CyborGlogger: A Computational Framework for Real-time CyborGlogging

by

Raymond Chun Hing Lo

A thesis submitted in conformity with the requirements for the degree of Master of Applied Science
Graduate Department of Electrical and Computer Engineering
University of Toronto

Copyright © 2010 by Raymond Chun Hing Lo
Abstract

CyborGlogger: A Computational Framework for Real-time CyborGlogging

Raymond Chun Hing Lo

Master of Applied Science

Graduate Department of Electrical and Computer Engineering

University of Toronto

2010

CyborGlogs are lifelong log files of personal experiences that are captured without conscious thought or effort. By creating cyborglogs on a continuous basis, we can enable various novel applications where the lifelong records are used as memory aids or for personal safety. This thesis presents the development of a mobile application that provides the tools to instantly capture, archive, recall, and share our personal experiences on widely available cameraphones. To achieve this goal, a client-server computational framework is designed and implemented to support real-time interaction among the users. Three fully functional prototypes supporting three major mobile platforms (J2ME, Symbian, and iPhone) are presented to show the feasibility and flexibility of this framework. Finally, this thesis shows an early prototype which demonstrates the idea of mediated reality on cameraphones using various integrated sensors. This prototype explores the possibilities of developing truly intelligent wearable applications on mobile devices in the future.
Acknowledgements

This thesis would have never been possible without the support and mentorship from my supervisor Prof. Steve Mann and Dr. James Fung. Today, I still vividly remember the time we spent together developing the first version of CyborGlogger on a mobile phone when I was still an undergraduate student. Without the initial guidance and collaboration with these great individuals from the research lab, I would have never been able to achieve what I have accomplished at this stage of my life.

During these years of research on CyborGlogger, I have received much support from individuals who are also passionate about this work. First, I would like to thank Dr. Stefanos Pantagis for donating the Sony Ericsson K800 cameraphone to the lab. His generous support in the early stage of this project strongly encouraged me to further pursue my research in this field. Then, I would like to acknowledge the support from SDI Corporation (especially Sadek Ali and Adnan Ali). I would also like to thank The Innovation Group at the University of Toronto for assisting us in the process of filing a patent on this invention. Furthermore, I would like to thank all the examination committee members, Prof. Paul Chow, Prof. Khai Truong, and Prof. D. Hatzinakos for providing their comments on my thesis.

Finally, I would like to thank everyone who has shown their love and care to me in my life. In particular, I would like to show my greatest gratitude to my parents for giving me the opportunity to study in Canada. Without their sacrifice, none of this would have ever happened.
# Contents

1 Introduction  
   1.1 CyborGlogging and Sousveillance ............................. 1  
   1.2 Mobile Platforms ............................................. 2  
   1.3 Organization of Thesis ....................................... 5  

2 Related Work  .......................... 7  
   2.1 EyeTap .................................................. 7  
   2.2 Microsoft SenseCam and MyLifeBits ......................... 8  
   2.3 Nokia LifeBlog ............................................. 9  
   2.4 Mobile Blogging ............................................. 9  
   2.5 Reality Mining ............................................. 10  
   2.6 Summary ................................................. 10  

3 Cyborglogging with Cameraphones: Steps toward Equiveillance  11  
   3.1 Introduction .............................................. 11  
   3.2 Surveillance and Sousveillance .............................. 11  
   3.3 Equiveillance through a Cameraphone System .................. 12  
      3.3.1 Concomitant cover activity .............................. 17  
   3.4 Philosophical, moral, and ethical questions ................... 18  
   3.5 Conclusions .............................................. 18
# Glogger Framework

4.1 Glogger Overview .................................................. 21

4.2 Implementation of the Glogger Mobile Application .................. 23

4.2.1 Platforms ......................................................... 25

4.2.2 System Overview ................................................ 27

4.2.3 Data Acquisition ............................................... 27

4.2.4 Significant Event Detection .................................... 30

4.2.5 Data Packaging and Archival ................................. 34

4.2.6 Content Sharing ............................................... 36

4.3 Implementation of the Glogger Server ........................... 40

4.3.1 Overview ......................................................... 40

4.3.2 Glogger Engine ................................................ 41

4.3.3 Glogger Database ............................................... 43

4.4 Evolution of Glogger ............................................... 43

4.4.1 Glogger Prototypes ............................................. 45

4.4.2 Glogger Live (for J2ME-enabled Phones) ...................... 46

4.4.3 GloggerVS2 (for Symbian Phones) .......................... 51

4.4.4 iGlogger (for iPhone) .......................................... 53

4.5 Conclusions ......................................................... 57

# Glogger Extensions

5.1 Introduction ......................................................... 59

5.2 Design Challenges and Related Work ............................ 60

5.3 Image Acquisition and Data Preparation ......................... 61

5.4 Scale-Invariant Feature Transformation .......................... 62

5.4.1 Keypoint Localization ......................................... 62

5.4.2 Local Extrema Detection ...................................... 64

5.4.3 Filtering Keypoints ............................................. 64
5.4.4 SIFT Keypoint Descriptor ........................................ 67
5.4.5 Complexity Analysis Summary .................................. 68
5.5 Results and Applications .......................................... 69
5.6 Going Beyond SIFT .................................................. 69

6 Conclusions ................................................................... 75
6.1 Summary of Contributions .......................................... 75
6.2 Future Work ............................................................ 76
6.2.1 Improve Significant Event Detection Algorithms .......... 76
6.2.2 Mediated Reality and Cyborglogs ............................... 77

Bibliography .................................................................... 78
List of Tables

4.1  iPhone 3GS vs SenseCam .................................................. 25
4.2  Multimedia File Formats Supported on Glogger .......................... 42
5.1  A Summary of the Time Complexity Analysis .............................. 70
5.2  A Summary of the Space Complexity Analysis ............................ 71
List of Figures

1.1 An Early Example of a CyborGlog .................................................. 3
1.2 Neck-worn Wearable Setup ......................................................... 5
3.1 Preparing for ACM-CFP 2005 Opening Keynote ................................. 13
3.2 Neck-worn Sousveillance Wearable Setup ........................................ 14
3.3 Glogger Instant Communications .................................................. 15
3.4 A CyborGlog showing annotated images ........................................ 16
3.5 CyborGlog Storyboard ................................................................. 19
4.1 System Architecture Overview of Glogger ........................................ 22
4.2 Screenshots of the Glogger Mobile Application and the Glogger Server Interface ................................................................. 24
4.3 System Overview of Glogger Mobile Application .................................. 26
4.4 A Flow-chart of the Passive Capture Algorithm on Glogger .................. 28
4.5 Structure of a Cyborglog ............................................................... 35
4.6 System Overview of the Glogger Engine ........................................... 41
4.7 The User Directory Structure of the Glogger Server Database ............... 44
4.8 Glogger Map Interface ................................................................. 45
4.9 Glogger J2ME One-click Upload Interface ......................................... 46
4.10 Glogger J2ME Camera and StoryBoard Creation Interface .................. 48
4.11 Glogger J2ME StoryBoard Feed Interface ......................................... 48
Chapter 1

Introduction

1.1 CyborGlogging and Sousveillance

Cybernetic organisms (*cyborgs*) are defined by way of a synergy between human and machine such that the operation of the machine does not require conscious thought or effort on the part of the human [27, 33]. CyborGlogging (*glogging*) is a concept that involves capturing one’s own life on a continuous basis, without requiring conscious thought or effort, from a first-person perspective [32]. The lifelong log files of personal experiences that are captured effortlessly this way are called CyborGlogs (*glogs*) [33]. With the advent of the World Wide Web, CyborGlogs can also be shared with others instantly.

One key difference between CyborGlogs (*glogs*) and WeBlogs (*blogs*) [24] is that CyborGlogs are generated passively without user intervention while WeBlogs are usually composed through a post-documentary process which is performed after the events happened [31]. In contrast, WeBlogs are often unable to document every detail of one’s life because users are unable to remember, as opposed to CyborGlogs where personal experiences are recorded continuously and documented as they happen. One early example of a CyborGlog is the campus fire recorded by Mann with his EyeTap wearable device.
in 1995 as shown in Figure 1.1. Continuous recording often results in the serendipitous capture of important moments, such as the campus fire.

Lifelong CyborGlogs can be used as a personal visual memory prosthesis, particularly for the elderly and patients with Alzheimer’s disease [25, 36]. For example, scrolling through CyborGlogs of family photographs could bring back childhood memories of precious moments. CyborGlogs can also be used for personal safety since the continuous capture of data can allow them to function like a “black box” flight recorder in an aircraft that provides evidence as to why an accident occurred. These applications lead to not only the personal uses of CyborGlogs, but also the broader social impact of CyborGlogs for *sousveillance* purposes.

The term “surveillance” is derived from the French words “sur” (above) and “veiller” (to watch). Likewise, the term “sousveillance” is derived from the French words “sous” (below) and “veiller” (to watch) [31, 33]. Typically, surveillance cameras look down from above, both physically and hierarchically where person(s) of higher authority (e.g. store manager, security guards, or the like) watch over workers, suspects, or citizens. In contrast, the term personal “sousveillance” has been used to describe the recording of an activity by a participant in the activity where cameras are brought down to eye-level for human-centered recording of personal experiences. An important aspect of “sousveillance” or “inverse surveillance” is that it emanates from individuals recording their personal experience and their immediate vicinity, rather than the recording/monitoring of individuals by an outside party.

### 1.2 Mobile Platforms

As digital cameras and personal computers become more ubiquitous, it is becoming increasingly convenient and inexpensive to capture and archive hundreds to thousands of multimedia (image, video, or audio) files. The miniaturization of portable devices and
Figure 1.1: An early example of a CyborGlog created by Prof. Mann with his EyeTap device and wearable computer (WearComp) on February 22, 1995. Mann was able to capture and transmit the images of the event as the event was still happening. The figure shows a screenshot of the HTML-based webpage rendered on the NSCA Mosaic web browser. The webpage summarizes the event with images and text annotation and can be found at http://www.wearcam.org/previous_experiences/eastcampusfire [31].
advants in sensor technology have allowed the use of various types of commodity electronic devices for capturing our everyday experiences and interactions with our surroundings. These portable devices are capable of continuously recording data not only from a camera and a microphone, but also from various types of sensors such as an accelerometer to extract a user’s context. As mobile technology evolves, it is now possible to record one’s entire life by utilizing widely available off-the-shelf cameraphones at a much more reasonable cost.

Previous research on continuous lifelong capture focused on prototyping with wearable devices such as EyeTap [32] (a wearable device that enables first-person experience capture) and Sousveillance Dome [31] (a neck-worn wearable webcam that takes images passively as shown in Figure 1.2). Later, Microsoft also developed a neck-worn wearable digital camera called SenseCam [25] that takes photographs passively. However, these devices are often heavily customized and thus difficult to build, repair, and replace by ordinary individuals. Therefore, this thesis focuses on the design and implementation of a computational framework called CyborGlogger (or Glogger) on ubiquitous mobile platforms for personal experience capture and sharing.

The rapid evolution of mobile technology has turned mobile phones into an ideal platform for CyborGlogging for the following reasons. First, modern cameraphones (e.g., Apple iPhone 3GS) have a rich set of sensors including an accelerometer, a Global Positioning System (GPS) receiver, and a digital compass. Second, the wireless connectivity of mobile phones allows them to communicate with other devices or connect to additional biosensors which can be worn by the users. Third, the widespread use of cameraphones allows them to be easily replaced or upgraded. This also provides a platform for large-scale software deployment to reach a large number of users around the world. Finally, the small form factor of mobile phones makes them ideal to be carried around or to be worn in our daily life.
Chapter 1. Introduction

In order to serve as a useful retrospective memory aid, a wearable camera needs to be very practical. Issues such as ease of use during capture and replay are paramount. The motivation for SenseCam is to extend and unify the body of work reviewed here to create a small, low power, easy to operate and carry, wearable camera, which can automatically capture images of a person’s day using purely onboard sensing.

### 3. SenseCam

SenseCam is a small digital camera that is designed to take photographs automatically, without user intervention, whilst it is being worn (Figure 1). Unlike a regular digital camera or a cameraphone, it does not have a view finder or a display that can be used to frame photos. Instead, it is fitted with a wide-angle (fish-eye) lens that maximizes its field-of-view. This in turn means that nearly everything in view of the wearer is captured by the camera. Examples of the images taken by SenseCam are shown in Figure 2. In addition to the camera functionality, a number of different electronic sensors are built into SenseCam. These sensors are monitored by the camera’s microprocessor, and certain changes in sensor readings can be used to automatically trigger a photograph to be taken. For example, a significant change in light level, or the detection of body heat in front of the camera can be used as triggers. Additionally, an internal timer may be used to trigger photograph capture, causing a photo to be taken automatically every 30 seconds, for example. SenseCam also has a manual

![Figure 1.2](image)

**Figure 1.2:** Pictures of the neckworn wearable devices sorted in chronological order from left to right. (a) A picture of the sousveillance dome, a necklace webcam, worn by a user [33]. (b) The Microsoft SenseCam device worn by a user [25]. (c) The cameraphone becomes a wearable device, worn with a strap around the neck of the user. The camera, when resting, has an upside-down ergonomic design [38].

#### 1.3 Organization of Thesis

The goal of this thesis is to develop a real-time computational framework for Cybor-Glogging on mobile platforms. In particular, this thesis aims to create a software-based framework and a set of mobile applications which enable cameraphones to capture one’s life continuously.

The organization of the thesis is outlined as follows. In Chapter 2, we provide a list of related work surrounding the idea of CyborGlogging. In Chapter 3, we present a conceptual framework called “equiveillance” and the underlying motivation behind creating a mobile application called Glogger as the “sousveillance” or “inverse surveillance” platform. The concept of “equiveillance” presented in Chapter 3 forms the basis of discussion for the Glogger application that runs on commodity cameraphones. However, to create a
mobile application which enables real-time capture and sharing of personal experience, a number of technical issues need to be addressed. In Chapter 4, we present the design and implementation of the Glogger framework. In addition, this chapter shows the evolution of the Glogger application with the three prototypes developed on three major mobile platforms (J2ME, Symbian, and iPhone). In Chapter 5, we present one possible extension of the Glogger framework by implementing a real-time image tracking algorithm on cameraphones. Lastly, in Chapter 6, we summarize the significant contributions and possible future work of this thesis.
Chapter 2

Related Work

This chapter highlights the significant research on lifelong personal experience capture in both academia and industry.

2.1 EyeTap

EyeTap is a wearable device that allows the eye of the wearer to function both as a camera and a display [32]. The unique arrangement of the EyeTap device allows it to capture and analyze rays of light passing through the eye, modify them, and resynthesize them. One of its main features is its ability to record exactly what a person sees. By wearing EyeTap for personal experience capture over the last 30 years, Mann [32] extended the idea of lifelong log to lifelong cyborglogs, which are recordings generated without the user’s conscious thought or effort. He also explored the social implications of lifelong capture and proposed the notion of inverse surveillance (termed “sousveillance”) [39]. However, the relatively high cost as a result of the custom components required to build the EyeTap system has hindered the widespread use of this technology. This work, therefore, focuses on exploring the idea of CyborGlogging and sousveillance on the ubiquitous mobile platform.
2.2 Microsoft SenseCam and MyLifeBits

The Microsoft SenseCam device is a digital wearable camera that takes photographs passively. It contains a number of electronic sensors including a passive infrared (body heat) sensor, a light meter, and an accelerometer [25] similar to the left-most neck-worn wearable devices as shown earlier in Figure 1.2 [33]. The onboard camera has an ultra-wide angle to capture the scene in front of a person (but not necessary what a person sees exactly). Image capture can be triggered by various sensors or alternatively, this can happen at a fixed time interval. After a day of usage, the device would typically capture over 2000 images [23]. Several case studies also describe the use of SenseCam to improve autobiographical memory in patients with neurodegenerative conditions such as limbic encephalitis and Alzheimer's disease [8, 25]. However, this custom hardware designed by Microsoft is not widely available and could only be obtained for research purposes. In contrast, the work proposed in this thesis aims to support a large number of compatible mobile phones and provide additional features such as real-time experience sharing. This thesis also presents a web service that manages the lifelong logs and allows for access from anywhere in the world provided that Internet is available.

Another approach from Microsoft is called MyLifeBits which is a software, primarily inspired by Bush’s memex introduced in the previous section, for organizing all types of digital media, including documents, images, sounds, and videos in a person’s life [22]. More recently, the MyLifeBits project was integrated with the SenseCam idea in an attempt to visualize and organize the massive set of passively taken photographs along with the sensor data [21]. One aspect this project has not addressed is the importance of social interaction (i.e., allowing peer-to-peer interaction and commenting) through instant sharing of the content.
2.3 Nokia LifeBlog

Nokia’s LifeBlog is a commercial application for mobile phones and desktop computers that automatically creates multimedia diaries from the user’s mobile data [42]. It collects all kinds of data from the mobile device, including images, videos, multimedia messages, and blog entries. The collected data are synchronized with a personal computer for archiving and visualization. It also makes use of context data such as timestamps, location information, and other metadata to describe these multimedia data and make them searchable. However, LifeBlog does not focus on the real-time sharing of mobile data. Additionally, this solution does not allow for passive capture; instead, it purely relies on the user’s input to the mobile phone and thus the mobile data are only generated on-demand and synchronized to a personal computer afterwards.

2.4 Mobile Blogging

Commodity mobile phones with a variety of sensors (cameras, accelerometers, GPS, and health monitors) are already widely used and are becoming more ubiquitous each year. This has prompted researchers to utilize cameraphones as a real-time publishing tool allowing users to upload and share multimedia contents anywhere and anytime [2, 5, 14, 16, 20]. Davis et al. [16] proposed a system called the Mobile Media Metadata 2 (MMM2) that allows users to upload, archive, and share images with cameraphones. Similarly, Cemerlang et al. [14] and Beale et al. [4] also presented systems that allow users to post blogs instantly and support social interaction on mobile phones.

Others have also attempted to create mobile applications that capture not only photographs and text annotations, but also other contextual information such as location data and proximity data about the users. For example, Bamford et al. [2] implemented a system that allows users to create location-based blogs (termed LocoBlog) on mobile phones with Bluetooth GPS receivers. Likewise, Gaonkar et al. [20] created a system
called MicroBlog that enables cameraphones to generate and share multimedia along with location data using WiFi, GSM, and GPS based localization schemes. However, these blogging systems mainly focus on the creation of blogs with photos and text annotations rather than the passive capture and analysis of input data in real-time, as demonstrated in this thesis.

2.5 Reality Mining

Since most people today carry their mobile phones with them in their daily life, mobile phones can potentially be used to continuously collect user-centric data such as communication, proximity, location, and activity information for analyzing human social behaviors. The Reality Mining Project has applied machine learning techniques to analyze the complex social system captured from mobile phones [18]. This project has demonstrated some important uses of the continuously recorded human-centric mobile data. One potential use is to provide insights into the underlying relational dynamics of individual and social communication patterns [19]. However, multimedia contents, such as images, are not recorded and analyzed in this project.

2.6 Summary

This chapter provided an overview of the important research projects in the wearable and mobile computing domain specifically for the purpose of personal experience capture. It is clear that more researchers are beginning to utilize mobile phones as the new computing platform because of its widespread adoption (with billions of active mobile phones in the world) and the ever increasing capabilities of these devices.
Chapter 3

Cyborglogging with Cameraphones: Steps toward Equiveillance

3.1 Introduction

In this chapter, we introduce the notion of “equiveillance” as a conceptual framework for understanding the balance between surveillance and sousveillance. In addition to this conceptual framework, we introduce a practical embodiment of equiveillance in the form of a new program called “CyborGlogger” (Glogger) that runs on most modern cameraphones, along with the server architecture to support Glogger. The work described in this chapter is published in the Proceedings of the 14th annual ACM International Conference on Multimedia [38]. The detailed implementation and design of Glogger are described in Chapter 4 of this thesis.

3.2 Surveillance and Sousveillance

Sousveillance stems from the French words “sous”, meaning “below”, and “veiller”, meaning “to watch”. Research on the topic of surveillance is well established by way of a number of IEEE and ACM conferences as well as various journals, such as Surveillance &
Society. More recently, the notions of “inverse surveillance” as well as personal experience capture (both referred to as “sousveillance”) have emerged.

Surveillance connotes a kind of “archicentric” omniscient “eye-in-the-sky” (God’s eye or authoritarian view), in which cameras are affixed to buildings or other architectural elements. Conversely, sousveillance involves the recording of an activity by a participant in the activity. Panoptic surveillance often requires secrecy (i.e., a centralized optical system that ensures total transparency in one direction and zero transparency in the other direction).

Sousveillance usually involves a peer-to-peer approach that decentralizes observation to produce transparency in all directions. Sousveillance seeks to reverse the otherwise one-sided panoptic gaze. Sousveillance is related to (even if the opposite of) the tradition of surveillance and the artistic practice explored by surveillance. The opening keynote for the 15th Annual Conference on Computers, Freedom & Privacy, by the Association of Computing Machinery (ACM CFP 2005) was a panel discussion on Equiveillance [1]. Additionally, a sousveillance workshop, performance, and deployment was coordinated for the conference, including the creation of 500 sousveillance devices that were made and deployed at this event (Figure 3.1).

### 3.3 Equiveillance through a Cameraphone System

In this section, we present a cameraphone application called “CyborGlogger” (or Glogger for short) that enables cameraphones to be used in a sousveillance CyborGlogging system. Glogger is a readily available\(^1\) tool that makes possible an artistic form of personal expression. It allows the practice of sousveillance to be performed and resulting images and narratives to be produced. The creation of such a tool that is freely available moves the exploration of sousveillance a step forward from previous work.

\(^1\)Available for download at http://m.glogger.mobi
Figure 3.1: Preparing for ACM-CFP 2005 Opening Keynote: The Opening Keynote address was a panel discussion on equiveillance. Additionally, a “maybecamera” sousveillance device was given to each attendee. (a) ACM CFP 2005 Dome sewing party: Alex Cameron (upper left), John Gilmore (at right), and numerous others participated in the dome sewing party organized by ACM CFP 2005 conference Chair, Deborah Pierce. (b) Manufacture of ACM CFP 2005 sousveillance bags: Five hundred “maybecameras” were manufactured for ACM’s CFP 2005. The domes were made by a vacuum forming process, on a mandrel that could handle four at a time. They were then laser-drilled, using an automated CNC process. Finally, a team of approximately 20 volunteers hand-assembled the units by sewing them onto the conference bags, (one for each each conference attendee). Wearable wireless video cameras were inserted into some, but not all of the bags (hence the name “maybe camera”). Additionally, some (but not all) of the bags that had no cameras in them had flashing red LEDs and other “fake” circuitboards. During the conference, images from the real working camera bags were broadcast over the Web and in various places throughout the conference hall and surrounding hotel space, on large screens.
Chapter 3. Cyborglogging with Cameraphones: Steps toward Equiveillance

Figure 3.2: The art of sousveillance brings down the cameras from the ceilings guard towers, and lamp posts [see (a)], and re-situates them in a human-centered rather than architecture-centered context, down from the heavens, and down-to-earth, at the level of the average person [see (b)]. The cameraphone becomes a wearable device, worn from a strap around the neck. The camera, when resting, has an upside down ergonomic design [see (c)].

Glogger includes several specific features for sousveillance and continuous personal experience capture that promote a different usage from typical point and shoot cameraphone usage. One basic user interaction of Glogger is the “one click” image capture and upload where images are transmitted to a remote server in real-time. Also, Glogger can be set to capture continuously without any key presses or user interaction. The cameraphone may be worn, say dangling around the neck, and the camera will capture a first-person perspective set of images from the user’s everyday life (See Figure 3.2).

The Glogger system has several differences from other online photosharing communities. Firstly, Glogger is aimed at creating real-time personal narratives. In Figure 3.3, images and texts are taken by the user and published at the same time as the user is experiencing them. This differs from photo sharing sites which focus on online, web-accessible albums, but do not necessarily emphasize real-time upload or developing narrative. For
example, a Glogger user may take a series of many images to describe the event as it develops in time, whereas traditional photographic sharing may seek to take only a few descriptive images of an event.

Real-time automatically-generated panoramas provide a new way to visualize what users can see. The field of view capturing an experience is expanded through the Glogger system. Moreover, these panoramas automate the task of aggregating similar pictures (i.e. through automatic scene-change detection) into the same “orbit” [41]. Figure 3.4 shows a panoramic image taken with the system.

The cameraphone becomes a “great equalizer” not merely in the sense of a political “weapon” but more importantly in the balanced sense of equilibrium that it can afford. Thus, although it is tempting to see SUR and SOUS as binary, us-versus-them opposites, we are hoping to build a system of equiveillance, that is, the possibility that these two very different social practices might somehow result in some kind of equilibrium. This work contributes to the discussion of the use of ubiquitous cameras and Glogging of events by discussing them within the context of sur/sousveillance and equiveillance and
Figure 3.4: A CyborGlog showing annotated images
the surrounding legal and ethical issues.

### 3.3.1 Concomitant cover activity

In contrast to other photosharing sites, where images are uploaded from PCs, Glogger uses cameraphones which are commonly carried by people in their day to day lives. This further allows the exploration of “concomitant cover activity”. The main force of equiveillance comes from an uncertainty as to whether or not the device is being used as a camera at any given moment. This stems from its various other uses. The other non-picture-taking purpose of the device provides for a concomitant cover activity.

In photography (and in movie and video production), it is desirable to capture events in a natural manner with minimal intervention and disturbance. Current “point and click” photographic or video apparatus create a visual disturbance to others and attract considerable attention on account of the gesture of bringing the camera up to the eye. Even if the size of the camera could be reduced to the point of being negligible (e.g. no bigger than the eyecup of a typical camera viewfinder, for example), the very gesture of bringing a device up to the eye is unnatural and attracts considerable attention, especially in establishments such as gambling casinos or department stores where photography is often prohibited.

The Glogger system’s continuous capture allows a cameraphone to be worn, without giving an unusual appearance to others (such as a potential assailant). Such an apparatus might also be of use in personal safety. Although there are a growing number of video surveillance cameras installed in the environment allegedly for “public safety”, there have been recent questions as to the true benefit of such centralized surveillance infrastructures. Most notably, there have been several examples in which such centralized infrastructure has been abused by the owners of it (as in roundups and detainment of peaceful demonstrators). Moreover, “public safety” systems may fail to protect individuals against crimes committed by the organizations that installed the systems. The
apparatus allows the storage and retrieval of images by transmitting and recording images at one or more remote locations. Images may be transmitted and recorded in different countries, so that they would be difficult to destroy, in the event that the perpetrator of a crime might wish to do so.

Moreover, as an artistic tool of personal expression, the apparatus allows the user to record, from a first–person–perspective, experiences that have been difficult to so record in the past. As a result, the system can be used to capture everyday life without the conscious thought or effort of the user. Alternatively, if the user wishes to manually override the automated Cyborglogging system, he or she can make deliberate narratives, complete with annotations, as illustrated in Figure 3.5.

### 3.4 Philosophical, moral, and ethical questions

One of the virtues of equiveillance is an increased reciprocal transparency in the operations of powerful entities engaged in surveillance. Such reciprocal transparency has become necessary, in part, because surveillance often takes place surreptitiously, i.e., without the knowledge and consent of the people who are being surveilled, whereas Sousveillance tends to be a more open process.

Naturally, as with any new technology, there will be both advocates as well as opposers. When faced with the moral or ethical dilemma of when to run Glogger, we consider, as a base-level of operation, the notion of equiveillance. Equiveillance doctrine says that as long as surveillance is present in the environment, that a person ought to have a moral and ethical right to engage in sousveillance.

### 3.5 Conclusions

We have presented “equiveillance” as a conceptual framework, along with a practical embodiment by way of a program called Glogger. This chapter has explored the balance
**TTC Strike**

5/5 of 2 ratings

You have already rated this story more...  

Got myself to Finch station and figured that TTC was on strike.

Uploaded: July 28, 2006, 5:39 pm by raymonldo84

Surprised at the subway. Found the doors locked, a sign saying no service. Unannounced transit strike.

Uploaded: July 28, 2006, 5:42 pm by fungja

Cannot go anywhere except going back home. Wasted two YRT tickets and 2 hours for nothing.

Uploaded: July 28, 2006, 5:46 pm by raymonldo84

**Figure 3.5:** Example of a deliberately constructed, communal narrative, as an alternative to a free-running CyborGlog.
between surveillance and sousveillance, and discussed the need for this balance. We have shown how new technologies, such as cameraphones, can be used to create “sousveillance” communities that can lead us toward equiveillance by providing a counterpoint to the existing well-established surveillance practice.
Chapter 4

A Real-time Computational Framework for Cyborglogging

This chapter presents the system design and implementation of a fully functional computational framework called Glogger that is built for mobile platforms. At the time of writing, the work described here has been tested by over 70,000 mobile phone users worldwide and has undergone numerous revisions based on the feedback from the community. Additionally, a patent of this invention has been filed [37] through The Innovations Group at the University of Toronto.

4.1 Glogger Overview

Glogger is a computational framework which consists of two key components. These include (1) a server component for managing data and handling user records, and (2) a client component for collecting data and handling user interactions. For example, the server component which is responsible for data storage and retrieval can communicate with different mobile platforms through the standard Hypertext Transfer Protocol (HTTP) [7] and markup languages that are widely supported on mobile phones. To demonstrate the practical use of this framework, this thesis shows several implementations of the Glogger
Mobile Application tailored to three popular mobile platforms to allow Glogger to run on a large set of mobile devices. Despite differences in the hardware and software, these different prototypes are able to communicate with one another seamlessly due to the use of a server-client architecture as well as the use of standard open protocols and interfaces [12].

Figure 4.1 provides a high-level system overview explaining the relationship between different components in Glogger. Mobile clients, which include mobile phones, wearable computers, desktop computers, or the like, interface with the Glogger Server through the Glogger Application API using standard open protocols such as HTTP and standard markup languages such as Hyper Text Markup Language (HTML) and Extensible Markup Language (XML) [9]. The Glogger Engine is responsible for record keeping and multimedia file conversion (e.g., resizing images or transcoding videos for optimal viewing on various resource-limited devices). The Glogger Server has a file database storing all
cyborglogs which are represented as a stream of multimedia files such as image, video, and audio. To efficiently manage the cyborglogs, all cyborglogs are indexed in an SQL Server Database with text annotations and GPS locations. On the other side, the Glogger Mobile Application functions as a data collector for sensor data about the user’s context and also provides the Graphical User Interface for handling interaction with the user. Glogger Mobile Application may also interface with other sensors such as EyeTap (a pair of wearable digital eyeglasses [35]) to enable first-person perspective recording of personal experiences. In one of the wearable cameraphone prototypes, a remote game controller (Wii Controller\(^1\)) is connected to the mobile phones via Bluetooth on a Nokia N95\(^2\) cameraphone. Users can trigger image or video capture by clicking on a dedicated button on the remote controller. The cameraphones in the Glogger system also function as a computational unit for analyzing signals from the integrated sensors or on-body sensors and controllers that are connected wirelessly.

### 4.2 Implementation of the Glogger Mobile Application

This section presents the design and implementation of the Glogger Mobile Application, the client-side cameraphone program that is designed to capture one’s life on a continuous basis without any user intervention. The cameraphone could be worn by the user (see Figure 3.2 for some of the wearable prototype configurations) and it automatically collects information about the user’s interactions with various sensors.

\(^1\)Controllers for Nintendo Wii. ([http://www.nintendo.com/wii/what/controllers](http://www.nintendo.com/wii/what/controllers))

\(^2\)Nokia Nseries. ([http://europe.nokia.com/find-products/nseries](http://europe.nokia.com/find-products/nseries))
Figure 4.2: Screenshots of the Glogger Mobile Application and the Glogger Server Interface. The server is a central data repository for content sharing. Users can access the latest videos and images directly from the cameraphones with the GUI provided by the Glogger Mobile Application or through a web browser.
Table 4.1: Comparison of hardware specifications between the Apple iPhone 3GS cameraphone and the Microsoft SenseCam v2.3 wearable device [25].

<table>
<thead>
<tr>
<th>Hardware Specification</th>
<th>iPhone 3GS</th>
<th>SenseCam v2.3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Camera Resolution</td>
<td>Up to 3.2 MP</td>
<td>VGA</td>
</tr>
<tr>
<td>Angle of View</td>
<td>70 degrees</td>
<td>119 degrees</td>
</tr>
<tr>
<td>Audio Recording</td>
<td>Yes</td>
<td>Under Development</td>
</tr>
<tr>
<td>Video Recording</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Passive Infrared(PIR) Sensor</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>GPS</td>
<td>Built-in</td>
<td>External</td>
</tr>
<tr>
<td>Digital Compass</td>
<td>Built-in</td>
<td>No</td>
</tr>
<tr>
<td>Wireless Connectivity</td>
<td>WiFi, Cellular, Bluetooth</td>
<td>Bluetooth</td>
</tr>
<tr>
<td>Proximity Sensor</td>
<td>Yes</td>
<td>No</td>
</tr>
</tbody>
</table>

* Can be used to determine if a person is in the field of view with body heat.

* Can be used to determine if an object is in the user’s proximity.

4.2.1 Platforms

There are thousands of cameraphones available in the market. Among all these cameraphones, there are fortunately only several major mobile platforms, each focusing on a different set of desirable features. It is important to first identify the limitations of and to decide whether mobile devices are truly a suitable choice for lifelong capture. As shown in Table 4.1, the iPhone 3GS, a modern cameraphone developed by Apple, has already met some of the key hardware requirements in the Microsoft SenseCam device, a wearable device which is specially designed for lifelong logging purposes.

Although cameras in modern cameraphones have a narrower field of view (about 70 degrees instead of 119 degrees in SenseCam), companies such as Digital King\(^3\) have

\(^3\)http://digital-king.jp/
invented conversion lenses to extend the field of view of the camera to up to 180 degrees. Alternatively, one can extend the field of view by creating panoramic images as shown in Figure 3.4. In addition to these hardware limitations, mobile phones are also generally not designed for wearable and continuous capture purposes. For example, most cameraphones may go to sleep mode and terminate the current running application. Therefore, it is also important to choose the mobile platforms carefully to allow cameraphones to be programmed to meet these additional requirements. This thesis focuses on three popular mobile platforms, namely Symbian, and iPhone, and J2ME. The limitations of each platform are explained further in Section 4.4. For consistency, the examples shown in the following sections are mainly based on the latest prototype developed for the iPhone 3GS.
4.2.2 System Overview

Figure 4.3 provides a high-level overview of the system design of the Glogger Mobile Application on a modern cameraphone. The data flow of the application is divided into four system operations: (1) collect, (2) analyze, (3) package, and (4) transmit. The first step is data acquisition. The Glogger Mobile Application collects various sensor data from the integrated or remote sensors and stores them in a temporary memory buffer (see Section 4.2.3). The temporary memory buffer is used to minimize overheads in the file I/O operations and to allow for faster memory access in the next step. The second step is data analysis for significant events. The collected data are analyzed to detect significant events such as loud noise and sudden motions as described in Section 4.2.4. The third step is data packaging and archival. The application packages the buffered data into a cyborglog (see Figure 4.5 and Section 4.2.5 for the directory structure and file formats used) and writes into the persistent storage to avoid data loss. The fourth step is the data transmission. The application sends the most recently created cyborglogs to a remote central server for archival and sharing through a standard HTTP POST request. Figure 4.4 summarizes the above steps in a flow chart.

4.2.3 Data Acquisition

The first step is to acquire sensor data from the cameraphone. Typically, cyborglogs are images or videos with text annotations as seen in Figure 1.1. As technology evolves, other sensors such as accelerometers, GPS receivers, and digital compasses are also commonly integrated into cameraphones today, providing much more contextual information that could be used to monitor user status and infer user behaviors. Figure 4.3 shows a list of sensor data inputs the Glogger Mobile Application collect in each iteration.

This rich set of sensor data can serve multiple purposes if they are analyzed intelligently. For example, the accelerometer could be used to detect the user’s motion or
Figure 4.4: A flow-chart of the passive capture algorithm on Glogger.
to determine the orientation of the device. One example usage of the accelerometer is to turn off the logging feature when it is stationary and oriented downward or upward. This simple approach can prevent unnecessary capture, and thus reduce power consumption. Although these data are not extensively analyzed in this thesis, it is still important to capture this valuable information for further analysis using techniques such as those introduced in the Reality Mining project in the future [18].

Temporary Memory Buffer

Sensor data are sampled continuously. One simple approach is to write the sampled data to disk when it is available. However, this approach leads to excessive overheads in each file I/O operation and greatly impacts the performance of the application. A better solution is to cache the sensor data in a temporary memory buffer instead of writing to disk continuously. The temporary memory buffer is reset at runtime and it stores the GPS location data, acceleration data, and compass heading data. When the buffer is full, the application writes the data to local persistent storage. For large data that are sampled at a lower rate (e.g., images), the data are written to persistent storage instead to avoid overflowing the temporary memory buffer.

The buffer size and data sampling rate have several effects on the behavior and execution time of the application. A smaller buffer can reduce the latency in content sharing as the captured data are only transmitted after the buffer is full. However, a small buffer increases the overheads in the data packaging and transmission process. On the other hand, a large buffer can reduce such overheads but can increase the latency in content sharing and also the risk of data loss and corruption. The data sampling rate can impact the responsiveness of the application. A low data sampling rate can improve battery life because it reduces the overall resource consumption. In contrast, a high sampling rate can improve the accuracy of the detection algorithm but can greatly impact the battery life. Therefore, it is logical to adjust these parameters dynamically based on the user’s
context. For example, if the user is at home, the system may sample at a lower rate and only take photos every minute or so.

In the current prototype implementation for iPhone 3GS, the default length of the buffer is set to 30 seconds and sensor data are sampled at 15 Hz (see Figure 4.3). Additionally, audio files are also segmented into 30-second clips.

### 4.2.4 Significant Event Detection

The second requirement is to detect significant events and capture images at a variable frame rate. This feature is particularly useful for pervasive capture because it can reduce unnecessary resource consumption when the user is idle or when the device is not worn (i.e., when no significant events are detected). When significant events are detected, the Glogger Mobile Application will automatically trigger image capture and record these events in a database. Furthermore, to ensure that the captured image is not blurred due to motion, the accelerometer is used to detect shaking. The shutter waits until the cameraphone is stable. This demonstrates how various sensors can be used jointly to create more intelligent wearable applications on mobile platforms. Three simple real-time algorithms for detecting loud noise with a microphone, shaking of the device with an accelerometer, and changing of view point using a digital compass are presented next.

The first algorithm detects shaking with an accelerometer. Listing 4.1 shows the implementation of the algorithm as a simple function written in C. The function computes the absolute difference between acceleration values, and it returns true when the changes are greater than a sensitivity threshold.

```c
// a function that detects camera shake
// float aX, aY, aZ are the acceleration values
4 boolean DetectShift(float aX, float aY, float aZ)
{
    float KDetectShiftThreshold = 0.01;
    boolean iDetectShift = false;
```
float diff = 0;
float diff2 = 0;
diff = fabs(oldX-aX) + fabs(oldY-aY) + fabs(oldZ-aZ);
diff2 = fabs(oldX2-oldX) + fabs(oldY2-oldY) + fabs(oldZ2-oldZ);

if (diff>KDetectShiftThreshold || diff2>KDetectShiftThreshold)
    iDetectShift=true;

// store the old values
oldX2=oldX;
oldY2=oldY;
oldZ2=oldZ;
oldX = aX;
oldY = aY;
oldZ = aZ;
return iDetectShift;
}

Listing 4.1: The camera shake detection algorithm

The second algorithm detects loud noise with a microphone as shown in Listing 4.2. The function takes the peak power reading (in decibels) from the microphone and feeds it through a low-pass filter. If the result is greater than the desired threshold (0.5 by default), then the function returns true.

boolean detectBang (float micPeakPower) {
    const float ALPHA = 0.05;
    const float LOWPASS_THRESHOLD = 0.5;
    // convert decibels to linear
    float peakPowerForChannel = pow(10, (0.05 * micPeakPower));
    // apply low-pass filter
    lowPassResults = ALPHA * peakPowerForChannel +
        (1.0 - ALPHA)*lowPassResults;
    if (lowPassResults > LOWPASS_THRESHOLD)
return true;
return false;
}

Listing 4.2: A simple algorithm which detects sudden loud noise such as banging and screaming

In Listing 4.3, an algorithm for detecting any changes in orientation with a digital compass is shown. The function compares the absolute difference between the current compass reading with the old one that is saved whenever a photo is taken. If the difference is greater than the field of view of the camera (i.e., the camera is looking at a different direction that may not have been captured), then it returns true.

```java
boolean detectViewChange(float new_compass_value) {
    // when the reading is unknown
    if (compass_value < 0) {
        return false;
    }
    // for fish eye lens
    const float VIEW_ANGLE_FISH_EYE = 180;
    // for a normal lens
    const float VIEW_ANGLE_NORMAL = 55;
    // for the amount of overlap needed
    const float OVERLAPS = 0.5;

    if (fabs(old_compass_value - new_compass_value) > VIEW_ANGLE_FISH_EYE * OVERLAPS) {
        return true;
    }
    return false;
}

Listing 4.3: A simple algorithm which detects view changes
```

These algorithms can be easily integrated into the capture loop to trigger image capture as shown in Listing 4.4.
// a callback function, triggered by a timer which runs at 15Hz by default
void TimerCallbackFunction() {
    // get acceleration values from the accelerometer sensor
    float *accel = getAcceleration();

    // get peak power from the microphone
    float micPeakPower = getMicPeakPower();

    // get compass heading from the device
    float compass_heading = getCompassHeading();

    // detect noise
    if (detectBang(micPeakPower)) {
        initCapture = true;
        // log the event to backtrack later
        logEvent();
    }

    // detect view changes
    if (detectViewChange(compass_heading)) {
        initCapture = true;
        logEvent();
    }

    // periodic capture
    if (counter > REFRESH_HZ*PHOTO_TAKEN_PERIOD) {
        forcePhotoTaken = true;
        counter = 0;
    }

    // only take photo if the device is stable enough or override with a counter
    if ((initCapture && !DetectShift(accel[0], accel[1], accel[2])) ||
        forcePhotoTaken) {
        // reset flag...
Listing 4.4: A piece of code which combines the three algorithms from Listings 4.1, 4.2, and 4.3

Despite the simplicity in these algorithms, the sensor-based algorithms described in this section have provided a feasible low-power solution (an important design objective and consideration), thus allowing the application to run much longer.

4.2.5 Data Packaging and Archival

When the temporary memory buffer is full, the sensor data are packaged into a single cyborglog file with the structure shown in Figure 4.5 using the zlib library [17]. There are several advantages to this approach. First, packaging can reduce the overall file size of the cyborglogs, especially since the number of files can be substantial after long-term usage. Second, the periodic packaging also automatically segments the data stream into smaller parts, thus reducing the risk of data corruptions. For example, if the application were terminated due to exceptions, only the last 30 seconds of data stored in the buffer would be lost. Consequently, this improves the reliability of the system.

Figure 4.5 also shows the formats that are used to represent the data inputs. Sensor data are parsed and stored in CSV file format or XML file format, an example of which is shown in Listing 4.5. In our prototype, the XML file format is used for ease of debugging and portability. To save storage space, the CSV file formats can be used instead.
Figure 4.5: Structure of a cyborglog created on cameraphones. A cyborglog is composed of a set of sensor data including images, accelerometer data, GPS data, audio, and additional annotations provided by users (e.g., comments or text description of the events).

```xml
<acceleration>
  <data><x>0.036224</x><y>-0.742599</y><z>-0.633926</z></data>
  <data><x>0.036224</x><y>-0.742599</y><z>-0.652039</z></data>
  ...
  <data><x>0.054337</x><y>-0.742599</y><z>-0.670151</z></data>
</acceleration>

<heading>
  <data>
    <trueHeading>186.522097</trueHeading>
    <headingAccuracy>25.000000</headingAccuracy>
    <magneticHeading>197.000000</magneticHeading>
  </data>
  ...
  <data>
    <trueHeading>209.522097</trueHeading>
    <headingAccuracy>25.000000</headingAccuracy>
    <magneticHeading>220.000000</magneticHeading>
  </data>
</heading>
```
Listing 4.5: An example of various sensor data captured by the Glogger Mobile Application and converted to XML file format for portability

4.2.6 Content Sharing

To share cyborglogs with others, users can transmit the content to the Glogger Server through the HTTP POST protocol, Multimedia Messaging Service\(^4\) (MMS) protocol, or Simple Mail Transfer Protocol (SMTP) [43]. In our prototypes, the HTTP POST protocol is used extensively because of its flexibility and its widespread adoption in most mobile platforms such as Java, Symbian, and iPhone. Listing 4.6 shows an example of transmitting data to Glogger Server through HTTP POST protocol in Java. The construction of the HTTP POST request is rather generic and can be easily ported to other platforms.

\(^4\)Multimedia Messaging Service V1.3 (http://www.openmobilealliance.org/Technical/release_program/mms_v1_3.aspx)
public int uploadImage(byte[] byteImage, String serverName, String userName, String password, String commentString) {
    String functionString = "nothing";
    try {
        HttpConnection c = null;
        DataInputStream is = null;
        DataOutputStream dstream = null;
        StringBuffer b = new StringBuffer();
        // get the device’s system property
        String sysProp=getSystemPropertyString();
        try {
            c = (HttpConnection) Connector.open("http://www.glogger.mobi/upload/upload.php");
            // Set the request method and headers
            c.setRequestMethod(HttpConnection.POST);
            c.setRequestProperty("Connection", "Keep-Alive");
            c.setRequestProperty("User-Agent", "Profile/MIDP-2.0 Profile/CLDC-1.1");
            c.setRequestProperty("Content-Type", "multipart/form-data; boundary=****4353");
            // Getting the output stream may flush the headers
            dstream = new DataOutputStream(c.openOutputStream());
            dstream.write("---****4353\r\n".getBytes());
        }
    }
// Send in the username

dstream
    .write("Content-Disposition: form-data; name=username"
          + "\r\n\r\n" + userName.trim() + "\r\n" + "-----4353\r\n"
          .getBytes());

dstream
    .write("Content-Disposition: form-data; name=password"
          + "\r\n\r\n" + password.trim() + "\r\n" + "-----4353\r\n"
          .getBytes());

dstream
    .write("Content-Disposition: form-data; name=systemProperty"
          + "\r\n\r\n" + sysProp.trim()+ "\r\n" + "-----4353\r\n"
          .getBytes());

dstream
    .write("Content-Disposition: form-data; name=comment"
          + "\r\n\r\n" + commentString.trim() + "\r\n" + "-----4353\r\n"
          .getBytes());

dstream
    .write("Content-Disposition: form-data; name=function"
          + "\r\n\r\n" + functionString.trim() + "\r\n" + "-----4353\r\n"
          .getBytes());

dstream
    .write("Content-Disposition: form-data; name=\"uploadedfile\""
          + "filename=\""
          + "abc.jpg"
          + "\r\nContent-Type: image/jpeg\r\n\n"
          .getBytes());

// now send the image

int index = 0;
```java
int size = 1024;

do{
    if((index+size)>byteImage.length) {
        size = byteImage.length - index;
    }
    dstream.write(byteImage, index, size);
    index+=size;
} while(index<byteImage.length);

dstream.write("\r\n\r\n\n").getBytes();

// get the return message from the server
is = c.openDataInputStream();
int ch;
while ( (ch = is.read()) != -1) {
    b.append((char) ch);
}
String result = b.toString().trim();
if(result.equals("0")) {
    return GLOGGER_OK;
} else if(result.equals("1") || result.equals("2")) {
    return GLOGGER_BAD_PASSWORD;
} else if(result.equals("3")) {
    return GLOGGER_ZEROBYTES;
} else {
    return GLOGGER_BAD_CONNECTION;
}
}

} catch (Exception e) {
    return GLOGGER_BAD_URL;
}

} finally {
    if (is != null)
        is.close();
    if (dstream != null)
```
Listing 4.6: An example function for sending images to the Glogger Server using Java in real-time. The body of the HTTP POST request is composed of the username, the password, and the raw bytes of the image data. A `multipart/form-data` type is used to encode the message and other sensor data such as location information can also be appended to the message.

4.3 Implementation of the Glogger Server

In this section, we present the design of the Glogger Server which handles the archival, retrieval, and sharing of cyborglogs.

4.3.1 Overview

The Glogger Server is a web server with Linux, Apache 2, MySQL, and PHP installed. A set of multimedia codec libraries such as FFmpeg\(^5\) are also installed to allow for multimedia file conversions. Figure 4.6 shows an overview of how cyborglogs are handled in the Glogger Server. The Glogger Engine, which is the core of the server mainly written in PHP, is responsible for handling user logins, conversion of multimedia files, commenting of content, as well as rendering of the content. Most importantly, the engine is designed to automate time-consuming tasks in managing the incoming stream of data. Users can

\(^5\)FFMpeg is an open source tool for converting between different multimedia file formats (available at [http://ffmpeg.org/](http://ffmpeg.org/)).
simply enable continuous capture on their mobile client and images or videos will be automatically uploaded and archived to the Glogger Server. These images and videos can be viewed by others in real-time on mobile devices or desktop computers through the Glogger web interface. On the other hand, the Glogger Database is responsible for indexing the data for efficient retrieval with an SQL database.

4.3.2 Glogger Engine

One requirement of the Glogger Engine is to allow users to access cyborglogs from anywhere on a variety of devices ranging from mobile phones to desktop computers. This requires a design which automatically converts cyborglogs to a more widely compatible format. Cyborglogs are a stream of multimedia (typically image, video, and audio) files augmented with additional sensor data such as the GPS location and acceleration values. Since the raw data received from different users are in various multimedia formats and sizes, it is important to convert these data to a mobile-compatible and web-
compatible format to ensure consistent user experiences. The Glogger Server supports a number of common multimedia file formats as shown in Table 4.2. Since ImageMagick (http://www.imagemagick.org/script) and FFmpeg (http://www.ffmpeg.org) are used for image conversion and video conversion, respectively, the Glogger Server can potentially support over 100 different image and video formats accordingly to the specifications list in these two multimedia libraries.

Figure 4.6 shows the generic workflow of the Glogger Engine. First, the user identification information, timestamps, and the raw data are extracted from the HTTP POST request. Once the user identification is verified, the raw data are then moved to the user directory. To organize the increasing number of cyborglogs, user directories are created dynamically based on the timestamps of the cyborglogs as shown in Figure 4.7. This way, all cyborglogs created on the same day are stored in the same directory to allow for easier retrieval and organization. To allow the cyborglogs to be viewed on a wide range of devices, the Glogger Engine converts the raw data to a more mobile-compatible format. For example, all audio files are converted to the MP3 file format to allow for real-time HTTP streaming. Finally, a new entry is inserted into the SQL database along with an update to the user’s status.
4.3.3 Glogger Database

To efficiently retrieve records from the Glogger Server, all cyborglogs are managed with an SQL Database Server. The use of an SQL database allows cyborglogs to be linked to one another with tags, storyboards, or other collection structures. The database is divided into tables that are customized for handling different data. The key tables in the Glogger Server include a *user* table which handles user account information such as the user id and password, a *tag* table which links cyborglogs and allows for efficient text search, and a *cyborglog* table which handles the metadata such as timestamps and location data.

All raw data are stored in the file system with the directory structure and naming convention shown in Figure 4.7. Each record in the Glogger Server is created based on timestamps to ensure no overlaps occur. The hierarchical structure also provides an intuitive way for navigating through the records.

Currently, there are about 80,000 images and videos uploaded to the Glogger Server by users around the world. To demonstrate the widespread use of Glogger, some of the latest images and videos uploaded to Glogger are shown in Figure 4.8. Based on the distribution observed in the world map interface, there are more users using the Glogger Mobile Application in the United Kingdom and Asia than in North America, possibly due to the relatively high mobile phone penetration rate in the United Kingdom or due to the relatively expensive data plans in North America.

4.4 Evolution of Glogger

Over the past few years of continuous development, the Glogger Mobile application evolved from a simple one-click capture and upload image sharing application to today’s context-aware wearable application which can last for several hours under continuous operation. This section shows the interface design and the features incrementally enabled
Figure 4.7: The User Directory Structure of the Glogger Server Database. This illustrates how files are hashed into a directory based on the timestamp. However, in the initial design, only a single directory is created per user on the Glogger Server. This approach is only adopted in the iPhone prototype to organize the data in the local database.
4.4.1 Glogger Prototypes

To demonstrate the practicality and extensibility of the Glogger framework, three different prototypes of the Glogger Mobile Application\(^6\) were implemented on three major mobile platforms (J2ME, Symbian, and iPhone). The prototype applications that have been released and tested include Glogger Live v7 for J2ME-enabled mobile phones, Glogger VS2 for Nokia phones with Symbian OS (S60), and iGlogger for Apple iPhone with iPhone OS 3.1. To support the interactions between these different versions of Glogger, a web service is established and made available at http://www.glogger.mobi. A brief summary of the challenges and features added to Glogger are described in the following sections.

\(^6\)The source code of Glogger is available for download from the public cvs repository at http://www.eyetap.org/viewvc
4.4.2 Glogger Live (for J2ME-enabled Phones)

The initial prototype described in Chapter 3 is a Java Mobile Information Device Profile (MIDP) 2.0 application that runs on J2ME-enabled phones. Since then, this application has undergone several major releases and a number of modifications to support various new features and to fix a number of usability and compatibility problems.

The first feature, which is also one of the novel features at that time, is called the one-click image capture and sharing. This feature allows users to capture, archive, and share their experiences with a simple click of a button. The interface for this feature, which is illustrated in Figure 4.9, is designed with simplicity in mind to minimize user distraction.

To access the camera on J2ME-enable phones, a package called Mobile Media API (MMAPI)\(^7\) is required to support additional multimedia features such as image capturing and audio recording. Listing 4.7 shows the source code for initiating the camera and displaying on screen using the MMAPI in Java. Notice that the camera initiation steps varies

\(^7\)Java ME, Mobile Media API (MMAPI); JSR 135. (http://java.sun.com/products/mmapi/)
across different mobile manufacturers. For example, Sony Ericsson mobile phones uses the locator “capture://video_audio” instead of the standard locator “capture://video” to initialize the camera with video recording feature. Once the camera is initialized, it is now possible to capture an image with the function as shown in Listing 4.8.

```java
1 public void initPlayer() {
    if (myPlayer == null) {
        PreferenceNoForm pf = new PreferenceNoForm();
        // Set up the camera according to the specification
        if (pf.getPhoneBrand().equals("Nokia")) {
            myPlayer = Manager.createPlayer("capture://video");
        } else if (pf.getPhoneBrand().equals("Motorola")) {
            myPlayer = Manager.createPlayer("capture://video?encoding=rgb565&
            width=160&height=120");
        } else if (pf.getPhoneBrand().equals("Sony Ericsson")) {
            myPlayer = Manager.createPlayer("capture://audio_video");
        } else {
            myPlayer = Manager.createPlayer("capture://video");
        }
        myPlayer.realize();

        // Grab the video control and set it to the current display.
        vc = (VideoControl) myPlayer.getControl("VideoControl");

        if (vc != null) {
            vcRef = append((Item) vc.initDisplayMode(VideoControl USE_GUI_PRIMITIVE,
            null));
        }
        myPlayer.start();
        isInit = true;
    } else {
        myPlayer.prefetch();
    }
```
The second important feature is to create storyboards (narratives) collaboratively directly on the cameraphone as shown in Figure 3.5. Such a feature allows two or more
users to publish an event at the same time from their own perspectives. Figures 4.10 and 4.11 show the interfaces for creating and viewing storyboards respectively on a cameraphone.

Additional features such as continuous capture and video broadcasting are also implemented in later releases as demonstrated in Figure 4.12. The algorithm for broadcasting video is similar to the continuous logging algorithm illustrated in Figure 4.4. The video stream is segmented into 30 second buffers and transmitted to the Glogger Server. The server then transcodes the video buffers to FLV file format for real-time HTTP streaming. Despite the extra latency (approximately 45 seconds when a 30 second buffer is used) in this approach, the buffering approach has created a rather smooth playback experience. The extra latency has little effect on the overall user experience mainly because remote viewers are not able to notice the small delay.

The use of the Java platform has allowed Glogger to reach a large number of users by allowing Glogger to run on a large number of phones. Based on the statistics gathered from GetJar.com (a popular mobile application sharing website), approximately 170,000 users downloaded and installed the Glogger Live v7 application on their mobile phones. Among these users, around 70,000 users joined Glogger through the on-device registration process in the Glogger Live v7 application. However, a number of users have reported that their phones are incompatible with the MMAPI. Fortunately, at the time of writing, the support of the MMAPI library is much more common on most modern cameraphones.

Despite the portability of the Java platform, there are several fundamental issues prohibiting further development of Glogger on this platform. The first issue is the image acquisition latency on these J2ME-enable phones [13]. Image acquisition is one of the most important steps in the continuous lifelong capture. In addition, the relatively poor performance (compare to Symbian) and the lack of low-level feature support has

---

8Glogger Live for Mobile / Free Download (available at http://www.getjar.com/mobile/6448/glogger-live)
Figure 4.12: A web interface for viewing the live video feed from the wearable camera worn by the author during the CyborGLOGGER Performance of Globaleyesation [34] for ACM SIGGRAPH 2007 Art Gallery. Peers from Toronto were able to experience the event remotely from the author’s point of view and comments on the event as it was happening through the Glogger Server.
halted further development of any performance critical application on this platform. To overcome these limitations, a different platform, namely the Symbian OS, is explored next.

### 4.4.3 GloggerVS2 (for Symbian Phones)

The second prototype of Glogger is developed on the Symbian OS (S60 platform). Since this platform allows developers to create *native* Symbian C++ applications, there is a noticeable gain in performance compared to the Java platform, particularly in imaging related tasks [13, 26]. One key objective of developing the GloggerVS2 application is to utilize various sensors on the phone and create a wearable application that can capture one’s life in a *smarter* way. Instead of blindly capturing images in a fixed interval as shown in the last chapter, this application also records location data and utilizes the accelerometer to detect camera shake to reduce image blur.

The first feature explored on this platform is the use of an accelerometer to detect camera shake and hold off the shutter until the camera is stable. A screenshot of the detector running in real-time is shown in Figure 4.13. This feature is designed to reduce the chance of capturing blurry images when the cameraphone is worn.

Moreover, the new Glogger application supports important camera control features such as auto-focus and manual exposure adjustments similar a hand-held camera. Figure 4.14 shows a screenshot of the main interface of GloggerVS2. Notice that the interface is similar to the one-click capture interface as shown in Figure 4.9.

The GloggerVS2 application is also released to the public, allowing users around the world to capture and share images to Glogger in real-time. Figure 4.15 shows the distribution of users who have shared images to the Glogger Server with the GloggerVS2 application.

In addition to better programming support, modern cameraphones such as the Nokia N95 also come with NTSC TV output that allows the cameraphone to output video
Figure 4.13: (a) The camera is stable and the 'Ready to Capture' notification is shown on the left corner of the screen. (b) The camera is shaking and the 'Camera is Shaking' warning is shown instead.

Figure 4.14: The main interface of GloggerVS2.
Chapter 4. Glogger Framework

Figure 4.15: A map of users who contributed content to Glogger with GloggerVS2 (the Symbian version of Glogger) around the world.

signals to a wearable display as shown Figure 4.16. Figure 4.17 shows the head-mount configuration of a wearable setup with the Nokia N95 cameraphone. Despite the unusual appearance of the wearable setup, this configuration allows a person to record from a first-person perspective (i.e., record what a person sees). This setup allows a person to concentrate on an event or activity while at the same time providing the person with instant feedback from the system. However, the head-mount setup adds extra complexity and obtrusiveness, thus reducing its mainstream appeal in comparison with the neck-worn prototype.

4.4.4 iGlogger (for iPhone)

A wearable prototype of Glogger has also been developed on the Apple iPhone platform. This platform allows developers to create native applications that are written in Objective-C [28]. Fortunately, the iPhone SDK allows source codes to be written in
Figure 4.16: An example of camera-aiming (viewfinder) through the wearable eyepiece.

Figure 4.17: (a) The Nokia N95 cameraphone is bonded to a head-mount. The cameraphone sends the video signal to a wearable eyepiece through the NTSC cable connected to the side of the phone. (b) A photo of the wearable setup taken at a different angle. Notice that the Nokia N95 cameraphone is powered by an external power supply for a day-long use.
native C or C++, allowing rapid development and portability of the source code.

In Figure 4.18, two screenshots of the prototype interface are illustrated. Users can start and stop continuous capture by pressing on the button at the bottom left corner. To avoid triggering recording accidentally, the proximity sensor and accelerometer are used to automatically lock the touch screen when the phone is in a worn position (i.e., the phone is leaning against the body of the user).

Several power saving features have also been implemented in this prototype to extend the battery life. The first power saving feature is to shut off the screen with the proximity sensor. Also, an accelerometer is used to turn off image capture when the device is oriented upward or downward for a long period of time (meaning a device might be on a table idling). Other context-aware approaches can be used to reduce the chance of unnecessary capture for power saving purpose. For example, one may want to turn off the logging automatically when the user is idle (e.g. writing a thesis with a laptop, watching TV, or sleeping). However, during the pilot study, a significant amount of customization is still needed to determine the ‘best time’ for capturing an event. Passive capture with simple sensor-based algorithm is found to be effective in most cases.

It was found that the power consumption of the device increases dramatically when the real-time sharing feature is enabled. By turning off real-time sharing, the application can run continuously for 3 hours, capture around 500 images, use about 100 MB of disk space with all sensor data recorded on a full charge of battery.

**Mediated Reality on iPhone**

Finally, an application demonstrating the idea of mediated reality, *Wayfinder*, has also been developed using a combination of sensors on the iPhone 3GS. This prototype opens up one possible extension of Glogger – a wearable mobile application that assist a person’s day-to-day life through mediated reality.

The interface of this application is intuitive. Users can navigate and find directions
Figure 4.18: (a) A screenshot of the iGlogger prototype interface on the iPhone 3GS. On the left, there is some vertical text displaying the various sensor values for debugging purposes. Overlaid at the bottom of the screen is a graph of the accelerometer values. (b) A screenshot of the interface when a significant event is detected. At the center of the screen is a caption indicating that a significant event was detected by the audio level and recorded in the database.
Figure 4.19: A picture of the Wayfinder application running on the iPhone 3GS. A 3D map is rendered in OpenGL and overlaid onto the viewfinder. Information such as street names, building names, as well as direction are also displayed.

and names of buildings on a virtual map by looking through the viewfinder as shown in Figure 4.19. In the current prototype, street names, building names, as well as a 3D map are rendered and superimposed on top of the real-world scene. The use of mediated reality on mobile phones can open up a new era of human computer interaction in the wearable and mobile computing domain.

4.5 Conclusions

This chapter presented the design and implementation of the Glogger framework which enables modern cameraphones to passively capture one’s life. The design of the server-
client framework was explained showing how various components work in the Glogger system. Three different prototypes were presented to show the evolution of Glogger from a simple real-time image sharing application to a wearable cameraphone application that can detect changes in the environment with various sensors. Despite the rapid advents in mobile technologies, the framework presented in this chapter is able to keep up due to the adoption of standard protocols and formats.
Chapter 5

Glogger Extensions

In this chapter, we present one possible extension of the Glogger application – a feature-based tracking algorithm. In particular, we describe the implementation of Lowe’s Scale-Invariant Feature Transformation (SIFT) [30] on J2ME-enabled mobile phones.

5.1 Introduction

Scale-Invariant Feature Transformation (SIFT) is a well-known algorithm for detecting and extracting local feature descriptors that are invariant to changes in illumination, image noise, rotation, scaling, and in viewpoint [30]. One application of this algorithm is to construct image panoramas from a set of random images [10]. In Chapter 3, we presented a server-side approach that allows users to create panoramic images in real-time on the Glogger Server using the video orbits algorithm [40]. In this chapter, we explore a client-side approach by first implementing the SIFT algorithm as a building block for performing real-time panorama on mobile phones in the future.

However, running the full-feature (i.e., multi-octave and multi-scale) SIFT algorithm is a computationally intensive task that may not be feasible due to the limited resources on a mobile phone. To minimize the computational resource usage, we first analyze the effect of each parameter on the execution time of the SIFT algorithm and then we select...
the appropriate parameter setting for implementation on mobile phones. Furthermore, we introduce several strategies to minimize the runtime memory usage and identify the resource bottlenecks in the SIFT algorithm.

This chapter is organized as follows. First, we show the necessary steps for acquiring images from a cameraphone on the J2ME platform and preparing these images for the SIFT algorithm. Second, we unfold the key steps in the implementation of the SIFT algorithm: keypoint localization, local extrema detection, and SIFT keypoint descriptor creation. Third, we present a summary of the time and space complexity analysis of each key step in the SIFT algorithm to identify the potential resource bottlenecks. Finally, we show the experimental results and future applications of this work.

5.2 Design Challenges and Related Work

Programming imaging algorithms on mobile phones is often a challenging task because of the memory and computational constraints on such platforms. Most computationally intensive applications cannot be directly ported to the mobile platform and it is often necessary to redesign an algorithm to reduce its complexity [3, 15, 44]. One possible solution is to reduce the resolution of input images and disable other features to improve performance at the expense of accuracy and robustness. In this section, we show the full implementation details of the SIFT algorithm along with some of the possible optimizations that can be made to reduce the required computational resources, thereby allowing SIFT to run on mobile platforms.
5.3 Image Acquisition and Data Preparation

In the first step, we acquire an image from the cameraphone using the Mobile Media API\(^1\) (MMAPI). Listing 5.1 shows a Java source code for capturing a 320x240 image from the cameraphone and converting it to a one-dimensional array of pixel data in RGB values.

```java
int width = 320;
int height = 240;
byte[] raw = mVideoControl.getSnapshot("encoding=jpeg&width=\"width\"&height=\"height\";
Image image = Image.createImage(raw, 0, raw.length);
byte[] src_image;
src_image = new int[width * height];
// get the RGB value and store it in the src_image array
image.getRGB(src_image, 0, image.getWidth(), 0, 0, image.getWidth(),
image.getHeight());
```

**Listing 5.1:** A simple demonstration of capturing images with the Mobile Media API. mVideoControl is an instance of the VideoControl class. An instance of the image is created using the Image.createImage() function with the raw bytes received from the camera. Then, the image is converted to an RGB image with the getRGB() function.

After the image is acquired, the input image is converted to a grayscale image by extracting the green channel using the bitwise shift operator (\(\gg\)) and AND operator (\&). The grayscale image is stored in a 16-bit integer array as shown in Listing 5.2. Notice that the grayscale image is multiplied by 256 to extend the range from 0-255 to 0-65536.

```java
/**
 * Convert to grayscale by taking the green channel.
 */
public void convertToGray() {
```

---

\(^1\)http://java.sun.com/products/mmapi/
```java
int tmp;
int rgb;
for (int i = 0; i < src_image.length; i++) {
    rgb = src_image[i];
    tmp = ((rgb >> 8) & 0xff) * 256;
    // since tmp ranges from 0–255, we multiply this by
    // 2^8 such that it will range from 0 to 2^16 = 65 536.
    // overwrite the old value
    src_image[i] = (tmp);
}
```

Listing 5.2: This code demonstrates the conversion of color images into grayscale images as well as the usage of the bit shift operator and binary mask in Java.

## 5.4 Scale-Invariant Feature Transformation

In the following sections, we show the necessary steps in extracting SIFT features from the image acquired from the cameraphone in the last step.

### 5.4.1 Keypoint Localization

The first step in the SIFT algorithm is to locate stable keypoints that are invariant to image translation, scaling, rotation, and have high tolerance to noise and distortion [29]. The multi-scale and multi-octave approach, described in [30], provides a higher repeatability when matching against a large database of feature. However, at the same time, performing these operations on a mobile phone could be infeasible. To reduce the computational resource, we have adopted the simpler approach stated in Lowe’s earlier paper where only 1 scale is used [29].
Difference-of-Gaussian (DoG) Pyramid

To detect locations that are invariant to scale changes (e.g., changes in the size of an image), we can search for stable features across all possible scales, using a continuous function of scales known as scale space [30]. To create the scale space of an image, the image is convolved with a variable-scale Gaussian:

\[
G(x, y, \sigma) = \frac{1}{2\pi\sigma^2} e^{-\frac{(x^2+y^2)}{2\sigma^2}}
\] (5.1)

where \((x, y)\) is the pixel location and \(\sigma\) is the standard deviation of a Gaussian function. Since the 2D Gaussian function is separable, we can reduce the complexity of the convolution operation from \(O(m \times n \times N \times M)\) to \(O(n \times N \times M) + O(m \times N \times M)\), where \(m \times n\) defines the dimension of the kernel filter and \(N \times M\) defines the dimension of the input image, by applying two passes of the 1D Gaussian function below in the horizontal and vertical directions separately.

\[
g(x, \sigma) = \frac{1}{\sqrt{2\pi\sigma}} e^{-\frac{x^2}{2\sigma^2}}.
\] (5.2)

The Difference-of-Gaussian Image \(G(x, y, \sigma)\) can be obtained as follows.

\[
D(x, y, \sigma) = (G(x, y, k\sigma) - G(x, y, \sigma)) \ast I(x, y)
\] (5.3)

\[
= L(x, y, k\sigma) - L(x, y, \sigma)
\] (5.4)

where \(L\) is the image produced from the convolution of a variable-scale Gaussian, \(G(x, y, \sigma)\), with the input image, \(I(x, y)\), and \(k\) is a constant factor. The value of the constant factor \(k\) can be determined by \(k = 2^{\frac{1}{s}}\) where \(s\) is the number of scales in an octave (a set of scale space images). To obtain another octave, the Gaussian image is down-sampled by a factor of 2 using a bilinear interpolation, and then the process is repeated.

The runtime and space complexity of the operations defined in Equation 5.4 mainly depends on the image size. To reduce the execution time and space usage, we have not doubled the size of the original image as suggested in Lowe’s paper [30] to obtain
additional keypoints. Also, we have used only 1 scale and created 1 octave for our experiments to reduce the memory usage. The extra scales and octaves, however, are critical to the stability and robustness for matching images that are taken from arbitrary viewpoints.

5.4.2 Local Extrema Detection

We can detect the local maxima and minima of $D(x, y, \sigma)$ by comparing each sample point with its eight neighbors in the same scale as well as the nine neighbors in the scale above and below. The point is selected as a keypoint candidate if its value is larger or smaller than all of these neighbors. In our experiment, only three Difference-of-Gaussian images are created, which is the minimum required. Although this reduces the repeatability of the algorithm as fewer potential keypoints are extracted, the additional speedup and reduction in both space and time complexity of the algorithm are highly desirable with the resource-constrained mobile platform.

5.4.3 Filtering Keypoints

Once a keypoint candidate has been found, we apply two thresholds to eliminate points that have low contrast and are poorly localized at the long edge of the image. To eliminate keypoint candidates with low contrast,

$$D(x) = D + \frac{1}{2} \frac{\partial D^T}{\partial x} x$$  \hspace{1cm} (5.5)

can be used for rejecting unstable extrema with low contrast where $D$ is the extremum value and $x$ is the location of the extremum. In our experiment, all extrema with a value of $|D(x)|$ less than 0.05 were discarded assuming that image pixel values were in the range $[0, 1]$.

To eliminate long-edge responses, we reject all keypoints with a large principal curvature. The principle curvature can be computed from the eigenvalues of the 2x2 Hessian
matrix, $H$:

$$
H = \begin{bmatrix}
  D_{xx} & D_{xy} \\
  D_{xy} & D_{yy}
\end{bmatrix}
$$

(5.6)

where $D_{xx}$, $D_{xy}$, and $D_{yy}$ are the second-order partial derivatives of the function computed at the location and scale of the keypoint. Since the eigenvalues of $H$ are proportional to the principle curvature, only their ratios need to be taken into account.

To check if the ratio of principle curvature is below a threshold, we only need to compare if

$$
\frac{\text{Tr}(H)^2}{\text{Det}(H)} < \frac{(r + 1)^2}{r}
$$

(5.7)

where $H$ is the Hessian matrix and $r$ is the threshold variable. Both of the partial derivatives in Equations 5.5 and 5.7 can be computed efficiently using pixel differences as shown in Listings 5.3 and 5.4. The computation time of this step is rather low (approximately 20 floating point operations) and contributes very little to the overall resource consumption.

```
1 /* Compute the gradient. */
Dx = 0.5 * (at(+1,0,0) - at(-1,0,0)) ;
Dy = 0.5 * (at(0,+1,0) - at(0,-1,0)) ;
Ds = 0.5 * (at(0,0,+1) - at(0,0,-1)) ;

5 b[0] = -Dx ;
7 b[1] = -Dy ;
9 b[2] = -Ds ;
11 /* Compute the score for filtering low contrast */
score = at(0,0,0) + 0.5 * (Dx * b[0] + Dy * b[1] + Ds * b[2]) ;
```

**Listing 5.3:** Pseudo-code for computing the threshold for eliminating low contrast keypoints using pixel difference approximation. The $at(x, y, s)$ function returns the pixel value at position $(x, y)$ and at scale $s$.

```
1 /* Compute the Hessian. */
Dx = 0.5 * (at(+1,0,0) - at(-1,0,0)) ;
```
Figure 3: This figure shows several screenshots of the J2MES IFT application running on the Sun Java™ Wireless Toolkit emulator. A synthetic input image (a circle) is used to demonstrate the results in each step of the SIFT algorithm. The results of Gaussian Blur, Difference of Gaussian, and Gradient and Angle Estimation are shown in order from left to right.

Figure 4: Left – The initial 912 keypoints locations at maxima and minima of the difference-of-Gaussian function. Middle – 201 keypoints remain after applying threshold on minimum contrast. Right – 161 keypoints remain after applying the threshold on the contrast and threshold on the ratio of principal curvatures.

Figure 5.1: Screenshots of the application running on the Sun Java Wireless Toolkit emulator. Left: The initial 912 keypoint locations at the maxima and minima of the Difference-of-Gaussian function. Middle: 201 keypoints remain after applying threshold on minimum contrast. Right: 161 keypoints remain after applying the threshold on the contrast and threshold on the ratio of principal curvatures.

3 \[ Dy = 0.5 \times (at(0,+1,0) - at(0,-1,0)) ; \]
4 \[ Dxx = (at(+1,0,0) + at(-1,0,0) - 2.0 \times at(0,0,0)) ; \]
5 \[ Dyy = (at(0,+1,0) + at(0,-1,0) - 2.0 \times at(0,0,0)) ; \]
6 \[ /* Compute the score for principle curvature */ \]
7 \[ score = (Dxx+Dyy)*(Dxx+Dyy) / (Dxx*Dyy - Dxy*Dxy) ; \]
8 \[ threshold = (r+1)(r+1)/r ; \]

Listing 5.4: Pseudo-code for computing the threshold for eliminating long-edge responses using the pixel difference approximation approach.

In our experiment, we found that it is usually better to set a high contrast threshold because images acquired from the cameraphone are usually noisier and suffer from artifacts such as aliasing. Also, a higher threshold can increase the performance of the application because there are fewer keypoints descriptors to construct and match in later steps. The effect of these operations are shown in Figure 5.1.
5.4.4 SIFT Keypoint Descriptor

In this section, we discuss the implementation and complexity of creating a SIFT keypoint descriptor. In our implementation, we have used the same parameter settings as suggested in Lowe’s paper where a 4x4x8 =128 elements feature vector is used for each keypoint [30]. The key steps for computing a SIFT descriptor are gradient and angle estimation, the peak detection, and keypoint description.

Gradient and Angle Estimation

The magnitude of the gradient, \( m(x, y) \), can be estimated using the sum of the squares of the pixel differences on the Gaussian smoothed image, \( L \), with the scale:

\[
m(x, y) = \sqrt{(L(x+1, y) - L(x-1, y))^2 + (L(x, y+1) - L(x, y-1))^2}.
\] (5.8)

Similarly, we can estimate the orientation, \( \theta(x, y) \), of the gradient using the pixel differences as follows:

\[
\theta(x, y) = \arctan((L(x+1, y) - L(x-1, y))/(L(x+1, y) - L(x-1, y))).
\] (5.9)

Two important floating point functions used in these steps are the square root and \( \arctan \) functions. We can see that a potential speedup can be obtained by applying a look-up table for the \( \arctan \) function as an example.

Peak Detection - Orientation Histogram

The orientation histogram is constructed by first creating an array of \( n \) elements where \( n \) is the number of bins in the histogram. We have used a 36-bin histogram for peak detection which covers every 10 degrees. To reduce the computation time, we pre-compute the gradient image, angle image, and also the Gaussian windows for the weight of the image patch in advance. For this step, we follow exactly the same implementation as in Lowe’s paper [30] except that peak detection is done by selecting the 90\(^{th}\) percentile in the
histogram instead of the 80th percentile to reduce the number of keypoints created at the same location.

**Keypoint Representation**

We encode the image patch into a keypoint descriptor by first rotating the image relative to the dominant orientation in the last step. Then, the descriptor is constructed by summarizing each subsample patch into a histogram. The size of the descriptor is \( n \times n \times o \) where \( n \times n \) is the number of subregions in a descriptor and \( o \) is the number of bins in the orientation histogram in each subregion. To reduce the effects of illuminance changes, each vector in the histogram array is normalized to unit length. This step is rather time consuming when the number of keypoints is large.

**Matching Descriptors**

The matching algorithm used in this experiment is a simple brute-force algorithm which compares every pair of descriptors between two images. We define that two descriptors \( D_1 \) and \( D_2 \) match if and only if the Euclidean distance between two descriptors \( D_1 \) and \( D_2 \) multiplied by 1.5 is not greater than the distance of \( D_1 \) to all other descriptors. We would expect a speedup in this step if a more efficient search algorithm such as nearest-neighbor search [6] is applied. However, since only a small set of images are compared in this experiment, the brute-force algorithm works well as long as the number of keypoints is small.

**5.4.5 Complexity Analysis Summary**

The complexity analysis of all key steps in this algorithm is summarized in Tables 5.1 and 5.2. We can see that the input image size has a significant impact on the runtime and space complexity of the SIFT algorithm, especially for the Gaussian smoothing operation.

In addition, the complexity of keypoint descriptor creation highly depends on the
dimensions of the descriptor and also on the number of keypoint descriptors processed. It is important to minimize these parameters based on the requirements of the application. For example, one may want to use a fewer number of keypoints for orientation tracking to improve the real-time performance. On the other hand, a larger number of keypoints should be used for non time-critical applications such as image panorama stitching because they do not need to provide instant feedback to users.

Since the image size is halved in every octave, the number of octaves does not contribute significantly to the memory usage. We can easily show that the space complexity of the algorithm increases logarithmically to the number of octaves. On the other hand, the space complexity of the algorithm increases linearly to the number of scales in the Difference-of-Gaussian (DoG) Pyramid.

5.5 Results and Applications

Figures 5.1 and 5.2 are screenshots of the application running on the J2ME Wireless Toolkit emulator. We have also successfully installed the application on several different J2ME compatible phones such as the Motorola E6 and the Nokia N82. The application takes about 3 seconds to process a 160x120 image and it takes approximately 2 seconds to match against approximately 300 keypoints on the Nokia N82. The results are also shown in Figures 5.3 and 5.4, demonstrating the robustness of the algorithm under changes in view angle and occlusion in the scene. Despite the relatively low performance, this work has unfolded the key components in the algorithm that can be optimized as well as the key trade-offs in each parameter.

5.6 Going Beyond SIFT

The implementation of the SIFT algorithm shown in this chapter is still not able to achieve the necessary frame rate (e.g.,10-15 fps) for real-time interaction. Based on the
Table 5.1: A summary of the time complexity analysis

<table>
<thead>
<tr>
<th>Operation</th>
<th>Time Complexity</th>
<th>Variables</th>
</tr>
</thead>
<tbody>
<tr>
<td>Image Resize (bilinear)</td>
<td>$O(N \times M)$</td>
<td>$N \times M$ - image dimensions</td>
</tr>
<tr>
<td>Gaussian Blur (separable)</td>
<td>$O(n \times N \times M)$ + $O(m \times N \times M)$</td>
<td>$N \times M$ - image dimensions, $n$ - width of the kernel, $m$ - height of the kernel</td>
</tr>
<tr>
<td>Keypoint Filtering</td>
<td>$O(K)$</td>
<td>$K$ - number of keypoints</td>
</tr>
<tr>
<td>Gradient Estimation</td>
<td>$O(N \times M)$</td>
<td>$N \times M$ - image dimensions</td>
</tr>
<tr>
<td>Angle Estimation (separable)</td>
<td>$O(N \times M)$</td>
<td>$N \times M$ - image dimensions</td>
</tr>
<tr>
<td>Keypoint Descriptor Construction</td>
<td>$O(K \times n^2 \times B)$</td>
<td>$K$ - number of keypoints, $n$ - width of the descriptor, $B$ - number of bins in the orientation histogram</td>
</tr>
<tr>
<td>Keypoint Matching</td>
<td>$O(o \times n^2 \times D^2)$</td>
<td>$o$ - number of orientations, $n$ - width of the descriptor, $D$ - number of descriptors</td>
</tr>
</tbody>
</table>
Table 5.2: A summary of the space complexity analysis

<table>
<thead>
<tr>
<th>Object</th>
<th>Space Complexity</th>
<th>Variables</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input Image</td>
<td>$O(N \times M)$</td>
<td>$N \times M$ - image dimensions</td>
</tr>
<tr>
<td>Difference-of-Gaussian Image</td>
<td>$O(L \times N \times M)$</td>
<td>$N \times M$ - image dimensions, $L$ - number of scales</td>
</tr>
<tr>
<td>Keypoint Descriptor</td>
<td>$O(n^2 \times o)$</td>
<td>$n$ - width of the descriptor, $o$ - number of orientations</td>
</tr>
<tr>
<td>Image Pyramid (Octaves)</td>
<td>Less than $2 \times N \times M$</td>
<td>$N \times M$ - image dimensions</td>
</tr>
</tbody>
</table>

Figure 5.2: Screenshots of the application running on the Sun Java Wireless Toolkit emulator. A synthetic input image (a circle) is used to demonstrate the results in each step of the SIFT algorithm. The results of the Gaussian Blur, Difference-of-Gaussian, and Gradient and Angle Estimation operations are shown from left to right.
Figure 5.3: The final matching results from the application running on the Sun Java Wireless Toolkit emulator. The green lines connect the corresponding keypoints between the two images. (a) Synthetic images are created to demonstrate the robustness of SIFT under image translation. The circle and square are translated to different locations, and the algorithm has successfully matched the circle and square properly. Notice that the keypoint at the bottom right corner of the square fails to match because the edge of the circle was also taken as part of the feature. (b, c) The source images are rotated 15 degrees and matched against the original image. The algorithm successfully matched 15 keypoints between the two images (approximately 80 keypoints for the house image). (d) Matching the Google and Yahoo logos. There are 18 false matches between the two logos out of approximately 200 keypoints.
earlier complexity analysis, we can see that the key performance bottlenecks are the image convolution step and descriptor creation step. More recently, a faster feature tracking algorithm called Speeded Up Robust Features (SURF) [3] have been released. These advancements open up the possibility of implementing real-time feature tracking and scene recognition for outdoor augmented reality on mobile devices as demonstrated by [44].

To experiment with the SURF algorithm on mobile phones, the OpenCV Library\(^2\), which already contains SURF and hundreds of other imaging algorithms, was compiled and ported to the iPhone platform. The SURF algorithm is similar to the SIFT algorithm, except integral images and descriptors based on sum of Haar wavelet responses are used to improve the performance and robustness of the tracking algorithm [3]. With minor optimizations and fine tuning of the parameters (e.g., using low resolution images

\(^2\)The Open Computer Vision Library is available for download at [http://sourceforge.net/projects/opencvlibrary/](http://sourceforge.net/projects/opencvlibrary/)
with 2 scales and 3 octaves), we are able to obtain a close to real-time performance (over 3 frames per second on a 160x120 image with about a hundred keypoints detected on the iPhone 3GS) for orientation tracking using SURF. Even though a higher performance was obtained using the SURF algorithm on the iPhone, the new extension has not yet been integrated into Glogger due to the increased power consumption. Despite the current technological limitations, further investigation of computer vision algorithms can potentially benefit Glogger in numerous ways. For example, one possible application is to continuously analyze the scene to provide annotation about the user’s context in real-time. A future direction for the Glogger application may be to provide smart features to turn Glogger into a truly intelligent wearable mobile application.
Chapter 6

Conclusions

6.1 Summary of Contributions

In this thesis, a real-time computational framework, called Glogger, for continuous life-long capture of personal experience is presented. This framework is built upon the idea of CyborGlogging pioneered by Prof. Mann. By translating his visions and previous work that were originally developed for EyeTap into a practical embodiment on mobile platforms, this work has resulted in a new wearable cameraphone application which captures, archives, retrieves, and shares personal experiences in real-time. Unlike ordinary cameras where users usually only capture on-demand, the proposed wearable cameraphone application aims to record from a first-person perspective and closely monitor a user’s status to detect abnormal events using the built-in microphone, accelerometer, and digital compass, as presented in Chapter 4. This work is a synergy of multiple domains – mobile computing, wearable computing, and pervasive computing. This thesis has addressed not only the key technical challenges of developing an experience capture application on mobile phones for use in our everyday life, but it has also presented the use of Glogger as a tool for exploring the conceptual framework of “equiveillance” discussed in Chapter 3.

An important feature of this framework is its compatibility with commodity mobile
devices that are widely available in the market. This has resulted in a growing community of over 70,000 members at the time of writing. To support the growing community, the web service\(^1\) and mobile applications within the Glogger framework are being improved everyday, allowing more users to join the community and interact with one another. A considerable amount of effort and time have been dedicated to improving user experiences by constantly upgrading existing prototypes and creating new ones to support the growing list of platforms.

### 6.2 Future Work

This work explored the feasibility of using Glogger as a tool to continuously capture, archive, retrieve, and share one’s personal experience in real-time. However, there are still a number of research problems that remain to be addressed. In the following sections, several future directions of research and further improvements of Glogger are outlined.

#### 6.2.1 Improve Significant Event Detection Algorithms

To extend the battery life, the significant event detection algorithms described in this thesis were designed to be primitive. In the future, as technology permits, more sophisticated algorithms and approaches can be taken to improve the system. For example, one approach is to dynamically adjust the data sampling rate and other thresholds in the application based on the user’s behavior. Although users may change these parameter settings manually as needed with the current implementation, it will be better if these settings can adapt to the user’s need automatically without explicitly specifying them. In addition to detecting motion and analyzing audio signals, Glogger may also determine the context of users based on proximity or the physical state of the user with biosensors. Real-time analysis of these data can help determine the optimal moments to capture

\(^1\)Glogger website is available at: [http://www.glogger.mobi](http://www.glogger.mobi)
one's life. On the other hand, there are moments in our life that capturing is either prohibited or not desired (e.g., in the washroom or change room). One future approach is to automatically enable and disable certain recording features under different settings.

### 6.2.2 Mediated Reality and Cyborglogs

The use of mediated reality demonstrated in the latest prototype provides a novel method for navigating through a 3D map in a more engaging way. The development of this prototype also inspires other possibilities of visualizing cyborglogs with the use of mediated reality. Instead of overlaying street information, Glogger may retrieve all cyborglogs that were captured in the proximity and overlay them onto the world. Since the orientation as well as the location of the images taken are also logged by Glogger, it is possible to create *windows* of cyborglogs that one can walk through and vividly *re-experience* what others have seen in the past.
Bibliography


[35] S. Mann and J. Fung. EyeTap devices for augmented, deliberately diminished,
or otherwise altered visual perception of rigid planar patches of real world scenes.  


