Fall 2007

A prototype system for human-computer interaction logging, post-processing, and data visualization for the Project54 system

Edward Bourbeau
University of New Hampshire, Durham

A PROTOTYPE SYSTEM FOR HUMAN-COMPUTER INTERACTION LOGGING, POST-PROCESSING, AND DATA VISUALIZATION FOR THE PROJECT54 SYSTEM

BY

EDWARD BOURBEAU

BS, University of New Hampshire, 2005

THESIS

Submitted to the University of New Hampshire

In Partial Fulfillment of

the Requirements for the Degree of

Master of Science

in

Electrical Engineering

September, 2007

This thesis has been examined and approved.

[illegible]

Thesis Director, Andrew L. Kun
Associate Professor of Electrical Engineering

[illegible]

W. Thomas Miller, III
Professor of Electrical Engineering

[illegible]

William Lenharth
Research Associate Professor of Electrical Engineering

7/20/07

Date

ACKNOWLEDGMENTS

First of all, I thank my thesis advisor, Dr. Andrew L. Kun for his guidance, insights, patience, and encouragement throughout the course of my research.

I also thank Dr. W. Thomas Miller, III and Dr. William Lenharth for serving on my thesis committee. I thank Dr. Miller for helping me find solutions to many troubling issues throughout the course of my research.

I thank Jacob LeBlanc, Nathan Purmort, and Eric Ramsey for helping me by testing the Interaction Logger within the Project54 test vehicles. I also thank them for helping me get the Interaction Logger deployed into police cruisers.

I thank the members of the Project54 design team for helping me throughout the entirety of my research. You all kept it fun.

Last, but by no means least, I thank the police officers who so generously volunteered to provide me with the information I needed to conduct my research.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

A PROTOTYPE SYSTEM FOR HUMAN-COMPUTER INTERACTION LOGGING, POST-PROCESSING, AND DATA VISUALIZATION FOR THE PROJECT54 SYSTEM

by

Edward Bourbeau

University of New Hampshire, September, 2007

Police officers and other emergency responders have been using the Project54 system in their vehicles for many years. Over this time it is likely that certain trends have developed regarding how they use the system to make their daily tasks easier and safer. This thesis examines the use of human-computer interaction logging, post-processing and data visualization techniques to quantify and graphically present how police officers utilize the Project54 system. Specifically, data was retrieved from two deployed police cruisers that identified their use of Project54's speech user interface (SUI) and graphical user interface (GUI), as well as the vehicles' original hardware controllers. That information was then analyzed and five different sets of data visualizations were generated based on the analysis results. The visualizations were reviewed by eight members of the Project54 design team, whose feedback indicated that the visualizations were successful at relaying conclusive results from the quantitative analysis.

CHAPTER 1

INTRODUCTION

Modern police cruisers are outfitted with a wide array of equipment used to allow police officers to perform their duties. However, there are two sides to such a proliferation of in-vehicle electronic devices. While the additional hardware increases the functional capabilities of police cruisers beyond a level that has ever previously been possible, such advanced systems create a whole new set of distractions for police officers.

To aid officers in controlling their myriad in-vehicle equipment, the Project54 laboratory at the University of New Hampshire, in conjunction with the New Hampshire Department of Safety, developed the Project54 system as a solution for the in-vehicle device integration [1]. In terms of its high-level operational components, the Project54 system provides police officers with the option of using either a touch-screen graphical user interface or a push-to-talk button-driven speech user interface (SUI) on top of the original hardware controls already present in police cruisers. Figure 1.1 shows a typical Project54 system installation, with attention drawn to the methods of device control available to police officers. These methods of control include the steering-wheel-mounted push-to-talk button that enables the directional microphone to accept speech commands, the console-mounted keyboard and touch-screen monitor that provide a graphical user interface (GUI), and the original in-vehicle hardware controllers. The original controllers are also console-mounted and (from top to bottom) control the lights, radio, and radar equipment. With Project54, system integration is seamless to the officers and control is as simple as the touch of a button or the utterance of a speech command.

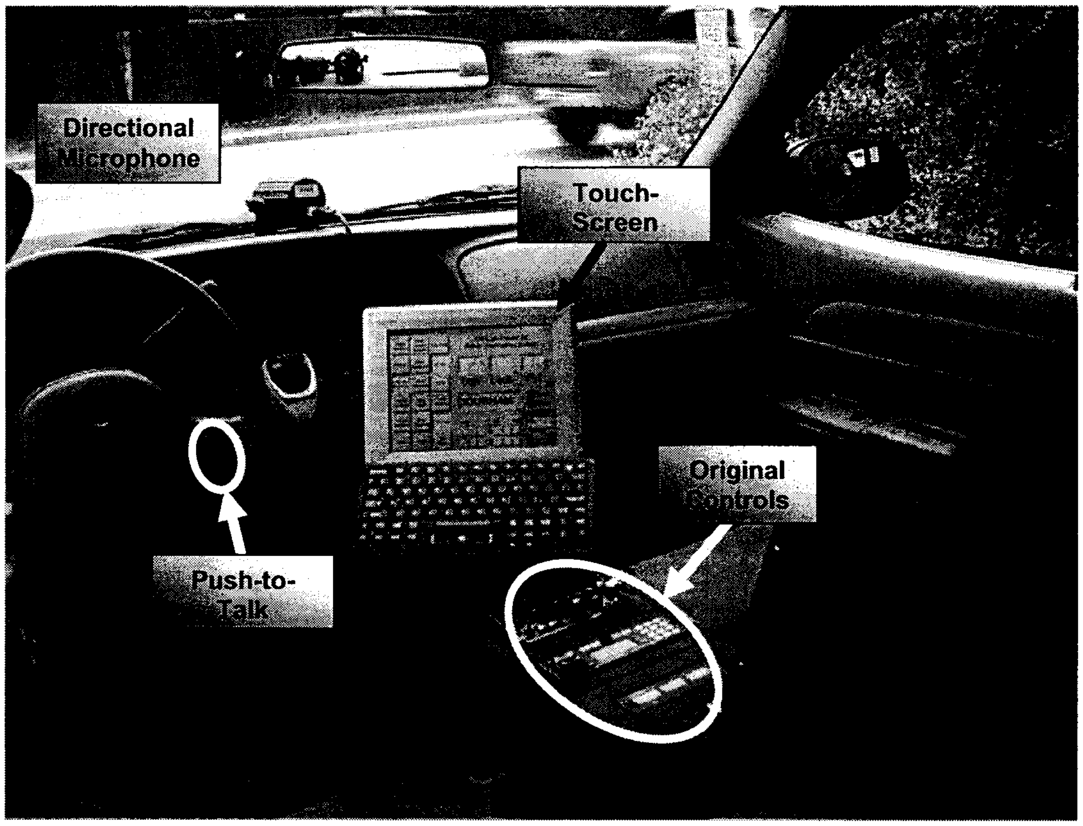


Figure 1.1 Typical Project54 system in-vehicle installation

1.1 Problems

One of the major factors contributing to the success of the Project54 software-based package is that it was designed with police officers in mind. To make sure the product suited their needs, officers were involved throughout various phases of development (e.g. planning and testing). Qualitative feedback (interviews, questionnaires, etc.) and quantitative feedback (data collection) from the officers was used throughout the design process. These insights have been an invaluable source for designers to draw upon in order to develop a product the officers would find intuitive to understand and natural to use. There is perhaps no better example of collaboration between the Project54 design team and police officers than the extensive work put into providing officers with device control via speech recognition. Since control via speech commands posed the advantage of not requiring an officer's hands to leave the steering wheel or eyes to leave the road in order to control devices, it was imperative that the speech recognition was as accurate as possible so that police officers would feel confident enough to use it as their primary means of device control. The data collected from police officers enabled designers to determine the most effective way to implement practical speech recognition.

To date, the speech recognition development process has generated results with which both developers and police officers alike can be satisfied. However, as is often the case with research and development projects, the use of police input to inform design may open the door to more possibilities for future versions of Project54 software. Since the information gathered from police officers had largely been related to the SUI, system developers knew (more or less) how police used the SUI but beyond that there was not much information available. In other words the problem was that, aside from collected speech data, there was not enough available information that provided insights into the nature of a police officer's use of the Project54 system. There was also no quantitative measure of whether or not police tended to prefer using Project54 over the more traditional device controllers.

1.2 Goals

The goals of this research were two-fold. The first goal was to develop the data analysis tools necessary for providing Project54 software developers and law enforcement officials with comprehensive information regarding police officers' Project54 system usage tendencies. Specifically, the system usage of interest was the number of interface interactions executed by the police officers, not the number of tasks they executed. The significance of this subtle difference is that tasks may be composed of multiple interactions and indicate a user's preference for a particular interface. On the other hand, interactions are important because the more interactions officers have to execute to perform their duties the more their driving performance suffers because their attention is moved from operating their vehicles to interacting with their equipment [2]. Finally, the analysis tools needed to be automated so that any analyses would be capable of being performed with little more than a mouse-click.

The second goal was to investigate the effectiveness of different analyses at conveying conclusive results to both the system designers and the law enforcement officials. The data analysis was meant to provide insight as to whether police officers tended to prefer using Project54 over traditional device controls. Beyond this, the data analysis would investigate if the control interface an officer uses had any correlation to certain contexts. For this thesis, the scope of these contexts was limited to determining if the control interface selection was dependent on the task performed, dependent on the active Project54 window, or dependent on whether or not the police cruiser is stopped.

1.3 Approach

In this research we proposed to provide the interaction-data analyses through the development of software that monitored and logged SUI, GUI, and hardware controller activities within police cruisers. Figure 1.2 shows the high-level block diagram used to pictorially describe the proposed logging and analysis processes and how they build upon the Project54 architecture. The upper portion of the figure contains those parts of this thesis project that were developed by others, during earlier work. Of significance there is that pre-existing applications are sending each other messages via the Project54 Application Manager. The portion of Figure 1.2 that lies below the horizontal dashed-line represents the proposed contributions of this project. The P54Gui block is shown overlapping into both the top and bottom portions of the diagram because, while the P54Gui component existed before this project, we proposed to update it to accommodate GUI event logging. We proposed to add an interface to the P54Gui component to provide the Interaction Logger access to records of specific GUI usage data. We also proposed to develop a usage log analyzer that would use the text files created by the Interaction Logger to develop the different visualizations alluded to in Figure 1.2. With the exception of the log analysis and visualizations (discussed in Chapter 5), the rest of the information presented in Figure 1.2 will be discussed in Chapter 3.

The research sequence was subdivided into three steps. It is important to keep in mind that these steps merely represented a logical grouping of tasks and not the actual order in which the tasks were undertaken. The first proposed step was to build the architecture for the in-vehicle data collection. This task had been simplified by building the logging capability on top of the pre-existing inter-application text messaging system [3]. The Interaction Logger was able to receive feedback messages from those applications with which it was registered. However, additional support had to be developed within the Project54 GUI software to capture specifics regarding button-press and key-stroke activities.

The second proposed step of this project was to develop a comprehensive testing phase. The data gathering software was tested extensively within a laboratory setting in order to ensure proper functionality. Once the testing satisfactorily concluded that the logging software was functioning properly, the application was deployed into actual police cruisers from the New Hampshire Department of Safety. The field testing took place for twenty-eight days, at which point the in-vehicle logging automatically ceased. The length of time was preset as an adjustable Windows Registry value (default value of twenty-eight). This was done at the officers' request. At the conclusion of the field testing period the data was retrieved from the vehicles for analysis.

The third proposed step was to create post-processing software capable of automatically parsing the raw data collected from police cruisers into different information tables. The different tables were used as the basis from which data visualizations were developed. These visualizations used multiple data dimensions as well as colors and even image overlays whenever applicable, in order to depict the results of the in-vehicle usage logging both for developers to base future applications on and for law enforcement officials to monitor how effectively they are able to carry out their duties, using Project54.

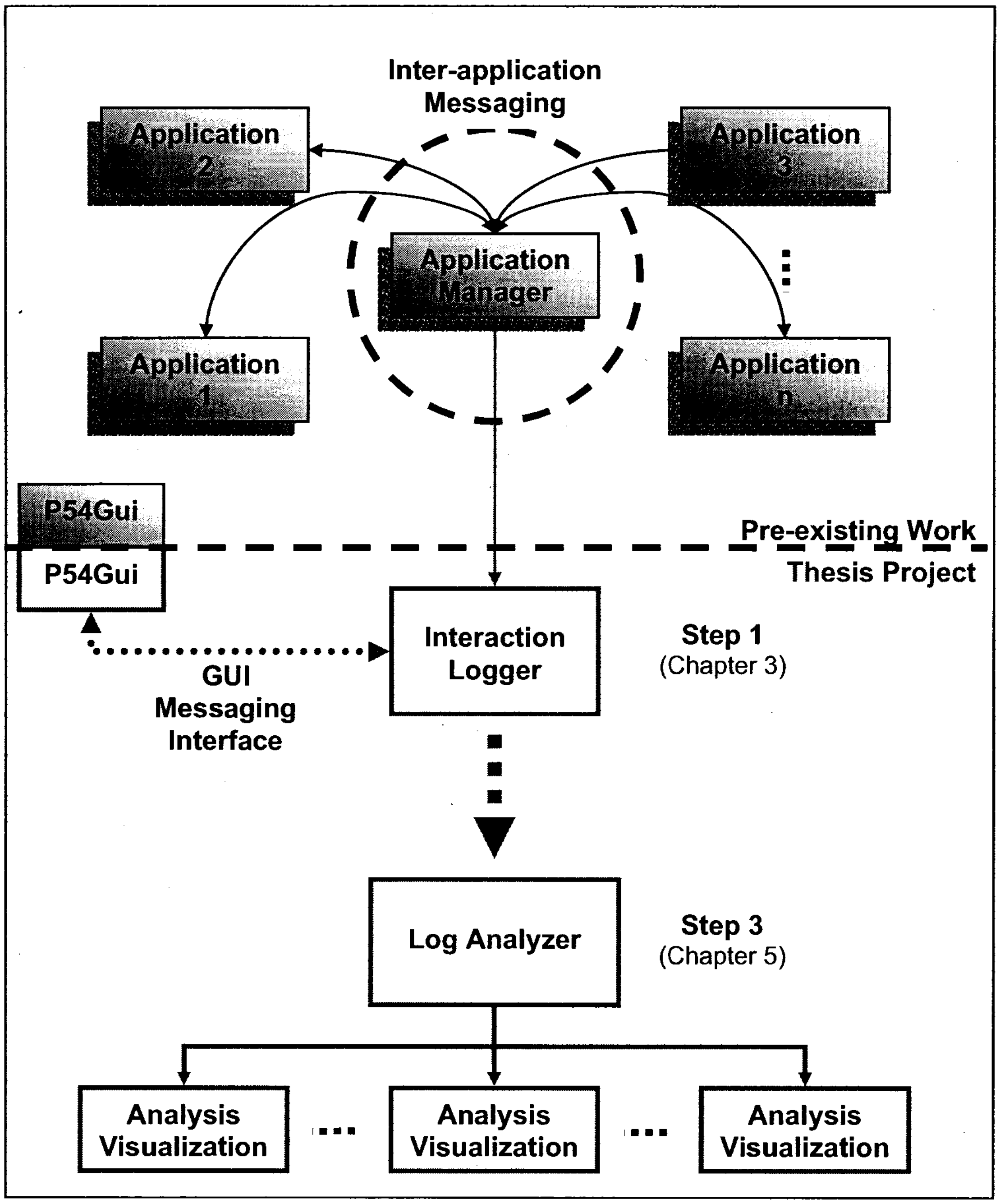


Figure 1.2 High-level block diagram of Project54 interaction logging/analysis implementation

1.4 Thesis Organization

This thesis is organized into six chapters and two appendices. The first chapter describes the motivating factors behind this research, including the problem addressed, the desired goals, and the approach used to develop the project.

Chapter two provides a brief background of existing work in the fields of data logging and log file analysis. Context-sensitive computing is also discussed as such background information will be useful when trying to provide context awareness to future versions of Project54 applications.

Chapter three details the development of the interaction logging software. Not only does this include a discussion about the logging-software design and implementation, but also the changes made to the Projec54 architecture to better facilitate detailed logging. Technical information such as registry settings and some of the more important functions used within the application are also outlined.

Chapter four contains the methods and results for the testing procedures used for the Lab Car, the driving simulator, and field-testing. Police cruiser deployment details are also provided.

Chapter five explains the data analysis undertaken for this project, including the development of the various data visualizations used to form conclusions. The post-processing includes scanning through all the original data and parsing out different portions of it in order to focus on the individual portions to form conclusions. These visualizations are the results from which conclusions will be drawn.

Chapter six summarizes and draws conclusions from the work done during the course of this thesis. Suggestions for future work with data logging and analysis are also provided.

Two appendices were included at the end of this thesis. The first appendix contains a copy of the questionnaire form administered to the police officers who volunteered for this research. The information from this questionnaire is intended to provide some context for interpreting the quantitative results received from the officers' vehicles. The second appendix contains a copy of the Institutional Review Board (IRB) approval letter that gives permission to use human test subjects for this thesis research.

CHAPTER 2

BACKGROUND

2.1 Introduction

Recording system usage characteristics is an important and useful tool in human-computer interaction (HCI) studies. Data logging is a robust, easily implemented approach to automatically gathering and subsequently analyzing information that may remain transparent to the system user [4, 5]. Hilbert and Redmiles discuss how data logging may also provide the sort of objective user-feedback information which questionnaires, interviews or other similar feedback evaluations cannot [4]. Such feedback could indicate how successfully a system gets utilized, which has major implications for future designs. The major challenges involved in evaluating HCI events are creating an efficient data collection approach and implementing informative data analysis. A balance needs to be struck between too much information and too little. Collecting too much information could slow system response down – a very unsatisfactory result for emergency responders. On the other hand, too little information could make performing an accurate analysis of user interactions impossible.

Post-processing usage information should also be as robust as the data collection process, while also being automated in order to reduce the burden placed on humans of analyzing voluminous data. Analysis results would be the most useful if they were effectively able to boil down the potentially immense amount of gathered information into clearly comprehendible HCI event representations. Research into ubiquitous computing has displayed promise for the use of HCI events beyond only demonstrating the nature of a user's interactions with a system. HCI data represents context information which can be used to guide the computer in interactions with humans.

2.2 Data Collection

Given that contemporary computers possess vast processing power, one's first instinct may be to use brute force in gathering the information while making data analysis the priority. However, even with the ability to post-process voluminous files quickly, it is still important that the data collection process gets planned intelligently so that log file sizes may be kept under control [4, 6]. In the case of retrieving data from vehicles on the road, Hilbert and Redmiles relate several motivating factors for efficiently acquiring logged information, such as the following [4]:

- In-vehicle computers may have severely downgraded performance and storage capabilities compared to, for instance, common home computers.
- Logging every possible human-computer interaction for a given program may generate otherwise-avoidable lags in that program's execution.

- Desirable usage information may become buried by less interesting information.
- It is likely that, due to limited access to the vehicles, large log files would eventually take up so much memory that overall system performance would degrade to unacceptable levels.

While logging too much data may lead to the loss of information-resolution, not logging enough data could be just as likely to generate its own problems which would also adversely affect information integrity. Such problems resulting from insufficient data collection include the sacrifice of valuable information at the expense of reduced processing time; also logs could be so sparse as to make robust data analyses virtually impossible [4].

Badre and Santos recognized that the most effective method for monitoring HCI events was to use an automated approach [7]. Their solution, the Computer-Human Interaction Monitoring Engine (CHIME) was a knowledge-based design that was capable of automatically distinguishing relevant HCI events. The system employed “smart” logging because it was created with a set of guidelines as to how the HCI events of interest could be identified.

In order to equip a design with the knowledge of what interactions are important, as CHIME did, filtering should be implemented within the data logging architecture. Information filtering may better streamline the collection process by ignoring information that is not of importance to a particular research endeavor.

Filtering would speed up the logging process, reduce the size of the log files, and, as a result, improve the performance of the subsequent data analysis.

2.3 Data Analysis

Using text files as a means to understanding the nature of particular human-computer interactions is as much an art form as it is a science. Hilbert and Redmiles discuss how, even if some level of discrimination in the data logging is employed, it may be difficult to separate the information of interest from the background [4]. This is especially true in cases involving very large amounts of collected data. Harrison et. al. developed a research tool for the express purpose of handling large amounts of data from different sources (i.e. video, audio, log records, etc.) [8]. The tool, Timelines, could capture and annotate data from HCI events. Timelines was also capable of associating that information with video and/or audio records of system usage (recorded in parallel with the data capture) in order to develop a complete picture of the user's interactions with a particular system. Once the data was annotated, it was displayed for subsequent qualitative and quantitative analyses. As its name suggests, Timelines is particularly well-suited for providing temporal data analysis. The analyses generated by the tool are, by nature, sequentially ordered blocks of information relating how a user was interacting with the system at any given time.

Usage data analyses are not only helpful for indicating how people tend to interact with a given system but also they can provide accurate records of the change in people's interactions with that system over time. Guzdial et. al.

performed a study of students in a class room setting and monitored their proficiency with a particular program over time [9]. The researchers were interested in learning if, as the students became more comfortable with the program, their use of that program would evolve in some fashion. By analyzing the students' interactions over time, the researchers were able to show that as the students' knowledge of program features increased, they were increasingly likely to use the program more efficiently.

Also of major concern is how best to display the data in a coherent and insightful manner, once the useful information has been extracted from the log files. To address this concern, researchers have developed different data visualization techniques to make various analysis abstractions palatable. For example, Guzdial, et. al. describe several different visualization techniques such as: scalar, one-dimensional, and two-dimensional analyses [10]. Scalar analyses generate a quantitative representation of the data. In other words, this approach would allow large volumes of records to be boiled down to categorized numbers. One-dimensional analyses result in chronological listings of events, while two-dimensional analyses are better suited for demonstrating how one set of data may be related to another data set. These data visualization techniques may be especially useful when put to the task of system usage analyses. According to Guzdial, usage data provides an image of which system functionality is taken advantage of by end-users [11]. Eick et. al. add that visualizations are also indispensable at making undesirable system usage traits (such as faults) clearly detectable at a glance [12]. It is often far more desirable to look at a picture of user interactions than it is to read through lines of text files to determine usage trends.

To add another level of expression, color may be added to enhance a visualization's ability to clearly present different data. Healey's research explored the important role colors play in identifying different features within visualizations [13]. The work has shown that the three most distinct colors for subjects to identify among different color groupings were red, green, and blue. However, the color palette may be expanded effectively as long as the selected colors are spaced evenly throughout the color spectrum.

While improving visualizations creatively it is important to make sure the images are flexible enough to apply to different data sets. Humphrey's research was focused not only on developing creative data visualizations but also making sure those visualizations were reusable [14]. Visualizations are, simply put, graphical representations of information which are meant to enhance an observer's ability to comprehend that information. It makes perfect sense that visualizations employ creative, so-called non-formal elements (titles, labels, backgrounds, etc.). This non-formal information enhances the presentation of the formal information (the collected data). In order to make visualizations reusable, a balance needs to be struck with regards to how much non-formal information is included. For example, too many non-formal elements may lead to static visualizations not pliable enough to handle myriad data sets.

2.4 Data-Derived Context Development

Besides painting a picture of human-computer interaction tendencies, information gathered from log files may also provide the basis for context-aware system development. In this sense, context could be explained as the reference or set of circumstances present during an HCI event. However, since this context information may be of a personal nature, it should be treated securely. Jiang and Landay drew attention to the issue of maintaining privacy in the face of the ever-evolving pervasive computing frontier [15]. Giuli, et. al. echoed the need for secure pervasive computing designs, specifically within the confines of motor vehicles [16]. Keeping private information secure must always be a priority when designing context-sensitive systems, in any environment.

Providing privacy is only one of the many challenges in creating successful context-aware applications. Implementation issues are a major concern and involve an intimate knowledge of the environment in which any context-sensitive system will be used. For instance, Lum and Lau developed their system for use in a mobile environment [17], while Voida, et. al. performed their research in an office setting [18]. Both projects were based around developing optimal time-saving strategies for information sharing over networks. However, in the mobile environment design, handheld computer limitations (cellular network bandwidth, reduced computational power, etc.) called for a solution that could use a software-based decision engine that could accurately interpret user preferences to manage computationally intensive content. In the office setting, researchers did not have to pay as much attention to data bandwidth and other handheld computer limitations. Instead the focus was on using the appropriate compliment of sensors to derive an accurate account of how workers manage their tasks. This information then had to be implemented within a system that was flexible enough to meet all the workers' requirements.

Several research projects illustrate the viability of data logging within context-sensitive computing. Since context-sensitive information was being generated for real-time applications, that data was readily available to be saved for future analyses and design iterations. The first such project, Smart Classroom Reconfigurable Context-Sensitive Middleware (RCSM) was done by Yau, et. al. [19]. The work addressed the lack of ubiquitous computing in a learning environment. The aim of this research was to develop a way for students and their teachers to spontaneously interact in a technology-intensive classroom. The approach was to modify personal digital assistants (PDAs) with sensors and other hardware in order to develop so-called "context-sensitive ad hoc communication" capable of determining the context of the interactions between different, independent groups of students and a teacher. The project used several measures from which context was derived, including the location of the PDAs, and lighting levels. The system was also capable of storing information which was then used to generate other files for classroom use, though not in real-time.

The ContextPhone project, developed by Raento, et. al., focused on the disparity between common smart phone operating system capabilities and the support for desirable phone features [20]. The designers planned, among other goals, to make their smart phone able to provide context as an informational tool and support existing phone applications. The context derived by this smart phone was based on sensor information including location and user interactions. This data was then logged, and used to drive further software design iterations. This process was especially beneficial in the early stages of development.

Ranganathan, et. al. created ConChat to address the lack of expressiveness in interpersonal electronic communications [21]. They planned to use context cues as means to enhance a chat program so that it would more closely mimic an actual face-to-face conversation. The program was able to automatically track and relay environmental characteristics between the users as well as allow the users to supply their own contextual information, such as mood and whether or how busy they are. Users were allowed to select the contexts they wished to send or receive which added another level of personalization to the program. Conversations and context cues could also be stored and analyzed for future development.

These examples echo Loke's argument that providing context sensitivity to systems should improve their usability [22]. Benefits to adding context-sensitive functionality include more efficient user interface designs and improved human-computer interactions. However, there was also the understanding that context-aware systems would be more successful if they were designed with humans in mind. In order to meet the users' needs not only was real-time context information supplied to the system but also it was stored and used to drive further design implementations. Rehman, et. al. believe that this logged context data would be the most useful if it is used to improve the communication experience between humans and machines, as opposed to being used as a system control input alone [23].

CHAPTER 3

INTERACTION LOGGER ARCHITECTURE

3.1 Introduction

As stated in the thesis introduction, the first proposed step of this project was to enable real-time user-interaction logging within police cruisers. The selected approach to accomplish this task was to design an application that would receive and record feedback messages from the other Project54 applications, when those applications were called upon by the user to perform a control operation (e.g. change a radio channel). Aside from this software used to direct the HCI event logging, other Project54 system alterations had to be put in place. Additions were made to the GUI component that would allow button presses and text field entries to be logged. A COM interface was also added in order to transmit those button and text field HCI event messages from the GUI component to the HCI event logger. The following section provides background for the Project54 messaging architecture [3] and its role in user interaction logging. Other sections within this chapter describe the details involved in the logging architecture development, including the alterations to the Project54 GUI component, linking the GUI to the HCI event logger, and the logger software design itself. The end result of this phase was to have an application capable of interfacing with the Application Manager messaging system as well as with a newly-developed P54Gui messaging system, as shown in Figure 3.1. The figure shows the inter-application communication lines that make interaction logging possible.

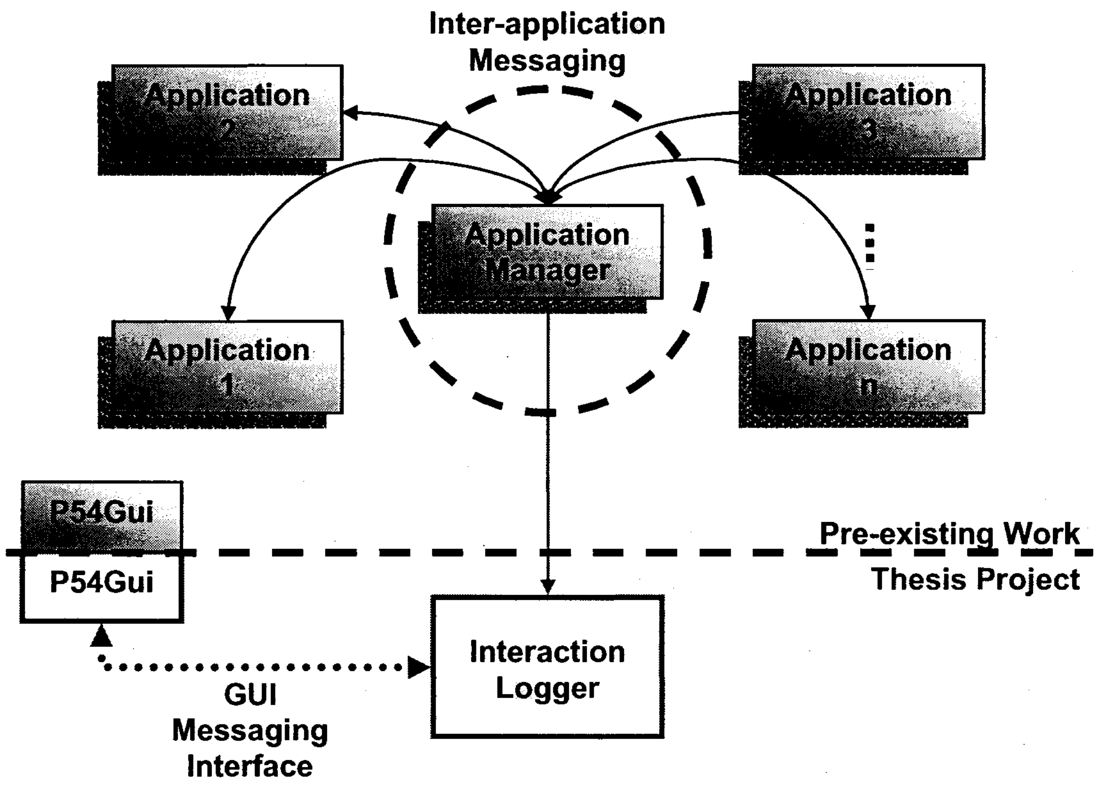


Figure 3.1 The Application Manager handles inter-application messaging between all the existing Project54 applications and the Interaction Logger

3.2 P54 Text Messaging System Overview

At its most basic level, Project54 may be described as a package comprised of several independent software control modules linked together by one central application. An example of one of the software control modules is the program written to provide speech and graphical user interfaces for a police cruiser's light bar. The central application is the Project54 Application Manager. As the connection point for the various software control modules, one of the main functions of the Application Manager is to facilitate communication between the various applications via the Application Manager's message coordinator. This inter-application communication is performed via text messaging. For the purposes of this thesis, the messages of interest are those related to the so-called status of every Project54 application, or what any application is doing at any given time.

Since the Application Manager is responsible for redirecting all inter-application messages from the source application to their proper destination, it is important to keep the message traffic to the Application Manager at a minimum. More message traffic means more processing time and greater potential for system lags. For this reason, applications only transmit status messages when that information is requested in advance by another application.

The request for status updates consists of the requesting application sending out a communication packet of the following format:

Message(*source, destination, message id tag, message text*)

The *source* and *destination* fields correspond to the names of the source application and destination applications, respectively. The *message id tag* and *message text* fields are used by the destination application during the process of handling received messages. The destination may apply a specific *message id* *tag* to certain source feedback messages that will only have meaning within the destination program. The *message text* contains the source module's status information. This status information sent between software control modules is characterized by the keyword, "STATUS", as the first word within the *message text* field of an inter-application communication packet. For example, if the Patrol Screen application wished to know the status of the radar application it would register for feedback from the radar application with the *message text* "FEEDBACK ON", using the previously described message format. The radar application would then add the Patrol Screen application to its queue of programs that are registered for status updates. Whenever the radar has a status change a message will be sent to all the applications registered for feedback, such as "STATUS FRONT ANTENNA" (in this case informing the Patrol Screen that the front antenna is on). However, if no application is registered for feedback messages, no messages will be sent to the Application Manager for disbursement.

This inter-application communication system functions well at what it was designed to do – provide updates from one program to another on a need-to-know basis. The usefulness of such messaging information can be expanded upon because applications may not only register for feedback messages from specific applications, but also may register as a sniffer and view *all* message traffic passing through the Application Manager. Among the benefits of using the message sniffer functionality are that it is automatically ensured that all available inter-application messages will be received by the Logger. Also, more information will be available beyond the standard feedback messages alone, such as the active Project54 window during any given speech or hardware user interactions.

Unfortunately there is no "STATUS GUI" message that would indicate the use of the in-vehicle touch-screen to control a device. This particular lack of status updates is because the Project54 GUI component software is not set up to provide feedback messages to the Application Manager, like other Project54 applications do. However, it is possible to add feedback functionality to the P54Gui component which, once sent to the Interaction Logger, would allow the application to monitor and record all of the interfaces an officer may use to control the various in-vehicle devices.

3.3 P54Gui Adaptations

The P54Gui is the software component that provides Project54 with its GUI functionality. The GUI attributes directly related to this project were the touch-screen buttons and the text fields (primarily used during records checks). In order to provide the Interaction Logger with information related to GUI usage, software changes had to be made to the Button Control class, the Text Field Control class, as well as to the Window Control class. These three classes contained within the P54Gui component are responsible for painting and refreshing the GUI screens with buttons, and text fields, as well as providing the functionality for those buttons and text fields. The aim of the software changes was to provide functionality that would record GUI usage characteristics and pass that collected data to the Interaction Logger.

The particular characteristics of interest relating to button-press user interactions included the following:

- the name of the active window during the button-press activity,
- whether a button was pressed down or released,
- at what time the button-press activity took place,
- the name of the button used.

To record the name of the active window during a button state change, the name of the active window had to be passed from the Window Control class to the Button Control class, since the Window Control class was the only location in which the active window name was available. The function *loadWindowLabel* was added to the Window Control class in order to make the window label name available to any other P54Gui class. In other words, the Button Control class made a call to the Window Control class's new *loadWindowLabel* function in order to gain access to the active Project54 window during a button-usage event. The Button Control class stored the results of this function call in the *m_WindowLabel* array. Functionality for identifying button state changes (pushed down or released) already existed within the P54Gui component's Button Control class. Once a button's state changed, a call to the new Button Control class function *logButtonPress* was made. This function is responsible for creating a date and time stamp corresponding to when the usage event takes place. The date and time stamps were created using the *time.h* standard library and the resulting information was stored in the *szTimestamp* array. The name of the button used during an interaction was already available within the Button Control class. This information was therefore accessible by the *logButtonPress* function. The *logButtonPress* function was able to combine the time of a button interaction, the name of that button, and the name of the active window during that interaction into one message, which was stored in *g_szGuiMessage*. This message was then passed to the Interaction Logger. The process for this message transmission is described later in this section. Table 3.1 summarizes the list of additions to the Button Control class that were used to implement button-press logging and a brief description of what each item was responsible for doing. The second item in the table refers to a Registry setting which will be discussed later in this section. The flow chart shown in Figure 3.2 represents the algorithm used by the *logButtonPress* function to create the log file entries for the GUI button usage events. This approach waits for a button state to change, captures the specified interaction information, and sends that data to the Interaction Logger.

Additions	Description
Time.h	This standard library was used to generate date and time information for the button usage messages in the mm/dd/yyyy hh:mm:ss format.
m_LogButtonPresses	The P54Gui component sets this Boolean value to true only when the P54Gui Registry setting that gives permission for GUI event logging is enabled.
*ControlWindow * Parent*	This class pointer provides the button control class with access to the loadWindowLabel() function, contained within the window control class.
m_WindowLabel	This string stores the name of the active window at the time a particular GUI button press occurs.
loadWindowLabel(m_WindowLabel)	This function is called within the Button Control class to retrieve the name of the active window from the Window Control class, when a GUI button press occurs.
b_LogData	The P54Gui component sets this global Boolean value to true only when the Interaction Logger is ready to receive GUI interaction event messages.
logButtonPress	This function places the timestamp, active window name, button name, and button activity associated with a particular GUI button event into the g_szGuiMessage array, and sends the information to the Interaction Logger.
g_szGuiMessage	This global character array stores the button-usage message to be sent to the Interaction Logger. This message contains the timestamp, active window, button name, and button state for each button-press activity.
szTimestamp	This character array is located within the *logButtonPress* function and stores the date and time at which a button event occurs.

Table 3.1 Descriptions for the additions made to the P54Gui button control class

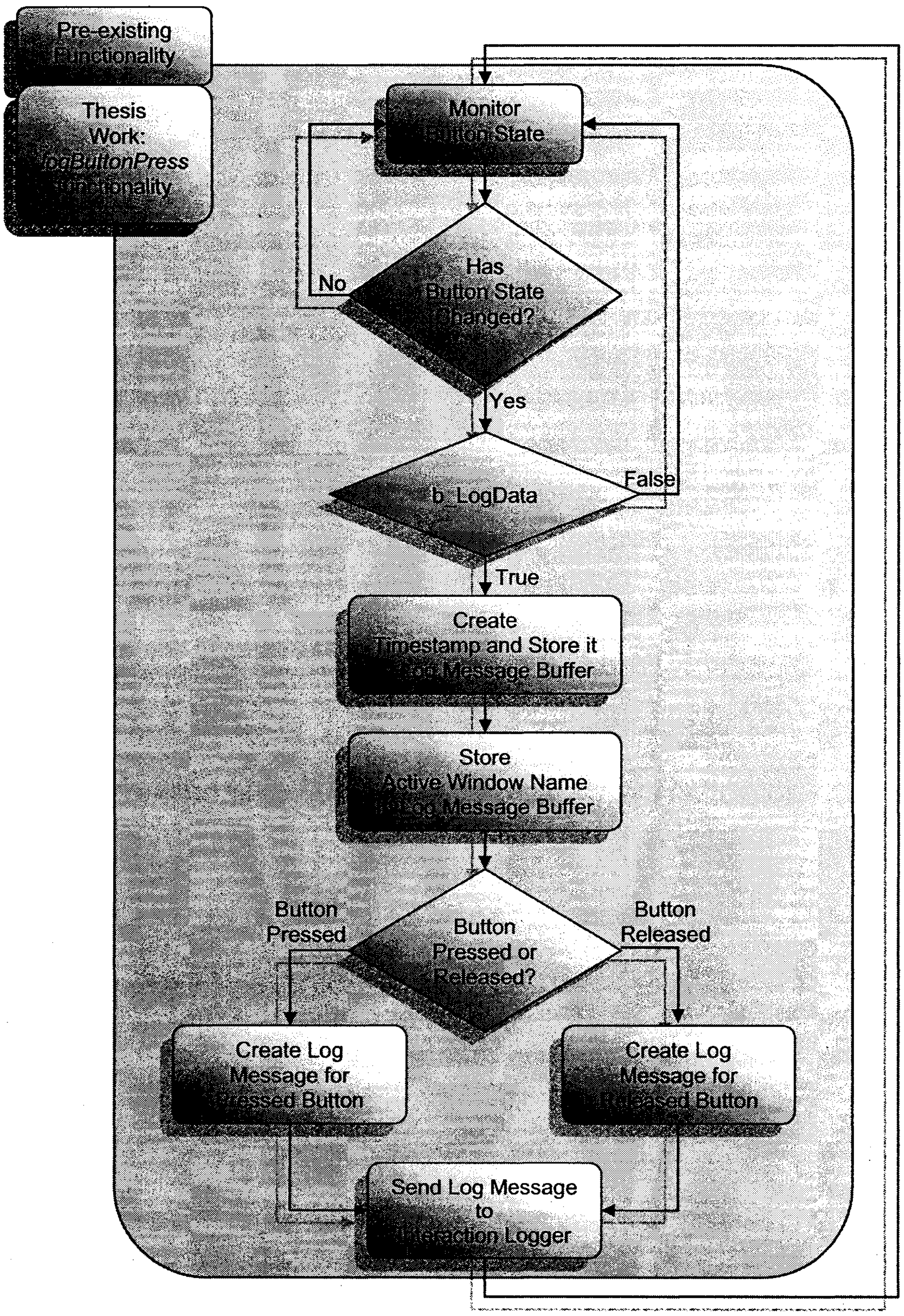

Pre-existing Functionality
Thesis Work: logButtonPress functionality
Monitor Button State
Has Button State Changed?
No
Yes
b_LogData
False
True
Create Timestamp and Store it in Log Message Buffer
Store Active Window Name in Log Message Buffer
Button Pressed or Released?
Button Pressed
Button Released
Create Log Message for Pressed Button
Create Log Message for Released Button
Send Log Message to Interaction Logger

Figure 3.2 P54Gui button usage logging algorithm

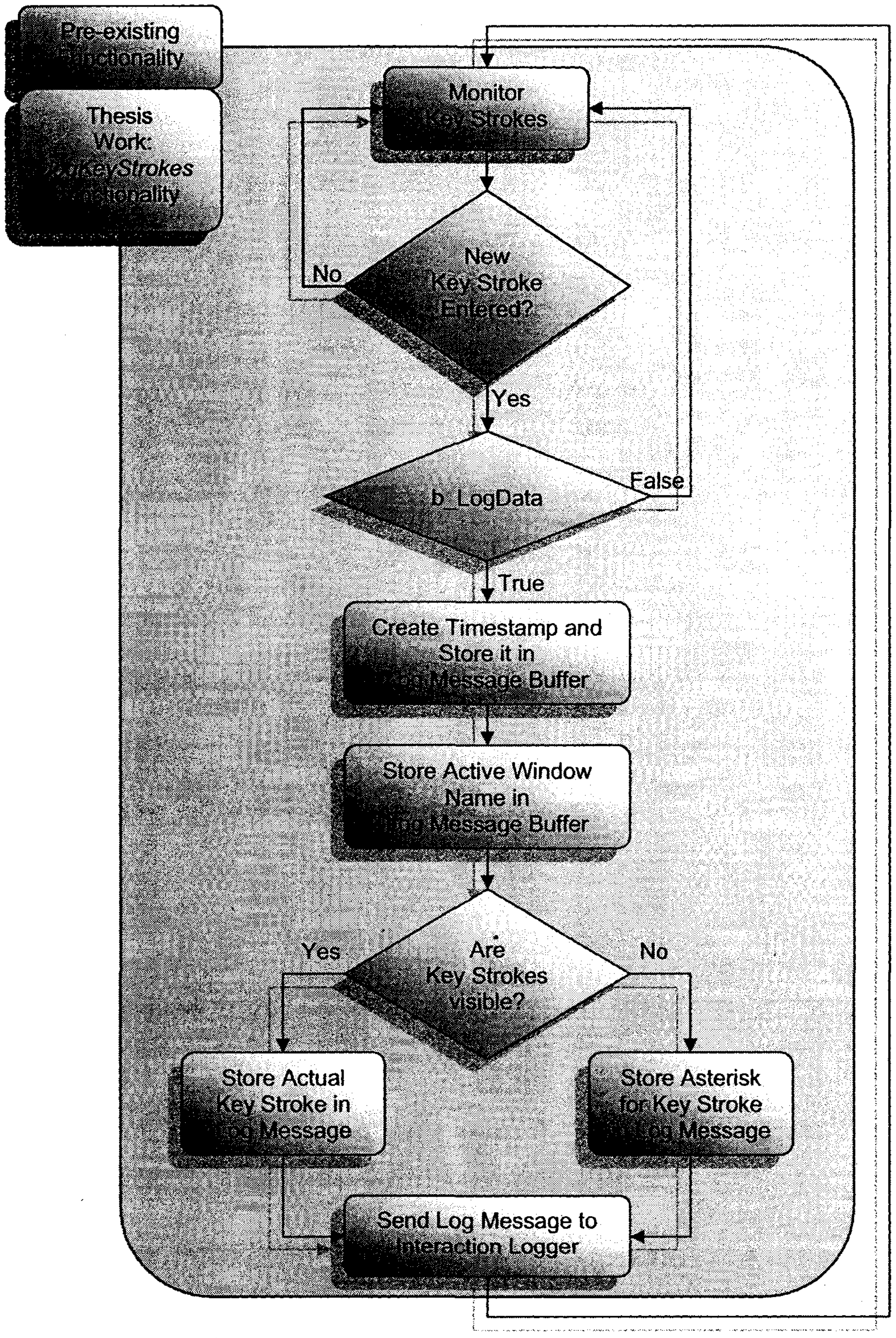
The characteristics of interest with regard to text field user interactions included:

- the active window in which the text field was located,
- the keystrokes entered into the active text field,
- the time at which the text field was used,
- the x and y coordinates of the active text field.

To record the name of the active window during a text field user event, the name of the active window had to be passed from the Window Control class to the Text Field Control class, since the Window Control class was the only location in which the active window name was available. The function *loadWindowLabel* was added to the Window Control class in order to make the window label name available to any other P54Gui class. In other words, the Text Field Control class made a call to the Window Control class's new *loadWindowLabel* function in order to gain access to the active Project54 window during a text field usage event. The Text Field Control class stored the results of this function call in the *m_WindowLabel* array. Functionality for identifying keystrokes within text fields already existed within the P54Gui component's Text Field Control class. Once a key stroke was detected, a call to the new Text Field Control class function *logKeyStrokes* was made. This function is responsible for creating a date and time stamp corresponding to when the usage event takes place. The date and time stamps were created using the *time.h* standard library and the resulting information was stored in the *szTimestamp* array. The special coordinates for the text field used during an interaction were already available within the Text Field Control class. This information was therefore accessible by the *logKeyStrokes* function. The *logKeyStrokes* function was able to combine the time of a text field interaction, the coordinates of that text field, and the name of the active window during that interaction into one message, which was stored in *g_szGuiMessage*. This message was then passed to the Interaction Logger. The process for this message transmission is described later in this section. Table 3.2 summarizes the list of additions to the Text Field Control class that were used to implement text field key stroke logging and a brief description of what each item was responsible for doing. The second and third items in the table refer to Registry settings which will be discussed later in this section. The flow chart shown in Figure 3.3 represents the algorithm used by the *logKeyStrokes* function to create the log file entries for the GUI text field usage events. This approach waits for a key stroke to be entered into a text field, captures the specified interaction information, and sends that data to the Interaction Logger.

Additions	Description
Time.h	This standard library was used to generate date and time information for the button usage messages in the mm/dd/yyyy hh:mm:ss format.
m_ShowKeyStrokes	The P54Gui component sets this Boolean value to true only when the P54Gui Registry setting that gives permission to log GUI keystrokes is enabled. Otherwise, the characters are logged as asterisks. Keystrokes entered into the Password text field are always logged as asterisks, no matter what the state of *m_ShowKeyStrokes* is.
m_LogKeyPresses	The P54Gui component sets this Boolean value to true only when the P54Gui Registry setting that gives permission for GUI event logging is enabled.
*ControlWindow * Parent*	This class pointer provides the button control class with access to the loadWindowLabel() function, contained within the window control class.
m_WindowLabel	This string stores the name of the active window at the time a particular GUI button press occurs.
loadWindowLabel(m_WindowLabel)	This function is called within the Button Control class to retrieve the name of the active window from the Window Control class, when a GUI button press occurs.
b_LogData	The P54Gui component sets this global Boolean value to true only when the Interaction Logger is ready to receive GUI interaction event messages.
logKeyStrokes	This function places the timestamp, active window name, text field coordinates, and key entered associated with a particular GUI text field usage event into the g_szGuiMessage array, and sends the information to the Interaction Logger.
g_szGuiMessage	This global character array stores the button-usage message to be sent to the Interaction Logger. This message contains the timestamp, active window, button name, and button state for each button-press activity.

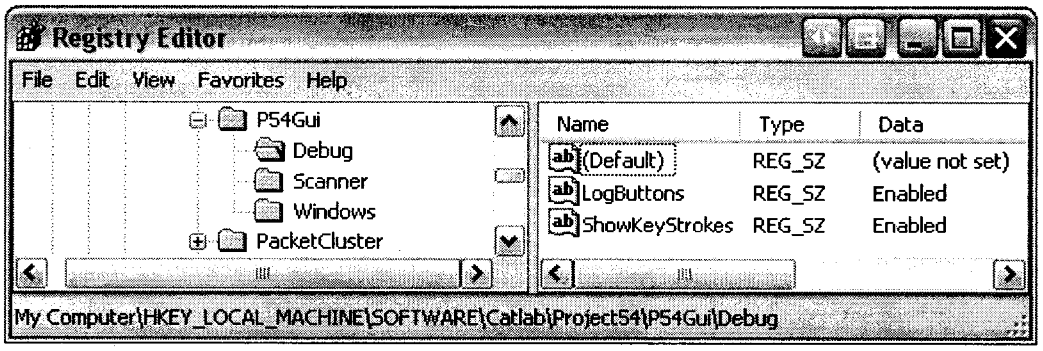
Table 3.2 Descriptions for the additions made to the P54Gui text field control class



Pre-existing Functionality
Thesis Work: LogKeyStrokes Functionality
Monitor Key Strokes
New Key Stroke Entered?
No
Yes
b_LogData
False
True
Create Timestamp and Store it in Log Message Buffer
Store Active Window Name in Log Message Buffer
Are Key Strokes visible?
Yes
No
Store Actual Key Stroke in Log Message
Store Asterisk for Key Stroke in Log Message
Send Log Message to Interaction Logger

Figure 3.3 P54Gui text field usage logging algorithm

As was mentioned previously, Windows Registry settings have been added to the P54Gui Registry folder to provide more flexibility as to when and how GUI interactions would be logged. The two Windows Registry keys were LogButtons and ShowKeyStrokes. Setting LogButtons to "Enabled" would allow GUI usage events to be logged. LogButtons is a bit of a misnomer as it not only governs when button-press events may be logged, but also when key stroke usage may be logged. The ShowKeyStrokes value is used to determine whether or not the key strokes entered into text fields will be shown as asterisks when they are logged. For instance, if a user types "hello" into a text field with ShowKeyStrokes disabled, the fact that characters were typed into the text field will be logged but, instead of displaying "hello", the log will contain the string, "*****". However, if ShowKeyStrokes is enabled, "hello" will be recorded as the string, "hello" in the log file. The Windows Registry information is presented in context in Figure 3.4. The figure shows the location within the Windows Registry of the P54Gui user interaction log values and their settings. The two values could either be set to "Enabled" or "Disabled".



Name	Type	Data
(Default)	REG_SZ	(value not set)
LogButtons	REG_SZ	Enabled
ShowKeyStrokes	REG_SZ	Enabled

My Computer\HKEY_LOCAL_MACHINE\SOFTWARE\Catlab\Project54\P54Gui\Debug

Figure 3.4 Windows Registry settings relating to the P54Gui usage logging functionality

While a solution was implemented that allowed the P54Gui component to track specific GUI usage events (button presses and keystroke entries), there was still no means of transferring that information from the P54Gui component to the Interaction Logger. The solution developed for this issue was to create two Component Object Model (COM) objects [24] that would facilitate data transmission from the P54Gui to the Interaction Logger: a logging object for the P54Gui and a GUI message handler object for the Interaction Logger. Figure 3.5 shows the added COM objects, including their interfaces and methods. The pre-existing P54Gui interfaces were not changed, but one was added – *IGuiLoggerControl*. The *IGuiLoggerControl* interface contained the two methods, *startLogging* and *stopLogging*. As their names suggest these two methods may be called by another application (in this case the Interaction Logger) to signal when GUI logging should begin and end. The Interaction Logger's message handler object receives the GUI messages once they are sent from the GUI. This process is carried out by the object's *getData* method, via the *IDataLogger* interface. Table 3.3 provides a brief summary of the interfaces and methods developed for this research.

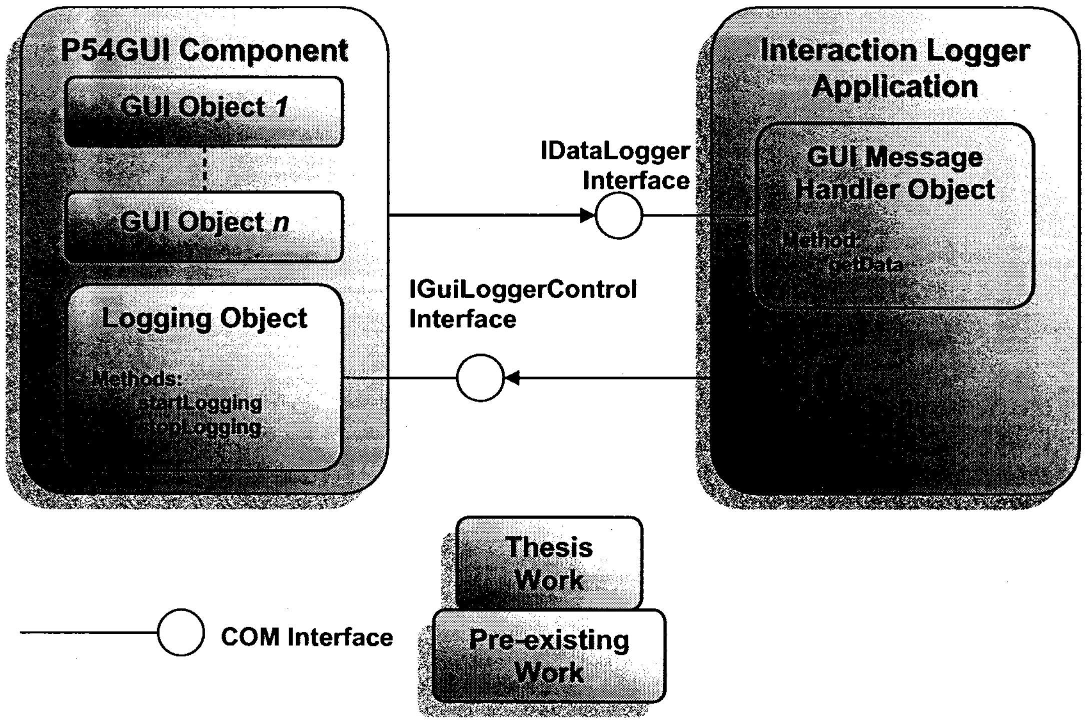


Figure 3.5 COM objects that facilitate communication between the P54Gui component and the Interaction Logger

Interface	Interface Methods	Description

IGuiLoggerControl	startLogging()
	stopLogging()
		This P54Gui interface alerts the GUI component when another application requests GUI usage event information. The Interaction Logger accesses these methods by calling startLogging and stopLogging.

IDataLogger	getData(*message*)	This Interaction Logger interface has one method – getData. The Logger will receive feedback messages from any application that calls the getData function.

Table 3.3 Summary of interfaces and methods added for GUI interaction event logging

The sequence of events is that the Interaction Logger must inform the P54Gui component that GUI interactions are desired, by calling the P54Gui component's *startLogging* interface method. When this method is called the P54Gui will set the global Boolean value *b_LogData* to true and create text messages containing GUI activities, as they occur. Once a GUI event takes place the P54Gui component sends the information about that interaction to the Interaction Logger via the *IDataLogger's* g*etData* interface method. Before the Logger shuts down it calls the P54Gui's *stopLogging* method to signal that no further GUI activity messages are needed. It is not until this point that *b_LogData* is reset to false.

The P54Gui called the g*etData* method when either one of two events transpired – the state of any GUI button changed or a keystroke was entered into a GUI text field. In order to prevent either the button control class or the text field control class from calling g*etData* while that method was busy, synchronization was used to give sole access to the first event (button press or keystroke) to call this function. That event had priority until the data could be safely sent to the Logger. On the Interaction Logger end of the process, the *getData* method receives the GUI event messages. This *getData* method waits for its message buffer to fill up (occurs when a GUI log message is sent) and then makes a call to the *logMessage* function (discussed in the next section) to log the GUI usage event. Back on the P54Gui side, once the message has been transmitted to the Logger, the message buffer is flushed in preparation for another GUI interaction to occur.

3.4 Interaction Logger

The brief background on Project54 inter-application text messaging as well as the explanation for the P54Gui adaptations necessary for GUI event logging provided the groundwork for the initial phase of this thesis work – logging user interaction events. As has been mentioned previously, the Interaction Logger was designed to monitor and record the SUI, GUI, and hardware usage events that could occur within a police cruiser. This section will describe the Logger software design approach as well as many of the details regarding its implementation.

Before any programming could be started it was important to have a plan put in place for what the Interaction Logger was going to accomplish. As was mentioned in Chapter 1, there needed to be a tool capable of recording all the events going on within a police cruiser, not just speech. With such a tool, designers and law enforcement officials alike would have access to information directly related to what aspects of Project54 user interfaces officers tend to prefer and in what situations the Project54 interfaces may be used. To ensure that accurate results were being generated, the interaction event recording had to be invisible to the officers. Certainly their consent to participate in an HCI study was required but once the software was installed on a car's computer it needed to function in the background, not interfering with the officer's daily workload. This requirement meant that the Logger could not have a GUI of its own. Once installed, the Logger had to operate automatically, without any external commands issued to it. Furthermore, the software had to be streamlined enough to avoid creating noticeable system lags. Any performance degradation would very likely become a nuisance to officers using the system, to the point that it might cause the officer to alter his or her system usage behavior. In general, any factors that would cause an officer to use the Project54 system in an uncharacteristic fashion could generate misleading event logs and should be avoided.

With these considerations in mind, implementation of the Interaction Logger could begin. Two of the first issues addressed dealt with how best to initialize and eventually shut down the application. Normally, these two program aspects would be considered benign and no formal discussion would be necessary. However, in order to maintain an accurate log of user event activity, a list of the other programs running on Project54 needed to exist. The programs on that list needed to shut down before the Interaction Logger to avoid missing any events that might occur after the application had stopped logging.

The Logger's startup routine, depicted in the high-level block diagram shown in Figure 3.6, includes elements that make use of the Project54 messaging system as well as certain Windows Registry settings. The "BROADCAST STARTUP" message shown in the first block of the figure is a startup command sent from the Application Manager to all the Project54 applications running within a given vehicle installation. Each program, the Interaction Logger included, must then initialize its startup routine and report that it is loaded and ready to run, by sending the message "STARTUP" back to the Application Manager.

Once the application is issued the initialization command, it then verifies whether or not it should log user interaction events. This process is done in two steps. First, the program makes sure permission to log information has not expired. Second, the program ensures that logging permission has been enabled. Both processes are done by checking the appropriate values within the Windows Registry, shown in Figure 3.7. The Registry value LogDuration is the length of time (in days) after the installation date. This LogDuration value provides a clear time frame for data collection to occur. Since the value is adjustable, data collection may be done in a flexible manner, on a vehicle-by-vehicle basis. The default value for LogDuration is 28 days. Once the time span allowed by LogDuration has elapsed, the Logger will automatically set LogData to disabled. The LogData Registry value indicates whether or not permission has been granted to proceed with logging interactions. The value should be set to either "Enabled" or "Disabled", depending on whether or not event logging is allowed. For all intents and purposes the order of the two verification steps is irrelevant; once logging permission is denied for either reason the application merely runs in the background without logging any information at all.

The remaining three blocks shown in Figure 3.6 involve processes that will only be executed when the application is set up for logging. As was mentioned earlier, in the section regarding inter-application communication, by registering as an application message sniffer the Logger is capable of monitoring all the communications occurring between other applications. Tracking the message traffic is used both to determine the active Project54 window and generate a count of the number of other applications also loaded onto Project54. The latter feature is noted in the following block and plays a key role in the shutdown routine (discussed shortly). The other action listed within the next block is the creation of a version list text file. This file contains the version of each Project54 application, as shown within the Component Versions folder of the Windows Registry. Since not all police departments have the same Project54 system setup, knowledge of each application's version list would allow data analysis to be better-tailored to individual fleets' installations. The last major step included within the block diagram is the Logger's registration for feedback messages from other Project54 applications. Figure 3.8 shows the list of applications within the Windows Registry from which the Interaction Logger could request feedback information.

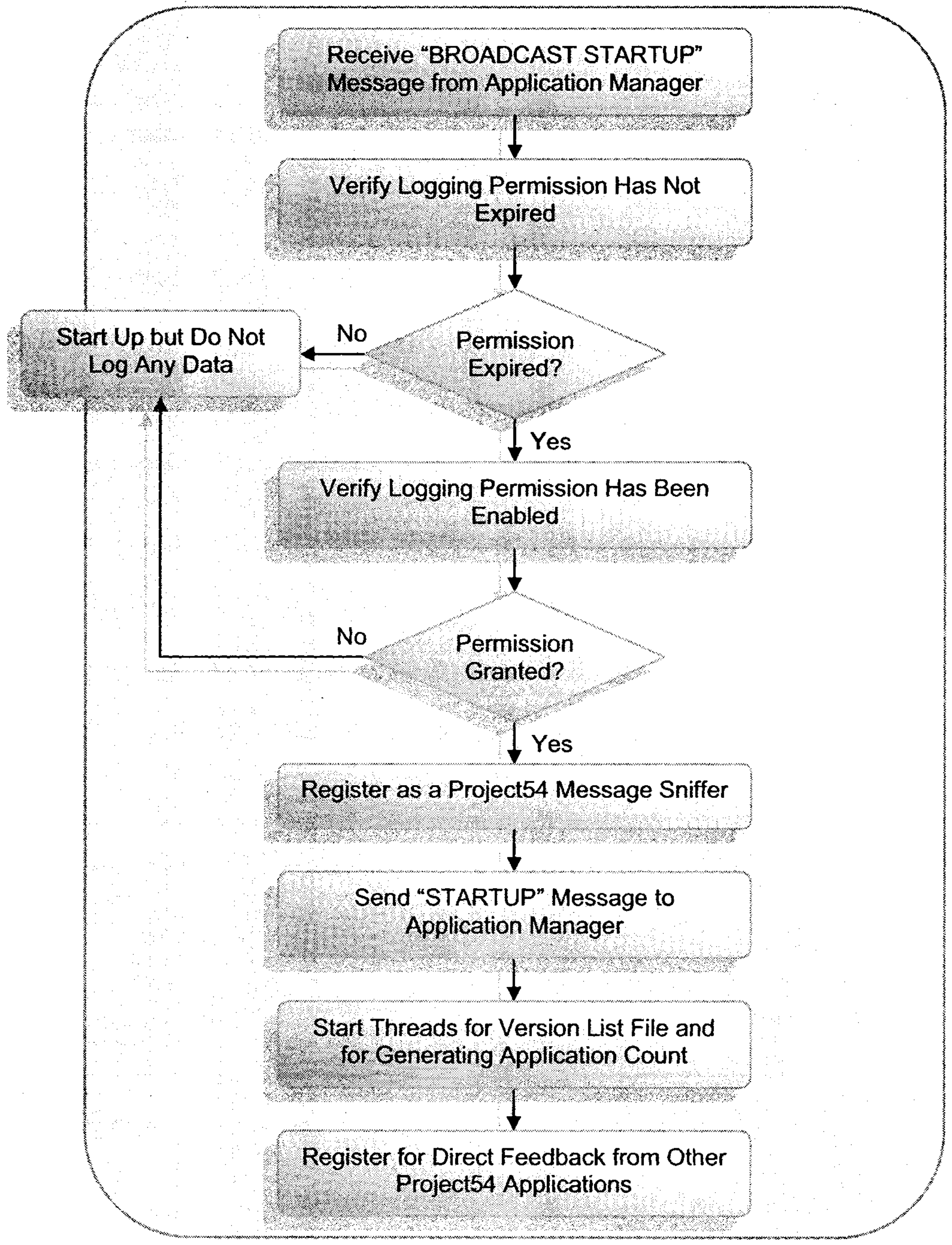
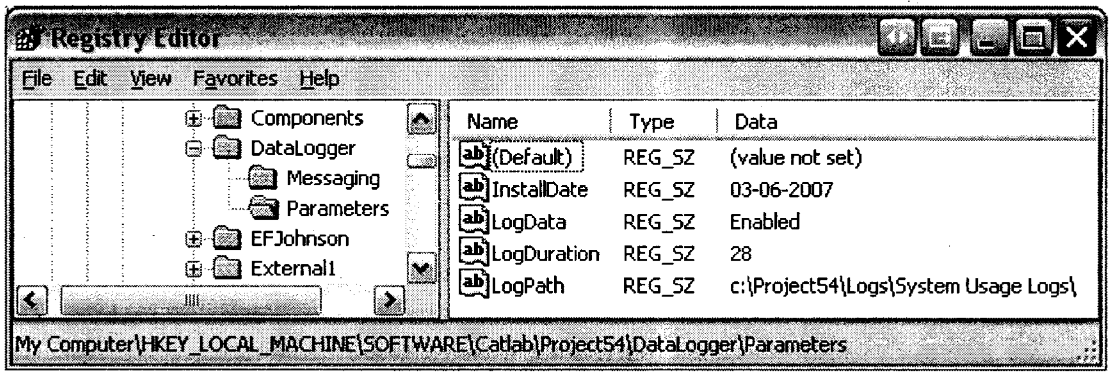

Receive "BROADCAST STARTUP" Message from Application Manager
Verify Logging Permission Has Not Expired
Permission Expired?
No
Start Up but Do Not Log Any Data
Yes
Verify Logging Permission Has Been Enabled
Permission Granted?
No
Yes
Register as a Project54 Message Sniffer
Send "STARTUP" Message to Application Manager
Start Threads for Version List File and for Generating Application Count
Register for Direct Feedback from Other Project54 Applications

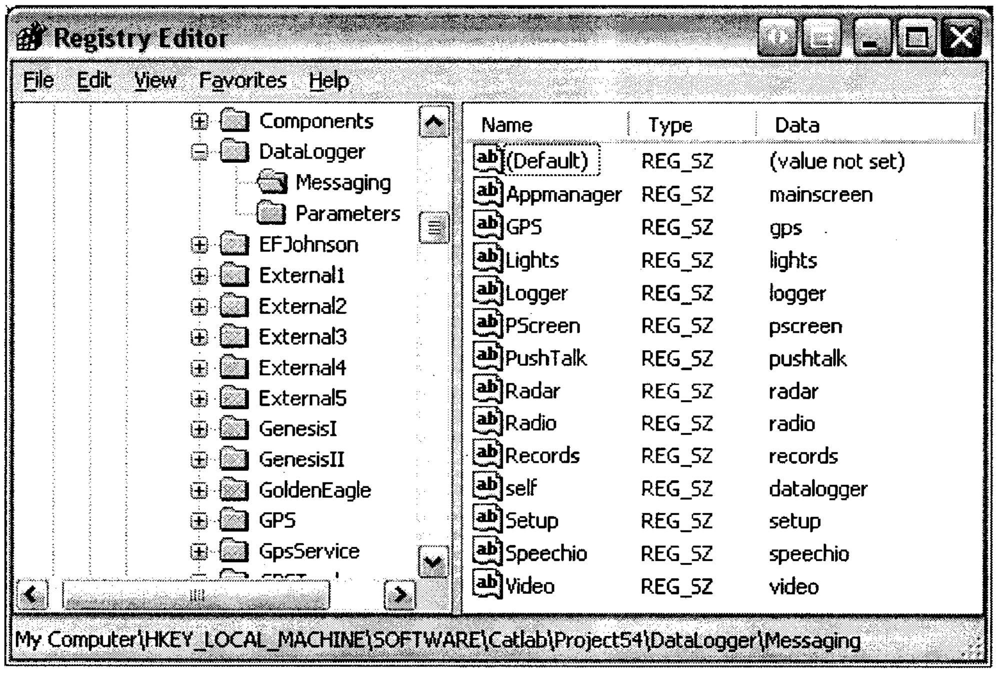
Figure 3.6 Interaction Logger program start routine



Name	Type	Data
(Default)	REG_SZ	(value not set)
InstallDate	REG_SZ	03-06-2007
LogData	REG_SZ	Enabled
LogDuration	REG_SZ	28
LogPath	REG_SZ	c:\Project54\Logs\System Usage Logs\

My Computer\HKEY_LOCAL_MACHINE\SOFTWARE\Catlab\Project54\DataLogger\Parameters

Figure 3.7 Interaction Logger registry parameters



Name	Type	Data
(Default)	REG_SZ	(value not set)
Appmanager	REG_SZ	mainscreen
GPS	REG_SZ	gps
Lights	REG_SZ	lights
Logger	REG_SZ	logger
PScreen	REG_SZ	pscreen
PushTalk	REG_SZ	pushtalk
Radar	REG_SZ	radar
Radio	REG_SZ	radio
Records	REG_SZ	records
self	REG_SZ	datalogger
Setup	REG_SZ	setup
Speechio	REG_SZ	speechio
Video	REG_SZ	video

My Computer\HKEY_LOCAL_MACHINE\SOFTWARE\Catlab\Project54\DataLogger\Messaging

Figure 3.8 List of applications from which the Interaction Logger requests feedback

Though registering for feedback may seem redundant since the Logger is already capable of sniffing message traffic, registering for feedback messages from applications has a distinct advantage over sniffing for this particular application. Due to the ability to create an application-specific ID for received messages (as discussed in the Project54 inter-application messaging section) the Logger only needs to pay attention to status messages that bear the proper ID. However, if the sniffer functionality was not taken advantage of, it would be far less convenient to determine the active window corresponding to user interaction and more difficult to ensure that the Logger was the last application to shut down. If, on the other hand, the software only took advantage of the sniffer functionality, it would be conceivable that important status information would not get logged due to the lack of any feedback clients for a given application to send messages to.

Once messages are received by the Interaction Logger they are handled according to the algorithm shown in Figure 3.9. When registering for feedback, the Logger provides the other programs with the unique message ID tag "DIRECTFEEDBACK" during a feedback request. The ID tag of each incoming message is checked when received by the application. If the message does not contain the tag "DIRECTFEEDBACK", it is a message picked up by the message sniffer. Since the system has already started up, the only sniffed messages of interest are the "SHOW WINDOW" messages sent every time the active window is changed. If the message does not contain the DIRECTFEEDBACK ID tag and it is not a SHOW WINDOW message, then it is ignored (i.e. not recorded).

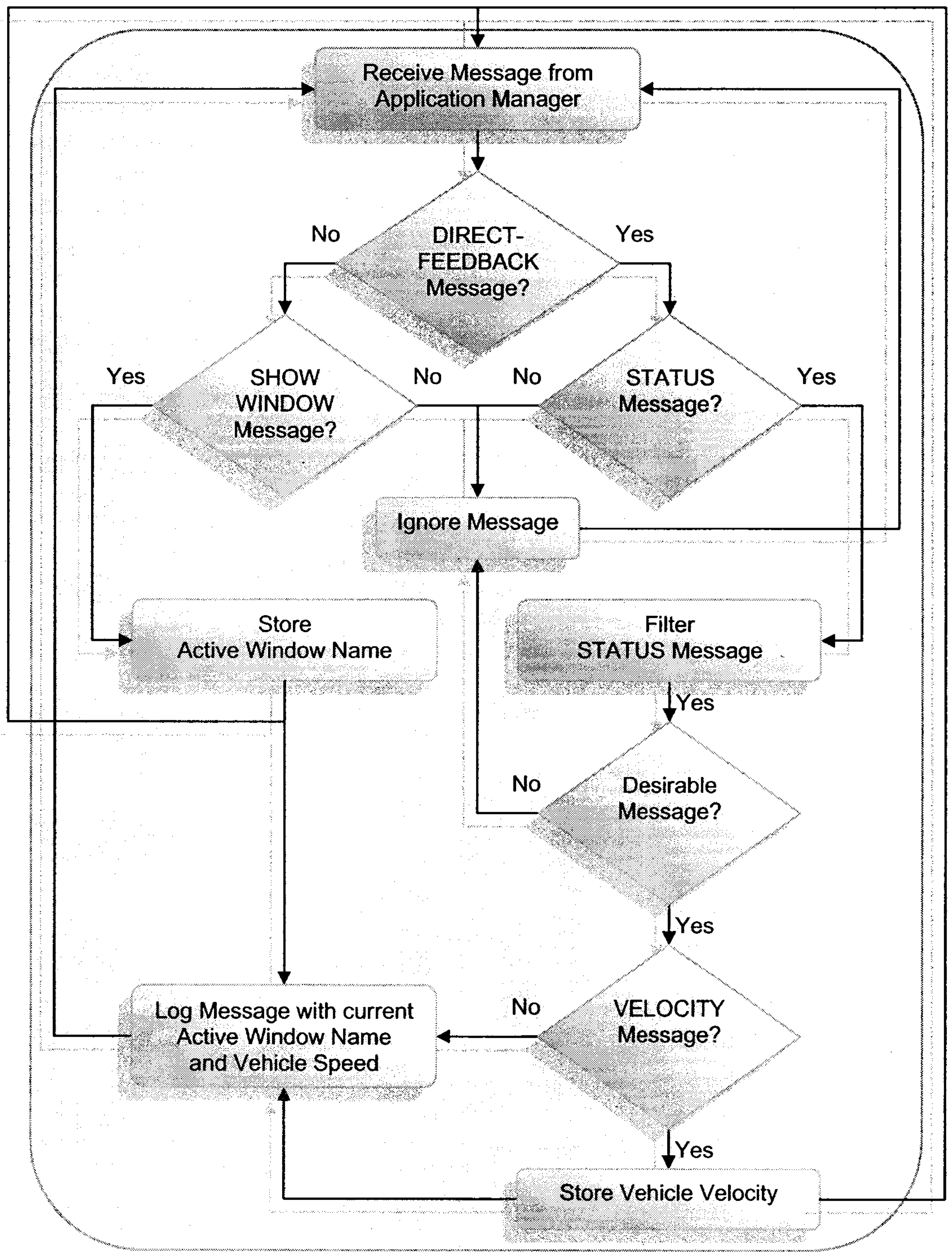


Figure 3.9 Interaction Logger message handling algorithm

Though the messages that do contain the DIRECTFEEDBACK tag are sent to the Interaction Logger specifically, it is still not guaranteed that those messages should be recorded. If the received message is not a status message (as described in the messaging section) it will be ignored. If, on the other hand, a status message is sent directly to the Interaction Logger that message must be screened before it can be logged. The screening process involves comparing the incoming status messages to the list of status messages shown in Table 3.4. If the received message matches any of those messages shown in the table, the message is ignored. The listing shown in Table 3.4 was comprised through a heuristic filtering process. During testing, it was determined that certain messages did not provide significant contributions to the information collection endeavor, but they did get transmitted frequently. Therefore, those less-important messages were filtered out to save storage space and preserve the clarity of the user interaction event information. The listed radar messages were ignored because they represented the *results* of an officer's actions (e.g. Turning on an antenna array *results* in knowing another car's "Target Speed"). The listed records and record queries message were ignored to prevent private information from being logged during this research project. The listed lights messages were ignored because they only report whether or not the light bar control head is active. This information is obvious during tasks performed using the light bar. Finally, the listed radio messages were ignored because they deal with monitoring radio traffic, not necessarily an officer's use of the radio itself. Special mention needs to be made regarding the "STATUS CHANNEL" entry within the

Table 3.4 listing. Messages that contain information regarding STATUS CHANNEL VOL (i.e. radio channel volume) are not filtered out, while all other STATUS CHANNEL data is ignored.

The use of the message filtering process was reduced with the help of selective feedback registration. If an application was created using the feedback handler found in the FEEDBACK.CPP file, specific feedback messages could be retrieved from that application without receiving all possible feedback data. For example, since the STATUS SPEECHIN message is the only information from the Speechio application that relates a user interaction, a feedback request such as:

```
Message(self,speechio,L"DIRECTFEEDBACK",L"FEEDBACK SPEECHIN ON")
```

could be sent to the Speechio application. Any other status messages Speechio might be able to send to the Interaction Logger would automatically be filtered out, without being transmitted to the Logger in the first place. Since not all applications have been built with the FEEDBACK.CPP file, this pre-filtering cannot completely remove the need for the Interaction Logger's own filtering functionality.

If a message does not get filtered out, it is checked for one more useful piece of information, whether or not it contains vehicle velocity data. If the message contains velocity information that data is stored and amended to all of the logged interaction information. If a message is not filtered out but does not contain vehicle velocity information, that message treated as an interaction and logged with whatever vehicle speed has already been saved. Since the in-vehicle GPS units update the vehicle speed every few seconds, the speed that gets logged along with the user interaction is an accurate one.

Project54 Application	Ignored Messages
Radar	STATUS PATROL SPEED STATUS TARGET SPEED STATUS LOCK SPEED
Records	STATUS ALERT STATUS QUERY STATUS PING
Record Queries	STATUS NEWQUERY STATUS ENDQUERY STATUS QUERYINPROGESS STATUS ADD STATUS QUERY STATUS RECEIVEDRECORD
Lights	STATUS LIGHTS CONTROL HEAD STATUS CONTROLHEAD
Radio	STATUS BUSY STATUS CHANNEL*

Table 3.4 STATUS messages not logged by the Interaction Logger application

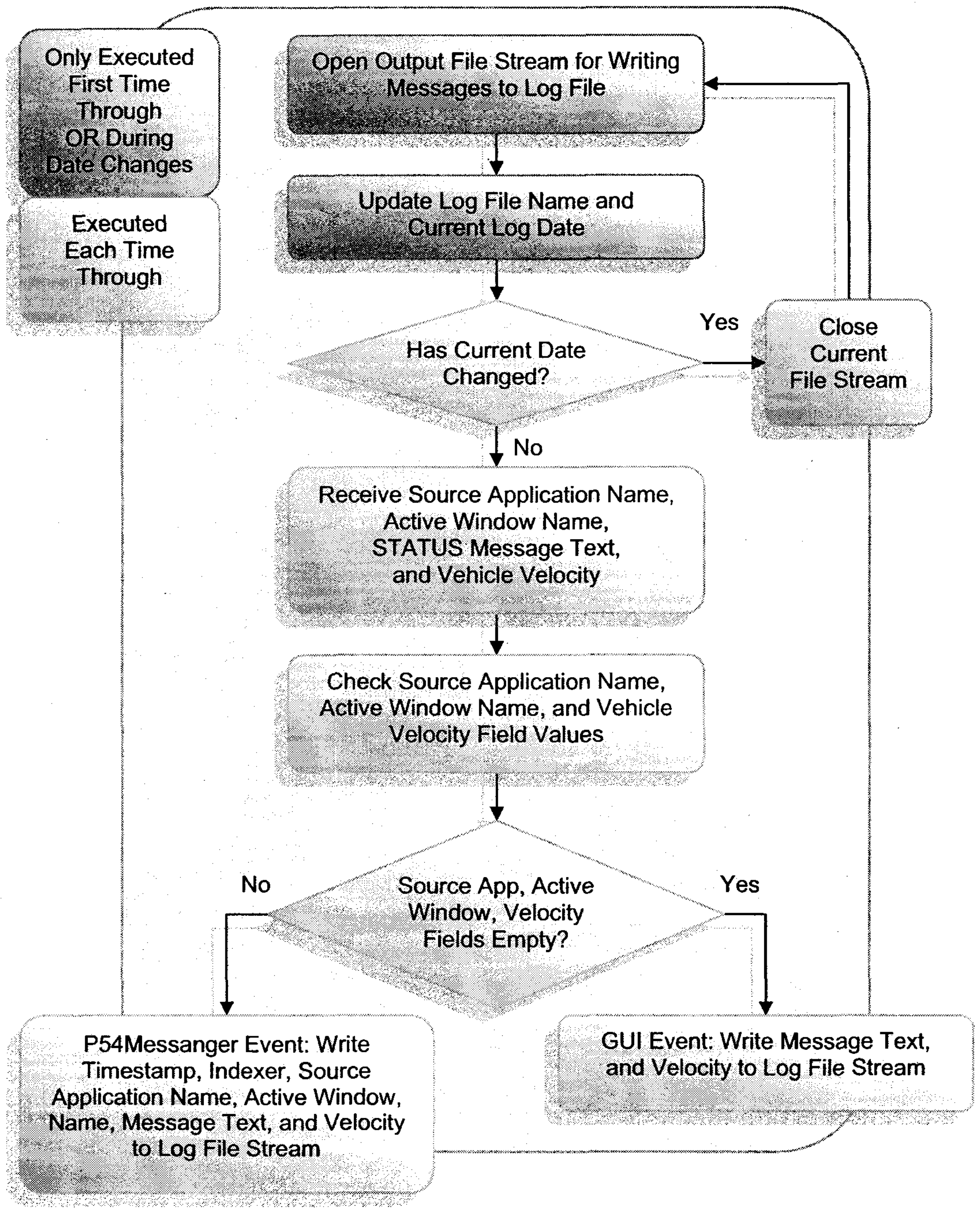
Once the status messages have been identified as direct feedback information and screened to weed out less-important data, they are ready to be recorded. The process, shown in Figure 3.10, indicates both the logging startup procedure and how all subsequent interaction information is recorded. The first time data is to be recorded (and each time a new day starts), the application must open a file stream to which that information will be written. The file stream is left open for the duration of the application's execution. The file stream will also close if the days change during logging, in which case the previous day's file stream will close and the new day's file stream will open. To keep the data logs' nomenclature simple, the name of a file is the same as the date on which that file was created. In other words, if a file was created on May 4, 2007, the name of that file would be 05-04-2007.

This file naming scheme makes it necessary to check the date in order to determine when a new log file has to be created. The check is performed each time a new message is ready to be recorded. By checking each message's date, it can be assured that no gray area would exist in which messages get logged in the wrong date's text file. If data recording occurs for more than one day without the computer restarting, the Logger will still be able to automatically detect a change in the date, close the previous date's log file, create a text file for the new date, open that new file, and write the buffered interaction message to the new date's file, with no detectable real-time delay.

With the properly-dated file stream ready to receive interaction data, the application waits for incoming messages to record. The logging process determines, based on the information available, whether the message pertains to a GUI interaction or a text message interaction. The differentiation between GUI messages and P54text messages is important because the messages have different formats. This decision process is based on whether or not the Logger has the following information: the source application's name, the active window name, and the vehicle's velocity. When this information is absent, the application deems the message present to be GUI interaction data. In the case of receiving

GUI interaction data, the vehicle velocity is appended to the original GUI message. If no GPS information is available, the velocity data appended to the corresponding usage logs is the string, “N/A”, to avoid knowingly recording a false speed. The text message data actually arrives at the message logger in pieces that must be put together before being written to the file stream. The pieces are the timestamp, an index (based on the number of milliseconds that have elapsed since system startup), the source application’s name, the active window’s name, the status message itself, and finally the vehicle velocity. Once the information is packaged in that format, it is recorded in a text file for later analysis.

The Interaction Logger’s shutdown routine is slightly more involved than most other Project54 applications’ shutdown processes. This is because, to ensure that no interactions are missed during system shutdown, the Interaction Logger must verify that it is the last application to terminate. Figure 3.11 shows a high-level block diagram for the Logger’s shutdown implementation. Once the Application Manager transmits the “BROADCAST SHUTDOWN” command to all the Project54 applications, the Interaction Logger checks its count of the number of currently-running applications. This count was created during startup by sniffing the number of “STARTUP” messages sent to the Application Manager. Similarly, during shutdown the Interaction Logger sniffs the number of “SHUTDOWN” messages each application sends to the Application Manager once they are ready to terminate. Each time a “SHUTDOWN” message is sniffed, the count of active applications is decremented by one. Once the count indicates that the Interaction Logger is the only application yet to shut down, the program will close the data log text file stream and terminate.



Only Executed First Time Through OR During Date Changes
Executed Each Time Through
Open Output File Stream for Writing Messages to Log File
Update Log File Name and Current Log Date
Has Current Date Changed?
Yes
Close Current File Stream
No
Receive Source Application Name, Active Window Name, STATUS Message Text, and Vehicle Velocity
Check Source Application Name, Active Window Name, and Vehicle Velocity Field Values
No
Source App, Active Window, Velocity Fields Empty?
Yes
P54Messanger Event: Write Timestamp, Indexer, Source Application Name, Active Window, Name, Message Text, and Velocity to Log File Stream
GUI Event: Write Message Text, and Velocity to Log File Stream

Figure 3.10 User interaction logging process

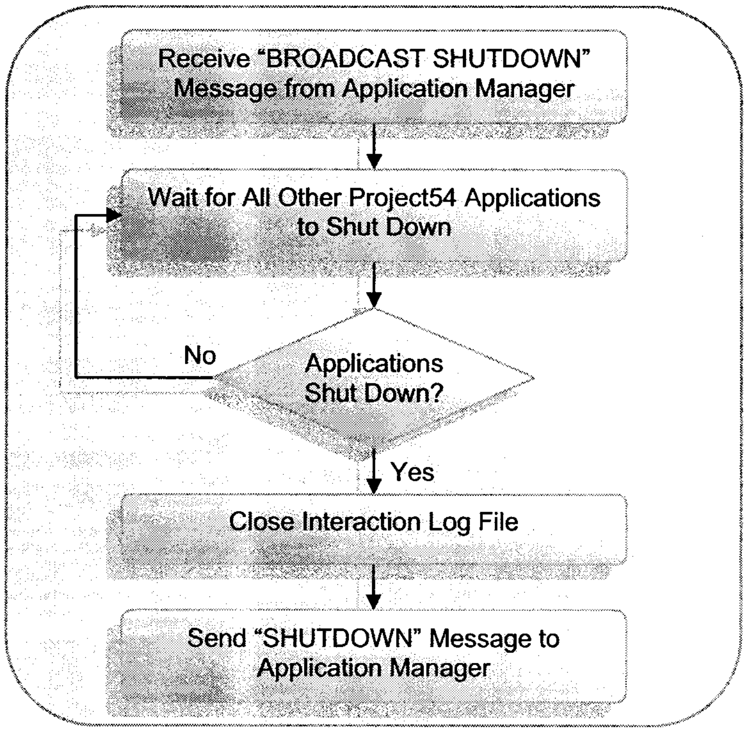

Receive "BROADCAST SHUTDOWN" Message from Application Manager
Wait for All Other Project54 Applications to Shut Down
No
Applications Shut Down?
Yes
Close Interaction Log File
Send "SHUTDOWN" Message to Application Manager

Figure 3.11 Interaction Logger program shutdown procedure

CHAPTER 4

SOFTWARE TESTING AND DEPLOYMENT

4.1 Introduction

The second proposed step of the user interaction logging tool development was to perform testing to verify that Project54 HCI event information could be accurately gathered from police cruisers. As part of the initial development, the application was tested on a desktop computer where any noticeable bugs were removed from the program. More intensive testing was performed by loading the software into a laboratory car mock-up, a driving simulator, and two test vehicles. The tests were conducted in three phases – simulated HCI event recording under simulated driving conditions using the laboratory car (Lab Car), actual HCI event recording under simulated driving conditions using the driving simulator, and actual HCI event recording under actual driving conditions using two Project54 test cars. Once the tests proved the software was stable and functioning properly, it was deployed in two state police cruisers for actual user interaction data collection. All participants of any data collecting procedures had a signed consent form on record, prior to the collection of their data.

4.2 Lab Car Testing

Once the interaction logging software was realized, the program was then tested in the Lab Car. The Lab Car, shown in Figure 4.1, is, practically speaking, the front seat of a police cruiser. The Project54 system within the Lab Car is equipped with an IDB network and assorted hardware devices, such as a radio, lights, siren, GPS, and radar. The IDB network connects the devices to a console-mounted embedded PC. This testing setup, with its hardware components and software settings, adequately represents the system currently installed within a typical New Hampshire state police cruiser.

Figure 4.1 Project54 lab car

To keep the installation process simple, a batch file was created that could automatically load the Interaction Logger, the updated P54Gui software, and the necessary Windows Registry settings onto a Project54 system setup. The folder that contained the installation batch file also contained the Interaction Logger, as well as P54Gui and Records applications that had been updated to accommodate the Logger. Also included in the folder were the text files that contain data to be loaded into the Windows Registry and the Project54 Application Selection program.

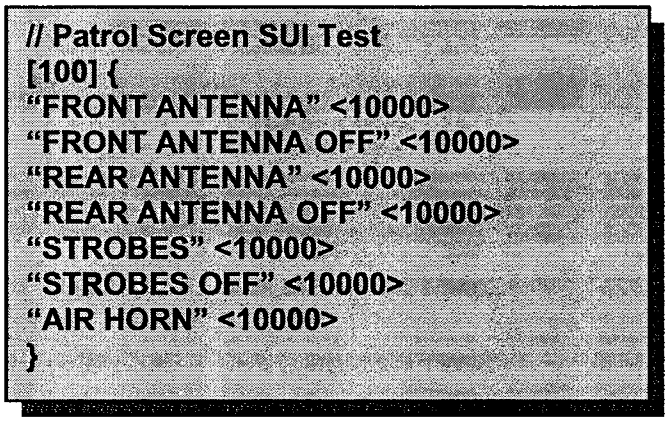
With the application loaded onto the Lab Car automated tests were conducted on button press user interactions and speech command user interactions separately. Samples of the files used to conduct these tests are shown in Figure 4.2 (the GUI file) and Figure 4.3 (the SUI file). In order to make sense of the information provided in these figures, Table 4.1 has been included as a key. The GUI test file example indicates that the test started within the Project54 Patrol Screen. Certain buttons (indicated by their column and row coordinates) had simulated presses occur once every ten seconds. The SUI test file sample also indicates that the test was conducted with the Patrol Screen as the active window. In this case, a simulated speech command (the text between the quotations) was issued once every ten seconds. These testing procedures were made possible by executing the test file commands within a pre-existing Project54 automated interface testing application [25].

Code	Description
{...}	Braces delimit a block of test commands.
[percentage]	Execute this line or block of commands based on the percentage indicated in brackets.
<milliseconds>	Once a command has been given, the test will wait the indicated number of milliseconds before moving to the next line.
(column, row)	The button coordinates are given to the test file in this format, to issue button presses.
"SIMSPEECH COMMAND"	The commands within quotations are interpreted as simulated speech but are treated by the system as standard speech commands.

Table 4.1 Test file command reference

```
// Patrol Screen GUI Test
[100] {
(6,2) <10000>
(7,2) <10000>
(8,2) <10000>
(6,3) <10000>
(7,3) <10000>
(8,3) <10000>
(6,4) <10000>
(7,4) <10000>
(8,4) <10000>
}
```

Figure 4.2 Sample GUI test file



```
// Patrol Screen SUI Test
[100] {
"FRONT ANTENNA" <10000>
"FRONT ANTENNA OFF" <10000>
"REAR ANTENNA" <10000>
"REAR ANTENNA OFF" <10000>
"STROBES" <10000>
"STROBES OFF" <10000>
"AIR HORN" <10000>
}
```

Figure 4.3 Sample SUI test file

Since the Lab Car tests were automated, it was possible to perform constant testing for long periods of time. In this case testing was done on the button press interaction events and speech command interaction events separately over the course of one entire weekend each, spanning from Friday evening to Monday morning (approximately sixty six hours a piece). After those tests were complete, another round of testing was done in which both GUI and SUI commands were issued in ten second intervals. This round of testing went on for two weeks (approximately three hundred hours).

A program was written to verify that the recorded data matched the automated SUI and GUI commands. The process involved first manually looking at the recorded data to verify that the first iteration of commands matched the testing script. Once that step was completed the first iteration of recorded data was used as the benchmark to which all other iterations of recorded data were compared. After more than 1.4 million lines of recorded data (covering more than fourteen thousand iterations of automated SUI and GUI commands) were checked, no anomalous log entries were discovered. Test results concluded that the logging software was able to accurately keep track of Project54 user interactions without generating systems crashes.

4.3 Driving Simulator Testing

After the Lab Car tests were completed and positive results were generated, the program was then tested in the Project54 driving simulator. The driving simulator, shown in Figure 4.4, is, similarly to the Lab Car, the front seat of an automobile, but with the addition of a bank of computers and a projector array capable of displaying virtual driving scenarios under various conditions. Also like the Lab Car, the driving simulator is outfitted with Project54 software. However, the driving simulator does not make use of various hardware devices, such as lights; instead the simulator emulates most device functionality within software (as is the case with the radar, for example). However, unlike the Lab Car, the driving simulator is able to simulate vehicle speed. For testing purposes, the simulator sent vehicle speed messages via IP messaging to the Project54 radar application which were interpreted as radar "patrol speed" data. Those radar patrol speed messages were then sent from the radar application to the Interaction Logger, to be appended to the end of every recorded user interaction message. Since there is no autonomous driving capability within the driving simulator, testing had to be performed manually.

The test procedure itself consisted of performing scripted tasks both in simulated driving and parked conditions. Table 4.2 contains an example of the script used to carry out the simulator testing. The tasks listed in the table were performed both under simulated driving conditions and under simulated parking conditions. In both cases the tasks were also performed using both the SUI and the GUI. The individual tests lasted for approximately fifteen minutes apiece and were performed a total of ten times by five different members of the Project54 team. The result of the tests, verified using the same procedure in which the Lab Car logs were inspected, indicated that the Interaction Logger could accurately record user interactions as well as the appropriate driving condition (moving or stopped).

Order of Tasks Performed
Turn Front Antenna ON
Turn Lock ON
Turn Lights & Siren ON
Turn Lights & Siren OFF
Turn Lock OFF
Turns Rear Strobes OFF
Turn Rear Antenna ON
Turn Rear Antenna OFF

Table 4.2 Example of one script used to test interaction logging on the driving simulator

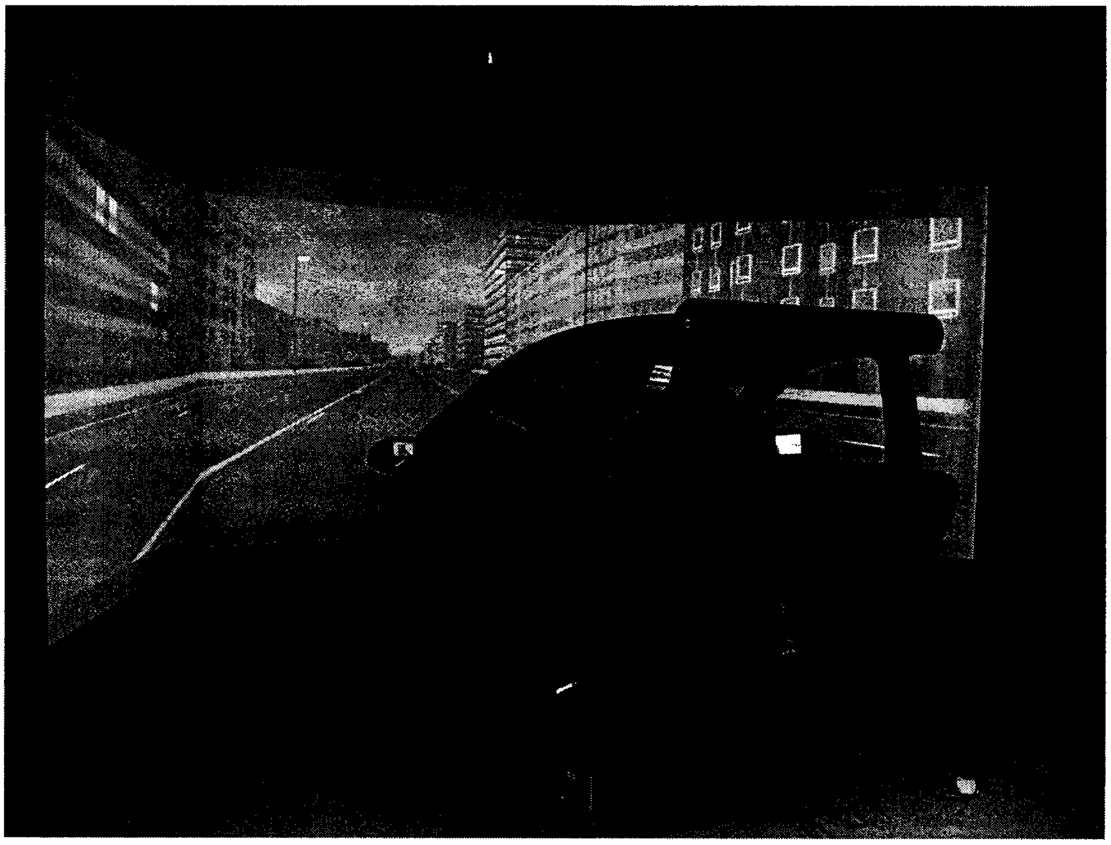


Figure 4.4 Project54 driving simulator

4.4 Project54 Test Vehicle Testing

The third set of test conditions was realized during road tests, using the Project54 show car and Chevrolet Impala to collect interaction data. The show car and Impala are both Project54-equipped vehicles, identical in every respect to a New Hampshire State Police cruiser. The cars are outfitted with the same hardware (GPS, radar, radio, lights, siren, etc.) that may be found within a state police cruiser as well as the same Project54 software configuration. The biggest advantage to using the show car and Impala for testing was that they were able to recreate actual in-vehicle device usage more accurately than either the Lab Car or the driving simulator.

The Show Car and Impala testing both consisted of four experienced, authorized Project54 employees driving while using the SUI, GUI, or hardware controls to operate the in-vehicle equipment. The operators' system usage was unscripted and only served to ensure the interaction logging application was stable. The testing went on for approximately ten hours with none of the test subjects detected system lags or any other system performance issues during any of the tests conducted.

4.5 Police Cruiser Deployment

Once all the test results were collected and reviewed it was evident that the interaction logging software was stable and could accurately record in-vehicle user interactions involving both Project54 interfaces (SUI and GUI) and the standard device control heads. The last step as far as the information gathering process was concerned was to implement interaction logging within actual police cruisers. Two New Hampshire state police officers volunteered to be test subjects for this user interaction evaluation. The Logger software was loaded in the two police cruisers, using the batch-file installer, and recorded usage data whenever the cruiser's embedded computer was turned on and running Project54. The data logging went on for twenty-eight days, at which point the data was collected from the cruisers.

Since the officers were not actually testing any software in this case, they were asked to refrain from using either the Project54 interfaces or the standard device controls in a manner that would be different from the way in which these controls are normally used. It was very important to make sure that the usage information that was recorded represented normal, day-to-day activities even if that meant the Project54 system never got turned on. Also as part of this evaluation, the officers were asked to fill out a questionnaire (See Appendix A). Among other things, this questionnaire gave the officers the opportunity to state how they felt they utilized the Project54 system during the course of their shifts. The results from this questionnaire were compared to the data collected directly from the police cruisers as part of this preliminary evaluation.

CHAPTER 5

DATA ANALYSIS AND VISUALIZATION

5.1 Introduction

After twenty eight days, the interaction logs were retrieved from the police cruisers that were used in cooperation with this research. Table 5.1 provides a summary of the amount of data gathered from the two participating police officers. All told, there was approximately five megabytes of information available from both police cruisers that needed to be analyzed. To this end, a program was created to post-process the data by way of parsing information from the original log files and placing it into new files. The new files were then used to generate data visualizations, meant to illustrate system usage trends. This chapter details the design of both the information post-processing application and the data visualization program. Visualization examples are also included to demonstrate the usefulness of the quantitative analysis.

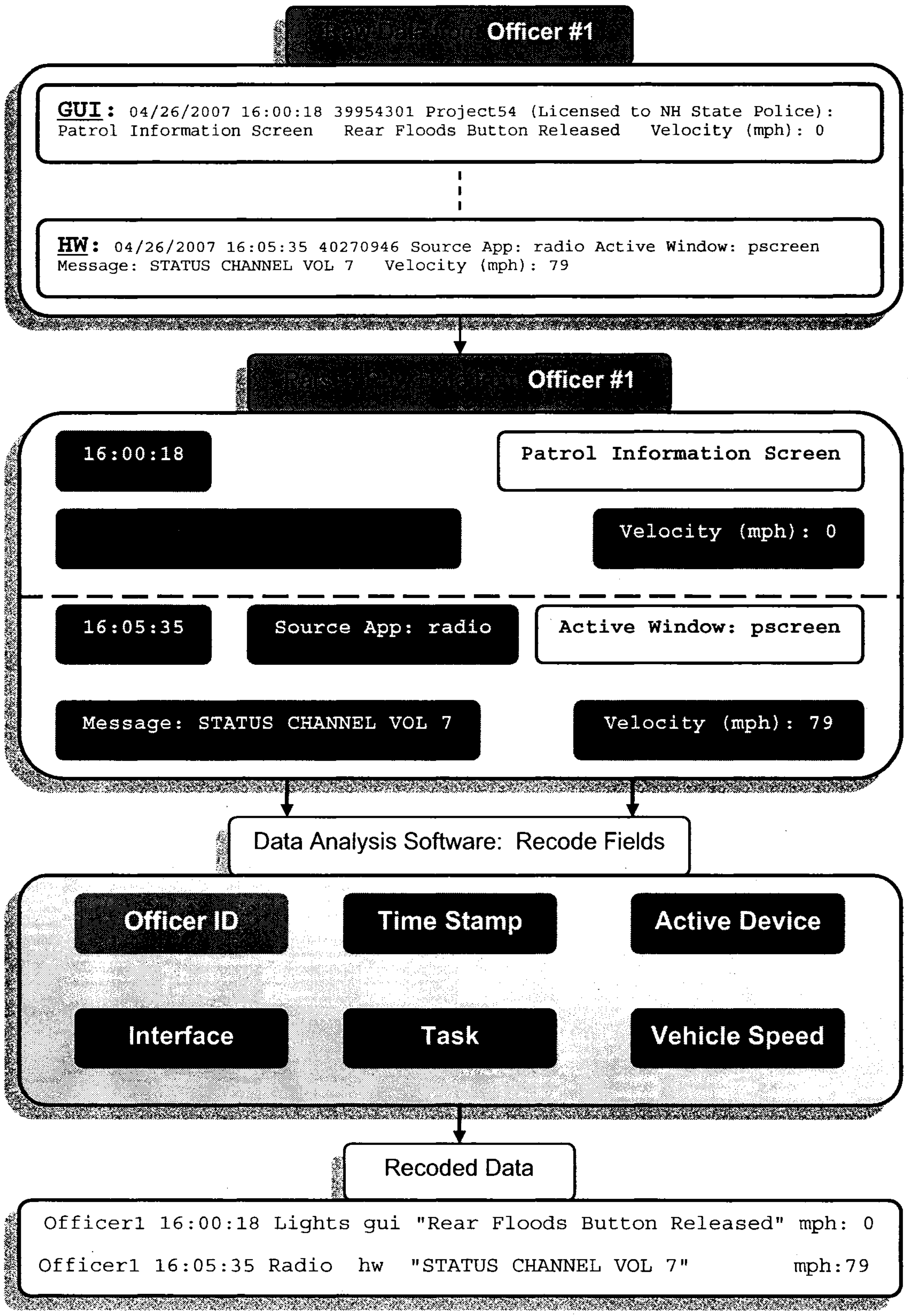
	Officer #1	Officer #2	Combined	Averaged
Totals Days of Data	23 days	19 days	42 days	21 days
Total Files of Data	23 files	19 files	42 files	21 files
Total Amount of Data	3.6 MB	1.8 MB	5.4 MB	2.7 MB
Mean Daily Log File Size	157 kB	95 kB	129 kB	126 kB
Total Number of Messages	28,611	14,456	43,067	21,533
Mean Daily Message Count	1243	760	1025	1,001

Table 5.1 Summary of collected data statistics from two deployed police cruisers

5.2 Data Post-Processing Development

The data analysis program provides an automated solution for determining a police officer's Project54 usage characteristics. The software functions by applying two main data analysis techniques – data selection and recoding – to the raw data input stream [5]. Data selection is a process by which the user interaction events of interest are separated from "noise" data (irrelevant data). Since a large amount of undesirable information was never logged in the first place, the selection process was minimal in that it only applied to ignoring certain status messages. For example, if a speech command was issued to turn strobes on, the corresponding event log sequence would contain both "STATUS SPEECHIN STROBES" and "STATUS STROBES". In this case the SPEECHIN message contains the user interaction while the second message represents system feedback, not an action taken by the officer. To avoid double-counting this event, the "STATUS STROBES" message is ignored. Similarly, when a button press is used to turn strobes on, the button-press message is logged and the accompanying "STATUS STROBES" message is ignored.

Data recoding involves producing a new event log based on the results of the selection process. Once data selection identifies information as being important, that information is reorganized into a new text file. This step is especially useful considering not all of the raw data log events follow the same format. For instance, button press log entries do not contain the same data fields as the speech or hardware entries. This is because the status message format, discussed in Chapter 3, is not supported by the GUI application. Recoding the raw data makes such format discrepancies irrelevant because once events are recoded all the information is presented in the same fashion for analysis. A graphical representation for the data recoding procedure is shown in Figure 5.1. The uppermost portion of the figure contains snippets from two different log entry lines (separated by a dashed line), taken from one of the officers' records. For the sake of fitting the figure better, the log entries have been edited. The boxes around the different data fields within the "Parsed Raw Data from Officer #1" block are color-coded to match corresponding fields within the "Data Analysis Software: Recode Fields" block. Even though it has not been used in the example illustrated by this figure, the box containing the Active Window field in the second log message has been included to better illustrate the difference in log entry formats.



```
GUI: 04/26/2007 16:00:18 39954301 Project54 (Licensed to NH State Police):
Patrol Information Screen   Rear Floods Button Released   Velocity (mph): 0

HW: 04/26/2007 16:05:35 40270946 Source App: radio Active Window: pscreen
Message: STATUS CHANNEL VOL 7   Velocity (mph): 79
```

```
Officer1 16:00:18 Lights gui "Rear Floods Button Released" mph: 0
Officer1 16:05:35 Radio  hw  "STATUS CHANNEL VOL 7"        mph:79
```

Figure 5.1 Data Analysis Recoding Procedure

Once events had been identified as user interactions (the selection process mentioned earlier), those data were parsed into the appropriate recoding fields (also color-coded in the figure). For example the time stamp, shown in the brown box of each log entry, can be thought of as being placed into the recoding field's time stamp bin (also shown as a brown box), then dumped into the recoded data stream, unchanged. However, the items which refer to the particular task an officer carries out (shown in the blue boxes) possess information regarding both the task performed itself, and the interface used to perform that task (i.e. SUI, GUI, or hardware). The interface is evident based on the format of the message in that "STATUS SPEECHIN" messages must involve the SUI, "STATUS ..." messages must involve the hardware, and any other messages must involve the GUI. Therefore, any items contained in the raw data's blue boxes may be thought of as placed into *both* blue recoding fields' bins and then dumped into the "Recoded Data" file. When completed, each line of the recoded data file would hold the same fields of information: the officer whose data is being analyzed, a date stamp, a time stamp, the device/application used for the interaction, the interface used for the interaction, the specific interaction itself, and the vehicle's speed during the interaction.

Also note that the contents of the blue box in the first raw-data event, "Rear Floods Button Released", have a different font color than the other data. The color change is intended to signify the third piece of information that may be gleaned from the message – namely that "Rear Floods" has to do with the lights application. While the GUI-related log messages do not contain their own field to specify the device used during a particular task, knowledge of each device's features/components is enough to identify which device is being used. "Rear Floods" messages belong to lights, "Front Antenna" messages belong to radar, "Log On" messages belong to records, etc. The same knowledge could have been applied to those log messages within the "Source App" message field (the red box) but it was easier to simply use the "Source App" information when it was readily available.

With some exceptions, the recoding procedure shown in Figure 5.1 was all that was required to decipher interactions directly involving the Project54 system, that is – the SUI and the GUI. However, developing a method for determining hardware interactions had to take other factors into consideration. For one, timing played a role in determining hardware usage. If log entries occurred too rapidly, it was evident that a human did not perform them. Specifically, for status messages to have been considered as candidates for hardware interactions, they had to have occurred at least one tenth of one second after the previous known user interaction. Though one tenth of one second may seem low for a threshold setting, viewing the log data indicated that this time was both too fast for human responses and too slow for computer feedback responses.

Timing cues alone were not enough to judge hardware usage. Determining hardware interactions also involved monitoring each device's operational status (i.e. "ON" or "OFF" in most cases), and updating that device's status whenever a known interaction took place. In other words, if a speech command was issued to turn the rear floods on, there would be an accompanying status message that indicated the rear floods were on. While this status message would not indicate a hardware interaction took place, the recoding application would still have to update the rear flood's status from "OFF" to "ON". If, however, a status change was observed in the front antenna without an accompanying SUI or GUI event, it would indicate that a hardware interaction had taken place.

The principle behind identifying hardware usage is shown in Figure 5.2. The figure begins with reading lines from the data logs, two at a time. If the "Current Line" (the first of the two lines read) contains either a GUI or a SUI event message, the data on the line is recoded. Failing either of these two options, the message is checked to see if a status change has occurred, with the application updating the device's status when needed. If the message happens to be at least a tenth of a second after the previous known interaction, while simultaneously not occurring within the same millisecond as the next line's event, it is likely that the current line's event represents a hardware interaction. The lack of certainty comes from some caveats regarding the in-vehicle devices (the bottom-most block in the figure). These device exceptions had to be handled individually to ensure the accuracy of the hardware interaction accounting. In order to test the procedure, representative selections of the logs were individually analyzed by the algorithm and compared to manual observations to ensure the algorithm matched human perception of events.

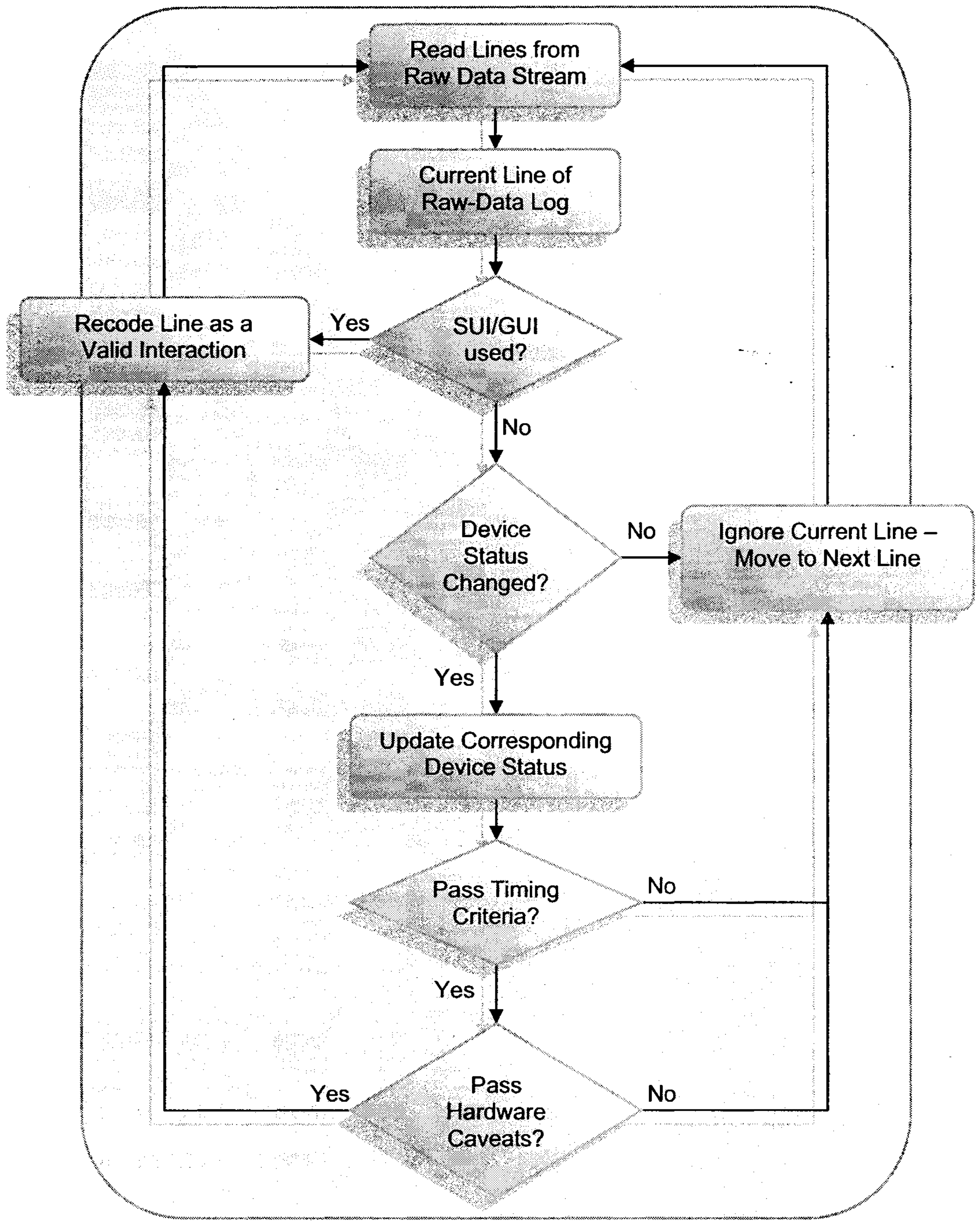

Read Lines from Raw Data Stream
Current Line of Raw-Data Log
SUI/GUI used?
Yes
Recode Line as a Valid Interaction
No
Device Status Changed?
No
Ignore Current Line – Move to Next Line
Yes
Update Corresponding Device Status
Pass Timing Criteria?
No
Yes
Pass Hardware Caveats?
Yes
No

Figure 5.2 Hardware Usage Identification Algorithm

In particular, there were two pieces of in-vehicle equipment that generated event logs which were inconsistent with the algorithm shown in Figure 5.2 – the light bar and the radar. The list of possible user interaction message structures, shown in Table 5.2, was determined by manually looking through the officers' log files for data irregularities, in a similar fashion to Nathan Purmort's radio traffic analysis [26].

The first four types of logged message blocks were handled easily by the algorithm in Figure 5.2. Accounting for more than five thousand of the almost seven thousand total logged interactions (76% of the total interactions), the first four types were by far the most common. The remaining types of logged message blocks presented some conflict. Type 5 shows the case of speech command messages getting logged in a counter-intuitive order. There were perhaps twenty or so instances of such speech command logging present in the data available for this research that would not have been detected because the algorithm did not account for receiving a feedback message before the speech command that generated it. The solution for this project was to manually go through a copy of the raw data files and flip the order of logged events whenever it was clear that the speech command was out of logical order with its resulting device feedback message. In the future, however, an automated solution to this problem should be employed.

Types 6 through 9 are examples of light bar message groupings that are too complicated for the basic interface identification algorithm to handle. Types 6 and 7 provide GUI and SUI examples (respectively) for the use of light bar strobes toggles. Both officers used a Whelen™ light bar and control head which employed a three-way switch that toggled the state of the strobes, between front strobes, rear strobes, and all strobes. The nature of this switch was such that only one strobes state could be active at a given time. In other words, if Front Strobes was active, Rear Strobes and All Strobes had to be off. The GUI buttons and SUI mimicked this behavior in such a way that issuing a Front Strobes "ON" command while another strobes state was active would automatically release either of the other two strobes GUI buttons (if either were already active), turn off the other strobes state, then activate the Front Strobes. In order to handle this data series properly, the strobes states were still updated, according to the process of Figure 5.2, but the three messages comprised only one GUI/SUI interaction, as opposed to, say, a GUI/SUI interaction and a hardware interaction (since, at first look it would appear as though there was an unaccounted-for hardware command).

Types 8 and 9 indicate examples for the use of the Project54 Lights and Siren functionality. There are situations in which police officers commonly turn on their Front and Rear Strobes, their Wig Wags, and their Wail Siren. To speed this process along Project54 developers added the Lights and Siren command to the GUI and SUI. Since the functionality of activating those three Lights and Siren functions only exists within Project54, it is possible to already have, for example, the Wig Wags on when the Lights and Siren button is pushed down. This does not affect the state of the Wig Wags but a log of the Wig Wags state is still recorded when Lights and Siren is pressed. The recoding program had to individually keep track of each of the states for the three Lights and Siren constituents and update them as needed. The program did not count the state change of any of the three devices as an interaction.

Types 10 and 12 represent examples for the use of the second piece of equipment in question, the radar. The radar used in both police cruisers is the Stalker radar, which comes with a remote control to perform hardware interactions. Type 10 demonstrates the act of activating the Front Antenna while the Rear Antenna is off. The main issue is the presence of the extra hardware feedback messages, “STATUS FRONT ANTENNA SELECTED” and the second occurrence of “STATUS FRONT ANTENNA”. The solution to this was to ignore all “STATUS X ANTENNA SELECTED” messages as they did not provide any information that was not readily available simply by observing the status of both antenna arrays themselves. In this case, the recoding program ignored the second antenna status message.

Type 11 represents those SUI/GUI commands issued to an antenna array when the other antenna array was already active. This type of message group was handled in the same fashion as the Type 6 strobes grouping, with the program ignoring the extra antenna status message.

Type 12 shows an example of an antenna array hardware control. In this case, one of the antenna arrays is already on when the remote control is used to turn the other antenna array on. Since the program has already ignored the “STATUS X ANTENNA SELECTED”, the only thing left for the program to do is to make sure that only the array getting activated is recorded as a hardware interaction, while the status of both arrays is updated.

	Logged Message Series	Interpretation
Type 1	XXX Button Pressed Down STATUS XXX or XXX Button Released STATUS XXX OFF	A single GUI Interaction
Type 2	STATUS SPEECHIN XXX STATUS XXX or STATUS SPEECHIN XXX OFF STATUS XXX OFF	A single SUI Interaction
Type 3	STATUS XXX or STATUS XXX OFF	A single Hardware Interaction
Type 4	Keystroke Entered: X	A single GUI Interaction
Type 5	STATUS XXX STATUS SPEECHIN XXX or STATUS XXX OFF STATUS SPEECHIN XXX OFF	A single SUI Interaction (Type 2 – Order Flipped)
Type 6	Front Strobes Button Pressed Down STATUS REAR STROBES OFF STATUS FRONT STROBES	A single GUI Interaction (Only seen when one of other 2 Strobes States was active)
Type 7	STATUS SPEECHIN FRONT STROBES STATUS STROBES OFF STATUS FRONT STROBES	A single SUI Interaction (Only seen when one of other 2 Strobes States was active)
Type 8	Lights & Siren Button Pressed Down STATUS LIGHTS AND SIREN STATUS STROBES STATUS WIG WAGS STATUS WAIL	A single GUI Interaction (Same Pattern for turning Lights & Siren "OFF")
Type 9	STATUS SPEECHIN LIGHTS AND SIREN STATUS LIGHTS AND SIREN STATUS STROBES STATUS WIG WAGS STATUS WAIL	A single SUI Interaction (Same Pattern for turning Lights & Siren "OFF")
Type 10	Front Antenna Button Pressed Down STATUS FRONT ANTENNA STATUS FRONT ANTENNA SELECTED STATUS FRONT ANTENNA	A Single GUI Interaction (Same Pattern for "OFF")
Type 11	STATUS SPEECHIN REAR ANTENNA STATUS FRONT ANTENNA OFF STATUS REAR ANTENNA STATUS REAR ANTENNA SELECTED STATUS REAR ANTENNA	A Single SUI Interaction
Type 12	STATUS REAR ANTENNA SELECTED STATUS FRONT ANTENNA OFF STATUS REAR ANTENNA	A single Hardware Interaction (Only seen when the other antenna was active)

Table 5.2 Summary of logged message structures that contain usage information

5.3 Data Visualizations

Once the raw data file recoding process was complete, the result was a series of concise data files with comparable information, one for each day of raw data logging for each officer plus an extra file for each officer that contained the total of all their days of interaction logging. In other words, the extra file was a concatenation of each individual officers' usage activity over the entire length of their participation in this research. That means that for this research there were a total of forty four recoded data files created. To give some sense of the amount of information used during this research, the statistics of these files is shown in Table 5.3.

	Officer #1	Officer #2	Combined
Total Days of Recoded Data	23 days	19 days	42 days
Total Files of Recoded Data	23 files + 1	19 files + 1	44 files
Total Amount of Recoded Data	380 kB	135 kB	515 kB
Mean Daily Recoded File Size	17 kB	7 kB	12 kB
Total Number of Interactions	4938	1793	6731
Mean Daily Interaction Count	214	94	160

Table 5.3 Summary of recoded data statistics from two deployed police cruisers

The next step in the post processing was to develop a MATLab-based visualization tool that could present the data graphically in order to accentuate the manner in which the Project54 system was used to carry out various user interactions. The different visualizations created by this tool are separated into so-called cells, a feature available in MATLab release R2006a. Using cells, the user can generate visualizations one at a time, which is faster than waiting for all the visualization figures to be produced. Also, the cells contain customization options such as creating visualizations that focus on what interactions take place while a police officer is driving.

The focus was not only on making sure data visualizations could be created automatically using this tool, but also to investigate which data visualizations were preferable. To that end, samples of the visualizations contained in the remainder of this chapter were shown to eight different research assistants within the Project54 design team, ranging in experience from several months to over three years. Not only did their feedback (which will be discussed during the introduction of each different set of visualizations) provide insight into which graphical representations were preferred, but also their input lead to several beneficial changes in the visualizations themselves. All of the visualization examples shown in this chapter are divided into two sections: information recorded while the police cruisers were moving and information recorded while the police cruisers were stopped. This information was received via GPS velocity data during the logging process. However, the police cruisers were not always within GPS signal reception areas so the counts of interactions and interface usage may not, in some cases, accurately reflect the total amount of activity going on within the vehicles. That being said, all identifiable user interactions were recorded during the logging process, regardless of whether or not the vehicle speed was available. The Logger noted the lack of vehicle speed data whenever such cases were present.

The first visualizations covered are the histograms. The histograms were included as an example of a visualization engineers were likely to feel comfortable dealing with, due to the likelihood of having come across them many times in the past. It came as no surprise that the engineers who viewed these visualizations found that, while the histograms tended to be the most "boring" of the visualizations, these presentations also required the least amount of explanation or time to understand. The main drawback to the histograms is that they can become hard to read, as is the case in Figure 5.3 through Figure 5.6. These four figures show interface usage while driving and while stopped (Figure 5.3 and Figure 5.4, respectively) as well as general device usage (lights, radar, radio, and records) while driving and while stopped (Figure 5.5 and Figure 5.6, respectively) from the two police officer participants. The data was averaged together over the course of all the days of their participation. That usage information is then presented as if it all transpired over the course of one day (one twenty-four hour period). While the information only covers a very small sample set, certain trends do tend to emerge. Among these trends are that the time corresponding to the evening commute tends to see a rise in activity, the GUI is the most frequently used interface while the vehicle is stopped, and the officers tend to spend their time running records checks when they are stopped more than any other in-vehicle activity.

The spikes shown in Figure 5.3 and Figure 5.5 are not anomalies. The spike at "Hour 1" (in both figures) is a result of one of the officers running records checks that made heavy use of the "Scroll Up" and "Scroll Down" GUI buttons during one of his shifts. The spike at "Hour 24" (in both figures) indicates that that same officer was turning his front antenna on and off repeatedly during the same shift that produced the "Hour 1" spike. This procedure is called "Hold Mode" and is done to avoid tipping-off drivers who may have radar detectors. The spikes shown in Figure 5.4 and Figure 5.6 are not anomalies either. All the GUI spikes (in both figures) are a result of one of the officers entering information into text fields during records checks over several shifts. The records checks also involved use of the "Scroll Up" and "Scroll Down" GUI buttons.

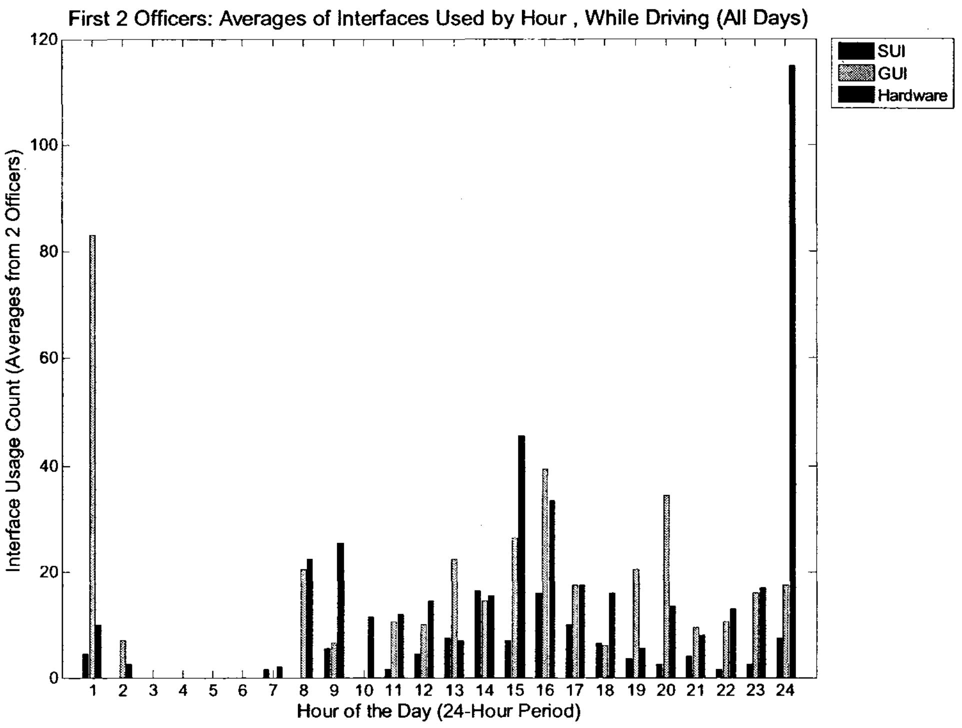


Figure 5.3 A histogram of interface usage while the police were driving

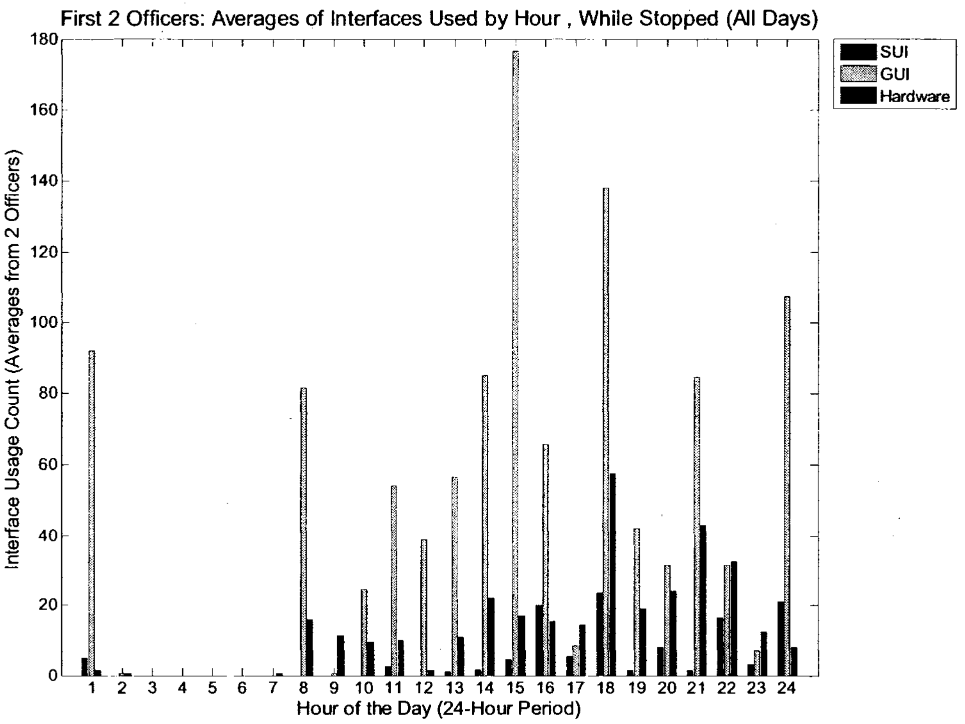


Figure 5.4 A histogram of interface usage while the police were stopped

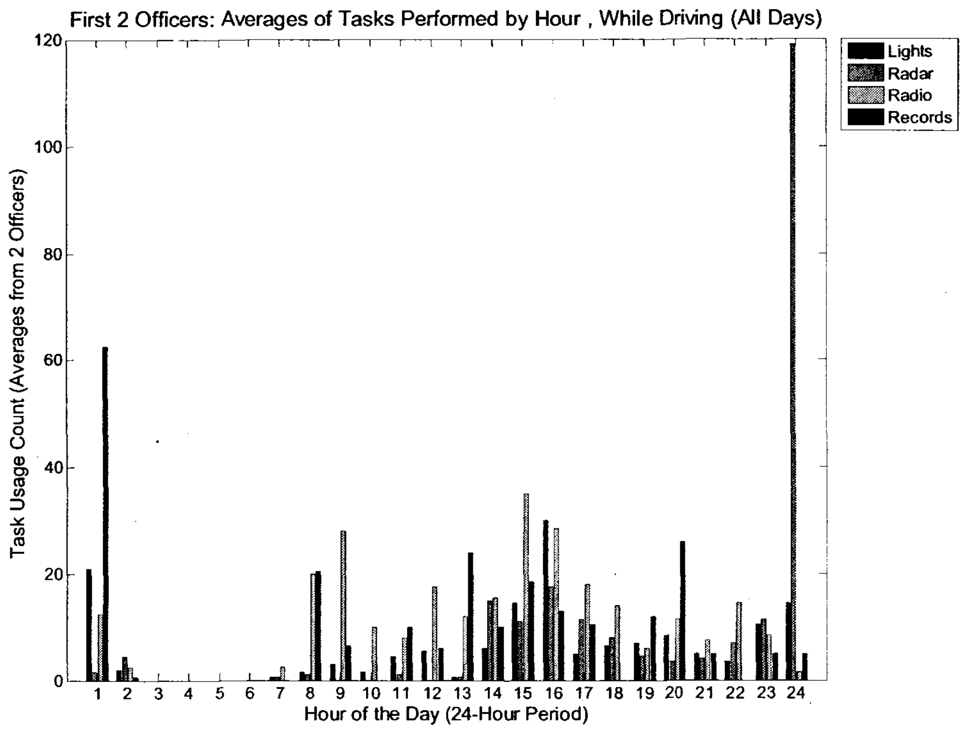


Figure 5.5 A histogram of interactions while the police were driving

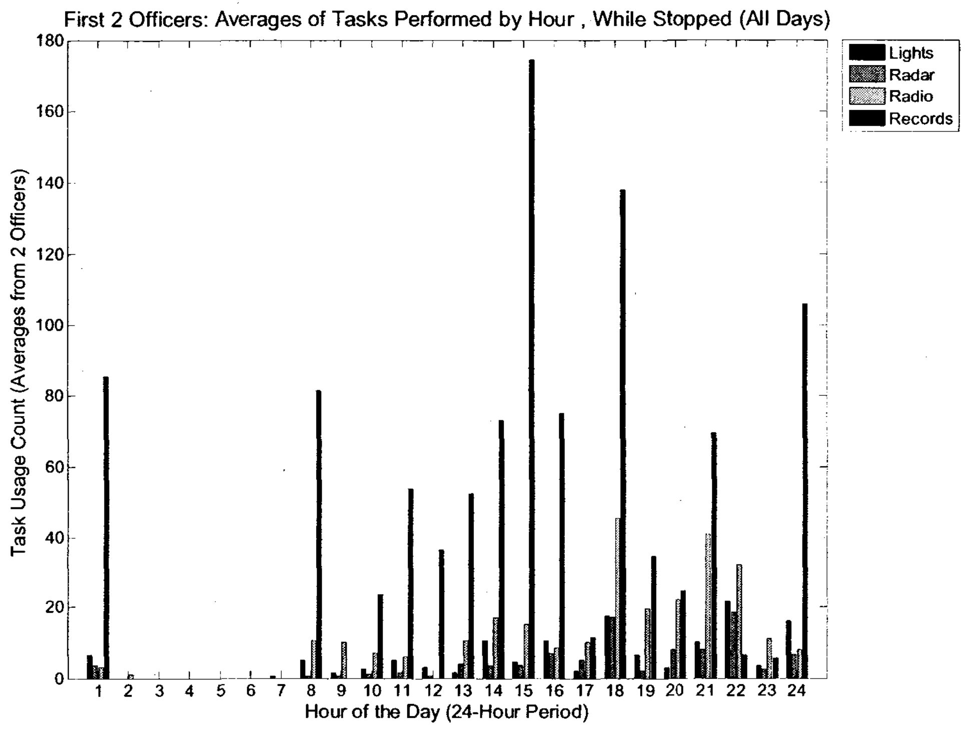


Figure 5.6 A histogram of interactions while the police were stopped

The interface usage plots shown in Figure 5.7 and Figure 5.8 represent the average usage (averaged by officer) of lights, radar, radio, and records for both participants over the course of all days in which data was collected. The rest of the histograms presented in this thesis act as subsets of these two figures.

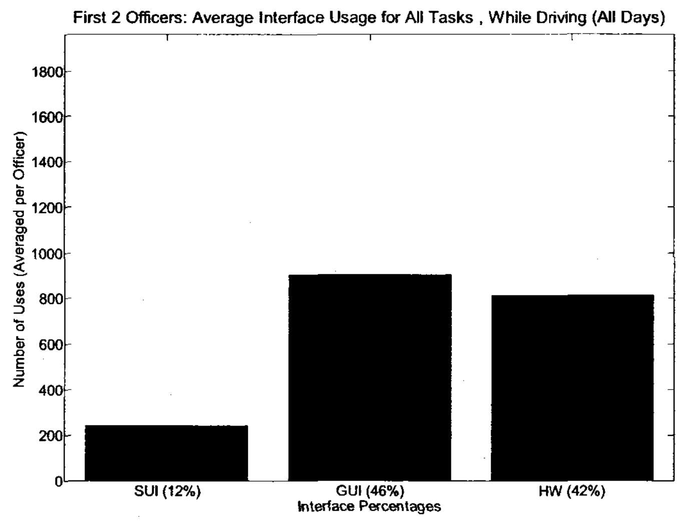


Figure 5.7 A histogram of in-vehicle interface usage while driving

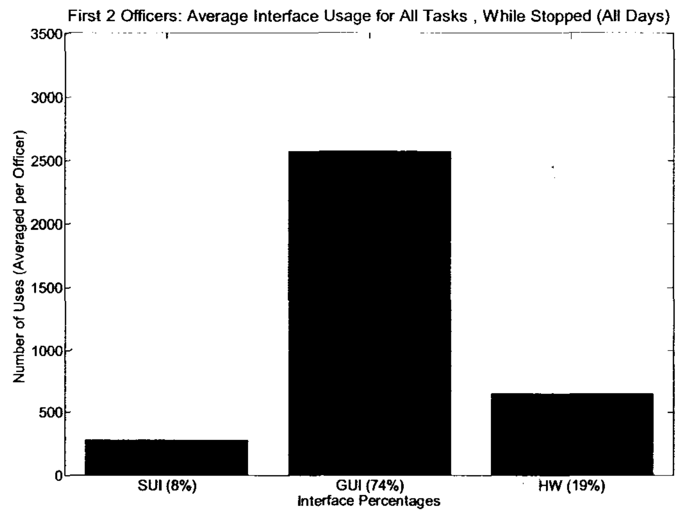


Figure 5.8 A histogram of in-vehicle interface usage while parked

The next set of histograms (Figure 5.9 through Figure 5.16) demonstrates the ability of the analysis program to generate results capable of examining any link between the device to be controlled and the interface used to control that device. The plots show the interface usage by task (lights, radar, radio, and records). Each task has been further divided into two subsets indicating the difference in interface usage while the officers were driving or stopped when performing their tasks. For example, there are two figures that show the average of the two officers' interface preferences for controlling lights – one figure show preferences while driving and the other show preferences while stopped. One of the interesting results of the light bar control plots, in particular, is that they tend to go against what the police officers stated on their questionnaires. More specifically, the officers said they preferred to use the original hardware controls more than Project54 to operate the lights. In general, the SUI tended to be the least popular method for controlling devices, while the GUI tended to be the most popular, regardless of whether or not the officer was driving at the time. The exception appears to be with the radio. By and large the radio control head was used to execute radio functions, however nearly all of the radio operations involved changing the volume, which will be shown later.

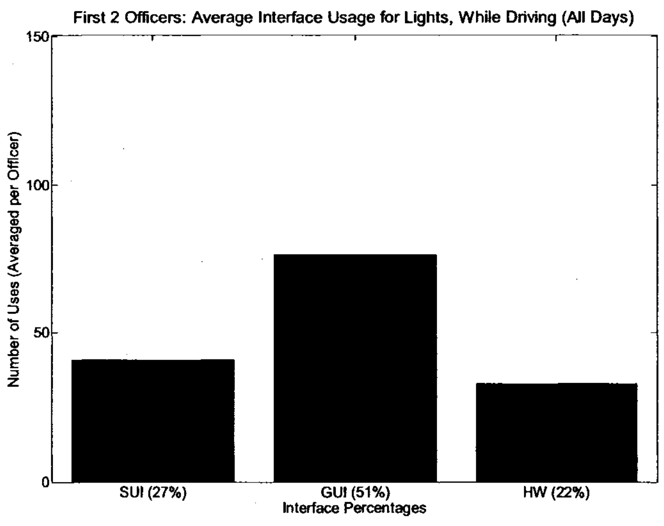


Figure 5.9 A histogram of in-vehicle interface preferences for controlling the light bar, while driving

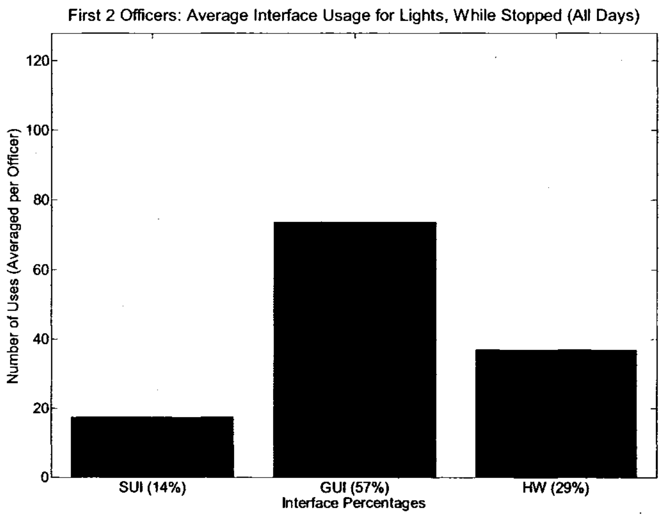


Figure 5.10 A histogram of in-vehicle interface preferences for controlling the light bar, while stopped

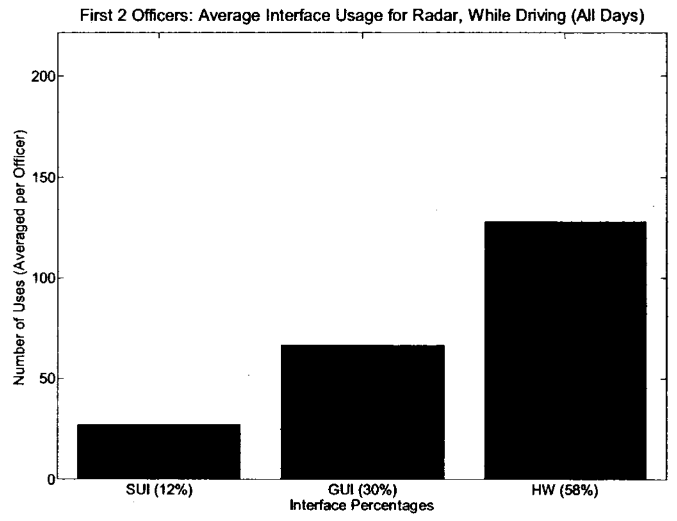


Figure 5.11 A histogram of in-vehicle interface preferences for controlling the radar, while driving

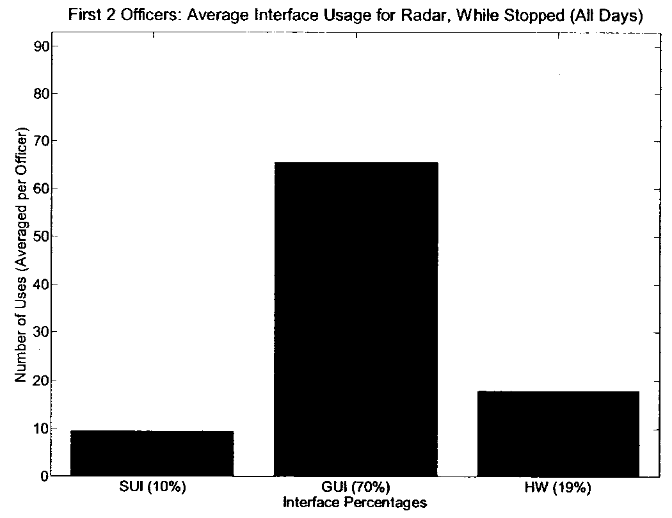


Figure 5.12 A histogram of in-vehicle interface preferences for controlling the radar, while stopped

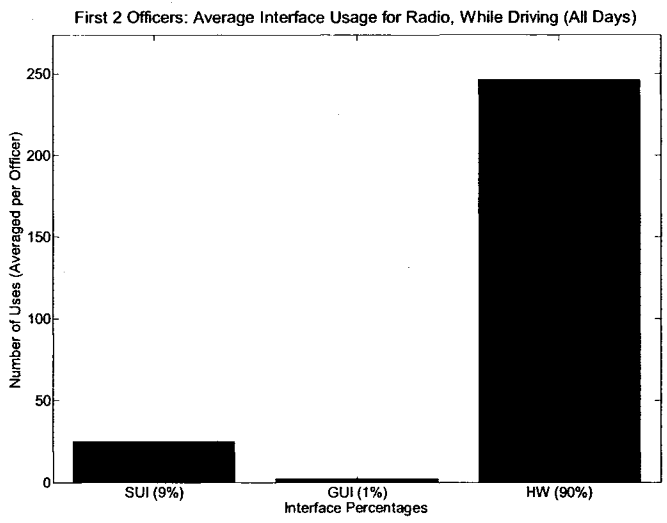


Figure 5.13 A histogram of in-vehicle interface preferences for controlling the radio, while driving

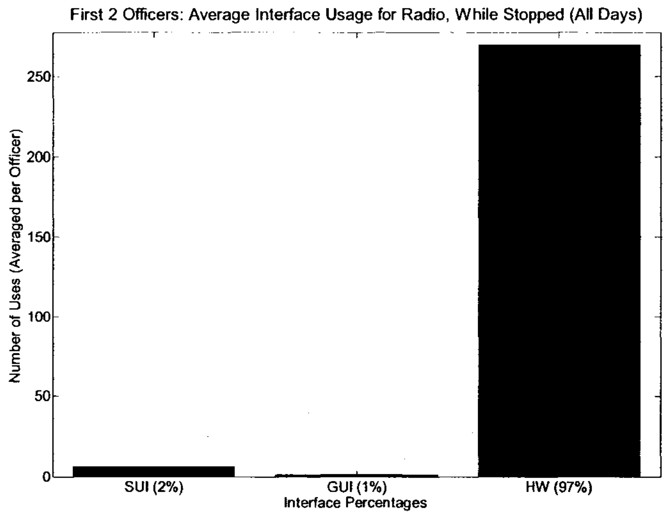


Figure 5.14 A histogram of in-vehicle interface preferences for controlling the radio, while stopped

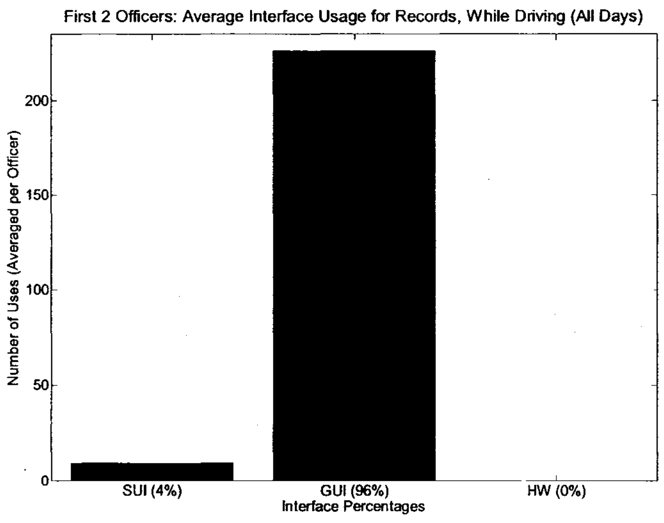


Figure 5.15 A histogram of in-vehicle interface preferences for performing records checks, while driving

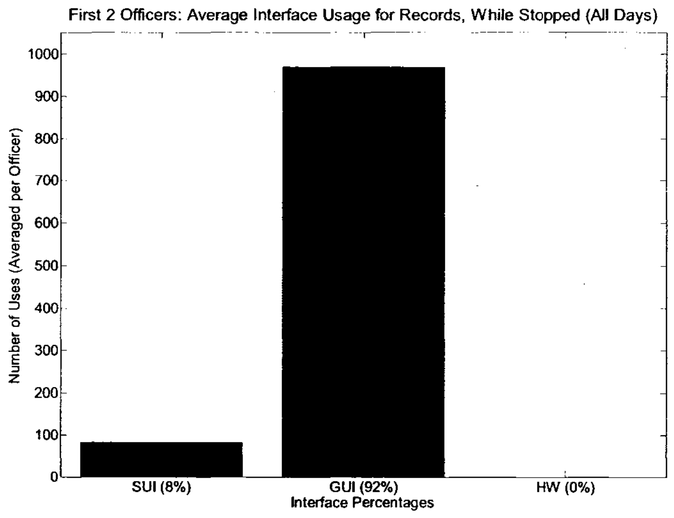
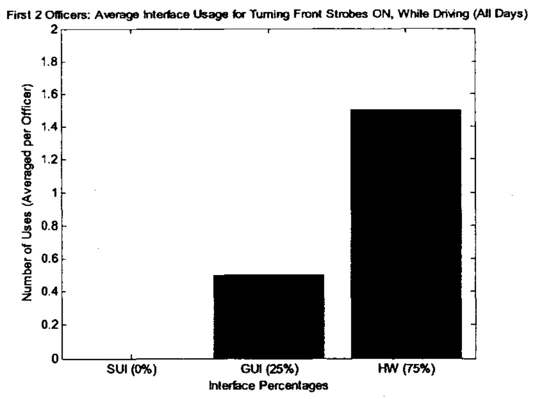
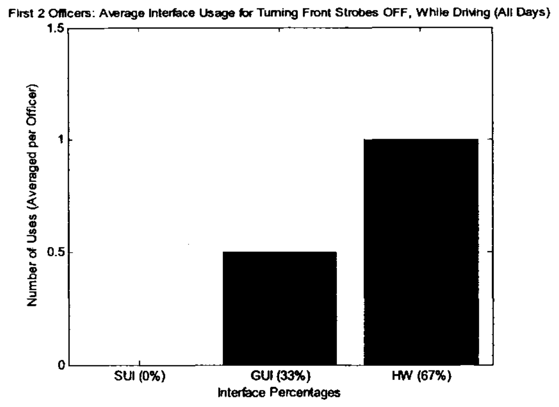


Figure 5.16 A histogram of in-vehicle interface preferences for performing records checks, while stopped

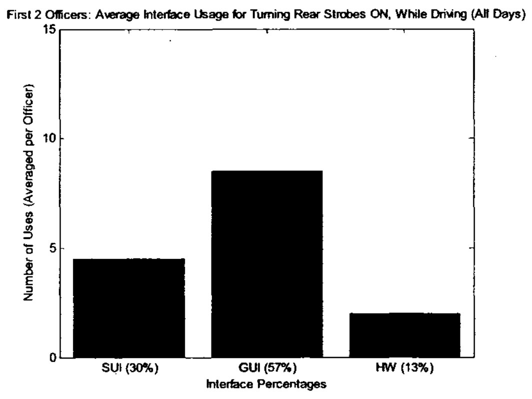
Due to what appeared to be a discrepancy between how the officers stated they preferred to operate their light bar assemblies and what the recorded data indicated they preferred, the next set of histograms takes a closer look at the lights usage (Figure 5.17 through Figure 5.20). The only apparent link between interface usage and whether or not the officer was driving is that the officer tended to rely more on the hardware controls and less on the SUI, when stopped as opposed to driving. The GUI was used consistently both while driving and while stopped. The results from the officers' questionnaire responses indicated that they use the original hardware controls to turn light bar functions on and Project54 interfaces to turn off light bar functions. However, the available usage information from these two officers conflicted with their questionnaire responses.



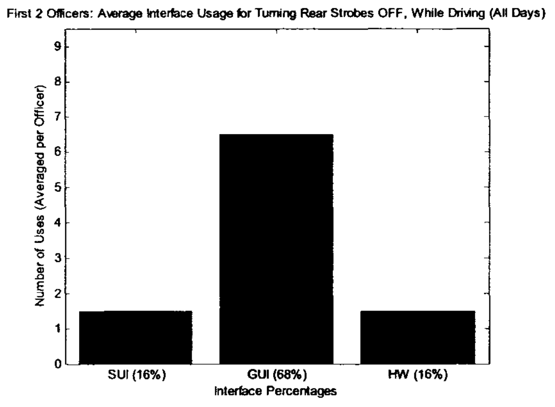
a)



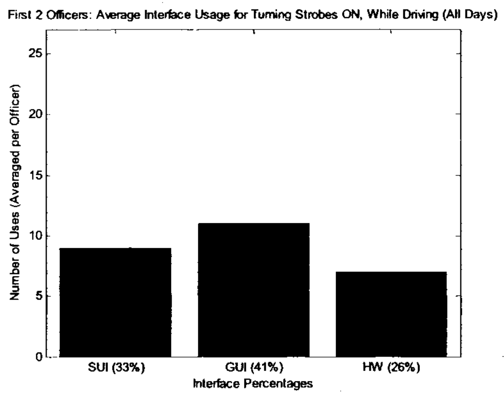
b)



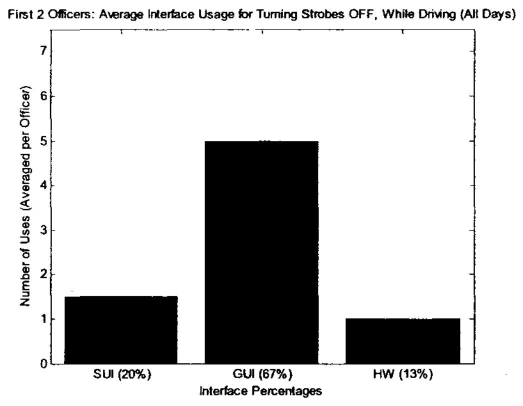
c)



d)

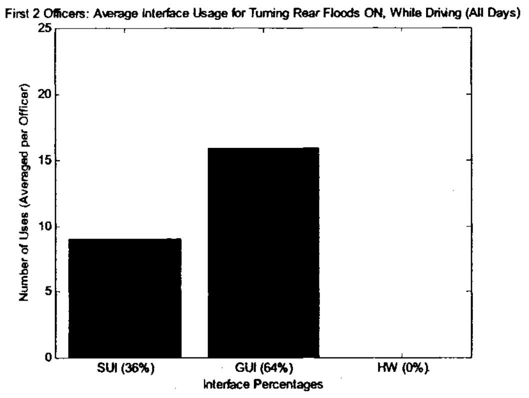


e)

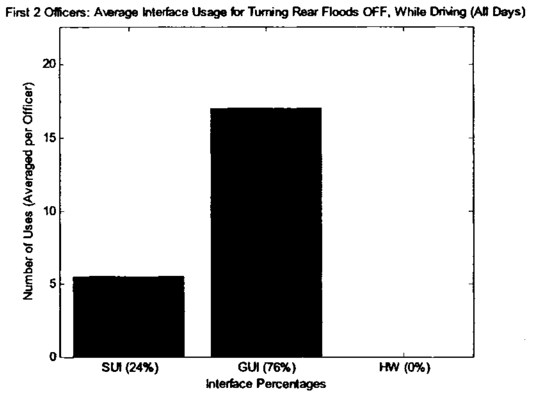


f)

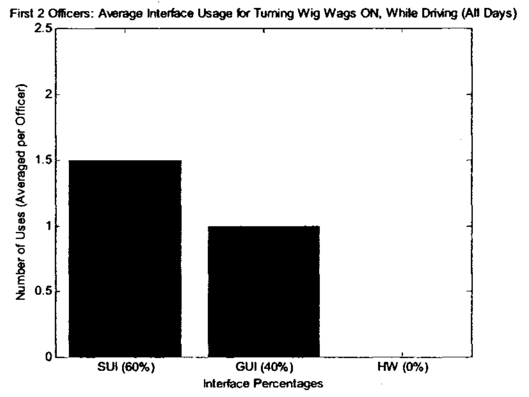
Figure 5.17 Histograms of in-vehicle interface preferences for controlling various strobes functions, while driving



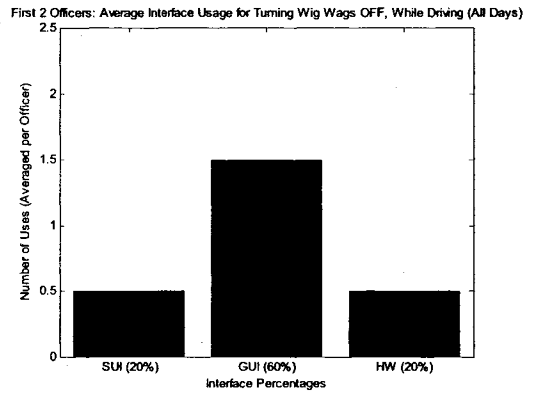
a)



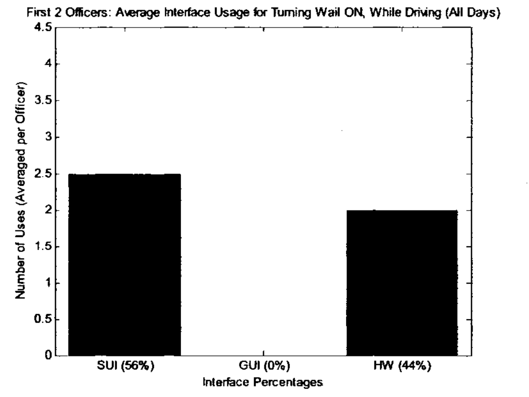
b)



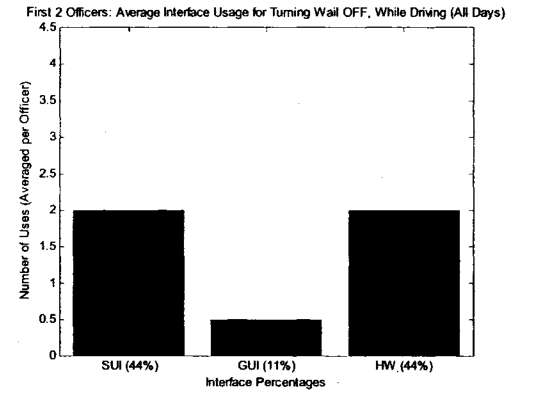
c)



d)

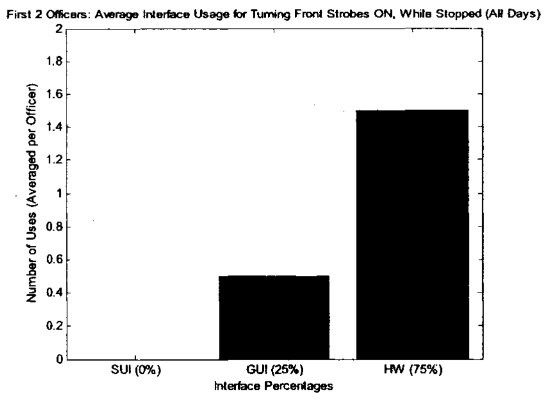


e)

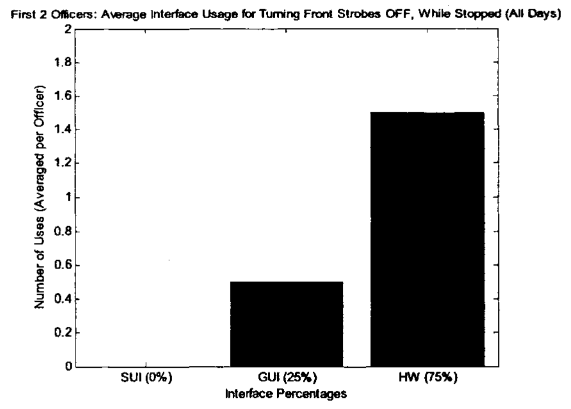


f)

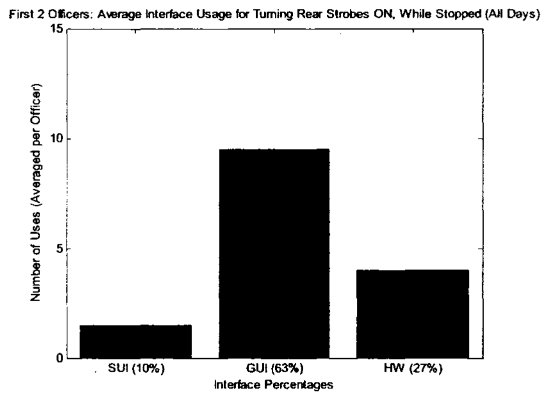
Figure 5.18 Histograms of in-vehicle interface preferences for controlling various non-strobes light bar functions, while driving



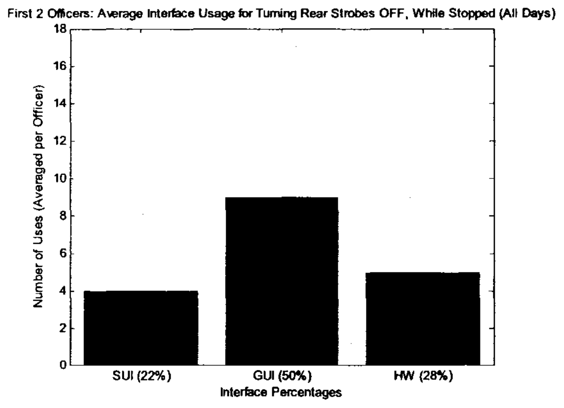
a)



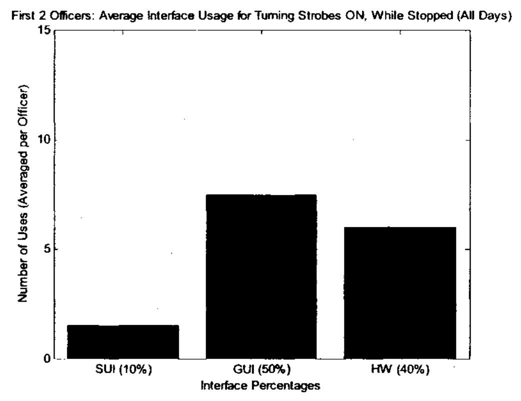
b)



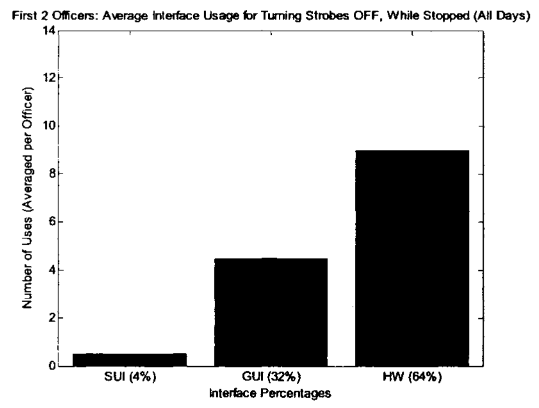
c)



d)



e)



f)

Figure 5.19 Histograms of in-vehicle interface preferences for controlling various strobes functions, while stopped

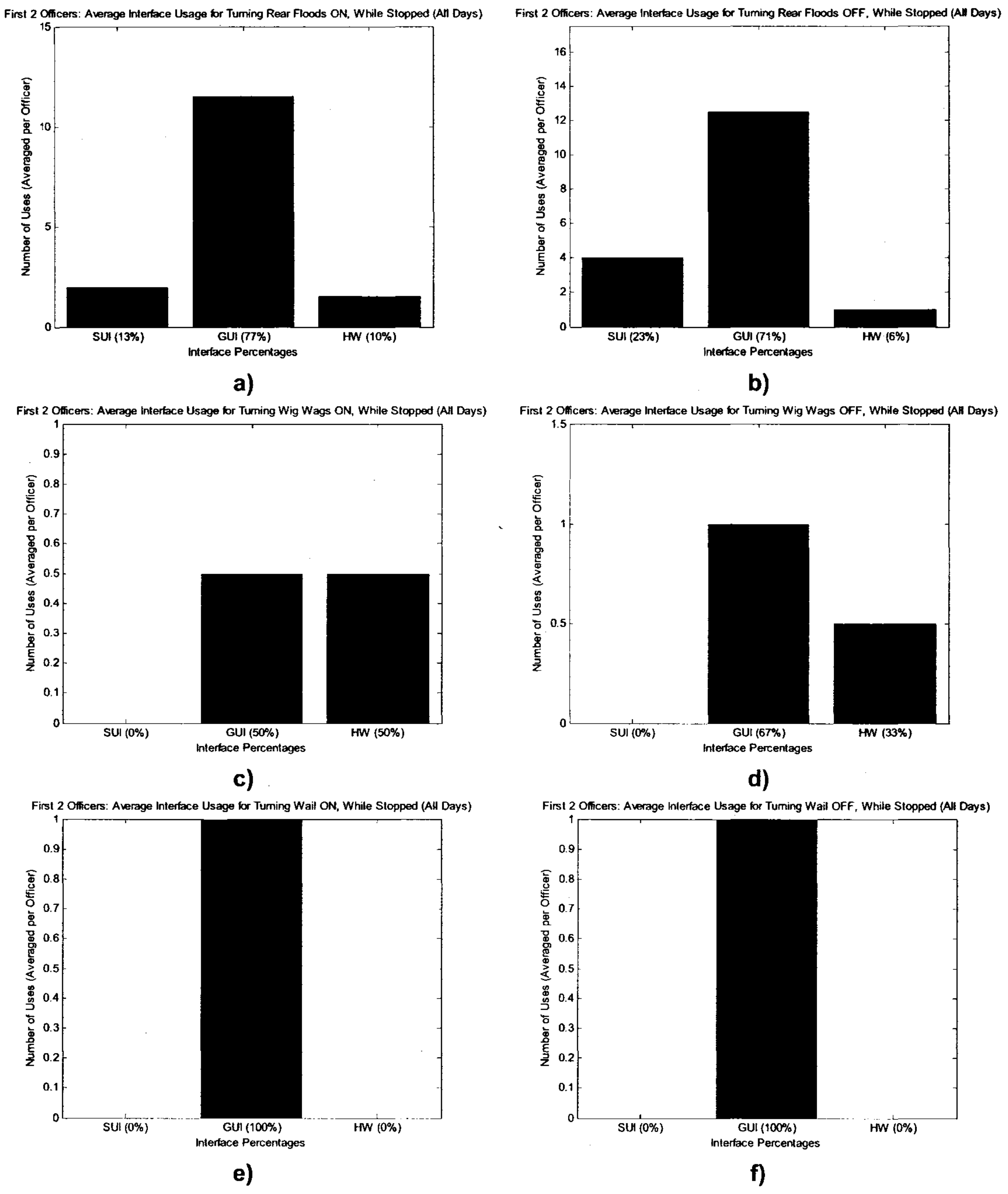
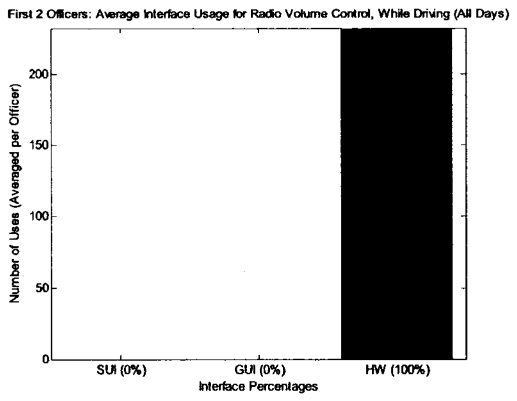
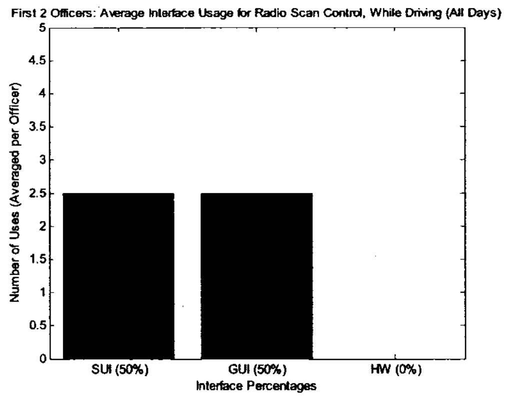


Figure 5.20 Histograms of in-vehicle interface preferences for controlling various non-strobes light bar functions, while stopped

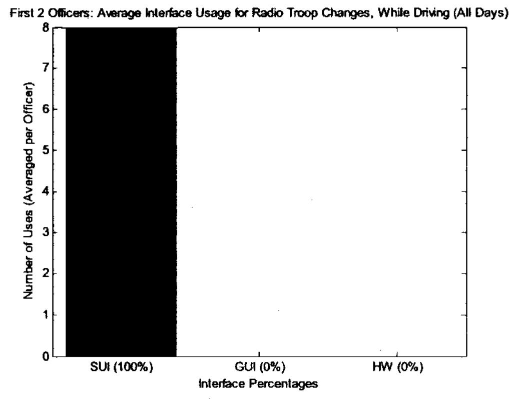
As stated earlier in this chapter, the radio appeared to be the only device that the officers tended to prefer controlling with the original hardware control head (See Figure 5.21 and Figure 5.22). Further investigation revealed that, far and away, the radio control head was used to change the radio volume. The other radio functions tended to be controlled via SUI just as much as via the hardware control head while the vehicles were moving. It appeared as though the officers' interface preferences were task-driven. In other words, they prefer to use a particular interface to perform a particular task, regardless of whether or not they are driving at the time.



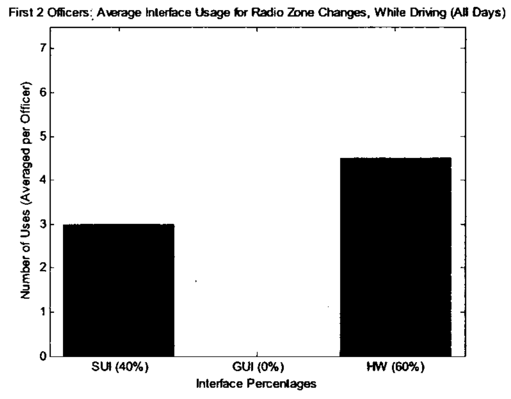
a)



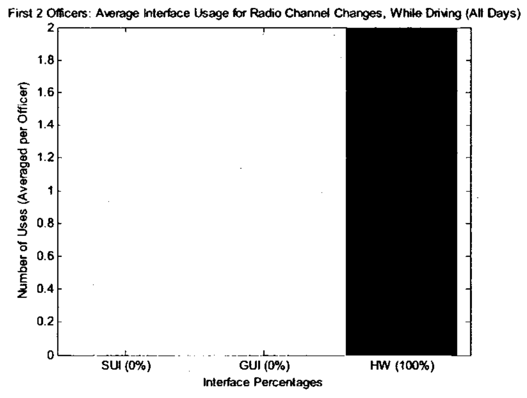
b)



c)

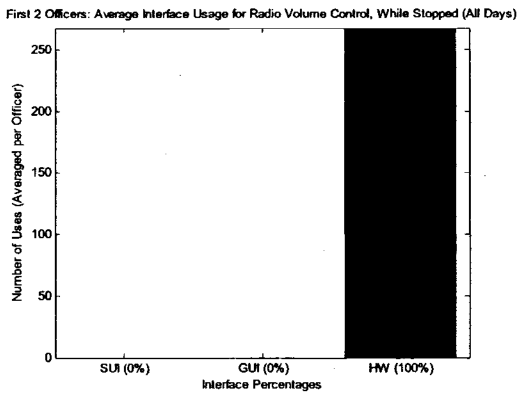


d)

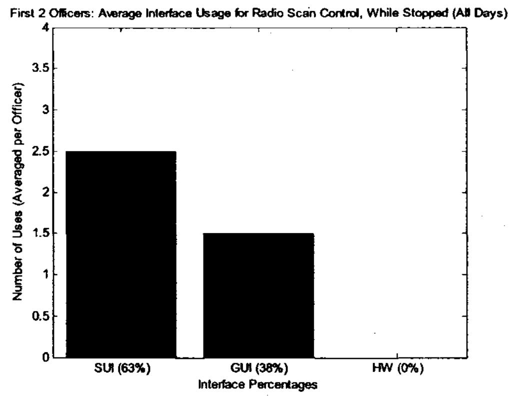


e)

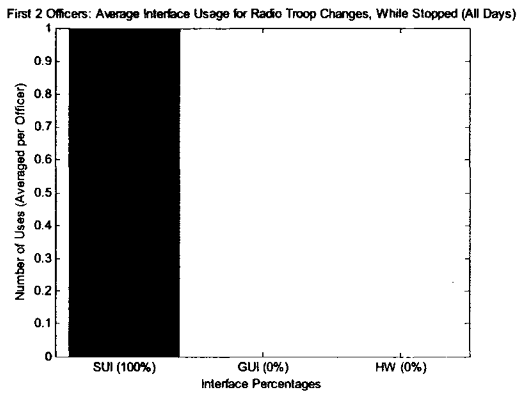
Figure 5.21 Histograms of in-vehicle interface preferences for controlling various radio functions, while driving



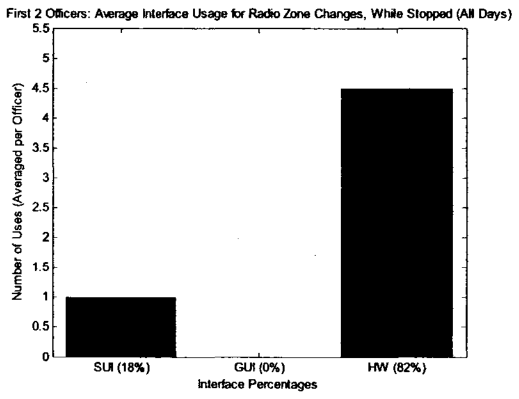
a)



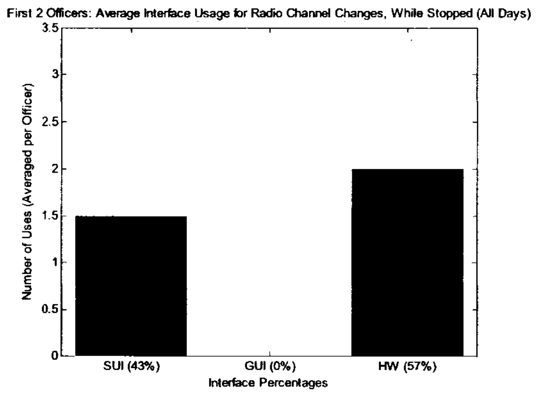
b)



c)



d)



e)

Figure 5.22 Histograms of in-vehicle interface preferences for controlling various radio functions, while stopped

More than the histograms, the remaining data visualizations truly utilize different techniques, including coloring, location, and object size, in order to convey the analysis results. With the exception of the 3D plots (the last visualizations described), the remaining visualizations were geared more towards law enforcement officials and other non-technical observers. This is not to say they do not convey the same level of information, however. To the contrary, the remaining visualizations were generated using the same information as the histograms, merely expressed in a different fashion.

The next set of visualizations (Figure 5.23 and Figure 5.24) was ranked fourth out of the five different visualizations, by the Project54 research assistants who reviewed the images. This was mostly due to two factors. First, the original image contained an extra "button" placed between the "Project54" button and the "Hardware" button, which only served to confuse observers as to its meaning. The second complaint was that these two images present the same information as the next set of images (Figure 5.25 and Figure 5.26), but with less information resolution regarding what Project54 interfaces were preferred. Still, Figure 5.23 and Figure 5.24 are useful in that they may immediately answer the questions, "Do police officers tend to prefer Project54 interface controls over the original hardware controls?" This is because the data that generates these visualizations takes into account the usage preferences for all tasks done by both officers, and for all days on which data was collected. The more the officers used Project54 to perform tasks, the more the colored police officer silhouette in the figure would turn green. This color indicator also had an arrow that pointed to the region of the color meter that corresponded to the officers' level of Project54 usage vs. hardware control usage. The result appears to be that for the two participating officers, Project54 was the preferred method for device control, based on available usage records.

In Figure 5.23 and Figure 5.24, the officer's silhouette changed color based on the average of interface usage for both participants. In other words the averages for SUI, GUI, and hardware usage were calculated in the MATLab visualization program. With those numbers on hand, the program was then able to treat the SUI, GUI, and hardware values as weights that affected how much red, green, and blue coloring was added to the silhouette. In order to actually add the color the x-y coordinates for a rectangle large enough to just fit the officer's silhouette were used in the program to define the region to which color was to be added. The original silhouette color was red with a black outline. The only other color present in the defined rectangle was white. The program ignored pixels within the defined rectangle that were either white or black and augmented the rest according to:

$$\begin{aligned} pixel(x,y,n) &= 255 - [255 \times (SUI + GUI)] + (255 \times hardware) \\ pixel(x,y,3) &= 0 \end{aligned}$$

The value *pixel* refers to the image pixel whose color is to be augmented. The values x and y were the coordinates within the defined rectangle at which color was to be added. The value n was either 1 or 2, and represented the red or green layer of the original image, respectively. In order to avoid blue colors within the image, n = 3 was set to zero (black). The value *SUI* is the count of times the SUI was used, averaged over both participants. The value *GUI* is the count of times the GUI was used, averaged over both participants. The value *hardware* is the count of times hardware was used, averaged over both participants. MATLab defines the maximum amount of color to be 255, so (255,255,255) is the red-green-blue (RGB) representation for white. For example, to add green color to the figure (n =2), the equations starts from an assumed full-green color value (255) and subtracts off an amount of green that has been weighted by the sum of SUI and GUI interactions. Then the program adds back an amount of green, weighted by the total number of hardware interactions.

The other addition to the original image, the arrow, was created by using MATLab's arrow annotation. The arrow was only free to vary about the y-axis, where it was bounded from a minimum value of 0.25 to a maximum value of 0.8. The values for these bounds were derived based on the notion that the total y-axis range of the image went from 0 to 1. Within the y-axis bounds, the arrow was free to move up or down based on:

$$arrow\_head = \frac{1}{2} - \frac{hardware}{2} + \frac{SUI + GUI}{2}$$

The value *arrow_head* refers to the location on the image of the head of the pointer arrow. The value *SUI* is the count of times the SUI was used, averaged over both participants. The value *GUI* is the count of times the GUI was used, averaged over both participants. The value *hardware* is the count of times hardware was used, averaged over both participants. In words, the arrow head started at the middle of the image (0.5) and moved up based on the amount of Project54 (SUI and GUI) usage and down based on the amount of hardware usage. Dividing both usage numbers by two was done to reduce the affect of the usage values on the arrow head's overall displacement from the middle of the image.

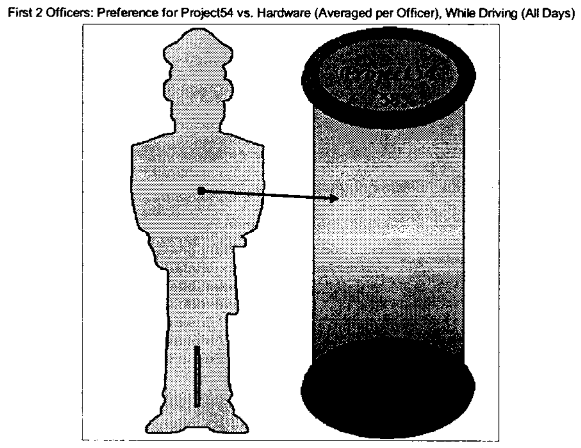


Figure 5.23 Use of color to contrast preference for Project54 vs. original controls, while moving

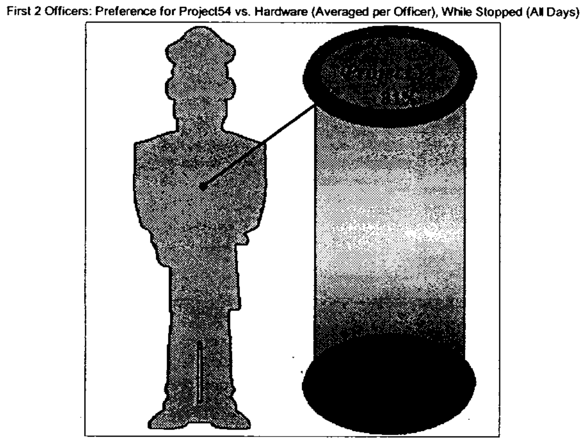


Figure 5.24 Use of color to contrast preference for Project54 vs. original controls, while stopped

Like Figure 5.23 and Figure 5.24, the images shown in Figure 5.25 and Figure 5.26 use color to represent interface preferences. Unlike the former two images, the latter two break down Project54 interfaces into SUI and GUI components. For this reason more than any other, those who reviewed the visualizations found this set of images to be the second-best. The image is straight-forward to understand – the police officer silhouette in the upper right-hand corner changes color depending on the relative popularities of the three interfaces. The black dot located within the color triangle will gravitate towards the most popular interface as well. The results of this analysis indicate that the two police officers tended to prefer the GUI over the other two interfaces, especially while stopped. SUI usage actually increased while the officers were driving, which is the ideal scenario.

The technique used to change the officer's silhouette color in Figure 5.25 and Figure 5.26 was very similar to the technique described for adding color to Figure 5.23 and Figure 5.24. The only change came in the equations used to alter the color, which were:

$$\begin{aligned}
pixel(x,y,1) &= 255 + (255\times SUI) - (255\times GUI) - (255\times hardware)\\
pixel(x,y,2) &= 255 - (255\times SUI) - (255\times GUI) + (255\times hardware)\\
pixel(x,y,3) &= 255 - (255\times SUI) + (255\times GUI) - (255\times hardware)
\end{aligned}$$

The value *pixel* refers to the image pixel whose color is to be augmented. The values x and y were the coordinates within the defined rectangle at which color was to be added. The value 1, 2, or 3 represents the red, green, or blue layer of the original image, respectively. The value *SUI* is the percentage of times the SUI was used, averaged over both participants. The value *GUI* is the percentage of times the GUI was used, averaged over both participants. The value *hardware* is the percentage of times hardware was used, averaged over both participants. Since the SUI button on the original image is red, SUI has a positive red color shift and negative green and blue color shifts. GUI and hardware usage obey the same trend, according to the set of equations.

The addition of the black dot within the original image was created by using MATLab's ellipse annotation. The image length is normalized so that the point (0, 0) represents the lower left-hand corner of the entire image and the point (1, 1) represents the upper right-hand corner of the image. The starting location of the dot is the center of the image, (0.5, 0.5). The dot is free to move within a defined triangular region that is located within the color-gradient triangle, with bounds at the three vertices, (0.33, 0.3), (0.49, 0.67), and (0.65, 0.3). These values correspond to locations within the original image, as defined by normalized x- and y-axes. Within the bounded "inner" defined triangle, the dot was free to move around according to:

$$x\_coord = 0.5 + \frac{hardware}{4.5} - \frac{GUI}{4.5}$$
$$y\_coord = 0.5 + \frac{SUI}{4.5} - \frac{GUI}{4.5} - \frac{hardware}{4.5}$$

The value *x_coord* refers to the dot's x-coordinate. The value *y_coord* refers to the dot's y-coordinate. The value 0.5 in each of the equations refers to the starting point for the x- and y-coordinates. The value *SUI* is the percentage of times the SUI was used, averaged over both participants. The value *GUI* is the percentage of times the GUI was used, averaged over both participants. The value *hardware* is the percentage of times hardware was used, averaged over both participants. In words, the dot's x-coordinate shifts to the right (positive x-direction) based on hardware usage and to the left based on GUI usage, according to their x-axis locations within the colored triangle. Similarly, the y-coordinate shifts up (positive y-direction), based on SUI usage and down, based on both GUI and hardware usage.

For example, if an officer used the SUI for 50% of the total interactions performed, the GUI for 30% of the total interactions, and hardware for the remaining 20% of the total interactions, the dot would be located at the coordinates (0.47, 0.5), according to:

$$x\_coord = 0.5 + \frac{0.2}{4.5} - \frac{0.3}{4.5}$$
$$y\_coord = 0.5 + \frac{0.5}{4.5} - \frac{0.3}{4.5} - \frac{0.2}{4.5}$$

For the scenario in which an officer used each interface for a third of the total interactions, the dot would be located at the coordinates (0.5, 0.43). The dot does not remain at the starting location due to the equations governing its movement. However, the importance of the dot's movement is that it gives an intuitive interpretation for how the police officers perform interactions.

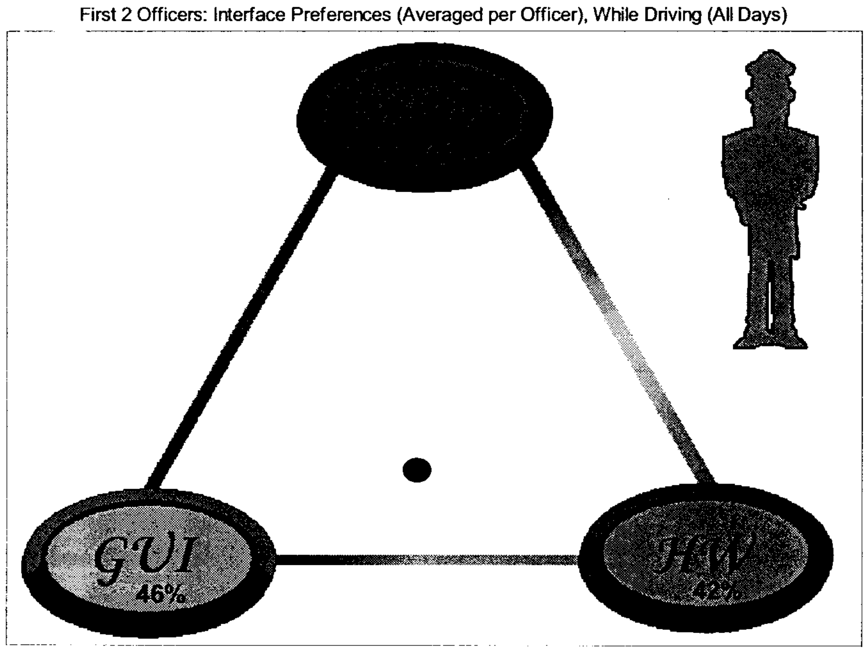


Figure 5.25 Use of color and location to indicate interface preferences, while moving

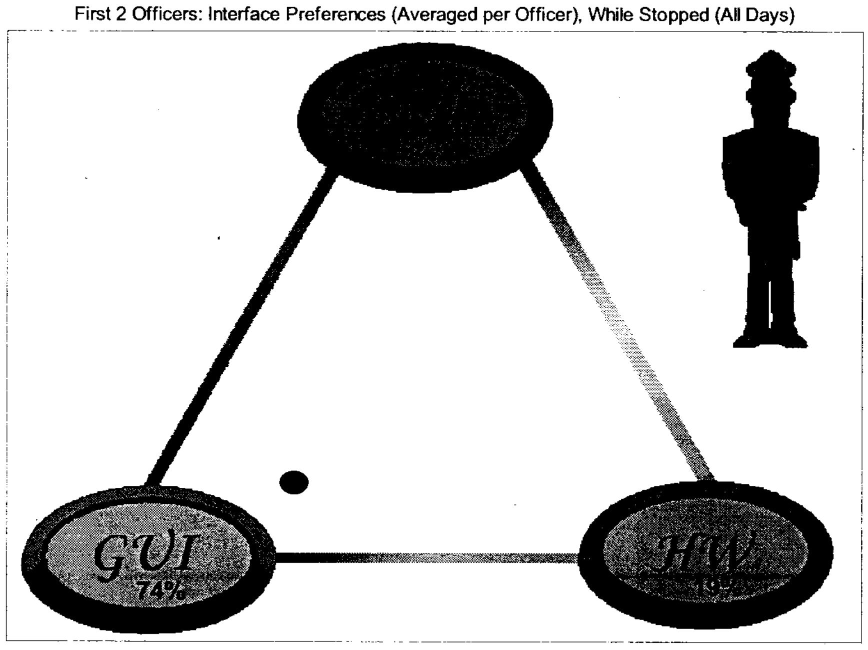


Figure 5.26 Use of color and location to indicate interface preferences, while stopped

The fourth set of visualizations, shown in Figure 5.27 and Figure 5.28 were the overwhelming favorite of those Project54 employees who viewed the five different data presentation styles. The two images show the results of a button-press analysis and a speech command analysis (respectively) performed while the active window was the Patrol Screen. The background of the image is an actual screen shot from the Project54 Patrol Screen that matches the Patrol Screen from the two officers' police cruisers. To show the frequency with which buttons were pressed or speech commands were issued, the visualization program gave the columns of buttons different intensities of color – blue for button presses and red for speech commands. The amount of color was determined within the visualization program by normalizing the number of count of button presses (or speech commands) for each button. Once the program normalized the values they were scaled up by a factor of 85 in order to allow 85 different color "chunks" to be defined for button-coloring purposes. Remember that the maximum amount of color is 255; dividing the maximum amount by 85 means that each color "chunk" is capable of changing the image's button color by 3 color units. Before the program applied colors to the buttons, it first set all the buttons to white. This was done mainly to aid observers in detecting unused buttons but it also simplified the coloring algorithm, which was:

$$button(i,n) = button(i,n) - \#ColorChunks(k)$$

The value *button* refers to a particular button on the Project54 GUI window; both *button* references deal with the *same* window button. The values for *i* ranged from 1 to 18 and corresponded to the number of buttons on the Project54 window. The values for *n*, used again to represent the RGB image layers, were 1 to 3. The values for *k* represent red and green for the GUI case (green and blue for the SUI case). The value #ColorChunks refers to the number of discrete color "chunks" used to color the buttons, as described earlier in this paragraph. In the GUI usage case, the more a button was pressed the more red and green were subtracted, leaving only blue. Similarly, in the SUI usage case, the more a speech command was used the more green and blue color was subtracted, leaving only red. The color bar next to each screen shot relates the amount of blue or red color to a percentage of button presses or speech commands, respectively. For example, approximately 18% of the total button presses were used to operate the Rear Floods, while the Patrol Screen was the active window. The images indicate a preference for using Project54 commands to control the strobes functions as well as the antenna arrays. There does not appear to be a bias towards one interface over the other in the button coloring, but that bias becomes clearer by noticing that button presses were performed 793 times, as opposed to speech commands, which were only performed 253 times within the same time frame.

Officer #1: Patrol Screen Button Usage (All Days)
Color Bar Indicates Percentage of Total Button Presses (Total Presses = 793)

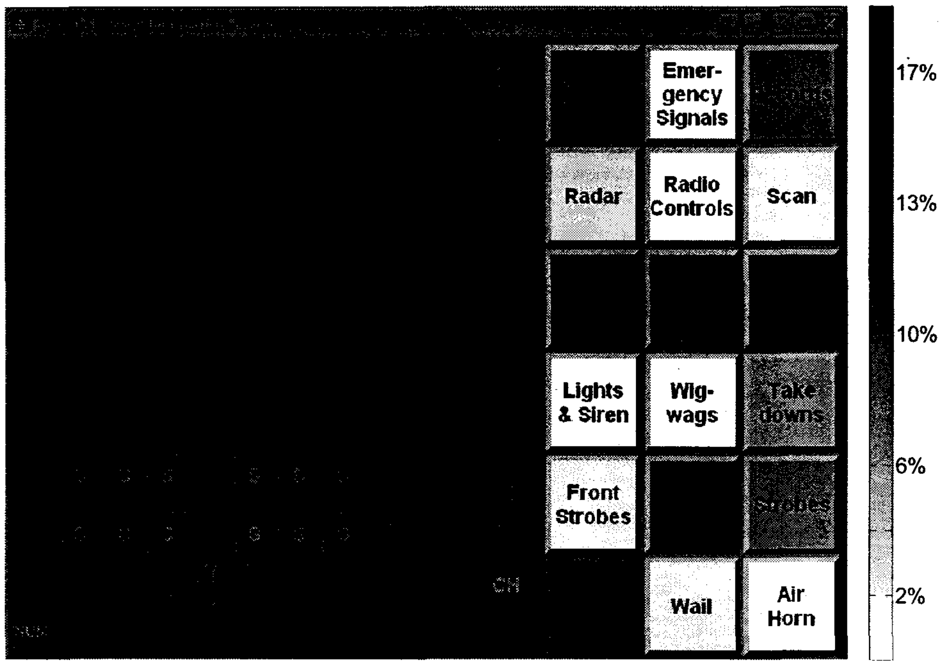


Figure 5.27 Use of screen shot and color overlays to indicate button preferences

Officer #1: Patrol Screen Speech Command Usage (All Days)
Color Bar Indicates Percentage of Total Commands (Total Commands = 253)

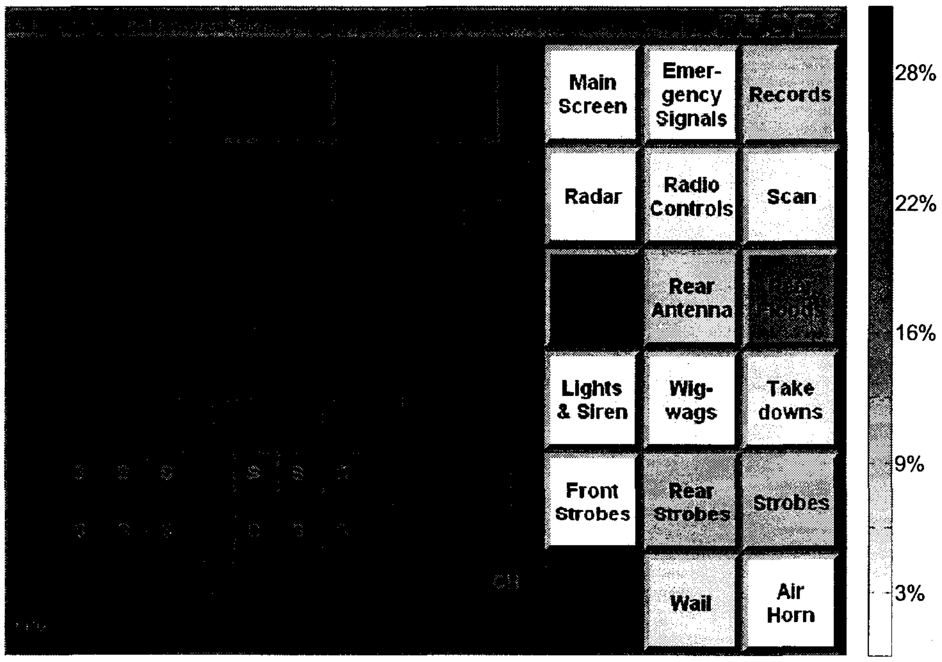


Figure 5.28 Use of screen shot and color overlays to indicate speech command preferences

The final two data visualizations to discuss, the 3D plots shown in Figure 5.29 and Figure 5.30 were overwhelmingly the least favorite data representation style shown to the review group. The major problem was being able to read the results of the representation, as it can be very challenging to represent three-dimensional data on a two-dimensional medium, such as paper. To make matters more interesting, this visualization actually contains four dimensions. The data shown in the figures is, again, averaged over the number of officers (in this case, 2). The percentage of each device usage that was performed by the SUI is plotted on the x-axis. The percentage of hardware used is plotted on the y-axis. The percentage of the GUI used is plotted on the z-axis. Therefore, the radar was controlled approximately 12% of the time with the SUI, 58% of the time with the radar remote control, and 30% of the time with the GUI. If the plot only contained these relative percentages there would be no way of knowing, overall, how much each device was used. It is for this reason that the fourth dimension, the sizes of the cubes themselves, is useful. The larger the cube appears, the more that corresponding device (or application) is used, relative to the other available devices. For example, in both images Records is the largest box, which means it is the most-often used. The legend has been added to further reduce ambiguity in reading the plots. The legend is sorted in a top-down fashion, based on the percentage of GUI used for each device. The Records application uses the GUI the most so it is listed first on the legend. The radio uses the GUI the least so it is listed last in the legend. Finally, dotted lines and projections are used as another measure to help make reading the visualizations easier. The red-colored lines may be traced down to the x-y plane where the hardware and SUI percentages may be read. The other two dotted lines may be traced to their respective coordinate planes in a similar fashion where projections have been placed by the program to aid reading the 3-D plot. The two plots indicate interface preferences while the officers are driving and while they are stopped. It is evident both that running records checks was the most-performed task while the cruiser was stopped and that the officers used speech more for controlling lights than for anything else while driving.

This last set of figures is an excellent example of creating visualizations with enough flexibility to display different data sets without the need for altering program code. Initially the program was designed to allow the x-axis to change length depending on how much the SUI was used. While this effectively zoomed in on the 3D cubes, it created the issue of needing to adjust the x- and y-label locations every time the SUI usage percentage changed. Though the updated visualizations do not zoom in on the cubes, they are suited to handle any set of interface usages.

First 2 Officers: Controller Usage, Based on Task (Averaged per Officer), While Driving (All Days)

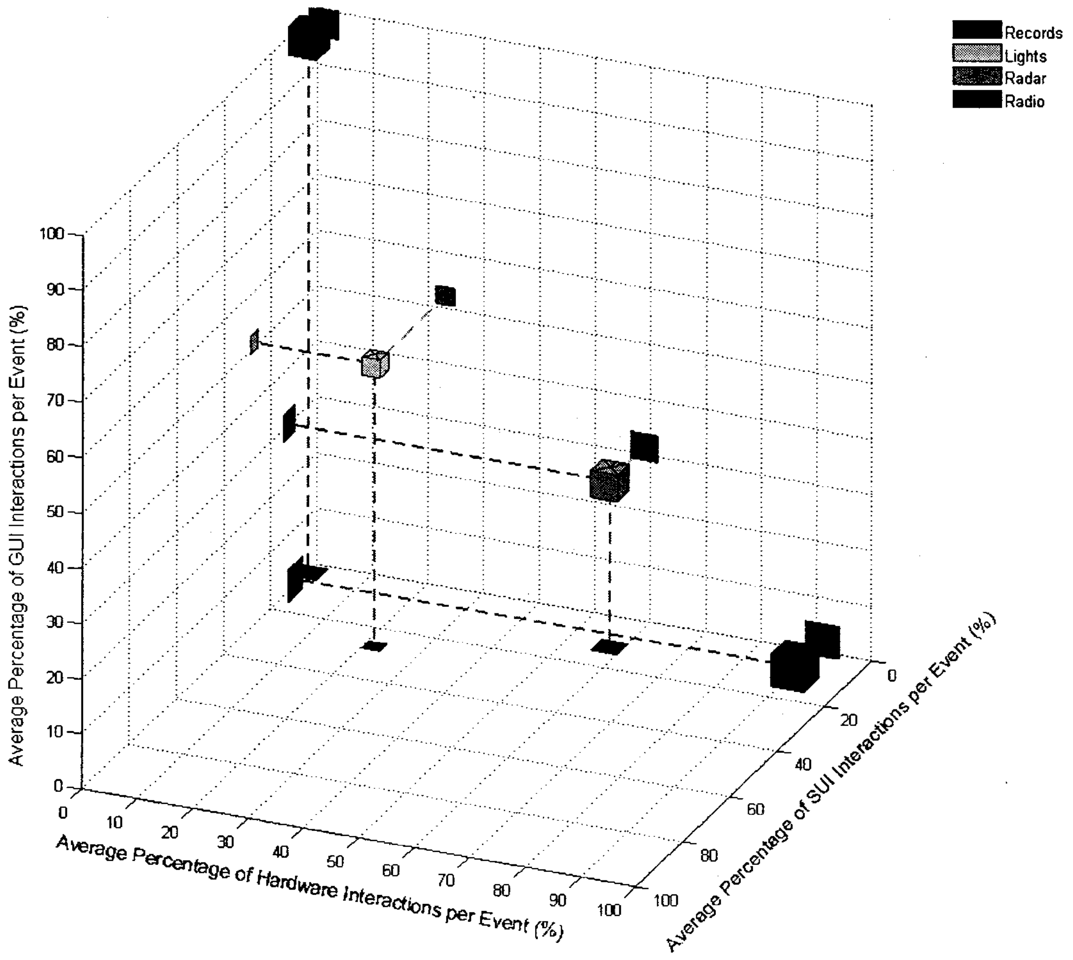


Figure 5.29 Use of 3-D imaging to represent tasks by interface preference, while moving

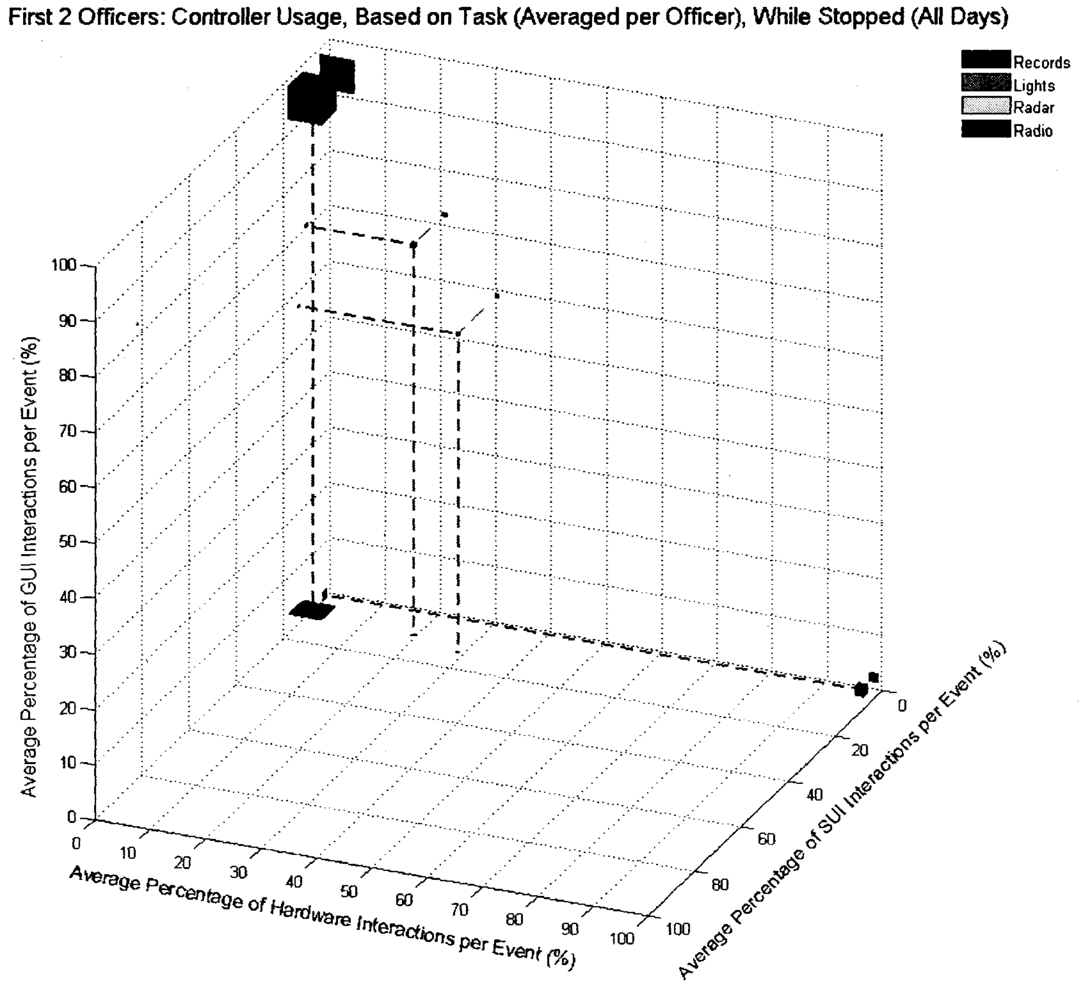


Figure 5.30 Use of 3-D imaging to represent tasks by interface preference, while stopped

CHAPTER 6

CONCLUSION

6.1 Research Conclusions

The first goal of this research project was to provide Project54 software developers and law enforcement officials with tools capable of conducting a comprehensive quantitative study into how police officers tend to use the Project54 system to operate the devices within their vehicles. This was accomplished by developing an application that could record all the user interactions within police cruisers and save that information in text files. The logging application employed knowledge of what messages constituted user interactions of interest, as suggested by Hilbert [4], Green [6], and Badre [7]. Those text files were then used as inputs to an analysis application that was designed to recode the raw data into comparable information fields. Finally, a visualization application was developed to display the results of the data analysis.

The second goal was to investigate the effectiveness of different analyses at conveying conclusive results to both the system designers and the law enforcement officials. This was accomplished by developing five different data presentations, a standard histogram plot, a 3D data plot, a screen shot with colored-button overlays, and two images that made use of coloring a cartoon police officer's silhouette in order to indicate interface preferences. The technique for using Project54 screenshots [5] with color intensifiers [10] exemplified the positive results that could be achieved by merging two distinct, accepted visualization methods in order to make use of the strengths from both. The 3D plot, the cartoon police officer images, and the screen shot with button overlays all made use of colors (especially red, green, and blue) in order to enhance each visualization's ability to convey conclusive data results, as explained by Healey [13]. The visualizations (especially the three-dimensional visualizations) were also created with flexibility in mind in order to maximize their reusability, which was suggested in Humphrey's work [14]. All the visualizations shown in this work were flexible enough to portray different data sets without making any alterations to the visualizations themselves. In other words, if different usage data from different officers was supplied to these visualizations, the results would be very similar to those shown in this thesis without having to make any changes to the program that generates the images. The five different visualizations were shown to eight research assistants within the Project54 design team to get their impressions. Once their feedback was received, the visualizations were altered to more clearly present the analysis results.

The research also involved a multi-tiered testing process that made use of the Lab Car, the driving simulator, road vehicle testing, and finally deployment into two police cruisers. The test results demonstrated that the data collection application was stable, did not induce any device operation delays, and accurately logged usage information as it was designed to do. The results of the vehicle deployment not only provided insights into how the participants were utilizing the device controls available to them, but also the data was useful for making improvements to the recoding algorithm. This was the case because the participants used the interfaces available to them in ways that were unaccounted for in previous test scenarios. The improvements to the data recoding algorithm ensured that future use of this same analysis program would generate valid results with minimal code refinements.

As a result of this thesis, Project54 developers were able to collect usage information from any Project54-equipped vehicles in service and use that information to extract interface usage patterns over the entire data pool.

6.2 Future Work

The current version of the Interaction Logger relies on GPS data for its vehicle speed information. This is undesirable for several reasons. First, the vehicles are not always within GPS range and so their speed is not always known. Second, many police cruisers do not posses GPS units and it may be the case that an officer's GPS unit could break during data collection. In either case the result would be that vehicle speed would not be known. Third, the GPS unit takes time to receive updates. During this span of time it is possible that the recorded vehicle speed would no longer adequately match the actual vehicle speed. In order to improve the likelihood of marking data records with accurate vehicle speeds, the next release of the Logger should make use of the OBDII application. This application receives vehicle speed updates directly from the cruiser's vehicle speed sensor so there is no chance of being out of range. Further, if the vehicle speed is the only feedback received from the OBDII application, the speed will refresh several times a second instead of once every several seconds so the recorded vehicle speeds would be far more accurate.

The Interaction Logger would also be improved if all the Speech input/output application's feedback messages were recorded. It could be especially beneficial to monitor commands that give insight into timing issues, such as how long it takes when a records query is initiated before the records are made available. Knowledge of potential timing issues associated with speech usage would also prove useful for applications that utilize real-time speech interactions, such as the Project54 GPS-based mapping software, currently in development.

Another worthwhile change that could be made to the Interaction Logger would be to eliminate its sniffer functionality. Currently the application only makes use of sniffed messages in order to determine the active window during Project54 user interactions. It may be beneficial to retrieve the active window information contained within the P54Gui component or to identify the active window based on the currently active speech grammar file. With the Interaction Logger's sniffer functionality replaced, there would still be knowledge of the active window but the messages received and handled by the logging application would effectively be cut in half.

Formal end-to-end testing of the data analysis software should be conducted to verify that the analyses employed in this research will be valid for new data sets. A Project54 system test script should be created with a known number of user interactions. That script should then be given to several different test subjects to perform. The test results should first be compared against the original script to verify that all activities were performed properly. Next, the number of interactions reported by the analysis software should be compared to the number of interactions within the script. The successful test will yield matching results after both comparison steps.

The addition of Active Window information should be included in all data analyses. Currently only the screenshot analysis makes use of the active window but, by providing knowledge of the Active Window to all analyses, further judgments may be made as to its relevance in how users interact with Project54. Adapting the analysis software to recode Active Window information will not be difficult because that information is already available within the recorded interaction text files.

To reduce the likelihood of interface usage scenarios that have been unaccounted for to this point in the recoding application's lifetime, it may beneficial to enhance the recoding algorithm. One way to accomplish this would be to read information from the raw data logs three lines at a time, instead of just two. Using this approach, the algorithm will have knowledge of previous data entries as well as future data entries. This will not only accurately catch user interactions that were logged out of logical order but also it could provide a cleaner approach to handling the message blocks that deal with strobes- and antenna-switching functionality.

Another improvement to the data analysis application would be to add data abstraction [5]. Data abstraction would allow groups of interactions to be combined to form a logical user tasks. For example, an officer may have to use several GUI button presses to change the radio volume to the desired level or use several keystrokes to generate a license plate check. Grouping the interactions into tasks ("Radio Volume Change" and "License Plate Check" for instance) would shift the focus from studying sources of driver distraction (multiple GUI interactions to accomplish certain jobs) to studying which interfaces the officers *prefer* to utilize to perform tasks.

REFERENCES

[1] A. L. Kun, W. T. Miller, III, W. H. Lenharth, "Project54: Introducing Advanced Technologies in the Police Cruiser," IEEE Spring VTC2002, Birmingham, AL, May 6-9, 2002

[2] Z. Medenica, A. L. Kun, "Comparing the Influence of Two User Interfaces for Mobile Radios on Driving Performance", Driving Assessment 2007, Stevenson, WA, July 9 – 12, 2007

[3] A. Pelhe, "One-to-One Communication Between Objects in the Project54 System Software," University of New Hampshire, Master's Thesis, September, 2003

[4] D. M. Hilbert, D. F. Redmiles, "Large-Scale Data Collection of Usage Data to Inform Design," Proceedings of INTERACT '01, Tokyo, Japan, July 9 – 13, 2001, pp. 569 – 576

[5] D.M. Hilbert, D.F. Redmiles, "Extracting Usability Information from User Interface Events," ACM Computing Surveys, Vol. 32, No. 4, December 2000, pp. 384-421

[6] P. Green, "Human Factors and New Driver Interface: Lessons Learned from a Major Research Project," 5th ITS-America, Washington, DC, March 15-17, 1995

[7] A. N. Badre, P. J. Santos, "A Knowledge-Based System for Capturing Human-Computer Interaction Events: CHIME," Georgia Institute of Technology Technical Report GITGVU-91, 1991

[8] B. L. Harrison, R. Owen, R. M. Baecker, "Timelines: An Interactive System for the Collection and Visualization of Temporal Data," Proceedings of Graphical Interface '94, Banff, Alberta, Canada, May 18 – 20, 1994, pp. 141 – 148

[9] M. Guzdial, C. Walton, M. Konemann, E. Soloway, "Characterizing Process Change Using Log File Data," Georgia Institute of Technology GVU Center Technical Report No. 93, 1993, pp. 93 – 44

[10] M. Guzdial, P. Santos, A. Badre, S. Hudson, and M. Grey, "Analyzing and Visualizing Log Files: A Computational Science of Usablity," Presented at HCI Consortium Workshop, February 2 – 6, 1994

[11] M. Guzdial, "Deriving Software Usage Patterns from Log Files," GVU Technical Report; GIT-GVU-93-41, Georgia Institute of Technology, 1993

[12] S. G. Eick, M. C. Nelson, J. D. Schmidt, "Graphical Analysis of Computer Log Files," Communications of the ACM, 37(12), December 1994, pp. 50 – 56

[13] C. G. Healey, "Choosing Effective Colours for Data Visualization," Proceedings from the Seventh IEEE Visualization 1996 (VIS '96), 1996, pp. 263 – 270

[14] M. C. Humphrey, "Creating Reusable Visualizations with the Relational Visualization Notation," Proceeding from IEEE Visualization 2000, Salt Lake City, UT, October 8 – 13, 2000, pp. 53 – 60

[15] X. Jiang, J. A. Landay, "Modeling Privacy Control in Context-Aware Systems," IEEE Pervasive Computing, Vol. 1, No. 3, 2002, pp. 59-63

[16] T. Giuli, D. Watson, K. V. Prasad, "The Last Inch at 70 Miles Per Hour," IEEE Pervasive Computing, Vol. 5, No. 4, 2006, pp. 20-27

[17] W. Y. Lum, F. C.M. Lau, "A Context-Aware Decision Engine for Content Adaptation," IEEE Pervasive Computing, Vol. 1, No. 3, 2002, pp 41-49

[18] S. Voida, E. D. Mynatt, B. MacIntyre, G. M. Corso, "Integrating Virtual and Physical Context to Support Knowledge Workers," IEEE Pervasive Computing, Vol. 1, No. 3, 2002, pp 73-79

[19] S. S. Yau, F. Karim, Y. Wang, B. Wang, S. K.S. Gupta, "Reconfigurable Context-Sensitive Middleware for Pervasive Computing," IEEE Pervasive Computing, Vol. 1, No. 3, 2002, pp 33-40

[20] M. Raento, A. Oulasvirta, R. Petit, H. Toivonen, "ContextPhone: A Prototyping Platform for Context-Aware Mobile Applications," IEEE Pervasive Computing, Vol. 4, No. 2, 2005, pp. 51-59

[21] A. Ranganathan, R. H. Campbell, A. Ravi, A. Mahajan, "ConChat: A Context Aware Chat Program," IEEE Pervasive Computing, Vol. 1, No. 3, 2002, pp. 51-57

[22] S. Loke, "Context-Aware Artifacts: Two Development Approaches," IEEE Pervasive Computing, Vol. 5, No. 2, 2006, pp. 48-53

[23] K. Rehman, F. Stajano, G. Coulouris, "An Architecture for Interactive Context-Aware Applications," IEEE Pervasive Computing, Vol. 6, No. 1, 2007, pp. 73-80

[24] D. Rogerson, "Inside COM," Microsoft Press, Redmond, WA, 1997

[25] R. L. Lynch, "The SpeechBot," Technical Report ECE.P54.2003.17, Electrical and Computer Engineering Department, University of New Hampshire, July 11, 2003

[26] N. Purmort, "Measuring New Hampshire State Police Radio Usage," Technical Report ECE.P54.2005.4, Electrical and Computer Engineering Department, University of New Hampshire, June 2, 2005

APPENDICES

APPENDIX A

QUESTIONNAIRE

The following questionnaire was administered to both participants in this research. Responses were indicated only to those questions which did not contain information of a personal nature. Both participants were between the ages of forty five and sixty, each with over ten years of service as a police officer. Both officers also had at least five years of experience using the Project54 system.

Subject ID: ___________ Date: ___________ Time: ___________

1. Gender

   Female ___ Male ___

2. Age: ____

3. Are you left-handed or right-handed?

   Left-handed ___ Right-handed ___

4. How long have you been a police officer?

   Exactly ________ Approximately ________

5. How long has Project54 been installed in your car?

   Exactly ________ Approximately ________

6. How often do you use the Project54 voice commands?

a) several times an hour b) several times a day _X X_
c) a few times a week d) never

---

7. How often do you use the Project54 graphical user interface (the touch-screen)?

a) several times an hour b) several times a day _X_
c) a few times a week _X_ d) never

---

8. How often do you use the original device controls (e.g. the lights switch)?

a) several times an hour _X_ b) several times a day
c) a few times a week _X_ d) never

9. Indicate the devices you prefer to control with speech commands.

Project54 window navigation___ Lights_X_ Radar_X X_
Radio_X_ Records_X X_ Video___

What (if any) reason do you have for preferring this type of controls for these devices?

---

10. Indicate the devices you prefer to control with the touch-screen.

Project54 window navigation___ Lights___ Radar___
Radio___ Records___ Video___

What (if any) reason do you have for preferring this type of controls for these devices?

11. Indicate the devices you prefer to control with the original device controls.

Lights___ Radar___ Radio___ Video___

What (if any) reason do you have for preferring this type of controls for these devices?

____

____

____

12. In what area or areas do you patrol regularly?

____

____

____

13. What does a typical shift for you consist of?

____

____

____

____

14. Do you have a routine that involves the use of the Project54 system?

____

____

____

____

15. Are there any upcoming events within the next month that will cause you to break any routines?

____

____

____

____

Please indicate your level of agreement with the following statements.

16. I am comfortable with using the Project54 system.

Strongly disagree___ Disagree___
Neither agree nor disagree___ Agree___ Strongly agree_X X_

17. I like using the Project54 system.

Strongly disagree___ Disagree___
Neither agree nor disagree___ Agree___ Strongly agree_X X_

18. I think the Project54 system is reliable.

Strongly disagree___ Disagree___
Neither agree nor disagree___ Agree_X_ Strongly agree_X_

19. I prefer using Project54 over the original device controls.

Strongly disagree___ Disagree___
Neither agree nor disagree___ Agree___ Strongly agree_X X_

20. I am satisfied with the accuracy of the speech recognition in my vehicle.

Strongly disagree___ Disagree___
Neither agree nor disagree___ Agree_X_ Strongly agree_X_

21. Using speech commands improves my productivity.

Strongly disagree___ Disagree___
Neither agree nor disagree___ Agree_X X_ Strongly agree___

22. Using speech commands makes operating my vehicle safer.

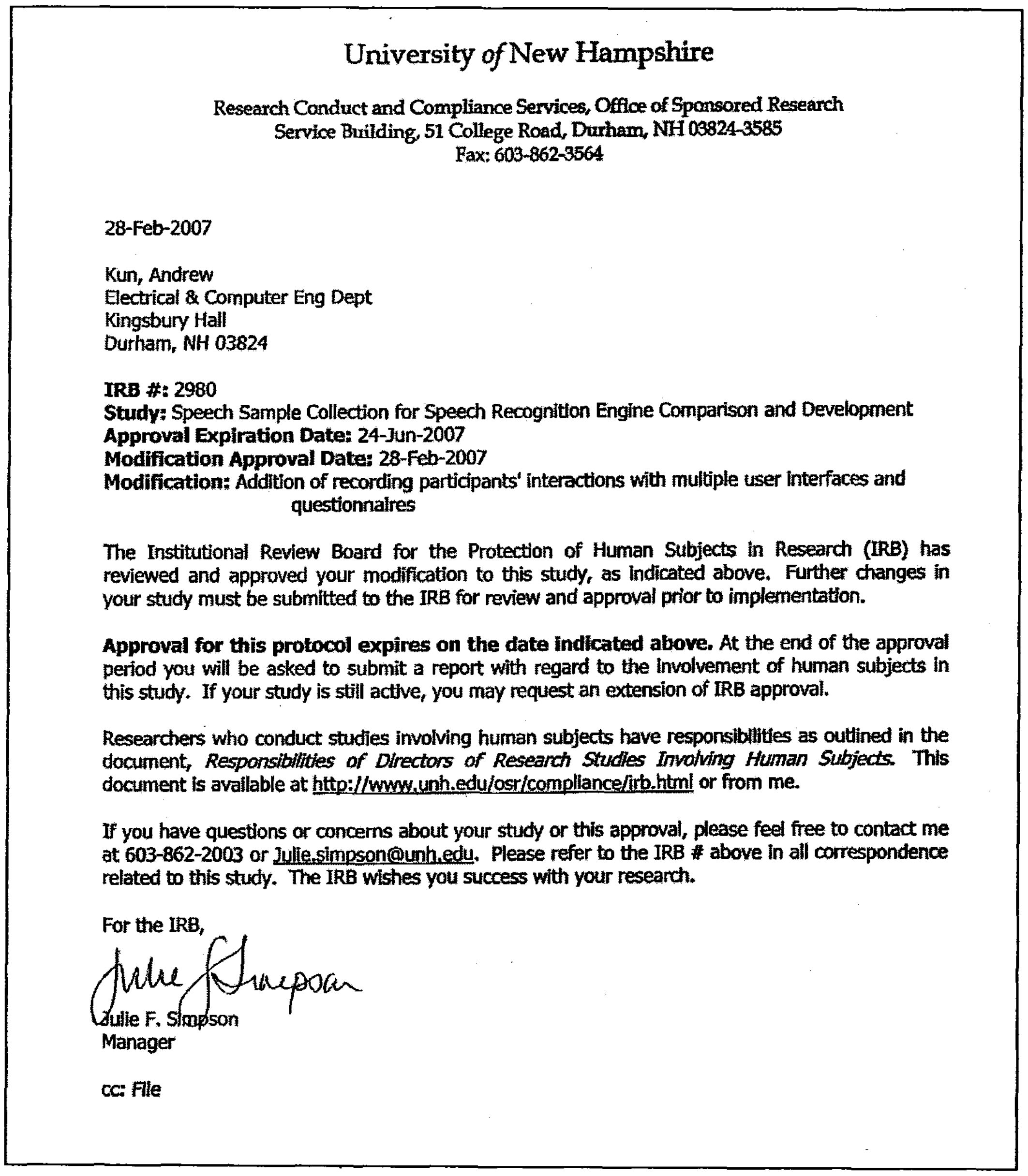
Strongly disagree___ Disagree___
Neither agree nor disagree___ Agree___ Strongly agree_X X_

23. The touch-screen buttons I like to use are located in the best place on the screen for me to use them.

Strongly disagree___ Disagree___
Neither agree nor disagree___ Agree_X_ Strongly agree_X_

APPENDIX B

INSTITUTIONAL REVIEW BOARD APPROVAL



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

28-Feb-2007

Kun, Andrew
Electrical & Computer Eng Dept
Kingsbury Hall
Durham, NH 03824

IRB #: 2980
Study: Speech Sample Collection for Speech Recognition Engine Comparison and Development
Approval Expiration Date: 24-Jun-2007
Modification Approval Date: 28-Feb-2007
Modification: Addition of recording participants' interactions with multiple user interfaces and questionnaires

The Institutional Review Board for the Protection of Human Subjects in Research (IRB) has reviewed and approved your modification to this study, as indicated above. Further changes in your study must be submitted to the IRB for review and approval prior to implementation.

Approval for this protocol expires on the date indicated 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 document, *Responsibilities of Directors of Research Studies Involving Human Subjects.* This document is available at http://www.unh.edu/osr/compliance/irb.html or from me.

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 Simpson

Julie F. Simpson
Manager

cc: File

APPENDIX C

DATA VISUALIZATION CREATION GUIDE

Once interaction records are retrieved from police cruisers they are already prepared for the full set of data analyses presented in this thesis. In order to use these analysis tools, the recorded interaction information must be saved in the following path:

C:\Project54\Logs\System Usage Logs\Officer ID Folder\

The name of the *Officer ID Folder* is a five-character folder name of the designer's choosing. The two folders created for this project used the two officers' badge numbers to name their respective folders. The folder name has to be five characters to make MATLab matrix string comparisons possible.

Before recoding interaction files for the first time, the P54 System Usage Analyzer program source code has to be altered. This one-time change involves going to the commented section at the start of the source code and adding:

```
public string m_OfficerSource = @"C:\Project54\Logs\System Usage Logs\
Officer ID Folder\";

public string m_OfficerRecode = @"C:\Project54\Logs\System Usage Logs\
Officer ID Folder\";
```

Again, the *Officer ID Folder* is named according to the designer's choosing. The rest of the both entries are formatted to following C# programming rules for creating a string that holds a path location name.

To generate visualizations for the data the MATLab visualization m-file has to be opened and the five-character *Officer ID* has to be added to each cell from which data visualizations are required.

The steps described for the analysis and visualization of interactions only need to be performed one time for each new officer that provides data. The various source file locations in which the changes are to be made (the MATLab code has several) are located at the beginning of the code (or MATLab cell) and are all marked with comments. Also, the additional lines of code will exactly match the lines that already exist for the two officers who have already participated in this research; all that will change is the five-character *Officer ID*.