PyOpticalTable

jdpicks

Dec 10, 2021

CONTENTS

CHAPTER

ONE

OVERVIEW

PyOpticalTable is a Python library designed to make it easy to create high-quality figures of optical setups, for use in papers, theses, or presentations. The figure creation is all done using *matplotlib <www.matplotlib.org>*, and **PyOpticalTable** essentially provides a user-friendly syntax for drawing the setup.

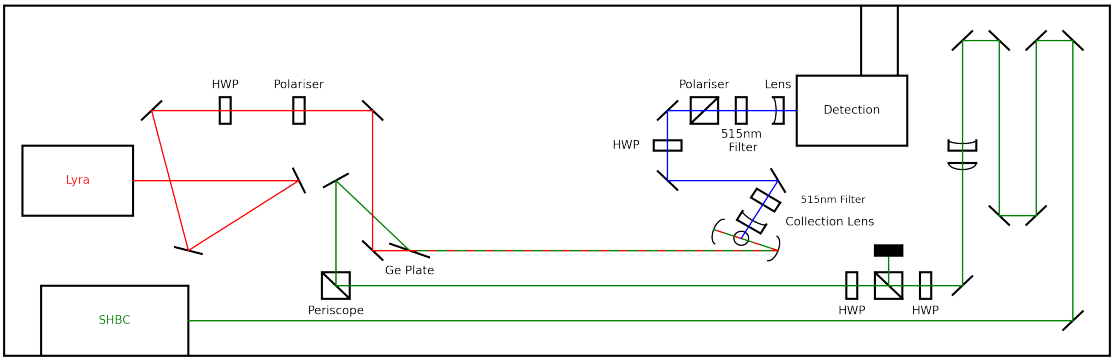


Fig. 1: Click image for the high-res version.

CHAPTER

TWO

WHY BOTHER WHEN INKSCAPE EXISTS?

Anyone who has used inkscape to try and draw a complex optical setup will probably have experienced the following:

- Clicking and dragging things to place the optics is easy initially
- Actually getting it to be precise, so that it looks professional, is harder.
- When you need to adjust the position of an optic because a proof-reader corrected it, it's a complete pain to readjust all the subsequent beam routing.
- When you need to resize the figure because the journal you're submitting to don't understand LaTeX, it makes everything look odd because the aspect ratio can change.
- Inkscape periodically will crash and freeze for no explicable reason.

So whilst Inkscape (or another point-and-click graphics editor) is easy to use at the start, it quickly becomes the bane of your existence. **PyOpticalTable** solves the problems listed above:

- Using PyOpticalTable, you can place all the optics precisely using a coordinate system that is logical and intuitive - and you can set a temporary grid over the table to make finding the coordinates very easy.
- Using PyOpticalTable, the trajectory of the laser beam(s) is defined *by the position of the optics*, just like on a real optical table. So if you move an optic, the beam trajectory automatically reroutes itself to compensate.
- Using PyOpticalTable, all the graphics for the pre-defined optical elements will maintain their appearance if you change the figure size, aspect ratio, or rotation. So whatever you need to do to make the figure fit into where you want it, you won't have to spend ages updating all of the individual components on the figure.
- Python IDEs freeze less often than inkscape (although this admittedly depends on IDE choice. Use vim for the sleekest experience.)

The downsides of PyOpticalTable are:

- Using PyOpticalTable will mean there's a bit of a learning curve if you don't already know Python.
- If your optical setup is very simple (like, two mirrors and a box simple), then maybe it's quicker to just use Inkscape.

CHAPTER

THREE

SCOPE

Currently the inbuilt optics in PyOpticalTable include mirrors, beamsplitters, lenses, waveplates (any kind of flat transmissive plate - crystals, windows…), and polarisers. Also available are ways to make lasers ('box' sources), and generic point sources where beams can come from. Things like beam dumps are also included. There are a variety of helper functions and routines that make adding/creating your own specific optical elements straightforward too - most things are built up from simpler elements like line objects or arc patches.

PyOpticalTable currently only deals with perfect laser beams, and so there isn't any native way to draw a diverging beam that has a realistic divergence affected by a lens (although this can be approximated by using two LaserBeam objects). It is *not* a ray tracing program, and has been designed around making professional-looking figures for publications/theses - not for accurately simulating all of the optical physics.

CHAPTER

FOUR

USING PYOPTICALTABLE

CHAPTER

FIVE

A SIMPLE EXAMPLE

Here we will make a drawing of a simple optical layout involving one laser beam, to illustrate how the package works. The python file `simple_example.py` in the *examples* folder contains the code shown here.

First we have to load the package:

```
import pyopticaltable as pyopt
```

You can either put the file `pyopticaltable.py` in your working directory, or add it to your path so python can find it wherever you are.

Now we need to set up our table:

```
table = pyopt.OpticalTable(20, 10, size_factor=10, show_edge=True, show_grid=True)
```

This creates an **OpticalTable** object called *table*. The dimensions of the table *in terms of matplotlib axis coordinates* are 20 by 10 (width by height). The point (0,0) is at the center of the table, so the point (-10, -5) is the lower left corner. The dimensions of the table *in millimetres* are given by the dimensions in axis coordinates multiplied by the **size_factor** - which is equal to 10 here. Thus, our final figure will be 200mm by 100mm. This allows you to make the axis coordinates a nice round number to make placing optics easy, but also allows you to tweak the figure size so it will fit in a precisely defined column width without making the optic placement annoying.

The two other keyword arguments affect how the table is displayed. **show_edge** draws a line around the table edge, so you can see where the limits of your table are. **show_grid** draws a faint grid over the table where the lines are spaced 1 matplotlib axis unit apart - so here there will be a 20x10 grid covering the table (note that you can also change the grid spacing, but the dimension divided by the spacing needs to be an integer). This grid is mostly useful for editing, as you can make the figure (running the script with just these two lines will make an empty table), and then mouse over the grid to see exactly what the coordinates of each point are. This makes it easier to get things to line up and be square - and you can then turn off the grid when you want to save the figure. See the docstrings for more information.

Now we can also initialise the beam we are going to use:

```
beam = pyopt.LaserBeam(colour='green')
beampath = []
```

The first line here creates a **LaserBeam** object that will eventually be green (any matplotlib-allowed colour works fine) - see the docstrings for other arguments you can use. In the second line we create an empty list called *beampath*, which we will use when we come to actually route our beam through the optics. Let's now place a laser that our beam can come from:

```
laser = table.box_source(-5,0, 4, 2, 0, colour='k', output_side='right',label='Laser',␣
↪textcolour='green', labelpad=0)
```

This will just put a box labelled *Laser* on the table, and the center of the box will be at (-5,0), and it is 4 grid spaces wide and 2 high (again see doctrings for detailed info). This box is an **OpticalElement** object that is called *laser*. The

output_side kwarg determines where the beam will come from - it will originate from the right hand side of the box we placed. Having placed the laser, we can now append this object to the list *beampath*:

```
beampath.append(laser)
```

The beam will originate from the first object in the list *beampath* and then hit subsequent optical elements in the order they are in the list. So it's easy to keep track of if you just append each object to the list after you make it. Alternatively, you can make all the objects and then add them to the list at once at the end - either works. Let's add some mirrors for our beam to reflect off.:

```
mirrorsize = 0.5
mirror1 = table.mirror(0,0,mirrorsize,45)
beampath.append(mirror1)

mirror2 = table.mirror(0,3,mirrorsize,45)
beampath.append(mirror2)

mirror3 = table.mirror(5,3,mirrorsize,-45)
beampath.append(mirror3)

mirror4 = table.mirror(5,0,mirrorsize,-45)
beampath.append(mirror4)
```

This will add four OpticalElement mirror objects to the table. Note that all of the optical elements we can add are methods of the OpticalTable class (i.e. the syntax is table.element()). The (x,y) position of the center of each mirror is defined, and the size of the mirror (length of the line representing it) is set globally in the **mirrorsize** variable. By default, any mirror you add lies flat along the x-axis, and so the final argument is the angle to rotate the mirror from the positive x-axis - 45 degrees (or -45 degrees) is common, but any is possible (note that you can rotate all other optical elements too).

After we add each mirror, we add the name each one to the *beampath* list, so that our beam takes the right route through them (note also that the beam will hit the mirror at the central point (x,y)). Let's try drawing the beam now:

```
beam.draw(table, beampath)
```

Now we have a green beam drawn on the OpticalTable *table* which follows the path defined by *beampath*. The nice thing is that the beam path is *defined by the positions of the mirrors* - try and move one of the mirrors and watch the beam re-route itself. You could also add the mirrors in a different order to *beampath* if you wanted it to take a different route.

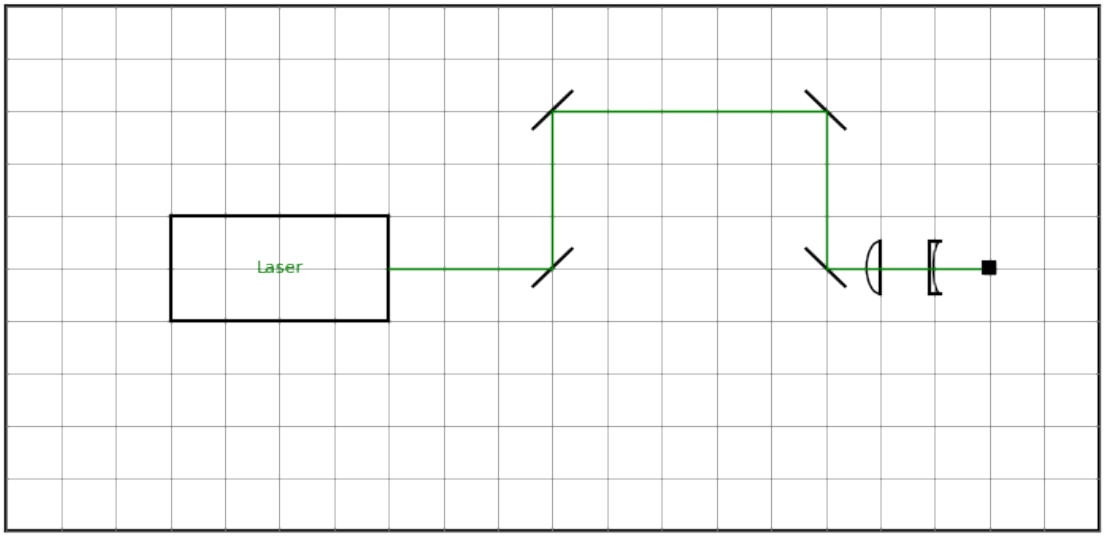
Now we can add some more optics though, so let's add these lines **before** we draw the beam:

```
L1 = table.convex_lens(6, 0, mirrorsize, 90)
beampath.append(L1)

L2 = table.concave_lens(7, 0, mirrorsize, 270)
beampath.append(L2)

dump = table.beam_dump(8, 0, 0.1, 0, fillcolour='k')
beampath.append(dump)
```

Adding two lenses like a telescope, and then putting a beam dump in to catch the beam at the end. There you have it! You can now turn off the grid and save the figure, or add more things, or do whatever you'd like to. You can see a more complex example in the *examples* folder too - one with multiple beams and things that aren't all at right angles.


Laser

CHAPTER

SIX

DETAILED DOCS

PyOpticalTable – Library for drawing optical layouts.

6.1 Classes

tools : helper functions for drawing and aligning optics correctly

OpticalElement : contains info specific to a certain element

OpticalTable : holds all elements, elements are methods of the table

LaserBeam: draws beam between all listed elements

class pyopticaltable.**LaserBeam**(*colour*, *width=1*, *style='-'*, *divergent=False*, *divergence_angle=None*)

LaserBeam defines a laser beam that goes through the defined setup.

When an instance of LaserBeam is created a beam with the given colour, linewidth, and linestyle is initialised, but not drawn. The beam is drawn when LaserBeam.draw(table, optics) is called. table is the OpticalTable instance to draw the beam onto, and optics is a list of the optics to pass through (in order of first to last hit).

The idea of this is that the beam will be redrawn automatically if any optical elements are moved, removing the need to tediously reroute the whole beam by hand. Each OpticalElement instance contains an (x,y) coordinate which is where the beam will hit the optic.

colour

Colour of the laser beam, any matplotlib defined colour is fine.

Type string

width

Linewidth of the drawn laser beam (as defined in matplotlib). The default is 1.

Type float, optional

style

Linestyle of the drawn laser beam, see matplotlib docs for more details. The default is '-' (unbroken line).

Type string, optional

divergent

NOT IMPLEMENTED

Type bool, optional

divergence_angle

NOT IMPLEMENTED

Type float, optional

draw(*table*, *optics*)

Draw a laser beam on table between the OpticalElement instances in optics.

The list optics is a list of OpticalElement instances, and the beam originates at the first element in the list, and passes through/reflects off each element sequentially. Thus, the ordering of the list is important to ensure correct beam routing.

Beam parameters (colour, linewidth, linestyle) are controlled when the LaserBeam class is initialised.

Parameters

- **table** (OpticalTable) – The instance of OpticalTable the beam is drawn on.
- **optics** (*list*) – List of OpticalElement instances.

Returns

Return type None.

class pyopticaltable.**OpticalElement**(*x*, *y*, *mode*, *angle=None*, *element_type=None*)

OpticalElement contains information specific to an optical element.

An instance of this class is created whenever an optical element is added to the table.

x

x coordinate of the optical element

Type float

y

y coordinate of the optical element

Type float

mode

mode of the optic - transmissive "t", or reflective "r" (UNUSED)

angle

rotation angle of the optic relative to the x axis

Type float

class pyopticaltable.**OpticalTable**(*length*, *width*, *size_factor=10.0*, *edgecolour='k'*, *grid_spacing=1.0*, *gridcolour='gray'*, *show_edge=True*, *show_grid=False*, *show_labels=False*)

OpticalTable is where OpticalElements are placed, and forms the main drawing canvas.

Calling OpticalTable will generate a figure to which optics (defined as methods of OpticalTable) will be added.

The length and width determine the internal table coordinates used to place optics, and then also determine the final figure size in conjunction with size_factor. Essentially then the coordinates can be made into a sensible range for ease of use, and then the final figure size controlled with size_factor. The table is centered on the origin so the coordinates range from -length/2 to length/2 and so on.

Parameters

- **length** (*int*) – Length (x dimension) of the table, in internal figure coordinates.
- **width** (*int*) – Width (y dimension) of the table, in internal figure coordinates.
- **size_factor** (*float, optional*) – The length and width are each multiplied by the size factor to give the dimensions of the final figure in millimetres. The default is 10.
- **edgecolour** (*string, optional*) – Colour of the border of the table, any matplotlib supported colour works. The default is 'k' (black).

- **grid_spacing** (*float, optional*) – Spacing of lines shown on the table grid, in internal figure coordinates. Must be chosen such that there is an integer number of lines over the length. The default is 1.
- **gridcolour** (*string, optional*) – Colour of the grid on the table, any matplotlib supported colour works. The default is 'gray'.
- **show_edge** (*bool, optional*) – If true then the edge of the table is shown. The default is True.
- **show_grid** (*bool, optional*) – If true then a grid is shown over the table. The default is False.
- **show_labels** (*bool, optional*) – If true then coordinate labels are added to the grid. The default is False

ax

Matplotlib axis containing the optical table figure.

Type AxesSubplot

aspect_ratio

Ratio of the length to the width of the table.

Type float

angled_line(*x*, *y*, *size*, *angle*, *colour='k'*, *show=True*, *get_coords=False*)

Generate a line centered at (x, y) of length size, rotated by an angle angle (in degrees).

The line automatically scales itself if the aspect ratio is changed.

This function is mostly called by the optical element generation functions but can also just be used to put a line somewhere (or get the coordinates of a line).

Parameters

- **x** (*float*) – x-coordinate of the centre of the line.
- **y** (*float*) – y-coordinate of the centre of the line.
- **size** (*float*) – Length of the line.
- **angle** (*float*) – Rotation angle of the line anticlockwise from the x-axis, in degrees.
- **colour** (*string, optional*) – Colour of the line, any matplotlib supported colour works. The default is 'k' (black).
- **show** (*bool, optional*) – If True then the line is plotted on the table. The default is True.
- **get_coords** (*bool, optional*) – If True then the coordinates of the two endpoints are returned as (x1, x2, y1, y2). The default is False.

Returns

- **x-dX** (*float*) – Smaller of the two x coordinates of the line (x1).
- **x+dX** (*float*) – Larger of the two x coordinates of the line (x2).
- **y-dy** (*float*) – Smaller of the two y coordinates of the line (y1).
- **y+dY** (*float*) – Larger of the two y coordinates of the line (y2).

beam_dump(*x*, *y*, *size*, *angle*, *colour='k'*, *fillcolour='k'*, *label=None*, *label_pos='top'*, *labelpad=0.25*, *textcolour='k'*, *fontsize=8*)

Draw a beam dump on the table.

A beam dump is simply a filled in block where a beam can terminate. The zorder=10 ensures that the beam dump is always drawn on top of the beam, such that it looks like the beam is effectively blocked.

Parameters

- **x** (*float*) – x-coordinate of the centre of the optic.
- **y** (*float*) – y-coordinate of the centre of the optic.
- **size** (*float*) – Size of the optic.
- **angle** (*float*) – Rotation of the optic anticlockwise from the positive x-axis, in degrees.
- **colour** (*string, optional*) – Colour of the optic, any matplotlib supported colour works. The default is 'k'.
- **fillcolour** (*string, optional*) – Colour to fill the block with. The default is 'k'.
- **label** (*string, optional*) – Text to put in the label for the box. The default is None (no label).
- **label_pos** (*string, optional*) – Position of the label relative to the box ('top', 'bottom', 'left', 'right'). The default is "bottom".
- **labelpad** (*float, optional*) – Additional padding to add between the label and the box. The default is 0.25.
- **textcolour** (*string, optional*) – Colour of the label text. The default is 'k' (black).
- **fontsize** (*float, optional*) – Font size for the label text. The default is fontparams['fontsize'].

Returns Instance of the OpticalElement class for this optic.

Return type *OpticalElement*

beamsplitter_cube(*x*, *y*, *size*, *angle*, *direction*, *colour='k'*, *label=None*, *label_pos='top'*, *labelpad=0.25*, *textcolour='k'*, *fontsize=8*)

Parameters

- **x** (*float*) – x-coordinate of the centre of the optic.
- **y** (*float*) – y-coordinate of the centre of the optic.
- **size** (*float*) – Size of the optic.
- **angle** (*float*) – Rotation of the optic anticlockwise from the positive x-axis, in degrees.
- **direction** (*string*) – Direction of the reflective surface in the beamsplitter, allowed values are "L" and "R".
- **colour** (*string, optional*) – Colour of the optic, any matplotlib supported colour works. The default is 'k'.
- **label** (*string, optional*) – Text to put in the label for the optic. The default is None (no label).
- **label_pos** (*string, optional*) – Position of the label relative to the optic ('top', 'bottom', 'left', 'right'). The default is "bottom".
- **labelpad** (*float, optional*) – Additional padding to add between the label and the optic. The default is 0.25.
- **textcolour** (*string, optional*) – Colour of the label text. The default is 'k' (black).

- **fontsize** (*float, optional*) – Font size for the label text. The default is fontparams['fontsize'].

Raises **ValueError** – Raised if a direction other than "L" or "R" is entered.

Returns Instance of the OpticalElement class for this optic.

Return type *OpticalElement*

box(*x*, *y*, *size_x*, *size_y*, *angle*, *colour='k'*, *standalone=False*, *label=None*, *label_pos='top'*, *labelpad=0.25*, *textcolour='k'*, *fontsize=8*)
Create a rectangular box of arbitrary size and rotation angle.

The box can be drawn on the table (if standalone=True), otherwise the corner coordinates of the box are returned from the function to use in other functions.

Parameters **x** (*float*) – x-coordinate of the centre of the box.

y [float] y-coordinate of the centre of the box.

size_x [float] Size of the box in the x direction.

size_y [float] Size of the box in the y direction.

angle [float] Rotation of the box anticlockwise from the positive x-axis, in degrees.

colour [string, optional] Colour of the box edge, any matplotlib supported colour works. The default is 'k'.

standalone [bool, optional] If true then the box is drawn on the table as it is, otherwise just the corner coordinates are returned for use in other functions. The default is False.

label [string, optional] Text to put in the label for the box. The default is None (no label).

label_pos [string, optional] Position of the label relative to the box ('top', 'bottom', 'left', 'right'). The default is "bottom".

labelpad [float, optional] Additional padding to add between the label and the box. The default is 0.25.

textcolour [string, optional] Colour of the label text. The default is 'k' (black).

fontsize [float, optional] Font size for the label text. The default is fontparams['fontsize'].

Returns

- *OpticalElement* – Instance of the OpticalElement class for this optic (if standalone=True)
- **corners_rot** (*list*) – List of the corner coordinates (anticlockwise from bottom left) of the box (if standalone=False).

box_source(*x*, *y*, *size_x*, *size_y*, *angle*, *output_side*, *colour='k'*, *label=None*, *label_pos='top'*, *labelpad=0.25*, *textcolour='k'*, *fontsize=8*)
Draw a box on the table that a laser beam can come from.

Box is centered on (x, y), and the beam will come from the midpoint of one of the four sides, determined by the string passed via output_side. Allowed sides are 'top', 'bottom', 'left', 'right'.

Parameters

- **x** (*float*) – x-coordinate of the centre of the box.
- **y** (*float*) – y-coordinate of the centre of the box.
- **size_x** (*float*) – Size of the box in the x direction.
- **size_y** (*float*) – Size of the box in the y direction.

- **angle** (*float*) – Rotation of the box anticlockwise from the positive x-axis, in degrees.
- **output_side** (*string*) – Which side of the box the beam will come from, allowed values 'top', 'bottom', 'left', 'right'.
- **colour** (*string, optional*) – Colour of the box edge, any matplotlib supported colour works. The default is 'k'.
- **label** (*string, optional*) – Text to put in the label for the box. The default is None (no label).
- **label_pos** (*string, optional*) – Position of the label relative to the box ('top', 'bottom', 'left', 'right'). The default is "bottom".
- **labelpad** (*float, optional*) – Additional padding to add between the label and the box. The default is 0.25.
- **textcolour** (*string, optional*) – Colour of the label text. The default is 'k' (black).
- **fontsize** (*float, optional*) – Font size for the label text. The default is fontparams['fontsize'].

Raises **ValueError** – Raised if an invalid output side is entered.

Returns Instance of the OpticalElement class for this optic.

Return type *OpticalElement*

concave_lens(*x*, *y*, *size*, *angle*, *colour='k'*, *offset_factor=0.05*, *lens_factor=4*, *label=None*, *label_pos='top'*, *labelpad=0.25*, *textcolour='k'*, *fontsize=8*)

Draw a concave lens on the optical table.

The beam passes through the point (x,y), which is the centre of the curved side of the lens.

Parameters

- **x** (*float*) – x-coordinate of the centre of the curved face of the lens.
- **y** (*float*) – y-coordinate of the centre of the curved face of the lens.
- **size** (*float*) – Size of the optic.
- **angle** (*float*) – Rotation of the optic anticlockwise from the positive x-axis, in degrees.
- **colour** (*string, optional*) – Colour of the optic, any matplotlib supported colour works. The default is 'k'.
- **offset_factor** (*float, optional*) – How thick the concave lens is. The default is 0.05.
- **lens_factor** (*float, optional*) – Controls how curved the lens is (how "lens-y" it looks). The default is 4.
- **label** (*string, optional*) – Text to put in the label for the optic. The default is None (no label).
- **label_pos** (*string, optional*) – Position of the label relative to the optic ('top', 'bottom', 'left', 'right'). The default is "bottom".
- **labelpad** (*float, optional*) – Additional padding to add between the label and the optic. The default is 0.25.
- **textcolour** (*string, optional*) – Colour of the label text. The default is 'k' (black).
- **fontsize** (*float, optional*) – Font size for the label text. The default is fontparams['fontsize'].

Returns Instance of the OpticalElement class for this optic.

Return type *OpticalElement*

concave_mirror(*x*, *y*, *size*, *angle*, *colour='k'*, *lens_factor=1*, *label=None*, *label_pos='top'*, *labelpad=0.25*, *textcolour='k'*, *fontsize=8*)

Draw a concave mirror on the optical table.

The beam will hit the mirror at the point (x, y).

Parameters

- **x** (*float*) – x-coordinate of the centre of the optic.
- **y** (*float*) – y-coordinate of the centre of the optic.
- **size** (*float*) – Size of the optic.
- **angle** (*float*) – Rotation of the optic anticlockwise from the positive x-axis, in degrees.
- **colour** (*string, optional*) – Colour of the optic, any matplotlib supported colour works. The default is 'k'.
- **lens_factor** (*float, optional*) – Controls how curved the mirror is (how "lens-y" it looks). The default is 1.
- **label** (*string, optional*) – Text to put in the label for the optic. The default is None (no label).
- **label_pos** (*string, optional*) – Position of the label relative to the optic ('top', 'bottom', 'left', 'right'). The default is "bottom".
- **labelpad** (*float, optional*) – Additional padding to add between the label and the optic. The default is 0.25.
- **textcolour** (*string, optional*) – Colour of the label text. The default is 'k' (black).
- **fontsize** (*float, optional*) – Font size for the label text. The default is fontparams['fontsize'].

Returns Instance of the OpticalElement class for this optic.

Return type *OpticalElement*

convex_lens(*x*, *y*, *size*, *angle*, *colour='k'*, *lens_factor=2*, *label=None*, *label_pos='top'*, *labelpad=0.25*, *textcolour='k'*, *fontsize=8*)

Draw a convex lens on the optical table.

The beam will pass through the point (x,y), which is the centre of the flat face of the lens.

Parameters

- **x** (*float*) – x-coordinate of the centre of the flat face of the lens.
- **y** (*float*) – y-coordinate of the centre of the flat face of the lens.
- **size** (*float*) – Size of the optic.
- **angle** (*float*) – Rotation of the optic anticlockwise from the positive x-axis, in degrees.
- **colour** (*string, optional*) – Colour of the optic, any matplotlib supported colour works. The default is 'k'.
- **lens_factor** (*float, optional*) – Controls how curved the lens is (how "lens-y" it looks). The default is 2.

- **label** (*string, optional*) – Text to put in the label for the optic. The default is None (no label).
- **label_pos** (*string, optional*) – Position of the label relative to the optic ('top', 'bottom', 'left', 'right'). The default is "bottom".
- **labelpad** (*float, optional*) – Additional padding to add between the label and the optic. The default is 0.25.
- **textcolour** (*string, optional*) – Colour of the label text. The default is 'k' (black).
- **fontsize** (*float, optional*) – Font size for the label text. The default is fontparams['fontsize'].

Returns Instance of the OpticalElement class for this optic.

Return type *OpticalElement*

generic_circle(*x*, *y*, *size*, *colour='k'*, *fill=False*, *fillcolour='k'*, *label=None*, *label_pos='top'*, *labelpad=0.25*, *textcolour='k'*, *fontsize=8*)

Draw a circle on the optical table centered at (x,y).

The radius of the circle is determined by the size parameter.

Parameters

- **x** (*float*) – x-coordinate of the centre of the circle.
- **y** (*float*) – y-coordinate of the centre of the circle.
- **size** (*float*) – Radius of the circle.
- **colour** (*string, optional*) – Colour of the edge of the circle. The default is 'k'.
- **fill** (*bool, optional*) – If True then the circle is filled with fillcolour. The default is False.
- **fillcolour** (*string, optional*) – Colour to fill the circle with. The default is 'k'.
- **label** (*string, optional*) – Text to put in the label for the box. The default is None (no label).
- **label_pos** (*string, optional*) – Position of the label relative to the box ('top', 'bottom', 'left', 'right'). The default is "bottom".
- **labelpad** (*float, optional*) – Additional padding to add between the label and the box. The default is 0.25.
- **textcolour** (*string, optional*) – Colour of the label text. The default is 'k' (black).
- **fontsize** (*float, optional*) – Font size for the label text. The default is fontparams['fontsize'].

Returns Instance of the OpticalElement class for this optic.

Return type *OpticalElement*

mirror(*x*, *y*, *size*, *angle*, *colour='k'*, *label=None*, *label_pos='bottom'*, *labelpad=0.25*, *textcolour='k'*, *fontsize=8*)

Draw a mirror on the optical table.

A mirror is simply an angled line. Laser beams will bounce off the middle of the mirror, at the point (x,y).

Parameters

- **x** (*float*) – x-coordinate of the centre of the optic.

- **y** (*float*) – y-coordinate of the centre of the optic.
- **size** (*float*) – Size of the optic.
- **angle** (*float*) – Rotation of the optic anticlockwise from the positive x-axis, in degrees.
- **colour** (*string, optional*) – Colour of the optic, any matplotlib supported colour works. The default is 'k'.
- **label** (*string, optional*) – Text to put in the label for the optic. The default is None (no label).
- **label_pos** (*string, optional*) – Position of the label relative to the optic ('top', 'bottom', 'left', 'right'). The default is "bottom".
- **labelpad** (*float, optional*) – Additional padding to add between the label and the optic. The default is 0.25.
- **textcolour** (*string, optional*) – Colour of the label text. The default is 'k' (black).
- **fontsize** (*float, optional*) – Font size for the label text. The default is fontparams['fontsize'].

Returns Instance of the OpticalElement class for this optic.

Return type *OpticalElement*

point_source(*x*, *y*, *label=None*, *label_pos='top'*, *labelpad=0.25*, *textcolour='k'*, *fontsize=8*)

Create a source for a laser beam at (x,y).

The source is not visible on the table, but just serves as a place where a beam can emanate from/pass through.

Parameters

- **x** (*float*) – x-coordinate of the point.
- **y** (*float*) – y-coordinate of the point.
- **label** (*string, optional*) – Text to put in the label for the box. The default is None (no label).
- **label_pos** (*string, optional*) – Position of the label relative to the box ('top', 'bottom', 'left', 'right'). The default is "bottom".
- **labelpad** (*float, optional*) – Additional padding to add between the label and the box. The default is 0.25.
- **textcolour** (*string, optional*) – Colour of the label text. The default is 'k' (black).
- **fontsize** (*float, optional*) – Font size for the label text. The default is fontparams['fontsize'].

Returns Instance of the OpticalElement class for this optic.

Return type *OpticalElement*

set_label(*axis*, *x*, *y*, *size*, *label*, *label_pos*, *labelpad*, *textcolour='k'*, *fontsize=8*)

Put a label on an optical element.

If the label is None then this does nothing.

Label coordinates are determined dynamically by the optic coordinates, optic size, and any padding added by the user. The position relative to the optic is determined by passing label_pos ('top', 'bottom', 'left', 'right').

Parameters

- **axis** (*AxesSubplot*) – Axis to put the label onto.
- **x** (*float*) – x-coordinate of the optic to be labelled.
- **y** (*float*) – y-coordinate of the optic to be labelled.
- **size** (*float*) – Size of the optic to be labelled.
- **label** (*string*) – Text to put in the label.
- **label_pos** (*string*) – Position of the label relative to the optic ('top', 'bottom', 'left', 'right').
- **labelpad** (*float*) – Additional padding to add between the label and the optic.
- **textcolour** (*string, optional*) – Colour of the label text. Default is 'k' (black).
- **fontsize** (*float, optional*) – Font size for the label text. The default is fontparams['fontsize'].

Returns

Return type None.

transmissive_cube(*x*, *y*, *size*, *angle*, *colour='k'*, *label=None*, *label_pos='top'*, *labelpad=0.25*, *textcolour='k'*, *fontsize=8*)

Draw a transmissive cube (i.e a square) on the optical table.

The square is centered at (x,y) and the beam passes through the centre.

Parameters

- **x** (*float*) – x-coordinate of the centre of the optic.
- **y** (*float*) – y-coordinate of the centre of the optic.
- **size** (*float*) – Size of the optic.
- **angle** (*float*) – Rotation of the optic anticlockwise from the positive x-axis, in degrees.
- **colour** (*string, optional*) – Colour of the optic, any matplotlib supported colour works. The default is 'k'.
- **label** (*string, optional*) – Text to put in the label for the optic. The default is None (no label).
- **label_pos** (*string, optional*) – Position of the label relative to the optic ('top', 'bottom', 'left', 'right'). The default is "bottom".
- **labelpad** (*float, optional*) – Additional padding to add between the label and the optic. The default is 0.25.
- **textcolour** (*string, optional*) – Colour of the label text. The default is 'k' (black).
- **fontsize** (*float, optional*) – Font size for the label text. The default is fontparams['fontsize'].

Returns Instance of the OpticalElement class for this optic.

Return type *OpticalElement*

transmissive_plate(*x*, *y*, *size*, *angle*, *colour='k'*, *offset_factor=0.05*, *fill=False*, *fillcolour='k'*, *label=None*, *label_pos='top'*, *labelpad=0.25*, *textcolour='k'*, *fontsize=8*, *zorder=2*)

Draw a transmissive plate on the optical table.

A generic transmissive plate but is useful for waveplates, crystals, filters, windows etc… with appropriate labelling.

The beam passes through the point (x,y), which is at the centre of the plate.

Parameters

- **x** (*float*) – x-coordinate of the centre of the optic.
- **y** (*float*) – y-coordinate of the centre of the optic.
- **size** (*float*) – Size of the optic.
- **angle** (*float*) – Rotation of the optic anticlockwise from the positive x-axis, in degrees.
- **colour** (*string, optional*) – Colour of the optic, any matplotlib supported colour works. The default is 'k'.
- **offset_factor** (*float, optional*) – How thick the transmissive plate is. The default is 0.05.
- **fill** (*bool, optional*) – If true then the plate is filled with a colour. The default is False.
- **fillcolour** (*string, optional*) – Colour to fill the plate with. The default is 'k'.
- **label** (*string, optional*) – Text to put in the label for the optic. The default is None (no label).
- **label_pos** (*string, optional*) – Position of the label relative to the optic ('top', 'bottom', 'left', 'right'). The default is "bottom".
- **labelpad** (*float, optional*) – Additional padding to add between the label and the optic. The default is 0.25.
- **textcolour** (*string, optional*) – Colour of the label text. The default is 'k' (black).
- **fontsize** (*float, optional*) – Font size for the label text. The default is fontparams['fontsize'].
- **zorder** (*int, optional*) – Zorder for the transmissive plate (controls drawing order). The default is 2.

Returns Instance of the OpticalElement class for this optic.

Return type *OpticalElement*

class pyopticaltable.**Tools**

Tools contains functions that help with drawing and aligning optics correctly.

Tools is normally not called by the end user, but it is used in other classes to ensure optics are placed and rotated correctly.

cosd()

Return cosine of an angle in degrees.

deg_to_rad()

Convert degrees to radians.

get_label_coords(*x*, *y*, *size*, *labelpad*)

Returns the coordinates where the label for an optic should be.

Calculates them based on the position and size of the optic. User defined labelpad can also be used to further move the label.

Label position is determined by the label_pos parameter.

Parameters

- **label_pos** (*string*) – String determining label position relative to the optic. Allowable values are 'top', 'bottom', 'left', 'right'.
- **x** (*float*) – x-coordinate of the optical element being labelled.
- **y** (*float*) – y-coordinate of the optical element being labelled.
- **size** (*float*) – size of optical element being labelled.
- **labelpad** (*float*) – additional space to put between optic and label.

Returns

- **label_x** (*float*) – x-coordinate of label position.
- **label_y** (*TYPE*) – y-coordinate of label position.

get_midpoint()
Return the midpoint of a line.

Line is loaded as a tuple of points ((x1, y1), (x2, y2))

Parameters **line** (*tuple*) – Contains two points (as tuples (x,y)) that define the line.

Returns **midpoint** – The midpoint of the line (midpoint_x, midpoint_y)

Return type tuple

inch_to_mm()
Convert inches to millimetre.

mm_to_inch()
Convert millimetre to inches.

rotate_point(*angle*)
Rotate a point by an angle theta around the origin.

Used for rotating optics around their centre. Uses a simple 2D rotation matrix (written out in longhand).

Parameters

- **point** (*tuple*) – Coordinates (x,y) of the point to be rotated relative to (0,0).
- **angle** (*float*) – Angle in degrees to rotate the point by.

Returns **point_rot** – Coordinates of the rotated point (x_rot, y_rot).

Return type tuple

sind()
Return sine of an angle in degrees.

CHAPTER

SEVEN

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

INDEX

Y