

Eyecursor

December 8, 2002

1 Abstract

Eyecursor is a project with the aim to track the eye for mouse cursor control to allow handicapped people the usage of the computer. To allow the usage for everyone the hardware has to be as simply and cheap as possible. The final result uses a standard webcam on an old motorcycle helmet to get a video from the eye's movement and tracks the pupil to transform the pupil's coordinates into screen coordinates.

2 About the project

2.1 The Team

Programming: Peter Hutterer, Matthias Bauer
3D world modelling: Wolfgang Hafenscher
Director application: Andrea Durchschlag

Coach: Michael Haller

This project was made in the 4th semester (March 2002 to June 2002) of the University for Applied Sciences Hagenberg, Media Technologies and Design (<http://mtd.fh-hagenberg.at>). It was part of the subject ASP and developed within 4 months in our freetime beneath the school.

2.2 Technologies used

- C with usage of the Win32API, OpenCV and the IPL
- Microsoft Visual Studio 6
- Macromedia Director 8.5
- discreet 3D Studio Max 4.0

Hardware:

- Philips ToUCam Pro
- an old Motorcycle helmet
- 2 standard PCs (650 and 800 Mhz) with a Windows 2000/XP installation

2.3 The award

For "Eyecursor" we won the 1st Price at the MTDGala 2002 (<http://www.mtdgala.net>) in the category "Mediatechnologies".

3 The project

3.1 The target

The project's main goal was it to allow the access to the computer without using the hands to control the mouse. To have everyone be able to use the program the hardware has to be as cheap as possible so we decided to use a standard webcam. This webcam should get a video stream from the eye where the software can start finding the center of the pupil to get the eye's coordinates. This coordinates should be transformed into screen coordinates to have the cursor point to the point where you currently are looking at. For demonstration purposes a small application should be developed where the user can move around in a 3D virtual world just using the eye as control device.

3.2 The hardware

One of the main problems is to get always the same video of the eye even if the user moves his head a little bit. To solve this problem we used an old helmet where we mountet the camera on. With some hinges to bring the camera in front of the eye. With this method the camera overlaps part of the eye's view, which could be solved by using a semitransparent mirror in front of the eye and the camera filming the eye over the mirror.

3.3 Eye tracking

3.3.1 The laser method

Our first thoughts were to let a simple laser or LED point to the eye to get a small red point in the eye. By tracking the angle between the darkest point (the pupil) and the red point you may get the pupils coordinates. Since we didn't have a laser and we thought it may be not a good idea to point a laser over hours into ones eye we trashed this idea, but it might be still useful for eye tracking if the usage time is short enough.

3.3.2 The floodfill method

The eye's pupil is black, we thought, so it must be easy to find. In each frame we searched for the black pixels and the average coordinate of the black pixels has to be the center. Since the pupil is not really black due to reflections we use a treshold to differ between "black" pixels and those not black enough. Due to the fact that there can be very dark shadows you get a loft of distortion in your image, about 3 to 7 big black spots and lots of standalone black pixels in opposition to the one great spot (which should represent the pupil) we thought of. This was the time when we used floodfill. On every black pixel we started the recursive floodfill algorithm to get all connected black pixels (within a 8 neighbourhood). For each started floodfill we count the number of connected pixel, the largest concentration has to be the pupil. The center (average coordinate) of the pupils black pixels has to be the center of the eye.

This version was the first runnable version and the results were nearly usable. The main problem was that if you are staring on a very light monitor (i.e. if you have the explorer window

open) the mirror image of the monitor covers about half of the pupils total area. The center point detected by the algorithm is not exactly in the center. There are also problems with very dark light because all shadows can be connected and the whole eye gets a big dark spot, which cannot be really solved (manipulating the treshhold did somehow work but not good enough). And as a last problem the recursive method caused the usual problems when used on large spots.

As a conclusion we decided that the method works fine but not fine enough to be usable for our target, even if we rewrite the recursive floodfill with the region labelling algorithm.

3.3.3 The prediction of the triangle height

One other problem of the floodfill was, that the delay between the eye movement an the final coordinates was about half a second to a second which is for a realtime application not usable.

To speed up the calculation we have to use scanlines to split up the image in just a handful of pixel data.

The scanline algorithm used a specified y coordinate in the image and started running from the left to the right edge of the image. First every pixel with a color value over a certain treshhold was marked as white, every pixel under a certain treshhold was marked as black.

In the next step within a stepwidth of 30 pixels the number of white and black pixels are counted. If the number of white pixels exceeds the number of black pixels all 30 pixels are marked as white or in the contrary marked as black. This was necessary because some standalone pixels can be darker than the pixels left and right which would cause problems lateron. As you may see, the problem with this method is, that maybe the pupil's edges could be falsified (if 5 dark pixel of the pupil are within a step with the 25 pixels left/right of the pupil the 5 pixel would disappear). To avoid this, the stepwidth has to be variable. If the last pixels within a stepwith are all black, the current step only goes to the last white pixel, then the next step starts. In example: your current step has 25 white pixels and then the 5 black pixels of your pupil. So the current step only goes to the last white pixel and the next step starts at the first black pixel.

This algorithm gets you a really exact mark of the pupil. However, due to shadows in the region around the lachrymal gland you get more than one dark regions in the scanline. We solved this problem by using those region which is closest to the center, which is nearly ever the right one. The center of the region is the x coordinate of the pupil's center.

The problem is, that this method won't really work with vertical scanlines, because the eyelashes and the lid cause to much distortion. So the solution was to calculate the y coordinate out of the x coordinates. The scanline method works very good with brown eyes but also works with blue eyes. Since the radius of the iris always stays the same you can calculate the distance of the center of this circle if you have the length of the secant through the circle. The secant is our region marking the start and endpoint of the iris. As an example: If your region's length equals the iris' diameter then the center has to be on the scanline. If it differs, the center has to in some distance, which can be easily calculated by using some simple formulars.

So you get two centers of the eye, but you still don't know if the center is north of your scanline or south of it. This can be solved by creating a new scanline and comparing the length of both scanline. If the northern scanline's length is smaller than the other one, the center is south of both of them, in the contrary the center is north. If the center is in between the two scanlines it is mainly the same.

One problem can be, if the length of the scanline is somehow wrong because of reflections in the eye or some other distortions. We solved this by using 30 scanlines and calculating the center of the eye for each combination of scanlines. So the southernmost scanline calculates the center position together with the scanline north of it, then with the next one and so on. The coordinates wich got the most hits, were seen as the eye center.

This method worked in real time and the coordinates were acceptable, but still not the best.

3.3.4 The Hough Transformation method

The Hough for circles needs a parameter space for your image (which is nothing but a matrix of the same size your image has). For each pixel in your image which might represent the edge of a circle you draw a circle in the parameter space with your eye's radius. Drawing a circle means you have to use the Bresenham algorithm to run through the parameter space matrix and on each position of the circle increase the matrix value by 1.

The point in the parameterspace with the highest value specifies the center of the circle which hits the most points in your original image.

This method is rather complex, even if you know the radius of the eye exactly, so it caused a delay of about 2 or 3 seconds, making it completely useless. To avoid it we combined the Hough with the scanlines. Instead of using the Hough for circles for the complete image we just used it on the edges of the iris we detected with the scanlines. The rest works exactly the same.

This method works in real time and is one of the two methods finally used in the program.

3.3.5 The average coordinate method

Sometimes the simplest methods work the best: The average method just counts the number of dark pixels (= pixels with a color value lower than a treshold) and adds their coordinates. Then the average of the x and the y coordinate is calculated and voila - that's the center. If the number of black pixels exceeds a certain number the treshold is decreased, if there are not enough black pixels, the treshold is increased. With this method it needs some time (about half a second to one second) until the treshold is optimized but it is still the method which works very simple and efficient.

This method is the second method finally used.

3.4 The final software

Our final software contains 2 main features: cursor positioning and automatic scrolling.

3.4.1 Cursor positioning

This function was the main goal of the whole project: To position the cursor where the user looks at. This was our greatest problem too, because the resolution of the webcam is much too low to realize this function correctly. At a resolution of 320x240 the eye's center moves around about 100 pixels if you look from left to right. If you just look from the left to the right edge of a standard screen this the movement is about 20 to 30 pixels. Now you have to transform 20 pixels towards a resolution of about 1024x768 or more. Even if the center detection would work 100% exactly the error would be about 50 pixels. This problem couldn't be solved, so we did a workaround: Our screen is split up into 9 areas of the same size. Every time the software realizes that you changed your view into another area the mousecursor is positioned within this area. As long as you look into this area, the mousecursor can be moved with your mouse.

3.4.2 Automatic scrolling

If this function is activated, the currently active window is scrolled up or down if you look at your screen's upper or lower edge. For MDI application this function for the active document.

3.4.3 Additional software

As an additional software we made a small world with 3D Studio Max and Macromedia Director where the user can move around by using the Eyecursor Software. With the keys 1 - 3 the user can decide whether to look, move or slide, with his eyes he can control the direction to look/move/slide.

4 Final conclusion

Working on this project was very interesting for all of us. Although this program never got out of the alpha stadium it was interesting to test the different methods to find the pupil. And the main thing we learned is how great the human brain is in comparison to a standard computer (at least in the section image recognition). Something everybody of us is able to - to find a pupil within a image of an eye - needed 4 months of hard work, endless hours of programming and the ability to overdrive frustration to get not even close to the capability of the human brain. The main goal of this project was not reached but a possible usage of the program would be automatic eyetracking during usability tests. It might even be interesting for the advertising industry or film industry because you can easily find out the point the user is looking at and so have the knowledge to position your advertisements better.

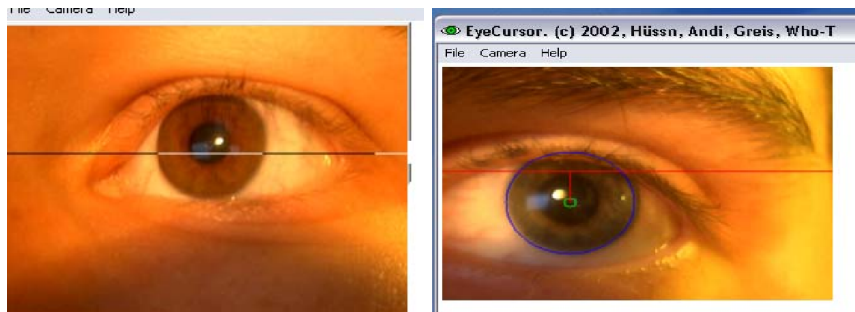


Figure 1: The scanline algorithm

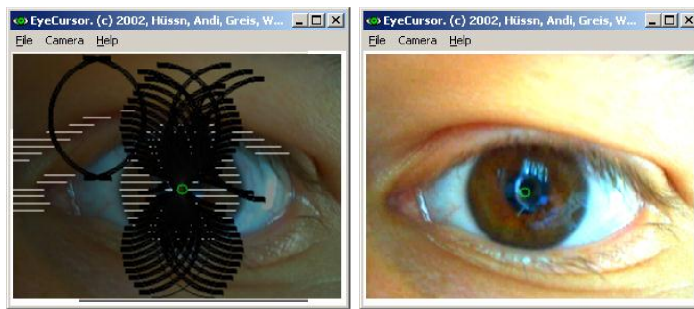


Figure 2: Hough transformation for circles and the average coordinate method

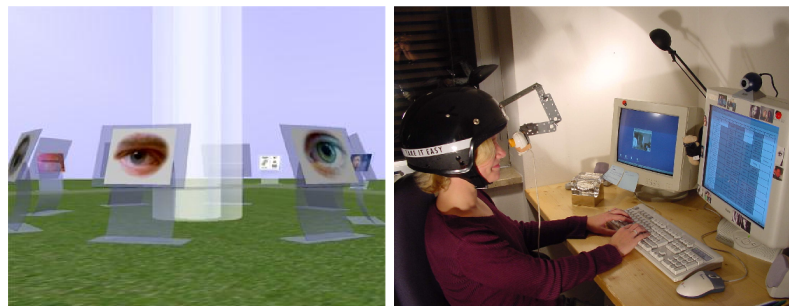


Figure 3: Snapshot of the 3d world and Andi using the *EyeCursor*