REAL-TIME 3D HEAD POSITION TRACKER SYSTEM WITH STEREO CAMERAS USING A FACE RECOGNITION NEURAL NETWORK

BY

JAVIER IGNACIO GIRADO
B. Electronics Engineering, ITBA University, Buenos Aires, Argentina, 1982
M. Electronics Engineering, ITBA University, Buenos Aires, Argentina 1984

THESIS

Submitted as partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Computer Science
in the Graduate College of the
University of Illinois at Chicago, 2004

Chicago, Illinois

ACKNOWLEDGMENTS

I arrived at UIC in the winter of 1996, more than seven years ago. I had a lot to learn: how to find my way around a new school, a new city, a new country, a new culture and how to do computer science research. Those first years were very difficult for me and honestly I would not have made it if it were not for my old friends and the new ones I made at the laboratory and my colleagues. There are too many to list them all, so let me juts mention a few examples.

I would like to thank my thesis committee (Thomas DeFanti, Daniel Sandin, Andrew Johnson, Jason Leigh and Joel Mambretti) for their unwavering support and assistance. They provided guidance in several areas that helped me accomplish my research goals and were very encouraging throughout the process. I would especially like to thank my thesis advisor Daniel Sandin for laying the foundation of this work and his continuous encouragement and feedback. He has been a constant source of great ideas and useful guidelines during my Thesis' program. Thanks to Professor J. Ben-Arie for teaching me about science and computer vision. Thanks to Andy Johnson and Jason Leigh for keeping me in track, and expand my vision of sci-fi and anime.

I would also like to recognize Maxine Brown and Laura Wolf who have provided me very helpful technical tips for organizing and writing several research documents, papers and this dissertation.

Most at all, I would want to express my sincere gratitude to my friends, Laura Wolf, Brenda Lopez-Silva and Cristian Luciano for their help, both personally and professionally.

JIG

TABLE OF CONTENTS

CHAPTER **PAGE**

TABLE OF CONTENTS (continued)

CHAPTER **PAGE**

LIST OF TABLES

TABLE **PAGE**

LIST OF FIGURES

FIGURE **PAGE**

LIST OF FIGURES (continued)

FIGURE **PAGE**

LIST OF ABBREVIATIONS

1D	One Dimensional
2D	Two Dimensional
3D	Three Dimensional
AGAVE	Access Grid Augmented/Autostereo Virtual Environment
ANN	Artificial Neural Networks
AVI	Audio Video Interleaved
CCD	Charge-Coupled Device
CMOS	Complementary Metal Oxide Semiconductor
CNS	Central Nervous System
CPU	Central Processing Unit
EVL	Electronic Visualization Laboratory
fps	frames per second
GUI	Graphic User Interface
HMD	Head Mounted Display
IEEE	Institute of Electrical and Electronics Engineers
IR	Infrared
LAMSTAR	LArge Scale Memory STorage And Retrieval
LCD	Liquid Crystal Display
NCC	Normalize Cross-Correlation
NN	Neural Networks
OS	Operating System
PC	Personal Computer
RF	Radio Frequency

LIST OF ABBREVIATIONS (continued)

SAD	Sum of Absolute Differences
SGI	Silicon Graphics, Inc.
SIMD	Single Instruction Multiple Data
SOM	Self-Organizing-Map
SSD	Sum of Squared Differences
SSE	Streaming SIMD Extensions
TCP/IP	Transmission Control Protocol/Internet Protocol
UIC	University of Illinois at Chicago
VR	Virtual Reality
WTA	Winner-Take-All
ZISC	Zero Instruction Set Computing
μP	Microprocessor

SUMMARY

Creating credible virtual reality (VR) computer-generated worlds requires constant updating of the images in all displays to have the correct perspective for the user. To achieve this, the computer must know the exact position and orientation of the user's head. Examples of current techniques for addressing head tracking include magnetic and acousto-inertial trackers, both requiring the user to wear clumsy head-mounted sensors with transmitters and/or wires.

This thesis describes a two-camera, video-based, tetherless 3D head position tracker system specifically targeted for both autostereoscopic displays and projection-based virtual reality systems. The user does not need to wear any sensors or markers.

The head position technique is implemented using Artificial Neural Networks (ANN), allowing the detection and recognition of upright, tilted, frontal and non-frontal faces in the midst of visually cluttered environments. In developing a video-based object detector using machine learning, three main sub-problems arise: first, images of objects such as faces vary considerably with lighting, occlusion, pose, facial expression, and identity. Second, the system has to deal with all the variations in distinguishing objects (faces) from non-objects (non-faces). Third, the system has to recognize a target face from other possible faces so it can identify the correct user to track.

This thesis introduces some solutions to problems in the face detection/recognition domain. For example, it discusses several Neural Networks (NN) per left and right channel, one for recognition, one for detection and one for tracking; real-time NN face and background training (a novel solution which allows new users to spend only two minutes training before being able to use the system); infrared (IR) illumination (to further reduce image dependency cause by room lighting variation) and global image equalization (in place of current trend of local equalization); algorithms highly tuned for the Intel Pentium IV vector processor; and a prediction module to achieve faster

frame rates once a face is been recognized. The goal is to reach real-time tracking, in our case 30 frames per second (fps), at 640 by 480 video-image resolution. The system has been evaluated on an ongoing autostereoscopic Varrier™ display project achieving 30 frames per second (fps) at 320x240 video-image resolution and 90% tracking position rate. In addition, this dissertation also includes a previous novel work in face detection using LArge Scale Memory STorage And Retrieval (LAMSTAR) neural network from which the current 3D tracker system is derived.

1. INTRODUCTION

1.1. Background

The Electronic Visualization Laboratory (EVL) is one of several research groups working on producing PC-driven, projection-based virtual reality (VR) displays. The use of high-end systems, such as EVL's CAVE® and ImmersaDesk are well established in application areas such as computational science, automotive engineering and chemical exploration. The next-generation of VR displays, both tiled LCD displays and projection-based, aim to eliminate encumbrances on the user. The trend is towards higher resolution displays where the user is not required to wear special glasses to view stereoscopic scenes. Analogously, the trend for interaction with these displays is towards lightweight and tetherless input-devices.

EVL and its collaborators are exploring the use of other modalities (such as vision, speech and gesture) as human-computer interfaces for this new generation of VR systems. Gesture recognition can come from either tracking the user's movements or processing them using video camera input. Gaze direction, or eye tracking, using camera input is also possible. Audio support can be used for voice recognition and generation, as well as used in conjunction with recording tele-immersive sessions. Used together, these systems enable tetherless tracking and unencumbered hand movements for improved interaction and collaboration within the virtual scene.

1.2. Motivation

The main objective behind this thesis is to replace current technologies used in commercial tracker systems. Beside their advantage, all of them have one disadvantage in common: sensors or markers (usually small pieces of reflective material) have to be attached to the objects to be tracked. Some of the sensors are heavy and/or bulky and/or need wires to connect them to the tracker system (receiver). If a wireless transmitter is provided to send data from the sensors to the system, still the tracker user has to carry it (including the batteries).

In case of markers, some tracker systems need a room in which the object being tracked has to be very well (controlled) illuminated in order for the markers to reflect the light. Therefore this tracker system is not suitable for projection-based virtual reality environment in which a dark or low-light room condition is desirable; otherwise the projection over the screen cannot be seen properly. On other systems the camera provides an infrared light and the marker will reflect it back to the camera and get its position, but still the user has to wear those markers.

As a conclusion, having to 'wear' something (sensor, markers, wires, etc.) is always an inconvenience so the motivation is to build a complete sensor-free or marker-free tracker system.

To perform this, the new system will rely only on two cameras in front of the object to be tracked (in this case the face). The system will have comparable features and characteristics to the system it tries to replace plus the sensor-free advantage. Some other possible advantages such as multiple face tracking, face recognition, hand detection, etc. will be mentioned at the conclusion of this proposal.

1.3. The Goal

My goal is to show that the face recognition, detection and tracking problem can be solved efficiently and accurately using image-based approach implemented with Artificial Neural Networks (ANN), also achieving real-time operation. Specifically, I will demonstrate how to recognize, detect and track upright, tilted, and non-frontal faces in cluttered grayscale video images at 30 fps and 320x240 pixel resolution using multiple neural networks (NN), arbitration among some of them and a simple prediction scheme. I will also show that the NN can be trained to achieve all these goals in a very short period of time (less than two minutes) making the tracker system very convenient for real-time operation.

1.4. Artificial Neural Networks

Artificial Neural Networks (ANN) are biologically inspired systems, which in very gross manner perform functions analogous to the most elementary functions of the biological Central Nervous System (CNS). Thus, ANN has a number of properties and characteristic of the CNS, such as:

- Learning from experience.
- Generalizing from previous examples to new ones.
- Recognition of complex patterns.
- Nonlinear analysis of multidimensional data.

The characteristics mentioned above result from ANN's internal structure, not from clever programming. These characteristics make ANN very useful tools for a variety of problems that conventional computers (and traditional 'step by step instruction' based programs) do poorly, if at all. To this class of problems we can include all areas of pattern recognition, identification and classification.

Object detection and recognition is an important and fundamental problem in computer vision and there have been many attempts to address it. ANN is just one of them.

This thesis is about a complete head tracker system, but in order to track a head a face has to be recognized and detected first. Therefore this thesis implements the object matching of image-based models (a face to be recognized) to video images (camera input) using neural networks (NN), and evaluates this approach in the face recognition, detection and tracking domain.

1.5. The Challenge of Face Detection and Recognition

Object detection and recognition is the problem of determining whether or not a window or part of an image belongs to the set of images of an object (recognition) or object classes (detection) of interest.

Thus, anything that increases the complexity of the decision boundary for the set of images of the object will increase the difficulty of the problem, and possibly increase the number of errors the system will make.

Suppose we want to detect or recognize faces that are tilted in the image plane, in addition to upright faces. Adding tilted faces into the set of images we want to detect or recognize increases the set's variability, and may increase the complexity of the boundary of the set. Such complexity makes the detection problem harder. Besides adding new images to the set of images of the object, there are more sources of variability to be considered in the face detection/recognition problem which will be difficult to include ahead in the set, for example:

- **Pose**. The images of a face vary due to the relative camera-face pose (frontal, 45 degree, profile, upside down), and some facial features such as an eye or the nose may become partially or wholly occluded.
- **Presence or absence of structural components**. Facial features such as beards, mustaches, and glasses may or may not be present and there is a great deal of variability among these components including shape, color, and size.
- **Facial expression**. The appearance of faces are directly affected by a person's facial expression
- **Occlusion**. Faces may be partially occluded by other objects. In an image with a group of people, some faces may partially occlude other faces.
- **Image orientation**. Face images directly vary for different rotations about the camera's optical axis.
- **Imaging conditions**. When the image is formed, factors such as lighting (spectra, source distribution and intensity) and camera characteristics (sensor response, lenses) affect the appearance of a face, for example color skin and shadows.

- **Image Scale and Size**. Face size accounts for the variability of face features and affect also face detection/recognition approaches which involve training by fixed size face databases (described later). Certain face detection/recognition methods are not size invariant therefore to detect/recognize faces in any size a proper preprocessing has to be applied
- **Background, cluttered scenes or complex background**. In realistic application scenarios a face could occur in a complex background and in many different poses and positions. Recognition systems that are based on standard face images are likely to mistake some areas of the background as a face. This has to be considered and in order to rectify this problem, more algorithms and processing have to be applied.

1.6. An Image-Based Approach using Neural Networks

The face recognition, detection and tracker system in this thesis are based on the following steps:

1. **Training**. Use of machine learning approach, specifically an artificial neural network, requires training examples. To reduce the amount of variability in the positive training examples, only the face in all possible poses (frontal, tilted, upright, downright, rotation, etc.) to be recognized, detected and tracked has to be input during the training process. If I have to do it manually with any new user the system has to track, it will defy the real-timeness I want to achieve due to the time it requires to manually extract the face in each pose from the background and input this information during the training process. To avoid this, I devised a methodology and corresponding algorithm to automatically train the NN with any new user's face, in all poses, and in a very short time. This per user training data can also be saved and easily recovered or loaded any time this person wants to use the system again in order to be tracked. To minimize the problem of mistakenly recognizing or detecting an area of the background as a face, the tracker system also automatically trains on the current background. I define the 'current background' as

the scenery view by the stereo-cameras when no people are in front of them and the tracker system is already running.

2. **Preprocessing**. To further reduce variation caused by lighting or camera differences most image-based approaches in detection and/or recognition first extract a window or sub-image area from the whole image and perform some kind of image preprocessing called 'image normalization' before is fed into the recognition/detection system. Standard algorithms, such as histogram equalization to improve the overall brightness and contrast in the images, lighting compensation algorithms that use knowledge of the structure of faces to perform lighting correction, etc., are very common preprocessing techniques [1, 2]. Unfortunately all these approaches impose a certain undesirable load to the CPU since I have to apply them for every window extracted from the video image. Real-time operation is one of the main goals in my thesis, so I have to use a different approach in order to normalize the input image without using too much CPU. Since this head tracker is a fixed part of a VR system, a controlled illumination environment is feasible. Using Infrared (IR) illuminators and IR filters in front of the camera lenses make the system mostly independent of room lighting and solve part of the variance problem. To further reduce this variation (mostly some fall-off in the cameras field-of-view corners), I only apply a global normalization to the whole video image (not each window). In this controlled environment I also compare both approaches (global vs. local normalization) and obtain not very conclusive results from the recognition/detection/tracking performance rate, but the global approach use much less CPU helping me to achieve faster tracking frame rates. And another very important advantage using IR is that I have to deal with only gray level images which means less bandwidth utilization between camera and system (higher camera frame rates), faster processing (one byte per pixel comparing to three bytes per pixel) and no skin color variation dependency (greatly affected by different room source-type lights).

3. **Image Resizing**. During training only the user's face in several poses is input as examples. But the size of the face remains constant during the whole process. This means that, although in all poses, the NN are been trained only with a specific face' size. In my thesis this size is the 'minimum face' size the tracker system is able to recognize, detect and track per user which establish a depth constrain because the farther the face is from the camera, the smaller it becomes. At runtime, I do not know the precise facial feature locations, and so I cannot use them to locate potential face candidates. Instead, I have to first perform an exhaustive search over all locations in the video image, then scale down the image (sub-sample) and repeat the process until I find all possible candidates locations. Currently, this is the technique I am using to recognize, detect, and track faces that are closer to the camera and therefore larger than the face' size the NN is been trained with. Clearly, this is a major bottleneck and CPU load and an improvement over this exhaustive search has to be developed.
4. **Recognition, Detection and Tracking**. After the video image is normalized in lighting (step **2**) and then searched in its original size and several scale-down size (sub-sampled, step **3**), all potential faces are examined to determine whether it is background or they are or not the specific user's face (the once the system is been trained on). Once the user face is recognized and located the system switches to a faster detection and tracking mode and in the next video image frames the face is searched only in a specific image sub-region. This sub-region is continuously determined by a prediction module based on head position of previous frames.
5. **Arbitration**. Since the tracker system uses two cameras (left and right) to determine the 3D head position, two different set of four (Recognition, Detection, Tracking and Background) NN are individually trained and used. Each NN learns different things from the training data, and makes different mistakes. During the recognition process each NN returns a '*confidence* number' which represent the likelihood of the user's face. These numbers have to be above certain *minimum* recognition *confidence* threshold, but at the end the tracker system decides between left and right

NN, whoever has the better *confidence* number. After this decision, the system switches to the left or right NN (whichever had the better *confidence*) to continue detecting and tracking the user's head. This allows the system to increase the tracking frame rate. If during detection and tracking the *confidence* number drop below an specified detection (not recognition) threshold, the tracker system re-initialize itself again and re-start in recognition mode until it finds the correct user again.

6. **3D Head Position**. The stereo cameras are identical and the coordinate systems of both cameras are perfectly aligned, differing only in the location of their origins. Therefore the z or *depth* coordinates axe is aligned (non-verged geometry), and the baseline (distance between cameras' optical center) is aligned to the cameras x coordinate axis. This camera's geometry constraint is called *epipolar constraint* and after determining the user's head position in the left (or right) video image it allows me to use of a standard image-based technique called *block–matching* to find the head in the right (or left) image using the same height position (y coordinate) of the left (or right) head. Then, once we have the left and right head position we can obtain the 3D coordinates through simple triangulation. In general this is referring as solving the *stereo-correspondence* problem.

Together these steps attempt to account for the source of variability described in the previous section and at the same time reach the target frame rate considered real-time for VR applications. These steps are illustrated schematically by **Figure 1.1**.

1.7. Evaluation

This thesis provides a theoretical and practical analysis of the accuracy, resolution in x and y axes for specifics depths (z axis), speed (tracking frame rate, lag and delays) and tracking stability of the algorithm developed. The thesis is been tested using EVL's Varrier™ Autostereo Virtual Environment (**Figure 1.2** and **Figure 1.3**), a high-resolution autostereoscopic display consisting of tiled LCD displays driven by a PC

cluster and fitted with this tracker system to track user's head position without the use of head mounted or hand held tracking devices. During several demos and an open house hosted for the IEEE VR 2004 conference at the Electronic Visualization Laboratory (EVL), University of Illinois at Chicago (UIC), March 29, 2004, more than a hundred persons used the tracker system and were able to view stereoscopic scenes without wearing any special glasses or using any tracking sensor. Each new user who wanted to experience the Varrier™ display was trained in less than two minutes and immediately recognized and tracked by system, even when the room was full of people (cluttered background with faces the system should not recognize and track) looking at the same Varrier™ display the new user was operating.

There is one very important issue in evaluating this thesis. My goal is to replace current technology in 3D head trackers with my system, so the performance has to be evaluated against current tracker systems in terms of accuracy, precision, resolution, tracking range and rate. It is not the main purpose of this work to develop state of the art face recognition and detection systems. And there is yet another problem: comparison with several other neural network based face detection and recognition systems will be difficult to perform because this tracker has to work (recognize, detect and track) with not only upright frontal faces where most of the research is focus, but tilted and rotated (non-frontal) faces. Although there are a few other detectors and recognizers (NN based) designed to handle tilted and non-frontal face, they have not been evaluated on large public datasets and/or they are not suitable for real time tracking (no published data on training time and/or detection/recognition speed, latency, etc.), so again performance comparison will not be possible. Good surveys, including rotational invariant face detectors, can be found in [3-6].

In the simple domain of upright and frontal faces, this current system has lower performance in terms of recognition, detection and false-positive rates compare to the best numbers gathered from the surveys [3-6]. My previous face detector research based on a modified LAMSTAR NN (**Chapter 6**) has comparable performance in this domain, but to reach real-time tracking using a NN approach (and also very

fast training) I have to simplify and re-design the NN architecture, trading-off performance in recognition, detection and false-positive rates vs. performance in tracking.

This work runs on a standard Personal Computer (PC) and only needs a pair of stereo-cameras, making it a very cost-effective tetherless solution comparing to commercial tether head tracker system. It is capable of tracking a person's head at 320 by 240 video image resolution (left and right) at 30 fps. The 30 fps is an actual limitation of the stereo-camera gear and the system had been tested with pre-record video inputs reaching tracking frame rates of 100fps at the same resolution. Overviews of the results are given in **Table 1.1** and **Table 1.2**, whereas **Table 1.3** summarizes the hardware and software platform utilized during this thesis. At the end of C**hapter 5** (section **5.3, Head Tracker Evaluation**) I will explain in more detail each term and how they are obtained.

Table 1.1: Overview of the results from the system described in this thesis as a face recognizer and detector using video-images.

System	Test Set	Tracker Error (wrong face or non-face)	Frame Rate at Image Resolution
Recognizer(*)	30 video images, faces in all poses (1 or 150 faces) (*)	3.3% - 0% (after re-trained) (for only one face) 16% - 6.6% (after re-trained) (more than one face)	9.2 fps @ 320x240
Detector(*)	30 video images, faces in all poses (1 or 150 faces)(*)	Face always detected (100%) but with 3 – 5 *false positives* (for only one face) 4% - 2% (after re-trained) but with 55 – 74 *false positives* (more than one face)	30 fps @ 320x240

(*) See section **5.3** for better explanation and description of terms.

Table 1.2: Overview of the results from the system described in this thesis as a 3D head tracker.

Tracker Performance rate(*)	After properly recognizing a user (84%-93.4%) it is able to track this user at 100% tracking rate (with prediction).
Tracking Frame Rate(*)	30 fps (camera video input)/100 fps (pre-recorded video input) (with prediction)
Recognition Frame Rate(*)	9.2 fps
Video Image Resolution	320 (Width) by 240 (Height)
Training time (new user)	Under 2 minutes
Type of Prediction	Always on. Predict next area of the input image to search for the face/head to speed up the tracker frame rate. It does not predict next head position to lower the latency or lag.
Input (movement) sensor	*Zoran ZR31112PLC* (color) CMOS image sensor
Input Protocol/Interface	IEEE 1394a (FireWire)
Output Protocol/Interface	UDP/IP over 100MBits/sec. Ethernet output port.
Tracking Latency(*)	82.5 ms ± 10 ms during an hour average. Measure between video image acquisition (stereo camera gear) and output of the 3D position data from the NN algorithms (see **Figure 3.1**).
Static Jitter and Drift(*)	± 2 mm in *x*, *y* coordinates and ± 3 mm in *z*.
Dynamic Jitter(*)	From ± 2 mm (best case) to ± 16 mm (worst case) (depends on the training).
Static Precision(*)	± 1 cm across the following volume: ±40 cm in *x*, ±40 cm in *y* and ±20 cm in *z*. Sampled every 10 cm.
Resolution(*)	4 mm ± 2 mm in *x* and *y*, and 3 mm ± 4 mm in *z*, across the following volume: ±40 cm in *x*, ±40 cm in *y* and ±20 cm in *z*. Sampled every 10 cm.

(*) See section **5.3** for better explanation and description of terms.

Table 1.3: Overview of hardware and software used in this thesis.

PC	**Oerating System**: *Microsoft Windows XP Professional*, v2002, Service Pack 1a **µP**: *Intel*® Pentium IV @ 3.4 GHz (Prescott), 1MB L2-Cache, 200 MHz FSB **Memory**: dual DDR400, 2x256 MB (512 MB) @ CL 2.0-tRCD 2-tRP 2-tRAS 6
Stereo Camera Gear	**Brand and Model**: *Videre Design* MEGA-D (STH-MD1-C), 30 cm baseline, Infrared internal block filter removed **Image Sensor**: *Zoran ZR32112*, CMOS, native resolution 1288x1032 (1.3 Megapixels) **Interface**: IEEE 1394a @ 400Mbps **Library and Drivers**: v3.2f
Lenses	**Focal Length**: 16 mm **Infrared Filter**: *Proline B+W 092 (89B) Dark Red 20-40 X*, 650 nm low pass filter
Tools, Compilers and Libraries	*Intel*® *Integrated Performance Primitive* (IPP) library v4 for Windows *Intel*® *C++ Class and Intrinsic Libraries for SIMD Operations* *Intel*® *C++ Compiler* v8 *Intel*® *VTune*™ *Performance Analyzer 7.1* All libraries, classes and tools are optimizing for Pentium 4 and Xeon™ processor.

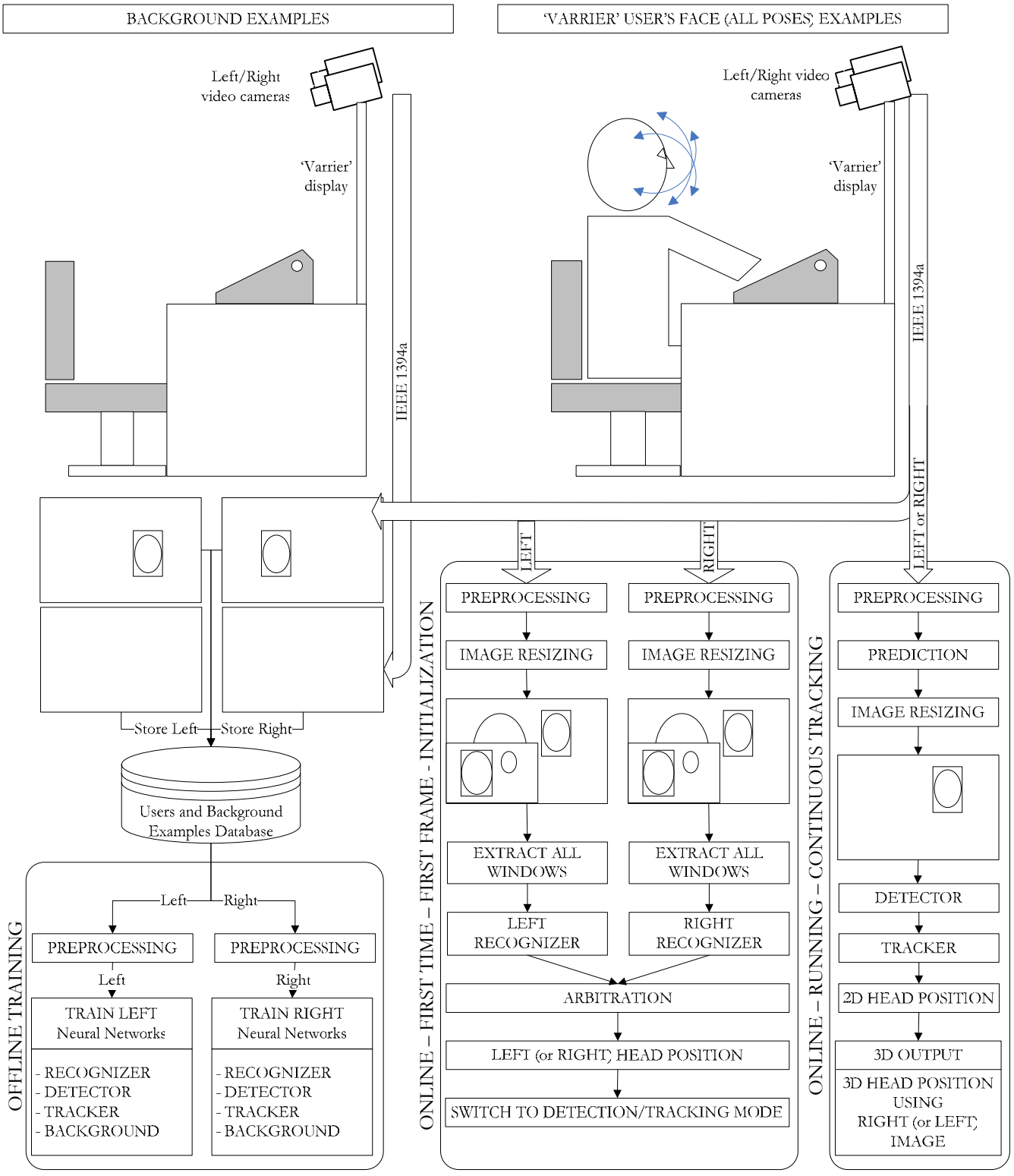

Figure 1.1: Schematic diagram of the main steps of the 3D head tracking developed in this thesis.

Figure 1.2: EVL's Varrier™ Autostereo Virtual Environment turned off (left) and turned on with a user (right).

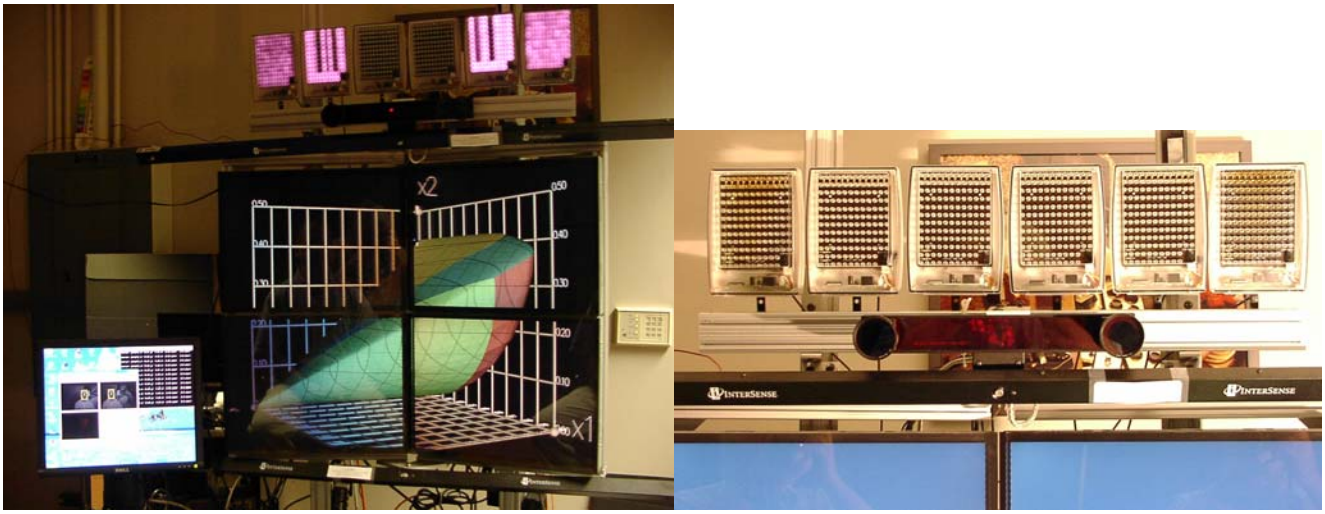

Figure 1.3: EVL's Varrier™ Autostereo Virtual Environment with a 3D scene running (left), and detail of the 3D camera based tracker system (thesis) with its infrared illuminators. Below the stereo camera gear is the InterSense IS-900 Precision Motion Tracker I use for comparison (left and right).

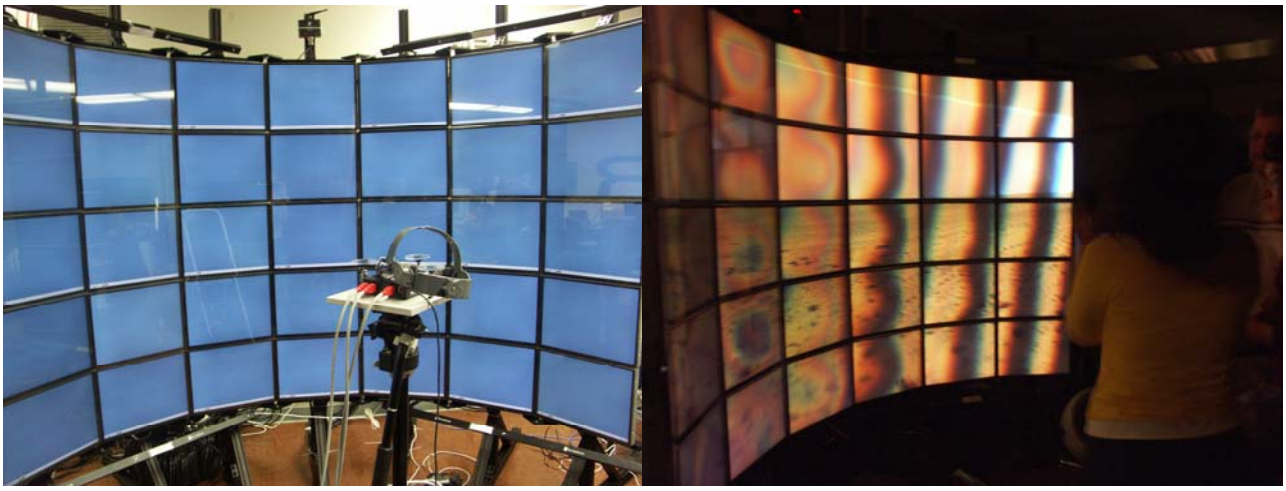

Figure 1.4: Next generation of EVL's Varrier™ Autostereo Virtual Environment. Bigger is Better!

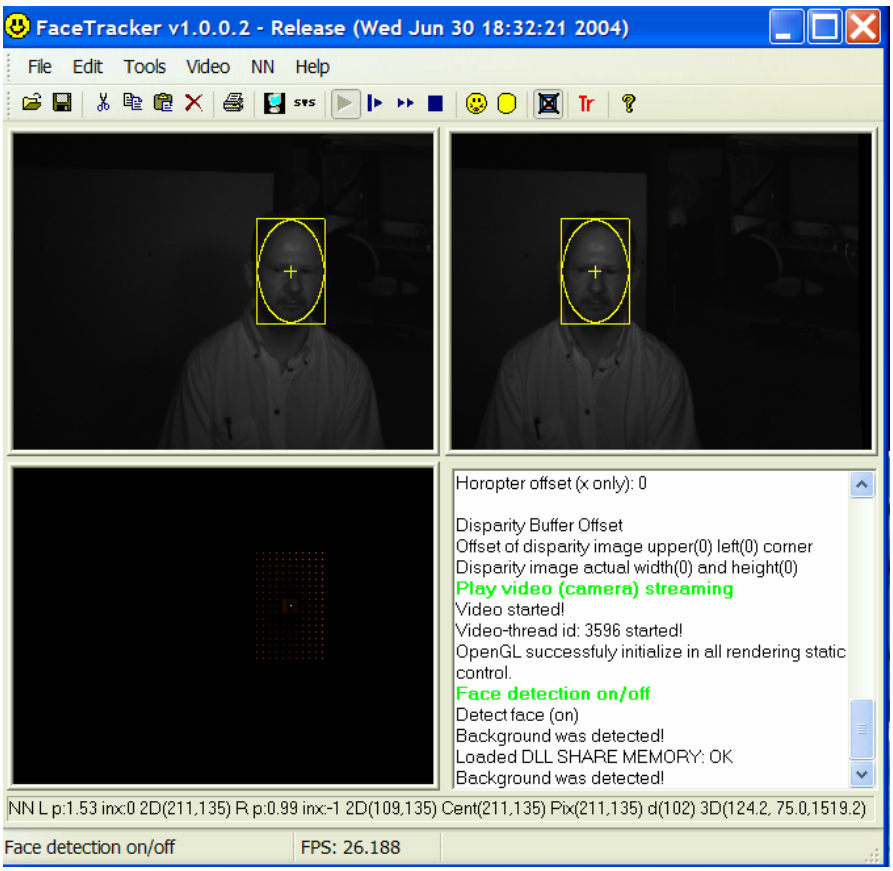

Figure 1.5: Detail of the tracker system Graphic User Interface (GUI), running and tracking.

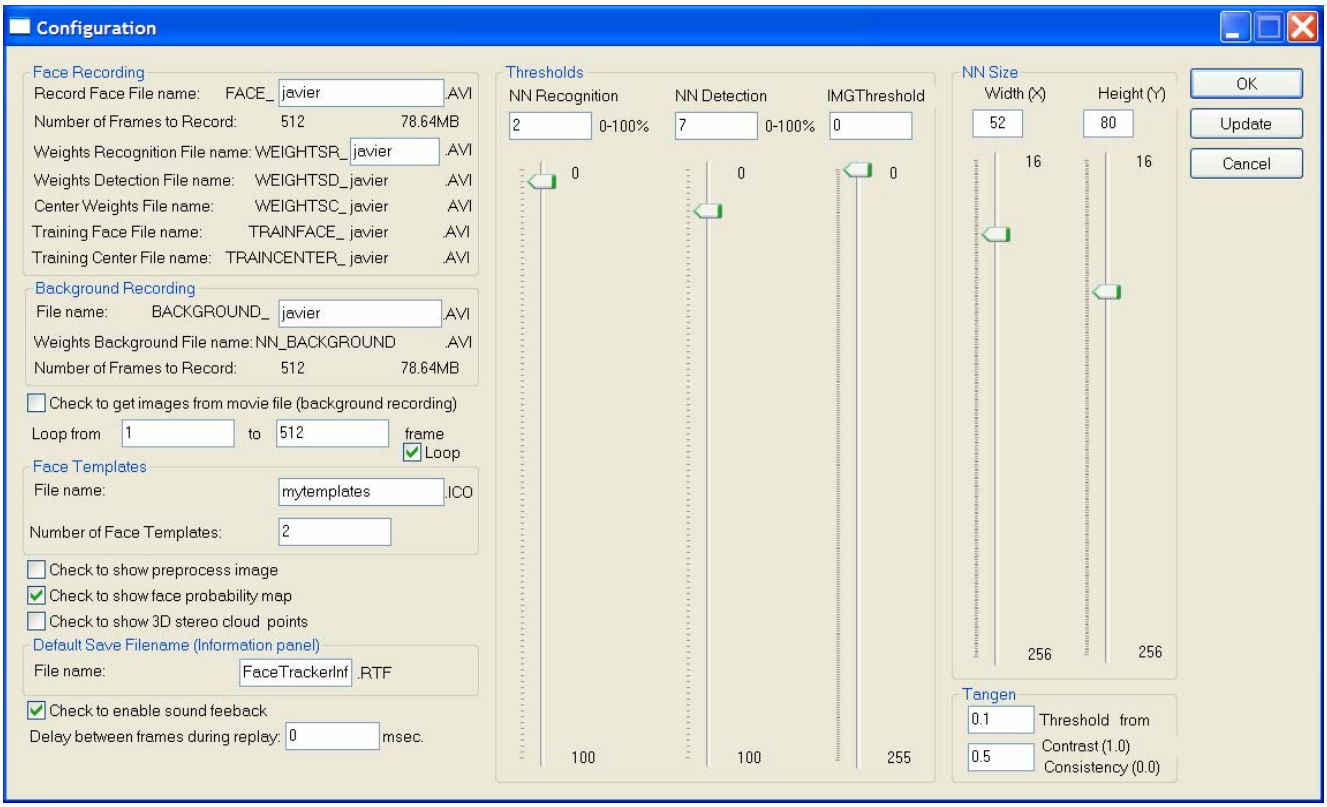

Figure 1.6: Detail of the tracker system Configuration GUI.

2. BACKGROUND

2.1. Introduction

Since the thesis is a complete head-tracker system based on cameras and using neural network algorithms, it would be a good approach to divide the background or previous works in several areas:

- Background in Virtual Reality (VR)
- Background in Commercial VR Tracker Devices

To understand the thesis objectives is important to have a brief introduction on VR and its related devices. Background in current position-tracker devices use in VR, specifically our laboratory, will help to specify the new tracking system in order to compete with these current technologies.

2.2. Background in VR

2.2.1. Brief Introduction of Virtual Reality and its Devices

Virtual reality (VR) is the science of illusion - a computer fabrication of a world. VR may best be defined as the wide-field presentation of computer-generated, multi-sensory information that tracks a user in real time. Well-known modes of virtual reality includes head-mounted displays (HMD) and binocular omni-oriented monitor (BOOM) displays [7]. In addition to these, the Electronic Visualization Laboratory (EVL) at the University of Illinois at Chicago introduced other modes: a room constructed of large screens on which the graphics are projected onto the three walls and the floor (CAVE® [8, 9]), and a drafting table format VR display (I-Desk [10] and PARIS [11]. These devices are called projection-based VR displays because they display 3D images on video projection screens or monitors.

The CAVE®, I-Desk and PARIS display stereo images. They present an image to each eye in order for the VR user to perceive depth information (one of the depth cues from the real word is given by stereo vision or binocular disparity). The computer sequentially generates two images, one for the left and one for

the right eye, and the user wears wireless Liquid Crystal Display (LCD) Shutter Glasses which alternately block and pass each image. An infrared signal - similar to the signal used for TV remotes - synchronizes the glasses to the computer images, so that the right image is shown when the right lens is transparent and the left image is shown when the left lens is transparent. Infrared emitters are placed so that wherever the user looks the glasses are always fully functional.

More information about VR in general and these devices in particular can be found at EVL web site (www.evl.uic.edu) and in [12].

Currently, our laboratory is researching a more exciting VR device called *Varrier*^{TM} [13, 14] whose goal is to develop a head-tracked, stereo virtual reality system utilizing plasma or LCD panels. It is an autostereoscopic display based on barrier-strip technology whose objective is ultimately to provide stereo imagery for VR that can be viewed without requiring the use of special glasses.

2.2.2. The Need and Importance of Tracker Systems in VR

Creating the computer-generated world around the subject (VR environment) means updating all 3D images in all displays and at the same time presenting the right perspective of these images to the VR user. The computer must know not only the exact position of the subject with respect to this world, but also where the subject is looking at (subject's point of view). This is to present him or her with the right perspective of this newly created VR environment. In other words, the position of the eyeballs must be known so that the correct viewer-centered perspective can be calculated.

Therefore, to perform this task the VR systems must have a tracking device (a sensor) that tracks the position in space of the eyeballs. Usually some part of the head is tracked and the position of the eyeballs is inferred from there. To be more specific, a sensor is attached to the stereo-glasses and the position of the eyeballs is inferred from the position of the glasses (**Figure 2.1**). Generally speaking these are called 'head-tracking devices'.

Once the eyes' position is tracked, the VR environment can be update accordingly to present the right perspective to the subject. A second sensor, attached to a wand, tracks the hand's movements allowing the user to interact with the virtual environment, for example, to pick up virtual objects (**Figure 2.1**). Sometimes it is necessary to track the subject's body parts, for example hands, arms, legs, etc., to map its positions and movements and represents them as an 'avatar' [15] (a generated computer graphics representation of a person) into the VR environment. The tracking data is used with avatars in networked applications, so that a collaborator in a Virtual Environments can see each other's position and interpret each other's gestures.

At this point, it becomes evident why tracking systems are increasingly important and a key factor in VR systems.

Figure 2.1: Hand-tracker system attached to a wand and a head-tracker system attached to a stereo-glasses.

2.3. Background in Commercial VR Tracker Devices

2.3.1. Tracker Systems Used in VR Environment

Magnetic trackers (tracker systems which use electromagnetic pulses and electromagnetic sensors) are generally used in VR, especially at EVL. They all have inherent advantages and disadvantages of devices that in general were not designed for the IDesk or CAVE®. They were designed for other applications like

short distance tracking of pilot's head in cockpit, motion capture for video applications, augmented reality, etc. Recently, some other tracking methods like a hybrid acousto-inertial have also been tested and used with better results, but still have some drawbacks. A brief survey of some of the most common commercial tracker systems used in VR environment (**Table 2.1**) will follow in order to extract among them their best features (like resolution, accuracy, stability, sampling rate, etc.) and summarize their advantages and disadvantages.

Table 2.1: Main characteristics of Commercial Tracker System used in VR

Company	Model	Tracking Method	Resolution (static)	Accuracy (static)	Sampling Rate	Interface	Comments
Ascension Technology Corporation	Flock of Birds®	Pulse DC Magnetic field	0.5mm@30.5cm 0.1°@30.5cm	1.8mm RMS 0.5° RMS	Up to 144 samples/sec	RS232@115K	- Each sensor provides 6DOF - Stand-alone system - Insensitive to occlusion - IR wireless option
Ascension Technology Corporation	pcBIRD®	Pulse DC Magnetic field	0.5mm@30.5cm 0.1°@30.5cm	1.8mm RMS 0.5° RMS	Up to 144 samples/sec.	ISA-Bus	- Each sensor provides 6DOF - Needs a PC system - Insensitive to occlusion - IR wireless option
Ascension Technology Corporation	SpacePad®	Pulse DC Magnetic field	N/A	N/A	120/sec. for one sensor 60/sec. for two sensors	ISA-Bus	- Low cost solution - Each sensor provides 6DOF - Needs a PC system - Insensitive to occlusion
Ascension Technology Corporation	MotionStar®	Pulse DC Magnetic field	0.76cm RMS 0.1° RMS at 1.52m 0.25cm RMS 0.2° RMS at 3.05m	0.76cm RMS 0.5° RMS at 1.52m 1.5cm RMS 1.0° RMS at 3.05m	Up to 144 samples/sec.	RS232@115K Ethernet	- Each sensor provides 6DOF - Sand-alone system - Insensitive to occlusion - IR wireless option
InterSense Inc.	IS-900	Acoustic (ultrasonic) and Inertial	1.5mm RMS 0.05° RMS	Stability of 4mm RMS 0.4° RMS	180 samples/sec.	RS232@115K Ethernet	- Each station (NOT sensor) provides 6DOF - Wireless only for position information - Stand-alone system - Sensitive to occlusion
Polhemus	STAR*TRAK®	AC Electromagnetic field	N/A	2.54cm 2.0°	120 samples/sec.	RS232@115K Ethernet	- Each sensor provides 6DOF - Sand-alone system - Insensitive to occlusion - 2.4 GHz RF wireless option
Polhemus	FASTRACK	AC Electromagnetic field	N/A	2.54cm 2.0°	120 samples/sec.	RS232@115K Ethernet	- Each sensor provides 6DOF - Sand-alone system - Insensitive to occlusion - 2.4 GHz RF wireless option

2.3.2.

Disadvantages in Tracker Systems Used in VR environment

Commonly all current commercial trackers used in VR environments have these main disadvantages:

Sensors. **Need to attach sensors on objects to be tracked**. Depending on the type of tracker system (brand, model, method, etc.) those sensors can be bulky and heavy and inconvenient to wear. In today's head-trackers configuration the head-sensor is attached to the frame of the active-stereo glasses or shutter glasses (see **Figure 2.1**). These glasses are heavy and bulky and can support easily the sensor. But new screen technologies, which have been developed for more than a decade, are beginning to show up in the market and they are also reaching the demanding screen sizes currently used in projection-based VR. Stereoscopic displays will allow VR users to wear passive-stereo glasses (polarized glasses) in order to perceive depth information and Autostereoscopic [13, 14, 16] will let VR users experience depth without any special glass (auto-stereo). A comprehensive explanation about these topics can be found at "*SPIE Proceedings of Stereoscopic Displays and Virtual Reality Systems I to VIII*".

With these new screen technologies, in the near future VR users will not find a place in his or her head to easily attach the head-sensor as before. Consequently, the user will be force to wear a special frame to attach it.

Wires. Sensors have to transmit their data to the tracker system. As a result, they need wires to connect them to the tracker receiver or main system (**Figure 2.1**). Some companies provide tracker systems with optional infrared devices to receive those sensor data in order to avoid hanging cables. To perceive depth information in our projection-based VR displays, the VR user must wear shutter glasses which use infra-red (IR) signals to synchronizes them with the computer generated projected images. The IR signals use by the transmitter could interfere with the IR signal from glasses, therefore using it in this environment is not possible. Other tracker systems can provide a radio frequency (RF) transmitter in place of an IR

transmitter in order to send sensors data to the main system, but still the tracker user has to carry it including the batteries which is also a disadvantage.

For pure electromagnetic tracker systems (**most common type in projection-based VR environments**) there are some additional **disadvantages**:

Magnetic materials. There are sensitive to conductive and magnetic materials [17] meaning that it may has to be recalibrated any time the environment changes. These, among other things, may cause the need to recalibrate the tracker system: adding or modifying any metal-made structure close to the VR system; adding or moving close to the sensor any device that can emit electromagnetic signals (monitors, computers, etc.); moving the VR system (in the case of the InmersaDesk2, a portable VR system, this happens any time it is moved or shipped to someplace else); moving or modifying the tracker electromagnetic pulse transmitter (antenna). In reality and based on the laboratory experience, magnetic trackers have to be recalibrated periodically and sometimes without any clear reason.

Calibration. **Need to be re-calibrated more than once, especially when magnetic and conductive materials are moved around**.

2.3.3. Real Specification of Tracker Systems used in VR environment

Calibration. To some extent, calibration can be used to correct tracker distortions but none of the calibration methods are likely to be optimal. Study of the noise and repeatability imply limits on calibration success. As an example, for the *Polhemus Isotrack* when the tracking is greater than 1.27mts from the source, the tracking signal is so noisy that no useful calibration can be expected [18].

In general, while calibration can reduce tracker error significantly, it can only do it to about 10 times the short-term stability (jitter) standard deviation [19]

Accuracy and Precision. In general, the systematic errors introduce biases in the computed range values and thus determine the **accuracy** of the range estimation method, while the random errors limit the **precision** of the method.

Tracker systems **exhibit substantial delay and increased non-linear inaccuracy with distance from the transmitter** [18, 20], meaning that the accuracy of the system decreases markedly as the distance from the sensor to the transmitter increases. Static distortion in the position signal affects the accuracy.

For the *Polhemus Isotrack* electromagnetic 3D tracker the distortion is significant and very noisy at distances greater than 1.5mts and is very sensitive to location [18]. While these results have been found for a particular tracker, they can be generalized to any position tracker in 3D space based on electromagnetic fields. Inside this distance, repeatability implies a limit to within 2.54 cm to 5.08 cm for separation of around 76.2 cm in calibration accuracy.

As a rule of thumb the smaller error a magnetic tracker can get after calibration will be about 10 times the jitter standard deviation [19].

All tracker systems are affected from random errors (they experience jitter); as a result, true precision cannot be determined leaving company specification as the only source of theoretical precision.

Short-term stability or jitter. For the *Flock of Birds*® with the extended range transmitter, transmitter-to-sensor separation of less then 500 mm led to saturation of the sensor inputs so a reading could not be taken in this region. The jitter was generally less than 0.5 mm and 0.05° for separation of 5-10 mts, and then rose to about 2 mm and 0.15° at 15 mts. As with the *FASTRACK*, the jitter appears to be a function of the square of the source-sensor separation because of the falloff of the magnetic field with distance [19].

Latency or Lag. Interactivity is an essential feature of virtual reality systems. Experience has indicated that lag or system end-to-end latency is one of the most important problems limiting the interactivity of virtual reality systems and therefore its quality. Other technological problems, such as tracker inaccuracy and display resolution, do not seem to impact user performance as profoundly as latency [21].

The system end-to-end latency is the time difference between a user input to a system and the display of the system's response to that input. The end-to-end latency is composed of tracker delay, communication delay, application host delay, image generation delay and display system delay [21]. Currently measurement of **system latency** on the *InterSense IS-900* gives an average of 58.5ms[22]. Based on laboratory measurements the Flock of Birds® system latency is worse.

2.3.4. Advantages in Tracker Systems used in VR environment

Despite their lack of accuracy, jitter, etc. and calibration problems, these types of trackers are popular because they are robust, place little constrain on user motion, and in the case of magnetic and inertial trackers, **are insensitive to occlusion** (needs for a clear line-of-sight).

3. PROPOSING NEW TRACKER SYSTEM

3.1. Introduction

It is impossible for the research community to measure and get the real specifications and performance of all available commercial tracker systems and models in the market used in VR. Not all parameters (like *dynamic distortion in position data, error orientation, long term stability* or *drift*, etc.) have been thoroughly researched, measured, and published with a useful conclusion (like how to calibrate it in order to avoid or diminish these problems). Some of these parameters are part of the specification post by the companies that develop and sell these tracker systems, but it has been proved (see section **2.3.3 Calibration** and **Jitter**) that companies' specifications sometimes differ widely from the real ones.

This chapter will specify the new tracker system based on the following four sources:

1. Laboratory experience or 'How good in terms of specifications and features this new system must be in order to replace the old one in this environment'
2. Research publications on this matter.
3. Considering the best spec. number among currently commercial tracker systems.
4. Since this is a camera-based tracker, the specified resolution will depend entirely on the camera characteristics and the VR environment in which the system is going to run.

3.1.1. A word of Tracker Latency (or Lag)

It is the delay between the actual physical movement of the tracked object and the output of its new position/orientation data at the host computer/application. Obviously VR users want the least possible latency because it has a profound effect in the quality of the experience of VR systems (see section **2.3.3 Latency**).

Tracker companies often don't specify their system's latency, and if they do, it is not clear what kind of latencies the companies are specifying. The user has to test the equipment, even with multiple configurations, in order to measure and optimize the tracker delay [22].

Also as shown in **Figure 3.1**, the total or "end-to-end" latency we want to minimize along the pathway is composed from several local ones whose numbers depend on multiple factors not entirely related to the tracker system design. For example for the same type of serial port (RS-232) interface, the connection between the tracker system and type of host (UNIX/SGI or MS-Windows/PC) can have some influence [22] in this latency, and of course the type of interface and their speed or bandwidth, drivers, programs, etc. associate with them can affect it. Most of the research in this subject measure these end-to-end latencies [22, 23] so we can 'guess' the local ones.

To narrow this problem I will only specify (and later measure) the latency of the tracker system (TrL, see **Figure 3.1**) between the video image acquisition (stereo camera gear) and the output of the 3D position data from the NN algorithms. Given the actual technology and speed of current PCs I am assuming that the latency between the NN output and the tracker PC output port is negligible.

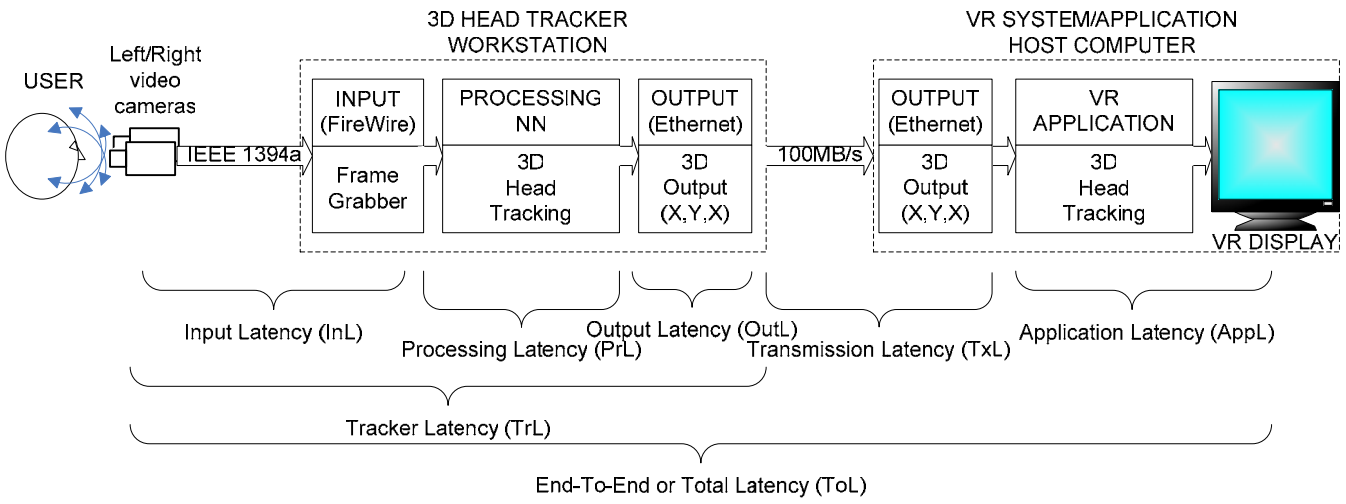

Figure 3.1: Global latency and its local latencies component parts

3.1.2. What is a Real-Time System?

Laplante and Stoyenko [24] stated that "The time between the presentation of a set of inputs and the appearance of all associated outputs is called the *response time*. A real-time system is one that must satisfy explicit bounded response-time constraints or *deadline satisfaction* to avoid failure". For example, in image and neural networks processing involving screen update for viewing and tracking continuous motion, the minimum constraint must be in the order of 33ms (assuming a 30Hz refresh rate display, 1/30Hz=33ms). But, timely deadline satisfaction is not the only constraint that head and gaze trackers have to fulfill to be considered real-time systems; latency or lag constraints, another deadline, should also be a vital part of this definition.

As a conclusion, a real-time tracker system is one whose logical correctness is based on the correctness of the output (accuracy and precision), timeliness (response time must be less than screen update or frame rate) and latency (delays below certain threshold).

3.2. Defining the New Tracker Systems Specifications

With all this in mind, what follows (**Table 3.1** and **Table 3.2**) is a comparison of commercial trackers against the proposed head tracker system. I made tentative specifications and features of the new system in order to try to replace the existing ones. Currently, the laboratory is using the *InterSense IS-900* motion tracker for the Varrier™ display, so I will consider its specs as an important reference.

Table 3.1: Comparison of current specification of some of the best commercial tracker including the ones we are using at our laboratory and the proposed camera-based tracker system.

Very Good **OK** **Worst**	Magnetic ***Flock of Bird***	Inertial Acoustic InterSense ***IS-900 VET(**)***	Camera-based with Markers	Camera-based Commercial ***Seeing Machine***	Camera-based Commercial ***TYZX***	Camera-based Commercial ***ARTtrack1***	Camera-based **Proposed** **Tracking System**
Object Tracking	**Yes**	**Yes**	**Yes**	**Yes**	**Yes**	**Yes**	**Yes**
Face Tracking	**No**	**No**	**No**	**Yes**	**Yes**	**Yes**	**Yes**
3D Head Position/Orientation	**Yes**	**Yes**	**Yes**	**Yes**	**No**	**Yes**	**Yes**
Tetherless	**No**	**No**	**Yes**	**Yes**	**Yes**	**Yes**	**Yes**
Markerless	**Yes**	**Yes**	**No**	**Yes**	**Yes**	**No**	**Yes**
Insensitive to Occlusion	**Yes**	**Yes**	**No**	**No**	**No**	**No**	**No**
Calibration	**More than Once** **Difficult**	**More than Once** **Medium**	**Once** **Medium**	**Once** **Easy**	**Once?** **(No Info)**	**Once** **Easy**	**Once** **Easy**
Operation Range (Min-Max)	**N/A to 10'**	**N/A to 10'**	**Best**	**N/A to 3.2'**	**6' to 30'**	**N/A to 30'**	**3' to 5'**
Accuracy Position	1.8mm RMS (<1m)	N/A	N/A	1mm	N/A	1mm RMS	**5mm (worst)**
Accuracy Orientation	0.5º RMS (<1m)	N/A	N/A	1º	N/A	0.5º	**N/A**
Position Resolution	0.5mm@30.5cm	1.5mm RMS	N/A	N/A	0.2" to 5.6"	0.2mm	**1cm (worst)**
Angular Resolution	0.1º@30.5cm	0.05º RMS	N/A	N/A	N/A	0.12º	**N/A**
Long Term Stability Pos.	N/A	4mm RMS	N/A	N/A	N/A	N/A	**0**
Long Term Stability Ang.	N/A	0.2º(P/R),0.4º(Y) RMS	N/A	N/A	N/A	N/A	**N/A**
Jitter	N/A	N/A	N/A	N/A	N/A	0.03mm	**N/A**
Latency	**60ms**	**10ms**	**N/A**	**30ms**	**N/A**	**< 40ms**	**80ms**
Frame Rate (best)	**144Hz**	**180Hz**	**>30Hz**	**60Hz**	**30Hz**	**60Hz**	**30Hz**
Special Hardware	**Yes**	**Yes**	**Yes**	**No**	**Yes**	**Yes**	**No(*)**
Interface	**Ethernet/Serial**	**Ethernet/Serial**	**Ethernet/Serial**	**Ethernet/Serial**	**Ethernet/Serial**	**Ethernet**	**Ethernet**

(*) We are currently using a special stereo camera gear from *Videre Design* but any pair of cameras that can be syncronized to grab frames at the same time could be use here.

(**) This is motion tracker the laboratory is currently using for the Varrier auto-stereoscopic display.

Table 3.2: I extract the best specs and propose the new tracker based on these.

Specification	BestMeasured or Specified	Propose	Comments
Latency	50ms (tracker only)/60ms (End-to-End)(1) See **Figure 3.1**	80ms (Tracker only) See **Figure 3.1**	(1) Since the paper from He D. [22] measure end-to-end latency, we can obtain the tracker latency subtracting from it the transmission and host application latencies (see **Figure 3.1**). I am estimating this number to be 10ms (very conservative), therefore my thesis has to have a latency of 50ms (60-10=50) if I want it to compete against this tracker (*InterSense IS-900*). **But,** as it will be explained later, due to cameras + frame grabber + input interfaces delays, a more realistic tracker latency of **80ms** is proposed.
Sampling Rate	180 Hz	30 Hz min.	A minimum quality need it to experience VR impose to these systems with at least a screen refresh rate of 30 Hz for each eye (projection-based VR) or 30 Hz for both eyes (autostereoscopic-based VR). This means the tracker has to have a valid output of at least at 30 or 60 fps in order to keep pace with screen updates. Unfortunately five years ago when I began my thesis I only found 30 fps stereo camera gears.
Position's Precision (or Resolution)	1.5 mm RMS(2)	Maximum of **2 mm ± jitter** across tracking range, which will be defined later.	**This number is critical** because is related directly to the cameras resolution, therefore to camera cost. **Important:** as will see next, using video cameras imply that the precision is not constant across the tracking range. (2) Best value taken from *InterSense IS-900*. Flock of Birds from Ascension Technology claimed more precision but at a closer distance and based on research publications, the error increases as the distance from the source increases.
Jitter (short-term stability)	0.5 mm(3)	2 mm	**Jitter** depends mainly on camera Signal to Noise ratio (S/N). In general jittering is produced by camera noise (especially on low light conditions), internal electronics (if an analog camera is used, the digital-to-analog converter would be critical) and other factors. The propose system can reduce it to certain degree using proper algorithms, pre-filtering (like applying convolution), quantization, sub-sampling, etc. but at the end would be a **trade-off with the system precision**. Better cameras have better S/N but they are more expensive. As we will se later, NN based tracker add a certain amount of jittering depending on the training. (3) Best value taken from [19] using the Flock of Birds tracker system.
Accuracy	2.54 cm to 5.08 cm(4)	1 cm	The accuracy will depend mostly on the camera calibration. (4) Best value taken from S. Bryson [18] using different a magnetic tracker (*Isotrack*).
Drift (long-term stability)	N/A	0	After cameras calibration and if they stay in the same position, orientation, illumination and room conditions they should not experienced any drift.

3.3. Specifying the Hardware and the Environment

Introduction. During the development of this thesis those previously proposed specifications have to be valid in VR environments where this new tracker system will be intended to be used. The laboratory developed and currently uses several VR devices (CAVE, ImmersaDesk, Varrier™, etc.). Since this thesis is stereo camera-based, several tracker specs like depth range, precision or resolution and accuracy, etc. will be entirely correlated not only with the camera specifications (image sensor resolution, cameras' baseline, cameras' combine field-of-view, focal length, lenses, etc.) but also with the location of the stereo camera gear with respect to the user in this VR environment (see **Figure 3.4** and **Figure 3.5**).

History. The ideal logical steps would be to define the VR environment in which the new tracker would run, determine the space-volume the VR user would move, and from there specify the cameras which should maintain the previously specified tracker resolution across this user-volume.

Unfortunately, this is not what happened. Five years ago I was not really sure in which VR device my thesis will be used, but I needed to begin researching and developing the NN algorithms to test the viability of my ideas. What I want was a test bed. So I assumed the tracker system will be running in the ImmersaDesk (**Figure 3.2** and **Figure 3.3**), determined the volume in which the user has to be tracked (**Figure 3.4**) and calculate the required stereo camera gear specs to meet the tracker specifications. Then, purchase it.

Using this gear I spent the following years developing and testing a real-time NN based face detector concluding than in fact ANN can be use successfully use for this purpose (**Chapter 6** and my SPIE paper [25]). But, by the time I finished this stage of my thesis a new more exciting VR device was developed at EVL, the Varrier™ autostereoscopic display (**Figure 1.2** and **Figure 1.3**), and of course it was the logical choice to use it as a final test bed. Since the whole VR environment changed and the stereo camera gear was now in a different location, to cover the user space-volume I had to recalculate the new camera lenses

specification. This gave me a new head-tracker resolution specs (**Figure 3.5**). In **Chapter 5** evaluated the head tracker system to obtain real resolution and compared against the theoretical ones.

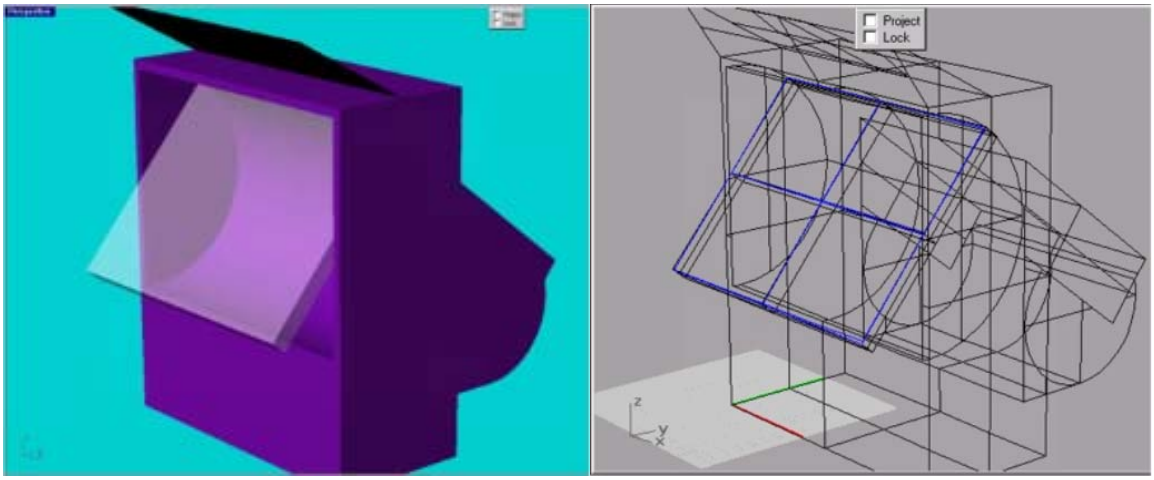

Figure 3.2: Different ImmersaDesk's view

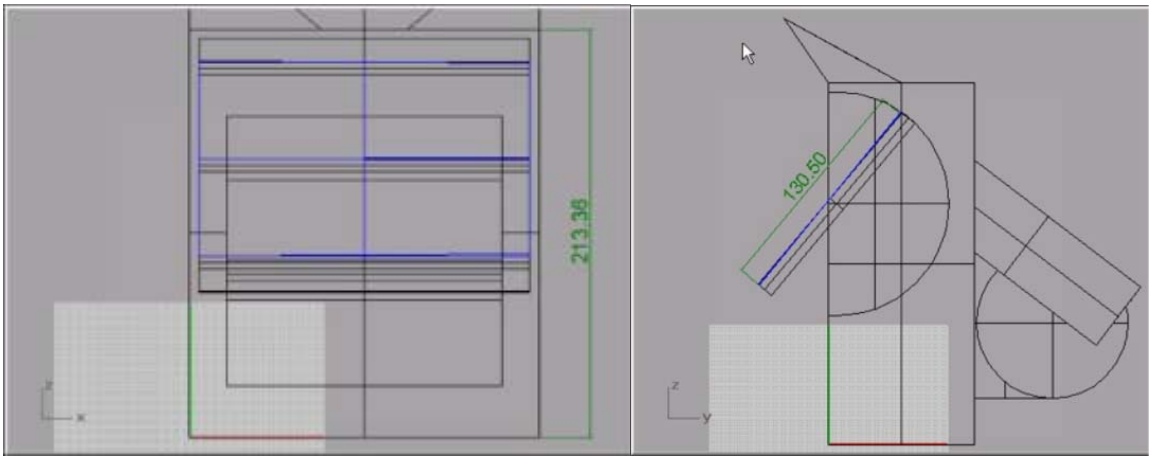

Figure 3.3: Different ImmersaDesk's views

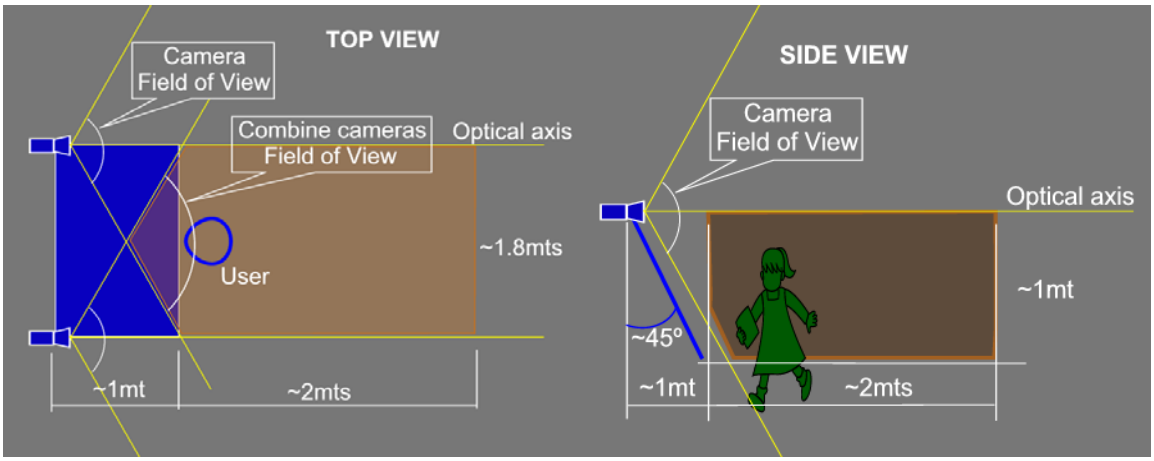

Figure 3.4: ImmersaDesk top and side measures and proposed camera setting and user-volume.

Two cameras vs. one camera. Research into monocular 3D head position tracking reveals that while it is theoretically possible to make 3D measurements of head-pose from a single viewpoint, the practical issues of finite image resolution and processing power place tight restrictions on the precision and accuracy of potential techniques [26, 27]. In using two cameras or view-points I could concentrate more into the NN algorithms for monocular face recognition, detection, and tracking, and at the end use standard computer vision proven techniques like image stereo correspondence to extract the 3D head position.

Commercial vs. custom made stereo gear. If I decided to use two separate equal cameras with external synchronization (to grab video frames at the same time) this would give me the greatest varieties of models and brands (camera specifications) to choose from and have more *confidence* in reaching the proposed head-tracker specs. But, then I would have to assemble myself the stereo gear and also perform the ext. sync by hardware of software, and beside, all I wanted was a hardware test bed to test the NN algorithms. Using my assumption that this system will be use with the ImmersaDesk, I found a company who sells a stereo camera gear that meets the necessary specs to fulfill the proposed tracker specifications.

Latency. On last word related to the cameras. Before acquiring a stereo camera gear or any camera, there is no way I can predict the latency or lag that its internal hardware (image sensor read-out, buffers,

etc.) and interface will introduce. I have to test it. The actual latency will be measure in the **Chapter 5**. Certain camera interfaces can be faster than others. For example, they can have buffered image sensors (more latency) or un-buffered image sensors (less latency). During this thesis I was and am more concerned about other specs like image resolution, frame rates, field-of-view, signal-to-noise ratio, etc.; aspects more related to the camera imagery than its lag. Once I get these numbers right, meaning, I achieve the required new tracker precision, accuracy, jitter, sampling rate and drift, I am almost sure the next generation of stereo camera gears will have faster frame rates at the same video image resolution, faster interfaces, lower noise and lower latencies.

Platform (Operating System and Hardware). I spent quite a while researching which platform would be the best for implementing this thesis. Most of my NN processing algorithms are based on vector operations (vector product, sum or subtraction of vectors, etc.). Since real-time tracking is one the main goals, I wanted to decide which hardware platform would be best suited for this task, not only from their vector processing capabilities point of view but also by their predicted processing power in the next five years in which I was hoping to end my thesis. This will allows me to begin implementing the tracker not worrying at the beginning about the speed performance.

This is not the place to explain the pros and cons of each reviewed platform so I am only going to mention the most important ones I researched and examined (but not limited too):

- General purpose microprocessor (μP) such as *Intel Pentium* and its Streaming SIMD Extensions (SSE) instruction set and *Apple Power Mac G4* and its *Velocity* vector processing engine (*Motorola's AltiVec* instruction set).
- Embedded vision *Intel Pentium*-based processor systems from *Coreco Imaging*, *Matrox* and *Pentek*.
- General Digital Signal Processor (DSP) like *Texas Instruments TMS320C62x* and *TMS320C67x* and single-chip multiprocessor *Texas Instruments TMS320C8x*.

- Specialized DSP such as *A436TM Parallel Video DSP Chip* from *Oxford Micro Devices, Inc.*, *TriMedia TM1300* from *Philips*, *MAP-CA* from *Equator Technology, Inc.*, *VT-5162 Vector Processor* from *Valley Technologies, Inc.*,
- Pipeline processing technology from *Datacube*.
- Zero Instruction Set Computing or *ZISC* technology. Neural Network on Silicon originally developed by *IBM*.
- Field Programmable Gate-Array (FPGA) based image-processing system like the Real-Time Stereo Vision on the *Programmable and Re-configurable Tool Set* (PARTS) [28] and FPGA based boards using FPGA from *Xilinx Corporation*.

Although general and specialized DSPs, special hardware and embedded vision systems were more appealing for this project than general purpose μP, their predicted speed (instruction per seconds) and memory bandwidth in years to come were not. So due to this and other factors like learning curve, presence in the market (for example, the *TMS320C8x* became obsolete), support, etc., I decided to implement my thesis using an *Intel* platform.

3.4. Theoretical tracker specificaions using Varrier™ autostereoscopic display as a VR environment

As it was stated before, the Varrier™ display is the current thesis' test bed, so what follows is the theoretical calculation of some of the new tracker specifications, area of racking coverage, etc., based on the chosen stereo camera gear specs and the volume range the VR user can move with respect to the cameras.

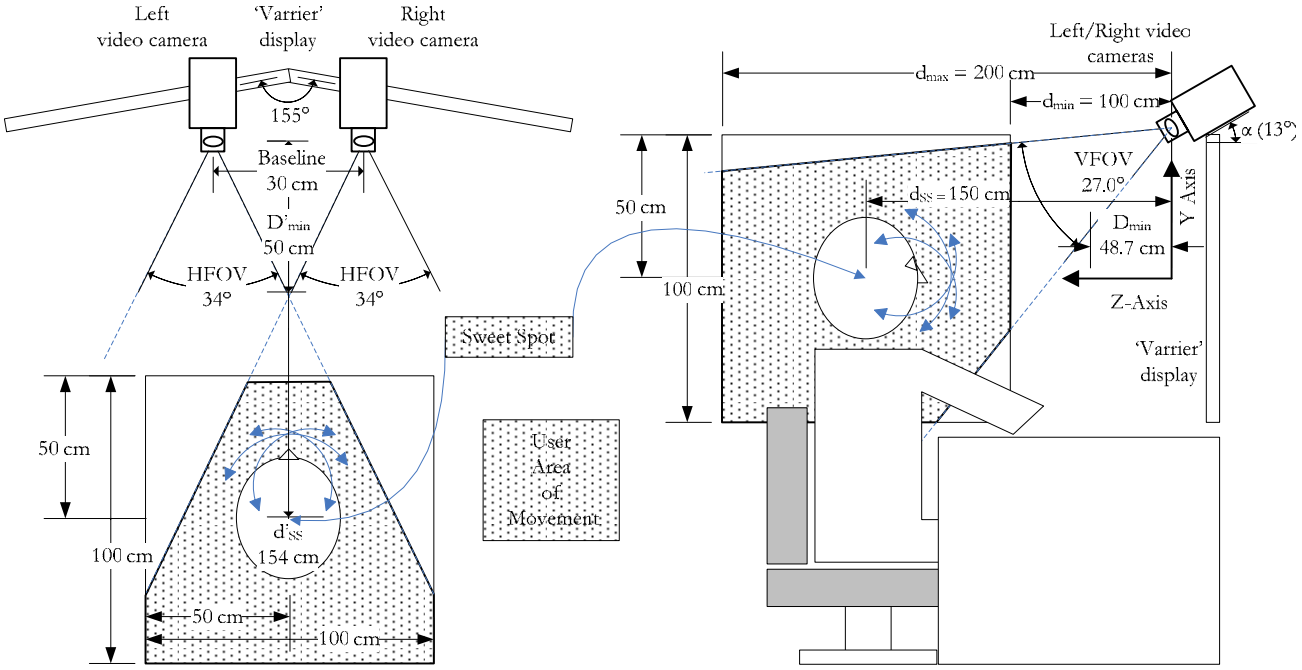

Figure 3.5: Schematic of top and side view of Varrier™ display with the new camera-based tracker system.

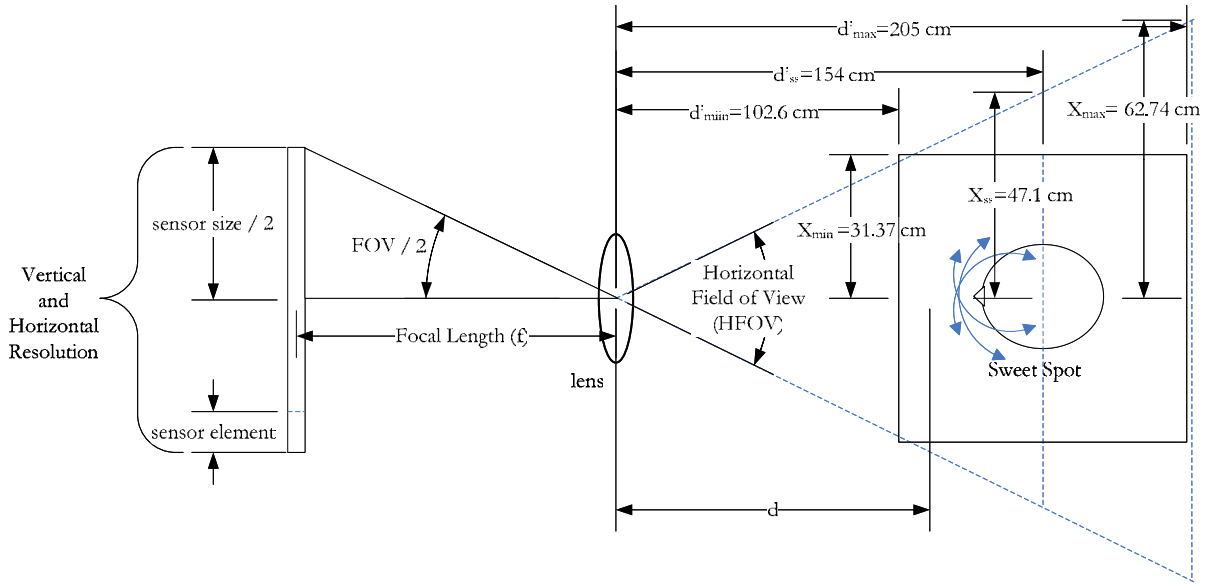

Figure 3.6: Field-of-view and resolution calculations. Top view.

From **Figure 3.6**, *ZR32112 Zoran* camera image sensor specifications, current camera lenses and using trigonometry we can obtain the Horizontal Field-Of-View (***HFOV***) and Vertical Field-Of-View (***VFOV***) using the following formulas:

HFOV = 2 • *Arctg*(((Sensor Element Size • Sensor Horizontal Resolution) / 2) / Focal Length)

HFOV = 2 • *Arctg*((0.0075 mm • 1288) / 2) / 16 mm) ≈ 34º

VFOV = 2 • *Arctg*(((Sensor Element Size • Sensor Vertical Resolution) / 2) / Focal Length)

VFOV = 2 • *Arctg*((0.0075 mm • 1032) / 2) / 16 mm) ≈ 27º

The lens' focal length ***f*** of 16 mm was chosen so the combined field-of-view originates far from the predetermined 'sweet spot' of 150 cm (see side view Sweet Spot ***SS***, ***SS'***, $\boldsymbol{D_{min}}$ and $\boldsymbol{D'_{min}}$ in **Figure 3.5**). This allows the user to lean over and still be in the cameras combined view. Remember that using stereopsis (3D or binocular vision) to extract depth information requires that the object must be seen by the two cameras at the same time. Using the baseline (distance between the cameras) of the stereo gear I already bought, it tilted down angle of ***α*** (see side view in **Figure 3.5** and top view in **Figure 3.6**) and trigonometry we can obtain:

D'_{min} = (Baseline / 2) / *tg*(HFOV / 2) = (30 / 2 cm) / *tg*(34º / 2) ≈ 50 cm

D_{min} = (Baseline / 2) / *tg*(HFOV / 2) • *cos*(α) = (30 / 2 cm) / *tg*(34º / 2) • *cos*(13º) ≈ 48.7 cm

$d'_{min} = d_{min}$ / *cos*(α) =100 cm / *cos*(13º) = 102.6 cm

$d'_{ss} = d_{ss}$ / *cos*(α) =150 cm / *cos*(13º) ≈ 154 cm

$d'_{max} = d_{max}$ / *cos*(α) =200 cm / *cos*(13º) = 205.2 cm

X_{min} = *tg*(HFOV / 2) • d'_{min} = *tg*(34º / 2) • 102.6 cm = 31.37 cm

X_{ss} = *tg*(HFOV / 2) • d'_{ss} = *tg*(34º / 2) • 154.0 cm = 47.1 cm

X_{max} = *tg*(HFOV / 2) • d'_{max} = *tg*(34º / 2) • 205.2 cm = 62.74 cm

Y_{min} = *tg*(VFOV / 2) • d'_{min} • *cos*(α) = *tg*(27º / 2) • 102.6 cm • *cos*(13º) = 24 cm

Y_{ss} = *tg*(VFOV / 2) • d'_{ss} • *cos*(α) = *tg*(27º / 2) • 154.0 cm • *cos*(13º) = 36 cm

Y_{max} = *tg*(VFOV / 2) • d'_{max} • *cos*(α) = *tg*(27º / 2) • 205.2 cm • *cos*(13º) = 48 cm

As was stated before in section **1.6** and **Figure 1.1** the system after recognizing a face uses only one camera to perform the head tracking. Once it obtain the 2D (***x*** and ***y***) face position, it look for the face in the other camera using the same ***y*** coordinates (same *epipolar* line) to perform the stereo-correspondence and

extract the 3D head position. Having this in mind and using similar triangles (**Figure 3.6**) we define the horizontal resolution Δx at a certain distance d' as the ratio between $X(d')$ and actual number of horizontal pixels (current image sensor width resolution):

$\Delta x = X(d') / (\text{Num. of Horizontal pixels})$

$X(d') = (\text{Sensor Element Size} \bullet \text{Sensor Horizontal Resolution} \bullet d') / \text{Focal Length}$

$X(d') = (0.0075 \text{ mm} \bullet 1288 \bullet d') / 16 \text{ mm} = 0.60375 \bullet d'$

Although the *ZR32112 Zoran* camera image sensor can reach 1288 by 1032 (width x height) pixel resolution, a limitation of the stereo camera gear is that it has to run at 320 x 240 pixel resolution in order to achieve 30 fps, possibly due to a limitation of it IEEE 1394a interface.

Since the movement area, center at d'_{ss} = 154 cm, determine that the user can move from d'_{min} = 102.6 cm to d'_{max} = 205.2 cm, we have the following *horizontal resolutions*:

$\Delta x_{min} = 0.60375 \bullet d'_{min} / 320 = 0.00188671875 \bullet 102.6 \text{ cm} \approx \mathbf{0.2\ cm}$

$\Delta x_{ss} = 0.60375 \bullet d'_{ss} / 320 = 0.00188671875 \bullet 154 \text{ cm} \approx \mathbf{0.3\ cm}$ (*Sweet Spot horizontal resolution*)

$\Delta x_{max} = 0.60375 \bullet d'_{max} / 320 = 0.00188671875 \bullet 205.2 \text{ cm} \approx \mathbf{0.4\ cm}$

Analogue, we can use the same formulas to obtain the *vertical resolution* Δy but since the stereo camera gear is tilted down α degrees with respect of the currently used Varrier™ display coordinate system (see **Figure 3.5**), all formulas are affected by a $cos(\alpha)$ factor:

$\Delta y = (Y(d') \bullet cos(\alpha)) / (\text{Num. of Vertical pixels})$

$Y(d') = (\text{Sensor Element Size} \bullet \text{Sensor Vertical Resolution} \bullet d') / \text{Focal Length}$

$Y(d') = (0.0075 \text{ mm} \bullet 1032 \bullet d') / 16 \text{ mm} = 0.48375 \bullet d'$

Since $d' = d / cos(\alpha) \Rightarrow d' \bullet cos(\alpha) = d$, so substituting

$\Delta y_{min} = 0.48375 \bullet d_{min} / 240 = 0.002015650 \bullet 100 \text{ cm} \approx \mathbf{0.2\ cm}$

$\Delta \mathbf{y_{ss}} = 0.48375 \bullet d_{ss} / 240 = 0.002015650 \bullet 150 \text{ cm} \approx \mathbf{0.3\ cm}$ (*Sweet Spot vertical resolution*)

$\Delta \mathbf{y_{max}} = 0.48375 \bullet d_{max} / 240 = 0.002015650 \bullet 200 \text{ cm} \approx \mathbf{0.4\ cm}$

4. DATA PREPARATION

4.1. Introduction

This thesis utilizes an image-based approach to face recognition, detection and tracking, and a statistical model (an artificial neural network or ANN) to represent each face or a face in each pose. An image-based face recognizer and detector determine whether or not a given sub-window of an image belongs to a set of images of faces (detection) or a set of poses of one particular face (recognition). The variability in the images of the faces due to, for example, variation of the illumination, camera characteristics, etc., may increase the complexity of the decision boundary to distinguish faces from non-faces (detection) or a particular face from other faces and non-faces (recognition). The following sections present techniques to reduce the amount of variability in face images. We first explain current *local* techniques to deal with this problem including their advantage and disadvantages. Then, we propose to run this Tracker System in a controlled environment, explain the steps to do this and depart from current techniques by proposing a novel *global* preprocessing which provides similar benefits but allows the system to achieve faster frame rates.

4.2. Preprocessing for brightness and contrast

Apart from the intrinsic differences among face poses and between faces (which will be taken care of during the training process in the next chapter) there are other major sources of variation: camera characteristics and lighting conditions, which can result in brightly or poorly lit images, and/or images with poor contrast.

4.2.1. Standard approaches

Here is where I depart from traditional image preprocessing methods applied for neural network face detection and recognition [1, 29-32]. During training and runtime these well established preprocessing techniques attempt to equalize intensity values (to compensate for any brightness differences over the face)

across the ellipse inside the sub-window where the tracker system is looking for a face (see sub-window and inside ellipse in **Figure 1.5**). The motivation is to have robustness to variations in the lighting conditions. However, there are limits to what 'dumb' corrections can accomplish with no knowledge of the structure of the faces or light sources. And no matter how intelligently we want to correct for lighting variations, it is never perfect [1]. In some cases while the overall intensity can be roughly normalized, the brightness across the face is not improved due to bright spots introduced into the image, sometimes probably because of specular reflections. And in other cases since some lighting models do not incorporate shadows, the shadows cast by the nose or brow will cause problems.

To explain graphically how this 'per sub-window' equalization methods works, I implemented a simple approach which was also been used in [31, 32]. I fit a function which varies linearly across the sub-window to the intensity values in an elliptical region inside the sub-window (see **Figure 4.1**). Pixels outside the ellipse represent the background, so these intensity values are ignored in computing the lighting variation across the face. If the intensity of a pixel ***x***, ***y*** is ***I(x, y)***, then I want to fit this linear model parameterized by ***a***, ***b***, ***c*** to the image:

$$[x\ y\ 1] \bullet [a\ b\ c]^T = I(x, y)$$

The choice of this particular model is somewhat arbitrary. It is useful to be able to represent brightness differences across the image, so a non-constant model is useful. The variation is limited to linear function to keep the number of parameters low and allow them to be fit quickly. Collecting together the contributions for all pixels in the elliptical sub-window gives an over-constrained matrix equation, which is solved by the pseudo-inverse method. This linear function will approximate the overall brightness of each part of the sub-window and can be subtracted from it to compensate for the variety of lighting conditions (see **Figure 4.1**).

Next, histogram equalization (see **Chapter 4** in [33]) is performed, which non-linearly maps the intensity values to expand the range of intensities in the sub-window. Again, the histogram is computed for pixels inside the ellipse (**Figure 1.5**). This will try to compensate for differences in camera inputs gains, as well as will improve contrast in some cases (**Figure 4.1**). I first compute the intensity histogram of the pixels inside the ellipse, where each intensity level is given its own bin. This histogram is then converted to a cumulative histogram [33], in which the values at each bin says how many pixels have intensities less than or equal to the intensity of the bin. The goal is to produce a flat histogram, that is, an image in which each pixel's intensity occurs an equal number of times. The cumulative histogram of such an image will have that property that the number of pixels with intensity less or equal to a given intensity is proportional to that intensity. In practice, **it is impossible to get a perfectly flat histogram** (for example, the input image might have a constant intensity), so the results is only an approximately flat intensity histogram.

Another common approach in image preprocessing is to apply the histogram equalization to the whole image, hoping that it will reduce the variability somewhat and without the background pixels having too much effect on the appearance of the face in the foreground.

Before I decided to tackle the problem in a different way (see next section) to obtain the best possible normalized image or sub-window, I tried to apply locally (to a sub-window) and globally (to the whole image) several different kinds of histogram equalizations. Among these were Histogram Normalization, Cumulative Histogram, Contrast Limited Adaptive Histogram Equalization by Karel Zuiderveld [34], sometimes even including automatic thresholding for discarding pixels with a high probability of belonging to the background and maintaining the foreground (face) pixels.

The results were mixed. In my attempts to equalize intensity values shiny bright spots would sometimes appear in the image (caused may be by specular reflection, another light source pointing to the camera, etc.) and disrupt the process of proper equalization. The resulting post-processed image, especially

the objects of interest (faces), end up with too much contrast or too 'flat' (no contrast) lowering the recognition, detection and tracker performance's rate.

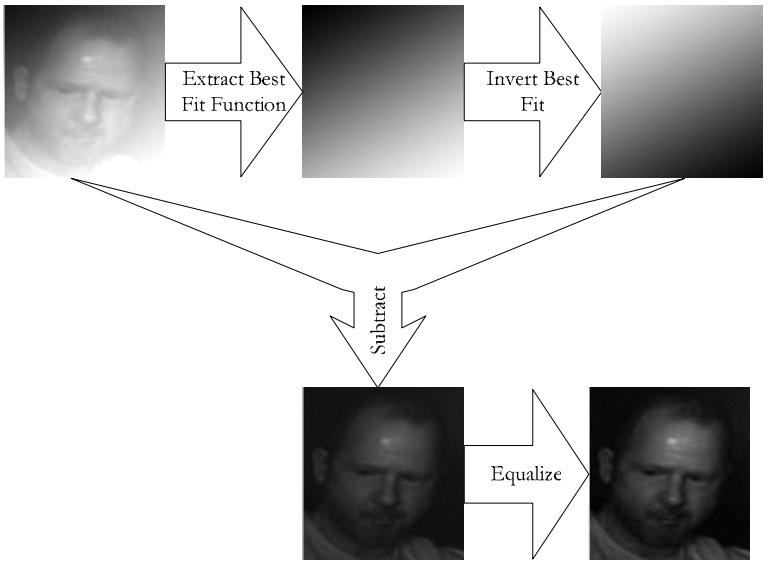

Figure 4.1: Extract the best fit illumination function from the sub-window, invert it and subtract it from the sub-window. Then, perform histogram equalization.

4.2.2. Thesis approach: Global Equalization in a Controlled Environment

Considering one of my particular thesis goals, real-time tracking, there is another point against these well established preprocessing methods: locally or 'per sub-window' intensity and histogram equalization takes valuable CPU processing time since the system has to repeat this process at each pixel position in which it has to check for a face.

Global or whole image intensity and histogram equalization preprocessing is faster because it is performed only one time, but usually achieves worse results than local preprocessing. This is because including all background pixels (non-face or faces the system is not analyzing) sometimes has too large an effect on foreground pixels (sub-window in which the system is looking for a face).

An ideal situation, at least from the room illumination point of view, would be to have a constant ambient light source, that is, light that exists everywhere without a particular source. Ambient light is a non directional light, having equal intensity everywhere, and does not produce highlights or shadows (it fills in the shadowed areas of a scene). Of course it would be impossible to obtain this 'ideal illumination' in real life.

With the following assumptions most of these preprocessing problems can be minimized:

Stereo camera gear. Suppose we already have an ideal ambient light. Since the same set of cameras (brand and model) is already fixed in position relative to the user, once their aperture and focusing controls are adjusted for the current environment, the image contrast, brightness and focus the tracker system attain from the stereo gear should be the same during its training as for during the tracking. The aperture regulates the amount of light that enters the camera and the focus maximizes the clarity or distinctness of the video image.

In order to avoid any variation in certain camera parameters, the camera automatic gain control and gamma correction are disabled, and their shutter speed is adjusted for the current illumination. Once the cameras are set, they should remain constant during training, tracking and for the current environment.

But even so, camera factors can not be ruled out. Some camera problems such as lens 'vignetting' or 'illumination falloff' (gradual darkening of the image towards the corners), distortion, image sensor pixel to pixel gain and offset variance, contribute to the difference in video images between training and tracking. This is because during training the user sits in a fixed position and moves his or her head (to train the tracker system), but during tracking the user can be in any part of the camera's field-of-view.

Infrared (IR) light illuminators. Several factors related to current room illumination can drastically affect the system performance in recognition, detection, and tracking. Additional factors are

significant changes in room lights between the neural network training and subsequence head tracking, room lights casting strong shadows across the scene (in the camera's field of view) including across any face, direct lights might cause specular reflections on faces, etc. To avoid having to deal with all these factors, my co-advisor (Daniel Sandin) suggested that I use IR illumination which proved to be a very clever way to be independent of room's lighting.

As a consequence of this idea, I first removed from the stereo camera gear the internal filters which are use to block IR light (most image sensors, including fortunately the *Zoran ZR32112PLC*, are very sensitive to IR). Second, I added IR low-pass band filters (650 nm) to each camera lens in order to sense only IR wavelengths. And last, I bought several IR light emitters (800 nm) and installed them on top of the stereo gear (**Figure 1.3**).

Using several IR multi-emitters pointing to slightly different locations to disperse their lights the resulting image still presents some illumination falloff (darkening of the image corners) but at least this is constant during training and tracking sessions.

4.2.3. Thesis Approach: Global Preprocessing Using Shading Correction

Virtually all imaging systems produce shading. By this we mean that if the physical input image is constant (let us say we cover the camera field-of-view with a white material), then the digital version of the image will not be constant. The source of the shading might be outside the camera, such as in non-uniform scene illumination, dirt and dust on glass (lens) surfaces, lens anomalies and distortion (optical vignetting, etc.), or the result of the camera itself where the CMOS or CCD image sensor (and internal electronics) *gain* and *offset* might vary from pixel to pixel. With this assumption the model for shading would be:

$$c[x, y] = gain[x, y] \bullet a[x, y] + offset[x, y]$$

where $a[x, y]$ is the digital image that would have been recorded if there were no shading in the object a, that is $a[x, y] = constant$, and $c[x, y]$ is the resulting image due to the camera shading (*Total shading*).

If we consider the illumination $I[x, y]$, it usually interacts in a multiplicative with the object $a[x, y]$ to produce the image $b[x, y]$:

$$b[x, y] = I[x, y] \bullet a[x, y]$$

with the object a representing various imaging modalities such as:

$$a[x, y] = \begin{cases} r[x, y] & \text{reflectance model} \\ 10^{-OD[x, y]} & \text{absorption model} \\ c[x, y] & \text{fluorescence model} \end{cases}$$

where at position $[x, y]$, $r[x, y]$ is the *reflectance*, $OD[x, y]$ is the *optical density*, and $c[x, y]$ is the concentration of fluorescent material. Parenthetically, we note that the fluorescence model only holds low concentration. The camera may then contribute for *gain* and *offset* terms, so that *Total shading* $c[x, y]$ is:

$$c[x, y] = gain[x, y] \bullet I[x, y] \bullet a[x, y] + offset[x, y]$$

In general we assume that $I[x, y]$ is slowly varying compared to $a[x, y]$.

Estimate of shading. We distinguish between two cases for the determination of $a[x, y]$ starting from $c[x,y]$. In both cases we intend to estimate the shading terms $\{gain[x, y] \bullet I[x, y]\}$ and $\{offset[x, y]\}$. While in the first case called '*a posteriori estimate*' we can assume that we have only the recorded image $c[x, y]$ with which to work, in the thesis (second) case we assume that we can record two, additional, calibration images. This second method is called '*a priori estimate*' and is the preferred method for shading estimation.

A priori estimate. Given the nature of this tracker it is possible to record test (calibration) images through the camera system. And unless someone tampers with the stereo camera gear or IR illuminators, we need to do this only one time.

The most appropriate techniques for the removal of shading effects is to record two images, $BLACK[x, y]$ and $WHITE[x, y]$. The *BLACK* image is generating by covering the lens leading to $b[x, y] = 0$ which in turn leads to:

$$BLACK[x, y] = offset[x, y]$$

The *WHITE* image is generated by using $a[x, y] = 1$ which gives:

$$WHITE[x, y] = gain[x, y] \bullet I[x, y] + offset[x, y]$$

The correction then becomes:

$$\hat{a}[x, y] = constant \bullet (c[x, y] - BLACK[x, y]) / (WHITE[x, y] - BLACK[x, y])$$

The *constant* term is chosen to produce the desired *dynamic range*. *WHITE* also can be any gray-level less than 1 (255 in our case since our cameras use one byte per pixel) but greater than 0. It will only affect the dynamic range and can be compensated with the *constant*.

Methodology and results. We use a board big enough to completely cover the cameras' field-of-view at the sweet spot tracker position. We employ and hang gray matte (*diffuse reflectance*) paper because the *surface reflection* of the human skin takes place at the *epidermis* surface and it is approximately 5% independent of the lighting wavelength and independent of the human race [35]. The rest of the incident light (95%) enters the skin where it is absorbed and scattered within the two skin layers. A matte material has similar reflectance properties.

To obtain the *WHITE*[*x*, *y*] we run the system, pass each video frame through a *median* filter (to remove possible salt and pepper impulse noise, see **Chapter 4** of [33]), average 512 frames (to reduce noise) and then record the resulting left/right images (**Figure 4.2**). Using *VTK* APIs I managed to view the same images in 3D (**Figure 4.3**), where the grey-level becomes the Z-axis (height). Here we can see in more detail some of the camera image sensors problems represented as holes.

Then, we cover the camera lenses and run perform the same algorithm to get *BLACK*[*x*, *y*] (not shown because it is really black).

Using the shading correction formula, we obtain $\hat{a}[x, y]$ for $c[x, y] = constant$. We can see the correction pattern in **Figure 4.4**.

In **Figure 4.5** we can see the system running with two faces but before applying the shading correction. Then, after applying the shading correction (**Figure 4.6**), we can see how effective this method is in compensating for any illumination and/or camera anomaly. The resulting post-processed images are a little bit dark so I compensate with the *constant* (**Figure 4.6**).

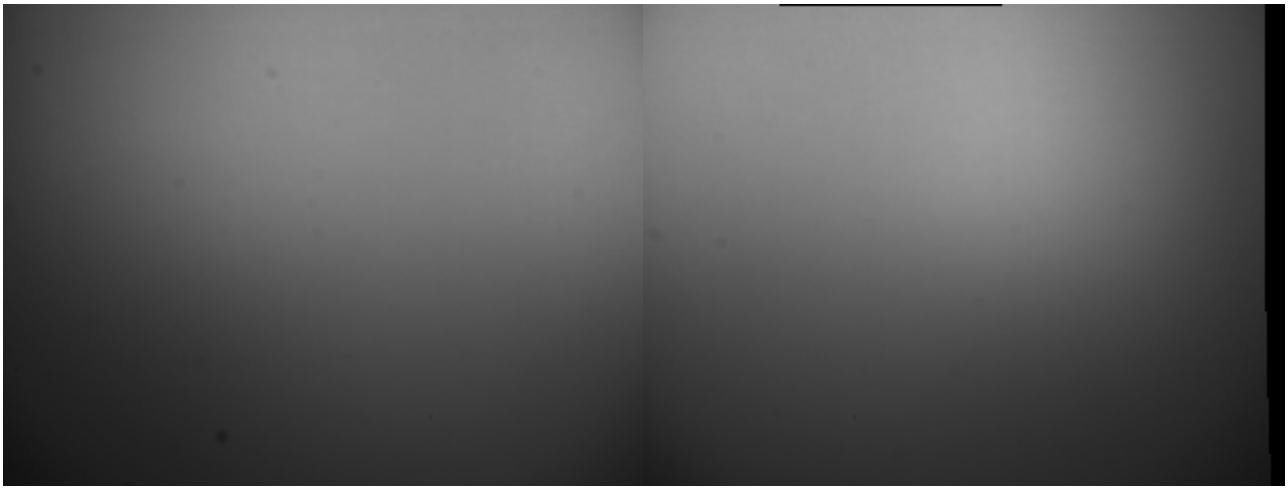

Figure 4.2: Left and right grey-level images after averaging

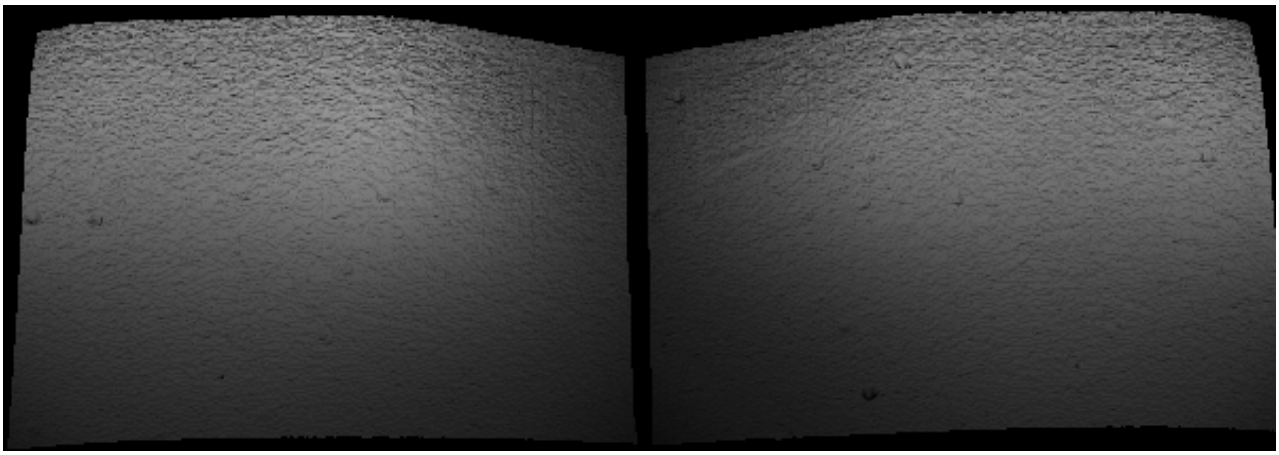

Figure 4.3: 3D view of the left and right images after the averaging

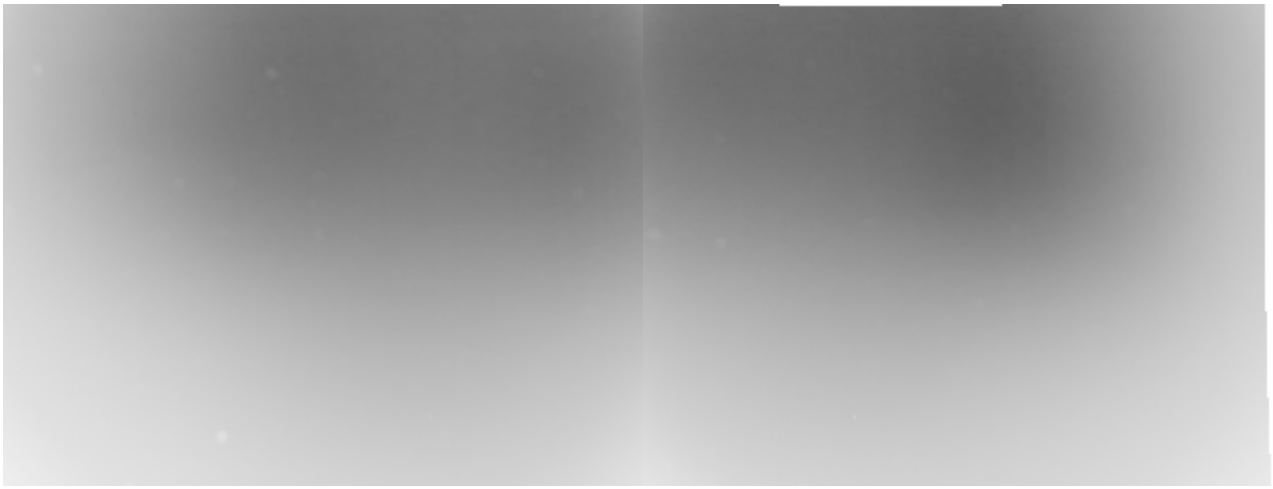

Figure 4.4: Left and right correction pattern

Figure 4.5: Left image with faces but without correction

Figure 4.6: Left image after correction and modifying the *constant*

4.3. Camera Calibration

For good stereo processing, the two images must be aligned correctly with respect to each other. The process of aligning images is called calibration. Generally speaking, there are two parts to calibration: *internal calibration*, dealing with the properties of the individual cameras and especially lens distortion; and *external calibration*, the spatial relationship of the cameras to each other. Both internal and external calibrations are performed by an automatic calibration procedure described in section **4** of the *Smallv-3.2pfd* manual that comes with the *Videre Design MEGA-D* stereo camera gear. The procedure needs to be performed when lenses are changed, or the cameras are moved with respect to each other.

From the internal and external parameters, the calibration procedure computes an image warp for rectifying the left and right images. In stereo rectification, the images are effectively rotated about their centers of projection to establish the ideal stereo setup: two cameras with parallel optical axes and horizontal epipolar lines. Having the epipolar lines horizontal is crucial for correspondence finding in stereo, as stereo looks for matches along horizontal scanlines. Without calibration, it is impossible for the stereo algorithms to find good matches.

Calibration parameters, along with other information about the stereo device settings, are stored in a parameter file that ends with the suffix ".ini". Parameter files are loaded automatically when the Tracker System start.

See an example of calibrated and un-calibrated cameras (stereo gear) in **Figure 4.7**.

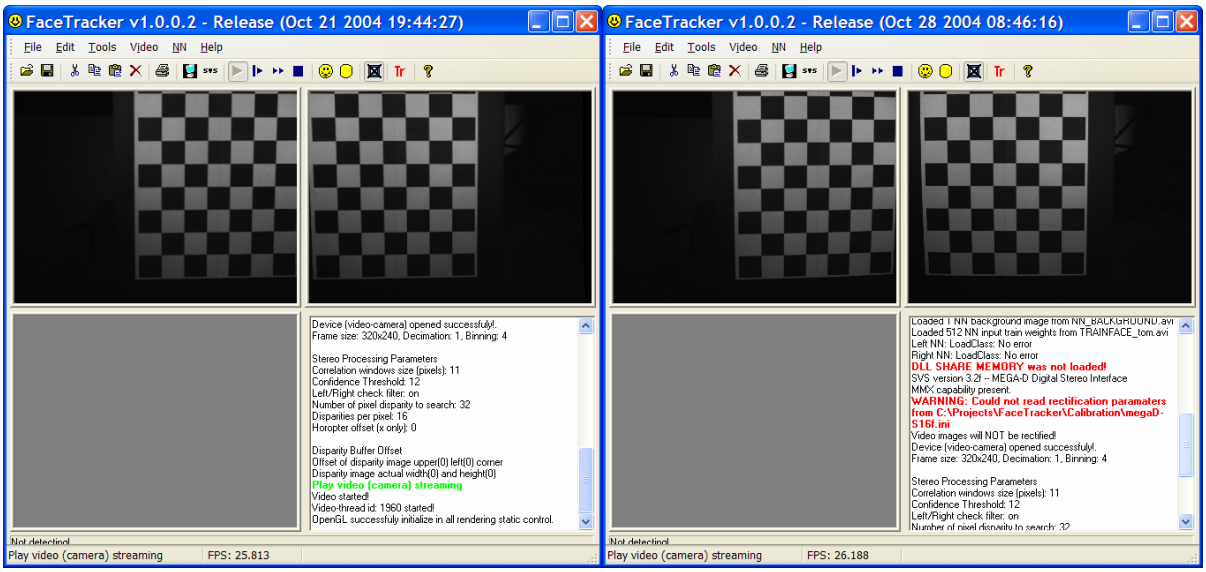

Figure 4.7: Calibrated and un-calibrated cameras.

5. HEAD TRACKER

5.1. Introduction

In this chapter I will present a neural network (NN) based algorithm to recognize, detect and track frontal and non-frontal views of a face in gray-scale images. After pre-processing (**Chapter 4**), the algorithm works by applying one or more neural networks directly to portions of the input stereo video images, and arbitrating their results. One left/right set of NN is trained individually to output the presence or absence of a specific face (*recognizer*). Another left/right set is trained to detect similar faces to this specific face (*detector*), and the last set is only trained to determine the presence or absence of a head (*tracker*).

During the recognition stage we use the arbitration between left and right NN, and heuristics to clean up the results and to improve the accuracy of the recognizer. After the face is recognized, the system switches into detection and tracking mode. At the end of the process, a simple but very fast stereo correspondence using a block-matching technique is used to obtain the 3D head position.

I will also explain the algorithm for performing a very fast training, more suitable for our VR environment, which will allow any new user to start using the tracker system in less than two minutes.

At the end of this chapter, I will evaluate the system during real time tracking and present its results.

5.2. Overview of tracking algorithm

The algorithm is composed of twelve basic steps (**Figure 5.1**): (1) Acquisition, (2) Shading correction (preprocessing), (3) Size reduction (sub-sampling), (4) Scanning; (5) Left *and* right recognitions; (6) Background test; (7) Arbitration; (8) Left *or* right detection; (9) Left *or* right tracking; (10) Prediction; (11) Stereo matching or correspondence (obtain 3D head position); and (12) Smoothing filter for the 3D output position.

(1) **Acquisition**. Process of grabbing left and right video frame.

(2) **Shading correction**. (Covered in C**hapter 4**).

(3) **Size reduction**. All neural networks receive a *face_width* by *face_height* pixel sub-window (region) of the image as an input, and then output a *confidence* number between 0% (*maximum confidence*) and 100% (*minimum confidence*). The reason why 0% represents the *maximum confidence,* and 100% the *minimum,* is because I use vector subtractions to compare input images against stored images inside the NN. As a result, similar images output *confidence* values close to zero, and dissimilar images greater than zero. During tracking, the *face_width* by *face_height* pixel region will be the *minimum* face-size that the system will be capable of tracking (*face_width* will be the face width in pixels and *face_height* will be its height). We will explain how the system will determine the *face_width* and *face_height* numbers (face-size) in the **Training** section.

To recognize, detect and track the face anywhere in the video input, the NN is applied at several locations in the image (see **Scanning** section below). To track the same face larger than the sub-window size the input image is repeatedly reduced (in size by sub-sampling) and the NN is again applied at each resulting image. This will allow the system to be size-invariant. To determine the optimal scale-down factor I use the following methodology:

- First, training the tracker system with a 'dummy'. The dummy is positioned at the 'sweet spot'.
- With the dummy in the original position, running the tracker and adjusting the threshold until the system begins recognizing the dummy. The algorithm was temporary modified to avoid switching to detection and tracking mode after the dummy is recognized.
- Moving the dummy forward until the system stop recognizing it. The closer to the cameras, the larger the size of its face (compare to the original *face_width* by *face_height* pixel size).
- In this position, trying different scale-down factor until the system begin to recognize again the dummy.

- Repeating the process until the dummy reaches the required minimum tracking distance.
- Record the number of sub-sampling we have to perform and each scale-down factors.

I repeat this process three times and average the results. The scale-down factors vs. distances from the camera are not linear (constant). Between a little bit farther than the 'sweet spot', at 160 cm, and the minimum distance of 102.6 cm, I obtained two scale factors of 1.25 and 1.31 after averaging the procedure three times. To simplify the algorithm and be in the safe side, I decided to use the minimum denominator (most conservative number). The scale down factor for this project is established in a *constant* = 1.2 and the number of time I have to perform the sub-sampling can be obtained from the following deduction:

In **Figure 3.6** we see that the 'sweet spot' distance is 154 cm and the minimum distance is 102.6 cm, therefore 154/1.2 = 128.3 cm, 128.3/1.2 = 107 cm, and 107/1.2 = 90 cm.

107 cm is close enough to 102.6 cm. and each time the system has to sub-sample the image it takes valuable CPU time. Therefore, I decided to scale the image down only twice.

(4) **Scanning**. When the tracker system starts, it automatically enters in recognition mode and stays in this way until the user's face is been recognized. For optimal performance during this mode, the *Recognizing* NN should test every pixel in the image (not including the borders) to look for the face. But given the nature of NN this test requires a lot of processing power and, as a result, it is very inefficient. NN are slightly invariant to position, meaning that if a face is recognized at the [x, y] position, chances are that the NN still will output a similar *confidence* number and recognize the face at [x+1, y+1].

Analyzing further showed that training the system with a *face_width* by *face_height* face size makes the *Recognizing* NN to have a *global maximum confidence* at [x_{face}, y_{face}] face position and *local minima* at [x_{face}+*face_width*/2, y_{face}+*face_height*/2]. A detailed analysis of this behavior showed that the *confidence* as a function of the face position around the *global maximum* is almost monotonically decreasing from x_{face} to

x_{face}+*face_width*/2 and from y_{face} to y_{face}+*face_height*/2. I mentioned 'almost' because there are ripples in the function, possible due to noise or internal NN behavior (see **Figure 5.5** and **Figure 5.6**). With this in mind, I developed the following efficient algorithm for recognition, lightly based on a computer vision region growing (seed) technique:

1. Given the width and height of the face, initialize the following variables and the algorithm:
   1.1. *x_offset* = *face_width*/10 and *y_offset* = *face_height*/10
   1.2. *x_step* = 3 • *x_offset* and *y_step* = 3 • *y_offset*
   1.3. Initialize a stack which will hold pixel coordinate numbers (*position stack*).
   1.4. Initialize *max_confidence* with the lowest possible confidence.
   1.5. Begin scanning for a face at pixel coordinates *x* = *face_width/2* and *y* = *face_height/2*.
2. Repeat:
   2.1. Check if this [*x*, *y*] pixel coordinate has already been tested for a face: **otherwise** continue with next step, but **if it has been tested**, advance to the next pixel *x* = *x* + *x_offset*. If we reach the end of the row (minus border *face_width/2*), go to *y* = *y* + *y_offset* and *x* = *face_width* and continue from there. If we scanned all columns (minus border *face_height*/2) go to next block (3) to check for *local maxima*.
   2.2. Extract an image sub-window of size *face_width* and *face_height* around this center pixel.
   2.3. Run the *Recognizing* NN with this sub-image and obtain the *confidence* = confidence output.
   2.4. Mark (flag) this [*x*, *y*] position as been tested to avoid testing the same location twice and wasting valuable time.
   2.5. If the *confidence* is equal or greater than the *recognition threshold*, save it in *save_confidence* = *confidence* and follow next steps, otherwise go to (2.1) and continue scanning for the tracker user's face.
   2.6. Save this [*x*, *y*] location in the *position stack* (push(*x*, *y*)).

2.7. If *save_confidence* is greater than *max_confidence*, update *max_confidence* with *save_confidence* (*max_confidence* = *save_confidence)* and mark this position as a *local maxima*. In later comparisons, will use this *max_confidence* value as a new *recognition threshold*. The idea behind this is always to go up-hill.

2.8. Repeat until the *position stack* is empty.

2.8.1. Get the last 'pushed' location from the *position stack* (pop(*x*, *y*))

2.8.2. For each 8-neighbors from the pushed position [*x*, *y*] that are located at a distance (in pixels) of *x_offset* and *y_offset*, do the following:

2.8.2.1. If this [*x*, *y*] pixel coordinate has already been tested for a face, go to test the next neighbor pixel (go to 2.8.2), **otherwise** continue with next step.

2.8.2.2. Run steps 2.2, 2.3 and 2.4 (run recognizer at this position and flag this location).

2.8.2.3. If the *confidence* is equal or greater than the *save_confidence,* continue with next step, otherwise go to (2.8.2) and continue testing another neighbor.

2.8.2.4. Save this [*x*, *y*] location in the *position stack* (push(*x*, *y*)).

2.8.2.5. Update *save_confidence* with this new *confidence* (to guarantee going up-hill).

2.8.2.6. Repeat 2.7 (compare against *max_confidence*, update it and mark or not as a *local maxima.*

3. If there is not any *local maxima*, exit the algorithm and return 'face not found'. Otherwise repeat for all *local maxima* that have been found (look for *global maximum*):

3.1. Get the [*x*, *y*] position of the *local maxima.*

3.2. From +*x_offset* to –*x_offset* and +*y_offset* to –*y_offset*, determine pixel by pixel the *confidence* at all locations around this [*x*, *y*] position (steps 2.2 and 2.3).

3.3. If any of this *confidence* is equal or greater than *max_confidence*, update *max_confidence* with this new *confidence* value and update *face_position* with this new location.

4. Finish the algorithm: return (output) 'face was found', *face_position* and *max_confidence*.

Once the face has been recognized, the tracker system predicts the position of the area (predicted sub-window, see **Figure 5.1** and **Prediction** below) in which the face might be (in the next frame), and then switch to 'detection' mode. In this mode, scanning and testing the image pixels to continue detecting and tracking the face is rather simple. As it was stated before, this is because there must be only one *confidence's global maximum*:

1. Only in the predicted image region, scan pixels every *face_width*/10 in *x*, and *face_height*/10 in *y* coordinates. Disregard scanning in the *face_width*/2 and *face_height*/2 borders.
   1.1. Extract an image sub-window of size *face_width* and *face_height* around each scanned pixel and feed the *Detecting* NN with this sub-image (similar to face recognition steps 2.2 and 2.3).
   1.2. If the *Detecting* NN output a *confidence* greater than the *threshold detection*, save the *confidence* in a position array (*confidence*[*x*, *y*] = output from *Detecting* NN at position [*x*, *y*].
   1.3. When the scan of the sub-windows is finished, look for *confidence's global maximum* number in each element of this *confidence* array. Returns this *global maximum* to the next block.
2. Switch to head tracking mode (change the *Detecting* NN with the *Tracking* NN). The *Detecting* NN proved to be sensitive to user's face pose and sometimes it jumps around the head center causing problems to the VR application (jittering). To smooth out this position we implemented this more general NN.
   2.1. With the *Tracking* NN in place, fine tune the position of the *Detecting* NN looking this time pixel by pixel around the position we got from the previous steps. The area is small: from -*face_width*/10 to +*face_width*/10 and from -*face_height*/10 to +*face_height*/10. Meaning testing around the *global maximum* to determine the absolute *maximum*.
3. Return this position to the system for further processing (**Prediction** and **Stereo Matching**).

The resulting algorithm is graphically explained in **Figure 5.2** and **Figure 5.3**. Here the system is tracking my face with a *face_width* of 48 pixels and a *face_height* of 80 pixels already determined during the training. In the same scene there is also a dummy face (my friends say that she is my girlfriend, isn't it sad? ☺). The display panel at the bottom shows in real-time a 2D color map in which oranges represent higher *confidences* and the white dot is the head position been determined by the tracker system. We can also observe in this case that during recognition the system tests at every 3•*face_width*/10≈15 and 3•*face_height*/10=24 pixels, but if there is a neighbor with higher *confidence*, the algorithm tests up-hill every *face_width*/10≈5 and *face_height*/10=8 pixels. Once the algorithm determines all *local maxima*, it tests at every pixel (this can be seen as little orange rectangles) to determine the *global maxima*. During detection and tracking (left of **Figure 5.3**.) the algorithm tests only in the predicted sub-window, testing every x=5 and y=8 pixels and looking for the *global maxima* Then it checks every pixel (from -5 to +5 in x and from -8 to +8 in y) around the *global maximum* to determine the *absolute maximum* (and therefore head position).

(5) **Left *and* right recognitions**. Each *Recognizing* NN (as *Detecting* and *Tracking* NN) is a Self-Organizing-Map (SOM) neural network. Each internal weight SOM is a *face_width* by *face_height* sub-window image but arranged in a one dimensional (1D) vector, in stead of two dimensional (2D). In general any vector inside any NN is a 2D image stored as 1D *face_width* by *face_height* array. Each vector is representative of a class of faces, more specifically a particularly average view of the face to be recognized. Currently, the *Recognizing* NN is composed of 256 *face_width* by *face_height* 1D vectors. A thorough explanation of SOM NN and how to train them is given in **Chapter 6**.

In this step, each left and right *Recognizing* NN is simply fed with the input pixels (in a form of a 1D vector) inside the ellipse bounded by the face-size sub-window (MxN). Each NN outputs a *confidence* number which represents, in percentage, the closest distance between the 1D input vector characterized by this image region and one of the 256 1D vectors stored inside the NN. This is called 'Winner-Takes-All' (WTA)

philosophy in the SOM jargon. Only one neuron (vector) is selected as output when the input vector is compared against all SOM vectors (neurons).

SOM NN uses the dot product between input vector and stored vector as the metric to determine who the winner is. The dot product is normalized, therefore equal vectors or closer (co-linear) ones output '1' or close to '1'.

In this thesis I departure from this well established method to determine the WTA because is too slow when you have to compare a lot of vectors (256 *face_width* by *face_height* vectors per each sub-window). As a metric of preference to determine the winner (closer), I decided to use the Sum of Absolute Difference (SAD) which will be explained in more detail in the **Stereo Matching** step. Therefore, the *confidence* is the SAD between the input sub-window vector and the winning SOM vector. Zero means equal vectors and the greater the number is the different the vectors are, and lower is the *confidence*. Both left and right output *confidences* go to the **Background test** and then to the **Arbitration** stage.

(6) **Background test**. Sometimes it happens that even though with the well trained NN the tracker system misunderstands some parts of the background scene for the face to track and tracks only there. This is called *false positive*. We will see, during the **Training** section, that I am not teaching the NN for a non-face (background) case despite all suggested research using NN for face detection and recognition ([1] and [3-6]).

Current research suggests that a large number of non-face images are needed to train a face detector or recognition, because the variety of non-face images is much greater than the variety of face images. For example, one large class of images which do not contain any faces could be pictures of scenery, such as a tree, a mountain, and a building. Collecting a "representative" set of non-faces is difficult to do. Practically, any image can serve as a non-face example (the space of non-faces images is much larger than the space of face images). The statistical approach of machine learning suggests that we should train the NN on precisely

the same distribution of images that will see at runtime. A representative set of scenery images could contains millions of windows, and training would be very difficult.

Since we are running the tracker system in an in-door controlled environment, the background is already established and it does not change very often (for instance, it does not have trees, mountains or cars). Our background is not 'dynamic'. Therefore, I devised a simpler solution which similar results (low *false positives*). Whenever the tracker user thinks that the background has been changed, he or she can command the system to take a 'snapshot' of the background (this means no person should be in the cameras' field-of-view during this moment).

For the background test, the algorithm uses the same input *face_width* by *face_height* sub-window which it was fed into the *Recognizing* NN (only pixels inside the bounded ellipse) as a 1D vector. If the *Recognizing* NN recognizes a face (the *confidence* number is greater than a certain threshold), we save the position and *confidence* number. We extract a *face_width* by *face_height* sub-window from the snapshot at the exact same location and use only the pixels inside the bounded ellipse to convert it to a 1D vector. Then, using SAD we compare both vectors (input sub-window and snapshot sub-window). This comparison is similar to the **Left *and* right recognition**. Note that both 1D vectors are comparables because they are of the same size. If the $confidence_{recognizer} > confidence_{background}$, the input region is indeed a face. Otherwise, ($confidence_{recognizer} < confidence_{background}$) the input region belongs to the background.

(7) **Arbitration**. We use the arbitration of left and right *Recognizing NN*s and heuristics to clean up the results and improve the accuracy of the recognition. For the following frames, the criterion is to choose the left/right *Detecting* and *Tracking* NN based which has the greater *Recognizing* NN *confidence* number. A much better solution would be to have several left and right *Recognizing NN*s in parallel and arbitrate among them. This would give us a greater recognition performance. But after some experiments, I found that even having one more *Recognizing* NN per channel (a total of four NN in parallel) dropped

the recognition frame rate to less than 1 fps. Even worse, it increased the training from two minutes to almost four. So I decide to compromise with the above mentioned solution. May be incoming faster PCs in the near future will allow to implement and test multiple *Recognizing NN*s per channel.

If both *Recognizing* NN *confidence* numbers are less than the threshold, the system will *not* switch to detection mode and will continue to try to recognize the user's face that it has been trained with.

(8) **Left *or* right detection**. Currently composed of 64 *face_width* by *face_height* 1D vectors, each vector represents different average views of the face. Once the arbitration stage determines which left or right NN has the best *confidence* number (and also above the threshold), the tracker goes to detection mode. During the following left or right video image frames, the system will try to detect a face only in a predicted left or right image region (sub-window). The *Detecting* NN *confidence* output is compared against a different *detection threshold*. Having two different thresholds, one for recognition and one for detection, allows the user to fine tune the tracking performance (see GUI **Figure 1.6**).

(9) **Left *or* right tracking**. Currently composed of 8 *face_width* by *face_height* 1D vectors, each vector represents different average views of the face. In this particular case, there are only few vectors trying to represent the 512 training input vectors (see **Training**). Therefore, after the process of training, each vector becomes a head blob instead of a detailed face view (see **Figure 5.4**).

With the position already determined by the *Detecting* NN, the tracker switches to head tracking mode (change the *Detecting* NN with the *Tracking* NN) and scans again around that last position. The *Detecting* NN is sensitive to user face-pose. For example, sometimes the user looks down or tilt his or her head a little bit, and the 3D head position jumps around the head center causing problems to the VR application (jittering). To smooth out this 3D position we implemented this more general NN. Without implementing this NN, the **Smoothing Filter** (final stage) needs more samples to successfully smooth out the 3D output position. Unfortunately this increases the latency more than the proposed 100ms.

(10) **Prediction**. Both left or right *Recognizing* NN (after arbitration) and *Detecting* NN send their position to this module. Thus, during the next video frame, the tracker will only look in a predicted position and sub-window size. The current work does not predict any 3D head position (which of course might decrease the tracker latency). This issue is open for future work.

To predict the coordinates of where to look next, I implement a simple gradient algorithm. I use the current position vector $[x_{current}, y_{current}]^T$ and the previous one $[x_{previous}, y_{previous}]^T$ to obtain a prediction vector (direction and magnitude) subtracting current minus previous vectors:

$[x_{prediction}, y_{prediction}]^T = [x_{current}, y_{current}]^T - [x_{previous}, y_{previous}]^T$, therefore the next predicted position will be

$$[x_{next}, y_{next}]^T = [x_{current}, y_{current}]^T + [x_{prediction}, y_{prediction}]^T$$

During the next video image frames, the size of the area in which the system will look into is fixed at 2•*face_width* by 2•*face_height*.

(11) **Stereo matching**. The *Tracking* NN determines the 2D position in one of the (left/right) video images. Then we have to use some stereo correspondence algorithm to get the 3D position using the remaining video image without using too many CPU cycles. Otherwise all the tuning, compromises and trade-off needed to reach real-time tracking will be lost in this critical and CPU intensive section.

Given a perfectly calibrated non-verge stereo camera gear (binocular geometry constraint or *epipolar constraint*) the z position is determined by the following *triangulation* formula [33]:

$$z = b \bullet f / d$$

where b is the baseline or distance in x direction between the left/right camera axis (see **Figure 3.5** and **Figure 3.6**), f is the focal length and d is the disparity or distance between points of a conjugate pair when the two images are superimposed.

Correspondence methods attempt to match pixels in one image with corresponding pixels in the other image. For simplicity, we refer to constrains on a small number of pixels of interest as *local constrains* as opposite as *global constrains* which involves the entire image or scan-lines. Local methods can be very efficient, but they are sensitive to locally ambiguous regions in images. Fortunately, given the way I designed and used the tracker system, neither occlusion regions nor uniform texture are likely to happen very often during the tracking. In case of region with uniform texture, using the whole face to perform the matching almost guarantees a very good stereo correspondence. In case of region occlusion, the NN are pretty robust to this problem. It is still able to track even with 1/3 of the face covered. Moreover, since the occlusion is likely to happen in both cameras' field-of-view, the face images will not be difficult to match.

Among the fastest of these local methods is the *block matching* methods. They seek to estimate disparity at a point in one image by comparing a small region about that point (the template) with a series of small regions extracted from the other image (the search region). The epipolar constraint reduces the search to one dimension. Three classes of metrics are commonly used for block matching: correlation, intensity differences, and rank metrics.

$$\text{Normalize Cross-Correlation (NCC)} = \frac{\sum_{u,v}(I_1(u,v)-\overline{I}_1)\cdot(I_2(u+d,v)-\overline{I}_2)}{\sqrt{\sum_{u,v}(I_1(u,v)-\overline{I}_1)^2\cdot(I_2(u+d,v)-\overline{I}_2)^2}} \tag{5.1}$$

Normalize cross-correlation (NCC) (see equation (5.1)) is the standard statistical method for determining similarity. Its normalization, both in the mean and the variance, makes it relatively insensitive to radiometric gain and bias. I implemented this but it proves to be too slow (CPU intensive) for real-time tracking.

$$\text{Sum of Squared Differences (SSD)} = \sum_{u,v}(I_1(u,v)-I_2(u+d,v))^2 \tag{5.2}$$

$$\text{Normailized SSD} = \sum_{u,v}\left(\frac{(I_1(u,v)-\overline{I}_1)}{\sqrt{\sum_{u,v}(I_1(u,v)-\overline{I}_1)^2}}-\frac{(I_2(u+d,v)-\overline{I}_2)}{\sqrt{\sum_{u,v}(I_2(u+d,v)-\overline{I}_2)^2}}\right)^2 \tag{5.3}$$

I also tried the sum of squared differences (SSD, see(5.2) and (5.3)) metrics which are computationally simpler than cross-correlation, and it can be normalized as well. But still, it degraded the tracker performance too much. The results were a little worse than NCC.

Finally, reading [36] which has a very good overview of these metrics, I found an intensity difference metric (SAD or *Sum of Absolute Differences*), which was perfect for my application. Even more, it can be specifically tuned for the *Intel Pentium IV* family processors (one of the reason I choose this platform) because it has an assembler instruction (Streaming Single Instruction Multiple Date Extensions 2) which subtracts sixteen bytes at a time and perform the sum of their absolute values. It is a key factor in my thesis which allows me to reach the real-time goal (most of the block matching and NN algorithms are implemented in macro-assembler or *intrinsic)*. I am using SAD not only in the stereo correspondence algorithm but also as a metric to compare input image vectors against each NN weights 'class-face' vectors and determine who the winner is. Remember that each NN is a SOM NN with the WTA philosophy.

Sum of Absolute Differences (SAD). Consider a template of $k \bullet l$ pixels with k rows and l columns. $I_L(i,j)$ is the intensity values of pixel (i,j) in the template in the left image. $I_R(i,j)$ is the intensity values of pixel (i,j) in the template in the right image. If we are working with gray-level images, then the SAD of these templates is calculated in the following way:

$$\text{SAD} = \sum_{i=1\ldots k} \sum_{j=1\ldots l} \left| (I_L(i,j) - A_L) - (I_R(i,j) - A_R \right|$$

$$A_L = \frac{\sum_{i=1\ldots k} \sum_{j=1\ldots l} I_L(i,j)}{k * l}$$ is average intensity of left template and

$$A_R = \frac{\sum_{i=1\ldots k} \sum_{j=1\ldots l} I_R(i,j)}{k * l}$$ is average intensity of right template

(5.4)

A_L and A_R are used to normalize the templates. Normalization reduces the effect of shadows resulting from different positions of the two cameras or from uneven lighting in the field-of-view *but* it slow down in the overall performance because it involves converting the bytes per pixel (unsigned characters) to float in order to perform the division.

Using the shading correction global preprocessing already explained in **Chapter 4** I was able to obtain similar results without having to perform this normalization process.

SAD = 0 if both template are the same. The values increase as the matching become worse. The highest possible value is (k•l•255) where intensity ranges from 0 to 255. So SAD values ranges from 0 to (k•l•255). This range can be transformed into a range from 0% (best) to 100% (worse) by dividing it by (k•l•255) and multiplying by 100.

Of course, the stereo cameras have to be calibrated in order to extract valid results from the *block matching* (see section **4.3 Camera Calibration**).

(12) **Smoothing Filter**. The last step is to smooth out the 3D position without adding too much latency. As it was explained before in section **Left *or* right tracking,** we could not apply the smoothing filter alone (without the *Tracking* NN) because in order to lower the jittering, I had to sample too many 3D positions (more than ten) before passing the information to the VR application. This increases the

latency to 150 ms which is not acceptable. The problem is that the tracker is not stable enough without the filter.

After I added the *Tracking* NN things improved, but still I observed some small jittering or sudden jumps around the head center. So I combined the *Tracking* NN with a small 3D position smoothing filter. I tried several 1D filters [33] like Gaussian, Averaging and Median. Finally the best compromise was a median filter using only four samples (four 3D previous positions).

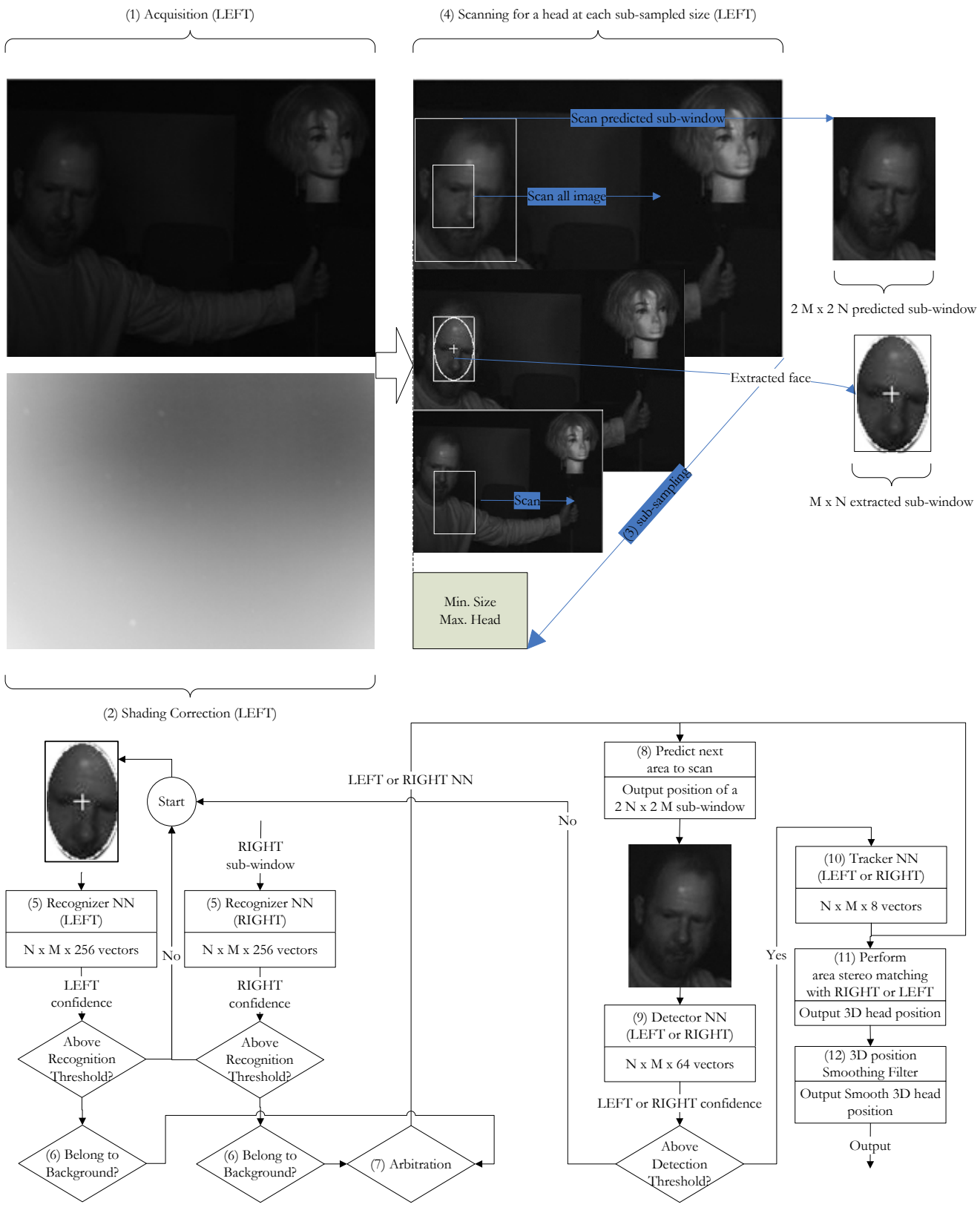

Figure 5.1: Overview of the tracker system basic steps

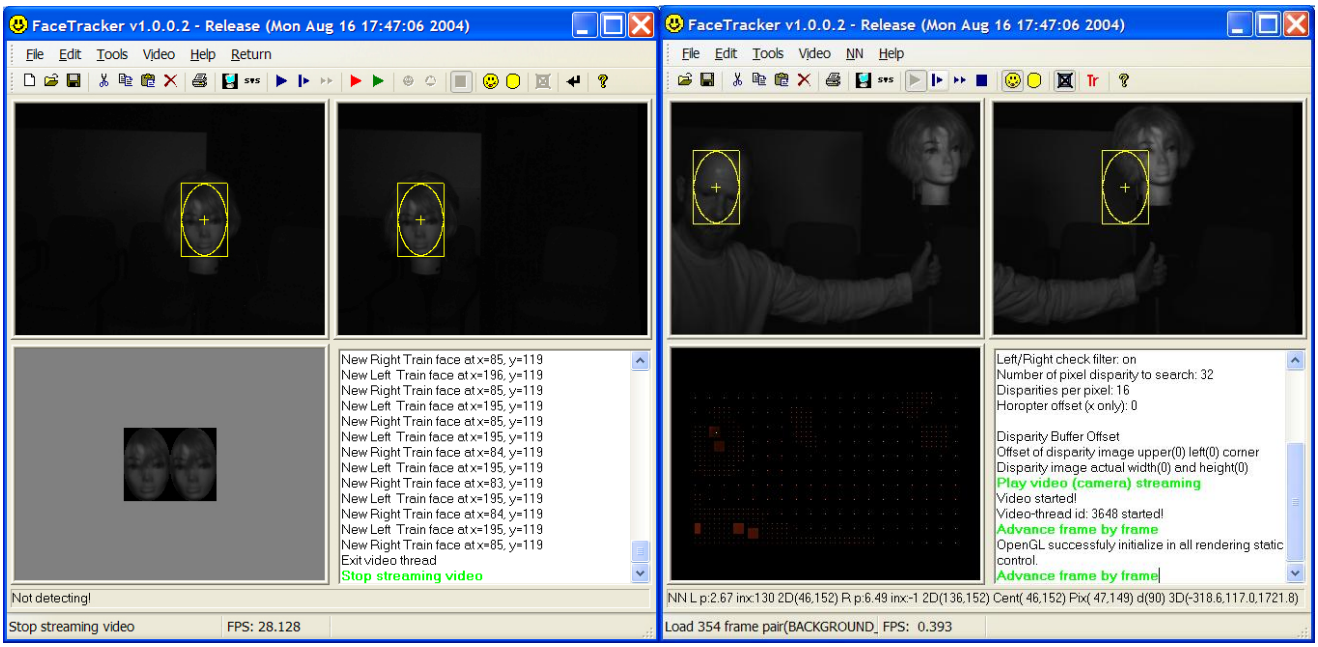

Figure 5.2: Dummy head during training (left). During tracking, a user already been trained and recognized, and the dummy (right).

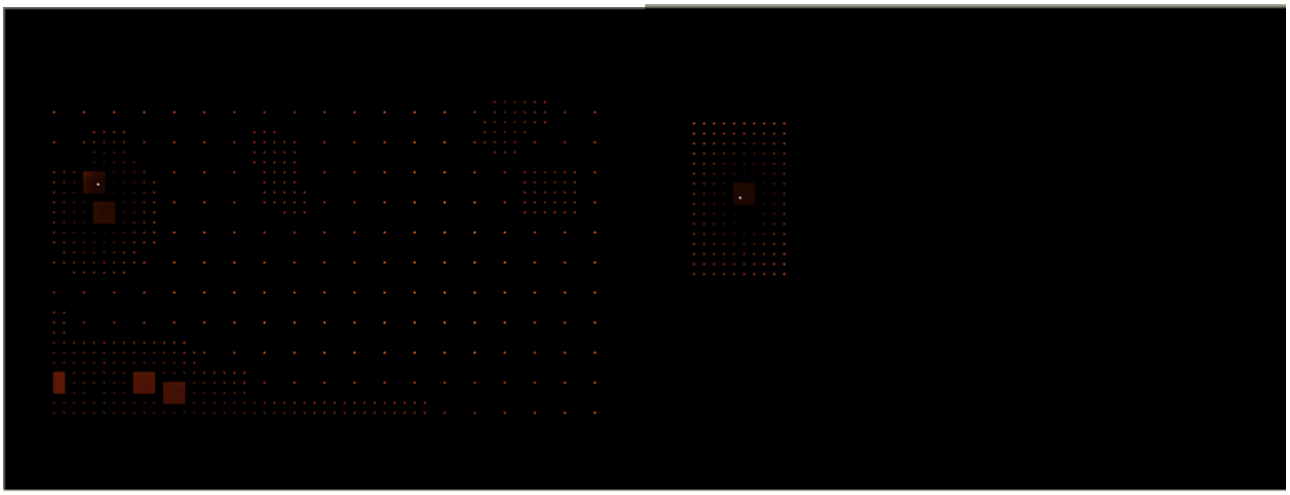

Figure 5.3: Detail of 2D *confidence* map during recognition and tracking.

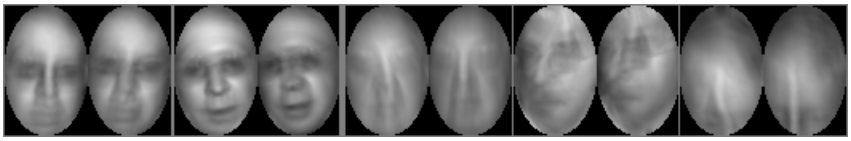

Figure 5.4: Example of *Tracking* NN internal weights. Each weight is represented by a *face_width* by *face_height* vector. As you can see, face details are almost lost and the face becomes more like a blob.

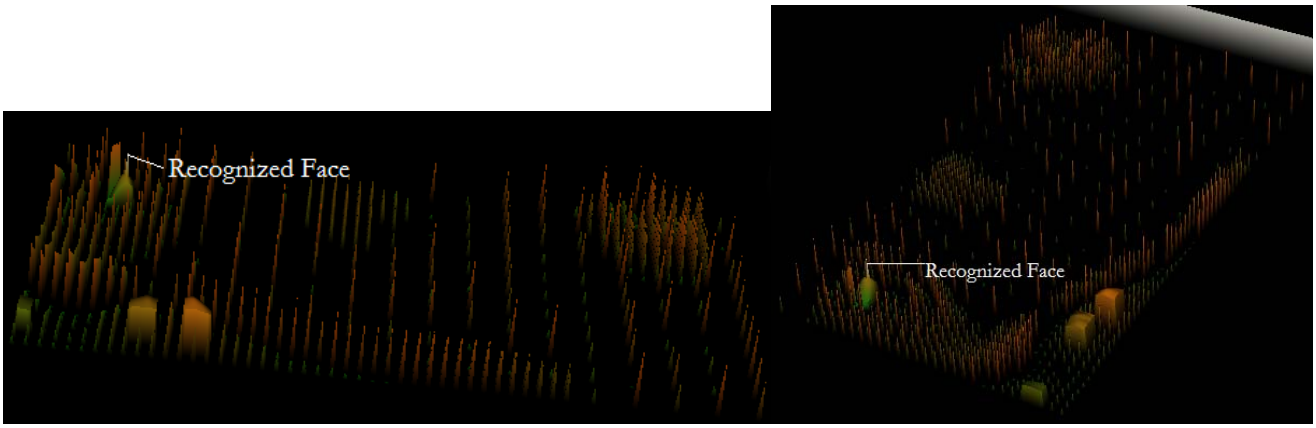

Figure 5.5: Detail of 3D *confidence* map during recognition.

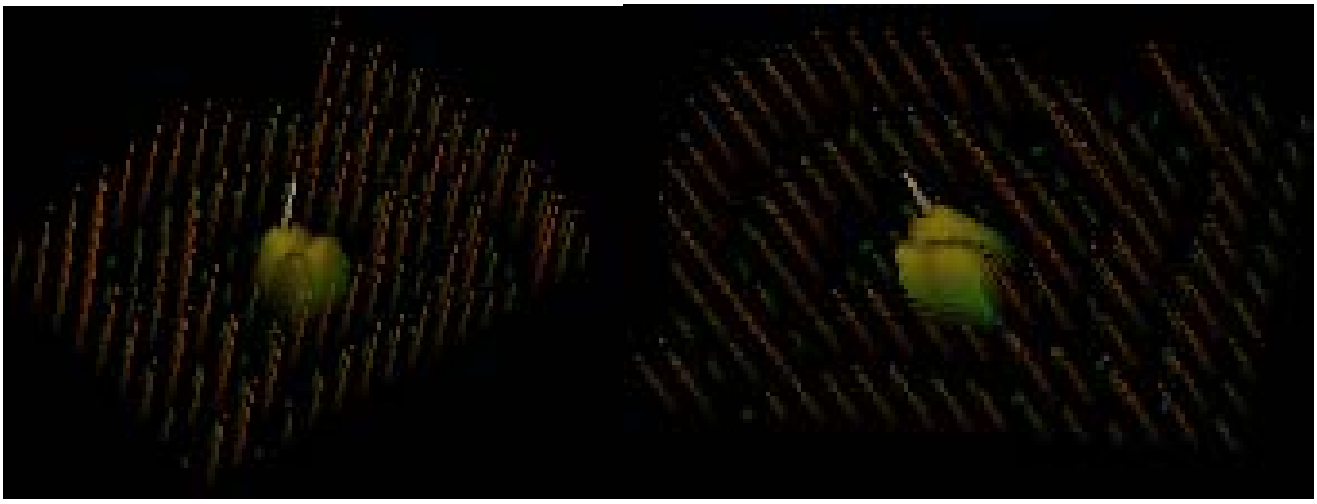

Figure 5.6: Detail of 3D *confidence* map during tracking.

Training

To recognize or detect (and therefore track) any object (a face in our case) all neural networks have first to know the object to be recognized or detected. This means that we have to use a 'learning' algorithm to teach the NN how to do it [3-6]. Face detection and recognition also provide interesting challenges to the underlying pattern classification and learning techniques. When a raw or filtered image is considered as input to a pattern classifier, the dimension of the feature space is extremely large (i.e., the number of pixels in normalized training images). The face-class (recognition) or classes of face (detection), and non-face images are decidedly characterized by multimodal distribution functions and effective decision boundaries are likely to be nonlinear in the image space. To be effective (taking an acceptable amount of time), either classifier must be able to extrapolate from a modest number of training samples (which is proven to be difficult), or must be efficient enough when dealing with a very large number of these high-dimensional training samples.

This means that we must present to the NN sufficient views of the face to extrapolate all possible poses of the face from the views, and then be able to recognize them. And here resides the big challenges: how do I automatically teach the NN without knowing a priori what a face is? And how do I do it in real-time so anybody can use the system without having to spend a lengthily training time before starting to be tracked? During the following section I will describe this issue, which I consider it is my real-contribution to the field. Most of the research in the area [3-6] consider the training as an off-line process, with a lot of human intervention. They don't usually report the time it takes them to perform the training, which based on the description of their methodology, seems to be very time consuming.

5.2.1. Methodology description

We ask the user to be tracked to sit down slightly behind the 'sweet spot' in front of the cameras, because the size of the face we get in the stereo video images will be the *minimum* face size the tracker system will be able to recognize, detect and track. It does not matter if he or she is in the exact center of the coordinate system. As it was explained during **Overview of tracking algorithm** (5.2), the system will

perform several image re-sizing preprocesses to consider a larger face when he or she leans over (get closer to the cameras) when is using the tracker.

The user is then recorded taking slowly runs through a series of subtle head poses (chin left/right, chin up/down, tilting left/right) over the course of 512 frames or 17 seconds (512 at 30 fps => 512/30 = 17 sec.). This process allows the system to accommodate for the natural changes in the user's face orientations (views) while using the virtual reality system. It is also desirable to slightly move forward (lean over) and backward so the NN can be trained with a little variability in depth and be more robust for changes in the z coordinate of the head position. Remember (section **5.2**) that we are scaling-down the images using a factor of 1.2. Therefore, any intermediate size (not the sub-sampling) has to be taken into account by the NN. The user can slowly move his o her body (and head) to each side as long the face is maintained in the combined cameras' field-of-view.

Once the recording is finished, the tracker system automatically presents the first left and right stereo frames superimposed with a left/right ellipses bounded by a box (**Figure 5.2**). Using the GUI (**Figure 1.5**) the user has to manually move each ellipse to the center of each face and then adjust their minor (x) and major (y) axis to approximately fit an oval. This is the only necessary human intervention that takes only a few seconds to do. Once the user finishes this task, the width and height of the bounding box will become the width and height of the face, and therefore, this will be the *minimum* user's face-size that the system will be able to track. On other words, this is the *maximum* depth (z coordinate) of the head position.

After this calibration, the user has to initiate the training process, which will take approximately two minutes to finish. The system saves not only the recorded video but also the training parameters (SOM internal weight vectors) in a form of video images (Audio Video Interleaved or AVI file) so we can visually check the already trained vectors. The user can recover and load these parameters at anytime later. The system will not need to be re-trained to track the user, unless there has been a change in the environment

(camera calibration, illumination adjustment, etc.). After this process, the system is all set to start tracking the user.

5.2.2. Algorithm description

The main goal is to automatically extract all face poses from the image sequence and train the NN with only these images in a reasonable amount of time (< 2 minutes).

I implemented two approaches in parallel: an elliptical head tracker as well as what I called 'an incremental NN training'. For the latter, I create a temporary *training* NN. As it was explained before (see section **5.2** above) I use the Sum of Absolute Difference (SAD) metric when I have to compare each input vector (image sub-window) against each weight vector in this *training* NN.

The reason why I could not use only ellipse-based trackers to extract the face image is because after I have implemented several different algorithms of this kind [37, 38] and [39, 40] I found all of them jitter around the face center (given the presence of only one face on the scene). Unfortunately, research papers never post this stability parameter. Given the nature of these algorithms, their instability is quite understandable because they use a grid of pixels to determine either the tangent or the Hough Transforms which decrease the center resolution. For this project, I finally choose a head detector based on Ellipse Hough Transform [37, 38], slower than [39, 40] but definitively more stable to compute the head-position.

For each individual left and right NN (they are trained independently):

1. Initialization:
   1.1. The system uses a 512 video frames for training, so we initialize a 2D *training* array (we call it *training* NN) of 512 rows (*weight* vectors) and *face_size* columns, where *face_size* has width (*face_width*) and height (*face_height*). This is the size of the sub-window with the ellipse inside which was determined above (read **Methodology description**).

1.2. Given any $[x_{center}, y_{center}]$ center coordinate, initialize the Ellipse Hough Transform function to look for the *ellipse maxima* in the range of $x_{center} \pm$ (*face_width* + 8 pixels) and $y_{center} \pm$ (*face_height* + 8 pixels).

1.3. When the system begins to train with the first recorded frame (when the user is supposed to be looking straight to the cameras), the ellipse and bounding box around the face has already been set and positioned at its $[x_{center}, y_{center}]$ face center. We run a Hough Transform based elliptical head tracker [37, 38] around this center, searching for the *maximum* Ellipse Hough Transform in the range of $x_{center} \pm$ *face_width*/2 and $y_{center} \pm$ *face_height*/2, and then adjusting the old center with the new one. Given the search range of $\pm$ *face_width*/2 and $\pm$ *face_height*/2 only one ellipse (*maximum*) should be found by the Hough Transform.

1.4. From now on (next 511 training frames), we will fix the *y* coordinate extraction point at this new y_{center}. This is because after training several subjects, I observed that when they look down (chin down) and if they are wearing t-shirts, sometimes the elliptical head tracker got stuck in its round collarless neckline (curly crew neck). Besides, we want to train the NN with the head center when the subject is looking either down or up because the system does not track the head orientation. On other words, from now on we fix the height of the head center.

1.5. We initialize a smoothing median filter (see **Smoothing** section **5.2** above) using this sample (coordinate center).

1.6. Using this new center we extract the region inside the box (image sub-window) and store it as a 1D vector into the *training* NN. This will become its first *weight* NN vector.

2. Repeat each of the next 511 training video frames:

2.1. For each pixel in the range of $x_{center} \pm$ *face_width*/2 and y_{center} do the following:

2.1.1. Extract an image sub-window of size *face_width* and *face_height* around this pixel.

2.1.2. Run the *training* NN with this sub-image and obtain the *confidence* = confidence output.

2.1.3. Look for the *maximum confidence* in this range and return this new $[x_{center}, y_{center}]$ center position at this *maximum*. The new located face should be at this coordinate.

2.2. **Explanation**. In the first case, the *training* NN has only one *weight* vector already stored (the rest 511 vectors are zero). So if the user move his or her face slowly, the second frame should have a similar face at a similar $[x_{center}, y_{center}]$ position respect to the first frame. Once this second face is extracted and entered into the *training* NN (which, up to this point, has only one 'image-face' stored from the first frame), it should locate this 'second face' without any problem, after all, there are similar. As training progresses, the *training* NN will be filled by previously found faces. Remember we are looking only in the range of last $x_{center} \pm face\_width/2$ and y_{center}. Hence only one face can occur in this image area. With the slow movement restriction, new faces will be similar in pose and position to the ones that were already stored. These new faces will be found immediately. The **advantage** of this method is that since the position center follows the face in a very smooth way (no jumpiness), after training the recognizer, detector, and *Tracking* NN with these highly correlated vectors, the resulting head position will also tend to be smooth. But there is a big disadvantage: using only this method alone might guarantee the stability, but not the precision. When the user head is moved to each side (chine left/chin right) this 'incremental NN training' tends to displace the head center to one side of the face or the other, showing some background. In certain way, this method follows the tip of the nose. Even worse, sometimes it never recovers and tends to center at a half face. We need another process in parallel which has to not only maintain this smoothness but also correct the position to the head center. Despite all drawbacks that elliptical head tracker alone might have (lack of smoothness, unreliability as face detectors, excessive *false-positives* and *false-negatives*, etc.), combining both methods provides me with an excellent solution for this 'automatic' NN training. Remember that we limit the ellipse search where only one face could be found in the sub-image and, given this constraint, elliptical head trackers are very good finding the center of the head (ellipse). **Summarizing**, the incremental NN training provides smoothness to the extracted training date, and the ellipse head tracker assures these training data are going to be located close to head center.

2.3. Find the face using the elliptical head tracker around the previous x_{center} face center position using the search range of [$x_{center} \pm face\_width/2$, y_{center}], smooth out its results and compare both reported centers (elliptical and incremental). If the x_{center} face center reported by the incremental NN training is drifting apart from the reported by the elliptical head tracker in more than $face\_width/10$ pixels, correct it, otherwise continue with the x_{center} provided by the *training* NN.

2.4. We store the new center in x_{center}, and y_{center} and continue with the next frame.

3. After we finish the whole training set we are ready to train each individual NN with 512 face poses taken from the *training* NN. In this case, I use two very well documented methods: simple vector averaging, and Kohonen-based algorithm for SOM training [41, 42] which is fully explained in **section 6.6** (**SOM Modules: description, input and training)**.

4. ***Recognizing* NN**. The recognizing NN has 256 weight vectors (neurons) and the *training* NN has 512 vectors. Each training vector represents a face pose with a unique characteristic: given the nature of head movements, all these internal 'feature vectors' are highly correlated so in this particular case, simple averaging in pairs to obtain 256 recognizer weight vectors from 512 training vectors gives me very good results. I compared these results against the Kohonen algorithm (**section 6.6**) and I obtained similar outcomes. The huge advantage is the speed: vector averaging is much faster.

5. ***Detecting and Tracking* NN**. Here we have to go from 512 training vectors to 64 (*Detecting* NN) and 8 (*Tracking* NN). Again, there are the same highly correlated training vectors but simple averaging implies taking 8 (detector)/65 (tracker) training vectors and average them into 1 detector/tracker weight vector. I tested both algorithms and evaluate the tracker performance (see section **5.3**) and obtain better results with the Kohonen method.

6. The resulting weight vectors are saved as AVI files under the user's profile. The user can visualize these vectors using the tracker system GUI and recover (load) this vectors for future use (not need to re-train again unless change in the camera settings or environment).

5.3. Evaluation

A number of experiments were performed to evaluate the system. We show an analysis of the tracker system at each stage (recognition, detection and tracking) and it corresponding errors running in real situations (conferences, demos, etc.). Stability, resolution and latency are also included. Results in tabular form can be found in section **1.7**.

5.3.1. Methodology

During several experiments, demos, etc. I recorded and collected several video images and choose thirty based on their contents (more than one person in front of the cameras, people's faces in very different poses, a particular face disappearing from the camera field of view and reappearing later, etc.). Each video segment has 512 frames (512/30 fps = 17 sec.). Once all thirty segments are put together I end up with one long test video of 17 • 30 = 510 sec. (8.5 min). There I counted 150 different persons (faces), each one in many different poses (see **Figure 5.7** as an example) across the video images.

At the same time, I gathered thirty different training video sessions (thirty different faces) with the condition that the face of the person with which the tracker system is being trained comes into view at least once during the test video (see **Figure 5.8**).

Figure 5.7: Snapshot of video images recorded to evaluate the tracker system

Figure 5.8: Snapshot of a sequence of training video images

5.3.2. Tracker System Error in Recognizer Only Mode

During this test I temporary disabled the 'change to detection mode', meaning that once the tracker system recognized a face it will not switch to detect any face again during next frames, but it will try to recognize the trained face over and over. I didn't disable the prediction module, therefore once the tracker 'locks' in the target face (right or wrong) it will continue during next frames to search only in a small predicted image-area. For each of the thirty different persons (faces) used for training, I ran the tracker using the test video with 150 different faces in which the trained face is also displayed. Since this is a tracking system I decided to define a *false positive* when the system 'locks' either in the wrong face or in a non-face. A 0% recognition error would be zero *false positives* in which all thirty faces were properly recognized.

After training with the thirty different faces and running the tracker with the test video thirty times, I obtained five cases where the system locks in the wrong face (sometimes even in a non-face object) giving a tracking error (in recognizer mode only) of 100%-25/30•100=16%. After re-training the system with those faces the tracker couldn't recognize I was able to obtain better results recognizing 28 out of 30 cases (two *false positives*), consequently dropping the error to 6.6%. A third attempt recognized 27 faces giving an error of 10%.

I also tested the tracker error for one face only. In this particular case, I trained the system with one user face and ran the recognizer using the same training-face video in which, of course, only one face was present. This is a simple case in which one particular user wants to use the system alone without any other person (face) present in the camera's field of view that could cause the tracker to fail (lock in the wrong face). I repeat this experiment for each thirty training videos and only one failed (100%-29/30•100=3.3%) but after re-training this face the error was 0%. I repeat this evaluation three times with the same results.

5.3.3. Tracker System Error in Detector Only Mode

Suppose we want to track *any* face to feed a multi-user and multi-view VR system. How well the tracker system can perform as a multi-face tracker? Unfortunately not very good, basically for three reasons:

1. **Faces Database**. In order to train the *Detecting* NN to detect any face, the NN must be trained with a huge amount of faces so it can internally learn a general face. For front face only, we can rely on several large databases like the FERET facial database from Georgia University (1,109 sets comprising 8,525 images), CMU and Harvard face database (1050 face examples) and ORL database of faces (400 images with 256 gray levels per pixel in PGM format). To reduce the amount of variation between images of faces we have to normalize them which mean aligns each one with one another (same 2D position, orientation and scale), extract them from their background and preprocess for brightness and contrast. A thorough explanation of each one of

these steps can be found in C**hapter 2** of [1]. These methodology, beside of being very time consuming (it is not fully automatic), still will not take into account all possible face's poses, but just only for frontal faces with slight rotation and tilt. To detect other poses based only in frontal face databases different algorithms need to be developed.

2. **Number of Internal Neurons**. To speed up the tracker frame rate (see **Chapter 5**) the *Detecting* NN module uses only 64 *face_width* by *face_height* 1D vectors (neurons) to store similar face poses (average user-face at similar positions). Based on my experimentation, this is enough for detecting the user's face in most of his/her poses during tracking and still avoid locking or tracking non-face objects. But definitively this is not enough to learn a general face, even less in multiple poses. Variability in the images of the face is going to increase the complexity of the decision boundary to distinguish faces from non-faces beyond the capability of this number of neurons. What is going to happen is that the NN will end up generalizing too much (too few decision boundaries) and detecting non-face objects as faces (*false positives)* or missing faces (*false negative*).
3. **Background Training**. Training a NN for the face detection task is challenging because of the difficulty in characterizing prototypical "non-face" images. For example, unlike upright face recognition in which the classes to be discriminated are different faces, the two classes to be discriminated in face detection are 'images containing faces' and 'images not containing faces'. It is easy to get a representative sample of images which contain faces, but much harder is to get representative samples of those which do not. To simplify the task, this tracker is trained with the fixed current background seen by the installed cameras, which is enough for head tracking purpose, but not enough for a general face detector.

Despite all the mentioned problems I went ahead and tested the tracker system as a face detector. To do so I performed the following procedure: I trained the *Detecting* NN with the thirty training videos (thirty

different persons in all poses) but adding the condition of not initializing the *Detecting* NN between training sessions. Once I finished the whole training I temporary modified the system to start in detection mode, to always scan the whole image-frame, and then to mark the number and position of faces it detects. This includes modifying the algorithm **Scanning** depicted in **Chapter 5** in order to search for all *local maxima*. Then I ran the system using again the same test video with 150 uniquely identified faces, counted the *true positives*, *false positives*, and *false negatives*, and summed up the *false positives* and *false negatives* as *tracker error* (after all, detecting a non-face or missing a face are errors from a head tracker point of view). I performed this test three times, re-initializing and re-training the *Detecting* NN every time. After this, I repeated the same process but this time using the training videos as test (only one face present per segment). **Table 5.1** summarized the results.

Table 5.1: Tracker error as a detector using 150 faces.

150 Faces	**True Positives** (right detection)	**False Positives** (non-faces)	**False Negatives** (missed faces)	**Tracker Error** (sum of False)
First Test	147 (98%)	52	3	55
Second Test	144 (96%)	68	6	74
Third Test	146 (97.3%)	57	4	61

Table 5.2: Tracker error as detector using one face.

1 Face	**True Positives** (right detection)	**False Positives** (non-faces)	**False Negatives** (missed faces)	**Tracker Error** (sum of False)
First Test	1 (100%)	5	0	5
Second Test	1 (100%)	4	0	4
Third Test	1 (100%)	3	0	3

5.3.4. Tracker Performance Rate

Here I will determine the performance of this system acting as a real tracker. On other words, how well it can track a user, independently of its precision, resolution, etc. Since the system start trying to

recognize the tracker user, any error in the recognition module will undoubtedly affect its performance as a tracker. The figures are obtained from the previous section. The tracker error during recognition goes from 6.6% to 16%, depending on the training. Therefore, when the system starts, the performance goes from 84% to 93.4%. After the user is correctly recognized, the system switch to detection/tracking mode and the prediction module assures that the system will only look in a small window around the user's face. This is the case in which the system has to detect/track only one face and, based on our previous evaluation, is a 100% certain the tracker will find the user's head.

There is one caveat about these results. During the evaluation as a detector (see previous section) we concluded that the system can produce a lot of *false positives*. The side effect of this is that if during tracking the user makes sudden movements, the prediction module can fail and the system will end up trying to find a face in a sub-window in which there is no tracker's face. Two things could happen here:

1. The tracker cannot find any object (face or non-face) whose *detection confidence* is greater than the *detection threshold* and, as a result, it switches back to recognition mode. In this case the tracker is functioning properly because it will look for the tracker's face again.
2. The tracker determines that there is one object (non-user face or non-face) with enough *confidence* (above *detection threshold)* to be interpreted as the tracker user's face. In this case the system is failing to track the right person. There are two workaround: to increase the *confidence threshold* or to restart the tracker system. But, there is a limit on how much we can increase the *confidence threshold*, otherwise we will reach a point in which the system can not detect and track anymore. As a rule of thumb, do not make any sudden/fast moves during tracking.

5.3.5. Frame Rate

During recognition (startup) the tracker system achieves only 9.2 fps, but when it is switched to detection mode it increases the frame rate up to 30 fps (stereo camera nominal frame rate). If instead of the stereo gear the input is a pre-recorded video (left/right stereo images), the tracker can go up to 100 fps.

5.3.6. Tracking Latency

As depicted in **Figure 3.1** the Tracker Latency (TrL) is composed by the Input Latency (InL) plus Processing Latency (PrL) plus Output Latency (OutL). The OutL is the time it takes the 3D position data (the tracker algorithm output) to reach the Ethernet port. Given the current technology (processor speed, TCP/IP stacks, OS, etc.) this is negligible (0 ms) compared to the others delays. As for the PrL, this is easy to calculate since it is the time it takes the whole tracker system algorithm to obtain the 3D head position once it receives the input images from the stereo cameras. Inserting (starting and stopping) the proper timers in the code allows me to easily calculate this delay. I ran the system for an hour and average the results obtaining a delay of 9 ms.. The standard deviation is almost zero because unless the OS decided to do something else, this number should not change. This is consistent with the 100 fps the system can achieve reading pre-recorded movie files in place of grabbing images from the stereo camera (again, reading movie files delays are negligible when using fast PCs with a lot of memory).

The most difficult part is to determine the Input Latency (InL) which is the time it takes the image captured by the camera sensor, to be converted to digital data, to travel through the various interfaces, to reach the PC (more interfaces and/or frame grabber board) and finally to be delivered to the input algorithm in charge of processes it (Tracker System). Even worse, most of the digital video cameras have internal buffers to make independent the sensor readout (frame rate) from their interface bandwidth (or speed).

My co-advisor, Daniel Sandin, devise a clever and simple procedure to approximate this latency (see **Figure 5.9**):

- Implement a program that can display a black or white square in the PC screen (screen desktop).
- Then, another piece of the program should be able to determine if the incoming image from the camera is black or white.
- Set the cameras pointing (and close) to the screen. The black or white square in front of the cameras should cover their field of view.
- Paint a black square and signal the cameras to grab one image (the stereo gear has the capability to run in 'continue' grabbing mode or 'single' mode in which they only capture an image when they receive an external command). At the exact same time start a timer.
- Wait in a loop until this 'black' image arrives into the computer (frame buffer). When it reaches the program, stop the counter and measure the time.
- Changes (re-paint) the square box to 'white'. Command the cameras to grab an image, start the timer, wait until the internal frame buffer changes to 'white', stop the timer and measure the time.
- Repeat switching to black and white and measuring the delay enough times to get a stable average and post the results.

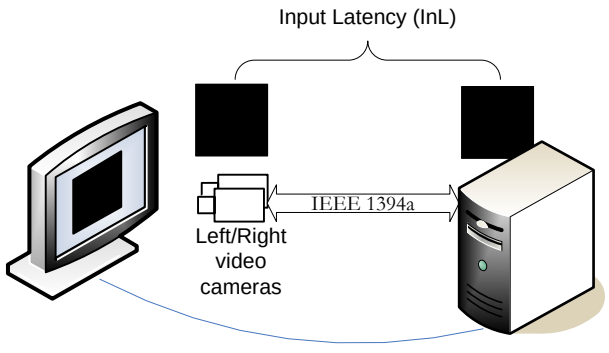

Figure 5.9: Latency measurement.

I left running this procedure for an hour, averaged the results, calculated the standard deviation and obtained 73.5 ms ± 10 ms. I assumed that the delay introduced by the 'grab a single image' command was much less that the input latency (grabbing the image and sending it to the PC).

As a conclusion the measured Tracker Latency (TrL) is around 82.5 ms ± 10 ms.

5.3.7. Static Jitter and Drift

Here I have to determine how stable the 3D head tracker position is if the face being tracked is not moving (it is in a fixed position). Also if the subject maintains its position for hours, does the output 3D position drift?

To test this I trained the Tracker with the 'dummy' head (**Figure 5.2**), stood the dummy in a stand in front of the cameras and let the system recognized and tracked the dummy head for an hour without moving it. I tried this experiment several times, every time re-training with dummy and tracking it at different positions (including different depths, far and close). The results were encouraging; after averaging the position data and calculating the maximum, minimum deviation I obtain a long term drift of zero pixels and a static jitter (or short term drift) of zero ± 2 mm in x, y coordinates and ± 3 mm in z.

5.3.8. Dynamic Jitter

This is difficult to measure automatically because when the user moves around, its head center moves too and there is not easy way to measure the drift or position variation from the center unless we know *a priori* the center path. The only way is to do it manually. For this purpose I used the same training videos and test video from previous sections. I trained the system for each individual user (this time only fifty) and run the test video in which this user should appear at least one time. Once the Tracker recognized the person, I modified the system to stop and begin frame by frame with a click of a mouse. Then, I visually followed the ellipse center (which should be centered at the user's head) and recorded any sudden jump or 'jitter' around this face center. I only wrote the maximum deviation among the three x, y and z coordinates

once per each running. I re-trained and repeated the test three times to observe the effect of training in the drift. **Table 5.3** summarizes the results.

Table 5.3: Maximum deviation per training session

	Training	Re-Training	Re-Training (again)
1	± 9 mm	± 2 mm	± 3 mm
2	± 5 mm	± 7 mm	± 5 mm
3	± 6 mm	± 4 mm	± 14 mm
4	± 8 mm	± 3 mm	± 9 mm
5	± 10 mm	± 12 mm	± 10 mm
6	± 2 mm	± 3 mm	± 7 mm
7	± 8 mm	± 9 mm	± 15 mm
8	± 5 mm	± 6 mm	± 4 mm
9	± 12 mm	± 5 mm	± 3 mm
10	± 15 mm	± 16 mm	± 8 mm
11	± 3 mm	± 13 mm	± 12 mm
12	± 16 mm	± 16 mm	± 4 mm
13	± 9 mm	± 13 mm	± 10 mm
14	± 14 mm	± 6 mm	± 4 mm
15	± 3 mm	± 13 mm	± 6 mm

The conclusion is that the dynamic jitter could go from ± 2 mm (best case) to ± 16 mm (worst case) and it depends definitively on the training.

5.3.9. Static Precision

The result of this test not only depends on the camera characteristics but mostly on the camera calibration program which is responsible for correcting its geometry and lens deficiencies.

I tested the static precision (tracker user is not moving) only around the sweet spot. I trained the Tracker with the dummy and centered it at the sweet spot using a stand. I also re-calibrated the system using the internal translation and rotation matrices so the center of the dummy's head position at the sweet spot will be the coordinate center (x=0, y=0, and z=0). With a ruler, I carefully moved in the x, y plane the head

center close to each border of the combine cameras' field of view, and measure the distances. I set the limits at ±40 cm in *x*, *y* plane and ±20 cm for *z*. I adjusted the internal scale matrix so the Tracker outputs the same 3D head position at each limit.

Once the system recognized the dummy and switched to tracker mode, I used a ruler to move the head center at 10 cm increments in *x,* parallel to the camera's baseline and maintaining *y* and *z* coordinates constant. I waited until the system output stable numbers, and then I recorded the Tracker 3D head position and compared it against the ruler. I repeated the procedure for *y* and *z*. **Table 5.4** summarizes the comparison results.

Table 5.4: Static precision. Measured vs. 3D head position Tracker ouput.

	Ruler Marks								
Measured *x, y, z*	-40 cm	-30 cm	-20 cm	-10 cm	0 cm	+10 cm	+20 cm	+30 cm	+40 cm
Tracker x	-40	-30.5	-20.6	-9.8	0.1	10.7	19.8	31	40.1
Tracker y=0	0.2	0.1	0.3	-0.1	0.1	0.1	0.2	0.4	0.2
Tracker z=0	0.3	-0.1	0.1	0.2	0	-0.1	-0.3	0.3	0.1
Tracker x=0	0.1	0.2	0.1	-0.1	0.1	0.2	0.1	0.3	0.1
Tracker y	-40.5	-30.3	-19.6	-9.3	0.1	10.8	19.1	29.3	40.2
Tracker z=0	0.1	0.2	-0.1	0.2	0.2	0.3	0.2	-0.3	0.1
Tracker x=0			-0.3	-0.2	0.1	0.2	0.1		
Tracker y=0			-0.2	0.1	-0.2	0.3	-0.2		
Tracker z			-20.5	-10.5	-0.1	11	19.6		

Looking at the **Table 5.4** we can conclude that the precision is about of ±1 cm in the *x*, *y* and *z* axis across a volume of ±40 cm in *x*, ±40 cm in *y* and ±20 cm in *z* and sampling every 10 cm.

5.3.10. Resolution

I measured the resolution using the previous **Static Precision** test. Each time I had to stop at any ruler mark to test the precision, before moving to the next mark I slightly moved the head in the *x*, *y* and *z*

directions. More specifically, with the ruler I moved the head until the tracker system output a different x, y or z position and then, measured this distance with the ruler. I repeated this for all the samples and calculated the average and standard deviation in x, y or z. The results were 4 mm ± 2 mm in x and y, and 3 mm ± 4 mm in z. These results were across a volume of ±40 cm in x, ±40 cm in y and ±20 cm in z. The bad results in z are possible due to the stereo correspondence algorithm which at the end is related to the quality of the NN training. The results in x and y are consistent with the theoretical values calculated in section **3.4**.

6. REAL-TIME CAMERA-BASED FACE DETECTION USING A MODIFIED LAMSTAR NEURAL NETWORK SYSTEM

6.1. Introduction

This chapter is going to present the original research which helped me to understand and gain experience in neural networks [25]. Although the results as a sole face detector were encouraging, in place of adding to this design the recognition part and continue to develop the tracker system, some drawbacks forced me to re-think the thesis and to re-design the neural networks (NN) architecture. Here follows the main problems with the original design:

- **Training Time**. The time to train these NN was too much (five minutes), definitely not very convenient in our VR environments. We want a new tracker user ready to use the system quickly. The final thesis can train a user in less than two minutes.
- **Maximum Frame Rate**. in this original design I could not made the detection frame rate reach more than 16 fps. With a proper (faster frame rate) stereo camera gear the already described tracker system can reach up 100 fps using the same PC hardware.
- **Use Color Information**. This system performs very well as a face detector but it use camera color information. This has two problems. First and more important skin color vary a lot with different types of illumination (tungsten, fluorescent, etc.) making more difficult the detection or recognition is this changes from the NN training to running the tracker. To control the illumination and be independent of this factor in the final design we decided to use infrared light which only produce gray-level images. Second, color cameras are more expensive, slower frame rate and with less resolution than the equivalent gray-level cameras (given the same CCD or CMOS image sensor they usually have to use more sensor pixels (depends of the color filter arrangement) per image pixel, also use more bandwidth to send color information, etc.).

6.2. Original Face Detector Description

What it is described here is a cost-effective real-time (640x480 at 16 Hz) face detector that will serve as the core of a video-based, tetherless 3D head position and orientation tracker system targeted for either auto-stereoscopic displays or projection-based virtual reality systems. It will be tested first using EVL's Access Grid Augmented/Autostereo Virtual Environment (AGAVE), a high-resolution autostereoscopic display consisting of tiled LCD displays driven by a PC cluster and fitted with a highly sensitive tracker system to track user's gaze and gestures without the use of head mounted or hand held tracking devices.

The attempt tracker system (**Figure 6.1**) will consist of two neural network-based face detectors working in parallel, each running through four distinct phases. They are: *pre-processing*, which includes input masking, image normalization, histogram equalization, and image sub-sampling; *detection*, where a modified LAMSTAR neural network takes the pre-processed image, scans for a face and outputs the coordinates of the corresponding box surrounding the face; *post-processing*, where facial feature extraction and stereo correspondence matching occurs to extract the 3D information; and the implementation of a *prediction module*, which is based on a neural network linear filter with a Tap Delay Line (TDL). The function of the prediction module is to inform the face detection modules where in the scene the face will likely be found, so as to avoid scanning the next whole frame for a face. If the face is not detected in the predicted position, the system will rescan the entire scene. The pre-processing and post-processing stages use computer vision techniques. This chapter addresses the pre-processing and detection phases.

Based on recent surveys, face detection approaches rely upon one or a combination of the following techniques: Feature-based, Image/View-based and Knowledge-based. The proposed face detector is based on a modified LAMSTAR neural network system along with a novel combination of the three techniques mentioned above.

At the input stage, after image normalization and equalization are achieved, the information is divided into sub-pattern categories, each representing a part of the raw image. Each sub-pattern is then fed to a neural network layer that is a Kohonen SOM (Self-Organizing Map) module. The outputs of all the neural network modules are interconnected by *correlation-links* which will be explain later, and can hence determine the presence of a face with enough redundancy to provide a high detection rate. The face detector is also rotationally and size invariant for front-view faces to a certain degree. This chapter will detail several aspects of the technique.

To meet real-time constraints (low latency and high frame rates), the algorithms are highly tuned for the new Intel 4 Family Processor micro-architecture, specifically its vector processor.

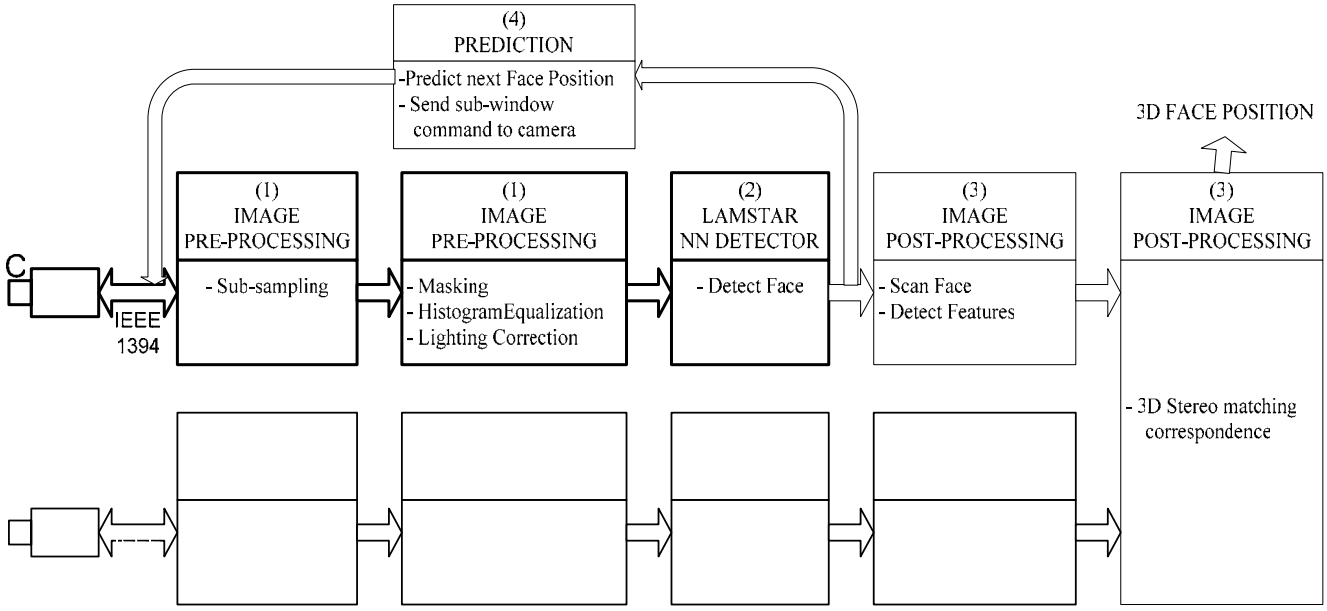

Figure 6.1: The attempted tracker system

6.3. Background

The core of the face detector is a system of artificial neural networks based on Graupe and Kordylewski's LArge Scale Memory STorage And Retrieval (LAMSTAR) network research, which targets large-scale memory storage and retrieval problems [43-46]. This model attempts to imitate, in a gross

manner, processes of the human central nervous system (CNS) concerning storage and retrieval of patterns, impressions, and sensed observations including processes of forgetting and recollection. It attempts to achieve this without contradicting findings from physiological and psychological observations, at least in an input/output manner. Furthermore, it attempts to do so in a computationally efficient manner using tools of neural networks, especially Self-Organizing-Map based (SOM) network modules, combined with statistical decision tools. Its design was guided by trying to find a mechanistic neural network-based model for very general storage and retrieval processes involved in, say, recalling the face of a previously encountered person. LAMSTAR follows a top-down approach based on neurophysiological observations that involve many parts of the CNS for the reconstruction of information. This general approach [45] is related to Minsky's idea [47] that the human brain consists of many agents, and a *knowledge link* is formed among them whenever the human memorizes an experience. When the knowledge link is subsequently activated, it reactivates the mental agents needed to recreate a mental state similar to the original. The LAMSTAR network employs this general philosophy of linkages between a large number of physically separate modules that represent concepts, such as time, location, patterns, etc., in an explicit algorithmic network. (Note: for consistency, this chapter uses the term *correlation links* instead of *knowledge links*.)

The LAMSTAR network has been successfully applied in fields of medicine (diagnosis) [43, 44, 46], engineering (automotive fault detection) and multimedia information systems [48]. Whereas the LAMSTAR design addresses large-scale memory retrieval problems, the face detection system is a comparatively small-scale retrieval problem. We are most interested in modeling our system design based on the LAMSTAR concept of using correlation links to retrieve stored information (faces).

Our real-time system design has two major constraints: the detection and subsequent tracking of a face must be performed in less than 33 ms to achieve 30 fps, and we only require the memory capacity needed to store one face. Hence, we limit our use of LAMSTAR concepts to processes of storage and retrieval, interpolation and extrapolation of input data, and the use of reward-based correlation-links

between modules to retrieve stored information. Since our coding approach deals only with pre-processed raw images, we use *input vector* or *input pattern* when referring to the sub-window image, *sub-vectors* or *sub-patterns* when referring to the post-processed segmented image, and *correlation-links* (*C-links*) when referring to the connection between modules.

6.4. System Overview

The modified LAMSTAR network is a self-trained/self-organized system of neural networks composed of Self-Organizing-Map (SOM) [42] modules (**Figure 6.2** and **Figure 6.3**), with added statistical decision tools. Interpolation takes place at these SOM levels, where the incomplete sub-pattern (minor variation or lack of data) is reconstructed as a result of the generalization property of the SOM network. The extrapolation, which is the primary function of associated memory models, is a result of correlation-based retrieval among SOM modules where one or more (but not all) sub-patterns are needed to reconstruct the entire pattern.

6.5. The Kohonen Self-Organizing-Map

Our system needs to map, in an unsupervised mode, whatever cluster of input data it is presented. Kohonen's Self-Organizing-Map (SOM) [42, 49] is best suited for this task because it is a self-organizing neural network that quickly integrates new input data into existing clusters by analyzing, then classifying the data using connection weights modified through different iterations.

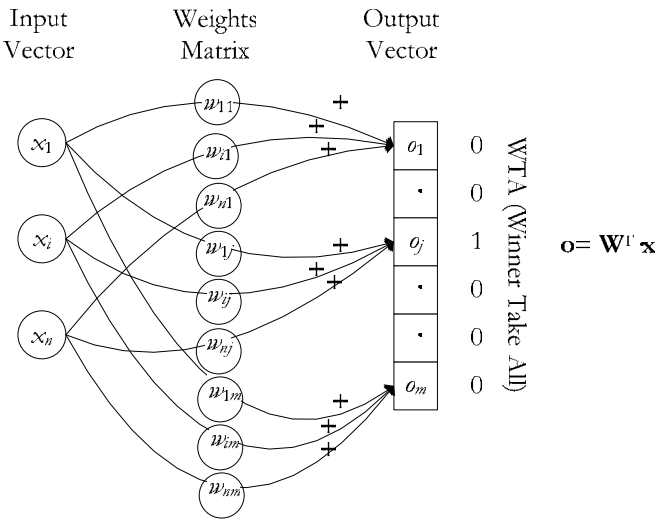

Figure 6.2: Kohonen Self-Organize-Map (SOM) network

Using the Lateral Inhibition technique, where distant neighbors are inhibited and closer ones reinforced, the Kohonen SOM is able to infer relationships and, as more input data is presented (in our case pre-processed video images), it creates the closest possible set of outputs for the given inputs. The output of a Kohonen SOM network (**Figure 6.2**) is a weighted sum of its inputs:

$$k_j = \sum_{i=0}^{m} w_{ij} x_i = \mathbf{w}_j^{\mathrm{T}} \mathbf{x}, \text{ where: } \mathbf{w}_j = \left[w_{1j} \ldots w_{mj} \right]^{\mathrm{T}}, \mathbf{x} = \left[x_1 \ldots x_m \right]^{\mathrm{T}} \tag{6.1}$$

After the equation weights (w_{ij}) are computed during the training phase, an unknown case is presented to the network. All outputs are found and the *maximum* output neuron is declared the winner, thus determining its class. The Kohonen neural network implements a "Winner-Take-All" (WTA) strategy [49], i.e., for any given input vector only one Kohonen neuron output is 1, whereas all others are 0:

$$k_{winner} > k_{j \neq winner} \Rightarrow k_{winner} = \sum_{i=0}^{m} w_{i\,winner} x_i = 1, k_{j \neq winner} = 0 \tag{6.2}$$

The neighborhood size parameter is used to model the effect of the Mexican hat or Gaussian hat function. Only input patterns whose neurons fall within the neighborhood size participate in training (i.e., learning) and weight changes (i.e., updates); all others are excluded. A good reference for these techniques and Kohonen SOM network implementations can be found at [50, 51].

6.6. SOM Modules: description, input and training

In this modified LAMSTAR network, each Kohonen SOM module represents a class of sub-patterns. The model assumes that the input patterns have been separated into sub-patterns before entering the SOM module (**Figure 6.3**). The network is thus organized to assign each neuron to a class of neurons (i.e., one SOM module) that best corresponds to the input sub-pattern. This SOM configuration yields very rapid matching with good error tolerance, and is capable of generalization.

Arrays of correlation links (C-links) horizontally connect the modules using coefficients determined by the statistical correlations between the various patterns considered. A coordinated activation of neurons between the various modules allows the network to recreate (interpolate) complex patterns and make associations (i.e., detect a face).

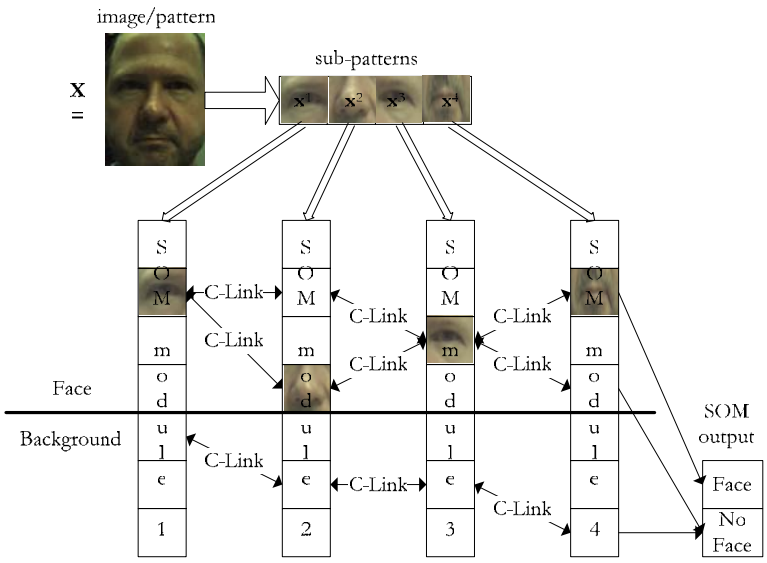

Figure 6.3: Simplified LAMSTAR diagram including the relationship between SOM models, SOM output layer and C-links

The input pattern/image is coded in terms of a real vector x given by:

$$\mathbf{x} = [\mathbf{x}^{1^T}, ..., \mathbf{x}^{i^T}, ..., \mathbf{x}^{m^T}]^T \tag{6.3}$$

The dimension of each *i'th* sub-vector varies according to the number of elements needed to describe a sub-pattern feature. To store data concerning the *i'th* category of the input pattern, each sub-pattern $\mathbf{x}^i$ is then channeled to the corresponding *i'th* SOM module. Thus, the dimension of a sub-pattern is equal to the amount of pixels representing a specific feature.

A winning neuron is determined for each input based on the similarity between the input vector $\mathbf{x}^i$ and weight vectors $\mathbf{w}^i$ (stored information). For a sub-pattern $\mathbf{x}^i$, the winning neuron is determined by the minimum Euclidean distance between $\mathbf{x}^i$ and $\mathbf{w}^i$:

$$\left\|\mathbf{x}^i - \mathbf{w}^i_{winner}\right\| = \min\left\|\mathbf{x}^i - \mathbf{w}^i_k\right\| \quad \forall\, k$$

where: $\mathbf{x}^i$ - input vector in $i'th$ SOM module
$winner$ - index of the winning neuron
$\mathbf{w}^i_{winner}$ - winner weight vector in $i'th$ SOM module
k - a number of neurons (stored patterns) in $i'th$ SOM module
$\|\square\|$ - Vector Euclidean distance: $\|\mathbf{x}-\mathbf{w}\| = \sum_{i=1}^{i=n}(w_i - x_i)^2$
where: n - dimension of sub-vectors **x** and **w**

(6.4)

The SOM module is a Winner-Take-All (WTA) network where only the neuron with the highest correlation between its input vector and its correspondence weight vector will have a non-zero output. The WTA feature also involves lateral inhibition such that each neuron has a single positive feedback onto itself and negative feedback connections to all other units.

$$\mathbf{o}^i_j = \begin{cases} 1, & for\ \left\|\mathbf{x}^i - \mathbf{w}^i_{winner}\right\| < \left\|\mathbf{x}^i - \mathbf{w}^i_j\right\| \ \forall\, winner \neq j \\ 0, & otherwise \end{cases}$$

where: $\mathbf{o}^i_j$ - output of neuron j in $i'th$ SOM module
$\mathbf{w}^i_{winner}$ - winning weight vector in $i'th$ SOM module
$winner$ - index of winning neuron in $i'th$ SOM module

(6.5)

The neuron with the smallest error determined by the previous two equations (6.4) and (6.5) is declared the winner and its weights $\mathbf{w}_{\text{winner}}$ are adjusted using the Hebbian learning law, which leads to an approximate solution:

$$\mathbf{w}^i_{winner}(t+1) = \mathbf{w}^i_{winner}(t) + \alpha \cdot (\mathbf{x}^i(t) - \mathbf{w}^i_{winner}(t))$$

where: $\mathbf{w}^i_{winner}(t+1)$ - the new value of winning weight vector $\mathbf{w}_{winner}$ in the $i'th$ SOM module
α - learning coefficient
$\mathbf{x}^i(t)$ - current input $i'th$ sub-vector
$\mathbf{w}^i_{winner}(t)$ - current wining weight vector

(6.6)

The adjustment in the LAMSTAR SOM module is weighted according to a pre-assigned Gaussian hat neighborhood function $\Delta(winner,j)$:

$$\mathbf{w}^i_j(t+1) = \mathbf{w}^i_j(t) + \Delta(winner, j) \cdot \alpha \cdot (\mathbf{x}^i(t) - \mathbf{w}^i_j(t))$$

where: $\mathbf{w}^i_j(t+1)$ - new weight of neighbor neuron j from winning neuron *winner*

$\Delta(j, winner)$ - neighborhood define as gaussian hat: $e^{\frac{|j-winner|^2}{2\sigma^2}}$ (6.7)

6.7. Training/Storage phase

The training of the SOM modules and the training of the C-links are performed in separate phases (see **Figure 6.4**):

- *Storage of new sub-patterns in SOM modules.* In the first case, given an input pattern **x** and for each $\mathbf{x}^i$ sub-pattern to be stored, the network inspects all weight vectors $\mathbf{w}^i$ in the *i'th* SOM module. If any previously stored pattern matches the input sub-pattern within a preset tolerance (*error* ε), the system updates the proper weights or creates a new pattern in the SOM module. It stores the input sub-pattern $\mathbf{x}^i$ as a new pattern, $\mathbf{x}^i = \mathbf{w}^i_j$, where index *j* is the first unused k^i_j neuron in *i'th* SOM module. If there are no more 'free' neurons, the system will fail, which means either the preset tolerance has to be increased to include more patterns in the same cluster of already stored patterns, or more neurons have to be added on the *i'th* SOM module.
- *Creation of C-links among SOM modules.* Individual neurons represent only a limited portion of the information input. For efficient storage and retrieval, only individual sub-patterns are stored in SOM modules, and correlations between these sub-patterns are stored in terms of creating/adjusting the C-links connecting the neurons in different SOM modules (**Figure 6.3**, **Figure 6.4** and **Figure 6.6**). This linkage distributes storage information horizontally between SOM modules. Consequently, in case of failure in one cell, one loses little information if a

pattern is composed of many sub-patterns. The neurons in one module of the network are connected to those in another by C-links. Correlation-link coefficient values C-link are determined by evaluation distance minimization as in equation (6.4) and (6.5) to determine winning neurons, where a win (successful match) activates a count-up element associated with each neuron and with it respective input-side link. During training/storage sessions, the values of C-links are modified according to the following simple rule (reward):

$$C_{k,l}^{i,j}(new) = \begin{cases} C_{k,l}^{i,j}(old) - \beta_{reward}(C_{k,l}^{i,j}(old) - C_{Max}), & \text{for } C_{k,l}^{i,j}(old) \neq 0 \\ 1 \quad , & \text{otherwise} \end{cases}$$

where: $C_{k,l}^{i,j}$ – correlation link between k'th neuron in i'th SOM module and l'th neuron in j'th SOM module (6.8)

β_{reward} – *reward* coefficient

If the correlation link between two sub-patterns already exists, namely, $C^{ij}_{k,l} > 0$ (a result from previous training), the formula of equation (6.8) updates (increases) the analyzed C-link. If there are no correlations ($C^{ij}_{k,l} = 0$), the system creates new C-link with initial value $C^{ij}_{k,l} = 1$.

6.8. Detection/Retrieval phase

The retrieval phase of the model (**Figure 6.4**) selects sub-patterns from the input pattern one at a time, than examines correlations with stored sub-patterns of a given class of sub-patterns in each SOM module. For example, one *i'th* SOM module could have previously stored noses, and will correlate any given input *i'th* sub-pattern and determine if there is a match or not (e.g., a nose or not).

The face is detected (or retrieved) by means of it C-links. Once all the winning neurons are determined, the system obtained all correlation-links coefficient values among all SOM modules. The output SOM layer (**Figure 6.2** and **Figure 6.6**), which all C-links are inter-connected, will determine the presence or not of a face.

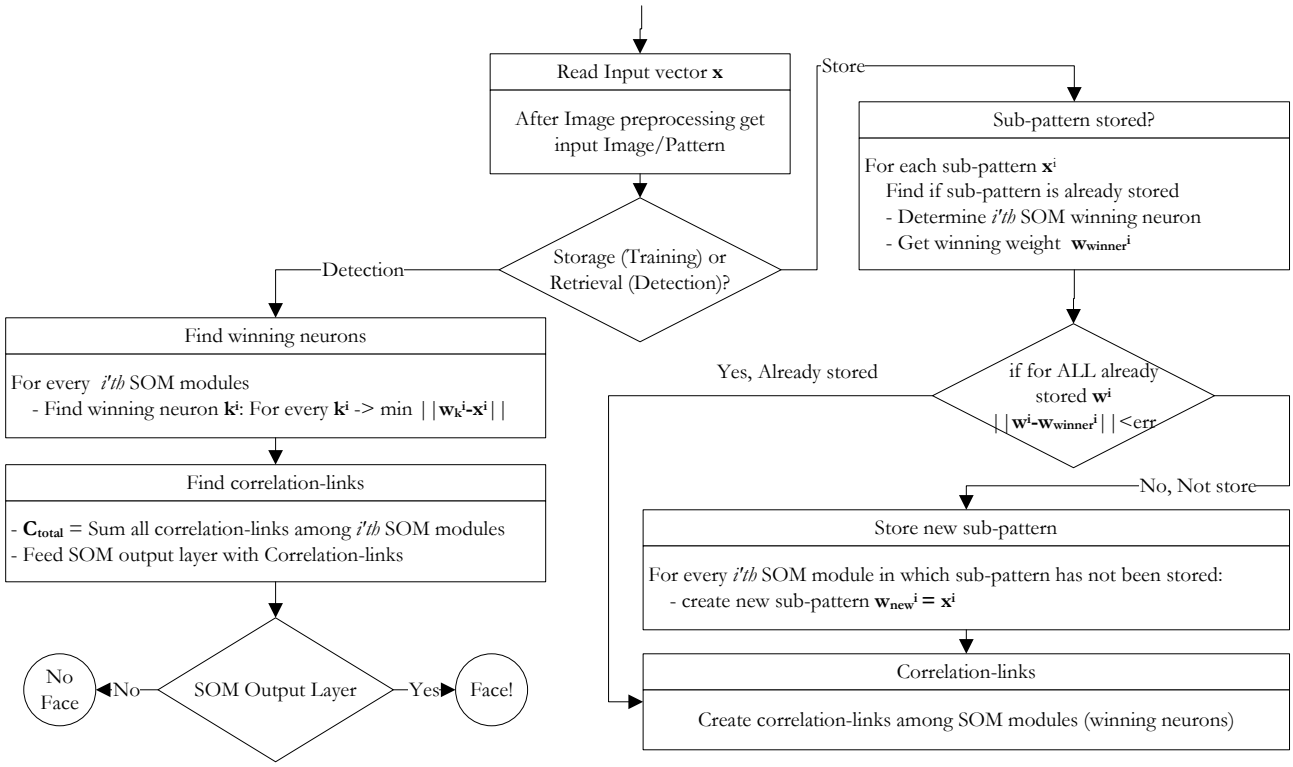

Figure 6.4: Flow diagram of the face training/storage and detection/retrieval

6.9. Methodology

Functioning as the head tracker in a virtual reality system, the face detector need only detect the face of the person intended for tracking. The system, therefore, accommodates each new user by retraining itself. Training is achieved through the following steps:

- With the face detector system set in training mode, the user faces a video camera and centers his face in a surrounding ellipse (**Figure 6.6**) shown on the face detector workstation screen. This is the training sub-window.
- A command initiates the training and the user slowly runs through a series of subtle head poses (chin left/right, chin up/down, tilting left/right) over the course of 10-15 seconds. This process allows the system to accommodate for the natural changes in the user's face orientation while using the virtual reality system.

- A command terminates the training stage. The user moves away from the camera's field-of-view. The user may opt to save the internal training parameters under a filename to later retrieve in lieu of retraining the system.
- A command initiates background training. The video camera takes a snapshot of the background scene and trains the neural network as non-face cases. Like in detection (**Figure 6.5**), it scans the whole image using a sub-sampling technique applied to every location of the image-scene to account for every possible face size [1]. This takes approximately five minutes. If the environment does not change, the user can save the background training information for later use.

This face detector first determines whether a given image sub-window belongs to the face category. Variability in the face images may increase the complexity of the decision boundary to distinguish faces from non-faces. To reduce this variability, the input image is pre-processed (**Figure 6.5**). During training and detecting phases, a pre-processing step [1] masks the background pixels from the face pixels with an ellipse mask (**Figure 6.6**). It then attempts to equalize the intensity values across the face pixels. A linear function is factored into the intensity values contained in the window, then subtracted out and corrected for extreme lighting conditions. Next, histogram equalization is applied to correct for different camera gains and to improve contrast. For each of these steps, the pre-processing is computed based on pixels inside the ellipse mask.

During the training process the system will train whatever face appears in the training sub-window in the image screen (**Figure 6.6**). Using an interpolation technique, the training sub-window is first reduced in size to match the internal face detection sub-window size, currently 64 x 85. This dimension is the *minimum* face width and height the system is able to detect and track.

During the detection process, it is still able to detect a trained face that appears larger than the trained sub-windows size, using a sub-sampling technique applied to every location of the image-scene. The input image is repeatedly sub-sampled by a factor of 1.2.

Once the image sub-window is pre-processed and reduced to a determined input vector size, the image is divided in non-parallel strips that cover local features that might important for face detection (**Figure 6.6**). Each non-parallel strip is a sub-vector (or sub-pattern) that is raw-fed into each SOM module. Currently there are ten SOM modules, each with a number of outputs ranging from five to thirty. There is also an output SOM layer inter-connected with all the other SOM modules. It only has two output neurons for the face and non-face case (**Figure 6.3** and **Figure 6.6**).

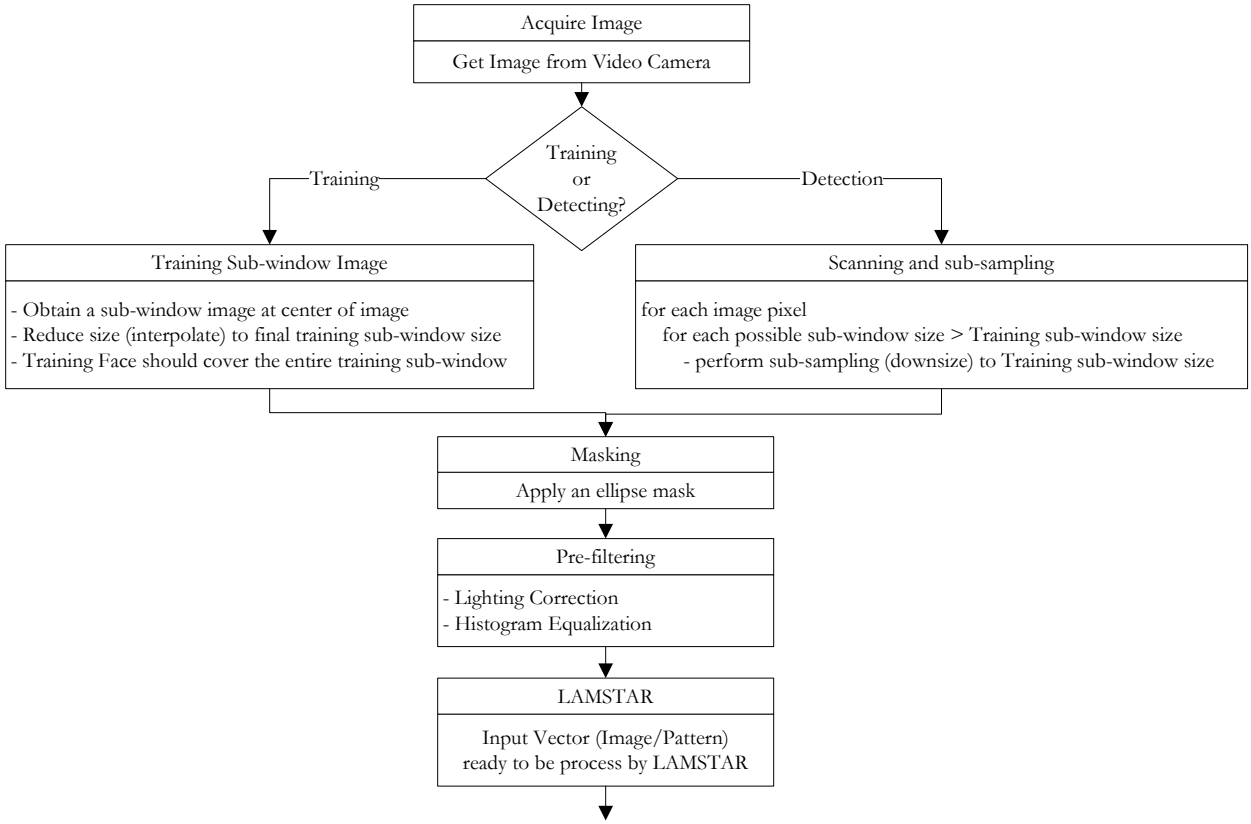

Figure 6.5: Flow diagram of the pre-processing stage

In order to properly use a Kohonen SOM network, it is recommended that all weights and input vectors are normalized, therefore during training and detection phases, sub-patterns and SOM weights are mapped to [-1, 1] and their lengths set to 1.

During the face and background training, sub-patterns are present and each SOM module is trained individually. All weights are initialized with random low values (< 0.2) and C-links coefficient with 0, which means no correlation at all. Once the SOM output is obtained using the WTA technique, the C-links between SOM modules and the output layer are established. During detection, the system selects the winners of each SOM module and through the C-links and weights the output SOM layer should be able to determine whether or not a face is present.

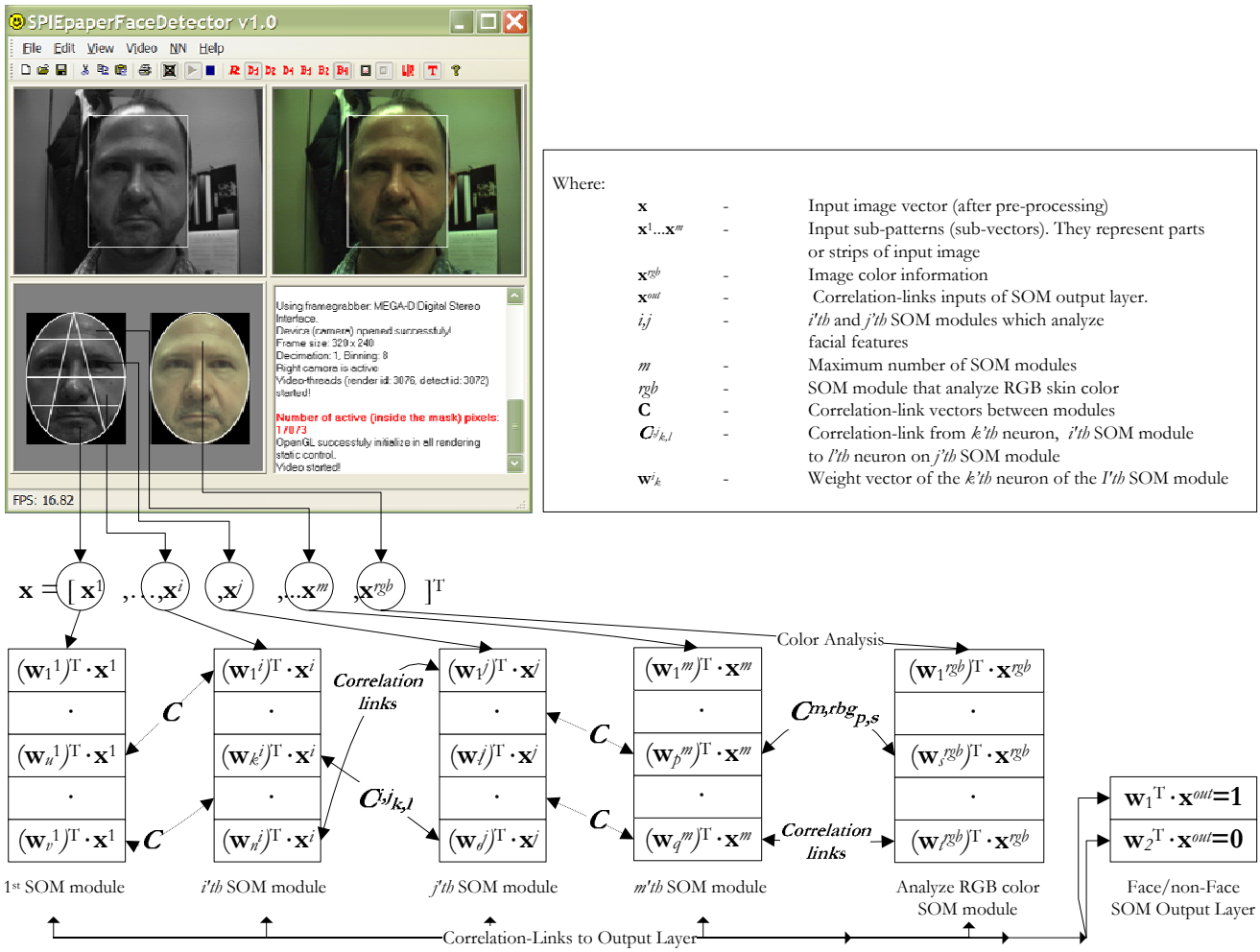

Figure 6.6: Snapshot of the face detector system including the relationship between SOM models

6.10. Implementation details

For video input, the face tracker system uses a *Videre Design MEGA-D* video camera. Its native image CMOS sensor resolution (*ZR32112*) is 1288 by 1032, and provides sub-sampling by binning (averaging) or decimation (removing) through its library developed by *SRI International*. It connects to the host using an IEEE 1394 at 400Mbps. An important and vital feature of this digital camera is its sub-window capabilities. Once a face is found in the scene, the face detector prediction module can send (not yet implemented) a sub-window command to the video camera to confine the next scan to the portion of

scene where the face is found. This substantially might reduce the bandwidth used between the camera and the host computer and increases the throughput of the face detector.

The system runs on a Dell 530MP with an *Intel® Dual Pentium IV Xeon* @1.7GHz and 768MB RAMBUS.

The pre-processing is developing using the *Intel® Integrated Performance Primitive* (IPP) library and the vector products involve in the internal SOM programming the *Intel® C++ Class Libraries for SIMD Operations*. The compiler is the *Intel® C++ Compiler* v7. The analyzer tool is *Intel® VTune™ Performance Analyzer 6.1*. All libraries, classes and tools are optimizing for Pentium 4 and Xeon™ processor.

6.11. Conclusions And Future Research

This research is intended to serve as the core of a video-based, tetherless 3D head position and orientation tracker system targeted for either auto-stereoscopic or projection-based virtual-reality systems.

Although still in early stages of development, this face-detection approach is showing promising results. We have successfully trained the system to detect a single face with it background in less than five minutes. Before the prediction model is enabled, even in a controlled environment where lighting and background remain constant, it can take from 0.5-16 fps to detect a face. The slow rate of 0.5 fps is due to the location of the face in the scene and the necessary successive image sub-sampling at each pixel position in order to detect arbitrary face sizes. Once the prediction model is enabled, the system achieves a detection rate of 16 fps.

This system is also able to locate and track a face at varying distances from the video camera, with the size of the face ranging from 640 x 480 pixels (maximum full frame) to 64 x 85 pixels (minimum due to internal face detection sub-window size).

A simple linear extrapolation prediction module localizes the scanning area to where the face is likely to appear in the scene. It takes the position velocity between the last two frames and uses it to predict the subsequent frame. This scheme improves system performance from 0.5 fps to 16 fps. However, when the system fails to detect a face, the system has to rescan the entire scene.

The performance of face detection and tracking is fairly robust for up-right frontal faces, achieving 90% of *true positive* detection rate (i.e., the face is a face). Once the system detects and locks onto a face, it only fails if the person's head rotates or tilts too much (lack of proper face training) or makes sudden movements (lack of good prediction and/or distorted face cause by video smearing). Sometimes, the system also fails with slight combinations of rotation and tilt; this could be due to either a lack of thorough training on the part of the user or a problem in the system's network design.

During the early stages of system development, when the system was only trained for face detection in controlled environments, the system had an average of 50% *false positive* (i.e., the face was not a face). After adding background training (non-face cases), the system improved and *false positive* detection went to 0%.

Tests are inconclusive in an uncontrolled environment, where more people can be present and the system might lock in a face that is not trained user's face.

Future work includes:

- Create a user information database. Each time the system learns to identify a new user face, a new record with weights and coefficients can be added and saved for future use.
- Refine the sub-pattern and SOM module arrangements to achieve better detection and faster system response.

- Add additional detectors, each one that can be independently trained, to create redundancy and therefore better reliability and accuracy.
- Stress the system in an uncontrolled environment and try to reach 30 fps (with prediction enabled).
- Continue developing the 3D head position and orientation tracker system.
- Currently, when the prediction module fails, the system has to re-scan the entire image, lowering performance to 0.5 fps. New techniques are being developed that use the coordinates of the last box surrounding the face (the face sub-window), increase the box size and then re-scan that area. The box size is incrementally increased and re-scanned until the face is found.
- Implement a better prediction module to avoid decreasing the frame rate. Currently under development is a more sophisticated prediction module based on a neural network linear filter with a Tap Delay Line (TDL).
- Decrease the bottleneck caused by image sub-sampling at each pixel position by developing a better algorithm for detecting arbitrary face sizes.
- Try different video camera with higher sensitivity, faster shutter speed and therefore less smearing.

This face-detection system is being designed to detect specific faces, not arbitrary faces; therefore, it differs from most of the general face-detection research in the literature. It is designed for use in a virtual-reality system with known users as a tetherless replacement for a magnetic head tracker. It avoids generalization, and doesn't lock on unknown persons who enter a scene, enabling the user to reliably control the virtual environment. Future research will, however, compare this specific algorithm with more general efforts, to determine the benefits and bottlenecks of this approach.

7. CONCLUSION AND FUTURE WORK

7.1. Conclusions

This thesis demonstrates the effectiveness of recognizing and tracking people using a view based approach with neural networks and its viability to replace current technology used by commercial trackers used in our VR environments. After evaluating the technology used by currently available commercial trackers in **Chapter 2**, this thesis successfully achieves most of the goals proposed in **Chapter 3**.

Chapter 4 shows a novel idea to preprocess the input video images to remove the variation caused by lighting and camera parameters. Unlike current local preprocessing techniques, we apply a global preprocessing based on shading correction. The results are similar to local preprocessing but with enormous benefit of speed which is translated in faster frame rates.

Chapter 5 is the core of this thesis. It describes all the different NN the system counts and their goals. There are three NN per left and right video input, each one specifically tuned for one purpose: face recognition, face detection, and head tracking. This chapter also explains a very efficient algorithm capable of training the NN with a user in less than two minutes.

After successfully been trained, the Tracker has a low initial recognition error of 6.6%, and 0% when it switched to detection/tracking mode. More impressive is the frame rate of 100 fps @ 320x240 it can achieve with a standard PC. Resolution and precision are within proposed margins.

The only immediate drawback which affects the jittering and tracker error as a recognizer, detector, and tracker, is the training process. Instead of training the system *with* the user, it seems like the system is training the user, teaching him or her how to move the head during training in order to obtain a better tracking performance (less jittering and recognition error). Most of the time, it is not necessary to re-train with the same user, but if the system is not performing well, re-training once more is generally enough.

The system has been shown during several important events and one IEEE VR conference with a total of more than sixty persons successfully using the Tracker. The good results obtained here encourage me to continue developing more robust and faster NN based trackers in order to use them on the next generation of VR display devices on which this laboratory is currently researching.

7.2. Future Work

- **Prediction**. Not only predict the next area of the video image to look for the head to speed up the tracker frame rate, but also in order to lower the tracker's latency predict the next head position in the next video frame.
- **Training of body parts or objects**. Research the feasibility of training the NN to detect and track other body parts (especially hands) or objects.
- **Background extraction from the face**. Once it is recognized, the left and right user face can be extracted from the background and use it for avatar projects [52].
- **Improve training methodology**. We have to yet establish a "good" methodology to train a new user, specifically how he or she has to move his/her head to obtain good training data and therefore good tracking performance. Sometimes the system is able to track the new user but the head position is not smooth enough and jumps around the real head center, or sometimes the system has difficulty recognizing the person. In these occasions we have to ask the user to sit down and perform the training again in order to get a better performance.
- **Create a user information database**. Each time the system learns to identify a new user face, a new record with NN weights and coefficients can be added and saved for future use. Then if the same user wants to use the tracker another day, he or she can query and recover the training data and use it without having to re-train the system again.
- **Multiple face detection, recognition and tracking**. Incorporating a user information database and given the increasing speed of PCs, examining this possibility is not far fetched. Multiple face recognition

can be very useful for customizing VR systems based on different users. For example a person walks in, the tracker recognizes him or her, and then using this information the host system run a specific VR application.

- **Move to next generation of Varrier™ autostereoscopic display**. During the evaluation of the thesis (see section **1.7**) I used the first version of the Varrier™ display (see **Figure 1.2** and **Figure 1.3**). I use this VR device solely as a test bed. The next goal is to implement this tracking technology into the next generation of bigger and more immersive Varrier™ display (see **Figure 1.4**).
- **Increment tracking frame rate**. Given current technology in processor speed, display frame rates, etc. is possible to experience VR systems in a more realistic and smooth ways than before. For example, the CAVE® runs typically at 120 Hz and a minimum of 60 Hz (30 fps per eye). MS-Windows screen displays have a minimum of 65 Hz. Varrier™ displays run now at 30 Hz, but we are targeting a minimum of 60 Hz. Since the algorithms I developed are able to track at 100 fps or more, to deliver this information at higher speed, I have to upgrade and test a new stereo camera gear capable of performing higher frame rates at the same or more video image resolution. There is another immediate solution but I have not test it yet: some cameras, including the ones I am using now, can perform "area-scan". This means that sending certain commands to the cameras from the application can force the camera's image sensor to read only a sub-window of the video image. This will effectively increase the frame rate due to the less use of bandwidth. Since I already have implemented a prediction module that tells the NN algorithms which image area to look next, I can use this information to feedback to the stereo cameras and consequently increase its frame rate.
- **Two viewpoints (one stereo camera) vs. multiple viewpoints (several cameras)**. It would be difficult if not impossible for the combined field-of-view of only two cameras to cover the space-volume in which a user can move and head-track using the next generation of Varrier™ displays (see **Figure 1.4**). Especially given the height of the stacks of tile LCD displays, it might be necessary not only

to install several cameras at the top, but also at the bottom so the user can look up or down and still be face-recognized. This, of course, implies that I have to develop a more sophisticated multiple view correspondence algorithm to extract the precise 3D head position, and in conjunction with algorithms for multiple face recognition and detection working as a unit.

- **Multiple *Recognizing* NN per left/right channel**. Currently to meet real-time restrictions, during recognition I am only arbitrating between left and right *Recognizing* NN. A much better solution would be to have several left and right *Recognizing NN*s in parallel and arbitrate among them. This would give us a greater recognition performance. Incoming faster PCs will allow me in the near future to implement and test this idea.

CITED LITERATURE

[1] H. A. Rowley, "Neural Network-Based Face Detection," Doctor of Philosophy's thesis in Computer Science, Carnegie Mellon University, Pittsburgh, CMU-CS-99-117, 1999.

[2] K.-K. Sung, "Learning and Example Selection for Object and Pattern Detection," Doctor of Philosophy's thesis in Electrical Engineering and Computer Science, Massachusetts Institute of Technology, A.I.T.R. No. 1572, 1995.

[3] M.-H. Yang, D. J. Kriegman, and N. Ahuja, "Detecting Faces in Images: A Survey," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 24, 2002.

[4] E. Hjelmås and B. K. Low, "Face Detection: A Survey," *Computer Vision and Image Understanding*, vol. 83, pp. 236-274, 2001.

[5] T. Fromherz, P. Stucki, and M. Bichsel, "A Survey of Face Recognition," University of Zurich MML Technical Report No 97.01, 1997.

[6] R. Chellappa, C. Wilson, and A. Sirohey, "Human and machine recognition of faces: A survey," *Proceedings IEEE*, vol. 83, pp. 705-740, 1995.

[7] J. Isdale, "What Is Virtual Reality? A Homebrew Introduction and Information Resource List," Version 2.1, October 8th 1993.

[8] C. Cruz-Neira, D. J. Sandin, T. A. DeFanti, R. Kenyon, and J. C. Hart, "The CAVE, Audio Visual Experience Automatic Virtual Environment," *Communications of the ACM*, vol. 35, pp. 64-72, 1992.

[9] C. Cruz-Neira, D. J. Sandin, and T. A. DeFanti, "Surround-Screen Projection-Based Virtual Reality: The Design and Implementation of the CAVE," *ACM Computer Graphics*, vol. 27, pp. 135-142, 1993.

[10] M. Czernuszenko, D. Pape, D. Sandin, T. DeFanti, G. L. Dawe, and M. D. Brown, "The ImmersaDesk and Infinity Wall Projection-Based Virtual Reality Displays," *Computer Graphics*, vol. 31, pp. 46-49, 1997.

[11] A. Johnson, D. Sandin, G. Dawe, T. A. DeFanti, D. Pape, Z. Qiu, S. Thongrong, and D. Plepys, "Developing the PARIS: Using the CAVE to Prototype a New VR Display," presented at Proceedings of the 4th International Immersive Projection Technology Workshop (CDROM), Ames, IA, 2000.

[12] G. Bishop, H. Fuchs, and e. al., "Research directions in virtual environments: report of an NSF invitational workshop," University of North Carolina at Chapel Hill, Chapel Hill, NC March 23-24 1992.

[13] D. Sandin, T. Margolis, G. Dawe, J. Leigh, and T. DeFanti, "The Varrier Auto-Stereographic Display," presented at SPIE, San Jose, California, 2001.

[14] D. Sandin, E. Sandor, W. Cunnally, M. Resch, T. DeFanti, and M. Brown, "Computer-Generated Barrier-Strip Autostereography," presented at Proceedings of SPIE, Three-Dimensional Visualization and Display Technologies, 1989.

[15] T. Ogi, T. Yamada, K. Tamagawa, and M. Hirose, "Video Avatar Communication in a Networked Virtual Environment," presented at INET 2000 Proceedings, Emerging Multimedia, Pacifico Yokohama Conference Center, Yokohama, Japan, 2000.

[16] S. Meyers, D. Sandin, W. Cunnally, E. Sandor, and T. DeFanti, "New Advances in Computer-Generated Barrier-Strip Autostereography," presented at Proceedings of SPIE, Stereoscopic Displays and Applications, 1990.

[17] M. A. Nixon, B. C. McCallum, W. R. Fright, and N. B. Price, "The Effects of Metals and Interfering Fields on Electromagnetic Trackers," *Presence*, vol. 7, pp. 204-218, 1998.

[18] S. Bryson, "Measurement and calibration of static distortion of position data from 3D tracker," Applied Research Branch, Numerical Aerodynamics Simulation Division, NASA Ames Research Center, Moffett Field, CA 94035 RNR Technical Report RNR-92-011, April 8 1992.

[19] R. L. Holloway, "Registration Error Analysis for Augmented Reality," *Presence: Teleoperators and Virtual Environments*, vol. 6, pp. 413-432, 1997.

[20] M. Ghazisaedy, D. Adamczyk, D. Sandin, R. Kenyon, and T. DeFanti, "UltraSonic Calibration of a Magnetic Tracker in a Virtual Reality Space," presented at Proceedings of the IEEE Virtual Reality Annual International Symposium (VRAIS '95), Raleigh, NC, 1995.

[21] S. Bryson and S. S. Fisher, "Defining, Modeling, and Measuring System Lag in Virtual Environments," presented at Proceedings SPIE, Stereoscopic Displays and Applications I, 1990.

[22] D. He, F. Liu, D. Pape, G. Dawe, and D. Sandin, "Video-Based Measurement of System Latency," presented at Proceedings of the Fourth International Immersive Projection Technology Workshop 2000, Ames, IA, 2000.

[23] S. R. Ellis, B. D. Adelstein, S. Baumeler, G. J. Jense, and R. H. Jacoby, "Sensor Spatial Distortion, Visual Latency, and Update Rate Effects on 3D Tracking in Virtual Environments," presented at Proceedings of the IEEE Virtual Reality, NASA Ames Research Center, Houston, Texas, 1999.

[24] P. A. Laplante and A. D. Stoyenko, *Real-Time Imaging, Theory, Techniques, and Applications*, First ed: IEEE Press, 1996.

[25] J. Girado, D. Sandin, T. A. DeFanti, and L. Wolf, "Real-time Camera-based Face Detection using a Modified LAMSTAR Neural Network System," presented at Proceedings of IS&T/SPIE's 15th Annual Symposium Electronic Imaging 2003, Applications of Artificial Neural Networks in Image Processing VIII, Santa Clara, California, USA, 2003.

[26] R. Newman, Y. Matsumoto, S. Rougeaux, and A. Zelinsky, "Real-Time Stereo Tracking for Head Pose and Gaze Estimation," presented at Fourth IEEE International Conference on Automatic Face and Gesture Recognition 2000, Grenoble, France, 2000.

[27] Y. Matsumoto and A. Zelinsky, "An Algorithm for Real-time Stereo Vision Implementation of Head Pose and Gaze Direction Measurement," presented at Proceedings of IEEE Fourth International Conference on Face and Gesture Recognition (FG'2000), 2000.

[28] J. Woodfill and B. V. Herzen, "Real-Time Stereo Vision on the PARTS Reconfigurable Computer," presented at Proceedings IEEE Symposium on Field-Programmable Custom Computing Machines, Napa, 1997.

[29] H. A. Rowley, S. Baluja, and T. Kanade, "Rotation Invariant Neural Network-Based Face Detection," Carnegie Mellon University and Justsystem Pittsburgh Research Center, Pittsburgh, PA 15213 CMU-CS-97-201, December 1997 1997.

[30] H. A. Rowley, S. Baluja, and T. Kanade, "Neural Network-Based Face Detection," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 1998.

[31] K.-K. Sung and T. Poggio, "Example-based Learning for View-based Human Face Detection," Massachusetts Institute of Technology and Artificial Intelligence Laboratory Center for Biological and Computational Learning A.I. Memo No. 1521, C.B.C.L. Paper No. 112, December 1994 1994.

[32] K.-K. Sung and T. Poggio, "Example-Based Learning for View-Based Human Face Detection," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 20, pp. 39-51, 1998.

[33] R. Jain, R. Kasturi, and B. G. Schunck, *Machine Vision*, First ed: McGraw-Hill, 1995.

[34] P. Heckbert, *Graphics Gems IV*, First ed: Academic Press, Inc., 1994.

[35] R. R. Anderson, J. Hu, and J. A. Parrish, "Optical radiation transfer in the human skin and applications in in vivo remittance spectroscopy," in *Bioengineering and the Skin*, R. Marks and P. A. Payne, Eds., First ed. Lancaster: MTP Press Limited, 1981, pp. 253-265.

[36] M. Z. Brown, D. Burschka, and G. D. Hager, "Advances in computational stereo," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 25, pp. 993- 1008, 2003.

[37] D. Maio and D. Maltoni, "Real-time face location on gray-scale static images," *Pattern Recognition*, vol. 33, pp. 1525-1539, 2000.

[38] M. J. Donahue and S. I. Rokhlin, "On the Use of Level Curves in Image Analysis," *CVGIP: Imge Understanding*, vol. 57, pp. 185-203, 1993.

[39] S. Birchfield, "An Elliptical Head Tracker," presented at 31st Asilomar Conference on Signals, Systems, and Computers, 1997.

[40] S. Birchfield, "Elliptical Head Tracking Using Intensity Gradients and Color Histograms," presented at IEEE Conference on Computer Vision and Pattern Recognition, Santa Barbara, California, 1998.

[41] T. Kohonen, "The Self-Organizing Map," *Proceedings of the IEEE*, vol. 78, 1990.

[42] T. Kohonen, *Self-Organization and Associative Memory*, Third ed. New York: Springer-Verlag, 2001.

[43] D. Graupe, *Principles of Artificial Neural Networks*, vol. 3, First ed. Singapore: World Scientific Publishing Co. Pte. Ltd., 1997.

[44] D. Graupe and H. Kordylewski, "A Large Memory Storage and Retrieval Neural Network for Adaptive Retrieval and Diagnosis," *International Journal of Software Engineering and Knowledge Engineering*, vol. 8, pp. 115-138, 1998.

[45] D. Graupe and W. J. Lynn, "Some Aspects Regarding Mechanistic Modeling of Recognition and Memory," *Cybernetica*, vol. 12, pp. 119-141, 1969.

[46] H. Kordylewski, "A Large Memory Storage and Retrieval Neural Network for Medical and Engineering Diagnosis/Fault Detection," Doctor of Philosophy's thesis in Electrical Engineering and Computer Science, University of Illinois at Chicago, Chicago, TK-99999-K629, 1998.

[47] M. L. Minsky, "K-Lines: A Theory of Memory," *Cognitive Science*, vol. 4, pp. 117-133, 1980.

[48] S. Chang, D. Graupe, and K. Hasegawa, "An Active Multimedia Information System for Information Retrieval, Discovery and Fusion," *International Journal of Software Engineering and Knowledge Engineering*, vol. 8, pp. 139-160, 1998.

[49] S. Kaski and T. Kohonen, "Winner-Take-All Networks for Physiological Models of Competitive Learning," *Neural Networks*, vol. 7, pp. 973-984, 1994.

[50] T. Masters, *Practical Neural Network Recipes in C++*. San Diego: Academic Press, 1993.

[51] V. Rao and H. Rao, *C++ Neural Networks and Fuzzy Logic*, Second ed. New York: MIS Press, 1995.

[52] J. Insley, D. Sandin, and T. DeFanti, "Using Video to Create Avatars in Virtual Reality," presented at Visual Proceedings of the 1997 SIGGRAPH Conference, Los Angeles, CA, 1997.

VITA

NAME	Javier Ignacio Girado	
EDUCATION	B. Electronics Engineering, ITBA University, Buenos Aires, Argentina	1982
	M. Electronics Engineering, ITBA University, Buenos Aires, Argentina	1984
	Ph.D., Computer Science, University of Illinois at Chicago, IL, USA	2004
FELLOWSHIPS	INTI National Institute of Industrial Technology, Buenos Aires, Argentina	
	Microprocessor Department	1984-1990
	Electronic Division	1983-1984
	Acoustic Division	1982-1983
HONORS	Member of the "Electronic Equipments Development Group", ITBA University, Buenos Aires, Argentina	1982-1983
RESEARCH EXPERIENCE	Research Assistant, Electronic Visualization Laboratory, University of Illinois at Chicago, IL, USA	Jan 1996-Aug 2004
	Research Assistant, INTI, Buenos Aires, Argentina	
	Microprocessor Department	1984-1990
	Electronic Division	1983-1984
	Acoustic Division	1982-1983
	Research Assistant, EE Department, Electronic Circuit Design course, ITBA University, Buenos Aires, Argentina	Aug 1983-Dec 1983
	Research Assistant, EE Department, "Electronic Equipments Development Group", ITBA University, Buenos Aires, Argentina	Dec 1982-Feb 1983

TEACHING EXPERIENCE	Research Assistant, *"Independent Study: From C to C++' course"*, Electronic Visualization Laboratory, University of Illinois at Chicago, IL, USA	Jan 1998-May 1998
	Lecturer, *"'C' Programming Language"* and *"Electronic Circuit Design"*, ITBA University, EE Department, Buenos Aires, Argentina	Mar 1985-Jul 1987
	Lecturer, *"Post-Graduate course: Microprocessor for Electronic Engineers"*, UTN University, EE Department, Buenos Aires, Argentina	Aug 1985-Dec 1985
	Teaching Assistant, *"Circuit Theory and Synthesis"* and *"Communication Systems I"*, ITBA University, EE Department, Buenos Aires, Argentina	1983-1984
	Research Assistant, *"Fast Fourier Transform and Digital Filters, Implementation using Microprocessors"* and *"SPICE 2G.6 (circuit simulation program) Seminar: Programming, use and applications in design"*, INTI, Electronic Division and Microprocessor Department, Buenos Aires, Argentina	1983, 1985
	Freelance Lecturer, Organized and conducted the following Seminars, Training Courses, Workshops and Help-desks: t*rained programmers in Object Oriented Technology (OOT) using IBM VisualAge and SmallTalk over IBM AS400 and IBM OS2/2 PCs; Introduction and Advance Lotus 1-2-3 v2.0/v3.0, including Advance Macro programming; Clipper; FoxBase; dBASE II/III/IV programming, including introduction to Relational Databases; RPGI;, Quick Basic; C and Turbo Pascal programming; PC-DOS OS, including PC Support and System Calls, Viruses and Anti-Viruses; MultiMate; Open Access; Perfect Picture; Lotus Freelance, Lotus Manuscript; Harvard Graphics; Diagraph; StatGraphics; WordPerfect*	
	Merck & Co., Buenos Aires, Argentina branch	Oct 1985-Jun 1995
	IBM, Buenos Aires, Argentina branch	May 1994-May 1995
	Raychem International Corporation, Argentina branch	Aug 1989-Dec 1990
	Proartel S. A. (TV Station), Buenos Aires, Argentina	Aug 1986-Feb 1987
	Atanor S. A., Buenos Aires, Argentina	Aug 1986-Mar 1987
	Banco Provincia, Buenos Aires, Argentina	Jan 1987-Jul 1987

PROFESSIONAL EXPERIENCE	Freelance Consultant, Project Leader, System Designer and Programmer, Network and System Administrator, Programmer, Technical Support	
	Quantum Market Research, Inc., Oakland, CA, USA	Dec. 2003
	Keytech S.A., Buenos Aires, Argentina, partner with Compression Labs, Inc, San Jose, CA, USA	Mar 1993-Jun 1995
	Hasar S.A., Buenos Aires, Argentina	Jan 1993-Aug 1995
	Broadcast Systems, Madrid, Spain	Jan 1991-Mar 1991
	Merck & Co., Buenos Aires, Argentina branch	Oct 1985-Jun 1995
	Merck & Co., Sidney, Australia branch	Jan 1991
	Merck & Co., Oakland, New Zealand branch	Jan 1990
	Merck & Co., Puerto Rico, USA branch	Jan 1989
	Merck & Co., NJ, USA Headquarters	Jan 1988
	IBM, Buenos Aires, Argentina branch	May 1994-May 1995
	Raychem International Corporation, Argentina branch	Aug 1989-Dec 1990
	Proartel S. A., Buenos Aires, Argentina,	Aug 1986-Feb. 1987
	Atanor S. A., Buenos Aires, Argentina	Aug 1986-Mar 1987
	Akapol S.A., Buenos Aires, Argentina	Mar 1987-Sep 1987
	AGFA-GEVAERT ARGENTINA S.A., branch of Agfa NV, Belgium	Oct 1988

COPYRIGHTABLE INTELLECTUAL PROPERTIES	Javier I. Girado, Daniel Sandin, "A two-way videoconferencing program with audio capability" Provisional Patent Application, University of Illinois at Chicago, IL, USA	Filed in January 2000
	Javier I. Girado, Felipe M. Girado, "Girado's Conditional Access Management System", TXu-726-267 COPYRIGHT HISTORY MONOGRAPH FILE (COHM) database, Computer program & screen displays	Date of Creation 1994 Date of Reg. 17 Nov95

PUBLICATIONS

J. Girado, D. Sandin, et al.,*"Real-time Camera-based Face Detection using a Modified LAMSTAR Neural Network System"*, Proceedings of IS&T/SPIE's 15th Annual Symposium Electronic Imaging 2003, Applications of Artificial Neural Networks in Image Processing VIII, Santa Clara, California, USA, SPIE

J. Leigh, J. Girado, et al., *"TeraVision : a Platform and Software Independent Solution for Real Time Display Distribution in Advanced Collaborative Environments"*, Access Grid Retreat 2002 Proceedings, La Jolla, CA, USA

J. Leigh, J. Girado, et al., *"AccessBot: An Enabling Technology for Telepresence"*, CDROM Proceedings of INET 2000, The 10th Annual Internet Society Conference, Yokahama, Japan

WORKSHOPS

EVL doctoral student Javier Girado Presented *"Global Tele-Immersion: Working in CyberSpace,"* as a keynote speaker at the *"Science and Technology and Companies: A Vision for the 21st Century"* workshop and plenary sessions, held in Barcelona, Spain. The event was sponsored by Barcelona University (UB) and the Technical University of Catalunya (UPC), and reflected the interest of both institutions to foster joint research projects and a technology exchange, and to promote the new cientific and Technological Park Barcelona 2000 (PCTB2000), Jan 20-21, 2000

PRESENTATIONS

Varrier™ auto-stereo system driven by a Real-Time 3D Head Position Tracker System with stereo cameras using a Face Detection and Recognition Neural Network (Ph.D. thesis). The event was an open house hosted for the IEEE VR 2004 conference at the Electronic Visualization Laboratory (EVL), University of Illinois at Chicago (UIC), March 29, 2004

VOLUNTEER WORK

Team member of GraphicsNet on-site support at SIGGRAPH 2000, New Orleans, Louisiana, August 2000

ACKNOWLEDGMENTS

Joseph A. Insley, Daniel J. Sandin, and Thomas A. DeFanti, "Using Video to Create Avatars in Virtual Reality". Visual Proceedings of the 1997 SIGGRAPH Conference, Los Angeles, CA, August 1997, pp. 128