The Watching Window

Tracking revisited

University of Twente
The Netherlands

Date: November 17th 2004

Author: Jeroen Broekhuijsen

At: Graphics department, University of Otago,
TKI, University of Twente

Supervisors: G. Wijvill
B. McKane
A. Nijholt

Abstract

This document describes my twelve week period at the University of Otago. In this time I've worked on the Watching Window project. In this project, the user interacts with a 3D world without the use of a keyboard or mouse. Interaction occurs by capturing motion and gestures with several cameras inside a controlled environment.

In those twelve weeks I have focussed on: *Improving the performance and robustness of the tracking system of the Watching Window.*

I have improved the system as a whole and, more specifically, changed the way the camera images are interpreted. I improved the tracking system to make it more robust and perform better. To increase the performance I've made algorithms faster or introduced algorithms to increase robustness. I've also introduced an EigenFaces method to verify the interpreted information.

Contents

1. Preface

The “Watching Window” has been an ongoing project with different people working with, changing and improving it. Recently there have been several major changes in architecture and set up. The old system, which used a backlit environment and older TV-cameras, has been replaced with an imaging filter system that has FireWire cameras. The organisational structure went through a major change as well during this process.

When I first arrived to work with the “Watching Window”, this major change had just taken place and the system was not yet in a fully functional state. Charles, another Dutch student, was also working on the project at the time.

My first priority was to create a working version of the system, which allowed me to get to know the system and perform the minor changes needed to make the system work.

After making the necessary changes there was still one major problem. The calibration of the cameras was not integrated in the system. The data the cameras were providing were not useful for the display program.

Charles worked on the calibration part of the system, and I helped him trying to locate the problem with the calibration data. In order to split the work I started working on the “3d-server” program to find whether it was functioning correctly, while Charles focussed on understanding the Calibration process and the calibration program itself.

When the calibration was integrated into the system I had more time to improve the vision system. At this time we were also able to add a third camera to get better accuracy.

Since then I have been working on the vision system to improve the tracking system.

In the following chapter I will shortly discuss the changes necessary to provide a working version of the Watching Window. Changes had to be made to install the new FireWire camera system, check the “3d server” and improve functionality in the “Watching Window Client” program.

I will then focus on the main problem, improving the performance of the tracking system in chapters 2 and 3, Tracking revisited and EigenFaces, where changes have been introduced to Vision system, but first I will introduce the Watching Window.

2. Watching Window

The Watching Window allows a user to interact with a computer without the usage of keyboard or mouse. The user is observed by several cameras and the images of these cameras are used to interpret the users' actions. Currently the system uses three cameras in total connected to a single computer.

The Watching Window is a booth. The user enters through a door on the side and can see the display on a large screen. One PC is used to handle all the information from the cameras, and the other is used to handle the interpreted information and provide feedback for the user. A rough description of the setup and layout of the booth can be found in Figure 1.

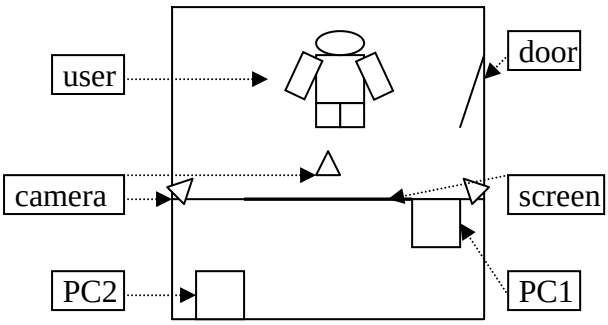


Figure 1. Overview of the Watching Window.

The first computer runs a program called "TwoCameras", which captures the images from the three cameras, interprets the data and sends this data to computer 2.
The second computer converts this data in the program called "3D server" and sends it to the display program called "Watching Window Client". This last program outputs to the screen in the middle of the booth.

The user is able to interact with the screen through motion or gestures and is allowed to walk freely in front of the screen.

The ability to walk freely in front of the screen has been made possible since the introduction of the new FireWire cameras.

2.1 FireWire cameras

FireWire cameras were introduced to replace the old TV cameras, which had been used in the booth previously. FireWire cameras allow a higher resolution of images, wider field of view and are more noise resistant.
Although these new cameras performed better, some new problems arrived after installing them. This led to several changes in the Watching Window programs.

Removed old TV-camera code

To change to the new FireWire system, the old TV-camera's were redundant. All old TV-camera relevant code could be removed.

Restructured code

While doing this we also changed the structure of the program to better match Rob Oudenaarde's [Rob04] specification, see Figure 2.

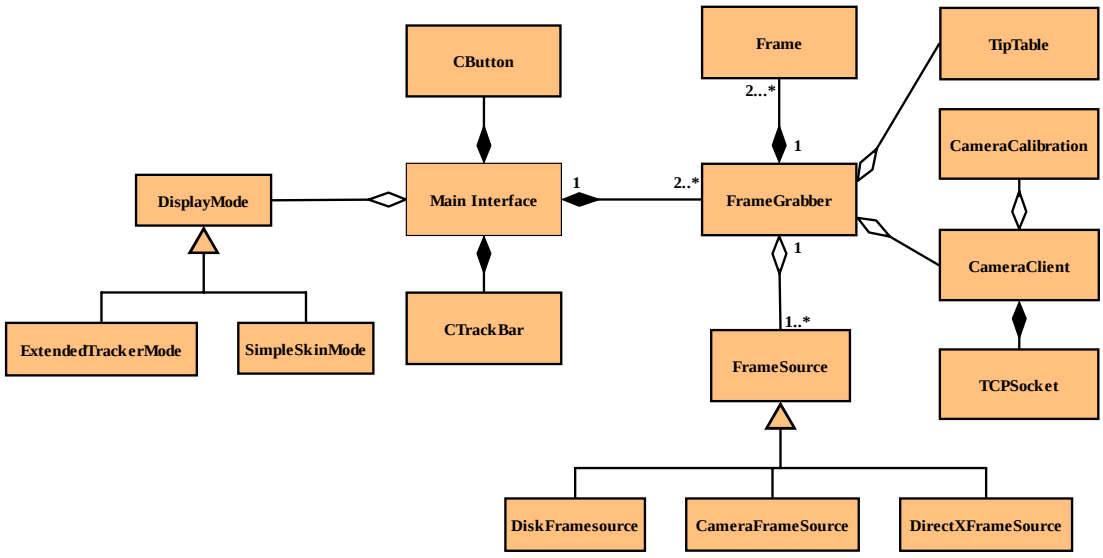


Figure 2. Class diagram for the "TwoCameras" application created by Rob Oudenaarde. [ROB04]

Some functionality was in the DisplayModes while it should be in the ImageFunctions and some functionality was in the FrameGrabber or ImageFunctions while it should be in the DisplayModes or vice versa.

Menu

The menu of the "TwoCameras" application contained deprecated functionality or functionality which no longer worked. Deprecated functionality was removed, new functionality was introduced.
Functionality to handle saving images was fixed to handle the new FireWire camera images and save correct bmp files of all sizes. Before, it was only possible to save the entire image. Now, sub images can be saved as well.

Rotated cameras

The cameras were mounted differently (figure 3) and the tracking code and image algorithms had to be adjusted to these rotations.

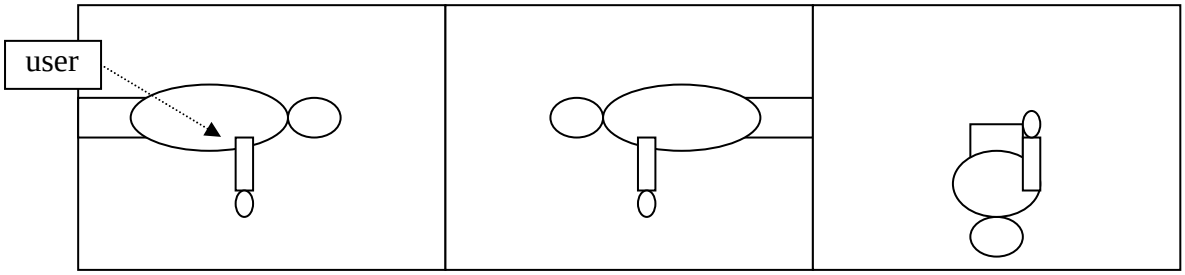


Figure 3. Left showing the Wall camera, middle showing the Door camera, right showing the Ceiling camera. As is visible all three cameras are rotated. The Wall camera is rotated 90 degrees clockwise, the Door camera 90 degrees counter clockwise and the Ceiling camera 180 degrees clockwise.

Lighting Conditions

The lighting conditions are different for these cameras, not only from the old TV cameras, but also from each other. This caused problems with the skin colour algorithm, which no longer worked. The main problems are the colour differences and whiting out in certain areas as you can see in figure 4 below. Especially in the skin colour algorithm this led to problems detecting the skin.

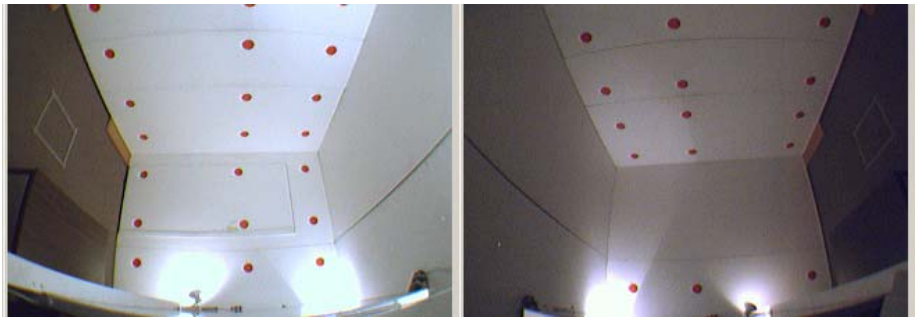


Figure 4. Left having a larger shutter time providing more light in the booth. Right having a shorter shutter time but still having the whiteout spots at the edge.

Noise

The amount of noise in the FireWire cameras is also different and behaved in a different way. This led to different threshold values in various algorithms to adjust for this new form of noise.

Performance

After the changes necessary for the FireWire cameras we tested the performance of the system.

The performance of the “TwoCameras” application was lower than with the old TV cameras. This was mainly because the skin colour mechanism did not work properly with the new FireWire cameras. After Charles rewrote the skin colour algorithm the system performed well, but not as good as we had hoped.

The FireWire cameras had a wider field of view and the images now covered almost the entire booth. Since a user is smaller in the image, less detail is available for the system to interpret the actions of the user, making it easier to mislead the system.

The system currently only looks for a head and a hand (regardless of left or right hand) and never verifies if what it has found is actually a head or hand, making it in many ways possible to fool the system into believing the head or hand is somewhere

else. The system also has many false positives which slow the process of finding a head and hand down. But these are the same problems encountered as with the old TV cameras.

The “Watching Window Client” application however did not display a correct view in respect with the users position. The 3D coordinates the program received were incorrect and did not reflect the actual position of the user in the booth.

It seemed the calibration process of the new cameras was not working properly. Charles had been working on the calibration program. To split the work I decided to focus on the “3D server” application to try and find the problem.

2.2 3D server

The "3D server" application is used to calculate coordinates from the camera image to real world coordinates. The server takes two image coordinates and then transforms them into one real world coordinate. The "Watching Window Client" reported incorrect data, but when I did the calculation by hand the calibration values seemed to work. There had to be something wrong in the 3d server to create this discrepancy.

I'll first explain how the 3d server actually calculates the 3d coordinates using the calibration information [Tsai86] and explain what went wrong in the process.

Steps to calculate from the image coordinates to the real world coordinates:

1. Receive the two 2d coordinates from the TwoCameras application.

Then for both coordinates do:

2. Input coordinate:
   See figure 5, the blue dot with image coordinate system, the thick lines
   t[x] = x where x is the input x coordinate
   t[y] = y where y is the input y coordinate

3. Centre adjustment:
   See figure 5, the blue dot with camera coordinate system, the thin lines
   $t[x] = x - centre_x$
   $t[y] = y - centre_y$

4. Warping:
   See figure 5, the green dot, adjusted for camera distortion

   $tempkappa = (t[x]^2 + t[y]^2) * kappa1$

   $difx = t[x] * tempkappa$
   $dify = t[y] * tempkappa$

5. Calculate Initial Vector t
   $t[x] = t[x] + difx$
   $t[y] = t[y] + dify$
   $t[z] = focal\ length\ camera$

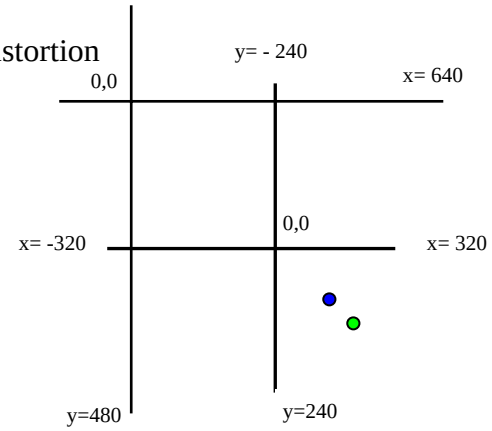


Figure 5. Warping an Input coordinate. Blue dot is the input coordinate, green dot is result coordinate.

6. Translation of vector **t** to centre of the real world coordinate system:
   t = t + translation vector (Camera Specific)

7. Rotation of vector **t** to the real world coordinate system:
   t = t * rotation matrix (Camera Specific)

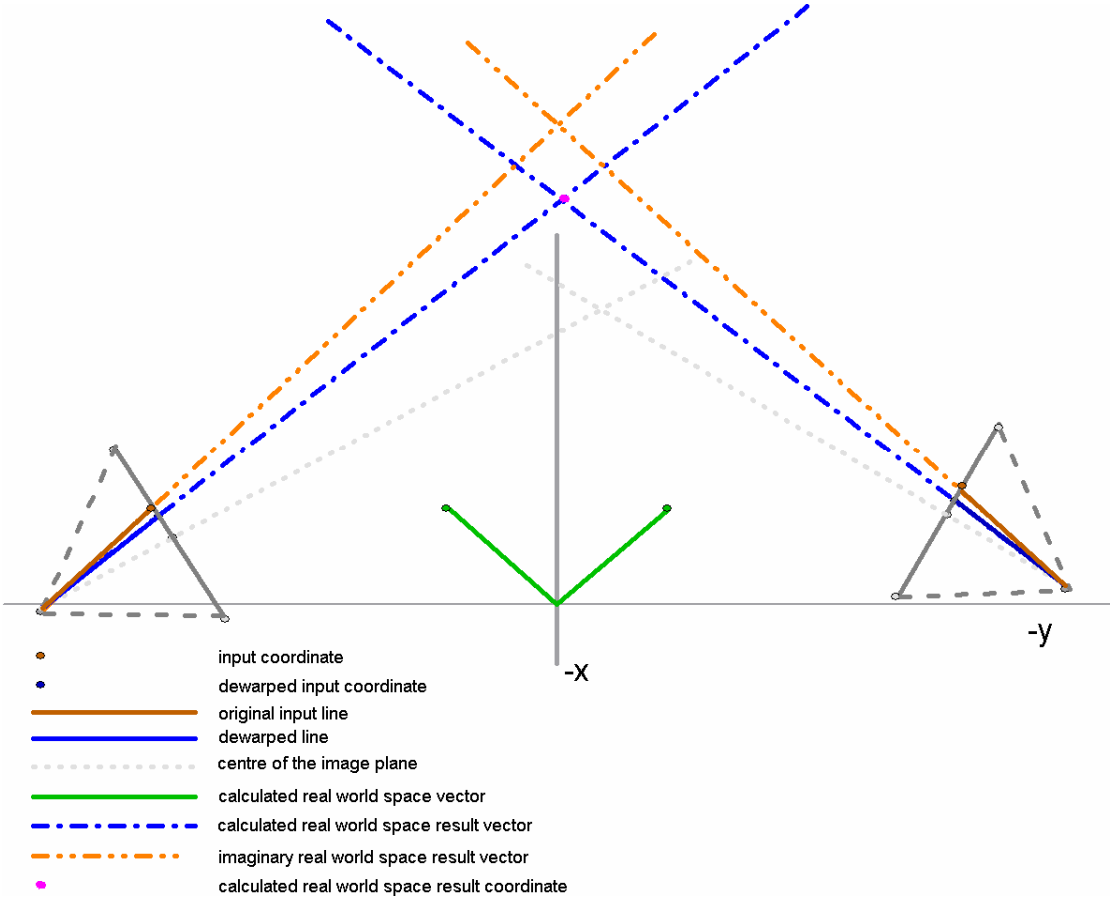


Figure 6. Description of calculation process for two 2d coorinates to one 3d coordinate.

Vector **t** is now a vector in world space from the origin of real world space, see the green lines in figure 6.

8. These two vectors need to be translated to the origin of the camera, see the solid blue line in figure 6.

9. Now we extend these two lines, see the dotted blue lines in figure 6, and find the point where the distance between them is the smallest, see the purple point in figure 6.

10. This point is the resulting real world coordinate.

This coordinate is very accurate, but if the original input coordinate is incorrect, the resulting real world coordinate will also be incorrect, but by a much larger factor.

This is why the information coming solely from the Wall and the Door camera is not enough. Information from the Ceiling camera is also needed to be accurate.

All three combinations of cameras, Wall – Door, Wall – Ceiling and Door – Ceiling are calculated. This provides more accuracy and averaging the resulting end-coordinates from all three calculations will give a better result.

Problems

Some things must have gone wrong in the process in the 3D server application. After I checked the entire process by hand I discovered the following faults in the system:

- The centre $_x$ and centre $_y$ coordinates were not the actual calculated centres from the calibration data but the standard value of x=320 and y=240.
- The resolution of the calibration was 640x480 while input coordinates were from a resolution of 320x240.
- The calibration files for the first two cameras were swapped, in the TwoCameras program, so the data provided to the calculation process were incorrect.
- Dewarping and warping did not work properly, a new method for warping and dewarping was necessary.
- A bug in communicating the data to the Watching Window Application program led to incorrect 3d coordinates.

2.3 Calibration

After solving the communication problem the data started to make sense and when the other issues in the 3d server were corrected, the accuracy was close to 1 cm.

Charles solved all the problems with the calibration program and made it possible to check the data, to see if it was valid.

2.4 Watching window client

The watching window display client runs the applications and takes the real world coordinates from the 3d server as input.

Since the data received from the server were now correct. Almost all “magic” numbers could be removed. These magic numbers were used to correct the data from the old TV-cameras application, which did not produce correct data.

With the coordinates coming in correctly it was noticeable that due to minor movements the calculated 3d coordinate seemed to jitter and could sometimes even be incorrect if the cameras found an incorrect position for the head or hand.
The incoming data needed to be averaged to filter outliers from the input. A simple averaging routine was included which took the average of the last three input values.

All programs were now working, but the drawing program still needed improvement. It had correct input data but a user was not able to stop drawing in the drawing program.
The easiest way to solve this was to define a zone in which the user could draw. This zone was within half a metre of the screen. If the hand was outside this zone the current line stops and the user can start a new line from another starting point.

Due to the way the hand is detected in the camera application, the coordinates were always a certain amount off. The centre of the hand is always used instead of the tip of the hand. A new “magic” number was used to correct this value and subtract 30 to 50 centimetres from the X coordinate of the hand.

When testing the application in a test environment another bug was found. Resetting after changing modes did not clean up the GL buffer properly. After fixing this last problem the application worked properly.

After solving the initial problems, the system worked properly and as fast as before, but a lot of false positives and false negatives still affected the performance and robustness of the system. I tried to improve this by changing the “TwoCameras” application, the tracking system.

3. Tracking revisited

Since the system as a whole functioned well, after making the described changes, I could focus on my main goal: *To improve the performance and robustness of the tracking system of the Watching Window*.

The Computer Vision part of the Watching Window is currently its most important part. It gathers the required data for the rest of the system. Most importantly it locates a persons head and hand as input for the “3d server” and “Watching Window client” applications.

The application called “TwoCameras” handles all the incoming video data from the cameras and processes it to generate camera coordinates of body parts to send to the 3d-server.

How does the application do this? As we can see from Rob’s report [Rob04] on the architecture and structure of the TwoCameras program there are three steps to get the information, see figure 7.

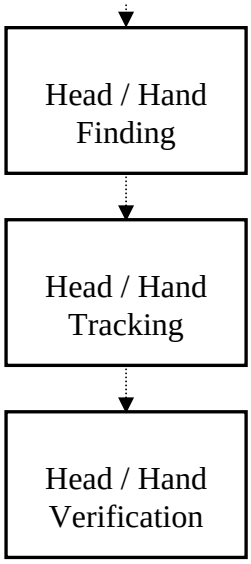


Figure 7. Three steps that make up the structure of the process that generates the coordinates for the head and hand.

In each of these steps a variety of methods, filters and algorithms is used.
The algorithms used can be found in Rob’s report [Rob04]. I will briefly discuss them again to show the changes made in all three steps.

Finding
- Movement detection
- Silhouette generation
- Box fitting

Tracking
- Texture matching

Verification
- Skin colour verification.

The same procedure is used for the head and the hand.

The new method has the same structure, but has additions and changes in the used algorithms.

The old method works as detailed in Figure 8. The new method is set up to work like Figure 9.

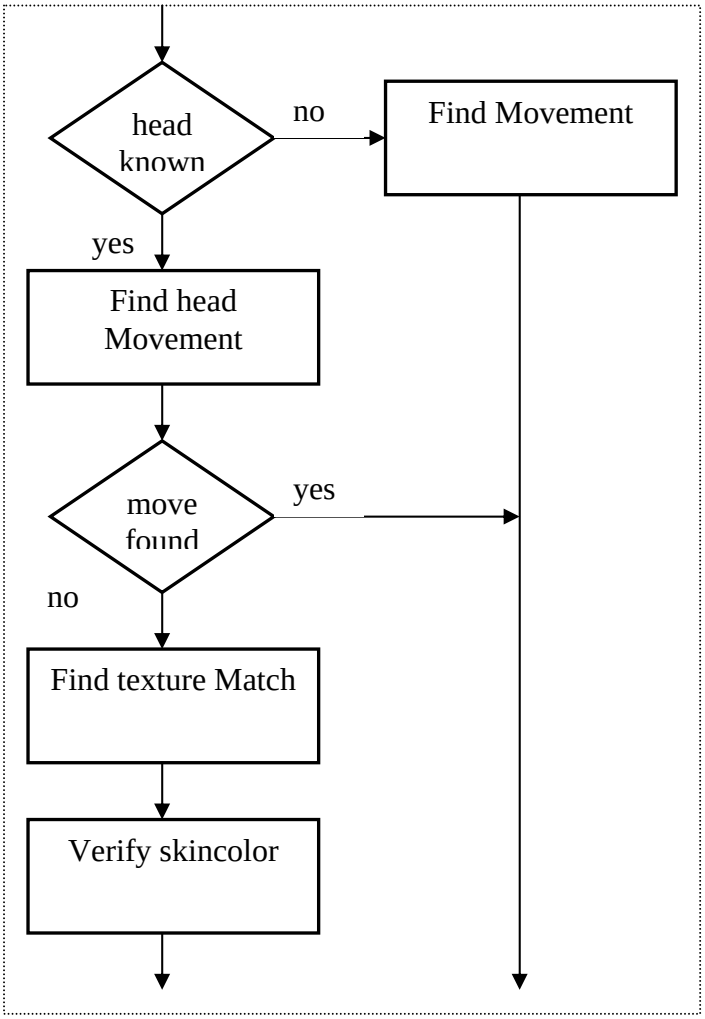


Figure 8. Old method of finding head / hand coordinates.

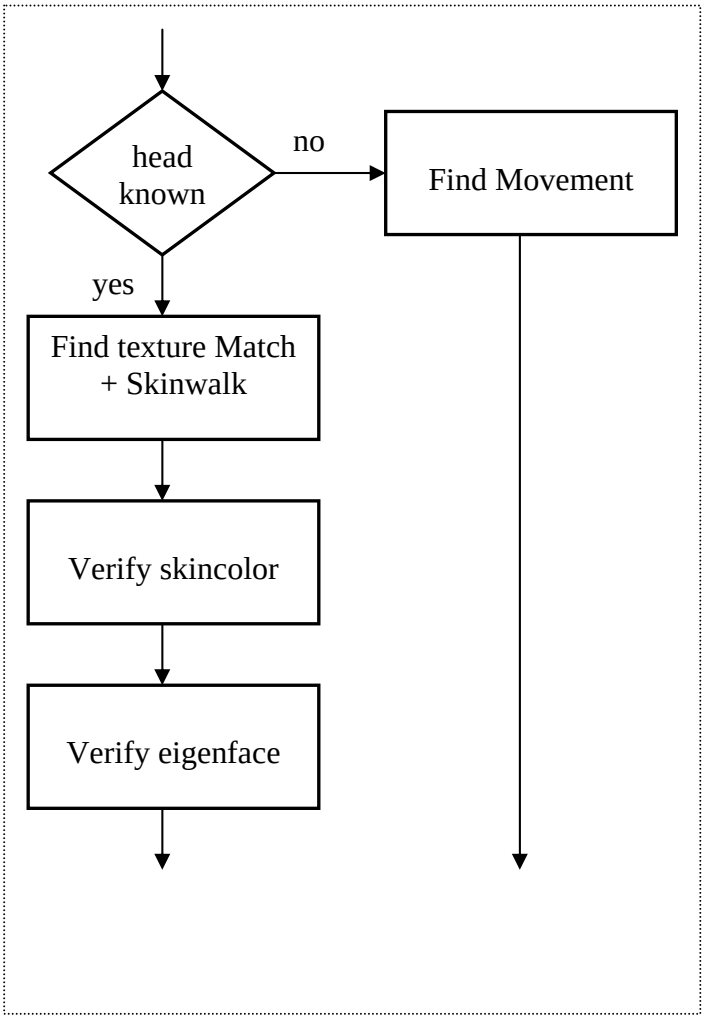


Figure 9. New method of finding head / hand coordinates.

3.1 Problems in the tracking process

After making the changes to make the system work, the system did not perform well enough compared to the old TV cameras. It was slower and did not track as well as the old TV cameras system since the skin colour method, used in several extensions to the system which Rob made, did no longer work.
As well as being slower, it was possible to trick the system in believing things that should not be. This was also possible with the old TV camera system.

Since the system only checks for skin colour, one of the ways was to cover your head with your hands and then the texture matching process would take a snapshot of the

hand and track the hand instead of the head. Also you could guide the texture tracking to a skin coloured region in the booth and put the head or hand coordinate there. These tricks are possible because the system has no way to validate if there actually is a head or hand present in the image. It only checks for skin colour and the algorithm usually generates too much or too little skin colour because of lighting problems. Because the texture used was only 10 by 10 pixels large, the entire head or hand would not fit inside the texture making it possible to track something else instead of the head or hand.

A method is needed to verify the actual presence of a head or hand in the tracking texture. This is why I added the EigenFaces approach.

To increase the speed and robustness I went through every step in the vision process and changed several methods to either increase robustness, or to increase speed.

3.2 Movement detection

After movement detection a box fitting process would analyze the silhouette and create a box around the head. Usually this box would be too big. A larger headbox leads to more processing time in the texture matching process. A smaller and better fitting headbox could be better for the speed of the system. First I will explain the old situation of the silhouette and then describe the new situation of the silhouette.

Old situation
The old method starts with movement detection, see figure 10.
Every pixel that passed a filter to check for neighbouring pixels was filled with a blob of 16x16 pixels (see figure 11). The topmost blob is then found and this creates the head area.

New situation
The silhouette procedure was improved to make a better silhouette, see Figure 12. Instead of using 16 by 16 pixel blobs, 2 by 2 pixel blobs were used and then connected both horizontally and vertically to create a solid shape.

The filter which used to generate the silhouette was changed to be more accurate, more noise resistant. It verifies that each pixel has at least 3 neighbouring pixels in an X shape. As can bee seen from the top of figure 11 and figure 12, the new method is slightly slower than the old method, 12 versus 10 frames per second, but provides a better fitting silhouette. The silhouette is only necessary when the head has not been found. The speed loss only occurs if the head is not found.

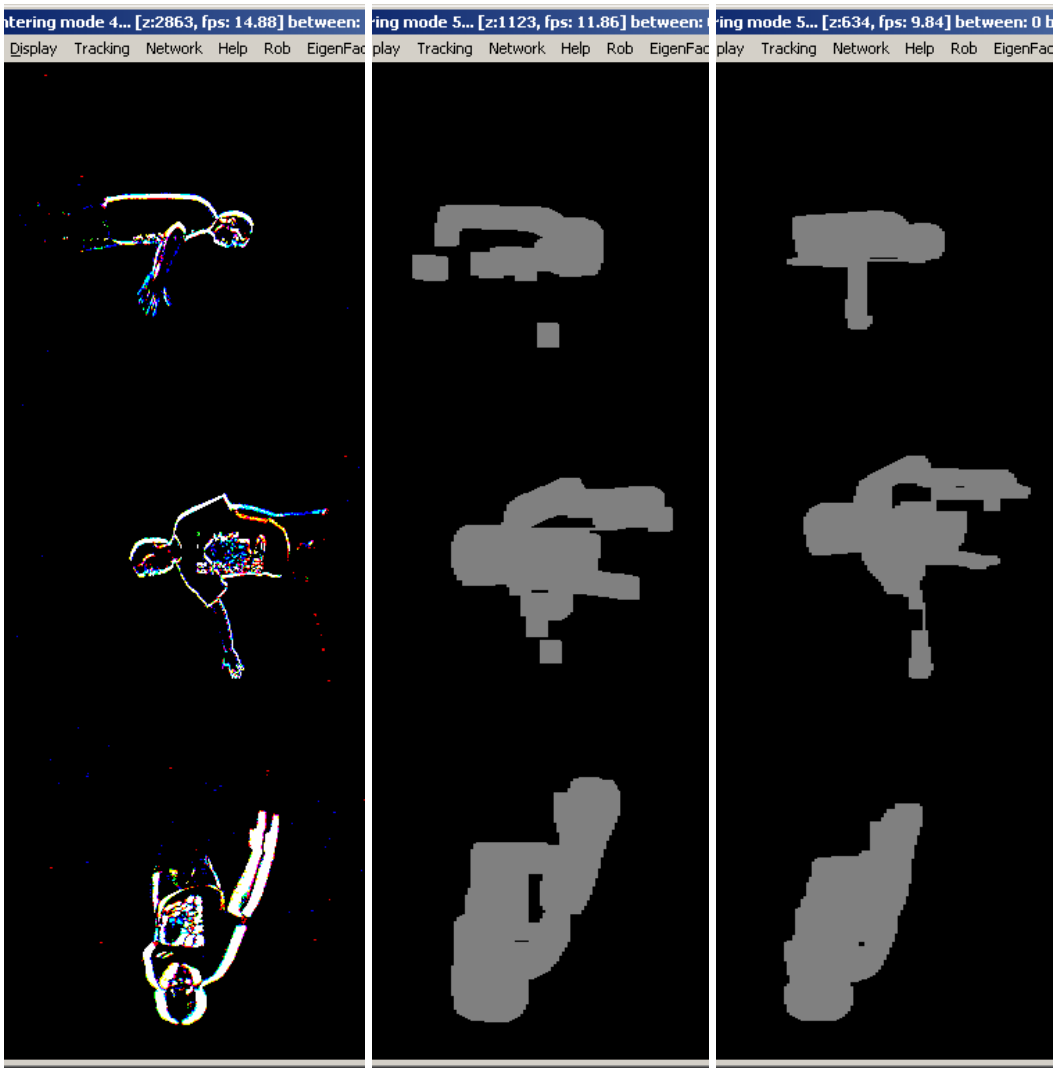


Figure 10. Movement masking. Figure 11. Old silhouette Figure 12. New silhouette

The head and the hand finding procedures are now separated and the hand finding function no longer generates a silhouette, but finds the first non-noise pixel.

There used to be a second round of movement detection in the old algorithm. This was removed since it led to false detections and the performance dropped significantly because the texture matching process would have less matches if a false detection occurred.

The new method however created a smaller silhouette shape of the head. Some threshold values needed to be changed accordingly, and the head has to be closer to the camera to create a positive detection. If a user is too far away the head can be too small to be accepted as a correct head.

3.3 Texture matching

Texture matching takes a lot of processing time if the head has been found. Making texture matching faster would not only increase the speed of the system, but also increase the speed of the texture matching process itself. If the system can handle more frames, the distance a user travels between frames is less. If this distance is less, a better texture match can be found in a smaller surrounding area, this cuts search time necessary for the texture matching process.

Old situation
A sub image was taken when the head was found and this sub image was checked against several sub images in the new incoming frame. This sub image used to be 10 by 10 pixels and was taken from the centre of the green box. It was matched with all other sub images of the same size around the original coordinate. The centre of the sub image with the highest score set the new coordinate for the next round.

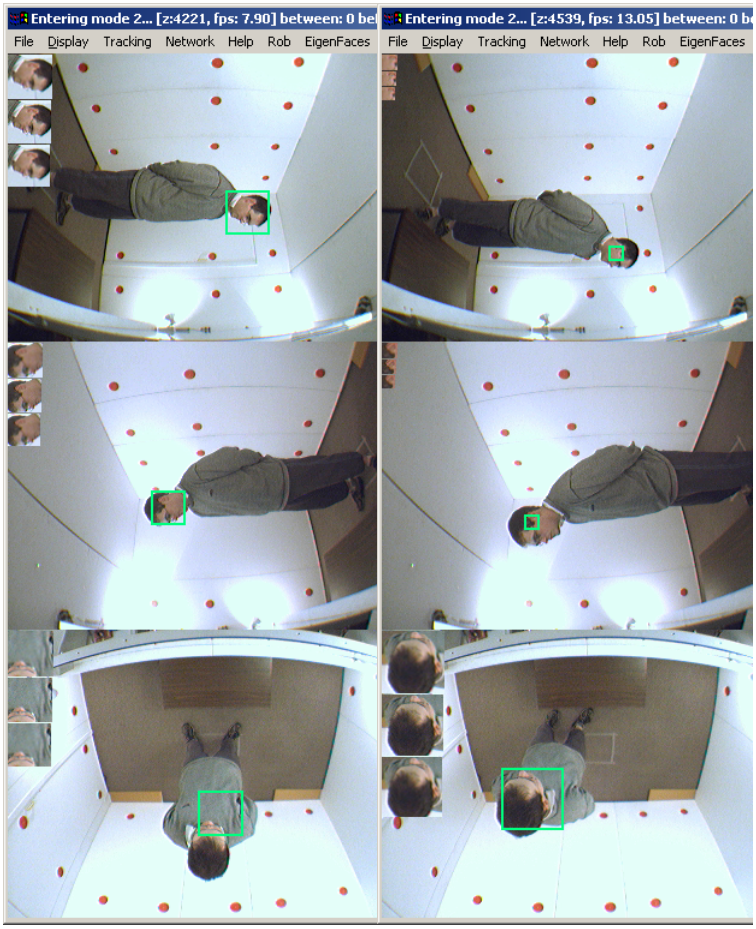


Figure 13. New style resizing before EigenFaces. In the bottom left picture the head image grew to the point where almost no face was left in the image and the sweater makes up almost the entire sub image. In the top right picture the head image shrank to the point where only a part of skin was left in the image.

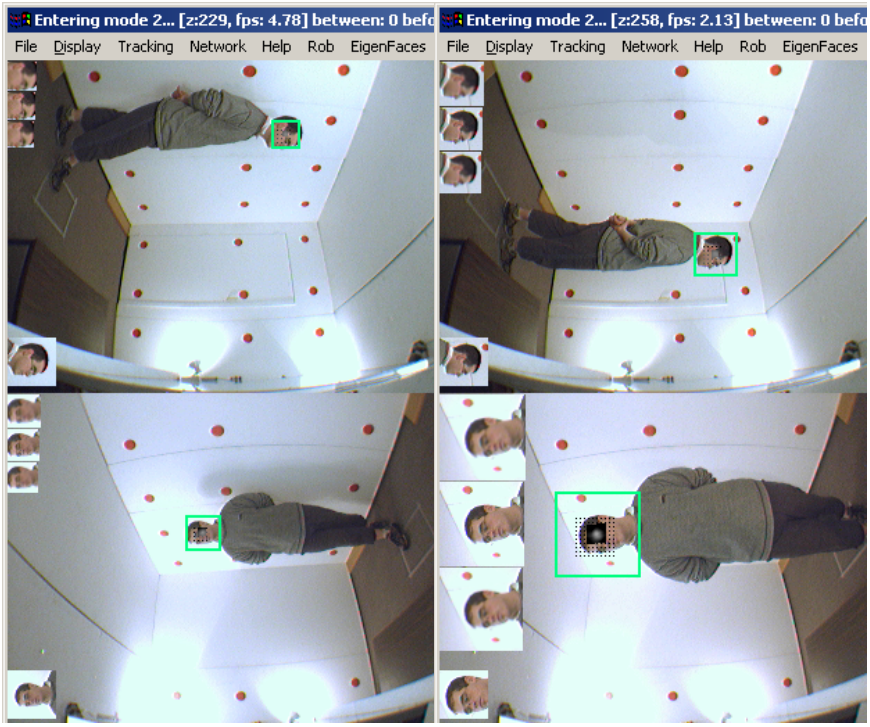


Figure 14. New style resizing after EigenFaces. In the left picture the head image shrank to match the person further away and becoming smaller in the image. In the right picture the head image was enlarged, to match the person walking closer to the camera and becoming larger in the image.

New situation

Now, instead of using 10 by 10 pixel sub images, the entire green box or a percentage of it, whatever its size is, can be the sub image. Furthermore, instead of checking all possible sub images only every second sub image was checked to reduce processing time by halve. This is similar to the way the black dots are spaced one pixel apart in figure 14.

All sub images were blurred to improve matching. Instead of blurring every sub image individually we blurred a range around the coordinate of the image and took sub images from the blurred frame.
After fixing an additional rounding error, blurring increased the matching score dramatically.

To see if an object came closer, or moved further away from the camera, the program not only checks with the same size image, but also checks with a smaller or larger size image. These larger and smaller sub images were created from the original sub image by bilinear resampling. The larger size, normal size and smaller size images are shown in the top left of the image for debugging purposes, figure 13 and 14. This however did not perform as well as was anticipated.

Sometimes the larger image would always give a higher score, and sometimes the smaller image would always give a better score, figure 13.

It is difficult to say why this happens. Apparently the same size image did not match well enough, and the larger/smaller size image had a better match.

One of the reasons could be the lack of key contrast points in the sub images, especially if the original position of the sub image was not correct and had only half of the face inside the sub image. In this case the background either tended to add to the image till the image was too large, or shrink so much that only a very small portion of the head was included in the sub image.

Later experiments showed that if the sub image had the head at the centre and the entire head was included, the enlarging or shrinking did seem to work since the entire head was now fully inside the sub image, providing more contrast points for the matching process (figure 14).

3.4 Skinwalk

After the texture matching Charles and I introduced a new method, the skinwalk.

First, a skin colour test was done to centre the sub image and then additional tests were performed just above the head/hand area to see if the sub image was positioned too low and the sub image needed to be “higher”.

Especially for the hands this was a welcome feature, since the system occasionally tracked the wrong spot when someone with bare arms steps into the booth.
Now the position of the found hand will slowly “walk” across the bare arm to the top of the outstretched arm (figure 15 and 16).
There are still problems when the user uses both hands since the system is not yet able to distinguish between the two.

Figure 15. Original Head/Hand position

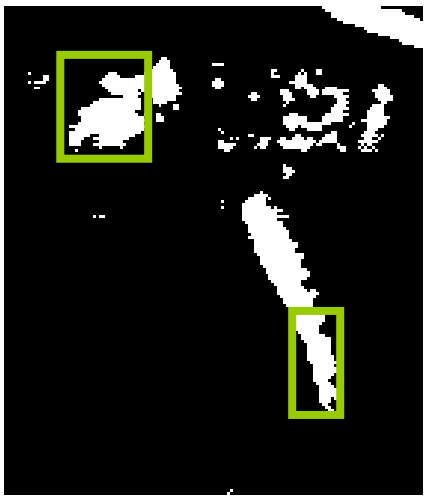


Figure 16. Position of Head/Hand after skinwalk at the outer edge of the head/hand.

3.5 Verify Skin colour

The old method no longer worked with the new FireWire cameras since the colour properties of the camera image are different.

Charles made a new skin colour method that is now used. The result is visible in Figure 15 and 16.

3.6 General Improvements

Next to the main algorithm to find the head and hand coordinates several other features were included as well.

Timer

First of all, to be able to tell if methods were actually faster, I introduced a timer in the menu bar, to indicate the time taken to execute the algorithm for all the cameras.
This provided us with a means to verify whether an algorithm actually performed faster or slower than before (figure 17).

Figure 17. Menu bar with timing indication, here the algorithm takes 391 ms.

Modes

Secondly, several old modes were removed from the program, and new several modes created.

The tutorial modes were changed to show how the new head/hand finding worked.
A GoodWorkingMode was added to indicate a stable mode, which should always work fast and robust. The ExtendedTrackerMode was changed to be an experimental mode in which features could be added or removed to see the impact on the performance of the system. This mode also has the EigenFace approach and texture matching approach included.

Extra filters

Several new filters and ImageFunctions were introduced to make the new algorithm possible.

- Median blur filter [Med04]
- Gaussian blur filter [Gau04]
- Bilinear resample algorithm [Vxl04]
- SimpleFilter (remove noise filter)
- FillSmallFrame
- Charles made the new skin colour filter

3.7 Eigenface check

At last I tried to verify the head with an EigenFace approach to see if the found head actually is a head, or just something else entirely. The eigenface check is discussed in more detail in the next chapter.

4. EigenFaces Approach

Since the lighting issues with the skin colour algorithm could not be easily solved I wanted to look for a solution in another direction. An EigenFace based approach seemed a simple and possible solution to verify if a head was present and to increase the robustness of the system.

4.1 Eigenface Method

The EigenFaces are calculated following the method from M. Turk and A. Pentland [Tr91] and using the VXL library [Vxl04].

I will describe the method broadly step by step and then explain what I've done with the method in the Watching Window Code.

Training set

First a training set is calculated from a set of input images which all have the face centred in the image.

1. Calculate the average face from all input images and then subtract the average image from each input image (figure 18).

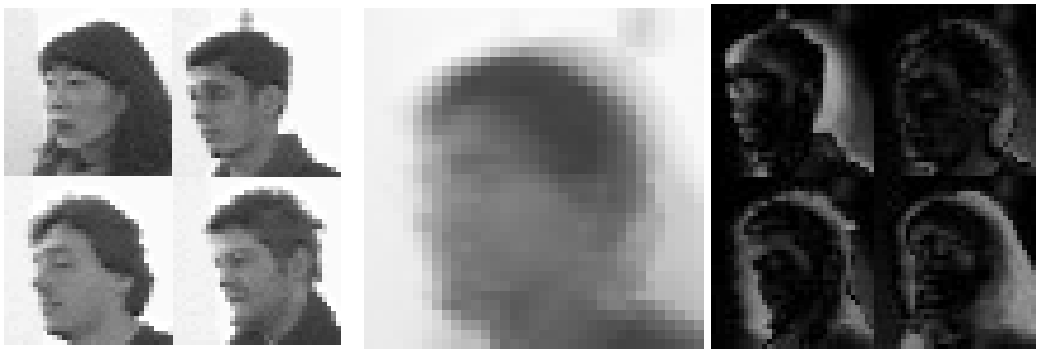


Figure 18. Left are four example input images. The average face image is in the middle. Right are the mean adjusted input images.

2. Using principle component analysis, a set of vectors, EigenFaces, is calculated that best define the distribution of the data. These make up the training set (figure 19).

Figure 19. These are five examples of EigenFaces from the training set.

4.2 Detection

After the training set is calculated we can try and determine whether a face is present in an input image, note that the face has to be centred in the input image.

1. Calculate the mean adjusted input image (figure 20 and 21).

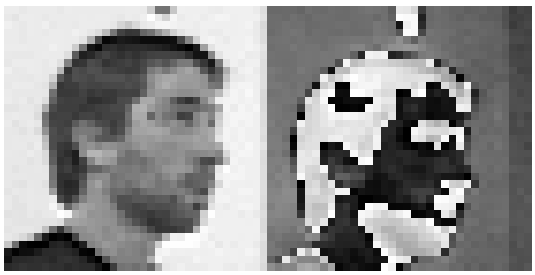


Figure 20. Left is the input image. Right is the mean adjusted input image.

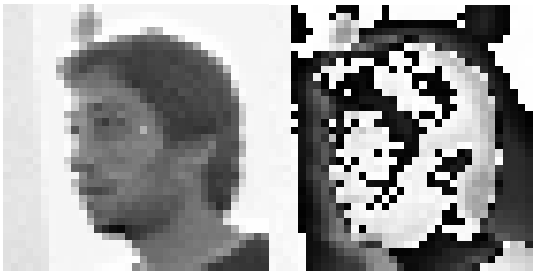


Figure 21. Left is the input image. Right is the mean adjusted input image.

2. Using the training set, project the input image onto the training set (figure 22 and 23).

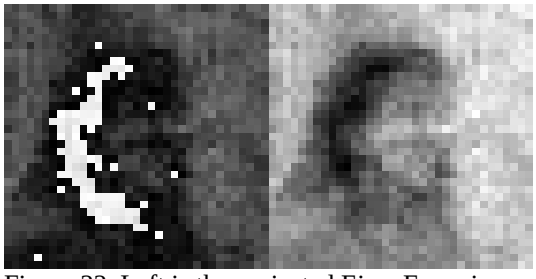


Figure 22. Left is the projected EigenFaces image. Right is the reconstructed face.

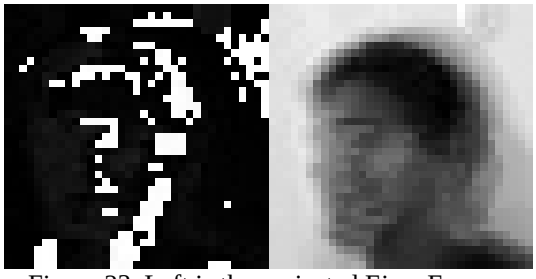


Figure 23. Left is the projected EigenFaces image. Right is the reconstructed face.

3. Calculate the distance between the projected and the mean adjusted input image, and calculate the distance between the original input image and the reconstructed face image.
4. Classify whether the input image has a face or not depending on the calculated distance.

Both distances are used to classify the input image. In some cases one threshold was not distinctive enough to classify the image. Especially if a uniform black or white image was presented, the score would still be high enough to pass. After introducing a second check, the distance between the original image and reconstructed image, these scored lower, the classification process performs better.

4.3 EigenFaceMode

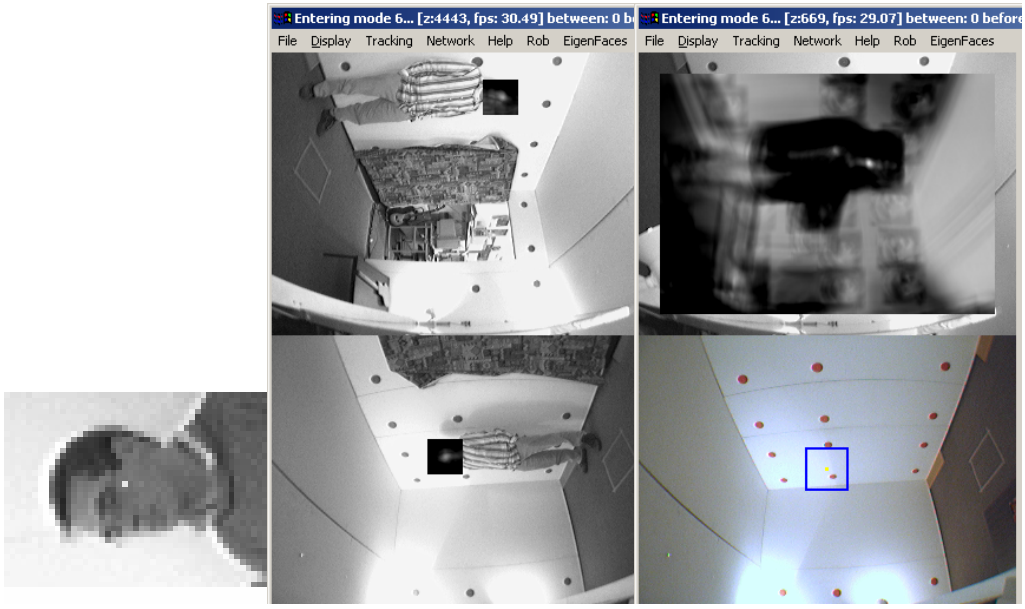
I implemented an EigenFaceMode. In this mode a left click determines a capture area, a right click then adds the current image in the capture area to the training set (figure 24).

Fig 24. Small section of the camera image. Here a sub image is selected to be added to the database.

In the menu exists a “calculate training set” button that activates calculating the training set from all training images currently in the database. Then the toggle mode Calculate EigenFaces can be turned on to see if there is a face in the designated capture area.

The program can be altered to calculate the face chance of:

1. The current sub image, see figure 25 left.
2. Each sub image where the centre of the sub image is within the capture area, see figure 25 middle.
3. All the sub images in the entire screen, see figure 25 right.



Figer 25. Left shows the face chance of a single sub image. The middle image shows the face chance of all sub images with the centre inside the capture area. The right image shows the face chance of all sub images in the screen.

4.4 Program functionality in EigenFaceMode

The functions listed below are all methods inside the EigenFaceMode in the TwoCameras program. These methods provide functionality for the EigenFace approach.

AddtoDatabase:

- Saves a sub-image to file and ads it to the list of files in the training.txt file.

LoadEigenvectors:

- Loads the eigenvectors.txt file and stores all Eigenvectors in the Framegrabber.tiptable.eigenvector

CalculateTheta:

- Calculates the average Euclidian distance of all images in the training.txt file.

ApplyMode:
Either:

- Calculates the average face, EigenFaces, and saves the average and eigenvectors to file and loads it to the TipTable.
- Loads the average and eigenvectors in the TipTable from file.
- Calculates the theta and saves this in a file
- Classifies a sub image or range of sub images and outputs a chance in the output image.

CalculateAverage:

- Loads all files in training.txt and calculates the average image. The first image is used to set the size. Then outputs the average image to a file and outputs to the screen.

CalculateEigenFaces:

- Loads the input image. Calculates mean adjusted input images. Stores them in A. Calculates A^T and calculates $C = A^T*A$. uses the VNL_Symmetric_eigensystem [Vxl04] matrix to calculate the orthonormal vectors into C. Then stores all the eigenvectors to file after normalizing them. And saves the normalized EigenFace images for debugging purposes.

ProcessMouseClick:

- Left: creates area around the point clicked and displays the box on screen in ApplyMode.
- Right: takes a sub image. Converts it to greyscale. Adds it to the database and displays on the screen which images was taken.

Classify:

- Classifies a selection on the screen with the chance of having a face in a sub image. Outputs a pixel with value 0 to 255 depending on the face-ness of the sub image.

5. Conclusion

Looking back on all the improvements to the Watching Window, it is now a working system where users can interact with the computer using only their head and hand.

The main goal, increasing the performance and robustness of the tracking process, has been achieved. Performance has increased from the old TV cameras system which had an average frame rate of 16 frames per second and the eigenface method provides a fairly accurate way to verify whether a head is present or not. It has become harder for a user to trick the system and the system tracks the user more accurately.

5.1 Performance

The GoodWorkingMode of TwoCameras can now operate with 3 cameras at 14 frames a second, and with two cameras at 20 frames a second, a significant improvement over the older Television Cameras, with better found coordinates.

After introducing the timer to estimate the performance of the algorithms a bug was found in the code grabbing the frames from the camera. Sometimes the camera would somehow go into “slow mode” and drop frames from the cameras. Unfortunately I haven’t been able to fix this problem, only localize it.

5.2 EigenFaces and Texture matching

The ExtendedTrackerMode has the EigenFaces method implemented and different size texture matching and operates at roughly 4 frames a second where in two of the three cameras the EigenFaces method is used.
This speed is not usable for the system, but if some improvements can be made to the coding this speed could be better.

The EigenfaceMethod is able to predict roughly whether a face is present in the image and always does a better job at finding the centre of the head than the skinwalk method.

This makes the texture matching process perform better. Unfortunately the speed of the computers doesn’t allow testing of the texture matching. Texture matching performs best if the difference between frames is low. If the frame rate is too low, the difference between frames will be too large to provide a match and the algorithm will switch back to the finding procedure.
If the speed of the computers is fast enough, texture matching will perform better and the actual robustness and performance can be tested properly. Some testing has been performed with a very slow moving user and this provided encouraging results where the size of the head varies with the distance to the camera.

Also the detail of the EigenFaces is too low to actually detect faces, ellipses are also considered to be very face-like, see figure 26. A more detailed training set could improve the false detection of an ellipse.

Figure 26. Left is the primary EigenFace image, it is in the shape of an ellipse. As can be seen in the right image, if an ellipse shape is present it will also be classified as a face.

Furthermore the background is a problem for EigenFaces. The training set has to be larger to allow different backgrounds, see figure 27.

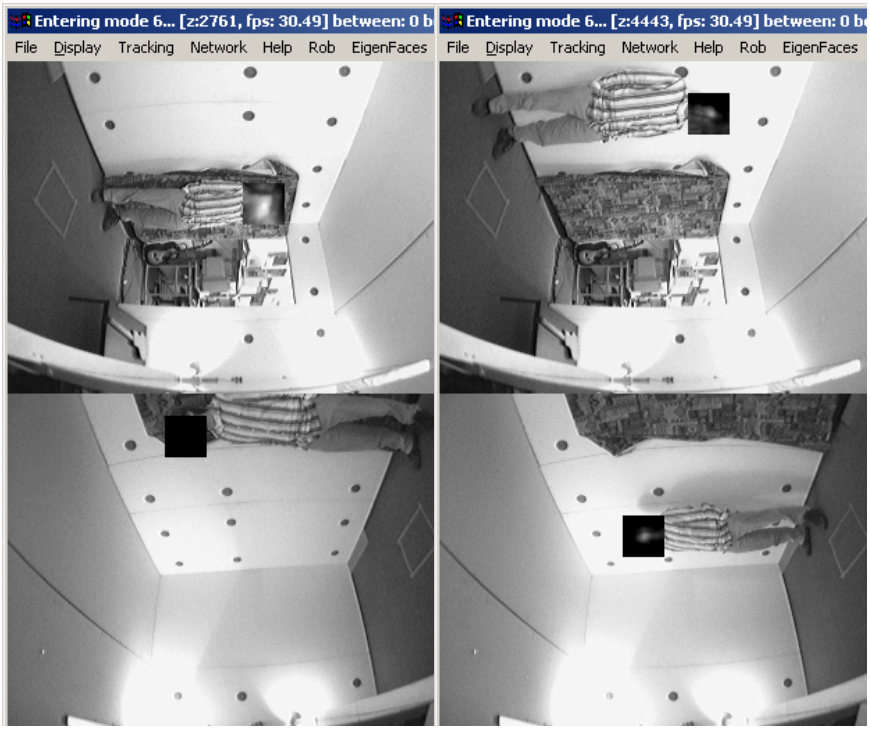


Figure 27. Different training sets were used for the camera in the top of the image and the bottom of the image. The top one was also trained for different backgrounds, while the bottom one was not. In the right image can be seen that the top training set still performs well on the white background while the left image shows that the bottom training set does not perform well with an alternative background.

Lastly, the size of the image makes a huge impact on performance. If the person is closer to the camera, the head is larger, and matches less with the database setup for average head size. If the person is further away from the camera, the head becomes too small to match well with the average head size.

6. Recommendations

EigenFaces

Since the EigenFaceMode does not have enough detail in the database to actually detect faces, a better database is needed. Larger, more detailed images could be used to train the system, after which the images can be scaled down to fit the input image, or the input image scaled up to fit the database images.

Instead of only a set of limited people, a wider set could be used to include even children, black people, and facial hair, especially if the system is used for demonstration purposes.

In the future, the booth is likely not to have a uniform white background. This will decrease the performance of the EigenFaces approach. The training set could be extended to include varying backgrounds. An example of a training set with varying background can be seen in figure 27.
Another approach could be to eliminate the need for a background in the training set and input images, but this requires better knowledge of the position of the head and foreground/background analysis.

The detection of the hands and/or gestures of the hand don't use the EigenFace approach at all. It could improve the detection of the hand instead of relying solely on the skincolor.

3d server

Instead of needing all three cameras to work it is better to use a voting mechanism that uses a majority of two out of three. When one of the cameras has a false positive the output is less affected if this camera is ruled out.
It could however be possible that the top camera is necessary to provide accurate results, especially for the position of the hand.

Network

Currently all three cameras are connected to one computer that handles all the data. This means that the processing power of the computer has to be divided over all three cameras. A lot more processing power will be available if each camera could have its own computer.
The main problem with this approach will be synchronizing all three cameras. A network protocol will need to be established to synchronize the cameras and exchange information on the position of the user.

Vision system

With the arrival of the new cameras old problems became apparent which had not been addressed yet. The cameras are always adjusting the white balance and colour balance to compensate for changes in their surroundings. This means that each input image will be different from the others.

Calibration for the light level and colour of each camera is needed to make the algorithms less dependent on the conditions inside the booth. Skin colour algorithms for example will no longer need to have separate variable values for each camera.

The EigenFace approach depends heavily on the rotation of the head. If the head is rotated, the detection rate will go down drastically. A database including rotations of the head could be added to the EigenFace database and the right database for detection can be selected if the rotation can be estimated by the vision process.

7. Literature

[Rob04] R. Oudenaerde, The Watching Window - Revisited. A report on upgrading the Watching Window, University of Otago, 2003 http://www.cs.otago.ac.nz/graphics/ww/papers.html

[Tr91] M.A. Turk, A.P Pentland, Face Recognition Using Eigenfaces. Proc. CVPR, June 1991. Page(s): 586 -591

[Gau04] http://homepages.inf.ed.ac.uk/rbf/HIPR2/gsmooth.htm Updated on: November 10th 2004

[Med04] http://homepages.inf.ed.ac.uk/rbf/HIPR2/median.htm Updated on: November 10th 2004

[Vxl04] http://vxl.sourceforge.net/ Updated on: November 10th 2004

[Tsai86] "An Efficient and Accurate Camera Calibration Technique for 3D Machine Vision", Roger Y. Tsai, Proceedings of IEEE Conference on Computer Vision and Pattern Recognition, Miami Beach, FL, 1986, pages 364-374,