

## DUGR GUI

This document describes how the DUGR GUI, currently under development, could be used to determine the extended UGR value according to CIE 232:2019.

The GUI is still evolving, neither the process nor the frontend is final. Bugs and errors may occur.

### 1. Choose whether the perspective distortion should be corrected or not

From v\_0.4 onwards 2 algorithms are implemented to calculate the DUGR value.

One that performs perspective correction and one that works on distorted images. You can use the checkbox to swap between both algorithms.

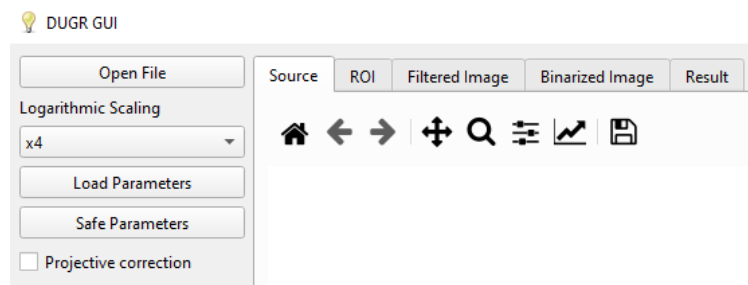


Figure 1: Checkbox for projective correction and "open file" button

### 2. Load a source file from the luminaire which is under evaluation

By using the "Open File" Button you will be able to select an image from the windows explorer. Currently supported image file formats are \*.pf files and pixel information stored in \*.txt files.

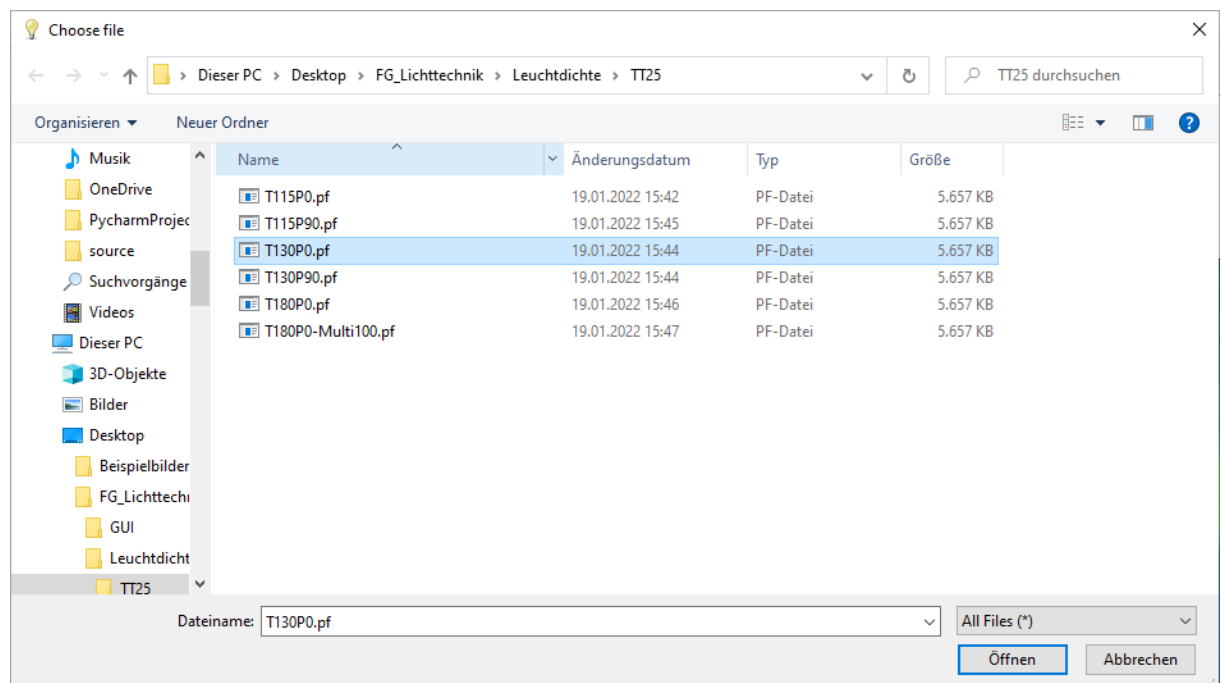


Figure 2: Loading a source file (\*.pf Image)

After loading an image the GUI Interface should look somewhat equal to figure 3.

(Depending on which algorithm is used the sidebar and tabs are different)

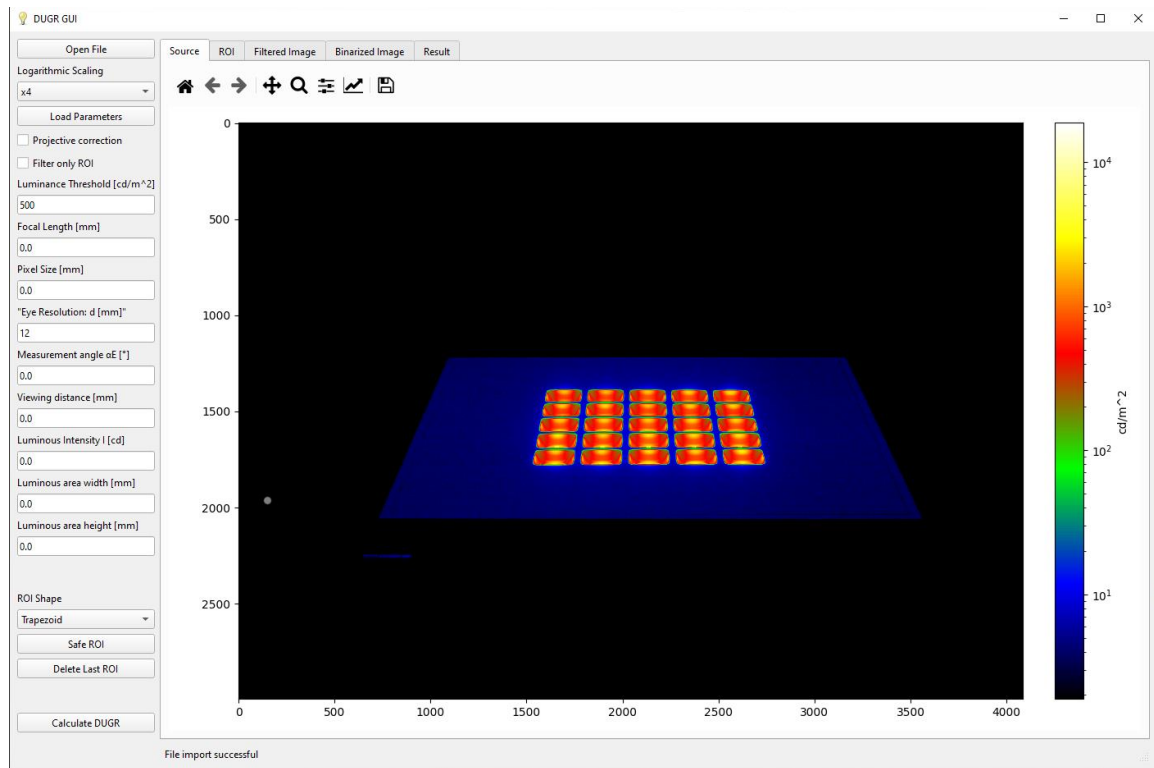


Figure 3: GUI Interface Source Tab projective distorted algorithm

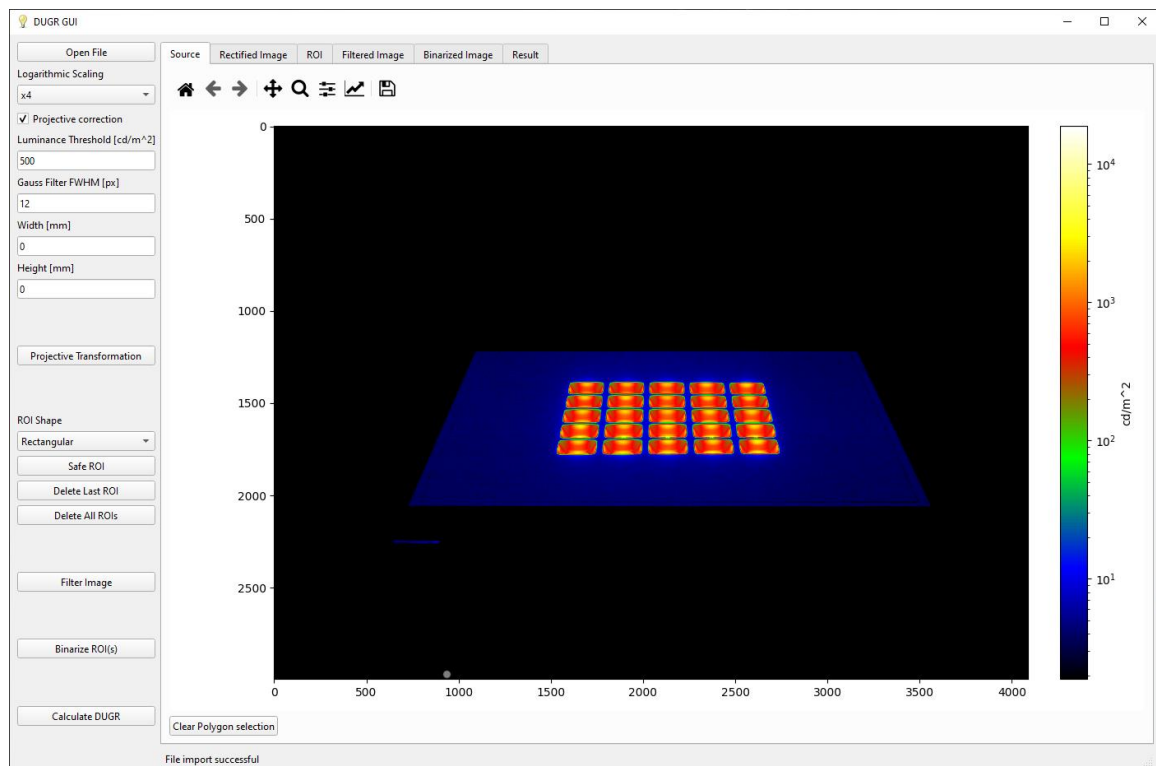


Figure 4: GUI Interface Source Tab projective corrected algorithm

As of version 1.0, a colormap has been implemented as in the LabSoft software from TechnoTeam.

In addition, it is possible from version 1.0 to display different logarithmic scalings of the image to possibly ensure a better contrast.

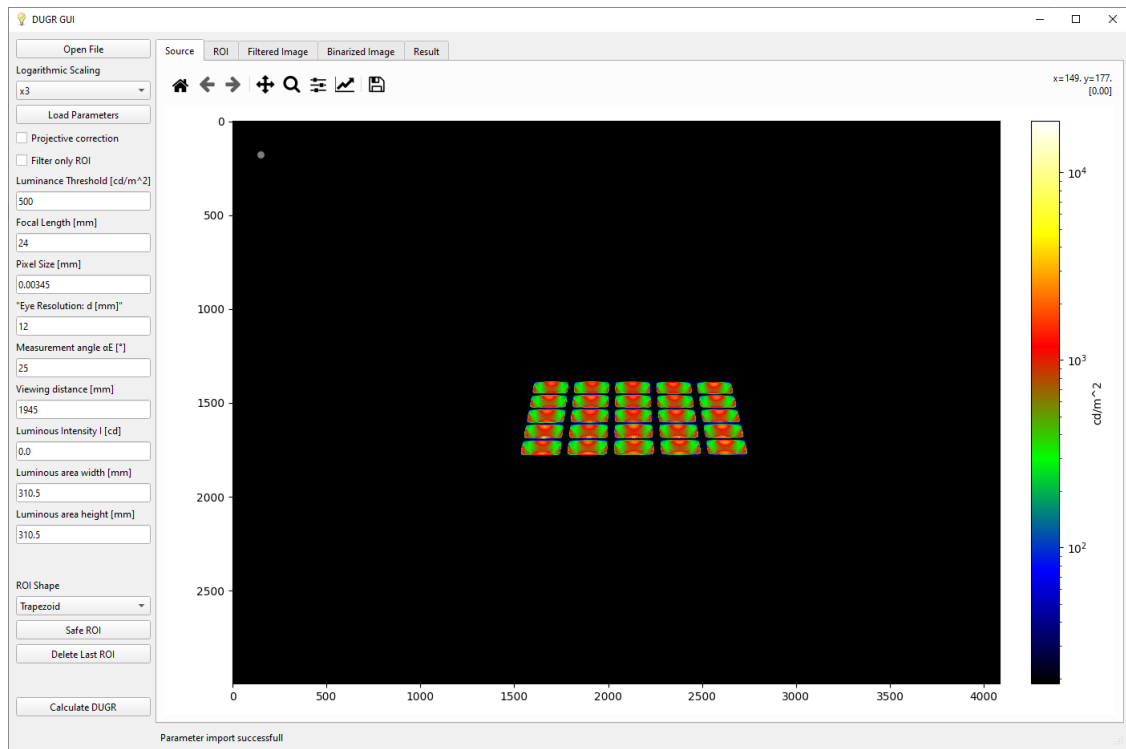


Figure 5: Log Scaling: x3

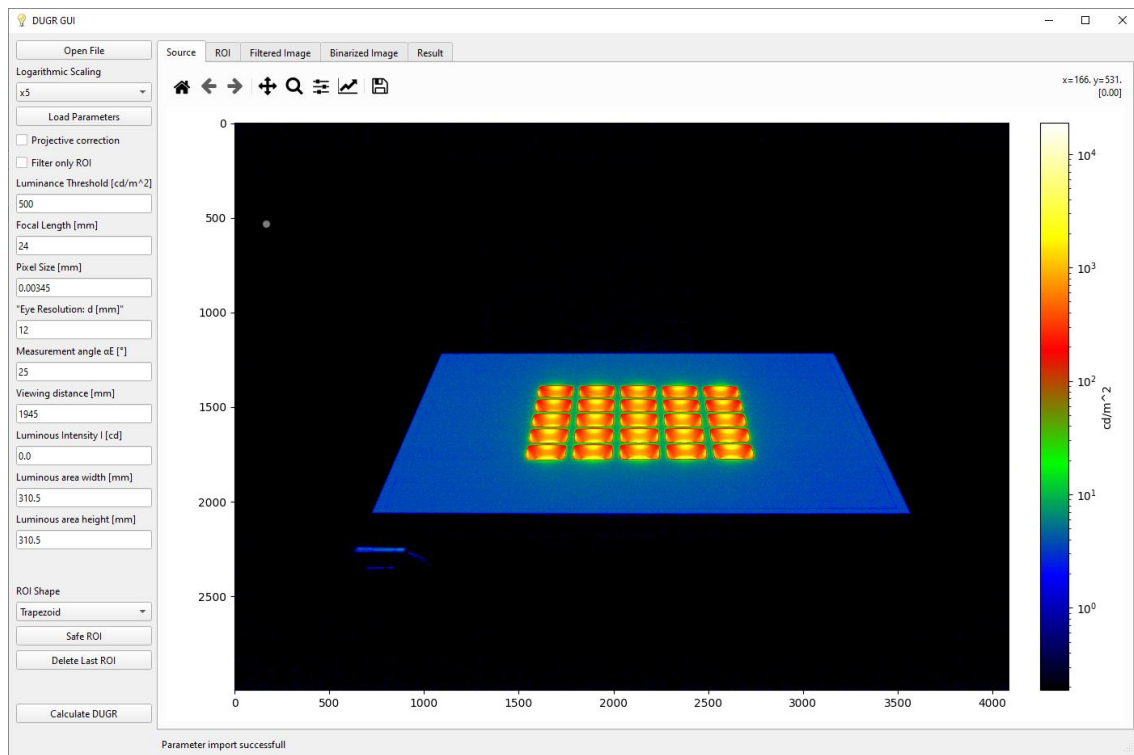
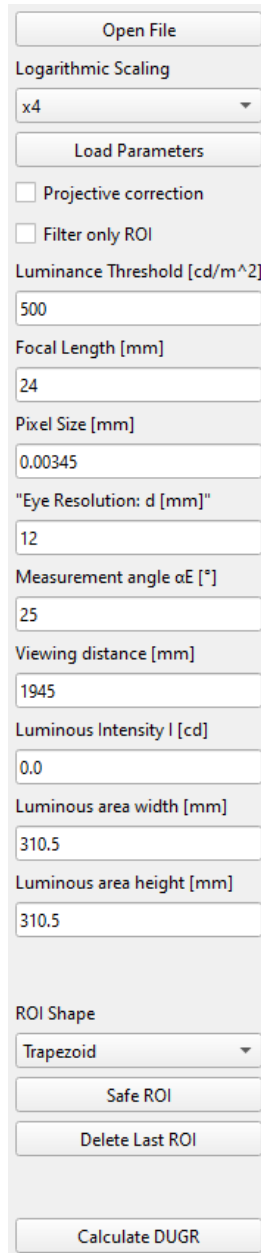


Figure 6: Log Scaling: x5

### 3. Projective distorted algorithm

The sidebar of the projective distorted algorithm has 8 parameters which have to be filled before executing the calculation. **The Luminous Intensity is optional** and can be used to calculate a second result if given.



The sidebar contains the following controls from top to bottom:

- Open File button
- Logarithmic Scaling dropdown menu (set to x4)
- Load Parameters button
- Projective correction checkbox (unchecked)
- Filter only ROI checkbox (unchecked)
- Luminance Threshold [cd/m<sup>2</sup>] input field (500)
- Focal Length [mm] input field (24)
- Pixel Size [mm] input field (0.00345)
- "Eye Resolution: d [mm]" input field (12)
- Measurement angle αE [°] input field (25)
- Viewing distance [mm] input field (1945)
- Luminous Intensity I [cd] input field (0.0)
- Luminous area width [mm] input field (310.5)
- Luminous area height [mm] input field (310.5)
- ROI Shape dropdown menu (Trapezoid)
- Safe ROI button
- Delete Last ROI button
- Calculate DUGR button

- **Luminance Threshold:** Threshold value to determine which pixels are assigned to the effective area and the effective luminance
- **Focal Length:** The focal length of the optical system (the lens). Together with pixel size this value is used to calculate the solid angles of each pixel. Therefore, the parameter should be the distance between the entrance pupil and the sensor center. However, using the focal length of the lens seems to be an acceptable approximation for the time being.
- **Pixel Size:** This parameter is the actual size of the pixels on the sensor (Pixel pitch). As said before this is used to calculate the pixel solid angles.
- **Eye resolution:** Minimum feature diameter based on the resolution of the human eye. This parameter defaults to 12mm (As proposed in CIE 232:2019)
- **Measurement angle:** Measuring angle with respect to the horizontal
- **Viewing distance:** Distance between the measuring device and the center of the luminaire. (In an ideal scenario this distance would also be measured from the entrance pupil)
- **Luminous Intensity:** This parameter is optional and can be used to generate a second result if you have determined the luminous intensity under the measuring angle. First measurements with this parameter have shown that luminous intensities under the relevant measuring angles are mostly error-prone due to the magnitude.
- **Luminous area width & height:** Indication of the extent of the luminous area. (This can be obtained from a \*.ldt file). Please note that height and width are the extensions based on the image (height is corresponding to the y axis and width is corresponding to the x axis)!

From v\_0.6 onwards it's possible to load the parameters from a predefined \*.json file. (This can be useful if you have several measurements with the same camera).

Afterwards, the button „Calculate DUGR“ can be used to start the calculation. The calculation steps of this algorithm are explained below.

If you want to load the 9 parameters from a preset \*.json file, the file has to look like the following:

```
{
  "lum_th": 500,
  "focal_length": 24,
  "pixel_size": 0.00345,
  "d": 12,
  "viewing_angle": 25,
  "viewing_distance": 1945,
  "luminaire_width": 290,
  "luminaire_height": 290
}
```

Figure 7: \*.json file for parameter import

#### 4. Calculation steps of the projective distorted algorithm

In order to execute the DUGR calculation with the projective distorted algorithm, the first step has to be drawing a region containing the luminaire onto the image.

For a more precise drawing you can use the zoom function (magnifying glass icon).

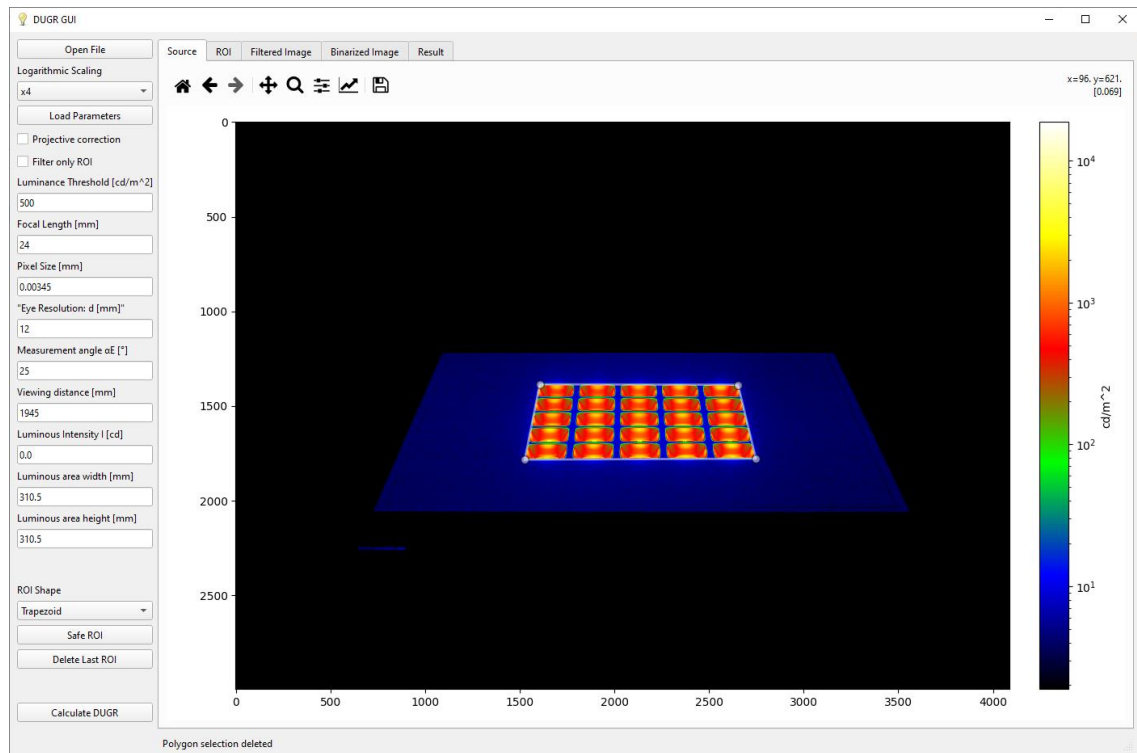


Figure 8: Drawing the ROI onto the plot (projective distorted algorithm)

If the Region is drawn onto the image you can use the button safe ROI.

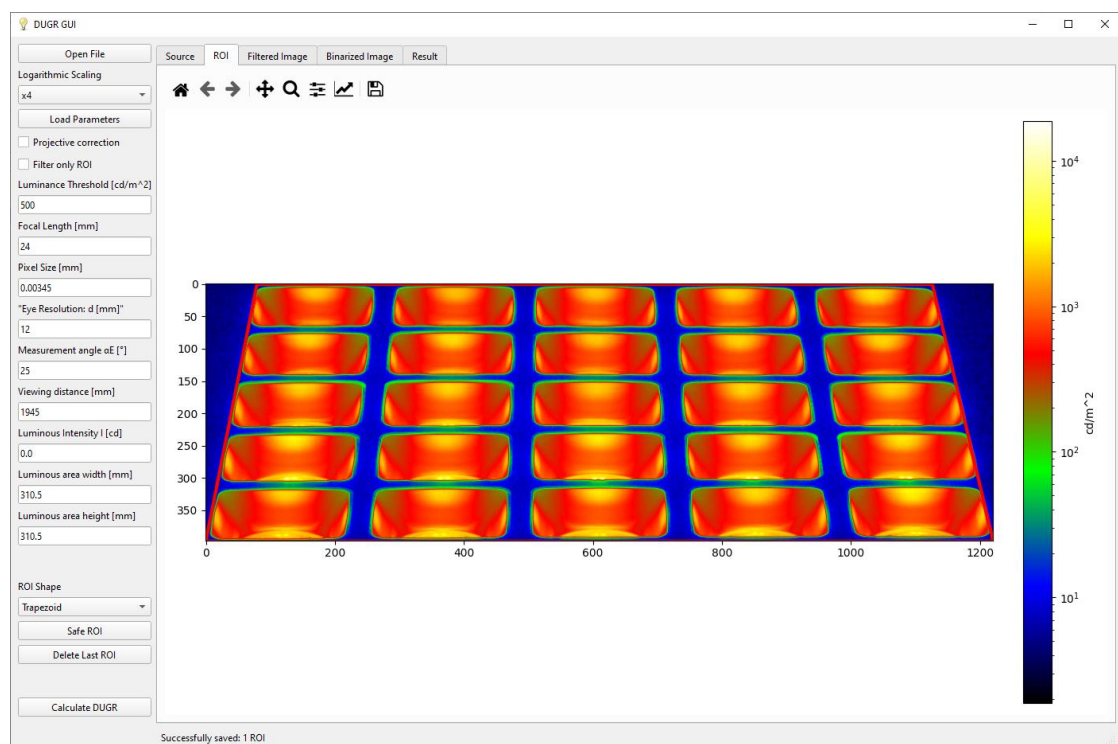


Figure 9: ROI tab after saving the ROI drawn before

From v\_0.7 onwards you can decide to either filter and binarize the whole image or only a window around the drawn ROI. The window is rectangular and half the filter width bigger than the biggest extension of the ROI.

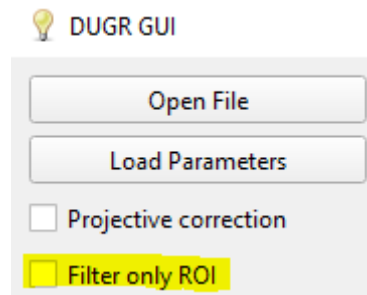


Figure 10: Filter only ROI checkbox

After saving the ROI and loading or filling out the parameters you can use the button “Calculate DUGR”.

In the tabs “Filtered Image” and “Binarized Image” you can check the result of the filtering and binarization operation after the calculation is completed.

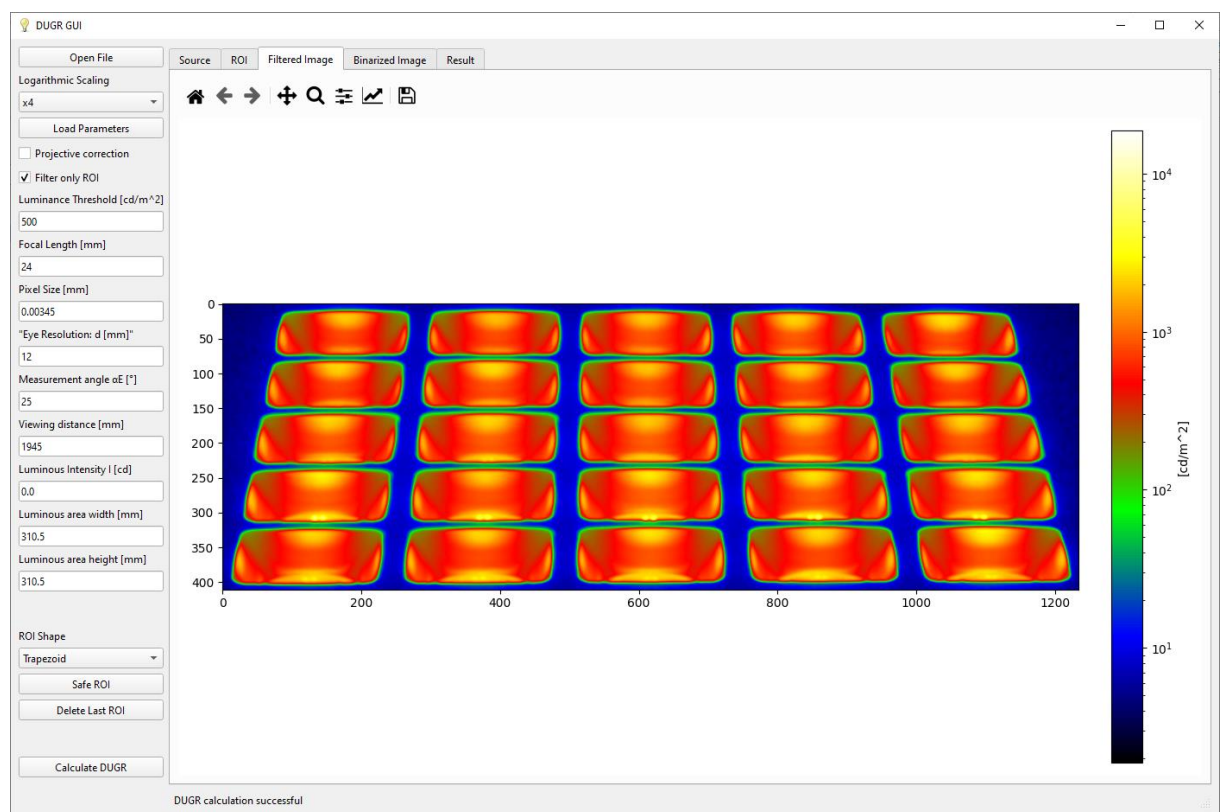


Figure 11: Filtered Image Tab

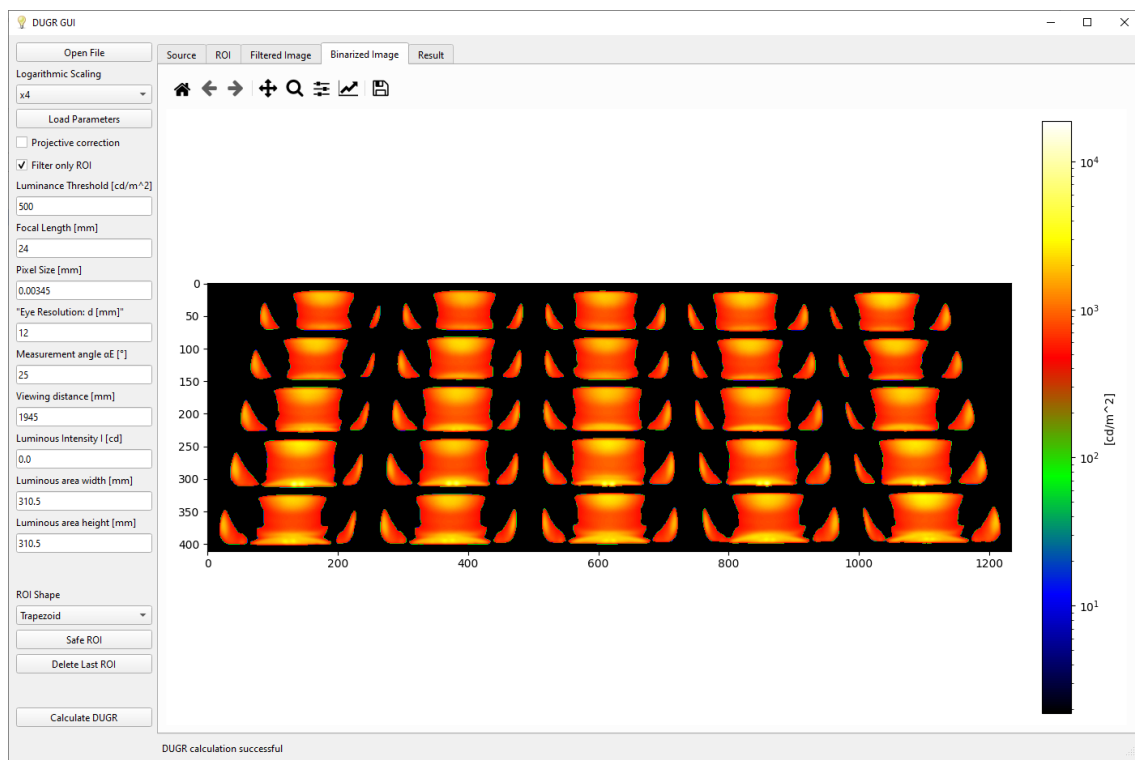


Figure 12: Binarized Image Tab

## 5. Result tab

After the Calculation has finished, the result tab should look similar to this:

The screenshot shows the DUGR GUI with the 'Result' tab selected. The table displays the following parameters and values:

DUGR_I	0.0
k <sup>2</sup> _I	0.0
A_p_new_I	0 [mm <sup>2</sup> ]
DUGR_L	0.9
k <sup>2</sup> _L	1.3
A_p_new_L	31584 [mm <sup>2</sup> ]
A_p	40745 [mm <sup>2</sup> ]
A_eff	13987 [mm <sup>2</sup> ]
Effective luminance	1001.08 [cd/m <sup>2</sup> ]
Mean Luminaire luminance	556.18 [cd/m <sup>2</sup> ]
Effective solid angle	0.003697 [sr]
Luminaire solid angle	0.009285 [sr]
Measurement angle αE	25.0 [°]
Viewing distance	1945.0 [mm]
Luminous area width	310.5 [mm]
Luminous area height	310.5 [mm]
Luminous intensity	0.0 [cd]
lum_th	500 [cd/m <sup>2</sup> ]
d	12 [mm]
Calculated optical resolution	0.00822 [°/px]
FWHM	4.01 [px]
Filter width	13 [px]
Filter sigma	1.701 [px]
rb min	2086.74 [mm]
ro min	0.03295 [°/px]

The status bar at the bottom indicates 'DUGR calculation successful'.

Figure 13: Result Tab

Please note that the parameters indexed with I are the ones calculated if luminous intensity is given. If it's not all those values default to 0.

### **Calculation of the parameters:**

By using the effective luminance the effective solid angle, as well as the mean luminance of the whole luminaire and the solid angle of the whole luminaire, the  $k^2$  parameter can be calculated as follows:

$$k_L^2 = \frac{L_{eff} \cdot \omega_{eff}}{L_s \cdot \omega}$$

Where the effective luminance  $L_{eff}$  is the mean value of all of the pixels above the luminance threshold and  $L_s$  the mean value of all of the pixels which are part of the luminaire.

$$A_{eff} = \omega_{eff} \cdot (viewing\ distance)^2$$

$$A_{p,L} = A_{luminous\ area} \cdot \cos(90^\circ - viewing\ angle)$$

$$A_{P_{new,L}} = \frac{A_{p,L}}{k_L^2}$$

Values calculated from the luminous intensity:

$$A_{P_{new,I}} = \frac{I^2}{L_{eff}^2 \cdot A_{eff}}$$

$$k_I^2 = \frac{A_p}{A_{P_{new,I}}}$$

The DUGR value can then be calculated using the following formula:

$$DUGR = 8 \cdot \log_{10}(k^2)$$

The calculation of the “solid angle image”, which is used for the calculation of effective and total solid angle of the luminaire is based on the procedure used by the labsoft for standard UGR estimation.

The results can be exported to a \*.pdf file. If the „Show all evaluation“ checkbox is selected all of the generated images during calculation are exported as well.



## 6. Perspective correction algorithm

For the perspective correction algorithm the GUI currently uses a transformation matrix calculated based on 4 points and the OpenCV function: *getPerspectiveTransform*

And mapped with the help of the OpenCV function: *warpPerspective*

([OpenCV: Geometric Image Transformations](#))

Currently the transformation is mapped to the width and height of the luminaire. For now, the image is downsampled to achieve a resolution of 1 mm/px. (Simplifies the process and shortens calculation time).

So in order to do the projective transformation the width and the height of the luminaire have to be entered and the four boundary points of the luminaire have to be drawn onto the previously imported source image (By clicking into the plot).

The status bar shows now if the required resolution is met by calculation the mm/px resolution based on the input.

By pressing the „Projective Transformation“ button the rectification is executed.

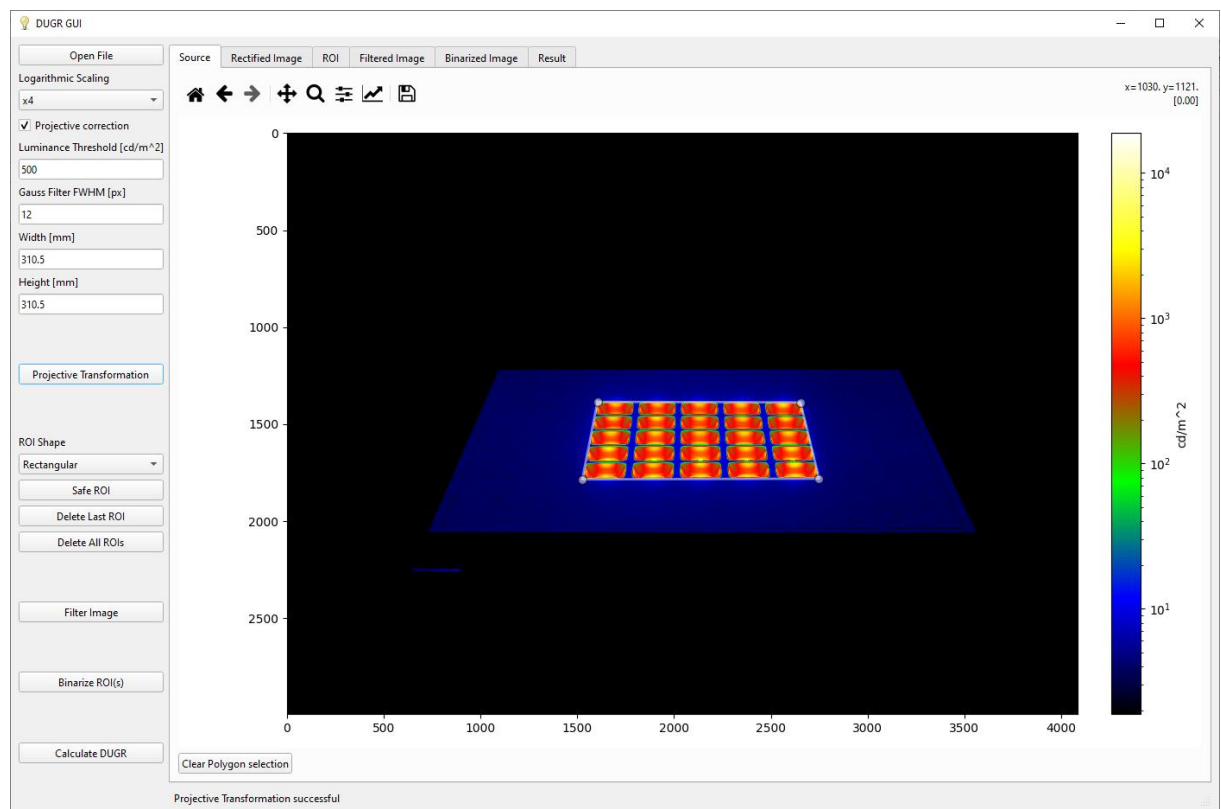


Figure 14: Parameters and Polygon for the projective Transformation

The result of the projective transformation can then be analyzed in the „Rectified Image“ Tab.

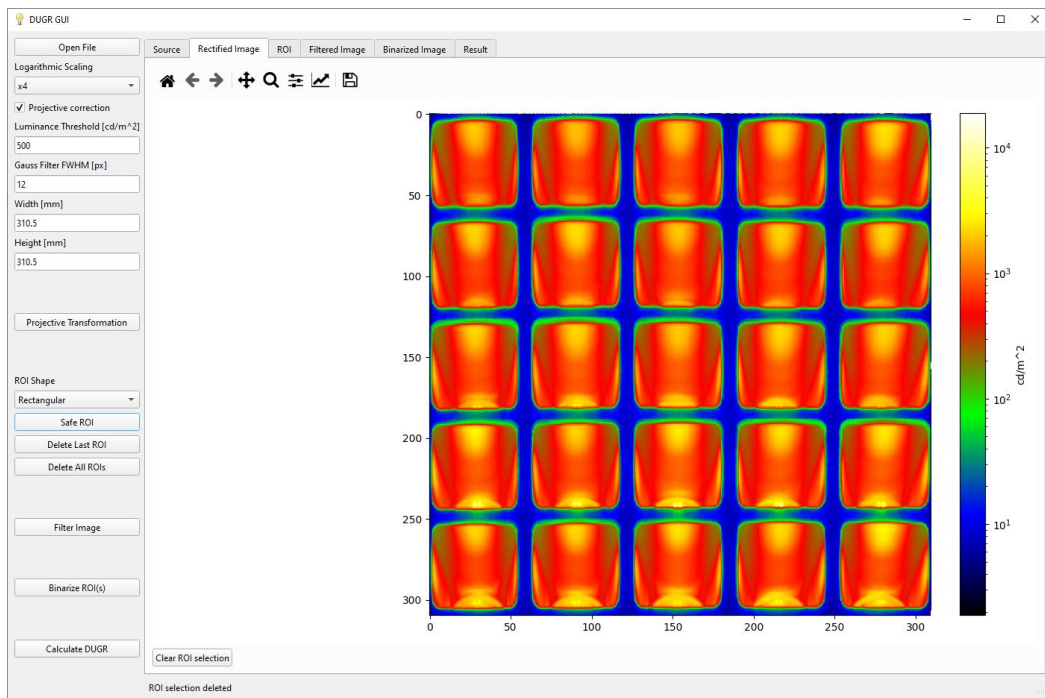


Figure 15: Image after projective transformation (Rectified Image Tab)

## 7. Draw ROIs

It is now possible to decide between two types of ROIs Rectangular and Circular. After selecting the required type the ROI can be drawn by clicking and dragging in the rectified image

In our example, the luminaire has two light emitting surfaces, so in the next step we draw two rectangle ROIs by clicking and dragging in the rectified image. After placing a rectangular ROI it can be saved by pressing the button „Safe ROI“.

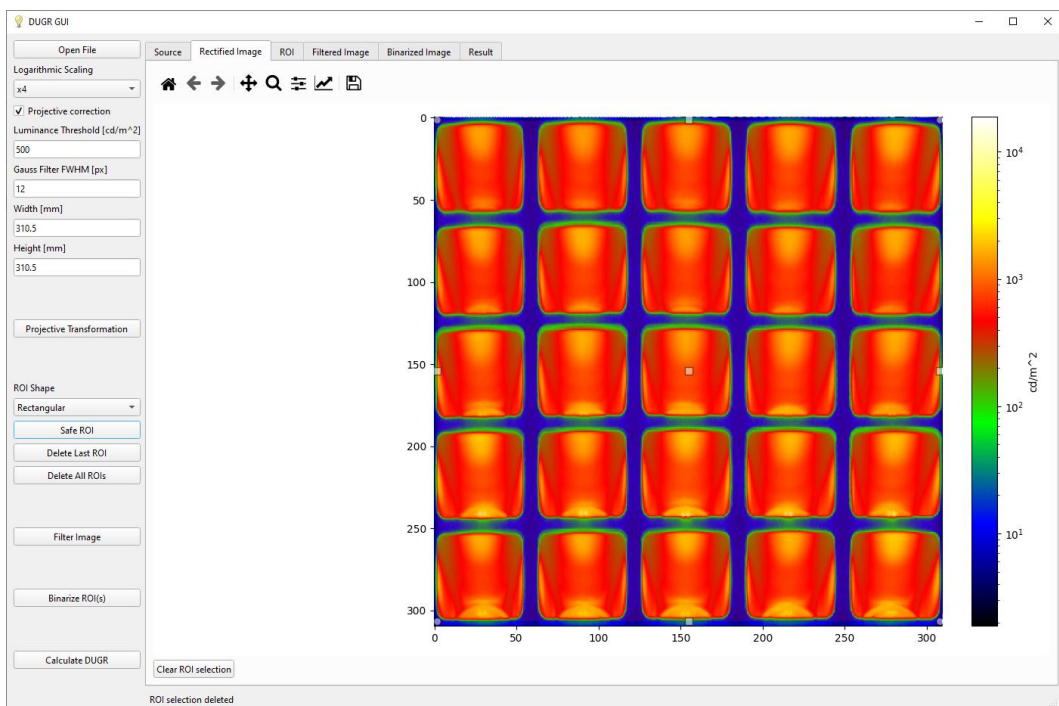


Figure 16: Drawing and saving a rectangular ROI

Each saved ROI can be viewed in the ROI tab. In addition, it is possible to delete either the last ROI or all ROIs via the respective buttons.

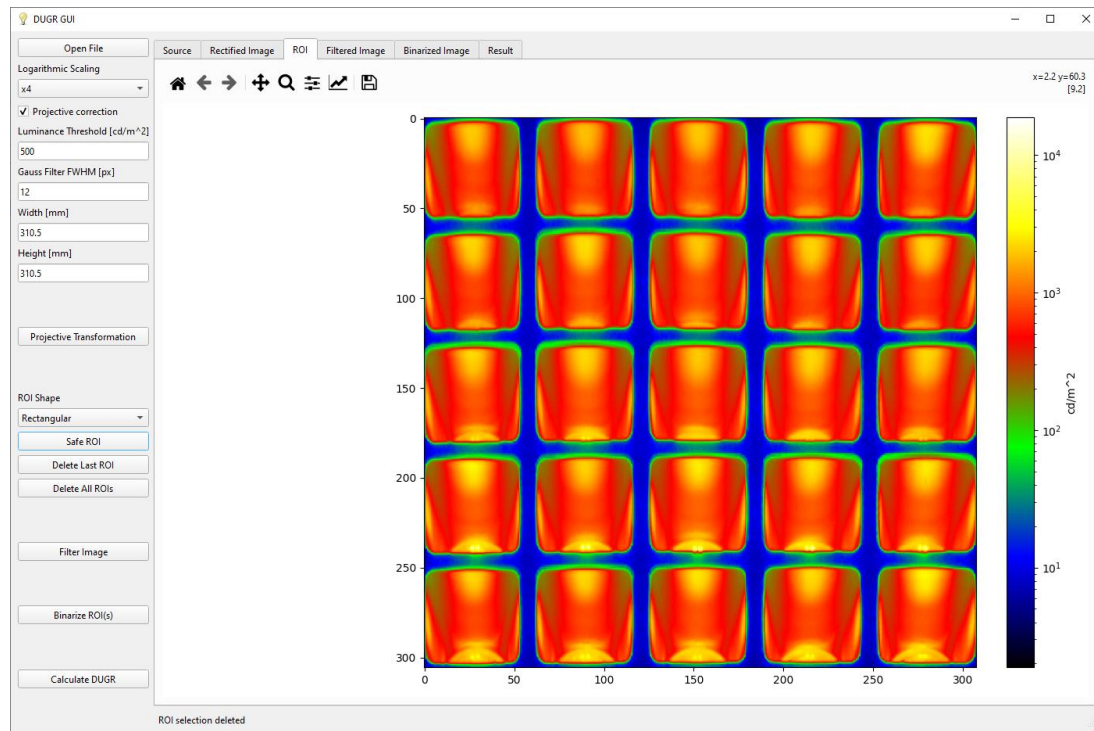


Figure 17: Viewing the ROIs in the ROI Tab

#### 8. Filter the image.

To account for human visual acuity, the image is filtered using a Gaussian filter. Since the image is scaled to a resolution of 1mm/px the default value for the full width at half maximum (FWHM) is 12mm, as suggested in CIE 232:2019. But it can be changed in the textbox.

Once the ROIs are determined and a full width at half maximum is defined, filtering can be performed via the filter image button.

Standard deviation ( $\sigma$ ) and filter width are calculated according to formulas 12, 13 and 14 from CIE 232:2019.

To ensure that no luminance information is lost during filtering, a projective transformed image of the luminaire is generated with a frame of half the filter width.

This is done by calculating factors for the luminaire dimensions based on the input for luminaire width and height and the previously drawn polygon.

With the help of these factors new polygon points are calculated and the resulting polygon is projectively transformed.

```

tw = tr[0] - tl[0] # Top width of the polygon (x - Dimension)
bw = br[0] - bl[0] # Bottom width of the polygon (x - Dimension)
rh = br[1] - tr[1] # Right side height of the polygon (y - Dimension)
lh = bl[1] - tl[1] # Left side height of the polygon (y - Dimension)

tw_factor = tw / luminaire_width # Top width factor
bw_factor = bw / luminaire_width # Bottom width factor

rh_factor = rh / luminaire_height # Right side height factor
lh_factor = lh / luminaire_height # Left side height factor

new_tw = (luminaire_width + 2 * border_size) * tw_factor # New top width based on factor and border size
new_bw = (luminaire_width + 2 * border_size) * bw_factor # New bottom width based on factor and border size
new_rh = (luminaire_height + 2 * border_size) * rh_factor # New right side height based on factor and border size
new_lh = (luminaire_height + 2 * border_size) * lh_factor # New left side height based on factor and border size

tl[0] = round(tl[0] - (new_tw - tw) / 2) # New top left x - coordinate
tl[1] = round(tl[1] - (new_lh - lh) / 2) # New top left y - coordinate

tr[0] = round(tr[0] + (new_tw - tw) / 2) # New top right x - coordinate
tr[1] = round(tr[1] - (new_rh - rh) / 2) # New top right y - coordinate

br[0] = round(br[0] + (new_bw - bw) / 2) # New bottom right x - coordinate
br[1] = round(br[1] + (new_rh - rh) / 2) # New bottom right y - coordinate

bl[0] = round(bl[0] - (new_bw - bw) / 2) # New bottom left x - coordinate
bl[1] = round(bl[1] + (new_lh - lh) / 2) # New bottom left y - coordinate

```

Figure 18: Code Snippet for the calculation of the projective transformed image with borders

The result is a rectified image of the luminaire with the calculated border size and can be analyzed in the Filtered Image Tab.

The previously selected ROIs are marked by red rectangles in the resulting filtered image.

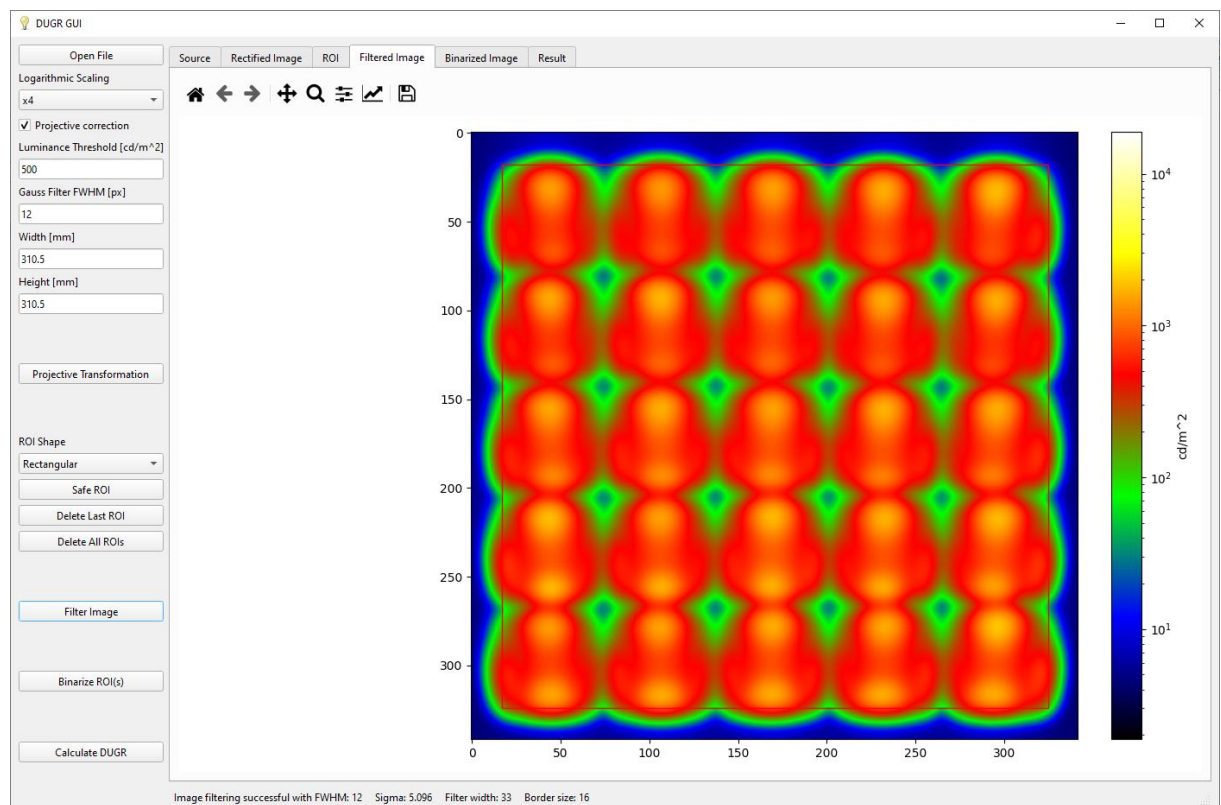


Figure 19: Gauss filtered image (Filtered Image Tab)

## 9. Thresholding the ROIs

In the next step, all pixels that are below the set threshold, in the previously defined ROIs, are set equal to 0. The default value for the threshold is  $500 \frac{cd}{m^2}$  as suggested in CIE 232:2019.

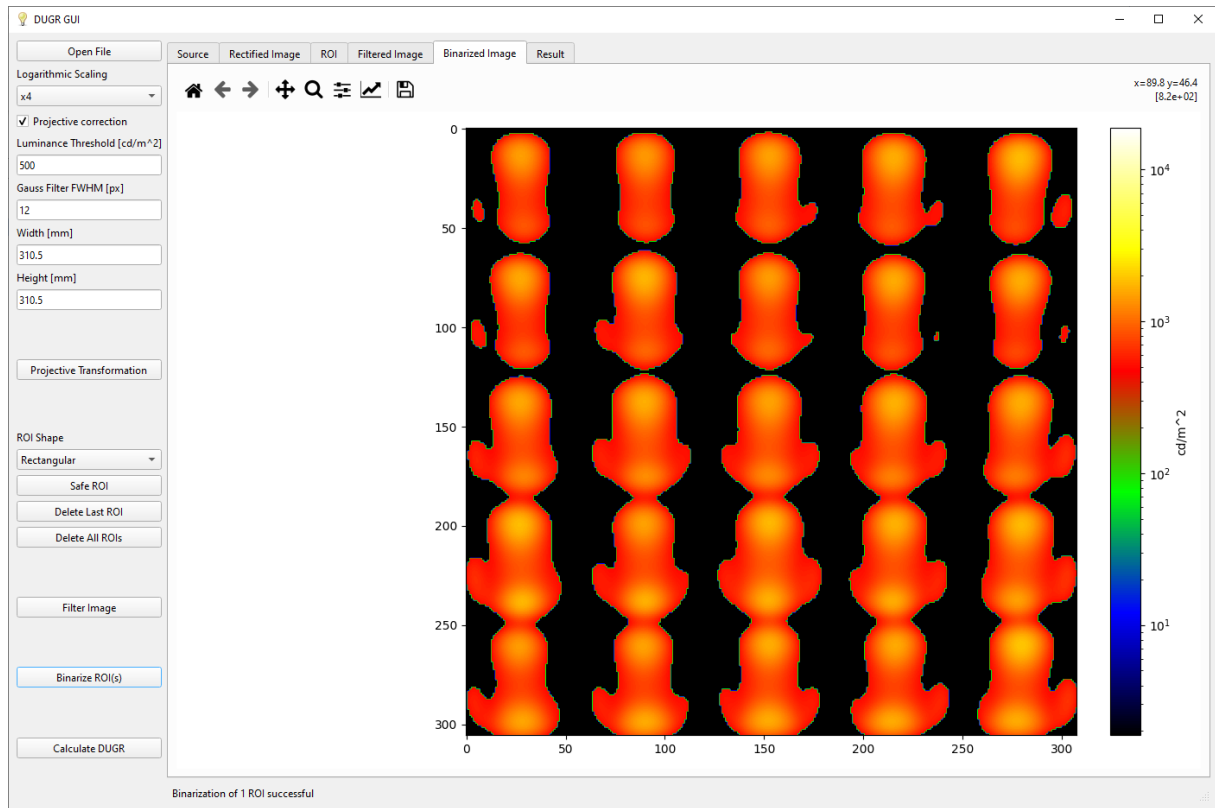


Figure 20: Binarized / Thresholded ROIs (Binarized Image Tab)

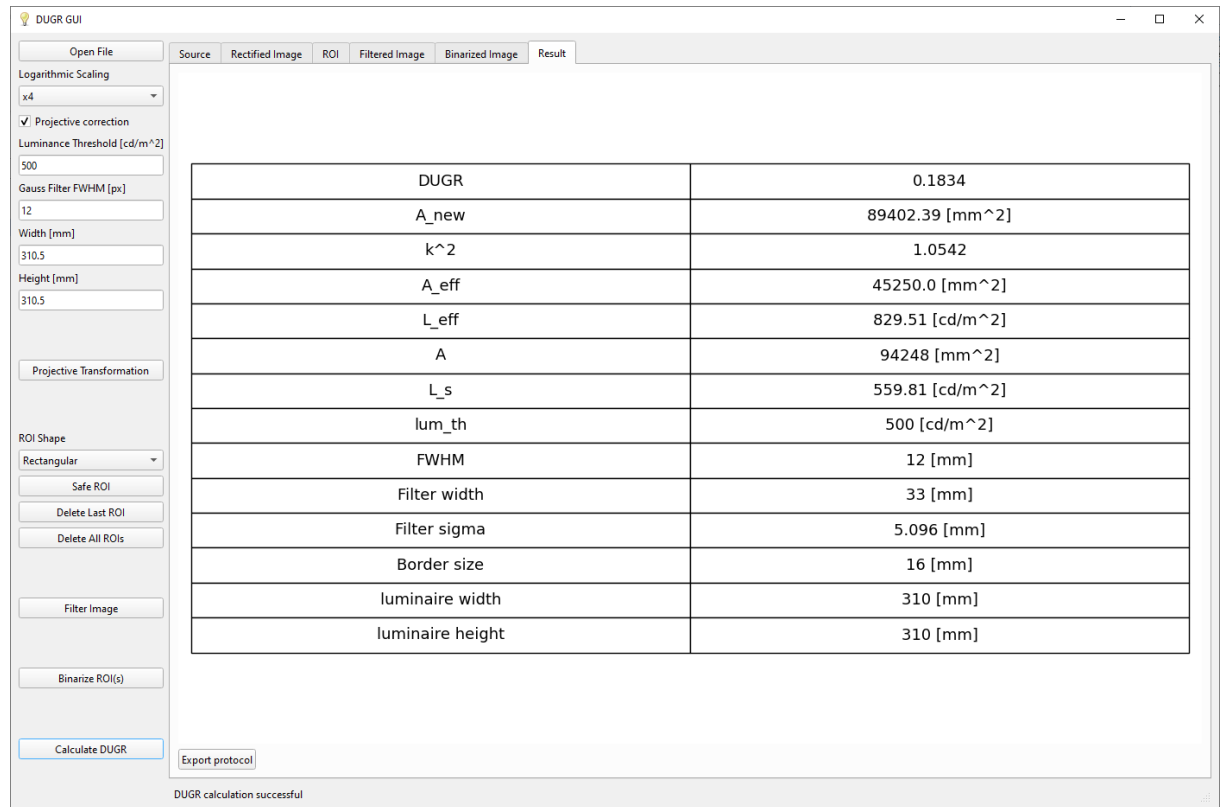
## 10. Calculating the results

After binarizing / thresholding the ROIs the „Calculate DUGR“ button can be pressed to calculate the results.

The results can be seen in a tabular form in the Result tab.

From v\_0.4 onwards it's possible to export the protocol as \*.pdf file. The file contains the evaluation images and the result table.

It the export can be done via the button or the shortcut CTRL+S



The screenshot shows the DUGR GUI with the 'Result' tab selected. The table displays the following data:

DUGR	0.1834
A <sub>new</sub>	89402.39 [mm <sup>2</sup> ]
k <sup>2</sup>	1.0542
A <sub>eff</sub>	45250.0 [mm <sup>2</sup> ]
L <sub>eff</sub>	829.51 [cd/m <sup>2</sup> ]
A	94248 [mm <sup>2</sup> ]
L <sub>s</sub>	559.81 [cd/m <sup>2</sup> ]
lum <sub>th</sub>	500 [cd/m <sup>2</sup> ]
FWHM	12 [mm]
Filter width	33 [mm]
Filter sigma	5.096 [mm]
Border size	16 [mm]
luminaire width	310 [mm]
luminaire height	310 [mm]

Below the table, there is an 'Export protocol' button. The status bar at the bottom indicates 'DUGR calculation successful'.

Figure 21: Result table in the Result tab

### Further Notes:

#### Shortcuts:

Open File	CTRL + O
Load Parameters	CTRL + P
Reset Polygon Selection (Polygon drawing) & Reset Rectangle Selection (ROI drawing)	Escape
Save protocol *.pdf file	CTRL+S



## Changelog

### [v 0.1 --> v 0.2](#)

#### New Functions:

- Check of the required resolution (1.2[mm/px]) based on input dimensions and the image
- Possibility to draw circular ROIs on the rectified image to calculate DUGR of luminaires which are not Rectangular

(--> Polygonal ROIs are going to be available in one of the next versions )

### [v 0.2 --> v 0.3](#)

#### New Functions:

- Possibility to load an image from a \*.txt file

The number of rows is estimated by counting the number of line breaks

The number of columns is then estimated by dividing the number of total values with the number of rows

```
664 def convert_ascii_image_to_numpy_array(image_path):
665     """
666
667     Function that converts an image from ascii format to a numpy array
668
669     Args:
670         image_path: Path to the image in the ascii format
671
672     """
673
674     with open(image_path, "r") as file_object:
675         pixel_values = file_object.read()
676         rows = len([pos for pos, char in enumerate(pixel_values) if char == "\n"]) # Find the number of rows
677         pixel_values = re.split('[\t\n]', pixel_values) # Split string of file into pixel values
678         pixel_values = pixel_values[:-1] # Pop last element (empty string)
679         columns = int(len(pixel_values) / rows) # Find the number of columns
680
681         image_array = (np.array(pixel_values, dtype=np.float32)).reshape((rows, columns))
682     return image_array
```

An example image as \*.txt file has been added to the dist folder.

### [v 0.3 --> v 0.4](#)

#### New Functions:

- The GUI has now 2 Different procedures implemented that can be changed via checkbox  
(Calculating DUGR on projective distorted and on projective corrected image)
- The Matplotlib frontend has been polished and the layout tightened.  
the matplotlib toolbar was added to all of the figures which also allows to view pixel values by mouse hovering aswell as zooming etc.
- Polygon and ROI selection can be undone by pressing escape now
- The Result table is now also a matplotlib figure -> allows pdf export and looks better.
- A possibility to save the measurement protocol to a \*.pdf file has been implemented
- A possibility to look at every evaluation image of the projective distorted algorithm via checkbox has been added (They can be exported to \*.pdf aswell)



## [v 0.4 --> v 0.5](#)

The calculation of the DUGR value on projective distorted images has been corrected.

In order to execute the calculation, the ROI of the whole luminaire has to be drawn onto the the source image.

NOTE: At this point it only works to select trapezoid ROIs. (Rectangular and Circular ROIs produce bugs that crash the software. Therefore they will be postponed to the next version)

A possibility to load a predefined parameter set from a \*.json file for the algorithm without projective correction has been implemented.

### Bugfixes:

- Fixed a bug where the polygon Selector didn't reset properly
- Fixed a bug where the result table was overwritten with multiple entries instead of updated correctly

## [v 0.5 --> v 0.6](#)

Improved the projective distorted algorithm

## [v 0.6 --> v 0.7](#)

- Option to only filter inside a rectangular window around the specified luminous area for the algorithm without projective correction

- Performance in terms of execution time improved

--> Execution time of calculation with filtering of a whole image with high resolution (4087x1996px) down to approx 1 minute

(Was around 6-7 Minutes before for images of such high resolution)

### Bugfixes:

- Fixed a bug where a trapezoid region with a bigger width at the top than the bottom side would crash the GUI
- Fixed a bug where the ellipsoid region selector was not selectable for the algorithm with projective correction
- Fixed a bug where loading a new source image did not reset the old axis and resulted in stretching the image

### [v 0.7 --> v 1.0](#)

- The colormap for luminance images has been changed to one similar to the one which is used by the LabSoft Software from TechnoTeam

- It is now possible to change between logarithmic scalings for the colormap.

--> This allows contrast enhancement in order to draw borders onto the image.

- A lot of Bugfixes regarding different procedures of the program.  
Deviation from the sequence often led to crashes of the software before.  
This behavior is now intercepted and a hint is given in the status bar.

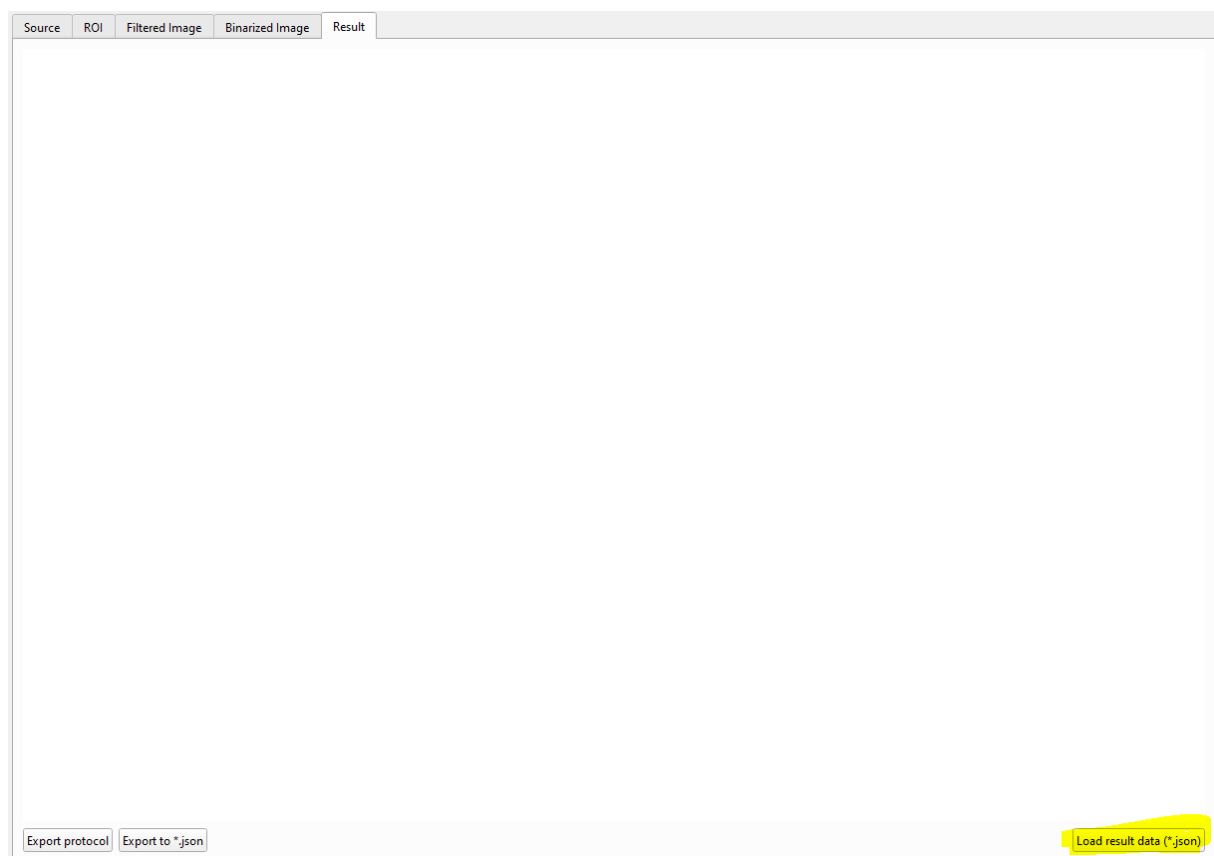
### [v 1.0 --> v 1.2.2](#)

- Changed the Python script architecture

-> Script splitted into separate Python Files to allow easier debugging

- Ability to save Parameters from the GUI to a \*.json File

- Ability to load Result \*.json Files into the Result Tab



- Small Bugfixes and stability changes

- Created an Installer for the GUI

[v 1.2.2 --> v 1.2.3](#)

- Added \*.csv handling for saving and loading the result tab

[v 1.2.3 --> v 1.3.0](#)

- Changed Parameter Layout
- Added Camera Presets (Pixel Size)
- Updated Pixel selection for Ellipsoid ROIs
- Updated Area Calculation for Ellipsoid / Round Luminaires
- Bugfixing for pixel mapping