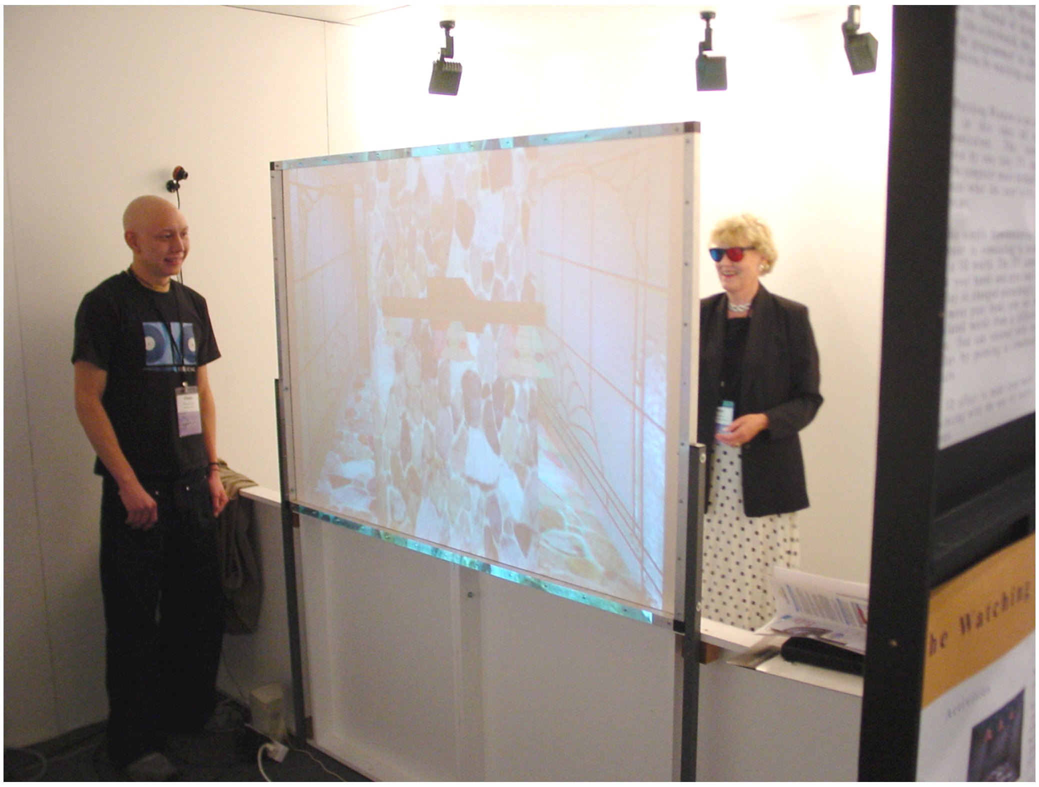
The Watching Window – Revisited

A report on upgrading the Watching Window



Date: 24 May 2004
Author: Rob Audenaerde, r.j.audenaerde@student.utwente.nl
Supervisors: Geoff Wyvill, Univeristy of Otago
Anton Nijholt, Universiteit Twente

Abstract

The Watching Window is an interactive 3D application. The users movement en gestures are the input of the system.

During my internship I implemented a new vision system for it. It tracks people near real-time, about 20 Hz. It finds the user's head location and hand location. Both the vision software architecture and the network architecture are changed and updated. The system is now more reliable and easily extendable.

Hardware changes are made, switching from tiny TV cameras to Firewire, which are more noise resistant.

Index

Chapter 1. Preface

In this document I describe what I did during my internship. I spent fourteen weeks at the Graphics lab of the Computer Science department at the University of Otago, New Zealand, where I have been working on the Watching Window project.

Firstly, I will give a global description of what the Watching Window is, and how I have been involved in the development of it.

My main focus during the project has been the computer vision of the Watching Window. The former implementation used a simple silhouette extraction method, based on a brightly lighted background. I changed that to a somewhat more advanced version. More information is found in the Computer Vision section.

Working on the old code base was difficult. There was almost no documentation and the code was not well organised. I revised the architecture and re-implemented and restructured almost all code. The results of that can be found in the Software Architecture section.

The Watching Window used a rigid peer-to-peer network model to transfer information from the cameras of the vision system to the display machine, to which the projector is connected. I replaced this model with a more flexible and more robust network implementation, which is described in the Network section.

Finally I give a short conclusion, some recommendations and an evaluation of my internship.

Chapter 2. Global Description

2.1 The Watching Window

"Computers have looked much the same for nearly twenty years. We are so used to the screen, keyboard and mouse that we forget that this appearance is merely a fashion and an accident of history. We should be able to communicate with a computer by speech and by gestures. Instead of using special tools like a keyboard, the computer can be programmed to determine our desires by watching and listening." – Geoff Wyvill

The Watching Window is an experiment in this idea of natural communication. The user is 'watched' by two tiny TV cameras and the computer must deduce from movement and gestures what the requirements of the user is.

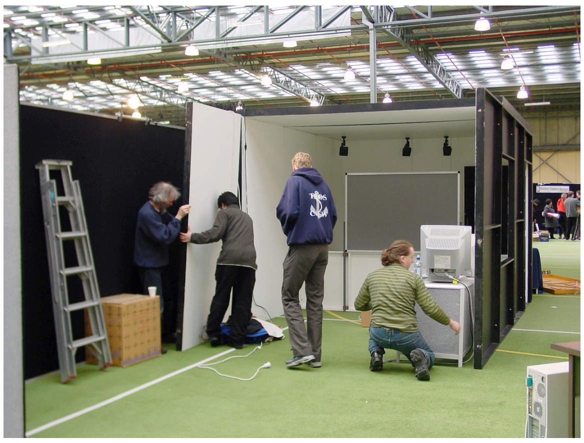


Figure 1 The Watching Window booth is being assembled for exposition purposes

In the current implementation of the Watching Window, the user enters a booth, a box about 3 × 2 × 2.5 meters, with a big screen on one side. The two cameras reside on the walls on either side of the screen. The booth is painted white and has bright lights on the walls. The user will become the one dark object in the booth, and in this way the cameras are able to track your hands and head.

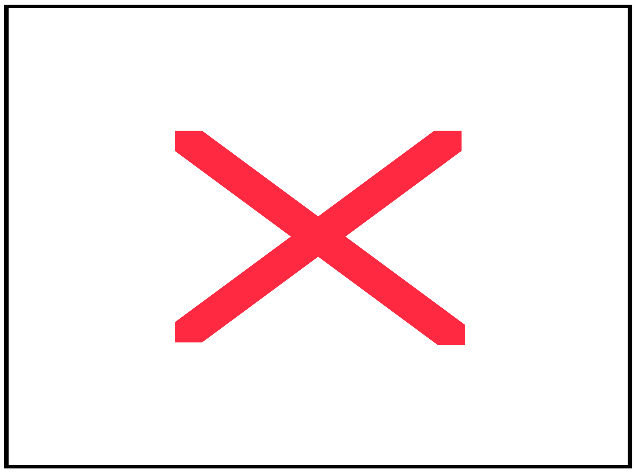


Figure 2 Top view of the watching window layout. The user stands in front of the screen and is observed by two tiny TV cameras.

The screen displays a 3D virtual world. The display is changed in accordance with how the user moves, and where the hands are. If the user moves the head, the simulated world is shown from a different angle. Interaction with the display is possible by pointing at simulated objects. The 3D effect is made even more convincing when using stereo glasses and 3D stereo display.

2.2 My contribution

This is my assignment as I wrote it at the start of the project:

"During my stay at the Graphics Lab, I will build a different vision system to do the eye tracking. The old system relies on a fixed white background and has some problems with varying lighting conditions.

The new system should be less dependent on constant background and constant lightning. The system should work real-time, or have at least an update rate of 15 Hz.

Several levels of improvement are possible. The hardware currently consists of two TV cameras, connected through special interface cards. The maximum frame-rate possible is 20 Hz. Is better hardware available? How can this be implemented in the system? Should we add another camera, or more?

The second level of improvement can be found in the software. The current vision system should be replaced, by a new system that does not depend on a fixed white background, but uses different means of recognizing people, their heads, eyes and hands.

My main focus will be on the software level. With my background and interest in computer vision, a better system is to be developed.

Chapter 3. Computer Vision

In this section I discuss the facets of the computer vision part of my project.

The processing of a picture starts by acquiring the image data. Then a set of operations is applied on the image data. The goal is to find such a set of operations, also called *filters*, so information on the location of the body parts in the image can be more easily deduced. The information that is found can be used to do more efficient image processing.

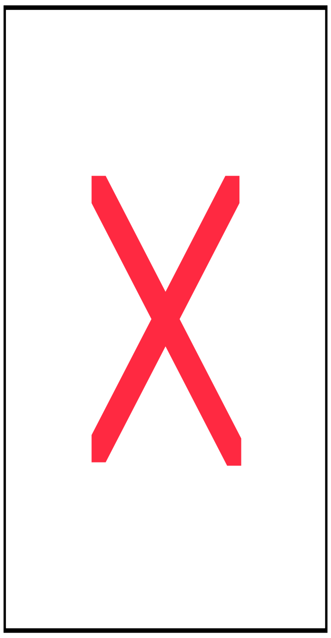


Figure 3 The computer vision process. An image is acquired, and then it is processed. Information can be extracted after processing, and used as feedback. Finally, body part coordinates are derived.

3.1 Data from the cameras

The first part of each vision system is the acquisition of the image data. This process involves converting photon-bombardments to electrical currents in the CCD. The currents are interpreted by a chip and converted to a signal for transport to the computers, which then processes them into an image. During this process there are many places where noise occurs.

3.1.1 Noise

This noise comes from different sources. The first real obvious one is that the image is just a set of discreet samples. By taking samples information is lost. This loss of information can influence the quality of the output image a great deal.

The second source of noise is the electromagnetic noise. We are working in a small area, with two computers with monitors that cause the signal that is being transported from the CCD to the PC to be influenced.

The third source of noise is the automatic balancing of the colours of the camera. When bright light shines on the CCD, it adjusts itself to be less sensitive, like human eyes do. The drawback in this is, that a pixels get a different colour value, although they are an image of exactly the same. For example, a pixel that was medium grey can become dark grey. The colour of people moving in front of the camera will thus also depend on the total amount of light the camera catches.

The Watching Window used two tiny TV cameras. The signal of the cameras is encoded as a TV signal, which consists of a 50 Hz interlaced signal. When trying to reconstruct an image out of that signal, movement will have a significant distorting effect, leaving one interwoven picture of the two interlacing passes. This causes unnatural images. Besides, it reduced the maximum frame rate to be 25 frames per second (fps).

3.1.2 Fighting the noise

It is not possible to take images without any noise. But is it possible to eliminate certain factors that cause noise. In the new set-up the TV cameras are to be replaced by Firewire cameras (IEEE-1394). The new Firewire cameras can grab images up to 30 fps, non-interlaced. In the implementation section I describe how to get the data from the cameras. To illustrate the differences between the cameras, I created a quick comparison.

3.1.3 Camera Comparison

	TV cameras	Firewire Cameras
Resolution(s)	320 x 240	320 x 240, 640 x 480
Colours	24 bit RGB	24 bit RGB
Signal Transport	Analog	Digital
Image Structure	Interlaced	Non-interlaced
Maximum speed	25 fps	30 fps (15 @ 640 x 480)

There is another difference: the colour saturation. I did not measure that, but I can say that the Firewire cameras have more colour saturation. This will benefit the vision process, because there is more information available.

3.2 Image Processing

In the next section I cover the image processing. I go through the different filters I implemented, and how they work. After that, I discuss how I combined those filters to extract useful information out of an image-sequence.

3.2.1 Movement

The first filter I implemented was the movement filter. It is based on the fact that it is only necessary to update the position of body parts when movement occurs. The problem is that movement is hard to define. The only thing that is measurable is the difference in colour of all the pixels. So I define movement as a difference in intensity (i) in each colour channel of each separate pixel. Then the amount of movement is encoded back in the pixel. To calculate the amount of movement (m) there are numerous different approaches available. I chose a computationally cheap one:

$$m_{channel}(t) = | i_{channel}(t) - i_{channel}(t-1) |$$

I calculate this for every colour channel: for red, green and blue. The total movement becomes:

$$total(t) = m_{red}(t) + m_{green}(t) + m_{blue}(t) \qquad (sum)$$

Another method:

$$total(t) = max\ (m_{red}(t), m_{green}(t), m_{blue}(t)) \qquad (max)$$

Both methods have properties why they should be used. The first method picks up more refined levels of movement. If there is movement in *blue* and *green*, the first one adds them together, so it results is a larger movement score. If there would only be movement in *green*, there would be less movement. Still, it should get noted as movement as well. The *max* version is in that realistic than the *sum* version.

When applying a threshold over this movement image, a binary image is formed, with areas that are moving, and areas that are not moving.

For the *sum* version:

$$i_{channel} = \begin{cases} 255, & if\ m_{channel} > \theta \\ 0, & otherwise \end{cases}$$

For the *max* version:

$$i = \begin{cases} 255, & if\ total > \theta \\ 0, & otherwise \end{cases}$$

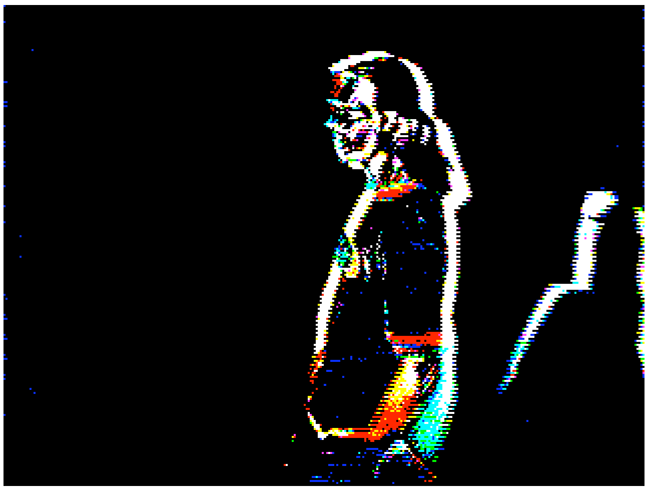


Figure 4 Movement filter in action. The movement is displayed with an applied threshold over each colour channel, using the *sum* method.

3.2.2 Skin colour

A common used technique in face detection is to find all parts in the image that are skin coloured. Studies (for example [1,6]) have shown the skin colour can be modelled as a single Gaussian distribution, or a mixture of them. I used only one, since research has shown that one is sufficient, considering all the noise effects.

To create such a distribution more effectively, the colour-space can be transformed to a *chromatic* colour space, where illumination (brightness, luminance) is removed. To achieve that, the colours can be converted to a chromatic colour space.

To convert a colour from RGB to the chromatic colour space, the following formulae apply:

$$\begin{aligned} c_{red} &= red \;/\; (red + green + blue) \\ c_{green} &= green \;/\; (red + green + blue) \\ c_{blue} &= blue \;/\; (red + green + blue) \end{aligned}$$

$$c_{red} + c_{green} + c_{blue} = 1$$

where c_{red} , c_{green} and c_{blue} are the chromatic red, green and blue.

The Gaussian distribution is found by analysing 76,800 samples of skin colour, taken from different persons, under different lighting conditions.

For every pixel the change of that pixel being skin coloured, P_{skin} , is evaluated under the Gaussian. This results in a 256 level greyscale image, where 0 denotes a $P_{skin} = 0$, and 255 a $P_{skin} = 1$.

By applying a threshold this image can be converted to a binary image, with the non-skin colour class pixels as black, and the skin colour class as white.

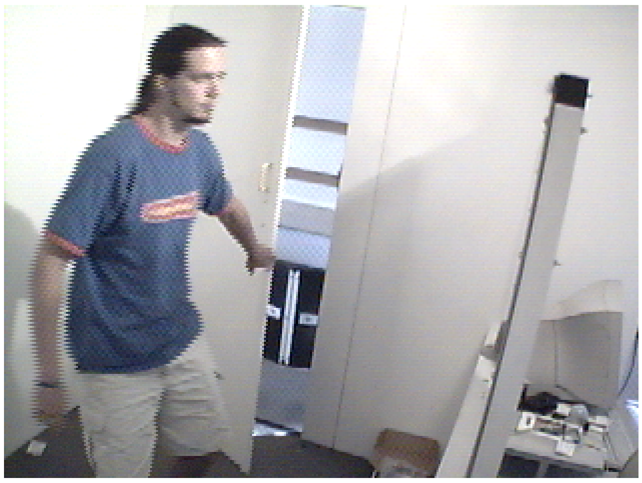


Figure 5 The original image

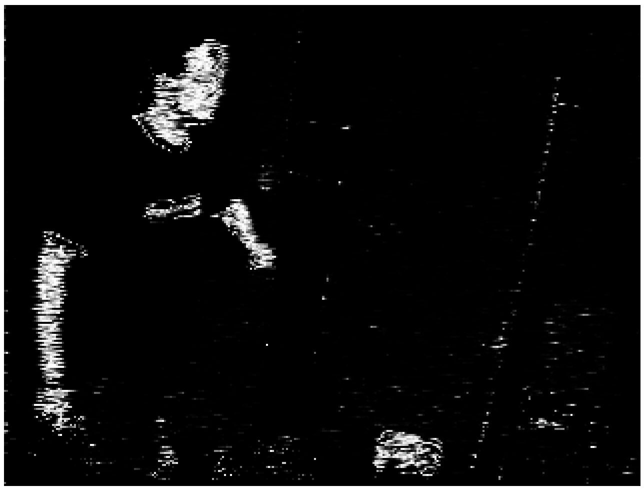


Figure 6 The image displays the P_{skin} of all pixels

3.2.3 Edge detection

Finding boundaries for body parts, like the head, can be done by edge-detection. A standard way of implementing edge-detection is by doing a convolution over the image with the *Sobel* kernels.

$$\begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

Figure 7 The Sobel kernels

When applying the first of the Sobel kernels, a resulting image will contain the gradient of the image in the y – direction (G_y), the second one gives the gradient in x – direction (G_x).

Combining both the gradient images can be done in two ways:

Strength:

$$i = sqrt(G_x^2 + G_y^2)$$

Orientation:

$$i = arctan(G_x / G_y)$$

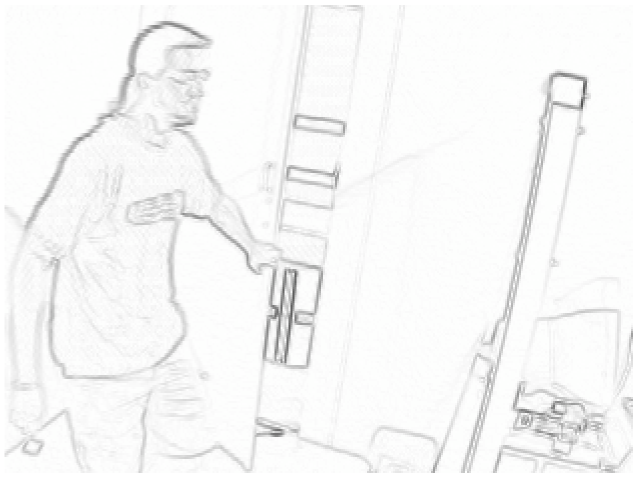


Figure 8 Edge strength filter in action

For edge orientation I use a slightly modified formula:

$$i = \begin{cases} |\, arctan(G_x / G_y) \,| & , \textit{if strenght} > \theta \\ 0 & , \textit{otherwise} \end{cases}$$

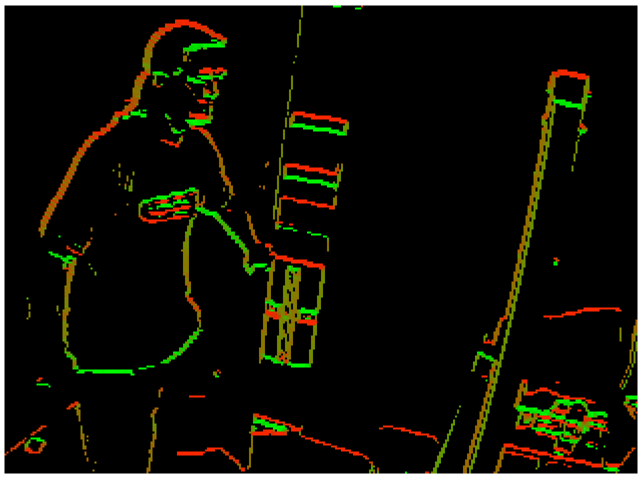


Figure 9 Edge orientation performed on an image. Red denotes a dark to light gradient from top to bottom and green a gradient from bottom to top. Horizontal direction has less significance and is therefore not coloured separately.

3.2.4 Silhouette from mask I

This *Silhouette from mask I* filter uses the *movement*-filter to create an image with only moving pixels.

For each row in the image, the first and the last moving pixel are found. All the pixels between them are filled with they original colour. The same operation is applied in the vertical direction. The resulting pictures will look like this:

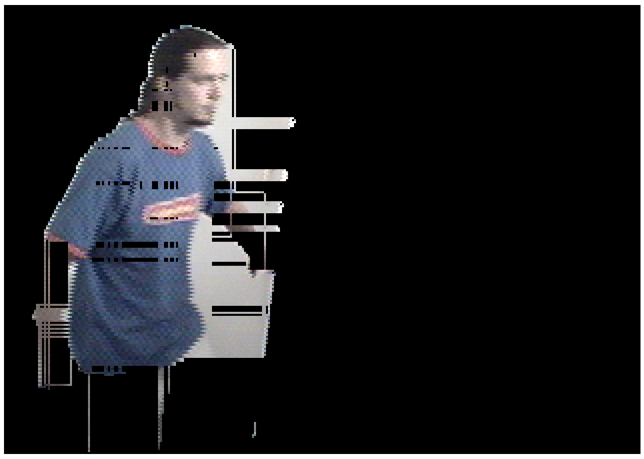


Figure 10 Silhouette from mask I filter in action

3.2.5 Silhouette from mask II

The *Silhouette from mask I* filter, which result is show in fig. 10 is not really reliable; it is much too sensitive to noise. As can be seen in the figure, there is a large part visible that is actually not moving. Another disadvantage of the *Silhouette from mask I* method is that it is computationally slow, as it needs to scan the entire picture.

Therefore I developed a rather simple, but robust method. It is based on simple binary image logic.

For each pixel the next is considered: if the pixel itself is moving, and at least 2 neighbours are moving, it will cause a 16 by 16 area around the pixel to be filled in the output image.

Afterwards, a quick horizontal scan is done to try to fill small gaps still appearing in the output image. Each pixel is checked if it is lit. If it is, the pixel at distance *window* away is scanned as well. If both pixels are lit, the entire gap is filled.

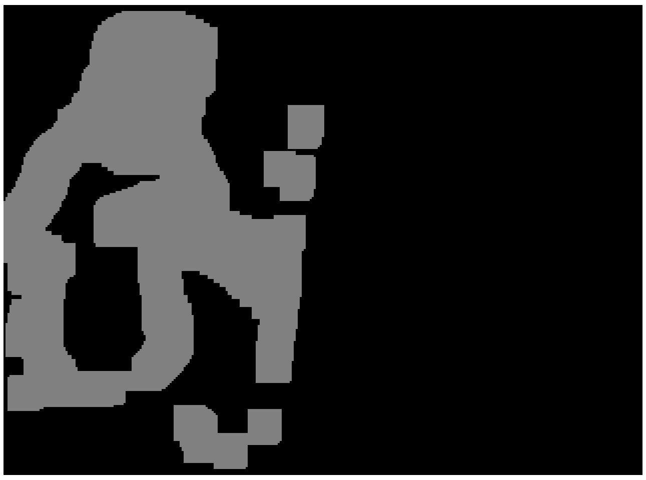


Figure 11 Result of Silhouette Masking II. Although the result may look a little worse than the other silhouette masking routine, it is more consistent and more useful.

3.2.6 Head finding in silhouette

3.2.6.1 Top finding

Top finding is a relatively easy process; the image is scanned from top to bottom and from left to right. As soon as a pixel is found with a non-background colour, it is marked as the top pixel. From the top pixel, we start the head-region finding [4].

3.2.6.2 Head-region finding

Starting from the upper-left pixel, the image is scanned with the following algorithm:

```
dy=0, x=top.x, y=top.y

while (dy < maxdy)
{
   if (x == left  ) return false;
   if (y == bottom) return false;
   if (pixel(x, y) == FOREGROUND)
   {
          x--;
          dy=0;
   }
   else
   {
          y--;
          dy++;
   }
}
left = x;
```

The image is scanned form the top pixel left. As soon as a pixel is found that does not belongs to the silhouette, scanning down starts. When a pixel of the silhouette is found, the algorithm start scanning left again. However, when after a number of steps no pixel is found while scanning down (it found a gap), a side of the head is found. This is most likely to be the side of the head on the left side.

The same algorithm is applied on the right side, finding the right side of the head. When instead of finding a gap the algorithm scans to the bottom, or to the edge, it can be safely assumed there is no interesting head to be found.

An example is shown in fig. 12.

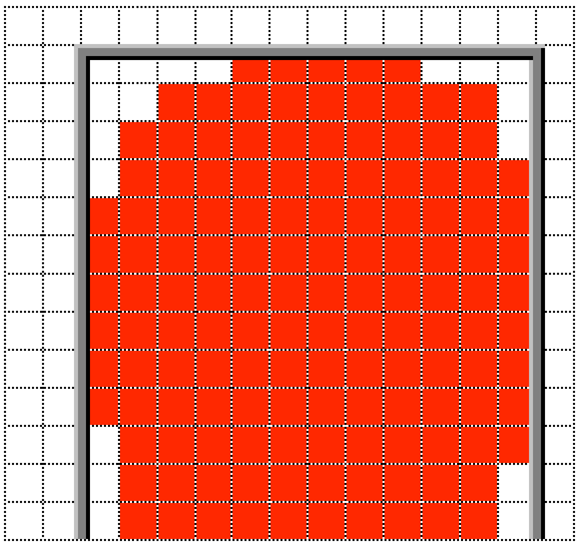


Figure 12 A head in pixels

3.2.7 Finding blobs

A binary image very often contains fields of pixels that are connected and somehow belong together. To find those connected-pixel fields, an algorithm is performed that finds those fields. It could be implemented by storing all pixel locations in a special container, but that would be to computationally too intensive. A work-around is to label all pixels belonging to the same class the same colour in an image.

As well as a separate label, there is other meta-data on the pixels. The algorithm also calculates the centre of mass of the pixel field, the bounding box and the number of pixels in the field.

3.2.8 Edge bending analysing

To try and find more information in the head silhouette, the shape of the edges from the top of the head to the neck can be analysed. As a person is seen “en profile”, this method can be used to find likely locations of the nose. If constraints are taken into account, for example that the nose is somewhat in the vertical centre of the head, this method can make a fair estimate of the nose location.

The method scans along the edge and marks the locations where the edge changes direction from left to right and vice versa. The nose is of course a profound landmark on the face and will introduce a nice bend in the edge.

An example of how this process works:

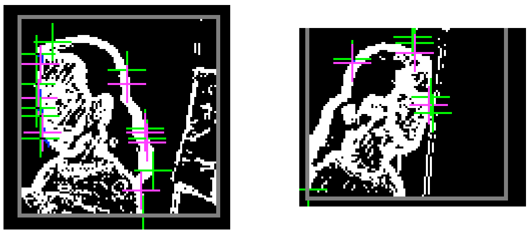


Figure 13 Edge bend analysis. Green crosses represent a change in direction in one way, purple represent a change in direction in the other way. The nose is characterised as having both changes in direction, at a certain distance from each other, and in a certain region in the head.

3.2.9 Enhance contrast

Eyes are the dark spots inside a head-region. To find eyes it would be useful to enhance the contrast in the head region, in order to make the eyes even darker. The method used for the enhancement of contrast is histogram equalization. [2]

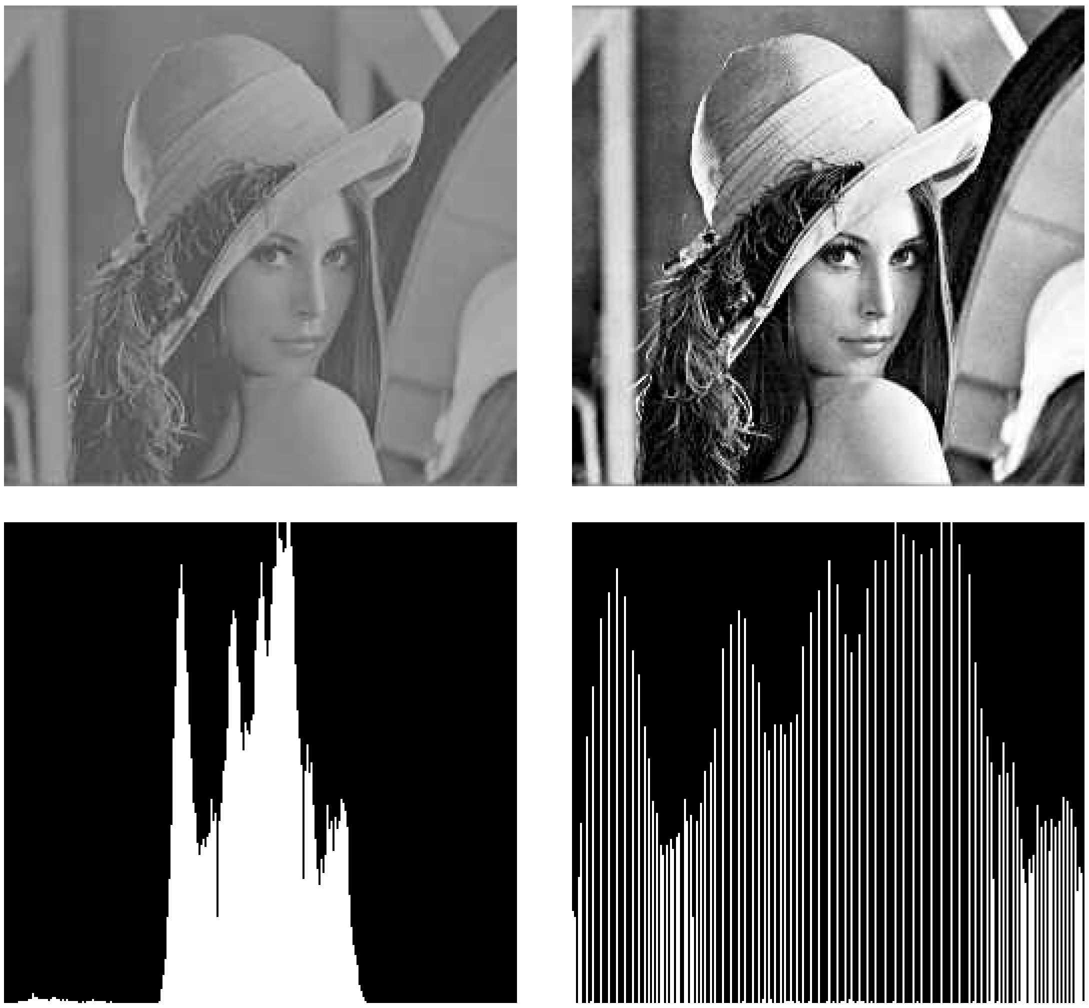


Figure 14 Histogram equalization in action. On the left an image that is lacking good contrast. In the histogram below the image it can bee seen that there are no real dark, nor real light colour in the image. If the histogram is stretched, as can be seen in the lower right image, the resulting image has more contrast.

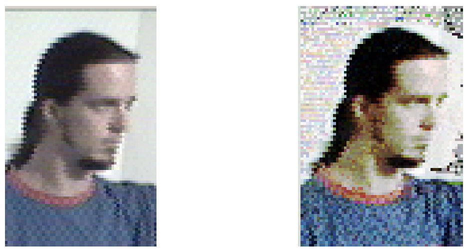


Figure 15 Histogram equalization on a test subject

3.2.10 Enhance local contrast

Another way of enhancing local contrast can be done in the method described next.

For each pixel the region around it is examined. The maximum and the minimum values of the region are stored. The distance from the pixel to the max value and the min value is calculated. The pixel will get the colour of whichever extremum it is the closest to.

In formula-form:

$$Pixel = \begin{cases} max, \text{ if } |max - pixel| > |min-pixel| \\ min, \text{ otherwise} \end{cases}$$

Examples:

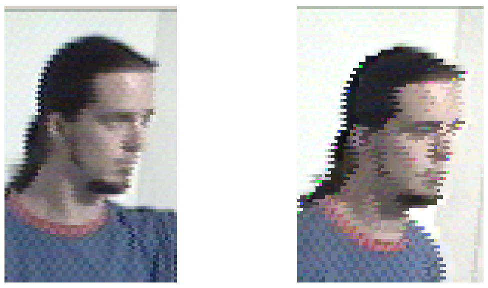


Figure 16 Local contrast enhancing

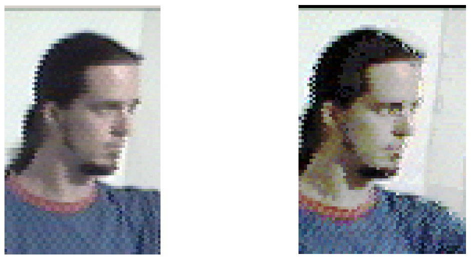


Figure 17 Both methods combined

3.2.11 Fast blur

The *fast blur* filter only is faster than the normal blur, if the area that is averaged is greater than 2x2 pixels, and if it is done on many location in the image. [REFERENCE]

For each location in the image, the sum of all pixels that are in the rectangle formed by the pixel and the upper-left of the image, is stored.

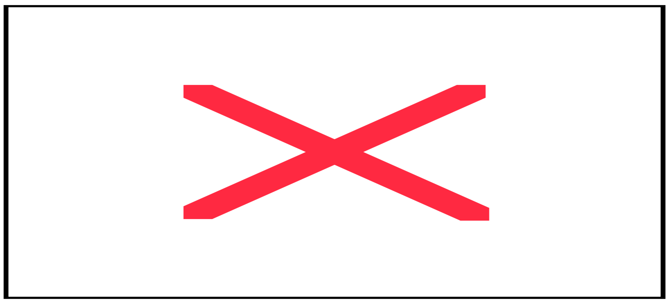


Figure 18 In the left square the values of pixels before calculation the sum. In the right image each pixel contains the sum of itself and all neighbour pixels in the upper-left area of the image.

The brightness in an area around a pixel can then be calculated using the following formula:

$$avg = (upper\_left\_limit + lower\_right - lower\_left\_limit - upper\_right\_limt) / size$$

In fig 18 the average brightness of the coloured square is, calculated using conventional blurring:

$$(5 + 6 + 8 + 9) /4 =7$$

This is equal to the fast blur result:

$$(45 + 1 - 12 - 6) / 4 = 7$$

The *lower_right* value has to be lessened by the *lower_left_limit* and the *upper_right_limit* to calculate the value in the area. But then the value of *upper_left_limit* is subtracted twice, so it has to be added that back to the total. Then to normalize the value, it is divided by the size.

This process is done for the total intensity, but can easily be extended to process each colour channel separately.

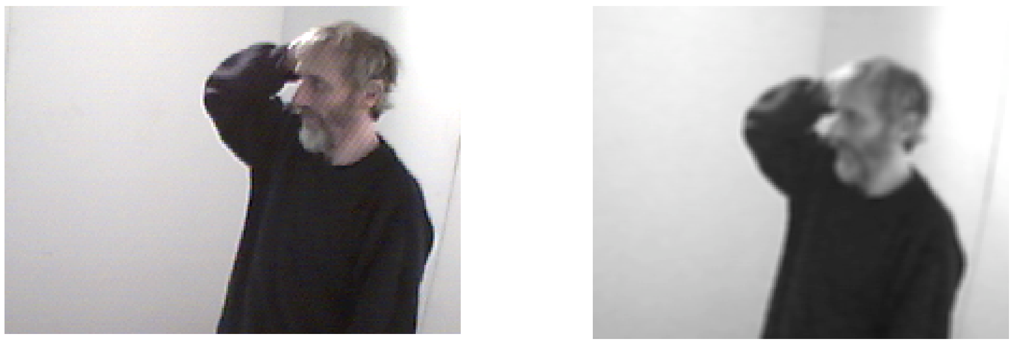


Figure 19 Geoff. In the left image, he is shown as the cameras see him. In the right image fast blur and a grey scaling is applied.

3.2.12 Face division

This filter is capable of roughly finding which way the user is looking. To do this, an area in the image where the head is most likely to be found is examined.

The area is divided in two triangular parts, by drawing a line from the upper left to the lower right, see also fig. 20. For each of the areas the average brightness and the average hue are calculated. The same operation is performed on the other division, with a line from the upper right to the lower left.

Then the division that causes the biggest difference is considered the division that decides where the head is looking. If the division was made using a line from the upper left to the lower right, the user is assumed to look left. If the differences are minimal, within a certain range, the user is assumed to look frontal (facing the camera or looking away from it), else the user is assumed to look right.

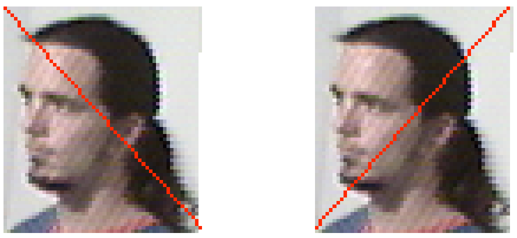


Figure 20 Division in left image has a greater brightness and hue difference than division in the right image, so the user is assumed to look to the left.

3.2.13 Mixing filter

The mixing filter takes in two images, taking a weighted average of them both.

It uses the formula:

$$new = \alpha \_ \ incoming + (1 - \alpha)\_ \ old$$

An example if found in fig. 21.

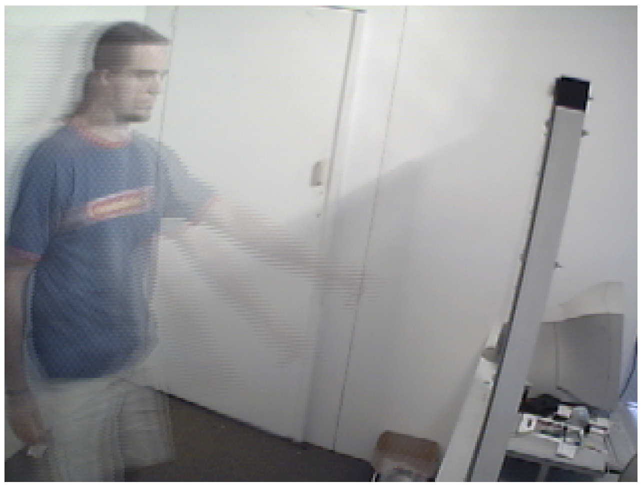


Figure 21 Mixing frames (notice the fading arm).

Images that are slowly updated and used for tracking are called Temporal Textures. It is like a photograph that is taken from (for example) the head. When the head is found again in the next frame, the photograph is updated a little to match the current match better. So in time the photograph, the texture, is slowly updated. Therefore it is given the name "Temporal Texture".

3.2.14 Spiral matching

The spiral matching is used to find a small image in a large image. It uses a hint where the small image can be found. This hint is in general the position where the small image was found in the previous frame.

If for example we found the head in the last frame, take a small "photograph" of it. We try to find that piece of image again in the next frame. The search starts at previous location where the photograph is taken. Then it searches around that location, in a spiralling fashion.

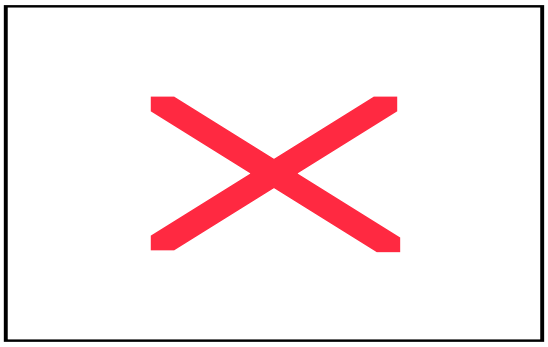


Figure 22 Spiral matching described schematically.

A maximum number of searches is defined, and the result of this filter will be the coordinates of the best match. Ideally, when the user is not moving, the coordinates are exactly the same as in the last frame.

3.2.15 Snakes

Snakes, or Active Contour Models, are used in several computer vision projects [5]. They allow searching for structures in the image that are not rigidly defined. In this project, they are perfect for finding the silhouette of the head. Although I have not completely implemented snakes, I think they can be a great help to find a nice head silhouette.

Snakes could be used in a very simple form. They consist of a set of control points, connected by straight lines. Each control point has a position, given by its coordinates in the image. The number and coordinates of its control points specify the snake. Moving the control points makes adjustments to the snakes.

The snake to be used has a closed loop in the image, though this is not necessarily true of snakes in general.

The points can be seen as mass points in a mass-spring system. In this system there are two forces that are applied to the mass points

1. Points move towards the line defined by its closest neighbours
2. Points move along the image gradient

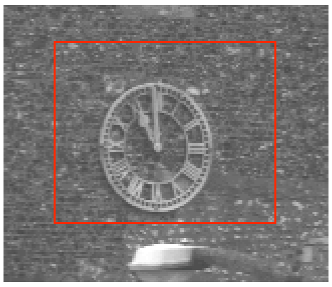


Figure 23 Snake in initial position

When starting in a position like fig 23, the points tend to move closer together. Because there is no profound gradient, they will keep moving until they reach the clock.

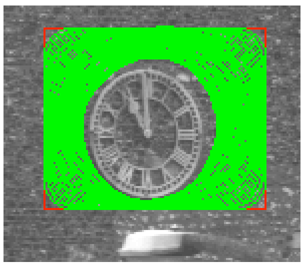


Figure 24 Snake during iterations

Once the points of the snake are close to the clock, they are pushed back by the strong gradient that is caused by the colour difference between the wall and the clock

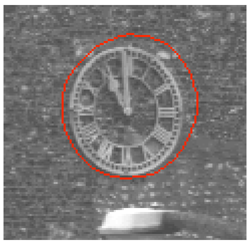


Figure 25 Snake after 20 iterations

After a certain number of iterations there is only a small amount of movement left of the control points of the snake. When the amount of movement falls under a certain threshold, the snake is said to be *stable*, and is no longer updated.

The snake system could be used after the first (rough) movement filtering. The bounding box around the head, found after analysing the silhouette, can be a good start. To make the snake work optimally, there has to be an artificial separation between the head and body.

The control points of the snake then define a good approximation of the head shape and can serve as basis for better algorithms.

3.3 Information Extraction

In this section I cover the Information Extraction phase of the computer vision process. A short explanation of the difference between localization and tracking is given. After that I discuss how to apply localization and tracking on the head. Then the same is done for the hand and the eyes.

3.3.1 Localization and tracking

When a specific body part is not know in the previous frame, the body part has to be found, to be *located*. If the body part is known however, we can switch to *tracking*. These two modes differ a lot. Localization tends to be slow and cumbersome, and tracking is generally fast and simple to understand. In the next sections I discuss how localization and tracking are done for head, hand and eye.

3.3.2 Head finding

In the next part I describe how I do the head finding. There are two approaches; each of them uses a combination of filters described before.

3.3.2.1 Method I: Movement, Skin colour and Finding Blobs

Localization
In short the localization algorithm looks like this:

1. Find the moving pixels.
2. Find all skin colour pixels.
3. Find all connected pixel areas with size greater than θ_1.
4. Find the topmost area.
    a. Take the centre of that area and label it the position of the head.
    b. Take the bounding box around that area and label it as head-box.

In *step 1* the moving pixels are found. The threshold for moving pixels is set very low, so there is also a little noise that is picked up as movement. Because humans don't stand entirely still even if they try, there will be a little movement, and the cameras will pick that up.

Step 2 eliminates all pixels that were found in *step 1*, but don't have skin colour. This removes a large part of the noise.

After s*tep 3* most of the noise is removed. Only large areas of skin coloured and connected pixels are left. It is very likely that a head and hands are found in these pixels.

Step 4 labels the found connected pixels as head .

Tracking
When the head is known the algorithm works in a similar way, but instead of looking through the whole image, it only scans a small area around the last found head and hand locations.

Results

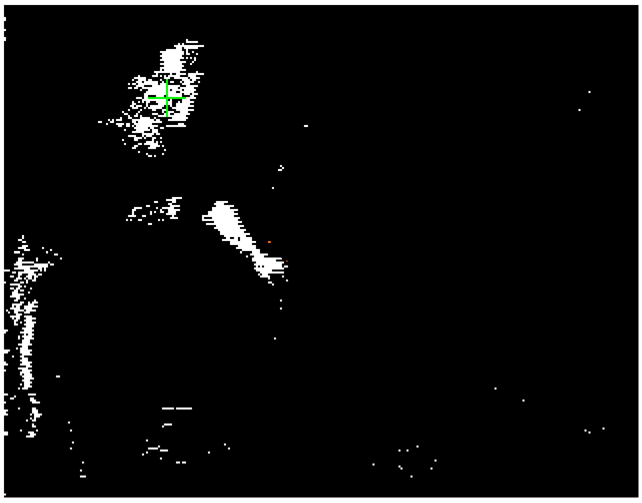


Figure 26 Head correctly localized. It has a green cross on it.

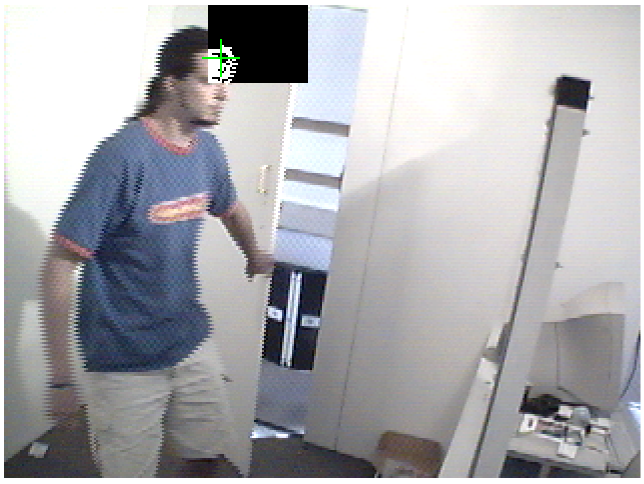


Figure 27 Head in tracking mode.

Disadvantages
This method has several disadvantages. The main weakness is the skin colour segmentation. It will classify pixels as skin colour when the user is wearing reddish or beige clothing. And the method, although using chromic colour space, is sensitive to lighting changes, due to the automatic level correction of the cameras.

Another disadvantage is that the movement-threshold cannot be too high; otherwise no moving pixels will be left. But if set to low, the carpet on the floor will be identified as moving skin as well.

The blob finding is computationally slow, and its speed will decrease dramatically if there is a lot of movement in the image.

Conclusion

This filter works, but not good enough. It can only be used to gain some information, but it is not reliable enough to deduce the head location, nor do head tracking. Useful as a start, but better method has to be found.

3.3.2.2 Method II: Silhouette masking, Head finding in silhouette, Spiral matching of Temporal Textures

The second combination of filters combined movement into silhouette, finding the head in the silhouette and doing tracking by using temporal textures and spiral matching.

Localization
In short the localization algorithm looks like this:

1. Find the moving pixels.
2. Find a silhouette
3. Find the head, using the top down edge walk
4. Check if the head has a head-size
5. Check if the head is skin coloured
6. If all checks are OK, assume the head is found and create a temporal texture, and update the TipTable

In *step 1* the interesting objects are located. Not only the main moving person will be found, but most likely also the shadow and some other noise. If the threshold is chosen carefully, most noise will disappear. A good value for the movement threshold is proved to be 50, which means the total difference must be 50 or greater for a pixel to be marked “moving”.

Step 2 and *step 3* will find a head shape as described in *Head finding in silhouette*. This head shaped object may in fact be not a head at all. To make the system more robust, the found object has to be verified. This verification process consists of two fast tests: size and skin colour check.

The size of the object found has to be a reasonable head size. This size can be adjusted in the program, minimum width of 20 pixels, and maximum of 100, are useful borders. These border values are not too strict, but offer the flexibility needed for the user to move closer to and away from the camera, and still be detected.

If all succeeds, a snapshot of the head is taken as a basis for the Temporal Texture. All the information found in the head detection process is stored in the TipTable.

Tracking
The tracking takes place when the head was known in the previous frame. It is based on the temporal texture of the head.

The algorithm:

1. Start to match the texture at the old position of the head
2. Try to match it to the neighbours, in a spiralling way
3. Stop after a certain number of searches.
4. Consider the highest match:
    a. If bigger than a threshold θ, then
        i. Make sure there is no movement above the found match
        ii. Make sure the found match contains enough skin colour.
        iii. Update the new position to be the position of this match.
        iv. Update the temporal texture to the new found match
    b. If smaller, consider the head "not found anymore" and start localizing it again

In *step 1* the temporal texture is matched to the sub image in the new frame, at the position where the head was found in the previous frame.

The match is calculated using the following score:

$$total\_difference = sum_{pixels}\ (difference\ (pixel_1\ ,\ pixel_2))$$

$$score = 1.0\ /\ ((total\_difference/size) + 1.0)$$

The difference method used is the same as in the *movement* filter.

In *step 2* this process is repeated for all the pixels in the neighbourhood of the old head position. The head will most likely not have moved much. So it is good to start looking around the old position and slowly spiralling outside. After a number of those matching steps, the search is stopped and the scores will be considered.
If no score was greater than a certain minimum value, then the head is considered lost, and the system will try to localize the head in the next frame. Otherwise, when there is a valid score there is a series of fast checks that try to make sure the head has been found, in the right position.

When the user in the booth is wildly banging his or her head, and wearing clothes that resemble the face colour, it may occur that the algorithm matches an area around the chest. To verify if it has indeed found the head, the area above the match is scanned for movement. The head should be the highest moving object in the image. If there is movement above the found match, it is likely a mismatch was found. Localization is then done in the area above the match, and if a head-like object is found in that area, the process starts again, considering the new possible head position.

If condition 4.a.ii. fails, if there is no skin colour found, the head is considered lost, and the localization step is taken again.

Results

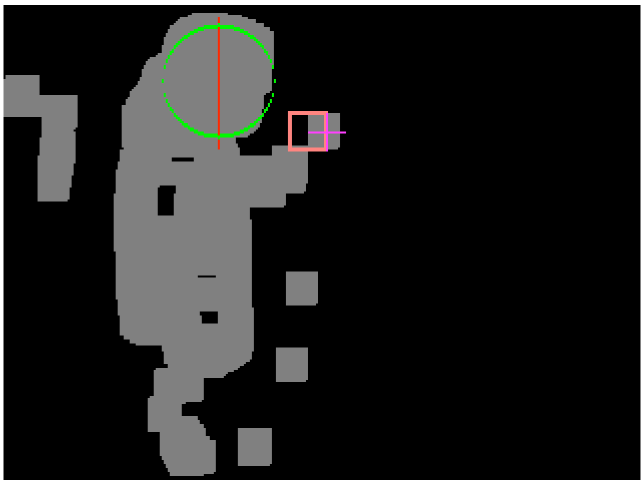


Figure 28 Localizing the head (green circle).

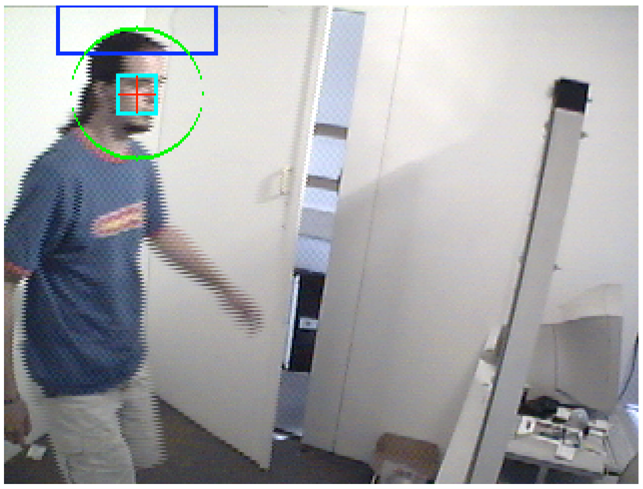


Figure 29 Example of filter combination (Tracking).

Disadvantages

The main weakness in this method is the computational power it takes to do the matching. To compare an image with another currently requires all pixels to be considered. This can be solved using Monte Carlo matching.

Conclusion

This filter works well. It can be used for head localization and tracking.

3.3.3 Hand finding

Finding the hands in the image is more difficult than finding the head. The head is bigger, has a certain size that does not change much over time, and does not move around fast.

Hands on the other side, can move fast, can change shape quickly, and are smaller than the head.

For hand finding I use almost the same techniques as head finding *Method II: Silhouette masking, Head finding in silhouette, Spiral matching of Temporal Textures*.

The main difference is the direction to start looking for moving pixels. While in head finding this will be fro the top, in hand finding it will be either form the left, or from the right, depending on the direction the user is assumed to look.

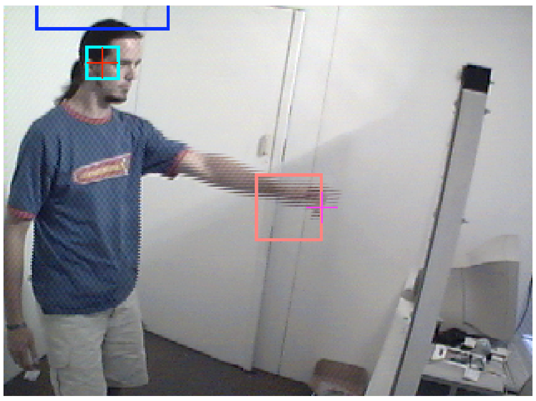


Figure 30 Head and hand tracking.

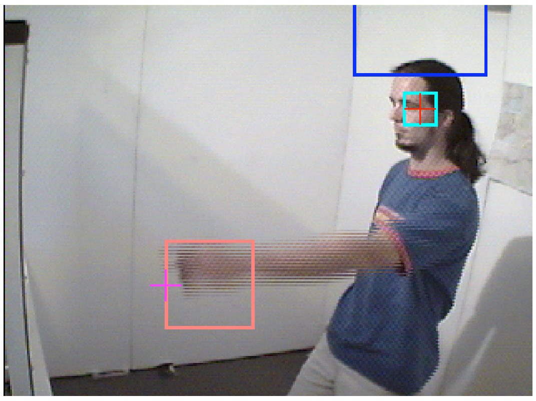


Figure 31 Head and hand tracking.

When the hand moves fast, the movement silhouette is not the best approximation. Movement includes the place where the hand was in the frame before and the movement becomes bigger than it is in reality. A snake around the hand, just as proposed to use in the head finding would increase the reliability.

In the images above the purple cross indicates the hand position as the system sees it. Note the interlace effects on the arm.

3.3.4 Eye Finding

Once the head is found, it is time to go and look for the eyes. To find eyes in a head, some assumptions can be made:

- The eyes are dark compared to its surroundings

Figure 32 In the left image Geoff's head is shown normally. Eyes and eye sockets are seen as dark patches in the face area. Normally it would not be easily noticed, because the human brain automatically compensates for this. But when an image is turned upside down (right image), it becomes clearer.

Other assumptions:

- An eye is not bigger than 1/10 of the total head size
- An eye is bigger than 3 x 2 pixels
- Eyes are vertically located halfway the face

With these assumptions the search for the eye becomes less difficult.

3.3.4.1 Using local contrast

Localization
My first approach was just to look for dark areas in lighter areas. Ideally, the resolution to look for eyes is dependant on the head size. But because the head size is not really accurate, only an approximation can be made. To find eyes the head is scanned at several resolutions, and the results are added.

The area is scanned, and for each $n \times m$ block of pixels the average brightness is calculated. Then the average brightness of all the surrounding blocks is calculated as well.

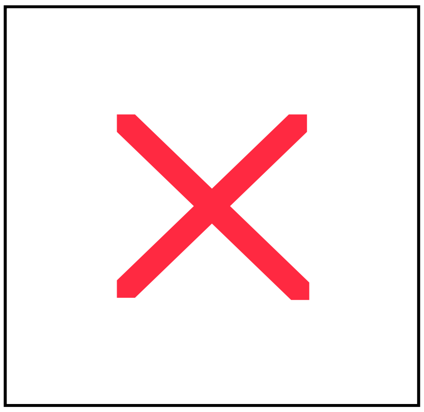


Figure 33 Centre area and neighbours

There are several methods of calculating the matching score. The first:

$$score = sum_{neighbours}\ (\ |\ c\text{-}neighbour\ |\ )\ /\ size$$

This scoring method had a big disadvantage: if there was an average centre, with light and dark areas around it, it gets a high score as well.
The second method:

$$score = c - avg_{neighbour}$$

This function works reasonably well, but also has the drawback that one extreme neighbour can confuse the entire system.

The third method:

$$score = sum_{neighbours} (c - neighbour) / size$$

In this method, if there are light and dark neighbours, they cancel each other out, lowering the total score and thus preventing false hits. This is the method I finally used.

To maximize the results, the area that is looked at processed using both the contrast enhancement filters.

Results

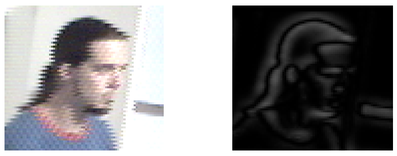


Figure 34 Before and after finding dark in white

Conclusion

I implemented both the contrast enhancing filters, but they did not improve the situation. Even on a person with a high contrast ratio between eyes (eyebrows) and face, it does not give a clear result.

3.3.4.2 Using edge information

Instead of looking at whole areas, it might be more useful just to look at gradients in the area. First only strength is considered; later on orientation is also used.

Edge strength

Filtering based only on edge strength. As can bee seen in the images below there is no clear eye found in the image.

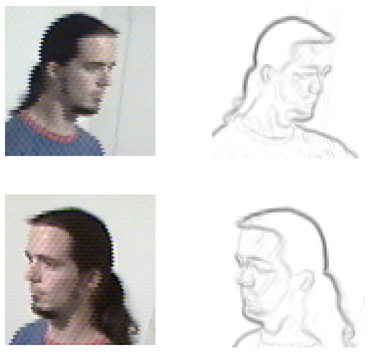


Figure 35 Edge strength results. The left images are the original; the right images represent the edge strength. The darker pixels represent the stronger edges.

Even though an image of a person with a high skin – eye contrast is processed, the information in the output image is not of high enough quality to extract information.

Edge orientation

Using edge orientation proved to be the most successful method I tried. The functionality is about the same as looking for dark areas in white areas. Instead of looking for white areas however, in this version areas that are red and green are considered.

The finding algorithm searches for areas where the top half of pixels are red, and the bottom half of the pixels are green. Areas with many red and green pixels get a higher score.

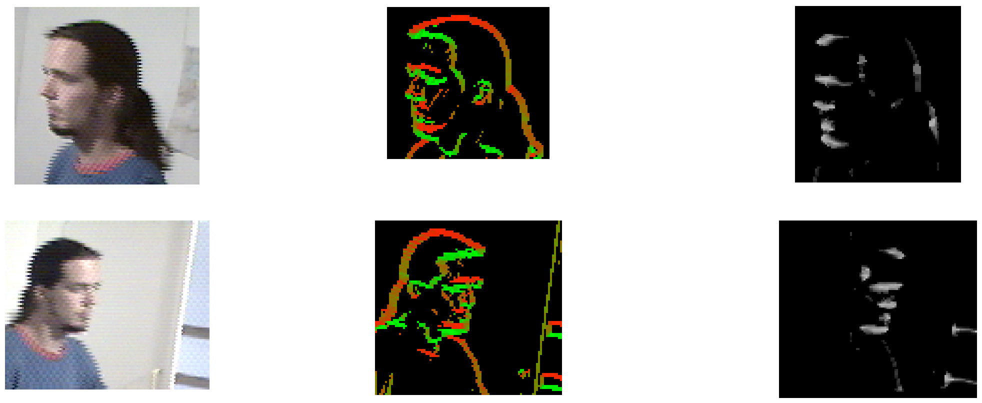


Figure 36 Results of the edge orientation method. Left the original images. In the middle the result of the *edge orientation* filter. In the right are the head area is scanned for eye-patches with size 4x4, 6x6 and 8x8. The scores are added together. The lighter the pixel is, the higher the chance is that it belongs to an eye. No extra information is used. When filtering the images on the right on for example the position within the head, and the head orientation, a fair estimate can be made on the eye position.

Conclusion

Edge strength alone is not enough to extract useful information. Edge orientation gives a fair result. However, when the methods are applied to a person with blond or grey hair, the results will become worse. This is the reason for not using eye finding in the final product.

3.3.5 TipTable and deciding where the head and hands are

All information is stored in the TipTable. After a complete image-processing step the TipTable can be used to look up information, like region of interest, head bounding box, head centre, temporal textures etc. This information can also be used in the next image processing iteration, so the image can be analysed more effectively.

Currently there is just one method used to determine the position of the head and the hand. In later versions, more algorithms may run in parallel and then all the information in the TipTable can be used to do an average, or a voting algorithm.

Chapter 4. Vision Software Architecture

4.1 Former situation

In the former situation there was no real architecture. Most functionality was stored in two big files. Then there was a separate tracker module, and a separate network module. For the camera calibration information there was a separate structure, and there was a structure for holding all the information on the frame grabbing. In short it could be viewed as this:

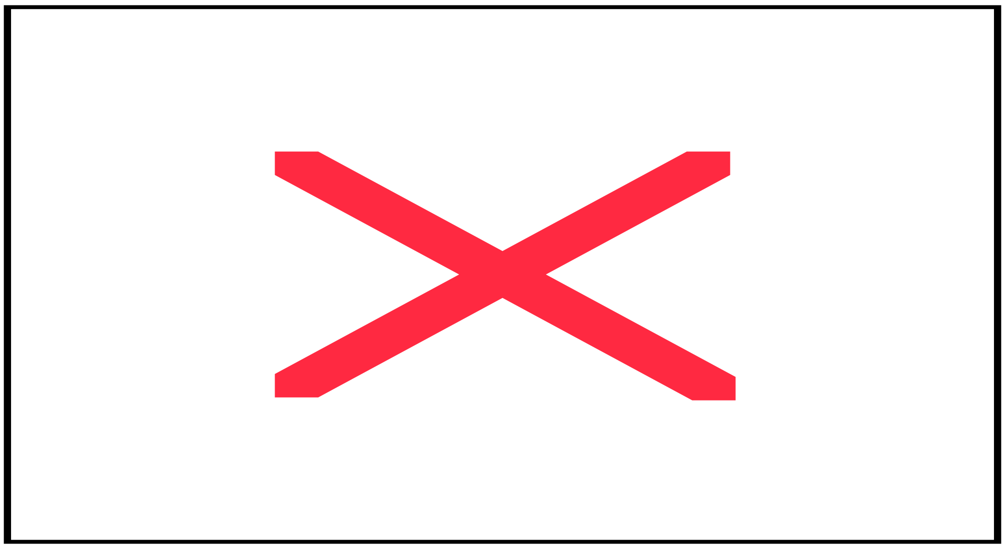


Figure 37 The former architecture as I reconstructed it. All the camera handling, display-mode information, background subtracting was done in the one main module. The image-processing code was stored in a separate file.

4.2 Current situation: separation of concerns

In the old system there was no separation of concerns. Different functionality was implemented in the same files and classes and it was not clear which class should be responsible for what tasks. I improved that situation.

4.3 Information flow

To better understand how the architecture works, it is useful to know how the information flows through the program. In the next diagram I will show this. The

image data is captured in the *Frame source*. Then it is passed on to the *Frame grabber*. The *Display mode* selects which *Image Functions* are to be applied to the image data, and how. The result will be shown in the *Output window*.
After the information on the location of the body parts is available, it is send to the *Camera client*, who will transfer it over the network to the 3D server.

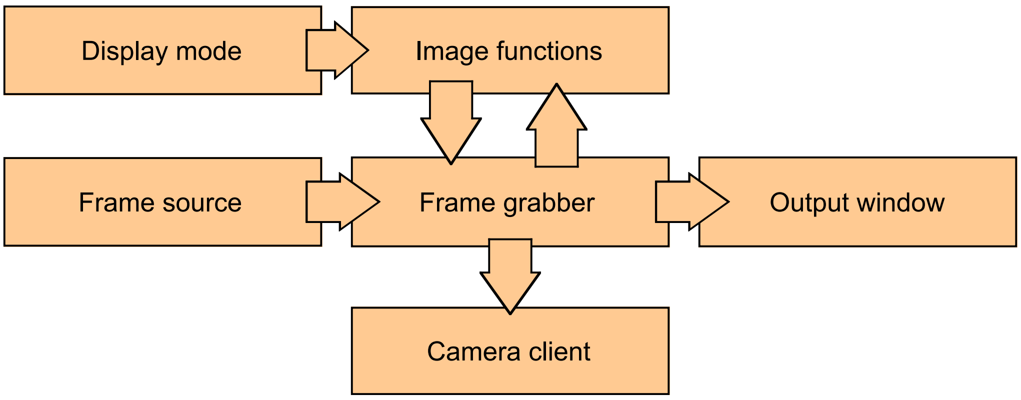


Figure 38 The information flow.

4.4 Class description

To show the new architecture, first a (slightly simplified) class diagram is given in fig. 39. Then I will go into some more detail in how the classes work.

4.4.1 Class diagram

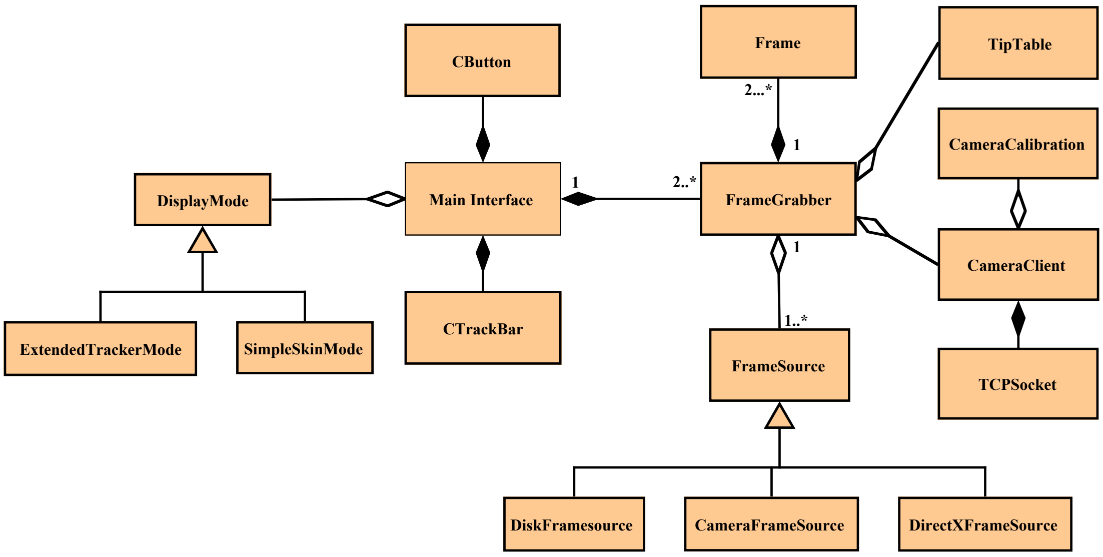


Figure 39 Simplified software architecture. Some classes are omitted to increase readability.

4.4.2 Blob

A blob is an information object; it currently describes an area of pixels (rectangle) in an image that have certain attributes in common. The pixels in the area that fulfil the requirements are known collectively as the blob. However, I do not store the pixels themselves in the blob, but only some information, for example: BoundingBox, CenterOfMass, NumberOfPixels, TotalX and TotalY.

4.4.3 BoundingBox

Simply said, a bounding box is just a rectangle. Most of the times it is used as the smallest rectangle that fits around a clouds of pixels.

4.4.4 CameraClient

The camera client class is responsible for the connection to the 3D server. It uses a TCP/IP connection to connect, and has its own protocol for talking to the server. More information on the network part in found in the Network chapter. For each framegrabber a camera client is defined.

4.4.5 CButton and CTrackBar

Graphical interface classes used to display trackbars (sliders) and buttons on the application.

4.4.6 DisplayMode

DisplayMode is the base class for all the display modes. The main program has a currentdisplaymode, which decides what to do with the incoming frames, how to process them and what to output.

In the current implementation the subclasses are the next:

4.4.6.1 ColorNormalizeMode

In this mode the normalized colours are displayed. The colours are normalized by removing the effect of brightness on a pixel.

4.4.6.2 EdgeDetectionMode

This mode performs the edge detection, strength based. The resulting image contains pixels with a grey-level between 0-255, where 0 is a zero-strength-edge, and 255 is the maximum edge.

4.4.6.3 EdgeOrientationMode

This mode performs the edge detection, based on orientation. The resulting image contains pixels with a red to green level, where red means an edge is top to bottom, and green the other way around.

4.4.6.4 EnhanceContrastMode

In this display mode, the contrast is enhanced by histogram equalization.

4.4.6.5 ExtendedTrackerMode

This is the main mode, where head and hand tracking are implemented.

4.4.6.6 GrayScaleOutputMode

Converts the image to greyscale.

4.4.6.7 MixMode

MixMode alpha blends the new frame to the current state frame using the *mixing* filter.

4.4.6.8 MovementMode

Uses the *movement* filter to display the images.

4.4.6.9 RawOutputMode

Displays what the cameras see. This mode is useful for making screen dumps.

4.4.6.10 SilhouetteMode

Displays the silhouette, based on the *Silhouette for Mask I* filter

4.4.6.11 SkinColorMode

Uses the *Skin colour* filter and applies a threshold to it that is adjustable in the GUI.

4.4.6.12 TrackerMode

Allows the user to click in the image, and the program will try to track the point that is clicked by using the S*piral matching* filter.

4.4.7 Frame

Frame is the basic frame holder. It contains the raw image data, and some meta-information, like width, height and pixel-type. This class also contains low-level functionality to clear the image.

4.4.8 FrameGrabber

The FrameGrabber is the class responsible for handling the frame, and holding all meta information on the image processing that is done on the frame.
It holds the TipTable, which contains information on body part locations, a CameraClient instance, and a pointer to a FrameSource.

4.4.9 FrameSource

FrameSource is the base class for all the possible frame sources. Currently there are three different frame sources.

4.4.9.1 CameraFrameSource

The TV cameras are handled in this class. The TV cameras are attached to a special capture card, the PXC series card.

4.4.9.2 DirectXFrameSource

The new Firewire cameras can be accessed through this class. It uses Microsoft DirectX DirectShow to interface with the camera. DirectShow has become a standard in computer vision projects, because all new cameras have support for DirectShow. This makes systems built on DirectShow flexible, for they can use a wide range of cameras.

Another advantage the DirectShow has, is that as well as getting data from cameras, it also reads movie files. DirectShow uses the video decoders that are used in Microsoft

Windows, which are easily installed. A large variety of video formats are available as input sources for the vision project, when using this frame source.

4.4.9.3 DiskFrameSource

This class can generate frames by reading them from the disk. Currently only reads BMP files.

4.4.10 ImageFunctions

This class has all the filter implementations, and is therefore a very big class. All image-processing functions are static.

4.4.11 Pixel

Pixel is a basic structure for the red, green and blue values of a pixel. There are some operations defined in this class, such as getting the intensity, or normalizing the colour.

4.4.12 Point2d, Point2i, Point3d

Basic types for passing on point information.

4.4.13 Snake

Snake is the class that implements snakes. The snake is effectively just a list of points. The class has its own update function. When the update function is called, one iteration is carried out.

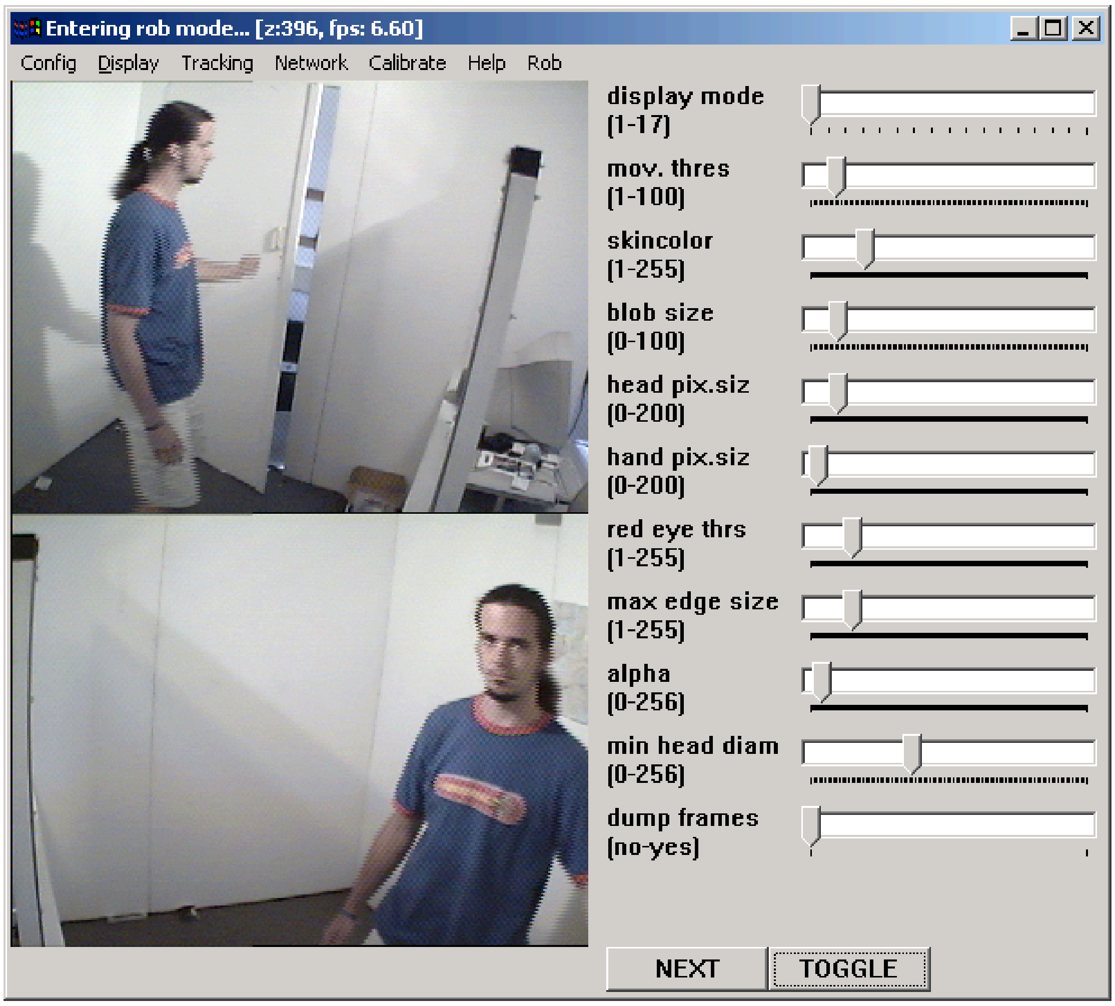
4.4.14 TipTable

The TipTable is the class for holding all interesting information that is found during the image processing.

In future, voting algorithms may be added to deduce information from the tips, and create even more reliable information.

4.5 User Interface

The user interface as it is currently implemented. It is built to easily adjust certain parameters.



display mode [1-17]
mov. thres [1-100]
skincolor [1-255]
blob size [0-100]
head pix.siz [0-200]
hand pix.siz [0-200]
red eye thrs [1-255]
max edge size [1-255]
alpha [0-256]
min head diam [0-256]
dump frames [no-yes]

Figure 40 This is the main user interface. On the left there are the two images, as produced by the cameras. On the right there is a large section of sliders, used to adjust all kinds of filter-settings. The NEXT and TOGGLE buttons are used in interactive mode. The NEXT button takes the next image, the TOGGLE button switches between automatic mode and interactive mode. In the menus above, all other things can be adjusted, the connection to the server can be (re-) established, and so on.

Chapter 5. Network Architecture

5.1 Introduction

In this chapter I shortly discuss the former situation, as it was before I stared working on the Watching window project, and how I solved most of the problems it came with, on a top level. More details, for example the network protocol, can be found in the appendix.

5.1.1 Former situation

The network implementation was basic. There was one computer responsible for the display and acting as server, receiving coordinates from the vision machine.

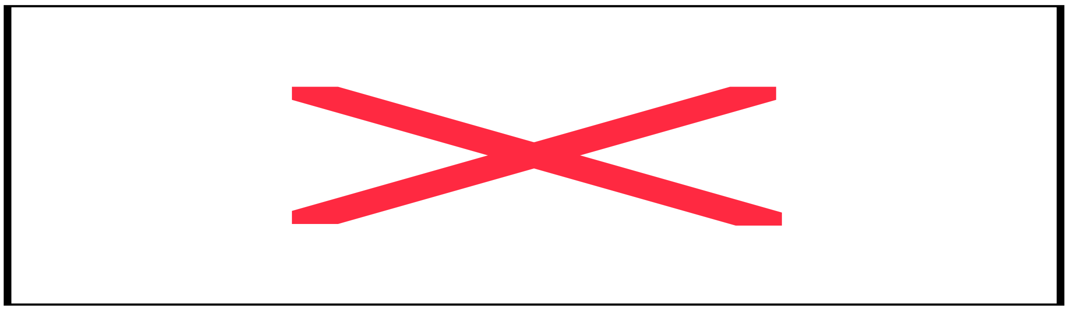


Figure 41 The old network architecture. Only one vision system and one dislay server are supported

Drawbacks were:

1. Only two cameras were supported.
2. All cameras had to be attached to the same machine.
3. The network connection code was unstable.
4. The method used to transfer coordinates contained a lot of overhead.

The support for two cameras is a drawback, because the more cameras that can be added, the more reliable the 3D position of body parts can be estimated.

The second problem has to do with computational power. Both of the images have to be processed by the same machine. When using algorithms that require a lot of calculations, the system responds more slowly.

The network connection code was not well structured, and unstable. Sometimes when connection failed, it would not try again, and sometimes the system could not connect at all. The reason behind this was unknown, but I didn't spend time on debugging the code, because from experience I know it is faster to completely replace it with something new.

The coordinate transfer system involved a lot of synchronizing overhead, because it was trying to add a timestamp to the network packets and synchronize both machines. There was no apparent reason for this, so I left it out.

5.1.2 New situation: Solving the problems

I changed from a single client, single server architecture to a different one where the server would be the machine that receives 2D coordinates from one or more vision machine (cameras). The server is also responsible for sending the data to the display clients, upon request.

5.2 Design

The new architecture is illustrated in fig. 42.

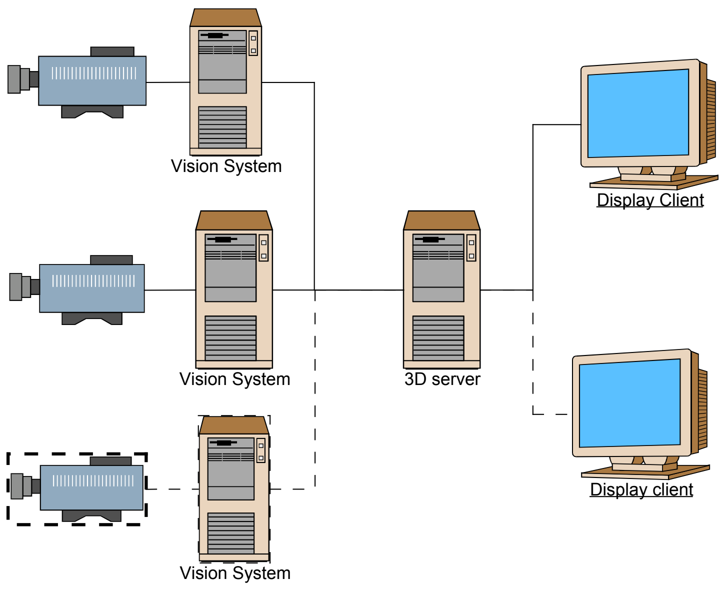


Figure 42 The network architecture. The system can easily be extended with extra cameras and extra displays.

5.2.1 Responsibilities

All the entities in the network structure have their capabilities and responsibilities. These responsibilities define what should be implemented where.

5.2.1.1 Camera client

- Know its camera's calibration, and be able to send it
- Be capable of processing the image
- Transferring the body-part data to the server
- Request the re-projected positions of the body-parts and use it for processing

5.2.1.2 3D Server

- Keep administration on camera's and displays
- Use multiple camera's to deduce the 3D location of body parts
- Transmit data to either client when requested

5.2.1.3 Display client:

- Know its own location in world space
- Retrieving 3D location of body-parts from 3D Server

- Do something appealing with the data

See the appendix for more detailed information on interactions.

Chapter 6. Server Software Architecture

6.1 Introduction

As part of the total Watching Window project, the 3DServer functions as the link between the vision part and the display part.

The 3DServer performs the necessary calculations to transform a set of points in camera space to world space. The camera clients upload their points. The display clients can request the calculated points in world space.

The server takes care of connecting, administration and calculation necessary for this functionality.

In this chapter I tell how the server is designed, and what the functions of all classes are.

6.2 Class description

In this section I describe each class shortly.

6.2.1 Class diagram

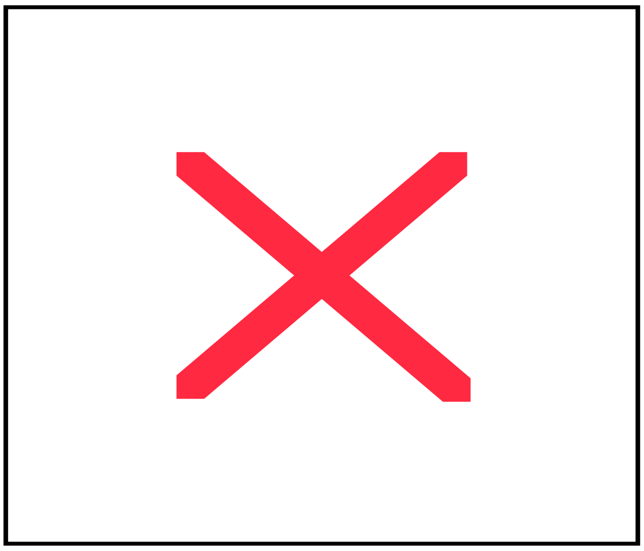


Figure 43 Class diagram of the server software

6.2.2 Camera

This class is responsible for administration of the camera. It therefore stores the calibration information, as well as a history of received camera space points.

6.2.3 Calibration Information

Each camera has its own calibration information. The calibration information consists of parameters that describe how a transformation form world space to camera space is made. It can also be used to calculate the line in world space when starting with a point in camera space.

6.2.4 2D to 3D Calculator

This class is the one that performs the calculation of the point in world space, using points in camera space of two cameras.

For each body part that is correctly found the next procedure is executed:

1. For each camera, a line is created from the projected body part through the focal centre into world space.
2. The two lines that are formed in that way are analysed
3. The shortest distance between the two lines is found
4. The point halfway that distance is taken as approximation of the world space point

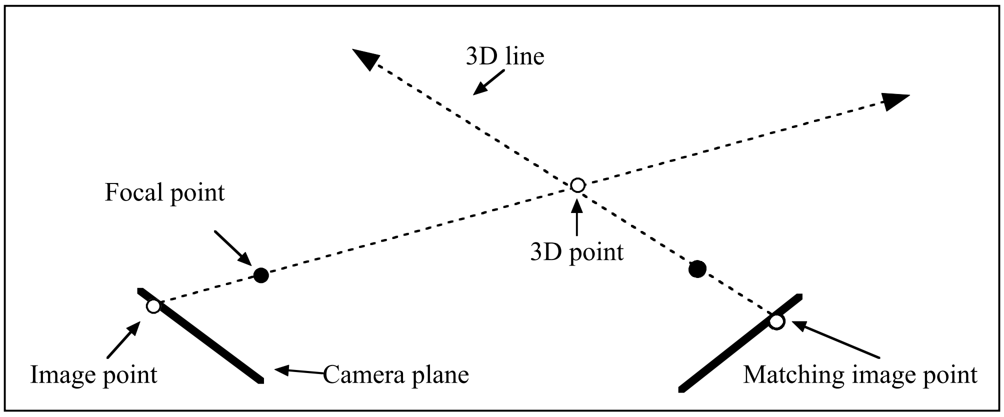


Figure 44 The top view of the construction of a 3D point out of two 2D points.

6.2.5 TCPSocket

This class listens to the network and creates a ClientHandler if there is an incoming connection.

6.2.6 3D Server

This is the main class. On creating this class a TCPSocket is created to listing to incoming network traffic. The 3D server is responsible for the administration of the ClientHandlers.

6.2.7 ClientHandler

ClientHandlers handle the communication with the clients. They check the incoming messages and handle in accordance to them. Each client is appointed to a ClientHandler.

6.2.8 BodyPartCoordinates

Body part coordinates are information objects in the 3D Server. They contain the 2D or 3D location of the body parts. They are receives from the camera clients and send to the display clients.

Chapter 7. Conclusions

The new vision system for the Watching Window tracks people close to real-time. It finds the user’s head location and hand location. The system is not yet capable of finding eyes in images, but basic functionality is provided to build such a system on.

Both the vision software architecture and the network architecture are improved; they are more flexible and reusable, they make developing easier. The entire system is now more reliable, maintainable and easily extendable.

The Firewire cameras provide a better signal for the computer. They are more noise resistant. Another advantage is that they have a higher frame throughput.

Chapter 8. Recommendations

This project has some loose ends, and some thing that can be improved

8.1 Computer Vision

8.1.1 Fully implementing snakes

Snakes provide very useful information at very low cost. Better head and hand silhouettes can be found. This will benefit the recognition process.

8.1.2 Eye localization and tracking

The eye localization and tracking should be made to work. It is currently unreliable. A complete different approach may be taken. It may provide more useful to create a database with faces, and create face models out of it. Then the head found in the image can be matched against the database, to find the eye position.

8.1.3 Adding cameras

Adding cameras could also increase the performance of the system.

8.1.3.1 Top of screen

A camera on top of the screen looking in the direction of the user will be able to get user-images that are most of the time frontal. Frontal images can then be used to do face detection on. Face detection on frontal views is a widely covered in the computer vision community; many papers are available on this subject.

8.1.3.2 Top of booth

A camera on top of the booth looking down along the screen enables the system to make more accurate approximation of the hand positions. Because the user in seen from a top view, it is easily detected if the hands are before the body, or above, or at the sides.

8.2 3D Server

8.2.1 Re-projection

Basic functionality is implemented to request a projection from a found body part in world space back to camera space. This code is untested, but theoretically it could provide very useful information. Once a world coordinate is found and it is a good approximation, the projected point in camera space can reduce the amount of image data that have to be searched. Instead of looking in a certain area the search can be refined to look only along the projected line or point.

8.2.2 History

Currently there is a history of one point per body part per camera stored as history. In other words, only the last point is remembered. This could be extended to a certain number, for example 10. This would enable the system to do movement-approximation on those points, which could in turn be used as feedback to the image processing, or to filter out noisy results.

Chapter 9. Evaluation

In here I will describe how my internship went, how things where organized and I give my personal opinion on how things went.

9.1 Product progress

At the start of my internship the goal of it was still little vague. After the first meeting Geoff and I decided it should be "eye tracking" for the Watching Window. A German student, Hanna von Tenspolde, also worked on the Watching Window, but she focussed on the display functionality.

The first weeks I mainly focused on getting into the code, and trying to understand what it actually did, or tried to do. I began replacing bit and pieces, and I threw out everything that did not make sense or did not have an apparent reason to be there.

While working on the code I started implementing new vision functionality. I started out using skin colour, but that was not the right way to go. So I changed this and I got quite good results for finding the head.

Then I spend a long time on finding the eyes. This was difficult, and I did not get results that satisfied me enough.

The process of building filters to find the head and eyes took several weeks; during these weeks I also gradually replaced old design with new design.

During the last two weeks I spend time on writing the DirectX DirectShow code, writing basic hand detection and improving the speed en quality of the head finding process.

At the time of writing this report there is a new Watching Window code base. It is faster and more flexible than the old system.

9.2 Communication

Each week there was a special meeting with Geoff, Hanna and Jayson Mackie. Jayson is the person who was responsible for making the Watching Window run and knew most about it. He also was the one that did searching for new cameras.

Every Monday we evaluated what we did the past week and updated our plans for next few weeks, working towards the end goal.

I worked with Hanna in one room, so we had plenty of opportunity to discuss things. Because she had little programming experience, many times I was helping her out explaining how to do the design and programming of code.

9.3 Personal opinion

I enjoyed my stay at the University of Otago. I worked hard and although I did not reach all my initial goals, personally I find the result is satisfying. There is a working product with a lot of opportunities. I had a wonderful time with all the other people around in the lab. There was a good atmosphere to work in.

Besides from just working, I broadened my knowledge on the subjects I covered. Unfortunately, one of the computer vision experts, Dr. Brendan McCane, was one a sabbatical year. I could probably have more information exchange with him, if he were around.

Chapter 10. References

[1] Rogério Schmidt Feris, Teófilo Emídio de Campos, Roberto Marcondes Cesar Junior. "Detection and Tracking of Facial Features in Video Sequences". Lecture Notes in Artificial Intelligence, vol. 1793, pp. 197-206, April 2000, Springer-Verlag press, http://www.springer.de/comp/lncs/index.html (MICAI-2000, Acapulco)

[2] K. R. Castleman. *Digital Image Processing*. Prentice Hall, 1996

[3] Bernhard Fröba, Andreas Ernst and Christian Küblbeck, Real-Time Face Detection. *http://www.embassi.de/publi/veroeffent/Froeba.pdf*

[4] Ghidary, S.S., Nakata, Y., Takamori, T. and Hattori, M. "Head and Face Detection at Indoor Environment by Home Robot." *Proceedings of ICEE2000*, May 2000, Iran

[5] M. Kass, A. Witkin, and D. Terzopoulos, "Snakes - Active Contour Models'" *International Journal of Computer Vision*, 1(4): 321-331, 1987.

[6] J. C. Terrillon, M. N. Shirazi, H. Fukamachi, S. "Comparative Performance of Different Skin Chrominance Chrominance Spaces for the Automatic Detection of Human Images," *Proceedings of 4th IEEE International Conference Face and Gesture Recognition*, pp. 54-61, 2000.

Appendix A Network Details

A.1 Interactions

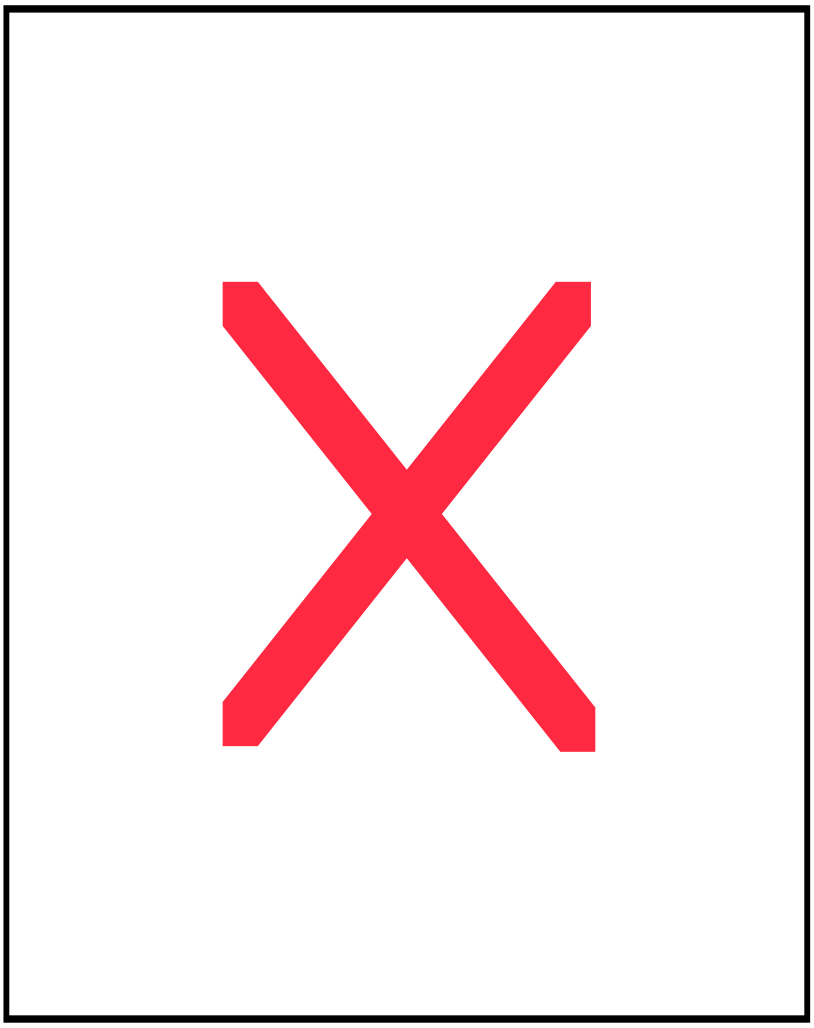


Figure 45 Network interaction between Camera Client and 3D Server

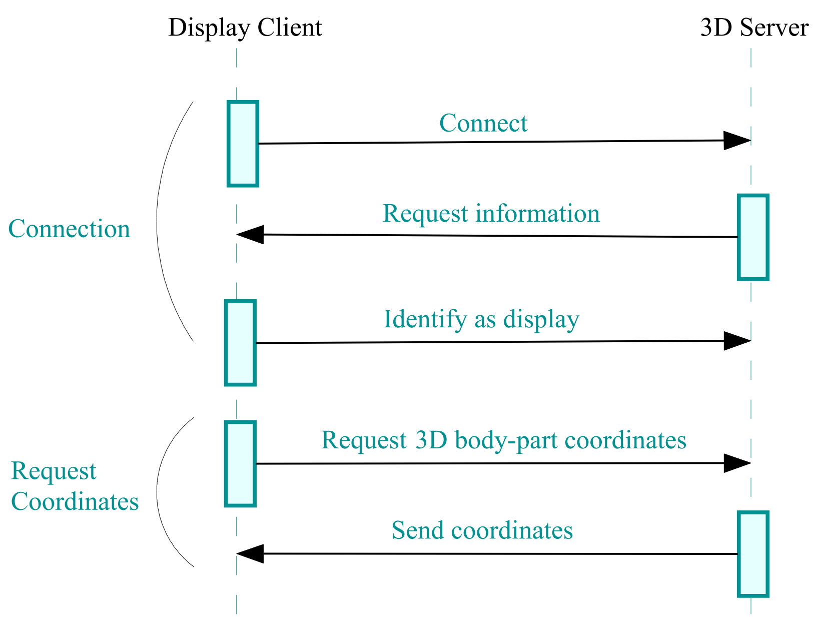


Figure 46 Network interaction between Display Client and 3D Server

A.2 Protocol - Packages

CAMERA_CALIBRATION_PACKET	1
CAMERA_COORDINATES_PACKET	2
CAMERA_REQUEST_2D_PACKET	3

CAMERA_LOGON

Field	Offset	Length	Type	Value
Name	0	6	ASCII -String	"CAMERA"

CAMERA_CALIBRATION

Field	Offset	Length	Type	Value
Type	0	1	BYTE	CAMERA_CALIBRATION_PACKET
Calibration matrix	0	?	Matrix	

CAMERA_PUSH_2D_INFO

Field	Offset	Length	Type	Value
Type	0	1	BYTE	CAMERA_COORDINATES_PACKET
Packed 3d points	1	85	Body 3d	

CAMERA_REQ_2D_INFO

Field	Offset	Length	Type	Value
Type	0	1	BYTE	CAMERA_REQUEST_2D_PACKET
Packed 2d points	1	85	Matrix	

DISPLAY_LOGON

Field	Offset	Length	Type	Value
Name	0	7	STRING	"DISPLAY"

DISPLAY_REQ_3D_INFO

Field	Offset	Length	Type	Value
Name	0	3	STRING	"REQ"

SERVER_CAMERA_CALIBATRION_CONFIRMATION

Field	Offset	Length	Type	Value
Acknowledge	0	27	STRING	"Got calibration info!\r\n"

SERVER_CAMERA_COORDINATES_CONFIRMATION

Field	Offset	Length	Type	Value
Acknowledge	0	16	STRING	"Got body info!\r\n"

SERVER_DISPLAY_3D_INFO

Field	Offset	Length	Type
Packed 3d information	0	125	Body3d

POINT3D

Field	Offset	Length	Type
X value	0	8	double
Y value	8	8	double
Z value	16	8	double
Known value	24	1	bool

POINT2D

Field	Offset	Length	Type
X value	0	8	double
Y value	8	8	double
Known value	24	1	bool

BODY3D

Field	Offset	Length	Type
Head	0	25	Point3d
Left hand	25	25	Point3d
Right hand	50	25	Point3d
Left eye	75	25	Point3d
Right eye	100	25	Point3d

BODY2D

Field	Offset	Length	Type
Head	0	17	Point2d
Left hand	17	17	Point2d
Right hand	34	17	Point2d
Left eye	51	17	Point2d
Right eye	68	17	Point2d

Appendix B Important class diagrams

B.1 FrameGrabber

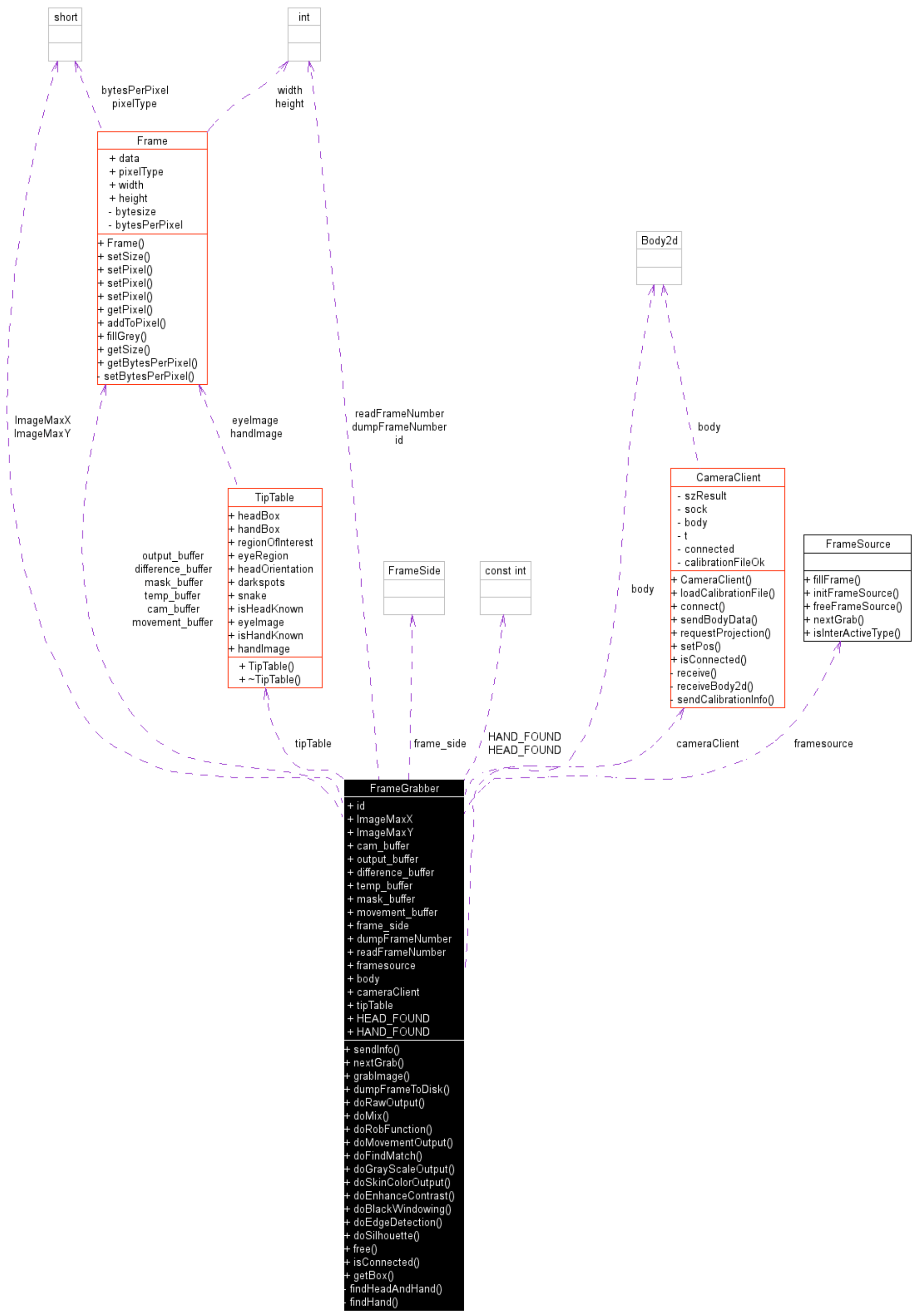


Figure 47 The class diagram of the FrameGrabber

B.2 FrameSource

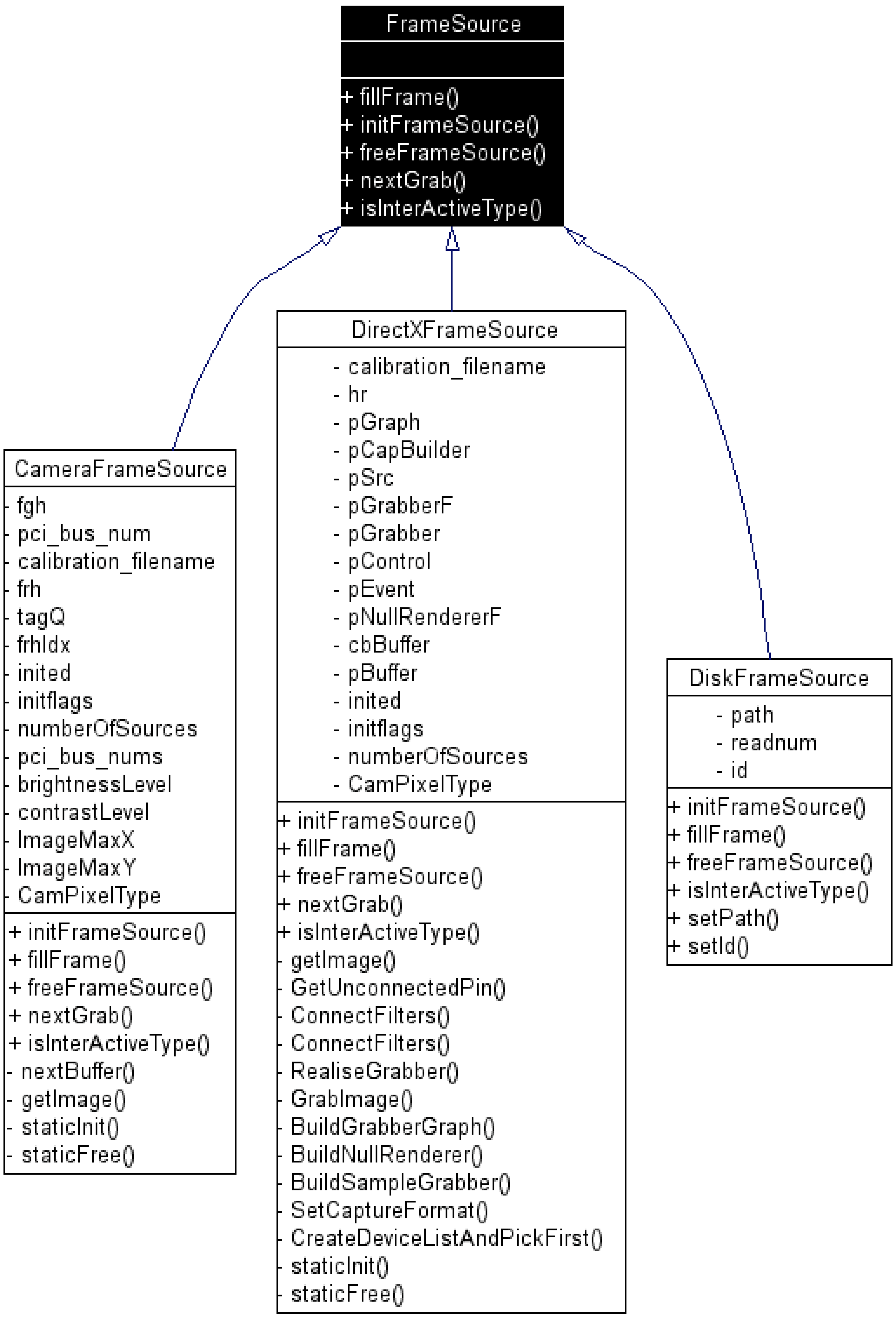


Figure 48 The class diagram of the FrameSource