



MANUAL v0.6.1



AragoJ 0.6.1 Manual

The up-to-date version of this manual is available from <https://github.com/AzWhaleLab/AragoJ/wiki>

Table of Contents

Why AragoJ?	1
AragoJ - Single camera photogrammetry software	1
Citation:	2
License	2
Architecture	3
Installation	3
Graphical User Interface (GUI)	4
Creating, saving and opening a work session.....	5
Create a new Session	5
Saving a Session	5
Saving a Session with a new name (Save as)	5
Open a saved Session.....	6
Importing images	6
Importing images via File menu	6
Importing multiple images	6
Drag & Drop	6
Known issues	6
Image Tools Menu	7
Description.....	7
Taking, transforming, renaming and deleting measurements	9
Taking a measurement.....	9
Transforming a measurement.....	10
Renaming or deleting a measurement.....	10
Measuring options	11
Assumptions	11
Choosing the pixel grid.....	11
Measuring method	11

Scaling	12
Scaling using fiducial markers	12
Scaling using pinhole camera model	14
Filters	17
Export	17
Camera calibration.....	18
Calibration models	18
Camera Calibration Interface	20
Calibrating the camera with a printed pattern.....	21
Calibrating the camera.....	23
Undistorting images.....	24
Expressions	24
Creating and storing expressions	24
Using expressions.....	25
ANNEX I.....	1
mXparser - built-in operators.....	1
mXparser - built-in Unary Functions	1
mXparser - built-in Iterated Operators	6
mXparser - built-in Calculus Operators	7
mXparser - built-in Mathematical Constants	7
mXparser - built-in Physical Constants.....	9
mXparser - built-in Astronomical Constants	10

Why AragoJ?

Domènec Francesc Joan Aragó, aka François Arago, was a French Catalan mathematician, physicist, geodesist, and astronomer. He is reportedly one of the first persons to understand the potential of photography as a powerful scientific and geodesy tool. As early as 1839 (the same year the daguerreotype was presented to the public), in one of his addresses¹ championing this new technology he stated

"(...) et les images photographiques étant soumises dans leur formation aux règles de la géométrie, permettront, à l'aide d'un petit nombre de données, de remonter aux dimensions exactes des parties les plus élevées, les plus inaccessibles des édifices."

which can be freely translated to something like

"(...) and the photographic images being subjected in their formation to the rules of geometry, will allow, with the aid of a small amount of data, to reconstruct the exact dimensions of the most elevated, the most inaccessible parts of the buildings."

AragoJ is a user-friendly Java application to help in the work-flow of single-camera, close-range photogrammetry, a technique that was developed in great part due to François Arago's vision.

More about François Arago can be found in [Wikipedia](#) and the reading of his own account of his unbelievable early life is highly recommended either in the original [French](#) or translated to [English](#).

AragoJ - Single camera photogrammetry software

AragoJ is a multi-platform stand-alone application, for acquiring planar measurements from photographs using simple tools.

Measurements are made in pixel units and then scaled to metric units (or other reference system) using one of two methods:

- An existing, known-size scale in the image;
- Using the camera specifications (image size and focal length) and known distance to the target for calculating the scaling factor based in a pinhole camera model.

The application reads and presents the metadata inscribed in digital images EXIF and other metadata files, which is useful for finding and using relevant information (geographic position, camera and image specifications, environmental information, etc).

Since scaling is based on a pinhole camera model, lens distortion in non-metric cameras is not accounted for when scaling measurements. To deal with that issue, the application includes a

module to self-calibrate the camera, that calculates the calibration parameters and optionally produces undistorted images.

The application also includes a module for storing mathematical expressions that can be called inside the software.

Measurements, calculation results, and image metadata can be exported as a csv file and work sessions can be saved.

Citation:

If you are publishing research that made use of AragoJ, please cite it as:

Aleixo, F., O'Callaghan, S. A., Ducla Soares, L., Nunes, P., Prieto, R. 2020. AragoJ – A free, open-source software to aid single camera photogrammetry studies. *Methods in Ecology and Evolution*. DOI: 10.1111/2041-210X.13376.

License

Copyright 2019 by Francisco Aleixo.

Licensed under the Apache License, Version 2.0 (the "License"); you may not use this software except in compliance with the License. You may obtain a copy of the License at

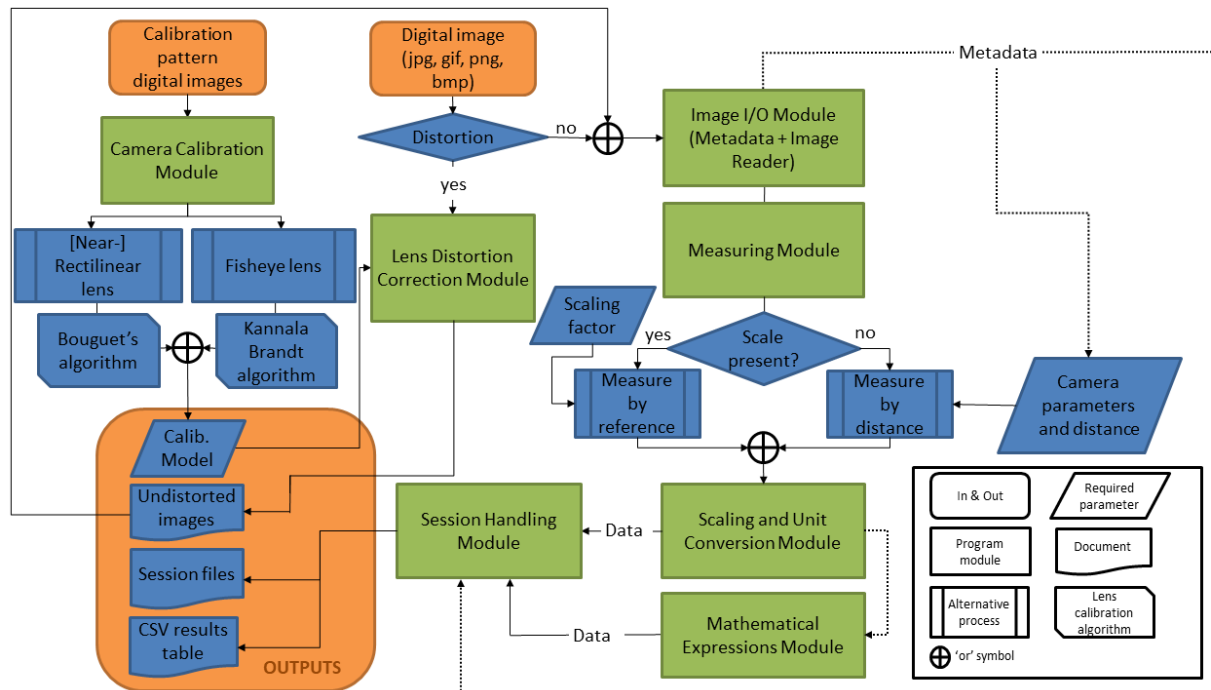
<http://www.apache.org/licenses/LICENSE-2.0>

Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

Architecture

AragoJ has a multi-modular architecture implemented in Java for its cross-platform capabilities. The GUI is implemented through JavaFX framework comprising three main panels (Navigation, Work, and [Info]rmation), along with Menu and Tool bars.

The software consists of seven modules: Camera Calibration; Lens Distortion Correction; Image I/O (Input-Output and Metadata Reader); Measuring; Scaling and Units Conversion; Mathematical Expressions; and Session Handling. In conjunction, these modules comprise a complete pipeline for data extraction and exporting.



Installation

The latest release and source code of AragoJ can be found in the GitHub repository <https://github.com/AzWhaleLab/AragoJ/releases/>.

You need to have Java installed (<https://java.com/en/download/>). Just unzip the AragoJ.zip file to a location of your preference. Double click the program's icon to launch it. This program is standalone and does not change any settings in the operating system. It can be run from an external disk or pen.

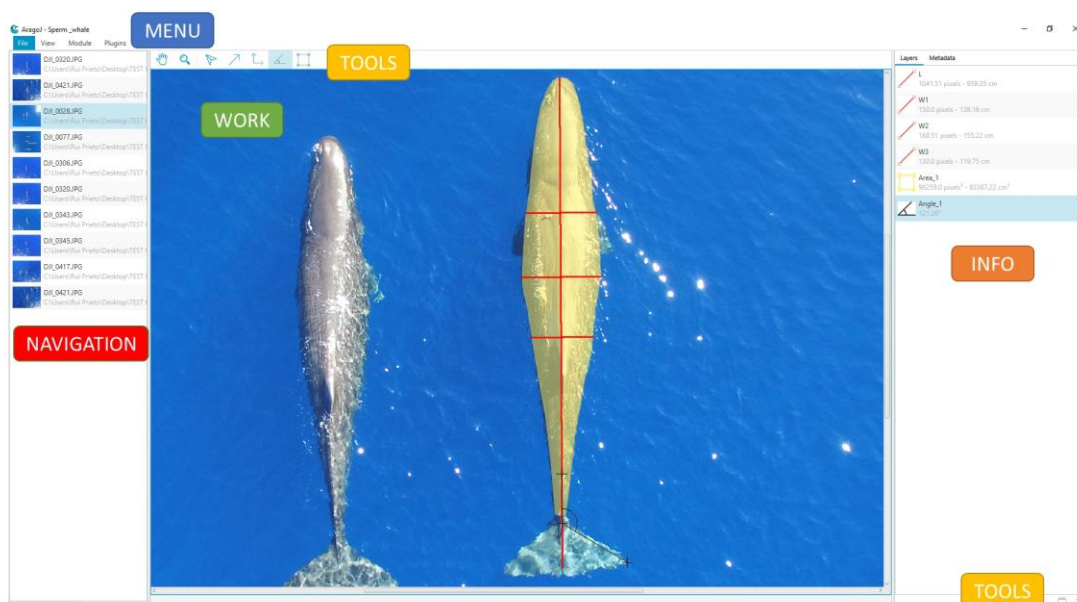
Make sure you install the correct version of Java for your system. The program functionalities may be affected if the Java version installed is not optimized for your system.

AragoJ has a module for camera calibration and correction of image distortions, based on [OpenCV](https://opencv.org/). To activate this module, currently it is necessary to download the file 'opencv_java340.zip' that is available from the AragoJ GitHub repository at https://github.com/franciscoaleixo/AragoJ/blob/master/wiki/opencv_java340.zip, and extract the single dll file to the 'libs' folder in the AragoJ directory: \AragoJ-0.4\libs. If that file is not installed, the [Camera Calibration](#) and image [Undistort](#) functionalities in AragoJ will not be activated but all other functionalities will remain unaffected.

Graphical User Interface (GUI)

When AragoJ is started, the initial screen contains the following components:

- The **Menu Bar** is used to manage sessions, import and export files, as well as to access and configure the different program modules and options.
- The **Navigation Panel** is where all the images imported into a work session are listed. A thumbnail is shown beside the image name and image location.
- The **Work Panel** is the main area of the program, where most operations over the image will be conducted. It works much like an image edition program.
- Two **Tool Bars**, located in the top-left corner of the Work Panel (*Image Tools*) and in the lower right corner of the Info Panel (*Miscellaneous Tools*), contain the image visualization, edition, measurement and calculation tools.
- The **Info Panel** is where the information about the image can be visualized. There are two tabs in this area:
 1. The *Metadata* tab shows all the metadata contained in the image's EXIF information.
 2. The *Layers* tab is where the measurements information is shown during work with the image.



Creating, saving and opening a work session

A “Session” is how work is stored by AragoJ. The work made during a Session can be saved and retrieved at a later time.

Create a new Session

To create a new session, choose

- File->New session (Ctrl+N)

You can work without saving the session but it is strongly recommended that you save it just after creation.

Saving a Session

To save a session, choose

- File->Save (Ctrl+S) or File->Save As (Ctrl+Shift+S)

Give any desired name to the file and save it to your preferred location. The file will be saved with an *axml* extension.

NOTICE: To keep the axml files to a manageable size, the images are not contained in the files, and only the image path (or paths) is saved. This means that if you want to open the session in another computer or if you change the image(s) location, the saved session will not open correctly the next time it is called, unless an identical path to the image(s) is found. To ensure that sessions will open in different machines, it is advisable that all files related to specific sessions are located in a folder in the root (for example, C:\AragoJ\Project1, where “AragoJ\Project1” can be replaced by any other folder names).

Saving a Session with a new name (Save as)

You can save an existing session with a new name by choosing

- File->Save As (Ctrl+Shift+S)

This can be useful for quickly backing up work or for making changes without losing previous work.

Open a saved Session

To open a session saved earlier, choose

- File->Open (Ctrl+O), and navigate to the desired *axml* file location.

Importing images

AragoJ accepts images in jpg, gif, bmp, and png formats natively. Support for other image formats can be added through plugins, consisting in simple Java library projects packaged in a JAR file. Guidance on how to create plugins is available on the GitHub repository. A plugin adding support to over 150 microscopy file formats, using the Bio-Formats library released by the Open Microscopy Environment Consortium (<http://openmicroscopy.org>) is included in AragoJ distributions after version 0.6.1, which can also be used as an example for the creation of additional plugins.

NOTICE: Images can only be imported after a Session has been [opened or created](#).

Importing images via File menu

To import images select

- File->Import images (Ctrl+I)

A window will open where you can navigate to the folder(s) containing the images. New images can be added to a session at any time, preserving the images that were already loaded.

Importing multiple images

You can select multiple images using the multiple file selection controls (Ctrl+A; Ctrl+Drag; Shift+Click; Ctrl+Click). After you select the files, press 'Open' and the images will be imported into the Navigation Panel.

Drag & Drop

Images can also be imported by "drag and dropping" multiple files into the Navigation Panel.

Known issues

The number of images that can be opened simultaneously in 32 bit systems is reduced due to memory limitations. In 64 bit systems the number of images that can be opened is theoretically unlimited.

Very large images may not display correctly or at all. This is due to tiling issues (depending on the hardware it can support from 4096x4096 to 16384x16384). The use of a tiling algorithm is planned in future releases to solve this issue.

Image Tools Menu

To start working with an image, select it from the list in the Import Panel and it will be presented in the Work Panel.

There are seven tools available in the Image Tools menu:



Description



Pan

Function

This tool is used to reposition the image inside the working area. After choosing the Pan tool, click and hold the left mouse button over the image and move it to the desired position. In alternative, use the keyboard arrows to move it up, down, right, or left. The sliders along the Y and X axes of the image can also be used to move the image.

Mouse alternative

If a wheel mouse is present, and even if the Pan tool is not selected, moving the wheel will move the image up and down. To move the image left and right, press 'Shift' while moving the wheel.



Zoom

Function

This tool is used to magnify/shrink the image. After choosing the Zoom tool, click over the image with the left mouse button to zoom in and with the right button to zoom out.

Mouse alternative

If a wheel mouse is present, and even if the Zoom tool is not selected, pressing the keyboard 'Ctrl' key while moving the wheel will zoom in and out.



Function

This tool is used to move and change lines and shapes. To be able to move or transform any line or shape that has been created using other tools, it is necessary to first select this tool.



Function

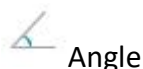
This tool draws segmented lines over the image. After selecting the tool, using the mouse left-click any point in the image to create a starting point. Click other points in the image to create successive line segments. After selecting the last point, right-click the mouse to finish drawing the line.

Line color can be changed using the color selector (see below).



Function

This tool is used to create lines at a given angle from another line segment (e.g. parallel; perpendicular). It can only be used if at least one line segment is already present in the image. Select this tool and a text box will be displayed in the right side of the Image Tools Menu where any angle can be inserted. Then, to draw a line at a corresponding angle of a pre-existing line segment, left-click the line segment once to choose it and then left-click again in any point to start drawing the line. Finish drawing the line by left-clicking.




Function

This tool returns the absolute angle between two structures in the image (varying from 0-180°). After selecting the tool, left-click the mouse to select any point over the first angle ray; continue by selecting a second point where the angle vertex is located and then select any point over the second ray to finish drawing the angle.

After the angle vertex has been chosen and while the point moving the tool to select the point over the second ray, the angle measurement is displayed on the left corner below the image.

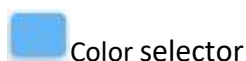


Function

This tool is used to draw polygons over the image. The tool functions much like the Line  tool.

The difference is that to close the polygon, the last point in a series of line segments must be positioned roughly over the starting point. When the mouse cursor is near the starting point it will be 'attracted' to it and by left-clicking over it, the polygon will be closed.

Polygon filling color can be changed using the color selector (see below).



Function

This tool is used to change colors of lines and polygons. It is only active for those tools. By clicking over the color selector icon, a color palette is displayed with 190 pre-loaded colors. If desired, custom colors can also be chosen using the Custom color selector (at the bottom of the color selector window). A pop-up window will be displayed with a new color palette, and colors can be chosen manually by moving the two cursors or introduced directly in the text box using either the RGB, HSB or HEX codes.

Taking, transforming, renaming and deleting measurements


Taking a measurement

Lines, angles and polygons will be drawn as a 'layer' over the image and can be changed, moved or deleted without affecting the image itself. There are two classes of lines that can be used either independently or simultaneously, and can be chosen from the View menu:

1. *Precision lines*: precision lines are thin lines, ending in a cross symbol, to enable choosing single pixels in an image as start and end points of a line segment. When the image is zoomed in, precision lines are obvious but when the image is zoomed out they may become difficult to see.
2. *Identifier lines*: identifier lines exist to help seeing the precision lines and to enable classifying different types of line segments by color. To change the line color, use the color picker that appears in the Tools menu once the Line tool is selected.

Each new line segment will appear in the Info Panel, under the Layers tab, and will be provisionally labelled incrementally, with names in the format 'Line_#' (where # stands for the line number). The same applies to polygons and angles.

Transforming a measurement

Once a measurement is taken it can be changed using the Select  tool.

- *Changing length and angle of a line*: click and hold either end of the line segment and move it to a new position. The other end will remain in the original position, leading to a change in angle and length simultaneously.
- *Changing length while angle is kept constant*: click and hold either end of the line segment while holding down the Shift key.
- *Changing the shape of a polygon*: click and hold any vertex of the polygon and move it to another position to re-shape it.
- *Changing an angle measurement*: click and hold any vertex in the angle and move it to another position to recalculate the angle.


NOTICE: In multi-segmented lines, the angle can only be kept constant in the first segment.

Renaming or deleting a measurement

Renaming

After a measurement has been taken, it can be renamed in the Info Panel by double clicking the provisional name [e.g. Line_#]. Right-clicking the mouse after a measurement has been chosen for renaming will open a box with several options (Undo, Redo, Cut, Copy, Paste, Delete, Select All). It is desirable that measurements belonging to the same category and taken from different pictures receive the same name if the goal is to export a table of results, since the names of the columns will correspond to the measurement names.

Deleting

If necessary, a measurement can be deleted by right clicking it in the Info Panel and choosing *Delete Layer*, or via the Trash Bin symbol  in the lower right corner (Miscellaneous Tools).

NOTICE: the remaining layer names and order will not be changed and new layers will still be named incrementally. If the layer names are not changed, this can create mismatches in category attribution among different images. To overcome this problem, just rename the layers before exporting results.

Measuring options

Assumptions

It is assumed that users of AragoJ are familiar with the basic principles of close-range photogrammetry theory. There are several excellent works on the theory and applications of photogrammetry as well as many on-line resources that can be found from the usual literature sources. The Wikipedia entry on photogrammetry can be found [here](#). The scientific publication that describes AragoJ ([reference](#) in page 2 in this manual) contains references to several relevant works on photogrammetry theory.

Presently, AragoJ assumes that pixels are square and, in consequence, that the conversion factor applies equally to the length and width of the image, which is an acceptable assumption for most current (2018) digital photographic cameras.

Measurements made by AragoJ also assume that images have no perspective [tangencial] ($\approx < 3^\circ$ camera tilt ref object) and/or optical [radial] (barrel, pincushion, mustache) distortions, **which is not realistic for most photographic images**.

To ensure that the measurements are as accurate as possible, images should be taken in a way that avoids perspective distortion and images suffering from optical distortion should be processed to reduce or eliminate the distortion prior to any measurements being taken. If necessary, AragoJ has a module for [camera calibration and image correction](#).

Finally, AragoJ **IS NOT** meant to be used in 3-D photogrammetry, and thus it cannot handle 3-D surfaces.

Choosing the pixel grid

AragoJ allows sub-pixel values, and three pixel selection options are available: free (line segments can start and end at any position inside a pixel), 0.5 pixel (line segments can start and end either at the center or the edge of a pixel) and 1.0 pixel (line segments can only start and end at the edge of a pixel). These options allow controlling for unrealistic precision and can be set can be chosen from the View menu.

Measuring method

Apart from angles, all measurements are taken in pixel units prior to scaling (see below). The linear measurement tool accepts any number of segments, allowing for the measurement of irregular and curved shapes. After a multi-segment line is created, the value that will be presented is the sum of the values calculated for each of its segments.

Polygon area is calculated from coordinate geometry, using equation (1)

$$Area = \left| \frac{(x_1y_2 - y_1x_2) + (x_2y_3 - y_2x_3) + \dots + (x_ny_1 - y_nx_1)}{2} \right| \quad (1)$$

with $x.n$ and $y.n$ being the coordinates of vertex n . Since the method is not applicable for area calculation of self-intersecting polygons, these are first decomposed using the Java class `FlatteningPathIterator`.

Scaling

The Scaling and Unit Conversion module is responsible for scaling measurements from pixel to metric units. For convenience, the module also includes a tool to convert from metric to other units. After scaling from pixel to metric units (see below), a conversion factor can be applied under the Module menu

- Module->Scale->Convert units

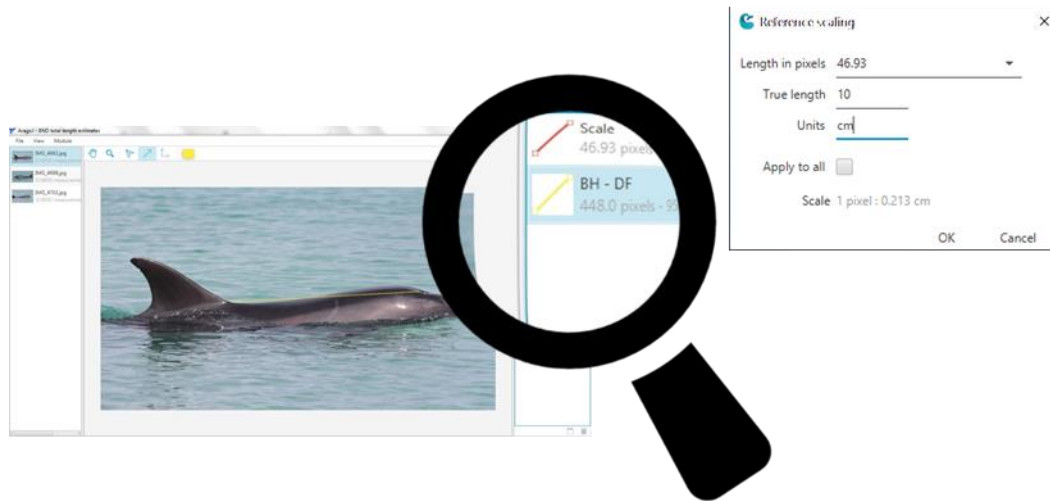
NOTICE: If the goal is to obtain relative metrics, scaling is not strictly necessary (the ratio between two metrics will be the same regardless of the units used).

Scaling using fiducial markers

In order to set a scale from an object of known size present in the image (fiducial marker[s]), first draw a line over that object. Once the line has been created, open the *Scale by Reference* Module by calling the *Scale by Reference* Module under the Module menu:

- Module->Scale->Reference

A pop-up window will open:



The scale line can be chosen from the drop down menu 'Length in pixels'. In alternative, a value (in pixels) can be inserted manually.

NOTICE: In some studies, multiple scales are used to obtain an average scaling factor in order to minimize error. Currently Arago! does not accept more than one scaling line, although that is planned in future releases. As a work-around, it is possible to obtain several scale measurements and calculate the average outside Arago! or using the [Expressions Module](#) and then inserting the resulting value in the 'Length in pixels' field.

To set the conversion factor, the true length of the object in the desired unit has to be inserted in the 'True length' field.

The desired unit abbreviation can be inserted in the 'Units' field.

The program will use equation (2) to scale all measurements (in pixels) made over the image:

$$Lm = \frac{(L_{px} * Sc_m)}{Sc_{px}} (2)$$

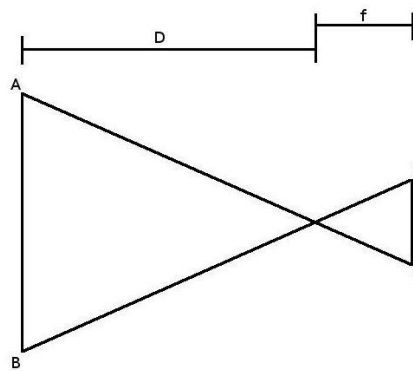
where $L.m$ stands for the length of the structure being measured, in the desired measurement unit; $L.px$ for the length of the same structure in pixels; and $Sc.px$ and $Sc.m$ are the lengths of the scale in pixels and in the desired measurement unit, respectively.

It is possible to set a common scale to all pictures in a Session by checking the 'Apply to all' tick box. This is especially useful if pictures are all taken at constant distance from object to camera, such as in microscopy or other controlled situations (for instance, animals moving through a single working alley).

NOTICE: any object used as a scale has to be in the same plane as, and parallel to the object to be measured, or the results will be biased.

Scaling using pinhole camera model

Using the simplest model, a pinhole camera, the size of the camera sensor and lens focal length coupled with distance to object determines the image scale or the resolution of the image.



The scale of an image is the ratio of the distance on the image to the corresponding distance at the object plane.

In the figure above, the distance 'AB' will be projected on the camera sensor on line 'ba', therefore, the image scale can be computed using the following formula:

- Equation [i]: $\text{scale} = (\text{distance } ba) / (\text{distance } AB)$

Using the similarity of triangles principle, it is also easy to conclude that

- Equation [ii]: $\text{scale} = (\text{lens focal length: } f) / (\text{distance to object: } D)$

Thus, it is possible to resolve for the distance AB

$$\text{distance } AB = [(\text{distance to object: } D) \times (\text{distance } ba)] / (\text{lens focal length: } f)$$

AragoJ uses that principle to convert pixel units to the metric system. Using information on the camera geometry and distance to object, the scaling factor is computed using the following equation:

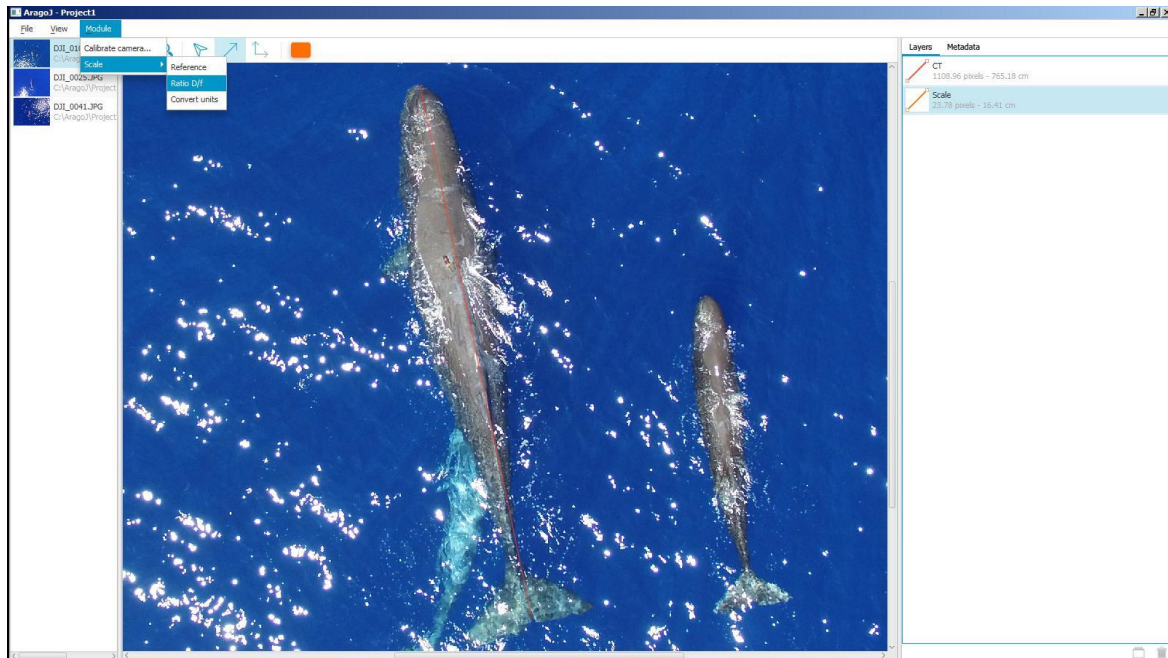
$$L_m = L_{px} \frac{S_w * (D + \beta) * 100}{Fl * imW} \quad (3)$$

where L_m [distance 'AB' above] stands for the length of the object in meters (m); L_{px} [distance 'ba' above] the length of the object in pixels; S_w is the width of the camera sensor in millimeters (mm); D the distance of the camera to the object (in meters); Fl the camera focal length (in millimeters); and imW the image width in pixels (by dividing the sensor width [in mm] by image width [in pixels] we obtain the pixel size in metric units).

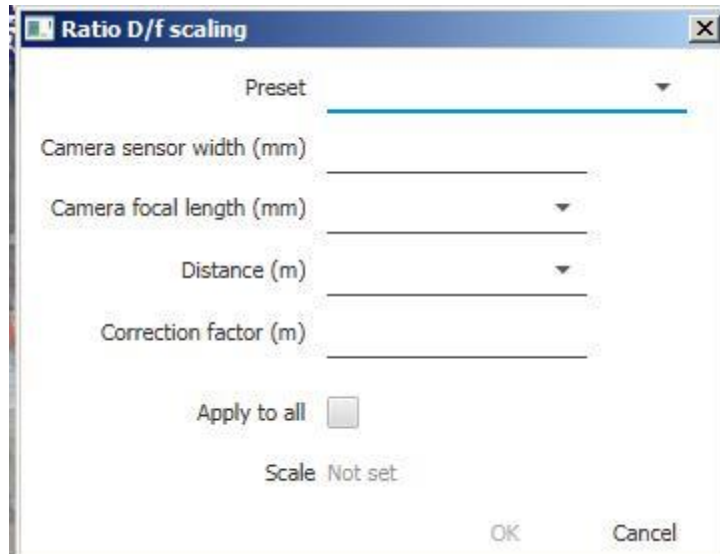
β in equation (3) is a correction factor for the camera to object distance. It can be useful in cases where a correction has to be applied to the distance to object measurement. For example, the height reported by a barometric altimeter, often used in drones, is referenced to the take-off location and can differ from the true height of the drone relative to the location where the image was taken (which is equivalent to the distance 'D'); if that difference is known, β can be used to correct for that. If no correction is necessary, the field for β should be left blank or set to 0.

The Module for scaling measurements using the geometric principles above is called D/f in AragoJ and is found under the Module menu:

- Module->Scale->Ratio D/f



and once it is invoked the following pop-up window opens



The *Camera Sensor Width*, in millimeters (mm), must be provided. It may be stated in the camera manual or it may part of the EXIF metadata. If necessary, a good [database](#) of camera sensor sizes is maintained by [OpenMGV](#).

Camera Focal Length, in millimeters (mm), can be imputed either manually or using the drop down menu to query the image EXIF information. Make sure that this information is accurate because even small changes in this value may have a big effect in the final result.

Distance to object, in meters (m) can also be imputed manually or using the EXIF information (depending on camera model the distance/altitude may be recorded with other EXIF metadata).

The *Correction Factor* β , can be set to any value in meters (m). If no correction is necessary, leave the field empty or set it to 0.

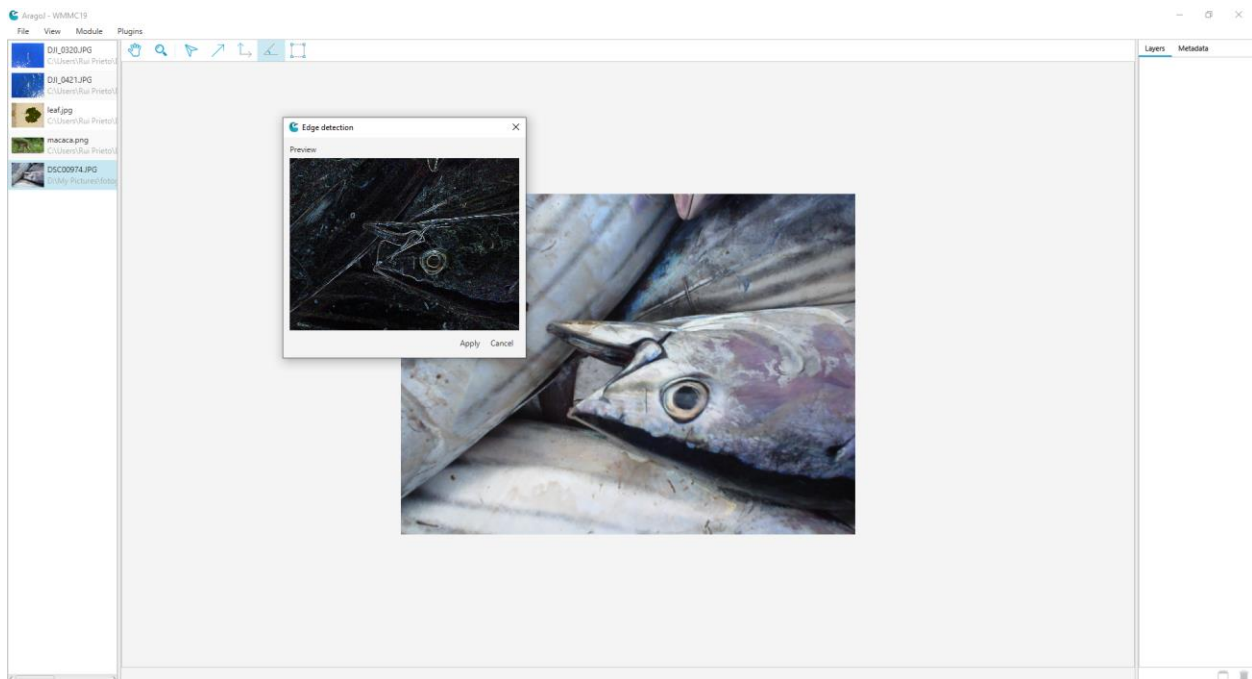
It is possible to set a common scale to all pictures in a Session by checking the '*Apply to all*' tick box. This is especially useful if pictures are all taken at constant distance from object to camera (for example, animals moving through a single working alley or an aircraft flying at a constant height).

Currently the *Preset* drop-down menu is inactive. In a future release, it will be possible to save the camera settings and recall them using this option.

Filters

Filters may be useful to highlight structures of interest in an image. Currently AragoJ implements two types of simple filters: 1) a sharpen filter to sharpen blurred images; and 2) a Sobel edge detection filter. Results are presented in a small pop-up window and if the user is happy with the result the filter can then be applied to the original image. Filters can applied and reverted by choosing

Module->Filters



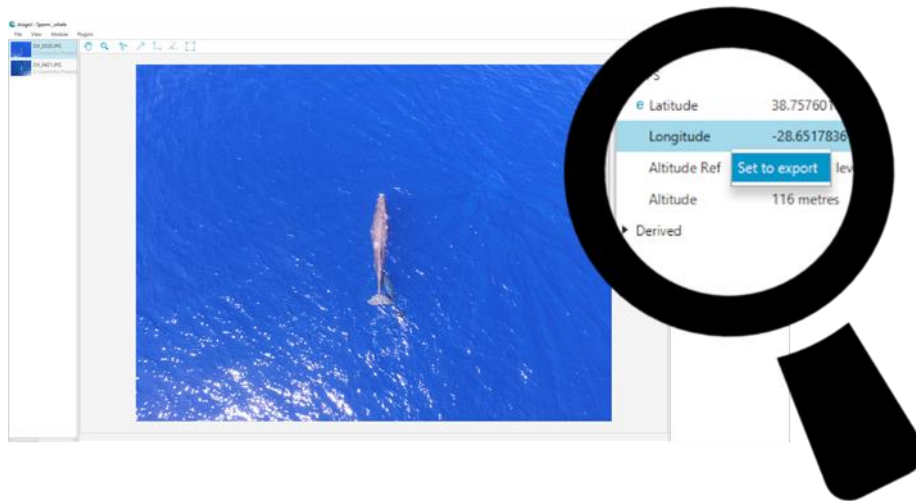
Export

Measurement results and selected metadata from the image EXIF information can be exported as a CSV file:

- File->Export as csv (Ctrl+E)

All measurements, in pixels and any other unit, will be automatically exported.

To export any metadata select the *Metadata Tab* in the Info Panel, and select the desired Metadata field by right-clicking the field name and selecting "Set to export"; if you desire to reverse the operation, just right-click over its name again and select "Unset to export".



Assuming all pictures were taken with the same camera model, this operation only needs to be performed once. Selected metadata will be exported for all pictures.

The exported csv will be organized with the images as lines and measurements and metadata as columns (fields):

	A	B	C	D	E	F	G	H	I	J	K	L
1	Source	Latitude	Longitude	TL (px)	TL (cm)	W1 (px)	W1 (cm)	W2 (px)	W2 (cm)	W3 (px)	W3 (cm)	
2	C:\AragoJ\Project1\images\DJI_0104.JPG	38.72735189	-28.65576075	1108.96	808.43	139.46	101.67	166.25	121.2	149.11	108.7	
3	C:\AragoJ\Project1\images\DJI_0025.JPG	38.75255678	-28.69888725	882.88	795.47	117.69	106.04	134.8	121.45	117.72	106.07	
4	C:\AragoJ\Project1\images\DJI_0041.JPG	38.71178436	-28.57540558	971.83	871.73	112.58	100.98	145.7	130.69	112.74	101.13	
5												
6												
7												

Camera calibration

Calibration models

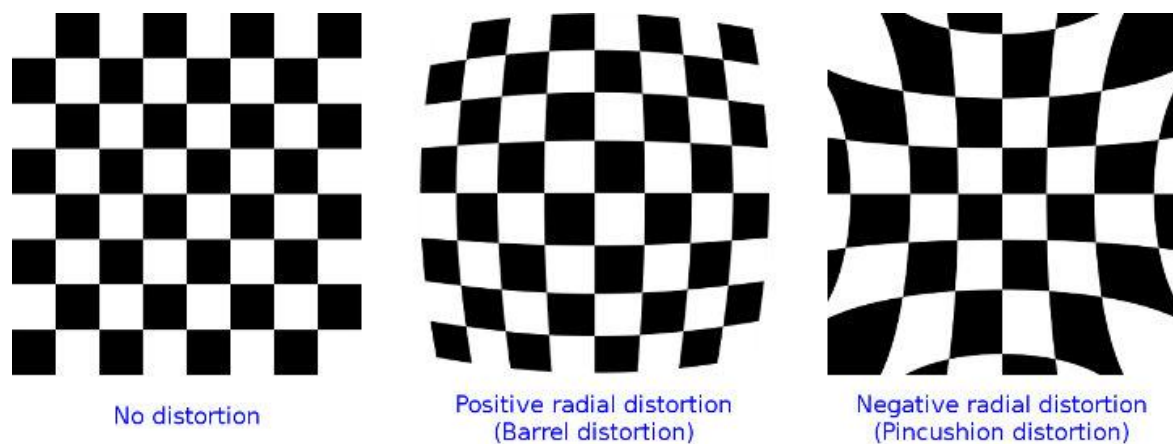
AragoJ assumes that all pixels in an image have the same dimensions, or in other words that there is no image distortion caused by the camera lens.

For true rectilinear lenses, the distortion effect is negligible or absent and measurements can be taken directly from the pictures. However, most consumer level lenses, even those considered 'rectilinear', will create some amount of distortion violating that assumption.

Distortions can often be identified by taking pictures from straight lines (e.g. walls with doors, windows, paintings frames), and seeing if straight lines are reproduced with some degree of curvature.

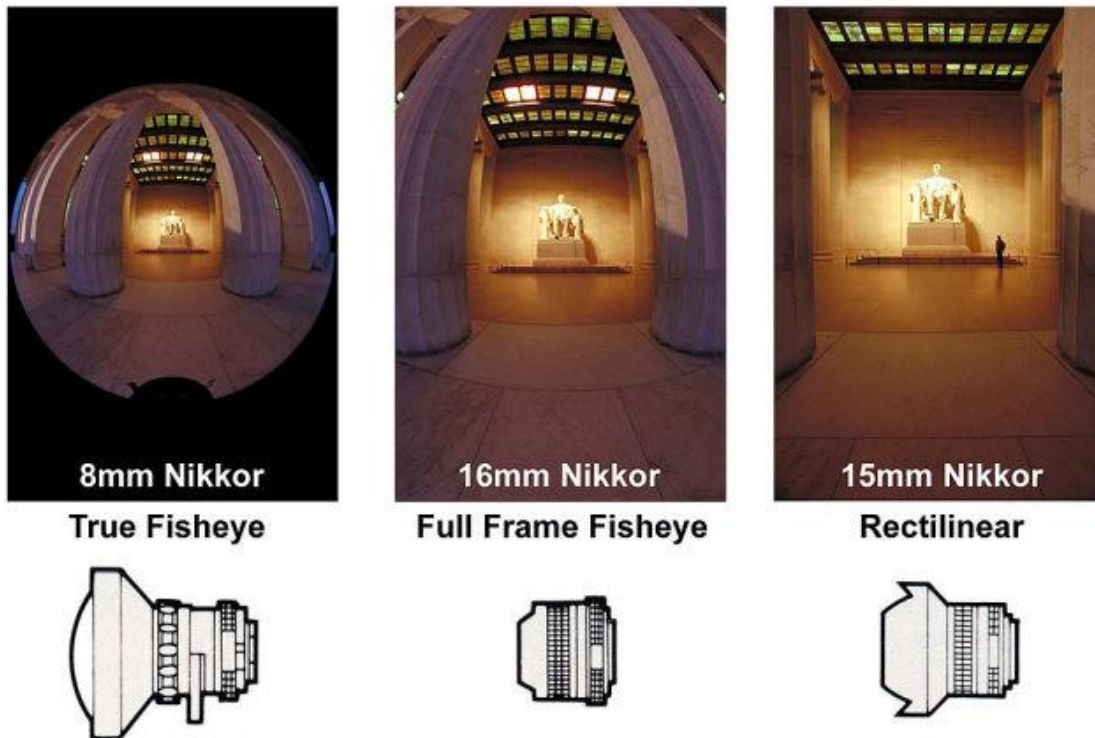
As long as the geometric parameters of the lens are known, the distortion can be corrected. These parameters can be estimated by taking a large number of pictures from a known-size pattern, from several orientations and distances, which is called geometric camera calibration or camera resectioning.

AragoJ includes a camera self-calibration module (*Calibration*), to calculate the calibration parameters that can then be used to undistort images. The *Calibration* module is based on the *Calib3d* module from [OpenCV](#), implemented through a custom C++ to Java wrapper. This module will account and correct for the two main types of radial distortion: positive (Barrel) and negative (Pincushion), illustrated in the next image



(Reproduced from [OpenCV](#))

The mathematical camera model used by OpenCV in the main calibration functions does not generalize well enough for all types of camera lenses, but it does offer ways of handling different types of lenses. Two options, for fisheye and for near-rectilinear lens types, are implemented in AragoJ. Fisheye lenses normally yield highly distorted images, with clearly round edges, while distortions from near-rectilinear cameras are more difficult to identify. Prior to camera calibration the optimal model type should be identified from the amount of distortion seen in the pictures.



(Scott Highton. Reproduced from [Virtual Reality Photography](#))

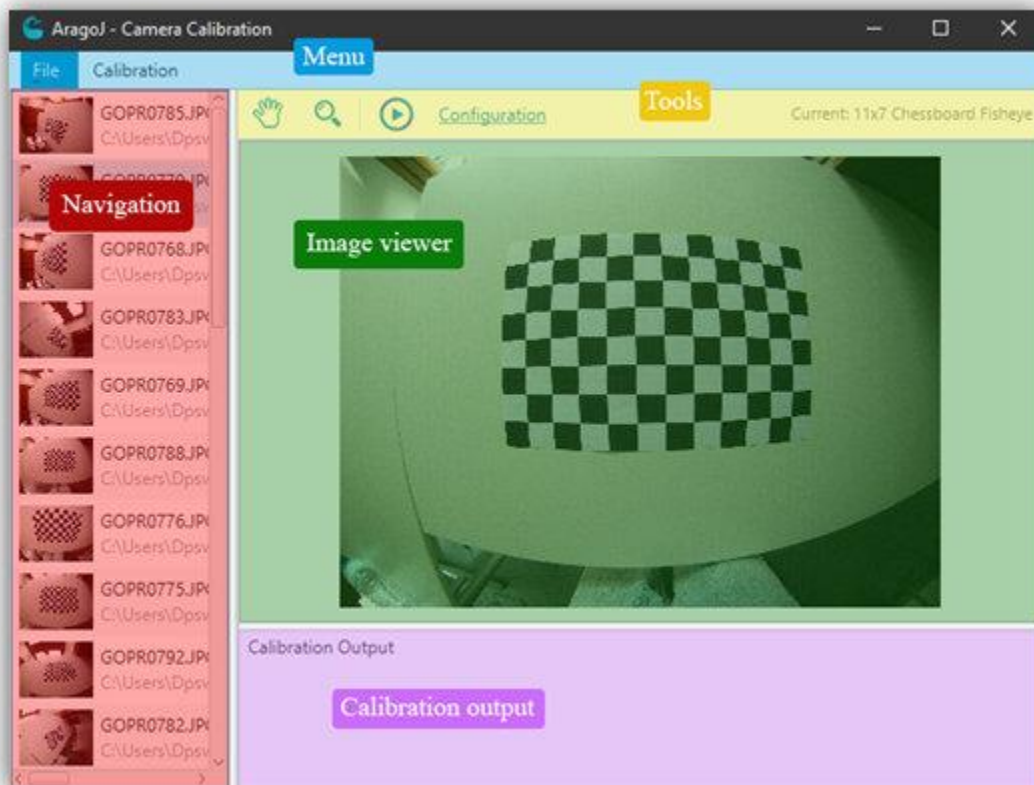
Camera Calibration Interface

The Calibration module has a dedicated screen and related menus that share a similar structure to the main AragoJ screen and is composed of the following components:

- The **Menu Bar**, which is composed of several utilities, namely:
 1. Saving and exporting the calibration model. Saving and exporting are equivalent, but when exporting the user can 1. choose where to export. If the user chooses to save, it saves automatically to a local directory;
 2. Import calibration images;
 3. Run calibration process;
 4. Configure calibration process.
- The **Navigation Panel**, where all the imported calibration images are listed. A thumbnail is shown beside the image name and image location. After calibrating, a status bar next to the thumbnail will appear either green if a calibration pattern was found for that image or red otherwise;
- The **Image Viewer Panel**, which is similar to the main AragoJ work panel. It has a top toolbar where the user is able to zoom and pan the selected image as well as run and configure the calibration. It also displays the current configuration on the right. After

calibrating, the calibration pattern is overlaid in each of the images that was used in the calibration, but not on those for which the pattern was not found.

- The **Calibration Output Panel**, where information about the results of the calibration process are shown, including elapsed time, extracted camera model name, number of images with pattern detected and overall reprojection error, which is the arithmetic mean of each image reprojection error, see equation.



Calibrating the camera with a printed pattern

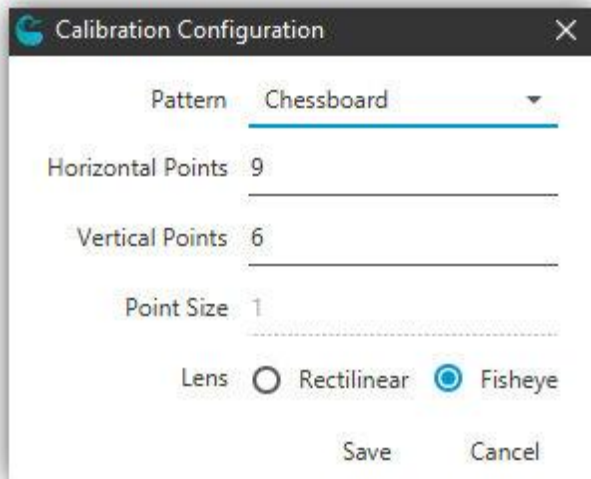
Preparing the calibration pattern

Several types of patterns can be used (squares, circles, triangles), and each have advantages and drawbacks that will not be discussed here for the sake of simplicity. Currently, AragoJ only admits square patterns, although the option for using circle patterns is also considered for future implementation.

This 9x6 [chessboard pattern](#) can be used, or you can create a custom one. The pattern should be printed with no size alterations and fixed to a flat surface.

Setting the calibration parameters

First it is necessary to set the calibration variables by opening the *Configuration* box under the Calibration menu:



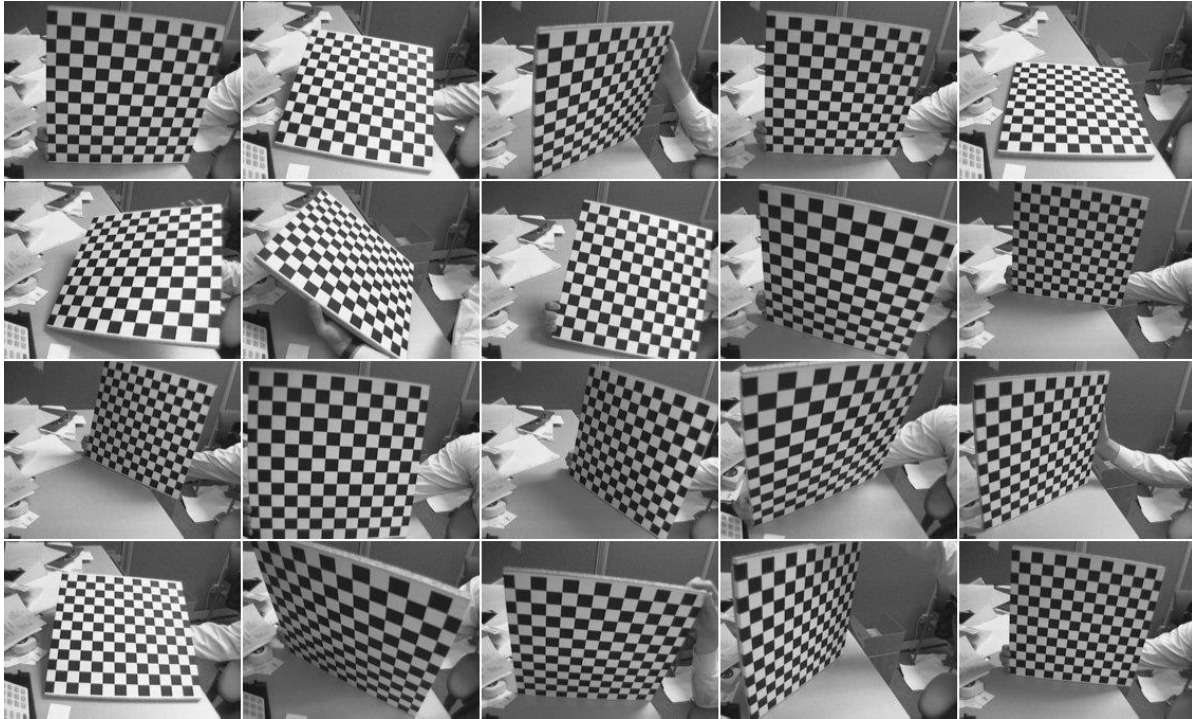
There are several variables that must be configured:

- **Pattern:** The calibration pattern that is present in the calibration images. Currently only chessboard patterns are accepted, but circle patterns will also be recognized in future releases;
- **Horizontal Points:** The number of calibration features in the x axis. In the case of chessboard, it is the number of chessboard corners in a row. In the case of a circle grid it is the number of circles in a row;
- **Vertical Points:** The number of calibration features in the y axis. In the case of chessboard, it is the number of chessboard corners in a column. In the case of a circle grid it is the number of circles in a column;
- **Point size:** Only applies to the circle grid pattern and is the rough size of the circles in millimetres. It relates to the way the blob detector, used to detect circles, works;
- **Lens:** The type of lens, either rectilinear (low deformation) or fisheye (large deformation), which is associated with different calibration models.

When all variables have been set, the configuration can be saved.

Creating a set of calibration pictures

Multiple pictures from the pattern have to be taken from different angles and distances for an optimal calibration, as shown in the image below



Brian moore81 [[CC BY-SA 4.0](#)], [via Wikimedia Commons](#)

The calibration pattern should cover most of the picture area and pictures should be well lit and have no shades or reflexes, as that may affect the ability of the program to detect the calibration features. Ideally, between 20-30 pictures should be taken from different angles.

Calibrating the camera

After the calibration images have been collected they should be imported into the Calibration module, using the File menu or by drag & dropping images to the Navigation Panel.

Then the calibration can be initiated by selecting **Run Calibration** under the Calibration menu.

After calibrating, the calibration pattern is overlaid on each of the images that was used in the calibration, but not on those for which the pattern was not found.

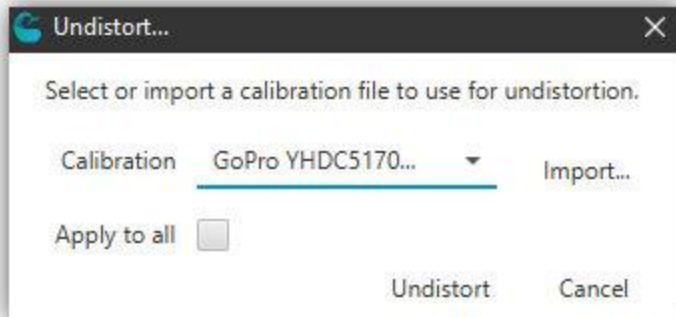
The calibration results are displayed in the *Calibration Output Panel*, including the calculated reprojection error that provides a qualitative measure of model performance, with low values indicating good performance (please refer to [OpenCV](#) and related documentation for further explanation). However, the best way of evaluating if a calibration model is good is by examining undistorted images (see [Undistorting Images](#)).

If the results from a calibration model are not acceptable, the process can be repeated as many times as required with new pictures taken from slightly different angles, and if necessary with better illumination.

Undistorting images

Saved calibration models can be used to correct lens distortions in images, using the *Undistort* module.

The *Undistort* module in AragoJ uses functions from [OpenCV](#), and can be opened by choosing *Undistort* under the Calibration menu, which will open the following box




As it launches, *Undistort* automatically checks the program file directory for any '.acalib' files and automatically imports them. Calibration files can also be manually located and imported from other locations.

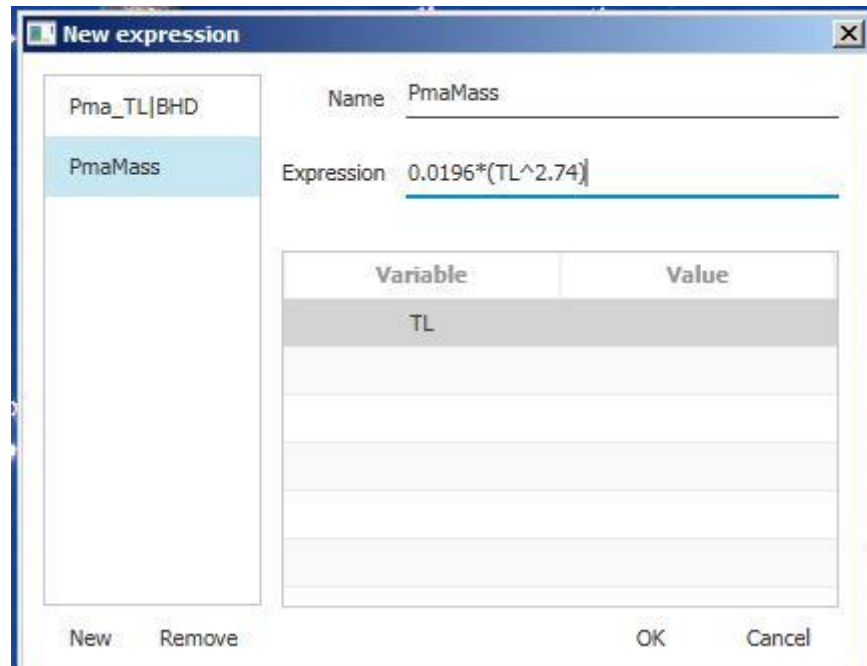
The 'Apply to all' checkbox simply applies the distortion correction algorithm using the chosen model to every image in the session if checked, or to only the selected image if unchecked. Undistorted images are saved with a suffix ('_u') relative to the original files' name, and the original images are preserved.

Expressions

Expressions of any type can be created in AragoJ using the *Expressions Module*. For example, it may be desirable to use an expression to calculate the area of a structure, or to derive measures such as size and weight from known allometric relations.

Creating and storing expressions

The *Expressions Module* is available through the *Miscellaneous Tools Menu*. Under the *Info Panel* choose the drawer symbol  to open the Expressions box:



To create a new expression, press 'New' in the lower left corner and start by giving a simple name to the expression in the Name field.

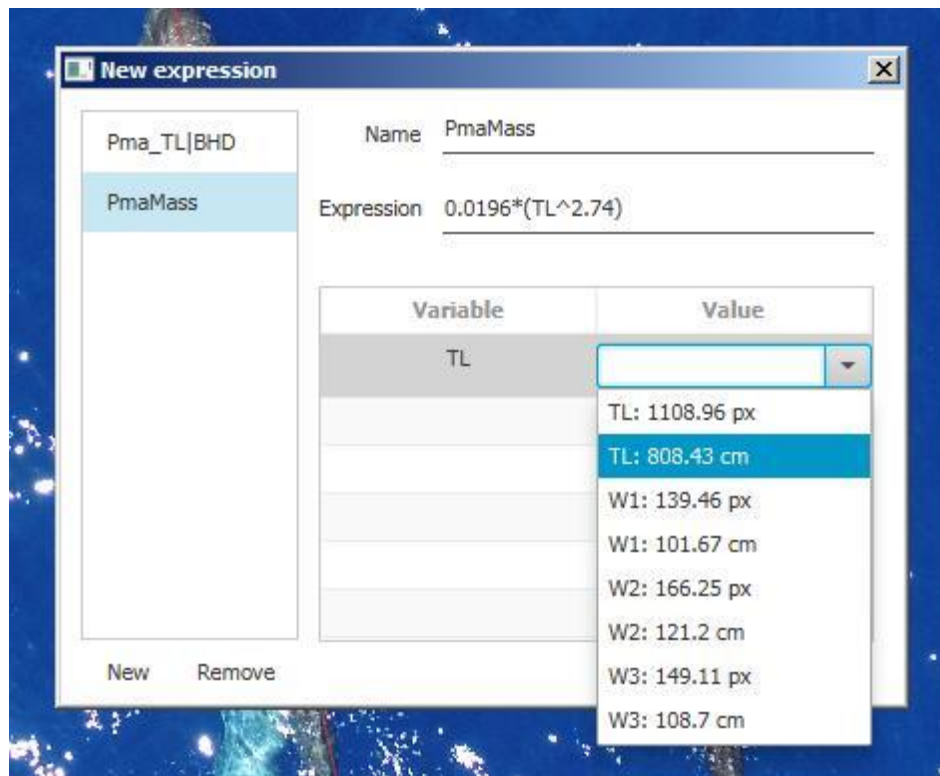
In the Expression field create the expression. Use letters for any variables in the expression; those will be presented in the list below the Expression field.

Press OK to save the expression.

A partial list of built-in operators, functions, and constants supported by the *Expressions Module* is presented in [Annex I](#).

Using expressions

To use an expression choose it from the Expression list in the left panel. You can assign values to the variables from measurements taken from the images, by opening the drop-down list under the "Value" column



The result will be written to the measurements list in the Info Panel and will be included as a new column if a CSV file is exported.

ANNEX I

This is a partial list of built-in operators, functions, and constants supported by AragoJ [Mathematical Expressions](#) module. AragoJ is a multi-platform stand-alone application, for acquiring planar measurements from photographs using simple tools and can be found here: <https://github.com/AzWhaleLab/AragoJ>.

Mathematical expressions in AragoJ are handled by mXparser (<http://mathparser.org>). For a complete list of options please visit <http://mathparser.org/mxparser-math-collection/>

mXparser - built-in operators

Key word	Category	Description	Example
+	Operator	Addition	$a + b$
-	Operator	Subtraction	$a - b$
*	Operator	Multiplication	$a * b$
/	Operator	Division	a / b
^	Operator	Exponentiation	$a ^ b$
!	Operator	Factorial	$n!$
#	Operator	Modulo function	$a \# b$

mXparser - built-in Unary Functions

Key word	Category	Description	Example
sin	Unary Function	Trigonometric sine function	$\sin(x)$
cos	Unary Function	Trigonometric cosine function	$\cos(x)$
tan	Unary Function	Trigonometric tangent function	$\tan(x)$
tg	Unary Function	Trigonometric tangent function	$\text{tg}(x)$
ctan	Unary Function	Trigonometric cotangent function	$\text{ctan}(x)$
ctg	Unary Function	Trigonometric cotangent function	$\text{ctg}(x)$

Key word	Category	Description	Example
cot	Unary Function	Trigonometric cotangent function	$\cot(x)$
sec	Unary Function	Trigonometric secant function	$\sec(x)$
cosec	Unary Function	Trigonometric cosecant function	$\operatorname{cosec}(x)$
csc	Unary Function	Trigonometric cosecant function	$\csc(x)$
asin	Unary Function	Inverse trigonometric sine function	$\operatorname{asin}(x)$
arsin	Unary Function	Inverse trigonometric sine function	$\operatorname{arsin}(x)$
arcsin	Unary Function	Inverse trigonometric sine function	$\operatorname{arcsin}(x)$
acos	Unary Function	Inverse trigonometric cosine function	$\operatorname{acos}(x)$
arcos	Unary Function	Inverse trigonometric cosine function	$\operatorname{arcos}(x)$
arccos	Unary Function	Inverse trigonometric cosine function	$\operatorname{arccos}(x)$
atan	Unary Function	Inverse trigonometric tangent function	$\operatorname{atan}(x)$
arctan	Unary Function	Inverse trigonometric tangent function	$\operatorname{arctan}(x)$
atg	Unary Function	Inverse trigonometric tangent function	$\operatorname{atg}(x)$
arctg	Unary Function	Inverse trigonometric tangent function	$\operatorname{arctg}(x)$
actan	Unary Function	Inverse trigonometric cotangent function	$\operatorname{actan}(x)$
arcctan	Unary Function	Inverse trigonometric cotangent function	$\operatorname{arcctan}(x)$
actg	Unary Function	Inverse trigonometric cotangent function	$\operatorname{actg}(x)$
arcctg	Unary Function	Inverse trigonometric cotangent function	$\operatorname{arcctg}(x)$
acot	Unary Function	Inverse trigonometric cotangent function	$\operatorname{acot}(x)$
arccot	Unary Function	Inverse trigonometric cotangent function	$\operatorname{arccot}(x)$

Key word	Category	Description	Example
ln	Unary Function	Natural logarithm function (base e)	ln(x)
log2	Unary Function	Binary logarithm function (base 2)	log2(x)
log10	Unary Function	Common logarithm function (base 10)	log10(x)
rad	Unary Function	Degrees to radians function	rad(x)
exp	Unary Function	Exponential function	exp(x)
sqrt	Unary Function	Square root function	sqrt(x)
sinh	Unary Function	Hyperbolic sine function	sinh(x)
cosh	Unary Function	Hyperbolic cosine function	cosh(x)
tanh	Unary Function	Hyperbolic tangent function	tanh(x)
tgh	Unary Function	Hyperbolic tangent function	tgh(x)
ctanh	Unary Function	Hyperbolic cotangent function	ctanh(x)
coth	Unary Function	Hyperbolic cotangent function	coth(x)
ctgh	Unary Function	Hyperbolic cotangent function	ctgh(x)
sech	Unary Function	Hyperbolic secant function	sech(x)
csch	Unary Function	Hyperbolic cosecant function	csch(x)
cosech	Unary Function	Hyperbolic cosecant function	cosech(x)
deg	Unary Function	Radians to degrees function	deg(x)
abs	Unary Function	Absolute value function	abs(x)
sgn	Unary Function	Signum function	sgn(x)
floor	Unary Function	Floor function	floor(x)

Key word	Category	Description	Example
ceil	Unary Function	Ceiling function	ceil(x)
not	Unary Function	Negation function	not(x)
asinh	Unary Function	Inverse hyperbolic sine function	asinh(x)
arsinh	Unary Function	Inverse hyperbolic sine function	arsinh(x)
arcsinh	Unary Function	Inverse hyperbolic sine function	arcsinh(x)
acosh	Unary Function	Inverse hyperbolic cosine function	acosh(x)
arcosh	Unary Function	Inverse hyperbolic cosine function	arcosh(x)
arccosh	Unary Function	Inverse hyperbolic cosine function	arccosh(x)
atanh	Unary Function	Inverse hyperbolic tangent function	atanh(x)
arctanh	Unary Function	Inverse hyperbolic tangent function	arctanh(x)
atgh	Unary Function	Inverse hyperbolic tangent function	atgh(x)
arctgh	Unary Function	Inverse hyperbolic tangent function	arctgh(x)
actanh	Unary Function	Inverse hyperbolic cotangent function	actanh(x)
arcctanh	Unary Function	Inverse hyperbolic cotangent function	arcctanh(x)
acoth	Unary Function	Inverse hyperbolic cotangent function	acoth(x)
arcoth	Unary Function	Inverse hyperbolic cotangent function	arcoth(x)
arccoth	Unary Function	Inverse hyperbolic cotangent function	arccoth(x)
actgh	Unary Function	Inverse hyperbolic cotangent function	actgh(x)
arcctgh	Unary Function	Inverse hyperbolic cotangent function	arcctgh(x)
asech	Unary Function	Inverse hyperbolic secant function	asech(x)

Key word	Category	Description	Example
arsech	Unary Function	Inverse hyperbolic secant function	arsech(x)
arcsech	Unary Function	Inverse hyperbolic secant function	arcsech(x)
acsch	Unary Function	Inverse hyperbolic cosecant function	acsch(x)
arsch	Unary Function	Inverse hyperbolic cosecant function	arsch(x)
arccsch	Unary Function	Inverse hyperbolic cosecant function	arccsch(x)
acosech	Unary Function	Inverse hyperbolic cosecant function	acosech(x)
arcosech	Unary Function	Inverse hyperbolic cosecant function	arcosech(x)
arccosech	Unary Function	Inverse hyperbolic cosecant function	arccosech(x)
sinc	Unary Function	Sinc function (normalized)	sinc(x)
Sa	Unary Function	Sinc function (normalized)	Sa(x)
Sinc	Unary Function	Sinc function (unnormalized)	Sinc(x)
Bell	Unary Function	Bell number	Bell(x)
Luc	Unary Function	Lucas number	Luc(n)
Fib	Unary Function	Fibonacci number	Fib(n)
harm	Unary Function	Harmonic number	harm(n)
ispr	Unary Function	Prime number test (is number a prime?)	ispr(n)
Pi	Unary Function	Prime-counting function - Pi(n)	Pi(n)
Ei	Unary Function	Exponential integral function (non-elementary special function) - usage example: Ei(x)	Ei(x)
li	Unary Function	Logarithmic integral function (non-elementary special function) - usage example: li(x)	li(x)
Li	Unary Function	Offset logarithmic integral function (non-elementary special function) - usage example: Li(x)	Li(x)

Key word	Category	Description	Example
erf	Unary Function	Gauss error function (non-elementary special function) - usage example: 2 + erf(x)	erf(x)
erfc	Unary Function	Gauss complementary error function (non-elementary special function) - usage example: 1 - erfc(x)	erfc(x)
erfInv	Unary Function	Inverse Gauss error function (non-elementary special function) - usage example: erfInv(x)	erfInv(x)
erfcInv	Unary Function	Inverse Gauss complementary error function (non-elementary special function) - usage example: erfcInv(x)	erfcInv(x)
ulp	Unary Function	Unit in The Last Place - ulp(0.1)	ulp(x)

mXparser - built-in Iterated Operators

Key word	Category	Description	Example
sum	Iterated Operator	Summation operator (SIGMA) sum(i, from, to, f(i,...) <,BY>)	sum(i, 1, 10, i^2), sum(i, 1, 10, i^2, 0.5)
prod	Iterated Operator	Product operator (PI) prod(i, from, to, f(i,...) <,BY>)	prod(i, 1, 10, i^2), prod(i, 1, 10, i^2, 0.5)
avg	Iterated Operator	Average operator avg(i, from, to, f(i,...) <,BY>)	avg(i, 1, 10, i^2), avg(i, 1, 10, i^2, 0.2)
vari	Iterated Operator	Bias-corrected sample variance operator vari(i, from, to, f(i,...) <,BY>)	vari(i, 1, 10, i^2), vari(i, 1, 10, i^2, 0.5)
stdi	Iterated Operator	Bias-corrected sample standard deviation operator stdi(i, from, to, f(i,...) <,BY>)	stdi(i, 1, 10, i^2), stdi(i, 1, 10, i^2, 0.01)
mini	Iterated Operator	Minimum value mini(i, from, to, f(i,...) <,BY>)	mini(i, 1, 10, i^2), mini(i, 1, 10, i^2, 0.3)
maxi	Iterated Operator	Maximum value maxi(i, from, to, f(i,...) <,BY>)	maxi(i, 1, 10, i^2), maxi(i, 1, 10, i^2, 0.4)

mXparser - built-in Calculus Operators

int	Calculus Operator	Definite integral operator ($\text{int}(f(x,...), x, a, b)$)	$\text{inf}(\sin(x), x, 0, \pi)$
der	Calculus Operator	Derivative operator ($\text{der}(f(x,...), x)$)	$\text{der}(\sin(x), x)$
der-	Calculus Operator	Left derivative operator ($\text{der}-(f(x,...), x)$)	$\text{der}+(\sin(x), x)$
der+	Calculus Operator	Right derivative operator ($\text{der}+(f(x,...), x)$)	$\text{der}-(\sin(x), x)$
dern	Calculus Operator	N-th derivative operator ($\text{dern}(f(x,...), x)$)	$\text{dern}(x^2, 2, x)$
diff	Calculus Operator	Forward difference operator $\text{diff}(f(x,...), x, \langle, h \rangle)$	$\text{diff}(\sin(x), x, 0.1)$, $\text{diff}(f(x), x, 0.1)$
difb	Calculus Operator	Backward difference operator $\text{difb}(f(x,...), x, \langle, h \rangle)$	$\text{difb}(\sin(x), x, 0.1)$, $\text{difb}(f(x), x, 0.1)$
int	Calculus Operator	Definite integral operator ($\text{int}(f(x,...), x, a, b)$)	$\text{inf}(\sin(x), x, 0, \pi)$

mXparser - built-in Mathematical Constants

Key word	Category	Description	Example
pi	Constant Value	Pi, Archimedes' constant or Ludolph's number	$2*\pi$
e	Constant Value	Napier's constant, or Euler's number, base of Natural logarithm	$e*3$
[gam]	Constant Value	Euler-Mascheroni constant	$2*[gam]$
[phi]	Constant Value	Golden ratio	$[phi]*3$
[PN]	Constant Value	Plastic constant	$2*[PN]$
[B*]	Constant Value	Embree-Trefethen constant	$[B*]*3$
[F'd]	Constant Value	Feigenbaum constant alfa	$2*[F'd]$
[F'a]	Constant Value	Feigenbaum constant delta	$[F'a]*3$
[C2]	Constant Value	Twin prime constant	$2*[C2]$
[M1]	Constant Value	Meissel-Mertens constant	$[M1]*3$

Key word	Category	Description	Example
[B2]	Constant Value	Brun's constant for twin primes	2*[B2]
[B4]	Constant Value	Brun's constant for prime quadruplets	[B4]*3
[BN'L]	Constant Value	De Bruijn-Newman constant	2*[BN'L]
[Kat]	Constant Value	Catalan's constant	[Kat]*3
[K*]	Constant Value	Landau-Ramanujan constant	2*[K*]
[K.]	Constant Value	Viswanath's constant	[K.]*3
[B'L]	Constant Value	Legendre's constant	2*[B'L]
[RS'm]	Constant Value	Ramanujan-Soldner constant	[RS'm]*3
[EB'e]	Constant Value	Erdos-Borwein constant	2*[EB'e]
[Bern]	Constant Value	Bernstein's constant	[Bern]*3
[GKW'l]	Constant Value	Gauss-Kuzmin-Wirsing constant	2*[GKW'l]
[HSM's]	Constant Value	Hafner-Sarnak-McCurley constant	[HSM's]*3
[lm]	Constant Value	Golomb-Dickman constant	2*[lm]
[Cah]	Constant Value	Cahen's constant	[Cah]*3
[LI]	Constant Value	Laplace limit	2*[LI]
[AG]	Constant Value	Alladi-Grinstead constant	[AG]*3
[L*]	Constant Value	Lengyel's constant	2*[L*]
[L.]	Constant Value	Levy's constant	[L.]*3
[Dz3]	Constant Value	Apery's constant	2*[Dz3]
[A3n]	Constant Value	Mills' constant	[A3n]*3

Key word	Category	Description	Example
[Bh]	Constant Value	Backhouse's constant	2*[Bh]
[Pt]	Constant Value	Porter's constant	[Pt]*3
[L2]	Constant Value	Lieb's square ice constant	2*[L2]
[Nv]	Constant Value	Niven's constant	[Nv]*3
[Ks]	Constant Value	Sierpinski's constant	2*[Ks]
[Kh]	Constant Value	Khinchin's constant	[Kh]*3
[FR]	Constant Value	Fransen-Robinson constant	2*[FR]
[La]	Constant Value	Landau's constant	[La]*3
[P2]	Constant Value	Parabolic constant	2*[P2]
[Om]	Constant Value	Omega constant	[Om]*3
[MRB]	Constant Value	MRB constant	2*[MRB]
[li2]	Constant Value	li(2) - logarithmic integral function at x=2	[li2]*3
[EG]	Constant Value	Gompertz constant	2*[EG]

mXparser - built-in Physical Constants

Key word	Category	Description	Example
[c]	Constant Value	<Physical Constant> Light speed in vacuum [m/s] (m=1, s=1)	[c]*3
[G.]	Constant Value	<Physical Constant> Gravitational constant (m=1, kg=1, s=1)	2*[G.]
[g]	Constant Value	<Physical Constant> Gravitational acceleration on Earth [m/s^2] (m=1, s=1)	[g]*3
[hP]	Constant Value	<Physical Constant> Planck constant (m=1, kg=1, s=1)	2*[hP]

Key word	Category	Description	Example
[h-]	Constant Value	<Physical Constant> Reduced Planck constant / Dirac constant (m=1, kg=1, s=1)]	[h-]*3
[lP]	Constant Value	<Physical Constant> Planck length [m] (m=1)	2*[lP]
[mP]	Constant Value	<Physical Constant> Planck mass [kg] (kg=1)	[mP]*3
[tP]	Constant Value	<Physical Constant> Planck time [s] (s=1)	2*[tP]

mXparser - built-in Astronomical Constants

Key word	Category	Description	Example
[ly]	Constant Value	<Astronomical Constant> Light year [m] (m=1)	[ly]*3
[au]	Constant Value	<Astronomical Constant> Astronomical unit [m] (m=1)	2*[au]
[pc]	Constant Value	<Astronomical Constant> Parsec [m] (m=1)	[pc]*3
[kpc]	Constant Value	<Astronomical Constant> Kiloparsec [m] (m=1)	2*[kpc]
[Earth-R-eq]	Constant Value	<Astronomical Constant> Earth equatorial radius [m] (m=1)	[Earth-R-eq]*3
[Earth-R-po]	Constant Value	<Astronomical Constant> Earth polar radius [m] (m=1)	2*[Earth-R-po]
[Earth-R]	Constant Value	<Astronomical Constant> Earth mean radius (m=1)	[Earth-R]*3
[Earth-M]	Constant Value	<Astronomical Constant> Earth mass [kg] (kg=1)	2*[Earth-M]
[Earth-D]	Constant Value	<Astronomical Constant> Earth-Sun distance - semi major axis [m] (m=1)	[Earth-D]*3
[Moon-R]	Constant Value	<Astronomical Constant> Moon mean radius [m] (m=1)	2*[Moon-R]
[Moon-M]	Constant Value	<Astronomical Constant> Moon mass [kg] (kg=1)	[Moon-M]*3
[Moon-D]	Constant Value	<Astronomical Constant> Moon-Earth distance - semi major axis [m] (m=1)	2*[Moon-D]
[Solar-R]	Constant Value	<Astronomical Constant> Solar mean radius [m] (m=1)	[Solar-R]*3
[Solar-M]	Constant Value	<Astronomical Constant> Solar mass [kg] (kg=1)	2*[Solar-M]

Key word	Category	Description	Example
[Mercury-R]	Constant Value	<Astronomical Constant> Mercury mean radius [m] (m=1)	[Mercury-R]*3
[Mercury-M]	Constant Value	<Astronomical Constant> Mercury mass [kg] (kg=1)	2*[Mercury-M]
[Mercury-D]	Constant Value	<Astronomical Constant> Mercury-Sun distance - semi major axis [m] (m=1)	[Mercury-D]*3
[Venus-R]	Constant Value	<Astronomical Constant> Venus mean radius [m] (m=1)	2*[Venus-R]
[Venus-M]	Constant Value	<Astronomical Constant> Venus mass [kg] (kg=1)	[Venus-M]*3
[Venus-D]	Constant Value	<Astronomical Constant> Venus-Sun distance - semi major axis [m] (m=1)	2*[Venus-D]
[Mars-R]	Constant Value	<Astronomical Constant> Mars mean radius [m] (m=1)	[Mars-R]*3
[Mars-M]	Constant Value	<Astronomical Constant> Mars mass [kg] (kg=1)	2*[Mars-M]
[Mars-D]	Constant Value	<Astronomical Constant> Mars-Sun distance - semi major axis [m] (m=1)	[Mars-D]*3
[Jupiter-R]	Constant Value	<Astronomical Constant> Jupiter mean radius [m] (m=1)	2*[Jupiter-R]
[Jupiter-M]	Constant Value	<Astronomical Constant> Jupiter mass [kg] (kg=1)	[Jupiter-M]*3
[Jupiter-D]	Constant Value	<Astronomical Constant> Jupiter-Sun distance - semi major axis [m] (m=1)	2*[Jupiter-D]
[Saturn-R]	Constant Value	<Astronomical Constant> Saturn mean radius [m] (m=1)	[Saturn-R]*3
[Saturn-M]	Constant Value	<Astronomical Constant> Saturn mass [kg] (kg=1)	2*[Saturn-M]
[Saturn-D]	Constant Value	<Astronomical Constant> Saturn-Sun distance - semi major axis [m] (m=1)	[Saturn-D]*3
[Uranus-R]	Constant Value	<Astronomical Constant> Uranus mean radius [m] (m=1)	2*[Uranus-R]
[Uranus-M]	Constant Value	<Astronomical Constant> Uranus mass [kg] (kg=1)	[Uranus-M]*3
[Uranus-D]	Constant Value	<Astronomical Constant> Uranus-Sun distance - semi major axis [m] (m=1)	2*[Uranus-D]
[Neptune-R]	Constant Value	<Astronomical Constant> Neptune mean radius [m] (m=1)	[Neptune-R]*3
[Neptune-M]	Constant Value	<Astronomical Constant> Neptune mass [kg] (kg=1)	2*[Neptune-M]

Key word	Category	Description	Example
[Neptune-D]	Constant Value	<Astronomical Constant> Neptune-Sun distance semi major axis [m] (m=1)	- [Neptune-D]*3