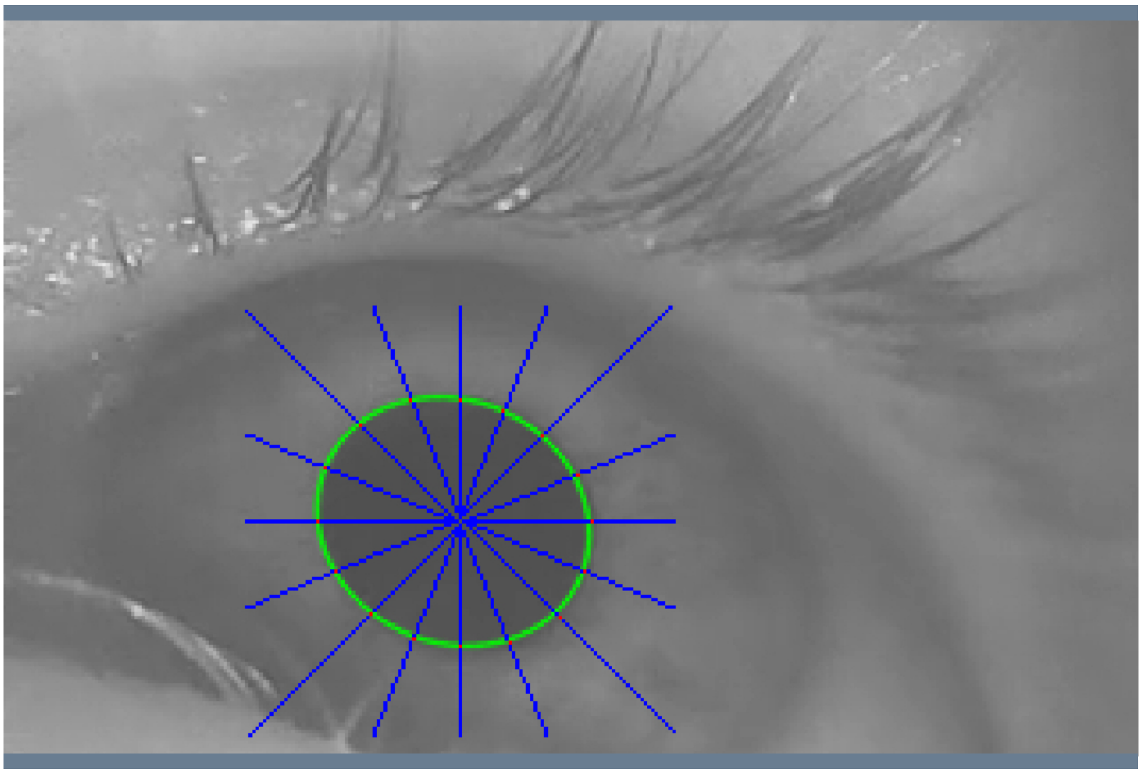


Pupil Detection for a Portable Eye-Tracking System

Bachelor Thesis

Degree course:	Electrical and Communication Engineering
Author:	Timothée Mollet
Tutor:	Prof. Dr. Theo Kluter
Expert:	Felix Kunz
Date:	30.07.2016

Berner Fachhochschule | Haute école spécialisée bernoise | Bern University of Applied Sciences

Versions

Version	Date	Status	Remarks
0.1	15.06.2016	Draft	Document created
0.2	30.07.2016	Draft	First Draft
1.0	30.07.2016	Final	Corrections

Management Summary

Eye-tracking offers important insights about hand-eye-coordination and visual search techniques of an athlete. An eye-tracking system for sports needs a high accuracy, portability and high temporal resolution.

Such a device is under development at the Institute for Human Centered Engineering. The system uses two cameras per eye to capture infrared images. The images should be used to determine the gaze direction of the athlete. An important component for that is the digital recognition of the pupil. This is currently not implemented satisfactory.

Until now, the detection of the pupil was achieved with a hardly tested algorithm. This algorithm should be improved so that it is more stable and accomplish a better accuracy. Additionally, the algorithm needs to run on the portable hardware of the eye-tracking system.

Test data is required to compare the accuracy of the algorithms. This data needs to consist of known actual values. To achieve that, an animated and accurate model of the eye and the eye-tracking system should be created. This model should be flexible so that pictures for different scenarios can be generated. The rotation of the eyes during the animation builds a reference for results.

The algorithm uses the pictures from the model to generate pupil data. A partner work uses the pupil data to calculate a gaze direction. It is now possible to determine measurement uncertainty with the known rotation of the model as reference.

Contents

1. Introduction

Eye-tracking is utilized to monitor eye movement. It is employed in a wide variety of fields such as marketing research and sports. In sports, eye-tracking offers important insights about hand-eye coordination and visual search techniques. These differ greatly between professional athletes and beginners. So the analysis can improve the efficiency of training.

An eye-tracking device for sports needs a high accuracy and temporal resolution to draw accurate conclusions. It should be portable to not interfere with the activity of an athlete.

1.1. Current Development

In collaboration with the sport research institute at the University of Bern, the microLab research group has developed an eye-tracking system for sports. This system called Gazelle uses two cameras per eye to capture infrared images at a high speed. The information from the cameras should be used to determine the gaze direction of the athlete. An important part of that is the detection of the pupil, which is not implemented satisfactory yet.[3]

1.2. Overview of the Eye-tracking System

The Gazelle eye-tracking system consists of two main parts. The first is the GazelleGlasses where the cameras are mounted and the data is sent down to the second main part that is GazelleCompute. As the name suggests this component is responsible for the processing of the data.

GazelleGlasses

GazelleGlasses are glasses that an athlete can wear without obstructing the activity. For the capture of an eye two cameras are placed below the eye and besides the nose. This positioning was chosen to not obstruct the view of the athlete and offer a good perspective on the eye. A front facing camera is placed between the eyes and provides a view that is close to the view of the athlete. The cameras are equipped with a filter to let infrared light through. This produces a better picture of the pupil. Infrared LED illuminate the eye for a brighter picture.

Data that is generated by the cameras need to be transferred to GazelleCompute. This is done by serializing the data and sending it over an Ethernet cable to the main compute unit.[3]

GazelleCompute

The main processing is done on a small portable unit. The units main components are a Zynq FPGA and a Tegra 3. Data from the glasses is deserialized and read by an FPGA. Initially the idea was to do the image processing on the Tegra 3 as it is a powerful mobile System on Chip.

Problems with the data transfer from the FPGA to the Tegra 3 scrambled that idea. The image processing is now required to run on the dual core ARM CPU that is on the Zynq FPGA.[3]

1.3. Find the Gaze Direction

The gaze direction should be determined With the image data from GazelleGlasses. This consists of the digital detection of the pupil as an ellipse for each camera. It needs to be very fast because it has to run on a dual core arm CPU with limited processing power. The algorithm for a stable detection of the pupil is described in chapter 3. A second algorithm is applied in a partner work that calculates the gaze direction with the two ellipses from one eye.

Reproducible test data is needed to verify the functionality of the whole process. Generating such data from real captures of the cameras would be ideal. This is not possible as the images can't be transferred to the Tegra 3 where the processing power to encode the video stream would be available. An alternative is an accurate model of the eye and the glasses. This model is presented in chapter 2. It allows to simply generate animations where the rotation of the eyes and many additional properties can be controlled. The output is pictures that reassemble captures of the actual cameras. Additionally, the parameters of every object is logged for every frame that is rendered.

The methods to calculate the gaze direction where tested with the pictures as input and the parameters as reference. The influence of different resolutions and algorithms is investigated in chapter 4

2. Model

A very important step in hardware and software development is testing. For an eye tracking system it requires a big effort to do this with real data. The reason for that is the information where the participant looks at each moment needs to be recorded alongside of the video streams. For this project real time data processing is needed as there is no hardware available on Gazelle Compute to encode or store the video streams of the cameras.

A simple way of generating new data where the gaze direction and the camera positions are known would simplify the process of optimising the algorithm.

2.1. 3D-Modeling Software

The 3D-Modeling software to create the model needs to fulfill a few requirements:

- Correct representation of refracting light
- Create animations
- API that allows scripting
  - Set position and rotation of certain objects at a certain frame
  - Set camera properties and which camera is active
  - Read the position and rotation of objects at any frame
- (optional)Be free
- (optional)Run on Linux
  - Server that can be used to render runs Linux containers
- (optional)Be easy to learn

Blender is able to match all requirements although the last one might be debatable. You can apply a material property to a surface. This includes refracting. Rotation and position of objects can be inserted as key frames and it interpolates the steps in-between.

Blender is also an open source project that is built with Python and has a powerful API that gives access to nearly everything.

2.2. Requirements for the Model

In order to create a flexible model that is close to the reality, the model needs to follow the following requirements:

- Correctly represent a human eye
- Accommodate for different eye parameters
  - Eye diameter
  - Distance between Eyes
- Correctly represent camera placement and rotation
- Correctly represent camera properties
  - Resolution

  - Frame rate
  - Field of View
- Gaze direction can be animated
- Classes rotation and position can be animated
- Front facing cameras records gaze direction

2.3. Human Eye

A side view of a human eye model is visible in 2.1. The camera can only see the front part of the eye. As a result only the cornea, anterior chamber and the iris need to be modelled correctly. The lens can be simplified by a black surface.

It might be counter intuitive that the cornea extends as much out of the eyeball as visible in 2.1. An easy way to verify that is to close the eye and move the eyes while holding a finger on the eyelid.

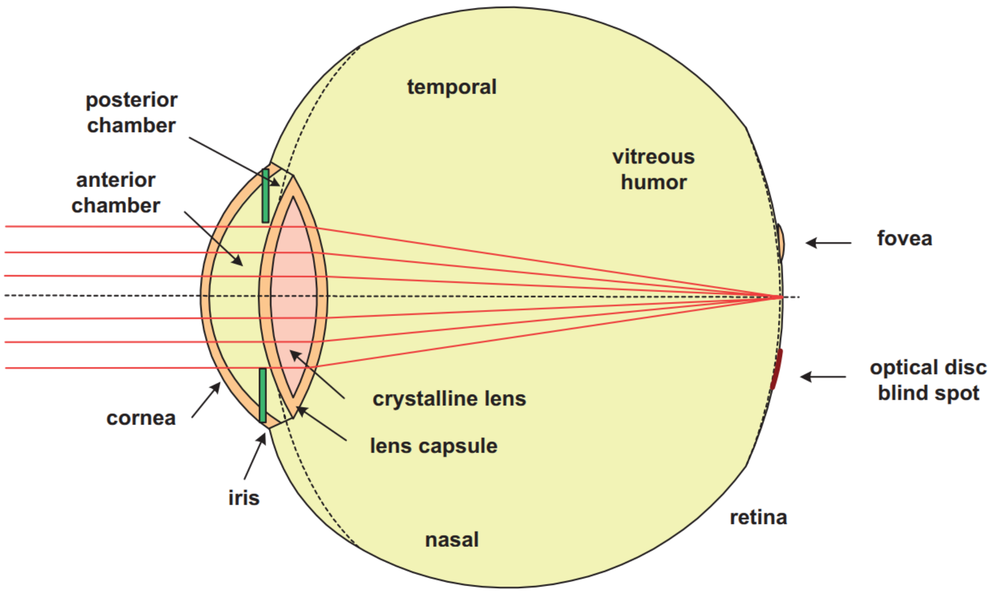


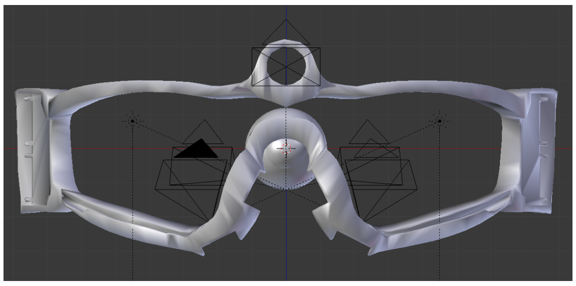
Figure 2.1.: Side view of a human eye model [2]

There are a few models with different values for the refracting indexes, radii and distances but they vary very little. So the simplified Gullstrand eye was chosen for the model.

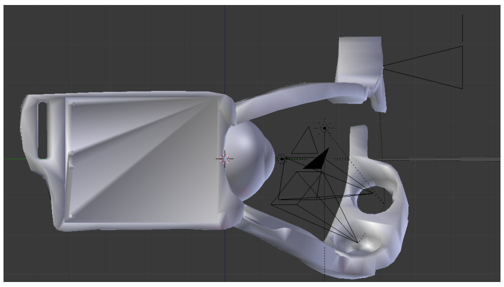
Notaton	Relaxed			Accomodated		
	Radius r [mm]	Thickness d [mm]	Index n	Radius r [mm]	Thickness d [mm]	Index n
cornea	7.70	0.50	1.376	7.70	0.50	1.376
anterior chamber	6.80	3.10	1.336	6.80	2.70	1.336
crystalline lens	10.0	3.60	1.4085	5.33	4.0	1.426
vitreous humor p	-6.00	17.187	1.336	-5.33	13.816	1.336

Table 2.1.: Properties of the eye with the simplified Gullstrand eye. [2]

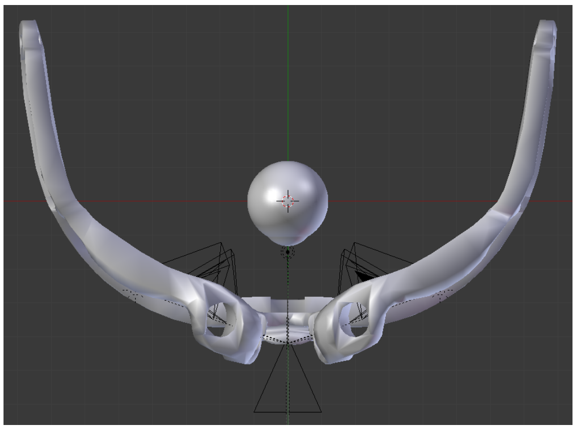
A human eye has on average a diameter of 24mm and the distance between the eyes ranges from 56mm to 72mm with the mean for men at 65mm and 62.6mm for women. [2]



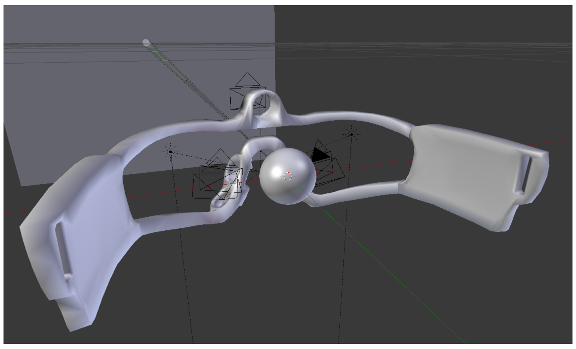
(a) Front view of base model



(b) Side view of base model



(c) Bottom view of base model



(d) 3D-view of base model that shows also the light cone that follows the eye rotation

Figure 2.2.: Finished base model with eyes in the center. A Script will place the eye correctly and generate an animation from the model that is shown here.

2.4. Base Model

While there are quite a few parameters that need to be adjusted, some will remain the same. This can be incorporated into a base model that can then be adjusted by a script for each render.

This base model is centred around an accurate eye model that is built with the properties mentioned in section 2.3. Every eye is a group that can be manipulated with a script as one object. This includes a directed light cone that points where the eye is looking.

An existing model of the glasses is used and placed to approximately represent the position on a human head. Cameras are placed into the their cutouts in the glasses. The rotation is approximated to capture an average positioned eye correctly. The light cones that represent the gaze direction illuminate a small portion of a wall, which is captured by the front facing camera and is visible in figure 2.2d. Similar to the eye, the cameras and the glasses build a single object that can be moved and rotated. The result can be seen in figure 2.2

2.5. Scripting

The base model alone is not enough to create meaningful results. Because there is no animation, the eyes are not placed and Blender would just render one camera. Additionally, the position and rotation of the eye and glasses needs to be logged for every frame. Otherwise, they can't be compared with the final gaze direction results. A simple way of configuring the script is needed to allow a wide variety of animations to be created.

Automate Animation

The script is written in Python because Blender offers a powerful Python API that allows a script to modify every property in a model. JSON is a simple data-interchange format that is easy to read and write for humans. Sites such as http://jsoneditoronline.org/ offer a convenient way to modify JSON. The script uses a JSON file with all the properties as input. It adjusts the values in the model and saves a modified blend file for each camera. This blend file is ready to be rendered without any further modification. The properties include static parameters such as the eye position and diameter. But also dynamic information for key frames such as the rotation of the eye and position of the glasses.

Automate Process

Even with the script, the process to generate render output is still very time consuming. First the script needs to be applied to base model, which produces the blend files that are ready to render. For each file a render job needs to be started.

A Makefile offers a convenient way to automate the stages further. Additionally, it makes it simple to clean the whole mess up once the generated or rendered files are not needed anymore. With the Makefile the whole process is shorter and easier to remember:

```
#export config file
export ANIMATION_JSON=animation.json
make
make render
```

After a few hours test data is ready to be used for analysis.

3. Pupil Detection

It would be very difficult to improve an algorithm, if there were no test data available. The generated pictures of the last chapter lay the foundation to improve the algorithm to detect the pupil digitally.

This chapter shows how the current algorithm tries to detect a pupil. Improvements are presented that achieve a better result.

3.1. Overview of Existing Algorithm

The general sequence of the algorithm can be described as follows:

1. Find the darkest spot
2. Extract rays starting from the darkest spot from the image
3. Transit the rays with a filter of certain length to detect edges
4. Fit an ellipse on the detected edges with the least square method

The darkest spot is determined by a raster with a fixed distance between points. For each point the average with the surrounding eights point is calculated. The point with the darkest average is the start point for the rest of the algorithm.

Outgoing from the start point, rays are stamped out by the sturstarburst IP. For more information about the starburst IP, visit "Eye Tracking For Sports, Chapter 18.5"[3].

The difference to the next pixel is calculated for each ray that is stamped out by the starburst IP. A filter is applied that adds multiple neighbouring differences together to build a score for each pixel. The pixel with the highest score is a detected edge.

Every edge point is handed over to a weighted least square method to find an ellipse that fits the points well. The residuals of the lest square fitting are used to calculate how well the ellipse fits on the points. [3]

Testing

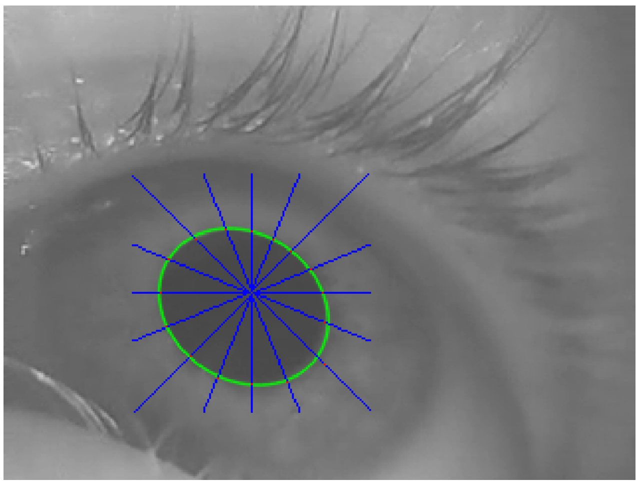
The algorithm was tested with pictures that were inserted at compile time. The output of the algorithm was then compared with the input picture. This is not efficient and is addressed in the next section.

3.2. Improve Testing

It is very time consuming to generate picture data that can be compiled. After that the output of the program still needs to be compared with the input picture. A simpler way would ba a tool that can read a video file and display the result of the algorithm on each picture.

The program "Gazelle View" that was developed during the project study can already open and play video files. Because of the modular design of Gazelle View it is simple to insert the algorithm during the decode phase and paint the rays, the detected edges and the fitted ellipse on the image.

All data that was produced by the algorithm may also hold information about bugs. Such data also helps by analysing problems with the algorithm. So every information that the algorithm generates is displayed beside each frame.


(a) Fitting of Ellipse without outliers

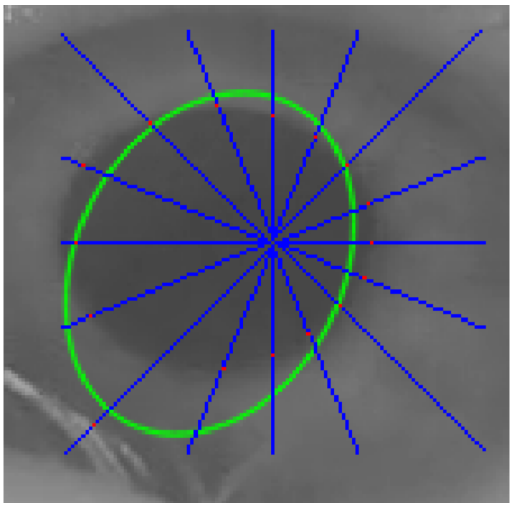

(b) Fitting of Ellipse with one outlier

Figure 3.1.: Comparison between results with and without an outlier. It shows the big impact of an outlier on the least square fitting.

The next step to improve testing would be to automate it and condense the performance of the algorithm to parameters such as the mean and standard deviation of the accuracy.

This should be done on a wide variety of test data. Ideally, with high frame rate pictures that were captured with the eye-tracking system. For each of those pictures, an ellipse has to be fitted manually and stored as a reference. Because this needs to be done for a lot of pictures, the process of generating the reference should to be simplified.

3.3. Outliers

The supplied algorithm does not differentiate between edges that are from the pupil, hair or the eyelid. The algorithm works fine when every edge is a correct one as seen in figure 3.1a. But as a consequence to the least square method, a single false detection severely alters the fitted ellipse. Such a case can be observed in figure 3.1b.

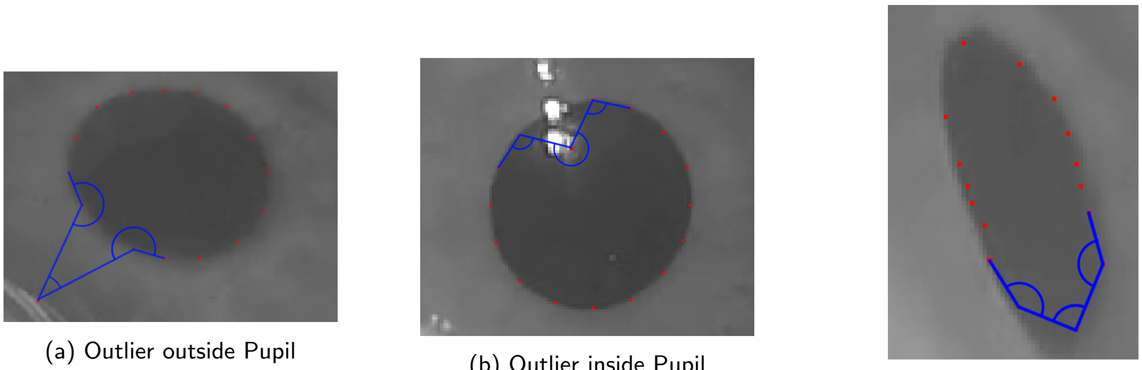
To minimize the influence of outliers there occurrence needs to be reduced. Additionally, outliers should be detected so they can be ignored.

3.4. Angle-Validation

To detect outliers it is necessary to find properties that differ strongly from the other edges. One of those properties gets visible when the angle for each edges gets calculated with the neighbouring edges.

In figure 3.2a an outlier is in the bottom left corner, outside of the pupil. The angle of the outlier is very small compared to the angle of neighbouring edges. Those on the other hand have a larger angle than the other edge points.

When an outlier is inside the pupil then the situation is reversed as visible in figure 3.2b where the outlier has a large angle and the adjacent edges very small angles.


(a) Outlier outside Pupil

(b) Outlier inside Pupil

(c) Accurate Transitions with small Angles

Figure 3.2.: Angles between edges around possible outliers

Multiple adjacent outliers differ in that it is not possible to make a clear statement what the angle will be. But the neighbours to outliers still have the same properties as described in the last paragraphs. For that reason an algorithm that wants to detect outliers should focus on detecting the neighbours of outliers.

In figure 3.2c three adjacent edges have angles that appear too small compared to its neighbours. None of them are outliers.

Overview of Algorithm

The algorithm to detect outliers based on the angles between edges involves four steps:

1. Calculate angles
2. Find 3 adjacent angles with small differences
3. Categorize angles
    - To big
    - To small
    - As expected
4. Determine outliers

Calculate Angles

The angle between the points is calculated with the vectors that stretch from the center point to the other points. For each vector the angle between the x axis and the vector is calculated with the atan2 function. The difference between the resulting two angles is the searched angle.

When the detected edges are close to each other, measurement uncertainty can greatly affect the angles at such points. To mitigate that effect the angle for a point is only calculated with transitions that are distant enough.

Find Adjacent Angles With Small Differences

Context is important to categorize an angle. Only angles that are as expected do not always require context. This is the case when there are three adjacent angles that fall within a narrow range of each other and create context for the neighbouring angles.

Categorize Angles

The found context leads the way to iterate over every edge and categorize the angle within the context. This is done as described in the decision making diagram in 3.3. Angles are classified into three groups. Angles that are as expected, too big or too small.

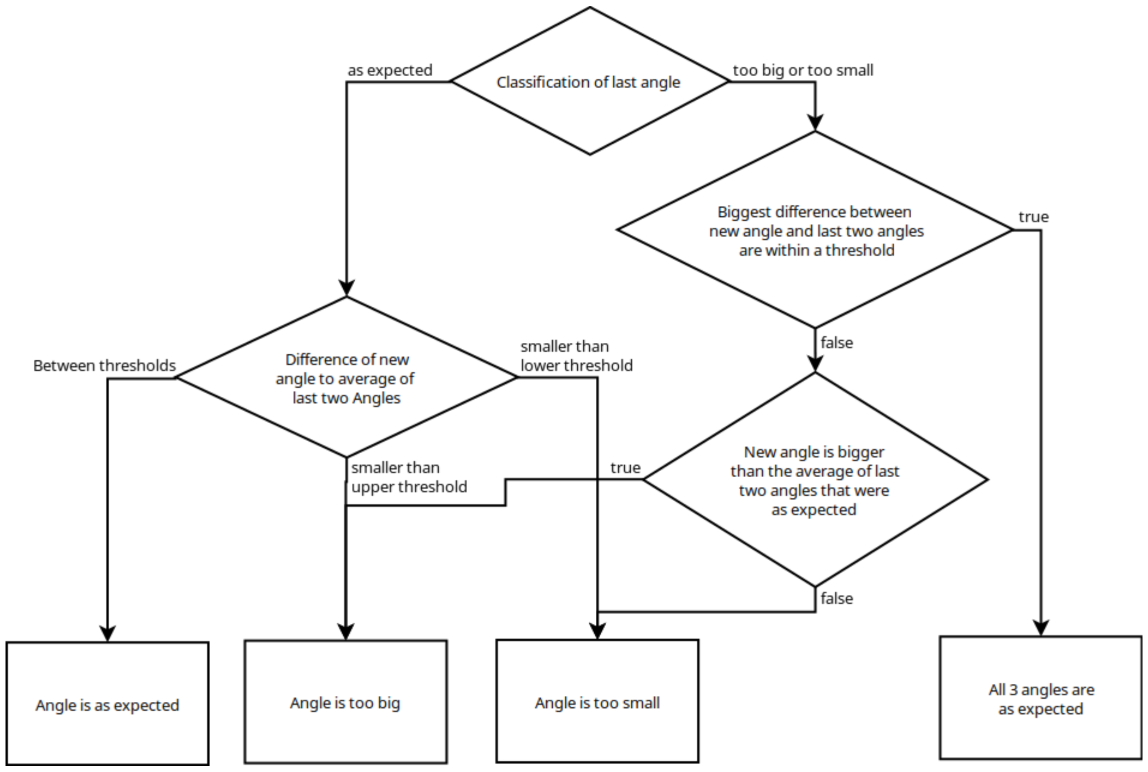

Classification of last angle
as expected
too big or too small
Biggest difference between new angle and last two angles are within a threshold
true
false
Difference of new angle to average of last two Angles
Between thresholds
smaller than lower threshold
smaller than upper threshold
New angle is bigger than the average of last two angles that were as expected
true
false
Angle is as expected
Angle is too big
Angle is too small
All 3 angles are as expected

Figure 3.3.: Decision making diagram for categorizing angles. Note that previous categorization might be overwritten by a later decision

Determine Outliers

As described in section 3.4 outliers are surrounded by angles that are not as expected. The categorized angles make that determination straight forward. An outlier is an edge where neither the angle before or after is categorized as expected. The situation shown in figure 3.2c constitutes an exception to this rule. This exception only occurs with adjacent small angles with neighbours that are as expected.

3.5. Colour of the Startpoint

The pupil has an uniform colour. So edges that transit from the pupil to the iris start with a colour close to the colour of the start point. Edges between the iris and the eyelid start with a colour that is brighter. This correlation can be exploited to improve the number of correctly detected transitions. By dividing the score during the filter with the difference between the colour of the beginning of the edge and the start point. To avoid a division with zero the absolute difference with an offset is used. The offset can also be used to control the influence of this part of the algorithm.

3.6. Border Detection

The colours of the starburst are represented by the values between one and 255. When a point of the starburst is outside the image, then it is set to zero. With that it is simple to detect the border. The behaviour in this case is different depending if there was already an edge or not. If there was already an edge we would like to keep that edge otherwise the result of this ray should be discarded.

To decide if an edge has already occurred a similar method as in section 3.5 is used. But a hard threshold is applied isntead of adjusting the score.

3.7. Exploit good Results

The high frame rate of the eye-capturing cameras results in small differences of the pupil position between frames. This can be used to improve the result because we already have an idea what the ellipse will look like. The start point can be adjusted to be the center of the old ellipse so the detected edges will be more equally distributed.

With the old ellipse it is also possible to calculate where a ray would transit this ellipse. This information is used to weight the score of a transition similar as the colour of the start point in section 3.5 was handled. The score is divided by the absolute distance in pixel to the old ellipse on the ray with a small offset.

3.8. Fixed Threshold

This section is about a method that replaces the ray transition that uses a filter with a threshold.

An edge is detected when the difference to the colour of the start point is greater than a certain threshold.

As shown in chapter 4 this produces better result than the filter. But it is possible that the fixed threshold causes problems under certain light conditions. Further analysis is needed to evaluate that risk. But even if it turns out to be a problem, there are ways to improve the idea. On such idea would be to calculate edges with multiple thresholds and use a Kalman filter to improve the result.

3.9. Special Cases

There are a number of special cases that need to be taken into account. They are likely to get forgotten but can cause major problems later on.

Closed eyelid The algorithm can't produce a valid result when an eyelid is closed. This needs to be detected somewhere, otherwise random results are produced. Without the angle validation this would be pretty straight forward as a closed eyelid would cause a bad fitting, which would cause high residuals. With the angle validation it gets a little more complex. Points that are classified as outliers are ignored. With fewer points it is simpler to get a good fit on random data. To minimize that risk an ellipse is only fitted when at least eight edges were not classified as outliers.

There is also the option to validate a fitted ellipse later by comparing it to other fittings in the same area.

Pupil partly visible It is still possible to fit an ellipse when only a part of the pupil is visible. As described in section 3.6 the border is already detected. If the ray did not transit an edge, then the ray is ignored. The quality of the fitting will generally be lower due to the reduced number of valid edges.

Light reflections	Light reflections inside the pupil will cause an inner outlier. With the angle validation edges that are detected because of a reflection should not cause problems. Everything will fail if the reflection is right at the start point. The colour of the start point is assumed to be dark and similar to the rest of the pupil in sections 3.5 and 3.6. The algorithm with a fixed threshold will fail too when this is the case. A reflection outside the pupil will likely cause an outlier outside the pupil when measurements described in sections 3.5 and 3.7 are not sufficient to prevent it. Generally reflections may cause problems and should be eliminated. Placing the infrared LEDs correctly and using glasses that filter infrared light could elliminate the problem.
Pupil not darkest spot	This case needs to be prevented at all cost. Otherwise, the start point will be completely wrong and a correct fitting is not possible.

3.10. Summary

This chapter gives a overview of the existing algorithm for digitally detecting the pupil. A convenient way to visually validate the result is presented in section 3.2. This enables a better analysis of the problems with the algorithm. The findings were applied to a variety of different methods to improve the result of the algorithm. Additionally, a different approach was proposed to detect edges with a fixed threshold. A number of special cases are discussed in section 3.9.

The validation of the algorithm is done visually. This is a good approach for a first step in improving the algorithm. Different algorithm that produce similar good results can not compared meaningful as there will be a personal bias remaining. For that reason the next chapter compares the performance of the algorithm that uses a fixed threshold and the algorithm that uses a filter to detect edges.

4. Testing

A partner work implemented a method to calculate the gaze direction with two fitted ellipses from different cameras of the same eye. Together with the algorithm described in the last chapter, it is possible to calculate the gaze direction from picture data. Results from both eyes are intelligently combined to improve the accuracy. More information about how this is accomplished can be found in the partner work[1]. The pictures and parameters generated in chapter 2 can now be used to compare the result of the algorithms with the parameters.

Animation

The animation for this test should cover a wide variety of gaze directions. Otherwise, directions that produce strange results could be missed. Additionally, no big jumps should occur as the algorithm for the pupil detection uses the last frame as reference.

A path that fulfils those requirements is shown in figure 4.1. It covers a wide area of 40° in all directions and follows the path with a steady speed.

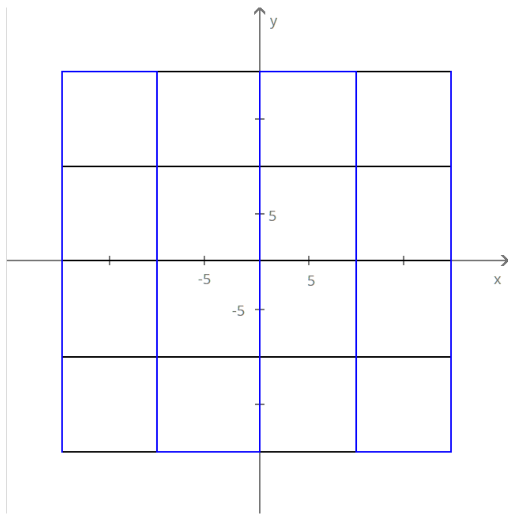
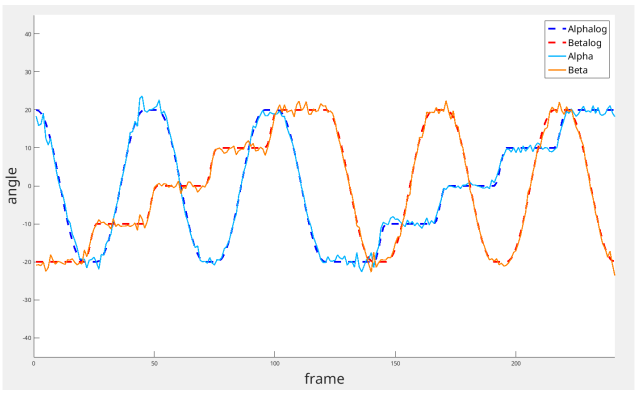


Figure 4.1.: Path that the eyes follow during the animation. It starts at {-20°, -20°} and follows the black in a horizontal zick-zack pattern path until it reaches {20°, 20°}. From there it follows the blue vertical zick-zack pattern path until it reaches the start point again.

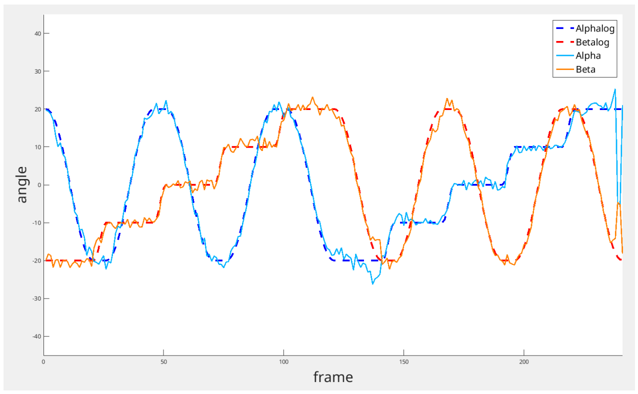
You would expect a higher resolution to produce a more accurate result. The animation was rendered at QVGA(240x320), VGA(480x640) and UVGA(960x1280) resolutions to test that hypothesis.

Results

The fixed threshold algorithm and the filter algorithm were both applied to every resolution. The result for the VGA resolution with fixed and filter algorithm are presented in figure 4.2. It seems that the filter algorithm performs worse than the fixed threshold algorithm. This is confirmed by a look at the standard deviation. The filter algorithm performs much worse at the lower VGA and QVGA resolutions as shown in figure 4.2a. A big contribution to this discrepancy is much higher maximal deviation.

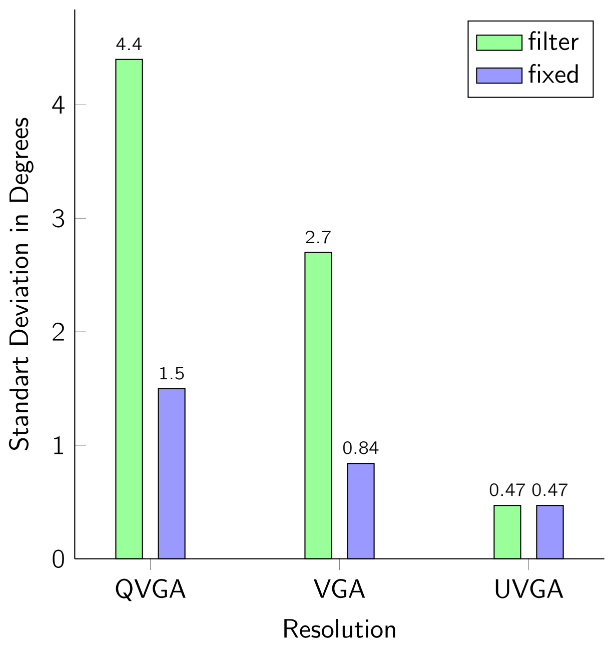


(a) Fixed Algorithm

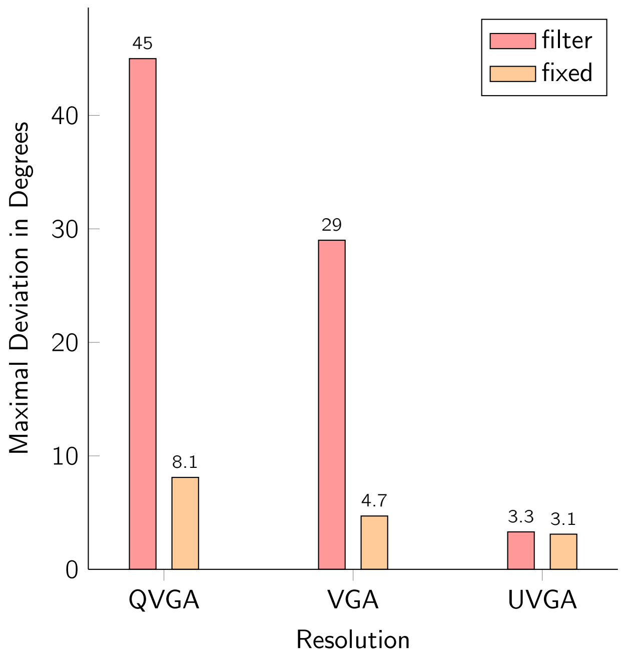


(b) Filtered Algorithm

Figure 4.2.: Comparison between the gaze direction obtained by the log of the animation(dashed) and calculated direction with the algorithms(solid). The results for the fixed threshold in figure 4.2a seem more accurate than the results for the filter algorithm in figure 4.2b.



(a)



(b)

Figure 4.3.: Figure 4.3a shows the standart deviation between the resolutions and different algorithm. The filter algorithm performs similar to the fixed threshold algorithm at the UVGA resolution. But at the lower resolutions it performs much worse. An explanation for that can be found in the maximal deviation in Figure 4.3b. The maximal deviation is much bigger for the filter algorithm.

Discussion

It is quite surprising how much better the fixed threshold algorithm performs at the lower resolutions. It has to be noted that the measurements are not with real data. Once the algorithm is implemented on GazelleCompute, measurements with actual data should be done.

Initially it was planed to configure the cameras with QVGA resolution and a frame rate of 120fps. The results presented in figure 4.3a show a reduction of the standard deviation of 44% for the fixed threshold algorithm. This is an improvement that strongly favors the VGA resolution. For the Gazelle project the higher resolution would come at the cost of a reduced frame rate of 90fps instead of the planned 120.

5. Further Work

The work done in this thesis and the partner work bring the Gazelle eye-tracking project a good step forward. However, there are still some areas that require work to make the eye-tracking system functional.

Implement Algorithm on GazelleCompute

The algorithm for the digital detection of the pupil needs to run on the ARM cores of the Zynq FPGA that is on GazelleCompute. There are still some problems to solve to achieve this. The first is synchronization. There are two cores available but at any time only one can transfer data to the Tegra 3. Furthermore, interrupts when new data is available need to be handled.

Test With Real Data

Tests with data generated from the animation were used for testing. Although this was enough to validate the principle, real data can show a wide variety of problems that don't occur with synthetic data. Such measurements would be possible after implementing the algorithm on GazelleCompute. Different light conditions and different participants should be tested to cover as many cases as possible.

Create Unit Test Framework

Fine tuning thresholds used in the algorithm need a repeatable set of tests that assess the performance of the algorithm. In the best case scenario this would consist of image data from GazelleGlasses. As there are problems with the data transfer to the Tegra, this could be impossible. A solution would maybe consist of reduced resolution and frame rate from only a single camera.

A unit test needs good ellipse fittings as reference. Creating a reference data set needs to be facilitated to allow many ellipses to be fitted by a human.

Improve Fixed Threshold Algorithm

The fixed threshold algorithm is currently implemented in a very crude way. There are still many improvements possible. An example would be an approach with different threshold and a Kalman filter for choosing a more accurate edge.

6. Conclusion

The realistic 3D model of the human eye and GazelleGlasses lays the foundation to test methods that calculate the gaze direction from images of the eyes. It is simple to perform additional test because no knowledge of the 3D modelling software is needed to create new test scenarios. The model is versatile as many different parameters can be configured.

The algorithm to digitally detect the pupil is much more stable and accurate. One of the reasons for that is an ingenious method to eliminate detected edges that don't lie on the border between the pupil and the iris. Those edges would otherwise distort the result and make an accurate eye-tracking impossible.

A test with data from the model shows that a higher resolution improves the accuracy of the eye-tracking.

Declaration of primary authorship

I hereby confirm that I have written this thesis independently and without using other sources and resources than those specified in the bibliography. All text passages which were not written by me are marked as quotations and provided with the exact indication of its origin.

Place, Date:	Biel, 30.07.2016
Last Name, First Name:	Mollet, Timothée
Signature:	....

Glossary

API Application Programming Interface.

ARM Advanced RISC Machine, family of RISC architectures for computer processors.

atan2 arctangent function with two arguments.

CPU Central Processing Unit, an important part of a computer.

FPGA Field-Programmable Gate Array, integrated circuit that can be reconfigured after manufacturing.

IP Intellectual Property.

JSON JavaScript Object Notation, data-interchange format.

LED Light Emitting Diode.

QVGA Quarter Video Graphics Array, resolution of 320x240.

RISC Reduced Instruction Set Computing, a CPU design strategy.

UVGA Ultra Video Graphics Array, resolution of 1280x960.

VGA Video Graphics Array, resolution of 640x480.

Bibliography

[1] L. Bleuer, "Eye-tracking for sports," Berner Fachhochschule, 2016.

[2] B. A. Herbert Gross, Fritz Blechinger, *Handbook of Optical Systems, Survey of Optical Instruments*. Wiley, 2008.

[3] D. W. Reto Pablo Meier, "Eye tracking for sports," Berner Fachhochschule, 2016.

List of Figures

List of Tables

A. Inserts

1. Guide to Generate an Animation
2. Content of CD-ROM

Insert 1

Guide to Generate an Animation

1. Organise a PC, laptop or server that is powerful enough to render and runs linux.
2. Install blender, git, ffmpeg and make on it.
   - On Ubuntu or Debian this is done with
     ```
     apt install blender make git ffmpeg
     ```
   - On Arch Linux this is done with
     ```
     pacman -S blender make git ffmpeg
     ```
3. Clone the git repository
   ```
   git clone https://github.com/timoll/eye-generator
   ```
   In case you want to contribute to the project, fork the project on github and clone your repository. Once you made meaningful changes you can push them to your repository and create a pull request.
4. Change directory into the cloned project
   ```
   cd eye-generator
   ```
5. There are already a few animations stored in the repository. Open a .json file for reference.
   ```
   gedit animation.json
   ```
   You can adjust the values manualy, write a script that generates the animation you want or modify it on `http://jsoneditoronline.org/`

Eye Position Left/Right	Position of the Left/Right Eye in centimeters.	
Diameter	Diameter of the eye in millimetres	
Last Frame	The last frame that will be rendered	
Right/Left Eye Keyframes	Keyframes of the Left/Right eye. Frames in between are interpolated	
	Frame	When the eye has this position
	Rotation	The direction the eye is looking {-90, 0, 0} is forward. Change x for up/down and z for left/right
Glasses Keyframes	Keyframes for the glasses	
	Frame	When the eye has this position
	Position	Position of the Glasses {0 , 0, 0} is a good start
	Rotation	The direction the glasses is pointing {-90, 0, 0} is forward.

6. Save the new json file in the same folder as the rest of the project
7. Export your file so make knows it
   ```
   export ANIMATION_JSON=myanimation.json
   ```
8. Create the different blend files with your animation
   ```
   make
   ```

9. (optional) Verify that your animation is right in blender

   ```
   blender leftup.blend
   ```

   In the bottom left corner select "Timeline" and play the animation. You can zoom out or in by scrolling and move around by pressing the middle mouse button.

10. Start the render, make sure this is done on the server if you have access to one.

    `make render` It will start the render detached so you can continue to use the command line. If you want to abort the render you need to stop blender

    ```
    pkill blender
    ```

    Note: this will kill every instance of blender so make sure you save open blend files first.

11. Wait, this process may take some time, especially if the pc is not that fast.

12. Once all frames are rendered you can inspect them in their folders. The blender output is saved as log in log/renderlu.log or similar for each camera.

    The position and rotation of the glasses and the rotation of eyes for each frame are logged also in the log folder

13. (optional) You might want to generate a movie files from the pictures. For the eye cameras just run the script `./genffmpeg`. It will generate video files in the folder videos.

14. Cleaning up. Once you have saved the generated data that you need. Clean up the folder by running

    ```
    make clean
    ```

    To delete all the blend files that are not needed and

    ```
    make clean_render
    ```

    To delete all files that where generated by the render.

Insert 2

Contents of CD-ROM

```
|-- eye-generator
|-- gazelle_view
|-- gazelle_view_algorithm
|-- matlab
|-- render_results
+-- thesis_latex
```