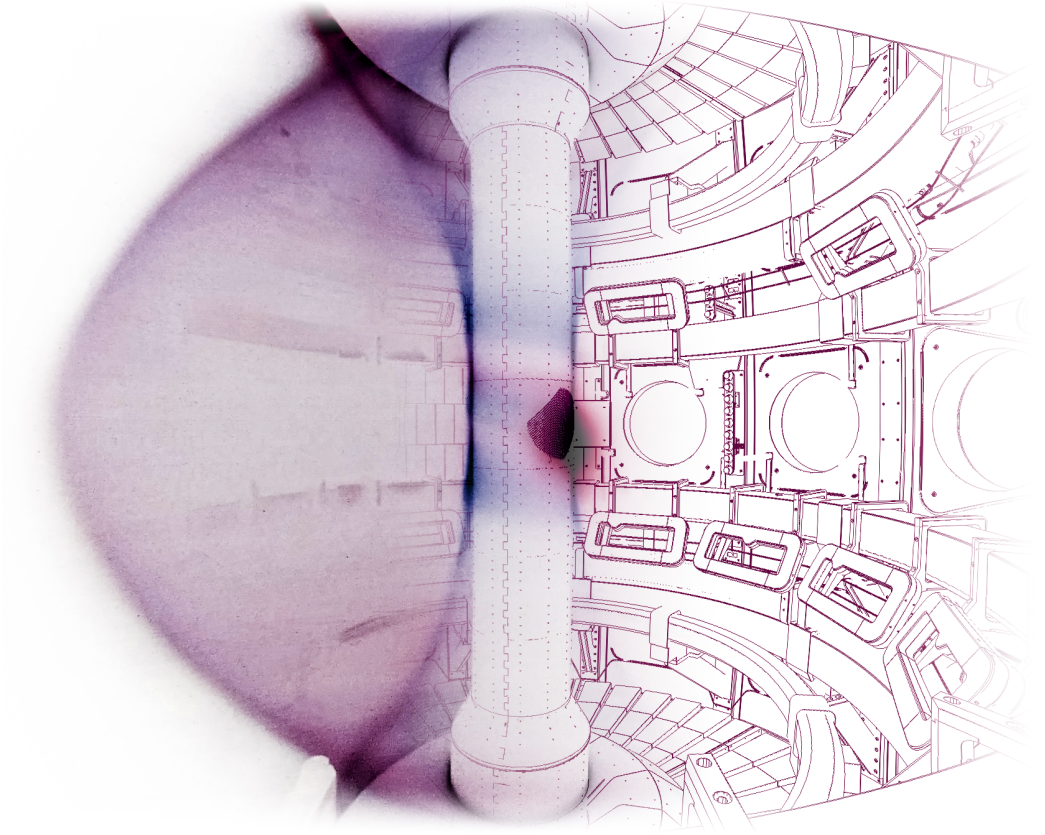

calcam

tools for camera spatial calibration and analysis

User Guide

Contents

1 Introduction & Background

1.1 Purpose & Scope of Calcam

Calcam is a python package whose purpose is to provide tools to spatially calibrate camera diagnostic syetems, i.e. to calibrate the mapping between pixel coordinates in camera data and 3D coordinates in real space in the lab. Calibrations are based on matching known features in camera images and features on a CAD model of the scene being observed. Calcam provides a graphical user interface for performing the calibration, and a Python API which provides tools to use these calibrations as part of your data analysis process.

1.2 Camera View Models

Calcam works by fitting a model which describes the relationship between 3D real-world coordinates and image coordinates. It supports two different models: one for "conventional" perspective projection lenses and one for "fisheye" projection lenses. In both cases, we wish to relate the coordinates of a point (X, Y, Z) in the lab frame to its pixel coordinates (x_p, y_p) in the camera image. First, we must consider the position and viewing direction of the camera in the lab frame, which is described by a 3D translation and rotation. The translation and rotation parameters are known as the *extrinsic* parameters in the model. Knowing these, we can apply a suitable translation and rotation to obtain the point of interest's coordinates in the *camera frame*: a 3D real space coordinate system where the camera pupil is at the origin and the camera looks along the positive Z axis. We denote the coordinates of our point of interest in the camera frame as (X', Y', Z').

In order to find the pixel coordinates of this point in the camera image, we start with a simple perspective projection, where the height of an object in the image is inversely proportional to its distance from the camera pupil:

$$\begin{pmatrix} x_n \\ y_n \end{pmatrix} = \begin{pmatrix} X'/Z' \\ Y'/Z' \end{pmatrix}. \tag{1}$$

The "normalised" coordinates (x_n, y_n) are then transformed by a model which describes the image distortion due to the optical system. This model depends on the lens projection being assumed, and models for the perspective and fisheye lens types are described in the following sections. Here we simply

denote the resulting distorted normalised coordinates as (x_d, y_d). Finally, the normalised, distorted coordinates are related to the actual pixel coordinates x_p, y_p in the image plane by multiplication with the "camera matrix":

$$\begin{pmatrix} x_p \\ y_p \\ 1 \end{pmatrix} = \begin{pmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_d \\ y_d \\ 1 \end{pmatrix}. \tag{2}$$

Here f_x and f_y are the effective focal length of the imaging system measured in units of detector pixels in the horizontal and vertical directions, and are expected to be equal for square pixels and non-anamorphic optics. c_x and c_y are the pixel coordinates of the centre of the perspective projection on the sensor, expected to be close to the detector centre. The parameters in the camera matrix, along with those describing the distortion model, constitute the *intrinsic* camera parameters, i.e. they are characteristic of the camera and optical system and are independent of how that system is placed in the lab.

1.2.1 Perspective Lens Distortion Model

The image distortion model for perspective projection lenses takes in to account radial (barrel or pincushion) distortion, and tangential (wedge-prism like, usually due to de-centring of optical components) distortions. The equation relating the undistorted and distorted normalised image coordinates in this model is:

$$\begin{pmatrix} x_d \\ y_d \end{pmatrix} = \left[1 + k_1 r^2 + k_2 r^4 + k_3 r^6\right] \begin{pmatrix} x_n \\ y_n \end{pmatrix} + \begin{pmatrix} 2p_1 x_n y_n + p_2(r^2 + 2x_n^2) \\ p_1(r^2 + 2y^2) + 2p_2 x_n y_n \end{pmatrix}, \tag{3}$$

where $r = \sqrt{x_n^2 + y_n^2}$, and k_n and p_n are radial and tangential distortion coefficients, respectively. The polynomial in r^2 in the first term describes the radial distortion while the second term represents tangential distortion.

1.2.2 Fisheye Lens Distortion

The fisheye distortion model only includes radial fisheye distortion. Unlike the perspective projection model, the polynomial describing the radial distortion is a function of an anglular distance from the centre of perspective, rather than a linear distance in the image:

$$\begin{pmatrix} x_d \\ y_d \end{pmatrix} = \frac{\theta}{r}\left[1 + k_1\theta^2 + k_2\theta^4 + k_3\theta^6 + k_4\theta^8\right]\begin{pmatrix} x_n \\ y_n \end{pmatrix}, \tag{4}$$

where $r = \sqrt{x_n^2 + y_n^2}$ and $\theta = \tan^{-1}(r)$.

1.3 Image Coordimate Conventions

Calcam follows the convention of using matrix/linear algebra style pixel coordinates for images, which is consistent with the way images are stored and addressed in 2D arrays. In this convention, the origin $(0, 0)$ is in the centre of the pixel in the top left corner of the image. The y axis runs from top to bottom down the image and the x axis runs horizontally left to right. It is important to note that since 2D arrays are indexed `[row, column]`, arrays containing images are indexed as `[y,x]`. However, functions which deal with image coordinates are called as `function(x,y)`. This is consistent with the way image coordinates are delt with in OpenCV.

1.3.1 Original and Display Coordinates

In camera diagnostics, the data comes out of the camera with some intrinsic orientation which may not be the 'right way up' in terms of easily looking at or understanding the images, e.g. if the camera is mounted upside down or the image is flipped by a mirror on the optical path. It is therefore desirable to work with the image transformed to be the 'right way round'. However, when performing bulk, programmatic analysis of video data it is not necessary or advantageous to perform this transformation, and is more efficient to work with the raw data as it comes out of the camera or is stored. Calcam therefore uses the concept of 'Display' and 'Original' image coordinates (thanks to Valentina Huber for this idea, and coining the phrase). Consider an object at a specific point in an image. Its coordinates in the camera image, as read straight from the camera, are the 'original' coordinates. If the image is transformed to be the 'right way up', the object's coordinates in the image will now be different. These coordinates in the 'right way up' image are called 'Display' coordinates. Calcam keeps track of the transofmration between original and display coordinates, so that the results can be used with raw data as it comes from the camera as well as processed 'right way up' data. It is therefore highly recommended that you do not perform any geometrical transformations to images before loading them in to calcam for calibration,

but make all these adjustments within calcam. This will leave the option open for using the raw data more efficiently in your analysis.

By default, calcam functions work with display coordinates unless specified otherwise. This is because the underlying openCV camera model fitting requires that the image being calibrated is not flipped, horizontally or vertically (this would add a minus sign to the pinhole projection which is not supported by the code). This is also an important point when calibrating images: ensure the image is not horizontally or vertically flipped when performing calibrations!

1.4 About this User Guide

This is the initial, somewhat rough and incomplete version of the Calcam user guide, written by Scott Silburn (`scott.silburn@ukaea.uk`). Feedback on both the code and the user guide are welcome. The installation and setup and API documentation sections assume a reasonable familiarity with the Python language.

2 Installation, First Import & Configuration

2.1 Hardware Recommendations

Since working with Calcam involves interactively working with CAD data, using a system with good OpenGL rendering performance makes the experience much more pleasent. Currently Calcam is only single-threaded, so there is no particular advantage to having more CPU cores. It is recommended to use calcam "locally" rather than over a remote connection or on a remote cluster, for 2 reasons: 1) From experience, OpenGL / VTK can be more tricky to get running, and 2) Performance when working with mesh data is typically not very good.

2.2 Software Environment Requirements

Calcam is compatible with Python versions 2.7 - 3.5. Since working with large images and CAD models can quickly become memory intensive, it is highly recommended to use 64-bit Python where possible. In addition to Python itself, Calcam requires the following modules / packages to be installed:

- **NumPy**
- **SciPy**
- **MatPlotLib**
- **VTK 6.0 or newer**
- **OpenCV 2.4 or newer (fisheye calibration only available with 3.0 or newer)**
- **PyQt 4.5 or newer**

Typically, VTK is the requirement which causes the most headaches in getting Calcam up and running. In future versions, specific VTK test scripts will be included to help troubleshooting this. On Windows, the easiest way to satisfy all of the above requirements is to install the Python(x, y) package and make sure the above modules are selected at installation time. If building VTK from source on your system, as well as the Python bindings, VTK must be built with Qt support.

2.3 Installation, Importing & Setup

Place the calcam directory containing the python files in your location of choice, preferably not in the root of your home directory (usually abbreviated in this document as '~') since `~/calcam` will be used to store user data. It is highly recommended to add the path containg the calcam folder to your `PYTHONPATH` environment variable, i.e. if you copy the calcam folder to `~/Pythom/calcam`, make sure `~/Python` is in your `PYTHONPATH`.

With your `PYTHONPATH` set up correctly, or in Python with the working directory in the same location as the `calcam` folder, the calcam module can be imported using `import calcam`. The first time calcam is imported by a given user, calcam will create directories for storing calibration input, calibration results and user defined code in your home directory (Usually `C:\users\<username>\calcam` on Windows or `/home/<username>` on Unix.). Once these have been created and you have checked for any warnings about missing required modules, you can move on to configuring CAD models using the templates provided (see next section).

2.3.1 Configuring CAD models

In the current version of Calcam, only .stl and .obj format 3D mesh files are supported (more formats will be added in future versions). It is suggested to break down the CAD model in to separate mesh files for different components or groups of components, and CalCam will allow each file to be included or excluded when working with the CAD model. In order to use a CAD model in Calcam, you must create a definition for the CAD model in a Python file in the foler `~/calcam/UserCode/machine_geometry/`. A detailed template for a CAD model defition is provided in `~/calcam/UserCode/machine_geometry/Example.py`, which will be created when Calcam is first imported. To define CAD models for use in calcam, please refer to this file to create your CAD model definitions.

2.3.2 Configuring Image Sources

As standard, Calcam can load camera images from all common image file formats. However, it may also be desirable to define your own source of images, e.g. fetching images from some central data store based on a pulse number, camera identifier etc. This can be achieved by defining custom image sources in python files in `~/calcam/UserCode/image_sources/`. A detailed, working template is procided in `~/calcam/UserCode/image_sources/Example.py`, which will be created when CalCam is first imported.

3 Performing Calibrations: The CalCam GUI

The CalCam GUI is used to perform the camera spatial calibration. The GUI can be started in two ways: from inside python with the calcam module imported, by calling the function `calcam.gui.start_calcam()`, or by executing the file `calcam/gui.py` as a Python script (i.e. if you have put calcam into `~/Python/calcam`, executing `python ~/Python/calcam/gui.py` from a terminal).

You should be greeted with a window which looks like figure 1. The main features of this window are, with reference to the numbers on the figure:

1. **CAD Display:** the left half of the initially black area will be used to display the CAD model.

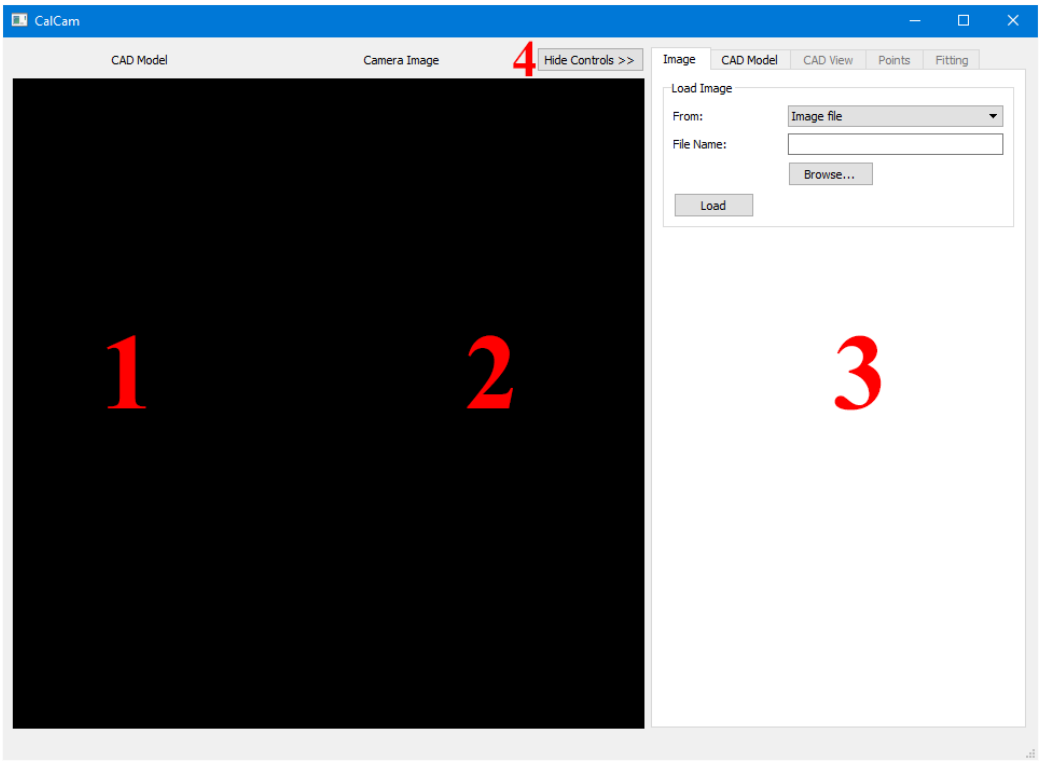

Figure 1: Calcam main window.

2. **Image Dipslay:** the right half of the initially black area will be used to display the camera image you are calibrating.

3. **Control tabs:** At the right-hand side of the window are a set of tabs which provide the controls for the program. These will be delt with in detail in the next sections. When the GUI is first loaded, only the Image and CAD model tabs will be enabled.

4. **Show / hide controls button:** Particularly on smaller screens, it is sometimes desirable to hide the control tabs to have a larger view of the image and CAD model while working. This button hides the control tabs, leaving the full window area for the CAD and image views. Clicking this button again re-opens the control tabs.

The following sections provide details on each step of performing a calibration.

3.1 Loading and adjusting camera images

The first step in performing a calibration is to load the camera image you wish to calibrate in to Calcam. The 'Image' tab which is used to do this is open by default when the Calcam GUI starts. This tab is used to load an image or adjust the currently loaded image, and is shown in figure 2.

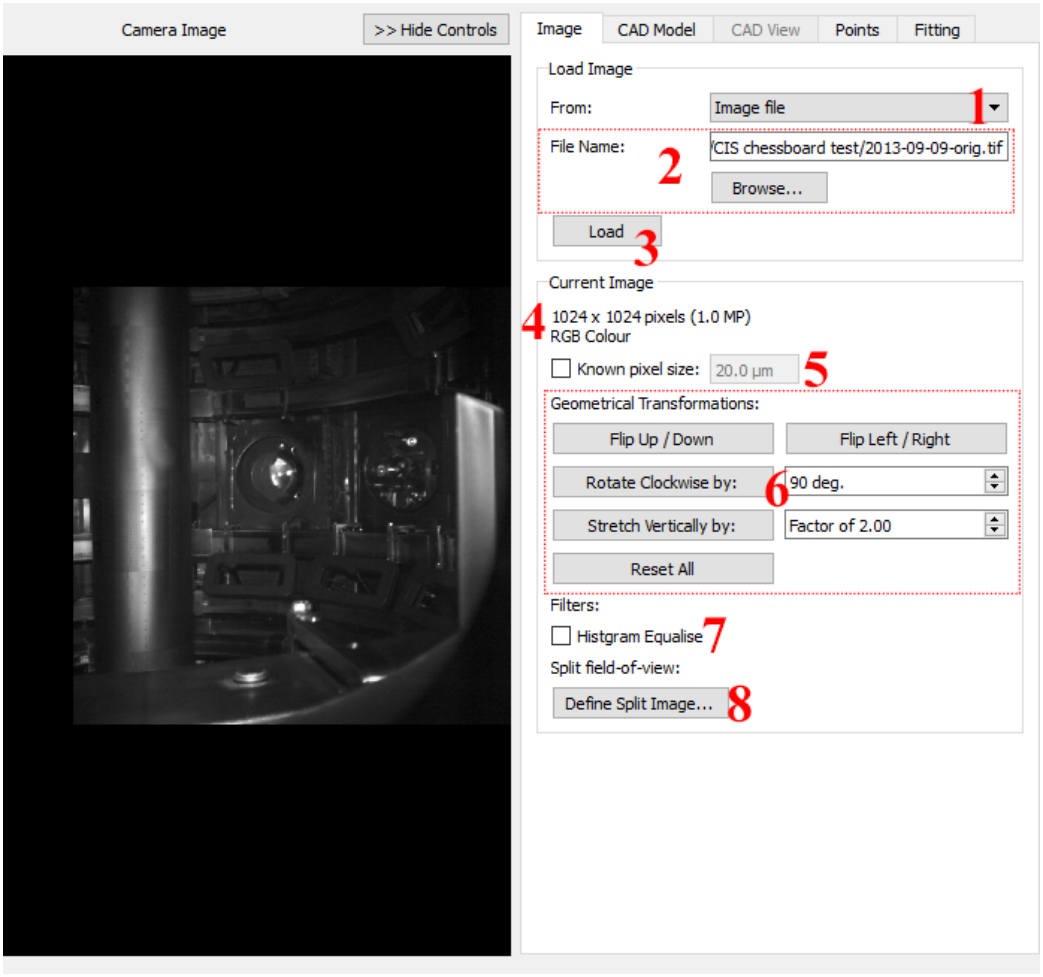

Figure 2: The image Tab.

3.1.1 Loading Images

At the top of the 'Image' tab is a group of controls for loading images in to Calcam. This contains controls for loading an image:

1. **"From" dropdown box**
   Selects where the image should be loaded from (what image source should be used). The default is 'Image File' which allows an image to be loaded from a normal image file. The other built-in option available is 'Previously Used Image' which allows quick loading of an image used before in Calcam. If you have any user defined image sources (as described in section 2.3.2) they will also appear as an option here.

2. **Image Load Options**
   The controls here set the options for loading an image, and the displayed controls depend on the image source selected. In the case of loading an image file, a file name box appears which can be filled in manually or populated using the 'Browse...' button to select a file.

3. **Load Button**
   Loads the image according to the options set above. The loaded image is displayed in the image view as shown in figure 2. Once the image is loaded, the 'Current Image' section appears, and the Points and Fitting tabs are enabled.

Remember, as explained in section 1.3.1, it is highly recommended not to perform any rotation, flipping etc of the raw image before loading it, but instead to do these actions within calcam, as explained in the next section.

3.1.2 Adjusting Images

With an image loaded, the Image control tab contains a 'Current Image' control group containing information and settings for the current image. If the image from the camera is not already the 'right way up', this can be corrected here. If the camera system has multiple fields-of-view on the same detector, this can also be defined in this section. The controls in the 'Current Image' section are:

4. **Image Information**
   The pixel resolution and colour type are displayed at the top of the

current image section. If the image dimensions are different in display and original coordinates (see section 1.3.1), both are displayed.

5. **Pixel Size**
   If the pixel size of the detector is known, it can be entered here. This is optional and does not effect the calibration except that calibrated focal lengths can be displayed in real length units rather than pixels. This is useful if you know what the effective focal length of the optical system should be, and want to sanity check the calibration.

6. **Geometrical Transformation Buttons**
   These buttons provide controls for rotating, flipping etc the image. If using these controls when some calibration points have already been defined, the calibration points are transformed with the image.

7. **Histogram Equalisation**
   This checkbox activates local histogram equalisation. This increases the contrast of features in the image, making them easier to identify and can therefore make it easier to accurately identify calibration points.

8. **Split field-of-view**
   Some camera diagnostics may have a split field-of-view, i.e. multiple, different views of the machine on the same detector, for example in optical systems using beam splitters or mirror boxes in the optical path. In this case, each sub field-of-view is calibrated independently. In order to do this, you must tell calcam which image pixels belong to which view. The 'Define Split Image' button opens a dialog where you can define the split field-of-view. Full documentation for this dialog will be added later.

3.2 Loading and working with CAD models

Once the image to be calibrated has been loaded, the next step is to load the CAD model against which to calibrate the image. Calcam has two tabs concerned with the CAD model, described in the following sections.

3.2.1 CAD Model Tab

The CAD Model tab is used to load or change CAD models, and turn on or off individual CAD model features in the view. This is shown in figure 3.

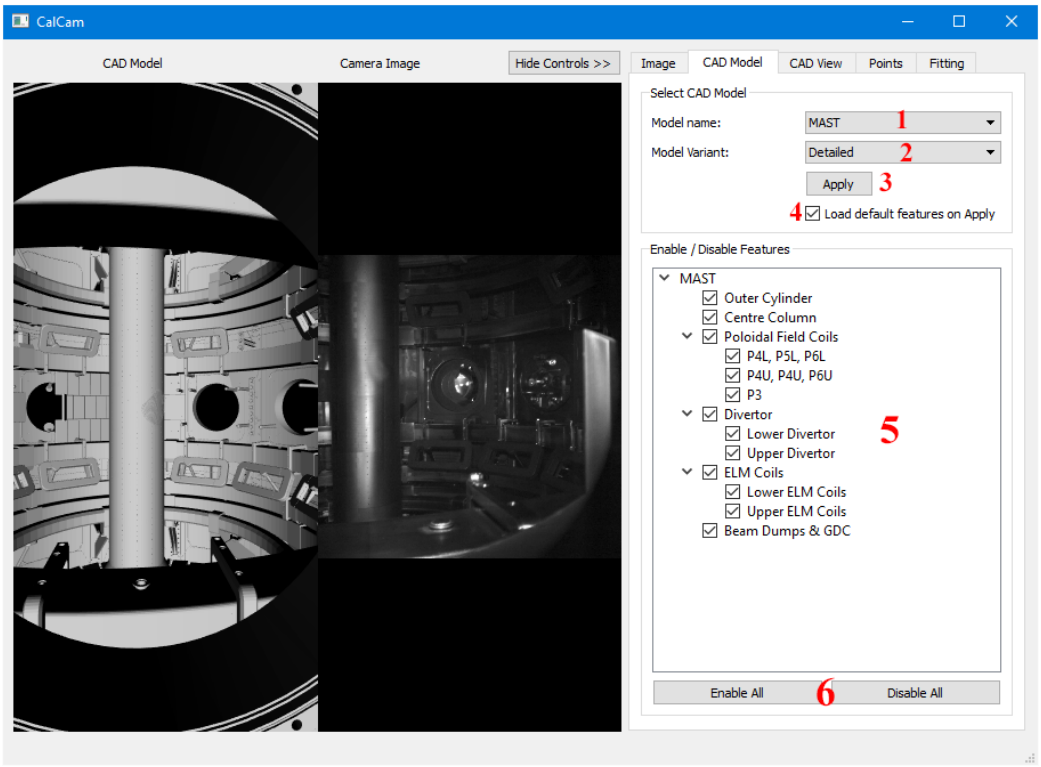

Figure 3: CAD Model Tab.

The controls on this tab are:

1. **"Model Name" dropdown box**
   This is used to select the CAD model to load, and lists all the CAD models which are configured (section 2.3.1 describes how to configure CAD models). When a model is selected, the "Model Variant" dropdown box is populated with the available variants of the model, and set to the default variant specified in the CAD model definition.

2. **"Model Variant" dropdown box**
   This is used to select a particular CAD model variant (as defined in the model configureation).

3. **Apply Button**
   Loads the selected CAD model.

4. **"Load default features on Apply" Checkbox**
   Controls what happens when the Apply button is clicked. If checked, which is the default, the Apply button will load all of the default features of the selected CAD model in to the CAD view. This can be undesirable for performance reasons if running on low specification systems when using large CAD models. If this box is unchecked, clicking Apply will populate the "Enable / Disable Features" box without loading the CAD data itself, allowing the user to load only the parts of the model they require.

5. **Feature Selection**
   When a CAD model is loaded, this box ontains a checkable tree of all the individual "features" in the model. Checking or un-checking a feature, or group of features, will turn it on or off in the CAD view. This is useful to improve performance if not all parts of the CAD model are required, or if otherwise it is convenient to change which parts of the model are loaded.

6. **Enable / Disable All buttons**
   These buttons enable or disable all the CAD model features in the above features list simultaneuosly.

3.2.2 CAD View Tab

The CAD View tab is enabled when there is a CAD model loaded, and can be used to control or see information about the current CAD viewport. The view can be changed either interactively with the mouse (as described in the next section) and/or using the controls on this tab. The CAD View tab is shown in figure 4, and has the following elements:

1. **CAD Defined Viewports**
   This section lists all of the views configured in the CAD model definition. Clicking the name of a view sets the current CAD view to the clicked one.

2. **Calibration Results**
   This lists existing calcam calibration results. Clicking the name of a result will set the current CAD view to match the results of that calibration.

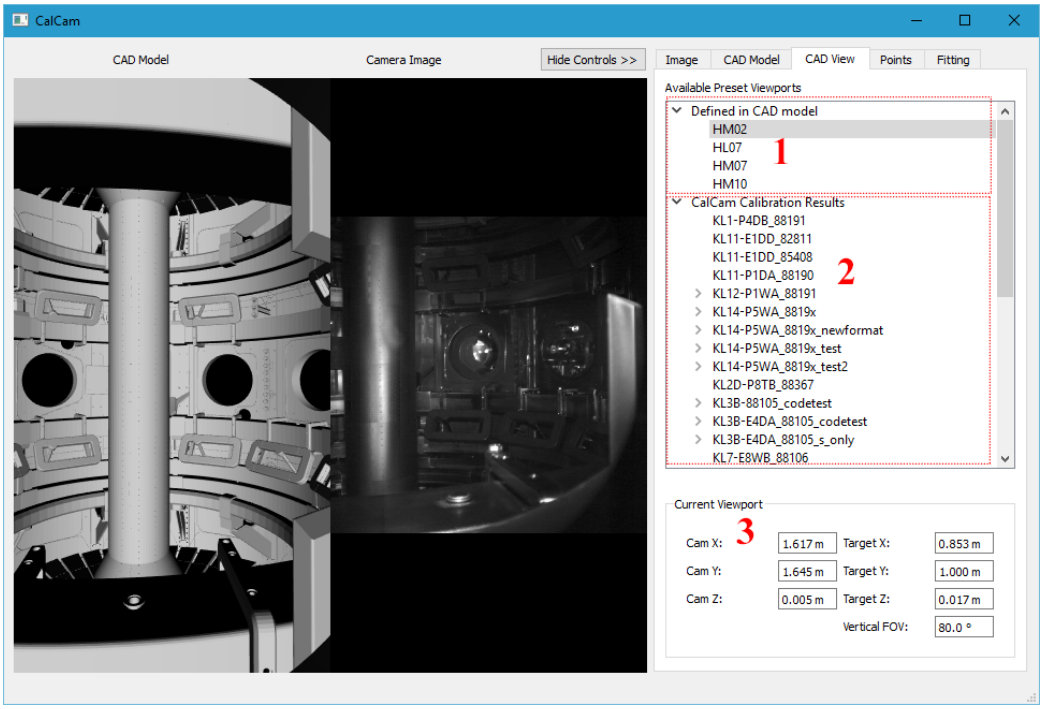

Figure 4: The CAD View tab.

3. **Viewport Data**
   This section contains boxes showing the current CAD camera position coordinates, viewing target coordinates (the point towards which the camera is lookling), and CAD vertical field of view. These can be edited manually to change the CAD view or are updated to show the current values when the view is changed by another means.

3.3 Defining points for calibration fitting

With a CAD model and image loaded, you can begin to define calibration points by identifying matching features in the CAD model and camera image. In order to easily navigate around the image and CAD model to precisely place points, both the CAD and image views can be navigated with mouse controls.

3.3.1 CAD View Mouse Controls

The following controls can be used to navigate around the CAD model, with the mouse over the CAD view:

- **Right Click + Drag**
  Look around (as in typical first person game controls)
- **Middle Click + Drag**
  Pan (translate) the camera sideways
- **Scroll Wheel up / down**
  Move the camera forwards or backwards
- **Ctrl + Scroll Wheel up / down**
  Reduce / increase the camera field of view

3.3.2 Image View Mouse Controls

The following controls can be used to navigate around the image view, with the mouse over the image:

- **Middle Click + Drag**
  Pan around the image
- **Scroll in / out**
  Zoom in / out of current mouse pointer location

3.3.3 Adding or modifying Point Pairs

Calcam uses 'point pairs' to perform the calibration, where a point pair consists of one point on the CAD model and its corresponding point on the image. Point pairs are displayed on the CAD and image views as red + cursors at the point locations. At any given time, one pair of points can be selected for editing. The selected point pair will be indicated with larger green + cursors.

Once you have identified a common feature on the image and CAD model, `Ctrl + Click` on the location on either the image or CAD view to create a new point pair. A point will be placed at the mouse location. Then click (without holding `Ctrl`) the corresponding point on the other view to finish creating the point pair. You should now see green + cursors on both the

CAD model and image. Clicking either the CAD model or image again will move the green cursor representing the current point to the clicked location. To start another point pair, `Ctrl + Click` again and repeat the process. The cursors showing the existing points will turn red, indicating they are no longer selected. In general, left clicking on either the image or CAD model will move the currently selected point to the clicked location. Clicking an existing cursor will select that point pair for editing, and holding `Ctrl` while clicking will start a new point pair.

If you start a new point pair but do not specify both CAD and image points (e.g. by `Ctrl+Click`ing on the image twice in a row), this will create 'un-paired' points which will be ignored when setting up the fit. The cursors of these points will be displayed in yellow. You can go back to these later to add their corresponding point by clicking on the yellow cursor to select it, then clicking on the corresponding point in the other view.

The currently selected point pair can be deleted by pressing the `Del` key on the keyboard, or clicking the "Remove current point pair" button on the Points control tab.

3.3.4 Points Control Tab

The points control tab contains 3 groups of controls used for loading, saving, inspecting and editing the calibration points, and is shown in figure 5. The controls are:

1. **Load Saved Point Pairs**
   This allows loading of previously saved point pairs. The dropdown box contains a list of previously saved point pairs which can be loaded for use with the current image. By default, any existing point pairs are removed when the 'Load' button is clicked, however un-checking the 'Clear current pairs first' option will cause the loaded points to be added to any existing ones instead.

2. **Current Points**
   This section contains information and options about the currently defined point pairs. At the top of this section, the number of point pairs currently defined is shown (and the number of un-paired points, if there are any). Just below this is information about the currently selected point pair, which usually displays the CAD and image coordinates of the points (The information displayed about the CAD point location

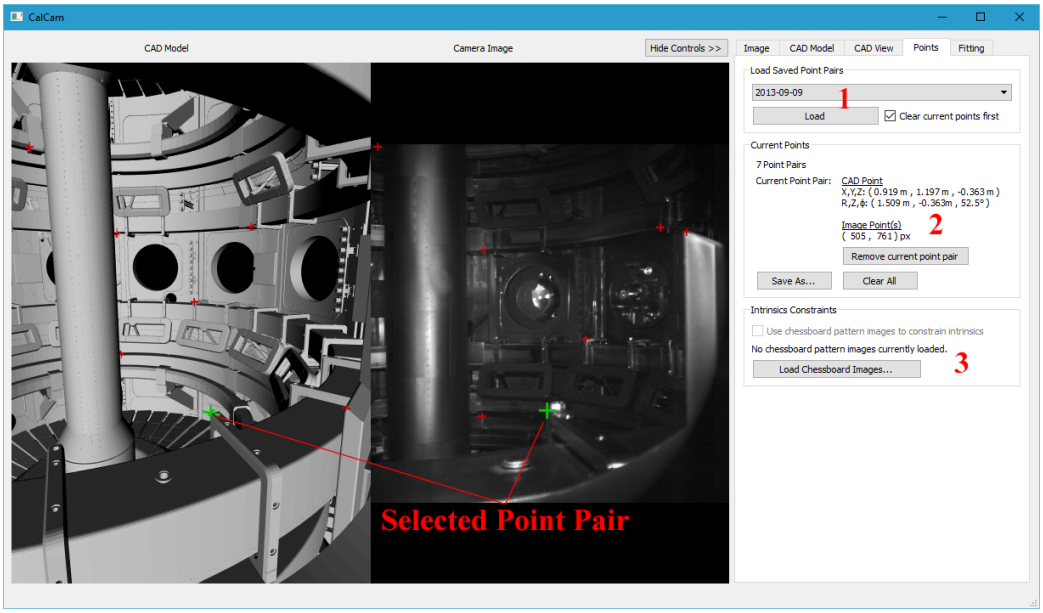

Figure 5: The Points tab.

can be customised by defining a custom `get_position_info` method for your CAD model definitions, see section 2.3.1). The "Remove current point pair" button removes the currently selected point pair, while the "Clear All" button deletes all existing point pairs. The "Save As..." button allows the current point pairs to be saved for later use.

3. **Intrinsics Constraints**
   In many examples of real images, only a relatively small number of point pairs can be accurately identified. Due to the large number of free parameters in the camera model fit (focal length(s), centre of perspective and distortion parameters) this can give poor quality results. It is possible to better constrain the fits by using images of a chessboard pattern, with known square size, taken with the same camera + optical system configuration. This adds aditional constraints on the intrinsic model parameters, meaning only enough points to reliably fit the extrinsic parameters need to be identified in the image of the machine. Clicking the "Load Chessboard Images..." button opens a dialog box where you can load chessboard images to use as additional constraints. Once chessboard images have been loaded, they can be included or ex-

cluded from the fitting using the checkbox in this section. This user guide will later include guidelines on preparing chessboard images; for the time being, suitable advice can be found by searching for OpenCV camera calibration with chessboard images or the MATLAB camera calibration toolbox.

3.4 Performing, checking and saving fits

Once you have defined a set of point pairs, fitting is performed with the controls in the Fitting tab, shown in figure 6. Before performing a fit, this tab contains only the 'Fit Options' section, with the 'Fit Results' section appearing once a fit has been performed.

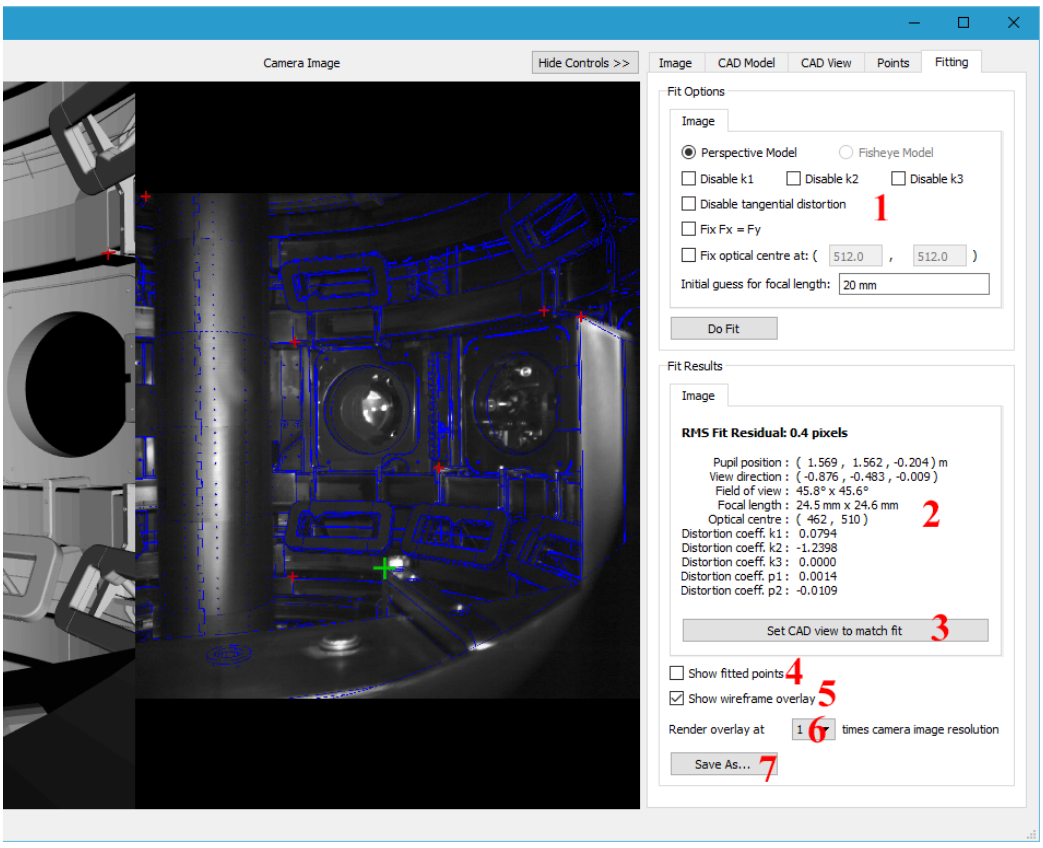

Figure 6: The fitting control tab.

3.4.1 Setting Fit Options and performing fits

Fit options are adjusted using the top section on the Fitting control tab, marked (1) in figure 6. The default options will typically produce good results for most images, however in some cases they will need to be adjusted to get a good quality result. For images with split fields of view, since each sub-field is fitted separately, tabs are displayed containing independent controls for each sub-field's fit options.

The first option to choose is whether to use the perspective or fisheye projection model: these two can be switched using the radio buttons at the top of the fit options section. The detailed options presented then depend on which model is selected:

Perspective Model Fit Options

- **Disable k_1...k_3**
  These options, when checked, set the corresponding coefficients in equation (3) to be fixed at 0 in the fit. This changes the order of the radial distortion model (and disables radial distortion entirely if all three are checked). Disabling higher order radial distortion terms can improve fits when the point pairs do not sufficiently constrain the distortion model, when the fitted results can have large erroneous distortions.

- **Disable Tangential Distortion**
  This option sets the coefficients p_1 and p_2 in equation (3) to be fixed at 0 in the fit, i.e. disables tangential distortion in the fitted model. This can be helpful if the fitting results in large erroneous values of these coefficients.

- **Fix $f_x = f_y$**
  This option fixes the focal lengths in the horizontal and vertical directions to be equal, i.e. fixes the image aspect ratio to 1. This is enabled by default, since for square pixels and non-anamorphic optics, which is the typical case, $f_x = f_y$ is expected. Un-checking this option can sometime help fit quality for some optical systems.

- **Fix Optical Centre**
  This option fixes the location of the centre of perspective at the specified pixel coordinates. I'm not sure why you would ever want to use this,

but since it's possible in the underlying OpenCV fitting, I thought I'd include the option.

- **Initial Guess for Focal Length**
  This is the initial guess for the focal length used when starting the fit. The OpenCV fitter seems quite robust to values far from the final result, and the default value has been chosen to work well for most test images. However, there may be some cases where it is desirable to manually set the initial guess for the focal length for the fitter to find the correct solution.

Fisheye Model Fit Options Note: fisheye model fits are only available if you are using OpenCV 3.

- **Disable k_1...k_4**
  These options, when checked, set the corresponding coefficients in equation (4) to be fixed at 0 in the fit, changing the order of the fisheye distortion model.

- **Initial Guess for Focal Length**
  This is the initial guess for the focal length used for the fitting. The OpenCV fitter seems quite robust to values far from the final result, and the default value has been chosen to work well for most test images. However, there may be some cases where it is desirable to manually set the initial guess for the focal length.

To perform a fit using the current fit options, click the "Do Fit" button underneath the fit options. Alternatively, the keyboard shortcut `Ctrl+F` also performs a fit with the current settings.

3.4.2 Checking fit quality and saving fits

As soon as a fit is performed, the fitted points are shown on the image as blue + cursors. These are the current CAD model points, converted to image coordinates using the fitted model, i.e. for a good fit these should lie on top of the current image points. The fitted points can be turned on or off using the "Show Fitted Points" checkbox (4), or pressing `Ctrl+P` on the keyboard. The RMS fit error and fitted extrinsic and intrinsic parameters (camera pupil position and view direction, field of view, focal length, centre of perspective

and distortion parameters) are dislayed in section (2). As with fit options, if the image has a split field-of-view, results for each field of view are shown on separate tabs. Note: for fits with small numbers of points, the camera model has sufficiently many free parameters that a very small RMS fit error can be obtained with a fit which is actually very bad!

A more robust visual check of the fit quality can be obtained by overlaying the CAD model in wireframe on top of the camera image, according to the fit results. This is be done by checking the "show wireframe overlay" box (5) or pressing `Ctrl+O` on the keyboard. The CAD model is then rendered in wireframe and superimposed on the image. For large images or CAD models this can be very slow and memory intensive, and to improve the rendering speed or avoid memory errors, the resolution of the overlay can be reduced using the dropdown box (6). Conversely, the resolution of the overlay can be increased to allow more detail to be seen. It is also possible to set the current CAD viewport to approximate the fitted model using the 'Set CAD view to match fit' button, which can be useful both for checking the fit and helping to identify point pairs.

Once a satisfactory fit has been obtained, the results can be saved using the 'Save As...' button (7) at the bottom of the panel. This will also save the point pairs used for the fit, if this has not been done already. Once a satisfactory fit has been saved, this completes the calibration process using the calcam GUI.

4 Using Calibrations: API Reference

Once you have performed one or more calibration fits and saved the results, the Calcam Python API provides tools to make use of the results. The following sections document the most important modules and classes in Calcam for doing this.

4.1 The `calcam.CalibResults` Class

The class `calcam.CalibResults(fit_name)` is used to represent calibration results, and has a number of methods for extracting useful information from the fitted model. To create a `CalibResults` instance representing a result saved with the name "`foo`" (this is the name used in the 'Save As...' box when saving the result), initialise a `CalibResults` instance with

the name as an argument: `MyFit = calcam.CalibResults(''fit_name'')`. This searches for a file `~/calcam/FitResults/fit_name.pickle`, which is where clacam stores its fit results, and loads the results stored in that file. A summary of the fit results, similar to that displayed in the Calcam GUI after fitting, can be seen by using the print statement on the resulting object, i.e. `print(MyFit)`. The main useful methods of the `CalibResults` class are:

`get_pupilpos([x_pixels,y_pixels,field,Coords])`

Get the camera pupil position, i.e. the origin of the camera's sight-lines, in the lab (i.e. CAD model) coordinate system.

Inputs

For the common case of an image without a split field of view, no arguments are necessary and this can be called as `get_pupilpos()`. For images with a split field-of-view, optional input arguments are:

- `x_pixels=`*None*`, y_pixels=`*None*
  Array-likes of floats containing the X and Y pixel coordinates you want to know the pupil position for. This is only useful or necessary for optical systems with split fields-of-view, where not all pixels necesserily have the same pupil position. Eiether this or `field` must be specified for split field-of-view images.
- `field=0`
  For split field-of-view cameras, an integer specifying the number of the sub-field you want the pupil position for.
- `Coords='Display'`
  String, either 'Display' or 'Original': if specifying x_pixels and y_pixels, this specifies whether the input x and y pixel coodinates are in display or original coordinates.

Outputs

- 3-element array of floats containing the camera pupil position `(X,Y,Z)` in metres.

get_fov([field])

Get the horizontal and vertical field of view of the camera in degrees.

Inputs

For the common case of an image without a split field of view, no arguments are necessary and can be called as `get_fov()`. For images with a split field-of-view, optional input arguments are:

- `field=0`
  Index of the sub field-of-view for which to get the view angles.

Outputs

- Two element array containing the `(horiz,vert)` field-of-view in degrees.

get_los_direction([x_pixels,y_pxels,Coords])

Get unit vectors representing the directions of the camera's sight-lines in the lab (CAD Model) coordinate system.

Inputs

If called with no input arguments, this function returns the sight-line directions of every pixel in the image. Optional inputs are:

- `x_pixels`=*None*,`y_pixels`=*None*
  Pixel coordinates for which to return the lines of sight. If none are given, results are returned for all pixels in the image. `x_pixels` and `y_pixels` can be any shape, as long as they are the same shape as each other.
- `Coords='Display'`
  String, either 'Display' or 'Original': if specifying `x_pixels` and `y_pixels`, this specifies whether the input x and y pixel coodinates are in display or original coordinates. If `x_pixels` and `y_pixels` are not given, this controls whether the output corresponds to the image in display or original coordinates.

Outputs

- NumPy array containing 3D unit vectors whose directions are the sight-line directions in the lab frame. If no pixel coordinates are given this will be a [hxwx3] array, where w and h are the image width and height, and the third axis contains the (X,Y,Z) components of the sight-line unit vectors. If `x_pixels` and `y_pixels` are given, the output array will be the same shape as `x_pixels` and `y_pixels` with an additional axis added to store the 3 components of the sight-line vectors (i.e. if x_pixels is a 5x5 array, the output will be 5x5x3).

`project_points(ObjPoints,[CheckVisible,RayData,RayCaster, VisibilityMargin,Coords])`

Project given 3D points in the lab frame in to pixel coordinates on the image. Optionally, checks whether the given points are visible in the image or occluded by geometry in the CAD model.

Inputs

- `ObjPoints`
  3D point coordinates to project on to the detector. These can be specified by EITHER an Nx3 array, where N is the number of points, or an array-like of 3 element arrays, where each 3 element array specifies a point.
- `CheckVisible`=*False*
  Bool, whether to check if the 3D points are visible to the camera or hidden behind CAD geometry. If this is set to `True`, either `RayData` or `RayCaster` must also be specified.
- `RayData`=*None*
  A `calcam.RayData` object containing sight-line information for this view, used for checking point visibility if this is requested. If both `RayData` and `RayCaster` are specified, this will not be used.
- `RayCaster`=*None*
  A `calcam.raytrace.RayCaster` object for this camera view and CAD model, used for checking point visibility if this is requested. If both `RayData` and `RayCaster` are specified, this will override `RayData`.

- `VisibilityMargin=0`
  A fudge factor to adjust the behaviour of `CheckVisibility`, I can't remember why I introduced this. This is a distance in metres, such that points this far behind a CAD feature will still be considered visible to the camera.
- `Coords='Display'`
  String, either 'Display' or 'Original': whether the resulting image coordinates should be in original or display coordinates.

Outputs

- `ImagePoints`
  A list of Nx2 numpy arrays containing the image coordinates of the given 3D points. Each array contains the list of image pixel coordinates [x,y] for a sub-field of view of the image. For simple images without a split field of view, `ImagePoints` is a one-element list where `ImagePoints[0]` is an Nx2 array containing the image coordinates. Points not visible to the camera, either because their projection is off the image edge, or they are occluded and CheckVisible is enabled, will have their image coordinates set to `[np.nan,np.nan]`.

`get_cam_to_lab_rotation([field])`

Get a 3D rotation matrix which rotates from the camera frame to the lab frame. This is mostly used internally in calcam, but might be useful.

Inputs

For the common case of an image without a split field of view, no arguments are necessary and can be called as `get_cam_to_lab_rotation()`. For images with a split field-of-view, optional input arguments are:

- `field=0`
  Index of the sub field-of-view for which to get the view angles.

Outputs

- 3x3 NumPy matrix containing the rotation matrix.

4.2 The `CADModel` Class

The functions described in the following sections require Calcam CAD model objects as inputs, therefore the CAD model class is very briefly documented here. This is the parent class for user-defined CAD models in calcam, and provides methods both for user interaction and interaction with the rest of calcam.

To create a CAD model instance representing a given model, use `model = calcam.machine_geometry.model_name(['model_variant'])`, where `model_name` is the name of the user-defined CAD model class and 'model_variant' is an optional string specifying the model variant (the default used when model_variant is not specified is configured in the CAD model definition). For more details about how these names are configured, see the example CAD model definition example/template as described in section 2.3.1.

The most useful methods of the CAD model class for the user are:

`enable_features(feature_list)`

Enable the given features of the model. When a new CAD model instance is initialised, features will automatically be enabled according to the configuration in the CAD model definition. Enabling features means their CAD data will be loaded and they will be included in rendering, ray casting etc.

Inputs

- `features`
  A list of strings with the names of the features to be enabled.

`disable_features(feature_list)`

Disable the given features of the model. When a new CAD model instance is initialised, features will automatically be enabled according to the configuration in the CAD model definition. Disabling features means their CAD data will not be loaded and they will be excluded from rendering, ray casting etc.

Inputs

- `features`
  A list of strings with the names of the features to be disabled.

`enable_only(feature_list)`

Enable only the given features of the model, disabling all others.

Inputs

- `features`
  A list of strings with the names of the features to be enabled.

`set_colour(Colour,[feature_list])`

Set the colour in which the CAD model should appear, either for the whole model or particular features.

Inputs

- `Colour`
  A 3-element list or tuple of values between 0 and 1 giving the (R,G,B) desired colour.
- `Features`=*None* A list of strings containing the feature names to which this colour should be applied. If none, the colour is applied to the whole model.

`enable_only(feature_list)`

Enable only the given features of the model, disabling all others.

Inputs

- `features`
  A list of strings with the names of the features to be enabled.

`get_enabled_features()`

Get a list of the currently enabled features.

Output

- `features`
  List of strings of the names of enabled features.

`colour_by_material(on)`

Set whether the CAD model features should be coloured according to their different physical materials.

Inputs

- `on`
  Bool, whether or not to turn on colouring according to material. The default is set in the CAD model definition.

`flat_shading(on)`

Turn on or off flat shading. When enabled, this disabled all lighting effects when the CAD model is rendered, showing it as flat colours.

Inputs

- `on`
  Bool, whether or not to turn on flat shading.

`colour_by_material(on)`

Set whether the CAD model features should be coloured according to their different physical materials.

Inputs

- `on`
  Bool, whether or not to turn on colouring according to material. The default is set in the CAD model definition.

4.3 The Render Module

The module `calcam.render` contains functions related to rendering images from calibration results. It contains one main important function:

`render_cam_view(CADModel,FitResults[,...])`

Render the given CAD model, from the point of view of a camera described by the calibration results object `FitResults`. This renders the CAD model including the correct image distortion.

Inputs

- `CADModel`
  A calcam CAD model object containing the CAD model to render.
- `FitResults`
  A `calcam.CalibResults` object containing the fit results to base the rendering upon.
- `filename=`*None*
  String, an image file name to save the resulting rendered image (including file extension).
- `oversampling=1`
  The rendered image size in pixels will be equal to this number multiplied by the actual camera resolution. Must be 2^N where N is an integer (including negative integers, if rendering at a smaller resolution than the original camera is desired).
- `AA=1`
  Factor by which to oversize the intermediate render, for antialiasing. Largeer values reduce blocky edges in the resulting images but quickly become very memory intensive to render.
- `Edges=False`
  Bool, if set to `True`, the CAD model is rendered in wireframe, otherwise it is rendered as solid.
- `EdgeColor=(1,0,0)`
  If rendering in wireframe, a tuple of 3 values from 0-1 specifying the RGB colour of the wireframe.

- `EdgeWidth`
  If rendering in wireframe, the thickness in pixels of the wireframe lines.
- `Transparendcy=False`
  Bool, whether to make the background transparent. If left at the default `False`, the background will be black.
- `EdgesMethod=None`
  A string, if rendering in wireframe, this can be used to override the CAD model's default edge type used for the wireframe.
  codeEdgeMethod=Simple shows all polygon edges of the CAD model, while `EdgeMethod=Detect` more intelligently detects edges.
- `Coords='Display'`
  Whether to render in original or display image orientation.

Outputs

- NumPy array containing the rendered image: a [hxwx3] or [hxwx4] (if using transparency) array containing the rendered RGB or RGBA image.
- If a filename is given, the function also saves the rendered image to a file with the given name (note: if transparency is enabled, any given file type will be overriden to PNG).

4.4 The RayCaster Class

The class `calcam.raytrace.RayCaster([FitResults,CADModel])` is used for ray casting, i.e. to find the points of intersection between the camera sight lines and machine structure. It can be initialised with or without specifying the calibration results and CAD model to use.

`set_cadmodel(CADModel)`

Sets the CAD model to be used for the raycasting: takes a calcam CAD model instance as input.

set_calibration(CalibResults)

Sets the calibration fit results used for the raycasting: takes a CalibResults instance as input.

raycast_pixels([x,y,binning,Coords])

Perform raycast to find points of intersection between pixel sight-lines and CAD geometry.

Inputs

- `x,y`
  x and y pixel coordinates for which to perform the ray casting. If none are specified, the function will ray cast each pixel on the detector. Can be any size or shape, as long as the size of x and y are the same.
- `binning=1`
  If ray casting the entire detector without specifying any pixel coordinates, controls the binning of the detector for raycasting. If binning is set to N, the ray cast is performed with the detector binned by a factor NxN.
- `Coords='Display'`
  If x and y are given, specifies whether the given coordinates are in display or original coordinates. If no x and y are given, determines whether the raycast is performed in original or display coordinates.

Outputs

- RayData - a calcam RayData object containing the raycast results.

4.5 The RayData Class

The `RayData(raydata_name)` class is used to store the results from raycasting. Existing results are loaded by initialising with the name of some saved RayData. This is the type of object returned by the `calcam.raytrace.RayCaster.rayc`

4.5.1 Properties

`ray_end_coords`

Numpy array containing the sight-line end points, i.e. the coordinates where the sight-lines intersect the CAD model. The shape of the array is the same as the shape of the input coordinates for the raycast, plus one extra dimension. The final dimension of the array will have length 3 and represent the X, Y and Z coordinates of the ray end point in metres.

`ray_start_coords`

Array containing the origin of the sight-lines, i.e. the camera pupil position. This will be the same as the result from `CalibResults.get_pupilpos`. The shape of the array is the same as the shape of the input coordinates for the raycast, plus one extra dimension. The final dimension of the array will have length 3 and represent the X, Y and Z coordinates of the ray start point in metres.

`x` and `y`

Arrays containing the x and y pixel locations of each raycast pixel. These are the same shape as the input coordinates for the raycast, and are always in display coordinates.

4.5.2 Methods

`get_ray_lengths([x,y,PositionTol,Coords])`

Returs the length of the sight-lines for the specified pixel coordinates, i.e. the distance from the camera pupil where the sight lines intersect the CAD geometry.

Inputs

- x and y
  x and y pixel coordinates at which to return the sight line length. Must be the same size as each other. If none are specified, returns the result for every pixel in the raycast result.

- PositionTol=3
  It is possible that the raycast was not performed at exactly the specified pixel x and y coordinates. Normally, this function will return the result from the closest point which was raycast, provided the distance of that point is within `PositionTol` pixels of the requested coordinates.
- `Coords='Display'`
  If x and y are given, specifies whether the given coordinates are in display or original coordinates. If no x and y are given, determines whether the output should be returned in original or display coordinates.

Outputs

- Numpy array containing the sight-line lengths in metres. If x and y are specified, this is the same shape as x and y. If no x and y are specified, this is the same shape as the x and y inputs to the raycast.

`get_ray_directions([x,y,PositionTol,Coords])`

Returns unit vectors representing the direction of the camera sight-lines in the lab frame. Does the same job as `CalibResults.get_los_direction` (maybe I should rename this one to match?).

Inputs

- x, y
  x and y pixel coordinates at which to return the sight line directions. Must be the same size as each other. If none are specified, returns the result for every pixel in the raycast result.
- PositionTol=3
  It is possible that the raycast was not performed at exactly the specified pixel x and y coordinates. Normally, this function will return the result from the closest point which was raycast, provided the distance of that point is within PositionTolerance pixels of the requested coordinates.
- `Coords='Display'`
  If x and y are given, specifies whether the given coordinates are in

display or original coordinates. If no x and y are given, determines whether the output should be returned in original or display coordinates.

Outputs

- Numpy array containing 3D unit vectors representing the sight-line directions in the lab frame.

4.6 The Geometry Matrix Module

The geometry matrix module is used to produce geometry matrices for tomographic reconstruction of plasma emissivity profiles, assuming toroidal symmetry. There is currently no documentation for this module in this document; the module RO is James Harrison.