

Interactive Advertisements for Smart TV Devices Using Face Detection

Rickard Westerlund

May 2012

Master's Thesis in Computing Science, 30 ECTS credits

Supervisor at CS-UmU: Jerry Eriksson

Supervisor at Dohi Sweden: Olof Brändström

Examiner: Fredrik Georgsson

UMEÅ UNIVERSITY
DEPARTMENT OF COMPUTING SCIENCE
SE-901 87 UMEÅ
SWEDEN

Abstract

This thesis report details a project to extend the AdGateway product developed at Dohi Sweden with a framework that allows advertisers to create interactive advertisements. The key feature of this framework is to provide augmented reality capabilities by taking advantage of face recognition and webcams on smart TV systems. The interactive advertisements are created as applications in JavaScripts which get downloaded on the fly when a user clicks on a banner for that type of advertisement. This implementation will both extend the back-end system to support the distribution of this new type of advertisement as well as the front-end for the supported devices in order to set up the environment and run the JavaScript application.

The Sony Google TV is the primary focus for the implementation but an extensive study is also included in the report to gather information about other possible platforms that could be supported. This report also studies in detail how the performance of face recognition can be improved by performing illumination normalization on input images.

The project could not be completed on time due to limitations in both hardware and software regarding face detection. Nevertheless a framework has been implemented, and solutions for the impediments are discussed in the report.

Contents

1	Introduction	1
1.1	Purpose	2
1.2	Goals	2
1.2.1	Evaluation	2
1.2.2	Design	3
1.2.3	Implementation	3
1.3	Related work	4
1.4	Overview	4
2	Method	5
2.1	Development Method	5
2.1.1	Project Management Tools	7
2.2	In-depth Study	7
2.3	Documenting and Tracking Results	7
3	Smart TV	9
3.1	Introduction	9
3.2	General	10
3.2.1	Social TV	10
3.2.2	Interactive TV	10
3.3	Google TV	11
3.4	Samsung TV	12
3.5	Apple TV	12
3.6	Yahoo! Connected TV	13
3.7	Discussion	13
4	Face Detection	15
4.1	Android Libraries	15
4.1.1	android.graphics.FaceDetector	15
4.1.2	android.hardware.Camera.FaceDetectionListener	16
4.2	OpenCV	16

4.3	JJIL	16
4.4	jviolaJones	17
4.5	OpenIMAJ	17
4.6	Discussion	17
5	Illumination Normalization	19
5.1	Overview	19
5.2	Quotient Image	20
5.2.1	Self-Quotient Image	21
5.2.2	Total Variation Quotient Image	23
5.3	Census Transform	25
5.4	Fused Logarithmic Transform	27
5.5	Discussion	28
6	Results	31
6.1	System Description	32
6.1.1	Framework	33
7	Discussion	35
7.1	Restrictions	35
7.2	Future Work	36
8	Acknowledgements	37
	References	41
A	Example Script	

List of Figures

2.1	The scrum process, where sprints are iterated.	6
5.1	Comparison of QI algorithms on a set of images from the Yale B database[37].	25
5.2	Shows the original images at the top and the results of the Census Transform below[17].	26
5.3	Shows the original image to the left and the results of the fLog transform on the right[25].	28
6.1	How the framework and related components interact with their execution environment. The arrows denote dependencies.	32

Chapter 1

Introduction

Interactive advertisements are like normal advertisements but allow for user interaction by accepting inputs from the user such as button presses and pass these on to the application which responds accordingly. The type of advertisements discussed in this report are banner ads provided by the AdGateway system.

AdGateway is a framework for providing advertisements on devices running on the Android and iOS platforms as well as web pages. This system consists of a front-end that application developers use to place a container on the screen of the application which will get populated by advertisements that gets retrieved from a back-end. These advertisements are retrieved as typical ad banners with an associated JavaScript file URL which will be executed by the front-end. Upon getting a request from the front-end, the back-end will return a targeted advertisement from the information it receives. A wide range of data can be provided to the back-end such as GPS location and weather conditions in order to retrieve a fitting advertisement.

The goal of this project is to extend AdGateway with the capabilities of handling a new type of advertisements that allow for interactivity with the users. In addition to interactivity this new type of advertisement should provide a renderable surface where the interaction can take place. This motivates building a framework for this purpose, where the provider of an advertisement of this kind would attach a script that can handle inputs from the user as well as execute commands for rendering on a surface.

Another major point is that this implementation will focus on smart TV¹ devices, more specifically Google TV but also other devices provided that the devices are similar enough to generalize the framework.

The key point of this implementation is to provide the means for creating an advertisement capable of giving an augmented reality² experience. For this purpose, a camera whether already on the device or an external webcam plugged in, will need to be available. Fur-

¹A “smart TV” is a TV set or set-top-box that typically integrates with the internet and has downloadable applications as well social media features.

²Augmented reality is a live view of the physical world that is augmented with computer-generated elements.

thermore the framework will support the retrieval of face detection data in the executed JavaScript applications as a type of user input.

This work will be done at Dohi Sweden which is a holding company run by Emanuel Dohi.

1.1 Purpose

If the goals are met this implementation will extend the AdGateway system to include more powerful types of advertisement for devices that support them which might prove more interesting to users[11] and garner more hits for an advertiser.

1.2 Goals

In order for this project to be completed a set of rough goals have been worked out to identify what should be produced both theoretically and practically.

1.2.1 Evaluation

During the startup of this project a light study was performed in the area of smart TVs to determine the most suitable platform for the implementation of the framework. Google TV was chosen because it ran on the Android platform and had an update scheduled in time for the implementation that would allow development for the device. Since this device can be developed for using Java, a language having large community support and libraries available, as well as being supported by AdGateway it became a good choice for the prototype. However, since the aim for the project is also to write a framework that can be supported by as many platforms as possible a more exhaustive study needs to be performed to determine additional platforms that need to be taken into consideration for the implementation.

This study will need look into the following details.

- Which manufacturers that produce smart TV devices.
- The platforms different brands use.
- Market share for different devices.
- Services already made available for these devices.
- The availability of cameras or the ability of plugging in a camera.
- Development options.
- How many applications are already available for download.
- Future plans for the different devices.

From this information other platforms for development will be identified and if enough time is given the prototype will be extended to include these.

Another area of research is face detection. Since much of the work relies on detecting faces in a given camera frame a study must be done to find libraries available for face detection in Java. These libraries need to be evaluated in terms of how fast they can process a camera frame and how reliably they can get information about a face from a low quality image. Furthermore, an in-depth study will be done in the area of illumination normalization to look at suitable algorithms for improving detection rates during face detection.

An important goal for the implementation is that it needs to perform well with real-time results. The frame rate for an application built around this framework will be measured on all relevant platforms as well as for different qualities for the captured camera image for a discussion about the results.

1.2.2 Design

The framework is the key component of this project and will receive careful attention to its design. Being a component of the AdGateway framework it must be built in the same HTML-based environment and be able to interact with the execution environment for the devices it is built for.

Another point is that the framework should be powerful enough to be able to expose relevant features for rendering, playing sounds and gathering input from different sources while still ensuring that any application using a particular feature not available for device it is being run on to still function properly.

1.2.3 Implementation

The implementation consists of two parts. One being the framework and one being an advertisement application. The advertisement is a prototype for a company selling glasses, having a banner ad that when clicked will load an application that allows the user to try out glasses in an augmented reality fashion. Face detection is used to properly place a pair of glasses on the user's face, being displayed on a rendering surface.

This work has been divided into the following set of milestones.

1. Get oriented in the Android API, especially the new USB functionality.
2. Connect a webcam to the Google TV and make sure data can be read from it.
3. Investigate how to make a good interactive user experience for the application.
4. Get face detection working on the Google TV using the webcam data.
5. Experiment with illumination normalization to increase detection rates if necessary.
6. Render graphics to the screen using the webcam capture as a background.

7. Begin the implementation of the framework overlaying the new type of advertisements from AdGateway onto the screen.
8. Find a way to make an overlay on the TV screen where AdGateway advertisements can be placed.
9. Rewrite the test application developed for Android so it uses the framework instead.
10. Extend to framework to cover other possible platforms.

1.3 Related work

Another thesis work performed Dohi Sweden by Johan Norberg relates to this work in the area of smart TVs and overlaying pieces of information onto the screen of a running TV show.

1.4 Overview

This section will provide an overview of the report.

- Chapter 2 Goes into detail about the methodology used during the practical and theoretical work of this project.
- Chapter 3 This chapter details a study on smart TV devices, explaining the term, which manufacturers exist, what the market looks like and how development for these devices works.
- Chapter 4 A study on available face detection libraries for the Android platform, where each library is described and tested to see how well they work on a Google TV.
- Chapter 5 This chapter studies algorithms for illumination normalization, which are employed to preprocess images before they are being used in a face detection algorithm. The purpose of this preprocessing is to eliminate lighting conditions and make facial features more pronounced in order to increase detection rates.
- Chapter 6 Describes the results of this project, with a detailed explanation of the workflow of the system and its functionalities. The chapter describes the functioning components which have been produced.
- Chapter 7 A discussion about the project, going into detail about why certain things could not be achieved and what could be done in the future. The non-functional components of the system will be discussed in this chapter and how these could be made to work.

Chapter 2

Method

This chapter discusses the methodology used for this project. Section 2.1 will give an explanation of the method used for development process as well as the tools used for tracking results. In section 2.2 the scope and basis for the in-depth study will presented. Section 2.3 will describe how the project's progress and results will be documented.

2.1 Development Method

For all internal projects at Dohi Sweden the Scrum development method is used. Scrum is a type of agile software development method that works in an iterative fashion where each iteration is called a sprint.

There are three roles in Scrum:

- **Scrum Master** - Ensures that the process is followed and that any problems that occurs are dealt with.
- **Product Owner** - The stakeholders and the business.
- **Development Team** - A team composed with all the different talents required to complete the project that should be self-organized.

It is interesting to note that the people on the development team are not necessarily focused on a single area but collaborates in getting the work done.

At the beginning of the process there is a product backlog which is a list of required features written down by the product owner. These are features are called stories and are written as a usage scenario with little technical details. Stories have an estimated business value attached to them. A meeting is held called storytime where the team has to evaluate the stories in the backlog and break these down into smaller stories and give an estimation how long they will take to complete.

A sprint in scrum is a time period that can range between a week and a month. The typical length of a sprint at Dohi Sweden is one or two weeks. At the beginning of a sprint there is a planning session where stories are broken down into specific tasks that need to be performed and put into a sprint backlog. These tasks are estimated for how long they will take to complete in hours. A task should not take too long to complete, in this case the task is further broken down into smaller tasks. Tasks should not be too small either. At Dohi Sweden the preferred time estimates for a task is two or four hours, and a maximum of eight hours. When a sprint is planned, the velocity¹ of the previous sprint should be taken into consideration as a guideline for the velocity to aim for the next sprint. Sprints have a particular goal that should be met, and at the end of a sprint it should be possible to conclude whether or not that goal has been met.

Every day there is a meeting called daily scrum where each member of the team gets to answer three question:

- What was done yesterday?
- What will you do today?
- Have you encountered any problems?

This meeting is kept very short, and should not take longer than 15 minutes. The purpose of this meeting is for the team members to state which tasks they're going to pursue from the sprint backlog and for the scrum master to find out about problems a team member may have encountered that needs to be resolved.

By having the members of the team assign tasks for themselves provides a benefit of flexibility where different roles can be interchanged or even letting another person get hired for completing the task if it proves to be too challenging.

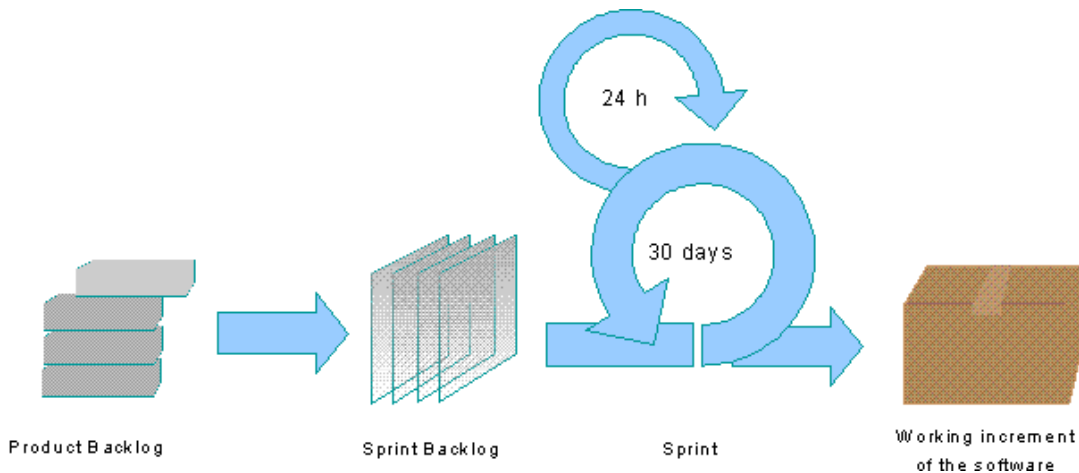


Figure 2.1: The scrum process, where sprints are iterated.

¹Velocity refers to the time, or points (measurement based on time), that the sprint took to finish.

An increment is the sum of all completed backlog items for a sprint and all prior sprints. It represents a working state for the product. These are made at the end of sprint alongside a review meeting where all the work is reviewed to find out what tasks have been completed and which have not been completed. The completed tasks are also demonstrated.

Furthermore, a retrospective meeting is held where each team member gets to reflect on the previous sprint and are asked what went well and what could be improved for the next sprint.

2.1.1 Project Management Tools

Dohi Sweden uses Pivotal Tracker² for tracking projects which is a tool for managing agile development projects. This tool provides the team with a backlog where tasks can be filled up and each team member can assign such a task for themselves so everyone can see who's working on what. These tasks can then be marked as complete when they're deemed complete, rejected after completion if something is missing and there is also support to add comments for these tasks. These comments can be helpful if a team member needs clarification on a task or if a problem has risen that the whole team needs to be made aware of.

Each task is associated with a number of points and is used by the tracker to display the velocity of the team and also used to produce statistics and generate burn-down charts.

This tool also provides time tracking features to keep track of working shifts and time spent on completing tasks.

2.2 In-depth Study

The in-depth study presented in chapter 5 focuses on illumination normalization, a pre-processing effort done typically on a grayscale image before performing face detection on it. Illumination normalization is used to eliminate the effects caused by varying lighting conditions in a scene so that the facial features of people present in an image becomes easier to distinguish for the face detection algorithm. The study will look at different algorithms to achieve this as well which ones may be appropriate to implement given limited processing power.

2.3 Documenting and Tracking Results

During the project the progress made each day will be written down so that it can be brought up on the daily scrum as well as providing a means to reflect on what has been done and if the project's goals can be met in time.

²<https://www.pivotaltracker.com/>

Pivotal Tracker will be used as a means for tracking what needs to be done, filing bug reports, receiving feed back as well as other features to move the project forward. Given that all features can be properly assigned a time estimate this tool can also tell whether the project is on time or not.

The implementation of the framework shall be documented as well. A diagram providing an overview of the system will be provided along with descriptions for each component. The public API of the framework will also be given special attention in the process of documenting.

Chapter 3

Smart TV

This chapter describes the study done on the subject of smart TVs. Section 3.1 gives an introduction on the area describing the terms in use and goes into detail what has been studied and the purpose of the study. In section 3.2 an overview of smart TVs is given. The following sections will go into detail about each manufacturer and the chapter concludes with section 3.7 where the different devices will be evaluated and taken into consideration for the implementation of the framework.

3.1 Introduction

Smart TV is a loosely defined term that denote a TV set or a set-top-box with functionality such as Web TV, streamed on-demand media, social media (e.g. Twitter and Facebook) as well as having applications which can be run on the unit. Like the name seems to indicate, the idea is to follow smartphones in their tracks and provide services that connects to the internet and having a larger focus on social media applications.

This study will look at what possibilities exist for development on existing smart TV platforms, look at what kind of services are already provided, the hardware in use as well as an analysis of what could be done in the future. Some of the things that will be looked at includes the technology (operating system, hardware), the range of applications available, potential market and more.

The rest of this chapter will continue with a section that will describe general characteristics of a smart TV unit as well as looking at some of the terms in use such as “social TV” and “interactive TV”. Following that there will be sections describing various providers take on smart TV implementations.

3.2 General

There is a wide range of units on the market that could be considered to be smart TV units. Logitech and Sony have units that are based on Google's Android platform and Apple has a unit called Apple TV which utilizes their iOS platform. Providers such as Samsung, LG and Philips have their own smart TV platforms which base their content on Flash, HTML and JavaScript. Roku is another provider who has launched a set-top-box with a large range of channels provided. Gaming consoles such as Xbox 360 and PlayStation 3 may also be considered to be smart TV units.

Common for all these units that are considered to be smart TV is that they provide applications for viewing streaming media and typically also have social media applications such as Twitter and Facebook. A smart TV either exists as a complete TV set or as a set-top-box, which has to be connected to a display commonly with a HDMI cable in order to be used.

Other names that have been used to describe smart TV are "connected TV" and "hybrid TV". This report will use the term "smart TV" to describe such units. The following terms can often be confused with smart TV and will be explained.

- **IPTV** - A system where television contents is provided over the internet protocol.
- **Internet TV** - Digital content delivery over the internet. It is different from IPTV as IPTV is something that is undergoing a standardization process.
- **Web TV** - A genre of digital entertainment, typically smaller episodic shows that are distributed using services like YouTube.

Despite difficulties for some manufacturers to get people interested in buying a smart TV[30] the market is growing, and by 2014 it's expected that more than 123 million units have been sold worldwide[3].

3.2.1 Social TV

A social TV experience implies that the viewer has access to technology that provides communication and social interaction while viewing a TV program or accessing some form of TV content. Much of the focus has been on having access to social media, but social TV also encompasses every aspect of communication. For example having a video call while watching a show would also be a social TV feature.

3.2.2 Interactive TV

The term interactive TV is used to describe techniques that allow viewers to interact with television content. In its simplest form, changing volumes and channels would be considered interactive TV. However, a truly interactive experience is gotten when the user is able to return information to the broadcaster. The return path can be provided by different means,

such as by telephone, SMS or internet. An example implementation of such interactivity is voting for different programs like the Eurovision Song Contest.

3.3 Google TV

For the Google TV platform there are a couple of different units available. There is a set-top-box developed with Logitech dubbed Logitech Revue as well as TV-sets and set-top-boxes developed in partnership with Sony. Logitech Revue uses an Intel Atom CE415 processor that has been clocked to 1.2 Ghz[20], 5 GB of RAM and has two USB 2.0 ports. Sony has two different set-top-boxes, one standard and one with a BluRay player built in along with TV sets that come in different sizes. These use an Intel Atom CE4170 processor that has been clocked to 1.66 Ghz[4], 8 GB of RAM and four USB 2.0 ports.

Common for all of these units is that they use the Android platform running on the Honeycomb version (having major version number 3). They also have ports for HDMI input and output, A/V receivers and cable/satellite integration. The USB ports can be used to plug in mice, keyboards, cameras and mass storage units. It's also possible to plug in special webcams to use in Skype calls. Google TV is not available outside of North America but is expected to be available in Europe by 2012[33].

Logitech has however resigned their partnership with Google and will not produce more units. Google is also planning on abandoning the Intel processors and switch to ARM[24] in the future.

Because Google TV uses Android as the operating system, most of the applications that have been built for Android-based smartphones can be run on a Google TV and an increasing number of these applications have found their way to the Google Market for Google TV. An application that has been written to make use of the NDK can however not be published which will unfortunately block a lot of interesting applications from finding their way to the TV.

Larger media companies like Hulu, CBS and ABC are blocking Google from using their services at the moment, other companies like Netflix have on the other hand chosen to partner with Google.

For developers there is, apart from the functionality that is already available from previous Android version, functionality for communication with USB devices that was introduced in Honeycomb as well functionality specific for the Google TV that allows for channel switching, listing of available channels and a communication protocol or using other devices as remotes for the TV. There are quite a few restrictions with how a developer can interact with the TV, it is for example not possible to read the contents from a TV channel and it's not possible to use picture-in-picture to display other contents[9].

3.4 Samsung TV

Samsung has their own TV-sets running on a Linux-based operating system. Their devices have 3 ports for USB 2.0, 4 ports for HDMI input and 2 ports for A/V-receivers as well as cable/satellite integration. The USB ports can be used to connect mass storage units, game controllers, microphones and health care devices. It's also possible to connect special webcams designed for Skype calls, and it might also be possible to connect webcams that are not officially supported[1].

The number of available applications for Samsung smart TVs is very large, with over 900 applications available and over 10 million downloads[35]. Most of the providers of streaming content have applications that can be downloaded and use to access their services. Samsung has also taken a step towards social TV with a special button on their remote control which can be used to access social media like Google Talk, Twitter and Facebook while watching TV.

Development for the unit can be done either with JavaScript and HTML or as a Flash application. Samsung has also made an SDK available for download for Windows. Apart from functionalities for creating a user interface and working with the file system you can use the SDK to communicate with the USB devices and activate/deactivate the 3D mode. The 3D mode is limited to showing 3D media, so it cannot be used to render stereoscopic images programmatically. WebGL is not supported on the devices so a developer has to use Flash to create applications with more advanced graphical capabilities. Apart from this there is also a communication protocol that other devices can use to act as a remote to the TV.

Samsung and other provides with similar units such as LG have also begun a partnership with Google and are working on delivering a device using Android as the operating system[26].

3.5 Apple TV

An Apple TV is a set-top-box dedicated to streaming media. It does not have any A/V receiver or cable/satellite integration. The first generation of Apple TV ran on a modified version of Mac OS X Tiger while the second generation is running on iOS. This second generation device has an Apple A4-processor, 256 MB of RAM and an HDMI output port.

There is no App Store available so the user is limited to using the services already provided with the box. A user can connect to iTunes, rent movies and watch clips on YouTube, Vimeo and Netflix among others. There is no real connection with social media so there are no applications for Twitter and Facebook. There is however a web browser.

Apple TV provides few possibilities for development since it is not possible to develop applications for the unit. It is however possible to connect to the device using AirPlay and using the unit as though you had connected an external display and sound system. Using this a developer can stream contents from an application running on another device such as an iPhone or an iPad and have it displayed on the Apple TV.

The device is very simple and does not provide all the functions that could be expected of a smart TV. There are however speculations about Apple having plans on releasing a real smart TV device that goes all the way with releasing an App Store[18].

3.6 Yahoo! Connected TV

Yahoo! has not built any devices of their own but have an engine for application development called Yahoo! Widgets, which previously had the name Konfabulator before being bought. This engine runs under Windows, Mac OS X and Linux. There is also a complete platform built around this engine called Yahoo! Connected TV which runs in a Linux environment with the Ubuntu distribution. This platform is in use by a couple of the larger developers such as Samsung, VIZIO, Toshiba, Sony and Hisense[15] and is available in larger parts of Europe[2]. Over 8 million units using the Connected TV platform have been sold worldwide[5].

Yahoo! Widgets provide, like the name implies, widgets, which is applications that are typically run in smaller windows on top of the screen. A widget can be developed with a few different size formats to choose from[7], where fullscreen mode is one of them.

Development is done in an HTML and JavaScript environment. Apart from the ability to do some standard system calls in JavaScript there are also binding to SQLite provided so a developer can implement a database. Another function provided in the SDK is the ability to receive commands from a remote unit over the network which can be used to implement a remote control for the TV.

3.7 Discussion

Many units provides some sort of base functionality where applications for social media and a couple of applications for streaming media is provided as well as having an SDK available for developers and a site where users can download and upload applications. Only Samsung appears to have made any attempt at bringing a social TV experience and very little has been done to use the capabilities of a smart TV to provide for an interactive TV experience. The processing power in these devices also appear to be sufficient to provide for simpler 3D rendering in real-time.

Yahoo! and Google appears to have a jump start in this business and it will be interesting to follow Google's partnership with Samsung. With any luck it might be possible to create stereoscopic OpenGL applications. Another thing that would be interesting is to experiment with the usage of webcams in an application for smart TVs, but as it stands right now few systems support common webcams and are very closed off from modification so it is not possible to write custom drivers for these cameras either. Samsung is on the other hand coming up with a new model that comes with a built in camera[13].

Another conclusion is that it is very difficult to developer applications that can appear on top of other. For example if one would like to have information that is displayed on top of the

screen, or at the side of the screen as one might want it when integrating social media on the device. TV applications appear to follow the same design principles as smartphones where only one application is being run on top of everything else and only houses one window. This makes working towards higher degrees of interactivity and social TV viewing much more difficult since a developer would have to make everything from scratch, importing media streams by themselves instead of using already existing services and then built content on top of that. Samsung has on the other hand provided the possibilities to specify different size formats for an application, allowing for apps to take the size of a ticker or a side pane but have chosen to only allow fullscreen applications for use in Europe[10].

Hopefully TV applications in the future will be more like desktop applications where an application can have multiple windows in custom sizes as well as having an open interface for retrieving contents that is displayed on TV channels and perhaps even internet-based services like YouTube and Netflix.

Chapter 4

Face Detection

This chapter will detail the investigation of available face detection libraries for the Android platform and decide which would be appropriate include in this project. These libraries have also been tested on a Google TV device running Honeycomb. The next sections will describe how face detection is achieved using the following libraries:

- The two different interfaces in the Android API.
- OpenCV
- JJIL
- jviolajones
- OpenIMAJ

This is then followed by a discussion in section 4.6 where the results of trying to get these libraries to work and perform face detection on a Google TV is detailed.

4.1 Android Libraries

The android platform provides two different facilities for face detection. One is an older interface that's been available since API version 1 as well as a more recent interface introduced in API version 14 (version 4.0).

4.1.1 `android.graphics.FaceDetector`

This interface has been available since the first version of the API and provides basic functionality for face detection. It works by taking an object of type `Bitmap` as input and

produces a list as output where each element of that list represents a uniquely identified face.

Each such identified face can be queried for information about the midpoint and distance of the eyes, the pose of the face as well as a confidence value expressed as a percentage which indicates how certain the algorithm is that it's actually a real face. The FaceDetector is restrained so that any face found must have a 40% confidence or else it's discarded[8].

This alternative does not appear to provide enough information to figure out where the eyes are actually located, furthermore there is no information about the size of the face.

4.1.2 android.hardware.Camera.FaceDetectionListener

In Android 4.0 a new interface for face detection was introduced which allows a programmer to create a listener class for face detection and attach it to an instance of the Camera class. Once attached, the listener will receive a list of objects represented uniquely identified faces through the user-defined callback function during each update of the camera during preview.

In order to receive face detection information it is also necessary to provide the camera with either a SurfaceHolder to draw preview contents on, or a SurfaceTexture used in OpenGL drawing[6].

The information about each face received in the callback includes coordinates for both eyes and mouth, the rectangular frame for the whole face as well as a unique and a confidence value. This confidence value is a value between 1 and 100 with 100 being the highest confidence.

4.2 OpenCV

OpenCV (Open Computer Vision) is a library written in C and C++ and was originally developed by Intel. This library provides many different features in the area of computer vision, such as face detection, motion tracking and gesture recognition. It also features many features many helpful features like camera handling and matrix math. This library has also been ported to the Android platform, using JNI.

For face detection there is an implementation of the Viola-Jones algorithm[32] that takes a training file as input to detect certain features. The OpenCV package also comes with training files that have already been produced to identify various features such as faces and eyes in photographs.

4.3 JJIL

Jon's Java Imaging Library is an open source imaging library that is written purely in Java. This library is targeted towards mobile applications with support for Android. The author

claims that the code for face detection works well for any camera phone[12], so it might have promising results for webcams as well.

4.4 jviolajones

This open source library is written purely in Java that implements the Viola-Jones algorithm and takes input files of the same format as OpenCV, and thus you could use the training files that come with the OpenCV package. In contrast to other libraries, this library is very lightweight since it is only meant to implement the Viola-Jones algorithm with no additional image processing functionalities.

4.5 OpenIMAJ

OpenIMAJ stands for “The Open Intelligent Multimedia Analysis toolkit for Java” and is an open source library which is a bit similar to OpenCV but also deals with audio and video processing, written completely in Java. This library uses the Viola-Jones algorithm like many others to perform face detection and comes with several built-in Haar cascades that are trained to detect faces and eyes among others.

4.6 Discussion

The early version face detector in the Android API does not provide the information about a face that suits the needs of this project and so will not be considered. While the newer interface for Android 4.0 provides all the necessary functionality it won't be possible to use it since the Google TV is running on the Android 3.0 (Honeycomb) versions. Another problem is that the Google TV does not have any built in camera and this newer interface only works with live camera previews and cannot be input with custom images.

OpenCV was the most interesting library due to its portability and its community support. Unfortunately it cannot be used with the current Google TVs, as these devices do not support native libraries. Even after finding workarounds and some tinkering with the build system it's still not possible to use OpenCV as it depends on dynamic libraries that does not exist on the device and cannot be provided by the application either.

JJIL works well with the Google TV, able to detect faces in given test images. It however showed some flaws, being unable to recognize faces in images fetched from a webcam. It is unclear why JJIL could not detect faces from the webcam images, but one reason could be the noisy environment and poor lighting conditions during testing which motivates the study in chapter 5.

jviolajones was not built with Android in mind and turned out to be difficult to integrate due to the fact that the library took filenames as arguments instead of already opened input

streams. This library could be modified to make it work but it seems better to focus on the other libraries that already work instead.

OpenIMAJ was written for Java and is claimed to work on Android because of these efforts[14]. During testing however the code for face detection would not compile. In particular, the piece of code that reads XML training data used the `javax.xml.stream` interface which is not present on the Android platform.

This investigation concludes that JJIL would be the safest to use, although it is not portable so efforts will be made to get OpenCV working.

Chapter 5

Illumination Normalization

This chapter details an in-depth study done in the field of illumination normalization. Efforts in getting good results for face recognition is often hindered by troublesome lighting conditions as well as variations in pose. Illumination normalization techniques aim to pre-process an image before it's passed on to the face recognition algorithm. The aim of this study is to find suitable algorithms that could help improve the detection rates in face detection on a smart TV.

5.1 Overview

Getting good detection rates while performing face recognition is proven difficult by a couple of challenges such as variations in illumination, pose and facial expressions. As is the case with this implementation a user is expected to be sitting in front of a television looking right at it with the camera either being integrated into the TV set or mounted on top of it. For this reason the pose is assumed to be neutral with the user directly facing the capturing device. The issues that are of primary concern for this implementation is the quality of the captured image, the resolution of the captured image (since a person sitting far from camera will show up tiny on a captured image) and lastly, varying lighting conditions. Another thing to note is that changes in illumination is one of the main problems in face recognition, and it has also been proven both experimentally and theoretically that changes in illumination cause greater differences in the image than the differences between individuals[22][37][23].

For these reasons, different techniques for compensating the effects of illumination will be studied. Since this project does aim to implement any face recognition features from scratch but rather use existing libraries the kind of techniques that will be looked at involve preprocessing the grayscale (luminance) images that are provided to the face detection routines.

There are numerous methods available in the area of face recognition, such as Eigenfaces, Probabilistic and Bayesian Matching, Active Shape Model, Local Feature Analysis and Elastic Bunch Graph Matching to name a few. While there are a lot of proposed algorithms

for face recognition that solve the problem in different ways, most of them are susceptible to the changes to differences in lighting[37].

Illumination normalization[37][28] is the process in which an image is transformed by some means so that it is considered to be lighting invariant. That is to say, the produced image can be said to be unaffected by changes in lighting conditions that were present in the original image.

In the area of illumination normalization there is also a lot of algorithms proposed, such as Illumination Cone[16], Quotient Image-based techniques[36][21][37][29], Anti-Face Method[31], Logarithmic Transform[25], Total Variation[38][37], Census Transform[23] among others.

This study will focus on the algorithms that do not require prior training and can preprocess an arbitrary image without the need for a learning database.

5.2 Quotient Image

The quotient image method for face recognition was developed as an alternative to class-based models where images are reconstructed using basis images for a particular class to determine whether that image belongs to the class or not. First the definition of a class of objects is given. For the purpose of having a precise definition that can serve as a suitable basis for analytical methods the term ideal class is used which refers to a set of 3D objects that have the same shape but can differ in the albedo¹ function for the surface. An object (in such a class) in turn refers to a set of images that represents the item in the picture under varying lighting conditions. In this case, as well as for the quotient image, the class being considered is human faces[36].

Quotient images represent illumination invariant signature images that could be thought of as neutral images, having canonical lighting conditions or viewing positions. With this process a novel image² is transformed into a neutral image and then matched against a data base of neutral images.

This method uses a 3D Lambertian reflectance model³ where cast shadows are not accounted for. Which means that the image space for the class can be described as:

$$p_i(x, y)n(x, y)^T s_j$$

where $p_i(x, y)$ is the albedo of object i that belongs to the class, $n(x, y)$ is the surface normal (which in the case of quotient images is pointing outwards from the image) and s_j is the direction of the point light source that can vary.

The definition of the quotient image Q_y of object y in relation to object a is defined as follows:

$$Q_y(u, v) = \frac{p_y(u, v)}{p_a(u, v)}$$

¹Albedo is the texture of the surface, or simply the color that's going to be reflected as light hits.

²An arbitrary image belonging to the class under consideration.

³This model is often used as the model for diffuse reflection in computer graphics, where the reflection is calculated by taking the dot product between the surface normal and the light direction.

, where u and v are the coordinates that range over the image[36]. The albedo functions for the two objects are unknown but they can be recovered analytically with the following proposition.

PROPOSITION 5.2.1: Given three images a_1, a_2, a_3 of object a where the lighting conditions in each image are linearly independent and an image y_s of object y that is illuminated by the light source s , then there exists coefficients x_1, x_2, x_3 that satisfy

$$y_s = \left(\sum_j x_j a_j \right) \oplus Q_y$$

where \oplus denotes pixel-by-pixel multiplication of images. Another thing to note is that the image space of object y is spanned by varying the coefficients[36].

This set of images of object a is referred to as a bootstrap set.

When the coefficients x have been produced the quotient image can be produced from the average image \mathcal{A} with the following formula:

$$Q_y = \frac{y_s}{\mathcal{A}x}$$

It shall be noted that because the quotient image algorithm requires an initial set of bootstrap images to produce a signature image and will therefore not be discussed further. The following sections will discuss derivatives of this algorithm that do not depend on training images which motivates having a basic understanding of the quotient image. Another interesting thing about this algorithm is that it shows that an illumination invariant image can be produced from a very small set of training images as opposed to alternatives that require far greater samples to produce good results[36].

5.2.1 Self-Quotient Image

The self-quotient image (SQI) is based on the Retinex method[19] and has advantages over the quotient image in that only one face image is necessary, it works in shadow regions and no prior alignment of an image is required[21].

It is pointed out by Haitao Wang and colleagues that the conditions assumed by the quotient image approach cannot be satisfied at the same time. These conditions are as follows[21].

- Images are illuminated by a single point light source with no cast shadows.
- All faces under consideration have the same shape (surface normal).
- Face images contain no shadows.
- The alignment between different faces in a set is known.
- A bootstrap set for estimating lighting direction is available.

The self-quotient image is defined as the ratio of the input image and its smooth versions. These smooth versions are derived from the Retinex algorithm.

Retinex algorithms are based on a reflectance-illumination model instead of the Lambertian model used in the quotient image approach. The reflectance-illumination model can be described as follows:

$$I = RL$$

where I is the image, R is the reflectance of the scene and L represents the lighting. This lighting component can be considered as the low frequency component of the image I and can be estimated with a low-pass filter:

$$L \approx F * I$$

where F is a Gaussian filter and $*$ is the convolution operation[21].

This leads to the ability to express R as

$$R = \frac{I}{L} = \frac{I}{F * I}.$$

The definition of the self-quotient image is done in similar terms.

Definition 5.2.1. The self-quotient image Q of the image I is defined by

$$Q = \frac{I}{\hat{I}} = \frac{I}{F * I}$$

where \hat{I} represents the smoothed version of I , F is the smoothing kernel and the division performed in the formula is done pixel-by-pixel.

The performance of the algorithm in various regions of illumination of an image is studied in three different cases[21].

EXAMPLE 5.2.1: For regions with small variations of the surface normal, the reflectance can be approximated with a constant C_1 as

$$n^T(u, v)s \approx C_1$$

This give the equation for the SQI as

$$Q = \frac{I(u, v)}{\hat{I}(u, v)} \approx \frac{p(u, v)}{(p(u, v) * F)C_1} = \frac{p(u, v)}{p(u, v) * F}$$

In this case Q is approximately free from illumination and calculated similarly to the quotient image but only depends on the input image.

EXAMPLE 5.2.2: In regions without shadow but large with large variations in the surface normal the equation for the SQI is as follows.

$$Q = \frac{I(u, v)}{\hat{I}(u, v)} = \frac{p(u, v)n^T(u, v)s}{F * (p(u, v)n^T(u, v)s)}$$

Such regions depend on the surface normal and the lighting and can there not be considered free from illumination.

EXAMPLE 5.2.3: In a shadowed region the values for luminance are low and vary less. Light in these regions are assumed to be uniformly distributed from all directions meaning that any surface normal in shadow forms a semi-hemisphere and therefore the dot product between the normal and the source of light is constant in the region. The same equation as in the first case can be used, but with the constant C_2 instead of C_1 , being defined as follows.

$$n(u, v)^T \sum_{i=1}^{\infty} s(u, v)_i = \sum_{i=1}^{\infty} n(u, v)^T s(u, v)_i = C_2$$

Just like in the first case, this case is also considered illumination free.

The filtering used for SQI is a weighted Gaussian filter which provides anisotropic smoothing. This filter has a normalization factor N that satisfies

$$\frac{1}{N} \sum_{\Omega} WG = 1$$

where G is the Gaussian kernel, W the weight and Ω the size of the convolution kernel. The convolution region is divided into two sub-regions M_1 and M_2 . If there is a large variation in luminance in the convolution region, i.e. an edge region, the filter will only convolute with the larger part M_1 , which contains the most pixels[21].

This has the effect of reducing halo effects because the weighted Gaussian kernel will only smooth one side of an edge region.

A final point about the SQI algorithm is that the division operation can magnify high frequency noise, especially in shadowed regions where there is a low signal to noise ratio. To compensate for this a non-linear transform is applied to Q . For this task a logarithmic transform can be performed.

5.2.2 Total Variation Quotient Image

The total variation quotient image (TVQI) uses the TV+L¹ model which minimizes the fidelity of a cartoon image using L¹-norm fidelity term. This method compares itself to the quotient image and the self-quotient image, boasting higher detection rates[37].

In the total variation framework the image f is modelled as the sum of an image cartoon u and a texture v with these variables being defined as functions. The cartoon consists of background and boundaries such as sharp edges whereas the texture contains small-scale patterns.

The TV+L¹ model is formulated as follows.

$$\min_u \int_{\Omega} |\nabla u(x)| + \lambda |f(x) - u(x)| dx$$

where $\int_{\Omega} |\nabla u(x)|$ is the total variation of u over its support Ω and λ is a scalar weight parameter.

While the reflectance of an image is helpful to take into consideration in the process of identification, it is not as important in face recognition where geometric features such as the eyes, nose and various shapes is more critical to distinguish different people[37].

The TVQI shares improvements with SQI, not requiring any training images and requires no alignment between images. In addition, the TVQI does not make any assumptions about the light sources in a scene.

By making the observation that the intensity and variance of small-scale signals in an image are proportional to the intensity of the background in their vicinities while acknowledging that the small-scale signals in dark areas must be amplified the normalized image is approximated as $v'_{x,y} = \frac{f_{x,y}}{u_{x,y}}$. In this equation each point (x, y) is sampled used the output u from the TV+L¹. The TVQI model is thus proposed in the following way[37]:

$$u = \arg \min_u \int_{\Omega} |\nabla u(x)| + \lambda |f(x) - u(x)| dx$$

$$TVQI = v' = \frac{f}{u}$$

In the TVQI model the choice of lambda is inversely proportional to the signal scale.

The division operator used in this model can introduce small variations and noise into the dark regions of an image. A solution to this problem is to preprocess the image using the TV+L² model[27] with a larger value for lambda before applying TVQI. The algorithm with this added preprocessing step is called +TVQI.

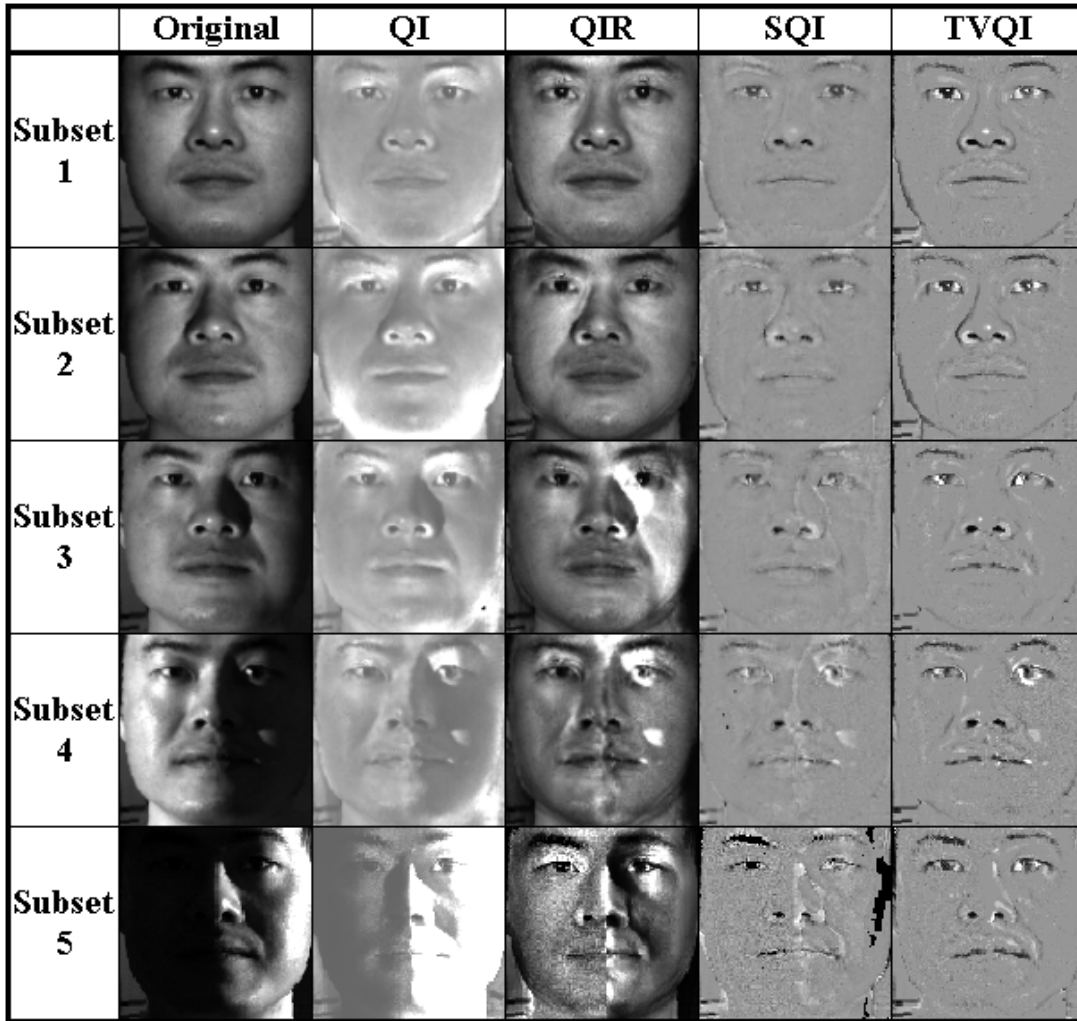


Figure 5.1: Comparison of QI algorithms on a set of images from the Yale B database[37].

5.3 Census Transform

The census transform[34] is a method for computing visual correspondence based on local ordering of intensities. This section will study a particular model proposed by Ji Hoon Kim and colleagues[23] where local features are generated using kernels of size 3x3 pixels applied on an input image.

Each element of these kernels are represented by the values 1 or 0, and are obtained by comparing the center pixel with its 8 neighboring pixels. This transform is defined in the following way.

$$C(p) = \oplus_{q \in N^{\circ}(p)} (I(p), I(q))$$

where $N(p)$ is the set of neighboring pixels of p and $p \notin N$, $I(p)$ the intensity at pixel p and the comparison operator ς is defined as:

$$\varsigma(x, y) = \begin{cases} 1 & \text{if } x < y \\ 0 & \text{if } x \geq y \end{cases}$$

The concatenation operator \oplus generates a bit sequence that represents the results of the comparisons[23].

Another transform that can be used is the modified census transform[17] which can have 511 variations of kernels instead of 256 by taking the average value of the nine pixels as a reference value. This transform is defined as:

$$\Gamma(p) = \oplus_{q \in N'} \varsigma(\bar{I}(p), I(q))$$

where $\bar{I}(p)$ is the average value.

The values of the kernel can be concatenated in $9! = 362880$ different ways. A test revealed that arranging the values backwards (right to left, downwards and up in the kernel matrix) gave the best results in performance.

After these values have been arranged sequentially they can be interpreted as a binary number and used to construct a normalized image.

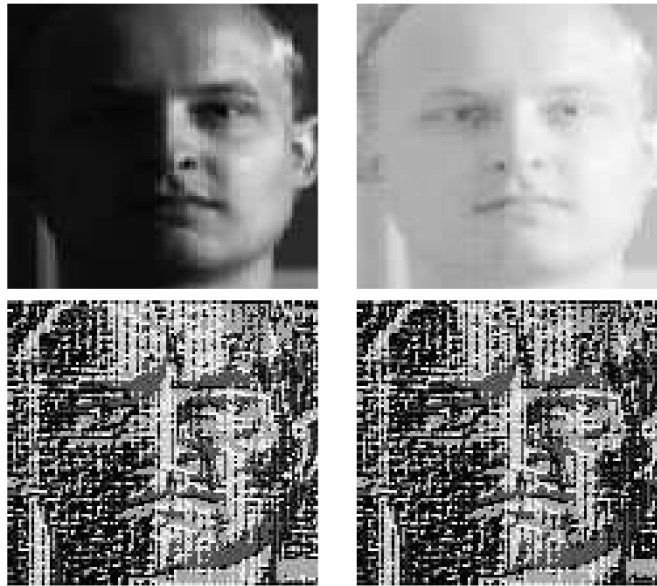


Figure 5.2: Shows the original images at the top and the results of the Census Transform below[17].

5.4 Fused Logarithmic Transform

One of the most commonly used methods in image enhancement is the logarithmic transform which is consistent with the logarithmic sensitivity of the human visual system for light perception[25]. While logarithmic transformation of images provides an increased contrast at lower ranges it also has the problem of compressing the contrast at higher ranges. The fused logarithmic transform proposed by Hung-Son Le[25] enhances image contrast at both low and high ranges. This method will work with arbitrarily unknown lighting conditions, which is to say that no assumptions are made on the light sources in a given input image, nor the absence of shadows.

Image fusion is a process that can be described as obtaining a set of two or more images of a given scene that can be assumed to be aligned and then combining these images into a composite image that will appear natural to the human eye. This method focuses on contrast fusion since the type of images used in face recognition are in the visible range. The aim of contrast fusion is to combine images with different contrast to get good overall contrast

The main idea behind the fused logarithmic transform (fLog) is that an image transformed by logarithmic transform can be improved by performing a contrast fusion with the original image, which does not suffer from the problems at higher ranges while preserving the improvements of the logarithmic transform at lower ranges.

Given an image $A(i, j)$ representing intensity in the dynamic range $[m : M]$ and where i and j serve as pixel coordinates the fLog transform can be described as follows.

First the image is quantized to the range $[0 : 255]$ with the following formula.

$$Q(i, j) = \lfloor 255 \times \mathbb{R}(A(i, j)) \rfloor$$

where the function $\mathbb{R} : [m : M] \rightarrow [0 : 1]$ is a linear mapping function. The logarithmic image can then be computed as:

$$B(i, j) = \lfloor \mathbb{R} \left(\left[\frac{\log(1 + p \times Q(i, j))}{\log(1 + p \times M)} \right]^{1/q} \right) \times 255 \rfloor$$

where $p \in [0, \infty)$ and $q \in [1 : 3]$.

The original image A is then fused with its logarithmic transform B into the composite image C using a multi-resolution spline fusion technique. In order to perform this fusion a binary mask M is also needed which can be produced through combining gamma corrected version of A and B according to the following formula.

$$M = \mathbb{R}(A^{\gamma_1} + B^{\gamma_2})$$

where γ_1 and γ_2 are two constants that have been found to work well by setting $\gamma_1 = 1$ and $\gamma_2 \in [2 : 3]$ according to Hung-Son Le[25].

The fusion can be performed by building Laplacian pyramids L_A and L_B for the images A and B respectively as well as a Gaussian pyramid G_M for the mask M . These pyramids are

then used to form a combined pyramid L_S using the following formula for each decomposition level l .

$$L_{S_l}(i, j) = G_{M_l}(i, j) \times L_{A_l}(i, j) + (1 - G_{M_l}(i, j)) \times L_{B_l}(i, j)$$

Finally, the image C is obtained by expanding and summing all levels of L_S .

It is also possible that the synthesized image might look soft in areas where the image A had bright areas with low contrast. The fusion algorithm can in this case be reiterated to produce a further improved image by using $\bar{A} = (255 - A)$ and $\bar{C} = (255 - C)$ as the two images to be fused with $\mathbb{R}(\bar{C}^{\gamma_1} + \bar{A}^{\gamma_2})$.



Figure 5.3: Shows the original image to the left and the results of the fLog transform on the right[25].

5.5 Discussion

By simply looking at the improvements in the rate of detected faces[23][25][36] it can be concluded that just implementing any of these algorithms for preprocessing would suffice. The question is which algorithm is most cost efficient on hardware such as a Google TV with limited processing power compared to a desktop computer.

The census transform is particularly interesting because of its relatively simple scheme where luminance is more or less jumbled around to produce an image with more pronounced facial features, making relatively cheap calculation as it makes its pass over an image.

It is also worth looking further into the quotient image algorithms as they can be computed fairly fast as well but the difficult theoretical background necessary to proceed in understanding them and being able to properly implement them might prove too challenging for this project.

The fused logarithmic transform is interesting and not too difficult to work with. It might however become a bit too CPU intensive to construct the Laplacian pyramids as well as iterating over them.

From this study it appears that the census transform would be a good starting point to experiment with for improving detection rates in the application. The fused logarithmic transform could also be experimented with for this project.

Chapter 6

Results

This chapter describes the results of the project. Following this introduction, section 6.1 will go into detail about the structure and workflow of the system and section 6.1.1 goes into further depth describing the framework.

From the study on smart TV devices in chapter 3 it was made clear that none of the devices had any built-in cameras and supported a limited range of webcams. For this project I was given a Sony Google TV, Samsung Smart TV, Apple TV and PS3 Eye webcam to work with. None of these devices recognized the webcam so it was necessary to plug it into a desktop computer instead and write a simple server that would stream the data from this camera to the Google TV. While this is not a viable solution for a commercial application, the study found that in the future some of these devices will come with built-in cameras that hopefully can be utilized. Since this was just a prototype it would not be a problem.

It was not possible to write applications for the Apple TV so the rest of the attention was given to the Samsung Smart TV. From what could be seen both practically and in the study on smart TV devices most of them are capable of running applications through JavaScript to some extent. Therefore development efforts were completely focused on the Google TV where it was already possible to bind JavaScript objects to callbacks in the Java code of an application.

The project could not be completed on time however, because big problems were discovered with face detection. It was not possible to use OpenCV or most other widely supported libraries for face detection as discovered in the study of such libraries in chapter 4. JJIL worked on example images running on the Google TV, but would never detect faces on the images from the webcam. It was not possible in the time frame given to implement illumination normalization to test with JJIL.

However, assuming that face detection was properly implemented the system would nevertheless be fully functional as it has been tested using mock data for face detection results. This system is described in the following section.

6.1 System Description

When an ad banner containing an interactive advertisement is clicked on the control of the end-user application is given to the AdGateway front-end and there are three primary components of interest visualized in figure 6.1. The AdGateway front-end works by retrieving

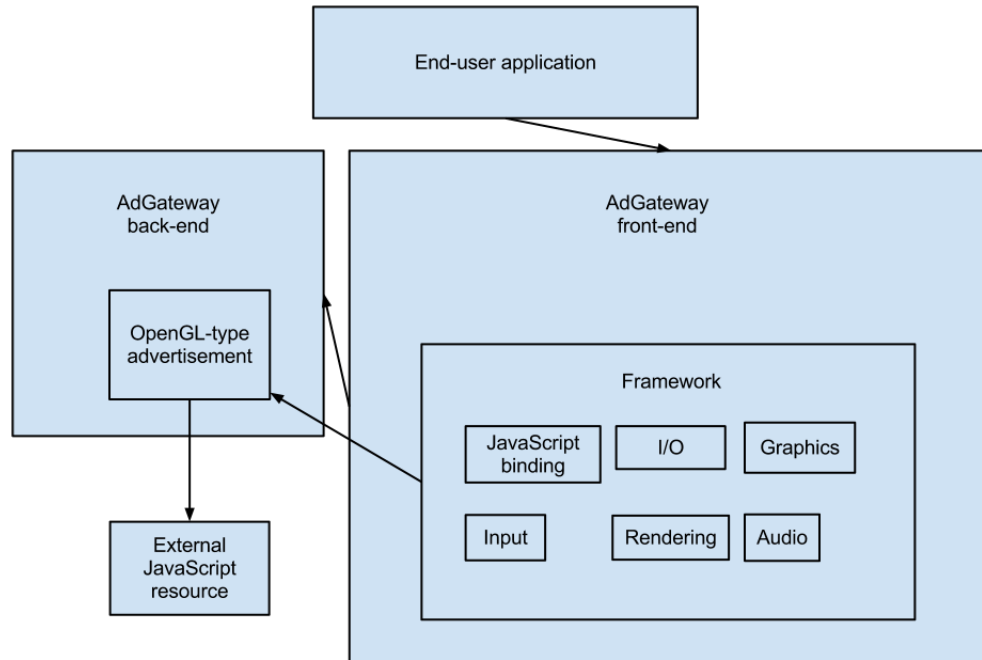


Figure 6.1: How the framework and related components interact with their execution environment. The arrows denote dependencies.

chunks of information representing different advertisements from the back-end. An addition to these types have been made with this implementation simply called “OpenGL” due to the fact that it creates an OpenGL rendering surface. Any user with a device that supports this type of advertisement (i.e. Android users) will be eligible to receive these kinds of advertisements.

More specifically the chunk of information sent from the back-end is XML-data and for the OpenGL type it has been extended to include a tag containing the URL for an external JavaScript resource.

On the front-end side, when an OpenGL-type advertisement is clicked on it will be handled by the framework which will set up an OpenGL environment for rendering and also retrieves the JavaScript provided by the information from the back-end. This script is then also executed and JavaScript bindings are made so both the framework and the JavaScript can communicate with each other through certain callback methods. In the following section

the framework is described in detail.

6.1.1 Framework

The framework is run like a typical render engine, having a main loop where each iteration draws something to the screen and time deltas are calculated for animations.

After the framework has set up its environment it calls `init()` on JavaScript object allowing the script to do some initialization such as preloading images used in rendering. At the start of each new frame the function `update(dt)` is called on the script where the renderer expects the script to do rendering, playing sounds, querying for information among other things through a certain object called `AdApp` which is made available to the script to communicate with the framework. The last function call the framework can make to the script is the `keyPressed(id, down)` which informs that the key with the given id was either pressed or released as indicated by the boolean variable `down`.

For the JavaScript, these are the available function calls on the `AdApp` object:

- **loadImage(url, name)** - Tells the framework to download the image from the given URL and bind it to the given name. This image will be converted to a texture in OpenGL for later use.
- **drawImage(name, transform)** - Issued in calls to `update(dt)`, will add a rendering request for the texture with the given name and modelview transformation. The special name “camera” can be given to render the latest snapshot from the camera if available. Due to limitations in the way objects are passed back via the JavaScript binding the elements constituting the transform need to be passed individually. In this case it means 3 float values representing x, y, and z for translation and scaling and 4 values representing angle, x, y and z for arbitrary axis rotation totalling in 10 float value parameters for the transform.
- **loadSound(url, name)** - Tells the framework to download the sound file from the given URL and bind it to the given name.
- **playSound(name)** - Tells the framework to play the sound file with the given name.
- **getScreenSize()** - Returns a JSON-stringified object that represents an object with the width and height of the screen in pixels. These values can be used to alter drawing on different platforms having different dimensions.
- **getDetectedFace()** - Returns a JSON-stringified object that represents an array of detected faces containing face rectangle and eye locations. If there are no detected faces this array will have a size of 0.

The idea is to have the script call the load functions during the call to `init()` and then later use them in the call to `update(dt)`. For an example of such a script refer to appendix A.

This framework is flexible and can be made to accommodate many different platforms due to all the work and data management being done by framework with the script only making requests.

Chapter 7

Discussion

This chapter will evaluate the project, looking at why it couldn't be completed in time, what went well as well as what can be done in the future. Section 7.1 will discuss the major restrictions prohibiting this implementation from achieving its goals. In section 7.2 some possibilities for new areas to explore related to this project will be touched upon.

During the startup phase, things were looking very good. I had no prior experience with the Android platform but was told it was very flexible and if some things wouldn't work out I could write my own drivers if needed. The initial study of smart TV platforms also did not dispel my beliefs that this could be done with the Google TV website saying cameras were supported with the USB connectivity. The fact that it would also get an application store and open up for development with Honeycomb was also promising.

The combination of no native camera and no possibility for running native code made the task of getting face detection, which was a major point in this project, almost impossible to achieve. It wasn't until very late that it was proposed that this project could be done just as well on an Android 4.0 phone using the new face detection interface. I had time to tinker around with it and if I was given more time to work on that there might've been a working implementation for these devices.

The framework on the other hand was relatively easy to get working, the JavaScript bindings in Android are very easy to use and having written a rendering engine which uses external JavaScripts for logic feels like a project of its own.

7.1 Restrictions

As has already been stated, the biggest restriction with this work is the lack of face detection. Another aspect is the limited functionality for overlaying programs on top of others and the ability to display the current TV show inside your own application. Since the idea was to have AdGateway banners provide advertisements during a currently playing show this is another goal that could not be achieved given the hardware I got to work with.

Since the Google TV platform does not allow native code, it was not possible to implement mutexes which made it difficult for the framework to synchronize with JavaScripts being executed. The reason synchronization is needed is in turn due to the fact that any functions called on a JavaScript object is asynchronous. This led to a workaround where scripts call `finished()` on the `AdApp` object during calls to `init()` and `update(dt)` to signal the renderer to continue processing. Depending on the user to give back control to the framework is not a safe solution and would not be acceptable in a finished product.

7.2 Future Work

In the study conducted on smart TV devices in chapter 3 it was discovered that Sony would release a TV with built-in camera and with such devices in the future it could be possible to get working face detection on the devices with which it would be possible to introduce augmented reality and interactivity to your applications which might be more in tune with the lean-back experience of TV viewing. It would also be interesting to experiment with a smart TV powered by Yahoo! Widgets to see if it might have better support for webcams and overlaying.

It is also worth considering other forms of interactivity, such as the possibility of using voice recognition. Voice recognition could for example be used to motivate a user to explore advertisements by providing a simpler and more laid back experience.

Chapter 8

Acknowledgements

I would like to thank the following people for their support throughout the course of this project.

- My class mates Johan Nilsson, Johan Norberg and Magnus Söderlund who performed their thesis work at the same time and were willing to discuss ideas and work together with me.
- Jerry Eriksson for his quick response time and availability, as well giving good feedback and advice.
- Olov Brändström for supervising me despite being very busy and not even available at the office most of the time.
- Linus Mähler Johnson who was helped with extra supervising when he was given the time.
- Emanuel Dohi for his encouraging attitude and creative ideas for taking different approaches.

References

- [1] Compatible webcam for samsung tvs - skype app. http://forums.cnet.com/7723-13973_102-395769/compatible-webcam-for-samsung-tvs-skype-app/, June 2010.
- [2] Yahoo! and samsung expanded availability of yahoo! connected tv to 26 new european countries. <http://www.yctvblog.com/blog/2010/11/02/yahoo-and-samsung-expanded-availability/>, November 2010.
- [3] Connected tvs forecast to exceed 123m units in 2014. http://www.displaysearch.com/cps/rde/xchg/displaysearch/hs.xsl/110425_connected_tvs_forecast_to_exceed_123m_units_in_2014.asp, April 2011.
- [4] Revue memory vs. sony memory - page 2. <http://forums.logitech.com/t5/Revue/Revue-Memory-vs-Sony-Memory/td-p/733218/page/2>, November 2011.
- [5] Yahoo! connected tv hits 8 million devices... and counting. <http://www.yctvblog.com/blog/2011/06/21/yahoo-connected-tv-hits-8-million-devices-%E2%80%A6-and-counting/>, July 2011.
- [6] Camera — android developers. <http://developer.android.com/reference/android/hardware/Camera.html>, May 2012.
- [7] Connectedtv yahoo! tv widgets - business opportunity. <http://developer.yahoo.com/connectedtv/business/index.html>, March 2012.
- [8] Facedetector.face — android developers. <http://developer.android.com/reference/android/media/FaceDetector.Face.html>, May 2012.
- [9] Frequently asked questions - google tv - google developers. <https://developers.google.com/tv/faq#pip>, March 2012.
- [10] Guide - - create with samsung d forum. http://www.samsungdforum.com/Guide/View/Developer_Documentation/Samsung_SmartTV_Developer_Documentation_2.5/Getting_Started/Samsung_Smart_TV/Smart_TV_Applications, March 2012.
- [11] Interactive tv ads are clicking with viewers - usatoday.com. http://www.usatoday.com/money/advertising/2008-07-06-interactive-tv_N.htm, May 2012.
- [12] jjil - jon's java imaging library, for mobile image processing - google project hosting. <http://code.google.com/p/jjil/>, May 2012.

- [13] Let the smart tv experience begin. — samsung smart tv. <http://www.samsung.com/us/2012-smart-tv/>, March 2012.
- [14] Openimaj / wiki / openimaj on android. <http://sourceforge.net/p/openimaj/wiki/OpenIMAJ%20on%20Android/>, May 2012.
- [15] Yahoo! connected tv: Movies, tv shows, internet on demand. <http://connectedtv.yahoo.com/developer/>, March 2012.
- [16] Peter N. Belhumeur Athinodoros S. Georghiades. From few to many: Illumination cone models for face recognition under variable lighting and pose. 2001.
- [17] Andreas Ernst Bernhard Fröba. Face detection with the modified census transform. 2004.
- [18] Tom Cheredar. Apple’s rumored itv: Analyst outlines three content scenarios. <http://venturebeat.com/2012/02/01/apple-itv-content/>, February 2012.
- [19] Glenn A. Woodell Daniel J. Jobson, Zia-ur Rahman. A multiscale retinex for bridging the gap between color images and the human observation of scenes. 1997.
- [20] Walter Galan. Logitech revue teardown. <http://forums.logitech.com/t5/Revue/Revue-Memory-vs-Sony-Memory/td-p/733218/page/2>, October 2010.
- [21] Yangsheng Wang Haitao Wang, Stan Z Li. Face recognition under varying lighting conditions using self quotient image. 2004.
- [22] Jun Miao Debin Zhao Gang Deng Jintao Li Hong Liu, Wen Gao. Illumination compensation and feedback of illumination feature in face detection. 2010.
- [23] Jong Geun Park Ji Hoon Kim and Chulhee Lee. Illumination normalization for face recognition using the census transform. 2008.
- [24] Richard Lawler. Google tv officially switching to arm, marvell armada 1500 cpu to lead the charge. <http://www.engadget.com/2012/01/05/google-tv-officially-switching-to-arm-marvell-armada-1500-cpu-t/>, January 2012.
- [25] Hung-Son Le. Face recognition: A single view based hmm approach. 2008.
- [26] Juang-Ah Lee. Samsung to sell google tv. <http://online.wsj.com/article/SB10001424052970204124204577151433502080906.html>, January 2012.
- [27] Stanley Osher Leonid I. Rudin and Emad Fatemi. Nonlinear total variation based noise removal algorithms. 1992.
- [28] Sotiris Malassiotis and Michael G. Strintzis. Robust face recognition using 2d and 3d data: Pose and illumination compensation. 2004.
- [29] Tatsuo Kozakaya Masashi Nishiyama and Osamu Yamaguchi. Illumination normalization using quotient image-based techniques. 2004.
- [30] Jared Newman. Logitech walks away from google tv after £62 million revenue failure. <http://www.computerworlduk.com/news/it-business/3317927/logitech-walks-away-from-google-tv-after-62-million-revenue-failure/>, November 2011.

-
- [31] Margarita Osadchy and Daniel Keren. Efficient detection under varying illumination conditions and image plane rotations. 2003.
- [32] Michael J. Jones Paul Viola. Robust real-time face detection. 2004.
- [33] Georgina Proadhan. Google tv to launch in europe next year. <http://www.reuters.com/article/2011/08/26/us-google-edinburgh-idUSTRE77P35820110826>, August 2011.
- [34] John Woodfill Ramin Zabih. A non-parametric approach to visual correspondence. 1996.
- [35] Don Reisinger. Samsung tv apps downloads near 10 million — the digital home. http://news.cnet.com/8301-13506_3-20104789-17/samsung-tv-apps-downloads-near-10-million/, September 2011.
- [36] Amnon Shashua and Tammy Riklin-Raviv. The quotient image: Class based re-rendering and recognition with varying illuminations. 2001.
- [37] Xiang Sean Zhou Dorin Comaniciu Terrence Chen, Wotao Yin and Thomas S. Huang. Illumination normalization for face recognition and uneven background correction using total variation based image models. 2004.
- [38] Jianhuang Lai Pong C. Yuen Xiaohua Xie, Wei-Shi Zheng. Face illumination normalization on large and small scale features. 2008.

Appendix A

Example Script

```
1 var x = 0.0;
2 var y = 0.0;
3 var zoom = 1.0;
4
5 (function () {
6     AdApp.finished();
7     AdApp.loadImage('http://www.acc.umu.se/~onionkgt/glasses.png', 'glasses');
8     AdApp.loadImage('http://www.acc.umu.se/~onionkgt/facerect.png', 'facerect');
9     AdApp.loadImage('http://www.acc.umu.se/~onionkgt/leyedot.png', 'leyedot');
10    AdApp.loadImage('http://www.acc.umu.se/~onionkgt/reyedot.png', 'reyedot');
11 })()
12
13 function update (dt) {
14     faces = eval('(' + AdApp.getDetectedFaces() + ')');
15     var f = null;
16     if (faces.length > 0) {
17         f = faces[0];
18     }
19     AdApp.drawImage('camera',
20         0.0, 0.0, 0.0,
21         1.0, 1.0, 1.0,
22         0.0, 1.0, 0.0, 0.0);
23     if (f != null) {
24         /*AdApp.drawImage('facerect',
25             f.face.left, f.face.top, 0.0,
26             (f.face.right - f.face.left)/2.0, (f.face.bottom - f.face.top)/2.0, 1.0,
27             0.0, 1.0, 0.0, 0.0);*/
28         AdApp.drawImage('leyedot',
29             f.lefteye.x, f.lefteye.y, 0.0,
30             0.05, 0.05, 1.0,
31             0.0, 1.0, 0.0, 0.0);
32         AdApp.drawImage('reyedot',
33             f.righteye.x, f.righteye.y, 0.0,
34             0.05, 0.05, 1.0,
35             0.0, 1.0, 0.0, 0.0);
36         var dist = Math.sqrt(Math.pow(f.righteye.x -
37 f.lefteye.x, 2) + Math.pow(f.righteye.y - f.lefteye.y, 2));
38         var midx = (f.lefteye.x + f.righteye.x)/2.0;
39         var midy = (f.lefteye.y + f.righteye.y)/2.0;
40         AdApp.drawImage('glasses',
41             midx, midy, 0.0,
42             dist, dist, 1.0,
43             Math.atan(f.righteye.y - f.lefteye.y, f.lefteye.x -
44 f.righteye.x)*180.0/Math.PI, 0.0, 0.0, 1.0);
45     }
46     AdApp.finished();
47 }
48
```

```
49 function onKeyPress (keyCode, state) {
50     // Up: 19
51     // Down: 20
52     // Left: 21
53     // Right: 22
54     // Ctrl: 113
55     // Rec: 130
56     // Rewind: 88
57     // Fastforward: 87
58     if (state) {
59         if (keyCode == 19) {
60             y += 0.1;
61         } else if (keyCode == 20) {
62             y -= 0.1;
63         } else if (keyCode == 21) {
64             x -= 0.1;
65         } else if (keyCode == 22) {
66             x += 0.1;
67         } else if (keyCode == 88) {
68             zoom /= 1.1;
69         } else if (keyCode == 87) {
70             zoom *= 1.1;
71         }
72     }
73 }
```