AFRL-RH-FS-TR-2020-0001

Multimodal-Multifunctional Image Fusion for Morphological and Functional Evaluation of the Retina

Edward A. Pier
Oceanit

Joel N. Bixler
711th Human Performance Wing
Airman Systems Directorate
Bioeffects Division
Optical Radiation Bioeffects Branch

2 January 2020

Final Report - June 2014 to March 2020

DESTRUCTION NOTICE - Destroy by any method that will prevent disclosure of contents or reconstruction of this document.

Air Force Research Laboratory
711th Human Performance Wing
Airman Systems Directorate
Bioeffects Division
Optical Radiation Bioeffects Branch
JBSA Fort Sam Houston, Texas 78234

NOTICE AND SIGNATURE PAGE

Using Government drawings, specifications, or other data included in this document for any purpose other than Government procurement does not in any way obligate the U.S. Government. The fact that the Government formulated or supplied the drawings, specifications, or other data does not license the holder or any other person or corporation; or convey any rights or permission to manufacture, use, or sell any patented invention that may relate to them.

This report was cleared for public release by the 88th ABW Public Affairs Office and is available to the general public, including foreign nationals. Copies may be obtained from the Defense Technical Information Center (DTIC) (http://www.dtic.mil).

"Multimodal-Multifunctional Image Fusion for Morphological and Functional Evaluation of the Retina"

(AFRL-RH-FS-TR- 2020- 0001) has been reviewed and is approved for publication in accordance with assigned distribution statement.

Digitally signed by SHORTER.PATRICK.D.1023156390
Date: 2020.02.10 07:46:18 -06'00'

PATRICK SHORTER, Major, USAF
Chief, Optical Radiation Bioeffects Branch

MILLER.STEPHANIE.A.1230536283 Digitally signed by MILLER.STEPHANIE.A.1230536283
Date: 2020.03.22 11:38:38 -05'00'

STEPHANIE A. MILLER, DR-IV, DAF
Chief, Bioeffects Division
Airman Systems Directorate
711th Human Performance Wing
Air Force Research Laboratory

This report is published in the interest of scientific and technical information exchange, and its publication does not constitute an official position of the U.S. Government.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704–0188

1. REPORT DATE *(DD–MM–YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From — To)*
02-01-2020	Final Report	June 2014 to March 2020

4. TITLE AND SUBTITLE

Multimodal-Multifunctional Image Fusion for Morphological and Functional Evaluation of the Retina

5a. CONTRACT NUMBER

FA8650-16-C-6678

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

Edward A. Pier, Joel N. Bixler

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

H0LY

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Air Force Research Laboratory
711th Human Performance Wing
Airman Systems Directorate
Bioeffects Division
Optical Radiation Bioeffects Branch

Oceanit
828 Fort Street Mall
Suite 600
Honolulu, HI 96813

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Air Force Research Laboratory
711th Human Performance Wing
Airman Systems Directorate
Bioeffects Division
Optical Radiation Bioeffects Branch

10. SPONSOR/MONITOR'S ACRONYM(S)

711 HPW/RHDO

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

AFRL-RH-FS-TR-2020-0001

12. DISTRIBUTION / AVAILABILITY STATEMENT

Distribution A: Approved for public release; distribution is unlimited. TSRL-PA-2020-0143. The opinions expressed on this document, electronic or otherwise, are solely those of the author(s). They do not represent an endorsement by or the views of the United States Air Force, the Department of Defense, or the United States Government.

13. SUPPLEMENTARY NOTES

14. ABSTRACT

Oceanit has developed RetinaView, a suite of tools for multi-modal retina image analysis and data fusion. RetinaView includes tools for data management, image display, image registration, image fusion, and specialized display of OCT and hyperspectral data. Its key innovations include: a relational database-backed data manager that presents as a reorderable file system, curved en face slicing of OCT volumes, and an optimization of thin plate spline warps so they can be computed in real time.

15. SUBJECT TERMS

Retina Multi-Modal Image Fusion

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Joel N. Bixler
U	U	U	U	22	19b. TELEPHONE NUMBER *(include area code)* 210-539-8172

This Page Intentionally Left Blank

TABLE OF CONTENTS

LIST OF FIGURES

1.0 INTRODUCTION

Researchers and clinicians have a broad range of imaging techniques available to evaluate the health of the eye. These include conventional fundus imaging, Scanning Laser Ophthalmoscopy (Scanning Laser Ophthalmoscopy (SLO)), Optical Coherence Tomography (Optical Coherance Tomography (OCT)), as well as multi- and hyperspectral imaging. Additionally, various forms of visual function testing can produce spatially resolved data.

Previously, each instrument manufacturer provided proprietary processing, analysis, and display software, but there was no openly available, general purpose tool that could fuse data from all sources and modes. The proliferation of proprietary tools created the following problems:

- Users were denied the synergy that comes from integrating different modes of analysis.
- It was difficult to customize proprietary software to use new data types or algorithms.
- Duplicated software development effort drove up the cost of imaging systems.

To address this need, Oceanit developed RetinaView, a cross-platform suite of tools dedicated to multi-modal retina image analysis and fusion. We describe RetinaView in detail in the following sections.

2.0 VIPA AND PYTHA

Oceanit has faced a similar challenge with aerial reconnaissance imaging. There too, each sensor manufacturer produces its own analysis and display software, making data fusion difficult. Oceanit addressed this problem by creating a Versatile Image Processing Architecture (Versatile Image Processing Architecture (VIPA)), which is a general purpose platform for image analysis designed to fuse heterogeneous data. VIPA consists of processing modules that act as sources, sinks and processors of data. Developers link these modules in an intuitive graphical environment (see Figure 1) to create applications.

Much of the functionality needed for aerial reconnaissance (e.g. noise suppression, feature enhancement, image registration, data fusion, input/output, and display) is also useful for ocular imaging, so we used VIPA to provide the underlying image processing functionality for RetinaView. However, VIPA follows a stream-oriented paradigm that is not conducive to the kind of event-based programming required for building user interfaces. We therefore developed a Python wrapper for VIPA, known as “Pytha”, to provide this functionality. We describe VIPA and Pytha below.

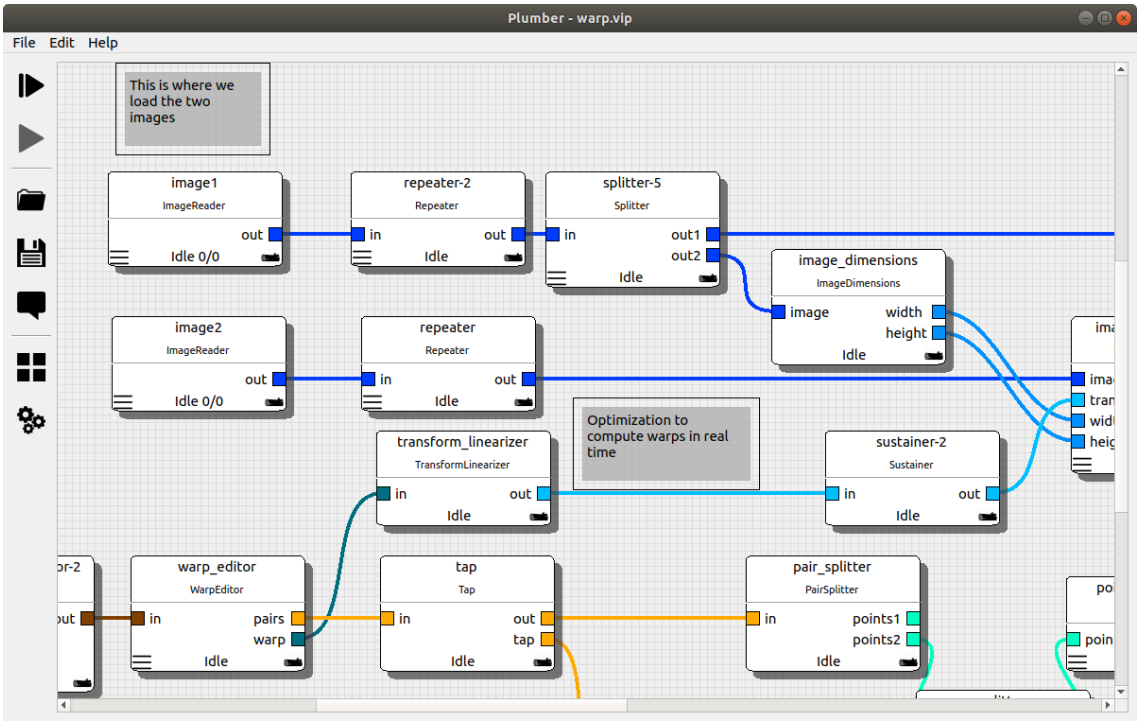

Figure 1. VIPA's graphical development environment, known as "The Plumber"

2.1 VIPA

A software platform for analyzing and fusing ophthalmic data has the following requirements:

- **Sensor agnostic** — It must isolate hardware characteristics and data storage formats from its internal representation of information.
- **Extensible** — Processing and registration of ocular images is the subject of ongoing research. The system must accommodate new algorithms, imaging modes and data formats for it to remain relevant into the future.
- **Parallelizable** — Image processing is compute intensive, and future advances will only increase this burden. The software must be scalable to take advantage of multiple processors and massively parallel graphics processing units.
- **Software Reuse** — There is significant commercialization potential in licensing the software to equipment manufacturers to reduce their software development costs. To achieve these savings the software must allow existing components to be repurposed for each manufacturer's needs.
- **Rapid Prototyping** — Software must serve, not restrict the user. The best way to achieve this is to place the software in users' hands and receive practical feedback

throughout the development cycle. It must be possible to create working prototypes quickly with low enough cost to discard those that prove impractical.

These requirements demand a paradigm shift in software development methodology. Conventional programming languages are cumbersome for image processing, because they concentrate on low level pixel-by-pixel details. For this reason Oceanit has developed a Versatile Image Processing Architecture (VIPA). VIPA was designed from the ground up to address the requirements listed above.

VIPA replaces traditional detail-intensive line-by-line text-based coding with a high level graphical development environment. Developers assemble reusable software modules into processing pipelines, through which images and other information flow. This modular design has several benefits.

VIPA's modularity makes it instrument agnostic, because input and output are isolated in modules that translate data file formats to and from VIPA's internal representations. Adding a new format only requires a new swappable module. VIPA has a rich set of internal image data types, supporting color and grayscale images with floating point or integer pixels of various bit depths. VIPA is not limited to images. It handles numbers, text, booleans, vector graphics, sound and more. Developers can add new data types to address a nearly unlimited problem space.

VIPA's modularity also makes it extensible. With traditional text-based programming, adding new functionality requires significant effort. With VIPA, a developer simply adds a new module that is then guaranteed to interoperate with all existing modules. VIPA modules can wrap well-tested and reliable third party and legacy code, saving considerable effort.

VIPA's modularity promotes software reuse. All modules communicate using a uniform protocol, so developers can reassemble modules written for one purpose to solve new problems.

VIPA is an excellent rapid prototyping tool. Its high level structure relieves the developer of pixel level details, and its graphical development environment provides an intuitive visualization of data flow and powerful debugging tools, such as performance indicators and module level tracing. Developers can create VIPA applications in minutes that would take days using traditional text-based programming.

VIPA is parallelizable. It represents processing threads with "Pump" modules, which developers can use to take advantage of multiprocessor hardware. They do not need to worry about race conditions, synchronization, thread lock and other issues that plague conventional multithreaded programming.

2.2 VIPA's Capabilities

We list some of VIPA's functionality below.

- **Input/Output** — VIPA can read and write all standard video formats and most still image formats used in computer graphics. Additional formats are easy to add.
- **Data display** — VIPA has a graphics display module that can show multiple images with line art and text overlays. Users can pan and zoom using intuitive gestures.
- **Color** — VIPA handles color and monochrome images and has modules for splitting images into Red/Green/Blue, and Hue/Saturation/Value color planes. It can match colors using a perceptually accurate color distance and create false color images.
- **Transforms** — VIPA has extensive support for spatially transforming images. It treats transforms as data that users can create, manipulate, combine and display. VIPA supports affine transforms, homolographic transforms, camera distortion transforms, thin plate splines and more.
- **Registration** — VIPA can register images using a number of techniques. These include area-based techniques, such as phase correlation, and feature-based techniques, which find and match distinctive features in a pair of images.
- **Pixel value manipulation** — VIPA can apply a broad range of linear and non-linear operations to adjust brightness, contrast, and color balance.
- **Computer vision** — VIPA combines the OpenCV computer vision library with the TensorFlow machine learning platform and custom algorithms to extract semantic information from images. For example, edge detection and object recognition can trace blood vessels in the eye, identify layers in OCT data and find other features due to disease or injury.

2.3 Pytha, a Python Wrapper For VIPA

While VIPA's modular, stream-oriented approach to data processing makes it possible to construct sophisticated applications quickly, it is not well suited to event-based programming, such as loading a file in response to a user request. VIPA's "Plumber" graphical development environment provides this level of user interaction, but it is not appropriate for end-users, because it exposes too much low level functionality. Therefore, an end-user application, such as RetinaView, requires a hybrid of VIPA and a traditional programming language. We therefore created a set of Python bindings for VIPA, known as "Pytha".

Python is a modern, high-level scripting language. It is designed for rapid development and data analysis – features that dovetail well with VIPA. Python ranks among the top

four in most assessments of programming language popularity and is currently on the rise as many top university computer science programs are beginning to use it as their core teaching language.

Pytha can do the following:

- **Load VIPA applications.** Developers can create VIPA applications using the Plumber and then integrate these into Python programs. Python can load multiple applications and run them at the same time or separately in response to external events or program states.
- **Start, stop and reset VIPA applications.** VIPA applications have a discrete lifespan that ends when the data sources reach end of stream. Some data sources have no end and therefore run indefinitely. The Plumber provides buttons to start an application, stop it at an arbitrary time, and reset the application to the beginning of the stream. Pytha can also perform these actions. For example, a Python program could wait for the user to specify an input file before starting VIPA processing. Then if the user specifies a subsequent input file, the Python application could stop and reset the VIPA processing and then restart it after changing the input.
- **Set module parameters.** Pytha can adjust module parameters at runtime.
- **Integrate Python and VIPA GUI components.** Python has bindings for the Qt Graphical User Interface (GUI) library used by VIPA. Developers can create GUI components on the Python side and place them in the same layout with VIPA GUI components. For example, a Python-controlled file menu could be used to set a VIPA input filename module parameter.
- **Pass data between Python and VIPA.** This is accomplished by a pair of modules. One passes data to an arbitrary Python callback function, and the other requests data from a callback. This is particularly useful for saving and reloading the state of a VIPA program (e.g. the warp transform between a pair of images) in response to a user request.

3.0 DATA MANAGEMENT AND THE FILE BROWSER

RetinaView is designed to work with large archives of retina imaging data. Data management is a significant challenge for any data collection or analysis effort. The typical approach is to define a file system directory structure. This has two main advantages: the data are organized hierarchically for efficient searching, and file systems are familiar to everyone and easy to navigate. The main draw-back of this approach is that one static directory structure may not fit everyone's needs. For example, the data collector might prefer a chronological organization, while the data analyzer might prefer grouping by characteristics of the images themselves.

A more sophisticated approach is to use a relational database to store image metadata. This frees the data from a static, pre-defined organization, and lets users regroup the data on the fly by querying the database. The down side of relational databases is that they require the user to learn a query language, like SQL. While SQL is familiar to most computer science majors, it is an unwelcome annoyance to most researchers.

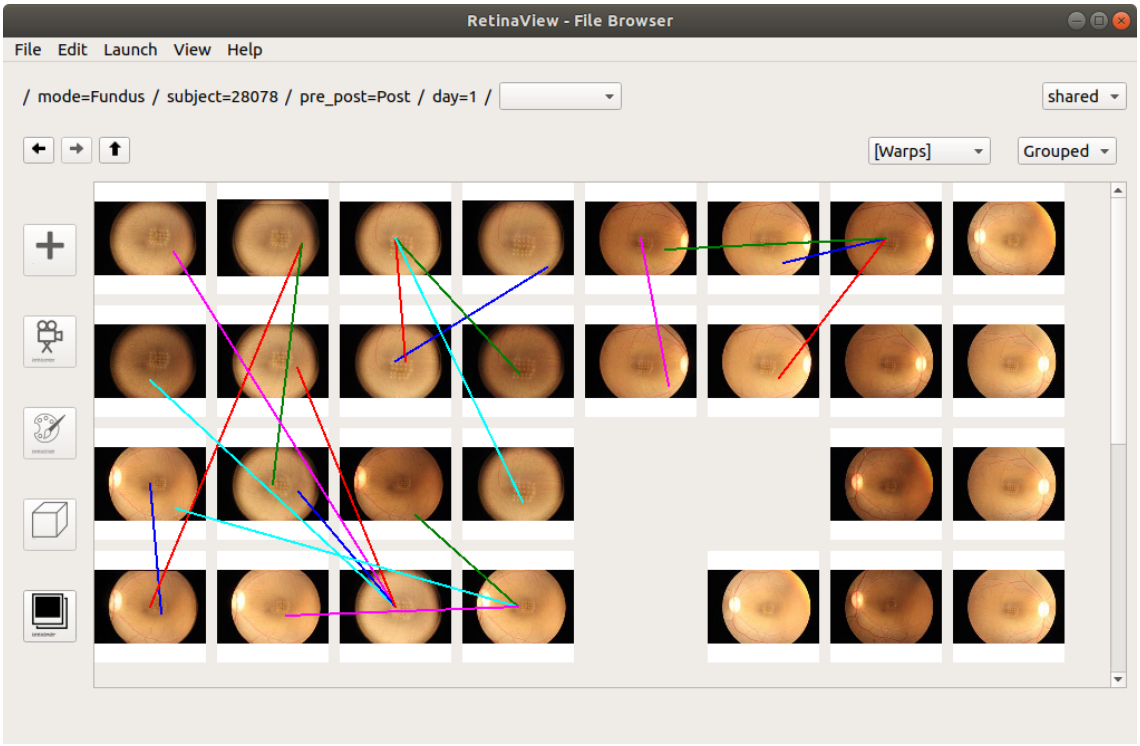

Figure 2. The File Browser

RetinaView takes a hybrid approach that uses a relational database on the back end, but presents a user interface that resembles a file system browser (see Figure 2). The unique feature of the RetinaView file browser is that users can rearrange the simulated file system hierarchy on the fly.

The core of RetinaView's database is a single table with one row per file and columns that list the metadata associated with each file. RetinaView places no restrictions on the number of columns, their names, or content, so the user can customize this table to meet the needs of a particular study or dataset.

The user interface presents icons that represent individual files. For images these are thumbnails of the image itself. For other file types (e.g. text files), it uses representative graphics. A file system directory structure maps onto an ordered list of database columns. For example, at the top level of the hierarchy, we might organize the data by imaging mode (Fundus, OCT, etc.), and then at the next level we might organize it by subject ID. Users construct this list by selecting columns from a "combo box" GUI element. They

can then edit this list with mouse gestures, dragging to rearrange elements or clicking to truncate the list.

Users navigate the hierarchy in the familiar way, by double clicking directory icons to open them and using buttons to move up in the hierarchy or go back to a previous directory. In terms of the database, a particular directory corresponds to a set of specific column values. These values are displayed at the top of the file browser in the file system hierarchy editor.

One of the main purposes of RetinaView is to register and combine sets of images. The File Browser indicates pairs of registered images using colored lines. By default the icons are grouped to minimize the lengths of these lines. The user can also choose to display the icons in raster order, sorted by any of the remaining columns.

The File Browser serves as the central access point for all of the RetinaView tools. Users can open a file using its default tool by double clicking on it. They can also use a specific tool by right-clicking on an icon and selecting the tool from a context menu. This behavior mimics standard operating system file browsers.

Users can also select groups of files by clicking on them. A left click selects a single file. A control-click toggles the selection state of a particular file, and a shift-click selects the clicked file and every image registered with it. A set of buttons on the left hand side of the File Browser lets the user combine the selected files in various ways.

4.0 INGESTING DATA AND THE ARCHIVE MANAGER

Datasets in RetinaView are organized into “archives”. Each archive has its own file property table, meaning it can have its own set of metadata. To appear in the RetinaView File Browser, an archive must be “ingested”. Ingesting a dataset, consists of creating the file property table and generating the image icon thumbnails.

Because the metadata can be different for each archive, RetinaView uses a plugin architecture for ingesting data. Users can supply Python code that follows a simple API for specifying the properties of a particular image. RetinaView comes with a number of plugins written for RHDO laser lesion datasets and a generic plugin that extracts only simple information about file format, etc.

Because imaging datasets can be large, RetinaView does not copy ingested files into dedicated disk storage. The idea is that the data are already archived in a way that was convenient for the data collector and is accessible to the data analyzers, typically on a shared file system.

Because RetinaView does not control the file storage, there is always a danger that the data could be moved so that RetinaView could no longer locate it. To address this problem,

RetinaView identifies files by their cryptographic hashes. This way we associate metadata with the file itself and not its location in the file system. Should the data be moved, RetinaView could recover by searching the modified file system and recomputing the hashes. As an added benefit RetinaView can also identify redundant copies of the same file and file corruption that happens after ingest.

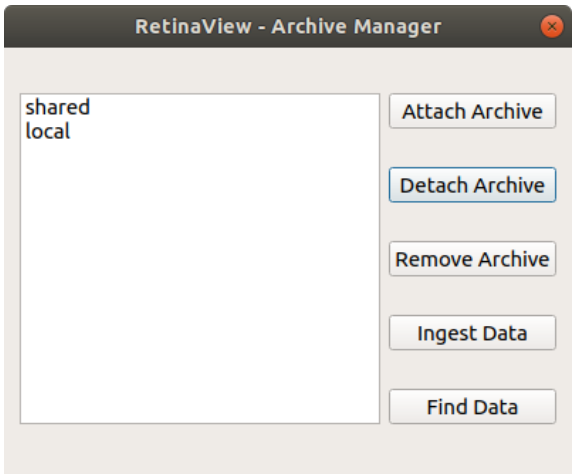

Figure 3. The Archive Manager

The user initiates a new ingest via the Archive Manager (see Figure 3), which can be launched from the File Browser. The Archive manager allows the user to "attach" to an archive that was previously ingested by a different user, "detach" an archive to no longer include it in the local list of available archives but still leave it available for others, and "remove" an archive, deleting its database tables and any generated files but leaving the original data intact.

While archive data may be located on a network shared file system, different users may not necessarily mount this file system the same way. Therefore, RetinaView provides a mechanism for specifying a different file system path prefix for a previously ingested archive. The Archive Manager's "Find Data" button initiates this process.

The parameters for an ingest are specified by the Ingest Wizard (see Figure 4), which steps the user through a series of input pages. The user specifies the directories to recursively search for data, the name of the archive, the directory in which to store generated files, and the Uniform Resource Locator (URL) of a database server. This allows users to create archives locally or for sharing with others. Finally, the user selects the ingest plugin.

After entering the ingest parameters, RetinaView presents the Ingester (see Figure 5), which lists the ingested files, displays the generated thumbnails, and indicates progress.

After ingest, users can edit the metadata for a file by opening it with the File Info tool (see Figure 6). Users can also associate free-form text notes with each file and add new

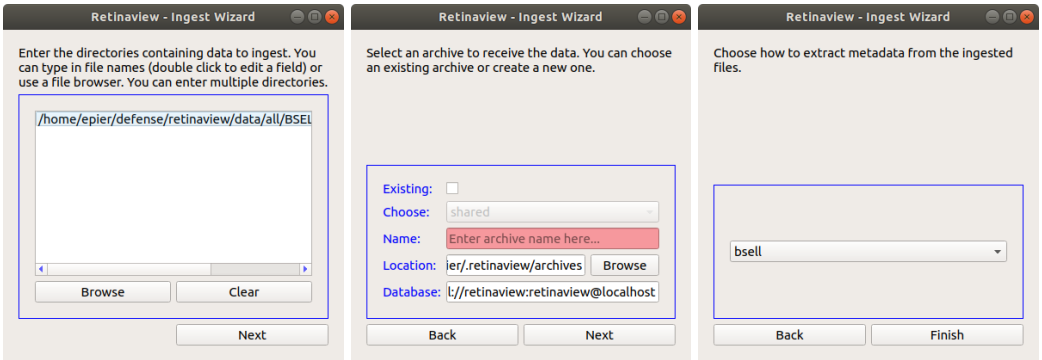

Figure 4. The Ingest Wizard

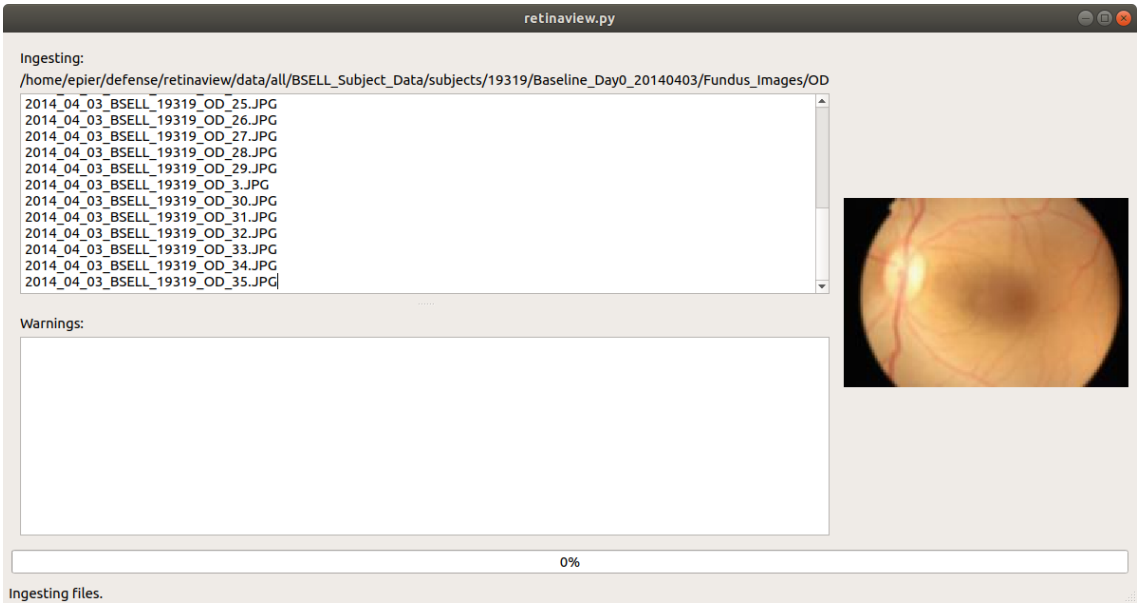

Figure 5. The Ingester

columns to the file properties table.

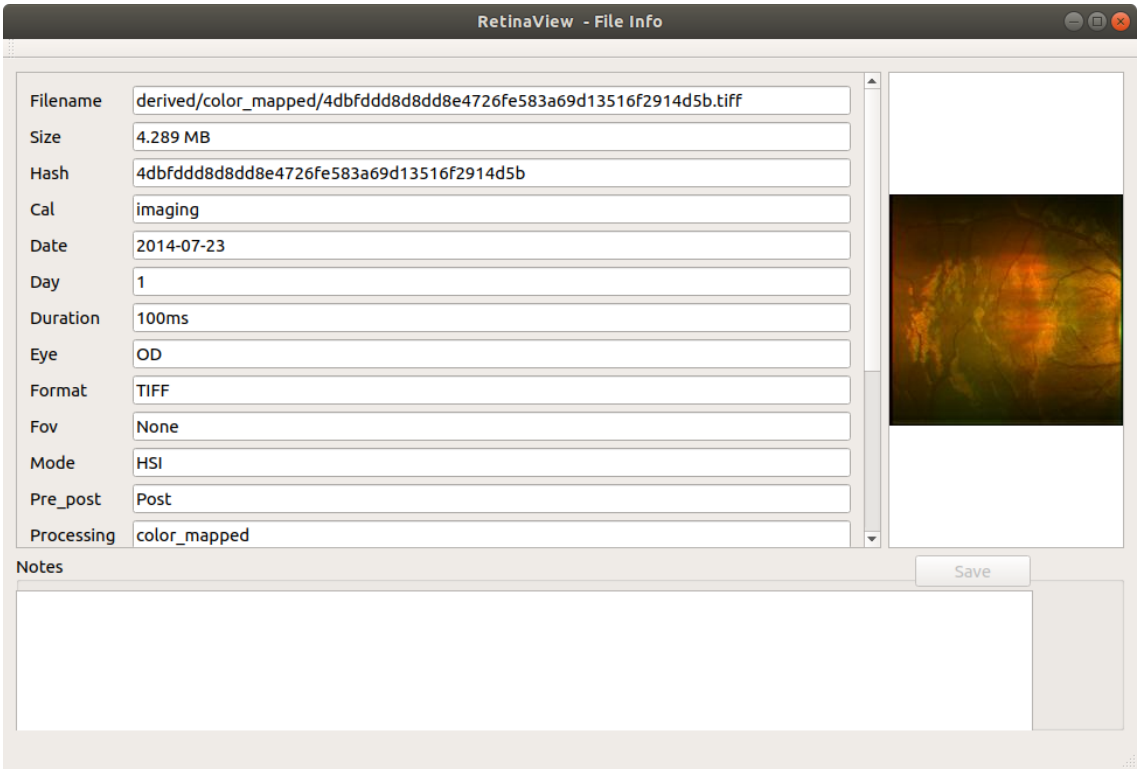

Figure 6. The File Info window

5.0 IMAGE VIEWER

The most basic thing to do with an image is display it. The Image Viewer (see Figure 7) handles this. Users can adjust the brightness and contrast using the sliders on the right and can pan and zoom using mouse gestures.

Users can also mark regions of interest and distance-measuring rulers on the image. Regions of interest can be drawn free-form, as circles, or pixel by pixel. Users can select and drag existing regions of interest. Users can also save these markings to disk and reload them later. If a pair of images are registered, then markings created on one image will automatically be transformed to the other.

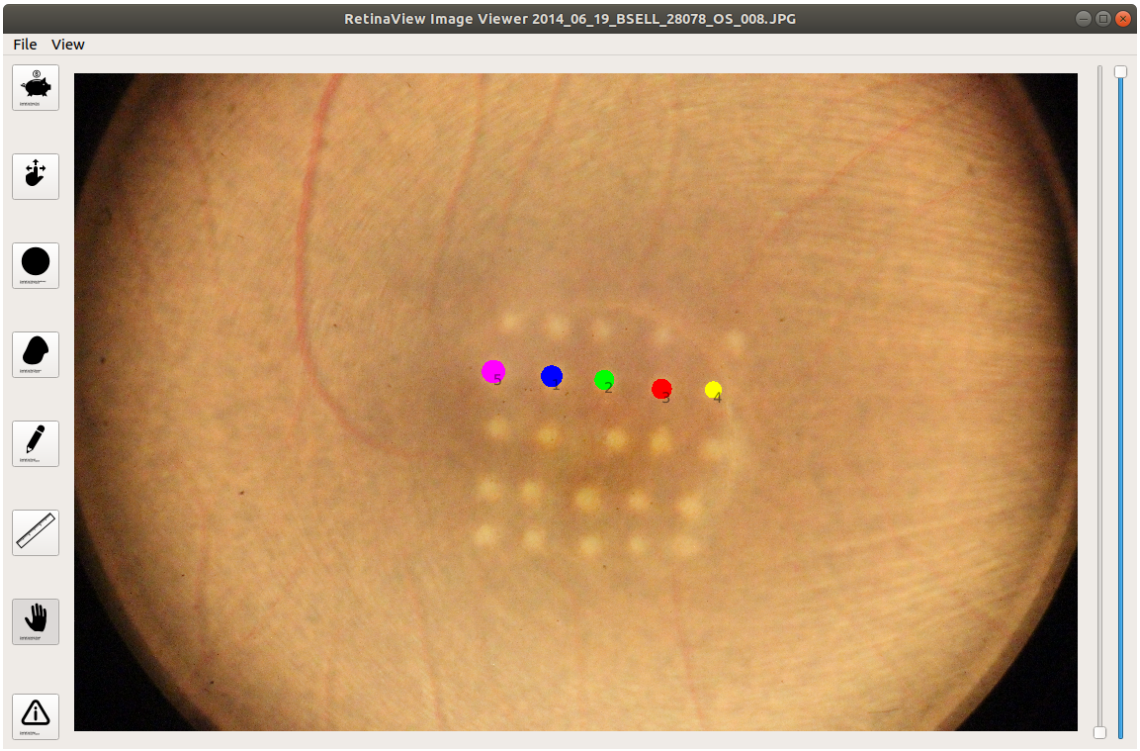

Figure 7. The Image Viewer

6.0 WARPER

The key feature of RetinaView is the ability to fuse images taken with a broad range of instruments. To do this it must be able to register images related by arbitrarily complex transforms. RetinaView's primary tool for registering images is the Warper (see Figure 8). The Warper displays a pair of images with the second (top) image transformed by a thin plate spline warp. A slider at the bottom controls the transparency of the top image and a button on the left allows the user to "blink" between the two images.

A thin plate spline is a mathematical technique for interpolating smoothly in two dimensions between a set of arbitrarily placed points. The warper allows the user to place and move these control points with mouse gestures. Typically, the user places control points on prominent features in the top image and then drags the control point to the corresponding feature in the bottom image.

Thin plate splines are computationally expensive. Ordinarily, a typical computer would not be able to display updates to the warp in real time as the user drags control points. RetinaView deals with this by approximating the warp with a piecewise set of affine transforms. This approximation can speed up the transform by a factor of a hundred or more.

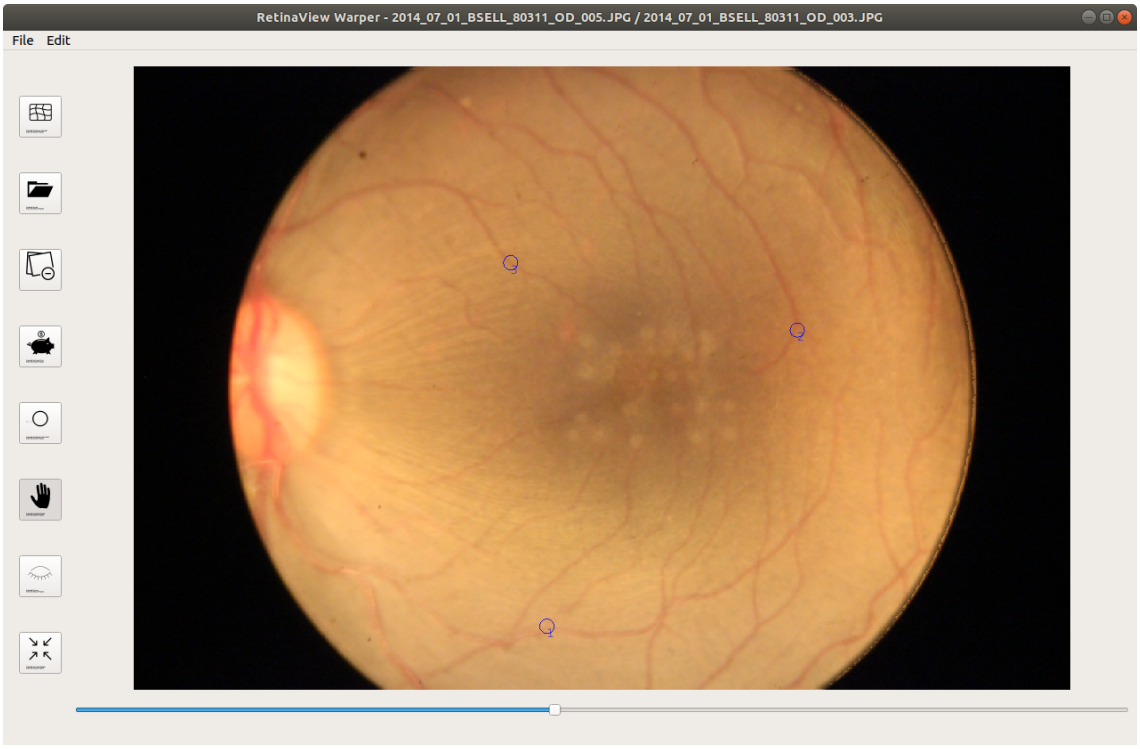

Figure 8. The Warper

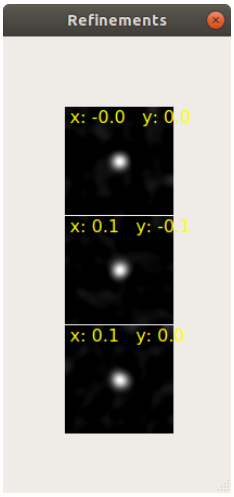

Figure 9. The Warper Refiner

A button on the left of the Warper triggers the Warp Refiner (see Figure 11), which uses phase correlation to optimize the placement of the control points. Phase correlation is a Fourier Transform-based technique that finds the optimal horizontal and vertical shift between a pair of images. It is computationally efficient and robust against noise, but it only works with images related by a translation. As long as the warp is roughly correct,

the transform in the local neighborhood of a control point is well approximated by a translation.

The result of the phase correlation calculation is an image where each pixel corresponds to a possible translation. The pixel brightness indicates the similarity of the original images after applying that translation. The Warp Refiner shows these correlation images so that the user can judge the quality of the resulting refinement. When phase correlation fails, the correlation image shows no clear bright spot. In this case the user may want to remove or replace the offending control point.

Users can save warps to external files or into the images' archive. Warps saved to the archive are indicated by connecting lines in the File Browser and can be used to transform regions of interest and other markings in the Image Viewer. Users can also load warps from disk or from the archive.

7.0 ALIGNER

The Warper is a powerful tool for registering arbitrary pairs of images, but it is only semi-automated. RetinaView can also use phase correlation to register large sets of images automatically with the Aligner (see Figure 10). As described in the previous section, phase correlation is limited to images related by a translation. It is mostly useful for sets of images taken in a time series with a single instrument, such as RHDO's hyperspectral imager.

The first step in using the Aligner is to specify the sets of images that it is to work on in the File Browser. The user constructs a directory structure and then navigates to a particular directory in that structure. The Aligner will then operate on each bottom level subdirectory of the current directory. This allows the user to arbitrarily select and group the data according to its metadata.

For each set of N images there are $N(N-1)$ pairs of images to which phase correlation can be applied. It takes a minimum of $N-1$ pairs to determine the registration between any two images, because the transforms can be chained, but additional transforms can be used to improve the accuracy of the overall solution by doing a least-squares fit.

There are two problems with computing every possible transform. First, it can be computationally prohibitive for large N. Second, while images adjacent in time may be related by small translations, the accumulated drift between images far apart in time may be large enough that the transform is not well approximated by a translation. In this case phase correlation may fail or produce inaccurate results.

The Aligner uses the following strategy. First, it computes the warps between adjacent images. It then computes a first approximation of the global transform between all images and extracts the set of all images whose transform is less than some maximum size. From

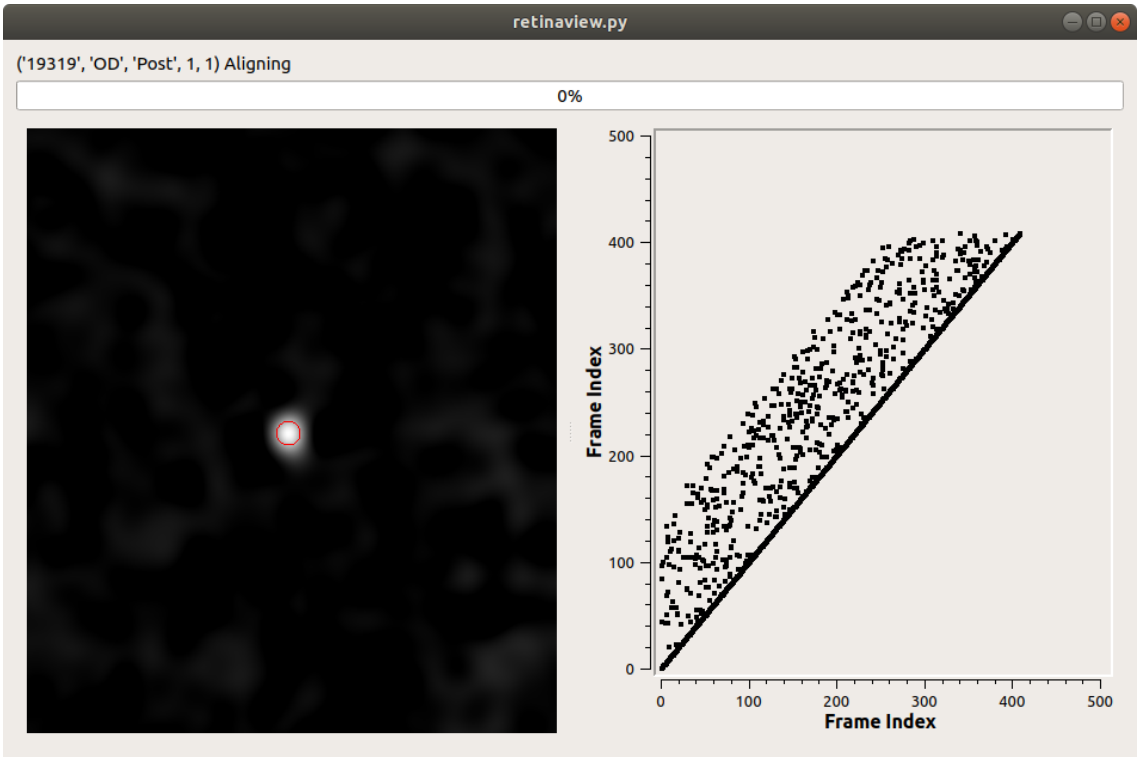

Figure 10. The Aligner

this set it selects 1000 images for additional phase correlation calculations. Finally, it recomputes the global transforms from all $N + 999$ pairs. This gives an accurate solution with a computation time that scales linearly with N.

The Aligner shows the correlation image for each pair and a plot indicating the pairs that it has considered. It also shows the global progress through the image sets. It saves the resulting warps to the archive. Note that a translation is just a warp defined by a single control point.

8.0 COMBINER

The Combiner (see Figure 11) fuses images that have been registered with the Warper or the Aligner. There are many ways to combine images, so the Combiner has a pluggable architecture that can accommodate additional fusion algorithms in the future. The current ways of combining images are:

- **Average** — This individually computes the average of the red, green, and blue color channels of the images and recombines the result into a single color image. The main advantage of averaging images is that it reduces noise. To maintain the full benefit of this noise reduction, the Combiner saves the average as a losslessly

compressed TIFF image with 16 bits per color channel. These images are large, so the user can export them from the File Browser as 8 bit per channel JPEG or PNG images for display.

- **Video** — This produces an MPEG4 video showing the images in sequence. This can be useful for showing change over time.
- **Color Mapped** — If the archive has a "wavelength" metadata column, then the Combiner can create a color image with the red, green and blue color channels computed as weighted averages. The weight depends on the wavelength in such a way as to mimic the color response of the human eye. This is useful for hyper- and multispectral data.
- **Cube** — This produces a three dimensional data structure with each layer of the cube corresponding to a different image. The resulting cubes can be displayed with the Spectral Viewer described below.

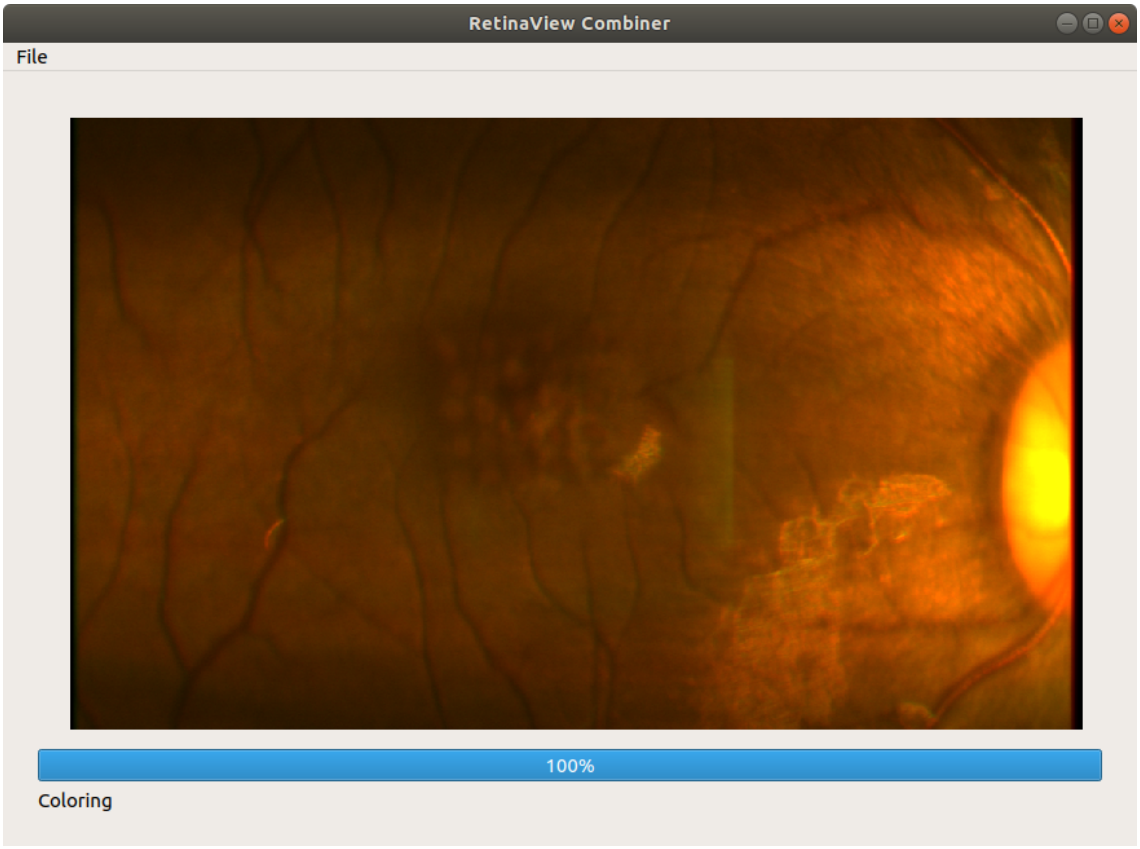

Figure 11. The Combiner

The user starts the Combiner from the File Browser by first selecting a set of images and then clicking the appropriate button for the desired algorithm.

9.0 SPECTRAL VIEWER

The Spectral Viewer (see Figure 12) displays hyperspectral datasets that have been combined into a cube. On the left the Spectral Viewer contains an embedded Image Viewer. By default the Image Viewer displays one of the images used to create the cube. The user can then mark regions of interest in this image, and the Spectral Viewer will plot their averaged spectra on the right.

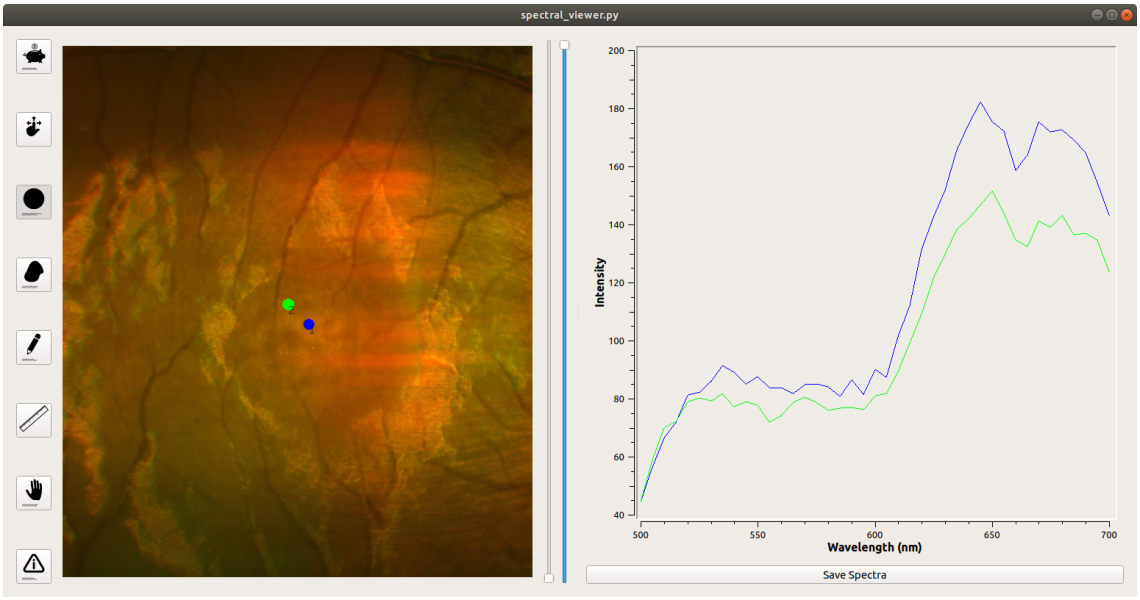

Figure 12. The Spectral Viewer

The user can display other images by dragging them from the File Browser onto the Image Viewer, as long as the image is registered to the cube. A common practice is to display a low noise average or color mapped image instead of the default image.

The user can save the spectra to a text file by clicking the button under the plot.

10.0 OCT SLICER

Optical Coherence Tomography (OCT) is a 3D imaging technique that has become the workhorse of ophthalmology. Many clinicians (particularly retina specialists) now forego the traditional fundus camera for a combined SLO/OCT imager. RetinaView has an OCT Slicer tool (see Figure 13) for displaying this OCT data.

The traditional way to view OCT data is to display individual B-scans (vertical slices into the retina) and allow the user to scroll through the scans. The OCT Slicer provides this capability along with the ability to slice along the other two Cartesian axes to provide transverse and en face views. However, the retina is curved, so a flat en face slice is of limited usefulness. To compensate, the OCT Slicer can also slice along curved slices that

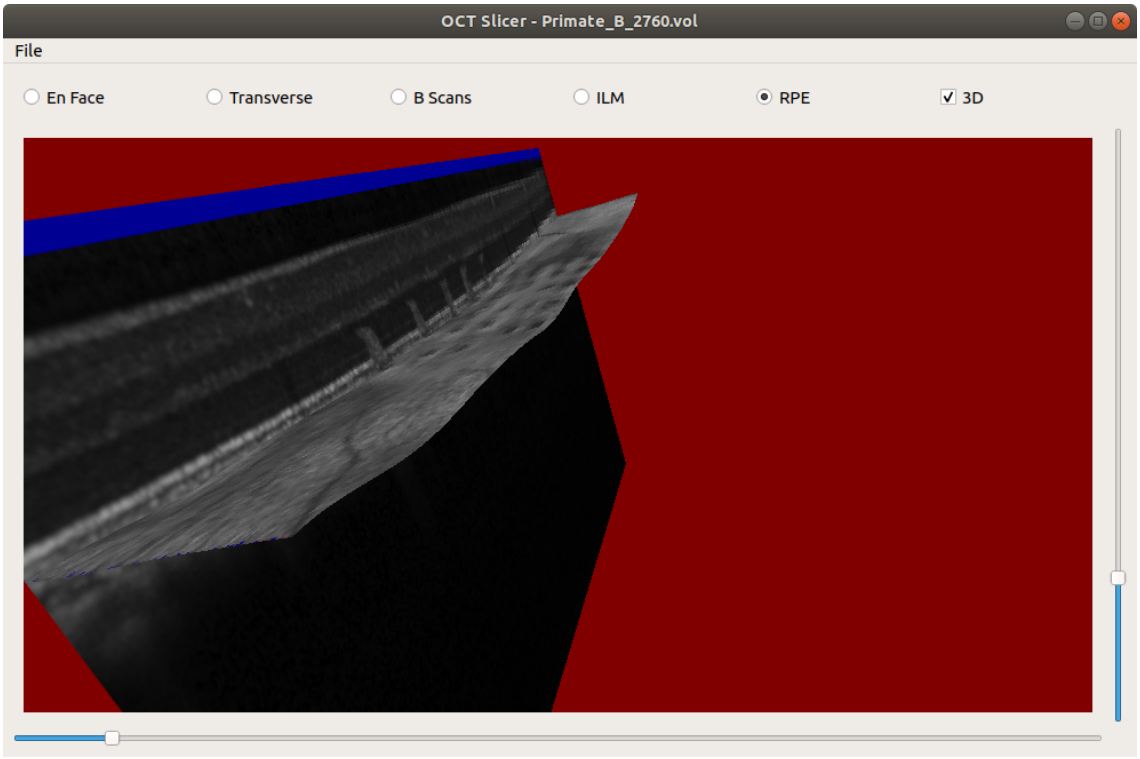

Figure 13. The OCT Slicer

mimic the curvature of the anatomical layers of the retina. Oceanit has been awarded a patent for this visualization technique. The user can choose between the Inner Limiting Membrane (ILM) and Retinal Pigment Epithelium (RPE) layer shapes. The user can also display a 3D rendering of intersecting B-scan and curved en face slices. The position of the slices is controlled with sliders.

The user can save slice images. One possible use case is to extract a slice just below the ILM that shows the blood vessels. This image can then be registered with other modalities using the Warper.

The OCT Slicer reads Heidelberg Spectralis "vol" files. These files contain the imaging data plus Three Dimensional (3D) voxel scaling data and the ILM and RPE layer shapes used for curved slicing.

11.0 FEATURE FINDER

RetinaView has preliminary support for using machine learning to detect features within images. The Feature Finder uses TensorFlow, which is Google's widely used open source platform for training and evaluating neural nets. Users must separately install The TensorFlow C API from https://www.tensorflow.org/install/lang_c . It is available for Linux, Windows and MacOS.

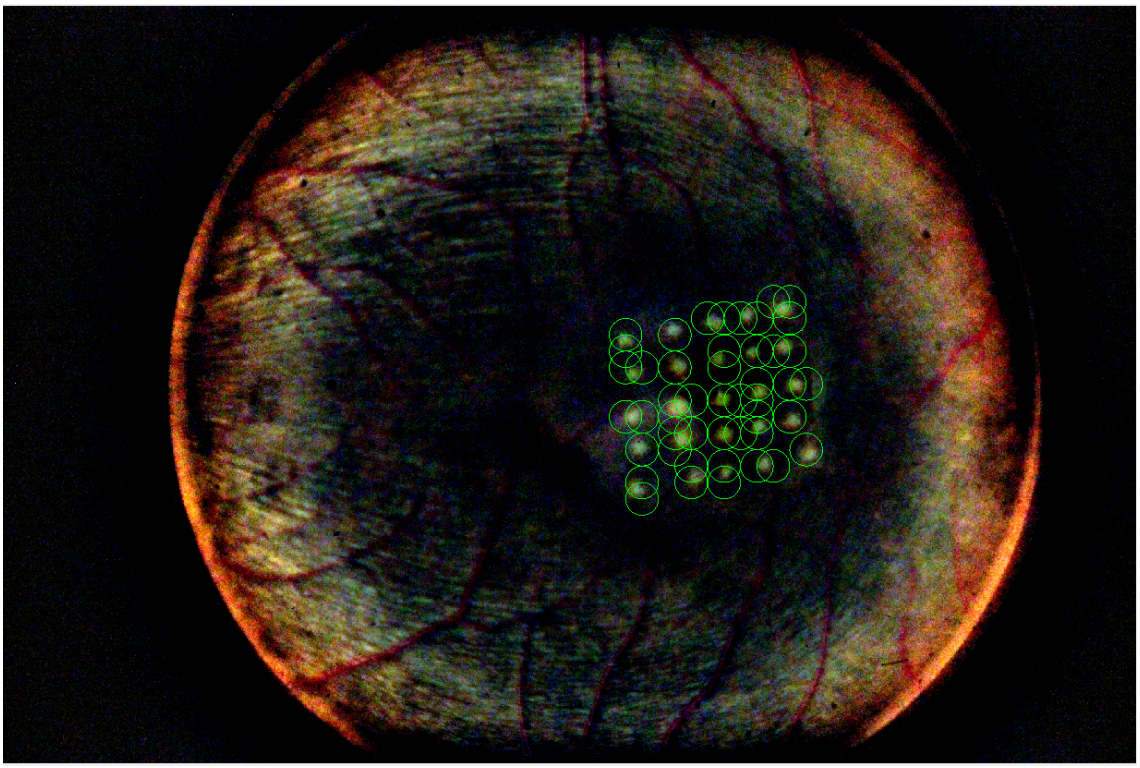

Figure 14. The Feature Finder.

The Feature Finder is currently designed to run a specific machine learning model that is trained to detect laser lesions in fundus images. It first flattens the image by subtracting a 200 pixel Gaussian-smoothed version of each color channel. It then divides the image into a series of overlapping 150x150 pixel "thumbnails" and evaluates the model on each. If the model indicates that a laser lesion is present in a given thumbnail, then the FeatureFinder marks the image with a a green circle (see Figure 14).

While we have demonstrated 95% or better accuracy with our laser lesion detection model, false positives are inevitable, given the large number of thumbnails in a typical fundus image. The Feature Finder therefore includes a slider that lets the user interactively adjust a confidence threshold. This lets the tool support a range of use cases, from an agressive check for anything resembling a lesion to a conservative check that is not likely to cause unnecessary alarm.

While this project has come to completion, future, separately funded work could generalize the Feature Finder to accomodate user-supplied TensorFlow models and preprocessing algorithms.

LIST OF SYMBOLS, ABBREVIATIONS, AND ACRONYMS

3D Three Dimensional

GUI Graphical User Interface

ILM Inner Limiting Membrane

OCT Optical Coherance Tomography

RPE Retinal Pigment Epithelium

SLO Scanning Laser Ophthalmoscopy

URL Uniform Resource Locator

VIPA Versatile Image Processing Architecture