Pupil Analysis GUI

This guide will cover how to analyze mouse pupil videos with the Pupil Analysis GUI MATLAB application. The Pupil Analysis GUI is designed to identify the pupil in videos of the eye and to measure pupil radius and position on each video frame. It can also measure frame luminance to enable the use of a blinking light as an alignment signal between videos and other recording systems.

The GUI allows you to interactively alter the pupil analysis algorithm parameters to find those most optimal to your specific videos. Once you have found those parameters, you can analyze the entire video through the GUI, or you can export those parameters to a mat file and process the video via a script.

Let's get started!

1. Overview

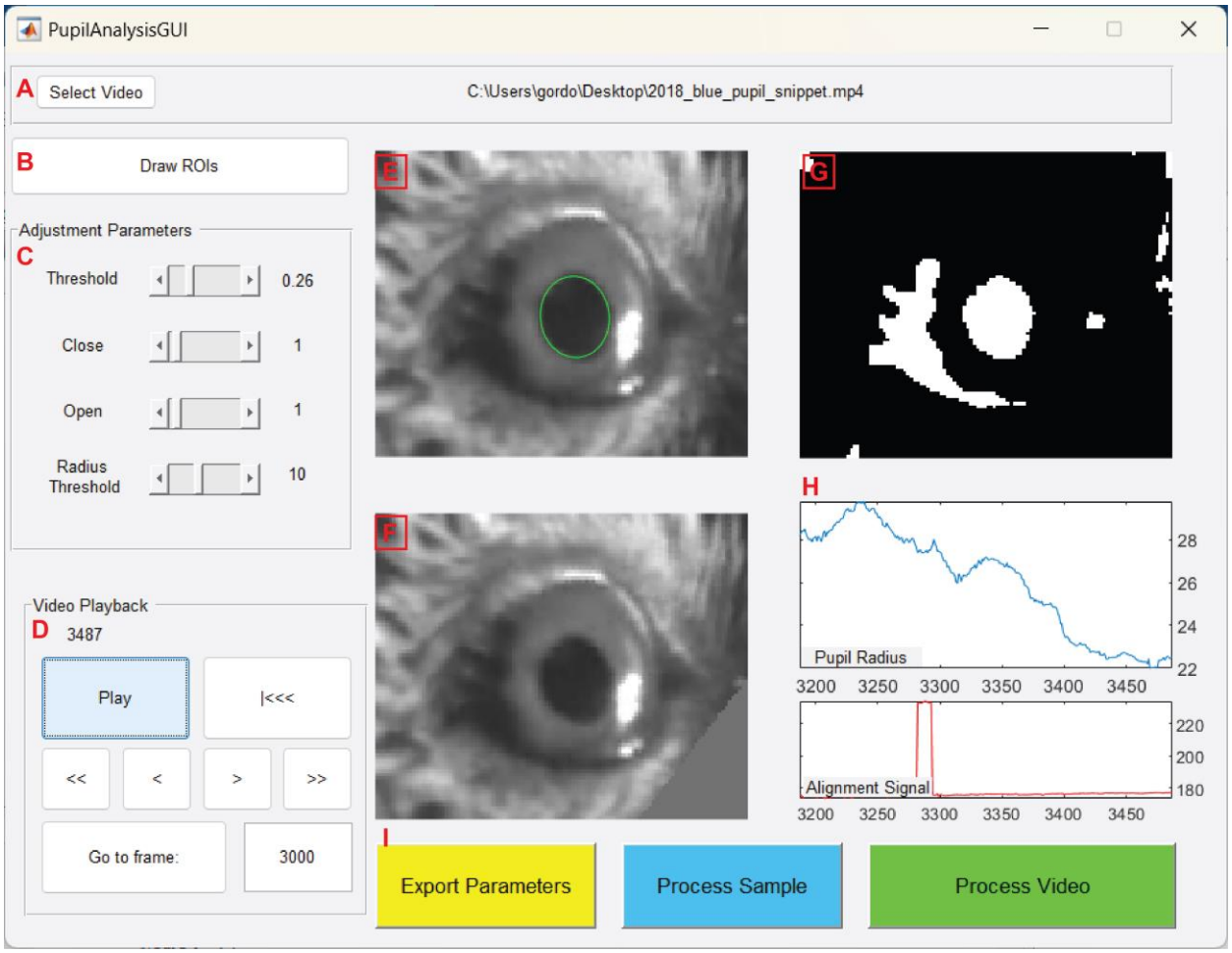

Figure 1-1: The Pupil Analysis GUI in Action

The GUI has multiple components to handle all steps of analyzing the pupil: loading a video, cropping it, setting algorithm parameters, navigating the video playback, running the algorithm, and saving

results. Here we will summarize each component. Note that when you start the program, all GUI elements other than **A** will be disabled.

A. Select Video

This button opens the file explorer to select a video. The GUI supports mp4 and avi files. It is recommended that you use mp4 files, as MATLAB appears to handle them more efficiently than avi files. Once you have selected a video, the first frame of the video will be displayed on the axes **E, F,** and **G**, and the rest of the GUI will be enabled.

B. Draw ROIs

This button will launch a sub-GUI where you can crop the video, draw an alignment signal ROI, and draw masks to obscure "distracting" areas of the video, such as highlights or shadows.

C. Adjustment Parameters

Here you can modify the parameters used to preprocess the frame ("Threshold", "Close", and "Open") and the minimum radius of the pupil ("Radius Threshold").

D. Video Playback

These buttons control the video playback. The Play button will start playing the video and running the pupil identification algorithm on each frame. The algorithm takes some time to run on each frame, so playback will be much slower than regular video speed. When the video is playing, all other GUI elements will be disabled. The other buttons allow you to rewind the video to the beginning, step forward and backward by a single frame or by five seconds, and to jump to specific frames in the video.

E. Original Video & Pupil Identification

This axis displays the current frame of the video, cropped but otherwise unprocessed. If a pupil has been identified, it will be highlighted with a green ellipse.

F. Masked Video

This axis displays the current video frame with masking ROIs applied. Masks can be added to the video via the "Draw ROIs" button (element B).

G. Processed Video:

This axis displays the current video frame after processing using the parameters set via the "Adjustment Parameters" box (element C). This will be a black-and-white video that has been thresholded (*i.e.* binarized by pixel value), inverted (such that white regions in the processed frame correspond to the darkest regions in the original video), and closed/opened (causing the white regions in the thresholded video to be smoothed out).

H. Measurement Plots

These two axes dynamically plot the results of the video analysis. The top axes (blue line) plot the radius of the identified pupil (green ellipse in the raw video window, element E). The bottom axis (red line) plots the alignment signal, which is the mean value of all pixels in the alignment signal ROI drawn using the "Draw ROIs" button. For both plots, the right-most data point corresponds to the current frame. The plot displays data for the previous five seconds. New values are only calculated

when the “Play” button is pressed, so jumping around the video will cause these plots to display empty spaces where data has not yet been processed.

I. Export Controls

These three buttons allow you to export results from your analysis. “Export Parameters” (yellow) will save all of the analysis parameters (including ROIs) to a mat file, which you can later use to analyze data in a script. “Process Sample” will run the algorithm on 30 seconds of the video and display the results, as a way of testing parameters before analyzing the entire video. After processing the video sample, the program will generate a summary plot, but data will **not be saved**. “Process Video” will run the algorithm on the entire video, generate a summary plot, and save the output measurements to a mat file. Longer videos can take a long time to process. For both the Export Parameters and the Process Video buttons, the outputs are saved to the same folder as the original video.

2. Analysis Walkthrough

Here we will go through the process of analyzing a pupil video step-by-step.

A. Running the GUI and Loading a Video

To start the GUI, first ensure that all required functions are on your MATLAB path. Then, type `PupilAnalysisGUI` into the MATLAB command line. The GUI will launch in a new window.

At launch, the only button available is "Select Video." Click this to open a file navigator and select your video.

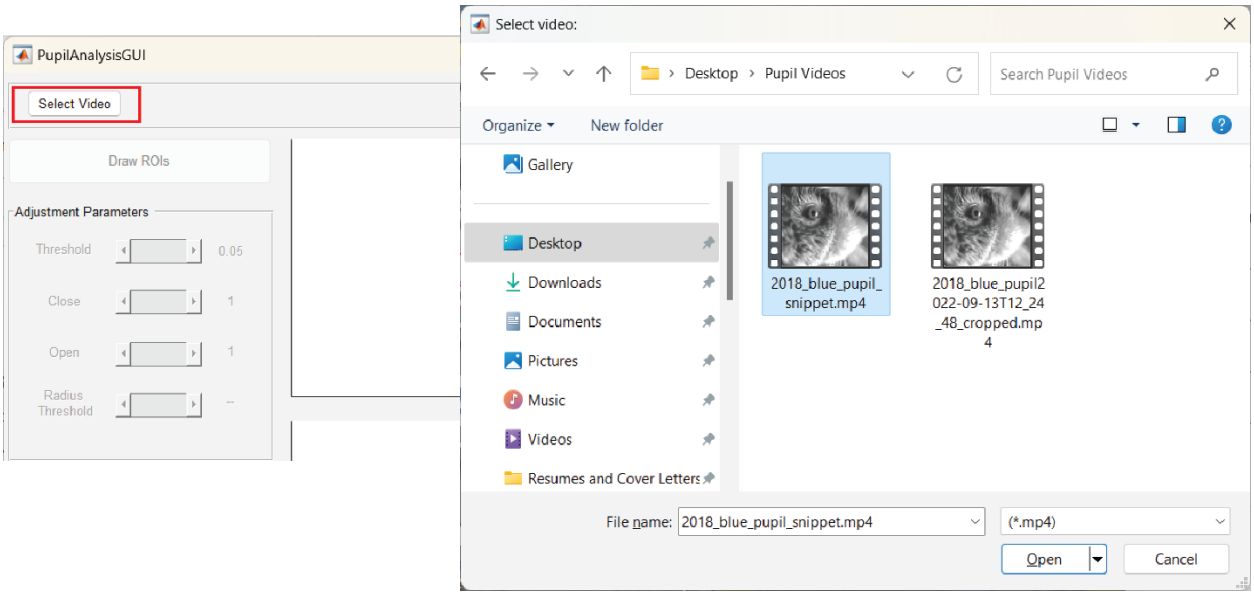

Figure 2-1: Loading a Video

Select your video and click "Open." If your video does not appear in the browser, adjust the file extension filter to look for avi files using the drop-down menu next to the File Name field. The GUI only supports mp4 and avi files.

Once you have selected a video, the rest of the GUI will be enabled, and you can proceed to the next step.

B. Cropping, ROIs, and Masks

Once you have a video loaded, the next step is to isolate the region of your video that contains the eye. To do so, click the "Draw ROIs" button. This will bring up a new GUI screen:

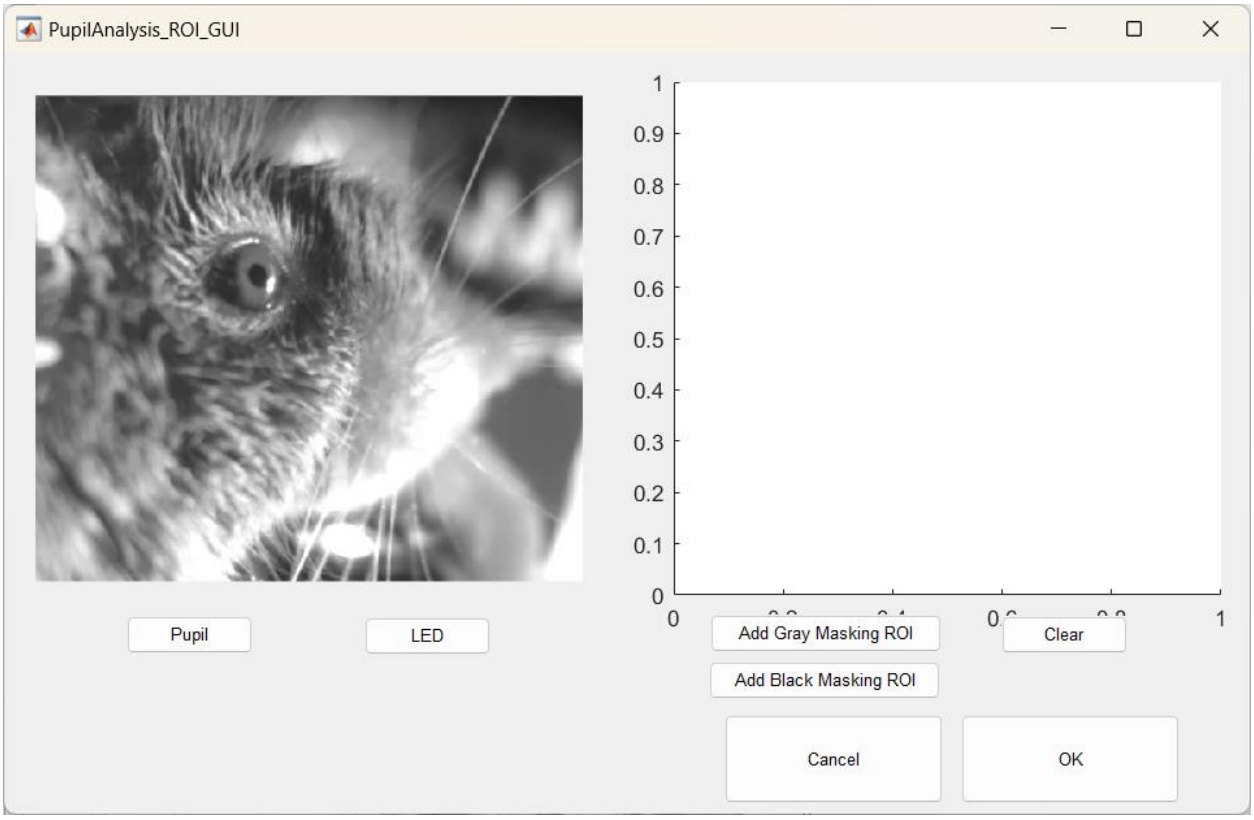

Figure 2-2: The ROI GUI upon startup

To isolate the pupil, click the "Pupil" button, then click-and-drag on the video frame to draw a rectangular bounding box around the eye. This will crop the video to the selected region, excluding the rest of the frame from the pupil analysis region.

Next, click the "LED" button to draw a box around the region of the frame containing the light you are using for your alignment signal. Note that this ROI does not need to overlap with the pupil ROI, it will still be analyzed even if it is in the "cropped out" area. If you are not using an LED for alignment, you can ignore this step, but the program will not display an alignment signal without an LED ROI.

After you have drawn the pupil ROI, the cropped video frame will be displayed on the right hand of the GUI. Here you can add masking ROIs to help reduce the effect of glares and shadows on the analysis algorithm. For example, in this video, the dark region in the lower right corner of the cropped frame might be mislabeled as the pupil. To obscure it, we can click "Add Gray Masking ROI" and click to create a polygon around that area. When analyzing the video, all pixels in that area will be replaced with a neutral gray.

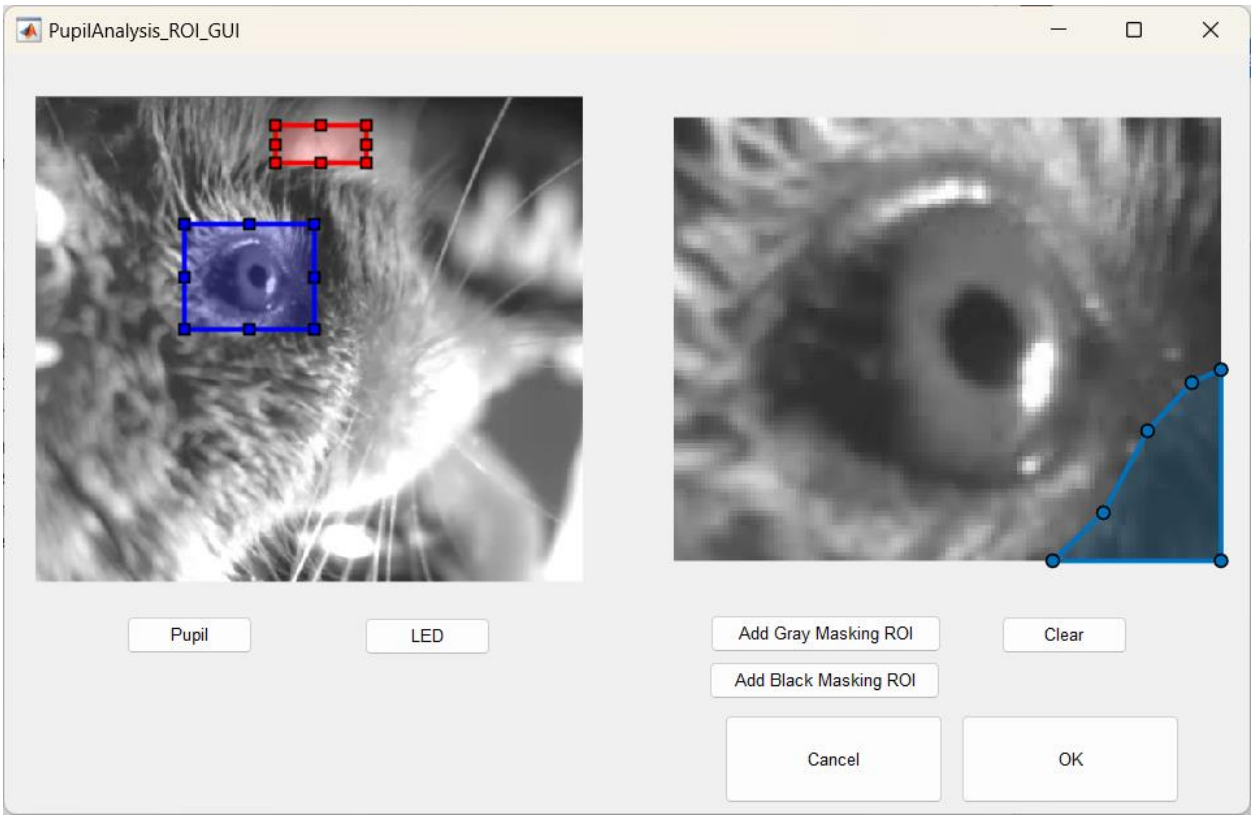

Figure 2-3: The ROI GUI With a Cropping ROI (blue, left), Alignment Signal ROI (red, left), and a Masking ROI (blue, right)

Clicking "Clear" will delete all masking ROIs. You can also right-click on individual ROIs and select "delete polygon" to delete individual ROIs.

Once you have drawn all ROIs, click "OK" to proceed. This will return you to the main GUI. The video axes will now display the cropped video, and the lower-left axes will display the video with the masks applied.

C. Set Algorithm Parameters

The next step is to use the controls in the "Adjustment Parameters" section of the GUI. There are four algorithm parameters you can adjust: *Threshold, Close, Open,* and *Radius Threshold*. The first three are preprocessing steps that are applied to the video frame before identifying putative pupil objects. Each time you adjust one of the parameters, the algorithm will be re-run on the current frame. To test your current parameters, you can play the video (using the video navigation panel) or step through frame-by-frame. Evaluate how well the algorithm tracks the pupil by observing how well the green ellipse matches the pupil in the top-left video axes.

i. Threshold

The first (and arguably most important) parameter is *Threshold*. This value ranges from 0 to 1 and corresponds to the pixel value at which to split the image into light and dark. Each frame is binarized such that all pixels below this value are converted to black and all pixels equal to or above this value are set to white. Then these values are inverted. The algorithm works by looking for white regions in

the binarized image that are approximately ellipsoid (the assumption being that the pupil is likely the darkest and roundest object in frame).

Use the threshold slider to adjust this value such that the pupil appears as a white region on the upper-right axes. Generally speaking, a threshold value of 0.25 to 0.35 works well, but this will depend heavily on your experimental setup and camera.

ii. Close and Open

The "Close" and "Open" parameters are used to smooth the binary image before looking for the pupil. They refer to the `imclose` and `imopen` functions in MATLAB. The parameter value specifies the size of the *structuring element* in either function, in pixels. Broadly speaking, *imclose* expands the white region of the binarized image to patch "holes," while *imopen* expands the black region of the image to remove small light patches. For lower-resolution videos, it is best to use small values for these parameters (less than five for both). High-resolution videos might require larger values for better performance.

iii. Radius Threshold

This parameter sets the lower bound for the pupil radius. If small shadows are labelled as the pupil, increase this value. This will exclude them from analysis. However, if this value is set too high, the pupil itself might be excluded, especially if it is particularly constricted.

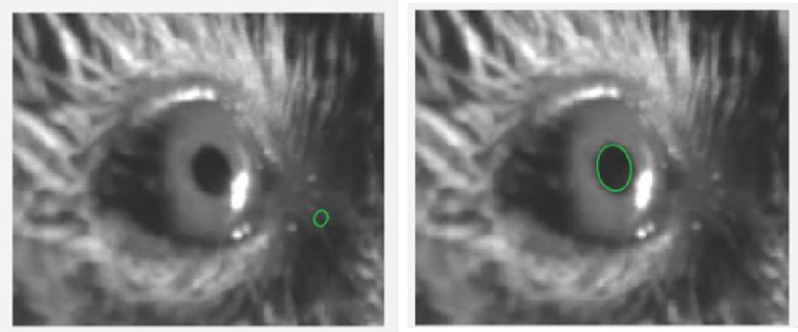

Figure 2-4: Left: A mislabeled frame due to the radius threshold being set too low. Right: The same frame correctly labeled after setting the radius threshold to exclude small objects

The pupil analysis algorithm always looks for the roundest object in the frame (after preprocessing the frame as described above). Very small patches of shadow can create objects that are only a few pixels across. After the *imopen* and *imclose* operations, these objects can be close to circular, and so will often be the roundest objects in frame. By setting the minimum radius, we can exclude these artifacts.

D. Evaluate Parameters

After you have set your initial parameter values, the next step is to test these parameters on a snippet of video.

Click the Play/Pause button to play through the video and observe how well the GUI tracks the pupil. When the algorithm is working correctly, the green ellipse will be drawn around the perimeter of the pupil on the vast majority of frames. If you find that the pupil is being mislabeled on certain a frame, you can pause the video there (or jump to it using the "Go to Frame" button) and adjust the parameters further.

Once you are satisfied with your parameters, the next step is to test them on a longer segment of the video. Click the "Process Sample" button, which will run the algorithm on one minute of the video, starting at the current frame. The algorithm takes several seconds to run, during which a loading bar will be displayed. After processing, the program will generate a summary figure:

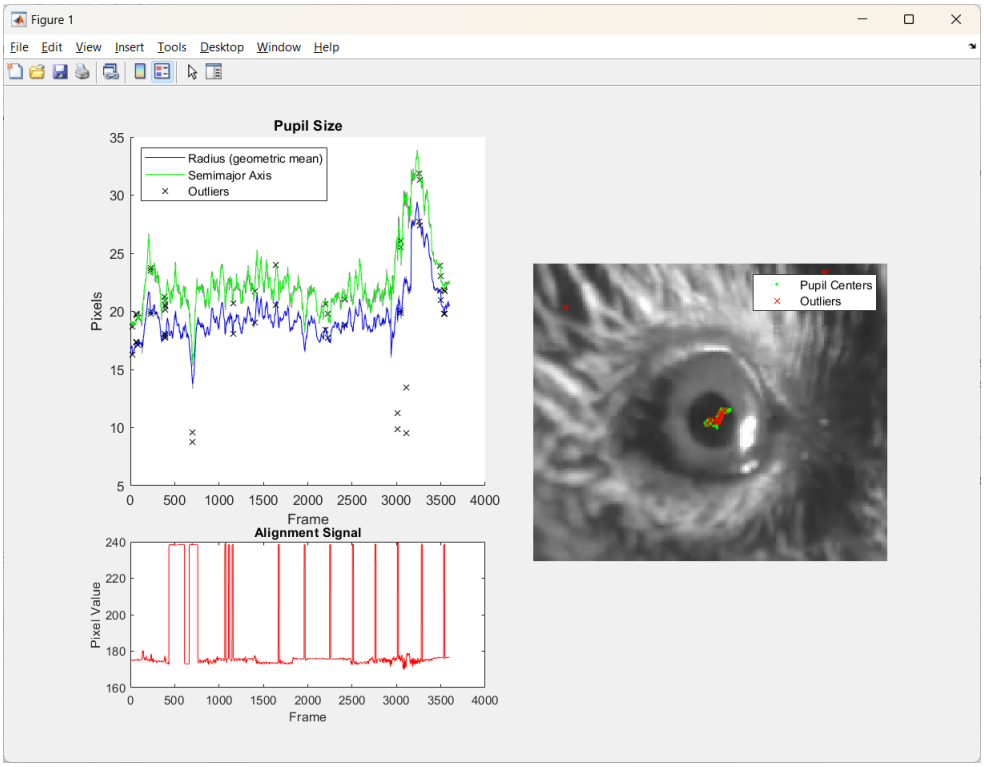

Figure 2-5: The summary figure generated by the Process Sample button. Top Left: Pupil radius (blue), semimajor axis (green), and excluded outlier values (black X). Bottom Left: Alignment signal. Right: Sample frame with pupil centers on each frame (green) and excluded outlier values (red X).

The upper-left plot displays the pupil size on each frame of the video. The pupil size is measured two ways: the *semimajor axis* of the ellipse, and the *radius*, which is defined as the geometric mean of the semimajor and semiminor axes of the pupil. The radius is the same value displayed in the GUI. For each metric, the raw measurement from each frame is plotted with a gray line and outlier values are plotted with a black X. A smoothed measurement with outliers omitted is plotted in color (semimajor axis in green, radius in blue).

The lower-left plot displays the alignment signal. The alignment signal is simply the mean value of each pixel contained in the alignment signal ROI on each vide frame. No frames are omitted, and there is no smoothing of the signal.

The middle-right axis displays a sample frame from the video and the center position of each of the fitted ellipses. Outlier values are plotted with a red X, and "good" values are plotted with a green point. The outliers are the same outlier points in the upper-left plot.

Outlier values are identified with the `isoutlier` function using a moving median method. Briefly, the algorithm looks at the pupil radius, the x coordinates of the pupil center, and the y coordinates of the center, over the course of one second of video. For each of these three vectors, any value that is at least four median absolute deviations away from the median of all values in that vector is flagged as an outlier. This will remove the occasional mislabeling of a shadow, sudden decreases in pupil size caused by blinks, and other similar artifacts. However, as this relies on a moving average method, it will struggle to identify mislabeling when there are several large movements in a row, such as caused by multiple blinks in rapid succession.

E. Process Video and Export Data

There are two ways to process an entire video.

i. Processing via the GUI

Clicking the "Process Video" button will run the algorithm on the entire duration of the video and generate a plot similar to that created by the "Process Sample" button. This is intended for analysis of shorter videos (less than 10 minutes), as longer videos will take a long time to process. Once the video is processed, the results will be saved in a mat file with to the same location as the input video and with a matching name. A second file with the parameters used to analyze the file will also be generated. For example, if the input video was named "myPupilVideo.mp4," the output data file will be named "myPupilVideo.mat" and the analysis parameters will be saved as "myPupilVideo_analysis_parameters.mat".

The data file has the following contents:

- `centroid`: a 2xn matrix of the position of the pupil on each of the *n* video frames. The first row of this matrix contains the pupil x coordinates; the second row contains the y coordinates.
- `radius`: a 1xn vector of the pupil radius on each of the *n* frames. As described above, the radius is the geometric mean of the semimajor and semiminor axes of the pupil ellipse. Note that this is *not* smoothed.
- `semimajorAxis`: a 1xn vector of the semimajor axis of the pupil.
- `isOutlier`: a 1xn logical vector indicating whether a frame was considered an outlier, *i.e.* mislabeled. True values indicate putative mislabeled frames.
- `radius_smoothed`: The pupil radius with outlier values removed, linearly interpolated, and smoothed over five frames with a moving mean.
- `semimajorAxis_smoothed`: The semimajor axis of the pupil with outliers removed, linearly interpolated, and smoothed over five frames with a moving mean.

The analysis parameter file has the following contents:

- `Threshold`: The pixel-value threshold used to binarize the video.
- `Min_Radius`: The minimum radius used to exclude small objects from analysis, in pixels.
- `Close`: The size of the structuring element used by the imclose function, in pixels.

- `Open`: The size of the structuring element used by the imopen function, in pixels.
- `PupilROI`: A four-element vector defining the position of the rectangular ROI used to crop the video. The vector has the form `[xmin, ymin, width, height]`.
- `IRROI`: A four-element vector defining the position of the rectangular ROI used to define the region of the video used to extract an alignment signal. So named because an infrared (IR) LED is usually used to generate an alignment signal.
- `Masks`: A cell array that defines the shape of the polygon of each gray masking ROI used in the analysis. If there are no masks used in the analysis, this will be empty array. Otherwise, each cell will contain a px2 matrix, where the p^{th} row defines the x and y coordinates of the p^{th} vertex of the ROI polygon.
- `Black_Masks`: A cell array defining the shape of the black masks. Identical in structure to the Masks variable described above.

ii. Processing via a Script

For longer videos, it can be more efficient and convenient to analyze them via a script. This way, you can run them overnight or on a server. To do so, first click the "Export Parameters" button, which will create a mat file and save your analysis parameters to it. This is identical to the parameters file described above. Then, you can use the `processPupil_wholevid` function to analyze that video.

The `processPupil_wholevid` function requires all of the analysis parameters stored in the parameter mat file, plus a `VideoReader` object corresponding to the video you want to analyze. Here is a simple example script to analyze a video named "myPupilVideo.mp4" using a parameter file named "myPupilVideo_analysis_parameters.mat"

```
videoPath = "path/to/video/myPupilVideo.mp4";
myVideoReader = VideoReader(videoPath);
parameterPath = "path/to/parameters/myPupilVideo_analysis_parameters.mat";
% Add the video reader object to the parameter struct array
analysisParameters = load(parameterPath);
analysisParameters.VR = myVideoReader;
outputData = processPupil_wholevid(analysisParameters);
```

You can write more elaborate scripts using the same function to analyze multiple videos in sequence.

3. The Analysis Algorithm in Detail

There are several steps to extracting the size and position of the pupil from a video frame. Steps A and B are performed by the `processPupilFrame_analysis` function. This function is called iteratively over each video frame. The last step, C, is performed by the `processPupil_postprocessing` function, which is called after processing the entire video.

A. Frame Preprocessing

i. Cropping and Masking

The video frame is cropped to the rectangular pupil ROI. Then masks are applied:

- Each pixel contained in a gray masking polygon have their values set to the median pixel value within the cropped frame
- Each pixel contained in a black masking polygon have their values set to 0 (darkest)

ii. Threshold and Invert

The image is binarized using the **`imthresh`** function and the *threshold* analysis parameter. The binarized image is then inverted such that the darkest regions in the frame are now white "objects" in the binary image.

```
im_bin = imbinarize(cropped_and_masked_frame,threshold_parameter);
im_bin = ~im_bin;
```

iii. Smooth with imopen and imclose

The white objects in the new binarized image are smoothed using the imclose and imopen functions, in that order. Both functions use a square structure element ("strel").

```
im_bin = imclose(im_bin,strel('square',close_parameter));
im_bin = imopen(im_bin,strel('square',open_parameter));
```

B. Identify Potential Pupil Objects

After binarizing the image, the **`regionprops`** function is used to identify regions of the frame that might correspond to the pupil.

i. Extract Features Using regionprops

`regionprops` is a built-in MATLAB function that identifies objects in a binary image (white regions on a black background) and computes various features about each region.

```
RP = regionprops(im_bin,'Centroid','MajorAxisLength','MinorAxisLength',...
                        'Circularity');
```

The resulting struct array RP will have a row for each of the white blobs in the binary image. By requesting these four parameters, we are specifying that the function should fit an ellipse to each of these blobs and return the resulting measurements.

ii. II: Use Features to Identify the Pupil

We use the major and minor axis lengths to compute a radius of each object (geometric mean). Then we exclude all objects with a radius less than the *radius threshold* analysis parameter.

```
all_radii = sqrt([RP.MajorAxisLength] .* [RP.MinorAxisLength]);
RP(all_radii < radius_threshold) = [];
```

Next, we look for the "roundest" object remaining, the object with the highest "circularity" value. We consider this to be the pupil on the current frame.

```
[~,pupilIndex] = max([RP.Circularity]);
RP = RP(I);
semimajorAxis = RP.MajorAxisLength;
radius = sqrt(RP.MinorAxisLength * RP.MajorAxisLength);
centroid = RP.Centroid;
```

The semimajor axis, radius, and centroid of this pupil object is saved. The above process is run on every frame in the video, resulting in a vector each for the radius, semimajor axis length, x position, and y position of the pupil on each frame. These vectors are saved in the output file, and can be used in analysis directly. The next step, smoothing, is not required, but is recommended.

C. Remove Outliers and Smooth Data

After processing the entire video (or video segment), the resulting pupil size and position vectors are cleaned and smoothed. The goal of this step is to reduce the impact of the occasional mislabeled frame, which can result from a mouse blinking or other large movements.

The following process is done using the custom functions **`processPupil_postprocessing`** and **`processPupil_findOutliers`**.

i. Identify Outlier Frames

First, we look for "outlier" frames, frames where any of the pupil radius, x position, or y position are dramatically different from those of its neighbors. In other words, we look for frames where the pupil changed size or position unexpectedly quickly. Such changes are almost always the result of a non-pupil object being picked up by the algorithm incorrectly.

To find these outliers, we first combine each of these three vectors into a single matrix, with each vector comprising the columns of that matrix. Then we use the built-in MATLAB function `isoutlier` to label outlier values. We consider a frame where *any* of the radius, x, or y position values are outliers to be an "outlier frame."

The default analysis algorithm uses a moving median method with a window corresponding to one second of video, or 60 frames, whichever is longer. The threshold for an outlier value is four median absolute deviations (MAD). The window size and MAD threshold can be specified in the **`processPupil_postprocessing`** function if the defaults to not work well for your data.

```
centroid = [x_position ; y_position];
outlierWindow = max(framerate, 60); % 1 second of video, min of 60 frames
outlier_test_mat = [centroid',radius'];
thresholdFactor = 4;
outlierBoolMat = isoutlier(outlier_test_mat,'movmedian',outlierWindow,...
    'ThresholdFactor',thresholdFactor);
outlierBool = any(outlierBoolMat,2);
```

The resulting **`outlierBool`** vector is a logical vector where "true" values indicate that the frame at that index is an outlier. It is returned as part of the algorithm output.

ii. Interpolate and Smooth

Next, we clean and smooth the pupil size vectors (radius and semimajor axis). Note that these smoothed vectors are saved *in addition* to the "raw," unsmoothed vectors.

First we remove outlier frames by replacing all values indicated by the **`outlierBool`** vector with an empty `NaN` value. Then we use MATLAB's built-in `fillmissing` function to linearly interpolate over those missing values. The code below describes how this is done on the radius vector, the same is also applied to the semimajor axis vector.

```
radius(outlierBool) = nan;
radius = fillmissing(currVar,"linear");
```

After interpolating, we smooth with a moving mean over five frames. This smooth window can also be specified in the `processPupil_postprocessing` function.

```
smoothWindow = 5;
radius_smoothed = movmean(radius,smoothWindow);
```

And with that we are done!