The Watching Window 2004



Date: 10 Dec 2004
Author: Wout Slakhorst, w.m.slakhorst@student.utwente.nl
At: Graphics lab, University of Otago, Dunedin NZ
Supervisors:
Anton Nijholt, Universiteit Twente (nl)
Geoff Wyvill, University of Otago (nz)
Brendan Mccane, University of Otago (nz)

Preface

This report is about my internship at the University of Otago in New Zealand. During my stay I worked on the Watching Window system. This report gives an overview of the entire system. I worked on most of the components, but for completion I included everything. In this way future students can use this report as a design and specification document. The Watching Window has been in development for four years at the Graphics Department of the University of Otago.

I chose this assignment because I didn't now much about imaging yet. Studying a field of research unknown to me is always a challenge. I wanted to go abroad since living in another country itself is an unique learning experience. Because of these reasons my choice of the University of Otago was easy.

Abstract

The Watching Window is an interactive 3D system where the user can interact with the system in a natural way. The user can see 3D objects from different angles by moving his or her head and the user's hands can be used as input to different applications.
During the last three months the system has been altered from a low-resolution color based system to a less static, high-resolution, grayscale based system. With this new system better tracking (hand and head) and the introduction of hand gesture detection have been realized.
Apart from the new features and improvements, the differences in implementation between different parts of the system have been removed. This will make it easier to exchange code between the different parts. The current Watching Window code not only works but also gives a good basis for further development.

Table of contents

1. Introduction

The ultimate goal of the Watching Window is a big screen mounted on the wall with integrated cameras, microprocessors and microphones. Users should be able to interact with this system using natural gestures and speech. This setup will remove the need for other peripherals like a mouse and keyboard. The current situation however is quite different. Current techniques don't enable a robust version of such a system yet.
The Watching Window project tries to bring this futuristic vision closer. The goals for this report have been set to include:

- robust head tracking
- hand tracking
- gesture recognition

The last goal has been added because there are some interesting developments at the graphics lab of the University of Otago. The Swirling Sweepers[1] developed by Alexis Angelidis is currently being implemented for the Watching Window by a masters student. This is a virtual modeling technique where it's possible to interact with a virtual piece of clay. This interaction is much easier when the user can use different gestures for different actions. Head tracking was previously based on detecting skin color, but it was felt that using skin color isn't the way to do it. This enables the use of bigger grayscale images which should come in handy for detecting hand gestures.
The first chapter will describe the environment of the system, the equipment used and the constraints this has on the system. A short description on calibration will follow, because accurate calibration is extremely important translating the position of tracked objects in the 3D environment. The main goals of head and hand detection will be described in chapters four and five respectively. These chapters will give a full description on all techniques used to accomplish the goals. Chapter five will also elaborate on the gesture detection. The last three important chapters will show the 2D to 3D conversion, the filtering and the system architecture. Other ideas tried will be discussed in chapter nine, followed by the conclusions for this report. Finally, recommendations will be made for future researchers working on the Watching Window.

2. Environment and equipment

The Watching Window is located in a controlled environment. Figure 1 gives a good look at the setup of the system. The cameras, indicated by red arrows, are located on the sidewalls and on the ceiling. The cameras are not in an upright position but are rotated in such a manner that they'll capture more of the user area and less of the walls and ceiling. The back projection screen in the center displays the application with which the user can interact. The booth has been painted white for reflecting light. The red dots on the wall are used for the calibration as explained in chapter three.

Figure 1

The Watching Window gives the illusion of a window by moving the frustum according to the head movement of the user. This gives a different result in comparison to moving the objects, objects aren't only seen from a different angle but they also get distorted, adding more realism to the illusion.

The current setup uses one 1 Ghz and one dual 500 Mhz computer. Future plans include the allocation of one computer per camera, which will remove the limitations on computing power for image processing. The current setup already permits the use of one computer per camera, since the network architecture is set up to handle each camera separately. The system runs on the windows platform. The booth itself influences the image processing. It guarantees that there won't be any background noise, which allows the use of easy and simple image processing algorithms. The current color of the walls however is not ideal since the cameras are saturated very quickly which make some parts of a human body become white, which in turn makes it hard to distinguish the part from the wall. There are already some plans to test new backgrounds for the booth.

Two big lights provide the lighting in the booth. Testing has shown that the head detection algorithms don't need the light and they will work fine without it. The hand detection, however, requires good lighting or it will not successfully detect the gestures made by the user.
Previous versions have used intense lighting to extract a human silhouette accurately. This however makes the system more dependant on the controlled environment. With the current lighting and algorithms this is less the case. As explained in the introduction, the final goal will be a system robust enough to run in an uncontrolled environment with marginal lighting.

3. Calibration

Accurate calibration is very important in the Watching Window. Like the old saying goes: 'A chain is only as good as its weakest link'. This holds for the Watching Window as well, the head and hand detecting can be really good but without accurate calibration the overall picture won't be convincing to the user.
The calibration is done by measuring the distance between a number of dots on the wall, and the four corners of the screen. The position of the dots can be seen in figure 1. The world coordinates can then be calculated from these distances. After the world coordinates are known, the camera properties are calculated using Tsai[]. Both the calculation of the world points and the calibration itself will be discussed further.

3.1. World point calculation

Typically a camera can see twenty-one points at a time. Tsai doesn't require that many points but the more points used in the calibration the more accurate it will be. As mentioned earlier, the world coordinates of the points are calculated from the distances between the points and the four corners of the screen (figure 2).

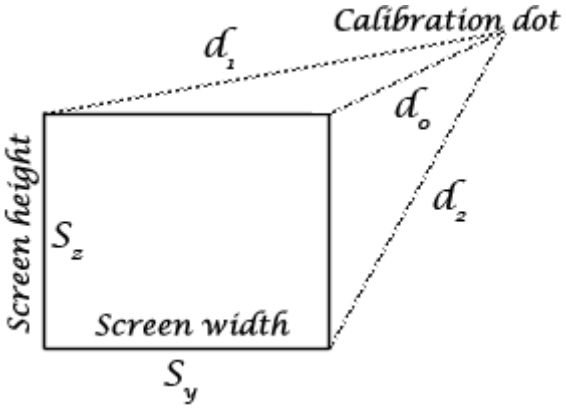


Figure 2

The distances can easy be calculated using formulas 1 to 4. Figure 3 gives is a close-up of the top triangle (d_0, d_1, S_y) as seen in figure 2.

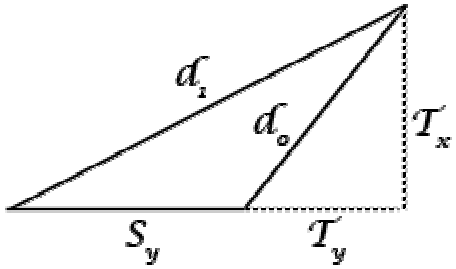


Figure 3

$$d_1^2 = T_x^2 + (S_y + T_y)^2 \quad (1)$$

$$d_0^2 = T_x^2 + T_y^2 \tag{2}$$

Combining these formulas gives:

$$d_1^2 = d_0^2 - T_y^2 + \left(S_y + T_y\right)^2 \tag{3}$$

This then only has to be solved for T_y:

$$T_y = \sqrt{\frac{d_1^2 - d_0^2 - S_y^2}{2S_y}} \tag{4}$$

where d_1, $d1$ and d_2 are the distances from the screen corners, S_y is the screen width, S_z is the screen height and T_x is a distance used in the equation. T_y is the y-offset of the right top corner. T_x can't be derived using formulas 1 to 4 since it's a line in two dimensions. T_z however, can be solved using the same formulas by replacing d_1 with d_2 and S_y with S_z. T_x can easily be calculated when both T_y and T_z are known. The resulting values are offsets to the screen corner. Adding these values to the world coordinates of the screen corner gives the world coordinates of the point.
Only three reference points are needed to determine the correct world point, but four points are available. For more accurate results, four calculations are done, each one using a different combination of reference points. These results are then averaged to give the final result.
A basic requirement for this way of calculating is that the reference points (screen corners) all lie within the same plane. Because these are the screen corners this condition is satisfied. It can be possible that the screen isn't entirely vertical, but this only has a small influence on the calibration. It becomes a bigger problem when the screen is skewed and thus the reference points aren't coplanar. This will generate big differences between the four calculations. If the difference is too big between the four calculations the point isn't used. Thus, if the screen is skewed badly, the difference will be bigger and the calibration will fail.

3.2. Tsai calibration

The calibration method explained by [2] recovers the interior orientation, the exterior orientation, the power series coefficients for distortion, and an image scale factor that best fit the measured image coordinates corresponding to known target point coordinates. The recovery of these properties is explained briefly below; for a more elaborate description see [2].

3.2.1. Interior

Interior orientation is the relationship between the camera coordinates and the image coordinates. The camera coordinate system has its origin at the center of projection, its z axis along the optical axis, and its x and y axes parallel to the x and y axes of the image.

The center of projection is at $(x_0, y_0, f)^T$, measured in the image coordinate system. F is the focal length of the camera. These values have to be recovered by the calibration.

3.2.2. Exterior

Exterior orientation is the relationship between the world coordinate system and the camera-centered coordinate system. The transformation from world to camera consists of a rotation and a translation. This rotation and translation must be recovered by the calibration.

3.2.3. Unknown horizontal scale value

A complicating factor in the calibration is that the discrete nature of image sampling is not preserved in the signal. The initially discrete sensor signal in analog form is filtered to produce a smooth video output signal that hides the transitions between cells of the sensor. This waveform is then digitized. The sampling in the horizontal direction is typically *not* equal to the spacing of sensor cells, and is not known accurately. The factor of the pixel spacing in the x and y directions therefore also has to be recovered. It is not possible to recover this extra parameter when using planar targets. For this reason the Watching Window cameras use points located at two walls for the calibration.

3.2.4. Distortion

Because cameras use a round lens, radial distortion occurs with centers along the optical axis. A point is displayed at a distance from the principle point that is larger or smaller than predicted by the projection equations. The distortion is worse further away from the center. The Watching Window only uses the first of the power series.

3.2.5. The algorithm

In calibration, a target of known geometry is imaged. In the Watching Window case these are the world coordinates of the points calculated earlier. Next to these coordinates are the corresponding image plane coordinates. These are obtained with the cameras. A search area is given for each calibration point. The camera will take a picture and look for the points. This is done by using red dots. The dots can easily be detected because of the high red value and low green / blue values in RGB color space.
These form the basic data on which the calibration is based. Tsai's method first tries to obtain estimates of as many parameters as possible using linear least squares fitting methods. In a subsequent step, the rest of the parameters are obtained using a nonlinear optimization method that finds the best fit between the observed image points and those predicted from the target model. Parameters estimated in the first step are refined in the process.
Details of the calibration method are different when the target is planar than when it is not. Accurate planar targets are easier to make and maintain than three-dimensional targets, but limit calibration since they can't retrieve the scaling factor. For the full algorithm see [].

3.3. Conclusion

The calibration currently is off target by a minimum of five centimeters. This error is too large for accurate control of objects with hand movement. The error is partially introduced by the measuring the distance between the dots on the wall and the four corners of the screen. When looking at the result of the calibration, the dots on the side walls have the biggest errors and are therefore the first to be ignored. The back wall has the most accurate dots. The more points used which aren't in the same plane, the more accurate the calibration will be. The side cameras typically use twelve dots on the back wall and only around five on the side wall. The top camera only uses two or three dots on the side walls, therefore too many points lie in the same plane which isn't good for the calibration. For the Swirling Sweepers application higher accuracy is needed. Therefore the calibration has to be improved. There has already been some discussion on how to improve the calibration. One suggestion is to determine the radial distortion manually. The reason for determining the radial distortion manually is an observation on the inaccuracy of the hand tracking. The hand tracking is less accurate when the hand is further away from the center of the image. This problem might be caused by the wrong model of the radial distortion.

4. Head detection and tracking

The previous system almost did the same as the current system, which will be described below. When the choice was made to use bigger, grayscale images, skin color checking didn't work anymore. Therefore Eigenfaces are being used to perform the final step in the detection process. An eigenface implementation[3] is developed by a third year student at the University of Otago. This implementation is heavily modified and implemented into the Watching Window.
To have a clear view on the entire process, the image processing will be described first, after that the eigenface theory and general working is explained. The final chapter on head detection and tracking will combine everything into a clear description. This part of the report will be concluded with the conclusion on head detection and tracking, explaining the improvements and problems of the new system.

4.1. Image processing

The processes used in the head detection will be explained here. The algorithms that change the image will be referred to as 'filters' from now on. The filters and other algorithms used for detecting the head all work after one another. One uses the output of the previous. In this way the goal of detecting the head is reached bit by bit. The process is done in five steps, the entire process will be explained in paragraph 4.2. The filters used are a low resolution filter, a movement filter, a silhouette creation filter, a head find algorithm, temporal image matching and the eigenface check.

4.1.1. Low resolution filter

The cameras are configured to grab 640 by 480 grayscale images. Most of the filters and algorithms don't require such a large image. This is why the low resolution filter changes the original image to an image of 320 by 240. This image is used to do the main part of the processing. The original image is currently only used for detecting the hand gestures (see chapter 5).
Since computing power is currently scarce, the filter only takes one out of four pixels (nearest neighbor algorithm). This means that no linear interpolation is done with the surrounding pixels. The filter should be altered to use linear interpolation as soon as more computing power is available. Or in other words: as soon as there is one computer for each camera.

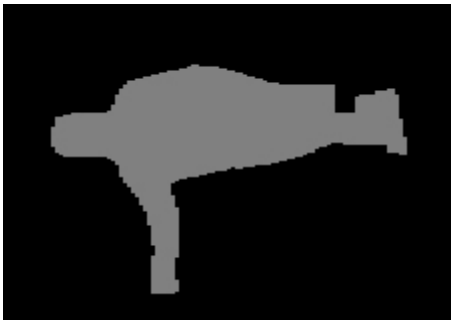
4.1.2. Movement filter

The main task for the movement filter is to filter out the non-moving pixels. The idea is that a human body always moves, even slight movements are picked up by the filter. If there isn't any movement than this shouldn't be a problem for the system, since the detected objects will be in the same place as before.
The result of this filter is a black and white image where the white areas have a difference in pixel value between two frames which is bigger than a threshold. This threshold has empirically been determined. Changes in lighting conditions influence the filter. To adjust for the change in lighting the threshold can be altered when running the main program. Figure 4 gives an example.

Figure 4

4.1.3. Silhouette creation filter

The silhouette creation filter looks for areas of white pixels. If an area has enough white pixels then the entire area will become white. Basically small groups of white pixels are enlarged to a 5 by 5 square. After this the gaps are filled by looking for adjacent blocks within a certain range. If a block is found, the area between the blocks is filled as well. The resulting effect is shown in figure 5.



Picture 5

4.1.4. Head find algorithm

The head find algorithm finds a most likely candidate for the head given a silhouette created by the previous filters. It actually finds the highest pixel which isn't black. After that it travels to top and bottom from this point expanding the area until it reaches black pixels. The width of the area is set to the height of the area.

4.1.5. Temporal image matching

Temporal image matching uses a previous taken image and tries to match it to a location of a new image. The location chosen is the location with the highest score , where the score is defined in equation 5:

$$S = \frac{1}{(\sum_{p_1=0,p_2=0}^{I} abs(p_{1,} - p_2)) + 1} \quad (5)$$

Where S is the score, I is the number of pixels in the temporal image and p_1, p_2 are pixels from the temporal image and a sub-frame of the new image. The algorithm searches in a spiral way for new locations and returns the location with the highest score.

4.1.6. Eigenfaces

Because of the change from color images to grayscale images, a new way had to be found to perform a check on an image to confirm it is a head. The choice has been made to use eigenfaces[4]. This technique works quite well on detecting heads as shown by earlier research. It uses grayscale images which makes it perfect for the current Watching Window. The eigenface approach requires training the system before it can detect heads accurately. Since the system will be trained on actual heads, it will also detect actual heads, with the exception of people wearing t-shirts with heads on it.
The next three paragraphs will explain the basics of using Eigenfaces: the training using the Watching Window and the detecting algorithm used to detect heads in the Watching Window.

4.1.6.1. Basics

The term 'eigenface' is derived from the fact that the generated eigenface are eigenvectors of a covariance matrix. All the training images are subtracted from the average image and then represented as vectors in this matrix. By using principal component analysis the eigenvectors of this matrix are returned. When these eigenvectors are viewed as images they look like 'ghostly' images of heads, hence the term 'eigenfaces'. Every image can then be shown as a linear combination of eigenfaces. This will be the representation of that image in a high dimension 'eigenface space', where the number of dimensions equals the number of used eigenfaces. The entire process of calculating the eigenvectors is beyond the scope of this report and can be read from[].

4.1.6.2. Training

In order to perform well, the eigenface approach needs a good training database. public databases usually only incorporate pictures of heads taken from in front of the person. The Watching Window cameras look at the user from the sides and from above. Further more the images are rotated as well, making the public databases more useless. Because of this unique environment, public databases are of no use.
A user in the Watching Window can freely walk around and look in any direction. This requires the eigenface training database to include images of heads at different angles and sizes. The first approach used all kinds of different images taken from different people and used those as training database. The resulting average images are shown in figure 6.

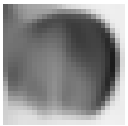


Figure 6

Clearly all the features like eyes, nose and mouth are hard to recognize. This was shown in the performance of the system. It didn't perform as well as expected. A new approach is to take pictures from heads where the user doesn't move. By taking images from

different angles and putting those images in different training databases a better result is achieved as described in paragraph 4.2.1. The images in figure 7 show the resulting average images of this approach.

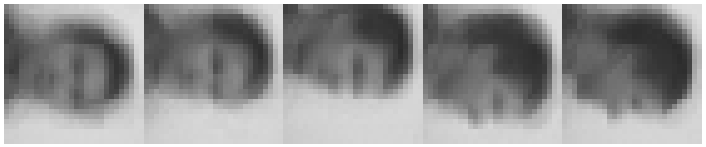


figure 7

In these images more features can be recognized. For every camera, images are taken from five different angles. Using more angles would make the system too slow and using less angles would decreases the performance. The result is five databases for each camera. These databases have fixed size images of 28 by 28 pixels.

All the images taken are by using the Watching Window user interface. In this interface two or three points can be chosen to align the images. The choice has been made to use the eyes and the nose (figure 8). If one of the eyes isn't visible, one eye will suffice. The interface will take images from 40 by 40 pixels. These have been cropped and resized to 28 by 28. By using the Watching Window itself to capture the images gives better results then using images obtained in another way.

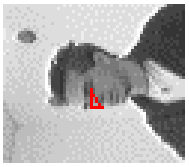


Figure 8

4.1.6.3. Detecting

Detecting heads with the eigenface approach requires an image which is the result from deducting the average image from an input image. The resulting image will then be represented as a combination of eigenfaces, giving a number of weights for this input image. The next step as described in [4] is to calculate the Euclidian distance from these weights to the training image weights. Because this is a rather slow way of doing things, the Watching Window calculates the distance from these weights to the origin in eigenface space. This is much quicker and thus more suitable for the Watching Window. The Watching Window uses 15 eigenfaces to represent an input image. The calculation of the distance for an input image only requires 15 computations, one for each weight. The original method described by [4] requires 15 computations for each training image! Instead of using the euclidian distance, the mahalanobis distance is used. This gives a normalization which enables the use of the same threshold per camera for the different databases. Equation 6 shows the mahalanobis distance.

$$dist = \sqrt{\sum_{i=0}^{m} \frac{x_i^2}{\lambda_i}} \qquad (6)$$

where m is the number of eigenfaces used, x is the input weight and λ the eigenvalue. The calculated distance alone doesn't tell us whether or not the input image was a head. A threshold has to be determined, where distances below that threshold are classified as heads and distances above are classified as non-heads. The threshold has been determined using the Watching Window, it has been set to a value which enables good tracking. In theory this is a threshold with no false-negatives and an unknown percentage of false-positives.

4.2. The complete process

By using all these filters and algorithms a head detector and tracker has been made. From an idle state the system can detect a head and then goes in tracker mode. The tracker mode has a speed advantage over the detection mode since it only has to evaluate a small piece of an image as opposed to the entire image. Both cases shall be discussed separately.

4.2.1. Detection

An user has to walk into the booth in order to interact with the system. This gives the system a good input to the movement filter (paragraph 4.1.2). The movement filter will convert the input image to an image which only shows the moving parts. The resulting image will be used by the silhouette creation filter. This will fill any gaps in the image produced by the movement filter. The head detection algorithm will do an initial guess on the head location. Finally a check is performed using the eigenfaces.
This check is done at nine locations, with three different image sizes and with all five eigenface databases. The check thus consists of a maximum of 135 checks per camera. To reduce the amount of processing, the eigenface checker exits as soon as it finds a head. The different sizes uses are 40 by 40, 36 by 36 and 28 by 28 pixels. The first two are resized using linear interpolation, which has the best quality/processing time ratio. The different image sizes are used from small to big, saving time with resizing the image when the 28 by 28 image gives a positive result. The nine locations are arranged as a three by three matrix and all points are two pixels apart. This seems small but works well in tracking mode.

4.2.2. Tracking

After the initial detection of the head is successful a 10 by 10 pixel temporal image is taken of the head. When the next frame is to be processed the temporal image is matched to a new location as described in paragraph 4.1.5. When a new location has been found the eigenface check is performed. A comparison with all the five different databases is not necessarily required. Instead a check will begin with the database used last time to successfully detect the head. If this check fails because the user turned his/her head, the databases of the adjacent angles are used (figure 7). This limits the total number of checks to a maximum of 81 instead of 135. If all checks fail the system goes out of tracker mode and tries to detect the head again. When the check is successful, the temporal image is updated.

4.3. Conclusion

The combination of simple filters and a more sophisticated eigenface approach works well. Heads are easily detected and the system can't be fooled with cardboard heads or by holding your hands above your head. Every now and then the system fails to detect the head and goes out of tracking mode. This effect usually occurs when an user bends over. This gives images the eigenfaces aren't trained with. When this happens it will immediately find the head again but with a speed penalty, this can be seen as a little glitch to the user. Future implementations might want to include training images with heads looking slightly up and down. Only the side cameras are used for head detecting, the top camera isn't of much use since the lighting conditions make the skin look very bright, making it more difficult to see the difference between the wall and face.

5. Hand detection

The hand detection has proven to be more difficult. For the head detection eigenfaces were used to perform the final check. Such an approach can't be used for the hand detection. The head is a relatively static object, a hand isn't. Hands can have all kinds of shapes, this increases the difficulty to detect them. However, some assumptions can be made when an user is using a hand to interact with the system: because the screen is in front of the user, he or she will reach out to touch the virtual objects. This places the hand in front of the body, where it's easier to detect. Given this assumption it's easy to detect a hand in a certain area. A user can only interact with the system if his or her hand is within half a meter distance of the screen. The top camera wasn't used for the head detection, but is very useful for the hand detection. Apart from the hand detection, the detection of gestures is also realized. The new system can distinguish between two gestures. Both the hand detection and the gesture detection are interwoven and don't work separately. The next paragraphs explain the used filters and the complete algorithm.

5.1. Image processing

5.1.1. Movement filter

The hand detection uses the same movement filter as described in paragraph 4.1.2. The only difference is that for the hand detection only a small portion of the screen is scanned depending on the camera.

5.1.2. Edge detection

Edge detection is done by doing a convolution over the image with the sobel kernels. The resulting image is a grayscale image in which the harder edges have a higher gray value. Figure 9 shows the sobel kernels.

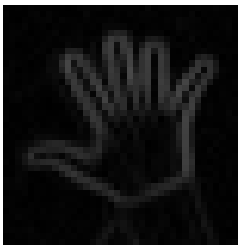
$$\begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

Figure 9

Each kernel is applied separately to the image. Both the resulting images are added together using equation 7.

$$R = \sqrt{I_1^2 + I_2^2} \tag{7}$$

Where R is the output image and I_1 and I_2 are the resulting images from each of the kernels. Figure 10 shows the result of the edge detection performed on a hand.



Picture 10

5.1.3. Edge thresholding

The edge detection can be modified into a filter where the result is a binary image. A pixel becomes black when it's lower than a certain threshold and white when it's higher. With a well chosen threshold this will remove unwanted edges and keep the more important edges.

5.1.4. Edge threshold determination

Rather than having a static threshold to be used for the edge thresholding. A more dynamic approach is also possible. By looking at the edges in an image an equation can be used to determine the threshold. For the hand detection equation 8 and 9 have been used.

$$V = \sum_{y=0}^{h} \sum_{x=0}^{w} (1 + \frac{(h-1-y)}{5}) * P_{x,y} \quad (8)$$

$$T_v = \frac{V}{255} * \frac{e}{100} \quad (9)$$

Where h, w is the height and width, $P_{x,y}$ is the pixel value at (x,y) and V is the total value for the image. The threshold T_v is given by equation 10 where e is a variable which can be between 0 and 255. This seems rather strange to do, but it'll become more clear when the complete process is discussed in paragraph 5.2.

5.1.5. Edge transition count

This filter counts the number of transitions. A transition is an area of two by one pixel in which the two pixels are not the same. Or in words: scanning from left to right, the number of times you go from white to black and from black to white. The filter does horizontal scans every five rows. The number of transitions are added up and divided by the number of row scans performed.

5.2. The Complete process

Each individual filter seams rather strange, but the combination of the filters give a good result. The process is able to both find the hand and the hand gesture made. Until now only the low-resolution images are used. For the gesture detection the high-resolution images are used to get a better view on the hand.
First the movement filter is applied to the scan area for each camera. This will result in a similar image as figure 4, but only of the arm. Then the most forward moving pixel is find by scanning all the rows starting at the bottom. Notice that the most forward moving

pixel is actually the lowest white pixel in figure 4. The position is then compared to two previous positions. If the total distance between the new point and the old points is more than a certain value the point is not considered to be a stable point. This reduces the possibility that random noise is detected as a hand.
This gives the basic idea where the hand is, the next step is to positively identify the point as a hand. The first step is to take a sub-frame of the original image containing the most forward moving points. The bounds of the sub-frame are three pixels back (towards wrist) and twenty-two pixels forward. On this sub-frame edge threshold determination is applied. Three things can happen at this stage. The point really is the most forward point of the hand which will give a good threshold for the edge thresholding. The point is not the most forward point of the hand in which case the threshold will be too high. If the threshold is higher than a certain value, it'll not be considered to be the hand. If it's lower than this value, but still quite high since it's not the most forward point of the hand, it'll have an negative effect on the hand gesture detection.
With the found threshold, edge thresholding is applied. This will give binary images as shown in figure 11.

Figure 11

Some of the images have clear distinctive features of hands others have not because the edge thresholding has thrown away many edges since it's not really the most forward point of the hand. Finally the edge transition count is applied. This will give positive detections of gestures when it really was the most forward point of the hand or it'll think it's an unknown gesture when it was not. You can clearly see that with this process the hand gesture detection is connected to the hand detection.
The detected gesture is different per camera, the actual gesture detected by the entire system is determined by the 3D server. This will be explained in paragraph 7.4. The point detected as a hand can be different between two frames. To make the movement go smooth, additional filtering is applied at the 3D server.

5.3. Conclusion

The process on its own doesn't give a robust hand gesture detector. Filtering is an absolute necessity to determine the right gesture. The method is influenced by background edges. This background noise is almost not present in the booth, but becomes a bigger issue when removing the booth. To keep this method working, the camera should be focused on the working area. If the camera is focused in the working area, the background will be out of focus and will therefore give less noise.
Another solution might be posture recognition where a hand is located at the end of the arm. Accurate posture recognition can retrieve the hand location. More complicated methods can then try and determine the hand gesture.

The detection of two hands is a totally different problem. Although it would be nice to have two hands to work with, the hands will block each others view from one camera, making it very hard to see what gestures are being made. Two cameras are needed to detect a gesture successfully. With the current camera setup it's possible to see a camera and both hands at the same place from the other camera (and vice-versa). That leaves one camera for detecting the gestures, which isn't enough. To solve this problem, the cameras have to be replaced so one camera can't see the other camera and both hands at the same location. Other solutions are to add another camera or solve the problem by software.

6. 2D to 3D transformation

All the image processing produces 2D camera coordinates. These coordinates have to be converted to 3D world coordinates. Because the cameras will be separated in the future, the 3D server is responsible for calculating the 3D coordinates. The next paragraphs will show how the world coordinates system is set up and how the 3D coordinates are calculated using the calibration info.

6.1. World coordinate system

The world coordinate system is a right-hand-orientated coordinate system. The origin lies on ground level at the center of the screen. Y is across the screen, Z is up and X is out of the screen. Figure 12 gives a good overview.

Figure 12

6.2. Coordinate calculation

The main purpose of the 3D server is to calculate the 3D world coordinates given a set of 2D camera coordinates. The calibration data described in chapter 3 is used to convert a 2D camera point into a 3D line. Two sets of camera coordinates from different cameras are needed to calculate a point in 3D world space. If two cameras successfully detected the head, the 3D world position can be calculated; see figure 13 for a graphical representation.

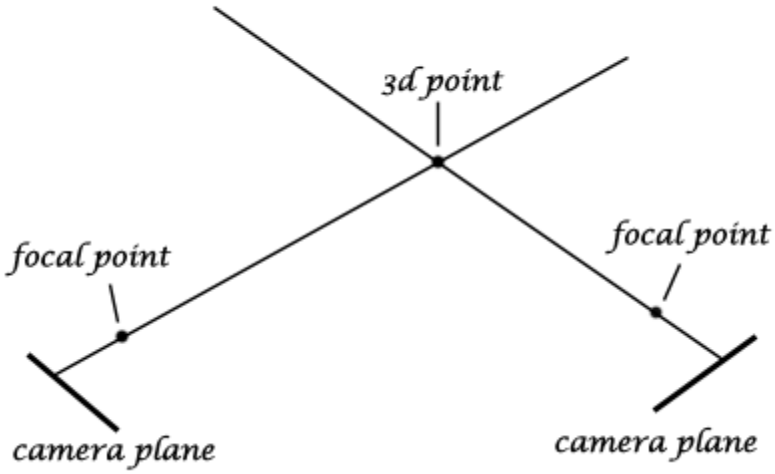


Figure 13

To calculate the position, a vector is constructed for each camera going from the camera image plane through the focal point. The shortest distance between these vectors is calculated. The position is the 3D coordinate in the center of the line representing the shortest distance.

7. Filtering

The 3D points calculated with the raw data from the cameras is not suited as input for the application. The data needs to be filtered first. The first filter tried is an average filter. It is easy to implement and can average out most of the noise. The drawback is that the filter gives a delay. Because the Watching Window is an interactive application, a delay is unwanted. Another possibility is to fit polygons onto the raw data. It has some promising result, but sill not the best result.
Therefore the choice has been made to use kalman filters. These filters produce smooth output while retaining a decent reaction time. The kalman filter also removes any sudden movements caused by the tracker to lose the head and finding it again at a different place. For the gestures a different filter is used. This filter can best be described as a majority vote filter. Aside from filtering the 3D server also decides which gesture is made.

7.1. Average filter

An average filter requires a certain window size to work on. The bigger the window size the smoother the result. But the average filter also makes the response worse. The smoother the result the worse the response. A delay, as told earlier, is unwanted in the Watching Window because this would brake the illusion of the interactive virtual world.

7.2. Parabola fitting

Parabola fitting tries to fit a second order parabola onto the data to be filtered. Since a parabola is used, this guaranties the result to be smooth. It has two problems: the parabola is based on a series of previous values which has a big influence on the current point to be filtered and it can't compensate for abrupt changes. Figure 14 gives a nice example of the results.

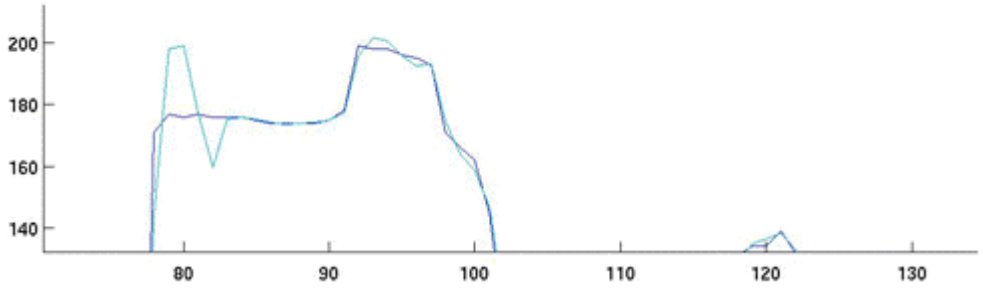


Figure 14

The light line shows the filtered data. It fits well to some parts but it also has a big error where the line goes from a steep angle to a shallow angle. In the Watching Window this would translate to a behavior where the view will move too far forward, too far back and then to the right position when making sudden movements. Since this is unwanted, the filter isn't used.

7.3. Kalman filter

The kalman filter[5] introduced by R.E. Kalman can be used as a predictor or as a filter. The predictor is used when there is a partially known input. The other part of the input is noise. The filter will predict the output based on a certain model. The Watching Window is using the kalman filter with no input and a known output.

7.3.1. Model

Before the equations for the filter can be given, some variables have to be explained first. The movement of the head and hand component wise, can be seen as a linear system with the following state equation:

$$x_{k+1} = Ax_k + w_k \tag{10}$$

and the output equation:

$$y_k = Cx_k + z_k \tag{11}$$

Where k is the time index, x is the state of the system, A and C are matrices, w is the process noise and z the measurement noise. The output equation is replaced for the Watching Window with an equation where the output is the measured point:

$$y_k = p_k \tag{12}$$

The state vector x models the position, velocity and acceleration for the head or hand:

$$x_k = \begin{bmatrix} p_k \\ v_k \\ a_k \end{bmatrix} \tag{13}$$

The matrix A in equation x can then be written as:

$$A = \begin{bmatrix} 1 & t & \frac{1}{2}t^2 \\ 0 & 1 & t \\ 0 & 0 & 1 \end{bmatrix} \tag{14}$$

so it conforms to laws of physics. Because the Watching Window only needs the position p_k, matrix C can be written as:

$$C = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix} \tag{15}$$

The model for the kalman filter is now established. The kalman filter for the Watching Window is given by equations 16 to 18:

$$K_k = AP_kC^T(CP_kC^T + S_z)^{-1} \tag{16}$$

$$\hat{x}_{k+1} = A\hat{x}_k + K_k(y_{k+1} - CA\hat{x}_k) \tag{17}$$

$$P_{k+1} = AP_kA^T + S_w - AP_kC^TS_z^{-1}CP_kA^T \tag{18}$$

Where K is called the kalman gain, P is called the estimation error covariance and $\hat{x}$ is the state estimate. S_z and S_w are the error matrices of the measurement and process noise respectively (equation x and y).

$$S_z = \begin{bmatrix} z^2 \end{bmatrix} \tag{19}$$

$$S_w = \begin{bmatrix} \frac{1}{4}w^2 & \frac{1}{2}w^2 & 0 \\ \frac{1}{2}w^2 & w^2 & 0 \\ w^2 & 2w^2 & 0 \end{bmatrix} \tag{20}$$

To filter the movement of the head and hand the predicted value $\hat{x}$ is used as the correct value.

7.3.2. Results

By changing the process and measurement noise the responsiveness and smoothness of the filter can be adapted. The noise values have been set to a level so there is no apparent delay to the user. The predicted value still has a small error, but is much less visible to the user. Figure 15 gives a good view of the result.

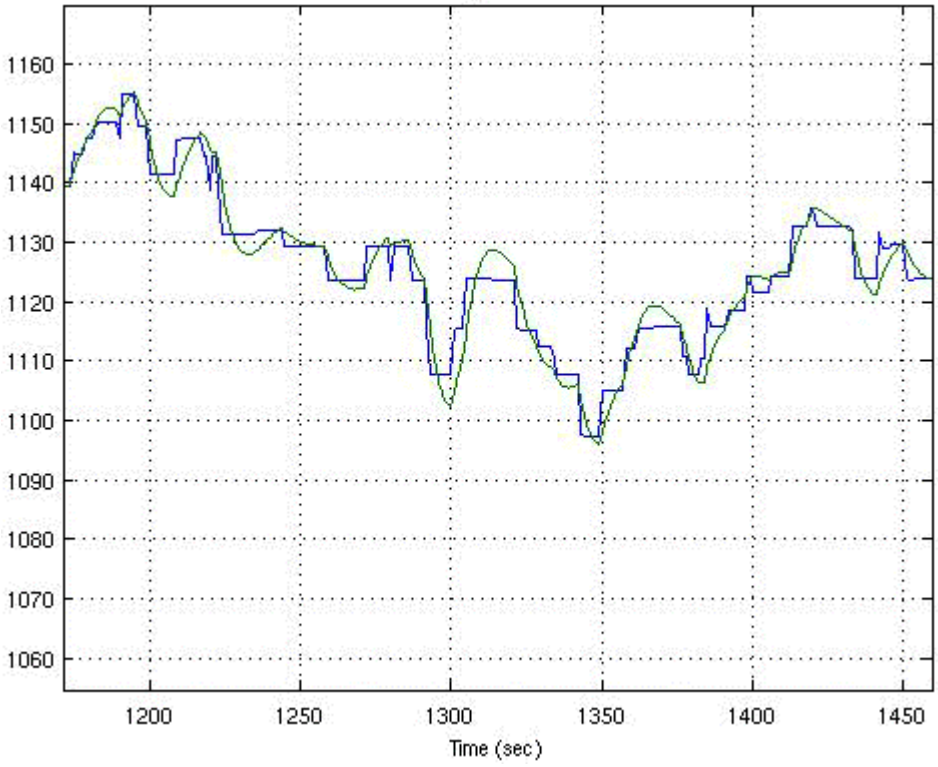


Figure 15

One problem with the kalman filter is that it can 'overshoot'. This means that it predicts the next value at a position too far given the current velocity. The filter isn't able to adapt quick enough to the new velocity resulting in the 'overshoot'. To counter this effect the responsiveness can be adapted, but this results in a less smooth result. The problem isn't that big because it only happens when someone moves his/her head really fast. At the moment of movement that person can't see the screen clear. This gives the filter the time to catch up. This doesn't hold up for the hand movement. The responsiveness for the hand movement filter has been raised, accepting a less smooth result.

7.4. Gesture filter

It is possible that each camera detects a different gesture. To figure out which gesture is really made a couple of filtering steps are done. The incoming gestures from the cameras aren't very consistent. Figure 16 shows this in a graphical form for a single camera.

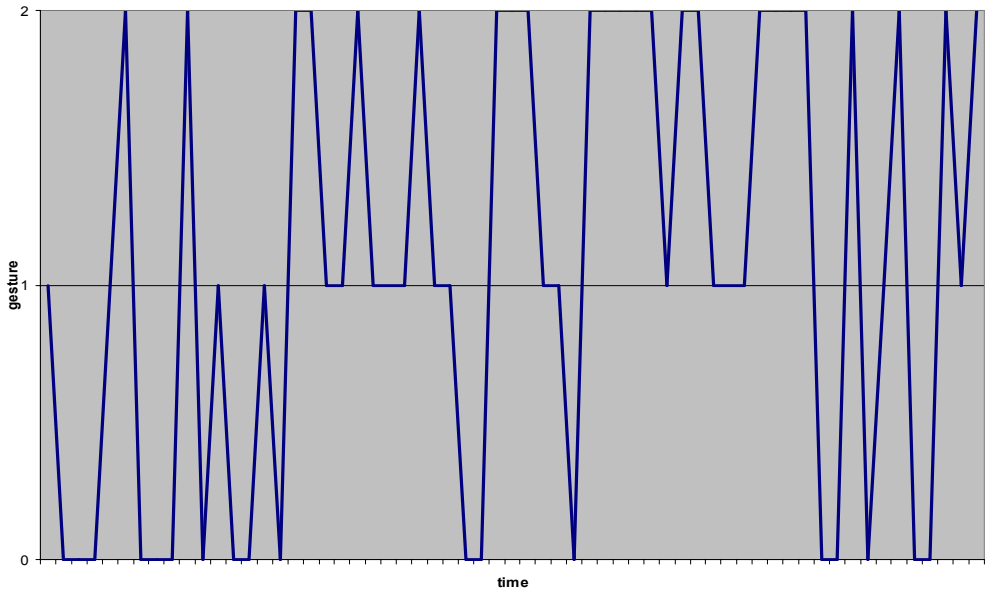


Figure 16

The three different gestures are represented as a number. 0 indicates an open hand, 1 an unknown gesture and 2, a closed hand or pointing finger. It is quite difficult to see what gesture is detected. To make this decision a majority vote is cast on the data. If at least 5 out of the 7 previous gestures are the same that will be the detected gesture for the camera. If no gesture has a majority the gesture is unknown. The result can be seen in figure 17.

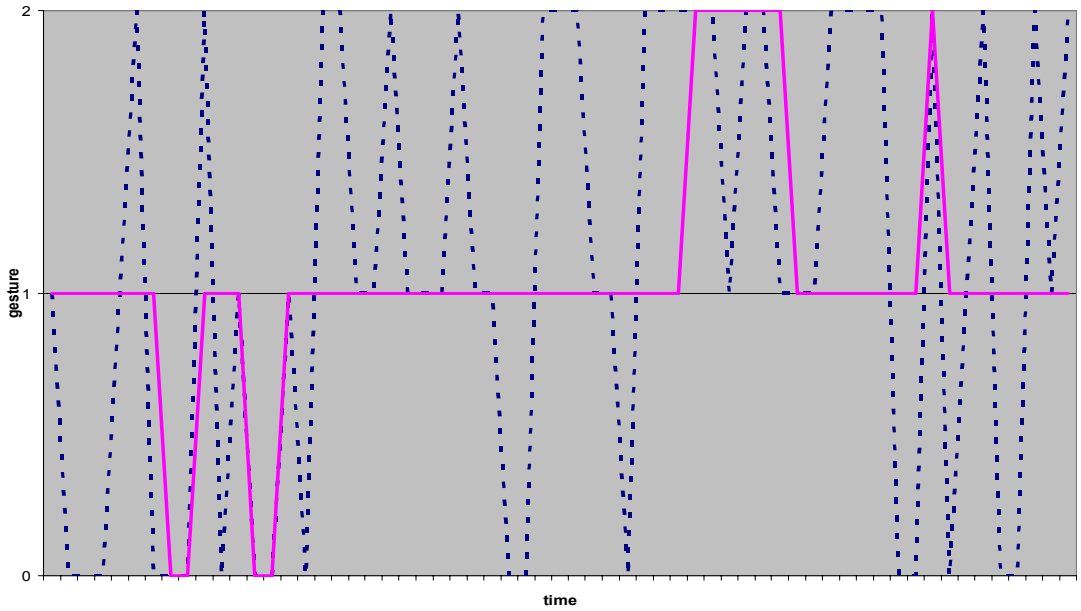


Figure 17

Most of the time the detected gesture is unknown. The result will be better when all three cameras are involved. If one camera positively identifies a gesture, this will be the overall detected gesture. Only one problem remains: what if two cameras detect two different gestures? This happens very often because the top camera is looking from a different angle than the side cameras. An open hand will be detected as an open hand by one camera but as a closed hand by another. Because this only happens when the user has an open hand, the open hand gesture precedes the closed hand gesture. So if an open hand gesture is detected by one camera, the overall detected gesture will be an open hand even if another camera has detected it as a closed hand.

7.5. Conclusion

It is difficult to see how good a filter works by looking at the figures but testing showed that the filters work well and makes the system 'feel' better. They remove the sudden changes caused by failure to detect the head and detecting it immediately after. It even looks like the head tracking didn't fail at all. A filter can only do a certain amount of filtering. For a better overall result, improvements have to be made to the image processing, calibration or world point calculation.

8. Architecture

The system architecture is divided in four parts: the server/client setup, the network protocol, the 3D server architecture and the camera client architecture. The display client architecture is not included since this differs per implementation. Any application can connect to the Watching Window by implementing the network protocol.

8.1. Server/client setup

The Watching Window uses a server/client architecture (figure 18). There is one 3D server which is responsible for the communication and the 2D to 3D coordinate conversion. Any number of camera clients and display clients can connect to the 3D server. Currently all the cameras are connected to one computer, but every camera has its own connection to the 3D server. Future plans include the allocation of one computer per camera (chapter 2), which is already supported by the architecture.

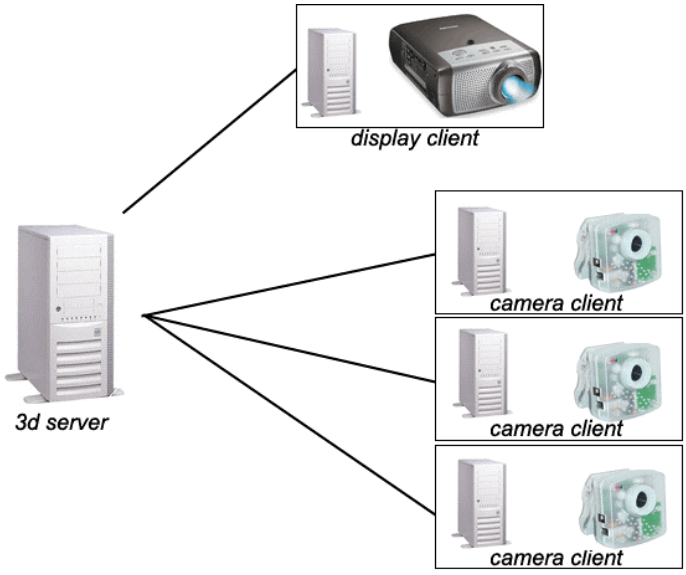


Figure 18

There are several client applications running on the display client. These applications can be seen as example applications which just show the capabilities of the Watching Window. There are more interesting developments happening at the moment. As mentioned in the introduction, a virtual modeling program is being made which will use the network protocol to communicate with the 3D server.

8.2. Network protocol

The previous network protocol stated that complete objects would be send between the clients and the server. A problem with this approach is that different platforms have different methods for storing objects. The result is that applications on different platforms couldn't communicate which each other. The ASCII standard is chosen to enable the communication between all different platforms. Sending string representations of variables requires more data to be send. This isn't a problem since only a maximum of 80

bytes is send each time. With an update rate of 60 times per second, the maximum transfer rate will approximately be 4800 bytes per second.

8.2.1. 3D Server - camera client communication

When a camera client connects to the 3D server through a socket, it will send the header package to the server. This package contains five items, each written as a four character number. The package type defines what kind of package it is. The client type tells the server what kind of client is connecting. The next member gives the camera id. The last two members are used to define if that camera has head and/or hand detection enabled. If one is set to false, the 3D server will ignore the coordinates coming in (see appendix A.1).
When the 3D server receives the header package from a camera client, it will respond with a 'ask for calibration info' package which triggers the camera client to send its calibration info. The calibration info is read directly from file. The file layout is determined by the calibration program. The 3D server parses the incoming calibration package and sets the internal calibration info for that camera. The initial communication is completed when the 3D server sends the 'ready to accept' package, indicating it's ready to accept camera coordinate data.
After the initial communication is done, the camera client can proceed sending 'body info' packages. These packages contain camera coordinates for the head and hand, and the detected gesture. The four character number is a fractal between zero and one multiplied by ten-thousand. The reason for this choice is that a fractal will be the same for all resolutions; a point (160,120) taken at a resolution of 320 by 240 will have the same fractal coordinates (0.5, 0.5) as a point (320,240) taken at 640 by 480. This makes the 3D server independent of any resolution used by the camera client.

8.2.2. 3D Server – display client communication

The communication between the 3D server and display client also starts with a header package. The display client only sends its type. The 3D server receives the package and sends the 'ready to accept' package. The display client and the server are now connected. The display client can request new data by sending the 'request data' package. The 3D server will compute a new set of 3D world coordinates and it will send these back along with the filtered gesture (see appendix A.2). Each of the 3D world coordinates are formatted as 12 character long doubles.

8.3. 3D Server architecture

Figure 19 shows a simplified class diagram of the 3D server. All variables and methods are not shown.

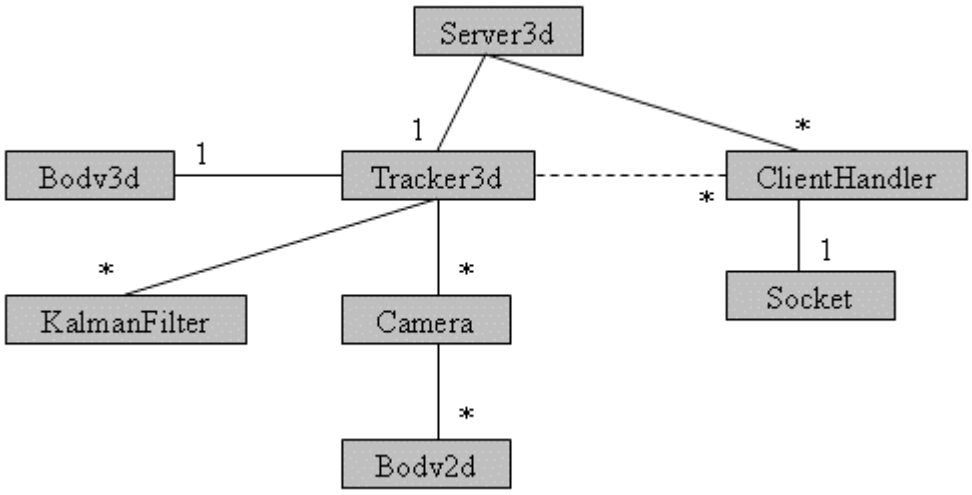


Figure 19

The main class is Server3D, this class initiates connections and holds the main processing class: Tracker3D. The Server3D class has one thread for each ClientHandler and one for setting up new connections. The ClientHandler class uses a socket to connect to either the camera client or the display client.
The Tracker3D class converts 2D coordinates into 3D coordinates. The Tracker3D class and the ClientHandler class have pointers to each other for easy data passing. If a camera connects to the 3D server a camera object is created as well. This object holds the calibration info and a history of previous received data from that camera. The Tracker3D class converts the Body2D data in the history to Body3D data. This data is filtered using the KalmanFilter class. A KalmanFilter object can be created to filter any 3D world coordinate.

8.4. Camera client architecture

Figure 20 shows a simplified class diagram of the camera client program. Currently all cameras are run by a single program. The rectangle around some of the classes indicate the architecture for a single camera implementation All variables and methods are not shown.

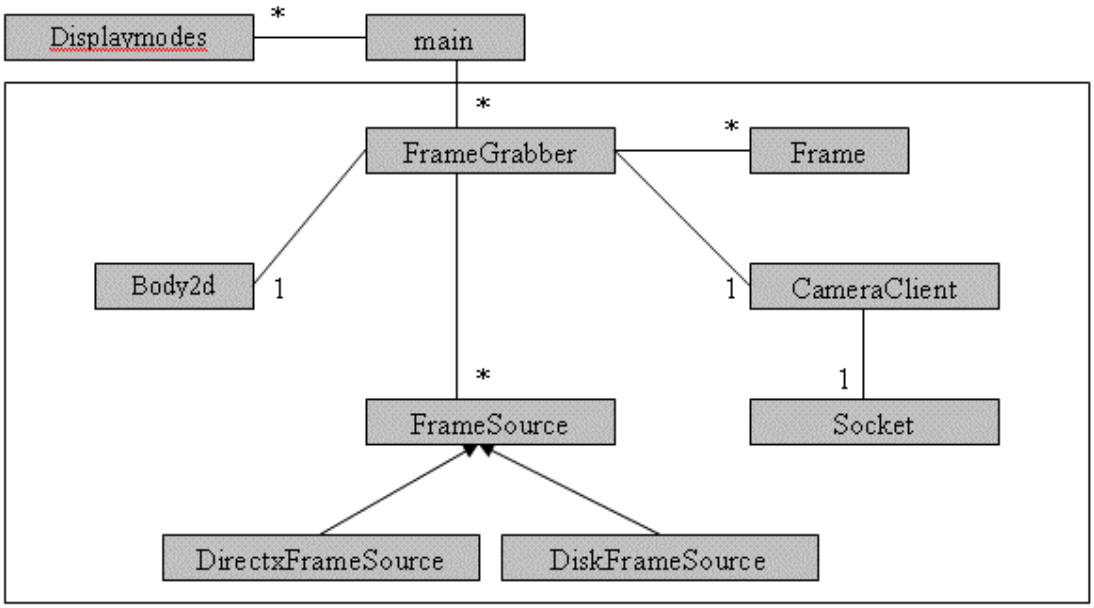


Figure 20

The main can hold any number of FrameGrabber objects. The FrameGrabber class is responsible for controlling the cameras and network connection. The DisplayMode class does the image processing. The FrameGrabber class holds the current detected body parts in a Body2D object, holds the last captured frames in Frame objects, communicates with the server through a CameraClient object and captures frames using the FrameSource class. The FrameSource class is extended by DirectxFrameSource and DiskFrameSource. DirectxFrameSource is responsible for capturing the frames using DirectShow. The DiskFrameSource class can load previously saved images from disk as if they were new captured images.

9. Other things tried

The first approach to improve the system involved looking at the 3D information rather than the 2D information from the cameras. The idea is to let the image processing produce multiple candidates per camera for the head or hand. These candidates can then be converted to a 3D volume. The intersections of these volumes would then eliminate certain candidates or even find the head or hand immediately.
There were several problems with this approach; the image processing was not set up to give a single candidate rather than multiple candidates. The results didn't look very promising and were really dependant on the image processing. Therefore it was decided to focus on the image processing rather than trying to improve this approach. Figure 21 shows voxel carving, where the intersections of the three green outlines produce the grayscale image.

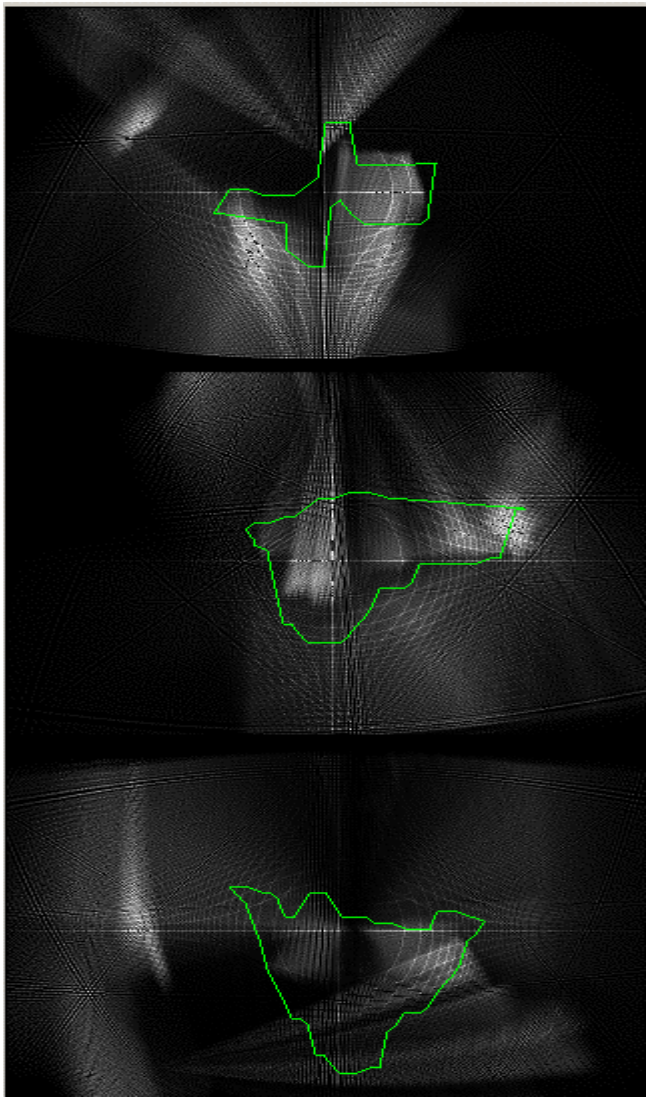


Figure 21

10. Conclusion

The Watching Window is a big project which has been under development for some years now. The goals presented in this paper fit well in the long term goal. The realization of these goals have created a version of the Watching Window which is capable of detecting the head, hand and the hand gesture of a user. The system runs at about 12 to 14 frames per second. This will be much higher when the system uses a single computer per camera setup.

The network code has been changed so it can be implemented on any platform. The current version can be extended to improve robustness of the head and hand detection. New gestures can be added to add functionality to the applications running on the display client.

All the code has been cleaned up, comments have been added and other things to know have been documented. This should provide a good basis for future work.

11. Recommendations

Some recommendations have already been made in the individual chapters. To sum up:

- Find an alternative for the current booth color. Studies on several different background textures should produce a better alternative.
- Allocate one computer per camera, this will give much more computing power. This can then be used to do more computational expensive algorithms or to speed up the system. An improvement in speed will have a positive effect on the detection rates.
- Improve the calibration data. The radial distortion could be modeled by hand. Different calibration objects and/or algorithms might give a better result.
- To improve the eigenface detection rate, training images from people looking down can be included. This should only be done when enough computing power is available.
- The performance of hand detection also depends on the background. The current implementation using edge detection will not work when the booth is taken away. Other solutions like posture recognition should be considered. When two hand gestures are to be detected, another camera is needed or the cameras have to be repositioned so the two hands are not on the same line as two cameras.

12. Personal evaluation

I enjoyed my time at the university of Otago. The first few weeks were quite hard. The previous student that worked on the Watching Window hadn't delivered his report yet, leaving me without any documentation. I worked my way in it by doing small projects first like implementing threads for the 3D server. The first idea I had cost me two or three weeks of my time. But the time I had left was enough to do some good work.
A lot of time was spent on trying ideas and testing. I could have done some more programming work to improve general architecture and readability, but my main concern was to achieve the main goals and write up in time.

Every Wednesday there was a meeting with Geoff Wyvill, Brendan Mccane and two other students working on the client applications for the Watching Window. These meetings were good for getting feedback and to make sure we were still working towards the main goals.

I think I did a good job. The Watching Window has a long term goal in which the current implementation fits perfectly. There is enough and accurate documentation to help the next student get to know the system.

13. References

[1] Alexis Angelidis, Marie-Paule Cani, Geoff Wyvill, Scott King. Swirling-Sweepers: Constant-Volume Modeling. 2004

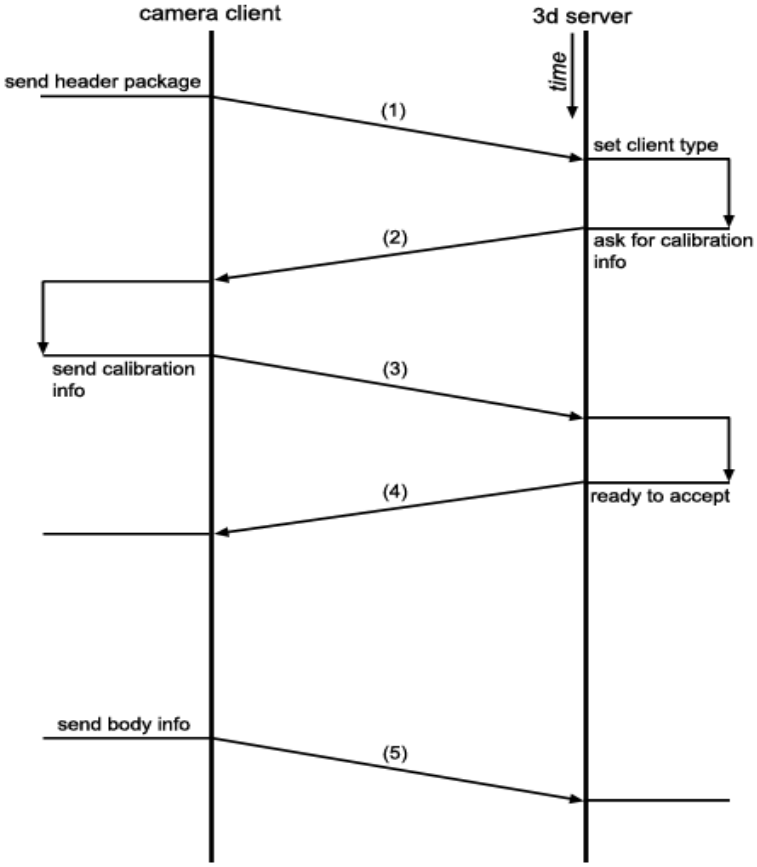
[2] Roger Y. Tsai. An efficient and accurate camera calibration technique for 3D machine vision. *Proceeding of IEEE conference on computer vision and pattern recognition*, Miami Beach, FL, 1986, pp 364-374

[3] G. Schuang, Eigenfaces for the Watching Window, 2004

[4] Alex P. Pentland, Matthew A. Turk. Face recognition using Eigenfaces. *Journal of cognitive Neuroscience, 3(1)*, 1991

[5] R.E. Kalman. A New Approach to Linear Filtering and Prediction Problems, *ASME*, 1960.

Appendix A.1.1



Appendix A.1.2

Package numbers correspond to numbers in appendix A.1.1

Send header (1)

Field	Offset	Length	Type	Value
Package type	0	4	ASCII	“0001”
Client type	4	4	ASCII	“0001”
Camera id	8	4	ASCII	Any number
Use head detection	12	4	ASCII	“0000” false “0001” true
Use hand detection	16	4	ASCII	“0000” false “0001” true

Ask for calibration info (2)

Field	Offset	Length	Type	Value
Package type	0	4	ASCII	“0002”

Send calibration info (3)

Field	Offset	Length	Type	Value
Package type	0	4	ASCII	“0002”
Calibration info	4	658	ASCII	From file

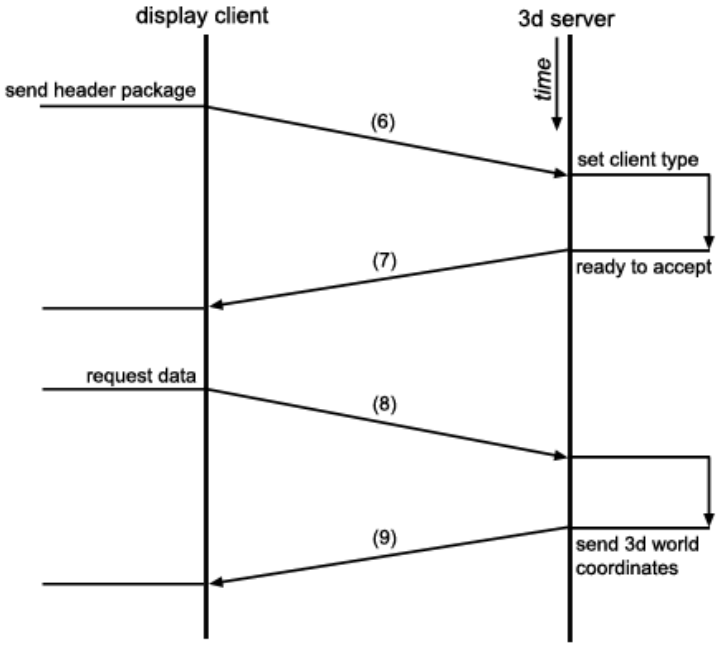
Ready to accept (4)

Field	Offset	Length	Type	Value
Package type	0	4	ASCII	“0003”

Send body info (5)

Field	Offset	Length	Type	Value
Package type	0	4	ASCII	“0004”
Head x	4	4	ASCII	Any number
Head y	8	4	ASCII	Any number
Hand x	12	4	ASCII	Any number
Hand y	16	4	ASCII	Any number
Gesture	20	4	ASCII	“0000” (open hand) “0001” (unknown) “0002”(closed hand)

Appendix A.2.1



Appendix A.2.2

Package numbers correspond to numbers in appendix A.2.1

Send header (6)

Field	Offset	Length	Type	Value
Package type	0	4	ASCII	“0001”
Client type	4	4	ASCII	“0001”

Ready to accept (7)

Field	Offset	Length	Type	Value
Package type	0	4	ASCII	“0005”

Request data (8)

Field	Offset	Length	Type	Value
Package type	0	4	ASCII	“0006”

Send 3D world coordinates(9)

Field	Offset	Length	Type	Value
Package type	0	4	ASCII	“0006”
Head x	4	12	ASCII	Any double
Head y	16	12	ASCII	Any double
Head z	28	12	ASCII	Any double
Hand x	40	12	ASCII	Any double
Hand y	52	12	ASCII	Any double
Hand z	64	12	ASCII	Any double
Gesture	76	4	ASCII	Any double