NTNU – Trondheim
Norwegian University of
Science and Technology

Identify level of interest of people to a object using kinect

Christoffer Lohne

Master of Science in Electronics
Submission date: February 2015
Supervisor: Andrew Perkis, IET
Co-supervisor: Jordi Puig, IET

Norwegian University of Science and Technology
Department of Electronics and Telecommunications

Summary

This paper explores whether it is possible to detect peoples interest toward an object by using a kinect camera. The kinect camera has a depth and a color camera and the most interesting part to use is the depth camera..The depth camera is interesting due to that each pixel contain the distance from the camera to the object. With a normal color camera each pixel contain the color value. The main focus in this master thesis have been to develop a system that can detect whether multiple people are looking toward an object by using kinect camera. The system consists of a kinect SDK and a mac computer running the program.

The paper starts by giving an introduction to the software-tools used and give the specifications of the hardware. Blob detection and face detection is the two computer vision methods used. The basic contents behind the methods are described in the following chapters. Blob detection is used on the frames from the depth camera. Face recognition is used on the frames from the color camera. In this paper the openFrameworks library is used. The most important library from openFrameworks is the OpenCV and kinect library. The OpenCV library has algorithms for face detection and blob detection. How these two methods are integrated to the system is described in the report.

The main challenge has been to implement the system and integrate the two methods. The system created is able to detect objects contour using the depth image. Then do a face detection on the detected object. This is to determine if the detected object is a person looking towards the camera. The system is tested on a test described in this paper. This test is designed to find out if the system is able to do what it's intended to on a simple scenario and more advanced scenarios. The test is also designed to find the limitations of the system. The results from the test is documented in the report and discussed.

Sammendrag

Denne rapporten utforsker mulighetene ved å bruke et kinect kamera til å observere folks interesse til et objekt. Et kinect kamera består av et fargekamera og et dybdekamera, hvor det er dybdekameraet som er mest interessant. I et dybdebilde inneholder hver piksel avstanden fra objektet til kameraet. I et fargebilde inneholder pikslene en fargekode. Hovedfokuset i masteroppgaven har vært å lage et system som klarer å oppdage hvorvidt en eller flere personer ser mot et objekt ved å bruke et kinect kamera. Systemet som har blitt laget består av et Microsoft kinect SDK kamera og en Mac datamaskin.

Rapporten starter med å gi en introduksjon til programvareverktøyene og maskinvarene som har blitt brukt i rapporten. Blob-detektering og ansikts-detektering er de to metodene som er brukt i rapporten. De grunnleggende teoriene til metodene er så beskrevet i rapporten. Blob-detektering er brukt på bildene fra dybdekameraet. Ansikt-detektering er brukt på bildene fra fargekameraet. I rapporten er det blitt brukt openFramework sine biblioteker. Det viktigste biblioteket er OpenCV biblioteket og kinect biblioteket. OpenCV biblioteket inneholder algoritmene for ansikts-detektering og blob-detektering. Hvordan de to metodene er integrert i systemet er forklart i rapporten.

Hovedutfordringen var å implementere systemet og hvordan intrigeringen av de to metodene ble gjort. Systemet som er blitt laget er i stand til å detektere konturen til objekter ved å bruke dybdebilde. Deretter gjør systemet ansikts-detektering på de oppdagede objektene. Dette er for å identifisere om det oppdagede objekt er en person som ser mot kameraet. Systemet er verifisert ved en test beskrevet i rapporten. Testen er laget for å finne ut om systemet er i stand til å detektere personer i enkle og mere avanserte scenarioer. Resultatene fra testen er dokumentert og diskutert i rapporten.

Preface

Commercial objects are shown on displays around the world. The reason for this is to attract attention from people toward the objects. An object can be a commercial sign, a new clothing collection, a screen showing the newest car video commercial, to mention a few. There is a lot of useful information in measuring the level of interest people have toward an object. This information can indicate how many people that have been exposed to the commercial object. This can be an indication on how successful the commercial is. The technology and the quality of cameras are at a level where it can be possible to detect the level of people's interest in a specific object. This project is aimed at exploring whether it's possible to use a kinect camera to detect the level of interest. The concept is to measure whether people are looking toward the object, and for how long. In this project the object is placed directly behind the camera.

One of the main motivators behind the project is the quality of free video analysis libraries that is available for academic and commercial use. Maybe the biggest of them is OpenCV. There are movies where this has been implemented and the results look very promising. Another interesting motivator is the capability of the Microsoft kinect camera. The Microsoft kinect camera contain a bunch of sensors, the most interesting being the depth camera. The depth camera has a remarkable possibility to capture the shapes of objects. The combination of OpenCV library and a Microsoft kinect camera needs to be investigated for a possible solution.

I chose the project because I wanted to get a better understanding of sensor technology. There are a lot of possibilities available. I am especially interested in the potential of combining embedded systems with sensors. This project is not going to include the embedded system part, but maybe this is something for the future. My supervisor introduced me to OpenFramework. OpenFramework was quite easy to understand, and had some good examples on how to use OpenCV and kinect camera.

Table of Contents

List of Figures

List of Tables

Abbreviations

NTNU	Norges teknisk-naturvitenskapelige universitet
MIT License	Massachusetts Institute of Technology License
osX	Mac osX is a operating system used on Mac computers.
OpenCV	Open Source Computer Vision
CV	Computer Vision
RGB	Read Green Blue

Table 1: Abbreviations

1 Introduction

This paper is about exploring the possibilities of using a combination of a color camera and a depth camera to measure the level of interest people show in an object. The goal is to be able to count the number of persons that have shown interest in an object. The solution needs to be able to recognize groups of people and distinguish each individual person.

Being able to measure the level of interest in an advertisement has high commercial value. The first idea about the project was trigged by a request from Dogu AS about their Salesscreen product. They wanted to count the number of persons and measure the level of engagement in front of the screen. This idea was brought to NTNU as a possible master thesis. The thesis was first discussed with Kjetil Svarstad, professor at NTNU. However, this topic was not within his domain of expertize, so his recommendation was to talk to Andrew Perkins. Andrew Perkins saw the possibility for this to become a master thesis. He wanted me to look into the possibility of using a depth camera.

The project started with a study of image and video algorithms. I got clues from my professor and supervisor on where to start the self study. This resulted in a theory that a combination of edge recognition and face detection could give a solution to the problem. Methods where identified and the next stage of the project started. I needed to find a way to implement this into a system. The recommendation from my supervisor was to use openFramework. OpenFramework is widely cross-platform oriented and has libraries containing algorithms for the methods that were needed. The implementation of the system was done on my Macbook pro. OpenFramework toolkit is easy to understand and there are plenty of good learning examples and introductions on their home page. When the code for the system was completed, the planning of the test activities started. Kinect one SDK is used in this project. The range and field of view of the depth camera is not optimal. This limited the possibilities for testing. The test created for the system is going to prove that the system can perform basic functions in a simple scenario. The more advanced tests are going to simulate more realistic scenarios.

The report is partitioned as follows:

- Ch. 1 - Introduction.
- Ch. 2 - Software-Tools. Brief introduction to the software-Tools used in the project.
- Ch. 3 - Hardware. Information about what hardware is used in the project and specifications.
- Ch. 4 - Computer Vision Methods. Basic theory about the computer vision Methods used and references.
- Ch. 5 - Implementation. Describing how the code for the system is

implemented. Explaining how the different computer vision methods are integrated.

- Ch. 6 - System Test. Describing how the system is going to be tested. Each step of the tests are described and the rational behind the test.
- Ch. 7 - Results. The results from the test.
- Ch. 8 - Discussion. Discussion of the results and an analysis of the system.
- Ch. 9 - Conclusion. A conclusion and ideas for further work.

1.1 Problem definition

Use a kinect camera to count the number of persons that have shown interest in an object. The system needs to be able to recognize groups of people and distinguish each individual person.

2 Software-Tools

This chapter is going to give an introduction to the software-tools used in this project. To get access to the desirer algorithms OpenFrameworks was used. OpenFramework has all of the libraries that were needed for the creation of this system. All of the programming was done in osX using Xcode.

2.1 OpenFrameworks

OpenFrameworks is an open source software toolkit. It is widely cross-platform oriented. It's distributed under the MIT License, opening it for private or public use. OpenCV is the tool that is used in this paper. It is further described in ***Chapter 2.2***. The reason for using openFrameworks is first of all that it is a framework that contains OpenCV and supports OSX. Further, the community that uses openFramework has a very active user group. There is a lot of information and examples that are shared between the communities. This makes it quite easy to find relevant examples and other comparable applications. OpenFrameworks consist of the following software tools:

- OpenGL,GLEW, GLUT, libtess2 and cario for graphise.
- rtAudio, PortAudio, OpenAL and Kiss FFT or FMOD for audio input, output and analysis.
- FreeType for fonts.
- FeeImage for image saving and loading.
- Quicktime, Gstreamer and videoInput for video playback and grabbing.
- Poco for a vaiety of utilities.
- OpenCV for computer vision.
- Assimp for 3D model loading.

2.2 OpenCV

OpenCV is an open source computer vision and machine learning software library. It has C++, C, Python, Java and MATLAB interfaces and supports Windows, Linux, Android and Mac OS. OpenCV consists of more than 2500 algorithms for Computer vision and machine learning. The fact that this CV library is free to use makes it attractive for projects and Startup Companies. From OpenCV homepage, well-established companies use this library. This indicates that the algorithms are at a high level of quality.

3 Hardware

This chapter will give an introduction to the hardware used in this paper. The specifications of the hardware will be listed.

3.1 Kinect camera

The sensor used in this paper is a Microsoft kinect one SDK sensor. This sensor was made as an extension to the gaming console X-box 360. This camera made it possible to use the person as an input to the gaming console instead of a controller. The kinect camera use a method called Skeleton tracking to detect the movements of the person. This system is able to detect up to two persons. The SDK version of the kinect Camera is a normal kinect camera with an extra power cable that makes it possible to use the kinect camera with a computer. The normal kinect camera draws its power from the X-box gaming console. The information is gathered from reference ***number [5]***.

3.1.1 Depth camera

The depth camera consists of an IR Depth Sensor and an IR Emitter. There is an IR Emitter in the kinect sensor that emits infrared light beams. The IR depth sensor detects the reflected infrared light. The reflected beams are converted into depth information measuring the distance between an object and the sensor. Specifications for the depth camera:

- Depth camera resolution 320 x 240
- 30 Frames per second
- Max Depth Distance 4.5 M
- Min Depth Distance 40 cm

3.1.2 Color camera

The Color camera consists of a RGB camera that stores three channel data in a 1280 x 960 resolution.

3.2 Computer

In this paper the code was written and tested on a Macbook pro. The mac runs mac osX operating system. The following specifications to the computer:

- CPU - Intel Core i7 2 GHz
- Memory - 8 GB

- Disk – 125 SSD
- Operating system – OS X 10.9.5

4 Computer vision methods

Humans only use a short amount of time to look at a picture and locate the people in the picture. We can read their facial expression to give us an indication of their level of interest in the object. To do this with a computer is not a trivial task. The approach to solve this task can be to first detect the shapes within the picture. This can be done by using blob detection, see ***chapter 4.1*** for further information. When the shapes within the picture are detected it is interesting to analyze the shapes to figure out if the shape is a human. If the shape is human, it needs to have a face. The limitation is that the person needs to face the camera. Face detection algorithms can be used to solve this problem, see ***chapter 4.2*** for further information.

4.1 Blob detection

Blob detection refers to a mathematical method used in computer vision to detect areas in an image where there is little change with a predetermined factor. The predetermined factor can be color or brightness. The area with little change is called the blob. There can be multiple blobs in a picture. The most interesting part of the picture are the areas with some form of similarity. Blob detection can identify regions of high interest. Then, further analysis can be applied to the regions of interest. This is from reference ***number [1] and [2].***

On a normal picture or a video stream, the edges of objects are not easily detected. This is due to the fact that it is very difficult to distinguish the object from the surroundings. By using a depth image, this becomes a much more trivial task. Instead of using the color or brightness as a measure of similarity, the physical depth can be used, as most objects have distinct depth features. This means that detecting objects becomes much easier.

4.2 Face detection

Face detection methods locates the position and the size of the faces. This is done by having a database with patterns that the algorithm uses on the image. Using Haar featured-based cascade classifiers is an effective object detection method proposed by Paul Viola and Michael Jones, ***reference number [3]***. The database contains several features that need to be trained on several images, some with faces and other whit out faces. In ***Illustration 1: Haar features there are some examples*** of features used in Haar object detection. ***(A)*** is a two-rectangle feature, ***(B)*** a three-rectangle feature, and ***(C)*** a four-rectangle feature. The number of possible feature combinations is high. How important each feature is is different, and some are almost pointless. This means that features can be given different weights that correspond to how important it is. The training of the database can end when acceptable level error rate or required number of features are found.

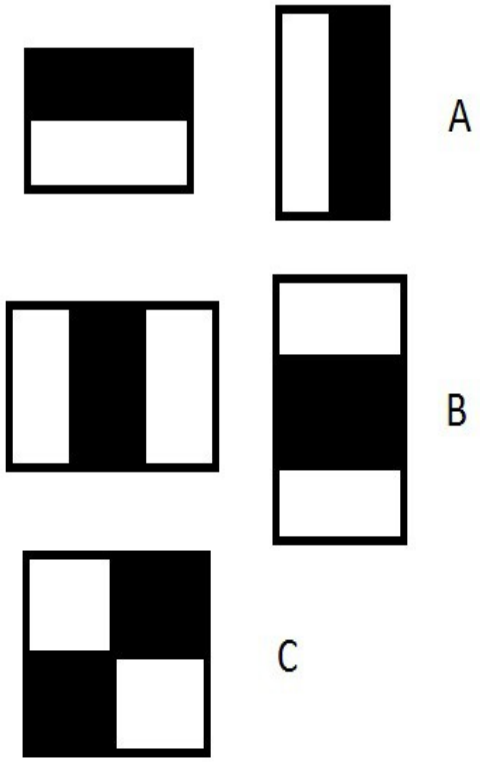

Illustration 1: Haar features example

5 Implementation

In this chapter the implementation of the system will be described. The code is written in c++ using the openFramework as a toolkit. The code can be found in ***Appendix B – Program code.***

5.1 Brief introduction of the system

The system is built up by three phases. Phase one is an initialization of the system where the system connects to the kinect camera to get the color and depth video feed. In addition the system is setting up the initial values for the blob detection and loading the Haar-cascade features. In phase two the system analyzes the video feeds. In this part of the code the blob and face detection are done. The last phase is presenting the results. This is done by displaying the two video feeds and showing the results from the analysis. The different phases can be found in the three main openFrameworks functions:

- Initialization in ofApp::setup()
- Analysis in ofApp::update()
- Displaying the results in ofApp::draw()

5.2 Initialization of the system

In this part of the code the system initializes the kinect camera, the blob detection and loads the Haar-cascade features. In addition, the initialization of variables and allocating images are done in this stage. The initialization of the kinect is done by using functions from the openFrameworks kinect library. The values set in the initialization phase are a result of testing different values. If there is a change in the set up, the values need to be reevaluated. For the kinect part this is the near and far threshold for the depth image. For the blob detection, different maximum number of blobs, minimum and maximum blob size. The face detection can be loaded with 4 different Haar-cascade features library.

5.3 Implementation of the analysis

There are two different analyzes done in this system. The first analysis is the blob detection. This function looks for the contours of objects by looking at similarity in the image. The function returns to where the contours are on the image, and collects information about the size and number of contours found. In the system this is done by first converting the depth image frame to a gray-scale image, where the brightness indicates the depth information, CvAnd function is used to find the pixels which are in union between the near and far field. The depth image is now ready to be sent to the blob detection algorithm. This system uses the ofxCvContourFinder class to perform the

blob detection. This class is from the openFramework OpenCV library.
When the blobs are detected the next stage is to determine if there are any faces in the blob. To do this, the system first needs to create an image from the blob. This is done by using the shape information of the blobs detected. This information is used to extract the area where the blob is detected from the gray-scale image. This image is called blobImg and the extraction is done by the function setPixelsSubRegion. The blob image is now ready to be sent to the face detection algorithm. The ofxCvHaarFinder class is used to perform the face detection. This class is from the openFramework OpenCV library. The number of faces detected in a blob is stored in a vector blobNrFaces. The information about the location of the detected faces is stored in a vector blobFaceArray. The blobImg is stored in a vector blobImgArray.

5.4 System output

In Illustration 2: System output example is an illustration of the output from the system. The image in the upper left part of the illustration is the depth image with the detected blob rectangle. To the right is the color camera gray-scaled image. Under the depth image is the blob image. If there are more blobs, they are shown to the right of the first blob.

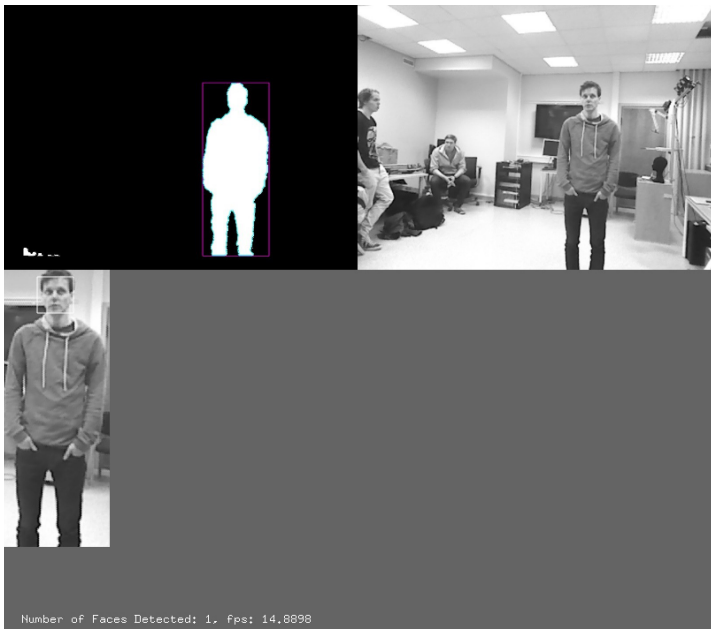

Illustration 2: System output example

6 System test

The goal of the system is to detect the number of people looking toward the camera. To demonstrate that the system is fulfilling this goal, several tests are designed. Some of the tests are scenarios to be performed by people. The system is going to be tested on multiple persons standing alone or in groups. Others are aimed at testing the limitations of the system: the range the system is able to detect objects, and at which angle a person needs to look towards the camera to be detected. The following sub-chapters explain the tests in more detail.

6.1 Initial test

Before doing the scenarios, it's required to verify that the system is able to detect an object. This is done by one person walking around in the room, and another person observing the outputs from the system. The objective of the test is to demonstrate that the system is able to detect an object by depth image.

6.2 Test 1: Basic functions

In this test there is going to be a simple scenario to test the basic tasks that the system need to be able to handle. The basic task is to find an object, and then detect if the object is a person looking towards the camera. The scenario consists of first only one person. The system needs to detect the person, and then determine if the person is looking towards the camera. Then another person enters the room, and the system needs to detect the other person too, and determine whether that person is looking towards the camera as well. The two persons are going to change between looking towards the camera and standing perpendicular to the camera. The steps in the scenario are as follows:

1. The first person stands right in front of the camera. The person looks towards the camera. There are no other objects in the room.
2. The first person turns 90 degrees from the camera.
3. A new person enters the room. The new person walks towards the first person and stops one meter away from him. The new person turns and looks towards the camera. The first person does not change his position.
4. The first person turns towards the camera.

6.3 Test 2: Basic test with group

In this test, the scenarios are a bit more advanced. This is to test the systems ability to handle people in groups. The scenario begins with two persons standing in front of the camera and looking towards each other as if they are having a conversation. Then the two persons' attention is drawn towards the camera. They move close together and look towards the camera. Then the system needs to be able to still detect how many persons

are looking towards the camera, even when they are standing close to each other. Then a new person enters the room. This is to test if the system can still detect a new person when there is already a group of persons in the room. This new person first stands alone, and then joins the group. The system needs to be able to detect how many people are looking towards the camera regardless of whether they are standing alone or in a group, even if they are first alone and then later in a group. The steps in the scenario are as follows:

1. Two persons stand in front of the camera with normal distance between each other as if they were having a conversation.
2. The two persons move closer to each other, shoulder to shoulder, and look toward the camera.
3. A new person enters the room. He walks towards the closest person and stops one meter from him. The new person turns and looks towards the camera.
4. The person to the right in the group turns 90 degrees from the camera.
5. The new person joins the group, standing shoulder to shoulder with the closest person. He turns and looks towards the camera.
6. The persons to the right, that is not looking towards the camera, turns and looks towards the camera.
7. The person in the middle turns 180 degrees from the camera.

6.4 Test 3: Blob distance

This test consists of a simple scenario where the goal is to find the range before persons become one blob. In this scenario there are two persons. The first person is going to stand still while the second person moves around. The movement of the second person is in two dimensions. The system needs to be able to detect the two persons. The minimum distance indicates how good the system can distinguish two persons by distance. The steps in the scenario are:

1. The first person and the second person stand on a line perpendicular to the camera. The first person stands in the middle of the room and the second person stands two meters from the person.
2. The second person moves towards the other person in a straight line until they become one blob.
3. The second person then turns 90 degrees and walks away from the camera. He stops when he becomes a single blob or reaches the wall.
4. The second person walks back to the other person and start walking diagonally away from the person. He stops when he becomes a single blob.

6.5 Test 4: Simulated reality

In this test the purpose is to simulate a realistic scenario. There are going to be up to five persons and a doll head in this test. First there is going to be two persons forming a

group to the left in the room. Later in the scenario a new group will be formed, and the size of the first group changes. The doll head is going to be used to simulate a child. In this scenario, the system needs to be able to handle multiple groups and movements. It needs to handle different size and shapes of persons. The steps in the scenario are as follows:

1. Two persons stand to the left in the room and are talking to each other. They are not looking towards the camera. They are later refereed to as the first group.
2. A new person enters the room and walks and stops in the right part of the room. This person looks towards the camera. This person will later form the second group.
3. A new person enters the room and walks to the person to the left to form the second group. The new person greets the other person and turns towards the camera.
4. The first person in group two now holds the doll head with abdominal height. The face of the doll is pointing towards the camera.
5. All of the people in the first group now turn towards the camera.

6.6 Test 5: Big group of people

This test is to detect if the system gives false results over a period of time. There are two persons in this test. First there is going to be only one person, and then later in the scenario there is going to be two. This is to first test how well it works on only one person, and later test how it performs with two. It is especially interesting to detect any false positive results during the test. The steps in this scenario are as follows:

1. All of the persons are standing side by side in the middle of the room. All of the persons are now looking toward the camera.
2. The second person from the right now holds the doll head.
3. All of the persons are now standing on two lines. There is a small distance between each person on the line so that the persons faces on the second line can be seen. All of the persons are looking towards the camera.
4. The persons on the edges move farther away.
5. All of the people stand as close to the middle as possible while they can still see the camera. They are all looking toward the camera.

6.7 Test 6: Over a period of time

This test is to detect if the system gives false results over a period of time. There are two persons in this test. First there is going to be only one person, and then later in the scenario there is going to be two. This is to first test how well it works on only one

person, and later test how it performs with two. It´s especially interesting to detect any false positive results during the test. The steps in this scenario are as follows:

1. The first person stands in the middle of the room looking towards the camera for 30 seconds.
2. Then the person turns 90 degrees away from the camera and stands still for 30 seconds.
3. Then a new person enters the room and stands one meter to the left the other person. The new person looks towards the camera for one min. The first person does not change position.
4. Both of the persons now look toward the camera for 30 seconds.
5. Both of the persons now turn 90 degree away from the camera. And holds this position for 30 seconds.

6.8 Test 7: Face angle

This test is to find what angle a person needs to be looking toward the camera. In this scenario only the doll head is going to be used. The doll head is going to be manually turned around. The steps in this scenario are as follows:

1. The doll head is placed in the middle of the room on a table. First the face is pointing straight towards the camera.
2. The doll head is slowly turned to the right 360 degrees.
3. The doll head is slowly turned to the left 360 degrees.

7 Results

In this chapter the main results from ***chapter 6 System test*** will be presented. During the test, a picture of each of the steps in the tests were recorded. Only the most relevant pictures will be presented and the other pictures can be found in ***Appendix C – All test results***. All of the tests were performed in the Sense-IT lab at NTNU. There were four volunteers involved in performing the scenarios. The volunteers age range from 24 to 28, and are all caucasian. The kinect version one was mounted on a tripod 1,25 meter above the ground during the testing.

7.1 Initial test

Illustration 3: Initial test: max depth range is a screen-shot taken at the moment when it was possible to see a human shape on the depth image. This was evaluated by a person looking at the depth image output. This range was measured to 3 meters. The person to the left of the detected person is out of range for the depth camera. Illustration 4: Initial test: minimum depth range is a screen-shot taken at the moment before the human shape disappeared on the depth screen. This distance was measured to 86 centimeters. The width of the area was measured to be around 2.5 meters.

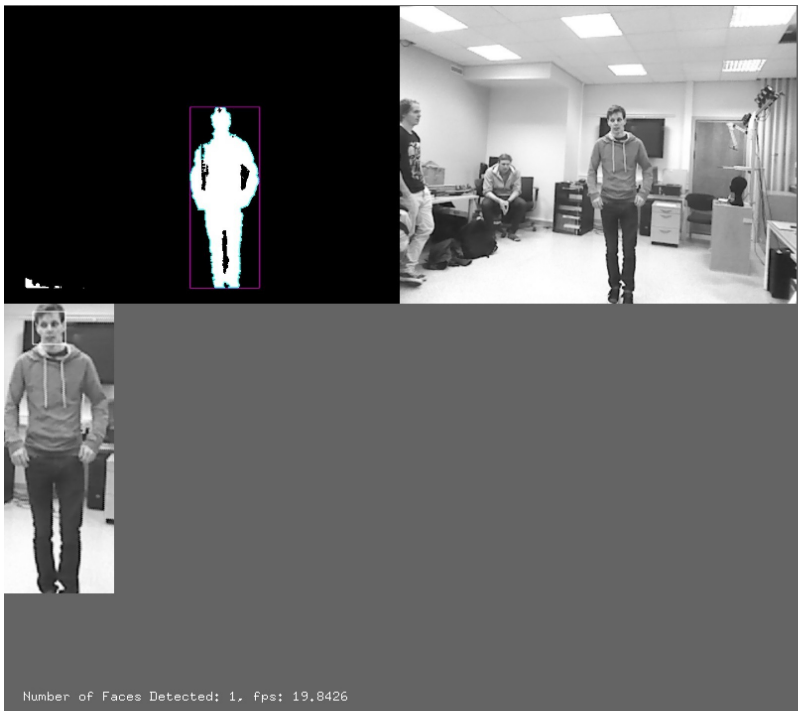

Illustration 3: Initial test: max depth range

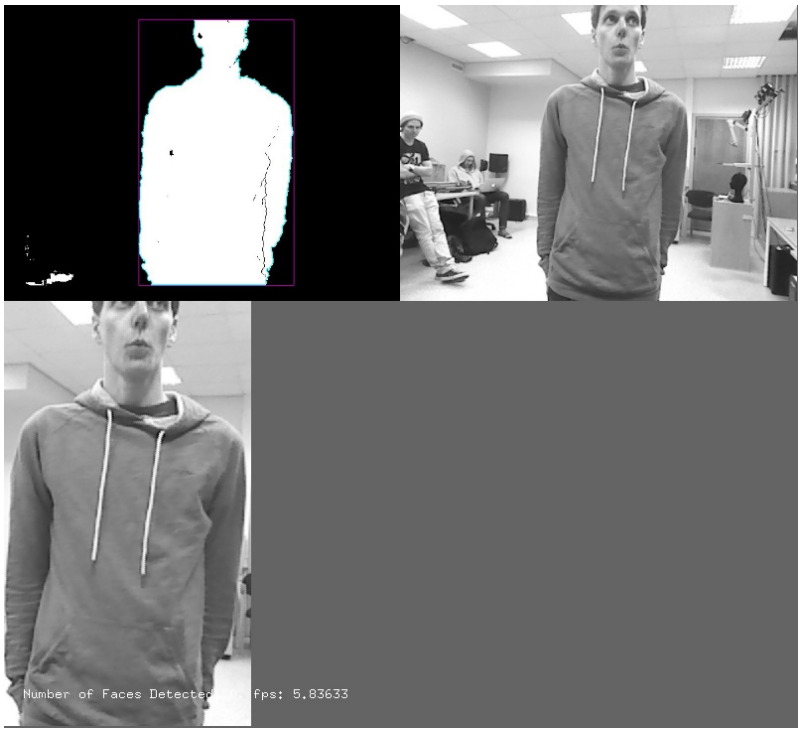

Illustration 4: Initial test: minimum depth range

7.2 Test 1: Basic test

In Illustration 5: Test 1: Step 1 shows that the system is able to detect one blob in the depth image. And from the gray-scale image the system is able to extract a blob shape and perform a face detection on this extracted section.

Illustration 5: Test 1: Step 1

In Illustration 6: Test 1: Step 4 shows that the system is able to handle two persons. It is still able to detect each of the persons in the depth image and do the extraction on the gray-scale image. The face detection detects 2 faces. This corresponds to what was happening.

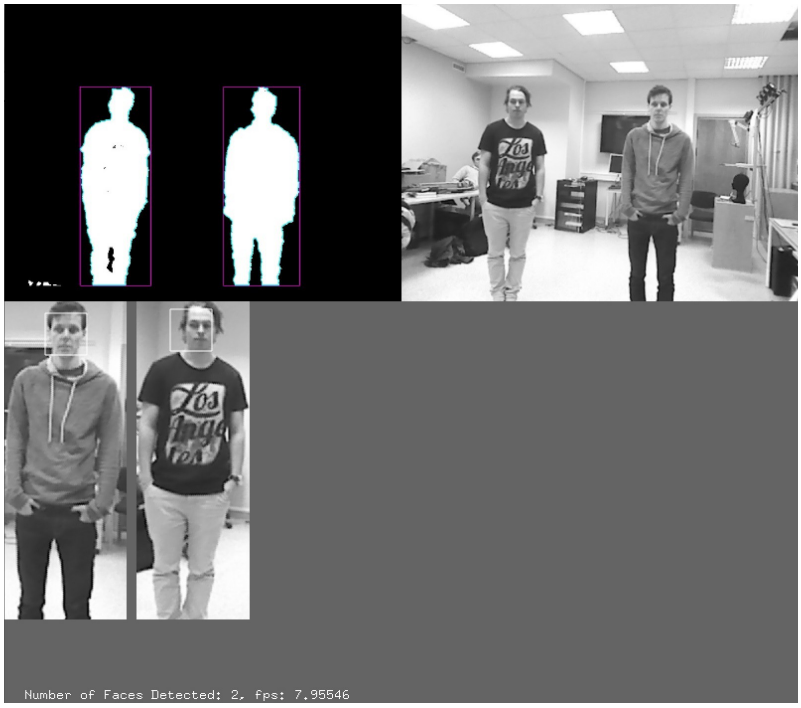

Illustration 6: Test 1: Step 4

7.3 Test 2: Basic test with group

In Illustration 7: Test 2: Step 3 the system detects two blobs. In the blob with two persons it is able to detect that there are two faces. It is also able to detect the face in the blob with only one person.

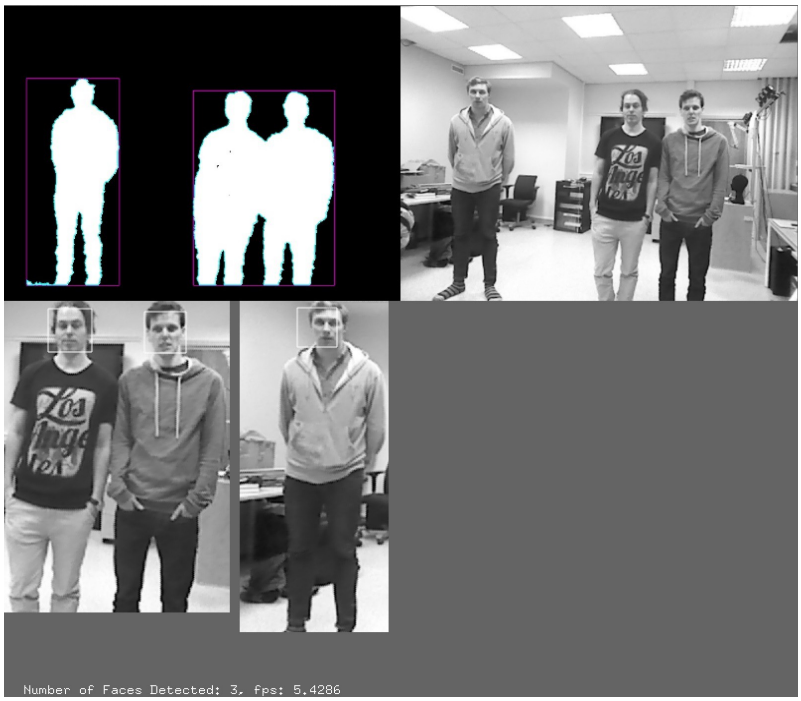

Illustration 7: Test 2: Step 3

7.4 Test 3: Blob distance

Illustration 8: Test 3: Step 2, two blobs, is taken right before the two persons become one blob in the depth image. The distance between the people is 13 centimeters. Illustration 9: Test 3: Step 2, one blob is right after they have become a blob. The distance between the persons is 10 centimeters. The second person needed to move 80 centimeters from the other person towards the camera to become a separate blob. When the second person was moving diagonally the distance was 20 centimeters between the two persons.

Illustration 8: Test 3: Step 2, two blobs

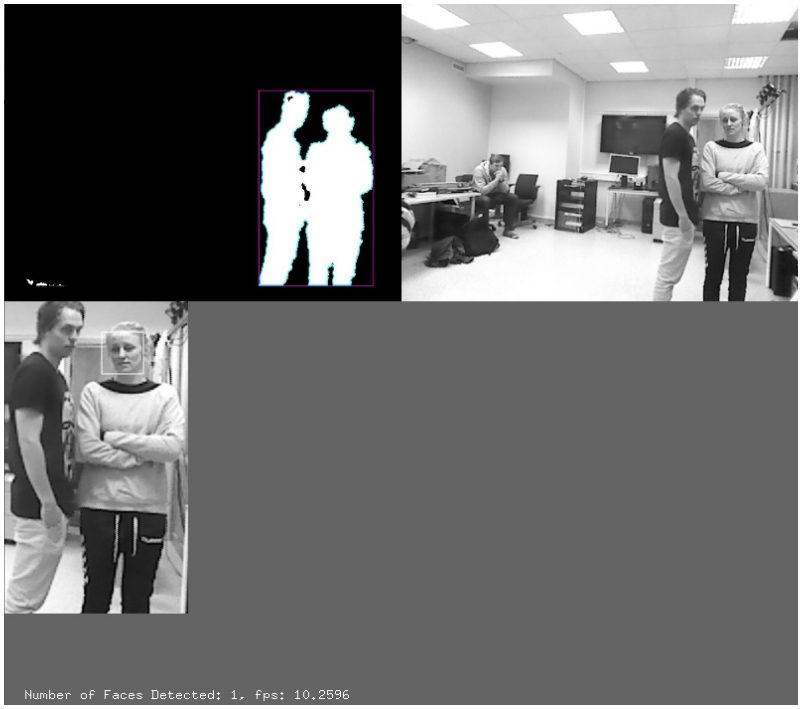

Illustration 9: Test 3: Step 2, one blob

7.5 Test 4: Simulated reality

The system is able to detect all the faces and blobs in this scenario. Illustration 10: Test 4: Step 5 is the last step in the scenario. In this picture there is one group of three faces and one group of two faces looking toward the camera.

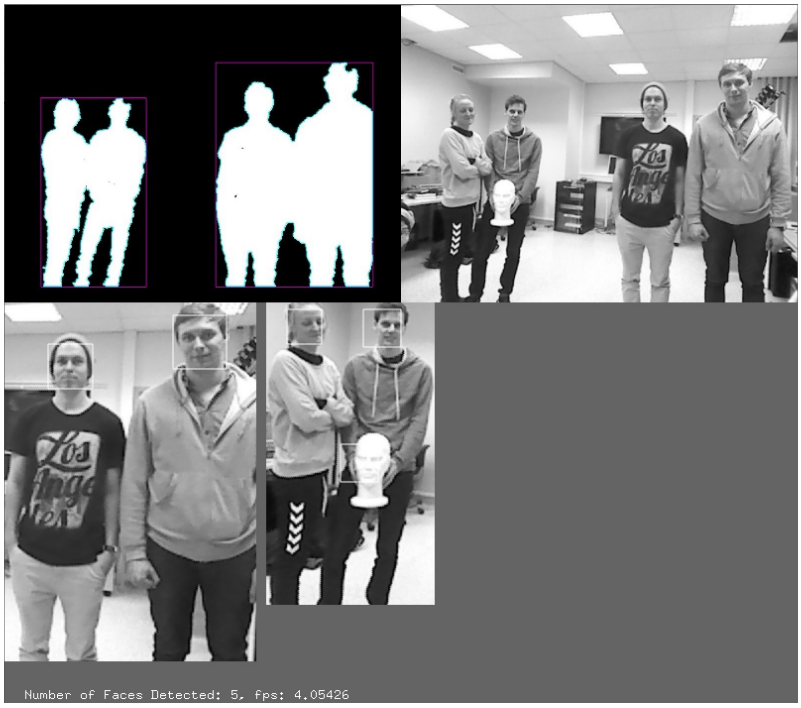

Illustration 10: Test 4: Step 5

7.6 Test 5: Big group of people

Results form testing the system on a bigger group of people are shown in Illustration 11: Test 5: Step 4 and Illustration 12: Test 5: Step 5. Step 4 and 5 are most interesting. In the images it shows that the system is able to detect all the people looking towards the camera even if they are standing apart or standing close together. In Illustration 11: Test 5: Step 4 the rectangle does not surround the face of the person standing to the right precisely. This is due to that the drawing of the rectangle do not take into count the gap between the blobs when they are drawn.

Illustration 11: Test 5: Step 4

Illustration 12: Test 5: Step 5

7.7 Test 6: Over a period of time

During this scenario only correct face detection was observed. There were no false positive face detection. During one of the other scenarios this was observed. This can be seen in Illustration 13: Test 2: Step 2. This false positive face detection was tried to be reproduced in this scenario. The images from this scenario are in ***Appendix C – All test results 9.1.1 Test 6: Over a period of time***

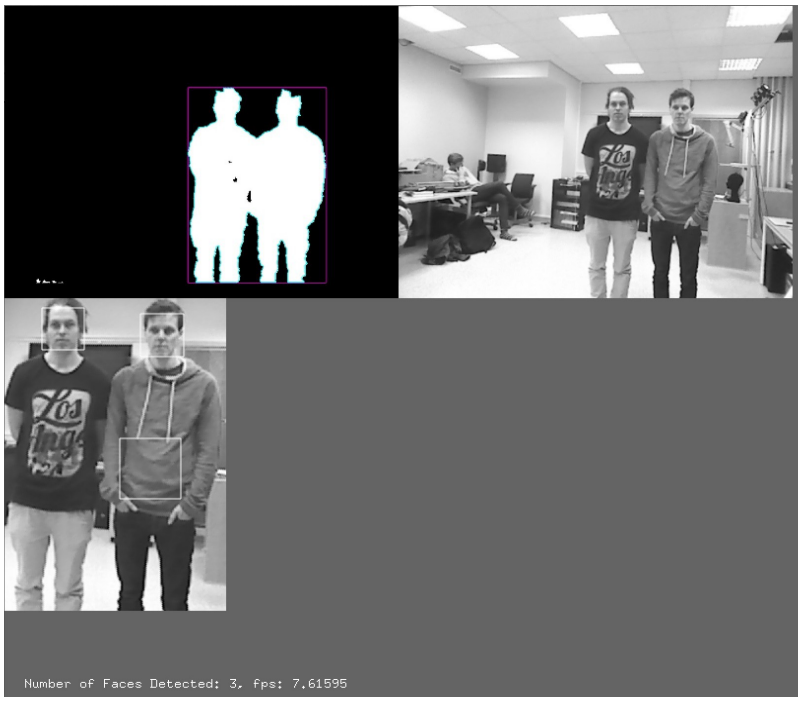

Illustration 13: Test 2: Step 2

7.8 Test 7: Face angle

During this test the system was able to detect the face when the doll was at a 25 degrees angle from the camera. Between 25 degrees and 27 the system was not sure if it was a face or not, it was flickering. When the angle was above 27 degrees it was not able to detect the face. During the other scenarios there were an interesting occurrence. If the person tilts its head the system was not able to detect the face. This can be seen in Illustration 14: Test 3: Step 3.

Illustration 14: Test 3: Step 3

8 Discussion

The results from ***chapter 7 Results*** show that the system is able to detect if people are looking towards the camera or if they are not looking. The system is able to detect objects using the depth camera, using both the depth and color camera to perform its tasks. From all of the illustrations in ***chapter 7 Results*** it is clear that using the depth image to do blob detection works great. During the test there were no problems with this part. The distinction between objects is quite clear in the depth image. There are some limitations with the depth camera though. First, the system is limited in the sense that it does not have a huge range of detecting objects from the depth camera. However, within the range of the depth camera, the system is good at detecting objects. Under the test in ***chapter 7 Results*** the system is able to detect each of the objects. Another limitation is the resolution of the depth camera. When comparing the resolution of the depth image and the color there is a huge difference. If the resolution of the depth camera could be improved, it could be possible to do more of the computer vision analysis here.

When it comes to how good the system is to detect whether people are showing an interest towards the camera, the results are a bit unclear. The system is able to detect if the persons are looking towards the camera with a 25 degree angle. From ***7.5 Test 4: Simulated reality*** it detects a person looking towards the camera. However, the observer would not claim that this person is actually looking at the camera. This indicates that the system is not able do determine if the person is just facing the camera or actually looking at it. This is a limitation when using face detection as an indication of whether people are looking toward the camera. There is another problem with using face detection, as false positive detection of a face was observed. This was observed in ***test 2 step 2***. This did only occur once during the testing. A false positive detection will give a wrong indication of how many people are actually looking toward the camera. Another limitation using the face detection is that it's not able to detect a face if the person tilts its face. If a person tilts its head, the system will not be able to detect it.

There is a limitation in the system due to the fact that it is not tracking the persons or object in the test scenarios. The system only tells how many people that are looking towards the camera in the moment. If tracking is added to the system, it should be possible to detect a face and determine how long the person has been looking towards the camera. With adding additional methods (Lucas-Kanade method) the system can be tested further to see if it meets this requirement.

9 Conclusion

The system developed in this project is able to detect peoples interest toward an object by using a kinect camera. The system is able to detect if one or up to five persons are looking toward the camera. It is able to detect multiple objects and perform face detection on each of them. The system uses the depth image to detect objects in front of the camera, and uses face detection on the color image. The face detection is done on the detected objects blob area. The system was able to fulfill all of the tests in the system test. However, some faults with the system were detected. One of them was that when a person tilted his head, the system could not detect the face. In addition, some false positive face detection occurred during the test. An example of this was when the system detected a face on the sweater to one of the participants. Further, the range of the depth camera limits the systems utilization. The range of the depth camera also limits the system's usability to work on larger groups of people, which reduces the systems' number of testing possibilities.

The following system capabilities were proven:

- The system can detect one or more objects, tested with up to four persons.
- The system is able to detect faces within a group of up to five faces.
- The system is able to detect multiple groups and detect faces within the groups.

The following weaknesses and limitations were demonstrated:

- The range of the depth camera limits the usability of the system.
- The system cannot detect faces when a persons head is tilted.
- The system sometimes detect false positive face detection.

9.1 Future work

A natural next step for the system is to add tracking. This will make it possible for the system to know if the detected blob is a new object or a previously detected object. If tracking is added to the system it will be possible to classify the blobs. This means that blobs can be classified as a person if a face is detected within the blob. Tracking makes it possible to know for how long the persons have been looking toward the camera. I believe that tracking in this system can be done by using Lucas-Kanade method.

By adding the tracking extension to the system, a more elaborated test can be performed. The system can be placed in a busy shopping mall to check whether it can detect different people looking toward an object. The results from this test can be compared to what a human is able to detect.

References and appendixes

References

[1] T. Lindeberg: *Detecting Salient Blob-Like Image Structures and Their Scales with a Scale-Space Primal Sketch: A Method for Focus-of-Attention*, International Journal of Computer Vision, 1993

[2] Lars Bretzner and Tony Lindeberg: *Feature Tracking with Automatic Selection of Spatial Scales*, Computer Vision and Image Understanding, 1998

[3] Paul Viola and Michael Jones: *Rapid Object Detection using a Boosted Cascade of Simple Features*, 2001

[4] The OpenCV Reference Manual, Release 2.4.9.0

[5] Microsoft Developer network Documentation. Kinect for Windows Sensor Components Specifications.

Appendix A – ZIP

Description about the files in the archive:

- finalApp.zip:

The files inside the zip contains the openFramework project files. It is strongly recommended to open this on a computer running osX and have Xcode installed. To run this program the openFrameworks of_v0.8.4_osx_release needs to be used. Unzip the zip and place the folder in of_v0.8.4_osx_release → apps → myApps. Open the finalApp folder and open finalApp.xcodeproj with Xcode.

Appendix B – Program code

Program code – Header

```
#pragma once

#include "ofMain.h"
#include "ofxOpenCv.h"
#include "ofxKinect.h"
#include "ofxCvHaarFinder.h"
```

```
class ofApp : public ofBaseApp{
public:
    void setup();
    void update();
    void draw();

    ofxCvGrayscaleImage setPixelsSubRegion(ofxCvGrayscaleImage  orgImage, int
x, int y,int width, int height);

    void keyPressed(int key);
    void keyReleased(int key);
    void mouseMoved(int x, int y);
    void mouseDragged(int x, int y, int button);
    void mousePressed(int x, int y, int button);
    void mouseReleased(int x, int y, int button);
    void windowResized(int w, int h);
    void dragEvent(ofDragInfo dragInfo);
    void gotMessage(ofMessage msg);

    void exit();

    ofxCvHaarFinder         faceFinder;
    ofxCvContourFinder          contourFinder;
    ofxKinect               kinect;

    ofxCvGrayscaleImage     depthImg; // gray scale depth image
    ofxCvGrayscaleImage     grayThreshNear; // the near thresholded image
    ofxCvGrayscaleImage     grayThreshFar; // the far thresholded image
    ofxCvGrayscaleImage         grayscaleImage;
    ofxCvColorImage         colorImg;
    ofxCvGrayscaleImage     blobImg;

    vector<ofxCvGrayscaleImage> blobArray;
    vector<ofxCvGrayscaleImage> blobImgArray;
    vector<ofRectangle>         blobFaceArray;
    vector<int>                 blobNrFaces;
    int                         nrBlobs;

    ofRectangle faceRectangle;
    int width;
    int height;
    int x;
    int y;

    int nearThreshold;
    int farThreshold;
    int angle;

    int threshold;
    int minBlobSize;
    int maxBlobSize;
    int nBlobsConsidered;

    //Blob stuff

    bool bLearnBg;
};
```

Program code – Setup

```
void ofApp::setup(){
    ofSetLogLevel(OF_LOG_VERBOSE);

    // enable depth->video image calibration
    kinect.setRegistration(true);

    kinect.init();
    kinect.open();        // opens first available kinect
    // print the intrinsic IR sensor values
    if(kinect.isConnected()) {
        ofLogNotice() << "sensor-emitter dist: " <<
kinect.getSensorEmitterDistance() << "cm";
        ofLogNotice() << "sensor-camera dist:  " <<
kinect.getSensorCameraDistance() << "cm";
        ofLogNotice() << "zero plane pixel size: " <<
kinect.getZeroPlanePixelSize() << "mm";
        ofLogNotice() << "zero plane dist: " << kinect.getZeroPlaneDistance()
<< "mm";
    }

    grayscaleImage.allocate(kinect.width, kinect.height);
    colorImg.allocate(kinect.width, kinect.height);
    depthImg.allocate(kinect.width, kinect.height);
    grayThreshNear.allocate(kinect.width, kinect.height);
    grayThreshFar.allocate(kinect.width, kinect.height);

    nearThreshold = 230;
    farThreshold = 80;

    ofSetFrameRate(20); // Limits the frames to 20 per second.

    // zero the tilt on startup
    angle = 0;
    kinect.setCameraTiltAngle(angle);

    //    testImgArray.resize(20);
    nrBlobs = 0;

    //Take picture


    // Initialize the blob detection
    threshold = 52;
    minBlobSize = 1000;
    maxBlobSize = (kinect.width*kinect.height)/2;
    nBlobsConsidered = 20;

    // Loading the Haar-Cascade Featurs
    //    faceFinder.setup("haarcascade_frontalface_default.xml");
    //    faceFinder.setup("haarcascade_frontalface_alt.xml");
    //    faceFInder.setup("haarcascade_frontalface_alt2.xml");
    faceFinder.setup("haarcascade_frontalface_alt3.xml");


}
```

Program code – update

```
void ofApp::update(){

    ofBackground(100, 100, 100);

    kinect.update();

    // there is a new frame and we are connected
    if(kinect.isFrameNew()) {

        // load grayscale depth image from the kinect source
        depthImg.setFromPixels(kinect.getDepthPixels(), kinect.width,
kinect.height);
        colorImg.setFromPixels(kinect.getPixels(), kinect.width,
kinect.height);
        // converts a color image to a grayscale image
        grayscaleImage = colorImg;

        //two thresholds - one for the far plane and one for the near plane
        //cvAnd to get the pixels which are a union of the two thresholds

        grayThreshNear = depthImg;
        grayThreshFar = depthImg;
        grayThreshNear.threshold(nearThreshold, true);
        grayThreshFar.threshold(farThreshold);
        cvAnd(grayThreshNear.getCvImage(), grayThreshFar.getCvImage(),
depthImg.getCvImage(), NULL);

        // update the cv images
        depthImg.flagImageChanged();

        // find contours which are between the minBlobSize and maxBlobSize
        contourFinder.findContours(depthImg, minBlobSize, maxBlobSize,
nBlobsConsidered, false, true);


        // clears the vector holding all of the previus blob images
        blobImgArray.clear();
        // clears the vector all of the old information about the detected
faces
        blobFaceArray.clear();
        // clears the vector that stored how many faces was detected in eatch
blob
        blobNrFaces.clear();
        nrBlobs = 0;

        for (int i = 0; i < contourFinder.nBlobs; i++) {

            // Extracting the rectangel of the blob
            faceRectangle = contourFinder.blobs.at(i).boundingRect;
            width = faceRectangle.width;
            height = faceRectangle.height;
            x = faceRectangle.x;
            y = faceRectangle.y;

            // creating a image with only the blob, from the grayscaleImage
            blobImg.allocate(width, height);
```

```
            blobImg = setPixelsSubRegion(grayscaleImage, x, y, width, height);

            // find the faces in the blob
            faceFinder.findHaarObjects(blobImg);

            // stores the number of faces detected in the blob
            blobNrFaces.push_back(faceFinder.blobs.size());

            // stores the information ablut the faces detected
            for (int i = 0; i < faceFinder.blobs.size(); i++) {
                blobFaceArray.push_back(faceFinder.blobs.at(i).boundingRect);
            }

            // Stores the blob image
            blobImgArray.push_back(blobImg);


            nrBlobs++;
        }
    }

}
```

Program code – draw

```
void ofApp::draw(){

    // Draws the Depth image
    depthImg.draw(0, 0, 400, 300);
    // Draws the Contour of the detected blobs
    contourFinder.draw(0, 0, 400, 300);
    // Draws the grayscale Image from the color camera
    grayscaleImage.draw(400, 0, 400, 300);

    // Drawing the blob images of the detected blobs
    int totalFaces = 0;
    int printWidth = 0;
    for (int i = 0; i < nrBlobs; i++) {
        blobImgArray[i].draw(printWidth+(i*10), 300);

        // Draws the detected faces from face finder
        ofNoFill();
        for(unsigned int j = 0; j < blobNrFaces[i]; j++) {
            ofRectangle cur = blobFaceArray[totalFaces];
            ofRect(cur.x + printWidth, cur.y + 300, cur.width, cur.height);
            totalFaces = totalFaces + 1;
        }

        printWidth = printWidth + blobImgArray.at(i).getWidth();
    }

    // Prints out the number of detected faces if there are any blobs
detected. Prints out the FPS too.
    if (nrBlobs > 0) {
        ofSetHexColor(0xffffff);
        stringstream reportStr;
        reportStr << "Number of Faces Detected: "  <<  totalFaces << ", fps: "
```

```
<< ofGetFrameRate();

        ofDrawBitmapString(reportStr.str(), 20, 700);
    }

}
```

Program code – setPixelsSubRegion

```
ofxCvGrayscaleImage ofApp::setPixelsSubRegion(ofxCvGrayscaleImage orgImage,int
x, int y,int width, int height)
{
    ofxCvGrayscaleImage targetImg;
    bool color = false;
    unsigned char * pixels = orgImage.getPixels();
    int totalWidth = orgImage.getWidth();
    int subRegionLength = width * height;
    unsigned char subRegion[subRegionLength];

    int result_pix = 0;
    for (int i = y; i < y+height; i++)
    {
        for(int j = x; j < x+width; j++)
        {
            int base = (i * totalWidth) + j;
            subRegion[result_pix] = pixels[base];
            result_pix++;
        }
    }
    targetImg.setFromPixels(subRegion, width, height);
    return targetImg;

}
```

Appendix C – All test results

Test 1: Basic functions

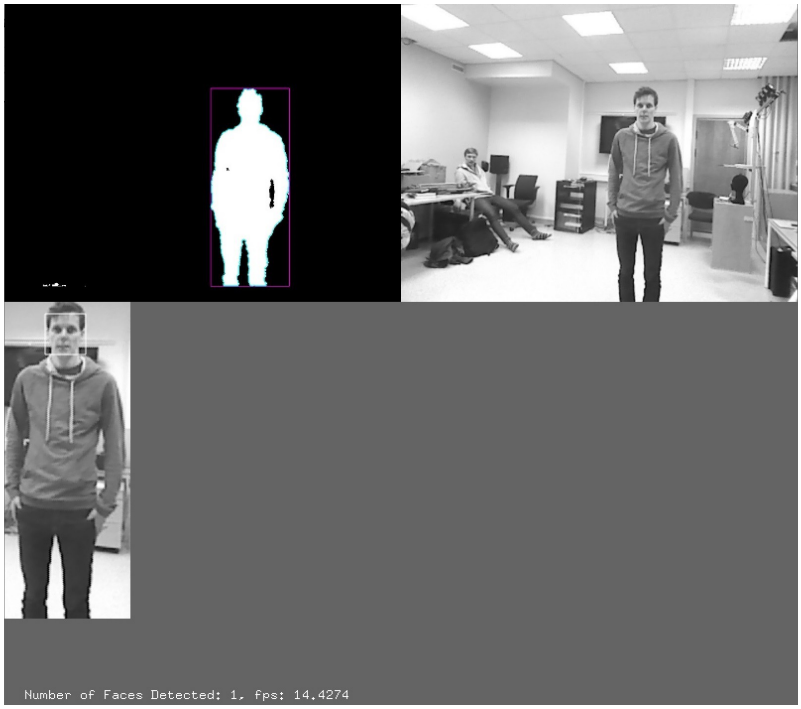

Illustration 15: Appendix C. Test 1: Step 1

Illustration 16: Appendix C. Test 1: Step 2

Illustration 17: Appendix C. Test 1: Step 3

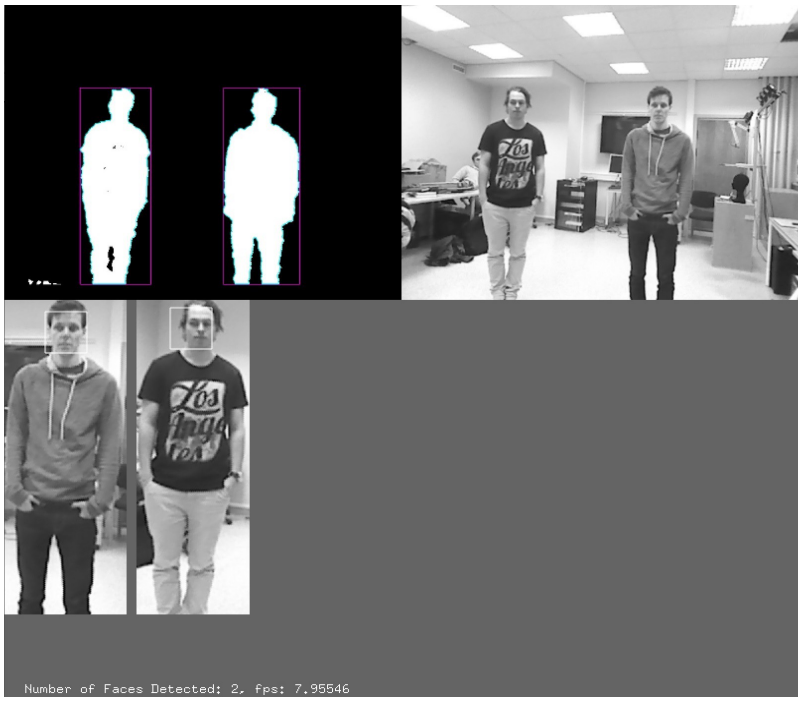

Illustration 18: Appendix C. Test 1: Step 4

Test 2: Basic test with group

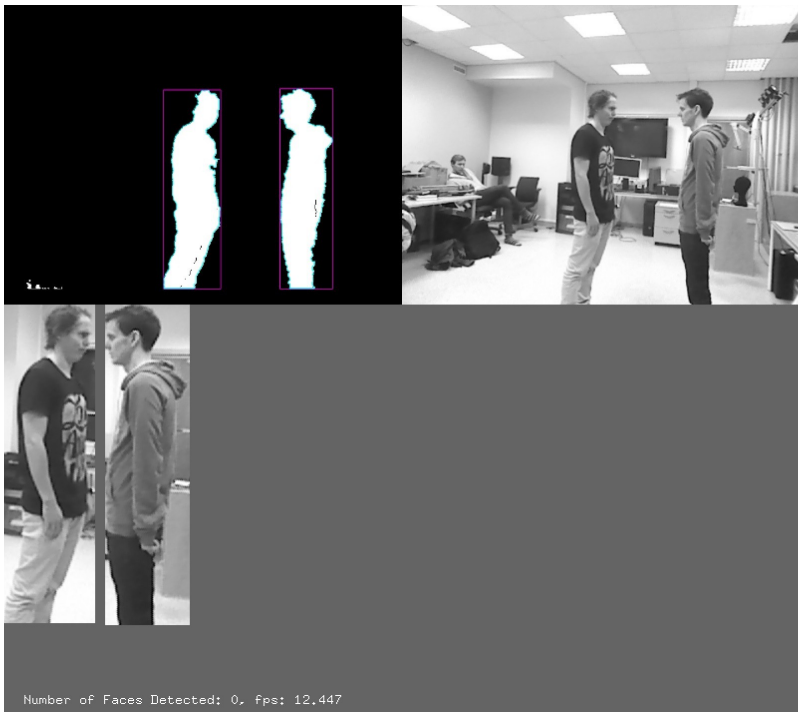

Illustration 19: Appendix C. Test 2: Step 1

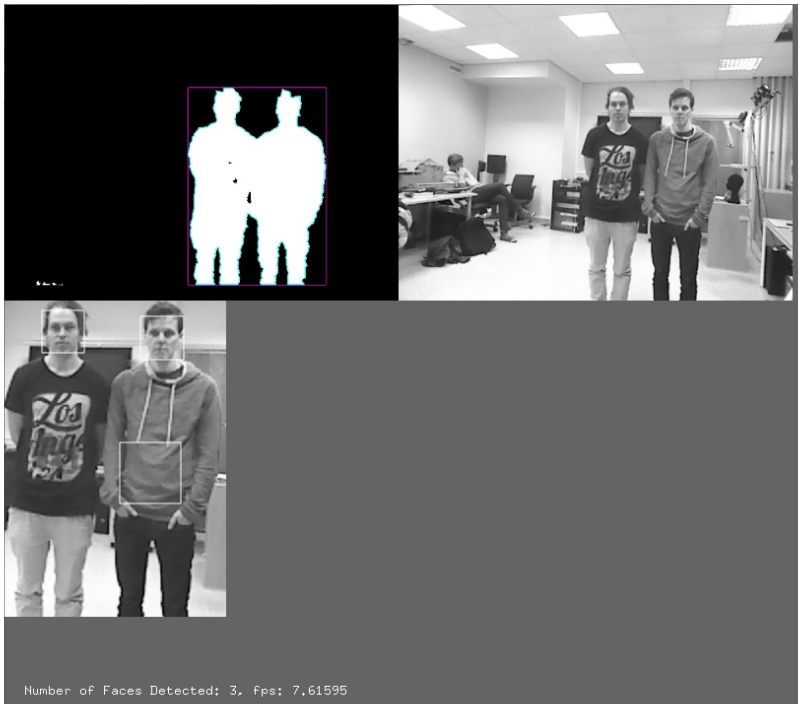

Illustration 20: Appendix C. Test 2: Step 2

Illustration 21: Appendix C. Test 2: Step 3

Illustration 22: Appendix C. Test 2: Step 4

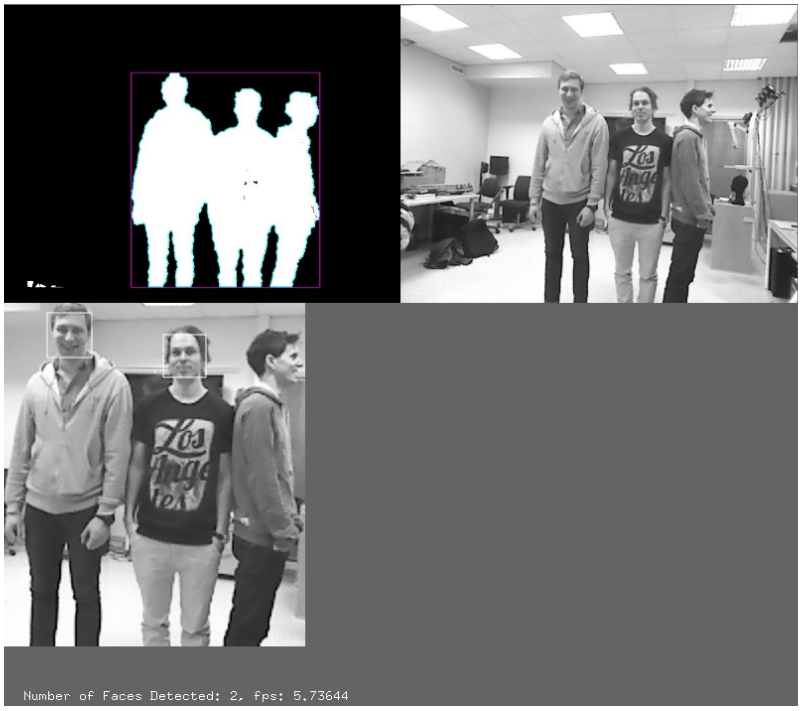

Illustration 23: Appendix C. Test 2: Step 5

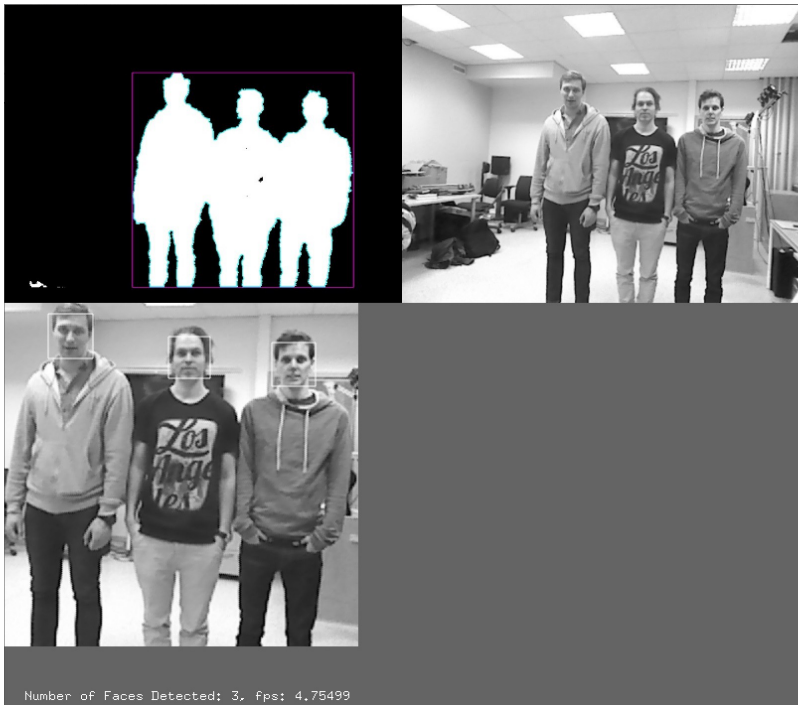

Illustration 24: Appendix C. Test 2: Step 6

Illustration 25: Appendix C. Test 2: Step 7

Test 3: Blob distance

Illustration 26: Appendix C. Test 3: Step 1

Illustration 27: Appendix C. Test 3: Step 2, two blobs

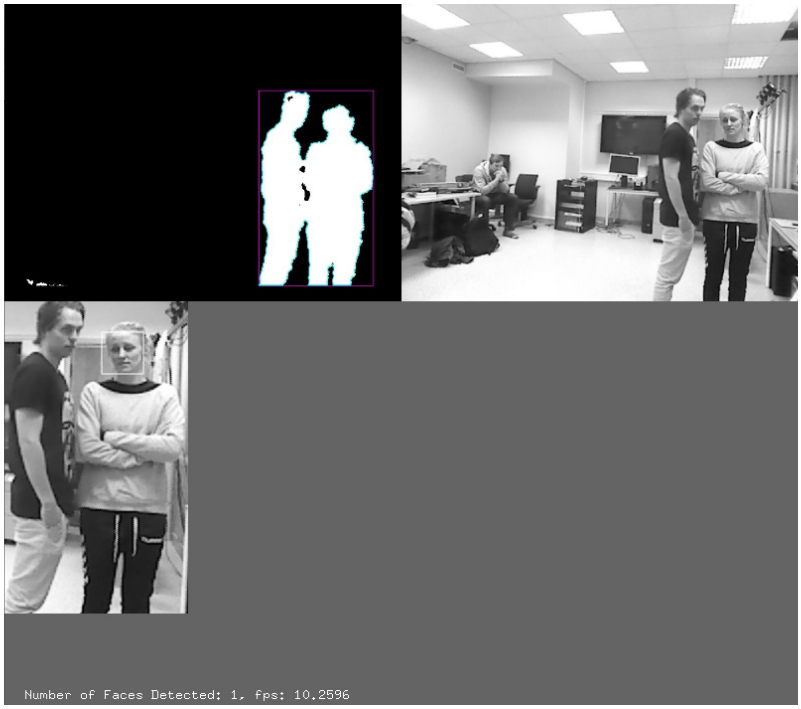

Illustration 28: Appendix C. Test 3: Step 2, one blob

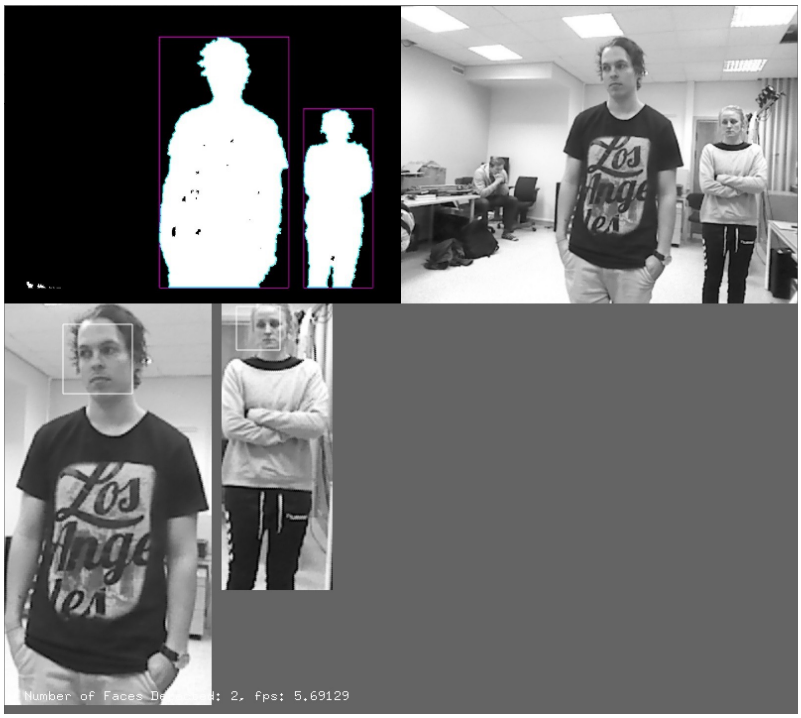

Illustration 29: Appendix C. Test 3: Step 3

Illustration 30: Appendix C. Test 3: Step 4

Test 4: Simulated reality

Illustration 31: Appendix C. Test 4: Step 1

Illustration 32: Appendix C. Test 4: Step 2

Illustration 33: Appendix C. Test 4: Step 3

Illustration 34: Appendix C. Test 4: Step 4

Illustration 35: Appendix C. Test 4: Step 5

Test 5: Big group of people

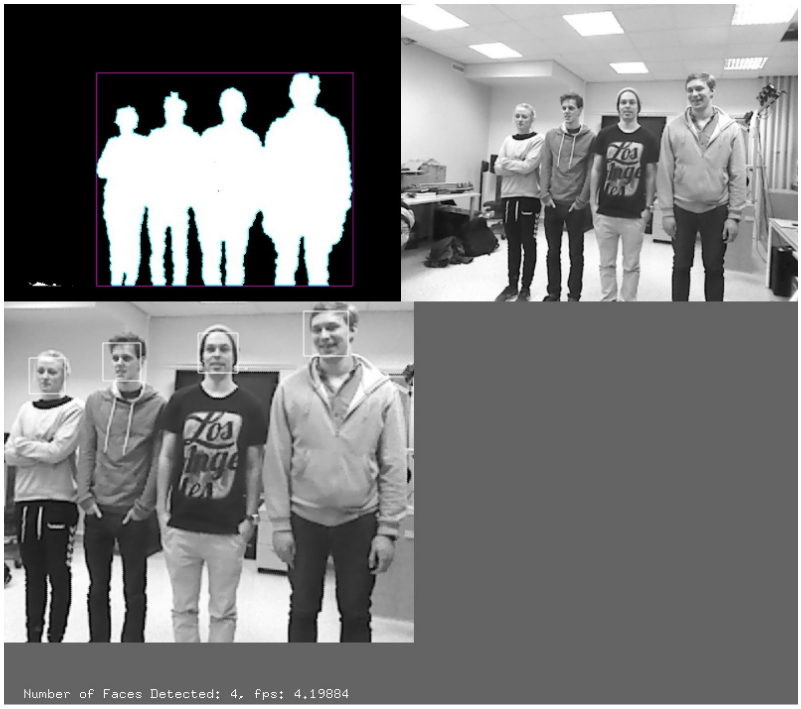

Illustration 36: Appendix C. Test 5: Step 1

Illustration 37: Appendix C. Test 5: Step 2

Illustration 38: Appendix C. Test 5: Step 3

Illustration 39: Appendix C. Test 5: Step 4

Illustration 40: Appendix C. Test 5: Step 5

9.1.1 Test 6: Over a period of time

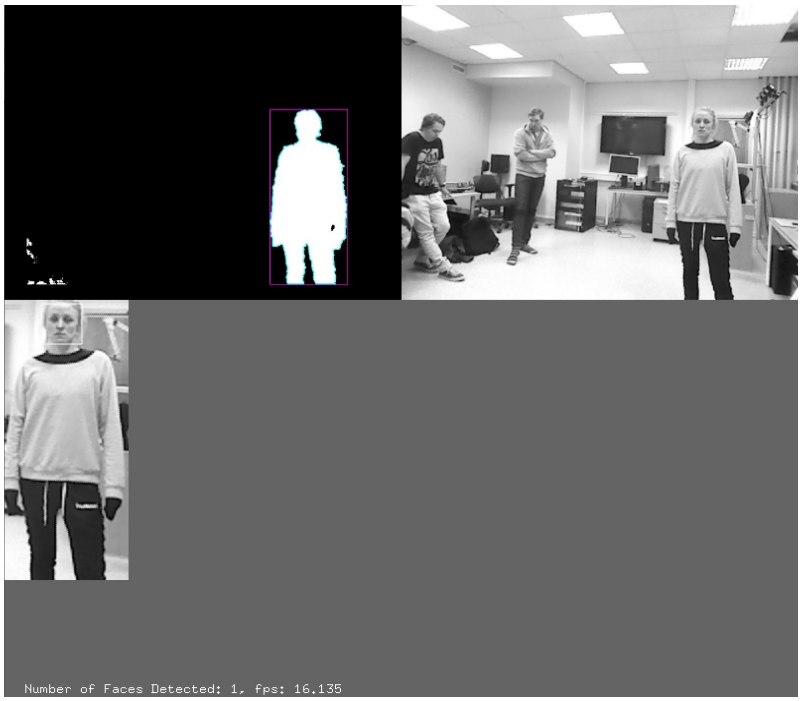

Illustration 41: Appendix C. Test 6: Step 1

Illustration 42: Appendix C. Test 6: Step 2

Illustration 43: Appendix C. Test 6: Step 3

Illustration 44: Appendix C. Test 6: Step 4

Illustration 45: Appendix C. Test 6: Step 5

Illustration 46: Appendix C. Test 6: Step 1, with a other person