Master Erasmus Mundus in
Color in Informatics and Media Technology (CIMET)

Eye Contact in Leisure Video Conferencing

Master Thesis Report

Presented by

Annick van der Hoest

and defended at

Gjøvik University College

Academic Supervisor(s): Simon J. R. McCallum

Jury Committee: Prof. Roman Bednarik
Prof. Luis Gómez Robledo

Eye Contact in Leisure Video Conferencing

Annick van der Hoest

2012/07/15

Abstract

The concept of *presence* refers to the sensation of being in a physically distinct location. Presence with respect to video conferencing refers to feeling as if together with the remote user rather than being conscious of the actual separation by distance. Eye contact plays an essential role in communication as it conveys additional nonverbal messages of the conversation. In video mediated distance communication eye contact is prevented due to the offset of the camera position to the user's gaze. In defiance of the imposed restriction, enabling eye contact in video mediated communication was the approach taken to improve presence in video conferencing. A particular focus was placed on private rather than professional video conferencing. The proposed solutions include a software based solution founded upon image processing techniques and a hardware based solution founded upon teleprompter technology. Two experiments were conducted to test the performance of the systems.

The first experiment was designed to measure the sense of presence within each solution, comparing them to the conventional situation where the camera is placed on top of the screen without correcting gaze. Evaluation of the results of the first experiment demonstrated that the solutions generated a significantly greater perception of eye contact than the conventional setup. The perceived eye contact was rated as an improvement by the majority of the participants. However we could not verify that enabling eye contact improves the sense of presence with respect to the conventional situation. This will require further experimental research. The second experiment was designed to measure the difference in sense of presence between the solutions. Evaluation of the results established the conclusion that the hardware solution induced a greater sense of presence than the software solution. Albeit both systems would profit from adjustments and improvements, both acquired positive feedback and showed great potential.

Preface

The issue of the inability to have eye contact when video conferencing has occupied various intellectuals [1, 2, 3, 4]. Research evaluated the importance of eye contact in conversation [5, 1] and was used to create eye contact enabling solutions [6, 7, 8]. While to date the area has not been exhausted, companies have developed and marketed diverse solutions [9, 10, 11]. The main issue with the available products on the market is that they aim at professional use, which is accompanied by a large expenditure. Driven by a personal motivation, the need for a consumer market suitable eye contact enabling solution was the inspiration to develop low cost yet fair quality solutions.

The proposed software solution in this work is based on image processing techniques. The majority of the researches in software based eye contact enabling solutions employ multiple cameras [8, 12, 7, 9]. The advantage of the proposed software solution in its current form is that no additional hardware is required, i.e. a single camera is sufficient. In order to increase performance a suggestion is made to include multiple cameras in future work. However in contrast to most of the researches there will not be a need to compute a virtual camera view. Performing a Gaussian blend to overlay template eyes requires less computing power which makes real time processing manageable across devices of different quality. Jerald & Daily [3] discuss a single camera solution similar to the proposed solution. A minor difference is that in their approach the camera is placed on the side of the screen. The resulting eye contact is upon an averted face due to excluding head position adjustment, which looks unnatural. In the proposed system this issue is avoided by placing the camera on top of the screen. A more essential difference is that their eye tracker is initiated by clicking on specific points in the eyes. In this work the eye capturing is automated by blink detection. The distinction between the systems is based on prioritising usability over precision. Our system requires limited user configuration so that users unfamiliar with the system could use it as part of leisure video conferencing. For research purposes the software solution was not only compared with conventional leisure video conferencing systems, but also compared to a hardware typed solution. Commercial hardware solutions to enable eye contact have been around longer. A low-budget hardware solution was developed in order to compare the software solution to a hardware solution that also aimed at leisure use.

The proposed hardware solution in this work is based on teleprompter technology. A number of the existing solutions on the market are based on the same technology. An example is TelePresence Tech [10], which is a high quality large system, suitable for professional distance business meetings but not for leisure use due to its size and cost. Another commercial product is the iris2iris [11], which is a computer monitor with a beam-splitter at 45° in front and a vertical facing camera at the bottom. The product being less professional is also less costly than TelePresence Tech, though still not within consumer market boundaries. Its cost, requirement for a phone connection and cumbersomeness, by including an entire additional monitor to the hardware users already own, make the system unsuitable for consumer use. A teleprompter

technology based marketed solution that aims at consumer use, is the SeeEye2Eye [13] which is a device that is to be placed over an integrated or external webcam. The device is portable and affordable but the drawback is the size of the display. The eye contact with the remote user is gained at the cost of losing a large part of the display area. This work proposes a hardware solution that employs the entire display of the device used. The use of a netbook or tablet for video communication implies user acceptance of intrinsically smaller screen sizes than laptops or computer monitors. When using the teleprompter on these devices the user will not be sacrificing display area to enable eye contact. The hardware solution proposed was developed with cheap materials, maintains acceptable quality and does not involve a loss of display area. These features combined make the solution distinct from existing teleprompter based solutions, satisfying the intention for private use.

Acknowledgements

I would like to thank my supervisor Simon McCallum, Associate Professor at Gjøvik University College. His comforting and encouraging words in the tough moments never failed to boost my confidence and ambition. I am grateful for all the inspiration and motivation given by his two hour lectures to short questions. Thank you.

Much gratitude and appreciation also goes out to Jayson Mackie, research assistant for the Game Technology Group at Gjøvik University College. Not only did his contribution make everything happen, he was also always available to teach and assist me in the process of learning C++. Without you I would not be able to look anyone else in the eye but myself. Thank you.

Finally I would like to thank my family and friends, who succeeded very well in taking my mind off the Thesis from time to time. Cheers!

Contents

List of Figures

List of Tables

1 Introduction

The topic of this Master Thesis is eye contact in leisure video conferencing. This was chosen based on a personal motivation. A passion for traveling is accompanied by a prolonged absence from home and family. Being able to talk to and importantly see family and friends while being great physical distances apart, makes non-professional video conferencing an invaluable tool. The concept associated with the sensation of truly being with another person that is physically somewhere else, is presence. In Knudsen's extensive research in the field of presence [14], the objective was to improve the quality of long distance communication by increasing the sense of presence. This is followed in this work and the focus was on enhancing users' sense of presence. The conventional setup in leisure video conferencing prevents eye contact between users due to the offset of the camera position to the user's gaze. This creates a problem because eye contact plays an important role in communication as it conveys additional nonverbal messages in the conversation [1, 5]. In achieving a sense of presence eye contact seems to provide a substantial contribution [14]. Therefore the topic of the Thesis concerns enhancing presence by providing solutions that enable eye contact in video mediated communication.

Eye contact incorporated video conferencing systems exist for professional video conferencing [9, 10, 11]. However, the available systems require a large expenditure and are therefore not suitable for leisure video conferencing. Therefore this Thesis presents two solutions that enable eye contact in casual video conferencing:

- A software solution based on image processing techniques
- A hardware solution based on teleprompter technology

Two experiments were conducted to examine the user experience for the proposed solutions. Evaluating the results of the experiments served to answer the following research questions:

1. Does enabling eye contact in video conferencing improve user experience with respect to the sense of presence?
2. Does the software or the hardware solution induce the greatest sense of presence, designating this to be the preferred solution?

The structure of the paper is composed by the consecutive chapters addressing background, methodology, results, conclusion and future work. Chapter 2 is the background chapter which provides information on the different fields concerned with presence and telecommunication. It discusses related works to solutions for eye contact in video conferencing. Moreover the technologies used in related works and in the proposed solutions are discussed. Chapter 3 is the methodology chapter which describes the proposed software and hardware solutions. This includes a discussion of the various initial implementations and explanations for failures. Moreover an overview of existing user experience measurements methods is given, from which the employed user experience measurement method was adopted. Chapter 4 is the results chapter and discusses the conducted experiments, including the experimental setups, the experimental results and the evaluation of the results from which a conclusion is drawn. Chapter 5 is the conclusion chapter which summarizes the main points and results of the presented work and provides answers to the posed research questions based on the obtained conclusion of the experimental results. Chapter 6 is the future work chapter and poses ideas for continuation and improvement of the presented work.

2 Background

This paper proposes a software and hardware based solution to the problem of eye contact in the field of video conferencing. This chapter provides the backgrounds concerning the different fields that meet in this project. The primary aim of the project is to improve *presence* which is a concept explained in Section 2.1. The approach taken to improve presence in video conferencing was enabling eye contact, due to the role of eye contact in communication. The importance of body language and in particular eye contact in communication will be discussed in Section 2.2. This is followed by an overview of different existing approaches and related researches to improve eye contact in video conferencing in Section 2.3. Next multiple camera geometry is discussed, which is the theoretical background for a common approach to a software solution for the eye contact problem namely generating a view from a virtual camera. This approach was used in a number of related researches but was not the approach used in this work. The opted approach for a software solution was gaze correction based on image processing techniques. In order to correct the gaze in a live video stream the eyes need to be tracked in real time. Eye tracking is discussed in Section 2.5. The location of the eye pixels and their intensity values are used for the gaze correction. Image processing techniques that can be used for the gaze correction are discussed in Section 2.6.

2.1 Presence and TelePresence

Presence is defined by Steuer [15] as a person's experience of being in an environment. When one is perceiving the world without using any media, the sense of being present in the physical world is trivial. However, in the case of mediated perception one can be present in a physical environment but *feel* present in the mediated distinct or remote environment. Steuer defines this as **telepresence**. In this paper the area of interest is telepresence. In literature there is a convention of using the term presence to refer to telepresence. This tendency will be followed in this paper.

According to Enlund [16] there are three factors that determine the sense of presence: *individual preconditions present*, the *characteristics of the content delivered* and the *sensory environment* (i.e. technologies used). Individual preconditions differ among people, such as the ability to imagine things. Due to its subjective nature, it is difficult to influence this factor. Therefore the content and technology factors are more feasible to influence in order to produce and improve a sense of presence.

The characteristics of the content is an important factor in creating a sense of presence. This is addressed by Knudsen [17] who states that presence is established by storytelling rather than by using advanced technologies. This becomes clear when looking at examples of dreaming or reading a good book. One can be completely engaged in the story by a gripping story or dream. An example of an experimental project that demonstrates this point is Knudsen's dance performance project BOOM [18]. In this project the audience was situated in a physical space, facing a stage with two percussionists. In another physical space the dancer was present together with the producer Knudsen of the project. The performance composed of the dancer and the percussionists was set in another space, namely a virtual environment. The synchronous performances were captured by video and put in a virtual room which was then displayed on screen visible to the audience. This project uses telepresence technologies as a medium for art creation. A sense of presence is obtained because the story is captivating. Distance education is another example where experiments show that the technology can become transparent and the students feel present in 'class', depending on the content of the class [16].

The sensory environment is made up by the technologies used. Presence technologies include all technologies that are used to make a user feel present in a remote environment. This involves technologies such as mp3 players when used to listen to a spoken book, head mounted displays used in gaming and Voice over IP to deliver voice communications over internet protocol. The Sections 2.1.1, 2.1.2 and 2.1.3 address the content and technology factors to induce presence in arts, gaming and video conferencing respectively.

2.1.1 Presence in Arts

The example of Knudsen's dance performance project BOOM [18] is mentioned in Section 2.1, which demonstrates the importance of content to induce a sense of presence. The example uses art to induce presence rather than advanced technology. Conversely, technologies are also used to improve the sense of presence in arts. For example, at the Coachella Music Festival the deceased Tupac performed live on stage. This was the latest digital resurrection carried out at the time of writing, using technology patented by Musion [19]. This technology is based on *Pepper's ghost*. Pepper's ghost is a technique used to create the illusion of a person or object being in a location that it is not [20].

It was used in the 1860's in theatres to materialize people or objects in a scene [21]. The illusion involves two rooms, a main room and a hidden room. The hidden room is not visible for the viewers. A piece of glass is placed between the viewers and the main room. This piece of glass shows the reflection of the hidden room. When the hidden room is dark there is no reflection visible, then the main room is brightly lit such that it is clearly visible to the viewers. When the hidden room is lit there is a reflection visible and the main room is dimmed slightly in order to compensate for the extra light coming from the reflection of the hidden room. In this state the viewers can see the main room with the reflection of the hidden room. To create the illusion of a person or ghost or object to appear out of nowhere in the main room, there are two approaches. The first approach is to make the hidden room as a mirrored image of the main room. Then when the reflection of the hidden room is visible, the viewer sees only that what is additionally present in the hidden room. The other approach is to make the hidden room entirely black with only light-coloured objects in it. Only the objects will have a reflection in the mirror, because black does not reflect light. The result is that viewers see people or objects appear in the main room, when these are not physically there [20]. The more real this illusion is perceived by the viewers, the greater their sense of presence because they are then more engaged in the spectacle. Musion uses more advanced technology but the principle is based on the Pepper's ghost illusion. A high definition projector is placed above and in front of the ceiling. An high definition image or video stream is then projected onto a reflective surface that is placed in front of the stage. The viewers are unable to see the reflective surface. The image is then reflected onto the Musion Eyeliner foil, which is placed at a 45 ° angle across the stage. The viewers see the projected video in the reflection of the foil. However the foil is sufficiently thin and smooth such that it is imperceptible to the viewers. The result being that the viewers perceive what is being projected as 3D images on stage [21]. A similar setup is used to build teleprompters, which enable a newsreader to simultaneously read the news and look into the camera. Teleprompter technology is also used in video conferencing to enable eye contact between users and is discussed in Section 2.3.1. Musion Systems [19] technology create the illusion of displaying 3D holographic images on stage. Holography and photography differ in their approach to capture an object or scene. Photography captures the light or the electro-magnetic radiation, whereas holography captures the light scatter coming from the object. Digitally a photographic image is stored as pixel values that resemble the colour and light intensity (for example the red, green and blue intensity values for an RGB image). Holography registers the amplitude and phase of the wave fronts (J. Campos, personal communication, May 24, 2011). Because a hologram contains light scatter information, the appearance of the image changes when it is seen from a different perspective. This defines a hologram as 3 dimensional (J. Campos, personal communication, May 24 ,2011). A photograph shows the same image, regardless of the perspective of the viewer and thus is 2 dimensional. The Musion Eyeliner systems use 3D HD images and project them as 2D HD images. Just as an image in a mirror appears to be 3D, the images reflected by the foil appear to be 3D. Looking at these seemingly 3D images displayed on a 3D stage, causes the viewers' minds to create the 3D illusion [21]. The success of using Musion Eyeliner technologies lies both in the content and the technology. The technology has been used to have deceased artists perform again, for alive artists to perform with animated characters or for alive artists to perform in different locations in the world at the same time [21]. The content in these examples is highly captivating and induces presence. However, the technology used in the examples is of high quality and therefore assists the content in inducing presence.

2.1.2 Presence in Gaming

In gaming the content of the game plays an apparent important role in the sense of presence of the gamer. In an article named *What Makes a Game Good?* [22] there are many aspects described that point to the content of the game. The story should be original, stay fresh when played repeatedly and contain surprises. The structure should be such that at the start all players have an equal chance of winning, the rules should be consistent and in the case that the game is not based on chance the players should be given the ability to affect the progress of the game. The tension caused by the game should be a balance between difficulty and skill of the player. When the game is too difficult, the player gets frustrated and when the game is too easy the player gets bored (S.J.R. McCallum, personal communication, October 19, 2011).

The technology factor of presence applied to gaming spans a broad field because theoretically anything can be turned into a game and anything can be used as game inputs. Computer games include many different types such as sports, racing, war and social games. Many games can be played either as a single player against the game or as a multi-player game against other people.

Many gaming devices use joysticks or button controls as inputs for the game but there are other possibilities. In car racing games a control can be used in the shape of a steering wheel. For shooting games a toy gun can be used as input. Nintento's Wii uses body movements as input, where the player holds a wireless device which can detect movements in three directions. Thus in a tennis game the player holds the device and makes an arm movement such as one would make when playing real tennis. This is thus closer to reality which means the player it likely to feel more like playing real tennis than when using a number of keys or joystick as input. Microsoft has introduced the Kinect which approaches presence even more than the Wii.

Kinect

Microsoft Kinect^{TM} [23] is equipped with an infrared (IR) projector, an IR camera and a RGB camera. The IR projector projects a known IR light pattern at a certain depth. The objects in the scene distort this pattern which is recorded by the IR camera. The offset of the recorded IR pattern compared to the known pattern gives information about the depth of the objects. In this way the foreground and background can be distinguished from each other. By deducting sequential frames from each other, information is gathered about the motion that occurred in the scene. The Kinect also does object recognition which means it can recognize whether there is a person in front of the camera. The combination of these features enables the Kinect to recognize the movement a person is making. The result is a control-free system that uses body movement as input. On top of this the Kinect does voice recognition which means that the player can use his voice to give commands to the Kinect. Using voice and body movement as input for the game makes the game feel more real, thus presumably increases presence, but presence goes further.

Controlling Avatar Motion

In first person games the player experiences the game from the perspective of the avatar. The Wii and Kinect are able to detect motions performed by the player but the avatar motion cannot be controlled completely by the movement of the player. This is because the setting of the games is a virtual environment which is potentially infinitely large whereas the player is in a room with limited dimensions. An approach to let the player control the avatar motion completely is to capture the motions of the player and apply *motion compression* [24]. This maps the motion performed by the player in the limited space to the avatar motion in the infinite space and is explained by Groenda et al [24] as follows. The player wears a head mounted display that shows the virtual game world to the player and is equipped with four microphones and a gyroscope. In the room there are four speakers mounted in different locations, each generating a unique sound. This setup enables knowing where the player is located in the room and where his head is directed to. The first step in the motion compression algorithm is path prediction. Humans tend to look in the direction they want to go thus a target path can be predicted based on the head motion. The target path is where the avatar in the virtual world wants to go. This path is then mapped onto a user path in the user environment. In the mapping the distances and turning angles are kept locally equal but the curvature is adjusted such that the user path stays within the dimensions of the room. The last step involves guiding the player to follow this mapped user path, which then results in the intended target path in the virtual environment. The user should not notice that he is actually walking a different path than the avatar is walking in the game environment. Humans constantly adjust their direction in order to walk straight to their target. By slightly rotating the gaze of the avatar the player automatically compensates for this rotation. The result is that the player is walking in a curved transformed path in the user environment as shown in Figure 1a) but the avatar follows a straight path to the target as shown in Figure 1b). Groenda et al. [24] discuss the application of controlling avatar motion to the games Quake and PacMan, resulting in respectively MCQuake and PaMCan.

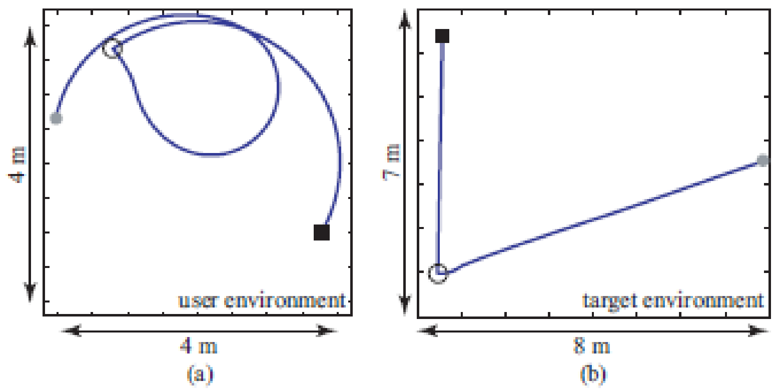


Figure 1: Motion compression. a) The transformed target path resulting in a curved user path in the user environment b) The corresponding target path in the virtual environment. Reprinted with permission from “Telepresence Techniques for Controlling Avatar Motion in First Person Games” by Groenda et al. (2005) [24].

Battlefield 3 Simulator

The Gadget Show took feeling present in gaming to a whole different level [25]. They built a simulator for the first person shooter game Battlefield 3. Battlefield 3 uses the graphics engine Frostbite 2, which had the best graphics and sound to date available. Moreover it uses a character and animation technique that make the characters in the game appear as realistic as possible to date. The Gadget Show created a simulator for the game with the intention to make the player really feel present in the game. The game was projected in a 360 ° video dome provided by Igloo Vision. The game shows a field of 180 ° and by tracking the direction in which the gun is pointing that the player holds, the game is always projected in front of the player. When the player turns around the game is projected there, leaving the player to always see the game. The dome has limited dimensions unlike the virtual world of Battlefield 3 but instead of using a motion compression algorithm they used a hardware solution namely an omni-directional treadmill designed by MSE Weibull. For every direction at every speed, the treadmill corrects for the player motion by moving him back to the centre. In this way the player can walk and run anywhere without actually moving more than a certain distance and without noticing the correction. Furthermore they used an appBlaster gun. This is a wireless toy gun with a mobile smart phone mounted on top. The gun has two triggers where the front one is used for aiming and the rear one for shooting. The information is then sent from the mobile phone to the computer that runs the game.

They use infrared cameras for motion detection and hacked the Kinect in order to also detect jumping and crouching motions from the player. A pixel mapping algorithm is used to control the lighting in the dome, which consists of 800 LEDs. Thus, the ambient lighting corresponds to the displayed pixel values on the screen. Apart from controlling the lighting, the pixel mapping algorithm is used to control a number of paintball guns located at different positions around the dome. When the player is shot the screen lights up red from the blood, which is used as an input to trigger those paintball guns located at more or less the position where the shot in the game came from. Having the game displayed 360 ° around the player, with corresponding ambient lighting in the entire playing area, the ability to move around freely without limitations and actually being shot when being shot in the game leads to a truly immersive game. The problem with this simulator is that it is not built for mass sale and for every game lover to have at home. The approach taken in this paper to improve presence in gaming therefore focuses on the eye contact problem, which can reach and thus help improve gaming for a greater public.

2.1.3 Presence in Video Conferencing

In leisure video conferencing the content can be assumed to be sufficiently captivating to be maximising the influence of the content factor on the sense of presence. Participation is voluntary and the content is controlled by the users themselves. Therefore in the field of leisure video conferencing the only factor that can be influenced in order to improve presence is the sensory environment. Similar reasoning is true for work related video conferencing. Participation might be voluntary or not but the content of the telecommunication being work related is sufficiently engaging for a presence sensation. In video conferencing the content is not in the control of the providers but should nevertheless be inducing presence.

The technology factor for video conferencing has evolved from telecommunication, which started centuries ago with using smoke signals to convey information over distance. Around the 1870's telephones were introduced, which over time evolved to mobile telephones. With the improvement of technology the sense of presence in distance communication has improved as well. Hearing the voice clearly of the other participant without any delay helps the sense of presence in the communication. But presence in telecommunication did not stop there. Apart from audio data transmittance arose the ability to send live video feed of the participants during the conversation and video conferencing was born. Conventionally computers and internet are used as the means for video conferencing but some smart phones like the iPhone can be used as well. Examples of video conferencing systems that are broadly used by the public are Skype, Google+ hangout and MSN messenger. In these systems participants can either chat with their contacts using written text, audio only or audio and video feed. These systems can be used for one-to-one chat or can involve more participants. In the case of more participants the video feed of the other participants are shown next to each other on the screen. Although the sense of presence in these systems is greatly improved with respect to regular phone calls, it is still a different experience than actually being present in the same room as the other users. There are more professional systems on the market that come closer to real communication between people. Examples are TelePresence by Cisco, HALO by HP and RPX HD by Polycom [26]. These systems support multi-user and multi-site videoconferencing setups. Users at one site sit at a table directed at a number of screens, each of which displaying the participants at a distant site. Especially if the tables are matching, this can look like and feel as if all participants are sitting at one large table, regardless of their geographic location. However there are still obstacles in the way to a true presence sensation in communication. The first problem is not being able to touch the distant communication partner. The next problem is that video telecommunication is displayed in 2D while real life communication takes place in 3D. Finally there is the problem of not being able to have eye contact in video telecommunication due to the offset position of the camera. Providing a solution for the latter problem is the aim of this paper.

Sense of Touch

It seems impossible to be able to touch a distant person, just like it seemed impossible many years ago to be able to see and hear a distant person live, like we can do now with video chat. Although it is impossible to send your hand through the screen and stroke the other person, some approaches to simulating touch over distance exist. An example is the BEAMING [27] project. This project creates a shared virtual space for the participants who then interact as avatars. The participant perceives the conferencing session as being in a shared virtual environment with remote participants, however there is the possibility for users to be embodied as a physical robot at a distant site. This enables the user to have the physical robot touch objects at the distant site for the user. Moreover haptic technology is able to give touch feedback, in the sense of applying force, vibration or motion as a result to particular cues in order to simulate touch for the user. The advantage is that the avatars are able to move around freely in a shared 3D space and can have eye contact. Moreover the aim is to add body gesture translations to the system to account for cultural differences. The avatars will automatically adapt gestures made by the participant it represents, to the appropriate gesture understood by the other participants. A disadvantage of using avatars in a shared virtual space is that the look-and-feel of avatars in a virtual world does not consort with a professional business meeting. Even out of a professional context the question remains if people would appreciate speaking to the avatars of their friends and family rather than their real selves.

3D Display Technology

In reality we perceive the world as 3 dimensional but most of what we display on screens is done in 2 dimensions. However it is possible to display footage evoking the illusion of depth. This is done by recording the scene in stereoscopic view, which means recording the same scene from two slightly different perspectives. When displaying the views they are alternated rapidly, such that the viewer does not notice the change of the perspective of the views but perceives depth. Using 3D display technologies could greatly enhance the sense of presence both in gaming and video conferencing activities, however this is not the focus of this paper. The approach used to improve presence in this paper deals with the ability of having eye contact. 3D technology involves the use of glasses which prevents the users from having eye contact. Therefore the use of 3D technology is not included in the proposed solutions in this paper.

Eye Contact in Video Conferencing

A pressing problem in video conferencing is the inability to have true eye contact among participants [1, 2, 3, 4]. In common video chat sessions the user is looking at the screen, where the other person is displayed, but the camera is located on top of the screen. The user is therefore not looking straight into the camera which prevents eye contact. Moreover in multiple user video conferencing participants are unable to tell which participant someone is looking at, causing a gaze awareness problem. The more professional systems available on the market present different solutions to this problem. These solutions involve hardware solutions such as see-through displays or half-silvered mirror systems. These are able to place the camera behind the displayed participants such that eye contact is achieved. Other solutions are based on software approaches. Multiple camera setups are used in order to either compute images as if they were recorded from a virtual camera placed in the centre of the screen, or to adjust the eye position taking the offset of the camera into account in order to simulate eye contact. These hardware and software solutions are discussed in more detail in Section 2.3.

2.2 Body Language and Eye Contact

People use body language in communication, in fact nonverbal communication reveals more information than verbal communication [28]. According to Mehrabian [29], the pioneer researcher in body language, only 7% of the message is conveyed strictly by words. Another 38% is influenced by paralinguistic aspects, which means the tone in which things are said. An example of this is sarcasm where one says one thing but means another, expressed by the tone in which it is said. The remaining 55% is captured in body language.

Body Language Gestures

In '*The Definitive Book of Body Language* Pease and Pease [28] discuss the meaning of several body language gestures. They state that people are for a large part unaware of their use and reading of body language. Although body language can be learned and observed consciously, people acquire body language skills by subconscious copying and learning from the people around them. This leads to cultural differences in body language, such as the thumbs up gesture which means 'good' to Westerners but it means 'up yours' to the Greeks (and it means '1' to Italians whereas it means '5' to Japanese). People do not analyze the gestures but merely copy them, knowing how and when to use them and knowing what others around them mean when they use them. Apart from learning body language by observing, there are also inborn and genetic gestures. Eibl-Eibesfeldt [30] found that children that were born deaf and blind use smiling expressions which could not have been learned or copied. Smiling is therefore an inborn gesture. People form 60% to 80% of their initial opinion about a person they meet for the first time within 4 minutes, which is merely based on body language. Even when giving a handshake, the person who has the upper hand and thus having the palm down is the dominant authoritative person of the two, leaving the person with the palm up to be submissive. An equal status is manifested in equally upright positions of the hands. The non-threatening meaning of showing the palms stems from back in the days when weapons were normally hidden at the wrists, thus showing the palms shows there are no weapons hidden. The 'Heil Hitler' gesture was a palm down gesture, showing authority. The showing of the palms is one of many body language aspects people are unaware of, yet use them in daily life. For example in meetings where one tries to dominate the other the handshake is unwittingly given in a palm down fashion.

Lying

Lies are detected by reading body language signals that contradict the words spoken by the lier. Gestures such as covering the mouth, or touching the nose among others indicate a person might be lying. Children cover their mouth with their hands when they lie, which is a gesture continued in adulthood more subtly by using a finger or a position where the hand touches the face covering the mouth slightly. When the listener covers the mouth this indicates a suspicion of the speaker telling a lie. The brain subconsciously tries to stop the deceitful words coming out of the mouth. A person that rubs the nose might have an itchy sensation in the nose as a defense rather than lying being the cause. However telling a lie releases chemicals that cause tissues in the nose to swell. This leads to an increase in blood pressure which makes the nerve endings in the nose tingle. Moreover it physically inflates the nose, even though it is not perceivable with the naked eye it is visible when using special imaging cameras [31]. This is known as the 'Pinocchio Effect'.

Eye Signals

Women are better at reading body language. Magnetic Resonance Imaging (MRI) brain scans show that women have between fourteen to sixteen areas of the brain to evaluate people's behaviour whereas men have four to six [28]. This is possibly due to the natural mother role of women to nurse babies with whom she only has nonverbal communication. Even though women are better than men at reading body language, the difference becomes much smaller when it comes to reading facial expressions. The face and facial expressions play an important role in body language as it is one of the primary sources to identify emotions in people [32]. In particular the eyes are important to recognise emotions [1]. A research was conducted to measure the accuracy in reading emotions from a person's eyes [33]. A photograph was shown that only revealed a thin strip of the face which included the eyes and men scored 76% and women 88% accuracy. People are in general better at decoding eye signals than body signals [28]. Eye contact can provide information attributes as attentiveness, competence and credibility of the speaker [5]. When a speaker engages in more eye contact, the audience perceives the speaker as more attentive, competent and credible. Different gazing behaviour conveys social, intimacy or power messages [28]. Social gazes remain in a triangle between the other person's eyes and mouth. Intimate gazing at close distance goes from the eyes to the chest and at greater distance goes down to the groin. In the case of power gazing the gaze is directed at a triangle from the eyes *up* to a point at the forehead. Subconsciously we sense the message people convey with their gazing behaviour. Women lowering their head and looking up communicates humbleness, because it is associated with a child's gaze. Since children are much smaller than adults they tend to look up. Raised eyebrows send a submissive message as well, in contrast to lowered eyebrows which send an aggressive message. When women use make-up to their eyebrows they tend to draw raised eyebrows which helps to provoke men's protective feelings for her. Dilated pupils indicate a state of excitement whereas negative stimulation causes a contraction of the pupils. Romantic atmospheres are associated with dimly lit places, which causes pupils to dilate. This sends the message of being interested in each other. In poker games a player's pupils dilate when seeing a good hand. Wearing dark glasses helps to prevent giving away the excitement about the good cards.

Body language is an important part of the conversation, where the eyes in particular convey additional messages about the emotional state of the conversationalists [1]. Eye contact needs to be established in order to be able to perceive these messages. Video conferencing in its conventional setup does not allow for eye contact, due to the offset of the camera location to the user's point of gaze. However Grayson & Monk [2] have shown that video conferencing users can learn to distinguish between the remote user looking at them or looking somewhere else. Grayson & Monk make the assumption that when a user can make this distinction, the user will interpret the image of the remote user looking down as eye contact as if it were mutual gaze in a face-to-face conversation. Our conviction is that knowing that the person is looking at your image does not provide the same information as eye contact. The first research question aims at investigating the influence of enabling eye contact on a user's sense of presence, based on the objective raised by Knudsen [14] that increasing the sense of presence improves the quality of the long distance communication.

2.3 Solutions for Eye Contact

The approach taken to improve presence in leisure video conferencing was enabling eye contact because eye contact seems to provide a substantial contribution to improve presence [14]. This section discusses existing solutions and related researches to enabling eye contact in video conferencing. Hardware solutions have been around longer and existing solutions are discussed in Section 2.3.1. Software solutions are more novel and related researches and existing solutions are discussed in Section 2.3.2.

Enabling eye contact to improve presence is not necessarily limited to the field of video conferencing. The field of gaming has focused on improving presence by improving the graphics, improving and using different technologies as discussed in Section 2.1.2 and making the characters and avatars more realistic, but no work has been done on obtaining eye contact. A software solution being hardware independent could be suitable for multi-player games to improve players' sense of presence.

2.3.1 Hardware Solutions for Eye Contact

There are commercial hardware based solutions in order to obtain eye contact in telecommunication. Most of these solutions are limited to professional use of the systems by cost and portability. The hardware used is expensive and can require a specific setup that is not practical in home situations but are preferred to be set in a conferencing room devoted to video conferencing. A simplistic but effective approach is to use large screens and have the user sit far away. For this to be effective the angle between the user's gaze and the camera location must be less than 6 seconds of arc. This is about the minimum disparity that the human eye can detect, which is called vernier acuity [34]. If the angle is smaller than the vernier acuity, the remote user will not notice the difference between the user looking at their image on the screen or looking straight into the camera and thus perceive eye contact. The most common hardware solution used to obtain eye contact while video conferencing is based on *teleprompter* technology. Other solutions involve display technologies.

Teleprompters

Teleprompters are used in newsreading. A newsreader looks straight into the camera, in front of which text is displayed which the newsreader can read while the camera only sees the newsreader. This is achieved by placing a display under the camera at a 90 ° angle and placing a beam splitter in between with a 45 ° to both the camera and the display. The beam coming from the display is reflected by the beam splitter towards the newsreader. The beam coming from the newsreader does not get reflected and goes straight through to the camera. A half-silvered mirror can be used as a beam splitter, or simply a piece of glass. The setup is visualised in Figure 2 where the newsreader sees the reflection of the display (the bottom beam in red) in the glass and the camera sees the newsreader through the glass (the top beam in green).

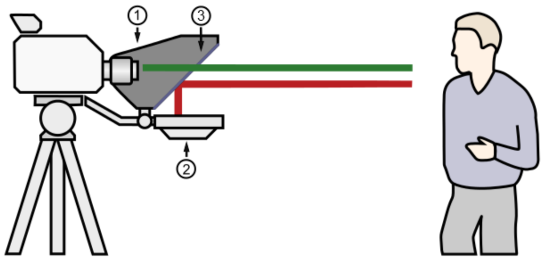


Figure 2: Typical teleprompter setup. 1) Camera. 2) Display. 3) Beam splitter.
Adapted reprint from http://en.wikipedia.org/wiki/Teleprompter.

TelePresence Tech [10] offers a number of products in different sizes and quality that are based on teleprompter technology. A camera is aligned at the eye level of the users such that eye contact is achieved, while the image is perceived by the means of a beam splitter, just like a teleprompter. The products can have life-sized dimensions, of which the purpose is professional telecommunication where the product is placed in a conferencing room without being moved around. TelePresence Tech delivers desktop products as well, where the device can be used as a regular monitor to a computer when not being used for telepresence. The focus of the products is professional use which means it exceeds the price range for regular consumers. The iris2iris is a smaller desktop device that can be plugged into a computer or laptop by a USB connection [11]. This device is based on the same principle of a teleprompter and though more compact its aim is still professional use and is also too expensive for the consumer market. An alternative solution for consumers is the SeeEye2Eye device [13]. This is an affordable device that is a teleprompter for webcams. It is placed over a webcam such that the user looks straight into the webcam while seeing the reflection of the display. Although made for consumer use, the disadvantage is that the display is rather small. Another disadvantage of teleprompter technology based solutions is that selective eye contact is not possible. When video conferencing with multiple participants, the user cannot have eye contact with one participant and not with the others. Looking straight into the camera gives the perception of eye contact to all the other users.

Display Technologies

Based on the idea of placing a camera in the user's gaze direction, other solutions to the eye contact problem use see-through display technologies. The camera sees through the display whereas the user sees what is being displayed on the display. Shiwa and Ishibashi [35] demonstrated a technology that can be used to create see-through screens. Switchable liquid crystal diffusers that quickly switch between a transparent and diffuse state. When the diffusers are in a transparent state, the camera captures images of the user and in the diffuse state the user sees the images rendered by a projector on the display. Izadi et.al [36] have advanced on this technology, introducing SecondLight. They use a switchable projection screen as an interactive surface. In diffuse state an image can be displayed on the screen and captured by a camera. The diffuse state enables touch detection much easier due to the light scattering property in this state. In the transparent state images can be displayed and captured by a camera on a surface above or in front of the screen. Object tracking can be used to find objects onto which the information is to be displayed. Motion detection can be used to recognise hand gestures that trigger a certain response. The state of the switchable projection screen can be altered sufficiently quickly such that it is imperceptible to the human eye. This means that distinct images can be displayed and recorded by camera simultaneously on a screen and on a surface above or in front of the screen. This type of screen could be used in telecommunication such that the participants can have eye contact. The camera is located behind the switchable projection screen and records the images in the transparent state, whereas the users see the displayed images when the screen is in the diffuse state. Another approach is to use *wavelength multiplexing* as done by Tan et al. [37]. In this approach the light that is projected on the display consist of wavelengths outside the sensitivity of the camera. Meaning the camera cannot observe these wavelengths and thus looks through the display, whereas the user can observe the wavelengths and perceive the displayed image. Regenbrecht & Hoermann [38] proposed a telepresence setup using a see-through screen as well. The images displayed on the transparent screen are known and can therefore be subtracted from the image recorded by the camera. Thus the user perceives the displayed image on the screen whereas the camera records through the screen which enables the participants to have eye contact. This latter approach is based on image processing techniques as well as display technology. The different solutions involving see-through screens are plausible solutions for the eye contact problem, however the technology is too expensive for consumer use and thus not the focus of this paper.

2.3.2 Software Solutions for Eye Contact

Common hardware that users have at their disposal is a PC with a single camera or a laptop with an integrated web cam. A solution proposed to the eye contact problem that does not involve more hardware than the commonly owned hardware is GazeMaster [6]. This is a software approach that first detects the head pose of the user and segments the eyes. The eyes are replaced by synthetic eyes and the face is texture-mapped on a head model which allows for rotation of the head. When a user looks at the screen while the camera is located on top of the screen, the head is tilted downwards. When one looks straight into the camera or straight into the other user's eyes the head is not tilted. Thus this is corrected by rotating the head model to a straight position. Using a general head model leads to the loss of personal features of the user, causing the user to slightly appear as an avatar. Moreover using synthetic eyes might remove the nonverbal content of the conversation contained in the participant's eyes. The idea of not using additional hardware in achieving eye contact between participants is appealing, however the tracking of the eyes proved to be too difficult with a single camera (J. Gemmell, personal communication, Januari 24, 2012). The system is not applicable for real time use in video conferencing.

The system proposed by Jerald & Daily [3] is a one camera system like the GazeMaster system. Jerald & Daily argue that vertical eye contact is more important than horizontal eye contact. Therefore the camera is placed beside the monitor, in order to have genuine vertical eye contact and they correct for the horizontal offset. In their system the user manually sets the angular offset for which the gaze should be corrected, by first looking at the remote user and pressing a key followed by looking at the camera and pressing another key. The user manually selects feature points in the eyes that are tracked and corrected when the user looks at the remote person, as if the user was looking in the camera. The correction is done by *spatial warping*. This technique is discussed in Section 2.6. The user configuration required to make the system work is not acceptable in a final product as mentioned in their paper, but it is sufficient for research purposes to demonstrate potential and try slight variations that give immediate feedback. The system does not correct for the head position which means that even though the eyes look into the camera the head is turned sideways, which looks unnatural. Both eyes have a different distance to the camera and therefore the amount of warping is different for both eyes and must be computed seprately.

The system proposed by Yang & Zhang [8] uses a pair of calibrated stereo cameras, one positioned on top and one below the display. The images obtained by the cameras are matched against a template which is a personalised face model of the user. In this fashion the head pose is tracked. Then the views of the cameras are matched, in order to match those parts of the images that are not included in the face model. Sufficient information is now available to generate a view as if a camera was placed in the centre of the screen, providing eye contact. The disadvantage of this system is the use of already calibrated cameras and a personalised face model. Videoconferencing users already own one camera already. It would maintain the ease of use to merely plug in an additional camera and be able to have eye contact with the remote participant.

When using two cameras at either side of the screen the disparity of the images is quite big. The number of reliable correspondence points that can be found in these images is quite low, which is why a personalised head model is used by Yang & Zhang to which the points are mapped. Civit & Montserrat [12] propose a system that involves four cameras, with two cameras at both sides of the screen. The small disparity of the optical centres of the cameras on the same side of the screen allow for a good search of correspondences. The positioning of cameras on opposite sides of the screen allows for correctly synthesizing the view from the virtual camera in the centre of the screen. When using only one camera on each side of the screen some parts of the image are not captured in the two views due to occlusion for example. The synthesized image then contains holes. By using two camera pairs these holes are eliminated. The system includes a camera calibration step, which means the user does not need to use already calibrated cameras. The greatest advantage of the system is that it runs in real time, thus it can be used while video conferencing. However to achieve real time performance 3D lookup tables are used for the camera pairs to generate the synthesized image, without considering temporal consistency between frames. This results in an unpleasant fluctuation effect in the video stream.

Dumont et al. [7] proposed another multi-camera system. The setup consists of six cameras arranged on a metal frame, possibly to be integrated into the monitor frame. The software framework is similar to that of Civit & Montserrat. Based on the captured images, a view is generated such that it is rendered from the centre of the screen. The view interpolation provides a dense depth map, due to the use of six cameras. The difference with the system proposed by Civit & Montserrat is that the depth information is computed dynamically. Thus there is no movement restriction nor fluctuation effect in the video stream, yet the system runs in real time. This is because Dumont et al. do the main processing on the Graphics Processing Unit (GPU) which has sufficient powerful computational resources to achieve real time performance. An eye tracking module runs concurrently on the Central Processing Unit (CPU) in order to find out where the virtual camera should be placed exactly. This is a promising system since it runs in real time and restores eye contact between participants. The disadvantage is that it involves six cameras, six times as many as users normally own. We would prefer a minimum of additional hardware to the standard hardware consumers have at their disposal.

The Fraunhofer Heinrich-Hertz Institute [9] has developed a software module called the Virtual Eye Contact Engine that uses two or three cameras to enable eye contact in video conferencing. A 3D model of the user's head is computed, using the different views of the cameras. A view is generated as if it was taken from where the remote participant is being displayed on the screen, enabling eye contact. The module performs in real time and renders high quality images. One of the objectives of the Fraunhofer Heinrich-Hertz Institute is to release a version in the future that runs on consumer laptops [39]. Being a research institute, the development of the products depends on financing by industry partners.

2.4 Virtual Camera View Generation

Several of the discussed software solutions in Section 2.3.2 that enable eye contact, use the approach of generating a view from a virtual camera [8, 12, 7, 9]. The virtual camera is rendered by generating a view from the given perspective, composed of the captured views by the multiple cameras. The theoretical discussion of camera view capture is discussed in Section 2.4.1, followed by merging multiple camera views in Section 2.4.2 and view generation from a different perspective in Section 2.4.3.

2.4.1 Camera Capture

A real world scene to be captured by a camera is 3 dimensional, whereas the image captured by a camera is 2 dimensional. Thus a camera maps a 3D scene to a 2D image plane which is called a perspective transformation [40]. Figure 3 shows a basic imaging system model which is a simplified representation of the mapping process. An object point in the 3D world scene is denoted by (X,Y,Z) which are its coordinates in the x, y and z direction. The corresponding image point is denoted by (x,y), where x,y is the location in the image plane to which the object point is mapped.

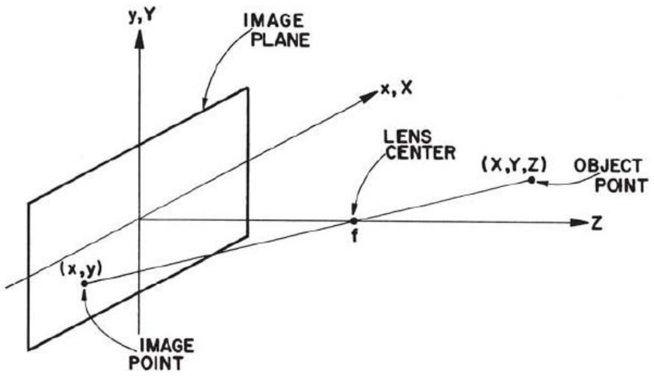


Figure 3: Basic imaging system model. Reprinted with permission from *Digital Image Processing* by Pratt (2007) [40].

The object and image point are related by an address mapping which depends on the intrinsic parameters of the camera. This means that the perspective transformation mapping is camera specific. The matrix P holds the perspective transformation corresponding to the camera and is called a camera matrix [41]. In mathematical form the mapping is represented as:

$$\tilde{w} = P\tilde{v} \tag{2.1}$$

Where $\tilde{v}$ is the homogeneous vector corresponding to v which holds the coordinates of the object point. Then $\tilde{w}$ is the homogeneous vector corresponding to w which holds the image point coordinates.

For each location in the 3D scene the corresponding location in the image plane can be found. However for each location in the image plane there are several corresponding locations in the 3D scene. This is visualised in Figure 3 where all locations along the direction of the line drawn between the image and object point correspond to the same location in the image plane. Consequentially it is impossible to extract depth information of a scene layout from a single camera captured image. Depth information is necessary to generate a view from a different perspective and can be extracted when using multiple cameras to capture the scene.

However before continuing to multiple camera geometry the simplified basic imaging system model is extended to a more realistic model. In the basic imaging system model the centre of the image plane corresponds to the centre of the world reference. In real life situations this is generally not the case. The position of the camera is relative to the reference world and its perspective to the scene must be taken into account. These are called extrinsic parameters [41]. Extrinsic parameters do not depend on the specific camera but on the position of the camera with respect to the 3D scene. The new camera imaging model is shown in Figure 4.

A camera is mounted on a gimbal centre such that the gimbal is fixed but the camera can rotate and tilt freely. The gimbal centre has an offset with respect to the reference world centre thus its position is denoted by (Xg,Yg,Zg). Equation 2.1 is based on the basic imaging model which merely takes the perspective transformation into account. In order to account for the offset of the gimbal centre to the world centre a translation matrix T_G is included in the expression. The

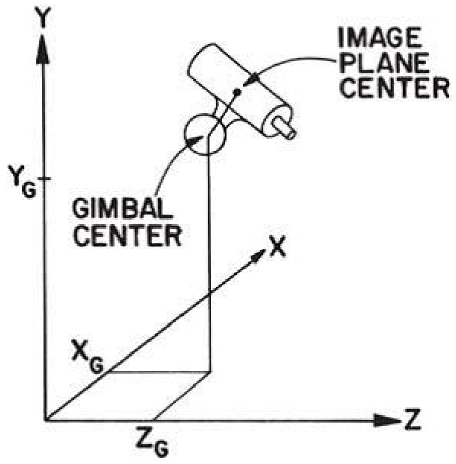


Figure 4: Camera imaging model. Reprinted with permission from *Digital Image Processing* by Pratt (2007) [40].

rotation in the horizontal direction and the tilt in the vertical direction of the camera is captured by a rotational matrix R. Finally, the offset of the image plane centre of the camera to the gimbal centre is contained in another translation matrix T_C. This results in the expression:

$$\tilde{w} = PT_C RT_G \tilde{v} \tag{2.2}$$

2.4.2 Multiple Camera Geometry

In order to retrieve depth information from a scene multiple camera views on the scene are required. Figure 5 shows the camera imaging model for two cameras. An object point X is mapped onto the image planes resulting in image point u for the first and image point u′ for the second camera. The points u and u′ are called correspondence points.

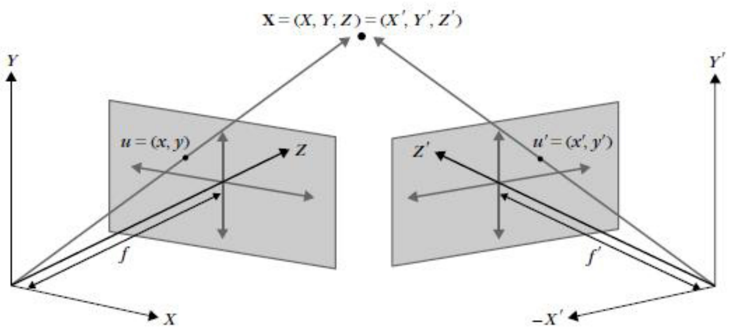


Figure 5: Two camera imaging model. Image coordinates u and u′ for object point X [41].

Epipolar Geometry

Hartley and Zisserman [42] explain the use of epipolar geometry to relate image views. For each image point X for which exist a correspondence pair the epipolar geometry can be visualised as shown in Figure 6 where the correspondence pair is denoted by x and x'. The camera centres are denoted by C and C' and the line connecting the centres is called the baseline. Connecting the camera centres and the point X results in the epipolar plane π. When capturing the scene by cameras C and C' the point X is mapped into the image plane of camera C resulting in x and it is mapped into the image plane of C' resulting in x'. The points where the image planes cross the baseline are the epipoles. An epipole is the image in the view of one camera of the other view. The intersection of the image planes with the epipolar plane is denoted by the epipolar lines l and l'. The epipolar geometry holds for any pair of images that capture the same scene from a different viewpoint and have their camera centres appear in each others' view. It describes the relation between two views, such that for a point x in an image view the corresponding point x' can be geometrically derived because x' has to lie somewhere along the epipolar line l'. When mapping a 2D image point to a 3D scene, a point x is mapped somewhere along the line CX. The corresponding point x' is mapped onto the 3D scene somewhere along the line C'X. These lines will intersect at X, which is the object point with coordinates that include depth.

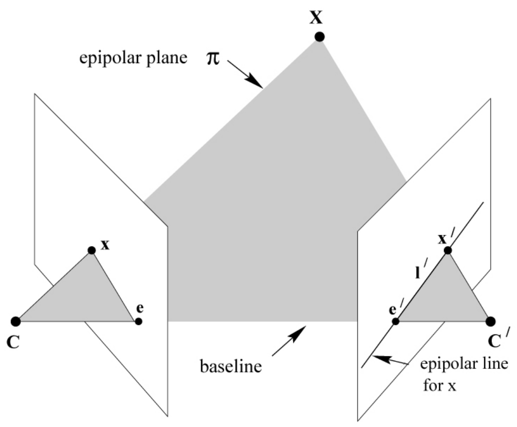


Figure 6: Epipolar Geometry. Reprinted from *Multiple View Geometry* by Hartley and Zisserman (2004) [42].

Fundamental Matrix

The fundamental matrix F is the algebraic representation of the epipolar geometry in the case that the intrinsic parameters of the camera are unknown [43]. It holds the perspective of the second image view with respect to the first image view and is defined as in Equation 2.3.

$$l' = Fx \tag{2.3}$$

By applying the fundamental matrix to an image point in the first image, the corresponding epipolar line is found. The correspondence point x' must lie somewhere along the epipolar line l'. The fundamental matrix includes the relative position of the camera centre with respect to the world centre and the relative position with respect to the other camera centre. Image pixel coordinates are used for its derivation. When the intrinsic camera parameters are known it is possible to work with normalised image coordinates. The matrix used to describe the epipolar geometry between the image views is then the essential matrix E. In order to estimate the fundamental matrix F a number of known correspondence points are used. Single Value Decomposition is a technique that finds the best fit for a matrix such that the error is minimised and is used when mapping the correspondence points using the matrix F.

Correspondence Points

A number of known correspondence points are required to determine the fundamental matrix F. In order to obtain correspondence points in image views without applying epipolar geometry, distinctive feature points in the images are used. Feature points are detected in the images separately. An example of a feature detector is the Harris corner detector, which has a strong invariance to rotation, scale, illumination variation and image noise [44]. The Harris corner detector detects corners as features, based on a specific intensity structure of a local neighbourhood by applying a local auto-correlation function on Gaussian smoothed images [45]. Other feature detectors are possible such as SURF and SIFT which find distinctive features based on different characteristics. Matching the feature points of the first image to the feature points of the second image determines the correspondence points. For all combinations of feature points a match strength is assigned by obtaining the cross-correlations of their image intensities. The feature pair with the highest correlation is elected as a match [45] which defines a correspondence pair.

2.4.3 View Generation

To generate a view as if taken from a different perspective or camera position with respect to the scene, merely one camera view is required. A 3D scene point is mapped to a 2D image point by applying matrices containing the intrinsic and extrinsic parameters of the camera. This is discussed in Section 2.4.1 and the obtained expression is shown in Equation 2.2. This expression is useful for understanding how views are generated from perspectives other than that of the capturing the camera [40]. Suppose that two views on a scene are obtained by using one camera. Then two expressions can be defined in which the matrices P and T_c are equal because the views were taken with the same camera. Moreover, $\tilde{v}$ is equal because it resembles the same object points. R and T_G describe the positioning (location and rotation) of the camera, thus these differ for the two views which leads to different $\tilde{w}$ vectors.

$$\tilde{w_1} = PT_C R_1 T_{G1} \tilde{v} \tag{2.4a}$$

$$\tilde{w_2} = PT_C R_2 T_{G2} \tilde{v} \tag{2.4b}$$

Let $\tilde{w_1}$ be the image points of the view that will be used to generate the virtual camera view. Then Equation 2.4a can be rewritten to express $\tilde{v}$ in $\tilde{w_1}$.

$$\tilde{v} = [T_{G1}]^{-1}[R_1]^{-1}[P]^{-1}\tilde{w_1} \tag{2.5}$$

The expression obtained in the above Equation 2.5 can be used to replace $\tilde{v}$ in Equation 2.4b.

$$\tilde{w_2} = PT_C T_2 T_{G2}[T_{G1}]^{-1}[R_1]^{-1}[P]^{-1}\tilde{w_1} \tag{2.6}$$

When the intrinsic parameters (i.e. matrix P) and the extrinsic parameters (i.e. matrices T_C, T_{G1} and R_1) for the camera capturing the scene are derived, the only unknown parameters on the right side of Equation 2.6 are matrices T_{G2} and R_2. These are the matrices containing the location and rotation of the other camera. These parameters can be selected accordingly to generate a view at the desired perspective. Using a single camera to capture the scene it is impossible to view parts of the scene that were not captured by the view used for the rendering of the new perspective. When rendering the new perspective the parts of the scene that were not captured by the camera are left out and appear as black. Therefore generating a virtual view on a scene requires the use of multiple views, such that there is data information in the whole scene as if taken from the virtual camera. The depth information of the scene shows which objects are to be displayed in the generated view, if the virtual camera was placed such that an object is occluded by another.

2.5 Eye Tracking

Eye tracking is used to find information about the interest of the viewer. The applications of the technology are mainly to use gaze as a control device for severely disabled people or to analyze the user attention while driving [46]. In this work eye tracking is used to locate the eyes such that the gaze can be corrected to be directed at the camera. Eye tracking measurements can either find the position of the eyes relative to the head or the position of the eyes in space. The latter is also called “point of regard” [47]. There are different ways to find the point of regard when the measurements give eye positions relative to the head. The first is to fixate a known head position such that the eye positions coincide with the point of regard. The second is to track the head position as well as the eye positions. Finally multiple ocular features can be measured to disambiguate head movement from eye rotation. The approaches in eye tracking can be divided into methods using additional hardware and computer vision image processing techniques. The Sections 2.5.1 and 2.5.2 discuss the relevant approaches belonging to the respective groups.

2.5.1 Eye Tracking using Additional Hardware

In this section three approaches to eye tracking are discussed, each using different special hardware in their method: electro-oculography, scleral contact lens with search coil and video-based combined pupil location with corneal reflection. Oculography means methods of recording eye position and eye movements.

Electro-OculoGraphy

In electro-oculography sensors are placed on the skin around the eyes of the viewer [47]. Electric potential differences are then measured which differ depending on the eye movement of the viewer. This measurement finds the eye positions relative to the head. In order to find the point of regard the head position should also be tracked.

Scleral Contact Lens with Search Coil

The sclera and the cornea form the outer cover of the eye ball. The cornea covers the iris and pupil and the sclera the rest. The sclera is therefore in human eyes the white of the eye. In this approach a contact lens is worn on the eye, where the lens extends over the sclera in order to prevent the lens from slippage [47]. A wire coil is embedded in the lens such that an alternating current electric field can be used to determine the induction current in the coil. In other words, by positioning the subject in an AC magnetic field, the position of the eyes and the eye movements are determined [48]. This is the most precise and most intrusive method available for eye movement measurements.

Video-Based Combined Pupil Location with Corneal Reflection

This approach measures the relative location of the corneal reflection of a light source to the pupil centre [47]. The light source used is usually an infra-red light source because it is invisible to the human eye and therefore less intrusive. The light source is placed at a fixed location with respect to the eye which causes the corneal reflection (called Purkinje reflection) to be relatively stable. The pupil centre is in different respective locations to the reflection, depending on the point of regard of the viewer. A calibration process captures the point of regard corresponding to the different relative positions of the pupil centre locations and the Purkinje reflections. This approach is able to measure the point of regard by measuring multiple ocular features (i.e. pupil location and corneal reflection). The positional difference between pupil centre and corneal reflection remains relatively constant with minor head movements, however the allowed head movement is limited. Furthermore there is the need for a calibration step and due to the use of an infra red light source the measurements cannot be carried out under daylight.

2.5.2 Eye Tracking using Computer Vision Techniques

The advantage of using image based features for eye tracking is that there is no need for additional equipment such as sensors, lenses or light sources.

Template Matching

Template matching finds the correlation between a template and fragments of the image by passing through pixel blocks in the image. The higher the correlation between template and fragment the higher the potential of a match. This is an approach generally used in object tracking and in the case of eye tracking images of the eye are used for the template [49]. The template consists of different gaze directed eye images. In this approach it is possible to use a personalised template where images of the user's eye gaze are used as a template. The advantage of this method is that it is easy to implement. However template matching is not robust against variations in illumination, scale, pose and shape. If the eyes in the image are in the shadow, small because the person is far away, rotated because the head is tilted or squeezed together, the performance decreases considerably and eyes might not be detected at all.

Model Based

In model based eye tracking a model is defined for the eyes which is then used as a template. Circles or ellipses can represent the pupil or the limbus (i.e. the border between the iris and the sclera which is the white of the eye) [50]. Parameters of the model can then be iteratively adjusted in order to fit the contour of the circle or ellipse better. A model can describe a circle and a coarse-to-fine gradient which represents the limbus, iris and pupil [51]. Geometrical shapes can be defined and serve as models for other facial features in order to find the eyes such as two parabolas modeling the eye lids [52]. Model based tracking is computationally expensive.

Feature Based

Feature based methods use the characteristic features of the eyes such as color, edges, and intensity distributions and gradients to locate the eyes [53]. In this approach a threshold is needed to decide whether a feature is present or absent in the image, which is generally left as a free parameter to be adjusted by the user [50]. This method can be combined with the use of infra red light sources to extract the pupil contour based on the intensity distribution [50]. Scale Invariant Feature Tracking (SIFT) [54] is an algorithm that locates points of interest in the image invariant to scale, rotation, change in illumination and noise. To increase robustness against these variations SIFT can be used to find the features to be used to track the eyes.

Haar-like features can be used for face detection [55]. A classifier is trained with positive samples (images with faces) and negative samples (images without faces). Distinctive features are extracted that describe faces. Applying the classifier to a region of interest of an image, discovers whether it is likely that there is a face in the region of interest in the image and where. The algorithm can be used to detect eyes in an image. The positive samples used to train the classifier are then eye images. Figure 7 shows an image that Zhou et al. [56] use to visualise the similarity of three Haar-like features to the structure of the eye. Applying Haar-like feature based face detection first sets the boundaries of the region of interest in an image for the Haar-like feature based eye detection.

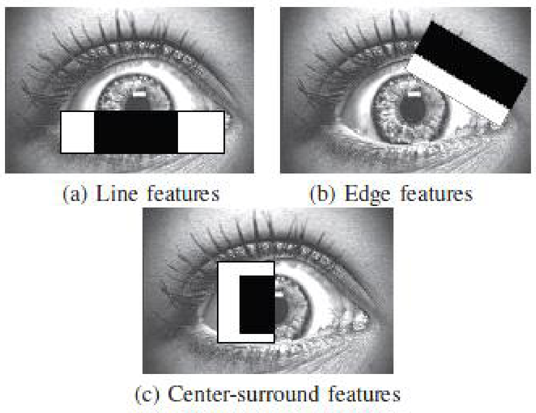


Figure 7: Comparison of Haar-like features and eye image. Reprinted with permission from “Haar-like Features Based Eye Detection Algorithm and Its Implementation on TI TMS320DM6446 Platform” by zhou et al. (2009) [56].

Appearance Methods

This method is based on the appearance of the image and uses machine learning to detect eyes. A classifier is trained by a large amount of training data consisting of images of eyes, in order to ‘recognize’ eyes in input images. The images used for training include eyes of different people, faces in different orientations and different illumination conditions. A neural network or the Support Vector Machine is used as a classifier [53]. In this approach an image is represented as a point in a high-dimensional space, thus an image can be considered as a n-component vector [57]. An example of this is discussed by Pentland et al [58] which is based on Principal Component Analysis. The images used for training are decomposed into principal components called “Eigeneyes”. The eigeneyes of the training set images form the orthogonal basis vectors of the “Eyespace” subspace [49]. The eigeneyes of a query image is compared to the eyespace in order to locate eyes in the query image.

2.6 Gaze Correction

When the location of the eyes is known, image processing techniques can be used to correct the user gaze. The software approach to enable eye contact in video conferencing proposed by Jerald & Daily [3] discussed in Section 2.3.2 uses a technique called spatial warping. The tracked eye feature points are warped onto the feature points of the eyes looking into the camera. In this work a Gaussian blend is used as the image processing technique to correct the gaze. The techniques are discussed in more detail.

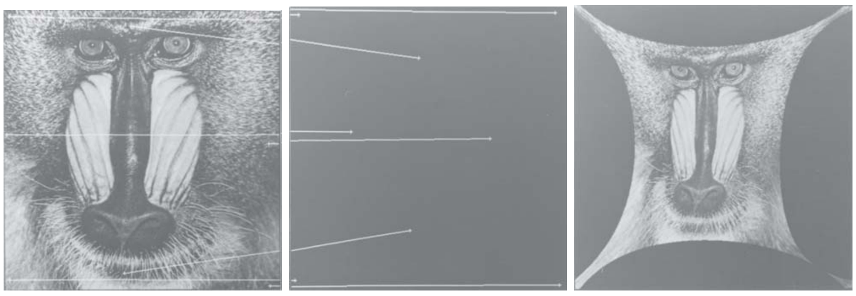
Spatial Warping

Spatial warping is a technique used to map pixels in a source image to different pixel locations in the target image. The following explanation of the technique is based on the explanation provided by Pratt [40]. Mapping pixels to different locations is an address mapping that is denoted as shown in Equation 2.7.

$$x = X(u,v) \ , \ \ y = Y(u,v) \tag{2.7}$$

The input location of the pixel u,v is mapped by function X to obtain the x-location in the target image and by a function Y to obtain the y-location in the target image. In the case of mapping the pixels of the captured eye gaze to the desired pixel locations of an eye gaze directed at the camera, the mapping functions are unknown and must be estimated. To estimate the mapping functions a set of *known data points* and *control points* are used. Known data points are the u,v locations of a number of pixels in the source image that are to be mapped to different locations in the target image. The x,y locations to which the known data points are mapped are the control points. Linear address mapping functions are easily extracted however in spatial warping and in the case of eye gaze correction the address mappings are higher order mappings. Representing the mappings between the known data points and the control points in matrix form enables the application of the Single Value Decomposition technique. This technique finds the best fitting coordinates for the mapping functions X and Y such that the error is minimised for all known data point - control point pair mappings. The process of spatial warping is visualised in Figure 8. For a number of known data points in the source image, the control point locations are found in the destination image. The address mapping functions are derived. Then the address mappings are applied to all pixels in the source image, resulting in the warped image.

With respect to eye gaze correction in videoconferencing, warping is applied solely on the eye pixels. In the video stream eye features are detected and tracked which are the known data points in the source image. In the calibration process the user looks straight into the camera and the same eye features are detected which are the control points. The mapping functions from the eye feature locations from the known data points to the control points are established. Then all pixels that the eye includes are mapped using these mapping functions. The result is an image where the user's head position and pose is the same as the captured video stream but the eyes are warped such that the eyes are directed straight at the camera. Note that the control points remain stable but the user can move his or her head and therefore the known data points can



(a) Known data points (b) Control Points (c) Warped Image

Figure 8: The process of spatial warping. For a number of known data points in the source image, the control point locations are found in the destination image. Address mappings are applied to all pixels in the source image, resulting in the warped image. Reprinted with permission from *Digital Image Processing* by Pratt (2007) [40].

have different locations which means the mapping functions can be different. Thus every frame the mapping functions should be calculated and the eyes warped. However a video conferencing user tends to have a quite stable gaze. Therefore in order to reduce computation time, the frame difference between subsequent frames can be taken first. If the frames differ in more than one pixel per tracked point then the mapping functions should be derived. If the frames differ in at most one pixel per tracked point then the mapping of the previous frame can be reused. This approach was used by Jerald & Daily [3] because they argue that a user does not notice a single pixel change. Moreover more than 50% of the frames differed by at most 1 pixel per tracked point which means this approach greatly improves performance of the system.

Blending

In image processing blending can be used to smoothen an edge transition. For example if there is a sharp edge between a light area and a dark area and this transition is desired to be smooth, then blending can be used. The pixel intensity values of the light area are mixed with the intensity values of the dark area along the edge in a certain ratio. A possibility is to use 50% of the light pixel intensity and 50% of the dark pixel intensity value. In this work blending is used to smoothen the transition from the template eye pixels to the current frame pixels. The template eyes are captured at the start of the session and are static, whereas the current frames can have changes in illumination. In order to make the overlaying of the template eyes obscure, blending is used. Not only the edges of the template eyes are blended with the current frame image, but all overlaying eye pixels are blended. In the centre of the eyes are the pupils and thus in the centre of the eyes the template eye pixels should have higher priority. At the edges of the template eyes the pixels should be closer to the current frame pixels in order to make the transition obscure. A Gaussian blending function was applied in calculating the pixel values of the gaze corrected image. The Gaussian function describes a symmetrical bell-shaped curve and the mathematical expression is shown in Equation 2.8 [59]. The A defines the height, μ defines the centre of the curve, σ defines the width of the curve and $e \approx 2.7183$ which is Euler's number. The curve is shown in Figure 9 with values $A = 1$, $\mu = 0$ and $\sigma = 1$.

$$y = Ae^{-\frac{(x-\mu)^2}{2\sigma^2}} \tag{2.8}$$

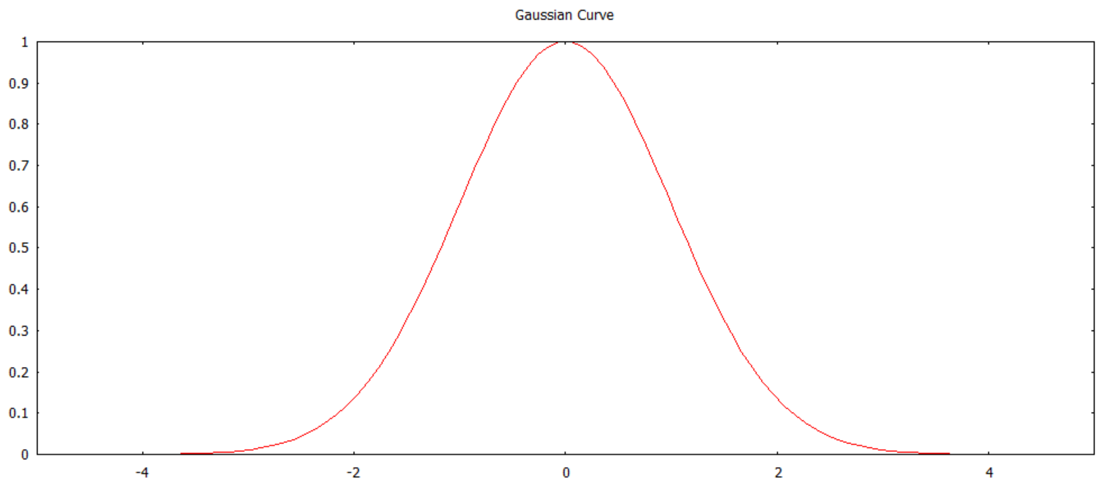


Figure 9: The symmetrical bell-shaped curve of a Gaussian function.

3 Methodology

This chapter describes the methodologies used for the proposed solutions to provide eye contact while video conferencing. Two solutions were proposed. The first solution is a software based solution based on image processing techniques described in Section 3.1. The second solution is a hardware based solution based on teleprompter technology described in Section 3.2. The performance of the solutions were determined by conducting user experience testing to measure users' sense of presence. Section 3.3 discusses several presence measurement approaches and the adopted approach in this work.

3.1 Software Solution

The software approach to enabling eye contact between video conferencing users consists of a program that detects the user's eyes and replaces them with template eyes. The source code can be found in Appendix E. An illustration of the program is given by a captured frame shown in Figure 10. The template eyes are obtained by having the user look straight into the camera and press a key, such that the frame is captured and the eyes are extracted. It is desirable for the program to be entirely automatic and unobtrusive for the user. However the thesis is time limited and therefore trade-offs must be made. The emphasis was placed on the replacement of the eyes since this is what enables eye contact, which is the topic of the thesis. The other parts of the program must function well enough to demonstrate the potential of the eye replacement and test the performance, however there was not sufficient time to perfect all parts of the program. The program is a continuation and extension to the Real Time Eye Tracking and Blink Detection with OpenCV software, provided by Nash Ruddin [60].

The aim is to obtain eye contact while video conferencing, thus the (gaze-corrected) processed images need to be transmitted over a network and received at another end. Video conferencing tools such as Skype can capture, send and receive images from a webcam between users. However they do not allow access and manipulation of the image data after it is captured by a webcam. Eye gaze correction is performed by altering the eye pixels. Therefore a networking application was developed by Jayson Mackie research assistant for the Game Technology Group at Gjøvik University College that allows image manipulation before sending it over the network. The architecture of the software as a whole is generally discussed in Section 3.1.1. The gaze correction functionality is the main concern of the presented work and is therefore discussed in more detail in Section 3.1.2.

3.1.1 Program Architecture

The software architecture is shown in Figure 11, which exhibits the program flow from capturing an image by webcam to displaying the processed image. The components above the dotted line take place at one user's end and the components below the dotted line take place at another user's end. The process displayed in Figure 11 is one-directional. In video conferencing the process is

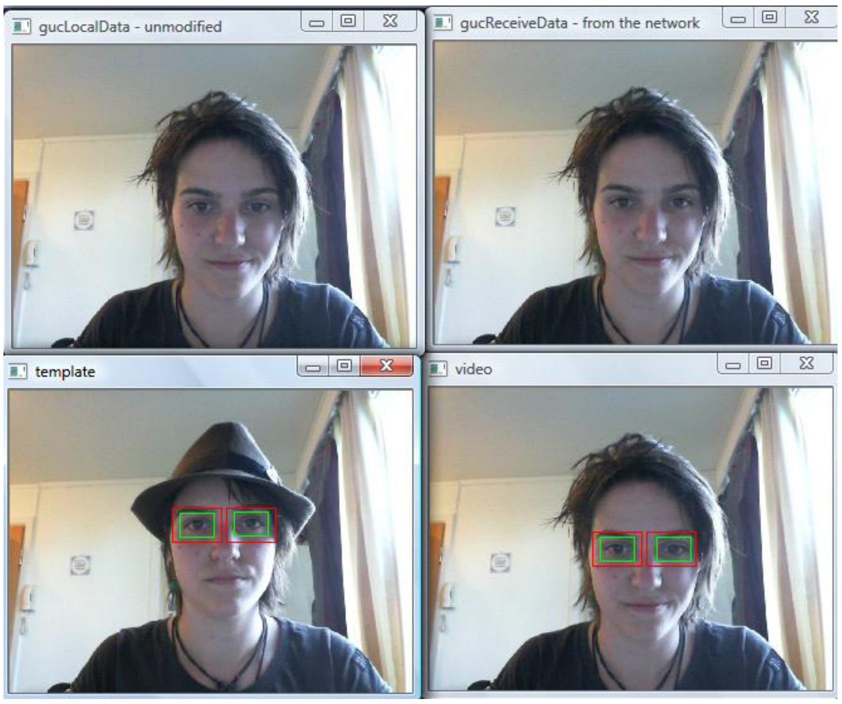


Figure 10: The software solution. A screen capture of the video stream. Left top shows the uncorrected frame. Right top shows the corrected frame. Left bottom shows the captured eyes used as template eyes to overlay on the current video frame. Right bottom shows the uncorrected current frame, with augmented green rectangles (inner rectangles) to indicate the eyes and augmented red rectangles (outer rectangles) to indicate the search window. The receiving party solely receives the corrected frame displayed at the right top.

bi-directional thus the same program flow occurs vice versa as well.

Images are captured by a webcam after which they are placed into a queue. The local processing component refers to the image processing applied to correct the eyes. The processed images are stored in another queue, after which they are send to the network. These images are received by the remote participant. The received images are stored in a queue and are then displayed. In order for the software solution to be applicable to video conferencing, the program is restricted to run in real time. The combination of storing images in queues and using parallel programming speeds up the processing and aids to meet the requirement to run in real time. The following subsections address the topics parallel programming, queues and networking. The gaze correction functionality of the program is discussed more in depth in Section 3.1.2.

Parallel Programming

Conventionally programs are written in a sequential manner, such that statements are executed one after another. The central processing unit (CPU) is where the processing takes place. The traditional approach to make computers faster was to increase the processing speed of the CPU [61]. The CPU performs a number of operations, which include arithmetic computations and transitioning to different parts of the program. A statement in a program tells the CPU with which

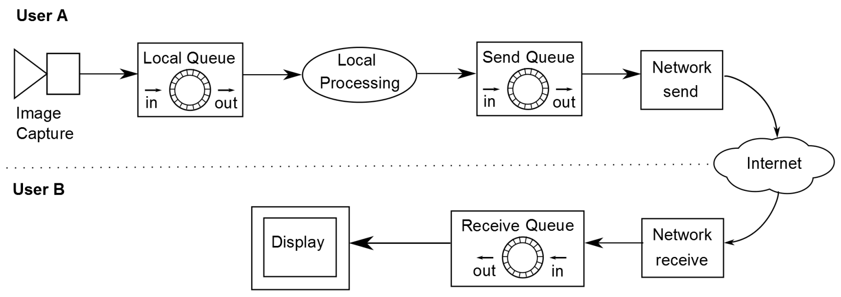


Figure 11: Program architecture. The one-directional program flow between user A and user B. Components above the dotted line take place at user A's end. Components below the dotted line take place at user B's end. Images are captured by user A's webcam. The images are stored in a queue. Image processing is applied to correct user A's eye gaze. The corrected images are stored in another queue and then send to the network. User B receives the processed images, which are stored in a queue. The images are then displayed to user B.

arguments it needs to perform which operation. The CPU consists of transistors, which are semiconductor devices able to switch and amplify electrical current [62]. The transistors that make up the CPU and together perform the operations, are thus connected by electricity. Electricity propels at a set speed, thus in order to increase the processing speed of the CPU the distance between the transistors was reduced. The maximum processing speed of the CPU has been reached because the transistors are positioned at a minimal distance to each other. Reducing the distance more would disrupt the electrical current. Therefore the approach taken to make computers faster is *parallel programming*. Parallel programming processes parts of the program in parallel rather than in sequence. This is done by using more than one processing unit. Multi-core systems contain a single chip with multiple processing units, which are called cores. Multi-CPU systems contain multiple chips, which each have one or multiple processing units (i.e. cores) integrated [63]. The processing units need to communicate with each other, because different threads can employ the same memory resource. The communication between processing units costs time. Therefore using multiple cores on the same chip rather than multiple processing chips with one core is faster because the distance between the units is smaller.

Traditionally a program is send to the CPU for processing as a whole, where the operating system (OS) divides it down into indivisible units of processing called threads [63]. These units of processing consist of code statements that cannot be split up and must be processed together. The code statements within one thread are processed sequentially in the order given. However, when the CPU is processing a thread and it awaits a response of network or hard drive access, it can start processing statements of different threads. The CPU can process one statement at a time, but the order in which threads are processed depends on how the OS schedules the processing. Not restricting the CPU to start processing another thread when one thread has been processed completely, prevents the CPU from being in an idle state when it needs to await a callback. However, different threads can require reading or writing access to the same memory resource. In the reading case this does not cause any conflict. However when different threads use and change the values in the same memory resource, 'mutexes' are used [64]. Mutex stands for mutual exclusion. When two threads share a mutex, then when one opens access to a shared memory resource, the memory resource is locked for the other thread. It remains locked until the thread that initiated access to the memory, no longer employs the resource. This is called thread synchronization and is the communication that is necessary to be able to process multiple threads on one core simultaneously. When using one core for processing the operating system handles the threading and thread synchronization. However using one core for processing, code cannot be processed truly in parallel because one core can process merely one statement at the time. In parallel programming multiple cores are used for processing. The programmer becomes responsible for dividing the program into threads that can be processed in parallel and assigning mutexes. The operating system handles which thread is processed by which core in order to achieve maximum efficiency.

The application programming interface (API) that is used to influence the threading in the provided software program is the widely used POSIX threads or Pthreads in short [64]. Pthreads are used to create, join and detach threads. A created thread can be given attributes such as whether or not it can be joined with other threads. Furthermore the API includes functionality to deal with synchronization. Threads can be assigned to share a mutex. Two threads that share a mutex cannot access the same memory location at the same time. This communication is accomplished by condition variables that signal when the memory location is occupied, which implies other threads are denied access [65].

The software solution provided to enable eye contact makes use of parallel programming and is divided into 6 threads. Figure 12 shows the one-directional program flow as shown in Figure 11 and the threads in which the program is divided. The tasks of the separate threads are:

1. The main function which includes camera capture, image processing and displaying the image.
2. Placing captured images into a circular queue.
3. Placing processed images into a circular queue.
4. Sending images to the network.
5. Receiving the images from the network.
6. Placing received images into a circular queue.

The queues in which the images are placed, prevent the program to slow down due to mutexes. Conflicting changes should not be applied to the same memory location at the same time, therefore memory that is being accessed by a thread is locked for other threads. When images are not placed into queues, the threads are paused until the memory access is unlocked. This means that the camera capture or image processing is paused until the previous image has been send to the network. By placing the images into queues, the mutex is placed on the queue. Thus the image processing of the next image can continue, while the previous image awaits its turn to be send to the network. Different data structures can be used as queues.

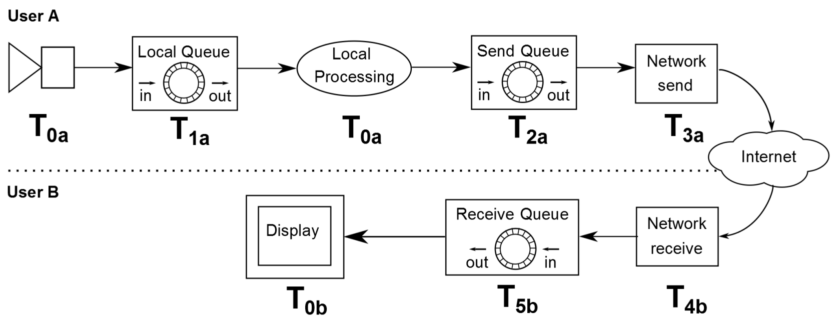


Figure 12: Program architecture of the program divided into 6 threads. One-directional program flow between user A and user B. The numbers indicate the thread number the component belongs to. The letter indicates the user's end on which the process occurs. 1: main function including image capture, image processing and image display. 2: captured image is placed in a circular queue. 3: processed image is placed in a circular queue. 4: processed image is sent to the network. 5: processed image is received from the network. 6: processed image is placed in a circular queue.

Queues

A queue is a data structure whose name and structure are analogous to a queue in the physical world. Data is stored in a 'first in first out' manner [66]. Each new image joins the queue at the end and leaves the queue when all the images in the queue in front have been taken out of the queue. In this way the order of the images is preserved, preventing the video stream to display nonconsecutive image frames.

A straightforward way to build a queue is linear, like in the physical world. In a linear queue the data can be arranged in an array, such that the front elements are removed and new elements are added at the end. The problem with an array is that it uses sequential memory locations. Considering that a video conferencing session can take longer periods of time, requiring a big chunk of sequential memory is not optimal. Therefore linked lists can be used. Linked list preserve the order in which images are added to the list, by pointing towards the memory location of a next or a previous element [66]. The elements are not physically stored in sequential memory, but stored wherever there is free memory available. Although linked lists use free memory more efficiently than arrays, it will still use up all the free memory after a period of time.

A solution to conserve a set chunk of memory that is allocated to the queue, is to use a *circular queue*. A circular queue is represented as a circle, such that when the last position in the circle is occupied a new element is added to the first position [67]. A visualization of a circular queue is shown in Figure 13.

A circular queue never exceeds the initial number of locations allocated to the queue. Physically the queue still has the shape of a linear array, but the theoretical shape of a circle is obtained by the means of pointers [67]. This is shown in Figure 14. There is a pointer that points to the start of the data, one that points to the end of the data and one that indicates the last position of the array, after which the next element should be added to the first position in the array. When the start and end pointers both point to the same element, this can mean the array is either entirely empty or entirely full. This ambiguity can be resolved in several ways, of which one

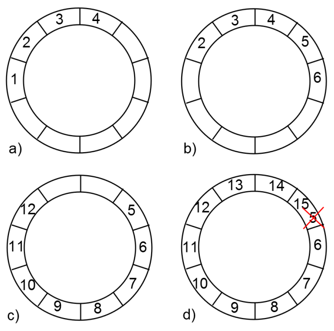


Figure 13: Visualization of circular queues. a) Insert four elements b) Insert two more elements at the end and remove an element from the start c) When the elements reach the end of the circle, new elements are inserted at the first positions of the circle d)When the circle is full, adding a new element will overwrite the oldest element in the queue.

example is to have the restriction to always keep one space open. One should be cautious about the characteristics of circular queues. When all possible positions in the array are occupied and a new element is to be added, the element will overwrite an older element in the queue.

In the provided software program circular queues are chosen as the data structure to store the images. The size of the circular queue is set to 2. If two images were stored in the queue and another image is added before an image is taken out, a frame is skipped when displaying the image frames. However the program runs sufficiently fast such that in practice the size of the queue does not exceed 1. Recall that the queues are used for mutexing threads, such that locked memory resources do not slow the program down.

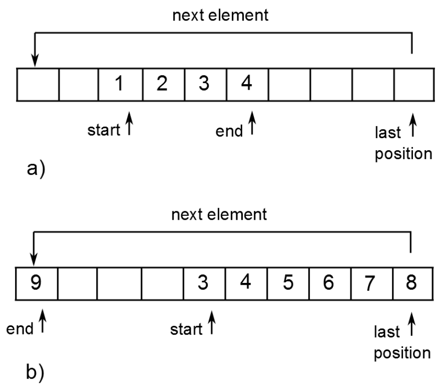


Figure 14: Circular queues structured as arrays, defined as circular by the means of pointers. a) Insert new elements at the end and remove from the beginning b) When the end pointer points to the last position of the array, insert new element at the first position of the array.

Networking

The gaze correction software is developed to be applied in video conferencing sessions, such that the users can perceive eye contact. Video conferencing is used to be able to communicate over distance. The devices used for the video conferencing sessions must be connected to each other to be able to send and receive data over distance. This connection is established by a network such as the widely used internet. A network application consists of a client side which initiates the communication process, and a server side which responds to an incoming client request [68]. A standard method for computer process communications is using *sockets*. A socket is a communication connection point that holds an IP address, a port number and defines which transport protocol is used [68].

On the server side a socket is created that holds the IP address belonging to the server device, and the port number used for the communication. The server socket has an open connection to the network and waits for a request to connect from a client socket. On the client side of the network a socket is created with the port number and IP address of the server it wants to connect to. A request to connect to that specific server is send over the network. The listening server socket receives a request to connect and determines that the IP address matches its own. The connection between the client and the server is now established. The client sends data to the server, which is then read by the server. In return the server sends data to the client (the echo) which the client reads. Either the client or the server can send a wish to terminate the connection, which is then acknowledges by the other connection endpoint. When both ends have sent a termination and acknowledgment message, the connection is aborted.

Sockets define which transport protocol is followed for sending the data over the network. A protocol is a set of rules used to exchange data. There are a number of different protocols available, one of which is the widely used internet protocol (IP). The data that is send over the network (i.e. internet) is generally too large to be send as a whole. The protocol divides the data into separate packets. A packet consists of a header and a payload. The header holds the source IP address, the destination IP address and other metadata. The payload holds the data that is to be transmitted [69]. The IP is responsible for locating the destination IP address and for routing the packets across the network. The packets are sent separately and thus can follow different routes in the network. This implies that packets can arrive in a different order than they were sent. Moreover packets can get corrupted, lost or delivered at the wrong IP address. To ensure reliable and ordered delivery of a data stream the Transmission Control Protocol is used [70].

The Transmission Control Protocol (TCP) requires an acknowledgment of reception for each packet that is sent, within a timer value [68]. If the timer expires before the good reception has been acknowledged, the TCP retransmits the packet. Finally the TCP is responsible for assembling all the packets in the correct order. Using TCP guarantees accurate delivery, however due to having to retransmit packets at times the delivery can be delayed by some seconds. Other protocols such as User Datagram Protocol (UDP) prioritize real-time delivery over accuracy [70].

The provided software program uses sockets, TCP and a local area network. On both ends of the video conferencing session there is a client socket and a server socket opened. The client socket requests a connection. When the IP address and port number of the client matches those that the server is waiting for, and the client is searching for the IP address and port number of that server socket, the connection is established. The server socket responds by sending the processed images to the client. The participants both receive the other participant's processed images. The IP address and port number of the remote participant's server socket are manually given as a command argument when executing the program.

3.1.2 Gaze Correction Functionality

The gaze correction functionality is contained in a class, such that for each captured camera frame an eye correcting method is called. The functionality consists of different parts which are discussed in chronological order, namely *eye detection*, *eye tracking*, *template capture* and *gaze correction*.

Eye Detection

At the start of the program users can see the live video stream of each other. This is equivalent to the conventional leisure video conferencing systems such as Skype. The first step is to locate the eyes of the users for which blink detection is used. Motion analysis techniques are used to detect blinking activity. The program keeps track of the difference between the current frame and the previous frame. The difference between the frames is stored in a difference image which is then thresholded. The resulting binary difference image is evaluated. If it contains exactly two connected components that are of approximately the same size (i.e. similar height and width), that have a small vertical distance and the horizontal distance between the components is reasonable in comparison to the components' widths then the connected components are assumed to be a pair of eyes. The eyes are defined as rectangles where the centres of the rectangles are the centres of the connected components. The width and height of the rectangles are hard coded, selected by testing out different sizes. The eye rectangles contain the location of the eyes within the frame.

The Real Time Eye Tracking and Blink Detection with OpenCV software provided by Nash Ruddin [60] provided a set of rules to determine whether the connected components are eyes or not. The rules were extended by restraining the position of the components; the connected components are not eyes if their position is at the far top or far bottom of the screen. This is inspired by the observation that video conferencing users tend to place their entire face in the image. This additional restriction decreases the false positives but can increase the false negatives. The position thresholds were chosen in an experimental approach by trying different values and evaluating the results, such that the performance was optimized. Thus the connected components in the difference image resulted from the eyes moving (i.e. blinking) where all other parts of the image were kept still. The difference image contains noise which is reduced by applying a convolution kernel which performs an opening morphological operation. The opening operation smooths the object contours of the components, breaks narrow connections between parts and eliminates small fragments [71]. Detecting the blinking of the eyes thus requires no movement except for the eye blinking motion, which is a limitation. However in leisure video conferencing situations users do not tend to move much, except for the lips when speaking and blinking out of reflection. Thus being still at the initialization stage is a reasonable requirement because blink detection to locate the eyes has the advantage that it is automated. The first round of experiments showed that having the blink detection as the single possibility to locate the eyes was an issue at times. With some participants the eyes were located correctly only after a couple of tries, which creates a feeling of annoyance. The program was extended with a non-automated approach to locate the eyes as a back-up if the blink detection failed. Users can manually click on the eye centres in the calibration stage, the clicked points will then be used as the location of the eye centres.

A different approach was considered namely Haar-like feature based eye detection, discussed in Section 2.5.2. The computer vision library OpenCv that was used for the implementation of this project, comes with Haar Cascade Classifiers. A Haar cascade classifier is a large number of Haar features together to form a stronger classifier. However in the presented implementation blink detection is not merely used for eye detection but also used to determine whether the gaze correction should be applied to the current frame. When the user blinks, the original frame image with closed is displayed. Due to the multiple use of blink detection, the choice was made to avoid the additional implementation of Haar-like feature based eye detection and simply use blink detection to detect the eyes. This choice was justified by the conviction that the performance of Haar-like feature based eye detection and blink detection would be analogous with respect to efficiency and accuracy.

Eye Tracking

After initially establishing the eye locations, the eye positions need to be tracked such that when the user moves his head the eye locations are updated to the current locations. The eye tracking method used was template matching. When the eyes are initially located, the images within the rectangles that define the boundaries of the pixels describing the eyes are stored and used as the templates for template matching. There is a short delay because at the time the user blinks the eyes are closed. A search window is defined around the eye rectangles and relative to the size of the eye rectangles, which are displayed as red rectangles. The search window defines the image to which template matching is applied with the stored templates. Template matching is performed by calling the OpenCV function cvMatchTemplate. This fuction takes a grayscale image and a grayscale template image and for each position of the template in the image it calculates the correlation based on pixel values. The output is a result image, which is a grayscale image where the lighter the intensity the higher the correlation. Thus this means that when a template image of an object is given, the result image shows the likelihood of the object appearing in the image and where. The index of the result image with the highest value indicates that placing the eye template at that location in the search window, gives the highest correlation. This means that the eyes will have moved to that location. Thus the best match returned by template matching gives the updated eye locations. The templates used as eye templates for the tracking are not updated during the tracking. Only when the eyes are lost and the eyes are localized afresh, the eyes at the time of the renewed capture will be used as templates.

The Ruddin's blink detection software [60] included eye tracking by template matching. However the eye tracking was improved and extended. The eye locations are only updated if the value of the best match returned is above a threshold. In the original blink detection software the threshold was hard coded, in the adjustments of the program the threshold depends on the first template match executed by the program. This was done to account for different quality cameras when running the program. The different qualities will result in different values for template matching. By making the threshold relative to the first match, the performance does not depend on the quality of the camera. The first match can be used to establish the threshold because the user can be assumed not to have moved in the time frame between localizing the eyes and performing the first template match. Thus the first template match can be considered a perfect match and the threshold is set to half the value of the first match. Moreover the blink detection software merely included one eye, which was extended to including the other eye. By including the other eye in the detection and tracking, the restriction could be added that the right eye can only be located at specific locations in the image. When requirements are not met, it can be assumed that the eyes are lost and the program resets. Then the eyes will have to be localized afresh. This was not included in the original blink detection software.

Alternative eye tracking software were examined. Opengaze is an open source gaze tracker for ordinary webcams [72], however it requires a Linux system and was therefore not suitable for this project. OpenEyes is an open source toolkit for eye tracking [73]. They provide Matlab eye tracking software that operate in combination with infrared or visible spectrum illumination. The requirement of additional hardware is a drawback of these software, as the aim is to merely use hardware users already own. The toolkit includes another eye tracker which makes use of two cameras, where a firewire camera is used to monitor the eye and a second camera is used to provide a frame of reference. The specific use of firewire cameras together with the requirement of an additional camera limit the generality of the tracker. TrackEye [49] is another open source eye tracker. The TrackEye implementation starts with a face detection. The face detection is either carried out by a continuously adaptive means-shift algorithm or by Haar face detection method. The continuously adaptive mean shift algorithm is based on climbing density gradients, which is not discussed in this work. The Haar face detection method is based on Haar-like feature detection explained in Section 2.5.2. The next step is eye detection, either by template matching like in the presented implementation or by adaptive eigeneye method which is an appearance method discussed in Section 2.5.2. Although openly online available there were issues with using the TrackEye software. TrackEye was built with Microsoft Visual C++ 6.0 and OpenCV b3.1 version. These are out-dated and in order make it compatible with Microsoft Visual Studio 2010 and the to date latest version OpenCv 2.4.0. the software would require a lot of time intensive modifications. Other open eye trackers found were SSIP2009 [74] and iGaze [75], which was written in C# using the EMGU library, but due to lack of programming experience the author was not able to get them to compile and run. Moreover with an eye on the future the author preferred to learn C++ over C# and OpenCV over EMGU as they are widely used with many applications. Before the start of the project the author was merely familiar with Matlab programming, therefore the choice was made to start from the simpler open source eye tracker blink detection which the author was capable of modifying to get it working. In this manner the author was able to learn to program in C++, learn to work with the OpenCV library and learn to use Microsoft Visual Studio 2010. The eye tracker provided by the blink detection was improved and performed sufficiently to demonstrate the potential of the gaze correction software.

Template capture

Now that the eyes are being tracked and thus their location is known in each frame, the template eyes are captured. These template eyes differ from the eye images used for template matching to track the eyes. The eye tracking template eyes are extracted automatically after the initial eye location establishment. The template eyes that are captured here will be used for the gaze correction in the next step. The user is required to look into the camera and press the 't' key.

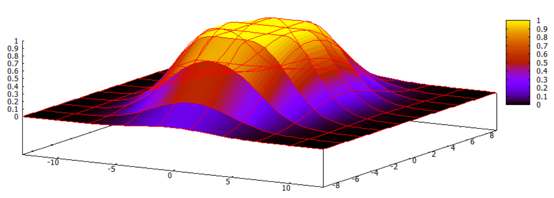
The image of the user looking into the camera is stored as a still frame image, from which the rectangles representing the eyes are extracted. The extracted eyes are stored as templates for the left and right eye and will be used for the gaze correction.

An attempt was made to restrict the gaze correction to only being applied when the user actually looks at the image of the remote user. The approach used was to capture 5 additional templates of the eyes, of the user looking at different locations on the screen. The 5 templates were captured by pressing the keys ’1’, ’2’, ’3’, ’4’ and ’5’ which captured templates of the eyes looking at the left top of the screen, the right top, the centre, the left bottom and the right bottom respectively. These templates were used to estimate which part of the screen the user was looking at. When the calibration process had been completed, the current frame eyes were compared to the 5 template eyes by the means of template matching. If the best match would be with the template eyes looking at the centre, then the gaze would be corrected and otherwise the original frame image would be displayed. The attempt failed due to the inaccuracy of the eye tracker. The current frame eye images differed from the template eye images not only by the eyes being focused on different screen locations, but also because the eye centres did not correspond exactly. One eye image could include the entire eyebrow whereas another could exclude the entire eyebrow, due to the eye tracker returning relatively differing eye centres. However when the eye tracker is improved, this approach could be included to apply gaze correction solely when the user looks at the image of the remote user. Another variation to this approach would be to only use template eyes of the user looking at the displayed image of the remote user. When template matching returns a value above a certain threshold the gaze correction is applied, if it is below the threshold the original unprocessed image is displayed. The threshold should be determined by testing different thresholds and evaluate the performance.

Gaze Correction

The gaze of the user is corrected by Gaussian blending the current frame eyes with the template eyes of the user looking into the camera. The first approach tried was by merely using the template eyes instead of the current frame eyes in the current frame image. This resulted in an unnatural looking image, because there was a sharp transition at the edges of the rectangles defining the eyes. Next a linear blend was attempted. At the centre of the eye rectangles merely pixels from the template eyes were displayed. At the edges of the rectangles merely pixels of the current frame eyes were displayed. In between the centre of the rectangle and the outer edges, a mixture between pixels of the template eyes and the current frame eyes was used. This mixture was defined by giving weights to the template eye pixels and the current frame eye pixels that for corresponding indexes added up to 1. The intensity value of the pixels were multiplied with the weight. When for an index both the template and current frame eye had a weight of 0.5 then the pixel in the resulting image consisted for 50% of the template eye pixel value and 50% of the current frame eye pixel value. The weights depended on the distance from the centre, the further away the smaller the weights of the current frame eye pixels and vice versa for the template eye pixels. Using a linear blend was an improvement to merely using the template eyes, however the result was not satisfactory yet. The linear blend was adjusted to a Gaussian blend.

Enabling eye contact by using pixels of the template eyes is a trade off with conservation of expression in the current frame eyes. In order to maximize conservation of expression in the


(a) Gaussian curve with flatted top

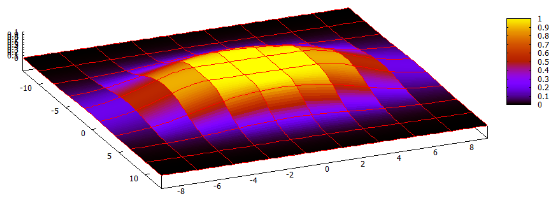

(b) Gaussian curve tilted to visualise shape of the eye

Figure 15: 2D Gaussian curve with flatted top

eyes of the current frame image a Gaussian blend was used. The Gaussian blend uses a greater amount of template eye pixels in the area of the pupil and a greater amount of current frame eye pixels in the outer areas of the eyes with respect to a linear blend. The Gaussian function is discussed in Section 2.6. The template eyes are 2 dimensional and therefore the expression in Equation 2.8 is adjusted to a 2D Gaussian function, which is the expression given in Equation 3.1.

$$f(x,y) = Ae^{-\frac{(\sqrt{x^2+y^2}-\mu)^2}{2\sigma^2}} \tag{3.1}$$

Figure 15 displays the shape of the 2D Gaussian curve with flatted top with the values $A = 1$, $\mu = 3.2$ and $\sigma = 2.5$ which was used as the blending function to blend the template eye pixels with the current frame pixels. The figure is rotated to visualise that the curve has the shape of an eye. Values of the Gaussian curve were stored in a 2D matrix of the same size as the eye rectangles. Each value is a number between 0 and 1 defining the weight of the template eye pixels. The values of the weights of the current frame eye pixels is obtained by subtracting the Gaussian value in the matrix from 1. The flatted top consists of a number of centre eye pixels with the value 1. This ensures that the centre of the eyes consist of pixel values of the template eyes only. This ensures that the perceived gaze direction after processing is to the camera. The values further away from the centre quickly decay to 0, thus the influence of template eye pixels quickly decreases to nullity ensuring maximum possible conservation of expressions in the eyes.

A different attempt was made to preserve expressions of the current frame eyes, which involved detecting skin pixels. Those pixels in the eye rectangles of the current frame image that consisted of skin pixels were not replaced. Those pixels in the eye rectangles of the current frame image that consisted of non-skin pixels were replaced by the template eye pixels. The desired result was that merely the white of the eye ball and the pupils would be replaced, conveying gaze direction while the eye lids were not replaced which would maintain the expressions. The skin detection algorithm discussed by Oliveira & Conci [76] is based on specific values in the Hue plane. The RGB image is converted to the HSV color space where those pixels in the H plane having values ranging between 6 and 38 are labeled as skin pixels. While the algorithm achieves good results in distinguishing between skin and non skin like clothes or objects, the edges separating the skin and non skin areas are not sharp transitions. This resulted in a failure to use skin detection in the eye rectangles in order to determine which pixels were to be replaced, due to an insufficient accuracy of the algorithm. The accurateness of skin detection needed to determine which pixels are to be replaced in the eye rectangles.

3.2 Hardware Solution Teleprompter

The hardware based solution is based on teleprompter technology which is discussed in Section 2.3.1. The idea is to have a teleprompter box suitable for tablets. This section is divided into two parts where the first part discusses the built model, namely a teleprompter box suitable for an Acer Aspire One netbook. The second part is a proposal for adjusting the built model such that it is compatible with tablets. Common used tablets such as the iPad or Android tablet do not have drivers for external cameras. Using an external camera is a requirement for the teleprompter based hardware solution. Therefore the netbook teleprompter model serves to demonstrate the potential of a tablet teleprompter model, provided that tablets will be congruent with external cameras. Due to the similarities of the netbook teleprompter to a tablet teleprompter, like the ease of sliding the device into the box and the size of the display, the netbook teleprompter is sufficient to test the performance of a potential teleprompter based solution for tablets.

The Model

The built model is a black wooden box that enables eye contact when the device used for video telecommunication is placed inside the box. The model is shown in Figure 16 and is based on teleprompter technology discussed in Section 2.3.1. Thus the display is laid down flat and a beam splitter is placed with a 45 ° angle to the display. Behind the beam splitter a camera is placed. In addition to holding the beam splitter and the camera in place, the box functions to keep out light coming from other sources than the display. The box is a teleprompter custom-built to an Acer Aspire One netbook. The material chosen was wood because it is resistant, cheap and easy to build with. The box has been painted black because black does not reflect light, which means that the beam splitter only reflects light coming from the display and not from the box, resulting in a better image of the reflected display. The material chosen for the beam splitter was glass. This is not the optimal material because glass does not produce the best reflection. It is see through and both sides of the glass generate a reflection of the display. This results in a double image of the display with a small offset, which means the image is not perceived as sharp. However the advantage of glass is that it is cheap and reflective and therefore the chosen material. The best material is a one sided mirror, which is see through with respect to the camera and a perfect reflection with respect to the display. An option is to use a regular mirror and to scratch out a little hole at the height of the camera. In this way the camera sees through the mirror and the user looks at the mirror which has a little hole sufficiently small not to be disturbing.

The difference between a netbook and a tablet is that a netbook has a keyboard and a tablet does not. In the model the keyboard is used as the back side of the box. This is a disadvantage because it is inconvenient to use the keyboard unless an external keyboard is used. Using an external keyboard increases the amount of effort which makes the step bigger to use the hardware. Using the keyboard as the backside of the box means that the backside cannot be used to support the camera thus the camera is mounted on a bar that is attached to the sides of the box. The external camera is connected to the netbook by USB. Having the camera at a fixed position means that the remote participant's eyes should always be in the same position on the screen, namely such that the reflection is directly in front of the camera. However the remote participant can have different heights, have their webcam at different distances to the face, have

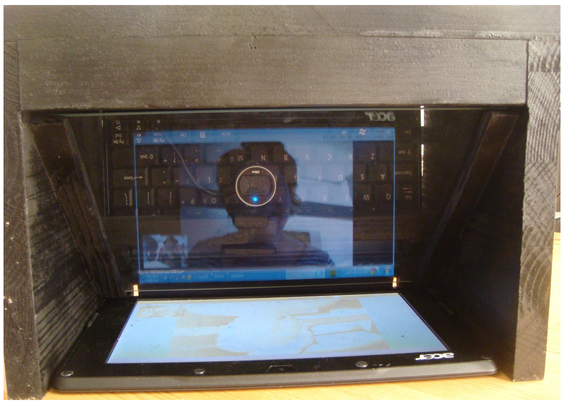


Figure 16: The hardware solution teleprompter model. The display laid flat, its reflection shown in the glass. The camera placed at the location of the remote user's eyes in the image. The black wooden casing keeps out other illuminations, preventing the desired reflection from being disturbed.

different postures, etcetera. These different possibilities cause the participant to appear in different positions on the screen, resulting in different locations of the eyes. A solution would be to have a model such that the position of the camera can be adjusted. In this model a different solution is chosen. Eye tracking is part of the software approach discussed in Sections 3.1 and the technology is discussed in Section 2.5. Using eye tracking to track the eyes of the remote participant, the position of the eyes can be send along with the video stream to the other user. When using full screen display of the remote participant the window cannot be moved around. However a minimized window can be displayed at different screen locations. The window can then be placed in front of the camera such that the eyes of the remote participant appear directly in front of the camera, because the position of the eyes of the remote participant is known and the camera position is fixed. This means that this approach uses additional software in order to guarantee eye contact and is thus not exclusively a hardware solution. However the component responsible for enabling eye contact is the teleprompter box. The software merely corrects for movements or different positioning of the remote participant's eyes.

The teleprompter should be positioned in a fixed place at home, there where the user usually video conferences. Although the box is portable it is likely that a user will not move the hardware around too much and therefore a comfortable fixed place is preferred. The intended use is personal rather than professional thus the location to place the hardware is at home. When engaging in video communication, the user can simply slide the netbook into the teleprompter box and have eye contact with the remote participant. (Note: when only one user possesses the teleprompter box, only that user can look at the other participant's eyes while looking in the camera. The result is that the remote participant perceives the user to be looking straight at him, however the user still sees that the remote participant is looking down or any direction depending on the positioning of the camera, but at least not looking in his eyes. True eye contact by the means of teleprompter boxes is achieved when both participants use such a teleprompter box).

Proposal for Extension to Tablets

Tablets are becoming more and more popular. They are smaller and lighter than laptops which makes them more mobile and their batteries last longer. Tablet owners can take them anywhere and use them any time. Tablets have a bigger screen than mobile phones and these features make tablets suitable for video telecommunication. The teleprompter box is not a necessity to use all the time but positively influences the user experience when using it. Thus when the user happens to be at home and engages in a video chat conversation, the quality of the call can be improved with minor effort. Slide the tablet in the teleprompter box and plug in the external camera which is embedded in the box. The main obstacle to overcome in order to use a teleprompter with tablets is the disability to use external cameras with tablets. This is not in my control and thus this section aims to show the potential of teleprompters for tablets, assuming they have the appropriate drivers.

The design of a teleprompter box for tablets is similar to that of the model as shown in Figure 16. It will be a black box with a beam splitter, such that the tablet can be slided in laid flat from the bottom and eye contact is enabled. Tablets have the advantage that they do not have a keyboard, but the keyboard is displayed on the display. This keyboard can still be used when the tablet is placed in the teleprompter. The typing is done on the actual display and the keys are shown in the mirrored reflection. Not having a keyboard as the backside of the box means that the camera can then be embedded on the backside of the box. The tablet could then simply slide in from the front. A straightforward manner to attach an external camera to a device is by USB. In the proposed model this is the case. However many common tablets do not have USB ports. In this case the tablet can be placed into a docking station which has USB ports. The docking stations for tablets are generally designed to have the tablet standing up vertically. The compound of the docking station and the tablet should be turned horizontally and slided into the teleprompter box such that the docking station closes one of the sides of the box. The box should be closed on all sides except for the front in order to minimize external reflections from entering the box which maximizes the perception of the reflection of the screen. The use of a docking station means that when sliding the compound in from the front, a part of the reflection of the screen is occluded by the docking station. Therefore to be able to view the entire screen the compound should be applied from one of the sides. When applying the compound from the left or right side the viewed display is rotated and this needs to be corrected. Therefore the preferred option is to apply the compound from the back. In this case the display is still mirrored but it is not upside down nor rotated. In order to correct for the mirroring a driver should be included. When sliding the device into the teleprompter it connects to the hardware, then the included driver automatically handles mirroring of the display.

The usage of the teleprompter box with a tablet will be similar to that of the netbook. In fact, the tablet model will be more convenient to use due to the difficulty with keyboard use in the netbook model. The results of the user experience test for this solution are discussed in Chapter 4.

3.3 User Experience Testing

The aim of providing eye contact while video telecommunicating is to improve the sense of presence. The concept of presence is discussed in Section 2.1. Presence is a user sensation and due to this complex nature there is not a standard approach to measure presence. A distinction can be made between subjective and objective measurements. Subjective measurements assess a user's conscious evaluation of his psychological state. Objective measurements assess automatically produced user responses, without conscious deliberation of the user. Van Baren and Ijsselsteijn [77] have performed extended research in presence measurement approaches. Based on their research the Sections 3.3.1 & 3.3.2 briefly explain the subjective and the objective measurement approaches respectively. From this overview the measurement method used in this Thesis is adopted and described Section 3.3.3.

3.3.1 Subjective Measurements

Subjective measurements assess users' conscious evaluation with respect to their sense of presence. The disadvantage is that subjective measures are user biased. The advantage is that the concept to be measured, namely presence, is a user sensation and thus inherently subjective to the user. The majority of the approaches to measure presence are *presence questionnaires*, which fall under subjective measurement methods. Apart from questionnaires there are other subjective measurements such as *continuous assessment*, *qualitative measures* and *subjective corroborative measures*.

Presence Questionnaires

Presence concerns the state of a user, whether a person *feels* emerged and present in a remote location. In the case of video conferencing the sense of feeling present refers to feeling as if the communication takes place face to face rather than by the means of any media. Questionnaires are the most common approach used to measure presence, because they directly address the state of the user. Presence is a subjective sensation and different questions can examine different aspects of presence. Section 2.1 discusses the concept of presence and the means by which to induce presence, namely by using a captivating content, advanced technology or both. Different methods used to improve presence require different questions to test their performance. For example questionnaires researching the performance of using 3D displays or avatars in a virtual environment (VE) setting to increase the sense of presence in video conferencing will contain different questions. Therefore there are a great number of experiment specific presence questionnaires available. An attempt has been made by Witmer & Singer [78] to create a general presence questionnaire, valid across media and content. Their Presence Questionnaire (PQ) is based on addressing factors that influence involvement and immersion, which they established as conditions for presence [78, 77]. The factors are:

- Control factors: The extent of user control and whether the consequences of user actions are logical and as expected, influence the immersion of the user.
- Sensory factors: Coherent stimulation of the senses and controlling the relation of the environment to the senses (such as changing viewpoint, which demonstrates the control of the relation of the environment to vision) stimulates immersion and involvement.

- Distraction factors: Isolation of the user by blocking physical world stimuli, the user's own ability to focus on the virtual environment and the quality of the interface influence the easiness with which a user gets distracted from the virtual world. This affects the immersion and involvement of the user.
- Realism factors: The realism of the scene, its consistency to reality and the meaningfulness to the user influence involvement.

The PQ questionnaire can be found in Appendix B.

Another popular general presence questionnaire is the Slater-Usoh-Steed Questionnaire (SUS) proposed by Slater et al. [79]. The questions used to measure presence are based on three indicators (from Slater et al. [79]):

- The subject's sense of "being there" - a direct attempt to record the overall psychological state with respect to an environment.
- The extent to which, while immersed in the VE, it becomes more "real or present" than everyday reality.
- The "locality", that is the extent to which the VE is thought of as a "place" that was visited rather than just as a set of images.

The SUS questionnaire can be found in Appendix C.

Continuous Assessment

In continuous assessment the user continuously rates the experienced presence, during the session. The advantage is that the ratings are not based on a memory of the presence feeling, but based on how the user is feeling at that exact moment. This means that changes in feeling present can be recorded as well. However, having to do the ratings during the session interrupts the user from the activity and can thus interfere with the sensation of presence.

Qualitative Measures

Qualitative measures do not quantify the user experience. Users do not have to choose an answer that best resembles their experience, but are given the freedom to express the experience in their own way and words. This results in a more accurate understanding of the user experience. However, the approach involves content analysis which is biased by the researcher's interpretation. Therefore the main purpose of qualitative measures is to explore and orientate in the field.

Subjective Corroborative Measures

Subjective corroborative measures do not directly assess presence, but evaluate attributes that *indicate* presence. For example a user might be asked after the session to describe the t-shirt the remote participant was wearing, or to recall which music was playing at the background. Another example is to have the user estimate a time duration, because different levels of engagement result in a different perception of how much time has passed. The difficulty is that certain attributes are assumed to indicate presence whereas in reality they might not.

3.3.2 Objective Measurements

Objective measurements address automatically generated user responses. Objective presence measurements are corroborative measurements, meaning that attributes are measured that *indicate* a sense of presence but do not directly assess presence. Presence is the concept of *feeling* present in a remote location which can only be directly addressed by asking the user the extent to which a state of presence is experienced. In order to do objective measurements, corroborative measurements are required. The advantage of objective measurements is that they are not prone to user bias, which is the case with subjective measurements. Examples of objective corroborative measures used to measure presence are *psychophysiological measures*, *neural correlates*, *behavioural measures* and *task performance measures*. These approaches are briefly discussed as explained by van Baren [77].

Psychophysiological Measures

Psychophysiological measures measure physiological attributes such as heart rate, skin temperature, skin conductance and potential difference of the skin of the face. These physiological responses are among others associated with emotion. In this paper solutions are proposed to enable eye contact in video communication, and it was hypothesized that this will increase the sense of presence. Being able to have eye contact might affect the emotions experienced by the user, but it will be small with respect to the effect of the content of the conversation.

Ocular measures are psychophysiological measures concerning the eyes which makes it inherently more appropriate to measuring user experience of video conferencing with enabled eye contact. Examples of ocular measures are measuring the amplitude of saccades (the series of small movements of the eyes when changing focus), fixation length or pupil response. Pupil responses are entirely uncontrollable by the user, which makes it a strictly objective measure. The idea is that the closer the physiological response of the user to a physiological response in a real world setting, the greater the sense of presence. However these methods are too obtrusive to use with the participants, which might influence the sense of presence.

Neural Correlates

Neural correlates include electroencephalogram (EEG) and functional magnetic resonance imaging (fMRI) techniques that measure the electrical activity and blood flow changes to the brain respectively. They are used to assess brain processes and activity to address presence. However there is little known about the neural processes concerning the sense of presence. This measure approach is therefore not suitable for this research.

Behavioural Measures

Behavioural measures intend to measure naturalistic behaviours such as facial expressions, reflexes or postural responses like a user's response of body sway when watching a clip taken from a moving car. In behavioural methods there is no user bias but there might be observer bias. For example if the number of smiles is used as an indication of presence, it can be hard to tell whether something counts as a smile or not.

Task Performance Measures

In task performance measures the users are given a task and the completion time or error rate is used to indicate presence. In remote learning environments the ability of the user to transfer the learned skill to the real world is used as a performance measurement. The disadvantage of this measurement is that there are other influences on a user's ability to perform a task, such as motivation and tiredness. These are uncontrollable factors.

3.3.3 Adopted Measurement Approach

The approach taken to improve the sense of presence in video telecommunication is enabling eye contact between the users. The subjective nature of the concept of presence indicates the need for a subjective measurement. The ease of use and unobtrusiveness for the user combined with the flexibility of a questionnaire to be experiment specific, have led to the choice of using a presence questionnaire as the subjective measurement. Existing general presence questionnaires such as Witmer & Singer's PQ and Slater et al.'s SUS cannot not be used per se, because many questions address irrelevant aspects. A question such as "*How well could you localize sounds?*" [78] does not give any information about the influence of eye contact on presence. Thus a questionnaire has been assembled by selecting and adjusting relevant questions from the popular presence questionnaires PQ and SUS, combined with experiment specific questions. The advantage of using questions from existing popular questionnaires is that the questionnaires have been developed and tested extensively and therefore consist of well-defined questions. The advantage of developing own questions is that they are experiment specific regarding the topic of this project. The questionnaires used to measure user experience after experiments 1 & 2 can be found in Appendix A. The presence questionnaires PQ and SUS can be found in Appendices B & C respectively.

Data Evaluation

The questions of the user experience survey were designed to measure a sense of presence. A statistical evaluation of the gathered data is necessary to be able to draw conclusions. When the scale used for answering the questions is a linear scale, the underlying data can be assumed to be linear with a normal distribution. Mean, standard deviation and a ttest are then suitable statistics to describe the data and draw conclusions. However the experiments conducted in this work measured a personal perception of to which extent the participant experienced a state of being. The difference between two distances on the scale are not necessarily equal, meaning that the scale is not necessarily linear and thus the underlying data cannot be assumed to be of a normal distribution. Therefore each possibility on the scale is treated as a category on its own. For categorical data the statistic used to determine whether two groups have significantly different opinions is the χ^2 (chi squared) test[80]. Thus the χ^2 test was performed for each survey question comparing two groups, either the group testing the hardware to the group testing the software or a group testing eye contact enabled to a group testing eye contact disabled. The χ^2 test was manually implemented in Matlab and the code can be found in Appendix F.

The mechanics of the χ^2 test are explained conjointly with an example. First the data is represented in a cross-tabulation. The first questions of the second experiment was "How engaged were you in the conversation?". The cross-tabulation is constructed by placing the groups under-

Observed	1	2	3	4	5	total
H	1	1	1	8	14	25
S	0	0	11	11	4	26
total	1	1	12	19	18	51

Table 1: Frequencies of each answer category and the sums of the rows and columns.

Expected	1	2	3	4	5	total
H	0.49	0.49	5.88	9.31	8.82	25
S	0.51	0.51	6.12	9.69	9.18	26
total	1	1	12	19	18	51

Table 2: Expected frequencies of each answer category and the sums of the rows and columns.

neath each other (i.e. the group testing the hardware and the groups testing the software) and calculating the sum of each row and the sum of each column as shown in Table 1.

The gathered data from the experiment is the actual observed data. If the groups did not have different opinions then the data would probably be distributed such that the frequencies are in proportion to the number of participants and the total frequency that the category occurred. Knowing the total number of times each category occurred and the total number of participants in each group, one can calculate the expected frequency per group for each category. This is visualised in Table 2.

Next the differences between the observed and expected values are squared and divided by the expected value, this generates all positive values representing the distance from the expected value. The greater the distances from the expected values the higher the indication that the groups have different opinions. Table 3 shows the differences and the sum of all the differences which is the total χ square value.

The obtained χ^2 value does not mean anything on its own. There are critical values that are related with levels of risk. This means that when the χ^2 value is higher than a certain critical value, the level of risk is the probability that the opinions of the group are the same. The confidence that the opinions of the groups differ is then 1- the level of risk. The critical values differ for different degrees of freedom. For the χ^2 test the degrees of freedom are calculated by taking the number of rows - 1 multiplied with the number of columns -1. In the above example there are 2 rows and 5 columns (because 2 groups are compared for 5 categories), leading to 4 degrees

Difference	1	2	3	4	5	total
H	0.53	0.53	4.05	0.19	3.04	8.34
S	0.51	0.51	3.90	0.18	2.92	8.02
total						16.36

Table 3: Differences between observed and expected values. The total sum of differences defines the χ^2 value.

	confidence				
Degrees of freedom	90%	95%	97.5%	99%	99.9%
1	2.706	3.841	5.024	6.635	10.828
2	4.605	5.991	7.378	9.210	13.816
4	7.779	9.488	11.143	13.277	18.467

Table 4: Confidence levels corresponding to critical values for 1, 2 and 4 degrees of freedom.

of freedom. The gathered data of the experiments conducted in this work are evaluated by comparing 2 groups for 3 categories (experiment 1) and 5 categories (experiment 2). Table 4 shows the levels of risk related to the critical values of 2 and 4 degrees of freedom, which was taken from the engineering statistics handbook provided by NIST (National Institute of Standards and Technology) [81].

The discussed example, based on the second experiment gathered data for the hardware and software solution, generated a χ^2 value of 16.36. According to Table 4 for 4 degrees of freedom this means that there is a 99% confidence that the data is significantly different. Thus we can be 99% confident that the opinion of a participant about engagement in conversation depends on whether the participant tested the hardware solution or the software solution.

4 Results

There were two experiments conducted to test the proposed hardware and software solutions. The first round of experiments aimed at discovering non trivial weak points that needed improvement, which were revealed during the testing. The experiment was designed to make a technical comparison between the hardware and software solution. Moreover both solutions were tested when eye contact was enabled and disabled.

The second round of experiments was designed to compare the hardware solution to the software solution, with respect to the sense of presence experienced by the participants. The participants were gathered on a voluntary basis for both experiments, however they were given a chocolate treat as an incentive for participation. The webcams used in both experiments for both solutions were Microsoft LifeCam HD cameras. In Section 4.1 the setup and results of the first experiment are discussed. In Section 4.2 the setup and results of the second experiment are discussed. In Section 4.3 an overall conclusion of the experiments is drawn.

4.1 Experiment 1

This experiment involved 11 participants testing the hardware and 11 participants testing the software solution. From the participants 45% were male and 55% were female. All participants were in their twenties. The video conferencing experience of the participants was partitioned into 64% of frequent users (i.e. using a video chat tool at least once a week) and 36% of regular users (i.e. using a video chat tool at least once a month). From the volunteers testing the software 64% had uncorrected eye sight, 27% percent wore glasses and 9% wore contact lenses. From volunteers testing the hardware 64% had uncorrected eye sight, 28% wore glasses and 28% wore contact lenses.

4.1.1 Experimental setup

The users were asked to look at a webcam captured image of themselves where eye gaze correction had been applied. They were given a topic and asked to talk to their own image about the topic. Then the eye correction was disabled and they were asked to talk to their own image about another given topic. The participants were told they could speak in their native language, as the specific content of the conversation was not relevant. The topic pairs proposed were:

1. a) What would you say to your future self, if you were talking to yourself in 5 years time?
   b) What would you say to your past self, if you were talking to yourself in your final year of school?

2. a) You have won the lottery, tell yourself what you will do with the money..
   b) You have only 24 hours to live, tell yourself how to spend your time..

In the software solution the eyes were captured by blinking after which the eyes were tracked. A template was captured by pressing the 't' key while looking into the camera. Enabled eye contact was established by turning on the gaze correction functionality. Disabled eye contact was established by turning off the gaze correction functionality. Turning the gaze correction on and off was controlled by keys. Rather than capturing and displaying the possibly processed images directly, a networking application developed by Jayson Mackie research assistant for the Game Technology Group at Gjøvik University College was used which sent the images to the local host. The images recveived at the localhost are grabbed and displayed. The networking application was used for two reasons. Firstly it approximates the video conferencing structure better, because in video chat sessions the images are sent over a network. Secondly, the slight delay caused by passing the images by the localhost rather than immediately displaying them, increased the tracking performance of the system. While the delay improves tracking performance, it is so little that the program still runs in real time.

Enabled eye contact in the hardware solution was established by using the webcam behind the glass within the wooden box. For a description of the hardware solution see Section 3.2. When the user is looking at the screen the captured image shows the user looking straight into the recipient's eyes. The recipient in the experiment being the participants themselves. Disabled eye contact was established by using a webcam placed on top of the box, which resembles a regular setup where the webcam is integrated or placed on top of the screen. When the user is looking at the screen the captured image shows the user looking down. Skype was used as a tool to display the image, which allowed for full screen display and smooth switching of the cameras without the need to end the chat session.

After the participants talked to their own image with eye contact enabled and disabled, they were asked to fill out a paper based survey which can be found in Appendix A. The survey asked about general user information such as gender and eye sight information. The survey contained two technical questions about the solutions and four presence questions where the latter were to be answered twice, namely with respect to the eye contact enabled and eye contact disabled. The survey was concluded with the question whether the participant felt the eye contact enabled was an improvement and an opportunity for comments was given.

The following questions were used to address technical aspects:

1. How would you rate the ease of use of the teleprompter/program?
2. How would you rate the image quality?

The following questions were used to address the sense of presence:

3. When you were talking to yourself, how close was it to talking in a mirror?
4. How engaged were you in the conversation with yourself?
5. Did you feel like you had eye contact with yourself?
6. How natural did the conversation seem, with respect to a real life conversation?

The answers were rated by participants between 1 indicating "not at all" and 7 indicating "very much".

		χ	h0	confidence
Technical	Ease of use	1.053	0	
	Image quality	1.333	0	
Software Solution	Like mirror	1,167	0	
	Engagement	6,667	1	95%
	Eye contact	7,200	1	95%
	Naturalness	1,577	0	
Hardware Solution	Like mirror	9,740	1	99%
	Engagement	1,952	0	
	Eye contact	14,864	1	99.9%
	Naturalness	3,543	0	

Table 5: Statistical Results Experiment 1. A rejection of the null hypothesis is shown by a 1. Then the corresponding confidence is given. A failure of the rejection of the null hypothesis is shown by a 0.

4.1.2 Results

Participants rated the answers to the questions on a scale from 1 to 7, based on their perceptions. The scale cannot be assumed to be a linear scale, as the difference between 4 and 5 (being neutral to a question and agreeing to a question) might deviate from the difference between 6 and 7 (indicating particular strengths of agreement). Each answer is then seen as a discrete answer. Therefore a χ^2 (chi squared) test is performed on the data, which is explained in Section 3.3.3. When 7 categories are used (each possible answer) in a comparison between two entities (such as comparing hardware to software) there are 6 degrees of freedom. There were 11 participants in the experiment which generated 11 answers per question, then 6 degrees of freedom is a lot. There is no need to find for each of the possible answers in specific whether it is correlated with hardware versus software. Thus the answers were grouped together in order to find appropriate results to draw relevant conclusions. Answers 1,2 and 3 were grouped together as negative (disagreement), 4 was kept in one group as neutral and 5,6 and 7 were grouped together as positive (agreement). The results of the technical comparison between the hardware and the software solutions are shown as bar graphs in Figure 17. The results of the hardware solution comparing enabled eye contact to disabled eye contact are shown as bar graphs in Figure 19. The results of the software solution comparing enabled eye contact to disabled eye contact are shown as bar graphs in Figure 18. The results of the χ^2 tests are shown in Table 5. For the technical questions the χ^2 test was performed between the answers of the software and the hardware testing. For the presence questions the χ^2 test was performed within the answers of the software and of the hardware testing, between the enabled and disabled eye contact. The original gathered data is given in Appendix D. The first round of experiments also aimed at discovering weak points that needed improvement. The second experiment was designed based upon the outcome of the results of the first experiment and included improvements. A discussion of the weak points and incorporated improvements concludes this section.

Technical Results

The technical results comparing the hardware and software solutions are shown in Figure 17. It was expected that the ease of use of the software would be rated lower due to the required user interaction of blinking and pressing keys. Whereas the teleprompter did not require any user interaction because the participants were not asked to handle the mirrored mouse but it was done for them. It was also expected that the image quality of the teleprompter would be rated lower due to the somewhat blurriness of the image caused by the double reflection of the glass. However the bar graphs show positive results for both solutions without an apparent difference neither in ease of use nor image quality. Referring to Table 5, it cannot be concluded that there is a statistical significant difference in the measured technical aspects when using the hardware and the software solutions. This means that if results of the experiments lead to favouring one solution over another it cannot be concluded that the difference is due to a technical difference.

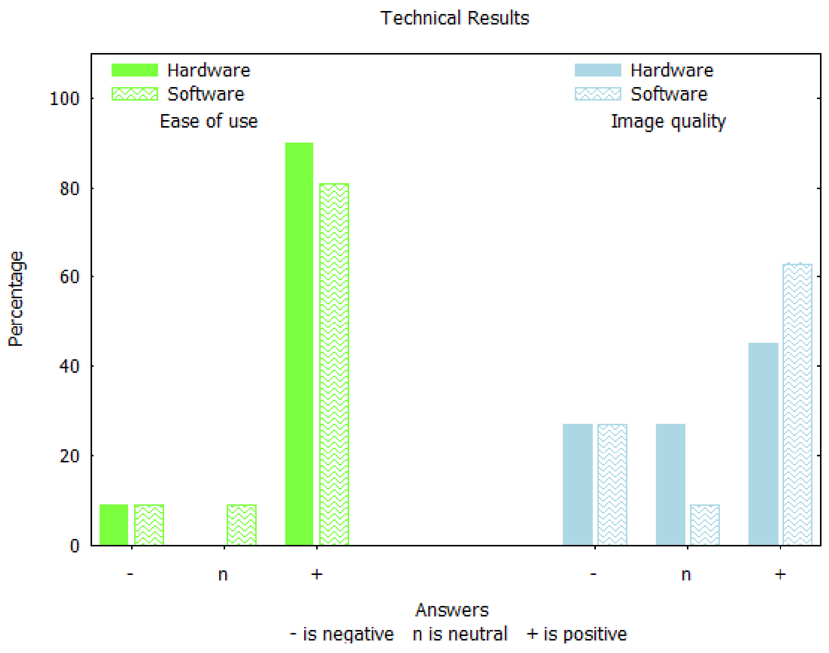


Figure 17: Technical results, testing the ease of use and image quality of the proposed hardware and software solutions.

Software Results

The results regarding the sense of presence when using the software program, comparing enabled and disabled eye correction, are shown in Figure 18. The most evident results concern engagement in conversation and eye contact perception. Engagement in conversation shows a peak at the positive end of the scale in the situation where eye contact was enabled by gaze correction. In the conventional setup where eye contact was not enabled, the engagement in the conversation was more neutral. Thus it can be concluded that applying gaze correction induces a greater engagement in the conversation (confidence level 95%), which indicates a greater sense of presence.

Eye contact perception shows a peak at the positive end of the rating scale in the enabled eye correction situation, whereas disabled eye correction shows a peak at the negative end of the rating scale (confidence of 95%). An additional question in the survey was asked, which examined whether the eye contact was an improvement or not. The question was answered with yes by 7 out of 11 participants and with no by the other 4. Different opinions could be due to different behaviours of the participants. There were participants that passively observed the effect of the gaze correction while talking to their own image. Other participants were actively testing the eye correction software by rotating and moving their head in and out. Comments of the participants included the word “funny” twice. Once the word “freaky” was used and ‘sometimes not natural’ was mentioned. Out of the 7 people answering that the correction was an improvement, the answer was followed 3 times by a ‘but’, commenting that it could or should be improved. Positive comments were also given such as “feels more natural” and “you can talk as you like, no need to force yourself to look at the camera”. One frequent video conferencing user (using video chat more than twice a week but not daily) commented that he likes to perform other tasks while video chatting. The participant said that the software is then useful because “you appear less distracted”.

The graphs corresponding to the alikeness to a mirror and the naturalness of the conversation show similar distributions of ratings suggesting no statistical difference between the enabled and disabled state. Indeed no valid conclusions could be drawn about the influence of enabling and disabling gaze correction on the alikeness to talking to a mirror and the naturalness of the conversation.

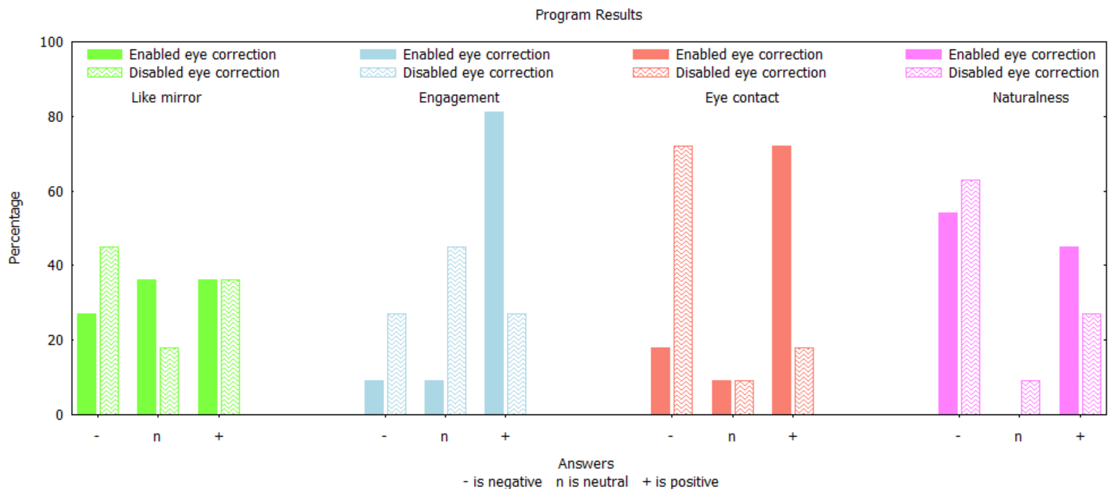


Figure 18: Software results, measuring the sense of presence using the software program. Comparing enabled and disabled eye correction.

Hardware Results

The results regarding the sense of presence when using the teleprompter, comparing the enabled and disabled eye contact are shown in Figure 19. The bar graphs corresponding to the questions concerning alikeness to a mirror, perception of eye contact and the naturalness of the conversation indicate a greater sense of presence when eye contact was enabled. Table 5 shows that in the cases of alikeness to a mirror and the perception of eye contact, the indication is strongly supported (with a confidence of greater than 99%). However no conclusion could be drawn in the case of the naturalness of the conversation.

The ratings regarding the engagement in the conversation are equally positive and negative for the enabled eye contact situation. For the disabled eye contact situation the engagement is rated more negative. The overall lack of engagement in either situation can be explained by the awkwardness of talking to oneself. Referring to Table 5, no conclusions could be drawn about whether enabling or disabling eye contact has a positive or negative effect on the engagement in the conversation.

The results show that the perceived eye contact was greater with enabled eye contact. The question asking whether the eye contact was an improvement or not, was answered with yes by 10 out of 11 participants 1 participant answered with no. The participant answering that the eye contact was not an improvement was a frequent user (using video chat more than twice a week but not daily) of video conferencing systems. The participant had trained himself to look in the camera when speaking, such that the remote partner would perceive his eye contact. The camera being positioned behind the screen meant that the participant looked at the image of the remote partner (or in the case of the experiment the image of the participant himself) which differed from what the participant had grown accustomed to. The participant commented: “No, because I felt a little bit uncomfortable when you’re looking straight in your eyes. But maybe you just have to get used to it and then it’s definitely an improvement.”

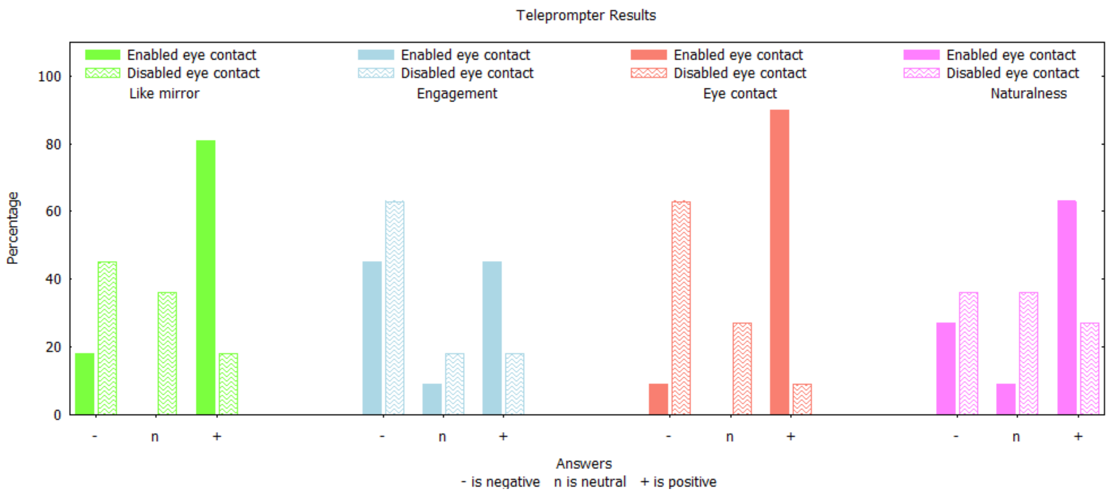


Figure 19: Hardware results, measuring the sense of presence using the teleprompter. Comparing enabled and disabled eye contact.

Improvements for Experiment 2

User experience comments included several times that it was awkward to talk to themselves, despite being able to speak in their native language. Both the software and hardware solution testing required users to talk to themselves. Therefore the induced awkwardness was consistent throughout this experiment. However the follow up experiment was designed such that the participants had a video chat conversation with another user rather than with themselves.

In testing the hardware solution Skype failed once. However this was a problem with the device used and after a restart it functioned well again. There were no other technical difficulties with the hardware solution and the experience showed that Skype was a good tool to use for the testing. The next experiment used Skype again to test the hardware.

In testing the software solution the main technical difficulty that was brought to light was that the eye capturing depended on blink detection. Blink detection is performed by motion analysis, using a difference of frames. For a detailed explanation see Section 3.1.2. When other parts in the image are moving as well, such as the lips or the camera itself because the table on which the device with the camera is positioned is moving, the blink detection fails. The frame difference shows not only the two connected components which are the eyes but also other motion and thus it fails to detect the eyes. In order to better this weak point an adjustment was made to the software. In the next round of experiments the eyes could be detected either by blinking or by clicking on the eyes.

The eye tracker in the software approach is the weakest point and needs improvement. However due to eye tracking being an entire subject in itself, the choice was made to settle for a working but not perfect eye tracker. The follow up experiment was designed such that it measured the sense of presence of participants who perceived eye contact with an opposite party. Applying the gaze correction one directional on the remote user's image was sufficient to measure participants' sense of presence. This gave the advantage that participants were not able to influence the gaze correction software performance. Thus an actively testing of the software that would lead to 'funny looks' was avoided.

The questionnaire of the first experiment was given out on paper, with 7 scale answers to the questions. The next experiment used the online survey tool LimeSurvey, which had a number of advantages over paper. The participants could be sent a link such that they were able to fill out the questionnaire at home in their own time and it could be submitted straight after they were done. Moreover LimeSurvey generated data files and statistical information automatically, as opposed to doing this manually when the paper questionnaires were used. The 7 scale answers were adjusted to 5 scale answers. The reason was that 5 scale answers better correspond to lexical representations for each answer category. For example a scale from 1 to 5 means "strongly disagree", "disagree", "neutral", "agree" and "strongly agree" or "very bad", "bad", "neutral", "good" and "very good".

4.2 Experiment 2

This experiment involved 26 participants that tested the software solution, of which 15 males and 11 females. The ages ranged from 19 to 61, however most participants were under 30 and the median was 23. About half had an uncorrected eye sight, about 20% used contact lenses and about 35% wore glasses. The video conferencing experience of the participants was partitioned into 42% of frequent users (i.e. using a video chat tool at least once a week), 23% of regular users (i.e. using a video chat tool at least once a month) and 31% of occasional video conferencing users (i.e. a few times a year) and 4% had never used a video chat tool.

There were 25 participants that tested the hardware solution, of which 16 males and 9 females. The ages ranged from 19 to 44, however most participants were under 30 and the median was 25. More than half had an uncorrected eye sight, but there were 2 participants using contact lenses and 8 wearing glasses. The video conferencing experience of the participants was partitioned into 68% of frequent users (i.e. using a video chat tool at least once a week), 16% of regular users (i.e. using a video chat tool at least once a month) and 16% of occasional video conferencing users (i.e. a few times a year).

4.2.1 Experimental setup

The setup for testing both the software and hardware solution was similar. The participants were asked to conduct a video chat with a member of the research team, namely the author. The experimenter used either the software or the hardware solution, such that the participants perceived eye contact. Although true eye contact can only be established when it is mutual, the eye correction was not applied to the image of the participants. The reason for this was that the participants would have to be trained in using the solutions. For the software solution this meant learning how the program captures the eyes, captures a template and how it sends the processed images. For the teleprompter this meant learning to operate mouse movements when the screen is mirrored. The first round of experiments indicated that the ease of use of both solutions is quite acceptable, however the participants were not required to handle the mouse in the mirrored display and were assisted in the steps to correct the eye gaze using the software. The focus was on ease of use after the initial setup, not on the ease of use of the setup itself. In order to avoid training the participants to use the solutions (which is time demanding and would thus expand the duration of a session, requiring volunteers to spare more of their time) and to avoid building a second teleprompter box, the gaze correction was only applied to one end of the video conferencing users. The participants perceived eye contact and the experimenter did not. However results are based on the participants' experiences rather than the experimenter's experience of the video chat session. The participants' experiences are independent of whether the experimenter perceives their eye contact, because this is not visible for the participants. Therefore correcting gaze merely one directional does not pose any issues in measuring user experience of the solutions.

The software was tested by using the networking application developed by Jayson Mackie research assistant for the Game Technology Group at Gjøvik University College. The application sends and receives images to and from a computer with a given IP address. On the experimenter's end the gaze correction processing was included before sending the webcam captured images. On the participant's end the gaze correction processing was not included. The experiments were carried out within one room, with a barrier separating the participant and the experimenter. This setup eliminated the need to include sound transmission in the current form of the software.

The hardware was tested by establishing a Skype video call with the participant. The experimenter called from the teleprompter box whereas the participants called from either their own device or a computer that was made available for the user testing. Using Skype enabled people to join the experiment from their own device, from any location in the world (provided that they had internet access and a webcam).

After the video chat conversation the participants filled out an online survey, developed and distributed by the online tool LimeSurvey. The questions included general questions such as age and video conferencing experience and nine questions measuring their sense of presence during the video chat. The presence questions were partly extracted and adjusted from commonly used questionnaires such as the *Presence Questionnaire* [78] and the *Slater-Usoh-Steed Questionnaire* [79]. These can be found in Appendices B & C respectively. The presence questions were divided into three main categories, namely *immersion, naturalness and eye contact*. Each category contained three questions of which two were positive (i.e. the higher the rank the greater the sense of presence) and one was negative (i.e. the higher the rank the lower the sense of presence). This was done to balance out the influence of the answers of participants that do not really read the questions but answer randomly, for example with the intend to please the experimenter, if there were to be any.

The following questions were used to address immersion:

- How engaged were you in the conversation?
- Did you get distracted from the conversation?
- How much were you involved by the visual aspects (the video part) of the video conferencing session?

The following questions were used to address naturalness:

- How natural did the conversation seem?
- To what extend did you feel as if the video conferencing session was as a real world conversation?
- Were you conscious about the media used during the video conferencing session?

The following questions were used to address eye contact:

- Did you feel like you had eye contact with the other video chat participant??
- How natural did the eye contact with the other participant seem?
- To what extend did you feel as if the eye contact with the other participant distracted you from the conversation?

The answers were rated by participants between 1 indicating “not at all” and 5 indicating “very much”.

4.2.2 Results

The software and hardware approach were compared with respect to the sense of presence experienced by the participants. The results per category into which the questions measuring presence were divided are shown in Figures 20, 21 & 22 which show immersion, naturalness and eye contact respectively. For each question a χ^2 (chi squared) test was performed between the set of answers related to the software solution and the set of answers related to the hardware solution. The χ^2 test is discussed in Section 3.3.3 and the results of the χ^2 tests are shown in Table 6.

		χ	h0	confidence
Immersion	Engagement	16,349	1	99%
	Distraction	10,281	1	95%
	Involvement	1,431	0	
Naturalness	Naturalness	12,038	1	97.5%
	Like real	5,460	0	
	Conscious media	5,075	0	
Eye Contact	Perception	7,700	0	
	Naturalness	9,313	1	90%
	Distraction	10,593	1	95%

Table 6: Statistical Results Experiment 2. A rejection of the null hypothesis is shown by a 1. Then the corresponding confidence is given. A failure of the rejection of the null hypothesis is shown by a 0.

Results Immersion

The bar graphs that demonstrate the distributions of given ratings by participants in percentages for the questions addressing immersion are shown in Figure 20. The higher the rankings on the questions regarding engagement in the conversation and video involvement the greater the presence. Conversely, the lower the ranking on the question regarding the distraction from the conversation greater the presence. The bar graphs show that participants testing the hardware solution were more engaged in (confidence 99%) and less distracted (confidence 95%) from the conversation than participants testing the software solution. The bar graph corresponding to the involvement by visual aspects shows a similar distribution for both solutions and no conclusion could be drawn.

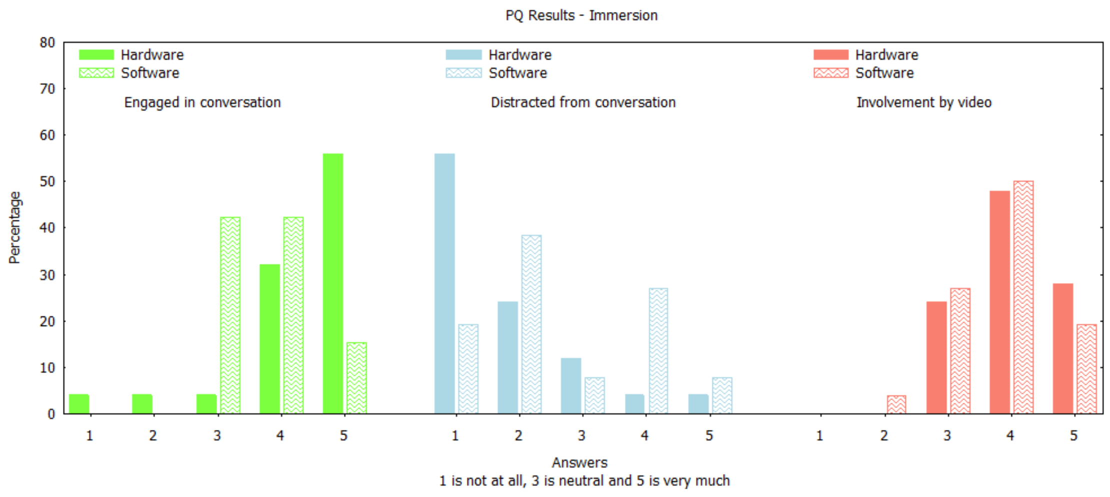


Figure 20: Presence results, comparing the sense of immersion when using the hardware and software solutions.

Results Naturalness

The bar graphs that demonstrate the distributions of given ratings by participants in percentages for the questions addressing naturalness are shown in Figure 21. The higher the rankings on the questions regarding the naturalness of the conversation and the alikeness to a real world conversation the greater the presence. Conversely, the lower the ranking on the question regarding the consciousness of the media used the greater the presence. The bar graphs show a greater naturalness of conversation when using the hardware (confidence level 97.5%). The bar graphs concerning consciousness of media and alikeness to a real world conversation show similar distributions for both solutions. The software solution results seem slightly more negative with respect to alikeness to a real world conversation and slightly more consenting to being conscious about the media used than the hardware solution results, but no conclusion could be drawn.

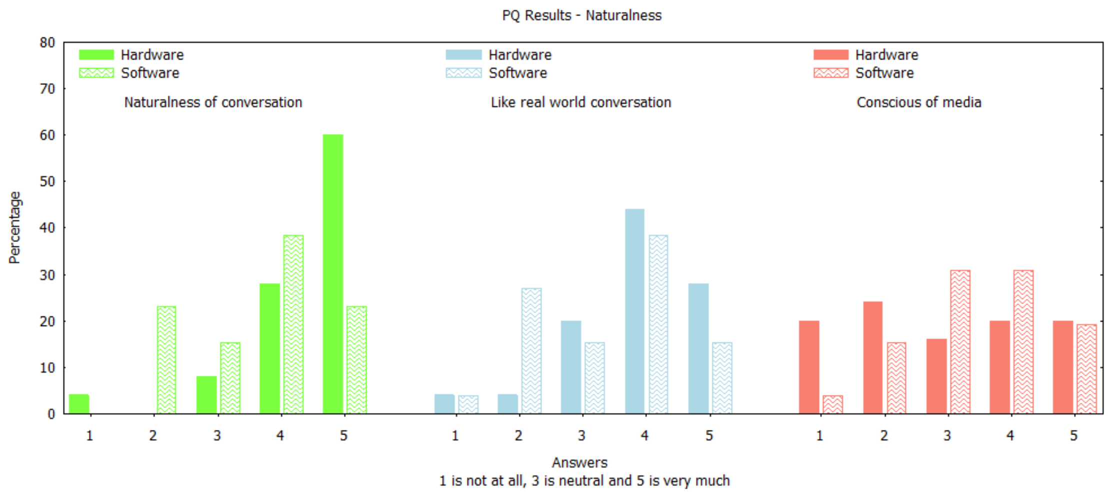


Figure 21: Presence results, comparing the sense of naturalness when using the hardware and software solutions.

Results Eye Contact

The bar graphs that demonstrate the distributions of given ratings by participants in percentages for the questions addressing eye contact are shown in Figure 22. The higher the rankings on the questions regarding eye contact perception and naturalness the greater the presence. Conversely, the lower the ranking on the question regarding the distraction by the eye contact the greater the presence. The graph shows that eye contact perception in general was good for both solutions. The peak in the graph indicates a greater eye contact perception when using the hardware solution. However Table 6 does not support this indication. This is because the χ^2 test was performed taking all 5 bins into account whereas both solutions scored high with respect to eye contact perception. Therefore another χ^2 test was performed on this specific question, distinguishing a good perception (i.e. rated by number 4) from a very good perception (i.e. rated by number 5). Results show that the hardware solution induced a greater eye contact perception than the software solution (χ value 4,375 and the hypothesis rejected with a confidence level of

95%). Furthermore a greater naturalness of the eye contact corresponds to the hardware testing, but only with a confidence level of 90%. Finally the induced eye contact is less distracting in the hardware case (confidence 95%).

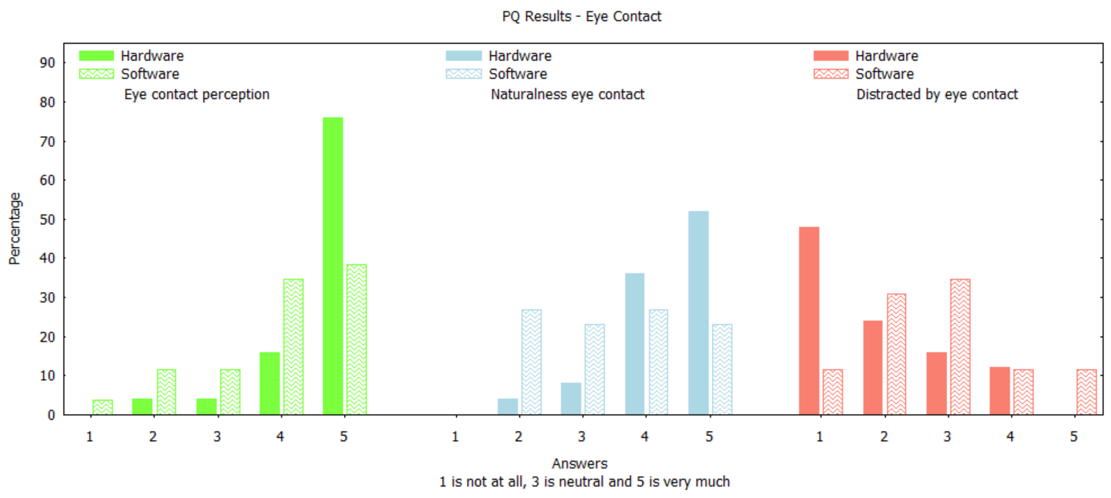


Figure 22: Presence results, comparing the sense of eye contact when using the hardware and software solutions.

4.3 Experiments Conclusion

The first experiment tested the technical aspects 'ease of use' and 'image quality' of the solutions. It was expected that the ease of use would produce a more negative user experience for the software solution, due to the program not being fully automatic. It was also expected that the image quality would be rated lower for the hardware solution due to the somewhat blurriness of the image. However no conclusions could be drawn about the technical differences. The results being mainly on the positive side of the scale, indicates that the user interaction in the program and the image blurriness of the teleprompter are acceptable. The ease of use of the systems do not pose an issue in the second experiment because the participants are not using the systems but are merely perceiving their effects.

The first experiment also measured the sense of presence when eye contact was enabled and disabled, both for the software and hardware solution. From the questions that were based on commonly used presence questionnaires, namely inquiring about engagement in the conversation and naturalness of the conversation, no conclusions could be drawn considering the hardware solution. For the software solution, the question inquiring about the engagement in the conversation demonstrated a greater engagement in the gaze corrected state. The lack of more valid results with respect to presence questions could be due to the awkwardness of talking to oneself, which was mentioned four times in the comments of the participants. It could also be that frequent users of video conferencing systems have grown accustomed to the offset of the remote user's gaze. Looking down could be automatically interpreted as eye contact when engaged in a video conferencing session. Then enabling eye contact does not influence the sense of presence.

The experiment would have to be conducted with participants talking to another person rather than to themselves, in order to draw valid conclusions about the difference in sense of presence in the conventional setup compared to an eye contact enabled situation. The experiment specific question inquiring about eye contact perception showed that both solutions succeeded in inducing a perception of eye contact. This does not necessarily imply that the perceived eye contact is an improvement. Therefore the questionnaire included the question whether the participants found the eye contact an improvement. For the software solution 63.6% and for the hardware solution 90.9% of the participants answered with yes.

The second experiment was designed to compare the performance of the solutions, measuring the sense of presence. Three categories were used to address presence while engaging in a video conferencing session. Each category contained three questions of which two were positive and one was negative, which is explained in Section 4.2.1. The results of the questions addressing immersion generated two valid results, favouring the hardware solution over the software solution. The results of the questions addressing naturalness generated one valid result favouring the hardware solution over the software solution. Finally the results of the questions addressing eye contact generated three valid results, all favouring the hardware solution over the software solution. Hence, in each category there was at least one question showing a preference of the teleprompter over the software solution. The reverse did not occur once. Neither was there an indication of the software solution being favoured over the teleprompter visible in the bar graphs. Thus from the proposed solutions as they were when tested, the hardware approach performs better than the software approach. A greater sense of presence is induced when using the hardware solution to enable eye contact than when using the software solution to enable eye contact while video conferencing.

5 Conclusion

In leisure video conferencing the camera is conventionally positioned with an offset to the user's gaze. This offset impedes eye contact perception in distance video communication. In video conferencing a user's sense of presence is the extent to which the user feels as if they are truly together with the remote user, as opposed to being aware of the physical separation. Eye contact plays a substantial role in communication. Thus allowing for eye contact in video conferencing is believed to induce a greater sense of presence. Existing systems that incorporate eye contact aim at professional use. These systems require a large expenditure and are therefore not suitable for the consumer market. Low cost solutions were developed, meeting adequate standards for leisure video conferencing. A software solution based on image processing techniques was proposed along with a hardware solution based on teleprompter technology.

The first research question to be answered was:

1. Does enabling eye contact in video conferencing improve user experience with respect to the sense of presence?

The results showed that both the proposed solutions succeeded in the goal to enable eye contact perception. The survey questions regarding the sense of presence showed that for the software solution employing gaze correction increased engagement in the conversation, but for the hardware solution no conclusions could be drawn. There was a lack of further results correlating enabled eye contact with an increasing sense of presence. A possible explanation is that video conferencing users have grown accustomed to the gaze offset of the remote user. In this case, adjusting the offset to better resemble reality would not improve the sense of presence. Another possibility is that the awkwardness of talking to oneself influenced the user experience.

The second research question to be answered was:

2. Does the software or the hardware solution induce the greatest sense of presence, designating this to be the preferred solution?

The results showed that the teleprompter induced a greater sense of presence than the software application, revealing the hardware solution as the preferred solution.

Based on the results and the positive feedback from the participants it can be stated that both the teleprompter and the gaze correction software show great potential. There is no need for employing sophisticated professional systems to enable eye contact in leisure video conferencing. However for the proposed solutions to be suitable as commercial products the suggested improvements and adjustment discussed as future work in Section 6 should be applied.

6 Future Work

This work proposed a software and a hardware solution for enabling eye contact in leisure video conferencing. Notwithstanding their great potential, both systems would benefit from improvements, adjustments and extensions. Suggestions are discussed regarding the hardware solution followed by suggestions regarding the software solution.

Future Work Hardware Solution

The teleprompter emerged as the preferred solution to enable eye contact, nevertheless the solution can be improved. A weak point is the blurriness of the captured image, due to the double reflection in the glass. Replacing the glass as the beam-splitter by a thinner reflective material (such as the patented Musion Eyeliner foil [21]) will improve image quality. Alternatively an opaque silvered mirror could be used, where a small space is left out or made transparent for camera capture.

The greatest disadvantage of the prototype in the current state is the difficulty of device usage while placed in the teleprompter. One of the sides of the box is composed by the keyboard of the netbook. Future adaptations of the prototype should be custom made for tablets, as proposed in Section 3.2, which would eliminate the issue of not being able to reach the keyboard because tablets have onscreen keyboards. Additionally a driver should be provided that auto corrects for the mirrored display in order to avoid counter intuitive responses to mouse movements.

Another drawback of the teleprompter box in its present form is that it is rather cumbersome. This could be overcome by using different material such as plastic instead of wood and adjusting the design to a model that folds for portability. In its folded state it could serve as a hard case cover for the tablet, in its unfolded state it would serve as a teleprompter.

Future Work Software Solution

The software program emerges as the less preferred solution to enable eye contact but nevertheless showed potential. The main weak point in the software is the eye tracker. The eyes are tracked by template matching, the pixel coordinates with the best match are used as the centre coordinates of the eyes. The method is able to track and localize the eyes but returns approximated coordinates. This imprecision causes the overlay of the template eyes to be slightly off at times, which creates a ‘funny’ or ‘freaky’ looking image. Therefore the highest priority in improving the gaze correction software should be improving the eye tracker. In Section 2.5 several eye tracking techniques are discussed. The solution ought to be applicable in leisure video conferencing which excludes eye tracking techniques that are obtrusive or use specialised external hardware. Including multiple cameras would provide more information on the scene, which would prompt a more educated approximation of the eye centre coordinates. External cameras are becoming cheaper and it is feasible to presume that in the future many devices will have multiple integrated cameras (for example the LG Optimus 3D smartphone, Microsoft Kinect and Fujitsu LifeBook AH572 laptop). However a single camera based solution would maintain the widest usability.

Future work should also include gaze localization. The eye correction should only be applied when the user looks at the image of the other user. An attempt was made to include this functionality in the current software program. The approach taken was to capture templates of the eyes of the user looking at different locations on the screen, including the centre where the image of the remote user is displayed. The eye correction should only be applied when the template match between the tracked eyes in the current frame and the template of the eyes of the user looking at the centre is greater than the template matches with the other captured template eyes. The attempt failed because the differences in the template matches were not sufficiently large. This was due to the eye template images differing from each other not just because of the user looking at different display locations, as desired, but because of the imprecision of the eye tracker. An improvement of the eye tracker could be associated with an improved performance of this attempted approach. Perhaps only an eye image template of the user looking at the centre is sufficient. If the template match with the current frame eye image is above a threshold then the eye correction functionality is allowed, otherwise it is denied.

Finally a feature that is not currently included is scaling. The captured template eyes used for the overlay are of a set size. The template eyes should be scaled up or down when the user moves in and out respectively, in order to maintain the correct ratio of the size of the eyes with respect to the face. The scaling feature can be based on the distance between the eyes. When the eyes are first captured the distance extracted is used as the ground truth. While the eyes are being tracked in the subsequent video frames, their distances are also extracted. The ratio of the ground truth distance and the current distance can be used as the scaling factor with which to multiply the template eyes before overlaying on the current image frame.

Bibliography

[1] van Eijk, R., Kuijsters, A., Dijkstra, K., & IJsselsteijn, W. JUN 2010. Human sensitivity to eye contact in 2d and 3d videoconferencing. In *Quality of Multimedia Experience (QoMEX), 2010 Second International Workshop on*, 76 –81.

[2] Grayson, D. & Monk, A. 2003. Are you looking at me? Eye contact and desktop video conferencing. *ACM Transactions on Computer-Human Interaction (TOCHI)*, 10(3), 221–243.

[3] Jerald, J. & Daily, M. 2002. Eye gaze correction for videoconferencing. In *ETRA*, 77–81.

[4] Criminisi, A., Shotton, J., Blake, A., Rother, C., & Torr, P. H. S. Efficient dense-stereo and novel-view synthesis for gaze manipulation in one-to-one teleconferencing. Technical report, 2003.

[5] Kleinke, C. L. 1986. Gaze and eye contact: A research review. *Psychological Bulletin*, 100(1), 78–100.

[6] Gemmell, J. & Zhu, D. 2002. Implementing gaze-corrected videoconferencing. In *In Proceedings of CIIT 2002*.

[7] Dumont, M., Rogmans, S., Maesen, S., & Bekaert, P. 2009. Optimized two-party video chat with restored eye contact using graphics hardware. *e-Business and Telecommunications*, 358–372.

[8] Yang, R. & Zhang, Z. JUL 2004. Eye gaze correction with stereovision for video-teleconferencing. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(7), 956–960.

[9] 2011. Fraunhofer Heinrich-Hertz Institute: Virtual Eye Contact Engine.
http://www.hhi.fraunhofer.de/en/departments/image-processing/applications/virtual-eye-contact-engine/. (accessed nov 2011).

[10] TelePresence Tech.
http://telepresencetech.com/. (accessed feb 2012).

[11] iris2iris. Online eye contact with IRIS: that feels like a face to face conversation.
http://www.iris2iris.com/en-UK/content/75/home-uitgebreid.htm. (accessed nov 2011).

[12] Civit, J. & Montserrat, T. JUL 2009. Eye Gaze Correction to Guarantee Eye Contact in Videoconferencing. *IEEE Latin America Transactions*, 7(3), 405–409. 13th Conference on Software Engineering and Databases, Gijon, Spain, OCT 07-10, 2008.

[13] SeeEye2Eye. Teleprompter for web cameras. http://www.bodelin.com/se2e/. (accessed nov 2011).

[14] Knudsen, C. J. S. *Presence Production*. PhD thesis, 2004.

[15] Steuer, J. 1992. Defining virtual reality: Dimensions determining telepresence. *Journal of Communication*, 42, 73–93.

[16] Enlund, N. 2000. The production of presence - distance techniques in education, publishing and art. In *ACS'2000 Proceedings*.

[17] Knudsen, C. J. 2002. Video mediated communication (vmc) - producing a sense of presence between individuals in a shared virtual reality.

[18] Andresen, S. & Knudsen, C. 2003. Boomdance forestillingen (boomdance performance). http://miki.nada.kth.se/t̃hesaint/boom.html. (accessed jan 2012).

[19] Musion Systems Limited: Musion Eyeliner 3D Holographic Projection. http://musion.co.uk/. (accessed apr 2012).

[20] Steinmeyer, J. 1999. *The Science Behind the Ghost*. Burbank.

[21] Dimensional Studios: Musion Eyeliner Holographic Projection System. http://www.eyeliner3d.com/. (accessed apr 2012).

[22] Kramer, W. 2000. What makes a game good? http://www.thegamesjournal.com/articles/WhatMakesaGame.shtml. (accessed jan 2012).

[23] Microsoft Kinect. http://www.xbox.com/kinect/. (accessed jul 2012).

[24] Groenda, H., Nowak, F., & Hanebeck, U. D. 2005. Telepresence techniques for controlling avatar motion in first person games. In *Intelligent Technologies for Interactive Entertainment, First International Conference, INTETAIN 2005, Madonna di Campiglio*, 44–53.

[25] Channel 5: The Gadget Show: E3 2011: Battlefield 3. http://www.youtube.com/watch?annotation_id=annotation_435312&v=eg8Bh5iI2WY&src_vid=nQR49JGySTM&feature=iv. (accessed feb 2012).

[26] Montserrat, T., Zuo, F., Waizenegger, W., Yellin, E., Divorra, O., & Schreer, O. 2010. Public Deliverable D.0.3 - State of the Art Update. 3DPresence- STREP FP7-ICT-2007-1 ref nr 215269, http://www.3dpresence.org. (accessed jan 2012).

[27] Friedman, D. BEAMING (Being in Augmented Multi-modal Naturally-networked Gatherings). http://beaming-eu.org/home. (accessed nov 2011).

[28] Pease, B. & Pease, A. December 2005. *The Definitive Book of Body Language*. Orion Publishing Group.

[29] Mehrabian, A. 1971. *Silent Messages: Implicit Communication of Emotions and Attitudes*. Wadsworth, Belmont, California.

[30] Eibl-Eibesfeldt, I. 1973. *Social Communication and Movement; The Expressive Behaviour of the Deaf-and-Blind-Born*. European Monographs in Social Psychology, 4. New York: Academic Press.

[31] Hirsch, A. 2003. Physical and verbal signs of lying. *Directions in Psychiatry*, 23, 15–19.

[32] Ekman, P. & Friesen, W. V. 1975. *Unmasking the face: A guide to recognizing emotions from facial clues*. Prentice-Hall.

[33] Baron-Cohen, S., Jolliffe, T., Mortimore, C., & Robertson, M. 1997. Another advanced test of theory of mind: Evidence from very high-functioning adults with autism or asperger syndrome. *Journal of Child Psychology and Psychiatry*, 38, 813–822.

[34] Roorda, A. 2002. Human visual system - image formation. *Encyclopedia of Imaging Science and Technology*.

[35] Shiwa, S. & Ishibashi, M. 1991. A large-screen visual telecommunication device enabling eye contact. *SID Dig.*, 22, 327–328.

[36] Izadi, S., Hodges, S., Taylor, S., Rosenfeld, D., Villar, N., Butler, A., & Westhues, J. 2008. Going beyond the display: a surface technology with an electronically switchable diffuser. In *Proceedings of the 21st annual ACM symposium on User interface software and technology*, UIST '08, 269–278, New York, NY, USA. ACM.

[37] Tan, K.-H., Robinson, I. N., Culbertson, B., & Apostolopoulos, J. JUN 2011. ConnectBoard: Enabling Genuine Eye Contact and Accurate Gaze in Remote Collaboration. *IEEE Transaction on Multimedia*, 13(3), 466–473.

[38] Regenbrecht, H. & Hoermann, S. November 2008. Eye-to-eye contact in videoconferencingmediated course advising situations technology and evaluation. Talk at Spotlight Colloquium, University of Otago New Zealand.

[39] 3DFocus. November 2011. Direct eye contact video conferencing to be commercialised. http://www.3dfocus.co.uk/3d-news-2/3d-technology/direct-eye-contact-video-conferencing-to-be-commercialised/6260. (accessed feb 2012).

[40] Pratt, W. 2007. *Digital image processing: PIKS Scientific inside*. Wiley-Interscience.

[41] Aghajan, H. & Cavallaro, A. 2009. *Multi-Camera Networks: Principles and Applications*. Academic Press.

[42] Hartley, R. I. & Zisserman, A. 2004. *Multiple View Geometry in Computer Vision*. Cambridge University Press, ISBN: 0521540518, second edition.

[43] Zhang, Z. April 1998. Determining the epipolar geometry and its uncertainty: A review. *International Journal of computer Vision*, 27, 161–195.

[44] Harris, C. & Stephens, M. 1988. A combined corner and edge detector. *Alvey Vision Conference*, 147–152.

[45] Torr, P. june 2002. Technical Report MSR-TR-56 - A structure and motion toolkit in matlab: Interactive adventures in S and M.
http://research.microsoft.com/apps/pubs/default.aspx?id=69936. (accessed Nov 2011).

[46] Valenti, R., Sebe, N., & Gevers, T. 2012. Combining head pose and eye location information for gaze estimation. *IEEE Transactions on Image Processing*, 21(2), 802–815.

[47] Duchowski, A. T. 2007. *Eye Tracking Methodology: Theory and Practice*. Springer-Verlag New York, Inc., Secaucus, NJ, USA.

[48] Van der Geest, J. & Frens, M. 2002. Recording eye movements with video-oculography and scleral search coils: a direct comparison of two methods. *Journal of neuroscience methods*, 114(2), 185–195.

[49] Savas, Z. June 2008. Trackeye: Real-time tracking of human eyes using a webcam.
http://www.codeproject.com/Articles/26897/TrackEye-Real-Time-Tracking-Of-Human-Eyes-Using-a. (accessed feb 2012).

[50] Li, D., Winfield, D., & Parkhurst, D. J. 2005. Starburst: A hybrid algorithm for video-based eye tracking combining feature-based and model-based approaches. In *Proceedings of the 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05) - Workshops - Volume 03*, 79, Washington, DC, USA. IEEE Computer Society.

[51] Daugman, J. G. November 1993. High confidence visual recognition of persons by a test of statistical independence. *IEEE Trans. Pattern Anal. Mach. Intell.*, 15, 1148–1161.

[52] Yuille, A., Hallinan, P., & Cohen, D. 1992. Feature extraction from faces using deformable templates. *International Journal of computer Vision*, 8(2), 99–111.

[53] Zhu, Z. & Ji, Q. April 2005. Robust real-time eye detection and tracking under variable lighting conditions and various face orientations. *Computer Vision and Image Understanding*, 98, 124–154.

[54] Lowe, D. G. November 2004. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60, 91–110.

[55] Mita, T., Kaneko, T., & Hori, O. 2005. Joint haar-like features for face detection. In *Computer Vision, 2005. ICCV 2005. Tenth IEEE International Conference on*, volume 2, 1619–1626. Ieee.

[56] Zhou, J., Jiang, L., Ji, Z., & Shen, L. 2009. Haar-like features based eye detection algorithm and its implementation on TI TMS320DM6446 platform. In *Imaging Systems and Techniques, 2009. IST'09. IEEE International Workshop on*, 89–93. IEEE.

[57] Tan, K.-H., Kriegman, D., & Ahuja, N. 2002. Appearance-based eye gaze estimation. In *Applications of Computer Vision, 2002. (WACV 2002). Proceedings. Sixth IEEE Workshop on*, 191 – 195.

[58] Pentland, A., Moghaddam, B., & Starner, T. 1994. View-based and modular eigenspaces for face recognition. In *IEEE Internation Conference on Computer Vision & Pattern Recognition*.

[59] Guo, H. sept. 2011. A simple algorithm for fitting a gaussian function [dsp tips and tricks]. *Signal Processing Magazine, IEEE*, 28(5), 134 –137.

[60] Ruddin, N. 2009. Real time eye tracking and blink detection with OpenCV (software). http://nashruddin.com/Real_Time_Eye_Tracking_and_Blink_Detection. (accessed feb 2012).

[61] The Gathering 2012 - Simon McCallum - Parallell Programming. http://vimeo.com/39913086. (accessed jun 2012).

[62] Knott, G. D. October 1974. A proposal for certain process management and intercommunication primitives. *SIGOPS Oper. Syst. Rev.*, 8(4), 7–44.

[63] Hennessy, J. L. & Patterson, D. A. 2006. *Computer Architecture: A Quantitative Approach*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 4th edition.

[64] Butenhof, D. R. 1997. *Programming with POSIX threads*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

[65] Barney, B. 2012. Lawrence Livermore National Laboratory Tutorial: POSIX Threads Programming. https://computing.llnl.gov/tutorials/pthreads/#References. (accessed apr 2012).

[66] Cormen, T. H., Stein, C., Rivest, R. L., & Leiserson, C. E. 2001. *Introduction to Algorithms*. McGraw-Hill Higher Education, 2nd edition.

[67] Smith, S. W. 1997. *The scientist and engineer's guide to digital signal processing*. California Technical Publishing, San Diego, CA, USA.

[68] Yadav, R. 2007. Client / server programming with tcp/ip sockets (ebook). http://devmentor.org. (accessed may 2012).

[69] Gont, F. July 2008. Security assessment of the internet protocol. Centre for the Protection of National Infrastructure. http://www.gont.com.ar/drafts/ip-security/index.html. (accessed may 2012).

[70] Comer, D. 2006. *Internetworking with TCP/IP: principles, protocols, and architecture*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 5th edition.

[71] Gonzalez, R. C. & Woods, R. E. 2002. *Digital Image Processing*. Pretince-Hall, Inc., Upper Saddle River, New Jersey, USA, 2nd edition.

[72] Zielinski, P. 2009. Opengazer: open-source gaze tracker for ordinary webcams (software), samsung and the gatsby charitable foundation.
http://www.inference.phy.cam.ac.uk/opengazer/. (accessed feb 2012).

[73] Li, D., Parkhurst, D., Babcock, J., & Winfield, D. 2006. openEyes: cvEyeTracker - Version 1.2.5 (software).
http://thirtysixthspan.com/openEyes/. (accessed mar 2012).

[74] Nagy, B., Romanca, M., Peter, R., & Matiukas, V. 2009. 17th summer school on image processing (SSIP-2009), Debrecen, Hungary.
http://www.inf.unideb.hu/ ssip/teams/team4/index.html. (accessed feb 2012).

[75] Koh, D., Munikrishne Gowda, S., & Komogortsev, O. 2009. Input evaluation of an eye-gaze-guided interface: kalman filter vs. velocity threshold eye movement identification. In *Proceedings of the 1st ACM SIGCHI symposium on Engineering interactive computing systems*, 197–202. ACM.
http://www.cs.txstate.edu/~ok11/igaze_emd_online.html (accessed feb 2012).

[76] Oliveira, V. & Conci, A. 2009. Skin detection using hsv color space. In *H. Pedrini, & J. Marques de Carvalho, Workshops of Sibgrapi*, 1–2.

[77] van Baren, J. & IJsselsteijn, W. March 2004. Deliverable 5 Version 1.0 (final) - Measuring Presence: a Guide to Current Measurement Approaches.
http://www8.informatik.umu.se/ jwworth/PresenceMeasurement.pdf. (accessed feb 2012).

[78] Witmer, B. G. & Singer, M. J. June 1998. Measuring presence in virtual environments: A presence questionnaire. *Presence: Teleoper. Virtual Environ.*, 7(3), 225–240.

[79] Slater, M., Usoh, M., & Steed, A. 1994. Depth of presence in virtual environments. *Presence-Teleoperators and Virtual Environments*, 3(2), 130–144.

[80] Ray, A. August 2006. Understanding chi square.
http://www.practicalsurveys.com/reporting/chisquare.php. (accessed jun 2012).

[81] NIST. April 2012. Critical values of the chi-square distribution.
http://www.itl.nist.gov/div898/handbook/eda/section3/eda3674.htm. (accessed jun 2012).

A User Experience Questionnaires

A.1 Questionnaire Experiment 1

The questionnaires used after the first experiment to measure the user experience of the participants testing the software (i.e. the program) and the hardware (i.e. the teleprompter) solution.

Number:					
Method:	PROGRAM				
Questions:	1a & 1b	/	2a & 2b		

Sex:	M	/	F		
Age:	__				
Eye sight:	Uncorrected	/	Contact lens	/	Glasses
	If there are any, please specify other eye sight information:				

Video conferencing experience (Skype/MSN with video):

- o Daily
- o More than twice a week, but not daily
- o Weekly
- o More than twice a month, but not weekly
- o Monthly
- o A few times a year
- o Never

TECHNICAL

1. How would you rate the ease of use of the program (i.e. capturing the eyes and the template)?
   (1 is very difficult, 4 is neutral, 7 is very easy)

   1 2 3 4 5 6 7

   Additional comments to this question? :

2. How would you rate the image quality?
   (1 is very poor, 4 is neutral, 7 is very good)

   1 2 3 4 5 6 7

   Additional comments to this question? :

PRESENCE

3. When you were talking to yourself, how close was it to talking in a mirror?
   (1 is not at all, 4 is neutral, 7 is exactly like a mirror)

   - Situation a: (having the eyes corrected, first question)

     1 2 3 4 5 6 7

   - Situation b: (Not having the eyes corrected, second question)

     1 2 3 4 5 6 7

   Additional comments to this question? :

4. How engaged were you in the conversation with yourself?
   (1 is not at all, 4 is neutral, 7 completely engaged)

   - Situation a: (having the eyes corrected, first question)

     1 2 3 4 5 6 7

   - Situation b: (Not having the eyes corrected, second question)

     1 2 3 4 5 6 7

   Additional comments to this question? :

5. Did you feel like you had eye contact with yourself?
   (1 is no eye contact at all, 4 is neutral, 7 is eye contact the entire time)

   - Situation a: (having the eyes corrected, first question)

     1 2 3 4 5 6 7

- Situation b: (Not having the eyes corrected, second question)

  1 2 3 4 5 6 7

  Additional comments to this question? :

6. How natural did the conversation seem, with respect to a real life conversation?
   (1 is not natural, 4 is neutral, 7 is very natural)

   - Situation a: (having the eyes corrected, first question)

     1 2 3 4 5 6 7

   - Situation b: (Not having the eyes corrected, second question)

     1 2 3 4 5 6 7

   Additional comments to this question? :

GENERAL:

7. Do you think the eye correction is an improvement?

8. Any other comments about your experience using the program?

Number:					
Method:	TELEPROMPTER				
Questions:	1a & 1b	/	2a & 2b		

Sex:	M	/	F		
Age:	__				
Eye sight:	Uncorrected	/	Contact lens	/	Glasses

If there are any, please specify other eye sight information:

Video conferencing experience (Skype/MSN with video):

- o Daily
- o More than twice a week, but not daily
- o Weekly
- o More than twice a month, but not weekly
- o Monthly
- o A few times a year
- o Never

TECHNICAL

1. How would you rate the ease of use of the teleprompter (i.e. talking in the box, not including setting up the video chat)?
   (1 is very difficult, 4 is neutral, 7 is very easy)

   1 2 3 4 5 6 7

   Additional comments to this question? :

2. How would you rate the image quality?
   (1 is very poor, 4 is neutral, 7 is very good)

   1 2 3 4 5 6 7

   Additional comments to this question? :

PRESENCE

3. When you were talking to yourself, how close was it to talking in a mirror?
   (1 is not at all, 4 is neutral, 7 is exactly like a mirror)

   - Situation a: (camera behind screen, first question)

     1 2 3 4 5 6 7

   - Situation b: (camera on top, second question)

     1 2 3 4 5 6 7

   Additional comments to this question? :

4. How engaged were you in the conversation with yourself?
   (1 is not at all, 4 is neutral, 7 completely engaged)

   - Situation a: (camera behind screen, first question)

     1 2 3 4 5 6 7

   - Situation b: (camera on top, second question)

     1 2 3 4 5 6 7

   Additional comments to this question? :

5. Did you feel like you had eye contact with yourself?
   (1 is no eye contact at all, 4 is neutral, 7 is eye contact the entire time)

   - Situation a: (camera behind screen, first question)

     1 2 3 4 5 6 7

- Situation b: (camera on top, second question)

  1 2 3 4 5 6 7

  Additional comments to this question? :

6. How natural did the conversation seem, with respect to a real life conversation?
   (1 is not natural, 4 is neutral, 7 is very natural)

   - Situation a: (camera behind screen, first question)

     1 2 3 4 5 6 7

   - Situation b: (camera on top, second question)

     1 2 3 4 5 6 7

   Additional comments to this question? :

GENERAL:

7. Do you think having the camera behind the screen is an improvement with respect to having the camera on top of the screen?

8. Any other comments about your experience using the program?

A.2 Questionnaire Experiment 2

The questionnaire used after the second experiment to measure the user experience of the participants testing the software and the hardware solution. The questionnaires were identical but they had a different title to distinguish which completed survey belonged to which solution.

Eye Contact Presence Questionnaire (Software)

This questionnaire is designed to test the user experience of the software solution proposed in the Master Thesis "Eye Contact in Video Conferencing"
Welcome to Eye Contact Presence Questionnaire. The following questions will ask about your video conferencing experience, in order to assess the performance of the proposed solutions.

There are 13 questions in this survey

General Information

1 [1] What is your gender? *

Please choose **only one** of the following:

○ Female
○ Male

2 [2] What is your age? *

Please write your answer here:

3 [3] How is your eye sight? *

Please choose **only one** of the following:

○ Uncorrected
○ Contact lenses
○ Glasses

I.e. do you wear glasses, contact lenses or neither?

4 [4] What is your video conferencing experience? *

Please choose **only one** of the following:

- ○ Daily
- ○ More than twice a week, but not daily
- ○ Weekly
- ○ More than twice a month, but not weekly
- ○ Monthly
- ○ A few times a year
- ○ Never

I.e. how often do you use video chat tools (such as Skype or MSN messenger) with video?

Immersion

5 [11]How engaged were you in the conversation? *

Please choose **only one** of the following:

- ○ 1
- ○ 2
- ○ 3
- ○ 4
- ○ 5

1 is not at all, 3 is neutral and 5 is very

6 [12] Did you get distracted from the conversation? *

Please choose **only one** of the following:

- ○ 1
- ○ 2
- ○ 3
- ○ 4
- ○ 5

1 is not at all, 3 is neutral and 5 is very

7 [13] How much were you involved by the visual aspects (the video part) of the video conferencing session? *

Please choose **only one** of the following:

- ○ 1
- ○ 2
- ○ 3
- ○ 4
- ○ 5

1 is not at all, 3 is neutral and 5 is very

Naturalness

8 [21] How natural did the conversation seem? *

Please choose **only one** of the following:

- ○ 1
- ○ 2
- ○ 3
- ○ 4
- ○ 5

1 is not at all, 3 is neutral and 5 is very

9 [22] To what extend did you feel as if the video conferencing session was as a real world conversation? *

Please choose **only one** of the following:

- ○ 1
- ○ 2
- ○ 3
- ○ 4
- ○ 5

1 is not at all, 3 is neutral and 5 is very

10 [23] Were you conscious about the media used during the video conferencing session? *

Please choose **only one** of the following:

- ○ 1
- ○ 2
- ○ 3
- ○ 4
- ○ 5

1 is not at all, 3 is neutral and 5 is very

Eye Contact

11 [41]Did you feel like you had eye contact with the other video chat participant? *

Please choose **only one** of the following:

- ○ 1
- ○ 2
- ○ 3
- ○ 4
- ○ 5

1 is not at all, 3 is neutral and 5 is very

12 [42]How natural did the eye contact with the other participant seem? *

Please choose **only one** of the following:

- ○ 1
- ○ 2
- ○ 3
- ○ 4
- ○ 5

1 is not at all, 3 is neutral and 5 is very

13 [43]To what extend did you feel as if the eye contact with the other participant distracted you from the conversation? *

Please choose **only one** of the following:

- ○ 1
- ○ 2
- ○ 3
- ○ 4
- ○ 5

1 is not at all, 3 is neutral and 5 is very

Submit your survey.
Thank you for completing this survey.

B Presence Questionnaire (PQ)

Presence Questionnaire proposed by Witmer & Singer [78].

1. How much were you able to control events?
2. How responsive was the environment to actions that you initiated (or performed)?
3. How natural did your interactions with the environment seem?
4. How completely were all of your senses engaged?
5. How much did the visual aspects of the environment involve you?
6. How much did the auditory aspects of the environment involve you?
7. How natural was the mechanism which controlled movement through the environment?
8. How aware were you of events occurring in the real world around you?
9. How aware were you of your display and control devices?
10. How compelling was your sense of objects moving through space?
11. How inconsistent or disconnected was the information coming from your various senses? 12. How much did your experiences in the virtual environment seem consistent with your real-world experiences?
13. Were you able to anticipate what would happen next in response to the actions that you performed?
14. How completely were you able to actively survey or search the environment using vision?
15. How well could you identify sounds?
16. How well could you localize sounds?
17. How well could you actively survey or search the virtual environment using touch?
18. How compelling was your sense of moving around inside the virtual environment?
19. How closely were you able to examine objects?
20. How well could you examine objects from multiple viewpoints?
21. How well could you move or manipulate objects in the virtual environment?
22. To what degree did you feel confused or disoriented at the beginning of breaks or at the end of the experimental session?
23. How involved were you in the virtual environment experience?
24. How distracting was the control mechanism?
25. How much delay did you experience between your actions and expected outcomes?
26. How quickly did you adjust to the virtual environment experience?
27. How proficient in moving and interacting with the virtual environment did you feel at the end of the experience?
28. How much did the visual display quality interfere or distract you from performing assigned tasks or required activities?
29. How much did the control devices interfere with the performance of assigned tasks or with other activities?

30. How well could you concentrate on the assigned tasks or required activities rather than on the mechanisms used to perform those tasks or activities?
31. Did you learn new techniques that enabled you to improve your performance?
32. Were you involved in the experimental task to the extent that you lost track of time?

C Slater-Usoh-Steed Questionnaire (SUS)

Presence Questionnaire proposed by Slater, Usoh S̃teed Questionnaire (SUS) [79] taken from the presence measurement approaches overview by Van Baren and Ijsselsteijn [77].

1. Please rate your sense of being in the virtual environment, on a scale of 1 to 7, where 7 represents your normal experience of being in a place.
2. To what extent were there times during the experience when the virtual environment was the reality for you?
3. When you think back to the experience, do you think of the virtual environment more as images that you saw or more as somewhere that you visited?
4. During the time of the experience, which was the strongest on the whole, your sense of being in the virtual environment or of being elsewhere?
5. Consider your memory of being in the virtual environment. How similar in terms of the structure of the memory is this to the structure of the memory of other places you have been today? By ’structure of the memory’ consider things like the extent to which you have a visual memory of the virtual environment, whether that memory is in colour, the extent to which the memory seems vivid or realistic, its size, location in your imagination, the extent to which it is panoramic in your imagination, and other such structural elements.
6. During the time of your experience, did you often think to yourself that you were actually in the virtual environment?

D Experimental Data

In Table 7 the original data gathered from the first experiment testing the software solution is displayed. Table 8 shows the original data gathered from the first experiment testing the hardware solution. In order to visualise and evaluate the results the data was later grouped together in 3 bins, namely negative, neutral and positive answers.

		1	2	3	4	5	6	7
	Ease of use	0	1	0	1	2	7	0
	Image quality	0	2	1	1	5	1	1
Like mirror	Gaze correction	0	2	1	4	3	1	0
	No gaze correction	1	2	2	2	1	2	1
Engagement	Gaze correction	1	0	0	1	7	2	0
	No gaze correction	2	0	1	5	1	2	0
Eye contact	Gaze correction	0	2	0	1	3	3	2
	No gaze correction	3	3	2	1	2	0	0
Naturalness	Gaze correction	1	2	3	0	3	2	0
	No gaze correction	2	0	5	1	1	2	0

Table 7: Original data experiment 1 software testing. Frequencies of answers on a scale from 1 to 7.

		1	2	3	4	5	6	7
	Ease of use	0	0	1	0	4	2	4
	Image quality	0	1	2	3	4	1	0
Like mirror	Gaze correction	0	0	2	0	2	5	2
	No gaze correction	2	1	2	4	1	1	0
Engagement	Gaze correction	0	0	5	1	2	3	0
	No gaze correction	0	1	6	2	0	1	1
Eye contact	Gaze correction	0	1	0	0	3	4	3
	No gaze correction	3	1	3	3	0	0	1
Naturalness	Gaze correction	0	3	0	1	1	5	1
	No gaze correction	3	0	1	4	1	2	0

Table 8: Original data experiment 1 hardware testing. Frequencies of answers on a scale from 1 to 7.

E Software Source Code

Listing E.1: Gaze Correction Class

```
/**
* This class performs gaze correction.
* Eyes are detected by blinking or by clicking on the eyes.
* Template capture by looking in the camera and pressing 't'.
* To show the gaze correction to the receiver press 's'
* To send original image to receiver disable gaze correction by pressing 'd'
* To restart tracking and template capture press 'r'
* When eyes lost press 'l', template is preserved
*
* This class in an adjustment and extension of the
* Real Time Eye Tracking and Blink Detection with OpenCV software
* @author  Nash <me@nashruddin.com>
* @license GPL
* @website http://nashruddin.com
*
* See the tutorial at
* http://nashruddin.com/Real_Time_Eye_Tracking_and_Blink_Detection
*
* Ruddin's software located an eye by blinking and tracked the eye
* by template matching within a search window.
*
* The software was extended to:
* - locate and track both eyes
* - have the alternative possibility to click on the eyes
* - capture eye templates
* - calculate a LookUpTable (LUT) filled with Gaussian numbers
* - Gaussian blend template eyes with the current frame image eyes
* - other improvements to the original code
* - altered to a class to function with the networking application
*/


#include "gucOCV.h"
#include "gazeSystem.h"
#include <iostream>
#include <string>


void gazeSystem::correctEyes(IplImage* img)
{
  cvNamedWindow(wnd_name, 1);
  cvCopy(img, frame, NULL);
  frame->origin = 0;

  if (stage == STAGE_INIT)
  {
    window = cvRect(0, 0, frame->width, frame->height);
    cvSetMouseCallback("video", &gazeSystem::on_mouse, this);
  }

  cvCvtColor(frame, gray, CV_BGR2GRAY);
```

```
nc = get_connected_components(gray, prev, window, &comp);

if (stage == STAGE_INIT && is_eye_pair(comp, nc, gray, &eye, &eye2))
{
  delay_frames(5);

  //Get template for template matching first eye
  cvSetImageROI(gray, eye);
  cvCopy(gray, tpl, NULL);
  cvResetImageROI(gray);

  //Get template for template matching second eye
  cvSetImageROI(gray, eye2);
  cvCopy(gray, tpl2, NULL);
  cvResetImageROI(gray);

  blinkCapture=true;
}

if (stage == STAGE_INIT && leftEyeClicked && !rightEyeClicked)
{

  eye = cvRect(
    p.x - (TPL_WIDTH / 2),
    p.y - (TPL_HEIGHT / 2),
    TPL_WIDTH,
    TPL_HEIGHT
    );

  cvSetImageROI(gray, eye);
  cvCopy(gray, tpl, NULL);
  cvResetImageROI(gray);

}

if (stage == STAGE_INIT && rightEyeClicked &&!blinkCapture)
{

  eye2 = cvRect(
    p.x - (TPL_WIDTH / 2),
    p.y - (TPL_HEIGHT / 2),
    TPL_WIDTH,
    TPL_HEIGHT
    );

  cvSetImageROI(gray, eye2);
  cvCopy(gray, tpl2, NULL);
  cvResetImageROI(gray);

}


if (stage==STAGE_INIT && ((leftEyeClicked && rightEyeClicked) || (blinkCapture) ))
{
  if (eye2.x < (eye.x+eye.width))
  {
    CvRect temp = eye;
    eye = eye2;
    eye2 = temp;

    static IplImage * tempIm= cvCreateImage(cvSize(tpl->width, tpl->height), 8, 1);

```

```
      cvCopy(tpl, tempIm, NULL);
      cvCopy(tpl2, tpl, NULL);
      cvCopy(tempIm, tpl2, NULL);

    }

    window_dist = (eye2.x) - (eye.x+WIN_WIDTH);
    stage = STAGE_TRACKING;

  }


  if (stage == STAGE_TRACKING)
  {

    CvRect tempWin;

    found = locate_eye(gray, tpl, &window, &eye, NULL);
    tempWin = cvRect(
      window.x + window.width + window_dist,
      window.y,
      WIN_WIDTH,
      WIN_HEIGHT
      );
    found2 = locate_eye(gray, tpl2, &window2, &eye2, &tempWin);

    if (eye2.x < (eye.x+eye.width))
    {
      CvRect temp = eye;
      eye = eye2;
      eye2 = temp;

      static IplImage * tempIm= cvCreateImage(cvSize(tpl->width, tpl->height), 8, 1);
      cvCopy(tpl, tempIm, NULL);
      cvCopy(tpl2, tpl, NULL);
      cvCopy(tempIm, tpl2, NULL);
    }


    if (is_blink(comp, nc, window, eye))
      blink = 2;


    image_corr_frame = cvCloneImage(frame);
    imageCorrected = true;

    //draw the green rectangle around the eyes 1 pixel larger
    //then when copying the eyes for template the green is excluded
    eyeRect = cvRect(eye.x-1, eye.y-1, eye.width+1, eye.height+1);
    eye2Rect = cvRect(eye2.x-1, eye2.y-1, eye2.width+1, eye2.height+1);
    DRAW_RECTS(frame, diff, window, eyeRect);
    DRAW_RECTS(frame, diff, window2, eye2Rect);

    //the corrected image should not get the green and red rectangles around it
    image_corrected = cvCloneImage(image_corr_frame);

    //get the position of the first eye
    eye_frame_pos=eye;

    //get the position of the second eye
    eye2_frame_pos=eye2;

    if ((found==0 && found2==0) ||  key == 'r')
    {
      stage = STAGE_INIT;

      if (wnd_tpl)
        cvDestroyWindow(wnd_tpl);

      window_dist=NULL;
      setThresh = true;
      //No need to create new LUT
      LUTempty = true;

      imageCorrected = false;
      templateCaptured = false;
      sendImage = false;

      leftEyeClicked=false;
      rightEyeClicked=false;
      clickedEyes=0;
      blinkCapture=false;

    }

    if(key == 't')
    {
      if (checkValidPtr(image_tpl))
        cvReleaseImage(&image_tpl);
      if (checkValidPtr(eye_correction_tpl_image))
        cvReleaseImage(&eye_correction_tpl_image);
      if (checkValidPtr(eye2_correction_tpl_image))
        cvReleaseImage(&eye2_correction_tpl_image);

      image_tpl = cvCloneImage(image_corr_frame);

      cvNamedWindow(wnd_tpl, 1);
      cvShowImage(wnd_tpl, frame);

      //get the template for the first eye
      cvSetImageROI(image_tpl, eye);
      eye_correction_tpl_image=cvCloneImage(image_tpl);
      cvResetImageROI(image_tpl);

      eye_tpl_pos=eye;

      //get the template for the second eye
      cvSetImageROI(image_tpl, eye2);
      eye2_correction_tpl_image=cvCloneImage(image_tpl);
      cvResetImageROI(image_tpl);

      eye2_tpl_pos=eye2;

      templateCaptured = true;
    }

  }

  if (key == 'l')
    stage = STAGE_INIT;

  if (key == 's')
    sendImage=true;

  if (key == 'd')
    sendImage=false;


  cvShowImage(wnd_name, frame);

  if (firstRound == false)
    cvReleaseImage(&prev);
  prev = (IplImage*)cvClone(gray);

  if (imageCorrected && templateCaptured && (found==1 && found2==1) )
  {
    if (blink==0 )
    {
      doCorrection(image_corrected, eye_frame_pos, eye_tpl_pos,
          eye_correction_tpl_image);
      doCorrection(image_corrected, eye2_frame_pos, eye2_tpl_pos,
          eye2_correction_tpl_image);
    }

    if (blink > 0)
      blink--;

    if (sendImage == true)
      cvCopyImage(image_corrected, img);
  }

  firstRound = false;
  if (checkValidPtr(image_corr_frame))
    cvReleaseImage(&image_corr_frame);
  if (checkValidPtr(image_corrected))
    cvReleaseImage(&image_corrected);


  key  = cvWaitKey(15);

}

void gazeSystem::on_mouse(int event,int x,int y,int flag, void* param)
{
  gazeSystem* tempGaze = (gazeSystem *)param;

  switch( event ){
  case CV_EVENT_LBUTTONDOWN:
    {
      tempGaze->blinkCapture=false;
      tempGaze->setxy(x , y);

    }
    break;
  }
}


void gazeSystem::setxy(int x, int y)
{
  p.x = x;
  p.y = y;
  clickedEyes++;


  if (clickedEyes==1)
```

```
    leftEyeClicked=true;
  if (clickedEyes==2)
    rightEyeClicked=true;
}
/**
* This is the wrapper function for cvFindContours
*
* @param  IplImage* img    the current grayscaled frame
* @param  IplImage* prev     previously saved frame
* @param  CvRect    window   search within this window
* @param  CvSeq**   comp     output parameter, will contain the connected components
* @return int          the number of connected components
*/
int  gazeSystem::get_connected_components(IplImage* img, IplImage* prev, CvRect window,
     CvSeq** comp)
{
  IplImage* _diff;

  cvZero(diff);

  /* apply search window to images */
  cvSetImageROI(img, window);
  cvSetImageROI(prev, window);
  cvSetImageROI(diff, window);

  /* motion analysis */
  cvSub(img, prev, diff, NULL);
  cvThreshold(diff, diff, 5, 255, CV_THRESH_BINARY);
  cvMorphologyEx(diff, diff, NULL, kernel, CV_MOP_OPEN, 1);

  /* reset search window */
  cvResetImageROI(img);
  cvResetImageROI(prev);
  cvResetImageROI(diff);

  _diff = (IplImage*)cvClone(diff);

  /* get connected components */
  int nc = cvFindContours(_diff, storage, comp, sizeof(CvContour),
    CV_RETR_CCOMP, CV_CHAIN_APPROX_SIMPLE, cvPoint(0,0));

  cvClearMemStorage(storage);
  cvReleaseImage(&_diff);

  return nc;
}

/**
* Experimentally-derived heuristics to determine whether
* the connected components are eye pair or not.
*
* @param  CvSeq*  comp the connected components
* @param  int      num  the number of connected components
* @param   CvRect* eye   output parameter, will contain the location of the
*                        first component
* @return int            '1' if eye pair, '0' otherwise
*/
int gazeSystem::is_eye_pair(CvSeq* comp, int num, IplImage* gray, CvRect* eye, CvRect*
    eye2)
{
  if (comp == 0 || num != 2)
    return 0;

  CvRect r1 = cvBoundingRect(comp, 1);
  comp = comp->h_next;

  if (comp == 0)
    return 0;

  CvRect r2 = cvBoundingRect(comp, 1);

  // if the components are lower than at 1/4 of frame, or higher than 2/3 of frame then
        return 0
  if (r1.y < gray->height*0.25 || r2.y < gray->height*0.25 || r1.y > gray->height*0.66
      || r2.y > gray->height*0.66)
    return 0;

  /* the width of the components are about the same */
  if (abs(r1.width - r2.width) >= 5)
    return 0;

  /* the height f the components are about the same */
  if (abs(r1.height - r2.height) >= 5)
    return 0;

  /* vertical distance is small */
  if (abs(r1.y - r2.y) >= 5)
    return 0;

  /* reasonable horizontal distance, based on the components' width */
  int dist_ratio = abs(r1.x - r2.x) / r1.width;
  if (dist_ratio < 2 || dist_ratio > 5)
    return 0;

  /* get the centroid of the 1st component */
  CvPoint point = cvPoint(
    r1.x + (r1.width / 2),
    r1.y + (r1.height / 2)
    );

  /* get the centroid of the 2nd component */
  CvPoint point2 = cvPoint(
    r2.x + (r2.width / 2),
    r2.y + (r2.height / 2)
    );

  /* return 1st eye boundaries */
  *eye = cvRect(
    point.x - (TPL_WIDTH / 2),
    point.y - (TPL_HEIGHT / 2),
    TPL_WIDTH,
    TPL_HEIGHT
    );

  /* return 2nd eye boundaries */
  *eye2 = cvRect(
    point2.x - (TPL_WIDTH / 2),
    point2.y - (TPL_HEIGHT / 2),
    TPL_WIDTH,
    TPL_HEIGHT
    );

  return 1;
}
```

```

/**
* Locate the user's eye with template matching
*
* @param  IplImage* img      the source image
* @param  IplImage* tpl      the eye template
* @param  CvRect*   window   search within this window,
*                             will be updated with the recent search window
* @param  CvRect*   eye      output parameter, will contain the current
*                             location of user's eye
* @return int                '1' if found, '0' otherwise
*/

int gazeSystem::locate_eye(IplImage* img, IplImage* tpl, CvRect* window, CvRect* eye,
    CvRect* newThingy = NULL)
{
  IplImage* tm;
  CvRect    win;
  CvPoint   minloc, maxloc, point;
  double    minval, maxval;
  int     w, h;

  eyesLost=false;
  /* get the centroid of eye */
  point = cvPoint(
    (*eye).x + (*eye).width / 2,
    (*eye).y + (*eye).height / 2
    );

  /* setup search window
  replace the predefined WIN_WIDTH and WIN_HEIGHT above
  for your convenient */
  win = cvRect(
    point.x - WIN_WIDTH / 2,
    point.y - WIN_HEIGHT / 2,
    WIN_WIDTH,
    WIN_HEIGHT
    );
  if (newThingy != NULL)
  {
    win.x = newThingy->x;
    win.y = newThingy->y;
    win.height = newThingy->height;
    win.width = newThingy->width;
  }

  /* make sure that the search window is still within the frame */
  if (win.x < 0)
    win.x = 0;
  if (win.y < 0)
    win.y = 0;
  if (win.x + win.width > img->width)
    win.x = img->width - win.width;
  if (win.y + win.height > img->height)
    win.y = img->height - win.height;

  // create new image for template matching result where:
  w  = win.width  - tpl->width  + 1;
  h  = win.height - tpl->height + 1;
  tm = cvCreateImage(cvSize(w, h), IPL_DEPTH_32F, 1);

  /* apply the search window */
  cvSetImageROI(img, win);

  /* template matching */
  cvMatchTemplate(img, tpl, tm, CV_TM_SQDIFF_NORMED);
  cvMinMaxLoc(tm, &minval, &maxval, &minloc, &maxloc, 0);

  /* release things */
  cvResetImageROI(img);
  cvReleaseImage(&tm);

  /* only good matches */
  if (setThresh==true)
  {
    //min_threshold = (maxval + minval)/1.5;
    min_threshold = 0.57*(maxval + minval);
    min_eye_threshold = 2*maxval;
    setThresh = false;
  }
  if (maxval < min_threshold)
  {
    //eyesLost=true;
    return 0;
  }

  /* return the search window */
  *window = win;


  /* return eye location */
  *eye = cvRect(
    win.x + minloc.x,
    win.y + minloc.y,
    TPL_WIDTH,
    TPL_HEIGHT
    );


  return 1;
}

/**
* Gaze correction with Gaussian blend
*
* @param  IplImage* img      the source image
* @param  CvRect*   eye_pos    the position of the eye
* @param  CvRect*   eye_tpl_pos the position of the template eyes
* @param  IplImage* eye_tpl    the eye template
*
* @return int            '1'
*/
int gazeSystem::doCorrection(IplImage* img, CvRect eye_pos, CvRect eye_tpl_pos,
    IplImage* eye_tpl)
{
  getLUT(LUTempty);

  cvSetImageROI(img, eye_pos);

  // Do Gaussian blending with LUT
  int LUTindex = 0;
  for (int y = 0; y < eye_pos.height; y++)
  {
    for (int x = 0; x < eye_pos.width; x++)
    {

      int indexEyeTpl = (eye_tpl_pos.y+y)*img->widthStep+((eye_tpl_pos.x+x)*3);
      int index = (eye_pos.y+y)*img->widthStep+((eye_pos.x+x)*3);

      img->imageData[index]= (unsigned char)img->imageData[index]*(1-LUT[LUTindex]);
      img->imageData[index] += (unsigned char) eye_tpl->imageData[indexEyeTpl] * LUT[
          LUTindex];

      img->imageData[index+1]= (unsigned char)img->imageData[index +1]*(1-LUT[LUTindex
          ]);
      img->imageData[index+1] += (unsigned char) eye_tpl->imageData[indexEyeTpl+1] *
          LUT[LUTindex];

      img->imageData[index+2]= (unsigned char)img->imageData[index+2]*(1-LUT[LUTindex])
          ;
      img->imageData[index+2] += (unsigned char) eye_tpl->imageData[indexEyeTpl+2] *
          LUT[LUTindex];

      LUTindex++;
    }
  }
  cvResetImageROI(img);

  return 1;
}


//this is LUT as the flatted pyramid

/**
* Create a 2D Look Up Table (LUT) filled with Gaussian values
*
* @param  bool LUTempty     only fill LUT if it is still empty
*
*/
int gazeSystem::getLUT(bool LUTempty)
{
  if (LUTempty == false)
    return 1;

  FILE *fp;
  fp = fopen("LUT_flatted.txt", "w");


  double a=1;
  double b=0;
  double temp;
  double c=2.5;
  double e=2.718281828;
  double LUTvalue;
  int LUTindex = 0;

  for (double y = -0.5*TPL_HEIGHT; y < 0.5*TPL_HEIGHT; y++)
  {
    for (double x = -0.5*TPL_WIDTH; x < 0.5*TPL_WIDTH; x++)
    {
      temp = sqrt((x*x) + (y*y));

      if (temp > 3)
      {
        b=3.2;
        LUTvalue = a*pow(e, -(pow(temp-b,2)/ (2*pow(c,2)) ) );
      }
      else
        LUTvalue = 1;

      //fill LUT matrix
      LUT[LUTindex] = LUTvalue;
      LUTindex++;
      fprintf(fp, "%6.3f", LUT[LUTindex]);

    }

    fprintf(fp, "\n");
  }

  fclose(fp);
  LUTempty= false;
  return 1;
}


int gazeSystem::is_blink(CvSeq* comp, int num, CvRect window, CvRect eye)
{
  if (comp == 0 || num != 1)
    return 0;

  CvRect r1 = cvBoundingRect(comp, 1);

  /* component is within the search window*/
  if (r1.x < window.x)
    return 0;
  if (r1.y < window.y)
    return 0;
  if (r1.x + r1.width > window.x + window.width)
    return 0;
  if (r1.y + r1.height > window.y + window.height)
    return 0;

  /* get the centroid of eye */
  CvPoint pt = cvPoint(
    eye.x + eye.width / 2,
    eye.y + eye.height / 2
    );

  /* component is located at the eye's centroid */
  if (pt.x <= r1.x || pt.x >= r1.x + r1.width)
    return 0;
  if (pt.y <= r1.y || pt.y >= r1.y + r1.height)
    return 0;

  return 1;
}

/**
* Delay for the specified frame count. I have to write this custom
* delay function for these reasons:
* - usleep() is not available in Windows
* - usleep() and Sleep() will freeze the video for the given interval
*/
void gazeSystem::delay_frames(int nframes)
{
  int i;

  for (i = 0; i < nframes; i++)
  {
    if (!frame)
      exit_nicely("cannot␣query␣frame");
    cvShowImage(wnd_name, frame);
  }
}

/**
* Initialize images, memory, and windows
* Constructor
*/
gazeSystem::gazeSystem(int imgWidth, int imgHeight)
{
  stage = STAGE_INIT;
  wnd_name  = "video";
  wnd_debug = "diff";
  wnd_frame = "frame";
  wnd_tpl = "template";
  wnd_corr = "corrected";
  WIN_WIDTH = TPL_WIDTH * 1.5;
  WIN_HEIGHT  = TPL_HEIGHT * 1.5;
  TM_THRESHOLD  = 0.1;
  setThresh=true;
  min_threshold = TM_THRESHOLD;
  min_eye_threshold = TM_THRESHOLD;
  templateCaptured = false;
  imageCorrected = false;
  sendImage = false;
  firstRound = true;

  leftEyeClicked=false;
  rightEyeClicked=false;
  blinkCapture = false;


  int delay, i;
  blink=0;
  LUTempty = true;
  FRAME_WIDTH = imgWidth;
  FRAME_HEIGHT = imgHeight;
  eyesLost=false;
  found = 0;
  found2 = 0;
  comp = 0;
  key = 0;

  clickedEyes = 0;

  p = cvPoint(0,0);

  storage = cvCreateMemStorage(0);
  if (!storage)
    exit_nicely("cannot␣allocate␣memory␣storage!");

  frame   = cvCreateImage(cvSize(imgWidth, imgHeight), IPL_DEPTH_8U, 3);

  if (!frame)
    exit_nicely("cannot␣query␣frame!");

  cvInitFont(&font, CV_FONT_HERSHEY_SIMPLEX, 0.4, 0.4, 0, 1, 8);

  kernel = cvCreateStructuringElementEx(3, 3, 1, 1, CV_SHAPE_CROSS, NULL);
  gray   = cvCreateImage(cvGetSize(frame), IPL_DEPTH_8U, 1);
  prev   = cvCreateImage(cvGetSize(frame), IPL_DEPTH_8U, 1);
  diff   = cvCreateImage(cvGetSize(frame), IPL_DEPTH_8U, 1);
  tpl    = cvCreateImage(cvSize(TPL_WIDTH, TPL_HEIGHT), 8, 1);
  tpl2   = cvCreateImage(cvSize(TPL_WIDTH, TPL_HEIGHT), 8, 1);

  if (!kernel || !gray || !prev || !diff || !tpl)
    exit_nicely("system␣error.");

  gray->origin  = frame->origin;
  prev->origin  = frame->origin;
  diff->origin  = frame->origin;

}

/**
* This function provides a way to exit nicely
* from the system
*
* @param char* msg error message to display
*/
void gazeSystem::exit_nicely(char* msg)
{
  cvDestroyAllWindows();
  exit(0);
}
```

Listing E.2: Gaze Correction Class Header

```
#ifndef _GAZESYSTEM_
#define _GAZESYSTEM_


#include <Windows.h>
#include <stdio.h>
#include "cv.h"
#include "highgui.h"
#include "cxcore.h"
#include "iostream"
//#include <math.h>

#define DRAW_RECTS(f, d, rw, ro)                \
do {                                            \
  cvRectangle(f, POINTS(rw), CV_RGB(255, 0, 0), 1, 8, 0);   \
  cvRectangle(f, POINTS(ro), CV_RGB(0, 255, 0), 1, 8, 0);   \
  cvRectangle(d, POINTS(rw), cvScalarAll(255),  1, 8, 0);   \
  cvRectangle(d, POINTS(ro), cvScalarAll(255),  1, 8, 0);   \
} while(0)

#define POINT_TL(r)   cvPoint(r.x, r.y)
#define POINT_BR(r)   cvPoint(r.x + r.width, r.y + r.height)
#define POINTS(r)   POINT_TL(r), POINT_BR(r)


class gazeSystem {
public:

//static const int FRAME_WIDTH  = 240;
//static const int FRAME_HEIGHT = 180;
```

```
static const int TPL_WIDTH  = 26;
static const int TPL_HEIGHT   = 18;
double WIN_WIDTH;
double WIN_HEIGHT;
double TM_THRESHOLD;
static const int STAGE_INIT = 1;
static const int STAGE_TRACKING= 2;

int        FRAME_WIDTH, FRAME_HEIGHT;
IplImage*    frame, * gray, * prev, * diff, * tpl, * tpl2;
CvMemStorage* storage;
IplConvKernel*   kernel;
CvFont       font;
char*      wnd_name;
char*      wnd_debug;
char*      wnd_frame;
char*      wnd_tpl;
char*      wnd_corr;


IplImage*    image_frame, * image_tpl, * image_corrected, *eye_correction_tpl_image, *
     eye2_correction_tpl_image, *eye_image, *imgGray, *tplGray;
IplImage*    image_corr_frame;
char*      lookWhere;
CvRect       window, window2, eye, eye2, eye_frame_pos, eye2_frame_pos, eye_tpl_pos,
     eye2_tpl_pos, eyeRect, eye2Rect;
//CvRect*      eye_frame_pos, *eye2_frame_pos, *eye_tpl_pos, *eye2_tpl_pos, *eyeRect, *
     eye2Rect;
bool       LUTempty, setThresh, eyesLost;
bool       templateCaptured;
bool       imageCorrected, sendImage, firstRound, leftEyeClicked, rightEyeClicked,
     blinkCapture;
double      LUT[TPL_WIDTH*TPL_HEIGHT];
double      min_threshold, min_eye_threshold;
//int        found, found2;
int        stage;
int        window_dist;
int        blink, found, found2;
CvSeq*   comp;
int    nc;
int    text_delay;
int    key;

int    clickedEyes;
CvPoint p;

void correctEyes(IplImage* img);
static void on_mouse(int event,int x,int y,int flag, void* param);
void setxy(int x, int y);
int  get_connected_components(IplImage* img, IplImage* prev, CvRect window, CvSeq**
     comp);
int  is_eye_pair(CvSeq* comp, int num, IplImage* frame, CvRect* eye, CvRect* eye2);
int  locate_eye(IplImage* img, IplImage* tpl, CvRect* window, CvRect* eye, CvRect*
     tempWin);
//int  doTplCheck(IplImage* img, IplImage** tpl, char * text);
int  doCorrection(IplImage* img, CvRect eye_pos, CvRect eye_tpl_pos, IplImage* eye_tpl)
     ;
int  getLUT(bool LUTempty);
int  is_blink(CvSeq* comp, int num, CvRect window, CvRect eye);
void delay_frames(int nframes);
//void init();
gazeSystem(int imgWidth, int imgHeight);
```

```
void exit_nicely(char* msg);

};

#endif
```

Listing E.3: Software Program Main Architecture.
Coded by Jayson Mackie research assistant for the Game Technology Group at Gjøvik University College

```

#include <omp.h>
#include "gucOCV.h"
#include "gazeSystem.h"

//==================================================================================

//initialize the gazeSystem object gazecorrect
gazeSystem* gazecorrect;

//when correctGaze is called, call the method correctEyes
void correctGaze(IplImage* img)
{
  gazecorrect->correctEyes(img);
}

//==================================================================================


void loopReceiveListener(gucQueue* q, gucQueue* output,int type = 0)
{
    IplImage* tempImage = tempImage = q->popImage();
    if ((tempImage == NULL) || (tempImage->imageSize == 0))
        return;

    // recieved image output output
    cvShowImage("gucReceiveData␣-␣from␣the␣network", tempImage);
    cvReleaseImage(&tempImage);
}

// do not change
void localEchoListener(gucQueue* q, gucQueue* outputQueue, int type = 0)
{
    IplImage* tempImage = tempImage = q->popImage();
    if ((tempImage == NULL) || (tempImage->imageSize == 0))
        return;

    cvShowImage("gucLocalData␣-␣unmodified", tempImage);

    static IplImage* pushImage = cvCreateImage(cvSize(tempImage->width, tempImage->
        height), IPL_DEPTH_8U, 3);

    // do image processing on tempImage->imageData
    // and afterwards, for example
    correctGaze(tempImage);

    cvCopyImage(tempImage, pushImage);

    if (outputQueue != NULL)
    {
        outputQueue->pushPtr(pushImage, true);
    }
}

void cameraCatureAndSend()
{
    if (!cvStartWindowThread())
        printf("Window␣Thread␣not␣available\n");

    cvNamedWindow("gucLocalData␣-␣unmodified");
    if (showSendingImage)
        cvNamedWindow("gucSendData␣-␣prior␣to␣sending");

    gucQueue* localQueue;
    localQueue = new gucQueue(FILL_QUEUE_FROM_IPLIMAGE);
    localQueue->setCaptureProperty(CV_CAP_PROP_FRAME_WIDTH, imageWidth);
    localQueue->setCaptureProperty(CV_CAP_PROP_FRAME_HEIGHT, imageHeight);
    localQueue->setCaptureProperty(CV_CAP_PROP_FPS, 30);

    gucQueue* sendQueue;
    sendQueue = new gucQueue(gucQueueTypes::FILL_QUEUE_FROM_USER_IPLIMAGE_TO_NETWORK,
        serverName, serverPort);

    // add the local image processing to camera queue
    localQueue->addListener(localEchoListener, sendQueue);

    localQueue->setCameraCapture(true);

    cvWaitKey(0);


    localQueue->quit("localQueue␣quit␣by␣main()");
    sendQueue->quit("sendQueue␣quit␣by␣main()");
}
void networkReceiveAndDisplay()
{
    if (!cvStartWindowThread())
        printf("Window␣Thread␣not␣available\n");

    cvNamedWindow("gucReceiveData␣-␣from␣the␣network");

    gucQueue* receiveQueue;
    receiveQueue = new gucQueue(gucQueueTypes::FILL_QUEUE_FROM_NETWORK_WITH_IPLIMAGE,
        serverName, serverPort);

    // add the display function to the receive queue
    receiveQueue->addListener(loopReceiveListener);

  cvWaitKey(0);


    receiveQueue->quit("receiveQueue␣quit␣by␣main()");

}

// do not change
void sendReceiveLoopWithLocalEcho()
{
    if (!cvStartWindowThread())
        printf("Window␣Thread␣not␣available\n");

    cvNamedWindow("gucLocalData␣-␣unmodified");
    if (showSendingImage)
```

```
        cvNamedWindow("gucSendData␣-␣prior␣to␣sending");
    cvNamedWindow("gucReceiveData␣-␣from␣the␣network");


    gucQueue* localQueue;
    localQueue = new gucQueue(FILL_QUEUE_FROM_IPLIMAGE);
    localQueue->setCaptureProperty(CV_CAP_PROP_FRAME_WIDTH, imageWidth);
    localQueue->setCaptureProperty(CV_CAP_PROP_FRAME_HEIGHT, imageHeight);
    localQueue->setCaptureProperty(CV_CAP_PROP_FPS, 30);

    gucQueue* sendQueue;
    sendQueue = new gucQueue(gucQueueTypes::FILL_QUEUE_FROM_USER_IPLIMAGE_TO_NETWORK,
        serverName, serverPort);

    gucQueue* receiveQueue;
    receiveQueue = new gucQueue(gucQueueTypes::FILL_QUEUE_FROM_NETWORK_WITH_IPLIMAGE,
        serverName, serverPort);

    // add the display function to the receive queue
    receiveQueue->addListener(loopReceiveListener);

    // add the local image processing to camera queue
    localQueue->addListener(localEchoListener, sendQueue);

    localQueue->setCameraCapture(true);

    cvWaitKey(0);


    localQueue->quit("localQueue␣quit␣by␣main()");
    receiveQueue->quit("receiveQueue␣quit␣by␣main()");
    sendQueue->quit("sendQueue␣quit␣by␣main()");

}

//==============================================================================


int main(int argv, const char** argc)
{

    int type = 0;// 0 is send, 1 is receive, other is both
    showSendingImage = true;  // show the intermediate image on the way to the network
        queue

    // image size options with lifecam 320x240, 424x240, 640x360, 800x448
    if (argv == 4)
    {
        type = atoi(argc[1]);
        imageWidth = atoi(argc[2]);
        imageHeight = atoi(argc[3]);
        strcpy(serverName, "127.0.0.1"); //if no ip adress given use localhost
        serverPort = 3333;
    }
    else if (argv == 6)
    {
        type = atoi(argc[1]); //send, receive or both
        imageWidth = atoi(argc[2]);
        imageHeight = atoi(argc[3]);
        strcpy(serverName, argc[4]); //to and from ip adress
        serverPort = atoi(argc[5]);  //with port number
    }
    else
    {
        imageWidth = DEFAULT_CAMERA_WIDTH;
        imageHeight = DEFAULT_CAMERA_HEIGHT;
        strcpy(serverName, "127.0.0.1");
        serverPort = 3333;
    }

    globalImageSize = imageWidth * imageHeight * DEFAULT_CAMERA_BYTE_DEPTH;
    printf("\nWidth\t:%d\nHeight\t:%d\nDepth\t:%d\nSize\t:%d\n\n",imageWidth,
        imageHeight, DEFAULT_CAMERA_BYTE_DEPTH, globalImageSize);

  //create an instance of the gazeSystem object
  gazecorrect = new gazeSystem(imageWidth, imageHeight);

    gucWinsock* w = gucWinsock::getInstance();


    if (type == 0)
    {
        cameraCatureAndSend();
    }
    else if (type == 1)
    {
        networkReceiveAndDisplay();
    }
    else
    {
        // this invokes the full queue/network startup
        sendReceiveLoopWithLocalEcho();
    }


    //system("pause");
    w->cleanupWinsock();
    exit(0);
}
```

Listing E.4: Main Program Header.
Coded by Jayson Mackie research assistant for the Game Technology Group at Gjøvik University College

```
#ifndef _GUCOCV_H_
#define _GUCOCV_H_

// supress warnings that I know are ok
#ifdef _MSC_VER
    #define _CRT_SECURE_NO_WARNINGS
    #pragma warning(disable: 4996)  // using old fopen not fopen_s
    #pragma warning(disable: 4482)  // c+11 use os enums
#endif

// system includes
#include <Windows.h>
#include <string>
#include <vector>
#include <queue>
#include <stdio.h>
#include <stdlib.h>
#include <assert.h>

```

```
// OpenMP
#include <omp.h>

// threading
#include <pthread.h>

// openCV
#include <opencv/cv.h>
#include <opencv/highgui.h>

#ifdef _DEBUG

#define checkValidPtr(a) ((unsigned int)a==0xcdcdcdcd?false:true)

#else

#define checkValidPtr(a) ((unsigned int)a==0xbaadf00d?false:true)

#endif

#define APPLICATION_FRAMERATE 30
#define APPLICATION_FRAME_DELAY 1000/APPLICATION_FRAMERATE/4

typedef enum {
    FILL_QUEUE_FROM_USER_DATA = 0,
    FILL_QUEUE_FROM_USER_DATA_TO_NETWORK,
    FILL_QUEUE_FROM_IPLIMAGE,
    FILL_QUEUE_FROM_IPLIMAGE_TO_NETWORK,
    FILL_QUEUE_FROM_USER_IPLIMAGE_TO_NETWORK,
    FILL_QUEUE_FROM_NETWORK_WITH_USER_DATA,
    FILL_QUEUE_FROM_NETWORK_WITH_IPLIMAGE
} gucQueueTypes;


typedef enum {
    USER_DATA = 0,
    IPLIMAGE,
    NETWORK_USER_DATA,
    NETWORK_IPLIMAGE
} gucQueueData;

#define QUEUE_SIZE 64 // big enough to not need threads, throw memory at it

#define DEFAULT_CAMERA_WIDTH     320
#define DEFAULT_CAMERA_HEIGHT    240
#define DEFAULT_CAMERA_BIT_DEPTH 24
#define DEFAULT_CAMERA_BYTE_DEPTH DEFAULT_CAMERA_BIT_DEPTH>>3
#define DEFAULT_CAMERA_IMAGE_SIZE DEFAULT_CAMERA_WIDTH*DEFAULT_CAMERA_HEIGHT*
    DEFAULT_CAMERA_BYTE_DEPTH

//#define JPEG_QUALITY 40
//#define JPEG_BUFFER_SIZE 256000

#define DEFAULT_SERVER_SEND_JPEG true

extern const int SERVER_SEND_JPEG;

extern int imageWidth;
extern int imageHeight;
extern int globalImageSize;
extern char serverName[128];
extern int serverPort;
extern bool showSendingImage;

#define TIMING
#define __TIMED_START if (timing) {
#define __TIMED_END }
#define __TIMED_CODE(a) if (timing) {a;}


// forward declare this API
class gucQueueBase;
class gucSendData;
class gucReceiveData;
class gucQueue;

#include "gucQueue.h"
#include "gucWinsock.h"
#include "gucNetworkBase.h"
#include "gucSendData.h"
#include "gucReceiveData.h"

#endif
```

Listing E.5: Instantiates global variables.
Coded by Jayson Mackie research assistant for the Game Technology Group at Gjøvik University College

```

#include "gucOCV.h"

const int SERVER_SEND_JPEG = false;

int imageWidth;
int imageHeight;
int globalImageSize;
char serverName[128];
int serverPort;
bool showSendingImage;
```

Listing E.6: Queue Class.
Coded by Jayson Mackie research assistant for the Game Technology Group at Gjøvik University College

```

#include "gucQueue.h"

/*
*
*   Private constructor
*
*/
gucQueue::gucQueue()
{
    capture = NULL;
    networkSend = NULL;
    networkReceive = NULL;
    dataReceived = false;
}

/*
```

```
*
*   Public constructor
*
*/
gucQueue::gucQueue(int qType, const char* address, int port, bool delayedStart)
{
    maxQueueSize = 2;
    cameraCapture = false;
    running = !delayedStart;
    gucQueue();
    queueType = qType;

    switch (queueType)
    {
    case (gucQueueTypes::FILL_QUEUE_FROM_USER_DATA):
        // nothing to do here, just wait for data
        break;

    case (gucQueueTypes::FILL_QUEUE_FROM_IPLIMAGE):

        // set up the camera
        capture = cvCaptureFromCAM( 0 );
        if( !capture ) {
            fprintf( stderr, "ERROR: gucQueue(FILL_QUEUE_FROM_IPLIMAGE): openCV camera 
                is NULL : camera %d\n",cameraID );
            getchar();
        }
        break;

    case (gucQueueTypes::FILL_QUEUE_FROM_IPLIMAGE_TO_NETWORK):

        // set up the camera and the network send object
        capture = cvCaptureFromCAM( 0 );
        if( !capture ) {
            fprintf( stderr, "ERROR: gucQueue(FILL_QUEUE_FROM_IPLIMAGE_TO_NETWORK): 
                openCV camera is NULL : camera %d\n",cameraID );
            getchar();
        }

        if( strlen(address) == 0) {
            fprintf( stderr, "ERROR: gucQueue(FILL_QUEUE_FROM_IPLIMAGE_TO_NETWORK): 
                address is empty\n");
            getchar();
        }

        networkSend = new gucSendData(this);
        networkSend->initialise(address, port); // this starts a thread
        addListener(networkSend);
        break;

    case(gucQueueTypes::FILL_QUEUE_FROM_USER_IPLIMAGE_TO_NETWORK):

        // setup the network send object
        if( strlen(address) == 0) {
            fprintf( stderr, "ERROR: gucQueue(FILL_QUEUE_FROM_USER_DATA_TO_NETWORK): 
                address is empty\n");
            getchar();
        }
        networkSend = new gucSendData(this);
        networkSend->initialise(address, port); // this starts a thread
        addListener(networkSend);
        break;

    case(gucQueueTypes::FILL_QUEUE_FROM_USER_DATA_TO_NETWORK):

        // set up the camera and the network send object
        if( strlen(address) == 0) {
            fprintf( stderr, "ERROR:␣gucQueue(FILL_QUEUE_FROM_USER_DATA_TO_NETWORK):␣
                address␣is␣empty\n");
            getchar();
        }
        networkSend = new gucSendData(this);
        networkSend->initialise(address, port); // this starts a thread
        addListener(networkSend);
        break;

    case (gucQueueTypes::FILL_QUEUE_FROM_NETWORK_WITH_USER_DATA):
    case (gucQueueTypes::FILL_QUEUE_FROM_NETWORK_WITH_IPLIMAGE):

        // set up the network receive object
        if( strlen(address) == 0) {
            fprintf( stderr, "ERROR:␣gucQueue(FILL_QUEUE_FROM_NETWORK_WITH_X):␣address␣
                is␣empty\n");
            getchar();
        }
        networkReceive = new gucReceiveData(this);
        networkReceive->initialise(address, port); // this starts a thread
        //addListener(??); // not required !!! whomever set up the network queue gave
             a listener
        break;
    }
    initialise();
}

/*
*
*   Initialiser to start the thread, this may be
*
*/
void gucQueue::initialise(const char* address, int port)
{
    accessingData = PTHREAD_MUTEX_INITIALIZER;

  // run the streaming server as a separate thread
  if (pthread_create(&queueThread, NULL, gucQueue::threadInitialise, (void *)this)) {
    quit("pthread_create␣failed.", 1);
  }
};

/**
 * This is the streaming server, run as a separate thread
 * This function waits for a client to connect, and send the grayscaled images
 */
void* gucQueue::threadInitialise(void* q)
{
    gucQueue* mySelf = reinterpret_cast<gucQueue *>(q);
    mySelf->threadLoop();
    return NULL;
}

void gucQueue::threadLoop()
{
    bool autoUpdateRequired = true;

    // the user data types only need to update when a push has occured
    if ((queueType == gucQueueTypes::FILL_QUEUE_FROM_USER_DATA) ||
        (queueType == gucQueueTypes::FILL_QUEUE_FROM_USER_DATA_TO_NETWORK))
        autoUpdateRequired = false;

    while (running)
    {
        if (dataReceived || autoUpdateRequired)
        {
            update();
        }
        Sleep(10);  // run the update not more than 100 times per second
    }
    int bob = 43;
}

void gucQueue::cleanup()
{
    clear();  // this might leak memory if called mid program, need a deep delete, !!!
        TODO
    if (checkValidPtr(capture))// ((unsigned int)capture != 0xcdcdcdcd) && ((unsigned
        int)capture != 0xbaadf00d) ) {  // <<<<<<<<<<<<<<<< HATE THIS!!!!  a NULL
        CvCapture* is not NULL but 0xcdcdcdcd
    {
        cvReleaseCapture(&capture);
    }

  pthread_mutex_destroy(&accessingData);
}
/**
 * this function provides a way to exit nicely from the system
 */
void gucQueue::quit(char* msg, int retval)
{

    if (running)
    {
        if (checkValidPtr(networkSend))
            networkSend->quit("networkSend␣quit␣by␣queue");
        if (checkValidPtr(networkReceive))
            networkReceive->quit("networkReceive␣quit␣by␣queue");
       // don't quit if the thread part is in the data
        pthread_mutex_lock(&accessingData);

        if (pthread_cancel(queueThread)) {
            pthread_mutex_unlock(&accessingData);
        quit("gucQueue␣pthread_cancel␣failed.", 1);
      }
        else
        {
            pthread_mutex_unlock(&accessingData);
            running = false;
          cleanup();
        }


        if (strlen(msg) > 0)
        {
            printf("\n");
          if (retval == 0) {
            printf("%s\n",(msg == NULL ? "" : msg));
          } else {
            fprintf(stderr, (msg == NULL ? "" : msg));
            fprintf(stderr, "\n");
          }
        }
    }
}

void gucQueue::update()
{
    IplImage* tempIplImage = NULL;

    switch (queueType)
    {
        case (gucQueueTypes::FILL_QUEUE_FROM_IPLIMAGE):
        case (gucQueueTypes::FILL_QUEUE_FROM_IPLIMAGE_TO_NETWORK):
        {
            if (cameraCapture)
            {
            // Get one frame
            if(!cvGrabFrame(capture)){              // capture a frame
                printf("Could␣not␣grab␣a␣frame\n");
                    exit(0);
                }
                tempIplImage=cvRetrieveFrame(capture);          // retrieve the
                    captured frame

                if( !tempIplImage )
                {
                    fprintf( stderr, "ERROR:␣gucQueue::update()␣-␣frame␣is␣null...␣:␣
                        camera␣%d\n", cameraID );
                }
                else
                {
                    // pushed the captured frame, no copy made
                    pushPtr(tempIplImage, true);

                    vector<callback_t >::iterator it;
                    for ( it=callbacks.begin() ; it < callbacks.end(); it++ )
                    {
                        if ((*it).callbackOwner == NULL)
                        {
                            (*it).callbackFunction(this, (*it).nextQueue, 0);
                        }
                        else
                        {
                            gucSendData* sd = (gucSendData*)(*it).callbackOwner;
                            sd->queueListener(this, (*it).nextQueue, 0);
                        }
                    }
                }
            }
            break;
        }
        case (gucQueueTypes::FILL_QUEUE_FROM_NETWORK_WITH_IPLIMAGE):
        case (gucQueueTypes::FILL_QUEUE_FROM_NETWORK_WITH_USER_DATA):
            // the user is waiting for this so pull the callback
            if (dataReceived)
            {
                dataReceived = false;

                vector<callback_t >::iterator it;
                for ( it=callbacks.begin() ; it < callbacks.end(); it++ )
                {
                    if ((*it).callbackOwner == NULL)
                    {
                        (*it).callbackFunction(this, (*it).nextQueue, 0);
                    }
                    else
                    {
                        gucReceiveData* rd = (gucReceiveData*)(*it).callbackOwner;
                        rd->queueListener(this, (*it).nextQueue, 0);
                    }
                }
            }
            break;

        case (gucQueueTypes::FILL_QUEUE_FROM_USER_IPLIMAGE_TO_NETWORK):
        case(gucQueueTypes::FILL_QUEUE_FROM_USER_DATA_TO_NETWORK):
        case(gucQueueTypes::FILL_QUEUE_FROM_USER_DATA):
            // the somebody (possibly the network) is waiting for this so pull the
                callback
            if (dataReceived)
            {
                dataReceived = false;

                if ((queueType == gucQueueTypes::FILL_QUEUE_FROM_USER_DATA_TO_NETWORK)
                    ||
                    (queueType == gucQueueTypes::
                        FILL_QUEUE_FROM_USER_IPLIMAGE_TO_NETWORK))
                {
                    vector<callback_t>::iterator it;
                    for ( it=callbacks.begin() ; it < callbacks.end(); it++ )
                    {
                        if ((*it).callbackOwner == NULL)
                        {
                            (*it).callbackFunction(this, (*it).nextQueue, 0);
                        }
                        else
                        {
                            gucSendData* sd = (gucSendData*)(*it).callbackOwner;
                            sd->queueListener(this, (*it).nextQueue, 0);
                        }
                    }
                }
            }
            break;
    }
}

void gucQueue::setRunning(bool r)
{
    running = r;
};

void gucQueue::setCameraCapture(bool c)
{
    cameraCapture = c;
};

int gucQueue::setCaptureProperty(int property_id, double value)
{
    return cvSetCaptureProperty(capture, property_id, value);
}

```

```
int gucQueue::addListener(gucNetworkBase* sd, gucQueue* next)
{
    callback_t newCallback;

    newCallback.callbackOwner = sd;
    newCallback.callbackFunction = NULL;
    newCallback.nextQueue = next;
    callbacks.push_back(newCallback);
    return callbacks.size();
}

int gucQueue::addListener(_callbackFunction, gucQueue* next)
{
    callback_t newCallback;

    newCallback.callbackOwner = NULL;
    newCallback.callbackFunction = _callbackFunctionPtr;
    newCallback.nextQueue = next;
    callbacks.push_back(newCallback);
    return callbacks.size();
}

int gucQueue::pushCopy(IplImage* img, bool updateRequired)
{
    IplImage* newImage = new IplImage(*img);
    return pushPtr(newImage, updateRequired);
}

int gucQueue::pushPtr(IplImage* img, bool updateRequired)
{
    if ((queueType == gucQueueTypes::FILL_QUEUE_FROM_NETWORK_WITH_USER_DATA) ||
        (queueType == gucQueueTypes::FILL_QUEUE_FROM_USER_DATA)||
        (queueType == gucQueueTypes::FILL_QUEUE_FROM_USER_DATA_TO_NETWORK))
    {
        fprintf( stderr, "ERROR: gucQueue::push() - Attempt to push data to an image 
            queue\n" );
    }
    pthread_mutex_lock(&accessingData);

    int result = -1;

    if (!img)
    {
        printf("gucQueue::push() with null IplImage\n");
        pthread_mutex_unlock(&accessingData);
        return result;
    }

    int s = items.size();
    while (s >= maxQueueSize)
    {
        items.pop();
        s = items.size();
    }

    item_t* newItem = new item_t;
    newItem->data = NULL;
    newItem->image = img;
    newItem->size = img->imageSize;
    newItem->type = gucQueueData::IPLIMAGE;


    items.push(newItem);
    dataReceived = updateRequired;

    result = items.size();

    pthread_mutex_unlock(&accessingData);

    return result;
}

int gucQueue::pushCopy(void *data, int size, bool updateRequired)
{
    //make a copy of the data to be stored so it is not lost
    void* dataCopy = malloc(size);
    memcpy(dataCopy, data, size);
    return pushPtr(dataCopy, size, updateRequired);
}

int gucQueue::pushPtr(void *data, int size, bool updateRequired)
{
    if ((queueType == gucQueueTypes::FILL_QUEUE_FROM_NETWORK_WITH_IPLIMAGE) ||
        (queueType == gucQueueTypes::FILL_QUEUE_FROM_IPLIMAGE)||
        (queueType == gucQueueTypes::FILL_QUEUE_FROM_IPLIMAGE_TO_NETWORK) ||
        (queueType == gucQueueTypes::FILL_QUEUE_FROM_USER_IPLIMAGE_TO_NETWORK))
    {
        fprintf( stderr, "ERROR: gucQueue::push() - Attempt to push image to an data 
            queue\n" );
    }

    pthread_mutex_lock(&accessingData);

    int result = -1;

    if (!data)
    {
        printf("gucQueue::push() with null data\n");
        pthread_mutex_unlock(&accessingData);
        return result;
    }

    item_t newItem;
    newItem.data = data;
    newItem.image = NULL;
    newItem.size = size;
    newItem.type = gucQueueData::USER_DATA;

    items.push(&newItem);
    dataReceived = updateRequired;

    result = items.size();

    pthread_mutex_unlock(&accessingData);

    return result;
}

int gucQueue::size()
{
    int size = 0;
    pthread_mutex_lock(&accessingData);
    size = items.size();
    pthread_mutex_unlock(&accessingData);
    return size;
};

void gucQueue::clear()
{
    pthread_mutex_lock(&accessingData);
    queue<item_t*>().swap(items);
    pthread_mutex_unlock(&accessingData);
};

void* gucQueue::popData()
{
    void* result = NULL;

    pthread_mutex_lock(&accessingData);
    if (items.size() == 0) {
        pthread_mutex_unlock(&accessingData);
        return NULL;
    }
    item_t* item = popFront();
    if (item)
    {
        result = item->data;
        items.pop();
        delete item;
    }
    pthread_mutex_unlock(&accessingData);
    return result;
}

IplImage* gucQueue::popImage()
{
    IplImage* result = NULL;

    pthread_mutex_lock(&accessingData);
    if (items.size() == 0) {
        pthread_mutex_unlock(&accessingData);
        return NULL;
    }
    item_t* item = popFront();
    if (item)
    {
        result = item->image;
        items.pop();
        delete item;
    }
    pthread_mutex_unlock(&accessingData);

    return result;
}

gucQueue::item_t* gucQueue::popFront()
{
    // !!!!! no mutex locking, responsibility of the calling function
    // !!!!! this is private so cannot be called by user
    item_t* dataCopy = NULL;
    if (!(items.empty()))
    {
        dataCopy = items.front();
    }
    return dataCopy;
};
```

Listing E.7: Queue Class Header.
Coded by Jayson Mackie research assistant for the Game Technology Group at Gjøvik University College

```
#ifndef _GUC_QUEUE_H_
#define _GUC_QUEUE_H_

#include "gucOCV.h"
#include "gucSendData.h"
#include "gucReceiveData.h"

#define _callbackFunction void (*_callbackFunctionPtr)(gucQueue*, gucQueue*,int)
#define _callbackMethod void (gucNetworkBase::*_callbackMethodPtr) (gucQueue*, gucQueue
    *,int)

typedef void (*_queueListenerFunctionType)(gucQueue*, gucQueue*,int);
typedef void (gucNetworkBase::*_queueListenerMethodType) (gucQueue*, gucQueue*,int);

using namespace std;

//============================================================================

class gucQueue {
private:
    int queueType;
    bool running;
    bool cameraCapture;

    int maxQueueSize;

    pthread_mutex_t      accessingData;
  pthread_t            queueThread;

    typedef struct {
        void* data;
        IplImage* image;
        int type;
        int size;
    } item_t;

    typedef struct {
        void *                      callbackOwner;
        _queueListenerFunctionType  callbackFunction;
        gucQueue*                   nextQueue;
    } callback_t;

    // for OpenCV modes
    CvCapture*           capture;
    int                  cameraID;

    // network objects
    gucSendData*    networkSend;
    gucReceiveData* networkReceive;
    bool            dataReceived;

    queue<item_t*>      items;

    vector<callback_t> callbacks;

    item_t* popFront();

    gucQueue();
    static void* threadInitialise(void* q);
    void threadLoop();
    void update();
    void cleanup();

public:

    gucQueue(int qType, const char* address = "", int cameraID = 0, bool delayedStart =
         false);

    void initialise(const char* address = "", int cameraID = 0);
    void quit(char* msg, int retval=0);

    int addListener(_queueListenerFunctionType, gucQueue* next = NULL);  // C functions
    int addListener(gucNetworkBase* owner, gucQueue* next = NULL);       // C++ methods

    // if the queue has an openCV camera attached to it
    int setCaptureProperty(int property_id, double value);

    //!!!! these do take a copy of the data
    int pushCopy(IplImage* img, bool networkPush = false);
    int pushCopy(void* data, int size, bool networkPush = false);
    //!!!! these do not take a copy of the data
    int pushPtr(IplImage* img, bool networkPush = false);
    int pushPtr(void* data, int size, bool networkPush = false);

    int size();
    void clear();
    void* popData();
    IplImage* popImage();

    void setRunning(bool r);
    void setCameraCapture(bool c);
};

#endif
```

Listing E.8: Class to send data to network.
Coded by Jayson Mackie research assistant for the Game Technology Group at Gjøvik University College

```

#include "gucOCV.h"
#include "gucQueue.h"
#include "gucSendData.h"


gucSendData::gucSendData(gucQueue* q)
{
    dataToSend = false;
    running = true;
};

void gucSendData::initialise(const char *serverIP, int port)
{

    strcpy(serverAddress, serverIP);
    serverPort = port;

    accessingData = PTHREAD_MUTEX_INITIALIZER;

  // run the streaming server as a separate thread
  if (pthread_create(&queueThread, NULL, &gucSendData::threadInitialise, (void *)this))
       {
    quit("pthread_create␣failed.", 1);
  }
};

/**
 * This is the streaming server, run as a separate thread
 * This function waits for a client to connect, and send the grayscaled images
 */
void* gucSendData::threadInitialise(void* sd)
{
    gucSendData* mySelf = reinterpret_cast<gucSendData *>(sd);
    mySelf->threadLoop();
    return NULL;
}

void gucSendData::threadLoop()
{
    update(); // needs to be split up to have an init and send part !!! TODO the bit
         below

    /*
    connect();
    while (running)
    {
        if (dataToSend)
        {
            sendData();
            dataToSend = false;
        }
    }
    */
}

void gucSendData::update()
{
  struct sockaddr_in server;

    long counter = 0;
    const long mod = 20;//00000000;

    printf("Server␣threadLoop␣started\n");

    bool startupComplete = false;
    bool startupFailed = false;
    while(!startupComplete) {

      // open socket
      if ((serversock = socket(PF_INET, SOCK_STREAM, 0)) == -1) {

            printf("Server␣threadLoop␣socket()␣failed\n");
        startupFailed = true;
      }
      else
            printf("Server␣threadLoop␣socket()␣success\n");

        if (startupFailed)
```

```
          {
            if (serversock)
                  closesocket(serversock);
          }

        // setup server's IP and port
        memset(&server, 0, sizeof(server));
        server.sin_family = AF_INET;
        server.sin_port = htons(serverPort);
        server.sin_addr.s_addr = INADDR_ANY;

        // bind the socket
        if (!startupFailed)
        {
            bool yes=true;
              setsockopt(serversock, SOL_SOCKET, SO_REUSEADDR, (char *)&yes, sizeof(int))
                  ;
            if (bind(serversock, (const sockaddr*)&server, sizeof(server)) == -1) {
                  printf("Server␣threadLoop␣bind()␣failed\n");
              startupFailed = true;
            }
              else
                  printf("Server␣threadLoop␣bind()␣success␣%s:%d\n",serverAddress,
                      serverPort);
          }

          if (startupFailed)
          {
            if (serversock)
                  closesocket(serversock);
          }

        // wait for connection
        if (!startupFailed)
        {
            if (listen(serversock, 10) == -1) {
                  printf("Server␣threadLoop␣listen()␣failed\n");
              startupFailed = true;
            }
            else
                printf("Server␣threadLoop␣listen()␣success\n");
        }

          if (startupFailed)
          {
            if (serversock) closesocket(serversock);
          }

        // accept a client
        if (!startupFailed)
        {
            if ((clientsock = accept(serversock, NULL, NULL)) == -1) {
                  printf("Server␣threadLoop␣accept()␣failed\n");
              startupFailed = true;
            }
            else
                  printf("Server␣threadLoop␣accept()␣success\n");
          }

          if (startupFailed)
          {
              printf("Server␣startup␣Failed,␣retrying␣in␣10s\n");
            if (serversock) closesocket(serversock);
            if (clientsock) closesocket(clientsock);
              Sleep(10000);
            startupFailed = false;
          }
          else
          {
              startupComplete = true;
          }
      }

      printf("Streaming␣server␣thread␣is␣sending␣data\n");
      /* start sending images */
      while (running)
      {
          int bytes = 0;
        int jpegSize = 1;
      int sending = 0;
          int imageSize = 0;

          if (dataToSend)
          {
          // send the frame, thread safe
          pthread_mutex_lock(&accessingData);

              // jpeg getting sent to compress it first
              if (SERVER_SEND_JPEG)
              {
                  /*
                  jpegSize = RGB24_to_jpeg(jpegBuffer, JPEG_BUFFER_SIZE, data,
                      CAMERA_WIDTH, CAMERA_HEIGHT, JPEG_QUALITY);
                  //printf("*");
                  //printf("Encoded image from %d to %d\n", CAMERA_IMAGE_SIZE, jpegSize);
                  //FILE *jpg;
                  //jpg = fopen ( "test.jpg" , "wb" );
                  //fwrite (jpegBuffer , 1 , jpegSize , jpg );
                  //fclose (jpg);

              if (jpegSize  && rarStreaming::frameTimer.time_m( ) > rarStreaming::
                  millisecondsPerFrame )
              {
                rarStreaming::frameTimer.setStartTick( );

                      sending = sizeof(int);
                      Sendall(clientsock, (const unsigned char *)&jpegSize, &sending);
                      sending = jpegSize;
                      bytes = Sendall(clientsock, (const unsigned char *)jpegBuffer, &
                          sending);
                }
                imageSize = jpegSize;
                if (imageSize > 100000) printf("oops - big file");
                  */
              }
              // just yuv422 getting sent
            else
          {
                  int sentBytes = 0;
                  if ((dataType == 0) && checkValidPtr(imagePtr))//((unsigned int)
                      imagePtr != 0xcdcdcdcd))
                  {
                      int iplImageSize = sizeof(IplImage);
                      int pixelSize    = imagePtr->imageSize;
```

```
                    int IplImageFlag = -1;
                    int sending = sizeof(int);

                    sentBytes = Sendall(clientsock, (const char *)&IplImageFlag, &
                        sending);
                    //if (sentBytes == 0)
                    //      printf("\nSend flag: %d\n", IplImageFlag);

                    if (pixelSize == globalImageSize)
                    {
                        sentBytes = Sendall(clientsock, (const char *)imagePtr, &
                            iplImageSize);
                        //printf("Send header: %d of %d   -   %.1f%%\n", sentBytes,
                            iplImageSize, sentBytes/(double) iplImageSize*100);

                        sentBytes = Sendall(clientsock, (const char *)(imagePtr->
                            imageData), &pixelSize);
                    //printf("Send image:  %d bytes\n", pixelSize );
                    }
                    else
                    {
                        printf("No␣image␣to␣send.\n");
                    }

                }
            }

            dataToSend = false;

        pthread_mutex_unlock(&accessingData);

            counter++;
            if (counter%mod == 0)
            {
                printf(".");
            }

            /*
            // if something went wrong, restart the connection
            if (sending != imageSize)
            {
              printf("threadLoop attempt reopen, bytes = %d of %d.\n", bytes, imageSize
                  );
             // printf("threadLoop connection closed.\n");
             if (clientsock) close(clientsock);

              if ((clientsock = accept(serversock, NULL, NULL)) == -1) {
                    printf("threadLoop accept() failed - reopened\n");
              }
              else
              {
                    printf("threadLoop accept() success - reopened\n");
              }
            }
*/
        }
        else
        {
            Sleep(2);
        }
    }

    fflush(stdout);
    printf("End␣of␣server␣loop?!\n");
}

// the data sender has to listen to the queue to know when something is ready to go
void gucSendData::queueListener(gucQueue* q, gucQueue* output, int type)
{
    dataToSend = true;
    dataType = type;
    static IplImage* tempImage = NULL;
    tempImage = q->popImage();
    if ((tempImage == NULL) || (tempImage->imageSize == 0))
        return;

    if (type == 0)
    {
        imagePtr = tempImage;
        dataPtr  = imagePtr->imageData;
    }
    else
    {
        imagePtr = NULL;
    }

    if (showSendingImage)
        cvShowImage("gucSendData␣-␣prior␣to␣sending", tempImage);
}

void gucSendData::cleanup()
{
  pthread_mutex_destroy(&accessingData);
}

/**
 * this function provides a way to exit nicely from the system
 */
void gucSendData::quit(char* msg, int retval)
{


    if (running)
    {
        pthread_mutex_lock(&accessingData);

        if (pthread_cancel(queueThread)) {
            pthread_mutex_unlock(&accessingData);
        quit("gucSendData␣pthread_cancel␣failed.", 1);
      }
        else
        {
            pthread_mutex_unlock(&accessingData);
            running = false;
          cleanup();
        }

        if (strlen(msg) > 0)
        {
            printf("\n");
          if (retval == 0) {
            printf("%s\n",(msg == NULL ? "" : msg));
          } else {
            fprintf(stderr, (msg == NULL ? "" : msg));
```

```
                fprintf(stderr, "\n");
            }
        }
    }
}
```

Listing E.9: Sending Class Header.
Coded by Jayson Mackie research assistant for the Game Technology Group at Gjøvik University College

```
#ifndef _GUC_SEND_DATA_H_
#define _GUC_SEND_DATA_H_

#include <winsock.h>

#include "gucOCV.h"
#include "gucNetworkBase.h"

class gucSendData : public gucNetworkBase {
private:

    pthread_mutex_t      accessingData;
  pthread_t            queueThread;

    void*            dataPtr;
    IplImage*        imagePtr;
    bool             dataToSend;
    int              dataType;
    bool             running;

    static void* threadInitialise(void* q);
    void threadLoop();
    void update();
    void cleanup();

public:
    gucSendData(gucQueue* q);
    void initialise(const char *serverIP, int port);
    void queueListener(gucQueue* q, gucQueue* output, int type = 0);
    void quit(char* msg, int retval=0);

};


#endif
```

Listing E.10: Class to receive data from network.
Coded by Jayson Mackie research assistant for the Game Technology Group at Gjøvik University College

```

#include "gucOCV.h"
#include "gucQueue.h"
#include "gucReceiveData.h"



gucReceiveData::gucReceiveData(gucQueue* q)
{
    waitingQueue = q;
    dataReceived = false;
    running = true;
};

void gucReceiveData::initialise(const char *serverIP, int port)
{
    strcpy(serverAddress, serverIP);
    serverPort = port;

    accessingData = PTHREAD_MUTEX_INITIALIZER;

  // run the streaming server as a separate thread
  if (pthread_create(&queueThread, NULL, &gucReceiveData::threadInitialise, (void *)
      this)) {
    quit("pthread_create␣failed.", 1);
  }
};

/**
 * This is the streaming server, run as a separate thread
 * This function waits for a client to connect, and send the grayscaled images
 */
void* gucReceiveData::threadInitialise(void* sd)
{
    gucReceiveData* mySelf = reinterpret_cast<gucReceiveData *>(sd);
    mySelf->threadLoop();
    return NULL;
}


void gucReceiveData::threadLoop()
{
    update(); // needs to be split up to have an init and send part !!! TODO the bit
        below

    /*
    connect();
    while (running)
    {
        if (dataReceived)
        {
            receiveData();
            dataToSend = false;
        }
    }
    */
}


void gucReceiveData::update()
{
  struct sockaddr_in server;

    printf("Streaming␣client␣thread␣is␣running\n");
    fflush(stdout);

  // open socket
  while ((serversock = socket(PF_INET, SOCK_STREAM, 0)) == -1) {
    //quit("socket() failed", 1);
    printf("Client␣No␣Socket␣yet,␣waiting␣2s\n");
    Sleep(2000);
```

```
}
printf("Client␣socket()␣success\n");

/* setup server parameters */
memset(&server, 0, sizeof(server));
server.sin_family = AF_INET;
server.sin_addr.s_addr = inet_addr(serverAddress);
server.sin_port = htons(serverPort);

/* connect to server */
while (connect(serversock, (struct sockaddr*)&server, sizeof(server)) < 0) {
  //quit("connect() failed.", 1);
  printf("Client␣Cannot␣connect␣%s:%d,␣waiting␣2s\n",serverAddress,serverPort);
  Sleep(2000);
}
printf("Client␣connect()␣success␣%s:%d\n",serverAddress,serverPort);

  fd_set socks;
  FD_ZERO(&socks);
FD_SET(serversock,&socks);

  while(running)
  {
    int sizeIncoming = 0;
      int bytes, received;

      if (FD_ISSET(serversock,&socks))
      {
      // get data type
      bytes = 0;
      if (running && (bytes = recv(serversock, (char *)&sizeIncoming, sizeof(int), 0)
          ) == -1) {
        quit("gucReceiveData␣recv␣failed␣on␣type\n");
      }

          if (sizeIncoming == -1)
          {

              IplImage* networkImage = new IplImage();

              //get the IplImage structure, this has no image data
          bytes = 0;
          if (running && (bytes = recv(serversock, (char *)networkImage, sizeof(
              IplImage), 0)) == -1) {
            quit("gucReceiveData␣recv␣failed␣on␣image␣header\n");
          }

              int bufferSize = networkImage->imageSize;

              if (bufferSize == globalImageSize)
              {
                  // get the image data
              // get image
                  IplImage* tempImage = cvCreateImage(cvSize(networkImage->width,
                      networkImage->height), IPL_DEPTH_8U, 3);
                  char* sockdata = tempImage->imageData;
              bytes = 0;
              for (received = 0; received < bufferSize; received += bytes) {
                if (running && (bytes = recv(serversock, sockdata + received,
                    bufferSize - received, 0)) == -1) {
                  quit("gucReceiveData␣recv␣failed", 1);
                }
                }

                    waitingQueue->pushPtr(tempImage, true);
                }
            }
        }
    }
    quit("End␣of␣client␣loop?!\n");
}


// the data sender has to listen to the queue to know when something is ready to go
void gucReceiveData::queueListener(gucQueue* q, gucQueue* output, int type)
{

}



void gucReceiveData::cleanup()
{
  pthread_mutex_destroy(&accessingData);
}
/**
 * this function provides a way to exit nicely from the system
 */
void gucReceiveData::quit(char* msg, int retval)
{
    if (running)
    {
        // don't quit if the threaded part is playing with the data
        pthread_mutex_lock(&accessingData);

        if (pthread_cancel(queueThread)) {
            pthread_mutex_unlock(&accessingData);
        quit("gucReceiveData␣pthread_cancel␣failed.", 1);
      }
        else
        {
            pthread_mutex_unlock(&accessingData);
            running = false;
          cleanup();
        }

        if (strlen(msg) > 0)
        {
            printf("\n");
          if (retval == 0) {
            printf("%s\n",(msg == NULL ? "" : msg));
          } else {
            fprintf(stderr, (msg == NULL ? "" : msg));
            fprintf(stderr, "\n");
          }
        }
    }
}
```

Listing E.11: Receiving Class Header.
Coded by Jayson Mackie research assistant for the Game Technology Group at Gjøvik University College

```
#ifndef _GUC_RECEIVE_DATA_H_
#define _GUC_RECEIVE_DATA_H_

#include <winsock.h>

#include "gucOCV.h"
#include "gucNetworkBase.h"

class gucReceiveData : public gucNetworkBase {
private:

    pthread_mutex_t     accessingData;
  pthread_t          queueThread;

    gucQueue*       waitingQueue;

    bool            dataReceived;
    bool            running;

    static void* threadInitialise(void* q);
    void threadLoop();
    void update();
    void cleanup();

public:
    gucReceiveData(gucQueue* q);
    void initialise(const char *serverIP, int port);
    void queueListener(gucQueue* q, gucQueue* output, int type = 0);
    void quit(char* msg, int retval=0);
};


#endif
```

Listing E.12: Networkbase Class. Sends tcp packet down socket.
Coded by Jayson Mackie research assistant for the Game Technology Group at Gjøvik University College

```

#include "gucOCV.h"


/*
====================
Sendall
Sends a tcp packet buf down socket s of length len
====================
*/
int gucNetworkBase::Sendall(int s, const char *buf, int *len)
{
    int total = 0;        // how many bytes we've sent
    int bytesleft = *len; // how many we have left to send
    int n;

    while( total < *len ) {
        n = send( s, buf+total, bytesleft, 0 );
        if ( n == -1 ) { break; }
        total += n;
        bytesleft -= n;
    }

    *len = total; // return number actually sent here

    return n == -1 ? -1 : 0; // return -1 on failure, 0 on success
}
```

Listing E.13: Networkbase Class Header.
Coded by Jayson Mackie research assistant for the Game Technology Group at Gjøvik University College

```
#ifndef _GUC_NET_BASE_H_
#define _GUC_NET_BASE_H_

#include "gucOCV.h"

class gucNetworkBase
{
protected:
    int     serversock, clientsock;
  char    serverAddress[255];
  int     serverPort;
public:
    virtual void queueListener(gucQueue* q, gucQueue* output, int type = 0) = 0;
    int Sendall(int s, const char *buf, int *len);

};

#endif
```

Listing E.14: Class singleton to start up winsock.
Coded by Jayson Mackie research assistant for the Game Technology Group at Gjøvik University College

```
#ifndef _GUC_NET_BASE_H_
#define _GUC_NET_BASE_H_

#include "gucOCV.h"

class gucNetworkBase
{
protected:
    int     serversock, clientsock;
  char    serverAddress[255];
  int     serverPort;
public:
    virtual void queueListener(gucQueue* q, gucQueue* output, int type = 0) = 0;
    int Sendall(int s, const char *buf, int *len);

};

#endif
```

F Matlab Source Code

Listing F.1: Matlab code evaluating the χ^2 test

```
% Function that evaluates the chi^2 test

% For each question evaluate the test
for r=1:9
    % in case of experiment 1
    % for r=1:10

    % in case of eye contact perception question
    % for r=7:7


    % Data1 and Data2 contain the software and hardware data

     data(1,:) = Data1(r,:);
     data(2,:) = Data2(r,:);

     % Data1 and Data2 contain the enabled eye contact and disabled eye
     % contact data
     % data(1,:) = Data1(r,:);
     % data(2,:) = Data2(r,:);

     % Data1 and Data2 contain the software and hardware data
     % perform test between bins 4 and 5
     % data(1,1)= Data1(r,4);
     % data(1,2)= Data1(r,5);

     % data(2,1)= Data2(r,4);
     % data(2,2)= Data2(r,5);

     % Calculate the totals per row and per column

    totalRow(1,1) = data(1,1) + data(1,2) + data(1,3) + data(1,4) + data(1,5);
    totalRow(1,2) = data(2,1) + data(2,2) + data(2,3) + data(2,4) + data(2,5);

    totCol(1,1)= data(1,1)+data(2,1);
    totCol(1,2)= data(1,2)+data(2,2);
    totCol(1,3)= data(1,3)+data(2,3);
    totCol(1,4)= data(1,4)+data(2,4);
    totCol(1,5)= data(1,5)+data(2,5);

    % Experiment 1
     % totalRow(1,1) = data(1,1) + data(1,2) + data(1,3);
     % totalRow(1,2) = data(2,1) + data(2,2) + data(2,3);

     % totCol(1,1)= data(1,1)+data(2,1);
     % totCol(1,2)= data(1,2)+data(2,2);
     % totCol(1,3)= data(1,3)+data(2,3);

    % Eye contact perception question
     % totalRow(1,1) = data(1,1) + data(1,2);
     % totalRow(1,2) = data(2,1) + data(2,2);
```

```

 % totCol(1,1)= data(1,1)+data(2,1);
 % totCol(1,2)= data(1,2)+data(2,2);


 % Get the overall total

total = totalRow(1,1) + totalRow(1,2);

 % Calculate the expected frequencies for each cell in the table

for i=1:2
    for j=1:5
 %  for j=1:3
 %  for j=1:2
        dataExp(i,j)=( totalRow(1,i)*totCol(1,j) ) / total;
    end
end

 % Find the squared difference between the observed and expected values

for i=1:2
    for j=1:5
 %  for j=1:3
 %  for j=1:2
        dataDiff(i,j)=((data(i,j) - dataExp(i,j))^2) / dataExp(i,j);

        if (dataExp(i,j)==0)
            dataDiff(i,j)=0;
        end
    end
end

 % Sum all the differences to get the chi value

sum=0;
for i=1:2
    for j=1:5
 %  for j=1:3
 %  for j=1:2
        sum=sum+dataDiff(i,j);
    end
end

chi = sum;

 % Critical values for chi correspond to a level of risk

 % Critical values corresponding to a degree of freedom of 4

    critval = [7.779     9.488     11.143     13.277     18.467];

 % Experiment 1:
 % Critical values corresponding to a degree of freedom of 2
 % critval = [4.605     5.991     7.378     9.210     13.816];

 % Eye contact perception question:
 % Critical values corresponding to a degree of freedom of 1
 % critval = [2.706     3.841     5.024     6.635     10.828];

    risk    = [0.1        0.05      0.025      0.01      0.001];

    % If chi is greater or equal than critical value, then null hypothesis
    % is rejected and the corresponding probability is stored

    hyp= 0;
    prob= NaN;
    for c=1:5
        if chi >= critval(1,c)
            hyp=1;
            prob=risk(1,c);
        end
    end

    % Store the chi values, null hypotheses and any probabilities in a
    % results table

    results(r,1) = chi;
    results(r,2)= hyp;
    results(r,3) = prob;
end
```