td></td>
</tr>
<tr>
<td>Figure 6: Latency of COULD for USER1</td>
<td>28</td>
<td></td>
</tr>
<tr>
<td>Figure 7: Trigraph Latency of COULD for USER1</td>
<td>29</td>
<td></td>
</tr>
<tr>
<td>Figure 8: Average Trigraph Latency of COULD for all users</td>
<td>30</td>
<td></td>
</tr>
<tr>
<td>Figure 9: Total Average Trigraph Latency of all users for COULD</td>
<td>31</td>
<td></td>
</tr>
<tr>
<td>Figure 10: Trigraph Latency of WOULD for USER1</td>
<td>35</td>
<td></td>
</tr>
<tr>
<td>Figure 11: Trigraph Latency of WOULD for USER1</td>
<td>36</td>
<td></td>
</tr>
<tr>
<td>Figure 12: Trigraph Latency of WOULD for USER1</td>
<td>36</td>
<td></td>
</tr>
<tr>
<td>Figure 13: Trigraph Latency of WOULD for USER1</td>
<td>37</td>
<td></td>
</tr>
<tr>
<td>Figure 14: Average Trigraph Latency of WOULD for USER1</td>
<td>37</td>
<td></td>
</tr>
<tr>
<td>Figure 15: Latency of COULD for USER1</td>
<td>38</td>
<td></td>
</tr>
<tr>
<td>Figure 16: Trigraph Latency of WOULD for USER2</td>
<td>40</td>
<td></td>
</tr>
<tr>
<td>Figure 17: Trigraph Latency of WOULD for USER2</td>
<td>41</td>
<td></td>
</tr>
<tr>
<td>Figure 18: Trigraph Latency of WOULD for USER2</td>
<td>41</td>
<td></td>
</tr>
<tr>
<td>Figure 19: Trigraph Latency of WOULD for USER2</td>
<td>42</td>
<td></td>
</tr>
<tr>
<td>Figure 20: Average Trigraph Latency of WOULD for USER2</td>
<td>42</td>
<td></td>
</tr>
</tbody>
</table>
ABSTRACT

TRIGRAPH LATENCY AS A METHOD TO INFER FATIGUE DURING TYPING

by

Bhagirathi Nagaraju

University of New Hampshire, September, 2006

Due to the use of computers for long durations and prolonged typing on keyboards many computer users are suffering from Repetitive Strain Injuries. In this thesis keystroke dynamics, specifically the changes in the trigraph latencies of some words are analyzed to infer fatigue during typing. These dynamics are collected using a software application and are represented as bar-graphs for data analysis. The technology developed is completely seamless and does not require any hardware modifications. The investigation here is one aspect of the integrated effort to infer fatigue from typing dynamics.

The thesis is divided into three phases. In the first phase, a software application is developed to collect the keystroke dynamics under consideration. In the second phase the collected data is displayed as bar-charts for analysis purposes. In the final phase to validate the software, data from three volunteers, who typed on the keyboard for four hours is used to predict their onset fatigue.
CHAPTER 1

INTRODUCTION

Computer systems are indispensable in almost all technical, business and industrial applications. The dependence of people on computers has increased tremendously in recent years. With the increase in computer usage there is prolonged use of the keyboard such as in high-volume data entry. Typing on keyboard for long time may cause Repetitive Strain Injuries (RSI) or other muscular skeletal disorders. In this thesis an analysis of keystroke dynamics is made to make user aware of the onset fatigue during typing.

1.1 BACKGROUND

Keystroke dynamics refers to the detailed timing information that describes exactly when each key is depressed and released as a person uses a computer keyboard [1]. The term describes an individual’s typing pattern, including latencies, key depress durations and keystroke pressure [2]. Each individual has a fairly unique typing pattern. Hence, this information can be used in the identification and authentication process of a computer user and thus provides an extra level of security.
Some of the other identifying biometrics used for systems which requires user identification includes hand geometry, thermal patterns in the face, blood vessel patterns in the retina and hand, finger and voice prints, and handwritten signatures. Keystroke dynamics is a relatively new method of biometric identification and provides a comparatively inexpensive and unobtrusive method of hardening the normal login and password process. It does not require additional hardware as it uses the keyboard to measure keystroke dynamics.

These keystroke authentication systems use latency as the key parameter. Gaines[3] used keystroke latency timings for user authentication. The observations were based on the digraphs (keystroke latency timings of key pairs) that occurred more than 10 times were analyzed. A test of statistical independence was carried out using the T-Test under the assumptions that the means of the digraph times at both sessions were the same, and with the assumption that the two variances were equivalent

Leggett, Um press and Williams [4] in their experiments considered latency valid if it fell within 0.5 standard deviations of the mean reference digraph latency, and the user was identified if more than 60 percent of the comparison between the test signature and the mean reference latencies were valid.

In the experiments conducted by Young and Hammon [2] the covariance matrix of the vectors of reference latencies was used as a measure of the consistency of the individual's signature and the Euclidean distance between the two vectors was used to compare a
number of attributes which included keystroke pressures and time to type a predefined number of words.

In this thesis the keystroke dynamics, specifically changes in trigraph latencies of some pre-specified words typed by a user are analyzed for inferring the fatigue developed due to long term typing.

1.2 REPETITIVE STRAIN INJURIES

Repetitive Strain Injuries occur from repeated physical movements doing damage to tendons, nerves, muscles, and other soft body tissues. It has been observed that large number of people who use computers for long durations report discomfort such as dry eyes, neck and shoulder tightness, back pain and exhaustion. This damage may lead to various musculoskeletal disorders of the upper limbs.

The increase in computer use and light-touch keyboards that permit high speed typing have resulted in an epidemic of injuries of the hands, arms, and shoulders. The thousands of repeated keystrokes accumulate and tend to damage the body. It can happen even more quickly as a result of typing technique and body positions that place unnecessary stress on the tendons and nerves in the hand, wrist, arms, and even the shoulders and neck. Lack of adequate rest and breaks and using excessive force almost guarantees trouble [5].

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
Over the years, computer-related injuries have increased alarmingly and beleaguered the modern office workplace, affecting the productivity of hundreds of thousands of workers, causing pain, injury and, in some cases, disability. The main cause of these computer related injuries is due to specific faulty movements and awkward postures inherent during typing on the keyboard. This type of computer usage may lead to fatigue, inflammation of muscles or tendons, compression and entrapment of nerves, as well as pain or weakness in the upper limbs and neck. This usage results in problems like Musculoskeletal Disorder and Occupational Overuse Syndrome [6]. Constant typing can also lead to Carpal Tunnel Syndrome. Repetitive Strain Injuries have become an epidemic since computers have entered the workplace in large numbers.

There is a serious need to make a computer user aware of the fatigue developed in the process of typing or doing any other repetitive tasks on the computer. Research has been done in determining the growing fatigue from long term typing on a laptop cursor plate and make user aware of the onset fatigue. In this research, Shibli [7] developed software which collects metrics associated with the use of the laptop cursor plate. Metrics considered were Contact Force, Contact Area, Contact Time and Contact Speed. These software measured metrics can be displayed at the user's desire for information about their interface with a laptop cursor plate. The technology is seamless to the laptop operation and the user and it does not require any hardware modification. With the knowledge of their fatigue level, the user can then modify their work habits to reduce fatigue.
In other research conducted by Chandra [8], software was developed which can provide the error rate and typing speed of a user when the user types continuously in a day. The results obtained were then used to see the effect of fatigue on an individual user. The primary purpose of the software was to provide a method for the user to monitor fatigue from use of the keyboard.

1.3 MOTIVATION AND PURPOSE OF THE THESIS

One of the best ways to prevent the computer user form injuries caused by repetitive stress is to make the user aware of the problem. This can be done by monitoring the onset of fatigue being caused by a user’s long-term typing on a computer. It is the purpose of this investigation to develop a system to record the keystroke dynamics which can used to monitor the onset of fatigue. In order to gather these identified dynamics the system has to be non-invasive and seamless to the user. Keeping the above considerations in mind, the technology developed has to be a software application that can access the hardware and gather the latencies of the key strokes. This software application should run in the background, without interfering with the user’s normal activities, and therefore has to be non-invasive. The thrust of the thesis entitled, “Trigraph latency as a method to infer fatigue during typing” is to provide a low-cost method for users to determine their growing fatigue level from long term typing on a computer key board. With knowledge of their growing fatigue level, the user can then modify their work habits to reduce fatigue and reduce the risk of developing CTS and other musculoskeletal disorders.
1.4 SCOPE OF THE THESIS

This investigation was divided into three phases. In the first phase the goal was to develop software that can gather the keystroke dynamics. This application would run in the background of the computer and record the dynamics required for the analysis. These keystroke dynamics include the trigraph latencies of some most commonly used words in English. The goal of the second phase of the thesis was to develop software which can analyze the data collected from the first phase. The final phase of the project involved human testing for proof of data collection and analyzing. Human testing involved data collection and analysis for a period of four hours with three computer users.

1.5 THESIS ORGANIZATION

The second chapter the software development for data collection and data analysis is described. The Human Interface with the software developed in Chapter 2 is presented in Chapter 3. Chapter 4 of this thesis document describes the human testing on the software developed. Data is gathered and displayed in the form of bar-carts for analysis. These results are discussed in Chapter 5.
CHAPTER 2

SOFTWARE DEVELOPMENT

Software development process is discussed in detail in this chapter. The software development is divided into two phases: data collection and data analysis.

The Data Collection process deals with the development of a software application called Recorder Application which gathers the trigraph latencies (Time taken to type three consecutive letters in a word.) of some pre-specified words and stores them in a text file. These pre-specified words are taken from the most commonly used four/five lettered words in written English. The application is a normal windows application which can be run as a background application or an application with graphical user interface.

The second part of the thesis deals with the analysis of the data collected from the Recorder Application mentioned above. The Trigraph Latency Analyzer Application is developed for data analysis. This application imports the data from the text files created in the Recorder Application and formats the data to display as bar-graphs for analysis purposes.
2.1 MICROSOFT .NET FRAMEWORK

The .NET framework is a new component of the Microsoft Windows Operating Systems. This framework provides an Application Programming Interface (API) to the services and APIs of classic Windows operating systems. This platform consists of a set of languages, including C# and Visual Basic .NET, a set of developing tools, including Visual studio .NET, a Comprehensive class of library for developing web and windows applications, as well as the Common Language Runtime (CLR) to execute objects within the framework [9].

The software environment for the programs written in .NET framework manages the program’s runtime requirements. The runtime environment which does this is the Common Language Runtime (CLR). The CLR includes a virtual machine, similar in many ways to the Java Virtual Machine (JVM). Thus, the programmers need not consider the capabilities of the specific CPU that will execute the program. The CLR also provides other important services such as security guarantees, memory management, and exception handling.

In .NET, programs are not compiled into executable files but they are compiled into Microsoft Intermediate Language (MSIL) files. When the program runs, the MSIL is compiled again, using Just In Time (JIT) compiler [9] and is executed.
Recorder Application is developed using C# .NET. The primary reason behind using this language was to learn and implement C# .NET. This is a simple language with only 80 keywords and a dozen of built-in data types. This language also includes all the support for structures, component-based, object-oriented programming. This application uses multithreading and multithreading in C# which is easy to understand and implement.

2.2 RECORDER APPLICATION

Recorder Application is a Windows multithreaded application with C# .NET as the underlying language. This application collects the keystroke dynamics, which in this case are the trigraph latencies of some pre-specified words.

The definition for Trigraph Latency concerning this thesis is the latency of three consecutively typed keys. Trigraph Latency of a word is the duration of the trigraph, the time duration between the first and the third key strokes of the word. This can be explained in detail with an example.

A sequence of trigraphs and latencies (ms) for word THERE are,

- “THE” 277ms, “HER” 255ms, “ERE” 297ms
- The time taken to type “THE” is 277ms, “HER” is 255ms and that of “ERE” is 297ms.
The goal of the Recorder Application is to gather the keystroke dynamics related to the trigraph latencies of some pre-specified words. These words are taken from the list of most commonly used words in written English [10]. The list of the pre-specified words is given below.

THERE, WHERE, WHICH, THESE, WHILE, WOULD, THING, THINK, RIGHT, WHOSE, THEIR, SINCE, COULD, THOSE, THIS, THEY, THAT, WERE, WHEN, THEN, HAVE, WILL, FROM, SOME, WITH, THUS, SUCH, THAN.

It has been observed that the words mentioned above are among the top 100 words used in written English. Hence, these words were chosen for the testing purposes.

The Recorder Application collects the trigraph latencies by monitoring continuously the key press events by the user. To make the application more object driven, the program is divided into three sections. The first section would take care of the key press events. The second takes care of the graphical user interface and the third records the key press events to calculate the trigraph latencies.

Recorder Application is a Windows program that can be set to run as a Windows application with GUI or as a background application. The first program with GUI offers user control over the application. This enables the users to start or stop the application at their will. The second program runs as a windows startup service program that runs as a background process without the need of any user intervention. The objective of this
program is to run behind the scenes and collect the keystroke dynamics related to the words typed by the user.

2.2.1 TECHNICAL DETAILS OF RECORDER APPLICATION

Recorder Application runs in the background as a windows start up service or as an application with GUI. Visual Studio .NET is used to build this application with C#.NET as the underlying language. The classes used to build this application are listed below.

- Keystroke_Record.cs
- Form.cs.
- Functions.cs

All these classes are built under a same namespace called Recorder, which has the same name as that of the application.

KEYSTROKE RECORDER CLASS

Keystroke_Recorder class which is built under the Recorder namespace keeps track of the key press events. This class uses events to model user key stroke actions. To capture the global key presses, a Win32 API function namely GetAsyncKeyState is used. This function takes a numeric keyboard scan-code as a parameter. Keyboard scan-codes for standard alphanumeric keys are quite straightforward. Keys A-Z are mapped to their uppercase ASCII equivalent. Keys 0-9 excluding the numeric keypad are directly mapped
to their ASCII equivalent. Extended keys such as delete or enter are mapped to unprintable ASCII codes [11].

The investigation of the keystrokes for the specified words being typed is done by spawning a thread which can keep track of the key strokes. A function called Run() is used to spawn up a new thread that runs an infinite loop and acts as the underlying polling mechanism. The event dealt with the Key Press Event uses the KeyPressEventArgs class, which is defined and included in the body of the Recorder namespace in Keystroke_Record.cs. The functions and delegates used to achieve this functionality are defined as follows.

[DllImport("user32.dll")] Win32 API call  
public static extern int GetAsyncKeyState (long vKey)  
public delegate void KeyPressHandler(object Keystroke_Record, KeyPressEventArgs KeyPressInfo);  
public event KeyPressHandler OnKeyPress;

FORM CLASS

FORM class is defined within the same namespace, Recorder. Microsoft Visual Studio.Net by default creates form class for a windows application. The default form is changed according to the GUI specification for this application. Two button functionalities are added in this class, one to start and other to stop the application. The start button functionality initializes an object for Keystroke_Record class and spawns the thread created for the key stroke investigation.

private void Keystroke_Record_KeyPress( object sender, KeyPressEventArgs e)
The above function records all the alphanumerics and replaces them with a white space for any other characters. This class also initializes an object of Function class which will be discussed in the next section. The application can be made as a background application by setting the forms visibility to false. So, the same application can be used as both a background and foreground application.

FUNCTIONS CLASS:

Functions class is the main functional block of the application. This class is also defined under the same namespace as the other two classes. The functions defined in the Functions class calculate the trigraph latency of the pre-specified. The algorithm used for designing the Functions class is as follows.

- Collect the words which are most commonly used in written English and store them in a string as pre-specified words.
- Get the information about the keystrokes from the Keystroke_Record class.
- Get the time stamp of the key strokes along with the information about the values of the key strokes.
- Check if the keystrokes match with any of the pre-specified words.
- If there is a match, calculate the trigraph latencies of the word and store them in a text file.
Step 1: The application needs a database of words which are most commonly used in
day-to-day written English. For this a thorough research has been made and the following
words were found to be among the first 100 commonly used words in day-to-day written
English [9].

THERE, WHERE, WHICH, THESE, WHILE, WOULD, THING, THINK, RIGHT,
WHOSE, THEIR, SINCE, COULD, THOSE, THIS, THEY, THAT, WERE, WHEN,
THEN, HAVE, WILL, FROM, SOME, WITH, THUS, SUCH, THAN.

These words can be stored in a file or in a string array for computing the latencies for
later use in the program.

Step 2: Input_Key() function in the Functions class is called from the main function when
ever a key is pressed. This function fills two arrayLists namely q_Letters and q_Time
with the input key presses and their respective time stamps respectively. The arrayLists
are used because they have the property of storing any kind of objects.

Step 3: To record the time stamps, C#’s DateTime class is used. This class has many
methods. The methods used in the program are as follows:

- DateTime.Now: This gives the Date and time at the current instant. This
  method is used here to get the time of the key press.

- Object.TimeOfDay.ToString(): The object here is the date time object. This
  particular function gives only the time of the day. That is the instant when the
key was pressed. The Date class represents times with 100ns resolution; it represents times in ticks, and each tick is 100ns.

- Object.ToString() is used to get the time of the day as hour:min:sec.

- The object derived from Time class is used to calculate the time difference.

The q_Time and q_Letters arrayLists are used to store the information of the time stamps and key presses respectively.

Step 5: Both arrayLists are filled up to the specified word length (that is 5 in this case). The q_Letter array list is converted to a string and compared with the pre-specified words. If there is a match then Step 6 is performed else the first element in the arraylists is removed.

Step 6: In Step 5, if there is a match with the stored key values and the pre-specified words in word list, a file named “word”.txt is created in append mode where word here is the matched word. For example if the match was with “THERE”, then THERE.txt is created and so on.

Trigraph latencies are the time spans between every first and third keystroke, second and fourth and finally between the third and fifth keystrokes. These are stored along with the current date and time, for future analysis. Once this is done, the ArrayLists are emptied for new values of key presses.
This program and method is completely a hardware free program. The user need not connect any peripherals or wires to the computer in order to work with this program. Also, the user need not shut down other programs in order to work with this program. This program works in the background without affecting any running programs. Therefore, the program will not affect the performance of other programs that are running at the same time. This program is seamless and completely non-intrusive. Also, this software is easy and straight forward to install.

2.3 TRIGRAPH LATENCY ANALYZER APPLICATION

Trigraph Latency Analyzer Application is built using Matlab. This application imports the data from the text files created by the Recorder application and displays the data as bar graphs. The data analysis process involves the following steps.

- Display the trigraph latencies of each word,
- Display the total latency for each word
- Display the average trigraph latency for each word
- Display the average trigraph latency for all the three users
- Display the average latency of each trigraph of a word for all the three users
2.3.1 TECHNICAL DETAILS OF THE TRIGRAPH LATENCY ANALYZER

In order to represent the data collected in many forms discussed in the above section, different Matlab functions are developed. The functional blocks which are constructed for the purpose are listed below.

- Func().
- Results().
- TL_plot().
- Avg_plot().
- Calculate_avg().
- Average_plot().

**FUNC**

Func() is called at the command prompt to start the application. This function gets the data ready for processing. The data is then imported and is sent to the Calculate_avg() function for displaying the average trigraph latencies of all the words from the three users. This function also calls Results() function.
RESULTS

Results() function gets the information about the location of the files where the data is stored. This function also helps in setting the paths and the file names in which the bar graphs are to be stored. The figures displayed are stored in word files automatically for future use.

TL_PLOT

TL_Plot() function displays the latencies all the trigraphs of a word in one single graph of each user and stores these graphs in separate word documents.

AVG_I_PLOT

Avg_I_Plot() function is called from the results() function. This function is called to plot the bar-graphs for the average trigraph latencies of each word for each user.

AVG_ALL_PLOT

Avg_all_Plot() function displays the average trigraph latencies of a word from all the users. This is used to study in general how the trigraph latency of a word varies.
CHAPTER 3

HUMAN INTERFACE

In this chapter the human interaction with the software developed for inferring the fatigue developed due to long term typing is described.

The human interface involves typing on the keyboard of a computer. As described in the aforementioned chapters, the keystroke dynamics related to trigraph latencies are collected by the Recorder Application. These dynamics collected are then analyzed for inferring on-set fatigue due to prolonged typing. For the application to function optimally, the users need to be a regular typist with decent typing skills and should have consistency in typing. The Recorder Application collects the dynamics of the keystroke, regardless of what and where the typing is performed, like typing a document or draft, the users could do their day-to-day work on the computer. An e-mail advertisement was sent to all the undergraduate and graduate students at UNH to obtain the volunteers. The advertisement sought good typists who would be able to type for 4 hours continuously. Three graduate students at UNH volunteered for this testing. The human testing was performed after getting the approval (Approval # 3726) from Institutional Review Board (IRB) and abided all the rules and regulations specified by IRB during the testing phase.
The primary task before starting the testing of the application was to educate the volunteers about the use of the software and its implementation. The scope, aim and purpose of this project were offered to the volunteers.

1. The project studies the keystrokes to monitor the on-set fatigue from computer users via identified typing-dynamics.

2. This project is a software application and thus does not require any hardware implementations.

3. The users of this application need only to perform their day-to-day activities on the computer, like regular typing, writing drafts etc.

4. This program runs in the background without any obtrusion or any need for interference required on user’s part, and is completely seamless.

5. This is a very simple project and needs minimal requirements and is absolutely risk free with no physical contact.

6. The software only records the information related to the dynamics under consideration. Thus, there is no log of the user’s activities on the computer.

7. This application does not capture any passwords or confidential information from the computer.

The application is easy to install. This process takes no more than two minutes. The users were given a demo about the functionality of the application before starting the testing process.
The Recorder Application developed consists of two programs, one which needs user intervention for initiation to start the application and the other which runs as a background application without any user intervention. The users were given the choice to choose any one of these applications.

The volunteers were given an essay of 30 pages about GENDER BIAS IN THE AUTOMOBILE REPAIR INDUSTRY and were asked to type for four hours to test the software. When the users were done typing the whole essay, they again repeated the same task till the stipulated time was reached.

3.1 RECORDER APPLICATION GRAPHICAL USER INTERFACE

The Recorder Application with graphical user interface requires user intervention to start the application. A shortcut icon is created on the desktop which needs to be double clicked for launching the application. The program when started opens a Window as shown in Figure 1.
The Recorder Application is initiated by clicking the “START” button to invoke the thread which collects the keystroke dynamics. The users can start their regular typing activities on the computer after starting the Recorder Application. Once the “START” button is clicked the status of the application which was “Not Recording Keystroke Dynamics” is set to “Recording Keystroke Dynamics” as shown in the Figure 2 below.

Figure 1: Recorder Application Initial Window

Figure 2: Recorder Application Initial Window
Once the application is started and the user begins to type and if there is a match with the pre-specified words, the trigraph latency is calculated and stored in a text file. As mentioned before the pre-specified words are the most commonly used words in written English. Once the data is collected the Recorder Application can be stopped by a click on the “STOP” pushbutton as shown in Figure 2.

The keystroke dynamics for the trigraph latencies collected for the pre-specified words are stored in text files. These text files are used by the Trigraph Latency Analyzer Application to analyze the results in Matlab.

The Recorder Application required human intervention to start and stop. In order to run this application in the background, the application’s form is set to invisible mode and registered as a Windows start up service. This enables the application to start as a background application as soon as the operating system is booted up. This program does not have a graphical user interface or front-end.

3.2 TRIGRAPh LATENCY ANALYZER

The Trigraph Latency Analyzer application is a program written in Matlab to analyze the data collected from the above Recorder Application. As mentioned before, the trigraph latencies are calculated for the pre-specified words and stored in text files named accordingly. For example if the word typed is “FROM” then the trigraph latency of this word is stored in a file named “FROM.txt”. The data is stored as follows.
This file consists of the date, time and the trigraph latencies as shown above. This data is imported into the Matlab file using the “import” function and properly formatted to display as bar charts for analyzing the effect of fatigue due to long term typing.

There are 28 pre-specified words and hence there will be 28 files for each user to be analyzed. Once the Trigraph Latency Analyzer is run, it produces graphs for each file.
For clarity a few of them are explained briefly and the rest are detailed in the following chapter.

The analysis process is explained as follows.

1. Display the trigraph latencies of each word for each user.

2. Display the average trigraph latency for each user

3. Display the latency of each word for each user

4. Display the average trigraph latency of each word for all the three users.

5. Display the total average trigraph latency for all the trigraphs for all the users

TRIGRAPH LATENCIES OF EACH WORD

For this discussion the word “COULD” has been chosen because it was typed many times by the users and also it has three trigraphs, which can give a good picture of trigraph latency variations due to onset fatigue. Figure 3 below shows a bar-chart displaying the trigraph latencies of the word COULD for USER1. The y-axis represents latency in milliseconds and the x-axis represents the time instants when the word “COULD” was typed. The data was collected for a period of 4 hours and displayed as bar-charts. The details are divided into three sets for ease of viewing. Each set contains 15 readings. The graph below shows the first fifteen trigraph latencies of the word “COULD” from 3:24PM to 4:11PM. The trigraphs “COU”, “OUL”, and “ULD” are represented as blue,
green and red bar graphs. Figure 4 displays the trigrams for the next fifteen occurrences of the word from 4:32PM and 6:18PM and Figure 5 shows the remaining occurrences and that is from 6:18PM to 6:49PM.

Figure 3: Trigram Latency of COULD for USER1

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
Trigraph Latency of COULD for USER1

Figure 4: Trigraph Latency of COULD for USER1

Trigraph Latency of COULD for USER1

Figure 5: Trigraph Latency of COULD for USER1

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
LATENCY OF THE WORD

This analysis gives the total time taken to type the whole word “COULD”. Figure 6 below shows the variation of the latency with time of the word “COULD” for USER1. The X-axis gives the number of times the word “COULD” has been typed in 4 hours duration. The Y-axis is latency in milliseconds.

Figure 6: Latency of COULD for USER1

AVERAGE TRIGRAPH LATENCY FOR EACH USER

The average trigraph latency of a word for each user is calculated by averaging all the trigraphs of a word. Figure 7 shows the average trigraph latency of “COULD” for USER1 for a duration of 4 hours.
AVERAGE TRIGRAPH LATENCY FOR ALL THE USERS

The analysis was made to see how the latency varies among the three users. Average trigraph latency for all the users is calculated by taking the mean of individual trigraphs of a word for all the users. Figure 8 shows the average trigraph latency of all the users for a time period of 4 hours. This average trigraph latency was calculated by adding all the trigraph latencies from the three users and taking the mean of the result. The length of the data for all the users was set to a constant length. This constant length was taken to be the minimum length among all the users.
The total average trigraph latency for all the users is calculated by taking the average of the trigraphs from all the users for a duration of 4 hours. Figure 9 below shows the total average trigraph latency of word “COULD”.

TOTAL AVERAGE TRIGRAPH LATENCY OF A WORD FOR ALL THE USERS
Figure 9: Total Average Trigraph Latency of all users for COULD
CHAPTER 4

RESULTS

This chapter focuses on the analysis of the data gathered by the Recorder Application. Three graduate students volunteered to test this application. Hence, three sets of data were collected from them for a period of four hours. Before the testing of the application, the users were given the choice to choose between Recorder Application with GUI and Recorder Application as a background application. All the volunteers opted to have the keystroke dynamics collected in the background without any intervention from their side. The results are shown in the form of a bar-chart representation for each volunteer.

As mentioned in the previous chapter the application runs in the background and collects the keystroke dynamics related to the trigraph latencies. These latencies are calculated when the user types a word which is in the list of pre-specifies words. The data is collected for a period 4 hours. This data is analyzed in Matlab. In these 28 words only those word were analyzed which occurred more than 15 times in the testing process. This filtering was done to optimize the opportunity to discover trends associated with on-set fatigue due to prolonged typing. In the Trigraph Latency Analyzer application the filtered words data is imported and represented in the form of bar-charts for data analysis.
This chapter explains in detail how the trigraph latency and total latency of a word for each user; average trigraph latency of a word for each user; average trigraph latencies of all the users; and total average latencies of each trigraph of all users vary with time. All these are explained with bar-charts in the coming sections.

In this chapter I have chosen to explain in detail about the above mentioned latencies for the word “WOULD”. This word is chosen for clarity purposes and as a proof of concept. The analysis of other filtered words would be discussed briefly at later time in this chapter. These words are “COULD”, “FROM”, “HAVE”, “SUCH”, “THAN”, “THAT”, “THEN”, “THEY”, “WERE”, and “WILL”.

4.1 DATA ANALYSIS – RESULTS

The Trigraph Latency Analyzer Application generates the bar-charts for all the pre-specified words. Here in this section the data analysis of the word “WOULD” is explained in detail to discover trends for inferring the onset fatigue on a computer user due to prolonged typing. The results displayed are for

- Trigraph Latencies of WOULD for a period of four hours.
- Total time taken to type WOULD for a period of four hours.
- Average Trigraph Latency of WOULD for each user.
- Average Trigraph Latency of the three users for a period of four hours.
4.2 VOLUNTEER1- RESULTS

In this phase of testing the user was asked to type continuously for four hours. The data was collected whenever the user typed any word which was in the list of pre-specified words.

TRIGRAPH ANALYSIS OF THE WORD “WOULD”

Figures 10 through 13 display the bar-chart for the trigraph latencies of the word “WOULD” for USER1 from 3:18 PM to 7:15 PM on 05/06/2006. These graphs show the trigraph latencies of “WOULD” at those particular instants of time shown on the x-axis. It can be observed that there is a slight trend for increased trigraph latencies with increased typing time. At certain points there is a deviation from usual; this is because the user took minor breaks whenever he/she was tired of continuous typing and developed fatigue. After the break the value of the trigraph latency dropped and then increased gradually. Here in the graphs the latency of the trigraphs “WOU”, “OUL” and “ULD” are represented with blue, green and red respectively.

It can be noticed from the graphs below that the trigraph latency of “WOU” was always lesser than that of “OUL” or “ULD” in the word “WOULD” and the trigraph latency
values for “OUL” and “ULD” are almost same. The average values of these trigraph latencies for the period of four hours are shown in the Figure 14 below. These values calculated for “WOU”, “OUL” and “ULD” were 162.616ms, 262.731ms and 269.387ms respectively. It can be concluded from the above observation, that typing “ULD” took longer time than typing other letters in the word “WOULD”. This increase in latency may be attributed to the position of the fingers during typing.

The variation from minimum to maximum for the trigraph “WOU”, was 109.375ms to 484.375ms. For “OUL” this variation was observed to be 218.75ms to 578.125ms and for “ULD” it was from 28.75 to 375.00ms.

![Figure 10: Trigraph Latency of WOULD for USER1](image)
Figure 11: Trigraph Latency of WOULD for USER1

Figure 12: Trigraph Latency of WOULD for USER1

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
Figure 13: Trigraph Latency of WOULD for USER1

Figure 14: Average Trigraph Latency of WOULD for USER1
LATENCY OF "WOULD"

In this analysis, the latency (total time) of the whole word "WOULD" is calculated and displayed in the form of bar-charts as shown below. The results displayed in Figure 15 show that the variation in the latencies was around ±20% for the period of four hours. These results indicate that for every three to four values of latency the graph increased gradually and then decreased for some time and again followed the same pattern. This user took rest for every time he/she got tired. But still the patterns are consistent for the period of four hours. The maximum time taken to type the word "WOULD" in the period of four hours was 780ms and the minimum was 343ms. The average latency of this word was 432ms.

Figure 15: Latency of COULD for USER1
4.3 VOLUNTEER2 – RESULTS

In this testing the user was asked to type continuously for four hours. Keystroke dynamics were collected whenever the user typed any word which was in the list of pre-specified words.

TRIGRAPh ANALYSIS OF THE WORD “WOULD”

Figure 16 through 19 displays the bar-chart of trigraph latency of the three trigraphs in the word “WOULD” for USER2 from 3:18 PM to 7:15 PM on 05/06/2006. In the first hour the trigraph latency of “WOULD” increased gradually with typing time. In the next three hours the increase was in sets. It shows the effect on fatigue on the typing patterns. That is the reason why the increase in the latency is not continuous. The latency of the trigraphs “WOU”, “OUL” and “ULD” are represented with blue, green and red respectively.

It can be observed that the typing pattern of USER2 is different from USER1 for the word “WOULD”. In this case, USER2 took more time in typing “ULD” among all the three trigraphs. Whereas in the case of USER1, the trigraphs “OUL” and “ULD” had almost the same latencies throughout the experiment. Also, it can be noticed here that, the latencies for “WOU” and “OUL” are close to each other unlike USER1.

The average latency values for the trigraphs “WOU”, “OUL” and “ULD” are 165.094ms, 211.38ms and 294.22ms respectively. These average latencies are plotted in the Figure...
In the case of USER1, the fatigue development was not as prominent as USER2. USER1 has a gradual increase in the value of latencies for the whole experiment.

Figure 16: Trigraph Latency of WOULD for USER2
Figure 17: Trigraph Latency of WOULD for USER2

Figure 18: Trigraph Latency of WOULD for USER2
Figure 19: Trigraph Latency of WOULD for USER2

Figure 20: Average Trigraph Latency of WOULD for USER2
LATENCY OF "WOULD"

In this analysis, the latency (total time) of the word "WOULD" for USER2 is calculated and displayed in the form of bar-charts as shown in Figure 21 below. In this case the steady increase in the graph is due to the latency variation in the trigraph "ULD". The other trigraphs maintained a very little variation throughout the experiment. The maximum time taken to type the word "WOULD" in the period of four hours was 718.750ms and the minimum was 343.750ms. The average latency of this word was 459.3160ms.

![Figure 21: Latency of COULD for USER2](image-url)
4.4 VOLUNTEER 3 – RESULTS

In this testing also the user was asked to type continuously for four hours. Keystroke dynamics were collected whenever the user typed any word which was in the list of pre-specified words.

TRIGRAPH ANALYSIS OF THE WORD “WOULD”

Figures 22 through 24 display the bar-charts of trigraph latencies of the three trigraphs in the word “WOULD” for USER3 from 3:39PM to 6:30PM on 05/06/2006. This user complained of cramps in the fingers due to continuous typing and he/she could only type for 3 hours. This user’s graphs are interesting. In the first hour the latency increased gradually with typing time for all the three trigraphs. From the second hour, the user complained of cramps in the fingers and took rest at regular intervals of time. So, the graph after an hour shows lot of fluctuations.

It can be noticed that the typing pattern of USER3 is lot different from USER1 and USER2 for the same word “WOULD”. This user had varying latency pattern after the first hour of the experiment. During the first hour “WOU”, “OUL” and “ULD” had increasing trigraph latency patterns. Trigraph latency of “WOU” was the least among the three trigraphs and “ULD” was greatest. From the graphs below, Figure 25, it can be seen that after the first hour, the trigraphs show a different pattern. There was no specific trend in the latencies. So, it is difficult to attribute the effect of fatigue to the typing patterns.
The average latency values for the trigraphs “WOU”, "OUL" and “ULD” are 325.60ms, 402.926ms and 421.210ms respectively. These average latencies are plotted in the Figure 25. For this user typing pattern could not give much of information about the fatigue developed in the user. This user developed fatigue very soon compared to other users. This can be inferred from the irregular typing pattern of the user after the first hour.

The variation from minimum to maximum for the trigraph “WOU”, was 156.375ms to 734.375ms. For “OUL” this variation was observed to be 218.750ms to 984.125ms and for “ULD” it was from 296.75 to 1062.500ms. Thus the increase in the latency gives us a clear picture of effect of fatigue.

Figure 22: Trigraph Latency of WOULD for USER3

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
Figure 23: Trigraph Latency of WOULD for USER3

Figure 24: Trigraph Latency of WOULD for USER3
Figure 25: Average Trigraph Latency of WOULD for USER3
LATENCY OF "WOULD"

In this analysis, the latency (total time) of the word "WOULD" for USER3 is calculated and displayed in the form of bar-charts as shown below. Figure 26 clearly indicates the effect of fatigue on the user due to prolonged typing. It can be observed that during the first hour of typing the increase was gradual. But in the later hours there was a visible increase in the latency but it was not gradual. It indicates that this user was most affected by the fatigue. The maximum time taken to type the word "WOULD" in the period of four hours was 1769ms and the minimum was 453.750ms. The average latency of this word was 706.421ms. These values also show the effect of fatigue was more on USER3 than all the other users.

![Figure 26: Latency of WOULD for USER3](image-url)
4.5 AVERAGE TRIGRAPh LATENCY FOR THE THREE USERS

In this section the average trigraph latency for the word is calculated for all three users. This gives a general idea how the fatigue affected the latency due to prolonged typing among all the three users. As explained in the previous sections, the trigraph latency increases gradually with time and decreases for some time and here it follows the same pattern. The difference in the typing pattern could be associated with the development of the fatigue due to long term typing.

Figure 27: Average Trigraph Latency for WOULD
TOTAL AVERAGE TRIGRAPH LATENCY OF WOULD FOR ALL THE USERS

The average trigraph latency of the trigraphs from all the users is calculated to analyze the on-set fatigue among all the users due to prolonged typing. Figure 28 displays these values as bar-charts. It can be observed that the average latencies for the trigraphs “WOU”, “OUL” and “ULD” are 204ms, 278.37ms and 327.69ms.

![Total Average Trigraph Latency for all users for COULD for a period of 4 hours](image)

**Figure 28: Total Average Trigraph latency for all users of WOULD for a period of 4 hours**

THE AVERAGE TOTAL WORD LATENCY FOR “COULD”

Figure 29 shows the average word latency of “WOULD” for all the users. This graph shows the general trend of how the total time for typing “WOULD” would change with typing time. The results show that the maximum time taken to type “WOULD” by the users was 531.250ms. The minimum value was 218.750ms. It can be observed from this
graph how the total latency increases with typing time. The effect fatigue due to prolonged typing can be inferred from this graph.

![Graph showing average total latency of WOULD for all users.](image)

**Figure 29: Average Total Latency of WOULD for all the users**

### 4.6 AVERAGE TRIGRPAH LATENCIES OF ALL THE USERS FOR OTHER WORDS.

Figure 30 through 41 show the average total latency for all the users for other filtered words. The other words which got filtered were “COULD”, “FROM”, “HAVE”, “SUCH”, “THAN”, “THAT”, “THEN”, “THEY”, “WERE”, “WILL”. These figures display the variation of the total latencies of these words with the typing time. It can be observed that the total latencies increased with the typing time. These graphs show clearly how the on-set of user fatigue due to prolonged typing. Detailed and extensive
data for all words is located on the attached CD labeled User1 Figures, User2 Figures and User3 Figures.

Figure 30: Average Total Latency of “COULD” for all users

Figure 31: Average Latency of “FROM” for all users

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
Average Total Latency of HAVE, for all users

Figure 32: Average Latency of “HAVE” for all users

Average Total Latency of SUCH, for all users

Figure 33: Average Latency of “SUCH” for all users
Figure 34: Average Trigraph latency of “THAN” for all users

Figure 35: Average Latency of “THAT” for all users
Figure 36: Average Latency of “THEN” for all users

Figure 37: Average Latency of “THEY” for all users
Figure 38: Average Latency of “THIS” for all users

Figure 39: Average Latency of “WERE” for all users
Figure 40: Average Latency of "WILL" for all users

Figure 41: Average Latency of "WITH" for all users
CHAPTER 5

CONCLUSIONS AND FUTURE WORK

The purpose of the thesis is to provide computer users an economical method to monitor the growing fatigue due to prolonged typing. The application developed here is completely seamless to the user. This collects the keystroke dynamics related to the words being typed and are stored in text files. The keystroke dynamics considered here are the trigraph latencies. This data stored in the text files is represented in the form of bar graphs for analysis.

In the first part of the thesis a software system was developed to collect the keystroke dynamics of the user with respect to some pre-specified words. This part proved to be successful as it could collect the required metrics without any time delays and performance issues.

The second part of the thesis dealt with the development of an application in Matlab that allows the trigraph latencies to be represented in the form of bar-charts and provide a mode for estimating the fatigue developed due to long term typing. The goal was to present the acquired data in many forms to analyze thoroughly. This phase was a success as seen from the results of the user testing and validation.
The third part required data collection for testing of the software with human subjects. The results obtained from three volunteers were properly documented and presented in the previous chapter. The analysis of these results inferred fatigue developed due to long term typing. Hence, this part also proved to be successful for the limited data set collected.

The first set of analysis, the analysis of the trigraphs of individual users showed how the typing patterns of each user varied and how the latencies of the trigraphs were affected due to on-set fatigue. It can be observed from these results how the typing patterns changed once the users started developing fatigue. For example, Figure 22 through 24 shows tremendous fluctuations in the typing patterns for USER3. During the first hour the user had gradual increase in trigraph latencies with typing patterns remaining constant. But from the second hour the typing patterns showed a lot of variation which can be attributed to the development of onset fatigue. Other two users also showed an increase in the trigraph latencies with typing time but there was no visible change in the typing patterns.

The second set of analysis dealt with the representation of the total time taken by the user to type a word. This depicted the way a user was typing the whole word. With this set of results the effect of fatigue can be inferred by observing the increasing trend of latency. For example, from Figure 26 it can be observed among all the users, USER3 developed fatigue quickly as compared to the other two users. This user's graphs show a visible increase in the total time taken to type a single word after typing for an hour. The other two users latency trends were constant most of the time.
The third type of analysis dealt with the total average trigraph latencies of each user. This gives information regarding the overall typing patterns of the user. These results can be used for comparing with data collected for longer duration to set a threshold for the trigraph latencies due to onset fatigue.

The fourth type of analysis was to study the graphs of Average Trigraph Latency of all the users. These results showed the general typing patterns among all the users and the variation of the trigraph latencies among them with increase in typing time. It can be observed from Figure 27 the trigraph latencies showed an increasing trend as the typing time increased, and the typing patterns varied with the typing time. This result can be compared with individual typing patterns to study if all the users behave in the same way when affected by fatigue. It is premature to conclude whether the individual user’s typing pattern follow the average typing pattern of all users from these results because the testing was done on limited number of subjects.

The graphs of average total latency of all the users also showed an increasing trend with the typing time. This gives an estimate of how fast the fatigue developed due to prolonged typing in the computer users. For example from Figure 27 it can be observed that there is an increase in the latency as the time increased.

The final analysis was to study the graphs of total average trigraph latency of all the users. This shows the general typing patterns among the users. For example from Figure 28 it can be observed that the three users average trigraph latencies almost followed the
same average typing patterns. This can also be used to set a reference for all the users for detecting the on-set fatigue among them.

Thus, it can be concluded that the change in typing patterns and increase in the latency with typing time can be used to make the user aware of onset fatigue due to long term typing. The average trigraph latencies, the total average trigraph latency and average latency of all the users can be used for comparing with individual users to study if these parameters behave similarly among all the users. These parameters can be used to develop a generic system for all the users to identify the onset fatigue during typing. Although these results give an indication of the user getting fatigued, the testing exercise has to be conducted on a larger scale for concrete conclusions. The testing exercise should be done for longer durations on more number of users for further conclusions.

5.1 ADVANTAGES

There are several advantages of this project compared to many existing systems which are used for biomedical projects. The advantages are listed below.

- This project provides a low cost means for determining the effect of fatigue due to prolonged typing on a computer keyboard.

- The GUI of the project is user-friendly and straightforward to use. Any user without prior knowledge can use this software.
• There is no hardware involved in this project. Hence there is no maintenance cost involved for this project.

• A bio-medical project generally requires hardware equipments for testing. Even though this project comes under that category there is no hardware equipment used and thus requires no contacts from the equipment to a person. Hence this project can be considered to be completely risk free.

• This application was developed with an objective to provide an easy and portable method to know how the user is getting fatigued due to long term typing and then could formulate and implement ways to reduce this.

• The data collected from the application is presented in the from of bar-charts. This kind of presentation makes the analysis very easy. The data is stored in files, thus making it available to be viewed any time and used for comparison purposes with other data.

• This software which runs in the background does not keep any track of user’s activity on the computer. This only captures the keystroke dynamics specified by the application. Hence there is no fear of stealing the data or any encrypted data from the user’s computer.

• This software takes very less time to be installed on a computer. Also, this application can run as a background application without any user intervention. Thus becoming completely seamless and non-obtrusive.
5.2 LIMITATIONS

This application needs some modifications and enhancements.

- The testing for this application took place in a controlled environment with some initial assumptions.

- The results presented in this thesis were only of three volunteers. It would have been more effective in analyzing the results if the testing subjects were more in number.

- The volunteers of this testing were UNH graduate students and their age varied from 23-26 years. This testing exercise needs to be extended among all the age groups who use computers for long time. By this we can draw conclusions for all age groups suffering from RSI.

- The testing was done for only 4 hours. But it needs to be done for longer period for having general conclusions about the on-set fatigue in the users due to prolonged typing.

5.3 FUTURE WORK.

Besides the limitations of the application there can be certain features added to this application in future. This application now runs only on Windows operating system.
It would be nice if an application is developed which can be platform independent.

The data analysis is done in Matlab. This task is taken care of by the administrator. It would be good if this analysis have a user control. Here in the discussion of the results it has been mentioned that the typing patterns are affected due to fatigue. It would be a good idea if these typing patterns were used for identification of the person and detect the on-set fatigue with the change in the typing patterns.
BIBLIOGRAPHY


[2] Young, J.R., Hammon, R. W., Method and apparatus for verifying an individual’s 
identity, patent number 4,805,222, US Patent and trademark Office, Washington D.C., 
1989.


Keystroke characteristics.” *Human Factor in Management Information Systems*. Jane M. 


“Repetitive Strain Injury. Prevent computer user injury with biofeedback: Assessment 
BIBLIOGRAPHY


[2] Young, J.R., Hammon, R. W., Method and apparatus for verifying an individual’s 
identity, patent number 4,805,222, US Patent and trademark Office, Washington D.C., 
1989.


Keystroke characteristics.” *Human Factor in Management Information Systems*. Jane M. 


“Repetitive Strain Injury. Prevent computer user injury with biofeedback: Assessment 


APPENDICES
namespace Recorder
{
    /// <summary>
    /// Summary description for Form1.
    /// </summary>
    public class Form1 : System.Windows.Forms.Form
    {
        private System.Windows.Forms.Button on_Start;

        private System.ComponentModel.Container components = null;
        private System.Windows.Forms.Button button1;
        private System.Windows.Forms.Label label1;
        private Functions function = new Functions();

        public Form1()
        {
            // Required for Windows Form Designer support
            InitializeComponent();
        }

        /// <summary>
        /// Clean up any resources being used.
        /// </summary>
        protected override void Dispose( bool disposing )
        {
            if ( disposing )
            {
                components?.Dispose();
            }

            base.Dispose( disposing );
        }
    }
}
    if (disposing) {
        if (components != null) {
            components.Dispose();
        }
    }
    base.Dispose(disposing);

#region Windows Form Designer generated code
    /// <summary>
    /// Required method for Designer support - do not modify
    /// the contents of this method with the code editor.
    /// </summary>
    private void InitializeComponent() {
        this.on_Start = new System.Windows.Forms.Button();
        this.button1 = new System.Windows.Forms.Button();
        this.label1 = new System.Windows.Forms.Label();
        this.SuspendLayout();
        // on_Start
        //
        this.on_Start.Location = new System.Drawing.Point(40, 96);
        this.on_Start.Name = "on_Start";
        this.on_Start.Size = new System.Drawing.Size(184, 32);
        this.on_Start.TabIndex = 2;
        this.on_Start.Text = "START";
        this.on_Start.Click += new System.EventHandler(this.on_Start_Click);
        // button1
        //
        this.button1.Location = new System.Drawing.Point(240, 96);
        this.button1.Name = "button1";
        this.button1.Size = new System.Drawing.Size(184, 32);
        this.button1.TabIndex = 3;
        this.button1.Text = "STOP";
        this.button1.Click += new System.EventHandler(this.button1_Click);
        // label1
        //
        this.label1.Location = new System.Drawing.Point(64, 24);
        this.label1.Name = "labell"
        this.label1.Size = new System.Drawing.Size(336, 48);
        this.label1.TabIndex = 4;
        this.label1.Text = "Not Recording Keystroke Dynamics";
    }
#endregion
this.label1.TextAlign = System.Drawing.ContentAlignmentMiddleCenter;

this.AutoScaleBaseSize = new System.Drawing.Size(5, 13);
this.ClientSize = new System.Drawing.Size(456, 262);
this.Controls.AddRange(new System.Windows.Forms.Control[] {
    this.label1,
    this.button1,
    this.on_Start});
this.MaximizeBox = false;
this.MinimizeBox = false;
this.Name = "Form1";
this.Text = "Recorder Application";
this.Load += new System.EventHandler(this.Form1_Load);
this.ResumeLayout(false);

#endregion

/// <summary>
/// The main entry point for the application.
/// </summary>

private void Form1_Load(object sender, System.EventArgs e)
{
    Form1 ActiveForm.Visible = false;
}

private void Keystroke_recorder_KeyPress(object sender, KeyPressEventArgs e)
{
    if (e.ModifierKeys == Keys.Shift)
    {
        char a = Convert.ToChar(e.KeyCode);
        function.note_TimeStamp(a);
    }
    {
        char a = '"';
        function.note_TimeStamp(a);
    }
    else
    {
        char a = Convert.ToChar(e.KeyCode);
    }
if(a!='_')
{
    function.note_TimeStamp(a);
}

Keystroke_Recorder keystroke_recorder = new
    Keystroke_Recorder();

//Start the application
private void on_Start_Click(object sender, System.EventArgs e)
{
    keystroke_recorder.OnKeyPress += new
        Keystroke_Recorder.KeyPressHandler(
            keystroke_recorder_KeyPress);
    keystroke_recorder.Run();
    label1.Text = "Recording Keystroke Dynamics";
}

//Stop the application
private void Stop_Click(object sender, System.EventArgs e)
{
    keystroke_recorder.exit();
    Application.Exit();
}

KEYSTROKE_RECORDER.CS

using System;
using System.Runtime.InteropServices;
using System.Reflection;
using System.Threading;
using System.Windows.Forms;

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
namespace Recorder
{
/// <summary>
/// Summary description for Keystroke_Recorder.
/// </summary?
public class Keystroke_Recorder
{

private int lastX = 0;
private int lastY = 0;
[DllImport("user32.dll")]
public static extern int GetAsyncKeyState (long vkKey);

public delegate void KeyPressHandler
          (object inputListener,
           KeyPressEventArgs KeyPressInfo);
public event KeyPressHandler OnKeyPress;
Thread thdMain;

public void Run()
{
    thdMain = new Thread(new ThreadStart(RunThread));
    thdMain.Start();
}

public void exit()
{
    thdMain.Abort();
}
private void RunThread()
{
    while(true)
    {
        Thread.Sleep(10);
        int i=0;
        for(i=1;i<Byte.MaxValue;i++)
        {
            if (GetAsyncKeyState(i) ==
Int16.MinValue+1 )
            {
                KeyPressEventArgs KeyPressInfo =
new KeyPressEventArgs(
               Control.ModifierKeys,i);
                if (OnKeyPress!=null)
                {
                    OnKeyPress(this,KeyPressInfo);
                }
            }
        }
    }
}
}
}

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
public class KeyPressEventArgs : EventArgs
{
    public KeyPressEventArgs(Keys ModifierKeys, int KeyCode)
    {
        this.ModifierKeys = ModifierKeys;
        this.KeyCode = KeyCode;
    }
    public readonly Keys ModifierKeys;
    public readonly int KeyCode;
}

FUNTIONS.CS

using System;
using System.Collections;
using System.IO;
using System.Threading;

namespace Recorder
{
    /// <summary>
    /// Summary description for Functions.
    /// </summary>
    public class Functions
    {
        // The initializations for storing the letters
        private static ArrayList q_Letters = new ArrayList();
        private static ArrayList q_Time = new ArrayList();
        private static ArrayList word_List = new ArrayList();
        private static string all_Words = "THERE|WHERE|WHICH|THESE|WOULD|WHILE|THING|THINK|RIGHT|WHOSE|THEIR|SINCE|COULD|THOSE";
        private static string Words_4 = "THIS|THEY|THAT|WERE|WHEN|THEN|HAVE|WILL|FROM|SOME|WITH|THUS|SUCH|THAN";
        private static string test_Word = "";
        private static string[] str_date = new string[5];
        private static int count = 0;
        private static string path = "";
        //Constructor of the class
        public Functions()
        {
            path = @"c:\test_Words"; //Setting the directory
            // Determine whether the directory exists.
            if (!Directory.Exists(path))
            {
                // Create the directory it does not exist.
            }
        }
    }
}
Directory.CreateDirectory(path);

count = 0;

//
public void note_TimeStamp(char a)
{
    DateTime Dt = DateTime.Now;
    if(count<5)
    {
        test_Word = string.Concat(test_Word,a);
        str_date[count] = Dt.TimeOfDay.ToString();
        count++;
        return;
    }
    else
    {
        check_Word(test_Word,str_date);
        test_Word = string.Concat(test_Word,a);
        count++;
        return;
    }
}

private void check_Word(string my_Word, string[] qt)
{
    int index = all_Words.IndexOf(my_Word);
    if(index != -1)
    {
        input_2_File(my_Word,qt,0);
        str_date = null;
        str_date = new string[5];
        count = 0;
        test_Word="";
    }
    else
    {
        //check for the words which are of 4 letters

        if(words_4.IndexOf(my_Word.Substring(0,4))!=-1)
        {
            input_2_File(my_Word.Substring(0,4),qt,0);
        }
    }
}
test_word = test_word.Substring(1);
for (int m = 0; m < 4; m++)
{
    str_date[m] = str_date[m + 1];
}
str_date[4] = "";

private void input_2_File(string my_word, string[] qt, int length)
{
    DateTime dt = DateTime.Now;
    DateTime Firstletter_time = new DateTime();
    DateTime Thirdletter_time = new DateTime();
    TimeSpan Latency = new TimeSpan();

    string temp_string = string.Concat(path, "\\")
    temp_string = string.Concat(temp_string, my_word);
    temp_string = string.Concat(temp_string, ".txt");

    FileStream file = new
    FileStream(temp_string, FileMode.Append, FileAccess.Write);
    StreamWriter objWriter = new StreamWriter(file);
    bool flag = true;
    int init_count = 1;

    for (int i = 0; i < my_word.Length - 2; i++)
    {
        for (int j = i; j < i + 3; j++)
        {
            if (flag == true)
            {
                *
                objWriter.Write(dt.Date.ToString());
                objWriter.Write(",");
                flag = false;
                objWriter.Write(DateTime.Parse(qt[0 + length]).ToShortTimeString());
            }
            if (init_count == 1)
            {
            
        75

    Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
Firstletter_time = DateTime.Parse(QString::fromStdString(qt[j+length]));

}  
if(init_count==3)
{
    objWriter.WriteString("","");
    Thirdletter_time = DateTime.Parse(QString::fromStdString(qt[j+length]));
    Latency = Thirdletter_time.Subtract(Firstletter_time);
    objWriter.WriteString(Latency.TotalMilliseconds);
}

init_count++;

init_count = 1;

objWriter.WriteString(objWriter.NewLine);
objWriter.Close();

}  
}  
}  
}
APPENDIX B

TRIGRAPH LATENCY ANALYZER CODE

FUNC.M

function func()

% check if the files have same number of files or not
userFiles = [];
cd USER1\;
userFiles{1} = dir;
cd ..;/
cd USER2\;
userFiles{2} = dir;
cd ..;/
cd USER3\;
userFiles{3} = dir;
cd ..;/

% check for common files in the three directories

count = 1;
for j = 3:length(userFiles{1})
    fileName = userFiles{1}(j,1).name;
    if (substr(fileName,-4) == '.txt')
        names{count} = fileName;
        count = count + 1;
    end
end

% Functionality Start

for i = 1:length(names)
    % Processing of USER1's files
    fileLoc = sprintf('USER1\%s', names{i});
    A = importdata(fileLoc,'',');
    results(fileLoc,'USER1',names{i});
% Processing of USER2's files

fileLoc = sprintf('USER2\%s',names{i});
B = importdata(fileLoc,',');
results(fileLoc,'USER2',names{i});

% Processing of USER3's files

fileLoc = sprintf('USER3\%s',names{i});
C = importdata(fileLoc,',');
results(fileLoc,'USER3',names{i});

calculate_avg(A.data,B.data,C.data,names{i});

end

%End of the function

RESULTS.M

function results(fileLoc,filePath,fileName)

fileLoc gives the exact location of the file being tested
filePath gives the exact path were the file is located
fileName gives the file name

legend_Names=[];

if(length(fileName) == 8)
flag_Lengh = 1;
word_length = 4;
else
flag_Lengh = 0;
word_length = 5;
end

%%%  title of the graph
%%%  title_Graph : Title for the graph.
%%%  file_TL_Graph : Variable in which the name of the file for which
%%%  the graph
%%%    file_avg_Graph : Variable in which the name of the file for
%%%  average
data_TL_Graphs : Path in which above said files are being stored.
%path_IL_Graphs : Directory in which above said IL files being stored
%legend_Names : Variable in which the legend names for the files are stored
%prepare_Graphs : Prepares the template for the graphs

[tile_Graph, file_TL_Graph, path_TL_Graphs, file_avg_Graph, path_avg_Graphs, legend_Names] = prepare_Graphs(fileName, word_Length, filePath);

%End Preparation of the graphs

%Plotting Individual Trigraphs for a word

TL_plot(tile_Graph, file_TL_Graph, path_TL_Graphs, legend_Names, fileName, word_Length, filePath, fileLoc);

%Plotting Average Trigraph Latencies of the words

Avg_plot(file_avg_Graph, path_avg_Graphs, legend_Names, fileName, word_Length, filePath, fileLoc);

%End of function

TL_PLOT.M

Function
TL_plot(title_Graph, file_TL_Graph, path_TL_Graphs, legend_Names, fileName, word_Length, filePath, fileLoc)
%Plotting Trigraph Latencies of the words
%Extract Data
A = importdata(fileLoc,',');
data_File = A.data;
textdata = A.textdata;
textData_File = [];
if(length(textdata)>15)
    for  i = 1:length(textdata)
        textData_File{i} = substr(textdata(i,2),0,4);
    end
%length_Data calculates the length of data
length_Data = length(data_File);

%plot 20 readings in a graph
index = floor(length_Data/15);
remainder = rem(length_Data,15);
for i = 1:15:length_Data
    if(index>0)
        bar(data_File(i:i+15-1,:));
ylim([0 700]);
        set(gca, 'XTick', 1:15,'XTickLabel',textData_File(i:i+15-1));
set_Graphs(title_Graph,legend_Names,path_TL_Graphs,file_TL_Graph);
    index=index-1;
    else
        bar(data_File(i:i-1+remainder,:));
ylim([0 700]);
        set(gca, 'XTick', 1:remainder,'XTickLabel',textData_File(i:i-1+remainder));
set_Graphs(title_Graph,legend_Names,path_TL_Graphs,file_TL_Graph);
    end
end

%End of the function

AVG_PLOT.M

%Plot the Lteany of te word and average trigraph latency for individual
%user
function Avg_plot(file_IL_Graph,path_IL_Graphs,legend_Names,fileName,word_Length
,filePath,fileLoc)

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
A = importdata(fileLoc,',');
dataFile = A.data;
textdata = A.textdata;
textDataFile = [];

% Plot the total average latency

title_Graph = sprintf('Average Trigraph Latency of %s for
%s',substr(fileName,0,word_Len),filePath);
size_data = size(A.data);
Y = mean(A.data,1);
bar(Y,0.3);
ylim([0,700]);
set(gca,'XTick',1:length(legend_names)','XTickLabel',
legend_names);
set_Graphs1(title_Graph,path_IL_Graphs,file_IL_Graph);

% Print the values in a file

file_path = sprintf('%s/avg.doc',filePath);
FID = fopen(file_path,'a');
name = substr(fileName,0,word_Len);
fprintf(FID,'%s
',name);
for i = 1:size_data(2)
  fprintf(FID,'%s\t: %3.3f\n',substr(name,i-1,3),Y(i));
end
fprintf(FID,'\n');
fclose(FID);

% Total latency of individual word

if(size_data(2) == 3)
title_Graph = sprintf('Latency of %s for
%s',substr(fileName,0,word_Len),filePath);
x = sum(A.data,2);
x_values = x - A.data(:,2)
bar(x_values);
ylim([0,1000]);
set_Graphs1(title_Graph,path_IL_Graphs,file_IL_Graph);
end

% End of the function

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function calculate_avg(user1,user2,user3,filename)

% Calculates the average for the three users

size_user1 = size(user1);
size_user2 = size(user2);
size_user3 = size(user3);

if(size_user1(1) < size_user2(1))
    if(size_user1(1) < size_user3(1))
        lesser = size_user1;
    else
        lesser = size_user3;
    end
else
    if(size_user2(1) < size_user3(1))
        lesser = size_user2;
    else
        lesser = size_user3;
    end
end

for i=1:lesser(1)
    y(i,:)=(user1(i,:)+user2(i,:)+user3(i,:))/3;
end

% Steps to get the file name

if(length(filename)==8)
    my_title = sprintf('Average Trigraph Latency for %s
',substr(filename,0,4));
    my_file= sprintf('%s. doc',substr(filename,0,4));
    name = substr(filename,0,3);
    word{1}=substr(filename,0,3);
    word{2}=substr(filename,1,3);
end

if(length(filename)==9)
    my_title = sprintf('Average Trigraph Latency for %s
',substr(filename,0,5));
    my_file= sprintf('%s. doc ',substr(filename,0,5));
    name = substr(filename,0,4);
    word{1}=substr(filename,0,3);
    word{2}=substr(filename,1,3);
    word{3}=substr(filename,2,3);
end
%Trigraph latency

bar(y);
ylim([0 700]);

if(length(filename)==8)
    legend(word{1},word{2})
else
    legend(word{1},word{2},word{3});
end
title(my_title);
xlabel('Time period of 4 hrs');
ylabel('Latency in ms');
save2word('avg_fig.doc');

%Average trigraph latency for all the users
%---------------------------------------------------------------
x = mean(y,1); % X gives the mean value of the trigraphs of all the users
bar(x,0.3);
ylim([0 700]);
if(length(filename)==9)
    set(gca, 'XTick', 1:3, 'XTickLabel', word');
end
if(length(filename)==8)
    set(gca, 'XTick', 1:2, 'XTickLabel', word');
end
my_title = sprintf('Total Average Trigraph Latency for all users for %s for a period of 4 hours ', substr(filename,0,5));
title(my_title);
xlabel('Time period of 4 hrs');
ylabel('Latency in ms');
save2word('avg_fig.doc');

file_Path = 'all_users_avg.doc';
FID = fopen(file_Path,'a');
name = substr(filename,0,length(word)+2);
fprintf(FID,'%s
',name);
for i = 1:length(word)
    fprintf(FID,'%s	: %3.3f
', substr(name,i-1,3),x(i));
end
fprintf(FID,'\n');
close(FID);
PREPARE_GRAPHS.M

function [tile_Graph, file_TL_Graph, path_TL_Graphs, file_IL_Graph, path_IL_Graphs, legend_Names] = prepare_Graphs(fileName, wordLength, filePath)

    tile_Graph = sprintf('Trigraph Latency of %s for %s', substr(fileName, 0, wordLength), filePath);
    file_TL_Graph = sprintf('%s.doc', substr(fileName, 0, 5));
    path_TL_Graphs = sprintf('%s/TL_graphs/', filePath);
    file_IL_Graph = sprintf('%s_all.doc', substr(fileName, 0, 5));
    path_IL_Graphs = sprintf('%s/IL_graphs/', filePath);
    var = [];
    for i = 1:wordLength-2
        var{i} = substr(fileName, i-1, 3);
    end

    legend_Names = var;

SET_GRAPHS.M

function set_Graphs(title_Graph, legend_Names, path_Graphs, file_Graph)
    ylabel('latency in ms');
    title(title_Graph);
    xlabel('Time of the day PM');
    legend(legend_Names);
    %ORIENT LANDSCAPE;

    cd(path_Graphs);
    save2word(file_Graph);
    cd ../;
    % cd ../;

%End of presentation of the data
University of New Hampshire
Research Conduct and Compliance Services, Office of Sponsored Research
Service Building, 51 College Road, Durham, NH 03824-3585 * Fax: 603-862-3564

May 18, 2006

Nagaraju, Bhagirathi
Elec & Comp. Engineering, Kingsbury
475 Wallis Road
Rye, NH 03870

IRB #: 3726
Study: Trigraph Latency as a Method to Infer Fatigue during Typing
Approval Date: 5/10/2006

The Institutional Review Board for the Protection of Human Subjects in Research (IRB) has reviewed and approved the protocol for your study as Expedited as described in Title 45, Code of Federal Regulations (CFR), Part 46, Subsection 110.

Approval is granted to conduct your study as described in your protocol for one year from the approval date above. At the end of the approval period, you will be asked to submit a report with regard to the involvement of human subjects in this study. If your study is still active, you may request an extension of IRB approval.

Researchers who conduct studies involving human subjects have responsibilities as outlined in the attached document, Responsibilities of Directors of Research Studies Involving Human Subjects. (This document is also available at http://www.unh.edu/osr/compliance/irb.html.) Please read this document carefully before commencing your work involving human subjects.

If you have questions or concerns about your study or this approval, please feel free to contact me at 603-862-2003 or Julie.simpson@unh.edu. Please refer to the IRB # above in all correspondence related to this study. The IRB wishes you success with your research.

For the IRB,

Julie F. Simpson
Manager

cc: File
John LaCourse