Experimental Manual for PR x RLT

Table of Contents

```
pyinstaller --clean main.spec
```

- build: directory where the exectuable files are built.
- hooks: directory for helpers to run pyinstaller.
- logs: Logs of the pipeline.
- thesis: directory where the Master thesis done on this code has been written. THERE ARE ALSO ALL THE DOCS TO READ FOR MORE INFORMATION ON THE PIPELINE AND ON THE DIFFERENT PROJECTS.
- htmlcov: directory where the test coverage is outputed.
- docs: documentation generated automatically with Sphinx. To update the documentation, go to the *docs* directory and write the following (be sure to pay attention to the errors and correct them):

```
make clean html
make html
```

To read the documentation, go to *docs/_build/html/ and double-click on **index.html**. Then you will be able to go through the documentation and find function informations.

Files

- .coverage and .coveragerc are used to create the htmlcov folder with pytest coverage of the pipeline
- environment.yml is used to create a new conda environment with:

```
conda env export > environment.yml
conda env create -f environment.yml
```

- main.spec and main2.spec are used to create pyinstaller executable (from Input_GUI and Input_GUI2)

Notes

htmlcov is created with commands:

```
coverage run --source=src -m pytest tests/
coverage report
coverage html
```

General tips at HOJG:

- Install with pip : pip install pandas --trusted-host pypi.org --trusted-host files.pythonhosted.org --user
- If git push fails with GitLab: GIT_SSH_COMMAND="ssh -i C:/Users/BardetJ/.ssh/id_ed25519" git push -vvv
- All data and results are saved in "V:\Studies\AOSLO_AOSLO_2025" in the different folders.

Background:

This project investigates the relationship between cone photoreceptor density and retinal layer thicknesses using a combination of high-resolution imaging techniques. We address the limitations of conventional foveal imaging by implementing a bilateral theoretical model that corrects systematic density estimation errors in Adaptive Optics Scanning Laser Ophthalmoscopy (AOSLO) data. Through spatial registration with Optical Coherence Tomography (OCT) measurements, we conduct correlation analyses that reveal significant eccentricity-dependent relationships between cone density and specific retinal layers.

We explain here the different steps of the pipeline, from the montaging of AOSLO images to the extraction of the photoreceptor densities and comparison with retinal layer thicknesses for healthy subjects.

How to run the pipeline

Generating the montage

1. Run via the *config.txt*:Write in the config file that can be found in *aoslo_pipeline/src/PostProc_Pipe/Configs/config.txt* the specific parameters that you want to run, especially everything under the [Montaging] section (*__do_montaging* must be set to **True** to run this step of the pipeline). Then go to the specific folder: `cd aoslo_pipeline/src/PostProc_Pipe` and run `python Start_PostProc_Pipe.py`.

More details on the parameters of this part of the pipeline can be found in the section Montaging: or in the dedicated README here

Extracting Layer Thicknesses

Simply go through the guide Notebook located in src/save_layer_features.ipynb NOTE that this notebook requires some familiarity with the CohortBuilder presented here or otherwise (with some modifications) the extraction of layer segmentations using RetinAI If usage of such tools is not possible, to skip this step the data should be available in the following formats

1. layer_thickness.json, one for each subject

```
{
  "UniqueID": {
    "SliceN": {
      "Layer Name(s)": {
        "vector": [Thickness_Values]
      }
    }
  }
}
```

2. layer_os.json, one for each subject

```
{
  "UniqueID": {
    "SliceNumber": {

        "vector": [Thickness_Values]

    }
  }
}
```

3. layer_cvi.json, one for each subject

```
{
  "UniqueID": {
    "SliceNumber": {

        "vector": [Thickness_Values]

    }
  }
}
```

Running the Pipeline

The `DensityAnalysisPipelineManager` class is responsible for steps 3, 4, 6, and 7 described above. Part of step 2 may be handled, but it requires the first two bullet points of step 2 (up to `locations.csv`) to be completed beforehand. Part 5, as already specified, is handled by the `save_layer_features.ipynb` notebook. Step 8 is handled by the `DensityStatistics` class.

An instance of `DensityAnalysisPipelineManager` is created for each subject, using a `ProcessingPathManager` and a `MontageMosaic` instance. The latter has already been described in step 2 above; the former is a class that manages the paths of a given subject; it contains the main path (e.g. `P:\...\_Results\SubjectXXX\SessionYYY\`) as well as informations about the subject as well as the paths to the different subdirectories used in the pipeline (e.g. `...\density_analysis_new\`, `...\layer_new\`, etc.).

The (density analysis) pipeline is then run by calling the `run` method of the `DensityAnalysisPipelineManager` instance. The arguments of this method are the following:

- `from_csv`: if `True`, the pipeline will load `...\densitiy_analysis_new\densities.csv` for the given subject. This requires that the densities have been fitted and saved in this file beforehand. If `False`, the pipeline will explicitely call the `get_densities_by_yellott` method; at this point, the following options are available:
  - `from_csv_raw`: if `True`, loads raw densities from `...\density_analysis_new\densities_raw_x.csv` and `...\density_analysis_new\densities_raw_y.csv`, and then performs the fitting step. If `False`, the densities are computed from scratch: ***this is the only case where a valid `MontageMosaic` instance is required*** (since the images & montage are needed to compute the densities).
  - `to_csv_dens`: if `True`, saves the raw & fitted densities in the corresponding `csv` files (as described above).
- `do_layer`: if `True`, steps 6 and 7 of above are executed: a `LayerCollection` manages the loading and processing of the layer data, a `LayerThicknessCalculator` computes the thicknesses, and a `DensityLayerAnalysis` compares the cone densities with the layer thicknesses.

The logic of when to load or not the corresponding `MontageMosaic` instance (either by building it from scratch or by loading it from the `pkl` in `...\montaged_corrected\`) is handled by `main` function in `src/cell/analysis/density_analysis_pipeline_manager.py`, which effectively runs the pipeline for a given subject.

In the `if __name__ == "__main__":` block of the same file, choose to run the pipeline on a single subject or on the whole cohort (iterating over the subject/session look-up table `processed.txt`); the `main` funcion is then called for each subject.

Simply run

```
python src/cell/analysis/density_analysis_pipeline_manager.py
```

to start the pipeline.

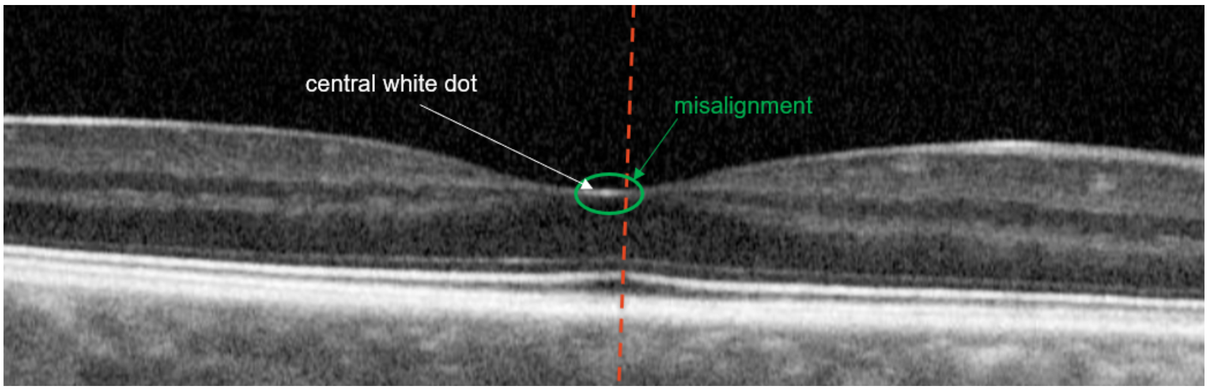
For the `DensityStatistics` class, nothing special; when all subject of the cohort have been processed, simply run

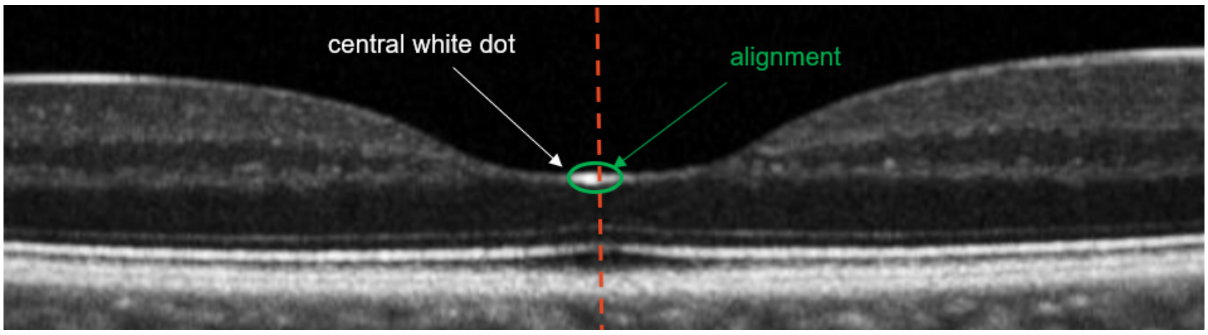
```
python src/cell/analysis/density_statistics.py
```

Data analysis

The results of the experimental data analysis are available by running the `PRxRLT_analysis.ipynb` notebook present in this folder.This notebook will extract the data computed from the

Important notes:

- Out of the 45 subjects of the healthy database, *only 33 are kept for the analysis* (see the notebook for details). The 12 subjects that are filtered out have misaligned OCT acquisition: on the central B-scan(s) -- i.e. the ones that contain a bright white dot at the bottom of the foveal pit, the peak of the OS layer (the tip of the little black triangle in the PR+RPE layer) is not vertically aligned with the said white dot. This is often because the OCT is *tilted*.
  - Here is an example of a misalignment (subject 25, dataset 48b2fd0e, frame 50/97: take a look on Discovery). We clearly see that the white dot is not vertically aligned with the peak of the OS layer (along the dashed red line, which is perpendicular to the retinal floor).


  - Here is an example of a nicely aligned OCT now, to compare (subject 15, dataset 5cd93c49, frame 49/97: Discovery)


Structure of the repository

Most of the following has already been covered in the steps description above, but here is a more detailed overview of the repository structure.

```
📦src
 ┣ 📁AST                                                    acquisition support tool
(unrelated)
 ┣ 📁cell
 ┃ ┣ 📁analysis
 ┃ ┃ ┣ 📜baseline_characteristics.py                        comparison of subject by age,
sex, etc.
 ┃ ┃ ┣ 📜cone_density_calculator.py                         cone density extraction
 ┃ ┃ ┣ 📜density.py                                         data structure to hold
densities
 ┃ ┃ ┣ 📜density_analysis_pipeline_manager.py               runs the whole density
pipeline, as described above
 ┃ ┃ ┣ 📜density_layer_analysis.py                          comparison between densities
& layer thicknesses
 ┃ ┃ ┣ 📜density_statistics.py                              global statitistics & plots,
saved to all_stats_new
 ┃ ┃ ┣ 📜helpers.py                                         defines eccentricity,
gather_results, etc.
 ┃ ┃ ┣ 📜normal_comparison_handler.py                       compares a subject to average
 ┃ ┃ ┣ 📜pvalue_processor.py                                helper for spearman's stats
 ┃ ┃ ┣ 📜result_writer.py                                   helper for
density_layer_analysis
 ┃ ┃ ┗ 📜__init__.py
 ┃ ┣ 📁layer
 ┃ ┃ ┣ 📜foveal_shape_extraction.py                         script to extract foveal
shape
 ┃ ┃ ┣ 📜helpers.py
 ┃ ┃ ┣ 📜layer.py                                           data structure to hold
thickness data
 ┃ ┃ ┣ 📜layer_collection.py                                loading and processing of
retinal layer data
 ┃ ┃ ┣ 📜layer_thickness_calculator.py                      extraction of thicknesses per
eccentricity
 ┃ ┃ ┣ 📜os_layer_extraction.py                             stript to segment and extract
OS layer
 ┃ ┃ ┗ 📜__init__.py
 ┃ ┣ 📁montage
 ┃ ┃ ┣ 📁AOAutomontagingPython
 ┃ ┃ ┣ 📁AOAutomontaging_master                             <- MATLAB scripts, as
described above
 ┃ ┃ ┣ 📁AST_v2
 ┃ ┃ ┣ 📜constants.py
 ┃ ┃ ┣ 📜matlab_reader.py
 ┃ ┃ ┣ 📜montage_element.py                                 holds an image and its
location on the montage
 ┃ ┃ ┣ 📜montage_mosaic.py                                  holds the elements of the
montage
 ┃ ┃ ┣ 📜montage_mosaic_builder.py                          effectively loads the images
and build the montage
 ┃ ┃ ┣ 📜montage_pipeline_manager.py                        runs the whole montaging
pipeline
 ┃ ┃ ┣ 📜README.md
 ┃ ┃ ┣ 📜run_auto_ao_montaging.py
```

```
  ┃ ┃ ┣ 📜ssim.py
  ┃ ┃ ┣ 📜ssim_processing.py
  ┃ ┃ ┗ 📜__init__.py
  ┃ ┃
  ┃ ┣ 📜affine_transform.py                         notably used for montaging to
represent loc of imgs
  ┃ ┣ 📜dark_region_finder.py                       used by cell detection
algorithms (unrelated)
  ┃ ┣ 📜processing_path_manager.py                  holds infos & paths to
subdirs of a subject
  ┃ ┗ 📜__init__.py
  ┣ 📂configs
  ┃ ┣ 📜autoIT_invoke_PowerAutomate.au3
  ┃ ┣ 📜autoIT_invoke_PowerAutomate_Matlab.au3
  ┃ ┣ 📜autoIT_invoke_PowerAutomate_signgo.au3
  ┃ ┣ 📜config.txt                                  config file for the whole
pipeline
  ┃ ┣ 📜parser.py                                   class eading & holding config
infos
  ┃ ┣ 📜read_config.py                              helpers for parser
  ┃ ┗ 📜__init__.py
  ┣ 📂Helpers                                       .outdated
  ┣ 📂InputData_Pipe                                .
  ┣ 📂input_GUI
  ┣ 📂input_pipeline
  ┣ 📂Logs
  ┣ 📂plotter                                       .plotters for each
corresponding module
  ┃ ┣ 📜baseline_characteristics_plotter.py         .
  ┃ ┣ 📜cell_detection_plotter.py                   .
  ┃ ┣ 📜cone_gatherer_plotter.py                    .
  ┃ ┣ 📜cone_mosaic_plotter.py                      .
  ┃ ┣ 📜density_layer_plotter.py                    .
  ┃ ┣ 📜density_plotter.py                          .
  ┃ ┣ 📜density_statistics_plotter.py               .
  ┃ ┣ 📜layer_plotter.py                            .
  ┃ ┣ 📜plotter.py                                  base class for all plotters
  ┃ ┗ 📜__init__.py
  ┣ 📂readme_images
  ┣ 📂shared
  ┃ ┣ 📂computer_vision
  ┃ ┃ ┣ 📜image.py
  ┃ ┃ ┣ 📜point.py
  ┃ ┃ ┣ 📜square.py
  ┃ ┃ ┣ 📜voronoi.py
  ┃ ┃ ┗ 📜__init__.y
  ┃ ┣ 📂datafile
  ┃ ┃ ┣ 📜coordinates_file.py                       stores e.g. locations of
detected cones on an img
  ┃ ┃ ┣ 📜dark_region_strategy.py                   different ways to segment out
vessels (dark areas)
  ┃ ┃ ┣ 📜datafile.py                               base class
  ┃ ┃ ┣ 📜datafile_constants.py
  ┃ ┃ ┣ 📜helpers.py
  ┃ ┃ ┣ 📃image_file.py                                 stores an image with specific
naming conventions
  ┃ ┃ ┣ 📃validators.py
  ┃ ┃ ┗ 📃__init__.py
  ┃ ┣ 🗀helpers
  ┃ ┃ ┣ 📃direction.py                                  class to improve use of
directions (X,Y)
  ┃ ┃ ┣ 📃exceptions.py
  ┃ ┃ ┣ 📃global_constants.py
  ┃ ┃ ┣ 📃intervals.py
  ┃ ┃ ┣ 📃metrics.py
  ┃ ┃ ┣ 📃os_helpers.py
  ┃ ┃ ┣ 📃strings.py
  ┃ ┃ ┣ 📃validators.py
  ┃ ┃ ┗ 📃__init__.py
  ┃ ┣ 🗀numpy
  ┃ ┣ 🗀plotting
  ┃ ┗ 📃__init__.py
  ┣ 📃classes.png
  ┣ 📃environment.yml
  ┣ 📃processed.txt
  ┣ 📃save_layer_features.ipynb                         detailed guide to download &
process layer features
  ┣ 📃save_layer_thicknesses.py                         outdated version of
save_layer_features.ipynb
  ┗ 📃__init__.py
```

Structure of the data

Most of the following has already been covered in the steps description above, but here is a more detailed overview of the subdirectories of `P:\AOSLO\_automation\_PROCESSED\Photoreceptors\Healthy\_Results\SubjectXXX\SessionYYY\`, for each subject/session.

```
📦SubjectXXX/SessionYYY
 ┣ 🗀montaged                                           MATLAB's outputs & manual
montage locations
 ┃ ┣ 📃ref_XX_combined_MODALITY.tif                        connected components from
MATLAB for each modality
 ┃ ┣ 📃all_ref_combined_MODALITY.tif                       MATLAB's attempt for
whole montage for each modality
 ┃ ┣ 📃IMG_aligned_to_refXX_MODALITY.tif                   loc of IMG with respect
to whole montage
 ┃ ┣ 📃AOMontageSave.mat                                   output of MATLAB's script
 ┃ ┗ 📃locations.csv                                       output of manual montage
correction
 ┣ 🗀montaged_corrected                                 final montage output,
overlaid on fundus & mosaic.pkl
 ┃ ┣ 📃mosaic.tif                                          final corrected montage
 ┃ ┣ 📃mosaic_fundus.tif                                   same, overlaid on eye
fundus
```

```
 ┃ ┣ 📃mosaic_flipped.tif                               flipped version to match
correct orientation
 ┃ ┣ 📃mosaic_fundus_flipped.tif                        same, overlaid on eye
fundus
 ┃ ┗ 📃mosaic.pkl                                       pickle of the
MontageMosaic instance holding the montage
 ┣ 📁layer_new                                      OCT-derived data
 ┃ ┣ 📃layer_thickness.json                             thicknesses for RNFL,
GCL+IPL, INL+OPL, ONL, PR+RPE
 ┃ ┣ 📃layer_cvis.json                                  same, but for Choroidal
Vascularity Index (from CB)
 ┃ ┣ 📃layer_os.json                                    same, but for Outer
Segment of PhotoRecept (homemade)
 ┃ ┣ 📃fovea_3d.csv                                     X,Y,Z coordinates of the
vitreo-retinal interface of OCT cube
 ┃ ┣ 📃fovea_3d_fitted_params.csv                       foveal shape parameters
fitted from OCT cube
 ┃ ┣ 📃LAYER.png                                        3d plot of the layers
 ┃ ┗ 📃triangle_LAYER.png                               same, including triangle
showing kept regions
 ┣ 📁density_analysis_new                           plots & csv results of
density analyis (main focus)
 ┃ ┣ 📃AXIS_LAYER.png                                   plots for each axis of
cone density vs layer thickness
 ┃ ┣ 📃layers_compared_to_density_AXIS.png              plots for each axis of
cone density vs all layers
 ┃ ┣ 📃cone_density_curve_smoothed.png                  plot of X & Y smoothed
cone density
 ┃ ┣ 📃cone_density_curve_fitted_new_AXIS.png           plot for each axis of raw
& fitted cone density
 ┃ ┣ 📃density_raw_AXIS.csv                             raw densities extracted
for each axis
 ┃ ┣ 📃density.csv                                      raw, smoothed & fitted
densities for X & Y
 ┃ ┣ 📃results.csv                                      all cone densities &
thicknesses for X & Y
 ┃ ┗ 📃spearman_AXIS.txt                                spearman's coef & p-
values for density vs thicknesses
 ┣ 📁compare_to_normal_new                          density & layers compared to
normal (CI)
 ┃ ┗ 📃FEATURE_AXIS_compared_to_normal.png              plots comparing each
feature (density/thickness) to normal
 ┗ 📃SubjectXXX_SessionYYY_EYE_LOC_DIM_ID_MOD.tif   registered images extracted
from the raw AOSLO videos
```

In Depth Explanation

Montaging:

- The images are montaged to reconstruct retina's mosaic. This step is crucial as it introduces a coordinate system centered at the fovea (very center of the retina, where the cone density peaks),

which allows to precisely locate what is observed in the initially independent images, and also to compare with other imaging modalities (e.g. OCT).
- The acquired images are 1.5x1.5 degrees (1° = 0.291mm = 480 pixels), and should overlap a little; this allows for the montaging algorithm to find the best overlap between images and reconstruct the mosaic. This is managed by a MatLab script, which does its best to construct connected components of images. The result is often not complete/correct though; a manual step is necessary to correct the montaging, and is done using the ***montaging GUI***, which allows to correctly position the images over the eye fundus. More details about this step are available here. The location of each component of the mosaic is stored in `P:\...\_Results\SubjectXXX\SessionYYY\montaged\locations.csv`.
- Based on the connected components and their locations, the class `CorrectedMontageMosaicBuilder` is used to build the `MontageMosaic`, which holds for instance the location of the fovea, as well as `MontageElement`s which themselves hold the images and their locations in the coordinate system.
- The montaging step takes a while to execute, so once the `MontageMosaic` instances are built once, they are `pickle`d and stored in `P:\...\_Results\SubjectXXX\SessionYYY\montaged_corrected\mosaic.pkl`.
- Once ran this step you should, in the same directory (`...\montaged_corrected\`), also find `mosaic.tif` which shows the reconstructed mosaic and `mosaic_fundus.tif` which shows the eye fundus (if provided) with the mosaic overlaid, as well as the mirrored versions (`*_flipped.tif`) -- this is because the AOSLO device acquires images in a mirrored way, so the montaged mosaic has to be mirrored back to be correctly oriented (convince yourself by looking at the fundus, which is correclty oriented, specifically the optic nerve head which should be on the right side of the en-face fundus for a right eye).

Cone density extraction:

- Each confocal image of the montage is then processed to extract the photoreceptor densities. For this work, this task does not require a precise count of each individual cone (this necessitates cell detection algorithms, a much more strenuous task -- see here for further details); instead, the cone density of a small region of interest (ROI) is estimated by Yellott's ring method, based on the Fourier tranform of the ROI (see Cooper 2019 and Cooper 2024). This algorithm is implemented by the class `YellottConeDensityCalculator`.
- The size of the ROI is adaptive, and is determined by the distance to the fovea (at this point, we only have an estimate of the location of the fovea; it will be refined later, see 4.): the closer to the fovea, the smaller the ROI, as cone density is expected to peak at the fovea and decrease monotonically with eccentricity (see graph here); eccentricity here means "distance from fovea", usually in degrees. Original images are 720x720 pixels, and the ROIs are 160x160 for eccentricities < 0.4°, up to 300x300 for eccentricities > 7°.
- Each ROI of the montage is processed by Yellott's method, giving an estimate of the cone density at that eccentricity (in cones/mm^2). The results are stored in `P:\...\_Results\SubjectXXX\SessionYYY\density_analysis_new\densities_raw_x.csv` and `...\densities_raw_y.csv` for the horizontal and vertical meridians respectively. Note that positive eccentricity for X-axis corresponds to the nasal meridian, and positive eccentricity for Y-axis corresponds to the inferior meridian; the logic of this is handled by the `eccentricity` function in here. Again, the fact that AOSLO images are mirrored is important to be taken into account here.

Fitting of theoretical cone density

- Very close to the fovea (< 0.5°) the cone packing is so dense that the resolution of the AOSLO confocal images isn't enough to resolve individual cones, and so estimates of cone density become unreliable (underestimation). Furthermore, the raw densities are sometimes very noisy, often because of image quality; near the periphery (> 6-7°) it happens that many density estimates are too high (overestimation), perhaps because of the presence of rods. To mitigate this, we fit a bilateral theoretical model of the cone density for horizontal and vertical axes.
- The fitting is handled by `YellottConeDensityCalculator` again, even though it would make sense to refactor it into a proper class.
- In details, the model reads $$\mathrm{cone\ density} \sim \exp\left(\texttt{A}*{\mathrm{mer}} - \texttt{B}*{\mathrm{mer}}\cdot\lvert\mathrm{ecc} - \texttt{C}*{\mathrm{mer}}\rvert + \frac{\texttt{D}*{\mathrm{mer}}}{\lvert\mathrm{ecc} - \texttt{C}*{\mathrm{mer}}\rvert + \texttt{E}*{\mathrm{mer}}}\right)$$ where $\mathrm{ecc}$ is the eccentricity (distance to fovea in degrees), $\texttt{A}*{\mathrm{mer}}$, $\texttt{B}*{\mathrm{mer}}$, $\texttt{C}*{\mathrm{mer}}$, $\texttt{D}*{\mathrm{mer}}$, and $\texttt{E}*{\mathrm{mer}}$ are the parameters to be fitted for each meridian $\mathrm{mer}$ (nasal, temporal, superior or inferior). Note that we constrain $\texttt{C}*{\mathrm{nasal}} = \texttt{C}*{\mathrm{temporal}} \equiv \texttt{C}*{\mathrm{X}}$ and $\texttt{C}*{\mathrm{superior}} = \texttt{C}*{\mathrm{inferior}} \equiv \texttt{C}*{\mathrm{Y}}$, and so $(\texttt{C}*{\mathrm{X}}, \texttt{C}_{\mathrm{Y}})$ denotes the refined location of the fovea ($\equiv$ cone density peak), used as the origin of the coordinate system.
- Other contraints include:
  - continuity at the fovea $\mathrm{ecc} = 0$ (for $\mathrm{X}$ and $\mathrm{Y}$ axis separately),
  - peak cone density (at the fovea) should comply with histological data (between 145k and 320k cones/mm^2, see e.g. Curcio 1990),
  - peripheral cone density should be at least 7200 cones/mm^2 (again, to comply with histology)
- The loss that is optimized in order to fit the model is carefully crafted to take into account the underestimation near the fovea and the potential overestimation near the periphery, in a smooth weighted way; density estimates that are too far from the theoretical curve are discarded to avoid fitting on outliers.
- From the model, $\texttt{width}$ (in °) & $\texttt{max_slope}$ of the peak are computed for each of the four meridians; the former is the width of the peak basis, based on the derivative of the fitted curve (should give a rough idea of where the turning point is), and the latter measures how steep the peak is.
- The raw, smoothed, and fitted densities (data spaced by 0.1°) as well as the $\texttt{width}$ & $\texttt{max_slope}$ parameters are stored in `P:\...\_Results\SubjectXXX\SessionYYY\density_analysis_new\densities.csv`.

Retinal layer thickness extraction:

- OCTs are an imaging modality that allows to visualize the different layers of the retina: it takes cross-sectional images of the retina ("slices") called B-scans, which when stacked together give a 3D representation of the retina and its layers. The OCT data is available on Discovery, a software of RetinAI which allows to visualize the B-scans and, in particular, to segment the different layers of the retina.

-
  - Out of these OCTs, we extract the following data for this work --, again all explained in more details in this guide notebook:
    - thickness of the retinal layers: CohortExtractor handles thickness computation (using segmentation provided by Discovery) of RNFL, GCL+IPL, INL+OPL, ONL, PR+RPE, Choroid as well as the CVI (Choroidal Vascularity Index, a measure of how much vessels), while the thickness of the OS layer (Outer Segment of the photoreceptors, which is the darker layer

within the PR+RPE layer which gets thicker close to the fovea) is extracted by a homemade script (see `src/cell/layer/os_layer_extraction`). These thicknesses are stored in `P:\...\_Results\SubjectXXX\SessionYYY\layer_new\layer_*.json`.
    - 3D model of the foveal shape: from the Discovery segmentation, we extract the foveal shape (defined by the thickness of retina, i.e. from the top of the vitreo-retinal interface to the bottom of the RPE) and build a 3D model of it. tThe script handling this task is found here. The raw data points (in 3D) are extracted and stored in `P:\...\_Results\SubjectXXX\SessionYYY\layer_new\fovea_3d.csv`, while the parameters of the 3D model are saved in `P:\...\_Results\SubjectXXX\SessionYYY\layer_new\fovea_3d_fitted_params.csv`. Again, more details about the extraction in the guide notebook.

Retinal layer processing module of the pipeline

- In order to be compared with the cone densities in the pipeline, the retinal layer thickness is processed by the `src/cell/layer` submodule (except the `_extraction.py` script that are used in the previous step). Notably, in order to populate the X and Y axes with the basically the same eccentricities as the cone densities (in order to be compared), the data within the *triangles* along each of the four meridians is kept; see `P:\...\_Results\SubjectXXX\SessionYYY\layer_new\triangle_*.png`.
    - The center of the coordinate system for the OCTs (in order to be aligned with AOSLO data's coordinate system) is computed using `get_cube_center`, which sets the center ($\equiv$ location of the fovea) as the location of the bright white dot in the central B-scan of the considered eye/subject.
    - This requires to have filled the look-up table `V:\Studies\AOSLO\data\cohorts\AOSLO healthy\lut_subject_to_white_dot_bscan.json` for all the subjects of the cohort.
    - It maps each subject to the number of the central B-scan of the OCT cube (see `V:\Studies\AOSLO\data\cohorts\AOSLO healthy\lut_subject_to_oct_data_paths.json` to see which cube to look at for your subject), i.e. the one that contains the bright white dot in the fovea pit. E.g., if the white dot is in the B-scan 48/97, please report 48 in the look-up table for this subject.

- Again, we have to make sure that the orientation of the X-axis is correct; this is handled by `LayerCollection`'s method `_invert_layer` in `src/cell/layer/layer_collection.py`. This class manages the loading and processing of the layer thickness data.

- Each pixel ($\equiv$ A-scan) of the OCT cube is associated with: - the thickness of the different retinal layers (RNFL, GCL+IPL, INL+OPL, ONL, PR+RPE, Choroid, OS) at this location, as well as the CVI, - the eccentricity of the pixel (in degrees) from the fovea.

- The `LayerThicknessCalculator` class handles the aggregation of the thickness data in the X and Y axes (data within the four triangles): an A-scan is kept if it lies in one of the four triangles (controlled by the `growing_factor` attribute), and then the thicknesses are averaged for each eccentricity ($\equiv$ radially from the fovea). The results are not saved at this point (and are pretty quick to compute anyway).

Cone density vs layer thickness module of the pipeline

- With the cone density (as a `Density` instance) and the layer thickness (as a dictionary of `Layer` instances) data in hand, the `DensityLayerAnalysis` class is used to compare the two.
- For each subject, it outputs results in `P:\...\_Results\SubjectXXX\SessionYYY\density_analysis_new`. It includes plots of cone density vs layer thickness for each of the retinal layers (RNFL, GCL+IPL, INL+OPL, ONL, PR+RPE, Choroid, OS) and the CVI, for the horizontal and vertical meridians separately, as well as an aggregated plot of the mentioned features (`layers_compared_to_density_*.png`). It also computes Spearman's correlation coefficients & p-values, and saves them in `spearmans_*.txt`. A summary of all the cone densities and layer thicknesses for all eccentricities is saved in `results.csv` (open with Excel)

Density statistics module of the pipeline

- The `DensityStatistics` class has several parts. It computes global statistics about the cohort, and its results are saved in `P:\...\_Results\all_stats_new`.
    - First, it gather the Spearmans' correlation coefficients and p-values for all subjects, and outputs the violin plots `spearmans_correlation_for_*.png`. These plots show the distribution of the correlation coefficients for each layer, and indicates what proportion of the subjects have a significant correlation (p-value < 0.05). It also saves an Excel sheet.
    - Then, the class `BaselineCharacteristics` outputs plots and Excel sheets stratifying for each feature (cone density or layer thickness) by some baseline characteristics (includes age, visit date, sex, axial length, spherical equivalence). These characteristics for each subjects are found in `V:\Studies\AOSLO\data\cohorts\AOSLO healthy\DATA_HC+DM.xlsx`.

Additional remarks

- the `step` keyword that pops-up everywhere is always 0.1 in pratice, it denotes the step size for the eccentricities.
- When wondering where a plot comes from in `P:\AOSLO\...\_Results`, Ctrl+Shift+F in this repo is your friend.