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 uncertanty 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, psychology, virtual reality and sports. In sport research eye-tracking offers important insights about the hand-eye data or 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. To interfere as little as possible with the activity of an athlete, it should also be portable.

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 of 120 frames per second. 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.

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 are 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 LEDs illuminate the eye for a brighter picture.

Data that are 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.

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 deserialised 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.

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 where the gaze point is known 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 that the information where the participant looks at each moment needs to be recorded alongside of the videostreams. 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.

To optimize the algorithms a simple way of generating new data where the gaze direction and the camera positions are known would simplify the process.

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
- Be scriptable
  - 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 properties to a surface. This includes refracting. Rotation and position of objects can be inserted as keyframes and it interpolates the steps in-between.

Blender is also an open source project that is built with Python and has an 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

  - Framerate
  - 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 modeled correctly. The lens can be simplified by a black surface.

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

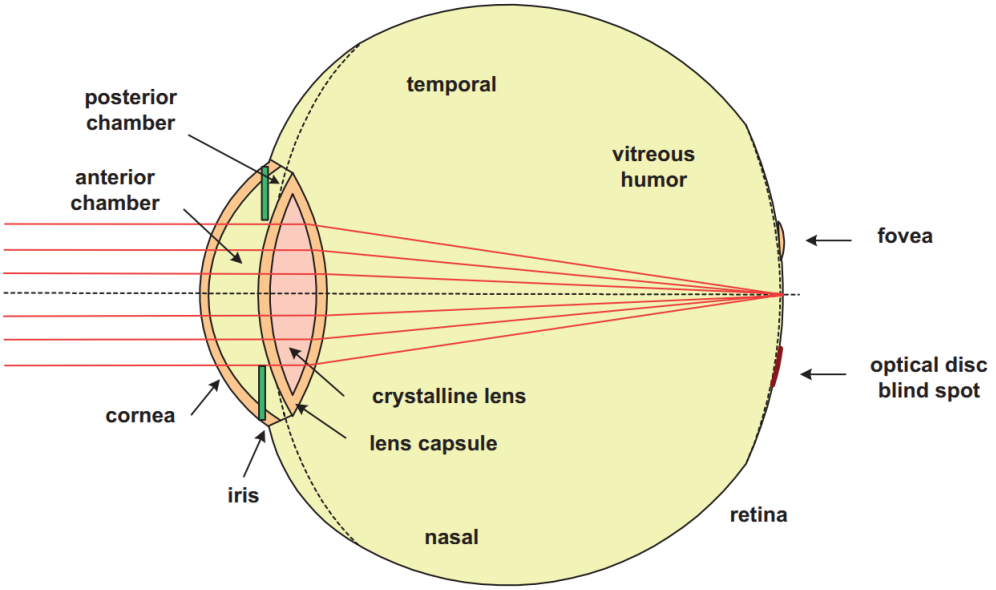


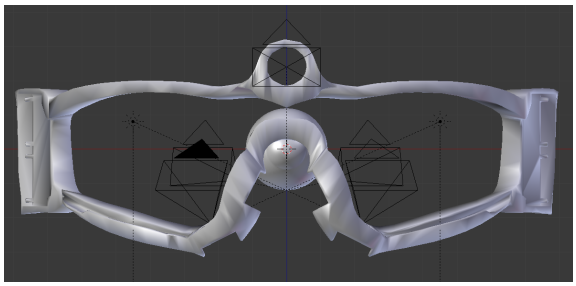
Figure 2.1.: Side view of a human eye model [1]

There are a few models with different values for the refracting indexes, radii and distances but they are close together in most cases. So the simplified Gullstrand eye was chosen for the model.

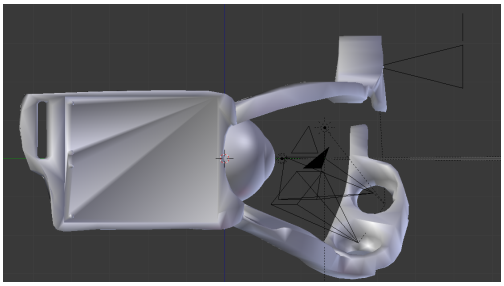
Notaton	Relaxed			Accomodated		
	Radius r [mm]	Thickness d [mm]	Index n	Radius r [mm]	Thickness d [mm]	Index n
cornea	7.70	0.50	1.376	7.70	0.50	1.376
anterior chamber	6.80	3.10	1.336	6.80	2.70	1.336
crystalline lens	10.0	3.60	1.4085	5.33	4.0	1.426
vitreous humor p	-6.00	17.187	1.336	-5.33	13.816	1.336

Table 2.1.: Properties of the eye with the simplified Gullstrand eye. [1]

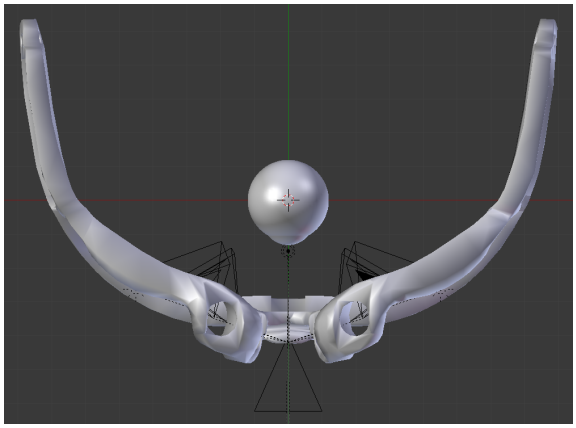
A human eye has on average a diameter of 24mm and the distance between the eyes ranges from 56mm to 72mm with the mean for men at 65mm and 62.6mm for women. [1]



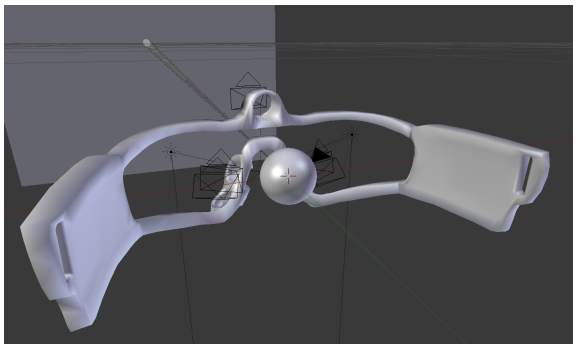
(a) Front view of base model



(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. Those 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 how they would be in the real world. Cutouts for the cameras are used to place them correctly while 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 that is captured by the front facing camera and can be seen in figure 2.2d. As with 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 as they are required for comparing the final gaze direction results. To allow a wide variaty of animations to be created a simple way of configuring the script is needed.

Automate Animation

This script is written in Python because Blender offers a powerfull Phython API that allows such 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 and adjusts the values in the model and saves a modified blend file for each camera that is ready to be rendered. The properties includes static parameters such as the eye position and diameter and the number of frames that should be rendered. But also dynamic information for keyframes such as the rotation of the eye and position and rotation of the glasses.

Automate Process

With the script the process to generate render output is as follows:

```
#Use script to generate render ready files
blender base.blend -b -P generate.py -- animation.json
#render the animation for each camera
blender -b leftDown.blend -E CYCLES -t 4 -a
blender -b leftUp.blend -E CYCLES -t 4 -a
blender -b rightDown.blend -E CYCLES -t 4 -a
blender -b rightUp.blend -E CYCLES -t 4 -a
blender -b scene.blend -E CYCLES -t 4 -a
```

This is still very time consuming and can be automated further.

A Makefile offers a convenient way to automate the stages further and make 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
```

An added benefit is also that make will start the render jobs detached and distribute the available cpu cores equally among the render jobs.

3. Pupil Detection

Without test data it is very hard to improve an algorithm. The generated pictures of the last chapter lay the groundwork 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 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 startpoint for the rest of the algorithm.

Outgoing from the startpoint rays are stamped out by the starburst IP that is described in "Eye Tracking for Sports, Chapter 18.5"

The difference to the next pixel is calculated for each ray that is stamped out by the starburst IP. Because the edge may not lie on a single pixel, surrounding differences are added together to build a score for each pixel. This score is positive for transitions from a darker pixel to a brighter. The pixel with the highest score is determined as detected edge. So only edges that transit from dark to bright can be detected.

Every edgepoint is handed over to a weighted least square method to find a ellipse that fits all the points well. The residuals of the lest square fitting are used to calculate how well the ellipse fits on the points.

Testing

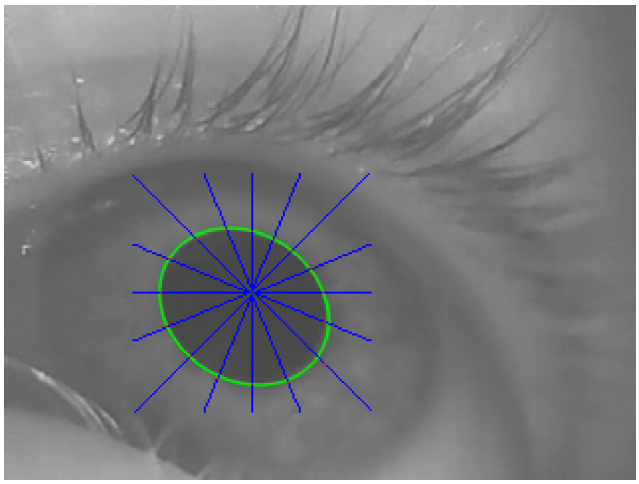
The algorithm was tested with pictures that were inserted at compile time with the parameters of the ellipse as output. The result was then compared with the result. This is not efficient and is addressed in the next section.

3.2. Improve Testing

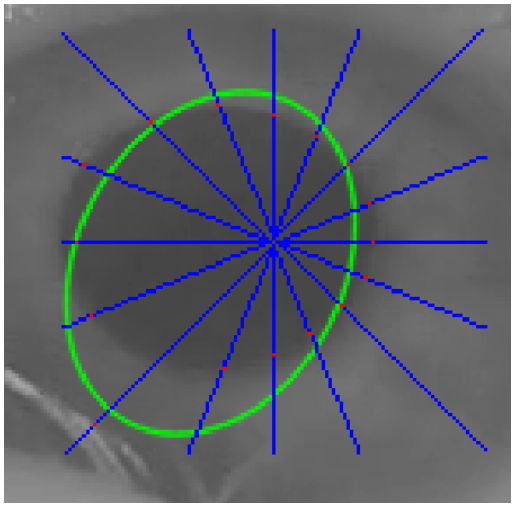
It is very time consuming to generate picture data that can be compiled and to compare the output of the program with what is expected. A better way is when a tool can read a video file and display the result of the algorithm visually on each picture.

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 important information to find bugs or problems with the algorithm. In order to display that data besides the frame all data is packed into a struct and displayed in a Treeview beside the 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 ideally condense the performance of the algorithm to parameters such as the mean and variance of the accuracy.

This should be done on a wide variety of test data, at best 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 golden reference. As this needs to be done for a lot of pictures the process of generating the reference needs to be simplified.

3.3. Outliers

The supplied algorithm does not differentiate between edges that are from the pupil or other edges from hair or the eyelid. The algorithm works fine when every edge is a right 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 as it happens in figure 3.1b.

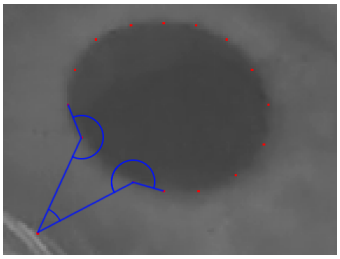
To mitigate the influence of outlier the occurrence needs to be reduced accompanied by the detection of outliers that are still detected.

3.4. Angle-Validation

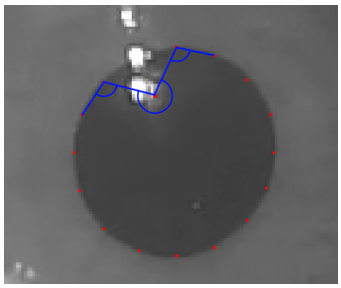
To detect outliers it is necessary to find properties that differ strongly from the other transitions. One of those properties gets visible when the angle for each transition gets calculated with the neighbouring transitions.

In figure 3.2a a outlier lies on the bottom left corner of the image and outside of the pupil. The angle of the outlier is very small compared to angle of the neighbouring transitions. Those on the other hand have a greater angle than the other transition points.

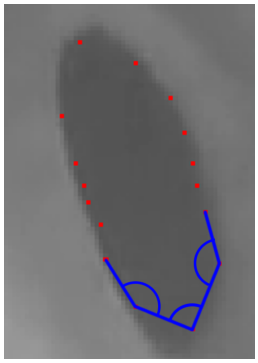
When a outlier is inside the pupil then the situation is reversed as visible in figure 3.2b where the outlier has a big angle and the adjacent transitions very small angles.



(a) Outlier outside Pupil



(b) Outlier inside Pupil



(c) Accurate Transitions with small Angles

Figure 3.2.: Angles between Transitions around 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 the 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 transitions 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 transitions 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 advantage of the atan2 over the conventional arctangent function is that it takes coordinates and returns the angle for the correct quadrant. The difference of the resulting two angles is the desired angle between the points.

When the detected transitions 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 transition 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.

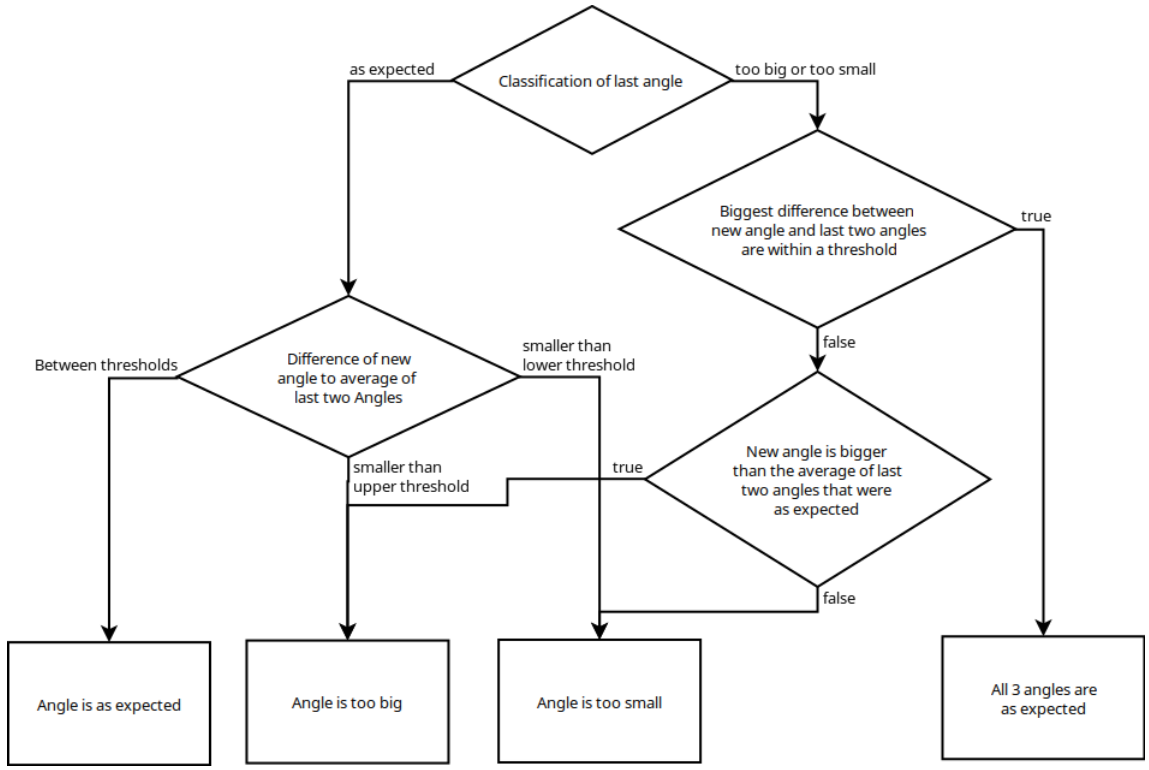


Figure 3.3.: Decision making diagram for categorizing angles

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 a transition where neither the angle before or after is categorized as expected. The situation described in figure 3.2c where accurate transitions build three adjacent angles that are smaller than expected constitute 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 transitions that go from the pupil to the iris start with a colour close to the colour of the startpoint. Transitions from the iris to an eyelid on the other hand 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 by the difference to the colour of the startpoint. To avoid a division with zero the absolute difference with an offset is used. Th offset can also be used to control the influence of this part of the algorithm.

3.6. Border Detection

The colours of te starburst are represented by the values between one and 255. When a point of the starburst is outside the image it is set to zero. With that it is simple to detect the border during the filter. The behaviour in

this case is different depending if there was already a 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 instead of adjusting the score of a transitions a hard threshold is applied.

3.7. Exploit good Results

The high framerate 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 startpoint can be adjusted to be the center ofthe 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 transits this ellipse. This information is used to weight the score of a transition similar as the colour of the startpoint 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 startpoint is greater than a certain threshold.

As shown in chapter 4 this produces better result than the filter. Although the fixed threshold might cause 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. Somewhere it needs to be detected when this is the case 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 less 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 be generally 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 the edges that are detected that way should be ignored.

In case the reflection is right at the startpoint everything will fail as the colour of the startpoint 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 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 major problems and should be eliminated. By placing the infrared LEDs correctly and using glasses that filter infrared light.

Pupil not darkest spot This case needs to be prevented at all cost. Otherwise the startpoint 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. Additionaly 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 ellipse from different cameras of the same eye. Together with the algorithm described in the last chapter they enable the gaze direction from picture data. The pictures and parameters generated in chapter 2 can now be used to compare the result of the algorithms with the parameters. 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.

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 **??**. It covers a wide area of 40° in all directions and follows the path with a steady speed.

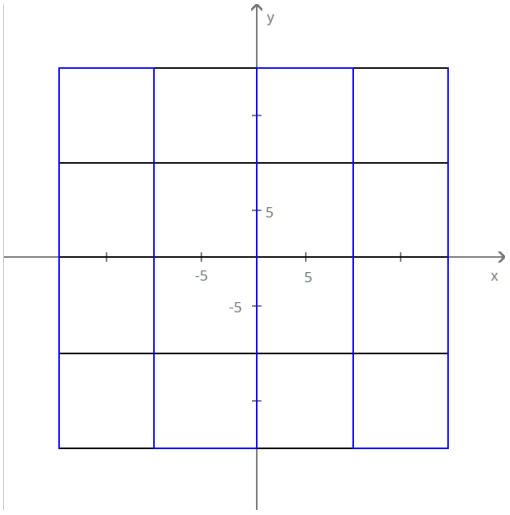
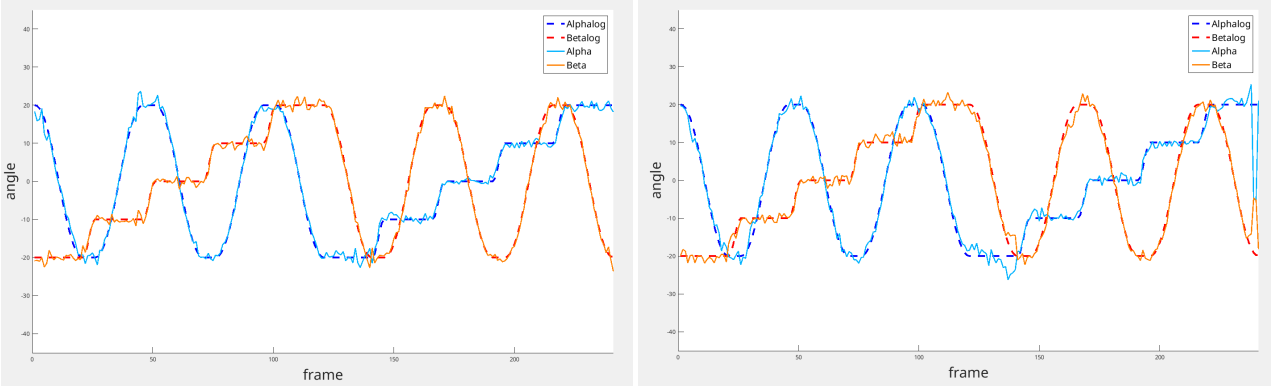


Figure 4.1.: Path that the eyes follow during the animation. It starts at {-20°,-20°} and follows the black in a horizontal zick-zack pattern path until it reaches {20°,20°}. From there it follows the blue vertical zick-zack pattern path until it reaches the start point again.

You would expect a higher resolution to produce a more accurate result. The animation was rendered at QVGA, VGA and UVGA resolutions to test that hypothesis.

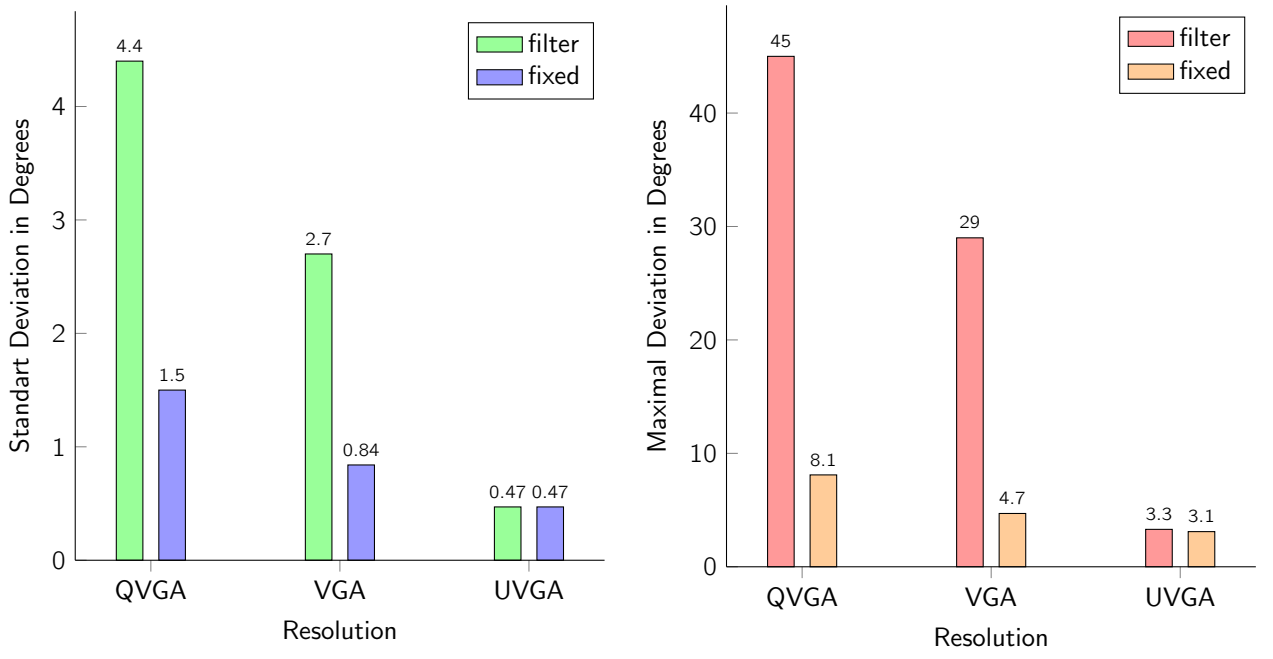
Results

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 filtered 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 it is shown in figure 4.2a. A big contribution to this discrepancy is the much worse 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 filtered algorithm in figure 4.2b.



(a) Fixed Algorithm (b) Filtered Algorithm

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 filtered algorithm.

Discussion

It is quite surprising how much better the fixed threshold algorithm performs at the resolutions that are more relevant for the Gazelle eye-tracking system. It has to be noted that the measurements are not with real data. Once the algorithm is implemented on GazelleCompute measurements with actual data can be done.

Initially it was planed to configure the cameras with QVGA resolution and a framerate of 120fps. The results presented in figure 4.3a show a reduction of the standard deviation of 44% for the fiexed threshold algorithm. This is an improvement that For the Gazelle project the higher resolution would come at the cost with a reduction of the framerate from 120fps to 90fps.

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 synchronisation. 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 valide the principle. Real data can show a wide variety of problems that don't occur with synthetic data. After the algorithm is implemented on GazelleCompute such measurements would be possible. 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 needs a repeatable set of tests that assess the performance of the algorithm. In a 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 framerate and 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 quite some improvements possible. Such as 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 as 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 / We hereby confirm that I / we 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/Burgdorf], 30.07.2016	
Last Name/s, First Name/s:	[Test Peter]	[Müster Rösä]
Signature/s:	.....	.....

Glossary

BibTeX Program for the creation of bibliographical references and directories in TEXor LATEXdocuments.

Index Index with keywords from text.

Bibliography

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

List of Figures

List of Tables

APPENDICES

A. 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.

B. Additional Appendix

B.1. Test 1

To an English person, it will seem like simplified English, as a skeptical Cambridge friend of mine told me what Occidental is. The European languages are members of the same family. Their separate existence is a myth. For science, music, sport, etc, Europe uses the same vocabulary. The languages only differ in their grammar, their pronunciation and their most common words. Everyone realizes why a new common language would be desirable: one could refuse to pay expensive translators. To achieve this, it would be necessary to have uniform grammar, pronunciation and more common words. If several languages coalesce, the grammar of the resulting language is more simple and regular than that of the individual languages. The new common language will be more simple and regular than the existing European languages.

B.1.1. Environment

It will be as simple as Occidental; in fact, it will be Occidental. To an English person, it will seem like simplified English, as a skeptical Cambridge friend of mine told me what Occidental is. The European languages are members of the same family. Their separate existence is a myth. For science, music, sport, etc, Europe uses the same vocabulary. The languages only differ in their grammar, their pronunciation and their most common words. Everyone realizes why a new common language would be desirable: one could refuse to pay expensive translators. To achieve this, it would be necessary to have uniform grammar, pronunciation and more common words.

C. Content of CD-ROM

Content of the enclosed CD-ROM, directory tree, etc.