

Real-Time Localization and Tracking of Multiple People in a Closed Environment with Facial Detection Using a Multi-Camera Setup



Master Thesis

Sebastiaan van Hese

Delft, May 2008



Acknowledgements

This research was mainly conducted at the Center of Advanced Information Processing at Rutgers University in Piscataway, NJ, USA. I would like to thank my supervisor at CAIP; Dr. I. Marsic, for inviting me to spend nine unforgettable months at his research department, his guidance while working on the Smart Trauma-resuscitation Room research project, his hospitality and his advice on several subjects concerning research and in general living abroad. Dr. A. Elgammel for sharing his knowledge of visual computing and guiding me through the design of this research.

A special thanks for my thesis supervisor in The Netherlands, Prof. Dr. Drs. Leon Rothkrantz for giving me the opportunity to spend 9 months at the Center of Advanced Information Processing, arrange funding and I am grateful for his inspiration and motivation while finishing my MSc. Thesis back home.

I would also like to thank my parents for the support and faith throughout my study and my brother and sister for their motivation in general. To all my friends in The Netherlands and the friends I made in America, they made the nine months in USA an unforgettable experience. I acknowledge the financial support of the Stichting Universiteitsfonds Delft, STIR, the Delft University Fund, the Fundatie van de Vrijvrouwe van Renswoude and the Faculty Fund.

CAIP

During nine months I have conducted research at the Computationally Advanced Infrastructure Partnerships (CAIP) Center at Rutgers University in Piscataway, NJ, USA.

CAIP is an Advanced Technology Center (ATC) sponsored by both industry and government. It conducts research in parallel and distributed computing, image and speech processing, graphics and data visualization, human/machine communications, robotics, and software engineering.

The CAIP Center is established in the Computing Research and Education (CoRE) Building on the Busch Campus in Piscataway.



Figure 1: Computing Research and Education building on the Busch Campus in Piscataway. CAIP Center is located on the top floors.

The Computationally Advanced Infrastructure Partnerships Center is rapidly becoming a leader in the field of high performance and collaborative computing, research and education. The CAIP center will also provide a platform for emerging collaborative computing initiatives which will incorporate multidisciplinary teams drawn from academia, industry and government.

CAIP shares the CoRE Building with related technology departments and centers, including the Departments of Industrial Engineering, Computer Science, Electrical & Computer Engineering,

and the Center for Discrete Mathematics & Theoretical Computer Science. CAIP supports facilities for high performance computing and has numerous additional computational facilities. The CAIP center is currently active on a wide variety of research projects like the Virtual Reality Laboratory, the Speech and Language Processing Lab, Machine Vision Lab, Biorobotics, and the Smart Trauma-Resuscitation Room project managed by Dr. Ivan Marsic which resulted in this thesis. While conducting research at CAIP Dr. Marsic was my supervisor.

Dr. Ivan Marsic is responsible for numerous publications and research initiatives including: Engineering Complex Collaborative Systems, Mobile Wireless Networks and QoS, Multimodal Human-Computer Interfaces and Visual Attention and Image Processing. He currently is associate professor at both the Department of Electrical and Computer Engineering and the Center for Advanced Information Processing.

Abstract

Author : Sebastiaan van Hese
Studentnumber : 1047418
University : Delft, University of Technology
Department : Human Computer Interaction
Thesis Title : Real-Time Localization and Tracking of Multiple People in a Closed Environment with Facial Detection Using a Multi-Camera Setup.

Thesis Committee : Dr. Drs. L.J.M. Rothkrantz
Ir. H.J.A.M. Geers
Ir. P. Wiggers
Ir. Z. Yang

In this thesis a novel system is presented capable of detecting and tracking multiple people wearing similar clothing using multiple synchronized and calibrated cameras. The system improves on existing systems in the following way:

Instead of segmenting objects using color models this research uses face detection, and geometrical classification to segment objects from one another.

The problem is tackled by developing some innovative algorithms and methods. It designs a parallel implementation of the Viola and Jones face detector to detect faces in different orientations. The information from multiple cameras about tracked objects is combined to produce better results. An innovative algorithm for object-location estimation given multi-view classification data is developed and a novel classification algorithm is presented based on previous position and orientation off the tracked object. Finally a system is constructed that combines the fields of face detection, tracking and multi-perspective reasoning.

With these algorithms a running prototype of the system is created. This prototype has been successfully tested using a simulated environment and also with real-live camera footage.

Table of Index

Abstract	vi
Chapter 1 Introduction	1
1.1 Problem definition	3
1.2 Goals	3
1.3 Challenges of the problem	4
1.4 Applications	6
1.4.1 Smart resuscitation bay	7
1.4.2 Aggression detection in public transportation	13
1.5 Roadmap	14
Chapter 2 Related Work	15
2.1 Overview	15
2.2 Face detection	15
2.2.1 Viola and Jones	15
2.2.2 Huang and Haizhou	17
2.3 Tracking methods	18
2.3.1 M2Tracker	18
Chapter 3 Theoretical Background and Problem Domain Description	21
3.1 Face detection	22
3.1.1 Parallel Viola and Jones	23
3.2 Environment setup	23
3.2.1 RFID tags	25
3.3 Vision	26
3.3.1 Video cameras	26
Chapter 4 Proposed Design	33
4.1 Face detection	34
4.2 Classification	37
4.3 Likelihoods	43
4.4 Location estimates	45
4.5 Kalman filtering	46
4.6 Combined location map	50

4.7 RFID identification	50
Chapter 5 Implementation	53
5.1 Development tools	53
5.1.1 OpenCV	54
5.1.2 Coin3D	55
5.2 Implementation of proposed algorithms	58
5.2.1 System	60
5.2.2 Face detection	61
5.2.3 Classification	64
5.2.4 Likelihood maps	67
5.2.5 Estimating object location	69
5.2.6 Kalman filter	71
5.3 Virtual environment	72
5.3.1 Camera calibration	77
5.3.2 Feeding images to the algorithm	78
5.4 Implementing the live environment	80
5.4.1 Setting up the scene	80
5.4.2 Capturing the scene	82
5.4.3 Feeding images to the algorithm	83
Chapter 6 Evaluation and Results	85
6.1 Software testing	85
6.1.1 Test Plan	85
6.1.2 Bugs	85
6.1.3 Time management	86
6.2 Face detection	87
6.3 Virtual environment	93
6.4 Live environment	96
6.4.1 The test scenarios	96
6.4.2 Test results	98
6.5 Evaluation of test results	115
Chapter 7 Conclusions and Future Work	117
7.1 Conclusions	117
7.2 Future work	119
References	121

Chapter 1

Introduction

Today the results shown by object detection and motion tracking demonstrate many opportunities in current situations and future applications. Both fields benefit from great attention in current society. The power to predict information about object identity, behavior and position is being used by monitoring services, security applications and information retrieval.



Figure 2:(a) Typical outdoor surveillance camera (b) Working algorithm detecting and tracking multiple people using one single camera.

Identification in this thesis is not only focused on finding credentials of a humanoid object but in a broader sense also on identifying certain tasks this object is performing or behavior it is showing. Together with other information, detection and tracking data results in predicting behavior and processed tasks, and gives semantic meaning to sensor data. Central to this monitoring task is the translation of the ‘where they are’ information to predict the ‘what they are doing’ question. This research focuses primarily on the ‘where they are’ question in specific environments. The results of this research can then be used to predict the ‘what they are doing’ question.

Many surveillance applications perform their tasks unobtrusively, meaning tracked objects themselves do not contribute to the working system by wearing devices. Examples of such devices can be probes sending information about geographical position or broadcasting devices sending identification information using radio frequencies. Also, when tracking humanoid objects, these applications usually cope with crowded scenes and tracking occluded objects. Furthermore lighting conditions, colors of people or environments and quality of image feeds are very determining for system results.

Some solutions to these difficulties have been presented, most successfully by using a setup with multiple cameras to aid the system with more spatial information and less uncertainty of other variables about the objects being tracked. Such a system is capable of detecting and tracking multiple people in cluttered scenes with various lighting conditions.

When other possible applications are presented that could use such a system, more constraints are placed in the domain description. The hospital is an environment where the use of tracking systems can give valuable information about placement of personnel that can be used by task analyzing expert systems. A problem with implementing current tracking systems in the hospital environment is the lack of discriminating features between objects. This is the result of prescribed uniforms worn by hospital personnel. These uniforms usually have identical colors and make the tracking of multiple objects harder because of low discriminative characteristics.



Figure 3: Emergency bay team working on a simulated resuscitation case. Note the similarity in clothing worn by personnel.

Current research is unable to work in an environment described above. Because current methods do not give an answer to this issue, this thesis is an attempt to solve that problem.

1.1 Problem definition

Digitally monitoring people can be described as a process of three steps. First sensors detect new objects in the image feed. These objects are then followed using tracking algorithms and finally behavior analysis is performed to add semantics to the data.

This research is focused on the first two parts of this process. The problem this research addresses is detecting and tracking multiple people using a multi-perspective approach in a closed environment.

Furthermore we introduce an important domain constraint that influences the nature of this research substantially. The problem needs to be solved in an environment where people are wearing similar clothing. This means that the difference between the various bodies is too small and unpredictable and makes building a tracking system based on these features an ineffective approach.

Besides the substantially large challenge described we also face other problems such as:

- Coping with different lighting conditions in the presented image feed.
- Cluttered scenes resulting in occlusion of people.
- Changes in people's posture result in difficult detection and tracking.
- Using multiple cameras results in the need for synchronization and calibration for each of them.

These problems are further investigated in **chapter 1.3**, challenges of the problem.

1.2 Goals

The most promising research in segmenting, detecting and tracking multiple people using a multi-perspective approach uses the variety of color patterns of the body to discriminate between

tracked objects (Davis, 2002). Because the environment of this research lacks the variety of color patterns this approach is not effective in this case.

Current progress in facial computer vision shows promising results for using methods based on face detection and motion tracking for this research. We would like to investigate the possibility of introducing facial detection combined with tracking algorithms in solving the stated problem.

Therefore the main goals of this thesis are as follows:

- Investigating the problem of discriminating and tracking multiple people using multiple cameras by using the fields of facial detection and tracking.
- Developing a face detector capable of multi-orientation face detection.
- Developing a classification algorithm based on object location data.
- Design an algorithm for object-location estimation given multi-view classification data.
- Design specific Kalman filtering for this tracking problem.
- Design an iterative system fed with camera images and implementing the communication flow between developed algorithms.
- Implement and demonstrate the working of the system by creating a digital environment that simulates the real domain.
- Create test cases and use real camera footage to demonstrate the working of the implemented system.

1.3 Challenges of the problem

In order for a successful result, some important obstacles are identified. Below a list of expected difficulties.

- Face detection currently is by far 100 percent infallible. Processed video footage is returning many false-positives and discarded valid facial detections. Some valid objects are discarded as facial detections and some faces will never be detected in the stream. By definition the system will be less dependent on the results of a single face detector on one camera for it uses a multi-perspective approach, but a appropriate handling of this problem still remains important for good results. The use of a dependable face detector is

essential. Furthermore, the detector can be aided in its task by feeding it with high quality and visually undistorted video material.

- The multi-perspective approach means that all cameras need to be carefully calibrated to the environment to make sure accurate geometrical calculations are possible during tracking. Calibration to the environment addresses problems like projecting a 3-dimensional world onto a 2-dimensional image. Calibration is necessary for tracking. Every pixel from any camera view needs to have geometrical meaning in the 3-dimensional environment being observed.
- How are objects classified from each other when information from body features are not discriminative enough? This thesis shows its relevance in this domain. The medical personnel is dressed in similar clothing. Most of them wear white, green or blue clothes. Because of this domain constraint, color histograms cannot effectively be used when tracking and localizing objects. Furthermore a way of classifying detected objects needs to be implemented. When using facial detection results can be classified by a combination of geometrical, orientation and facial characteristics. Especially the correct matching of detected faces to the belonging people models in the environment presents challenges. Due to variations in orientation of the person comparing detected faces with face models of people is a hard task. In **chapter 4**, Proposed Design, this topic is further investigated.
- The population is updated dynamically. Both people leaving and entering the environment should be handled appropriately. The system should recognize a new person not to be a member of the current population and add this person to the pool of inhabitants. People leaving the environment should not be deleted right away only until statistically is proven the person can be removed from the current inhabitants of the environment.
- Occlusion of objects in the environment is one of the bigger problems in multi-people tracking systems. One of the advantages of using a multi-perspective approach is that

occlusion of objects is less of a problem with larger crowds. People can be occluded by other people, static and dynamic objects or even by their own movements like waving their hand in front of their face. Although multiple cameras are used occlusion remains a problem that needs to be handled. When considering facial detection some applications will present even more challenges. In many domains people will actually wear certain attributes on their faces making face detecting harder.

- Quality of live camera footage is influenced by lighting conditions in the room, created either by artificial sources, natural sources, or both. For this research a closed environment is used. Therefore it is independent to any external influence like the sun. Carefully placed lighting conditions will be implemented when testing the system.
- When people move around in the environment they often change position and rotation. Head and body move independent of each other. Furthermore they can for example bend over while picking up something from the floor. Changes in people's posture result in difficult detection and tracking.

1.4 Applications

A system that implements the chosen strategy for solving the defined problem will be applicable in many real live situations. This system can be used in situations where people are harder to recognize considering clothing, occlusion etc. and can be extended as a regular detection and tracking system. This research is mainly focused on the domain it was designed for, the resuscitation bay (i.e. emergency room: ER) in hospitals, but other applications fit this research as well. This thesis provides a great opportunity for many applications that are restricted by similar constraints in the domain.

First described is the initial proposed application and the fitting of this research in that domain. Then one more application is presented.

1.4.1 Smart resuscitation bay

This thesis originated from a large research project currently researched by the CAIP department in New Brunswick, New Jersey (CAIP, 2007). I have worked in a research team during a period of 9 months on this project. Below is a description of the problem, goals and proposed design of this research project. Figures and several parts of this description are written by the group working on Smart Trauma-Resuscitation Room project (Ivan Marsic, 2008).

Trauma is a leading cause of death and disability in children and young adults. Because care early after injury has an important impact on outcome, initial management of the injured patient should be rapid, efficient and error-free. The initial resuscitation of an injured patient is prone to errors because workers often manage patients with unstable or fluctuating medical status, whose detailed medical history is commonly not available and who require time-critical decisions. Medical errors are prevented mainly through provider experience and redundancies in the evaluation process. The potential errors increase when treating injured children, because variable anatomy and physiology at different ages prevent standardization.

Information exchange during trauma resuscitation is unique, since clinical data is simultaneously and rapidly accrued by several providers and used to make decisions within minutes. The potential for errors increases because providers need to focus on their tasks while maintaining constant awareness of tasks of others. While information technologies have been used to support highly strained cognitive activities in time-critical settings, these methodologies have not been applied for trauma evaluation. Previous studies have focused primarily on individual team members who are operating sensors or computerized controls. This type of analysis will not be effective in environments, such as trauma resuscitation, with *structured teams* of members with different roles, teamwork being highly interdependent, and members relying primarily on judgment and experience rather than sensors. Studying teamwork in this environment therefore cannot rely on automatic data capture.

The resuscitation bay in hospitals is a very dynamic and busy room when used. An injured patient enters the room and the room becomes crowded by doctors, nurses, radiologists and other

personnel. To the unfamiliar eye this situation seems rather chaotic. But as time progresses and less people are moving around a smaller group of medical personnel remains in the room. For all personnel in this room the situation itself is quite familiar and less chaotic than seen by a new observer. During resuscitation every person in the room has a predetermined role. The composition of teams is usually very similar at different facilities and includes an attending surgeon, surgical residents, an anesthesiologist, an orthopedic surgeon, nurses, a respiratory therapist, a pharmacist, and an x-ray technician. Not everyone is needed during the complete resuscitation and people arrive and leave the resuscitation bay at a regular interval. It is essential for the group working on the patient to combine their efforts into an orchestrated process with the goal to stabilize the patient.

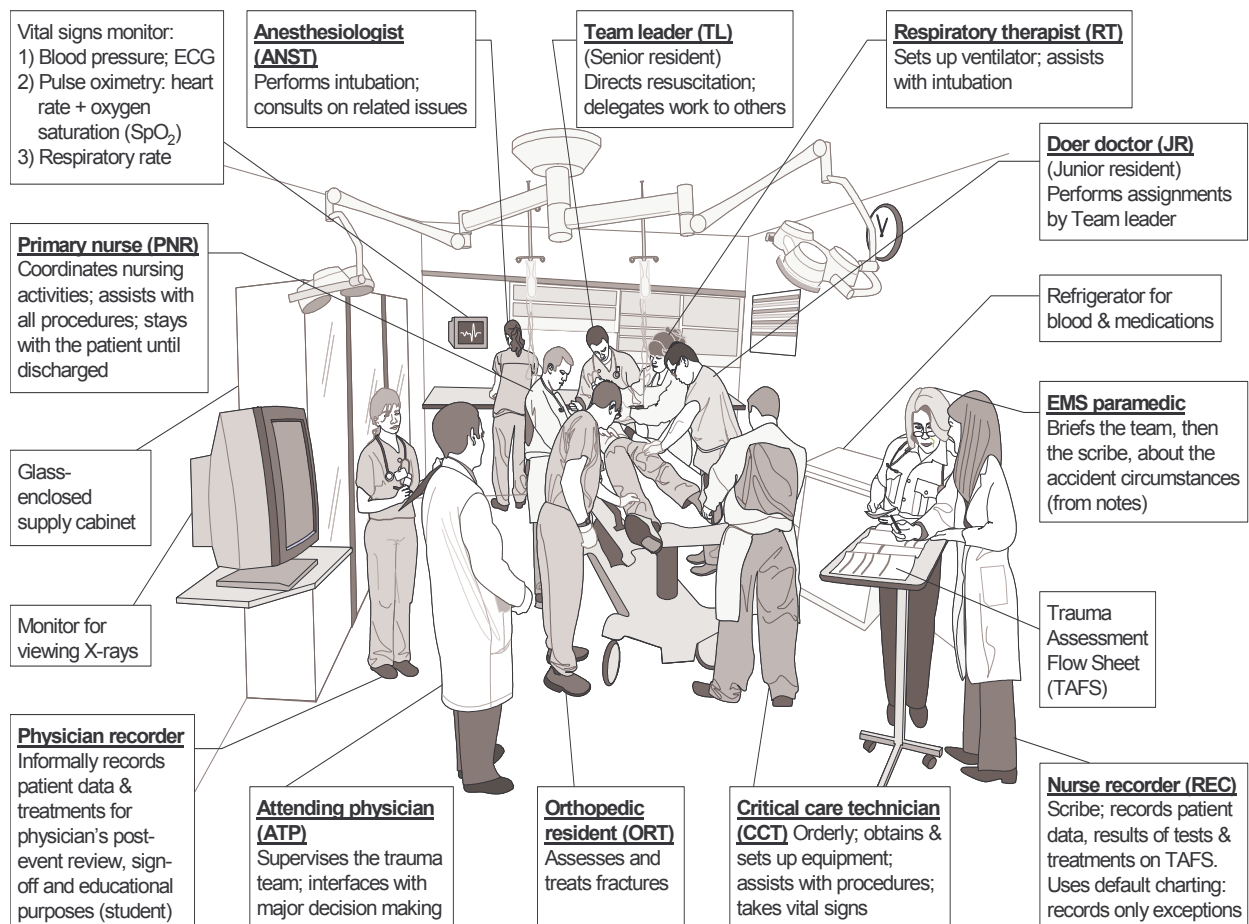


Figure 4: Emergency department (ED) trauma bay: Typical actors and props in a trauma resuscitation event.

The goal of the CAIP project is to lower the potential for medical errors and improve efficiency during resuscitation of the injured patient by introducing information technology in the trauma bay. Technological interventions include: (1) information communication and presentation technologies to improve team communication and information absorption, and (2) automatic monitoring of team activities and work progress to check for errors in procedure and decision making. The overall hypothesis of the project is:

The combination of information communication and presentation, and automatic monitoring during resuscitation will reduce medical errors and improve the efficiency of emergency trauma care.

This hypothesis will be tested by pursuing these four integrated aims:

1. To model and analyze teamwork in actual and simulated trauma resuscitations by (a) quantifying workload and team structure; (b) identifying key decisions made by the team and the input parameters for those decisions; and (c) performing logical and statistical analysis of error causes and correlates.
2. To develop a mobile collaborative data entry and display system that improves information exchange and decision making during trauma resuscitation.
3. **To develop an automatic system for real-time monitoring and tracking of teamwork, and alerting for anticipated errors and warning for already detected errors.** This function shows the relevance to this thesis.
4. To evaluate the impact of the information system on reducing communication errors and improving the efficacy of trauma resuscitation using human patient simulators.

The goal of this project is to lower the potential for medical errors and improve efficiency during resuscitation of the injured patient by introducing information technology in the trauma bay. Technological interventions include: (1) information communication and presentation technologies to improve team communication and information absorption, and (2) automatic monitoring of team activities and work progress to check for errors in procedure and decision

making. **Figure 5** illustrates an example of implementing such technology in the smart resuscitation bay.

Part of the smart resuscitation bay is a decision-support system able to devise a plan for resuscitating the patient and advising or critiquing a surgeon, can minimize errors greatly. By supplying the expert system with data about the patient and its environment the system can reason about resuscitation plans to stabilize the patient. Several sources of information are available in the resuscitation bay to supply the expert system with useful knowledge. Vital signs of the patient are a great source of knowledge about the patient’s status. The expert system preferably wants to critique a plan that is being executed. Then information about the tasks the resuscitation team is currently performing is very important. Vocal orders or statements given by team leader are for example great indications of future actions. Other useful information is the location of personnel in the resuscitation bay.

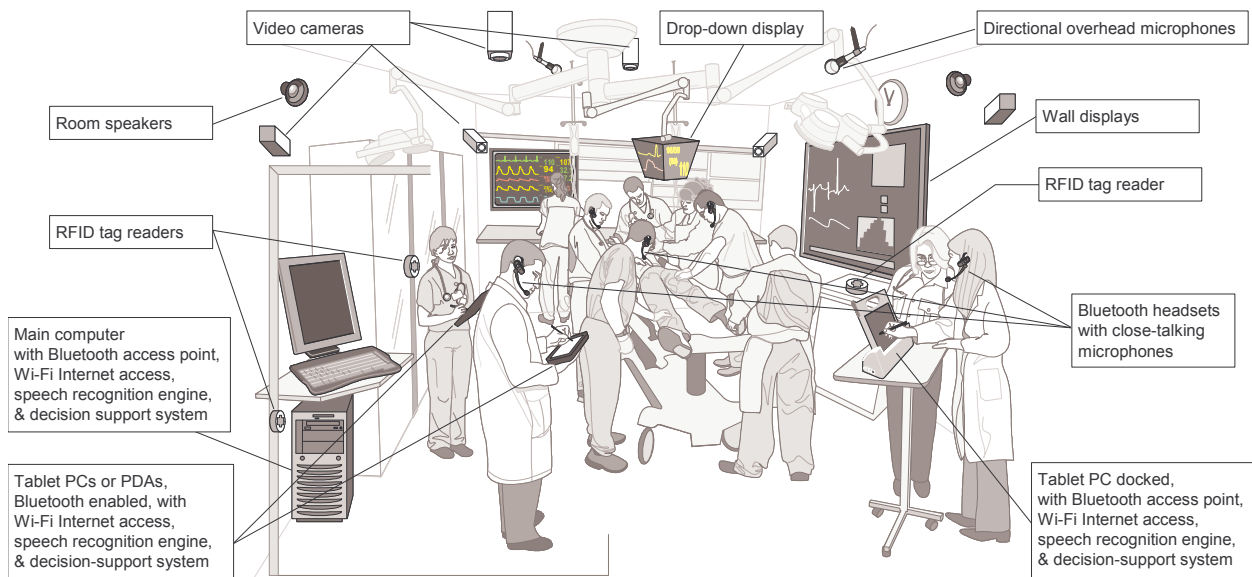


Figure 5: Hardware technologies of the a smart resuscitation room, TRU-Lab (Trauma Resuscitation Unit – Lab). Developed by CAIP.

There are various options for localizing the medical personnel in the trauma bay. When using camera images to calculate positions of objects in a room a preferred method is by using color histograms (Davis, 2002). But general policy in hospitals rule that proper clothing must be worn by all personnel. This means that a general resuscitation bay is mainly inhabited by identically

dressed people. Current tracking systems, especially tracking systems that work with multiple cameras, use information from clothing features like color to discriminate objects currently being tracked. When that source of information is lacking another method for discriminating moving objects needs to be used.

Although the research is only a small part of the project it contributes largely because location information of personnel can greatly increase the task evaluation that a reasoning system can provide.

The vision module will recognize people (face recognition), their location and orientation, and activity such as walking, reaching and pointing. Identifying individuals and their actions will be aided by knowledge of the typical location of individuals with different roles during resuscitation (e.g., the team leader typically is at the foot of the table) and the stable map of the furniture, other landmarks and patient location (**Figure 5**). The movement of individuals within this environment can also aid in speaker identification. Knowledge of the current and past location of individuals can be used to identify their role in the resuscitation.

It is very difficult to discern what people are doing based on video only. However, supplemented with other modalities and landmarks information, there are a number of standard tasks that the computer may be able to recognize automatically (e.g., inserting an intravenous) that may permit identification of individuals. Other important motions may help link moments in the video to specific actions that can be tagged.

Early systems for tracking people in controlled environments such as “smart rooms” or “parking lot surveillance” (Wern, 1997) (Wu & Nevetia, Oct 2005) have been successful in tracking individuals and locating their body parts. These systems are limited in their ability to track multiple people in crowded scenarios where visual occlusion is expected. Previous work on tracking multiple people under occlusion, however, has been based heavily on people appearance (such as color features) for discrimination, limiting the usefulness of this methodology in the domain of trauma resuscitation as will be explained below.

Tracking in crowded collaborative environments, such as trauma bay, presents major challenges due to:

- The need to track many actors, from 7 up to 20, and be able to resolve the data association problem. Tracking multiple targets is difficult since the state space dimensionality becomes high, posing problems for a tracking framework such as particle filters (Papageorgiou, 1998).
- The need to resolve interference by visual occlusion by other actors, furniture and equipment and the highly dynamic nature of their work (actors are walking, turning around or bending over the patient and handling equipment). In addition, actors may leave the field of view or even the room making them only partially visible or not at all. The tracker must be able to handle uncertainty due to occlusion and transient loss. The use of multiple cameras is essential to resolve occlusion.
- The absence of distinguishing uniforms. Scrubs or similar attire are commonly worn by providers of trauma care, effectively ruling out the use of discrimination based on clothing color. We need to find discriminative features for maintaining actor identity over time.
- The non-stationary background because objects get moved, rendering ineffective the traditional techniques, such as background subtraction.
- The absence of a constant directional view of the actors. During trauma resuscitation, providers will be attending to their work and will not be facing the camera all the time. For this reason, there will not be enough pixels on the person's face to allow for face recognition using traditional techniques. Training the face algorithm will also be challenging because the trauma team composition can change and may not be known in advance. A novel framework is required to discriminate between people faces in such settings. Adding to challenges for face recognition, team members may wear protective eyewear or surgical masks needed to prevent environmental contamination.

The problems outlined above may be partly alleviated by the limited volume to monitor, making it feasible to use several fixed cameras. The furniture and wall-fixtures are stationary and can serve as landmarks. Because visual tracking during trauma resuscitation is a significant challenge using existing visual tracking algorithms, this portion of our research work will likely lead to

novel solutions with potential applicability to diverse real-world applications. The framework that we propose is based on online adaptive feature selection and boosting to build representations of people's face and appearance from multiple views. Moreover, the vision module will work with speech recognition and other sensors to locate the individuals and disambiguate their actions.

1.4.2 Aggression detection in public transportation

Another application where this research could be useful is aggression detection in public transportation. The aggression in for example train compartments is a growing problem and can take many forms. Vandalism of train interior, aggression towards passengers or conductors and loud behavior in general. Because of this phenomenon less people choose the profession of conductor. Research has shown that aggression statistics have increased every year. 75% of the cases involve aggression towards conductors and within that 75%, 1 out of 10 incidents includes physical violence.

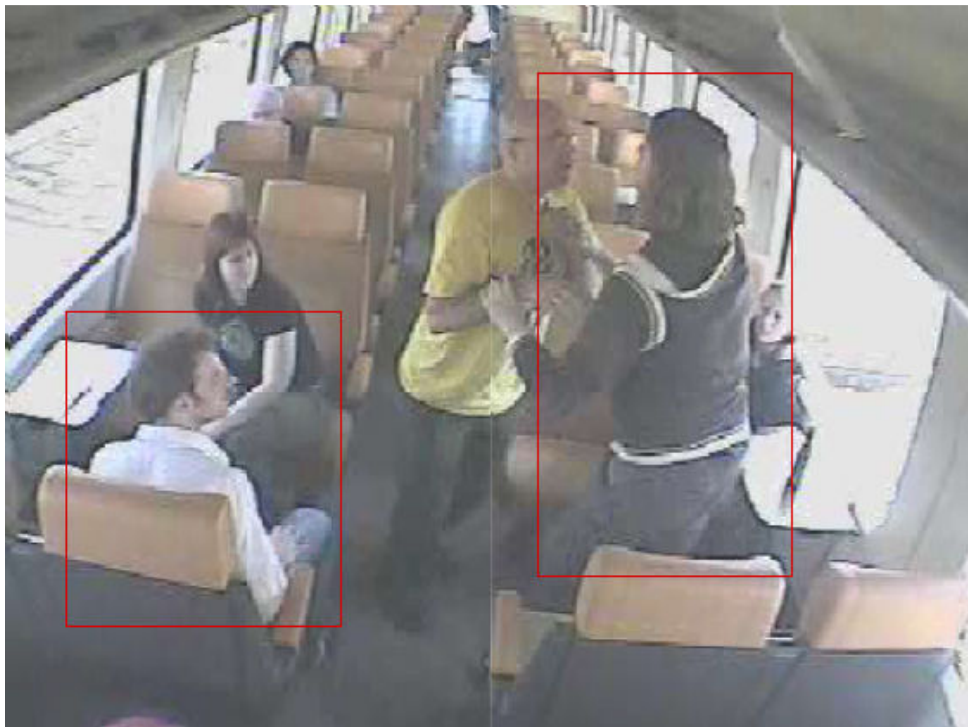


Figure 6: Video surveillance in train.

Public transportation usually places surveillance cameras near the ceiling. These cameras can be used to track people and monitor their behavior. By analyzing their position displacement unusual movement can be detected and can serve as a first warning to observation stations. Because this system works with face detection cluttered scenes can be observed from cameras mounted above the crowd.

When unusual movement is observed face detection can be used to detect aggression in facial expression. Verbal aggression with audio recognition.

1.5 Roadmap

This thesis is divided in 7 chapters. Each one of them explains one part of the research process. This chapter discussed the problem being presented and investigated the domain, the project goals and the challenges faced.

Chapter 2 will discuss related work and other research that has influenced this thesis. **Chapter 3** will introduce all the background theory and the description of the problem domain. **Chapter 4** shows the design of the proposed system and explains all the algorithms used. **Chapter 5** shows how from the design, the implementation was realized. First using a simulated environment and second with real-live cameras. **Chapter 6** shows the results achieved from the simulated environment and the real-live footage that was presented to the system. Finally **chapter 7** concludes this work, summarizes and discusses the results. Also possible future enhancements are listed.

Chapter 2

Related Work

2.1 Overview

There are several topics in this thesis for which there exists related work. This chapter will inform about interesting research and existing work that is used during this research.

2.2 Face detection

Face detection is defined as a computer technology that determines the locations and sizes of human faces in arbitrary (digital) images. It detects facial features and ignores anything else, such as buildings, trees and bodies. Face detection does not identify objects nor does it describe anything about facial expressions or emotions seen on the face.

Currently three notable face detector algorithms are possibly useful in solving this thesis' problem definition. The face detector that is most widely used and implemented in the standard OpenCV library is the **Viola and Jones** (Jones, May 2004) algorithm. Furthermore the university of Beijing is working on a successor based on algorithms proposed by Viola and Jones, created by **Huang and Haizhou** (Huang, Ai, Li, & Lao, 17-21 Oct. 2005). Finally **Idiap** is a possible face detector for the system. Below the design and algorithms used in the first two face detectors will be described in short overviews.

2.2.1 Viola and Jones

In 2004 Paul Viola and Michael Jones introduced a machine learning approach for visual object detection which is capable of processing images extremely rapidly and achieving high detection rates. This well known face detector is a variant of the AdaBoost (Yoav Freund, 1999) algorithm but implemented a new and radically faster approach in this field of research. It is based on the definition and rapid evaluation of simple features for object detection. These simple features are defined as 3 types of rectangular feature types.

- Two-rectangle feature type.
- Three-rectangle feature type.
- Four-rectangle feature type.

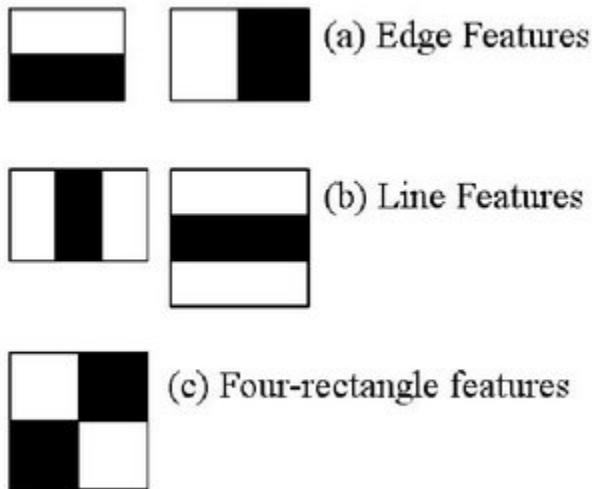


Figure 7: Visual example of the 3 types of rectangular features.

These rectangular features are chosen for their extreme computational efficiency. The algorithm then introduces the **integral** image. The integral image at location (x,y) is defined as the sum of the pixel values above and to the left of (x,y) , inclusive. Using the integral image representation one can compute the value of any rectangular sum in constant time. A single feature can be evaluated at any scale and location. It does not require a pyramid of scaling. Then a cascade of homogeneous classifiers is used, that can reject most negative examples early in the processes greatly reducing computation time.

By using the integral image method for fast calculation of features and a cascading network of classifiers it is possible to quickly remove non-face images to realize fast throughput.

The problem in this project with Viola and Jones is the fact that the face detector is trained on faces with one fixed rotation (e.g. frontal projection). Because of the multi-perspective nature of this research we need a detector that is capable of detecting faces in a variety of orientations.

More on this matter is discussed in **chapter 3**.

2.2.2 Huang and Haizhou

The Huang and Haizhou face detector characterizes itself in the ability to detect faces in multiple orientations. It introduces a novel **Width-First-Search (WFS)** tree structure that achieves higher performance in both speed and accuracy compared to Li's pyramid-structured MVFD (S. Z. Li, 2002) and Viola and Jones' decision-tree method for processing multi-view face detections. The WFS tree balances both the task of distinguishing between faces and non-faces and the identification of the pose of a face.

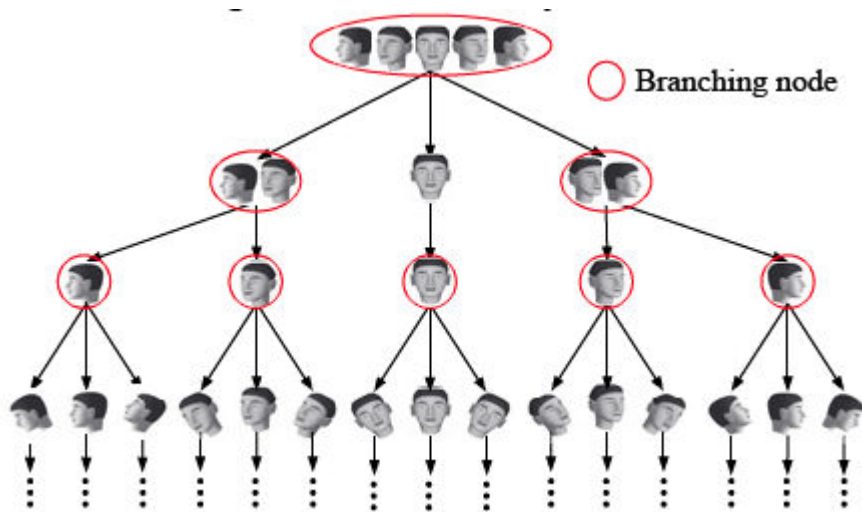


Figure 8: WFS tree structure used by Huang and Haizhou.

Using the WFS tree structure the detector is enhanced in both accuracy and speed. When we examine the output generated by this algorithm we see that both position and orientation of the detected faces are returned. **Figure 9** shows an illustration of how this looks.

An implementation of this face detector is not publically available. The Tsinghua University of Beijing was contacted to provide this face detector for research and educational purpose. But sadly the detector implemented already was commercially licensed and could not be shared with any educational institutions because of contractual agreements. This detector shows very promising results and especially the multi-view aspect of this face detector makes it very applicable for this project. Future use of this detector could have many benefits.

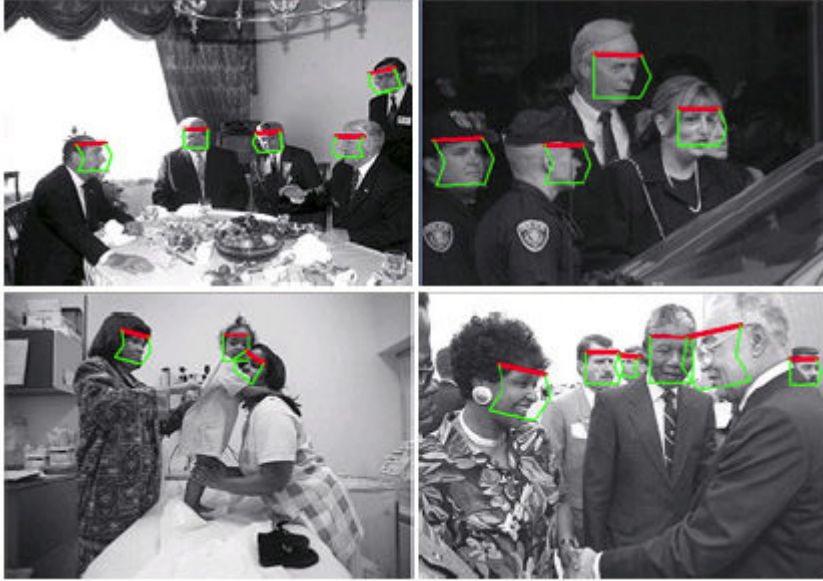


Figure 9: Example of detection results with the Haizhou algorithm.

2.3 Tracking methods

There are numerous tracking and detection algorithms, most of them based on single-camera usage. The problem with single-camera algorithms is the translation to 3-dimensional coordinates from a 2-dimensional camera view. Some tracking system present more useful algorithms for the domain of this thesis. Following is a list of the most closest related trackers to this research.

2.3.1 M2Tracker

The M2Tracker by Anurag, Mittal and Larry S. Davis (Davis, 2002) presents a basic framework from which this project can benefit. This system is capable of segmenting, detecting and tracking multiple people in a crowded scene also using a multi-perspective approach. The main working of the system concentrates on an entity called the **color model**. It notes the color distribution at different heights of the person being tracked. This information is used to discriminate between persons in the same environment. Because the domain of this thesis is limited to situations where people dress in similar clothing, using a color model in this system would not provide us with any useful information.

Other subjects used in the algorithms are more useful for this thesis. Geometrical calculations

and Bayesian occlusion aspects of the framework can work very well in this system as well. Furthermore the use of Kalman filtering to track object movement is also applicable to this situation.

The M2Tracker shows promising results but lacks the capability of tracking similar dressed people. This research will try solving exactly that problem.

Chapter 3

Theoretical Background and Problem Domain Description

This project combines various fields of research. In this thesis several new algorithms are designed and an iterative system is implemented integrating the communication between these algorithms. These algorithms and their integration is intensively described in **chapter 4**.

In order to explain and describe the system without concerning about theoretical details and characteristics of used methods, this chapter describes all fundamentals and background of the problem. These are categorized in the following topics:

- Face detection, explaining about the fundamentals of facial rotation and the parallel detection in this research.
- Environment setup. How the room is setup and what variables are used in the environment of the system.
- Vision. Describes the synchronization and calibration of cameras including basics about camera frames and views.

The diversity in the nature of this research makes it hard to just jump into the design and implementation of the program. Without a careful explanation of the general methods used and assumptions made in this research one could get lost in the detailed descriptions of the system. This chapter serves the proposed design and explains the problem domain description so it is easier to present the system more detailed in the following chapter. Fundamental concepts used in later chapters of this thesis are introduced here and methods used in the system are theoretically extensively explained.

3.1 Face detection

A reliable face detector is a very important part of this research in achieving its goals successfully. In the previous chapter the current state of technology was explained in the field of face detection. Although progress made with current developed face detectors is encouraging, most of these projects are difficult to make available for research and are often already commercially implemented. Both the effective and elegant face detector by Huang and Haizhou and the robust and fast face detector by Viola and Jones were described in **chapter 2**.

The OpenCV programming library includes the standard Viola and Jones face detector in its functions. The face detector is acknowledged as very reliable but is limited in the freedom available for different environments. The detector needs to be trained on a specific training set. This results in a detector that is very reliable in detecting faces that have a specific kind of orientation. One of the essential requirements of the system is the need for a face detector that is able to detect faces in multiple head orientations. The most basic orientation is a frontal face projection with the person looking straight to the camera.

The human head is able to rotate along three axes. And every rotation is identified by a specified term. The head is able to look left and right (*yaw*), up and down (*pitch*) and cock from side to side (*roll*).

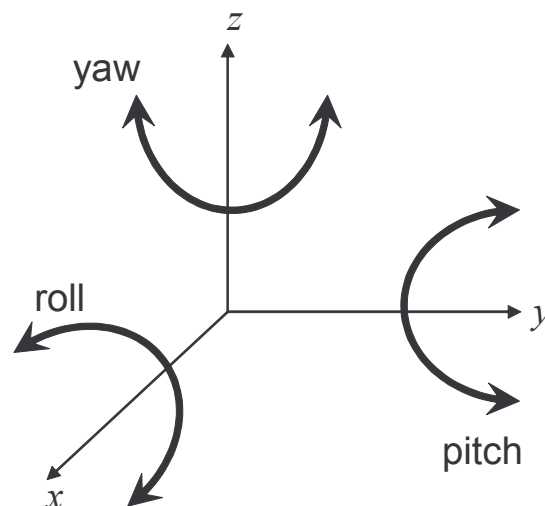


Figure 10: Coordinate system illustrating the three axes of rotation for a human head.

Because the basic Viola and Jones algorithm does not detect faces with variations in yaw, roll and pitch the basic face detector will be altered for this research. The theory of running detectors in parallel is used here.

3.1.1 Parallel Viola and Jones

The idea of the parallel Viola and Jones detector is very simple. Instead of using one detector trained on a specific set of faces, we run multiple detectors in parallel all trained with a specific training set. Each single face detector is trained with a data-set consisting of faces with identical yaw, roll and pitch. But every detector is trained on a different head orientation. The face detector for this research consists of three Viola and Jones face detectors running in parallel. The first detector is trained on a data-set consisting of faces in frontal projection and the other two detectors are trained with data-sets with a 90 degree yaw to the left or right respectively. The downside of this approach is that detecting faces will take three times as long as with a single Viola and Jones detector. Other detectors that incorporate multi-angle face detection like the Haizhou use pyramid algorithms to limit the searching time.

3.2 Environment setup

Several applications were proposed in the introduction. Every implementation of an application starts by creating the basic environment first. **Figure 11** illustrates this basic setup for this environment. The environment is a closed area that is monitored by cameras. To setup the basic environment the width and length of the ground floor is recorded and on every corner of this area a video camera is positioned. Video cameras are positioned slightly above eye level and are pointed toward the centre of the room. The cameras are slightly pitched to the floor and their position and look-at point is recorded. A virtual coordinate system is created with its origin in the centre of the room's ground floor. Locations of persons in the environment are measured along that coordinate system.

During runtime an important concept of the system is that of the **room model** \mathfrak{R} . The room model $\mathfrak{R}(t)$ represents the status of the environment on time t . The status of the environment is modeled by a few parameters.

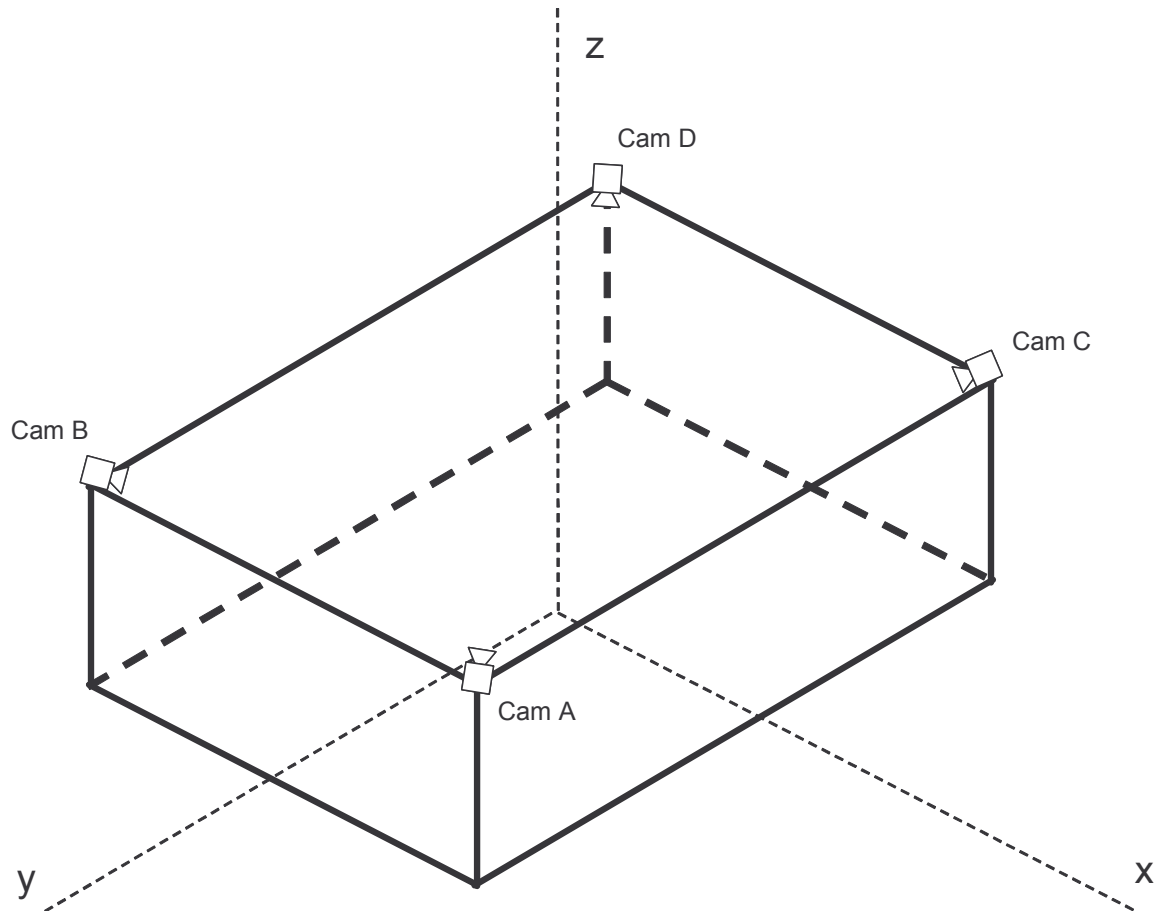


Figure 11: Illustration of the basic environment consisting of a closed environment and four cameras.

The room model contains a location map of estimated locations of the persons currently in the room, and their orientations. A **face model** \mathfrak{F} contains features about each person's face gathered during tracking in multiple steps in time. Examples of face model features are:

- Color distribution of the face.
- Relative locations of eyes, mouth or nose.
- Shape of the eyes, mouth or nose.

These features are represented by number-matrices representing the salient facial features. The room model \mathfrak{R} is the key output of the system that is the main subject of this thesis. It is continually updated, based on input images, to reflect the movement of persons. Together with the camera images the room model \mathfrak{R} is the main input to the system. It serves a temporal role in the system and gives information about previous time steps to aid the algorithm at time t .

3.2.1 RFID tags

Radio-frequency identification (RFID) is an automatic identification method, relying on storing and remotely retrieving data using devices called RFID tags or transponders.

An RFID tag is an object that can be applied to or incorporated into a product, animal, or person for the purpose of identification using radiowaves. Some tags can be read from several meters away and beyond the line of sight of the reader.

To aid tracking in the environment the RFID tags can be placed on people in the room. By using tag readers the information can be accessed if a tag is close enough to the reader. Tag readers will be installed in the entry to the environment and on "landmark" locations throughout the area. The readers will operate at short range (1 meter or less). When an inhabitant of the environment approaches a landmark location, their ID will be read out by the reader and the event will be time-stamped. RFID tags aid the tracking in two ways:

- Creating the possibility for recognizing instead of only discriminating people and tracking their motions. An identification label can be added to people.
- More accuracy during the classification phase, where detected face regions are classified to a certain person in the environment.

RFID tags are not a necessity to the system but add useful information for applications where identification is also wanted. Current RFID technology is not precise enough to function as a positioning system but can be used in the method described above. Motion of people can influence the RFID reading. For identifying people in closed environments people will usually stay at landmarks for a longer period of time and use of RFID can be implemented.

3.3 Vision

3.3.1 Video cameras

Four video cameras are positioned in every corner of the room. Each camera produces an image sequence while recording. To use the information that we see on images produced by the camera we often need to trace a ray into the scene through the specified object in the camera image. One basic concept of the camera is its **optical center**. This physical point in the camera is where every light ray from outside goes through the virtual image and gets bundled in one point in the camera. If we want to trace a ray through the scene we use the optical center and the point in the camera image to set up a line equation.

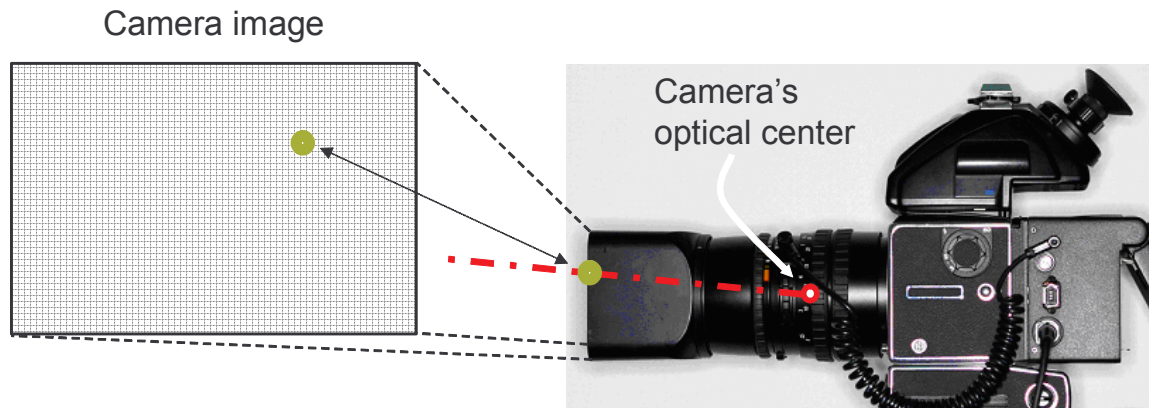


Figure 12: Illustration of ray tracing using the camera's optical center and a detected object in the camera image.

Frames and views

Four cameras oriented to the center of the scene are continuously recording to monitor the situation in the room. The array of images returned when taking one timestep is called a **frame**. This frame in time consists of a set of images, four in this case. Each image is recorded by one camera. A frame can be considered a certain moment in time. Every image that an individual camera will produce is called a **camera view**. So if all cameras are operating every frame consists of an equally amount of views. Views are made up of a certain amount of pixels that the camera records. The size of this pixel matrix depends on the camera resolution. **Figure 13** illustrates the concept of a camera view.

A view j represents a picture of the room from camera j taken at time t . Each pixel in view j is labeled by an (n,m) location in the view where n represents the horizontal coordinate and m represents the vertical coordinate in the view.

During runtime, from each camera j person f is detected by face detection from the associated view j . Detection of a face results in a rectangular **face region** in the view j .

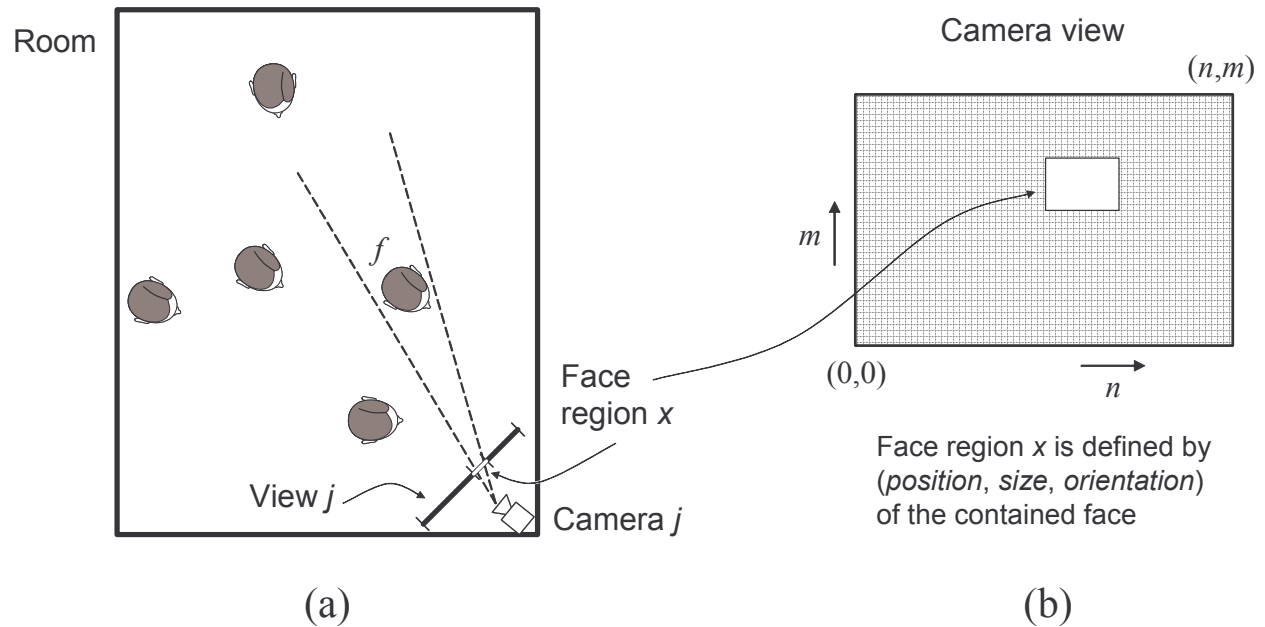


Figure 13: Illustration of concepts used in the localization and tracking system.

Every face region is defined by two pixel coordinates in the view. The first coordinate defines the upper left corner of the face region and the second coordinate defines the lower right corner. Together they define the boundaries of the rectangle. **Figure 14** illustrates the detected face regions and the stored pixel coordinates.

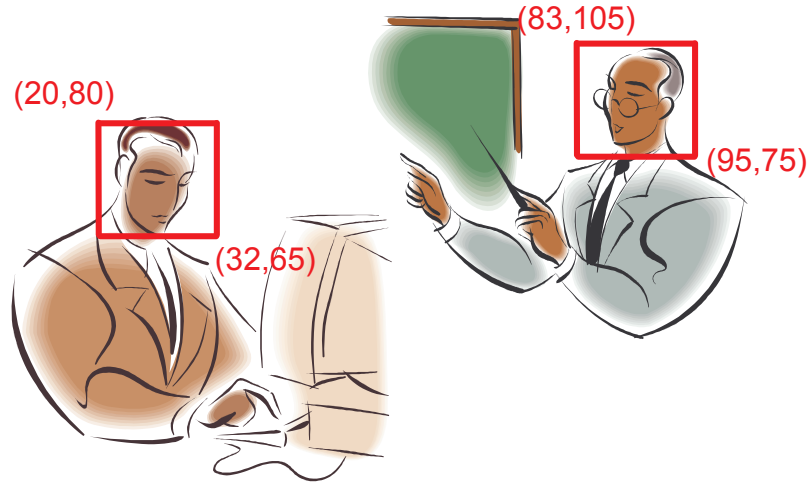


Figure 14: Illustration of the two coordinates stored for every detected face region.

Furthermore every face region is labeled by the head orientation detected by the parallel Viola and Jones face detector. The head orientation is defined by a vector with three entries noting the degree of yaw, roll and pitch:

$$orientation = \begin{bmatrix} yaw \\ roll \\ pitch \end{bmatrix}. \quad (3.1)$$

Cameras in the scene and the image sequences they provide need to be calibrated to the environment and synchronized with each other in order to function properly in the program.

Calibration

Calibration of the camera is the process of defining the position and the orientation of the camera in the environment so that a ray can be traced through the 3D scene (world-coordinates) for every pixel in the camera view (screen-coordinates). An equation of this ray is defined by two intersection points in the world-coordinate environment.

- The camera's optical centre.
- The pixel on the camera view presented in world-coordinates.

Ray tracing is necessary when the images of multiple cameras are combined and geometrical calculations about locations of objects are made in the 3-dimensional environment. The first step in calibration is defining the location of the camera's optical centre in the environment. This location is retrieved by measuring the dimensions of the environment and the camera's location within.

For this application we will use passive calibration. The camera parameters are assumed to be fixed, so the calibration is only performed once in advance. Using **Direct Linear Transformation (DLT)** we can describe a transfer function from a 3D coordinate to the corresponding 2D image coordinate and visa versa. DLT falls under implicit calibration, meaning only some parameters attached to the camera are required. There is no need to find out the exact value for each physical parameter of the camera used.

We describe the transformation from a 3D world coordinate $[x_w, y_w, z_w]^T$ to the corresponding 2D image coordinate $[x_i, y_i]^T$ using a linear equation:

$$\begin{bmatrix} t_1 \\ t_2 \\ t_3 \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \end{bmatrix} \cdot \begin{bmatrix} x_w \\ y_w \\ z_w \\ 1 \end{bmatrix}. \quad (3.2)$$

The image coordinates are then given by:

$$x_i = t_1 / t_3 \quad (3.3)$$

$$y_i = t_2 / t_3 \quad (3.4)$$

Substituting 3.3 and 3.4 into 3.2 and expanding the matrix product yields:

$$x_i t_3 = a_{11} x_w + a_{12} y_w + a_{13} z_w + a_{14} \quad (3.5)$$

$$y_i t_3 = a_{21} x_w + a_{22} y_w + a_{23} z_w + a_{24} \quad (3.6)$$

$$t_3 = a_{31} x_w + a_{32} y_w + a_{33} z_w + a_{34} \quad (3.7)$$

Substituting 3.7 into 3.5 and 3.6 yields two equations with 12 unknowns:

$$a_{11} x_w + a_{12} y_w + a_{13} z_w + a_{14} - a_{31} x_w x_i - a_{32} y_w x_i - a_{33} z_w x_i - a_{34} x_i = 0 \quad (3.8)$$

$$a_{21} x_w + a_{22} y_w + a_{23} z_w + a_{24} - a_{31} x_w y_i - a_{32} y_w y_i - a_{33} z_w y_i - a_{34} y_i = 0 \quad (3.9)$$

The 12 unknowns can be solved via using $N > 6$ points in terms of 3D world coordinates $x_w^j = [x_w, y_w, z_w]^T$ and corresponding 2D image coordinates $x_i^j = [x_i, y_i]^T$:

$$\begin{bmatrix} x_w^1 & y_w^1 & z_w^1 & 1 & 0 & 0 & 0 & 0 & -x_w^1 x_i^1 & -y_w^1 x_i^1 & -z_w^1 x_i^1 \\ 0 & 0 & 0 & 0 & x_w^1 & y_w^1 & z_w^1 & 1 & -x_w^1 y_i^1 & -y_w^1 y_i^1 & -z_w^1 y_i^1 \\ x_w^2 & y_w^2 & z_w^2 & 1 & 0 & 0 & 0 & 0 & -x_w^2 x_i^2 & -y_w^2 x_i^2 & -z_w^2 x_i^2 \\ 0 & 0 & 0 & 0 & x_w^2 & y_w^2 & z_w^2 & 1 & -x_w^2 y_i^2 & -y_w^2 y_i^2 & -z_w^2 y_i^2 \\ x_w^N & y_w^N & z_w^N & 1 & 0 & 0 & 0 & 0 & -x_w^N x_i^N & -y_w^N x_i^N & -z_w^N x_i^N \\ 0 & 0 & 0 & 0 & x_w^N & y_w^N & z_w^N & 1 & -x_w^N y_i^N & -y_w^N y_i^N & -z_w^N y_i^N \end{bmatrix} \begin{bmatrix} a_{11} \\ a_{12} \\ a_{13} \\ a_{14} \\ a_{21} \\ a_{22} \\ a_{23} \\ a_{24} \\ a_{31} \\ a_{32} \\ a_{34} \end{bmatrix} = \begin{bmatrix} x_i^1 \\ y_i^1 \\ x_i^2 \\ y_i^2 \\ x_i^N \\ y_i^N \end{bmatrix} \quad (3.10)$$

We set $a_{34}=1$ because scaling is irrelevant. Using least squares we can calculate the other 11 unknown. Now we can find a corresponding pixel of a 3D point. To find a 3D point given a image pixel we will use 3D reconstruction. 3.8 and 3.9 can be rewritten as:

$$\begin{bmatrix} a_{14} - a_{34}x_i \\ a_{24} - a_{34}y_i \end{bmatrix} = \begin{bmatrix} -a_{11} + a_{31}x_i & -a_{12} + a_{32}x_i & -a_{13} + a_{33}x_i \\ -a_{21} + a_{31}y_i & -a_{22} + a_{32}y_i & -a_{23} + a_{33}y_i \end{bmatrix} \begin{bmatrix} x_w \\ y_w \\ z_w \end{bmatrix} \quad (3.11)$$

From 3.11 we can see that we do not get one unique solution for x_w . We need two x_i that correspond to the same x_w . to reconstruct it uniquely. For this application we do not need a unique solution because we only project rays into the scene and we don't need exact 3d points. With 3.11 and least squares we can construct a ray using the optical center of the camera.

Synchronization

To use the dependency between objects detected in different views, every view j in a frame needs to be captured at the exact same moment in time. This means that when a frame is recorded every camera needs to record an image at the exact same time and add that image as a view to the frame. The theory behind synchronization of a batch of cameras starts with binding them through a network connection. Software for capturing images from each camera can then be synchronized. When using modern cameras two steps that are synchronized are:

- The starting time of capturing.
- The **frame rate**, recorded in frames per second.

Then every camera starts at the exact same time with recording and records the same amount of frames per second. Over a time span of several minutes the exact same amount of images is then recorded for each camera and each image is synchronized with the other cameras.

When older equipment is used it is possible that the capturing frequency varies between cameras. When this variety is large cameras will capture out of synch material. When using older cameras it is important to synchronize at frequent intervals. When using offline material, frames can also be distributed evenly over the recording so they are synchronized with each other.

Chapter 4

Proposed Design

In this chapter the proposed design for the core functionality of the system is explained. To track multiple people and estimate their locations we need information from multiple perspective, by extending the environment to a multi-camera setup we supply this extra information. By definition, if the functions of two lines in 3d space are known, it is possible to carefully estimate the place where those lines are closest together (i.e. the most likely position of the tracked object), meaning when two calibrated camera views detect the same person, that person can be localized by geometrical calculation.

The central part of the system begins when a frame of n views is recorded from the n (in this case 4) cameras. **Figure 15** illustrates the process that follows in every time step. In the following sections this system is explained intensively part by part but a short description is given here to explain the main flow of the process. Input to the system consists of one frame of n views and the room model $\mathfrak{R}(t-1)$. These inputs are fed to the face detector. The Parallel Viola and Jones algorithm is used for face detection in every view of the frame. The detected facial regions are labeled with the identity of a person in the room, the classification phase. Facial regions that can not be labeled with the identity of a person will remain unlabeled. Using the classification of the facial regions likelihood maps are created by bundling the facial regions that were labeled similar. Using the probability of its classification every facial region is represented on the map by a Gaussian model. Combining identically classified facial regions result in a likelihood map for every person. From a likelihood map a location for a person is predicted and this individual location map is the input to a Kalman Filter. The filter takes into account previous time steps to smooth out the motion of the person predicted by the system. The filter produces a final map that states the predicted location of the individual person. All final locations for every person are combined in the room model $\mathfrak{R}(t)$ and are fed back as new input to the system. A new frame arrives on time $t+1$ and the room model $\mathfrak{R}(t)$ is then used in the next loop through the system.

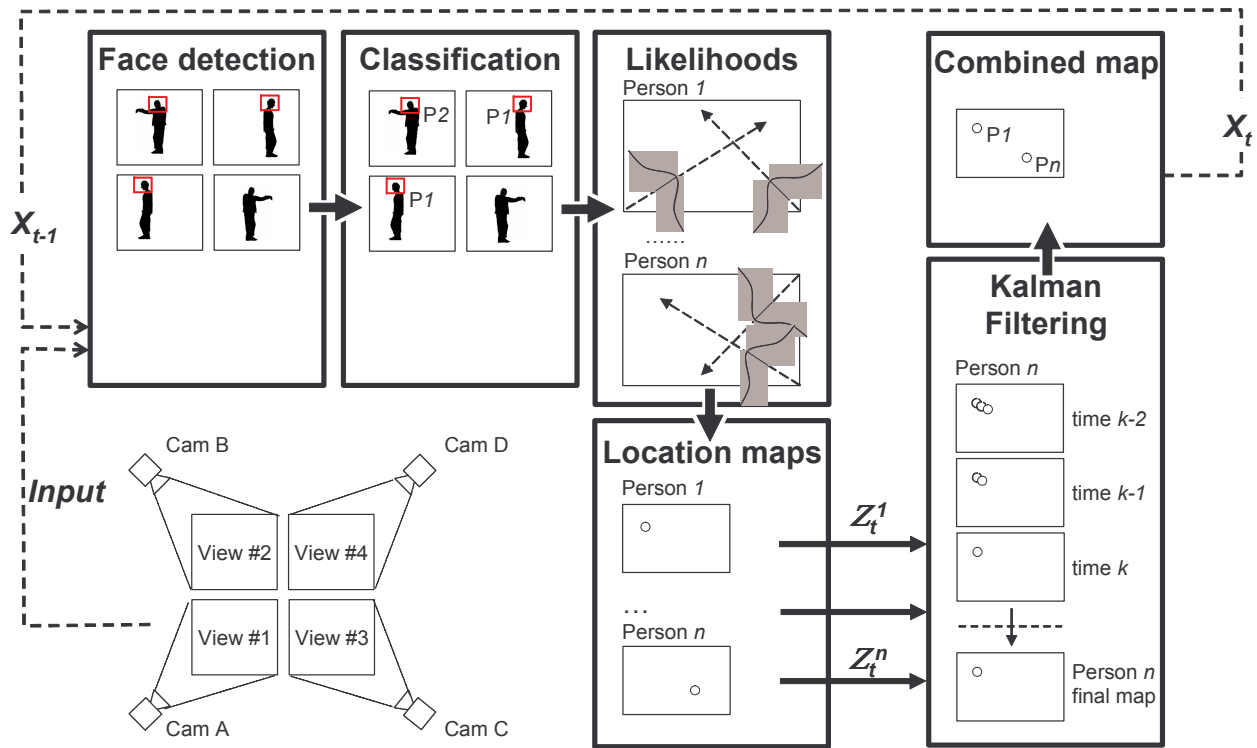


Figure 15: Illustration of the main flow of the program.

This diagram above shows the six phases the system walks through. We can summarize these phase once more:

1. Face detection.
2. Classification.
3. Likelihoods.
4. Location maps.
5. Kalman filtering.
6. Combined location map.

Each separate phase is explained in detail in the following sections.

4.1 Face detection

For every time t all n views are recorded from all n cameras and fed into the face detector. The objective of this phase is to localize every visible face in every view. An ideal face detector

would detect every face in every view. Variations in yaw, pitch and roll would have no effect on such a detector. Such an ideal detector is able to label every detected facial region with its associated yaw, roll and pitch. It would detect every face and would never produce a false positive. Current face detectors are not capable of such precision. Although more advanced face detectors are being developed none of them are currently available for research purposes. Therefore this research uses the Viola and Jones face detector that is openly available in the OpenCV function library. **Chapter 3.1** explains the methods to use the face detector as an detector for variations in yaw, pitch and roll.

The Viola and Jones detector is not *100%* accurate. This means some faces will have two or more detected face regions. Some detected face regions are false-positives and some faces in views remain undetected. Each of these situations needs to be processed when occurring.

Double detection of identical face

A detected face region *A* and a detected face region *B* are detected from the same face if one of two conditions is met illustrated by **Figure 16**. The second stored coordinate of face region *B*; the lower right corner of the rectangle, should be in the region right and below of the centre point of the rectangle face region *A*. Or the first stored coordinate of face region *B*; the upper left corner of the rectangle, should be in the region up and left of the centre point of the rectangle face region *A*. In both conditions the stored coordinates should be within the boundary of face region *A*.

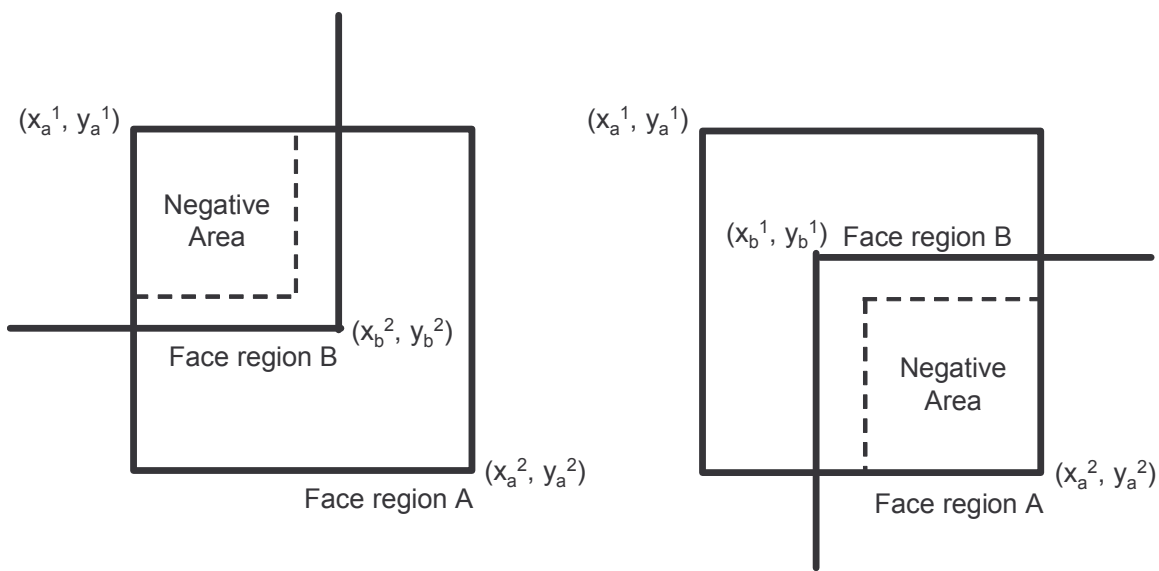


Figure 16: Illustration of the condition to be met for two face regions containing the same face.

These conditions can both be written in one formula

$$\begin{bmatrix} (x_a^2 - x_a^1)/2 + x_a^1 \\ (y_a^2 - y_a^1)/2 + y_a^1 \end{bmatrix} \leq \begin{bmatrix} x_b^2 \\ y_b^2 \end{bmatrix} \leq \begin{bmatrix} x_a^2 \\ y_a^2 \end{bmatrix} \vee \begin{bmatrix} x_a^1 \\ y_a^1 \end{bmatrix} \leq \begin{bmatrix} x_b^1 \\ y_b^1 \end{bmatrix} \leq \begin{bmatrix} (x_a^2 - x_a^1)/2 + x_a^1 \\ (y_a^2 - y_a^1)/2 + y_a^1 \end{bmatrix} \quad (4.1)$$

This situation occurs when both a facial region is detected with zero angle yaw and a facial region with yaw for the same detected face. This usually means the detector can not give decisive evidence if the orientation of a face is in either of those directions. If detected that the two facial regions belong to the same face (4.1) both face regions are deleted. Then a new facial region V is created by taking the mean from both coordinates for all identical face regions (4.2) and this new region is labeled with the mean of both yaw angles.

$$\begin{bmatrix} x_{newregion}^1 & \dots & x_{newregion}^2 \\ y_{newregion}^1 & \dots & y_{newregion}^2 \end{bmatrix} = \begin{bmatrix} (\sum_{i=A}^N x_i^1)/N & (\sum_{i=A}^N x_i^2)/N \\ (\sum_{i=A}^N y_i^1)/N & (\sum_{i=A}^N y_i^2)/N \end{bmatrix} \quad (4.2)$$

False positives

Whenever the face detector detects a face and produces a face region the possibility exists that this face region is a false positive. These false positives influence the accuracy of the system and should occur as few as possible. A preemptive measure is supplying the face detector with better training data. Also a better resolution and quality of views will benefit the accuracy of the detector.

When a false positive does get detected the face region will only be accepted when it meets a certain size-threshold δ . This threshold is dependant on the location of cameras in the room and the size of the room itself. It is reasonable to assume that faces appearing in the scene will not be smaller then threshold δ . This enforces a stricter detection of faces.

If a false positive does get stored as a valid face region the next phase of classification takes further care of this problem.

Undetected faces

The possibility of a visible face not being detected by the Viola and Jones detector exists. More accurate, detecting every face in all views is a daunting task due to variations in yaw, roll and pitch. The fact that the system relies on multiple cameras for tracking makes the program less dependant on the amount of true positives of the detector. A face does not need to be detected in all cameras for the person to be tracked correctly. But every detected face adds more certainty to the tracking of each person. The working of the face detector is very deciding in this project. By training the face detector with suitable data and supplying views of higher quality we try to reach the face detector's best potential.

Facial occlusion

Faces appearing in the scene can be occluded by various objects in the room or other people. The description of the Viola and Jones detector (**chapter 2.2.1**) mentions that the eye feature is the most discriminating rectangular feature. The paper claims that partial occlusion of the lower face still leads to decent detection results. Some of the possible applications of this system can present difficulties for face detection. In the emergency bay for example physicians often wear mouth caps while stabilizing a patient. This considerably occludes the face and presents a problem during face detection. With the rectangle features of the eyes possessing the most discriminating abilities of them all, the detection of those faces still remains possible.

4.2 Classification

Classification has the main objective of labeling every detected face region. The labels come from the set Ω of current people present in the room. The classification algorithm is supplied with a frame of n views each of them run through the parallel Viola and Jones face detector. Each face region in a view is stored as two coordinates for both the upper left corner and the lower right corner together with an estimate of the variation in yaw, pitch and roll of the head. The reason for classifying these face regions is the potential of localizing objects if it is known that several face regions belong to that same identical object.

The aim of the classification is to label every single object represented by a face region with a inhabitant number from set Ω . Unfortunately not every face region should be labeled with a person from set Ω . Two events cause these face regions to be detected from an object not in Ω .

- A new person enters the room, set Ω does not yet include this person and the correct label for classifying the face region is not available
- The face detector supplies a false positive.

Both situations are discussed later in this section. Temporal information is needed for the classification phase. Mainly knowledge retrieved from the room model \mathfrak{R} at previous time steps give clues about the correct labeling of face regions.

- From the location of a person in room model $\mathfrak{R}(t-1)$ estimate the probability of that same person showing up at a location calculated by a face region labeled to that person.
- From the variation in yaw, roll and pitch of a person in room model $\mathfrak{R}(t-1)$ estimate the probability of that same person showing current variation in yaw, roll and pitch calculated by a face region labeled to that person.
- By sensing a person's location by reading an RFID tag estimate the probability of that same person showing up at a location calculated by a face region labeled to that person. This assumes the labels in set Ω are connected to the identification given by the RFID tags. This connection is created at the very end of the program and will be discussed in **chapter 4.7**.

The classification process is based on the above list of available information. To describe the process of face classification the algorithm is now explained in detail step by step. Given a rectangular face region x in the input view, what is the probability that the face of person f is contained in x . We call this the **posterior probability**. The **classification problem** aims to maximize that conditional probability

$$\max_f P(f | x) \tag{4.3}$$

The classifier outputs the person f for which the probability $P(f|x)$ is maximum. Rewriting this equation using Bayes' theorem gives

$$\max_f P(f | x) = \max_f P(x | f) \cdot P(f) \quad (4.4)$$

The maximization problem is now broken up into two different parts: the *likelihood* and the *prior*. The **prior probability** $P(f)$ gives the probability that a random face region x contains an image of a person f . The prior at the current time t is estimated using the room model $\mathfrak{R}(t-1)$ at the previous time moment $t-1$. When calculating the prior, we need to account for the possibility that other persons occlude the person f , which is done as follows.

Figure 17 illustrates the parameters used in the calculation of the prior $P(f_0)$ in a camera j for the person f_0 . From the room model $\mathfrak{R}(t-1)$, we know the previous estimated locations of all persons in the room. Only the persons that are closer to the camera j than person f_0 can occlude the person f_0 in the view of camera j . In the example illustrated in **Figure 17**, persons f_1 and f_2 are further away, so they are not considered as possible occluding person f_0 . Only both persons f_3 and f_4 can possibly occlude person f_0 in the camera j view. Let Φ denote the set of indices of the persons that can occlude person f_0 in the camera j view. In our example, $\Phi = \{3, 4\}$.

The probability of the occlusion also depends on their distance from the ray r tracing from the optical centre of camera j to person f_0 . This probability is modeled by a Gaussian distribution: the closer the person f_k is to the connecting ray r , the higher probability that it occludes person f_0 . Given the distance d_k of person f_k from ray r connecting camera j and person f_0 , the probability of occlusion is denoted as $L(d_k)$. And L represents the Gaussian distribution with $\mu=0$ and $\sigma^2=1$. Then, the prior for a general person f is computed as

$$P(f) = L(d_f) \cdot \prod_{k \in \Phi} (1 - L(d_k)) \quad (4.5)$$

The **likelihood** $P(x|f)$ represents the probability of face region x (represented by location, size, and orientation of the head in the view), given that the face detector reports that it contains the face of person f .

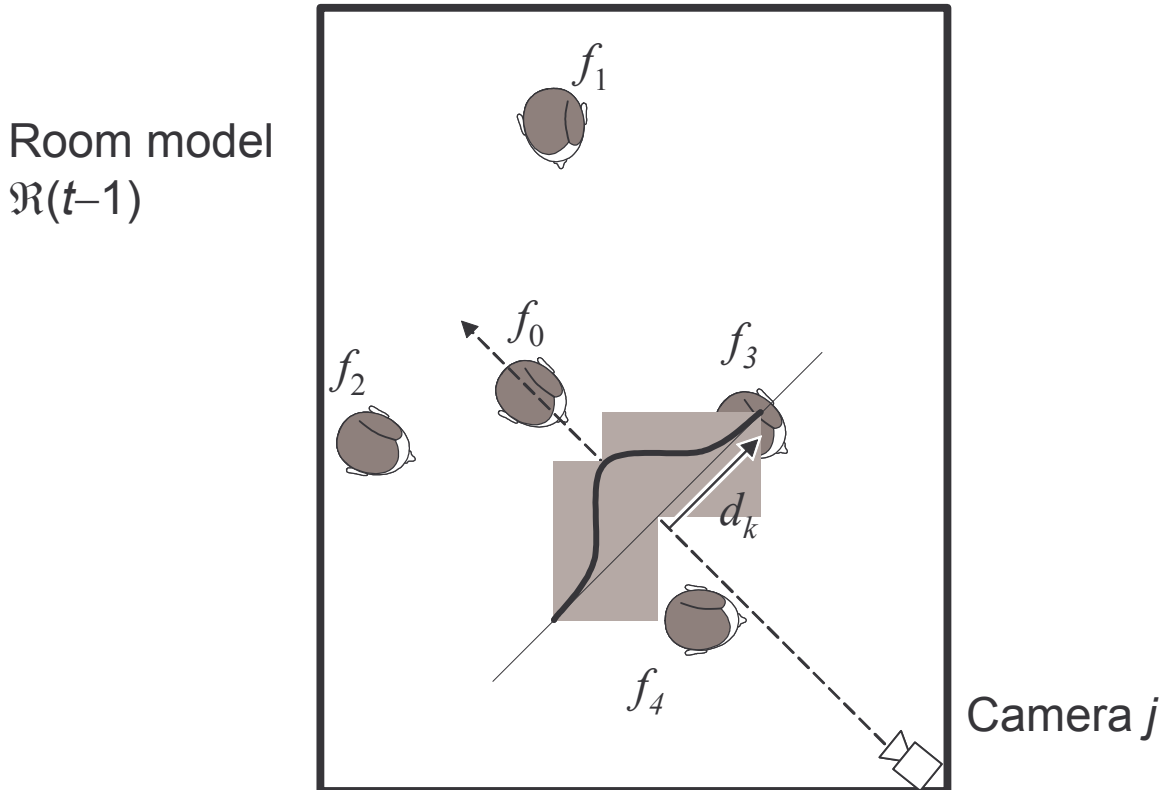


Figure 17: Illustration for the calculation of the prior probability $P(f_0)$ in camera j at the current time t , given the room model $\mathfrak{R}(t-1)$ at the previous time moment.

The key in estimating this probability is that we know that the face region belongs to a certain person, which allows us to compare the new values of location and orientation in the room model $\mathfrak{R}(t-1)$. As illustrated in **Figure 18**, given the current view in camera j , we draw a ray through the optical center of the camera and the center point of the face region x . Then we draw a line orthogonal to the ray that passes through the location of person f in the room model $\mathfrak{R}(t-1)$. Next, we calculate the distance Δl_f from the ray to the old location of person f in $\mathfrak{R}(t-1)$ along this orthogonal line. Then we input distance Δl_f to a Gaussian distribution centered at the person f old location in $\mathfrak{R}(t-1)$, which gives us the probability of the location displacement. Similar procedure is performed for calculating the probability of the orientation displacement. We take the old person f variation in yaw, roll and pitch from the room model $\mathfrak{R}(t-1)$. Next, we calculate

the displacement in yaw $\Delta\alpha_f$, pitch and roll between the values from the room model $\mathfrak{R}(t-1)$ and the new detected values. Then we input these displacements to a Gaussian distribution which gives us the probability of the orientation displacement.

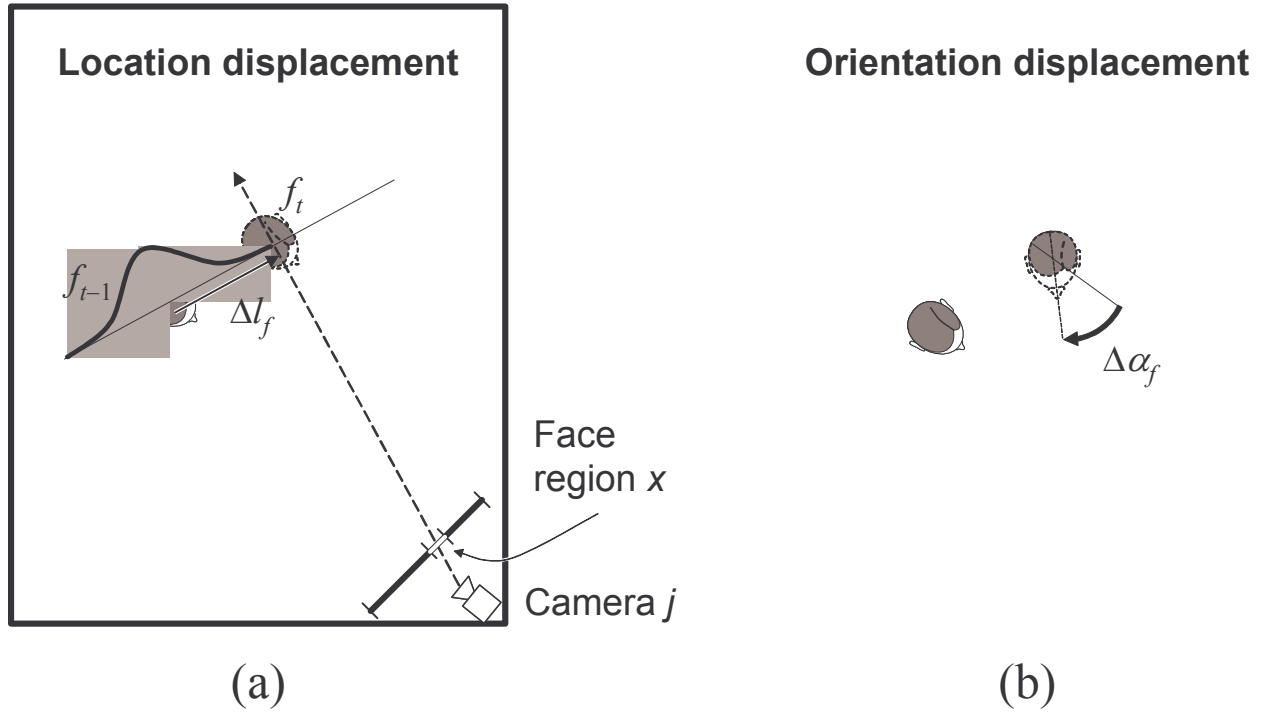


Figure 18: (a) Illustration for the calculation of the location displacement Δf over time in the room model \mathfrak{R} , which is part of the likelihood $P(x|f)$ calculation, in camera j at the current time t . (b) Orientation displacement $\Delta\alpha_f$ can include all three rotations: yaw, pitch, and roll.

The likelihood $P(x|f)$ is then calculated as a product of these two probabilities: the location displacement and the orientation displacement.

$$P(x | f) = P(pos(f)_t | pos(f)_{t-1}) \cdot P(ori(f)_t | ori(f)_{t-1}) \quad (4.6)$$

Then the classification problem (4.4) can be solved by multiplying likelihood and prior for every person f . The face region is then labeled to the person f with the highest posterior probability.

Classification accuracy

The result of the classification problem gives us the most likely person f to have its face contained in face region x . This classification has a final posterior probability. This probability tells something about the certainty of classification. A very low posterior probability even for the most likely person f can emphasize the possibility of a falsely labeled face region. Two reasons for this low posterior probability are the entry of a new person into the environment or the face region being a false positive. To ensure accuracy in the classification a face region will only be classified if the posterior probability of that region belonging to the most likely person f is higher than a certain threshold Θ . If the most probable case does not meet this threshold the face region remains unlabeled and further processing is given to the next phase in the model: the creation of likelihood maps.

$$Label(x) = \begin{cases} \text{"no - label"}, & P(f | x) < \Theta \\ f, & P(f | x) \geq \Theta \end{cases} . \quad (4.7)$$

Similar classification

It is important that two face regions in the same view are not classified to the same person. This means that every person from set Ω can only be classified to one face region in every view.

When this constraint is not applied ambiguity during the creation of likelihood maps appears and thus should be avoided. During face detection face regions that contain an identical face are combined together into one single face region. This implies that in every view no two regions are present containing an identical face. Then by definition it is incorrect to classify two face regions to the same person f .

When a situation occurs where person f has the highest posterior probability in two distinct face regions

$$\max_f P(f | x_1) = \max_g P(g | x_2) . \quad (4.8)$$

The face region with the highest probability will be labeled to person f . The other region will either pick the second most probable classification

$$Label \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{cases} \left(\begin{matrix} f \\ \max_{g=\Omega \cap f} P(g | x_2) \end{matrix} \right) P(g | x_2) > P(f | x_1) \\ \left(\begin{matrix} \max_{f=\Omega \cap g} P(f | x_1) \\ g \end{matrix} \right) P(g | x_2) \leq P(f | x_1) \end{cases}, \quad (4.9)$$

or the face region will remain unlabeled if it does not satisfy the threshold constraint (4.7).

4.3 Likelihoods

Objective of this phase is the creation of an individual likelihood map for every person in set Ω and the introduction of new persons to set Ω .

After completion each individual map M_f will tell us the likelihood of a person f standing at a location l_m on this map. The map represents the ground floor of the environment people are walking in.

The input of this phase is a set of labeled or unlabeled face regions each linked to a view. In this phase we will combine the information from different views and utilize the dependency of identical objects in distinct views. Basically the same process is repeated for every person in set Ω including a special process for all face regions that remained unlabeled.

Given is person f from set Ω . We gather all face regions that were classified to this person f . This creates a set of face regions G each one linked to a unique view. The constraint from equation (4.9) defines that every view is only represented once in G . **Figure 19** illustrates the following steps. For every face region $x \in G$, part of view v , trace a ray r through the optical center of camera v and the center of face segment x . Along this ray r project a perpendicular Gaussian distribution on the likelihood map of person f over the full length of ray r . This distribution is defined as

$$N(0, P(f | x)) \tag{4.10}$$

and is dependent on the posterior probability of the classification problem that resulted in the labeling of the face segment x to person f . A higher posterior probability will result in a higher likelihood around zero. To calculate an individual likelihood for location l_m we calculate the shortest distance from l_m to ray r and input that in the Gaussian distribution. For every location l_m on map M_f we multiply the likelihoods for every individual projected Gaussian distribution for all face regions.

$$l_m = \prod_{x \in G} L(\text{dist}(l_m, r_x)) \tag{4.11}$$

Figure 19 shows how two rays cross together at a location l_m on map M_f . When two Gaussians are projected along these rays the crossing of rays will result in a high likelihood circle shaped area around this crossing. This likelihood map will function as an estimator for the location of a person f .

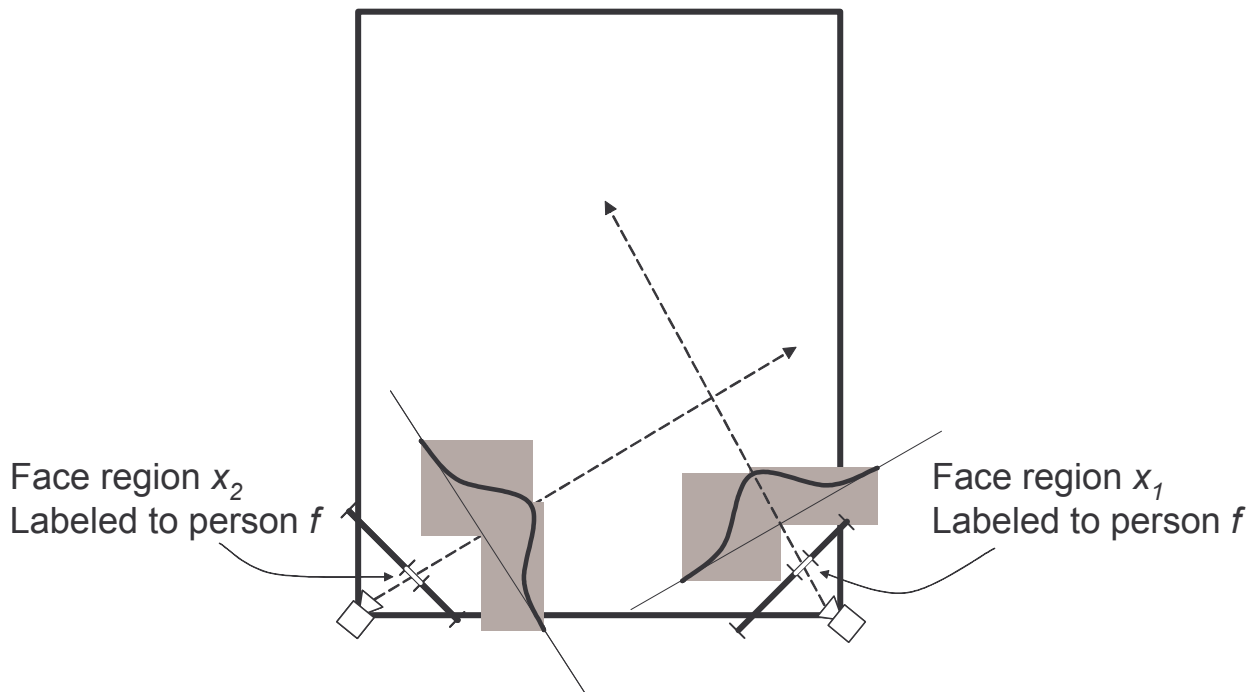


Figure 19: Illustration of an individual likelihood map of person f . Two face regions are detected.

Finally this likelihood map is normalized.

$$M_f = \frac{M_f}{\prod M_f} . \quad (4.12)$$

Unlabeled face regions

Unlabeled face regions are processed differently. Every unlabeled face regions is paired with all other unlabeled face regions not in the same camera view. If set G_u of all face regions contains 1 or less elements no action is performed and the next steps are skipped. This is necessary because a single unlabeled face region will not provide enough information to create a likelihood map.

If set G_u contains more than 1 element, trace for every face region $x \in G_u$, a ray r through the optical center of camera v and the center of face segment x . for each pair the shortest distance d between the two rays is calculated. Define t as a threshold defined by an experimented value based on the mean dimensions of the human face. If $d < t$, a new person g is added to the set Ω and a new likelihood map M_g is created.

4.4 Location estimates

The goal of this part of the system is to estimate locations of every person in set Ω by using the individual likelihood maps that were created for each of them in the previous phase. Output of this part will consist of a single location $l(x,y)$ for every person on the ground floor of the environment. Input is a likelihood map M_f for each person $f \in \Omega$. M_f consists of likelihood values of that person. The location of a person is estimated by identifying *regions* where the likelihood exceeds a given threshold.

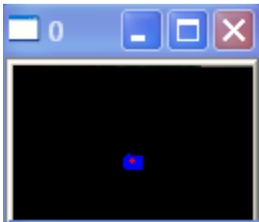


Figure 20: Illustration of region where likelihood exceeds given threshold, red dot shows the estimated location.

The estimate is then calculated using

$$l(x, y) = \frac{\sum_{u, u \in \text{region}} u \cdot M(u)}{\sum_u M(u)} \quad (4.13)$$

Figure 20 shows an example of a location estimation in the blue threshold region.

4.5 Kalman filtering

State estimation programs implement a model and an estimator. A model is analogous to a data structure representing relevant information about the visual scene. An estimator is analogous to the software engine that manipulates this data structure to compute beliefs about the world. The standard Kalman filter is such an estimator.

Many computer vision applications involve repeated estimating, that is, tracking, of the system quantities that change over time. These dynamic quantities are called the system *state*. The system in question can be anything that happens to be of interest to a particular vision task.

To estimate the state of a system, reasonably accurate knowledge of the system *model* and *parameters* may be assumed. Parameters are the quantities that describe the model configuration but change at a rate much slower than the state. Parameters are often assumed known and static. In this system a state is represented with a vector. In addition to this output of the state estimation routines, another vector introduced is a vector of *measurements* that are input to the routines from the sensor data, given by previous phases of the process. To represent the model, two things are to be specified.

- Estimated dynamics of the state change from one moment of time to the next.
- Method of obtaining a measurement vector z_t from the state.

An estimator should be preferably unbiased when the probability density of estimate errors has an expected value of 0. There exists an optimal propagation and update formulation that is the best, linear, unbiased estimator for any given model of the form. This formulation is known as the discrete Kalman estimator.

The Kalman filter addresses the general problem of trying to estimate the state x of a discrete-time process that is governed by the linear stochastic difference equation

$$x_{k+1} = Ax_k + w_k \quad (4.14)$$

with a measurement z , that is

$$z_k = Hx_k + v_k \quad (4.15)$$

What the filter eventually tries to do is estimating the state $x \in \mathbb{R}^n$ with a measurement $z \in \mathbb{R}^m$. The measurement in this case is the estimated location that is retrieved from the steps that were taken earlier. The random variables w_k and v_k represent respectively the process and the measurement noise. They are assumed to be independent (of each other), white, and with normal probability distributions

$$\begin{aligned} p(w) &\approx N(0, Q). \\ p(v) &\approx N(0, R). \end{aligned} \quad (4.16)$$

Both process and measurement noise covariance Q and R are constant in this model.

The $N \times N$ matrix A in the **difference equation (4.14)** relates the state at time step k to the state at step $k+1$, in the absence of process noise. The $M \times N$ matrix H in the **measurement equation (4.15)** relates the state to the measurement z_k .

If \hat{x}_k^- denotes a priori state estimate at step k provided the process prior to step k is known, and \hat{x}_k denotes a posteriori state estimate at step k provided measurement z_k is known, then a priori and a posteriori estimate errors can be defined as

$$\begin{aligned} e_k^- &= x_k - \hat{x}_k^- \\ e_k &= x_k - \hat{x}_k \end{aligned} \quad (4.17)$$

The a priori estimate error covariance is then $P_k^- = E[e_k^- e_k^{-T}]$ and the a posteriori estimate error covariance is $P_k = E[e_k e_k^T]$.

The Kalman filter estimates the process by using a form of feedback control: the filter estimates the process state at some time and then obtains feedback in the form of noisy measurements. As such, the equations for the Kalman filter fall into two groups: time update equations and measurement update equations. The time update equations are responsible for projecting forward in time the current state and error covariance estimates to obtain the a priori estimates for the next time step. The measurement update equations are responsible for the feedback, that is, for incorporating a new measurement into the a priori estimate to obtain an improved a posteriori estimate. The time update equations can also be viewed as predictor equations, while the measurement update equations can be thought of as corrector equations. Indeed, the final estimation algorithm resembles that of a predictor-corrector algorithm for solving numerical problems.

The output of the system is the input of a new process cycle. We will use the Kalman Filter to predict the future position of a person to aid the next process cycle in the classification phase. This predicted position can be compared to the results in the face detection phase retrieved from the camera footage. The Kalman filter can give predictions about future states by being supplied with data from previous steps. The Kalman filter smoothes out the input measurements, in this case the individual location estimates and cancels noise of the measurements. It will produce an location estimate for every person in set Ω for the next time step. In this system the input measurements can be processed in two ways.

- By combining all location maps to one array and feeding it to a single Kalman filter.
- Assigning each person to an individual Kalman filter. Every location map is assigned to a separate filter.

For this system instead of combining all location estimates and then feed them to a single Kalman filter every person is assigned to an individual Kalman filter. The reason for the individual approach is the fact that the set of current people in the room Ω is dynamic. This means that at any time t the set Ω can grow or shrink in size when people enter or leave the environment. If the set Ω is dynamic, the input vectors assigned to the Kalman filter will differ in size as well as the population changes in size. If a combined location array was the input measurement a dynamic vector would have been created because of the dynamic nature of Ω . By definition the Kalman filter is not able to process a state vector of varying size so the individual option is preferred. Every person is assigned to a separate Kalman filter. Then all the input vectors will have static size.

In this system the matrix A will be a 4 by 4 matrix and describes the transition model of the system. The columns from left to right representing respectively x , y , Δx , Δy .

$$A = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} . \quad (4.18)$$

Multiplication with x , the new x and y coordinates are computed by adding their respectively Δx and Δy components.

The input measurement represented by an estimated location on the ground plane of a person will be a vector with two elements, an x and y value. This means the state vector will have the following form:

$$\begin{bmatrix} x \\ y \end{bmatrix} . \quad (4.19)$$

The filter is supplied with the current location map of an individual person, which is the measurement that is currently taken. This measurement together with a set of mathematical

equations and a model of the process is used to estimate the actual state of the process. This makes sure sudden changes in movement by errors are smoothed out.

4.6 Combined location map

Goal of this phase is combining the individual location maps provided by the Kalman filter phase and output a ground map with the locations of every person in the room. Input in this phase is the individual location maps of every person $f \in \Omega$ provided by the Kalman filter phase. Each of these Kalman filters stores the estimated location as the new future position of that particular person. Then for each person in the environment these future positions are retrieved and combined in one array. These locations are presented in an array v of (x,y) location coordinates. The amount of persons in the environment is presented by n .

$$v = \left(\begin{bmatrix} x_1 \\ y_1 \end{bmatrix}, \begin{bmatrix} x_2 \\ y_2 \end{bmatrix}, \dots, \begin{bmatrix} x_n \\ y_n \end{bmatrix} \right) . \quad (4.20)$$

This array can now be used to draw a single ground map with every person plotted at a certain position. This data is then fed to the room model $\mathfrak{R}(t)$ to update its state and start a new cycle of the tracker. Together with a new frame of camera footage the process starts again at phase one. Because the system in the next cycle has the knowledge of every person's estimated position, it can give statistical predictions described in phase 2: classification.

4.7 RFID identification

After combining location maps and determining the final positions of all persons scattered through the room extra input can be used to enquire some valuable information. In various applications RFID tags can be worn by people in the environment. Whenever a person comes close enough to a sensor that person's identity is known and its location gets recorded. After determination of the location of people in the room these recordings can be used identify a person in the room. The result is that a person is not only discriminated from other people in the

room but is actually labeled with an own identity. For example the process estimates person f at a location l . If a RFID sensor at that moment senses a RFID tag in its perimeter it is very likely person f is in an area where his tag can be read. The sensor reads his identification and inputs this to the system. If person f did not yet have a label he is labeled with this identification. The label can later be used for more accurate classification too. RFID tags give the system the extra possibility of recognizing people instead of only discriminating them.

Chapter 5

Implementation

The theory has been worked out. Next is implementing these ideas. First this chapter describes the decisions made prior to programming about used tools and environments in **chapter 5.1**. Furthermore it describes the implementation process and results achieved from programming the following phases:

- The iterative system describing the communication flow between through the phases illustrated in **Figure 15** including:
 - The parallel face detector capable of multi-orientation face detection.
 - The classification algorithm based on object location data.
 - The algorithm for object-location estimation given multi-view classification data.
 - Specific implementation of Kalman filtering for this problem.
- Creating a virtual environment for testing purposes.
- Implementing the live environment.

5.1 Development tools

During development many tools were used to assist in the process. Usually a lot of alternatives were available for each topic. One of the first decisions that was made was which programming language and development environment to use. Eventually it was a choice influenced by external factors and nature of the problem. The general-purpose programming language C++ was eventually chosen over Java. Some arguments to use C++ are:

- C++ excels in high performance and a set of powerful features. Especially the real-time constraint of the problem make these properties an essential choice for this system.
- The language comprises a combination of both high-level and low-level language

features, which is beneficial for writing efficient code for real-time performance.

- A lot of research conducted at the department of CAIP developed code using the C++ language. Using the same environment results in easier access to information.
- The C++ language supports complete object-oriented programming, which makes structured programming much easier.
- This project wants to use the open source vision library OpenCV. It is completely written in C++.

After careful consideration decided is that all code will be programmed in C++ using the Microsoft Visual Studio integrated development environment. This is available on all workstations and presents a wide field of functions and tools useful while implementing.

5.1.1 OpenCV

OpenCV (Open Source Computer Vision) is a library of programming functions mainly aimed at real time computer vision. OpenCV includes the areas of:

- Human- Computer Interface.
- Object identification.
- Segmentation and Recognition.
- Face recognition.
- Gesture recognition.
- Motion tracking.
- Ego-motion.
- Motion understanding.
- Structure from motion.
- Mobile robotics.

A lot of functionality presented by the OpenCV library is useful for implementation of the proposed design. Some of the basics present in the OpenCV library that we can use for this system are:

- Matrix operations.
- Standard face detector, Viola and Jones. Needs to be adjusted to work in parallel.
- Basic Kalman filter.

The library of OpenCV is written in the C++ programming language. The functionality can be accessed from any programming language using an independent layer. But because this project is written in C++ it gives much more freedom accessing and adjusting elements from the OpenCV library. We choose to use the OpenCV library to assist in implementing the functionality of the algorithms.

5.1.2 Coin3D

For simulating and visualizing a scene where objects are tracked the 3D graphics development tool **Coin3D** is used.

Coin3D is a clone of object oriented retained mode 3D graphics API **Open Inventor**. It is an OpenGL based, retained mode 3D graphics rendering library. It is implemented in C++ and publicly released with open source code. The application programmer's interface (API) is fully compatible with SGI's Open Inventor, the *de facto* standard 3D graphics API for complex visualization applications. It is designed to create a higher layer of programming in OpenGL. Coin3D takes away the difficulty of low level library calls OpenGL presents to the user.

Coin3D uses scene graph data structures to render 3D graphics in real-time. Basic import, rendering, and interaction with a 3D object can be implemented in just a few lines of code, and programmer efficiency is greatly increased compared with programming directly with OpenGL. OpenGL code and Coin3D code can co-exist in the same application, which makes gradual migration from OpenGL to Coin3D possible.

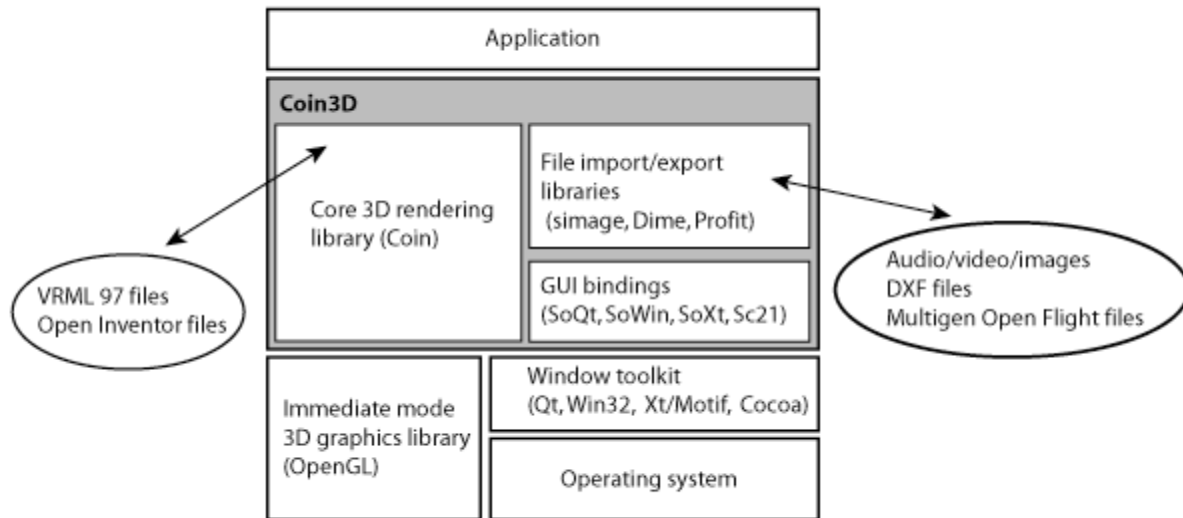


Figure 21: Basic workings of the Coin3D platform.

The core 3D rendering library Coin uses scene graph data structures to store and render graphics. Its data-driven design (redraws are only scheduled if the scene data changes) makes Coin3D well suited for user interface-based applications (e.g. scientific and engineering visualization applications), since it does not claim excessive CPU resources.

Being only a 3D rendering library, Coin needs a user interface binding to be able to open windows, handle user input etc. We will use the GUI Bindings that are available for the native Microsoft Windows GUI (SoWin).

Coin3D comes with several file import/export libraries: **simage** for loading and saving images, sound and video files; **Dime** for loading, manipulating, and saving DXF files; **Profit** for loading, manipulating, and saving MultiGen Open Flight 3D model files.

The Coin3D libraries page lists all available libraries with links to the individual project pages, latest releases, and Subversion information.

With Coin3D we will implement a basic virtual environment using object importing and scene construction capabilities. Then we will estimate facial positions on the four camera views and see if the algorithm gives useful results. This is further investigated in **chapter 5.3**.

Alternatives

There are numerous alternatives available to implement a simulated environment without the necessity of implementing complex OpenGL routines. Some of these that were considered are:

- Ogre 3D, one of the most popular open source 3D engines around. Excels in rendering capabilities but lacks the real ability of complete 3D engines, like physics.
- Irrlicht Engine, this engine is not as bloated as other engines but lacks usability when setting together an easy scene and scene viewer.
- Crystal space, Crystal Space is a very feature rich 3D engine but it is not very easy to use. One of the strongest points of Crystal Space is flexibility. Experiences tell us documentation is adequate but not 100% up-to-date in all respects. Crystal Space is a bit too bloated for this task. While CS is a very big project the structure is very modular. This makes CS less bloated than it could be.

Coin3D proved the only framework from which it was easy to extract 4 camera images and return them together. Furthermore accessing all of the frame buffers through C++ was much easier than with other engines. Finally the ready to use environment viewer turned out to be very useful while setting up the environment.

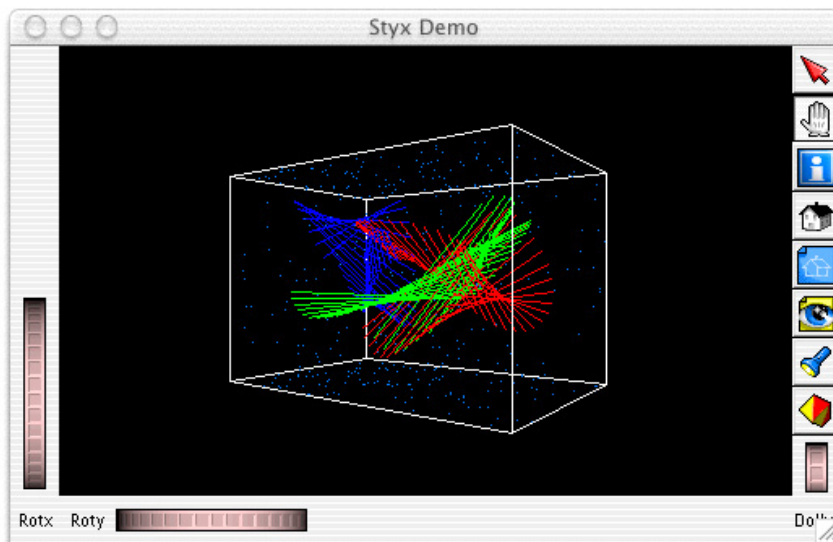


Figure 22: Basic Coin3D environment interaction window, capable of rotating, zooming, etc.

After careful consideration of the arguments above we chose to use Coin3D as our tool to implement the virtual environment.

5.2 Implementation of proposed algorithms

The proposed design will be implemented in the object oriented methodology using C++. The program needs to be modular so both virtual and live environment are independent from the implemented algorithm. The class diagram in **Figure 23** shows the basic outset for the program. The main object in this system is the **Room** class. One room instance is created and functions as manager over all functions in the system. It keeps track of the cameras placed, the inhabitants of the environment and the room characteristics like width and height. This instance was already introduced in **chapter 4**, as the room model \mathcal{R} . Four camera instances are created and added to the room. Each time step a frame is created by getting all four camera views and this frame is passed on to the room. In every view face segments can be detected in the face detection phase. Finally the identity class holds information about each inhabitant in the environment. It implements a individual Kalman filter and keeps track of likelihood maps and inhabitant location.

In **Figure 24** the flow of the program is illustrated in a sequence diagram. The program starts with the call of the main function in Facade. Here a room instance is created and several room characteristics are set. Examples of these characteristics are size of the room, defined by upper and lower corner coordinates and detail of the ground plane. The for every camera in the room an camera instance is created and passed to the room instance.

When the room is setup the program starts the main cycle by receiving frames from the extern environment, simulated or live. These frames hold four camera views and are fed to the room model every time frame. The room performs the following steps described in **chapter 4**: Face detection, classification, individual likelihoods, identity locations, Kalman filtering and combining the final locations.

The views are displayed with their face segments together with the final location map and the individual likelihood maps.

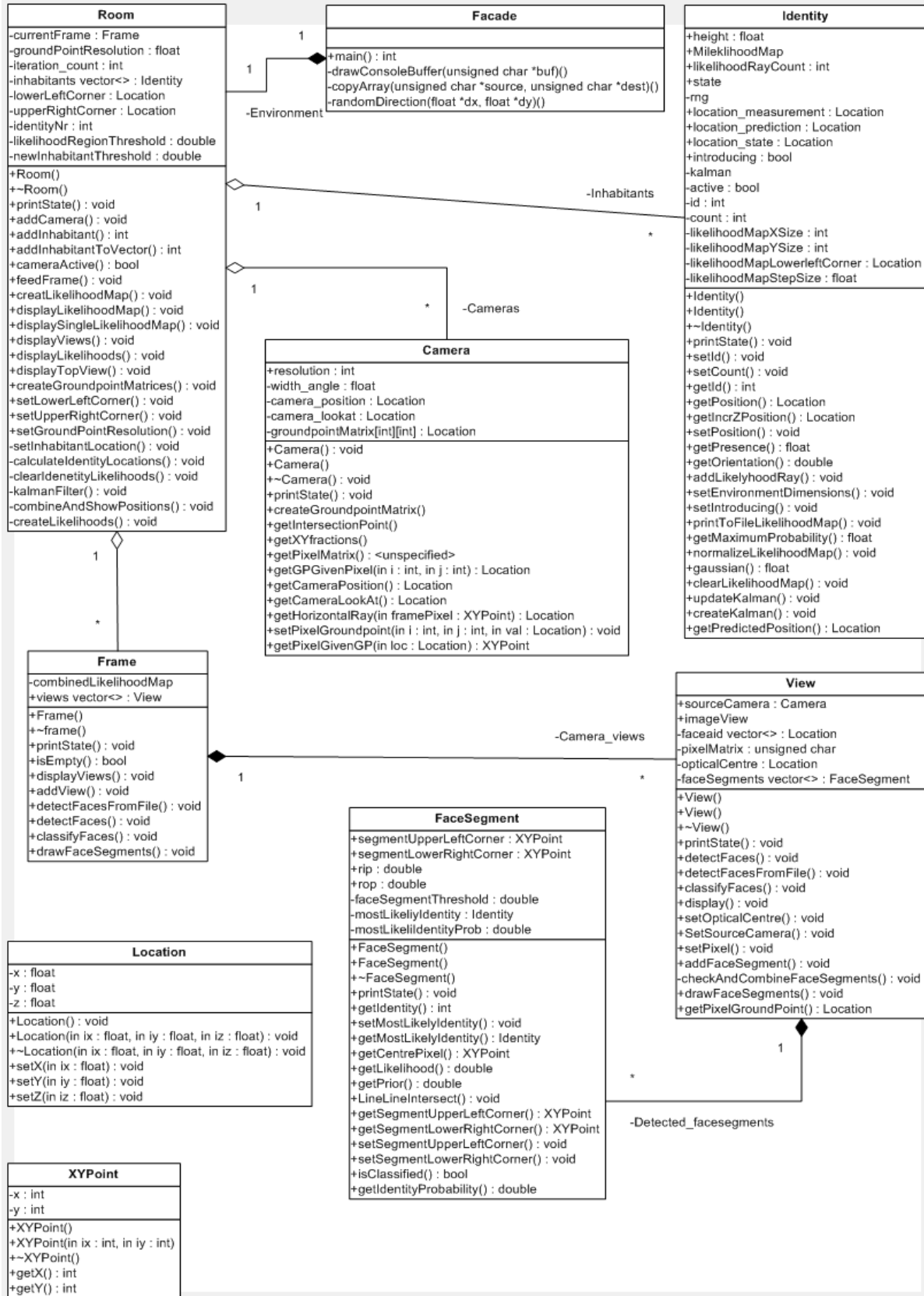


Figure 23: Class diagram of the system.

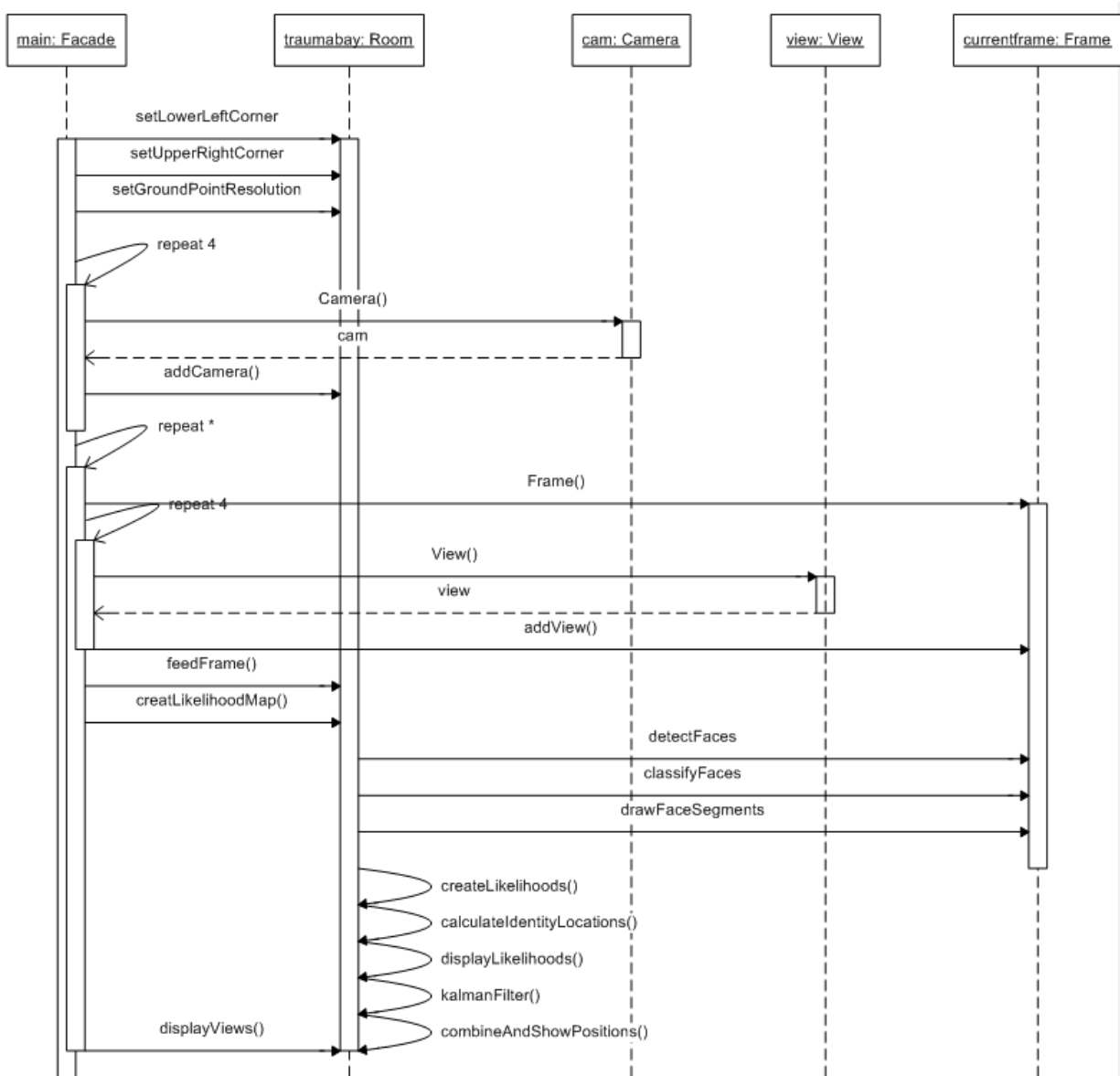


Figure 24: Sequence diagram of main flow in program.

5.2.1 System

As described a room instance is created from the main Facade file and then every timeframe the following function is called. Together with the input of new frames this is the complete iteration. Below is the main process that gets called every timeframe. The process is commented by text in green.

```

// -----
// Calls each function of the tracking system in order. Starts with face detection, then object
// classification,
// likelihoodmap creation, location estimation, kalman filtering, and finally shows the results
// and increases
// the iteration counter
//
// @return    void
// -----
void Room::createLikelihoodMap()
{
    //Detect faces using the simulated face detector which uses txt data
    currentFrame.detectFacesFromFile(iteration count);

    //Detect faces using the Viola and Jones face detector
    //currentFrame.detectFaces();

    //Classify the objects using the previous frame and the set of other inhabitants
    currentFrame.classifyFaces(inhabitants, previousFrame);

    //Draw the face segments and their classification onto the framebuffer
    currentFrame.drawFaceSegments();

    //Create for each person a likelihood map
    createLikelihoods();

    //Estimate each persons location based on their likelihoodmap
    calculateIdentityLocations();

    //Draw the likelihoodmaps in a new window
    displayLikelihoods();

    //Use a nindividual kalman filter for each estimated object location
    kalmanFilter();

    //Combine all locations from objects and show them in one groundfloor map
    combineAndShowPositions();

    //Increment the iteration counter
    iteration_count++;
}

```

5.2.2 Face detection

The parallel Viola and Jones detector is implemented using the openCV Viola and Jones detector and enhancing it with parallel functionality. Below the main algorithm implemented, comment is colored in green.

```

// -----
// Parallel Viola and Jones implementation
//
// @return    void
// -----
void View::detectFaces()
{
    //create two cascades, for front and profile detection
    CvHaarClassifierCascade* cascade_front;
    CvHaarClassifierCascade* cascade_profile;

```

```

CvMemStorage* storage=cvCreateMemStorage(0);
CvSeq* faces front;
CvSeq* faces profile;

//Load standard cascades from opencv directory
cascade_front = (CvHaarClassifierCascade*) cvLoad(file_frontal_training);
cascade_profile = (CvHaarClassifierCascade*) cvLoad(file_profile_training+ 0);

//Detect faces in the imagebuffer of this view using the front training.
faces front = cvHaarDetectObjects(
    imageView,
    cascade_front,
    storage,
    1.2, // scale the cascadeby 20% after each pass
    2, // groups of 3 (2+1) or more neighbor face rectangles are joined into a single
"face", smaller groups are rejected
    CV_HAAR_DO_CANNY_PRUNING, // use Canny edge detector to reduce number of false
alarms
    cvSize(0, 0) // start from the minimum face size allowed by the particular
classifier
);

//Detect faces in the imagebuffer of this view using the profile training.
faces_profile = cvHaarDetectObjects(
    imageView,
    cascade_profile,
    storage,
    1.2, // scale the cascadeby 20% after each pass
    2, // groups of 3 (2+1) or more neighbor face rectangles are joined into a single
"face", smaller groups are rejected
    CV_HAAR_DO_CANNY_PRUNING, // use Canny edge detector to reduce number of false
alarms
    cvSize(0, 0) // start from the minimum face size allowed by the particular
classifier
);

//For each detected frontal face
for(int i=0;i<(faces front ? faces front->total:0); i++ )
{
    //Get coordinates from the detected face
    CvRect* r = (CvRect*) cvGetSeqElem(faces_front, i);
    CvPoint pt1 = {r->x, r->y};
    CvPoint pt2 = {r->x + r->width, r->y + r->height};

    //Add facesegment to the stack
    FaceSegment *fs = new FaceSegment(XYPoint(r->x, r->y), XYPoint(r->x + r->width, r-
>y + r->height), 0, 0);
    addFaceSegment(fs);
}

//For each detected profile face
for(int i=0;i<(faces profile ? faces profile->total:0); i++ )
{
    //Get coordinates from the detected face
    CvRect* r = (CvRect*) cvGetSeqElem(faces_profile, i);

    //Add facesegment to the stack
    FaceSegment *fs = new FaceSegment(XYPoint(r->x, r->y), XYPoint(r->x + r->width, r-
>y + r->height), 0.5, 0);
    addFaceSegment(fs);
}

//Release Viola and Jones data
cvReleaseHaarClassifierCascade( &cascade_front );
cvReleaseHaarClassifierCascade( &cascade_profile );
cvReleaseMemStorage( &storage );

//Run algorithm for combining closely located facesgments
checkAndCombineFaceSegments();
}

```


A detected face region A and a detected face region B are detected from the same face if one of two conditions is met illustrated by **Figure 16**. The implementation of the **formula 4.1** and **formula 4.2** is given below. Comment colored green.

```

// -----
// Check if each facesegment is big enough and combines
// face segments that are close enough to each other
//
// @return void
// -----
void View::checkAndCombineFaceSegments()
{
    //For each face segment
    for(int i=0; i< faceSegments.size(); i++)
    {
        //Calculate the diagonal distance of a face segment
        double difx = faceSegments[i]->getSegmentLowerRightCorner().x - faceSegments[i]-
>getSegmentUpperLeftCorner().x;
        double dify = faceSegments[i]->getSegmentLowerRightCorner().y - faceSegments[i]-
>getSegmentUpperLeftCorner().y;

        //Check if diagonal is smaller then the facesegment threshold
        if(sqrt(pow(difx,2)+pow(dify,2)) < faceSegmentThreshold)
        {
            //If so, erase facesegment from stack
            faceSegments.erase(faceSegments.begin()+i, faceSegments.begin()+1+i);
            if(i>0)i--;
        }
    }

    //Check if any facesegments cover the same space and combine them together.
    for(int i=0; i< faceSegments.size(); i++)
    {
        for(int j=0; j< faceSegments.size(); j++)
        {
            XYPoint xy1a = faceSegments[i]->getSegmentUpperLeftCorner();
            XYPoint xy2a = faceSegments[i]->getSegmentLowerRightCorner();

            XYPoint xy1b = faceSegments[j]->getSegmentUpperLeftCorner();
            XYPoint xy2b = faceSegments[j]->getSegmentLowerRightCorner();

            if(
                //We are not dealing with the same facesegment
                (j != i) &&
                //and formula 4.1 in thesis
                (
                    (((xy2a.x - xy1a.x)/2 + xy1a.x) <= xy2b.x <= xy2a.x) && ((xy2a.y -
xy1a.y)/2 + xy1a.y) <= xy2b.y <= xy2a.y) ||
                    ((xy1a.x <= xy1b.x <= (xy2a.x - xy1a.x)/2 + xy1a.x) && ((xy1a.y <=
xy1b.y <= (xy2a.y - xy1a.y)/2 + xy1a.y))
                )
            )
            {
                //Update one facesegment by taking the mean of facesegment
coordinates
                faceSegments[i]-
>setSegmentUpperLeftCorner(XYPoint((int)(xy1a.x+xy1b.x)/2, (int)(xy1a.y+xy1b.y)/2));
                faceSegments[i]-
>setSegmentLowerRightCorner(XYPoint((int)(xy2a.x+xy2b.x)/2, (int)(xy2a.y+xy2b.y)/2));

                //Update one facesegment by taking the mean of both yaw and pitch
                faceSegments[i]->rip = (faceSegments[i]->rip+faceSegments[j]-
>rip)/2;
            }
        }
    }
}

```

```

        faceSegments[i]->rop = (faceSegments[i]->rop+faceSegments[j]-
>rop)/2;

        //Erase the other facesegment
        faceSegments.erase(faceSegments.begin()+j, faceSegments.begin()+1+j);
        if(i>0)i--;
    }
}
}
}

```

5.2.3 Classification

Main objective of the classification is labeling every detected face region. The labels come from the set Ω of current people present in the room. Below the main algorithm **formula 4.4** implemented, comment is colored in green.

```

// -----
// The following maximization problem is implemented
// maxf P(f|x) = maxf P(x|f) * P(f): Given a rectangular
// face region x in the input view, what is the probability that the face of person f is
// contained in x
// *** Formula 4.4 ***
//
// @param      inhabitants          Current inhabitants of the scene
// @param      previousFrame        The previous frame
// @return      void
// -----
void View::classifyFaces(vector<Identity*> inhabitants, vector<FaceSegment*>
previousFaceSegments)
{
    //Initialize
    double likelihood, prior, posterior;
    double posterior_temp;

    //for every facesegment of this view
    for(int i=0; i < faceSegments.size(); i++)
    {
        //First check which previous facesegment was closest to the current

        //Use as initial values
        float dis = -1;
        int closestFS = -1;

        //for every facesegment from the previous frame
        for(int ps=0; ps < previousFaceSegments.size(); ps++){

            //Get distance from old to new facesegment
            float a = sqrt(pow((double)previousFaceSegments[ps]->getCentrePixel().x-
faceSegments[i]->getCentrePixel().x,2) + pow((double)previousFaceSegments[ps]-
>getCentrePixel().y-faceSegments[i]->getCentrePixel().y,2));

            //If distance is smaller than record, save the record
            if(a < dis || dis == -1){
                dis = a;

                //And if previous facesegment is classified set closest facesegment
                //to that ID
                if(previousFaceSegments[ps]->isClassified()){

```

```

closestFS = previousFaceSegments[ps]-
>getMostLikelyIdentity()->getId();
    }
}

//Use as initial values
posterior temp = 0;
posterior = 0;

//For every inhabitant
for(int j=0; j < inhabitants.size(); j++)
{
    //Get the centre pixel of the facesegment
    XYPoint segmentXYPoint = faceSegments[i]->getCentrePixel();

    //Estimate the prior of person inhabitants[j] by calculating who
    //could block the ray from faceSegments[i] to inhabitants[j] giving all
    //the inhabitants including inhabitants[j].
    // *** Formula 4.5 ***
    prior = faceSegments[i]->getPrior(sourceCamera, inhabitants[j],
inhabitants);

    //Estimate the likelihood
    // *** Formula 4.6 ***
    likelihood = faceSegments[i]->getLikelihood(sourceCamera, inhabitants[j],
segmentXYPoint);

    //  $P(f|x) = P(x|f) * P(f)$ 
    posterior = likelihood * prior;

    //If new record and posterior bigger then face classification threshold
    if((posterior > posterior temp) && (posterior >
faceClassificationThreshold))
    {
        //Set most likely identity
        faceSegments[i]->setMostLikelyIdentity(inhabitants[j], posterior);
        posterior temp = posterior;
    }
}
}

```

And the prior probability implementation of **formula 4.5** below.

```

// -----
// Returns the prior. Gives the probability that a random face region x contains an image of a
// person f.
// When calculating the prior, we need to account for the possibility that other persons occlude
// the person f
//*** Formula 4.5 ***
//
// @param      cam                Source camera of this facesegment
// @param      inhabitant          Inhabitant prior is calculated for
// @param      other_inhabitants   Vector with all inhabitants in the room
// @return      void
// -----
double FaceSegment::getPrior(Camera *cam, Identity *inhabitant, vector<Identity*>
other_inhabitants)
{
    //Get camera position
    Location cam_pos = cam->getCameraPosition();

    //Get inhabitant position

```

```

Location inh_pos = inhabitant->getPosition();

//Calculate distance from the camera to the inhabitant
float inh distance = sqrt(pow(cam_pos.x - inh_pos.x,2) + pow(cam_pos.y - inh_pos.y,2) +
pow(cam_pos.z - inh_pos.z,2));

//Initialize the prior
float occlusion_prob = 1.0f;

//For every other inhabitant in the environment
for(int i=0; i < other_inhabitants.size(); i++)
{
    //Get the location of the other inhabitant
    Location oth_inh_pos = other_inhabitants[i]->getPosition();

    //Calculate distance from the camera to the other inhabitant
    float oth_inh distance = sqrt(pow(cam_pos.x - oth_inh_pos.x,2) + pow(cam_pos.y -
oth_inh_pos.y,2) + pow(cam_pos.z - oth_inh_pos.z,2));

    //If we are not dealing with the same inhabitant AND other inhabitant is closer to
the camera
    //This means the other inhabitant can occlude the current inhabitant
    if((other_inhabitants[i]->getId() != inhabitant->getId()) && (oth_inh distance <
inh_distance))
    {
        //Set x,y coordinates other inhabitant position
        float x = oth_inh_pos.x;
        float y = oth_inh_pos.y;

        //Set x,y coordinates camera position
        float x1 = cam_pos.x;
        float y1 = cam_pos.y;

        //Set x,y coordinates camera position
        float x2 = inh_pos.x;
        float y2 = inh_pos.y;

        //Calculate length of the shortest line between inhabitants[i] and
line(camera, inhabitant),
        //This line is perpendicular to line(camera, inhabitant).
        float A = x - x1;
        float B = y - y1;
        float C = x2 - x1;
        float D = y2 - y1;
        float dist = abs(A * D - C * B) / sqrt(C * C + D * D);

        //Formula 4.5: Get the gaussian probability using length of shortest line
and multiply
        occlusion_prob *= (1 - other_inhabitants[i]->getPresence(dist,0));
    }
}

//Return prior
float prior = occlusion_prob;
return prior;
}

```

And the likelihood implementation of **formula 4.6** below.

```

// -----
// Returns the likelihood. represents the probability of
// face region x (represented by location, size, and orientation of the head in the view),

```

```

// given that the face detector reports that it contains the face of person f.
// *** Formula 4.6 ***
//
// @param      cam                Source camera of this facesegment
// @param      inhabitant         Inhabitant likelihood is calculated for
// @param      cSegment           center of the facesegment
// @return     void
// -----
double FaceSegment::getLikelihood(Camera *cam, Identity *inhabitant, XYPoint cSegment)
{
    //Get camera position
    Location ray_a = cam->getCameraPosition();

    //Get other point on a line from the camera to the facesegment
    Location ray_b = cam->getRay(cSegment);

    //Get location of the inhabitant
    Location inh_pos = inhabitant->getPosition();

    //Set x,y coordinates inhabitant position
    float x = inh_pos.x;
    float y = inh_pos.y;

    //Set x,y coordinates camera position
    float x1 = ray_a.x;
    float y1 = ray_a.y;

    //Set x,y coordinates point on the ray through facesegment line
    float x2 = ray_b.x;
    float y2 = ray_b.y;

    //Calculate distance between inhabitant and ray through facesegment and camera
    float A = x - x1;
    float B = y - y1;
    float C = x2 - x1;
    float D = y2 - y1;
    float dist = abs(A * D - C * B) / sqrt(C * C + D * D);

    //Return the gaussian probability using length of shortest line
    return inhabitant->getPresence(dist,0);
}

```

5.2.4 Likelihood maps

Objective of this phase is the creation of an individual likelihood map for every person in set Ω and the introduction of new persons to set Ω .

After completion each individual map M_f will tell us the likelihood of a person f standing at a location l_m on this map. The map represents the ground floor of the environment people are walking in. Below the main algorithm of **formula 4.11** implemented, comment is colored in green.

```

// -----
// for each inhabitant create a likelihoodmap. Each individual map
// will tell us the likelihood of a person f standing at a location lm on this map.
// The map represents the ground floor of the environment people are walking in.
//*** Formula 4.11 ***

```

```

//
// @return void
// -----
void Room::createLikelihoods()
{
    //Clear likelihood maps
    clearIdentityLikelihoods();

    // -----
    // Put all non-classified facesegments in a new array faceSegmentArray
    // -----
    vector<FaceSegment*> faceSegmentArray;
    vector<int> faceSegmentViews;
    vector<View *> faceSegmentCameras;

    //For every view in the current frame
    for(int i=0; i< currentFrame.views.size(); i++)
    {
        //Get detected facesegments of the view in currentframe
        vector<FaceSegment*> faceSegments = currentFrame.views[i]->getFaceSegments();

        //For every facesegment in that view
        for(int j=0; j< faceSegments.size(); j++)
        {
            //If face segment is classified
            if(!faceSegments[j]->isClassified())
            {
                faceSegmentArray.push_back(faceSegments[j]);
                faceSegmentViews.push_back(i);
                faceSegmentCameras.push_back(currentFrame.views[i]);
            }
        }
    }

    // -----
    // Create pairs from array faceSegmentArray
    // -----
    vector<FaceSegmentPair*> faceSegmentPairs;
    for(int i=0; i< faceSegmentArray.size(); i++)
    {
        for(int j=i+1; j< faceSegmentArray.size(); j++)
        {
            //If facesegments do not come from the same camera view
            if(faceSegmentViews[i] != faceSegmentViews[j])
            {
                FaceSegmentPair *pair = new FaceSegmentPair(faceSegmentArray[i],
faceSegmentArray[j], faceSegmentCameras[i], faceSegmentCameras[j]);
                faceSegmentPairs.push_back(pair);
            }
        }
    }

    // -----
    // If rays through the facesegments of the pair are close enough, create a new inhabitant
    // -----
    for(int i=0; i< faceSegmentPairs.size(); i++)
    {cout << faceSegmentPairs[i]->getDistance();
        //Check distance
        if(faceSegmentPairs[i]->getDistance() < newInhabitantThreshold &&
faceSegmentPairs[i]->getDistance() > 0)
        {
            //Label face segments and add inhabitant
            Identity *inh = new Identity();
            faceSegmentPairs[i]->facesegment a->setMostLikelyIdentity(inh, 1.0);
            faceSegmentPairs[i]->facesegment b->setMostLikelyIdentity(inh, 1.0);
            addInhabitant(inh);
        }
    }

    // -----
    // For every classified facesegment create a likelihoodmap

```

```

// -----
//For every view in the current frame
for(int i=0; i< currentFrame.views.size(); i++)
{
    //Get detected facesegments of the view in currentframe
    vector<FaceSegment*> faceSegments = currentFrame.views[i]->getFaceSegments();

    //For every facesegment in that view
    for(int j=0; j< faceSegments.size(); j++)
    {
        //If face segment is classified
        if(faceSegments[j]->isClassified())
        {
            //Get the most likely identity from the facesegment
            Identity *sub = faceSegments[j]->getMostLikelyIdentity();

            //Project a ray over the likelihood map
            sub->addLikelihoodRay(currentFrame.views[i]->sourceCamera,
currentFrame.views[i]->sourceCamera->getRay(faceSegments[j]->getCentrePixel()));
        }
    }

    //Normalize all likelihoodmaps, they all consist of summed individual normalized
    //facesegment rays.
    for(int i=0; i < inhabitants.size(); i++)
    {
        inhabitants[i]->normalizeLikelihoodMap();
    }
}

```

5.2.5 Estimating object location

This part of the system estimates the locations of every person in set Ω by using the individual likelihood maps that were created for each of them in the previous phase. Output of this part will consist of a single location $l(x,y)$ for every person on the ground floor of the environment. Below the main algorithm of **formula 4.13** implemented, comment is colored in green.

```

// -----
// Estimate locations of every inhabitant by using their individual likelihood maps.
// Result consists of a single location l(x,y) for every person on the ground floor of the
// environment.
// *** Formula 4.13 ***
//
// @return void
// -----
void Room::calculateIdentityLocations()
{
    //Get row and column count of the environment
    int rows = (int)abs(lowerLeftCorner.y - upperRightCorner.y)/groundPointResolution;
    int cols = (int)abs(lowerLeftCorner.x - upperRightCorner.x)/groundPointResolution;

    //For every inhabitant
    for(int h=0; h< inhabitants.size(); h++)
    {
        // Cases when new inhabitant can be discarded
        // #1 If likelihoodmap has less than 2 rays and inhabitant is new
        if(inhabitants[h]->likelihoodRayCount < 2 && inhabitants[h]->introducing)
        {
            inhabitants.erase(inhabitants.begin()+h, inhabitants.begin()+1+h);
        }
    }
}

```

```

        h--;
    }
    else
    {
        //If good, inhabitant is labeled 'existing'
        inhabitants[h]->setIntroducing(false);
    }
}

//For every inhabitant
for(int h=0; h< inhabitants.size(); h++)
{
    //Initialize
    float x tel = 0;
    float y tel = 0;
    float tot prob = 0;

    //Get inhabitant specific gaussian maximum
    float prob_threshold = inhabitants[h]-
>getMaximumProbability()/likelihoodRegionThreshold;

    // predict a new position using the klaman filter when the likelihood map has only
one ray
    if(inhabitants[h]->likelihoodRayCount < 2 && !inhabitants[h]->introducing)
    {
        inhabitants[h]->predictKalman();
    }
    else
    {
        //For every groundpoint of the room
        for(int i=0; i<rows; i++)
        {
            for(int j=0; j<cols; j++)
            {
                //If likelihood bigger than threshold from chapter 4.4
                if
                (cvmGet(inhabitants[h]->MlikelihoodMap,i,j) >
prob threshold)
                {
                    //sum the x and y values multiplied by the
likelihood
                    y_tel += i * cvmGet(inhabitants[h]-
>MlikelihoodMap,i,j);
                    x_tel += j * cvmGet(inhabitants[h]-
>MlikelihoodMap,i,j);

                    //sum the probablities
                    tot_prob += cvmGet(inhabitants[h]-
>MlikelihoodMap,i,j);
                }
            }
        }

        //Divide the summed x and y by the total probability for a mean x,y
location
        float x = x tel/tot prob;
        float y = y tel/tot prob;

        //Set inhabitant position
        inhabitants[h]->setPosition(Location(x,y,0));
    }
}
}

```


5.2.6 Kalman filter

The Kalman filter estimates the process by using a form of feedback control: the filter estimates the process state at some time and then obtains feedback in the form of noisy measurements. The Kalman filter is implemented with the system model of **formula 4.18** by the code below. Comment is colored in green.

```
void Identity::createKalman()
{
    // Create state vector
    state = cvCreateMat( 4, 1, CV_32FC1 );
    state_set = false;

    kalman = cvCreateKalman( 4, 2, 0 );

    // The model of the system in matrix A (4.18)
    const float A[] = { 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1 };

    memcpy( kalman->transition_matrix->data.fl, A, sizeof(A));
    cvSetIdentity( kalman->measurement_matrix, cvRealScalar(1) );
    cvSetIdentity( kalman->process_noise_cov, cvRealScalar(1e-5) );
    cvSetIdentity( kalman->measurement_noise_cov, cvRealScalar(1e-1) );
    cvSetIdentity( kalman->error_cov_post, cvRealScalar(1));
}

void Identity::updateKalman()
{
    // -----
    // Create random generator
    // -----
    CvRandState rng;
    cvRandInit( &rng, 0, 1, -1, CV_RAND_UNI );
    cvRandSetRange( &rng, 0, 0.1, 0 );
    rng.disttype = CV_RAND_NORMAL;

    // -----
    // Predicting the future state
    // -----
    const CvMat* prediction = cvKalmanPredict( kalman, 0 );
    location_prediction.x = prediction->data.fl[0];
    location_prediction.y = prediction->data.fl[1];

    // -----
    // Inserting the measurement into the Kalman filter
    // -----
    CvMat* measurement = cvCreateMat(2, 1, CV_32FC1);
    measurement->data.fl[0] = location_measurement.x;
    measurement->data.fl[1] = location_measurement.y;
    cvKalmanCorrect( kalman, measurement);

    // -----
    // Adding noise to the state and set the prediction as new state
    // -----
    CvMat* process_noise = cvCreateMat( 4, 1, CV_32FC1 );
    cvRandSetRange( &rng, 0, sqrt(kalman->process_noise_cov->data.fl[0]), 0 );
    cvRand( &rng, process_noise );
    cvMatMulAdd( kalman->transition_matrix, state, process_noise, state );
    location_state.x = prediction->data.fl[0];
    location_state.y = prediction->data.fl[1];
}
```

5.3 Virtual environment

The environment of the trauma bay is full of unsuspected behavior. In order to investigate methods and test the algorithms created for the task of tracking, an easy to use and flexible simulation of this environment is needed. With the focus on testing and investigating in the earlier stages of the research, using the real environment would result in too many constraints and probably too much complexity. With current possibilities in virtual modeling the possibility to mimic the characteristics of the real domain are very realistic. By transferring the real situation to a virtual enables more flexibility during testing.

We discussed the Coin3D library as framework to create an simulated environment in previous chapters. The virtual environment allows for arbitrary placement of lights, persons and other objects and also enables variable camera placements and orientation. Useful setups can then later be transferred to the real environment. The goal is then to create a virtual world that would resemble the research domain best as possible. The Coin3D toolkit hides a lot of complexity from the developer, making the process of creating scenes a lot easier. Instead of focusing on the details of the 3D representation, models are easily imported to represent a realistic environment. Using the toolkit a basic room is created with the same size as the real trauma bay. Human models are loaded into the scene using *vrml* files. These files represent the geometry of a human person in detail. An arbitrary amount of models are introduced in the scene and cameras are set up. From the frame buffer of each virtual camera an image is taken for every time frame. These views are stored together as a single time slice and are presented to the main algorithm.

In **Figure 25** an overview is given from the virtual environment running on the windows desktop. In the top left corner we see estimations running of an early implementation of the main tracking algorithm. Below in the two larger boxes we see respectively a top view footage from the room and a combined 4-camera split screen. Three human models where imported in this model and the environment was created by setting up 4 walls and a floor tile.

For implementing the virtual scene Coin3D works with the concept of separators. Basically every scene has a root with multiple branches that build up the room. At every junction a separator is implemented to characterize that node with different variables like light, color or shape. In **Figure 26** is a graphic representation of such a separator.

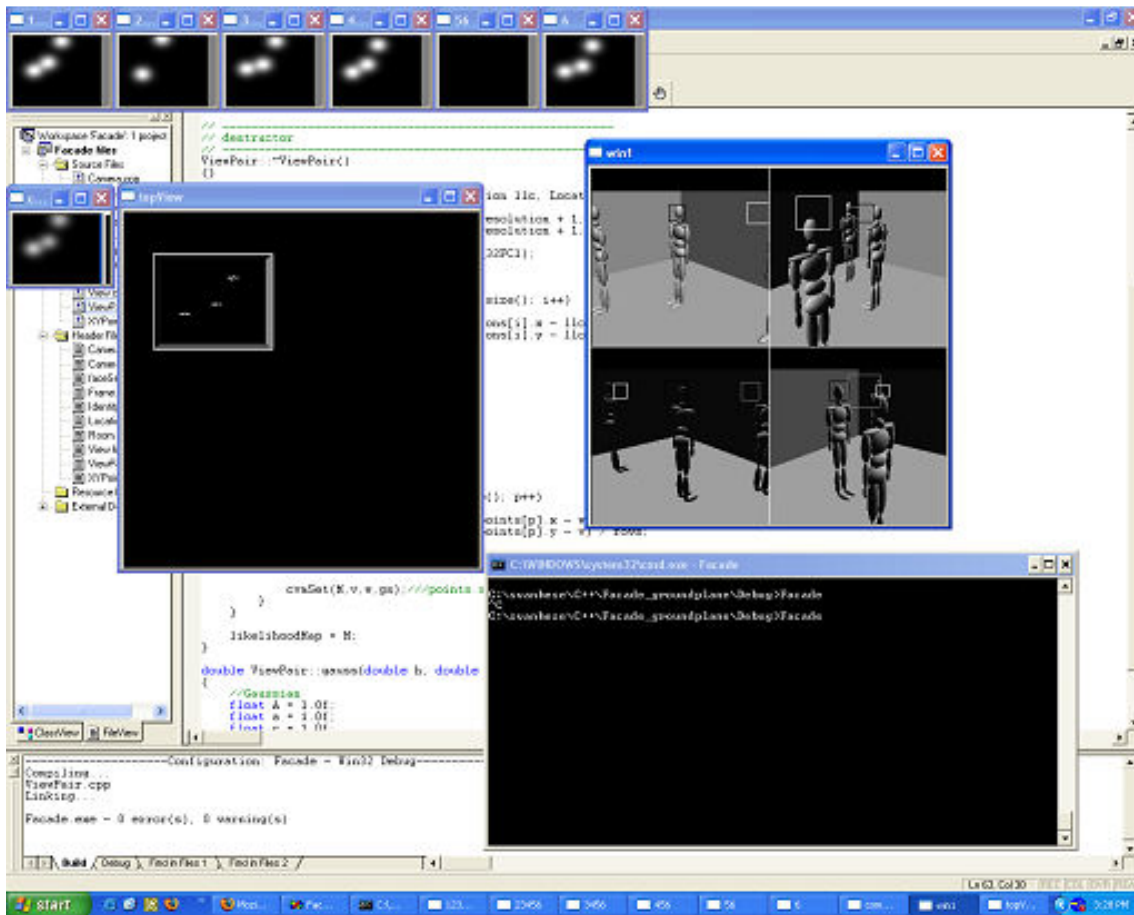


Figure 25: Desktop screenshot of running virtual environment with early implementation.

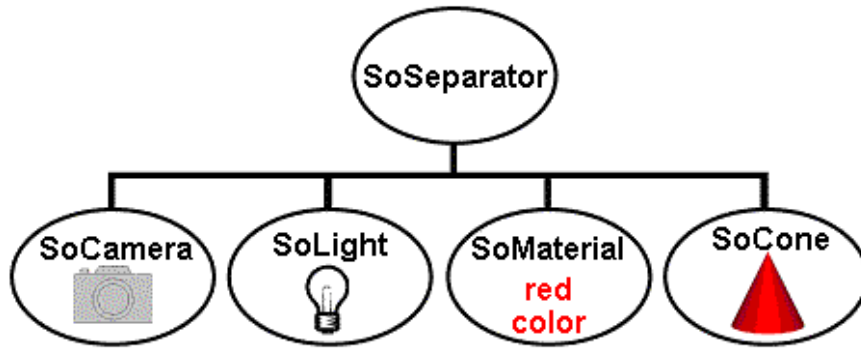


Figure 26: Visual representation of a Coin3D separator with various options.

First a root is established and a room and its inhabitants become its children. Then camera and light nodes are added as well. The walls are created and finally the light source is give a direction.

```

SoSeparator *root = new SoSeparator; root->ref();
SoSeparator *room = new SoSeparator;
SoSeparator *inhabs = new SoSeparator;

SoPerspectiveCamera * camera = new SoPerspectiveCamera;
SoDirectionalLight * light = new SoDirectionalLight;
SoCube * wall_l = new SoCube;
SoCube * wall_r = new SoCube;
SoCube * wall_b = new SoCube;
SoCube * wall_a = new SoCube;
SoCube * wall_f = new SoCube;

root->addChild(camera);
root->addChild(light);
root->addChild(room);
root->addChild(inhabs);

light->direction.setValue(1, -1.2f, -0.5f);
  
```

Then a graphical room model is loaded and two inhabitants are introduced to the scene.

```
SoSeparator *doc_1_model = SoDB::readAll(&in);
SoSeparator *room_model = SoDB::readAll(&in);

SoTransformSeparator *inhab_1 = new SoTransformSeparator;
SoTransform *inhab_1_transf = new SoTransform;
SoTransformSeparator *inhab_2 = new SoTransformSeparator;
SoTransform *inhab_2_transf = new SoTransform;

inhabs->addChild(inhab_1);
inhab_1->addChild(inhab_1_transf);
inhab_1->addChild(doc_1_model);
inhabs->addChild(inhab_2);
inhab_2->addChild(inhab_2_transf);
inhab_2->addChild(doc_1_model);
```

For every wall and the floor a new separator is established and a transformer is attached to position the walls. Then the separators are attached as child to each individual wall.

```
SoTransformSeparator *room_wall_l = new SoTransformSeparator;
SoTransform *room_wall_l_transf = new SoTransform;
SoTransformSeparator *room_wall_r = new SoTransformSeparator;
SoTransform *room_wall_r_transf = new SoTransform;
SoTransformSeparator *room_wall_b = new SoTransformSeparator;
SoTransform *room_wall_b_transf = new SoTransform;
SoTransformSeparator *room_wall_a = new SoTransformSeparator;
SoTransform *room_wall_a_transf = new SoTransform;
SoTransformSeparator *room_wall_f = new SoTransformSeparator;
SoTransform *room_wall_f_transf = new SoTransform;

room->addChild(room_wall_l);
room_wall_l->addChild(room_wall_l_transf);
room_wall_l->addChild(wall_l);
room->addChild(room_wall_r);
room_wall_r->addChild(room_wall_r_transf);
room_wall_r->addChild(wall_r);
```

```

room->addChild(room_wall_b);
room_wall_b->addChild(room_wall_b_transf);
room_wall_b->addChild(wall_b);
room->addChild(room_wall_a);
room_wall_a->addChild(room_wall_a_transf);
room_wall_a->addChild(wall_a);
room->addChild(room_wall_f);
room_wall_f->addChild(room_wall_f_transf);
room_wall_f->addChild(wall_f);

```

The current inhabitants are placed in the scene and scaled to proportion. The walls are scaled to set them in place.

```

inhab_1_transf->scaleFactor.setValue(1.0f,1.0f,1.0f);
inhab_2_transf->scaleFactor.setValue(1.0f,1.0f,1.0f);

inhab_1_transf->translation.setValue(1.0f,1.0f,1.0f);
inhab_2_transf->translation.setValue(absx2,0.0f,absy2);

room_wall_l_transf->scaleFactor.setValue(2.5f,1.0f,0.05f);
room_wall_r_transf->scaleFactor.setValue(2.5f,1.0f,0.05f);
room_wall_b_transf->scaleFactor.setValue(0.05f,1.0f,2.0f);
room_wall_a_transf->scaleFactor.setValue(0.05f,1.0f,2.0f);
room_wall_f_transf->scaleFactor.setValue(2.5f,0.05f,2.0f);

room_wall_l_transf->translation.setValue(0.0f,0.0f,-2.0f);
room_wall_r_transf->translation.setValue(0.0f,0.0f,2.0f);
room_wall_b_transf->translation.setValue(2.5f,0.0f,0.0f);
room_wall_a_transf->translation.setValue(-2.5f,0.0f,0.0f);
room_wall_f_transf->translation.setValue(0.0f,-1.0f,0.0f);

```

Then for every cycle in the algorithm the camera is placed on every room corner and a snapshot is taken that is placed in the global frame buffer. **Figure 27** illustrates this frame buffer printed in a window.

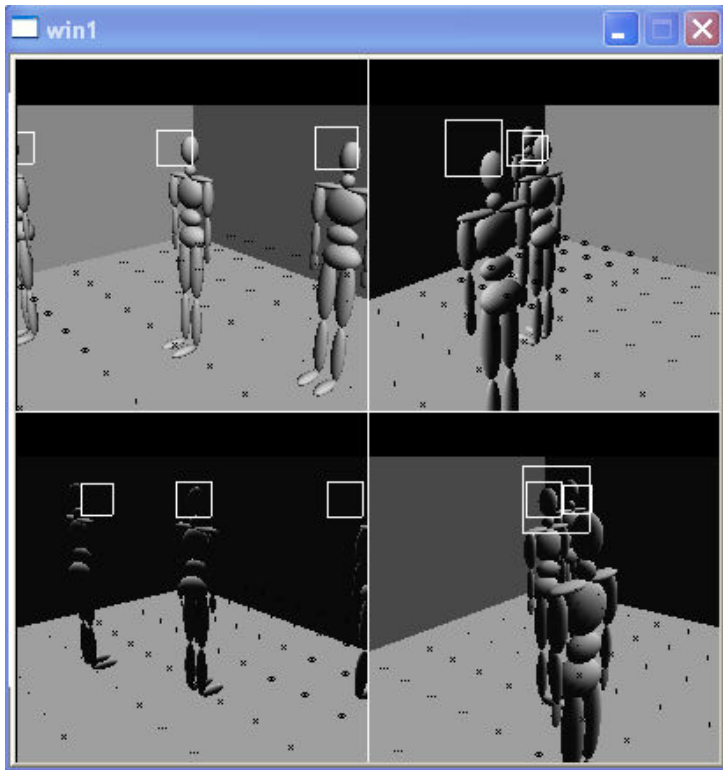


Figure 27: Visual environment created with 3 inhabitants.

5.3.1 Camera calibration

When we have calibrated cameras we can calculate the intersection point on the ground plane of every ray that is traced from the cameras optical centre through an image pixel.

Figure 28: Intersection points with the groundplane.

For the system to work with the input camera footage the virtual cameras need to be calibrated to the environment. The virtual environment also functions as a testing ground for camera calibration.

We use the DLT method explained in **chapter 3.3**. With this method we can get a 2D image view pixel from a 3D world coordinate. We tested the implementation by plotting a ground plane grid on the image view of each camera. This is visualized in **Figure 29**.

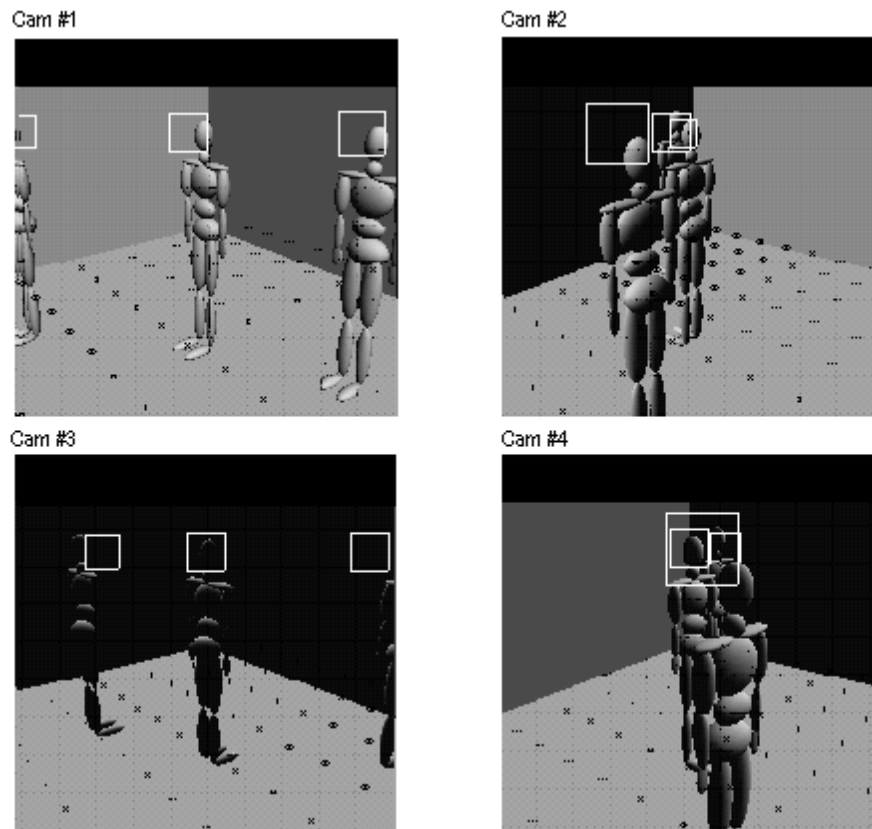


Figure 29: Illustration of the virtual environment created in Coin3D. On the ground plane is a plot of the calibration method that was used.

5.3.2 Feeding images to the algorithm

In order to implement a modular system the algorithms are implemented in a layered approach. This is especially important in the communication between the camera footage and the tracking algorithms. The camera environment produces a frame with four camera views. The tracking

algorithm is not interested in how this frame is produced. It can either be formed from the virtual environment or the live camera environment. This concept is emphasized in **Figure 30**.

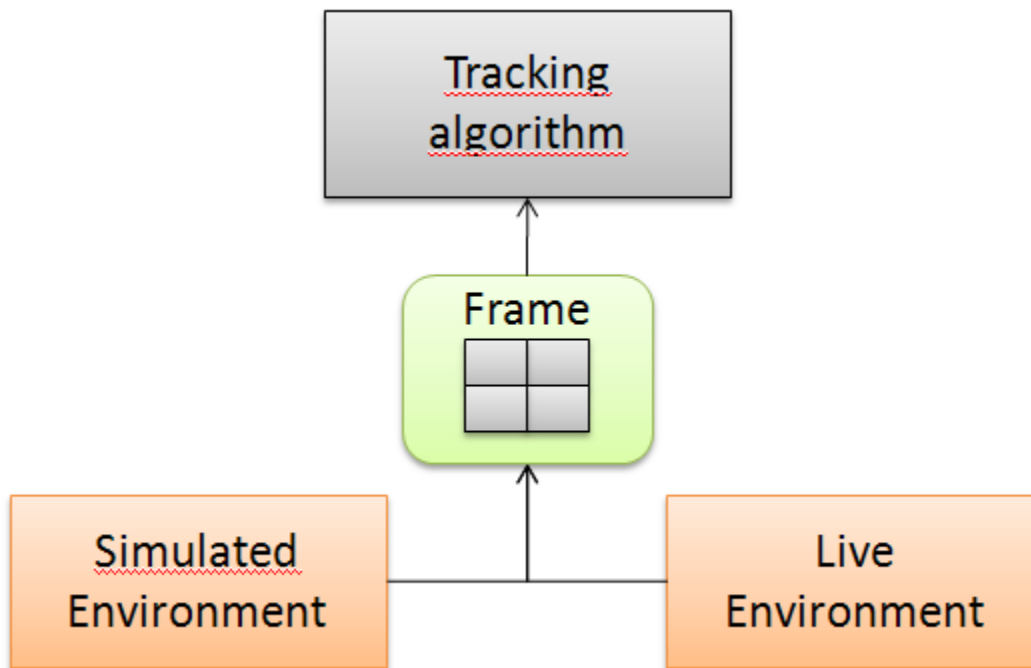


Figure 30: Illustrating the layered approach used in programming. Both environments are independent from the tracking algorithms.

The only thing that is used to transfer information from the environment to the algorithms is the frame class. This class enables storing an image buffer into its variables. Then this frame is passed on to the tracking algorithm and is processed. The code fragment below describes this process.

```
Frame frame;  
frame.addView(new View(imgbuffer_1, cam_1));  
frame.addView(new View(imgbuffer_2, cam_2));  
frame.addView(new View(imgbuffer_3, cam_3));  
frame.addView(new View(imgbuffer_4, cam_4));  
  
traumabay.feedFrame(frame);
```

To adapt the code to the live environment only the image buffer has to be filled by the live environment with real camera footage.

5.4 Implementing the live environment

The live environment was setup in the visual lab of the CAIP department in New Jersey. We needed the following items to construct the scene.

- A large room, approximately 12 meters long and 8 meters wide.
- Four identical cameras, capable of capturing color footage with at least a 320 x 240 resolution.
- Synchronization software for the four cameras. At each timestamp t we need from each camera the exact view that was taken at that timestamp t .
- One workstation to gather the live footage. This workstation needs enough horsepower to process all the incoming camera footage.
- Adequate lighting equipment to light the inhabitants of the scene evenly at every position.

5.4.1 Setting up the scene

The visual lab of the CAIP department had enough room to create a scene of realistic size. This lab was used during similar research so a lot of equipment was already available. Unfortunately the curtains that could completely enclose the scene were torn and could not be used in this research. They would have definitely resulted in better face detection rates because of the uniform background. Four identical cameras were found with the following specifications:

- Panasonic brand.
- Capable of recording with a 640 x 480 resolution.
- Color recording, 24 bit-depth.
- No compression used.
- Planes: 1.

These cameras were each positioned at predefined places in the room. They were stabilized on large tripods that could elevate the cameras optical position up to around 1.90 meter.

Figure 31: Schematic view of the live scene.

Two high-intensity discharge lamps were used to light the scene. They were placed on the long ends of the scene on face height. These lights are used for professional photography. Some of the characteristics of these lamps are.

- Low heat radiation.
- Color temperature is a characteristic of visible light. The temperature (in Kelvin) at which the heated black-body radiator matches the color of the light source is that source's color temperature. These lamps produce light of 6000K which resembles day light and are described as white light. This is excellent for maintaining proper light color to the cameras.
- They come with a screen that diffuses the light around the scene instead of bundling it to one direction.

- Build in reflectors maximize the light on the scene.

Finally the cameras are connected to two workstation. Each workstation handled the capturing of the footage of one camera pair. Using software available at the CAIP department at both workstations the camera pairs could be synchronized. The problem was that these two workstations needed to be synchronized together as well. The software available at the department capable of synchronizing each workstation was not working and parts had to be rewritten by myself. An initialization function was created and using network connectivity between the two workstations both programs could be connected to each other.

5.4.2 Capturing the scene

Finally after the complete environment was set, some actors where introduced to the scene and the first test footage was shot. In **Figure 32** and **Figure 33** two frames of this footage are shown.



Figure 32: Illustration of the live environment. Four camera views with three people in the room.



Figure 33: Illustration of the live environment.

5.4.3 Feeding images to the algorithm

Similar to the simulated environment approach the images are stored in the frame class and passed to the tracking algorithms. Below is a code fragment of this process.

```
string image1_string = imglocation + "\\cam1&2_backup\\test_01_" + str + ".bmp";
string image2_string = imglocation + "\\cam1&2_backup\\test_02_" + str + ".bmp";
string image3_string = imglocation + "\\cam3&4_backup\\test_01_" + str + ".bmp";
string image4_string = imglocation + "\\cam3&4_backup\\test_02_" + str + ".bmp";

IplImage *image_1 = cvLoadImage(image1_string.c_str(), 1);
IplImage *image_2 = cvLoadImage(image2_string.c_str(), 1);
IplImage *image_3 = cvLoadImage(image3_string.c_str(), 1);
IplImage *image_4 = cvLoadImage(image4_string.c_str(), 1);

// -----
```

```
// FILL FRAME AND FEED TO ALGORITHM
// -----
Frame frame;
frame.addView(new View(image_1, cam_1, FACESEGMENTTHRESHOLD));
frame.addView(new View(image_2, cam_2, FACESEGMENTTHRESHOLD));
frame.addView(new View(image_3, cam_3, FACESEGMENTTHRESHOLD));
frame.addView(new View(image_4, cam_4, FACESEGMENTTHRESHOLD));

traumabay.feedFrame(frame);
```

Eventually the live camera footage is passed real-time to the tracking algorithms but currently this is done offline. Each timeframe a bitmap is stored for every camera view. These images are loaded into the system using OpenCV functionality. Then these images are fed to the system by storing them in a frame instance and passing them on.

Chapter 6

Evaluation and Results

6.1 Software testing

This chapter summarizes the steps taken in software testing.

6.1.1 Test Plan

Testing the software basically encompassed the following steps:

- Test each algorithm in the system using the following information:
 - Input to the algorithm.
 - Expected output.
 - Measured output.
- These algorithms are:
 - Parallel face detection.
 - Classification.
 - Likelihood map creation.
 - Object-location estimation.
 - Kalman filtering over several frames.
- Then the complete system was tested by:
 - Feeding virtual camera views from the simulated environment.
 - Feeding live camera footage from the live environment.

During these tests, several bugs and anomalies were found. The next section explains those problems and the measurements that were taken.

6.1.2 Bugs

During software testing several bugs were found and treated accordingly. They were found in the

following form:

- Buffer overflow in image buffer or matrices. Usually because dimensions were wrongly interpreted or calculations incorrect.
- Counters that kept increasing while no more frames were presented. There was no upper limit on certain counters.
- Kalman filter not updating. Using the current position instead of the estimated position in calculations. This kept the filter in a static state.
- Cameras not calibrated right. Wrongly entered initial constants or calibration points.
- Width and height changed in matrix calculations. Usually resulted in buffer overflows or wrongly printed locations on the map.

6.1.3 Time management

An important feature of the system is that it should be real-time if it is ever to be implemented into a real live scenario. One factor in this is the amount of frames it can process every minute to keep the input real time. The system is separated by phase/algorithm below. We are using a Pentium 4 processor 550 HT 3.40 GHz with 512 MB internal memory. For each phase the amount of time taken processing one frame is plotted in **Table 1**.

Table 1: Illustration of processing times per frame for each phase in the architecture.

System phase	Process time for one frame	Frames / minute
Face detection	1952ms	30
Classification	86ms	697
Likelihood maps creation	53ms	1132
Object location estimation	62ms	967
Kalman filter	59ms	1016
Combining and printing results	31ms	1935

We can see that the Viola and Jones face detector slows the system down enormously. We have tried different settings for the Viola and Jones detector (For example bigger rectangle features), but this resulted in very poor detection rates. The current settings had to be used for a reasonable result (> 50% detection rate). Without the Viola and Jones detector the system can work real time with a frame count of 206 per minute using a resolution of 640×480 pixels.

6.2 Face detection

In **chapter 3.1.1** the concept of implementing the Viola and Jones face detector in parallel was introduced. We have created a Viola and Jones detector that is capable of detecting faces with 0° yaw and with 90° yaw (i.e. frontal and profile projection). We have used the Viola and Jones detector with the following variables.

- Scale the cascades 20% after each pass.
- Groups of 3 or more neighbor face rectangles are joined into a single `face`, smaller groups are rejected.
- Canny edge detector to reduce number of false alarms.

As a first illustration of the developed algorithm we applied it to several high-resolution pictures. Later in this chapter the face detector is tested with live camera footage. Hence the algorithm is tested in a systematic way. Below we present three real pictures from scenes taken at the Robert Wood Johnson University Hospital in New Jersey during a simulated resuscitation event organized to analyze the workflow and communication during such an event. This information was collected and used for this research. Faces are detected with a standard openCV training set for frontal and profile projection. Frontal detections are colored red and 90° - 90° yaw detections are colored green.

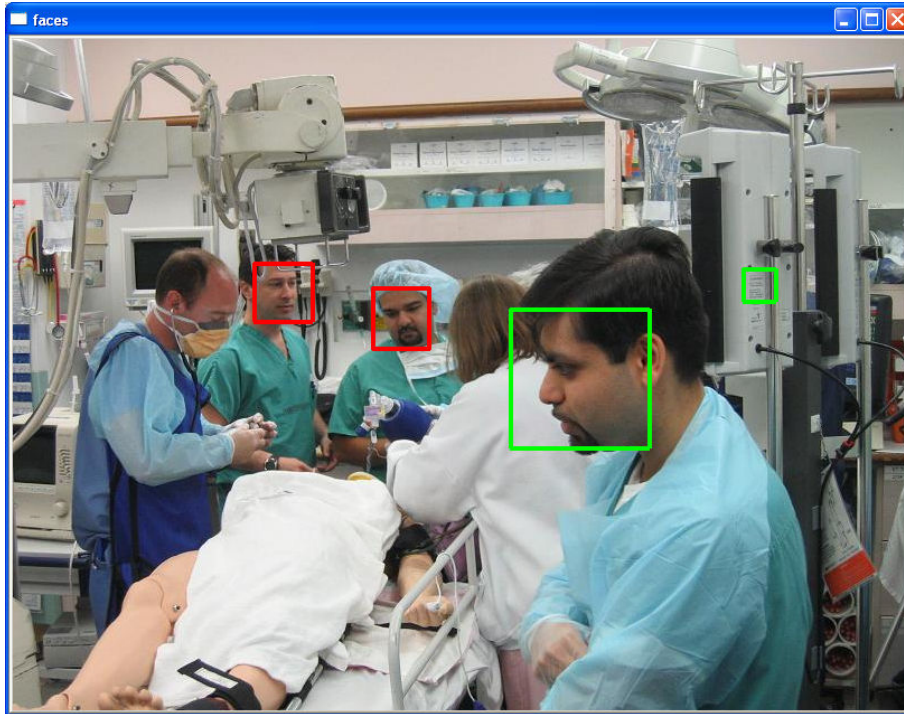


Figure 34: Face detection in a simulated resuscitation scene.

In the above scene all visible faces are detected. Except for the doctor wearing a mouth cap. Differences in yaw are detected and correctly colored.

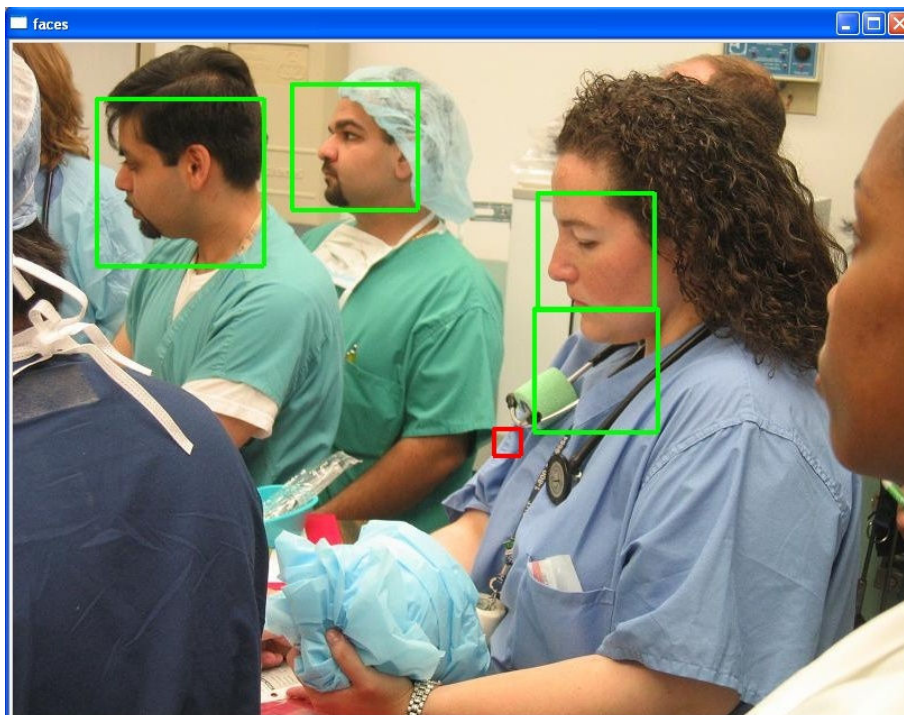


Figure 35: Face detection in a simulated resuscitation scene.

Once more all the visible faces are correctly detected in **Figure 35**. There is one false positive detected. It is very close to a face that is detected successfully.

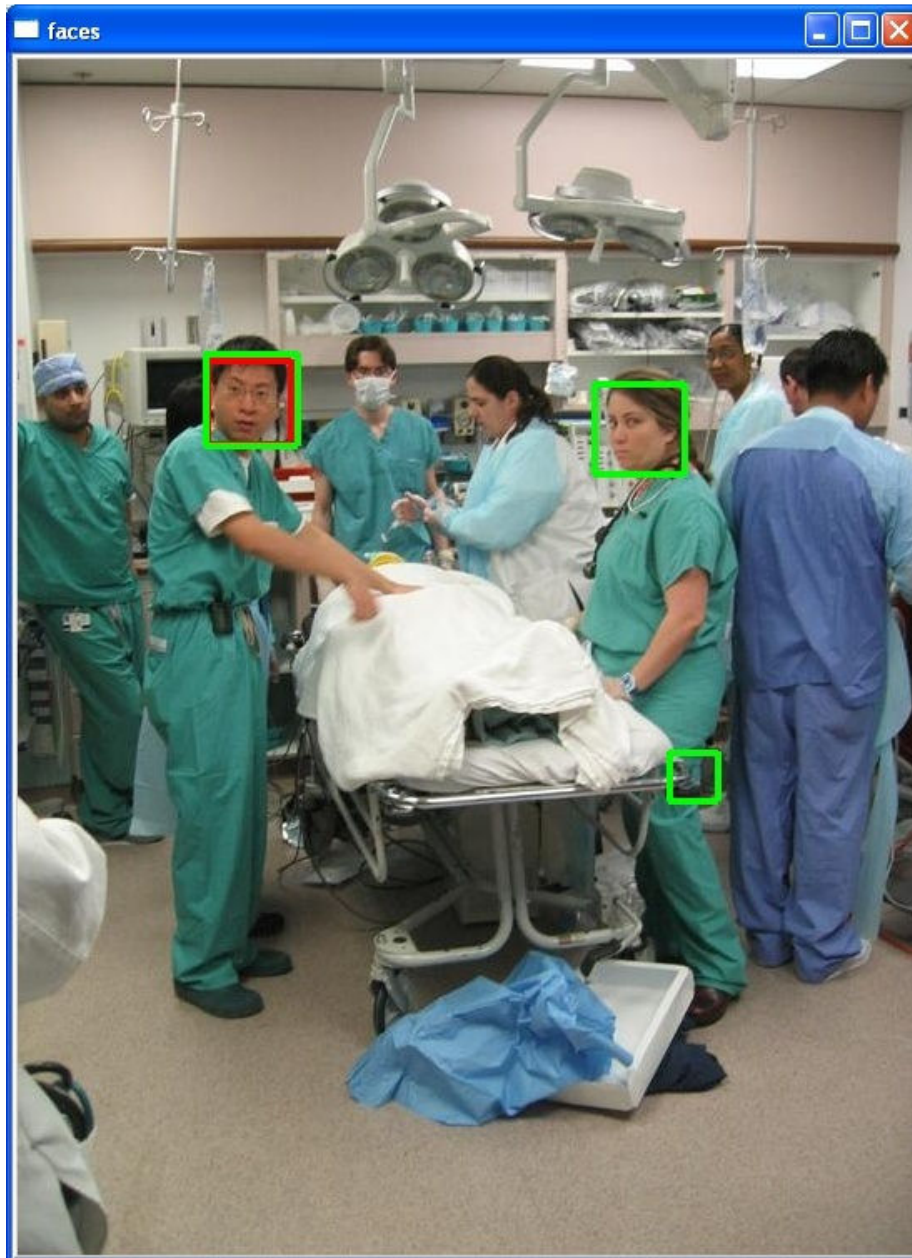


Figure 36: Face detection in a simulated resuscitation scene.

In **Figure 36** a lot of faces are not detected. One face is detected twice, with two different degrees in yaw. In **chapter 4.1** it was described that these two occurrences would be combined.

A mean is calculated for size and degree in yaw as variables of this face detection. These screenshots illustrate some of the problems the face detector is showing.

After testing the face detector on still images we conducted a few tests with the face detector on live camera footage. Two people are inhabiting this environment and we will investigate the detection rate of the face detector. **Figure 37** and **Figure 38** show two test cases of the face detector. They illustrate the errors commonly seen during testing.

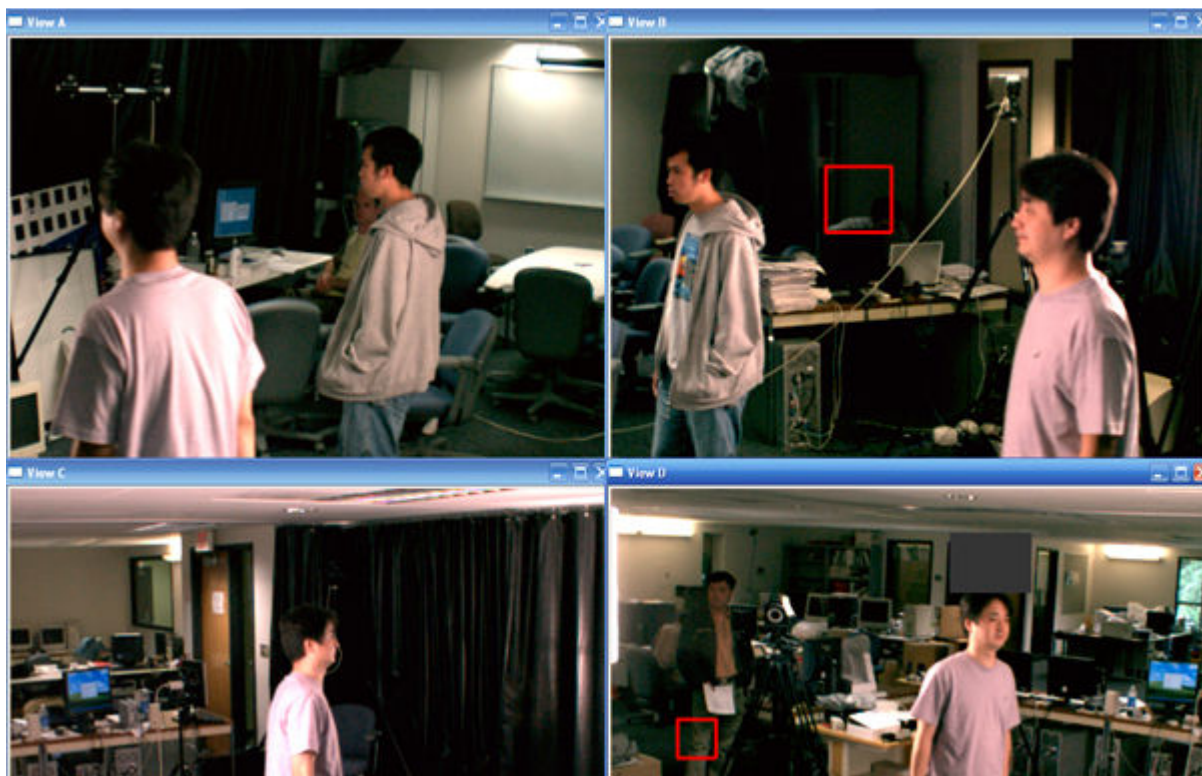


Figure 37: Illustration of face detection errors. Two false positives.

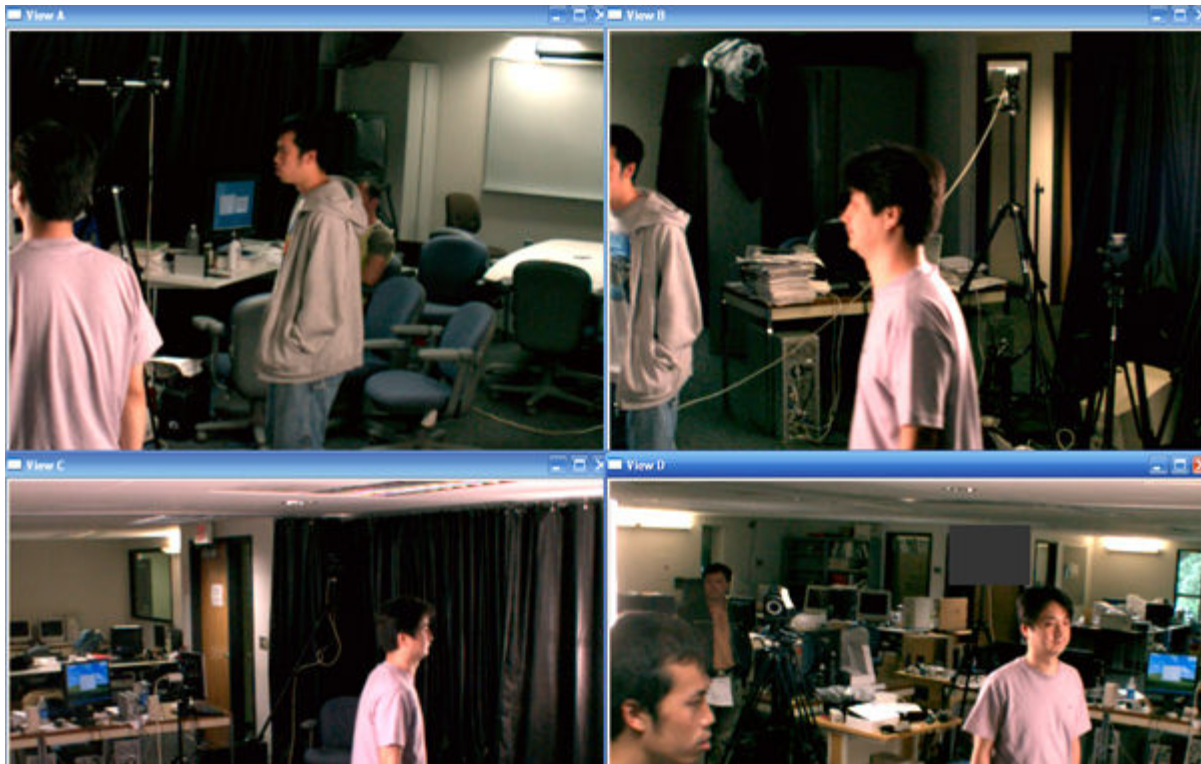


Figure 38: Illustration of face detection errors. No faces are detected.

In **Table 2**, 25 frames of live camera footage are plotted against the amount of false positives (detection of non-faces), true negatives (non-detection of faces) and correct detections.

Table 2: Illustration of 25 frames showing the amount of false positives, true negatives and correct detections of each inhabitant in the environment.

Frame number	False positives	True negatives	Correct detection
1	0	2	0
2	0	2	0
3	1	2	0
4	0	1	1
5	2	2	0
6	0	2	0
7	1	2	0
8	1	2	0
9	0	2	0
10	0	2	0
11	1	2	0
12	0	2	0
13	0	2	0
14	0	2	0

15	0	3	0
16	0	3	0
17	0	2	1
18	0	2	1
19	0	2	1
20	1	2	1
21	0	2	1
22	1	2	1
23	0	2	1
24	1	3	0
25	1	3	0

Very significant is the amount of missed detections against the amount correct detections. A lot of visual present faces are not being detected. When summing the totals over 100 frames the following statistics show up:

- False positives: 43.
- True negatives: 214.
- Correctly classified: 31.

The face detector only results in a **13%** detection rate. This is not reasonable enough for this tracking system to work. These face detection problems can have a variety of causes:

- Presented face examples hard to discriminate.
- Inadequate camera footage.
- Poor lighting conditions.
- Human posture shifts.

The same results appeared on similar tests with another frameset. It seems that the face detector is successfully implemented in parallel form and performs with a reasonable detection rate when applied to still shots. Unfortunately the performance of the detector on live camera footage was not as desired. We tried to train the face detector with specific training data gathered from the live scene but the detection rate was not better than with standard training. This phenomenon is common in many research projects at the TU Delft. Because the face detection is only the first

step of the system we wanted an alternative to test the remaining algorithms of the system. We simulated the face detection to test the remaining part of the system. When finished the system could try other solutions for a better face detector, for example an alternative from **chapter 2.2** or enhancing the face detector in future work which is described in **chapter 7.2**.

6.3 Virtual environment

For testing the virtual environment we introduced three inhabitants in the scene. Every cycle each inhabitant would move a predefined distance to a random direction. The position of each inhabitant p on the ground plane was used to locate the coordinate on the camera view. First from the position p the z -coordinate is incremented with the height of the inhabitant (in this case the standard height given by the imported model). A new ground intersection point is calculated by drawing a ray through the camera's optical center and the new incremented coordinate. Then with this new intersection point the appropriate pixel on the camera view is retrieved from the matrix A from camera calibration. **Figure 39** shows this process.

Figure 39: Illustration of the face detection algorithm in the simulated environment.

After the step of detecting the faces in the scene they are plotted on the frame buffer. **Figure 40** shows the scene with plotted ground points and detected faces. They are not yet classified and are all plotted with a white color.

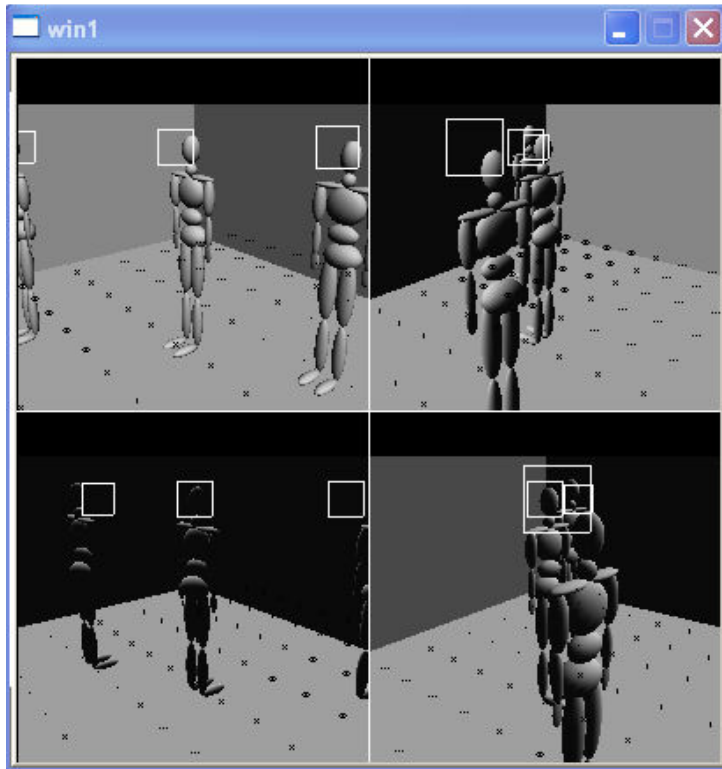


Figure 40: Face detection in the simulated environment.

Then the simulated environment and the tracking algorithm were combined. The algorithm during testing with the simulated environment was still in early phase of implementation but some important functionality could be checked. The same scene is shown in **Figure 41** but with the classification phase of the algorithm processed. Drawn in the image are the colored boxes that represent the detected faces. Their color represents the identity of a detected face. This color is chosen from a grayscale of 0 to 255 and step size differs between the amount of inhabitants in the environment. The step size is calculated by dividing the maximal grayscale value (255) by the amount of inhabitants.

If carefully investigated in every view the identity of each inhabitant is correctly plotted. The next steps in the algorithm were processed and the system produced an estimation of the position

of each inhabitant. After 50 cycles we stopped the process and plotted the graphical results. In **Figure 42** the output is shown.

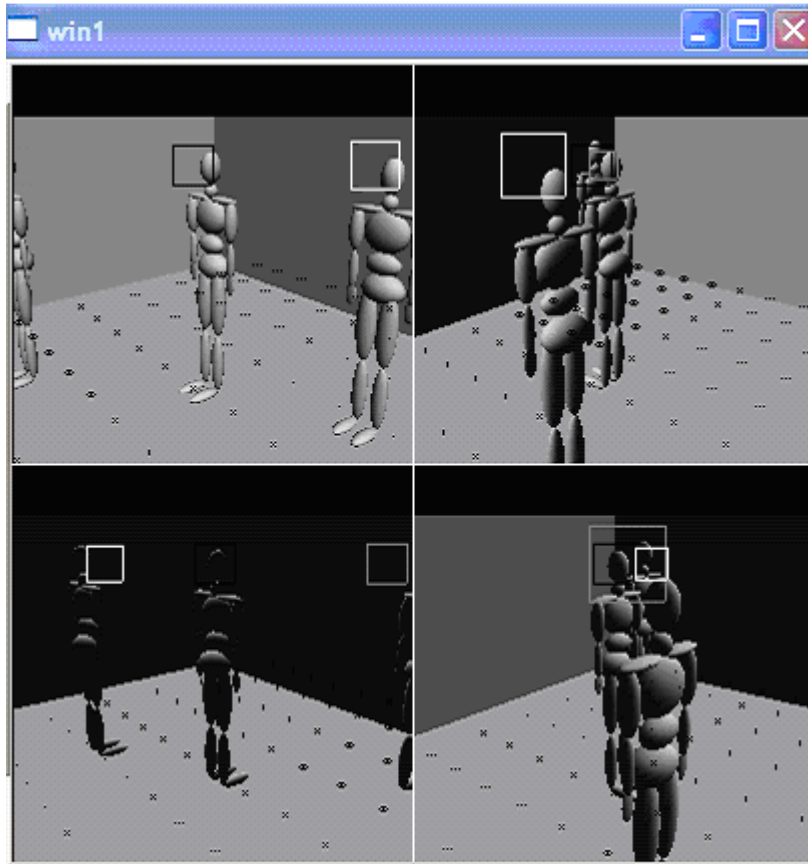


Figure 41: Face classification and color coding.

The likelihood map is showing the estimated presence of an inhabitant at a certain position. This early version did not yet discriminate between different inhabitant identities. We can compare these results with **Figure 43**, which shows an overview of the simulated scene that was constructed and processed until cycle 50.



Figure 42: Likelihood map after 50 cycles representing the estimated presence of an inhabitant.

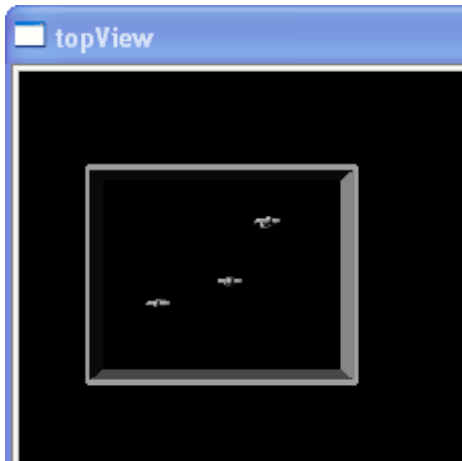


Figure 43: Top view of the simulated Environment at cycle 50.

The plotted results look promising. The positions of the inhabitants are more or less correctly estimated on the ground plane. Together they seem to be shifted a certain translation left and to the top but this could be because of small calibration errors. They can be investigated later on when more accuracy is needed.

After these results the main algorithm was further implemented and later tested on real camera footage. **Chapter 6.4** will describe these events.

6.4 Live environment

The most significant results are to be taken by testing with the live footage. For these images represent realistic examples of the environment. For testing the algorithm with live camera footage several testing scenarios are designed. These scenarios represent different environment characteristics.

6.4.1 The test scenarios

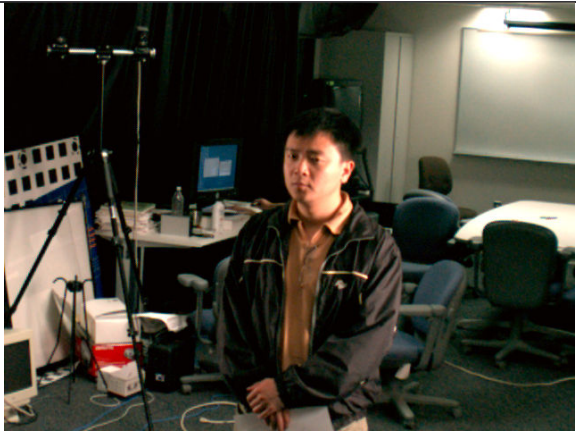

The testing scenarios try to encompass a wide range of possible situations. The scenarios start simple and gradually involve more people with more dynamic movement.



The following environment characteristics differ between scenarios:

- Amount of persons in the room.
- Whether persons are moving in the room.

The system will be tested using the following four testing scenarios:

Table 3: Summary of the 4 test scenarios showing only one of four camera views.

Testing Scenarios	
Scenario #1 Single person (45 frames) Static	
Scenario #2 Single person (100 frames) Dynamic movement	

<p>Scenario #3</p> <p>Two persons (100 frames)</p> <p>Dynamic movement</p>	
<p>Scenario #4</p> <p>Three persons (100 frames)</p> <p>Dynamic movement</p>	

6.4.2 Test results

The interface while testing shows us three distinct items.

- **Camera footage**

Four windows showing the live camera footage. In each window the face detection results are displayed by rectangles. Red rectangles represent detected faces that carry a label of one of the persons in the room, meaning they were classified to one of the inhabitants in the room. White rectangles are face detections that do not carry a label because they were not classified to a current inhabitant of the environment. In this overview it is easy to spot false positives and true negatives.

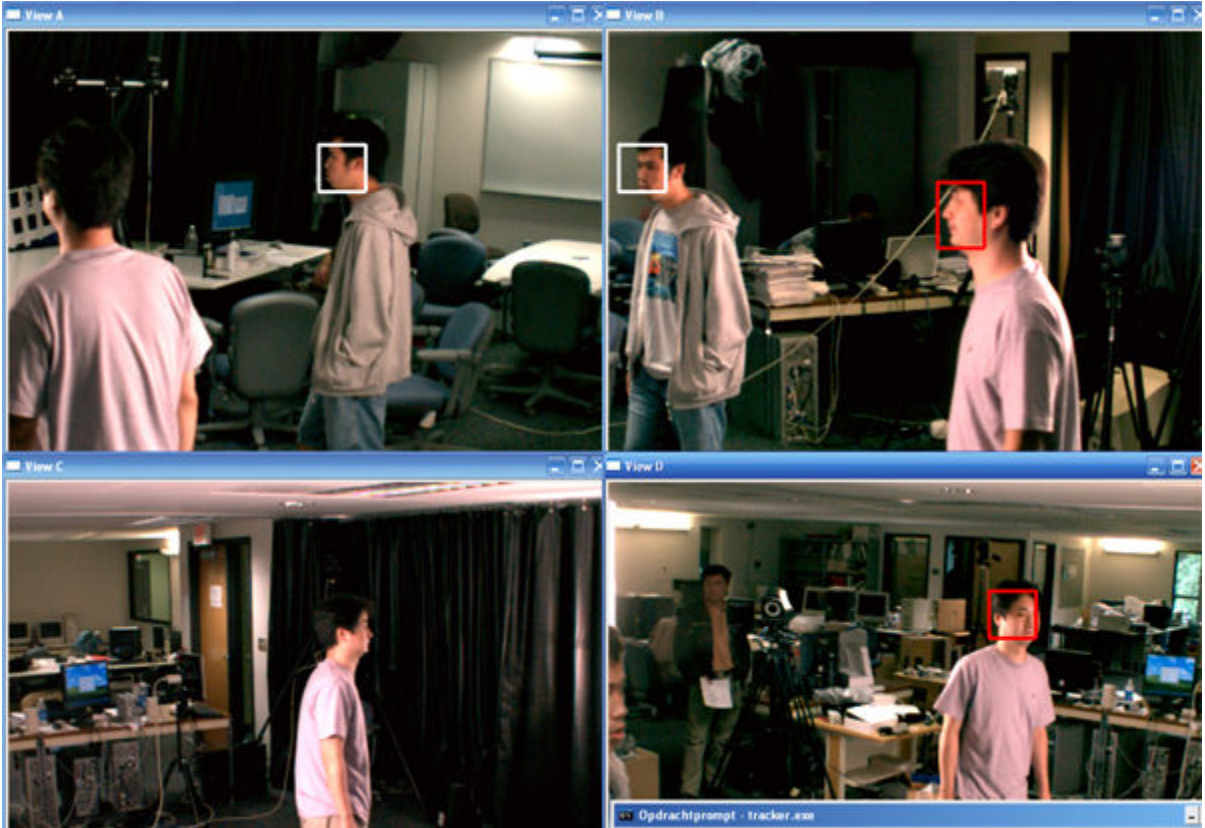


Figure 44: Illustration of camera footage showing four different angles. Rectangles show detected faces. Red illustrate labeled rectangles, white unlabeled.

- **Face detection classification**

Several windows show the individual likelihood maps for each present person in the environment. Each person represents one window. Every window shows the Gaussian projections from each face detection that was classified to that person into the environment. The blue region represents the threshold given in **equation 4.13**. From this region the current location of the individual is calculated. This position is represented by a red dot.



Figure 45: Illustration of two individual likelihood maps. Both represent an inhabitant in the environment. Red dot estimates the current position.

- **Final location map**

This window represents the estimated position of each person in the environment. Each person is represented by a white cross. This information comes directly from each prediction of the individual Kalman filters and are returned in one window.

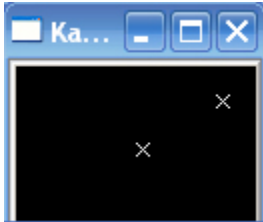


Figure 46: Illustration of the final location map.

Scenario #1

This scenario is the most basic. One person is standing in the room and is not moving. This basic scenario is a first test and shows how the algorithm operates. Below is a sequence of screenshots of the first 5 frames. In this scenario the Viola and Jones face detector is still used. It performs adequate enough on static inhabitants and is useable. The system starts exploring the environment. Few frames later the screenshots show a stabilized system.

In the first frame illustrated in **Figure 47** the face detector returns three face segments. Each of these rectangles is not classified because there is not yet one person active in the room. Two of these rectangles are correctly detected. One face detection is a false positive. In the face detection window (upper left corner) all three detected faces are added to the likelihood map of a new person. The red dot shows the current estimated position of that person. The final location map (upper right window) is still empty. No inhabitants are detected in the room.

The second frame shown in **Figure 48**, shows that all the face detections are classified to the new inhabitant. Therefore they are all colored red in the camera views. In the final location map (upper right window) we see a white cross inserted on the estimated position of this person.

The third frame illustrated in **Figure 49** shows the same information. We can see in the final

location map that the white cross is moving to the estimated point in the face detection window in the upper left window. This is the result of the kalman filter that is feeded with information every frame. It slowly converges the actual position.

In frame number 4 seen in **Figure 50** the false positive now is not detected. Two correctly classified face segments remain. This will result in a shift of the estimated position of the kalman filter because the likelihood map (upper left window) has changed to the current two face segments.

The fifth frame, illustrated in **Figure 51**, shows the return of the false positive in view D. But this time the algorithm has adapted to the actual position of the inhabitant and the estimated position retrieved from the kalman filter of the person does not coincide anymore with the position of this face segment. Therefore it is not classified to the inhabitant and remains white. The environment has now entered a stable state.



Figure 47: Screenshot taken while system processes frame #1.

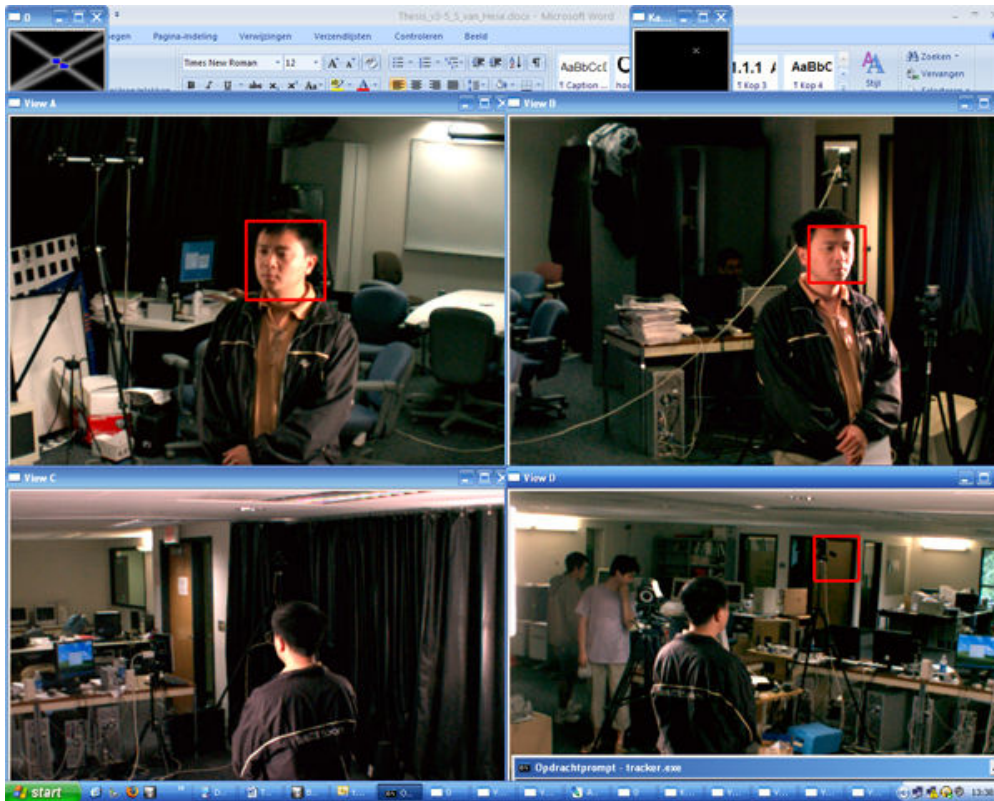


Figure 48: Screenshot taken while system processes frame #2.



Figure 49: Screenshot taken while system processes frame #3.

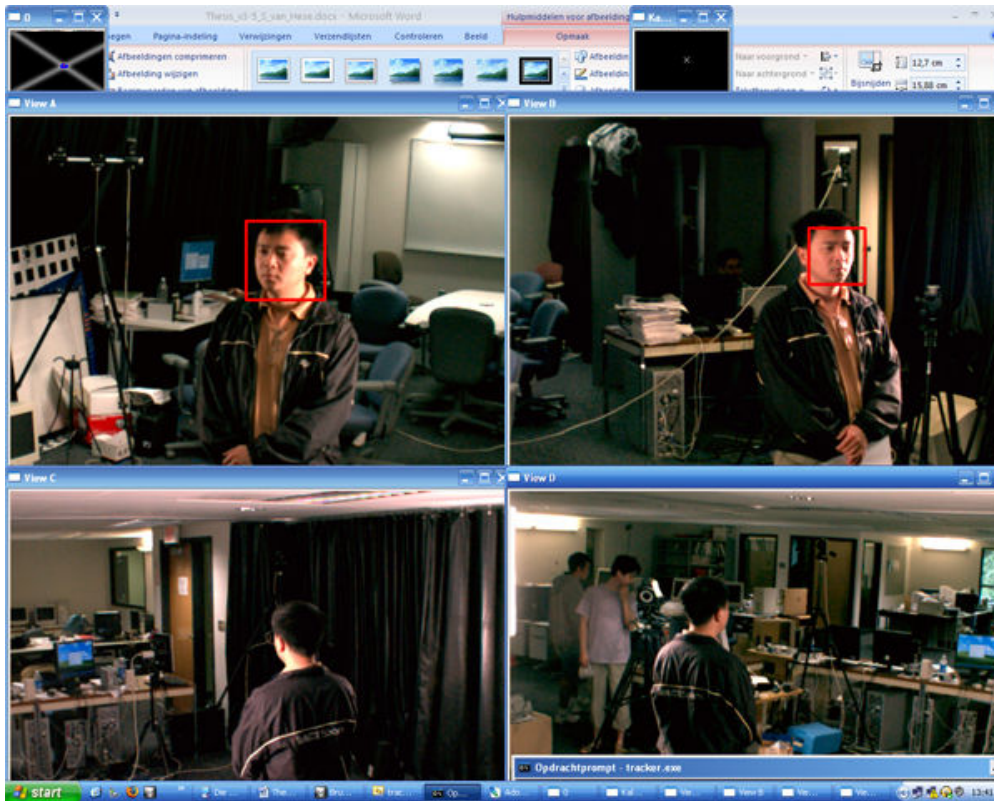


Figure 50: Screenshot taken while system processes frame #4.

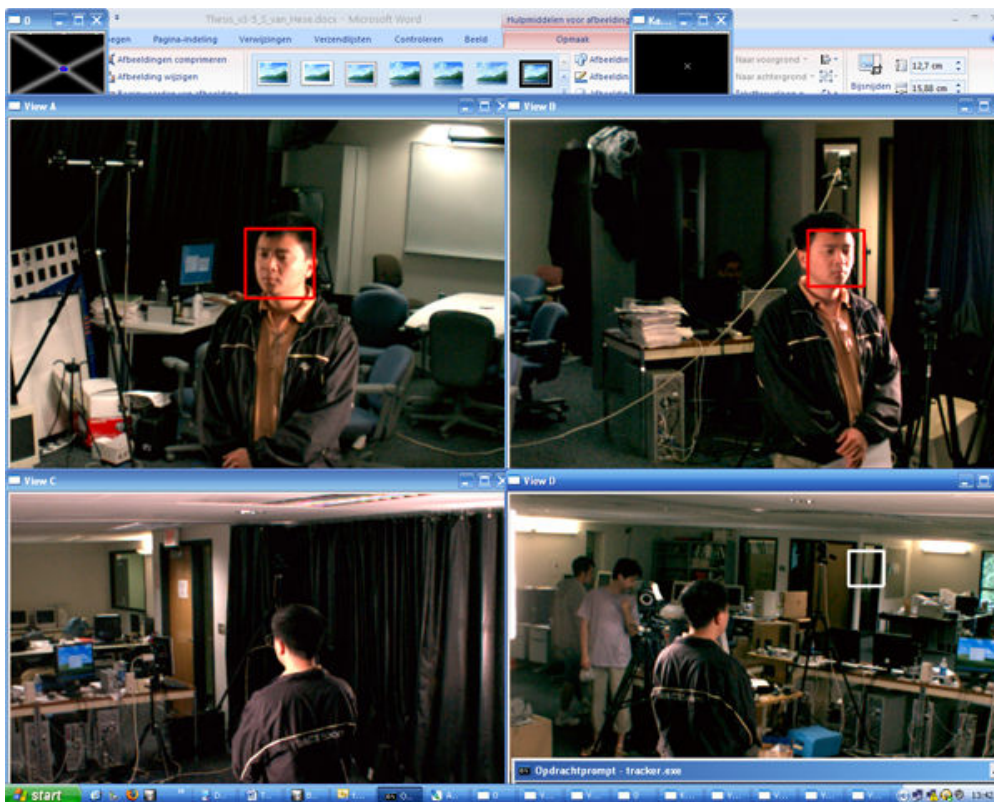


Figure 51: Screenshot taken while system processes frame #5.

After five frames the algorithm is aware of the position of the inhabitant and remains stable during all remaining frames. Other false positives will not influence the results because they will not be classified to the inhabitant. This process was described in **chapter 4.2**. Furthermore if one of the detected faces in view A or B disappears the estimated position of the inhabitant will not suddenly shift. The Kalman filter makes sure this is a gradual process. Below is illustrated the results of scenario #1 by printing the distance between the actual position of the inhabitant with the estimated position that is returned from the Kalman filter for each frame.

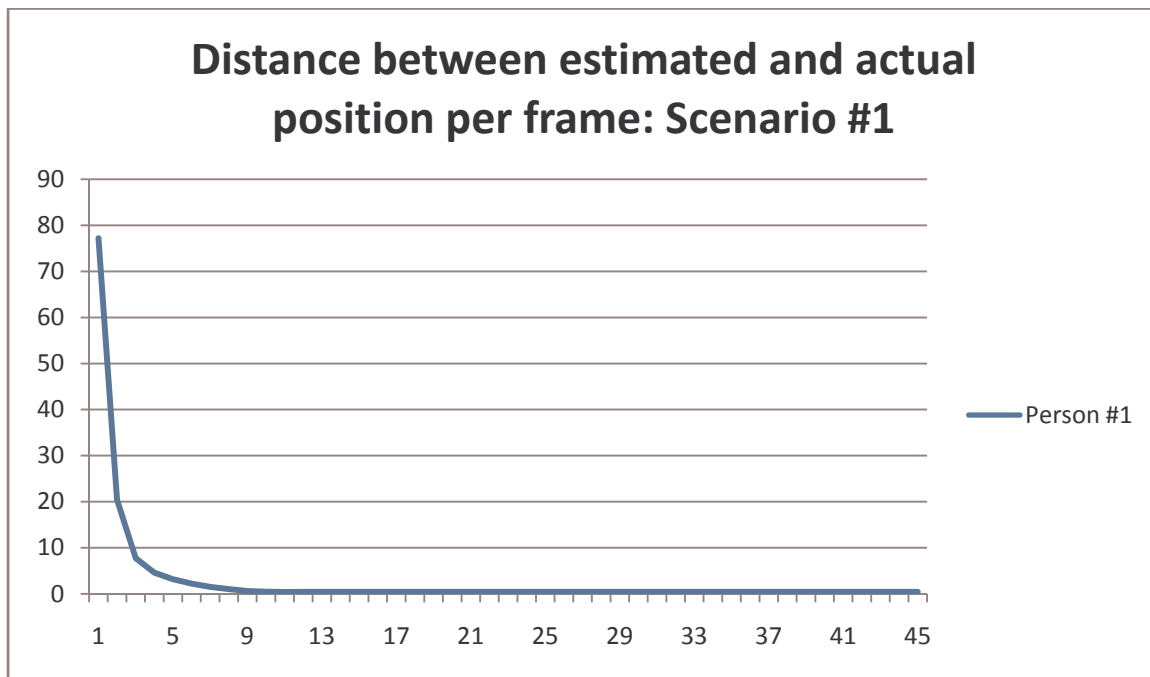


Figure 52: Distance between estimated and actual position per frame in scenario 1. Distance between estimated and actual position on the y-axis, frame number on the x-axis

The graph shows a rapid decrease of the distance between estimated position and actual inhabitant position. This is because the Kalman filter updates its predictions using previous time frames. After frame number 5 the estimated position has converged to the actual inhabitant position and false positives don't have anymore effect on this stable state.

Scenario #2

In this scenario one person is moving around in the environment. The basic tracking ability of

the system is tested on one object. Also in this scenario the Viola and Jones face detector is used for face detection. It performs adequate in this scenario with an detection rate higher than 60%. **Figure 53** shows a screenshot of the first frame processed by the system.



Figure 53: Screenshot taken while system processes frame #1.

During the first frames only view C (lower left window) shows a detected face. Because this is the only face detected no inhabitant is added to the environment because there is not enough geographical information. The location map window is empty and there is no likelihood window because there are no inhabitants.

When we make a step to frame 13 a face in view A is detected and a new inhabitant is added to the environment. This is illustrated in **Figure 54**.

Then frame 89 is illustrated in **Figure 55**. Due to several false positives in face detection a second inhabitant is added. The first inhabitant is recognized twice again and remains tracked. Eventually the second inhabitant will be discarded because of lack of new face detections.

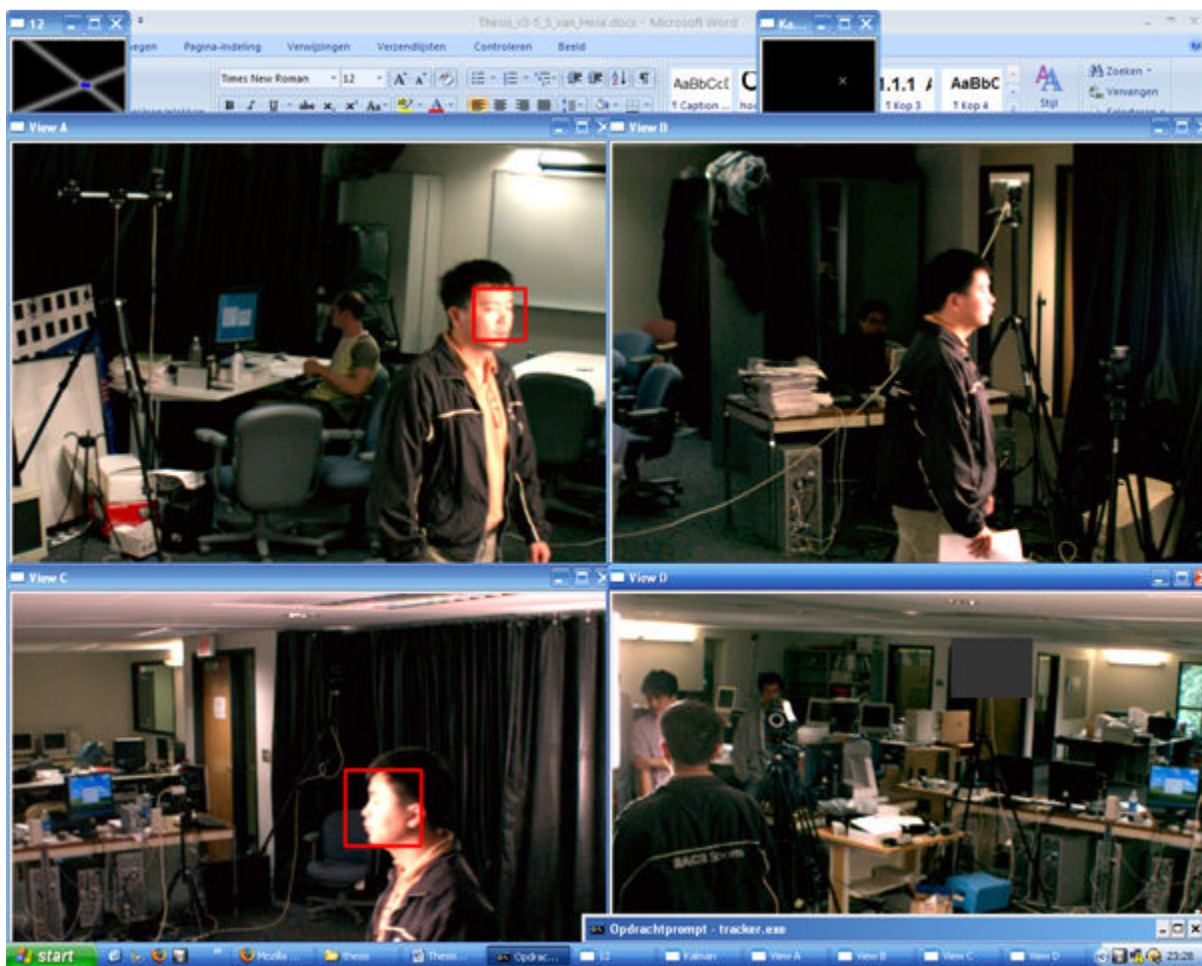


Figure 54: Screenshot taken while system processes frame #13.

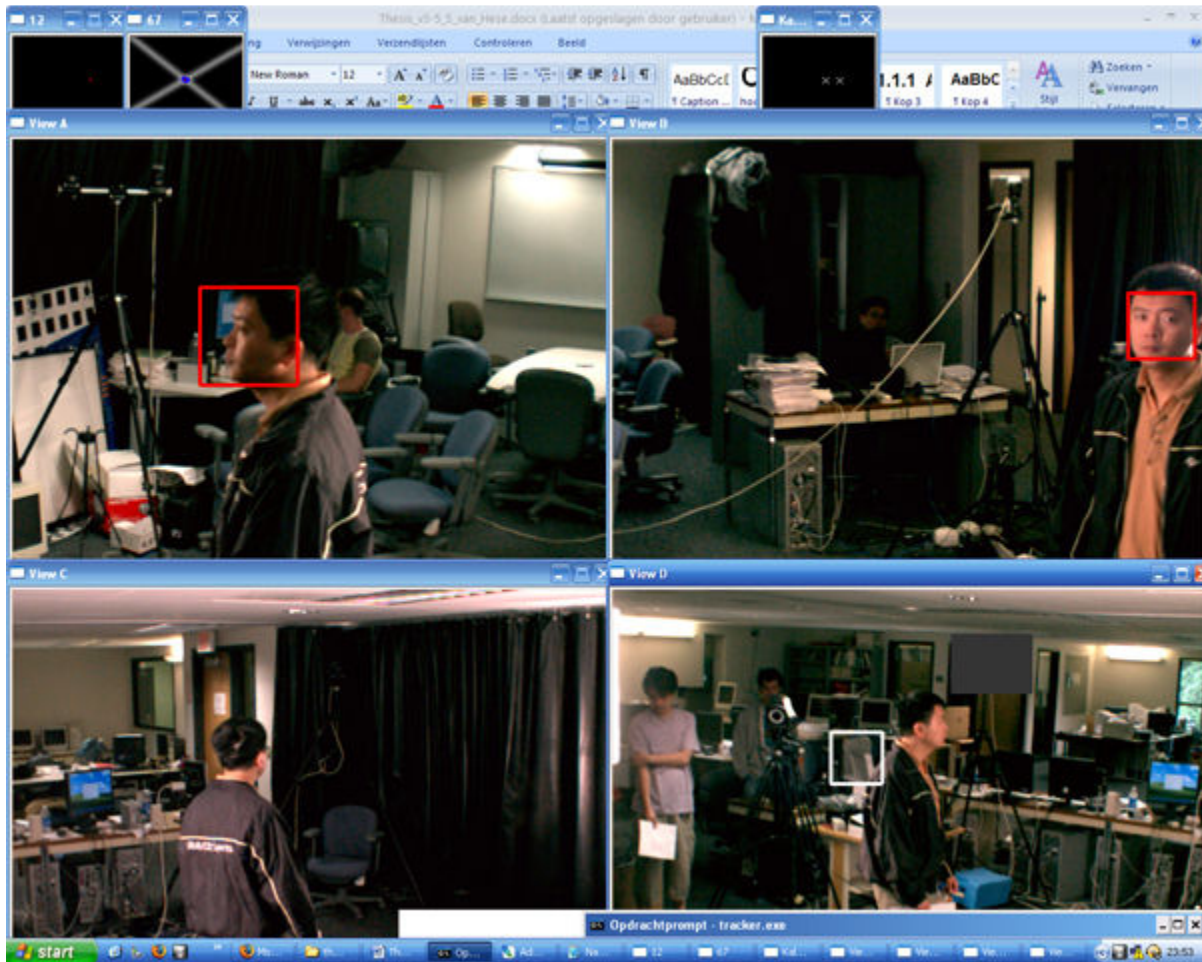


Figure 55: Screenshot taken while system processes frame #89.

At frame 13 a inhabitant is added because this is the first time two faces are detected. Then the distance between the estimated position and the actual position gradually decreases. During frames 25 and 85 mainly one face is detected and the Kalman filter does not get enough geometrical data to update its position. The distance increases. Around frame 85 two face detections again are present and the distance decreases again. The testing results are illustrated below.

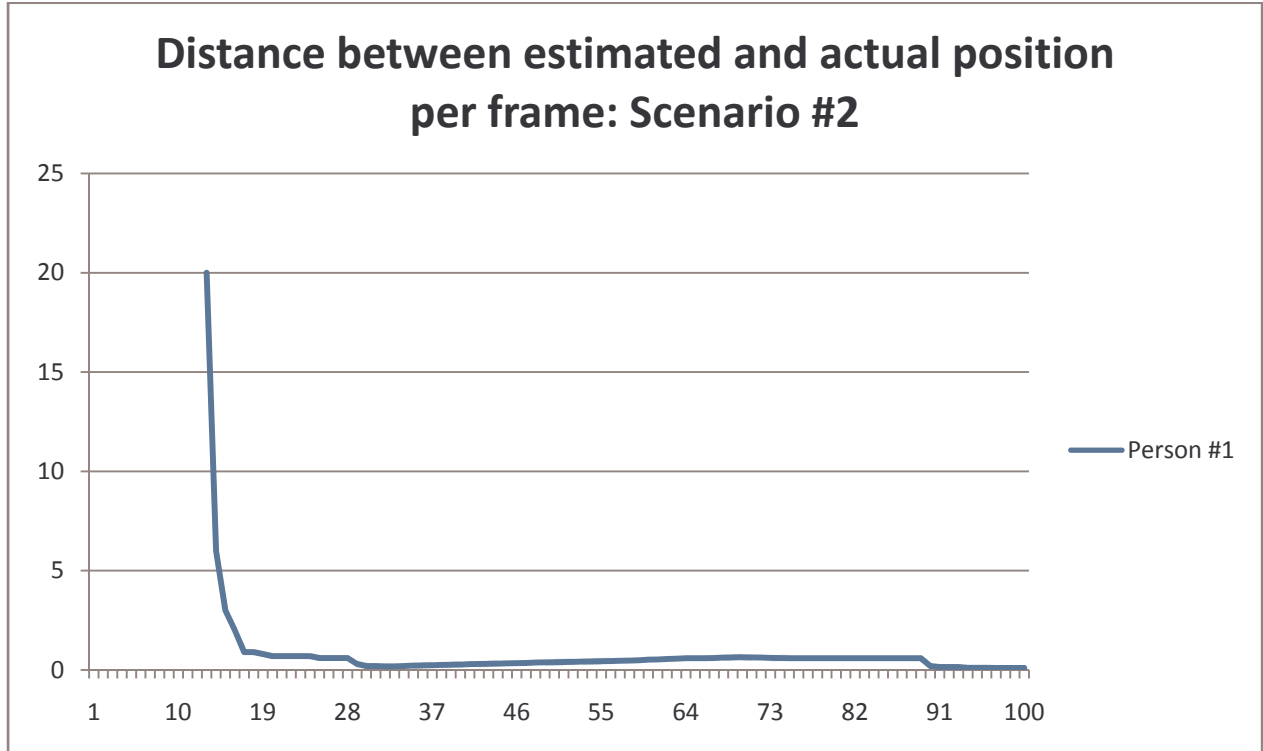


Figure 56: Distance between estimated and actual position per frame in scenario 2. Distance between estimated and actual position on the y-axis, frame number on the x-axis

Scenario #3

While testing this scenario adding two inhabitants to this environment seemed to have negative influence on the face detector. In the two previous scenarios the face detector worked reasonable and was sufficient because the environment was very basic. Now the errors produced by the face detector greatly influence the algorithms results. As mentioned in **chapter 6.2** the face detector only results approximately in a 13% detection rate. This is not reasonable enough for a tracking system to work.

For testing we decided to simulate the face detector by manually entering the face positions into the algorithm. This way we could test if the algorithm works appropriately. Two illustrations while processing scenario #3 are given in **Figure 57** and **Figure 58**.

Figure 57 shows in two views a face detection of the same person. This one person is tracked and located on the map. The second person is not added yet because only one face segment is known.

In **Figure 58** a second face is detected of the other inhabitant and his likelihood map is created together with an estimated position on the map. Now both inhabitants are shown on the location map in the upper right corner.

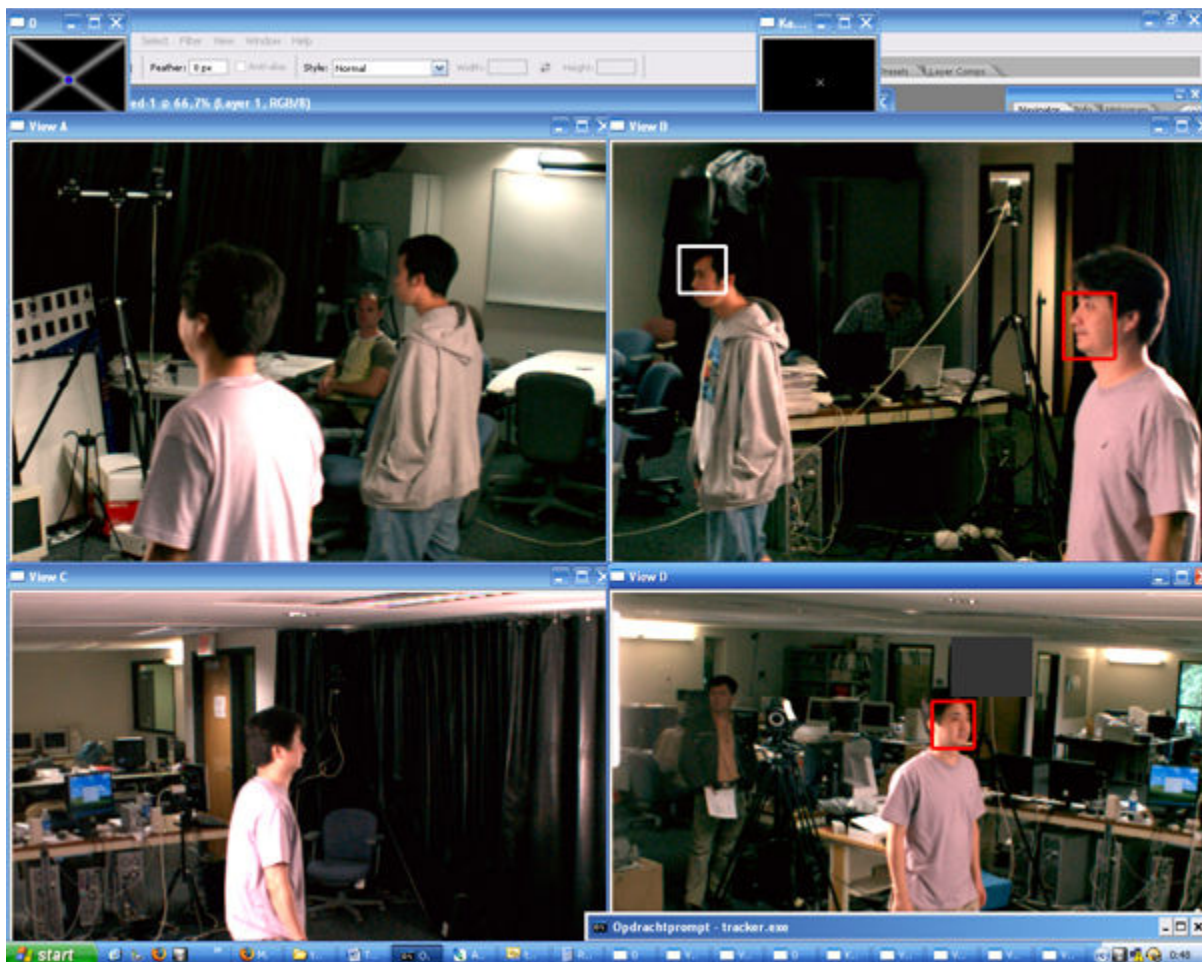


Figure 57: Illustration of the tracking system working in scenario 3.

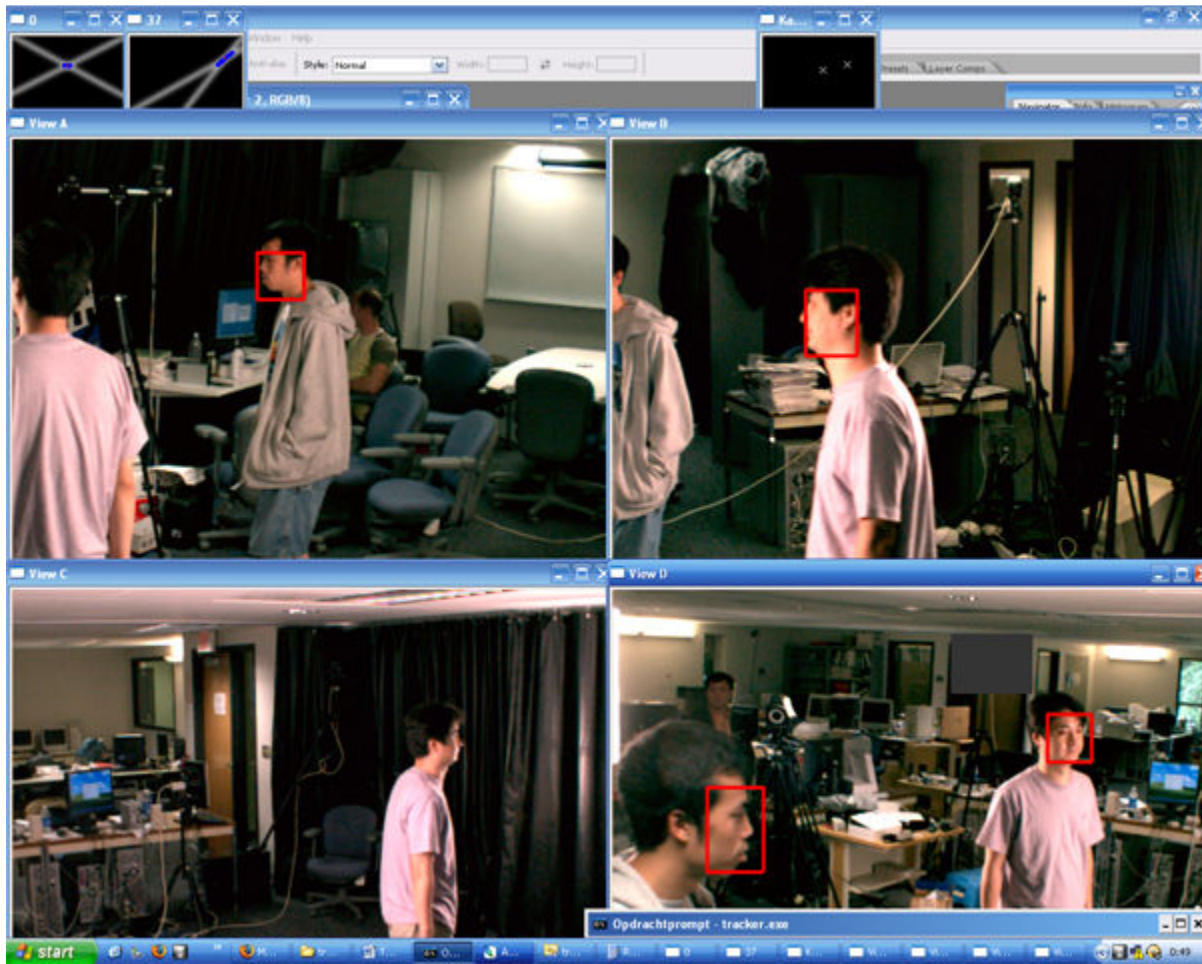


Figure 58: Illustration of the tracking system working in scenario 3.

The results are illustrated below. During frame 39 person #2 is detected in the environment and added to the inhabitant pool.

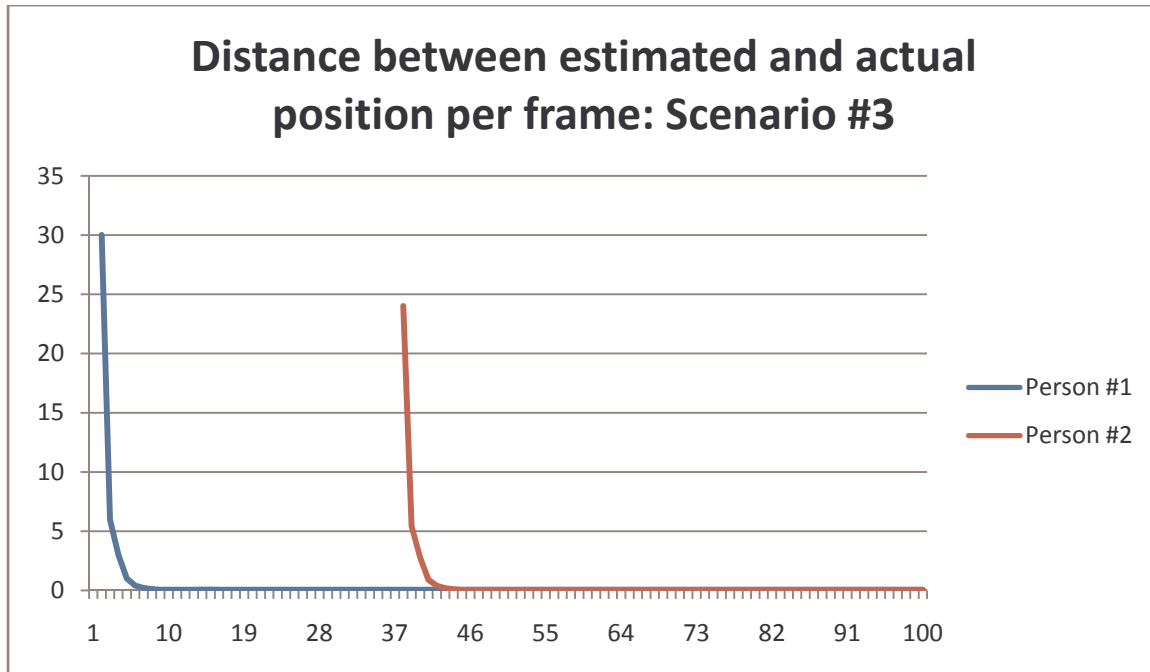


Figure 59: Distance between estimated and actual position per frame in scenario 3. Distance between estimated and actual position on the y-axis, frame number on the x-axis

To emphasize the tracking in scenario #3 below **Figure 60** is an illustration of the history of the location of both inhabitants during the test.

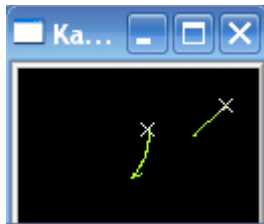


Figure 60: location trace of the two inhabitants.

In **Figure 57** and **Figure 58** we can see the movement of the two inhabitants. They are both directed at the top right corner. By tracing the path of the two inhabitants we can see their movement through the environment.

Scenario #4

The final scenario is a hard test scenario. Three people are moving around the room. **Figure 61** and **Figure 62** show two distinct moments in the scenario while being processed by the system.

Because the third person (white sweater) does not look up a lot, there is never more than one face detection of that person. This means that the person never is introduced to the scene.



Figure 61: Illustration of scenario #4.

During the process person 2 has a period where he only is detected in one view. The Kalman filter predicts its next position based on his previous movement. But the single face detection remains and the distance to the actual position of the person gradually increases until it cannot repair itself. When two new face detection segments present themselves a new person is created. This person used to be person 2. The old inhabitant is deleted from the environment. This result

is illustrated in **Figure 63**.



Figure 62: Illustration of scenario #4.

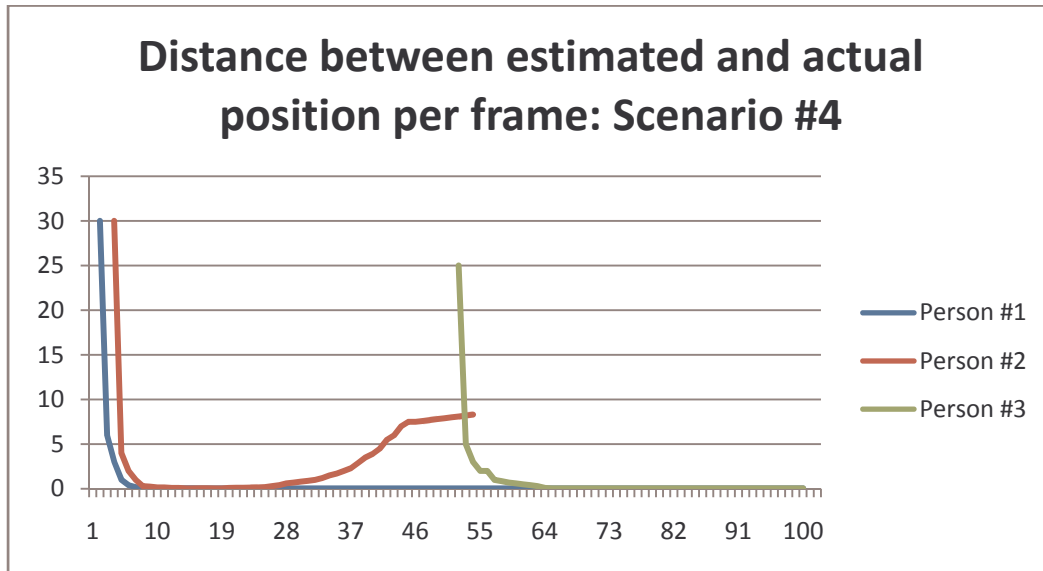


Figure 63: Distance between estimated and actual position per frame in scenario 4. Distance between estimated and actual position on the y-axis, frame number on the x-axis

Finally a graph is presented of this scenario showing the following information about detected face segments.

- Face segment unclassified, during the classification phase no inhabitant was found to classify this segment to.
- Correct classification, The face segment was classified to the correct inhabitant.
- Classification error. The face segment was classified to the wrong inhabitant.

Below for all frames the totals of the above mentioned categories are illustrated.

Around frame 55 we see a peak in correctly classified face segments. This is the moment where the system finds two new face detections and classifies them correctly to person 3, which is the new identity of person 2, after he was discarded.

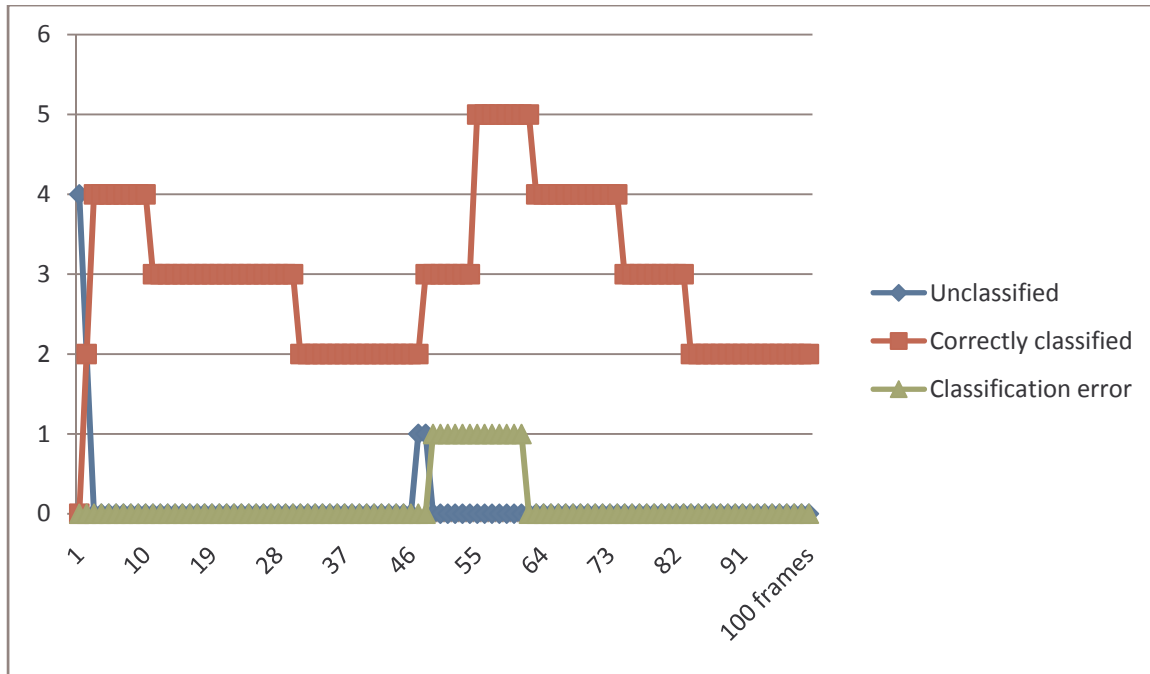


Figure 64: Illustrating the summed totals of classification results. Count of objects on the y-axis, frame number on the x-axis

6.5 Evaluation of test results

While testing using several testing scenarios we presented final results of the system. We will summarize in this section several items of the system and their final performance. If we compare the performance of each individual algorithm in the system we get a system that works very well if face detection is of a certain standard. This performance is shown in the following summary:

- **Face detection**, In the first two scenarios the face detector had a decent detection rate and is used in the final results. But in later scenarios only a **13%** detection rate was achieved and the detector was discarded. In scenario 3 and 4 a simulated detector was used to test the other algorithms of the system.
- **Classification**, Classifying detected objects by labeling them with an inhabitant worked very accurately in this system. **Figure 64** shows during the processing of 100 frames with three people walking in the scene a total of 298 correctly classified objects against 11 classification errors. A classification rate of **96.3%**. Even better statistics were retrieved

in the other test scenarios.

- **Object location estimation**, In all scenarios the location estimation works great. Each scenario shows the working of the location estimation. **Figure 52**, **Figure 56**, **Figure 59** and **Figure 63** show the distance between the estimated locations and the actual positions of each inhabitant. All three figures show an error **smaller than 0.5%** of the width of the room, for each inhabitant within 5 frames of its first detection.
- **Kalman tracking**, The specific Kalman tracking shows that it works when faces are not consistently detected. In scenario 2 multiple times faces are detected in one frame and in the next frame are not detected. The Kalman filter successfully predicts the estimated position of the next frame on previous data and supports the object location estimation by presenting it with up-to-date data. The results in **Figure 56** show that the Kalman filter is working accordingly. It keeps track off the object based on previous movement even when there is no new information.

More results of these individual steps are described in **chapter 7**.

Chapter 7

Conclusions and Future Work

7.1 Conclusions

In this paper, we have presented a system for segmenting, detecting and tracking multiple people using a multi perspective approach with synchronized cameras located in each room corner. For this system we have chosen an approach using face detection.

Summarizing the goals set in **chapter 1.2** we achieved the following results:

- The Viola and Jones was successfully implemented in parallel form. With still shots the face detector detected multiple variations in yaw at a reasonable detection rate. These screenshots were taken with a high resolution. Unfortunately the detection rate of the face detector on camera footage was not as desired. After training the face detector with specific training examples the detection rate was not better than with standard training. This phenomenon is seen in many researches performed at the TU Delft. Because the face detection is only the first step, we created a face detection simulator to test the remaining part of the system. The system showed outstanding performance and urges the need for a better face detector, for example an alternative from **chapter 2.2**.
- The classification algorithm worked perfectly. Together with the calibration of the cameras the algorithm is able to translate a point from the 2-dimensional camera view to the 3-dimensional space. From the estimated position of a inhabitant this algorithm calculated the likelihood of a detected face labeled to the same inhabitant. Objects were correctly classified to their representative inhabitant. **Figure 64** shows during the processing of 100 frames with three people walking in the scene a total of 298 correctly classified objects against 11 classification errors. A classification rate of **96.3%**. Even better statistics were retrieved in the other test scenarios.
- The algorithm for object-location estimation performed as expected. In the case of two

labeled objects the position of the inhabitant was estimated perfectly. Even when both labeled objects came from two cameras diagonally opposite, the threshold procedure took into account the overlap in likelihood rays and used the information only when a third labeled object appeared in the environment. The threshold cut-off created clusters sharing only the most informative points to estimate the inhabitants position. With three or more labeled objects for one inhabitant, the position was estimated accurately.

- The design and implementation of the Kalman filter was a great success. The model of the system was successfully described in matrices and implemented in the Kalman filter. If the object-localization did not return a location (not enough labeled face detections), the Kalman filter successfully predicted a new location based on previous data and served this input for a new iteration through the system. Movement of inhabitants was therefore tracked at all times. **Figure 60** shows this tracking in progress.
- When the new algorithms were developed an iterative system fed with camera images and implementing the communication flow between those algorithms was created. This linked all the components into a complete system and also served as an independent layer to the camera footage. The overall system was tested intensely and implemented the system of **Figure 15** successfully.
- The virtual environment was implemented completely with Coin3D and presented a great platform for testing developed algorithms and parts of the system. It greatly served its purpose as an easy adjustable environment, inputting camera frames into the system. It also served successfully as testing ground for the real camera footage. Camera calibration for example was thoroughly tested in virtual space and gave invaluable information for live deployment. A ground plane grid was successfully projected into the virtual scene and these methods were later successfully used in the live environment.
- Finally the live environment was setup and resembled environments common to the applications described in **chapter 1.4**. Probably even more light sources and covering up of the background by using drapes around the scene would have been beneficial to the face detection rate but would also imply a lack of similarity to future environments. The live environment successfully produced synchronized camera footage and presented it to the system. The cameras were calibrated and the complete system worked as expected. **Figure 52, Figure 56, Figure 59** and **Figure 63** show the final results of the estimated

positions against the actual positions of each inhabitant. All graphs show an error **smaller than 0.5%** of the width of the room, for each inhabitant within 5 frames of its first detection.

This system is successful in segmenting, detecting and tracking multiple people including handling occlusion of objects and can be used in many applications that have similar environments. Results are excellent when face detection rates are of reasonable standard. Getting these detection rates to reasonable standards is a hard but realistic task. In **chapter 7.2** more possible solutions to this problem are described.

7.2 Future work

This research leaves room for enhancements and future extensions. Some of these are:

- A possible improvement is to allow specifying the parts of the room which are excluded for people motion because of obstacles preventing it. For examples, in the trauma bay, team members cannot be located in the region where the stretcher is located.
- **Formula 4.6** may be extended to include history information from several previous time steps:

$$P(x | f) = P(pos(f)_t | pos(f)_{t-1}, \dots, pos(f)_{t-n}) \cdot P(ori(f)_t | ori(f)_{t-1}, \dots, ori(f)_{t-n}).$$

- Also **formula 4.6** can be extended to also include the likelihood of certain facial features that appear in the face detection. The formula would be extended as follows:

$$P(x | f) = P(pos(f)_t | pos(f)_{t-1}) \cdot P(ori(f)_t | ori(f)_{t-1}) \cdot P(facefeat(f)_t | facefeat(f)_{t-1}).$$

Where *facefeat(f)* returns a matrix with in each column a vector of a face feature used.

These face features can for example be:

- Mean color of the rectangle.
 - Eye or mouth position.
 - Size of the face.
- Unfortunately the detection rate of the face detector used on camera footage in this research was not as desired. Not only the detector itself can be replaced by an alternative, the detector can have some possible extensions that could make it perform better in this domain:
 - Currently for face detection one camera view is independent from the other cameras. The fact that more cameras are used in this research can enhance the face detector by using information from other camera views.
 - A detected face in one view A means a higher probability of detecting a face at the translated 2D position in view B of the imaginary line in 3D space from camera A to the face. This information can be used by creating likelihood maps for every view. Also a detected face without an equivalent in another view is more likely to be a false positive.
 - Algorithms can use the knowledge of a non-detected face in one view to minimize the occurrence of false positives.
 - When seeing a certain face in a certain ROP, you can make a prediction about the ROP shown in other views. That adds to the probability of face detection and lowers the false positive rate.

References

- Bergs, E. A. (2005). Communication during trauma resuscitation: Do we know what is happening? *Injury* 36(8) , (pp. 905-911).
- CAIP. (2007, August). *Center for Advanced Information Processing*. Retrieved August 2007, from Center for Advanced Information Processing: <http://caip.rutgers.edu>
- Clarke, J. R.-R. (2000). An objective analysis of process errors in trauma resuscitations. *Academic Emergency Medicine* 7(11) , (pp. 1303-1310).
- Davis, A. M. (2002). M2Tracker: A Multi-View Approach to Segmenting and Tracking People in a Cluttered Scene Using Region-Based Stereo. *IJCV'02*.
- Elgammal, A. M. (2005). Learning to track: Conceptual manifold map for closed-form tracking. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR '05)*. San Diego, CA.
- Elgammal, A. M. (2001). Probabilistic framework for segmenting people under occlusion. *Proceedings of the 8th IEEE International Conference on Computer Vision (ICCV '01)*, (pp. 145-152).
- Elgammal, A. M. (2003). Probabilistic tracking in joint feature-spatial spaces. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR '03)*, (pp. vol. 1, pp. 781-788). Madison, Wisconsin.
- Fitzgerald, M. C. (2006). Using video audit to improve trauma resuscitation—time for a new approach. *Canadian Journal of Surgery* 49(3) , (pp. 208-211).
- Gertner, A. S. (1998). TraumaTIQ: Online decision support for trauma management. *IEEE Intelligent Systems* 13, (pp. 32-39).
- Haritaoglu, I. D. (1998). A real time system for detecting and tracking people. *3rd International Conference on Face and Gesture Recognition*, (pp. 222-227).
- Huang, C. A. (17-21 Oct. 2005). Vector boosting for rotation invariant multi-view face detection. *Tenth IEEE International Conference on Computer Vision*, (pp. 446-453 Vol.1). ICCV 2005.
- Isukapalli, R. A. (2006). Learning to identify facial expression during detection using Markov Decision Process. *Proceedings of the 7th International Conference Automatic Face and Gesture Recognition*. Southampton, UK.
- Ivan Marsic, A. S. (2008). Quantifying adaptation parameters for information support of trauma

teams. *CHI Extended Abstracts 2008* , (pp. 3303-3308).

Jones, P. V. (May 2004). Robust real-time face detection. *International Journal of Computer Vision* , (pp. 137-154).

Lei, B. H. (1999). Reviewing Camera Calibration and Image Registration Techniques.

Mackenzie, C. F. (2003). Video techniques and data compared with observation in emergency trauma care. *Quality & Safety in Health Care 12(Suppl 2)* , ii51-ii57.

Marcel, S. (2007, December). *Sébastien Marcel - Face detection*. Retrieved December 2007, from Face detection: <http://www.idiap.ch/~marcel/demos.php>

Oakley, E. S. (2006). Using video recording to identify management errors in pediatric trauma resuscitation. *Pediatrics 117(3)* , (pp. 658-664).

Papageorgiou, C. T. (1998). A trainable pedestrian detection system. *IEEE International Conference on Intelligent Vehicles*, (pp. 241-246).

Ram, T. Z. Nevetia Bayesian Human Segmentation in Crowded Situations.

S. Z. Li, L. Z. (2002). Statistical Learning of Multi-View Face Detection. *ECCV 2002*.

Ventrella, J. (2005, January). *Yaw Pitch Roll*. Retrieved November 2007, from Yaw Pitch Roll: http://www.ventrella.com/Ideas/YawPitchRoll/yaw_pitch_roll.html

Wern, C. A. (1997). Pfunder: Real-time tracking of human body. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, (pp. 780-785).

Wu, B., & Nevetia, R. (Oct 2005). Detection of Multiple, Partially Occluded Humans in a Single Image by Bayesian combination of Edgelet Part Detectors. *ICCV'05*, (pp. 17-20). Beijing.

Yoav Freund, R. E. (1999). A Short Introduction to Boosting. *Journal of Japanese Society for Artificial Intelligence* , 14(5): (pp. 771-780).