Cnc25D Documentation

charlyoleg

October 08, 2016

Contents

Contents:

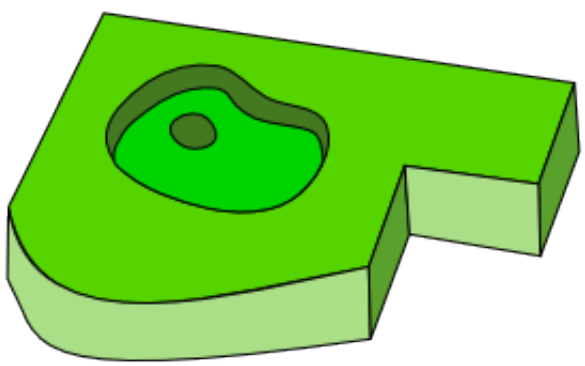
CHAPTER 1

Cnc25D Presentation

Cnc25D is the contraction of “CNC” and “2.5D”.

2.5D and cuboid assembly are good solutions for automated personal fabrication. The Python package cnc25d proposes an API and design examples related to those technologies.

- CNC (Computer Numerical Control) and 3D-printers let move from design files to the physical objects.
- 2.5D parts are objects that can be described as a pile of free 2D path linearly extruded along the third dimension.
- Cuboid assembly is any assembly emphasizing an orthogonal reference frame.



2.5D part

1.1 Cnc25D Python package content

1.1.1 Generic functions

In the Cnc25D Python package, you find functions that help you design parts to be made by a 3-axis CNC, to assemble those parts and to create DXF 2D plans of your design. In particular you get those functions:

- **cnc_cut_outline** takes as input a 2D polygon defined by a list of points and a CNC router_bit diameter and provides as output a millable 2D outline. Then, you just need to extrude this outline to get your 2.5D part.
- **place_plank** offers an alternative natural way to place a part in a cuboid assembly.

- **export_to_dxf** writes a DXF file with a projection of a cut of your design.
- **export_xyz_to_dxf** writes a DXF file with many projections of cuts of your design along the 3 axis, in a similar way as a medical 3D scanner.

DXF is 2D and is the most common design exchange file format. Usually, your CNC guy will need this file format to start his process flow. You can use LibreCAD to view and possibly to re-work your DXF files.

You can also output your design in the 3D STL format and use MeshLab to view and inspect your parts and design.

1.1.2 Design examples

The Cnc25D Python package comes also with some design examples, which are probably for most of the users the most useful things.

One good thing with Designing with Python script is that you get a *100% open-hardware* design because all conception micro-steps are pieces of code and can be shared and hacked using the tools of the software development such as git.

An other advantage of Designing with Python is that parametric design is natural. So you don't create an object but a family of objects with a set of parameters that individualize each manufactured object.

Designing with Python let you work in a similar way as software development. You write code, check the 3D result with the FreeCAD GUI, modify and expand the code and so on. This iterative work-flow is very efficient to capitalize work, reduce repetitive tasks, keep modification history, track bugs and co-work with people.

The complete list of *Cnc25D Design* is available in the section Cnc25D Designs.

Some realizations designed with *Cnc25D*:

- https://cubehero.com/physibles/charlyoleg/Box_Wood_Frame_N1
- https://cubehero.com/physibles/charlyoleg/Epicyclic_gearing_with_laser_cutter
- https://cubehero.com/physibles/charlyoleg/Epicyclic_gearing_with_3D_printer

1.2 Cnc25D Installation

The installation instructions are written for the Ubuntu systems.

1.2.1 Install Cnc25D on your system

This is the preferred method for most people.

- First, install FreeCAD (version 0.13 or newer), Python 2 and Tkinter (which is automatically installed with Python on Ubuntu).
- Then, install the Cnc25D package with the following commands. (The second command is because of a bug in the matplotlib dependency setup):

```
> sudo pip install Cnc25D -U
> sudo pip install matplotlib -U
```

- To create an design example, run the following commands:

```
> cd directory/where/I/want/to/create/my/3D/parts
> cnc25d_example_generator.py
> python box_wood_frame_example.py
```

1.2.2 Install Cnc25D in a virtual environment

This method has currently some issues because of PyQt4.

- First, install FreeCAD on your system (not in a *virtual environment*). You need the version 0.13 or newer.
- Then, create the virtual environment and install the Cnc25D package within it:

```
> cd directory/where/I/want/to/work
> virtualenv env_for_cnc25d
> source env_for_cnc25d/bin/activate
> pip install Cnc25D -U
> pip install matplotlib -U
> deactive
```

- Workaround for PyQt4:

```
> cp /usr/lib/python2.7/dist-packages/sip.so env_for_cnc25d/lib/python2.7/site-packages/
> cp -a /usr/lib/python2.7/dist-packages/PyQt4 env_for_cnc25d/lib/python2.7/site-packages/
```

- To create an design example, run the following commands:

```
> source env_for_cnc25d/bin/activate
> cnc25d_example_generator.py
> python box_wood_frame_example.py
> deactivate
```

- You can also run the generated design example with *freecad*. But *freecad* doesn't get the *virtualenv python package path* and doesn't read the environment variable *PYTHONPATH*. So, you must add the path to the *virtual python package* explicitly:

```
> source env_for_cnc25d/bin/activate
> freecad -P env_for_cnc25d/lib/python2.7/site-packages box_wood_frame_example.py
> deactivate
```

1.2.3 Work directly with the Cnc25D sources

Instead of installing the Cnc25D package, you clone the Cnc25D GitHub repository and work directly with it. This is the preferred method for the programmers:

```
> cd directory/where/I/want/to/work
> git clone https://github.com/charlyoleg/Cnc25D
```

Example of usage:

```
> cd Cnc25D/cnc25d
> python box_wood_frame.py
```

1.3 Cnc25D Usage

1.3.1 Use a design example

After installing Cnc25D, you get the executable **cnc25d_example_generator.py**. When you run this script, it asks you for each design example if you want to generate the script example. Answer 'y' or 'yes' if you want to get the script example. **cnc25d_example_generator.py** can generates the following Python script examples:

- **box_wood_frame_example.py** : The piece of furniture to pile up.

- **cnc25d_api_example.py** : This is not a design example, this shows how to use the API.

These scripts are the design examples. Edit one of these scripts, modify the parameter values, run the script. You get plenty of DXF and STL, that you can view with LibreCAD and MeshLab. You also get a txt file, that provides you a kind of report of your design. In summary, we run the following commands:

```
> cd directory/where/I/want/to/create/my/3D/parts
> cnc25d_example_generator.py
> vim box_wood_frame_example.py
> python box_wood_frame_example.py
> librecad bwf37_assembly_with_amplified_cut.dxf
> meshlab # import bwf36_assembly_with_amplified_cut.stl
> less bwf49_text_report.txt
```

This documentation contains one chapter per design examples that explains in particulary the parameter list.

1.3.2 Use a design example within FreeCAD

In the upper method, we have modified the design example script and then run it to get all the final design files. Even if we can iterate this method, this can be tedious as the generation of all the files requires time. So, probably we want to change a parameter value and just check the 3D result of the assembly. For this purpose, we use FreeCAD directly with one of those three methods:

Script as FreeCAD argument

Launch FreeCAD as following:

```
> freecad box_wood_frame_example.py
```

The design appear in the main windows. Rotate and zoom on your design to inspect it and make sure it is as you want it.

Script as FreeCAD macro

Launch FreeCAD and run the design example script from the macro menu:

```
FreeCAD Top Menu Macro > Macros ...
Within the pop-up window,
  in the field *Macro destination*, select the directory where is located your *design example script
  in the field *Macro name*, select your *design example script*.
  click on *Execute*
```

Script run from FreeCAD

Launch FreeCAD and run the design example script from the Python console:

```
Launch FreeCAD from the directory where is located your *design example script*.
> cd directory/where/I/want/to/create/my/3D/parts
> freecad

Enable 'FreeCAD Top Menu View' > Views > 'Python Console'
Within the FreeCAD Python console, type:
> execfile("box_wood_frame_example.py")
```

1.3.3 Make your design script

If you are interested in the Cnc25D API and want to create your own design with, create a Python script with the following snippet:

```
# import the FreeCAD library
from cnc25d import cnc25d_api
cnc25d_api.importing_freecad()
import Part
from FreeCAD import Base

# use the cnc_cut_outline function
my_polygon = [
  [  0,  0,    5],
  [ 40,  0,    5],
  [ 40, 40,    5],
  [  0, 40,    5]]
my_part_face = Part.Face(Part.Wire(cnc25d_api.cnc_cut_outline(my_part_outline).Edges))
my_part_solid = my_part_face.extrude(Base.Vector(0,0,20))

# use the place_plank function
my_part_a = cnc25d_api.place_plank(my_part_solid.copy(), 40, 40, 20, 'i', 'xz', 0, 0, 0)

# export your design as DXF
cnc25d_api.export_to_dxf(my_part_solid, Base.Vector(0,0,1), 1.0, "my_part.dxf")
xy_slice_list = [ 0.1+4*i for i in range(9) ]
xz_slice_list = [ 0.1+4*i for i in range(9) ]
yz_slice_list = [ 0.1+2*i for i in range(9) ]
cnc25d_api.export_xyz_to_dxf(my_part_solid, 40, 40, 20, xy_slice_list, xz_slice_list, yz_slice_list,
```

Further documentation at Cnc25D API Overview . Also look at the script example **cnc25d_api_example.py** that you can generate with the executable **cnc25d_example_generator.py**.

1.4 Links

1.4.1 Underlying technologies

Cnc25D rely on those open-source technologies:

- OpenCASCADE, the technology used by FreeCAD. Cnc25D doesn't use directly OpenCASCADE.
- FreeCAD, the new open-source CAD tool.
- Python, the popular programming language.

1.4.2 Source

The source code is available at https://github.com/charlyoleg/Cnc25D. Feel free to clone and hack it!

1.4.3 Python package

The Cnc25D package is available on PyPI.

1.4.4 Documentation

The Cnc25D release documentation is associated to the latest Cnc25D Python package release. The Cnc25D daily built documentation provides you the latest documentation updates.

If you have Sphinx installed on your system and you have downloaded the Cnc25D Github repository, you can generate locally the Html documentation with the following commands:

```
> cd Cnc25D/docs
> make html
```

With your browser open the local directory `file:///.../Cnc25D/docs/_build/html`.

1.5 License

The Cnc25D Python package is under GNU General Public License version 3 or any latter (GPL v3+).

1.6 Feedback and contact

If you find bugs, will suggest fix or want new features report it in the GitHub issue tracker or clone the Cnc25D GitHub repository.

For any other feedback, send me a message to "charlyoleg at fabfolk dot com".

1.7 Releases

Check the Cnc25D Release Notes.

CHAPTER 2

Cnc25D Release Notes

2.1 Release 0.1.11

Released on 2014-03-31

- low_torque_transmission
- gearlever

2.2 Release 0.1.10

Released on 2014-01-31

- refactoring/standardizing the designs with bare_design

2.3 Release 0.1.9

Released on 2013-12-13

- complete the Cnc25D API with generic functions for figures
- motor_lid
- bell
- bagel
- bell_bagel
- crest
- cross_cube
- gimbal

2.4 Release 0.1.8

Released on 2013-11-07

- add crenels to the gearwheel

- epicyclic-gearing
- axle_lid

2.5 Release 0.1.7

Released on 2013-10-07

- unify the test-environment of the macro-scripts
- use python-dictionary as function-argument for designs with many parameters
- gearring (aka annulus)
- gearbar (aka rack)
- split_gearwheel

2.6 Release 0.1.6

Released on 2013-09-25

- Use arc primitives for generating DXF and SVG files
- finalization of gear_profile.py and gearwheel.py

2.7 Release 0.1.5

Released on 2013-09-18

- GPL v3 is applied to this Python package.

2.8 Release 0.1.4

Released on 2013-09-11

- Python package created with setuptools (instead of distribute)
- add API function smooth_outline_c_curve() approximates a curve defined by points and tangents with arcs.
- integrate circle into the format-B
- add API functions working at the *figure-level*: figure_simple_display(), figure_to_freecad_25d_part(), ..
- remove API function cnc_cut_outline_fc()
- gear_profile.py generates and simulates gear-profiles
- gearwheel.py

2.9 Release 0.1.3

Released on 2013-08-13

- New API function outline_arc_line() converts an outline defined by points into an outline of four possible formats: Tkinter display, svgwrite, dxfwrite or FreeCAD Part.
- API function cnc_cut_outline() supports smoothing and enlarging line-line, line-arc and arc-arc corners.
- Additional API functions such as outline_rotate(), outline_reverse()
- All Cnc25D API function are gathered in the cnc25d_api module
- Box wood frame design example generates also BRep in addition to STL and DXF.
- Box wood frame design example support router_bit radius up to 4.9 mm with all others parameters at default.

2.10 Release 0.1.2

Released on 2013-06-18

- Box wood frame design example

2.11 Release 0.1.1

Released on 2013-06-05

- Experimenting distribute

2.12 Release 0.1.0

Released on 2013-06-04

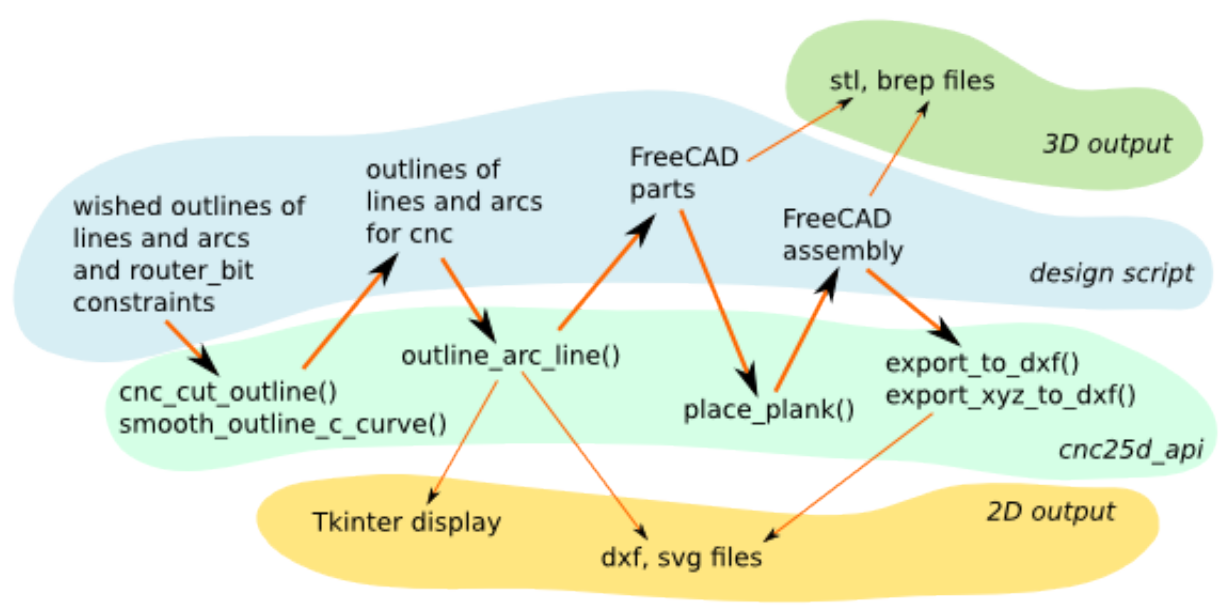
- Initial release

CHAPTER 3

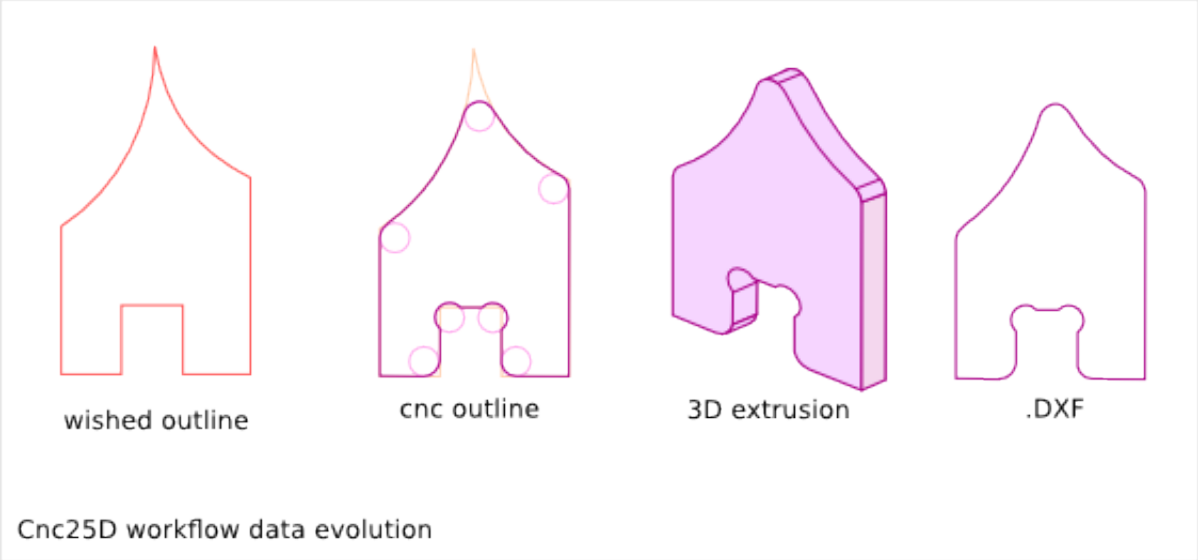
Cnc25D API Overview

3.1 Cnc25D Workflow

FreeCAD provides many GUI and API functions to sculpt and assemble 3D designs. **Cnc25D** proposes a script methodology and an API on top of the FreeCAD API to design 2.5D parts.



Cnc25D design workflow



3.1.1 The Cnc25D methodology

1. Create a list of 2D points you want that your outline go through. An outline is a list of lines and/or arcs. Other curve type must be generated by using multiple small lines.

2. Enlarge or Smooth the corners of the outline to do it makeable by a 3-axis CNC. The cnc25d_api.cnc_cut_outline() function do it for you. It returns a new list of 2D points defining the new lines and arcs of the new outline.

3. Exploit the 2D outline. This new outline can already be export as SVG or DXF. It can also be displayed using Tkinter. Finally, it can be converted intor FreeCAD Part outline to be extruded in 3D part.

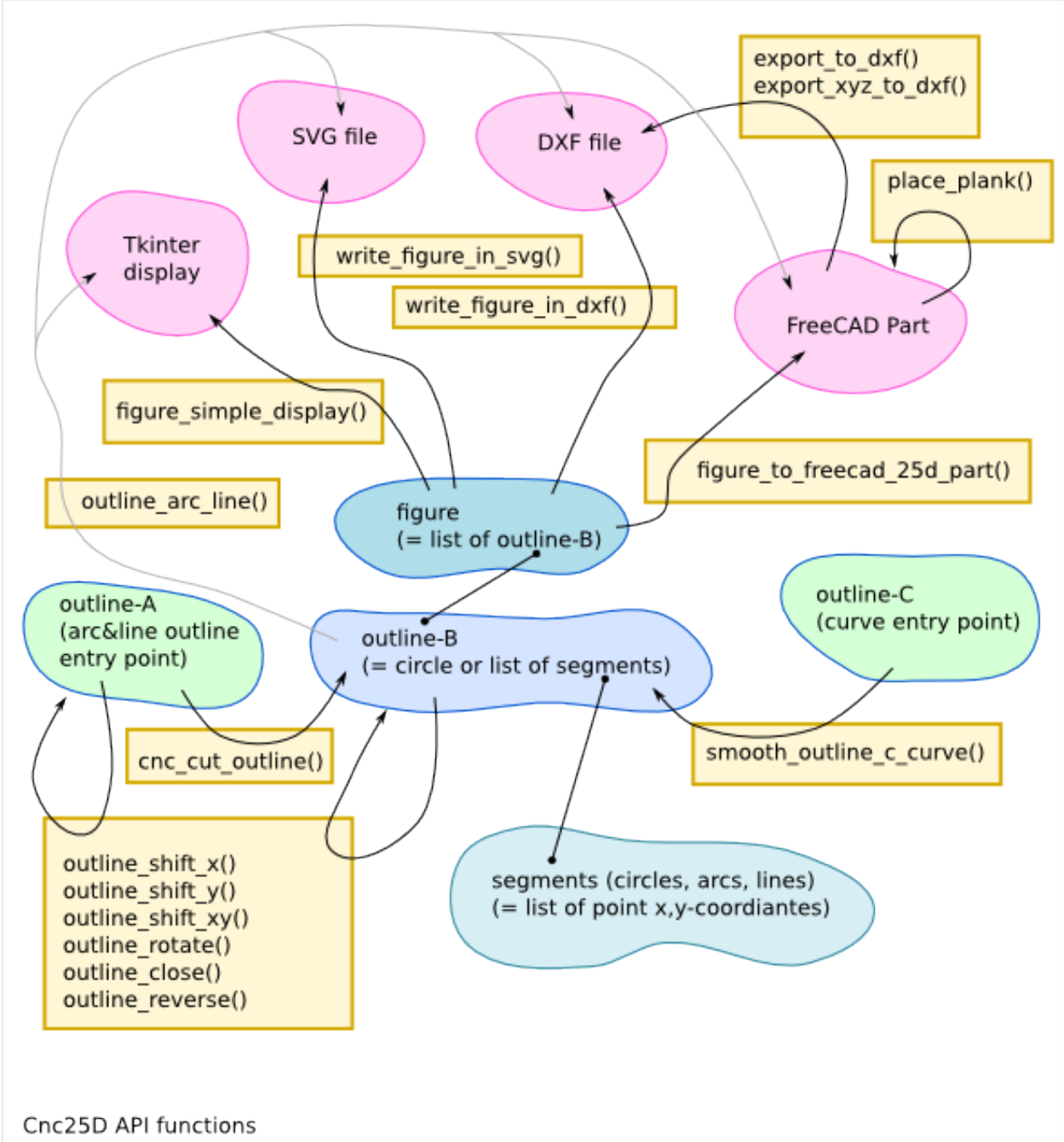
4. Create your 3D assembly. After creating the 3D parts with the FreeCAD Part API, cnc25d_api.place_plank() provides a more natural way to place 3D parts in an assembly than the standard rotate() and translate() methods.

5. Export your design. Export a cut of a 3D parts with cnc25d_api.export_2d(). Get a 3D scanning of your assembly with cnc25d_api.export_xyz_to_dxf()

3.2 Cnc25D API functions and class

```
cnc25d_api.importing_freecad() => 0
cnc25d_api.outline_shift_x(outline-AB, x-offset, x-coefficient) => outline-AB
cnc25d_api.outline_shift_y(outline-AB, y-offset, y-coefficient) => outline-AB
cnc25d_api.outline_shift_xy(outline-AB, x-offset, x-coefficient, y-offset, y-coefficient) => outline-
cnc25d_api.outline_rotate(outline-AB, center-x, center-y, rotation_angle) => outline-AB
cnc25d_api.outline_close(outline-AB) => outline-AB
cnc25d_api.outline_reverse(outline-AB) => outline-AB
cnc25d_api.cnc_cut_outline(outline-A, error_mark_string) => outline-B
cnc25d_api.smooth_outline_c_curve(outline-C, precision, router_bit_radius, error_mark_string) => outl
cnc25d_api.smooth_outline_b_curve(outline-B, precision, router_bit_radius, error_mark_string) => outl
cnc25d_api.ideal_outline(outline-AC, error_mark_string) => outline-B
cnc25d_api.outline_arc_line(outline-B, backend) => Tkinter or svgwrite or dxfwrite or FreeCAD stuff
cnc25d_api.Two_Canvas(Tkinter.Tk()) # object constructor
cnc25d_api.figure_simple_display(graphic_figure, overlay_figure) => 0
cnc25d_api.write_figure_in_svg(figure, filename) => 0
cnc25d_api.write_figure_in_dxf(figure, filename) => 0
cnc25d_api.figure_to_freecad_25d_part(figure, extrusion_height) => freecad_part_object
```

```
cnc25d_api.place_plank(freecad_part_object, x-size, y-size, z-size, flip, orientation, x-position, y-
cnc25d_api.export_to_dxf(freecad_part_object, direction_vector, depth, filename) => 0
cnc25d_api.export_xyz_to_dxf(freecad_part_object, x-size, y-size, z-size, x-depth-list, y-depth-list,
cnc25d_api.mkdir_p(directory) => 0
cnc25d_api.get_effective_args(default_args) => [args]
cnc25d_api.generate_output_file_add_argument(argparse_parser) => argparse_parser
cnc25d_api.generate_output_file(figure, filename, extrusion_height) => 0
```



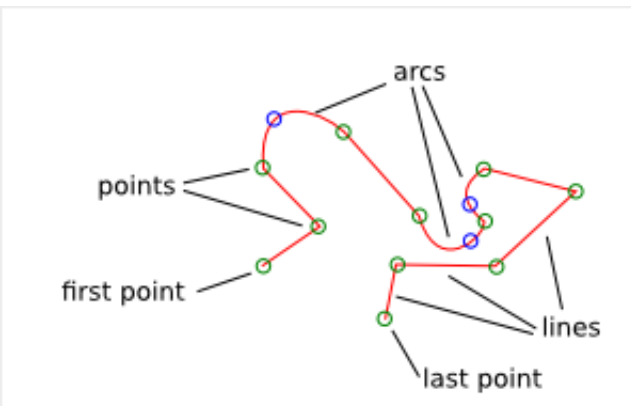
Cnc25D API functions

CHAPTER 4

Cnc25D API Outline Creation

4.1 Cnc25D outline

Cnc25D helps you to work on outline before extruding it into 3D parts. *Cnc25D outlines* are defined in the XY-plan and consist of a series of lines and/or arcs. A *line* is defined by a start point and an end point. An *arc* is defined by a start point, a passing-through point and an end point.



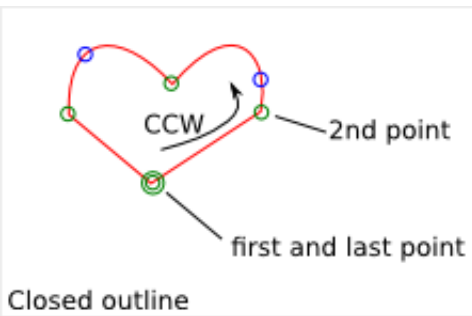
Outline example

Manipulating Cnc25D outline consists of working on 2D points. This requires much less CPU resources as invoking a complete 3D software. If you want to create other types of curve than lines or arcs, you must approximate those curves with multiple small lines.

Cnc25D outline vocabulary:

- outline: a series of segments
- segment: a line or an arc
- start-point: the starting point of a line or an arc
- end-point: the ending point of a line or an arc
- middle-point: the passing-through point of an arc (it doesn't have to be in the middle of the arc)
- first-point: the start point of the first segment of an outline
- corner: the junction between two consecutive segments.
- corner-point: the end-point of the previous segment or the start-point of the next segment

- *rbrr* : the router_bit radius request (how to transform a corner to do it millable by a router_bit of radius R?)
- closed outline: *True* if the end-point of the last segment is equal to the first-point
- outline orientation: *Counter Clock Wise* (CCW) or *Clock Wise* (CW) (this has a meaning only for closed outline)
- curved outline: outline representing a curve. The outline approximates the curve with some discrete points.
- tangent inclination: angle between the (Ox) direction and the oriented tangent of a point of an oriented curve.
- outline format A: pythonic description of an outline used as argument by the function cnc25d_api.cnc_cut_outline()
- outline format B: pythonic description of an outline returned by cnc25d_api.cnc_cut_outline() and used as argument by cnc25d_api.outline_arc_line()
- outline format C: pythonic description of a curved-outline used as argument by the function cnc25d_api.smooth_outline_c_curve()
- figure: list of format-B outlines



4.2 Cnc25D outline format A

In short, the *Cnc25D outline format A* is a list of list of 3 or 5 floats.

The purpose of the *Cnc25D outline format A* is to define your wished outline. In addition to the start, middle and end points of the segments, you define for each corner the associated *rbrr*. That means that you can request different router_bit radius for each corner. In general, you will set the same value for all corners of your outline. But you also have the flexibility to set different *rbrr* for each corner.

The first element of the *outline format A* list is the *first-point*. It is defines by a list of 3 floats: X-coordinate, Y-coordinate and the *rbrr* of the *first-point*.

The second element of the *outline format A* list is the first segment of the outline. If the first segment is a line, it is defines by a list of 3 floats: end-point-X, end-point-Y and the *rbrr* of the end-point of the segment. If the first segment is an arc, it is defines by a list of 5 floats: middle-point-X, middle-point-Y, end-point-X, end-point-Y and the *rbrr* of the end-point of the segment.

All elements of the *outline format A* list define a segment except the first element that defines the *first-point*. An outline composed of N segments is described by a list of N+1 elements. A segment is defined by 3 floats if it is a line or 5 floats if it is an arc. The start-point of a segment is never explicitly defined as it is the end-point of the previous segment. If the X and Y coordinates of the end-point of the last segment are equal to the X and Y coordinates of the first-point of the outline, the outline is closed.

rbrr (a.k.a *router_bit radius request*) defines how cnc25d_api.cnc_cut_outline() must modify a corner:

- if rbrr = 0, the corner is unchanged
- if rbrr > 0, the corner is smoothed to fit the router_bit radius *rbrr*

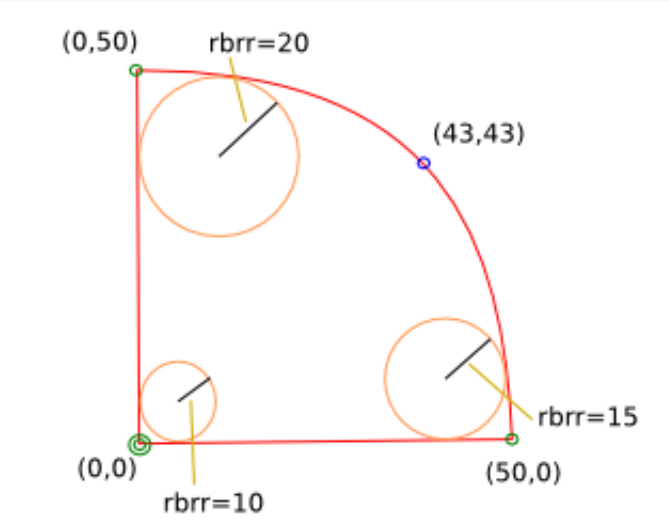
- if rbrr < 0, the corner is enlarged to fit the router_bit radius *abs(rbrr)*

Good practice: If the outline is closed, the *rbrr* of the last segment must be set to zero. If the outline is open (i.e. not closed), the *rbrr* of the *first-point* and the *rbrr* of the last segment must be set to zero.

The *outline format A* can be defined with *list* or *tuple*. The orientation of a closed outline can be CCW or CW.

outline format A example:

```
outline_A = [
  [  0,  0, 10],            # first-point
  [ 50,  0, 15],            # horizontal line
  [ 43, 43,  0,  50, 20],   # arc
  [  0,  0,  0]]            # vertical line and close the outline
```



Outline format A example

4.3 Cnc25D outline format B

The *Cnc25D outline format B* is either a *circle* or a *general outline*.

In short, a format-B circle is a list of 3 floats (center-x, center-y, radius). The *Cnc25D general outline format B* is a list of list of 2 or 4 floats.

The purpose of the *Cnc25D general outline format B* is to define an outline with points. In the general case, this is a simplification of the *outline format A*, where the *rbrr* information is removed.

The first element of the *general outline format B* list is the *first-point*. It is defines by a list of 2 floats: X-coordinate, Y-coordinate.

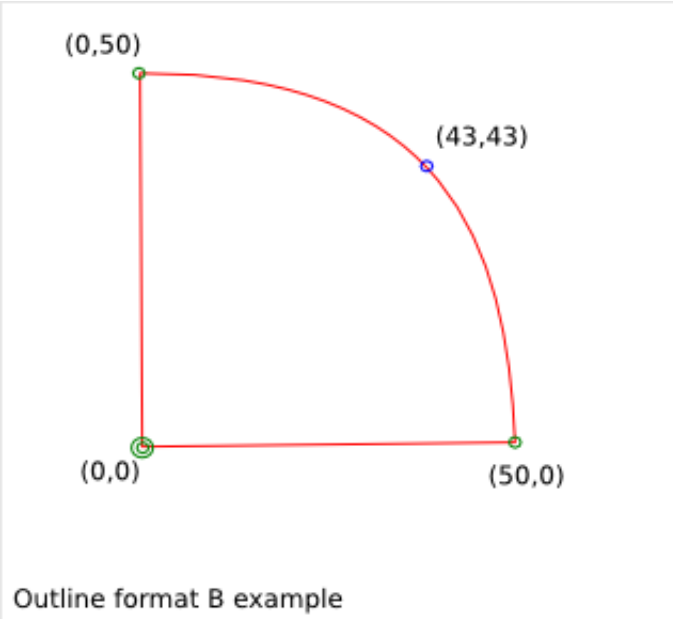
The second element of the *general outline format B* list is the first segment of the outline. If the first segment is a line, it is defines by a list of 2 floats: end-point-X, end-point-Y. If the first segment is an arc, it is defines by a list of 4 floats: middle-point-X, middle-point-Y, end-point-X, end-point-Y.

All elements of the *general outline format B* list define a segment except the first element that defines the *first-point*. An outline composed of N segments is described by a list of N+1 elements. A segment is defined by 2 floats if it is a line or 4 floats if it is an arc. The start-point of a segment is never explicitly defined as it is the end-point of the previous segment. If the X and Y coordinates of the end-point of the last segment are equal to the X and Y coordinates of the first-point of the outline, the outline is closed.

The *general outline format B* can be defined with *list* or *tuple*. The orientation of a closed outline can be CCW or CW.

general outline format B example:

```
outline_B = [
  [  0,  0],            # first-point
  [ 50,  0],            # horizontal line
  [ 43, 43,  0,  50],   # arc
  [  0,  0]]            # vertical line and close the outline
```



4.4 Cnc25D outline format C

In short, the *Cnc25D outline format C* is a list of list of 3 floats.

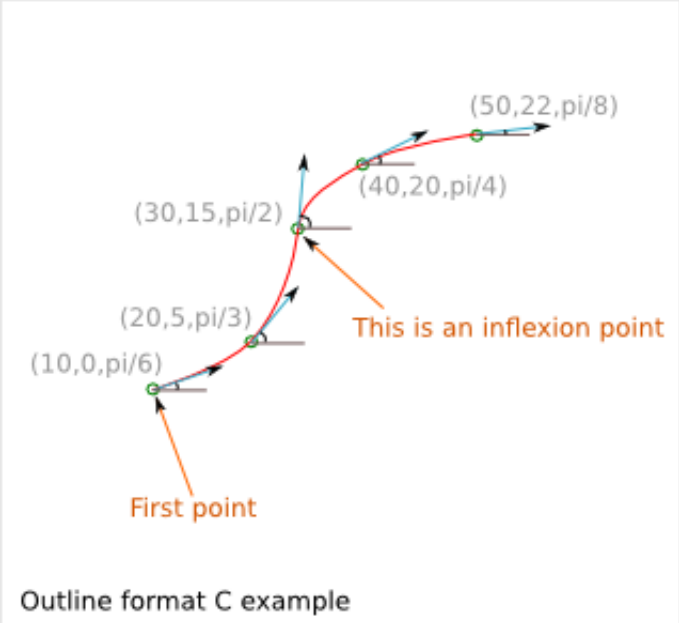
The purpose of the *Cnc25D outline format C* is to define a curved-outline with points and tangents. This is an extension of the *outline format B*, where the *tangent inclination* is added at each point. This format must be preferred to described a curved-outline.

Each element of the *outline format C* list is a curve sampling point. It is defines by a list of 3 floats: X-coordinate, Y-coordinate and the *tangent inclination angle*. The first element of the *outline format C* list is the *first-point*. The outline is oriented from the *first-point* to its last point. The *tangent inclination* is the angle (included in [-pi, pi]) between the (Ox) direction vector and the oriented curve tangent at the considered sampling point.

The *outline format C* can be defined with *list* or *tuple*.

outline format C example (the X,Y coordinates and the tangent inclination angle are rounded for a better readability):

```
outline_C = [
  [ 10,  0, math.pi/6],     # first-point
  [ 20,  5, math.pi/3],
  [ 30, 15, math.pi/2],
  [ 40, 20, math.pi/4],
  [ 50, 22, math.pi/8]]
```



The *Cnc25D outline format C* is used as argument by the function *cnc25d_api.smooth_outline_c_curve()*.

If the curved-outline contains one or several inflexion points, it is recommended to chose those points as sampling points. Thus the function *cnc25d_api.smooth_outline_c_curve()* is able to smooth the entire curved-outline. Otherwise segments containing an inflexion point are leave as line by the function *cnc25d_api.smooth_outline_c_curve()*.

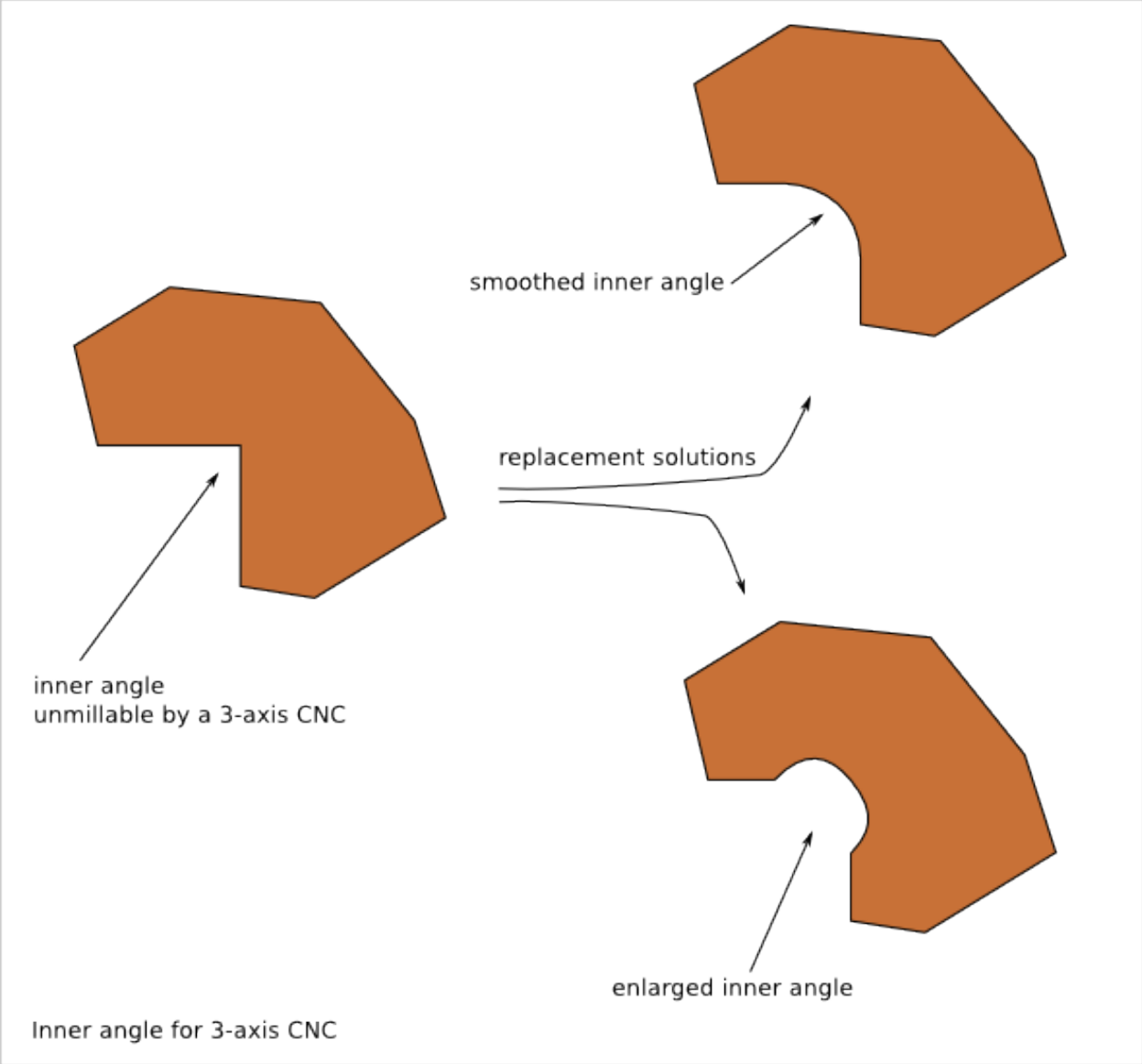
4.5 The function Cnc_cut_outline()

`cnc25d_api.` **cnc_cut_outline(** *list, string* **)**

Return a *list*.

4.5.1 cnc_cut_outline purpose

If you work with 3-axis CNC, your free XY-path gets actually some constraints due to the router_bit diameter. Real inner angle can not be manufacture and must be replaced either by a smoothed angle or an enlarged angle.

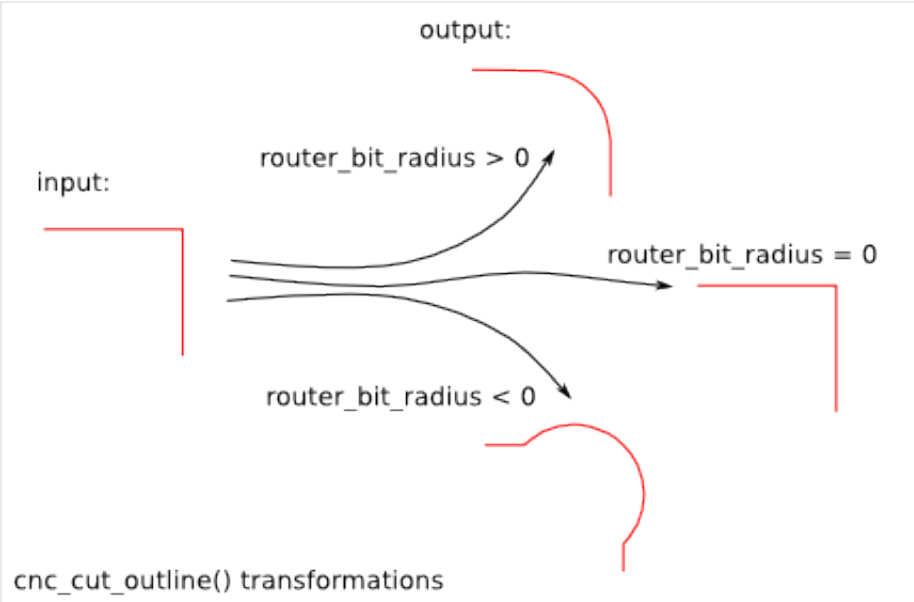


The *cnc_cut_outline* function aims at converting an outline defined by a list of points into an outline with lines and arcs makable by a 3-axis CNC. For each point, you choose if you want to enlarge the angle, smooth it or leave it sharp.

Look at the CNC Cut Outline Details chapter to get more information on when you should enlarge and when you should smooth a corner angle.

4.5.2 cnc_cut_outline usage

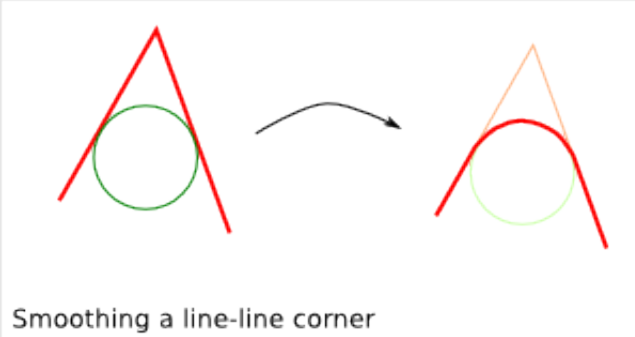
The cnc_cut_outline() function provides three possibilites as corner transformation: smooth, unchange, enlarge.

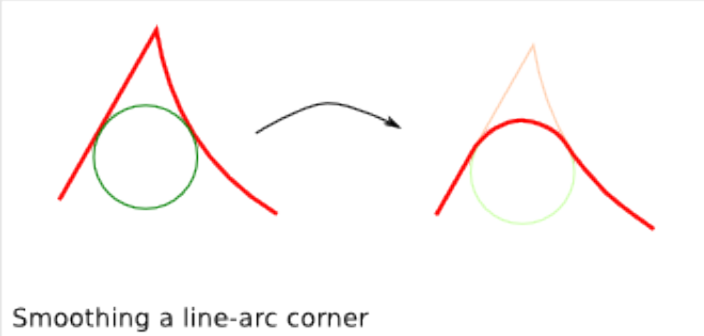


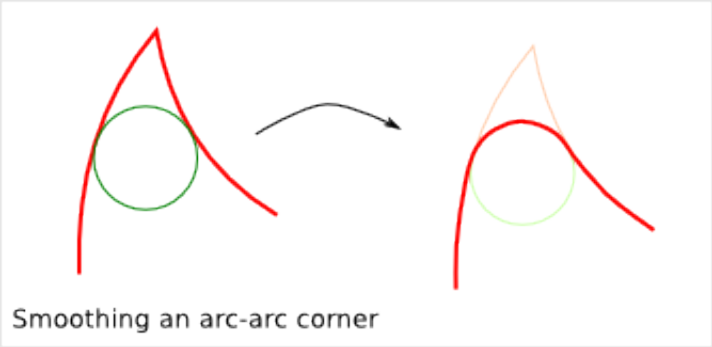
cnc_cut_outline() transformations

If *rbrr* (a.k.a. router_bit radius request) is positive, the angle is smoothed. If *rbrr* is negative, the angle is enlarged. If *rbrr* is zero, the angle is unmodified.

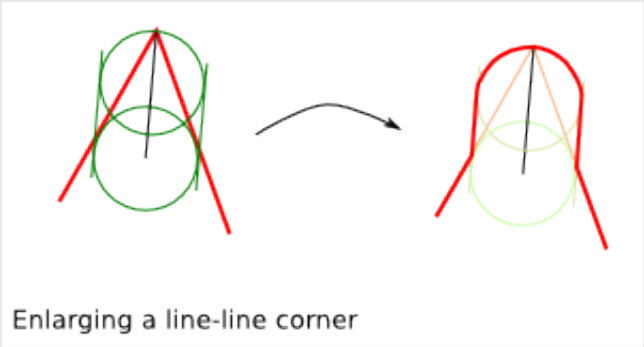
Smoothing a corner is a closed problem: there is only one arc of radius R (= *rbrr*) that is tangent to the two adjacent segments.

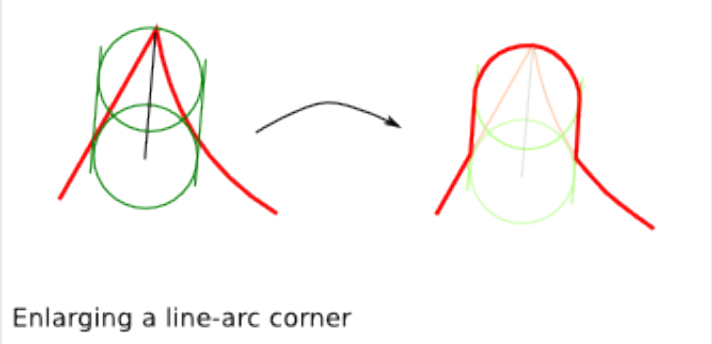

Smoothing a line-line corner

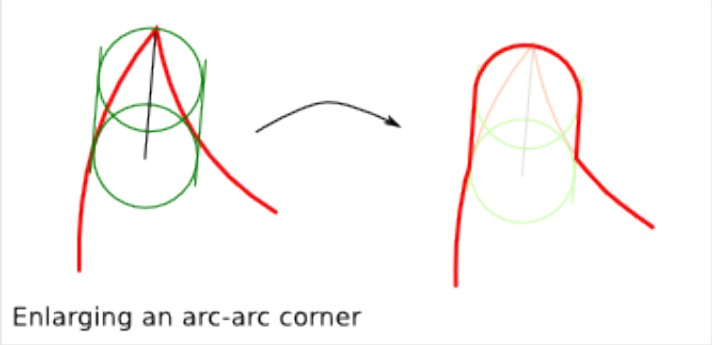

Smoothing a line-arc corner



Enlarging a corner is an open problem: there are several arcs of radius R (= *rbrr*) that can clear the wished outline. *Cnc25D* chose the arc of radius R (= *rbrr*) of which the center is on the line defined by the corner-point and the center of the associated smoothed corner. If you want an other solution, you can modify slightly your wished outline (in format A) to influence the final result as shown in the next paragraph *alternative enlarged corner*.







Notice that the interior of an closed outline is not influencing the process of smoothing or enlarging a corner. Only the local geometry (namely the two adjacent segments) influence this process.

The *cnc_cut_outline()* function needs as argument an outline of *format A* and returns an outline of *format B*. The

format B outline can easily be converted into a FreeCAD Part Object, that can be after some conversions be extruded:

```
my_outline_A = [
  [  0.0 ,  0.0,  0.0],   # this corner will be leaved sharp
  [ 20.0 ,  0.0,  5.0],   # this corner will be smoothed
  [  0.0 , 20.0, -5.0]]   # this corner will be enlarged
my_outline_B = (cnc25d_api.cnc_cut_outline(my_outline_A, "demo_my_outline_A")
my_part_face = Part.Face(Part.Wire(cnc25d_api.outline_arc_line(my_outline_B, 'freecad').Edges))
my_part_solid = my_part_face.extrude(Base.Vector(0,0,20))
```

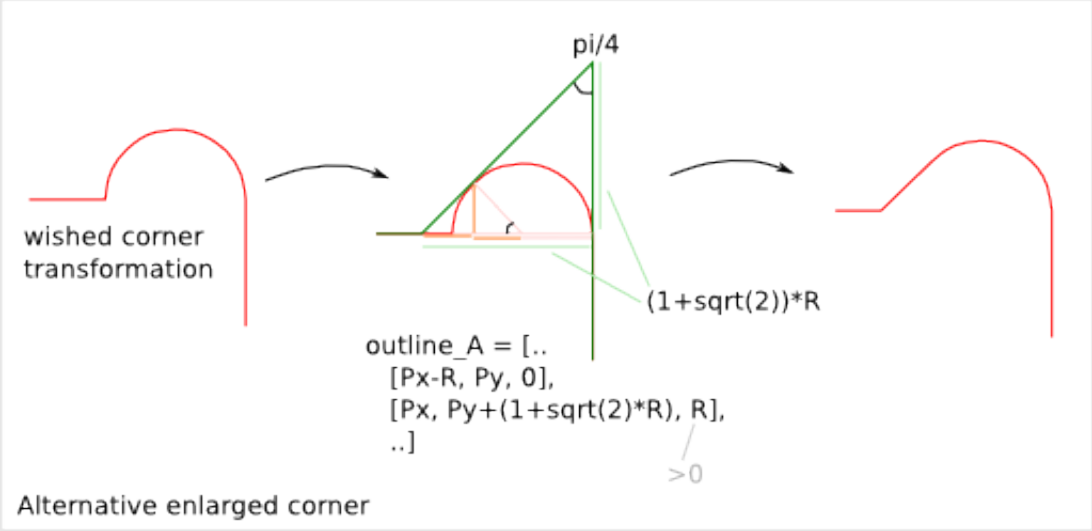
Look at the script *cnc25d_api_example.py* that you can generate with the executable *cnc25d_example_generator.py* for a more complete example.

If the requested *router_bit radius* is too large, the corner transformation may not be applied because of geometrical constraints. You get a *warning* or *error* message containing *string* set as argument. A good practice is to set *string* to the function name that calls *cnc_cut_outline()*. So you can find out which outline is not compatible with the requested *router_bit radius* in case of error. Below an example of warning message due to a too large *router_bit radius*. Thanks to the *string*, we know that the outline issue is located in the *plank_z_side* function:

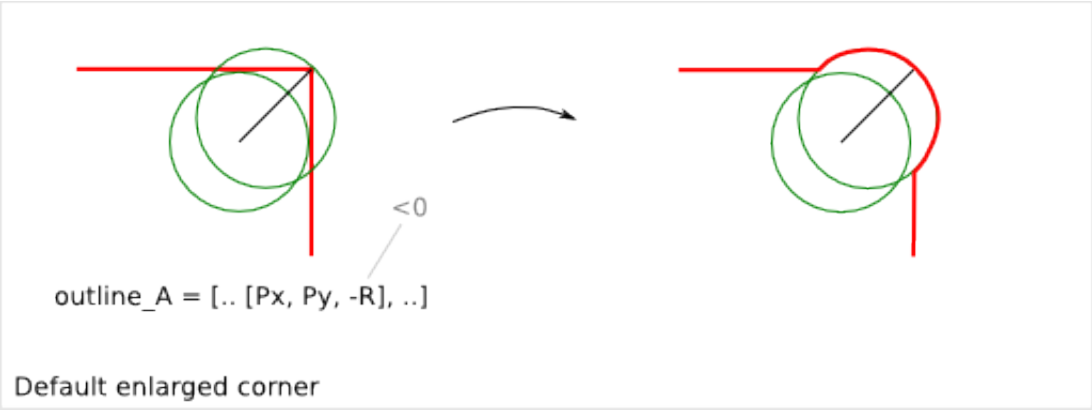
```
WARN301: Warning, corner plank_z_side.1 can not be smoothed or enlarged because edges are too short!
```

4.5.3 Alternative enlarged corner

As the problematic of enlarging a corner doesn't have a unique solution, you may want an other *enlarging corner* than the default one proposed by *cnc_cut_outline()*. For example, you may want to enlarge a corner without milling one of the adjacent segment. By changing the input outline, you can achieve it:



For comparison, the default result would be:



4.6 The function smooth_outline_c_curve()

`cnc25d_api.` **smooth_outline_c_curve(** *list, float, float, string* **)**

Return a *list*.

It reads a *format C outline* and returns a *format B outline* with the following characteristics:

- the outline is made out of arcs
- the outline goes through the sampling points
- the outline tangent at the sampling points has the requested direction (a.k.a. tangent inclination)
- the outline tangent is continuous

With an input *format C outline* of (N+1) points (i.e. N segement), the function *smooth_outline_c_curve()* returns a *format B outline* of 2*N arcs. If a segment contains an inflexion point, the arcs are replace by a line. If input points are aligned or almost aligned, arcs are also replaces by lines.

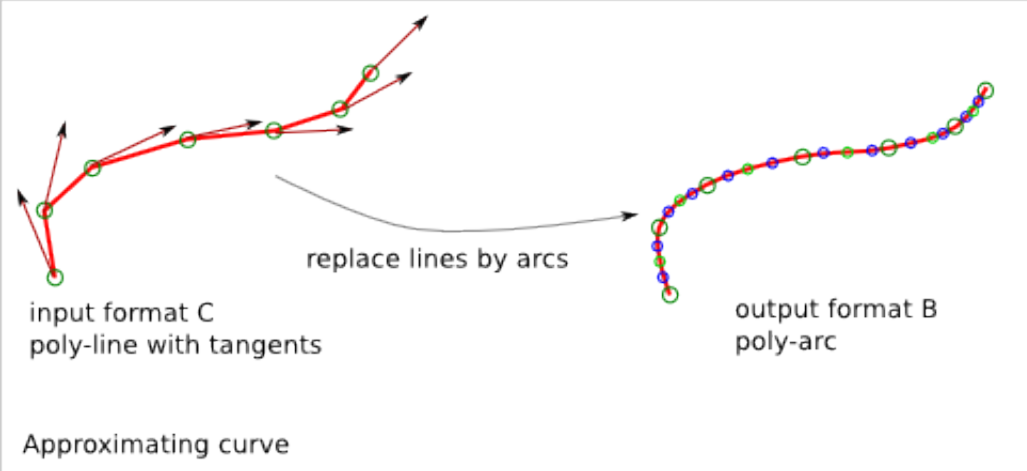
If the input curve contains *inflexion* points, choose these points as sampling points. This way, the function *smooth_outline_c_curve()* can returns an approximated outline containing only arcs. In this case, the outline tangent is continuous along the full path.

To approximate a mathematical or free-hand curve, it is better to use arcs than lines because with arcs you can keep the property of continuous tangent. Most of the 3-axis CNC can handle arcs at the motor driving level. So this function helps you to integrate your curve into a high quality workflow.

float **ai_precision**: defines the minimal angle to consider that points are not aligned and arcs must be created. Typical value: pi/1000.

flaot **ai_router_bit_request**: defines the minimal *radius of curvature* of the returned outline. If a computed arc has a radius smaller than *ai_router_bit_request*, a warning message is printed without changing the returned outline. Set *ai_router_bit_request* to your *router_bit radius*. If you get warnings, create a more regular curve or choose a smaller router_bit.

string **ai_error_msg_id**: this string is added in the error message and helps you to track bugs.



For more details on the implementation of *smooth_outline_c_curve()*, read the chapter Smooth Outline Curve Details

4.7 The function smooth_outline_b_curve()

`cnc25d_api.` **smooth_outline_b_curve(** *list, float, float, string* **)**

Return a *list*.

It reads a *format B outline* and returns a *format B outline* with the same characteristics as *smooth_outline_c_curve()*.

The function *smooth_outline_b_curve()* guests the curve tangent at each sampling point according to the previous and following sampling points and then computes the approximated outline with arcs using *smooth_outline_c_curve()*. The result is poorer than using *smooth_outline_c_curve()* because the curve tangents are approximated. Use this function only when you can not get the tangent inclinations at the sampling points.

4.8 Other outline help functions

Cnc25D outline format A and *B* reduce the description of an outline to the 2D coordinates of points. That's a drastic reduction of the amount of Data and still keeping the description accurate. But for complex outlines, a large list of point coordinates might become unreadable. It is preferable, to split a large list into comprehensive smaller sub-paths and then concatenate them. Often patterns will be used several times for an outline with some slight modifications like position (of course), scale, mirror or rotation. This is the purpose of the *outline help functions*.

The *outline help functions* accept as argument the *Cnc25D outline format A* and the *Cnc25D outline format B* and return the outline with the same format:

```
cnc25d_api.outline_shift_x(outline_AB, x-offset, x-coefficient)
cnc25d_api.outline_shift_y(outline_AB, y-offset, y-coefficient)
cnc25d_api.outline_shift_xy(outline_AB, x-offset, x-coefficient, y-offset, y-coefficient)
cnc25d_api.outline_rotate(outline_AB, center-x, center-y, rotation_angle)
cnc25d_api.outline_close(outline_AB)
cnc25d_api.outline_reverse(outline_AB)
```

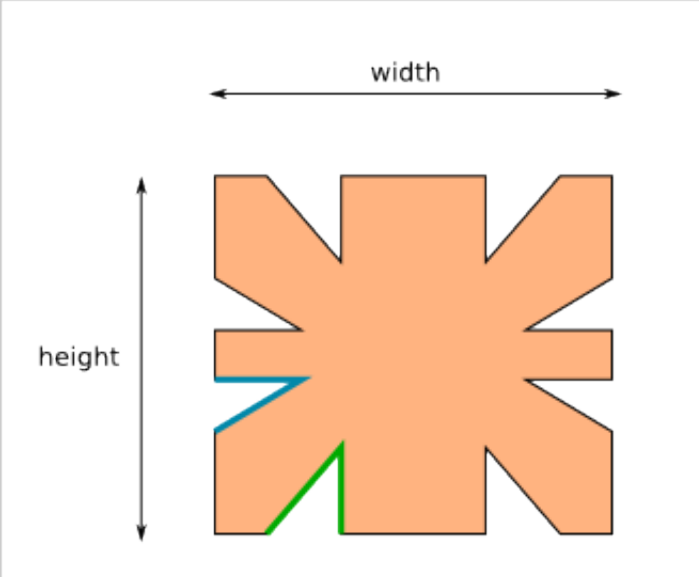
4.8.1 outline_shift

`cnc25d_api.` **outline_shift_x(** *list, x-offset, x-factor* **)**

`cnc25d_api.` **outline_shift_y(** *list, y-offset, y-factor* **)**
`cnc25d_api.` **outline_shift_xy(** *list, x-offset, x-factor, y-offset, y-factor* **)**

Return a list that defines a sub-sequence of outline.

The definition an outline can be quiet long and tedious. It might be useful to split a long list of points into several small sequences and concatenate them into one big list using the *.append()* and *.extend()* methods. Often it happens that sub-sequence patterns appear several times in one outline either shifted or mirrored. The functions *outline_shift_x*, *outline_shift_y* and outline_shift_xy can be use to help the reuse of outline sub sequences. Let's look at the following example.



Outline with repeated sub-sequences

If we want to define this outline brutally, we must create a list of 28 points. But we can also define first the blue and the green sub-sequences, which are each 3 points and create the complete outline out of them:

```
# We follow the points in the counter clock wise (CCW)
green_sequence = [
  [ 10,  0, 0],
  [ 20, 10, 0],
  [ 20,  0, 0]]
blue_sequence = [
  [  0, 25, 0],
  [ 10, 25, 0],
  [  0, 20, 0]]
width = 100
height = 80
my_outline = []
my_outline.append([0, 0, 0])
my_outline.extend(blue_sequence)
my_outline.extend(outline_shift_x(blue_sequence, width, -1))
my_outline.append([width, 0, 0])
my_outline.extend(outline_shift_x(green_sequence, width, -1))
my_outline.extend(outline_shift_xy(green_sequence, width, -1, height, -1))
my_outline.append([width, height, 0])
my_outline.extend(outline_shift_xy(blue_sequence, width, -1, height, -1))
my_outline.extend(outline_shift_y(blue_sequence, height, -1))
my_outline.append([0, height, 0])
```

```
my_outline.extend(outline_shift_y(green_sequence, height, -1))
my_outline.extend(green_sequence)
```

This code is easier to maintain.

4.8.2 outline_rotate

```
cnc25d_api.outline_rotate(outline_AB, center-x, center-y, rotation_angle)
return outline_AB
```

It applies a rotation of center (x,y) and angle *rotation_angle* to each points of the input outline.

4.8.3 outline_close

```
cnc25d_api.outline_close(outline_AB)
return outline_AB
```

If the input outline is open, it closes it with a straight line (from the end-point of the last segment to the first-point).

4.8.4 outline_reverse

```
cnc25d_api.outline_reverse(outline_AB)
return outline_AB
```

It reverses the order of the segments. If the outline is closed, that reverses its orientation (from CCW to CW or opposite). Notice that the *.reverse()* python method would not return a valid outline (format A or B) because of the *first-point* and the *middle-point* of arcs.

4.9 ideal_outline()

```
cnc25d_api.ideal_outline(outline-AC, error_mark_string)
return outline-B
```

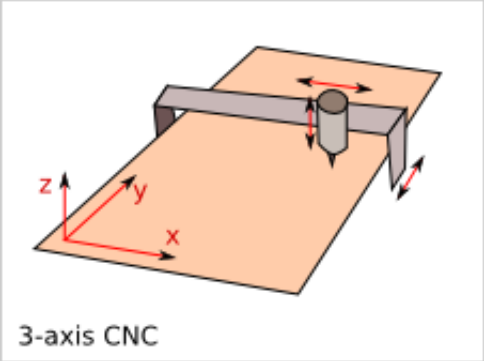
The function *ideal_outline()* lets you quickly convert a format-A or format-C outline into a format-B outline by dropping the additional information contained in the format-A and format-C. The returned format-B outline is probably to suitable for a 3-axis CNC. But you can display this *ideal* or *wished* outline in the Tkinter GUI to check the outline construction.

CHAPTER 5

CNC Cut Outline Details

5.1 Introduction to the automated cutting technology

Computer numerical control (a.k.a. CNC) lets cut material directly from computer design file (dxf, stl, g-code ...). This ensures precision, reproducibility, shape-complexity and automation.



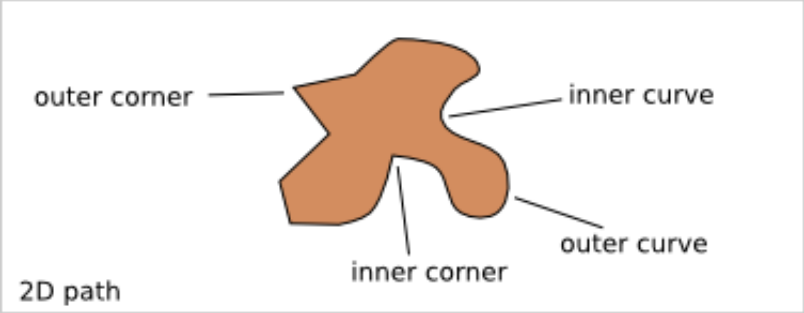
The 3-axis CNC can process:

- 2.5D : xy-path at z constant
- 3D: xyz-path in case of well adapted router_bit and path

Cutting technology:

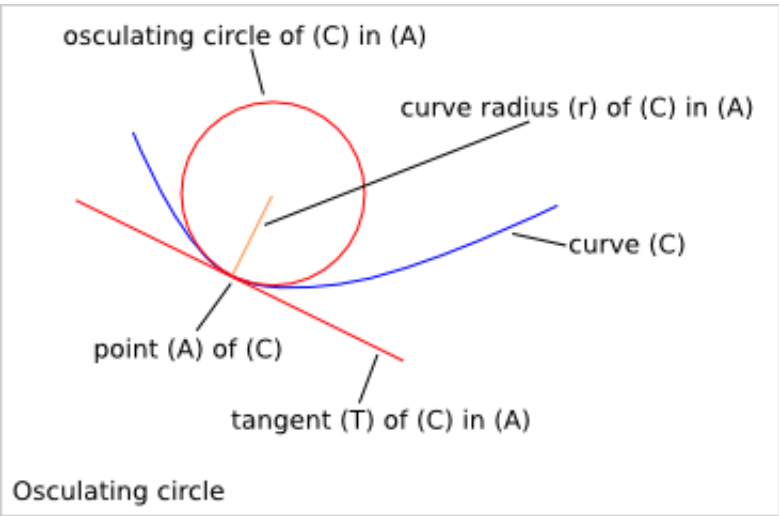
- laser cutter (Only 2D: cutting and engraving)
- water jet (Only 2D with a 3-axis machine)
- mechanical router_bit (2.5D and 3D depending on shape and router_bit shape)
- electrical discharge machining

5.2 2D path constraints

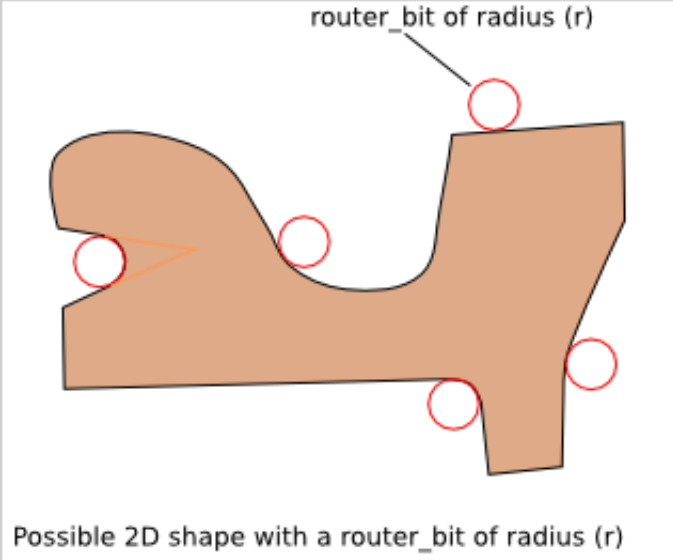


Minimal curve radius constraint:

- laser and water-jet requests no specific constraint
- For mechanical router_bit, inner curve must have a curve radius bigger than the router_bit radius.



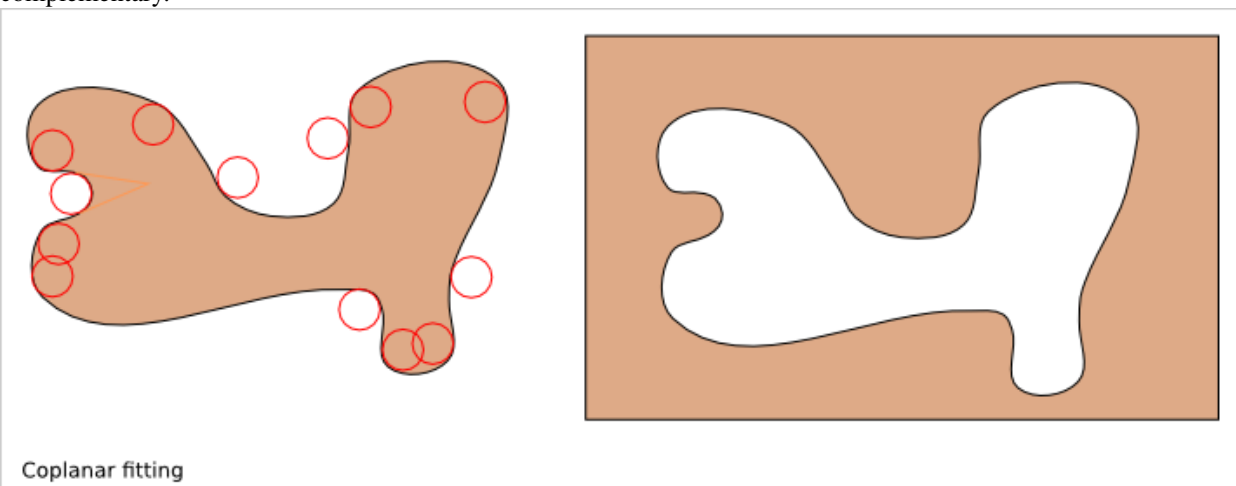
So inner corner can not be cut with router_bit. They must be replaced by inner curve. Tight inner curve must be smoothed to respect the minimal curve radius constraint.



Possible 2D shape with a router_bit of radius (r)

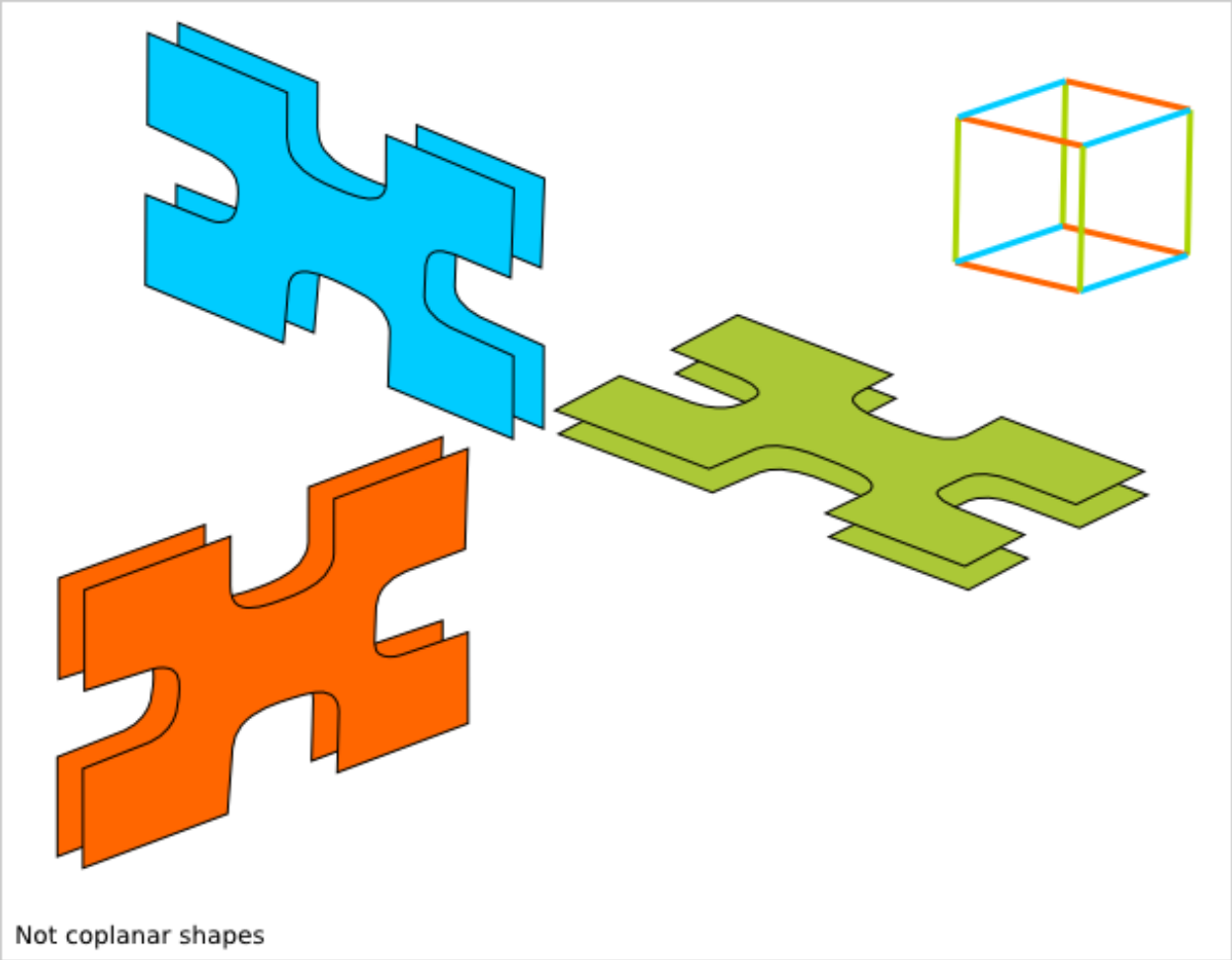
5.2.1 Coplanar fitting

If you want a perfect fitting between two coplanar shapes, then outer corners and outer curves must be rounded to get a minimum curve radius bigger than the router_bit radius. For a perfect fitting, two coplanar shapes must be complementary.



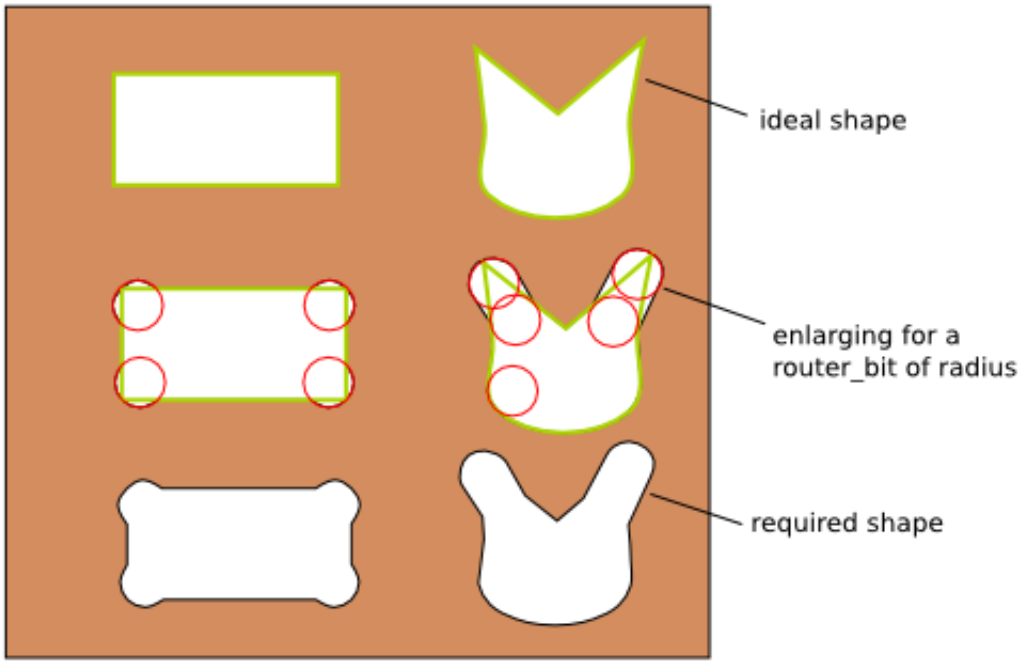
Coplanar fitting

5.2.2 Incoplanar fitting



If two parts, made out of 2D shape cut in a plan, are not coplanar, then rounding corner doesn't help the fitting of the two parts.

For fitting not coplanar shapes, we need to enlarge inner corners.

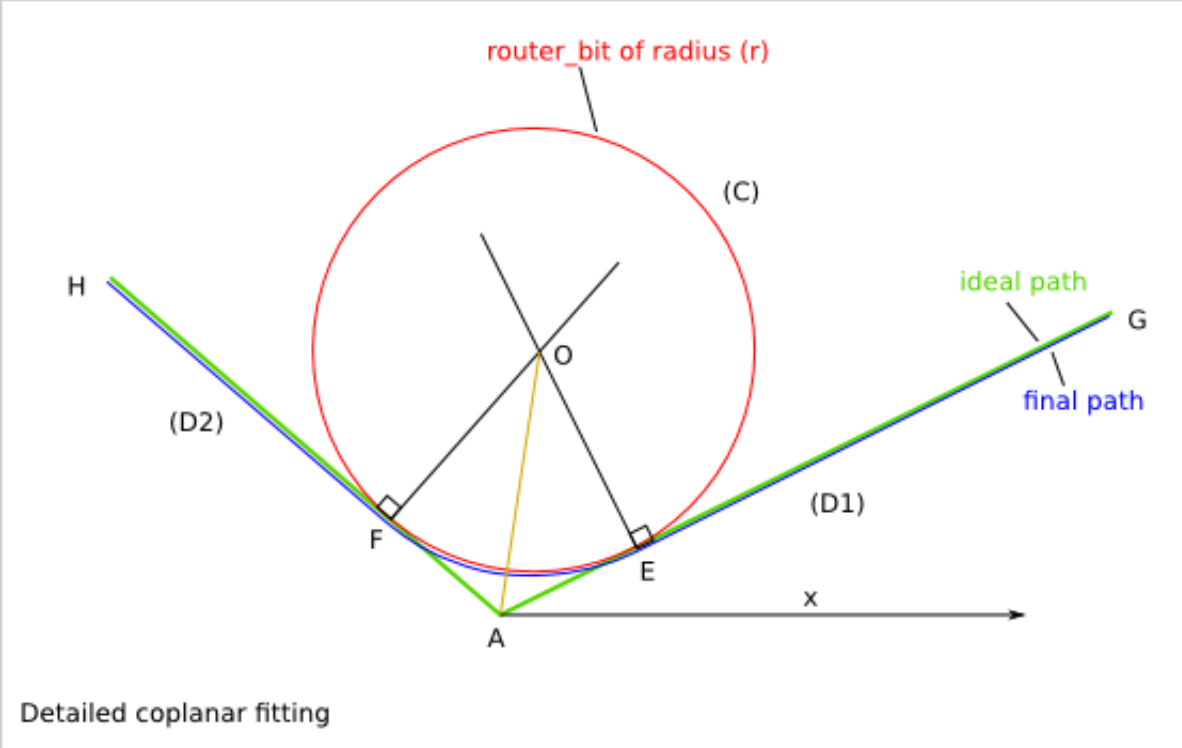


Not coplanar fitting

5.3 Coplanar fitting details

For fitting two coplanar shapes, the inner and outer corners must be smoothed.

This section details the calculation related to *smoothed line-line corner*. To get the calculation related to *smoothed line-arc corner* and *smoothed arc-arc corner*, check the *SVG* files *docs/smooth_corner_line_arc.svg* and *docs/smooth_corner_arc_arc.svg*.

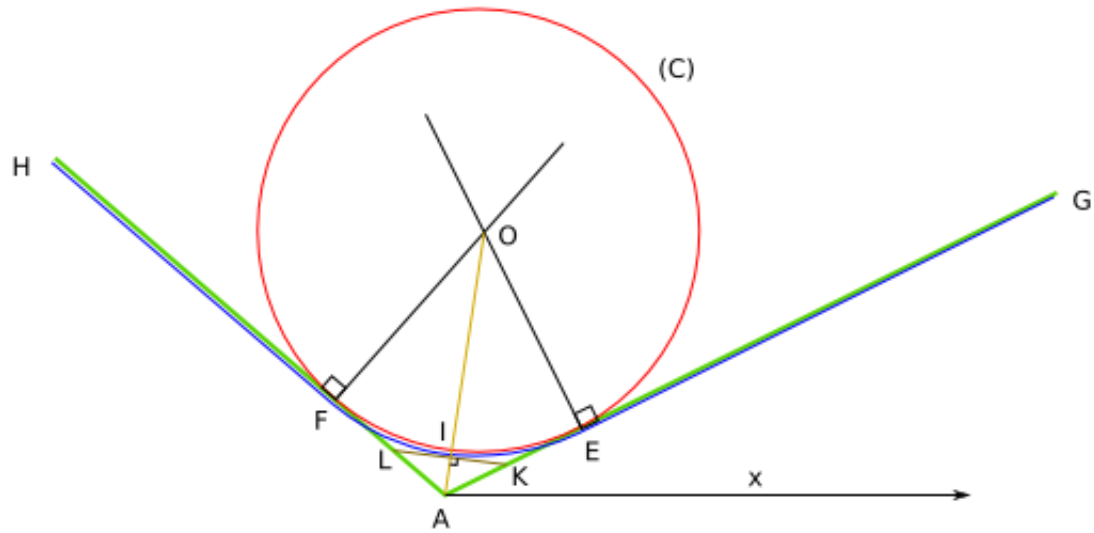



```
(D1), (D2) : two straight lines
A : intersection of (D1) and (D2)
(C) : circle or radius (r) tangent to (D1) and (D2)
E : intersection of (C) and (D1)
F : intersection of (C) and (D2)
O : the center of (C)
(EAF)=a is the angle between (D1) and (D2)

(C) is tangent to (D1), so (D1) is perpendicular to (EO)
(C) is tangent to (D2), so (D2) is perpendicular to (FO)
FO=EO=r, so O belongs to the bisector of (EAF)
We have AF=AE and (FA0)=(EAO)=a/2
AEO is right triangle in E
tan(EAO) = OE/AE
AE = r/tan(a/2)
sin(EAO) = OE/AO
AO = r/sin(a/2)

Knowing Gx,Gy,Ax,Ay,Hx,Hy, we want to calculate: a
(xAG) = atan((Gy-Ay)/(Gx-Ax))
(xAH) = atan((Hy-Ay)/(Hx-Ax))
a=(EAF)=(GAH)=(xAH)-(aAG)
a=atan((Hy-Ay)/(Hx-Ax))-atan((Gy-Ay)/(Gx-Ax))
```

Other method with the law of cosines $c^2=a^2+b^2-2*a*b*\cos(C)$

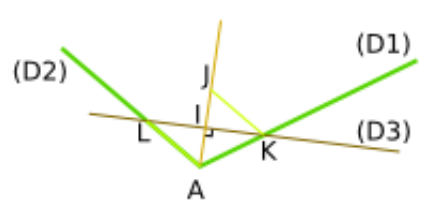
```
In the triangle GHA:
h=AG=sqrt((Gx-Ax)²+(Gy-Ay)²)
g=AH=sqrt((Hx-Ax)²+(Hy-Ay)²)
a=GH=sqrt((Hx-Gx)²+(Hy-Gy)²)
a=(GAH)=acos((h²+g²-a²)/(2*g*h))
```

```
Knowing Gx,Gy,Ax,Ay,Hx,Hy,a we want to calculate: Ex,Ey,Fx,Fy
Ex=Ax+(Gx-Ax)*AE/AG
=Ax+(Gx-Ax)*r/(tan(a/2)*sqrt((Gx-Ax)²+(Gy-Ay)²))
```



Arc third point for coplanar fitting

```
I is the intersection of (C) and (AO)
(D3) is the straight line perpendicular to (AO) and including I
K is the intersection of (D3) and (D1)
L is the intersection of (D3) and (D1)


The triangles KAI and IAL are similar so AL=AK
(LAI)=(IAK)=a/2
AI=AO-IO=r/sin(a/2)-r=r*(1-sin(a/2))/sin(a/2)
AK=AI/cos(a/2)=r*(1-sin(a/2))/(sin(a/2)*cos(a/2))=r*(1-sin(a/2))*2/sin(a)
AJ=AK+AL=(AI+IL)+(AI+IK)=2*AI
AI=(AK+AL)/2
Kx=Ax+(Gx-Ax)*AK/AG
```



Detailed arc third point for coplanar fitting

```
Knowing Gx,Gy,Ax,Ay,Hx,Hy,a we want to calculate: Ix, Iy
With E,I and F, we define the arc than can be build with a router_bit of radius r.
```

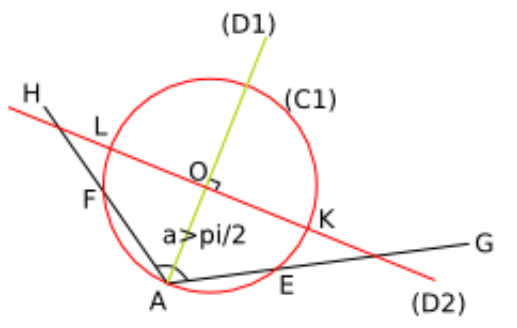
5.4 Incoplanar fitting details

For fitting two not-coplanar shapes, the inner corners must be enlarged.

This section details the calculation related to *enlarged line-line corner*. To get the calculation related to *enlarged line-arc corner* and *enlarged arc-arc corner*, check the *SVG* file *docs/enlarge_corner_arc_arc.svg*.

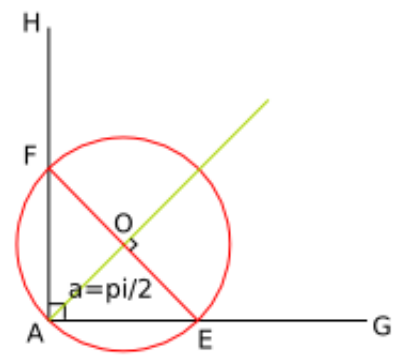
5.4.1 Angle types

Case of an inner obtuse angle



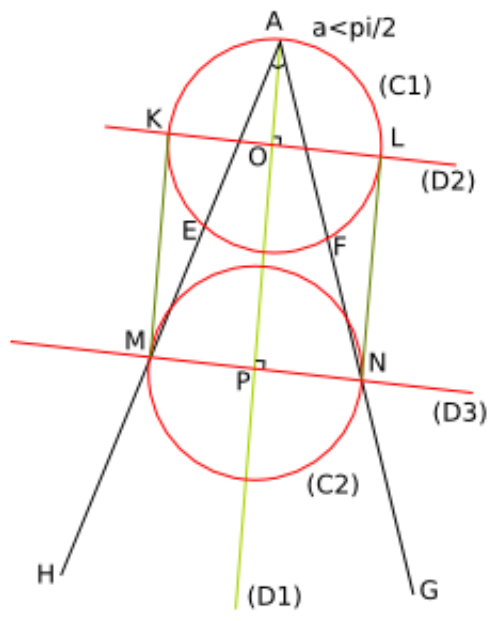
Not coplanar fitting with obtuse angle

Border case of an inner right angle



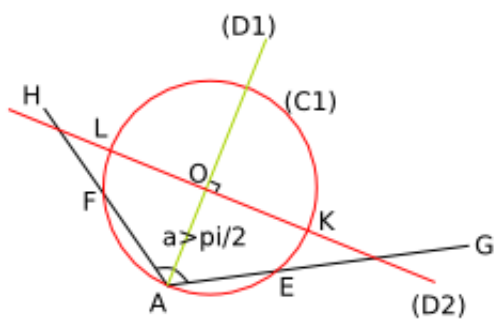
Not coplanar fitting with right angle

Case of an inner acute angle



Not coplanar fitting with acute angle

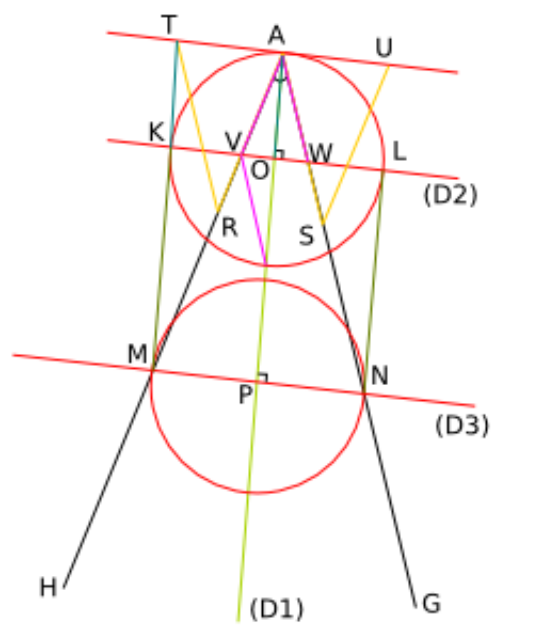
5.4.2 Calculation



Not coplanar fitting with obtuse angle

```
Let's consider three points A, G and H.
(D1) is the bisector of (GAH).
O is a point of (D1) such as AO=r
(C1) is the circle of center O and radius r
E is the intersection of (C1) and (AG)
F is the intersection of (C1) and (AH)
(D2) is the straight line perpendicular to (D1) and including O
K and L are the intersection of (D2) with (C1)
```

```
Let's calculate AE:
OA=OE=r
We define I, the orthogonal projection of O on (AE)
AI=EI because AEO is isosceles in O
AI=AO/cos(a/2)=r*cos(a/2)
AE=2*r*cos(a/2)
```



Detailed not coplanar fitting with acute angle

```
(D3) is the straight line perpendicular to (D1) and such that the length MN is equal to 2*r with M th
P is the intersection of (D3) and (D1).
AM=r/sin(a/2)

R is the middle of [AM]
S is the middle of [AN]
V is the intersection of (D2) and (AH)
W is the intersection of (D) and (AG)
AK=AR-AS+(AV+AW)/2
AR=AS=r/(2*sin(a/2))
AV=AW=r/cos(a/2)
```

CHAPTER 6

Smooth Outline Curve Details

6.1 1. Curve approximation

Most of the 3-axis CNC can handle arcs at the motor driving level. This means that arcs, like lines, can be done perfectly at the mechanical precision. All other curve types must be approximated either with small lines or in small arcs in an earlier stage of the design workflow.

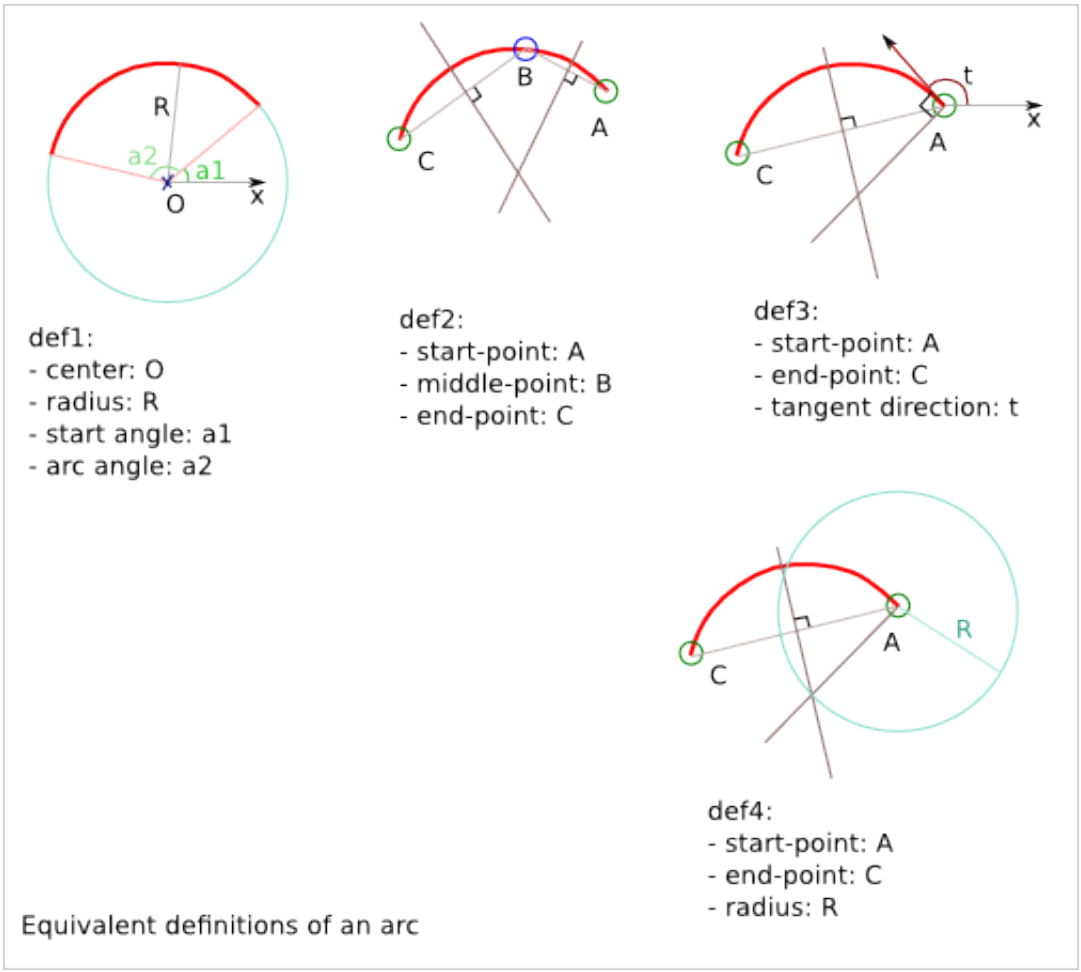
Approximating with lines is simple but you lose the continuity of the tangent along the path.

Approximating with arcs let you keep the continuity of the tangent along the path. This is probably what you want to approximate your mathematical curve or your free-hand curve.

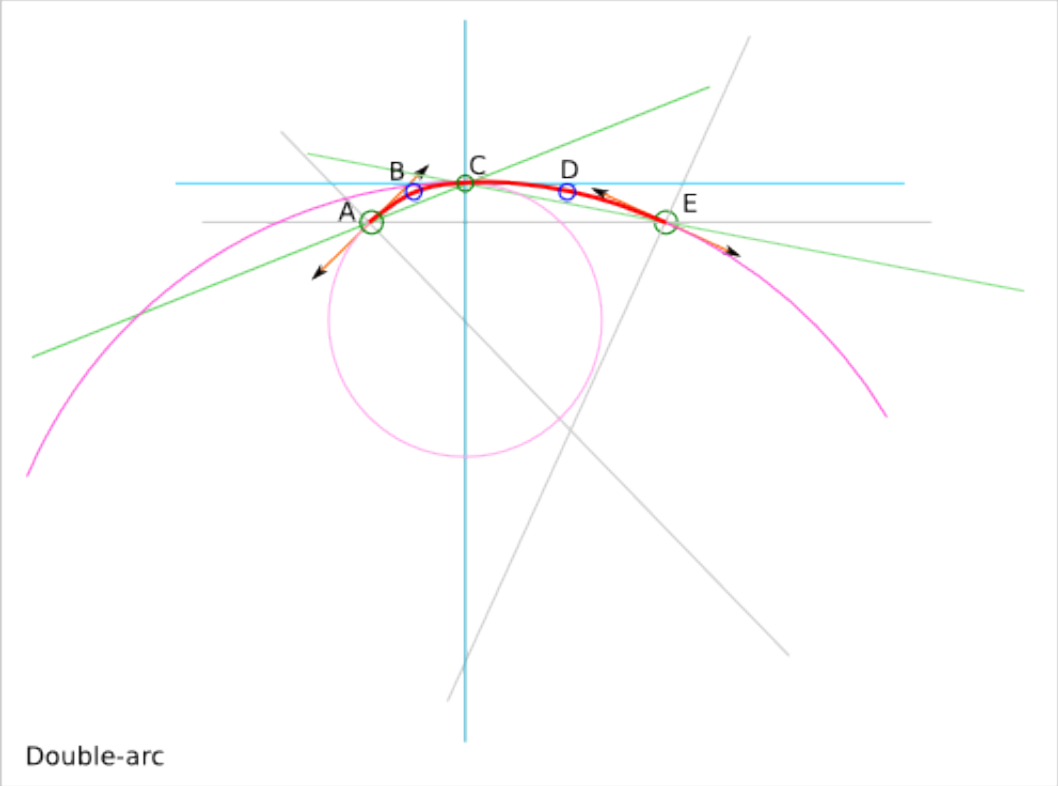
6.2 2. Double-arc solution

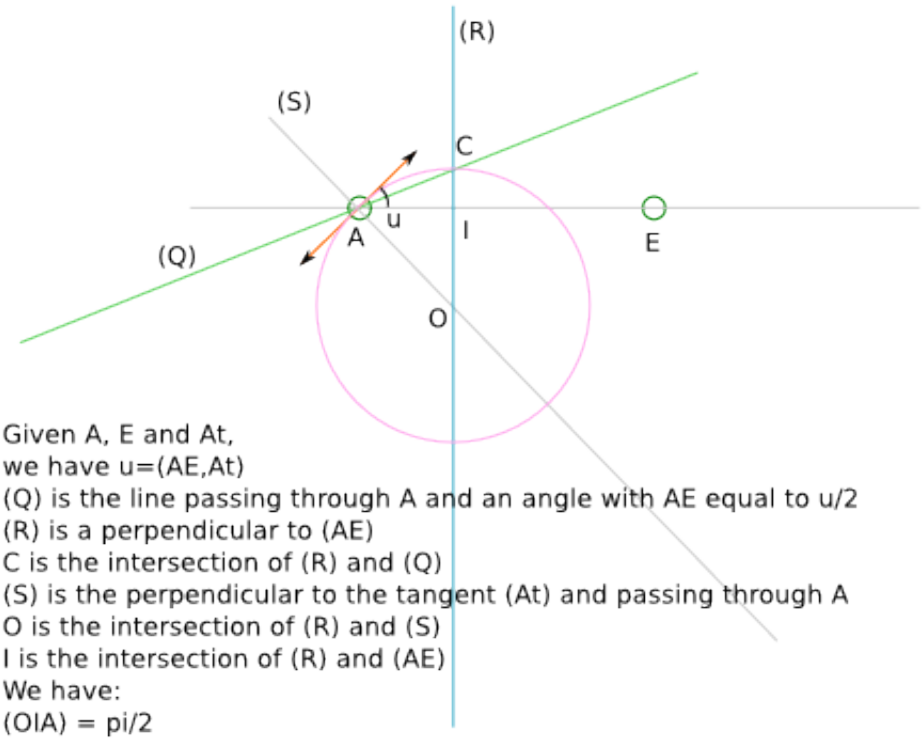
The function *smooth_outline_c_curve()* use the double-arc solution to approximate a *segment* of curve.

Given two points, A and E, and their tangent directions, you can construct two arcs that are joined in C with a common tangent direction (parallel to the line (AE)) and with the first arc that starts in A with the requested tangent direction and the second arc that ends in E with the requested tangent direction.



Equivalent definitions of an arc





Given A, E and At,
we have u=(AE,At)
(Q) is the line passing through A and an angle with AE equal to u/2
(R) is a perpendicular to (AE)
C is the intersection of (R) and (Q)
(S) is the perpendicular to the tangent (At) and passing through A
O is the intersection of (R) and (S)
I is the intersection of (R) and (AE)
We have:
(OIA) = pi/2
(IAO) = pi/2 - u
so, (AOI) = u
and (AOC) = (AOI) = u
(OAC) = (OAI)+(IAC) = pi/2 - u + u/2 = pi/2-u/2
(OCA) = pi - (AOC) - (OAC) = pi - pi/2 + u/2 - u = pi/2-u/2 = (OAC)
So, AOC is an isosceles triangle in O.
OA=OC
For any (R), the arc of center O, pass through A and is tangent to At
and pass through C and its tangent is parallel to (AE)
Doing the same process with the point E, we know that the jonction of
the double arcs must be on (Q) of A and the (Q') of (E).
So we choose C intersection of (Q) and (Q')

Double-arc: Construction of C

The file *docs/smooth_polyline.svg* contains other solution attempts.

CHAPTER 7

Cnc25D API Outline Utilization

7.1 Transformations at the figure-level

The description of a 2.5D part can require several outlines. Typically one outline is the outer shape of the part, the other outlines are holes in this part. In the Cnc25D API, a list of outlines is called a *figure*. After creating such a list, you can directly display this *figure*, write it in a file or extrude it in 3D with FreeCAD.

7.2 Display a figure in a GUI

```
cnc25d_api.figure_simple_display(graphic_figure, overlay_figure)
return 0
```

graphic_figure is a list of format-B outlines to be displayed in *red*. *overlay_figure* is optional and could be used to display an other figure in *orange* when the overlay is active. A common practice it to set *graphic_figure* with the outlines returned by *cnc_cut_outline()* and to set *overlay_figure* with outlines returned by *ideal_outline()*. So you can see your created format-A outlines and the final format-B outlines. Notice that you can also directly use format-A or format-C without converting them in format-B with *ideal_outline()*, but you will get a *warning* message.

If you want more control on the figure display like new *colors*, *width* or *animations*, then you should use *outline_arc_line()* and *Two_Canvas* directly.

7.3 Write a figure in a SVF file

```
cnc25d_api.write_figure_in_svg(figure, filename)
return 0
```

7.4 Write a figure in a DXF file

```
cnc25d_api.write_figure_in_dxf(figure, filename)
return 0
```

7.5 Extrude a figure using FreeCAD

```
cnc25d_api.figure_to_freecad_25d_part(figure, extrusion_height)
return FreeCAD Part Object
```

To create a 3D part from a *figure*, the function *figure_to_freecad_25d_part()* makes the assumption that the first outline is the *outer line* and the remaining outlines are holes.

7.6 Detailed transformations at the outline-level

After getting a *Cnc25D format B outline* from the cnc_cut_outline() function, you probably want to use this outline in CAD tools. The function *cnc25d_api.outline_arc_line()* lets you transform the *Cnc25D format-B outline* into one of this four formats: *freecad*, *svgwrite*, *dxfwrite*, *tkinter*.

```
cnc25d_api.outline_arc_line(outline-B, backend) => Tkinter or svgwrite or dxfwrite or FreeCAD stuff
  with backend=['freecad', 'svgwrite', 'dxfwrite', 'tkinter']
```

7.6.1 freecad

outline_arc_line(outline_B, 'freecad') returns *FreeCAD Part.Shape* object that can be used easily in the classic *FreeCAD* workflow:

```
my_part_shape = cnc25d_api.outline_arc_line(my_outline_B, 'freecad')
my_part_face = Part.Face(Part.Wire(my_part_shape.Edges))
my_part_solid = my_part_face.extrude(Base.Vector(0,0,20))
```

Notice that *FreeCAD* conserve the *arc* geometrical entity during its complete workflow. So after extruding the outline, slicing the part and then projecting it again in a DXF file, you still get the *arcs* you have designed in your original outline.

7.6.2 svgwrite

A *Cnc25D format B outline* is a 2D vectorial shape that can be transposed in a SVG file. *SVG file* is one of the usual input format for the 3-axis CNC tool chain. This snippet let you dump the *Cnc25D format B outline* in a *SVG* file:

```
import svgwrite
my_outline_B = [ .. ]
object_svg = svgwrite.Drawing(filename = "my_ouline.svg")
svg_outline = cnc25d_api.outline_arc_line(my_outline_B, 'svgwrite')
for one_line_or_arc in svg_outline:
  object_svg.add(one_line_or_arc)
object_svg.save()
```

Cnc25D relies on the *Python package* svgwrite from **mozman**. Use Inkscape to review the generated *SVG* file.

Warning: The SVG format supports the *arc* graphical object but the Python package svgwrite has not implemented yet the *arc* constructor. So *Cnc25D* transform each *arc* of the outline into a series of small segments. This might be an issue for certain *CNC tool chain* or for some designs.

7.6.3 dxfwrite

A *Cnc25D format B outline* is a 2D vectorial shape that can be transposed in a DXF file:

```
import dxfwrite
my_outline_B = [ .. ]
object_dxf = DXFEngine.drawing("my_outline.dxf")
#object_dxf.add_layer("my_dxf_layer")
dxf_outline = cnc25d_api.outline_arc_line(my_outline_B, 'dxfwrite')
for one_line_or_arc in dxf_outline:
  object_dxf.add(one_line_or_arc)
object_dxf.save()
```

Cnc25D relies on the *Python package* dxfwrite from **mozman**. Use LibreCAD to review the generated *DXF* file.

Warning: Like previously, the DXF format supports the *arc* graphical object but the Python package dxfwrite has not implemented yet the *arc* constructor. So *Cnc25D* transform each *arc* of the outline into a series of small segments. This might be an issue for certain *CNC tool chain* or for some designs.

7.6.4 tkinter

During the early phase of the design, you just need to view the outline (that still might be under-construction) without using the powerful *FreeCAD* or dumping files. This is the purpose of the *Tkinter GUI*. Check the design example *cnc25d_api_example.py* generated by the binary *cnc25d_example_generator.py* or check the file *cnc25d/tests/cnc25d_api_macro.py* to see how to implement this small *graphic user interface*.

```
cnc25d_api.Two_Canvas(Tkinter.Tk()) # object constructor
```

CHAPTER 8

Cnc25D API Working with FreeCAD

8.1 import FreeCAD

`cnc25d_api.` **importing_freecad()**

Modify the global variable *sys.path*.

FreeCAD comes with Python modules. But these FreeCAD modules are not installed in one of the standard directories. You will find the Python FreeCAD modules in a directory such as */usr/lib/freecad/lib*. To use FreeCAD from a Python script, you need either to set the PYTHONPATH system environment variable or to extend the sys.path Python variable.

Because you need to import FreeCAD at each beginning of scripts, this task as been implemented in the module *cnc25d_api.py* that is installed in a standard location. So, after installing Cnc25D, to use the FreeCAD modules, you only need to add those lines at the beginning of your Python script:

```
from cnc25d import cnc25d_api
cnc25d_api.importing_freecad()
```

The function *importing_freecad()* looks for the FreeCAD modules using a location list. If the function *importing_freecad()* doesn't manage to find FreeCAD on your system, you may need to edit the module *importing_freecad.py* and add the path to the FreeCAD modules to the *FREECADPATH* list.

8.2 place_plank()

`cnc25d_api.` **place_plank(** *FreeCAD.Part.Object, x-size, y-size, z-size, flip, orientation, x-position, y-position, z-posistion* **)**

Return a *FreeCAD.Part.Object*

FreeCAD provides the usual *rotate* and *translate* methods to place an object in a construction-assembly. Even if those methods are mathematically straight forward, they might require many *tries and errors* to find out the correct rotation to apply to an object to place it correctly in an assembly. The *place_plank()* function provides an alternative to the *rotate* method when you want to place a object in a cuboid assembly.

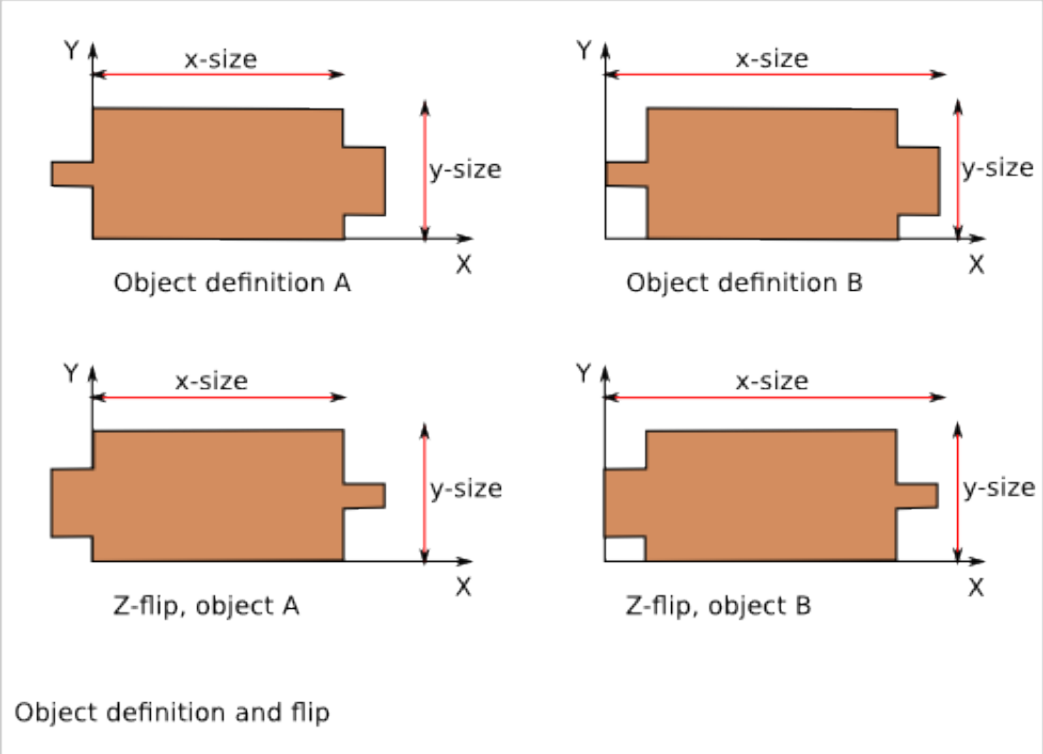
To help positioning object we have the following conventions:

- The largest size of an object defines the *main axis* of the object.
- The second largest size of an object defines the *second axis* of the object.
- During the object construction, we choose the X axis as *main axis* and the Y axis as *second axis*.

A cuboid assembly is a construction where most of the objects have their *main axis* parallel to the X, Y or Z-axis. To place an object, construed with the above conventions, in a cuboid assembly, you can define the rotation of the object with two natural parameters:

- the orientation of the *main and second axis*. There are just six possibilities: 'xy', 'xz', 'yx', 'yz', 'zx' and 'zy'. For example, 'yx' means that the *main axis* of the object is parallel to the Y-axis of the reference frame and the *second axis* of the object is parallel to the X-axis.
- the flip of the object. After defining the orientation of the *main axis* and *second axis*, there are still four possibilities called *flip*: 'identity', 'x-flip', 'y-flip' and 'z-flip'.

The *place_plank()* function uses this approach to place a object in an cuboid assembly. To realize flip and orientation, the *place_plank()* function needs to know the sizes along X, Y and Z of the object. Those sizes are virtual and you can play with them for your convenience.



A physical object can be defined in several ways respecting our *main and second axis* conventions. The choice of the definition influences the behavior of the *flip*. Knowing that, choose the most convenient definitions for your design.

Look at the Plank Positioning Details chapter to get more explaination on rotation, orientation and flip transformations.

8.3 Drawing export

FreeCAD provides very efficient methods for 3D export such as *.exportBrep()*, *.exportStep()* or *.exportStl()*. It also provides full customizable 2D export methods such as *.slice()* and *projectToDXF()*. **Cnc25D** provides simple functions that covers the most standard usage of the 2D export.

8.3.1 Cut export as DXF

`export_2d.` **export_to_dxf(** *FreeCAD.Part.Object, FreeCAD.Base.Vector, depth, path* **)**

Write the DXF file *path*.

The *export_to_dxf()* function performs two successive operations:

- It cuts a slice of the *FreeCAD.Part.Object* according to the direction *FreeCAD.Base.Vector* and the *depth*.
- It writes the DXF file *path* containing the projection of the slice.

If you are designing a 2.5D part, this function is useful to get the DXF file that will be used by the CNC workflow.

Usage example:

```
export_2d.export_to_dxf(my_part_solid, Base.Vector(0,0,1), 1.0, "my_part.dxf")
```

8.3.2 Cut export as SVG

`export_2d.` **export_to_svg(** *FreeCAD.Part.Object, FreeCAD.Base.Vector, depth, path* **)**

Write the SVG file *path*.

The *export_to_svg()* function performs the same operations as *export_to_dxf()* except it write a SVG file.

Usage example:

```
export_2d.export_to_svg(my_part_solid, Base.Vector(0,0,1), 1.0, "my_part.svg")
```

Warning: The function *export_to_svg()* only works when it is used in a script run from the FreeCAD GUI. This is because of a current limitation of the FreeCAD library function *Drawing.projectToSVG()*.

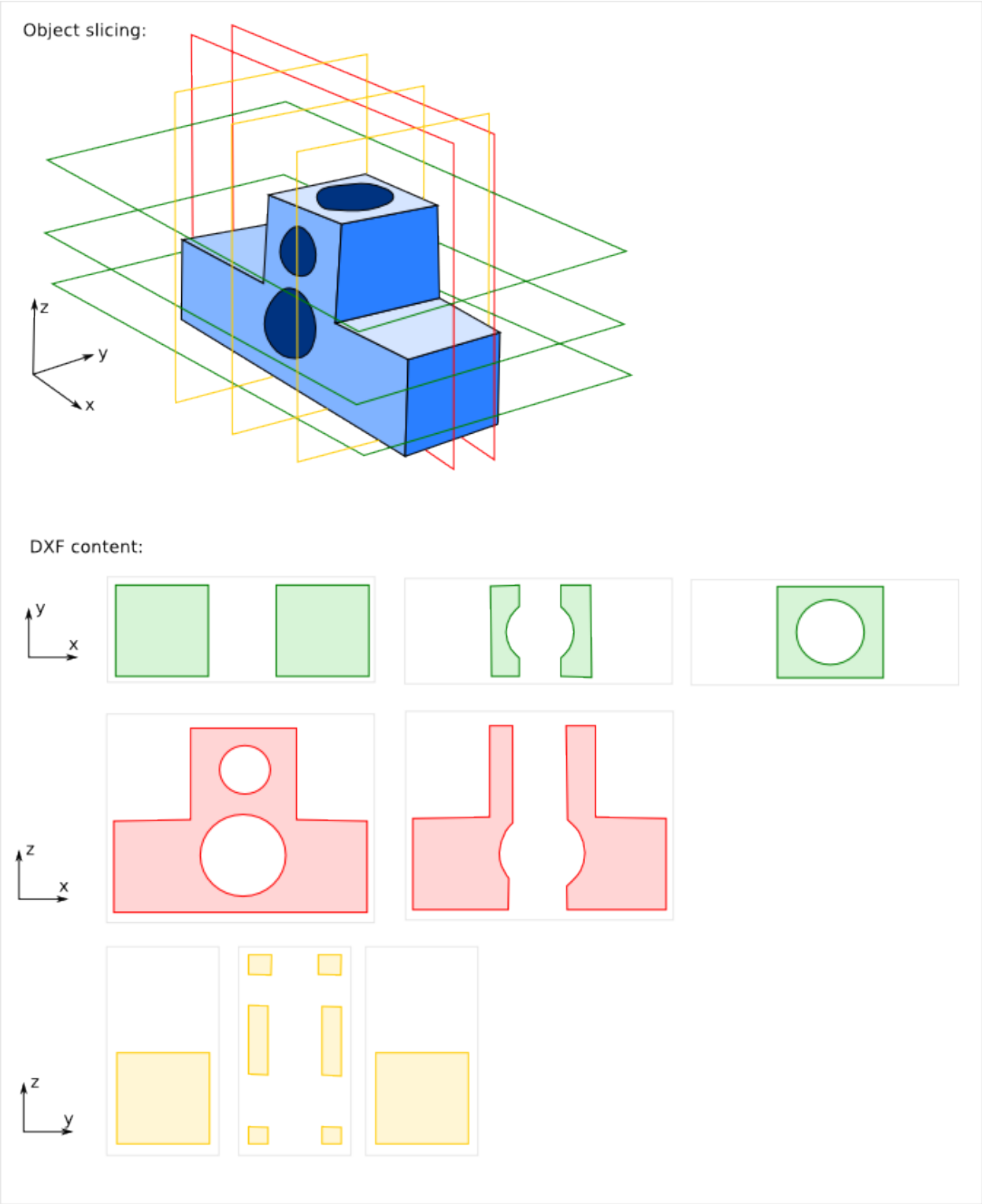
8.3.3 XYZ scanning

`export_2d.` **export_xyz_to_dxf(** *FreeCAD.Part.Object, x-size, y-size, z-size, x-list, y-list, z-list, path* **)**

Write the DXF file *path*.

The *export_xyz_to_dxf()* function cuts in many slices the *FreeCAD.Part.Object* according to the three directions of the reference frame axis X, Y and Z. The depth of the slices are provided by the three argument lists *x-list*, *y-list* and *z-list*. All the slices are placed in the plan XY and are written in the DXF file *path*.

The result looks like a medical scan. This is a more comfortable and readable document than the CAD tradition 3 views projections. This helps to show up weaknesses of designs if you choose good slices.



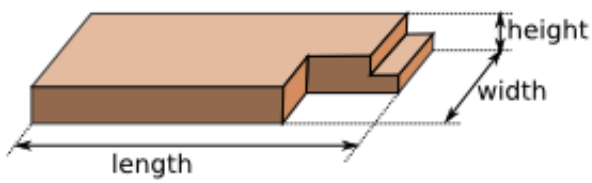
Usage example:

```
xy_slice_list = [ 0.1+20*i for i in range(12) ]
xz_slice_list = [ 0.1+20*i for i in range(9) ]
yz_slice_list = [ 0.1+20*i for i in range(9) ]
export_2d.export_xyz_to_dxf(my_assembly, 180.0, 180.0, 240.0, xy_slice_list, xz_slice_list, yz_slice_
```

CHAPTER 9

Plank Positioning Details

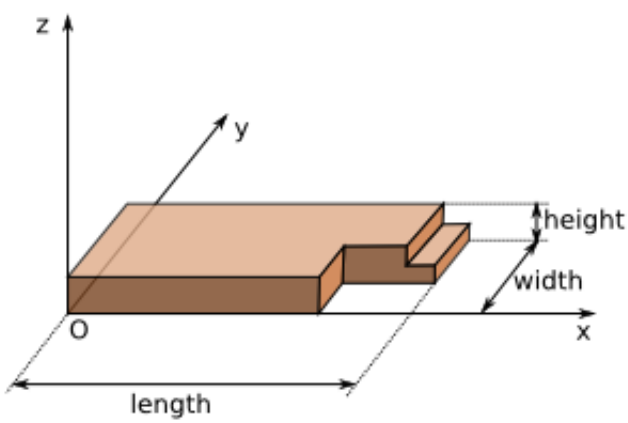
9.1 Plank definition



Plank with cuts

We call plank a 3D shape with a rectangular cuboid as construction base. The rectangular cuboid is defined by the three values: length, width and height with the relations: length > width > height. With addition ad-hoc conventions, any shape can be considered as a plank.

9.2 Plank reference frame



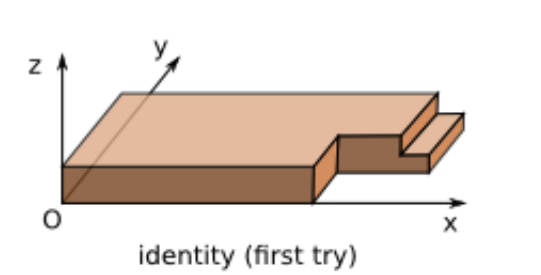
Plank reference frame

We choose the reference frame such as:

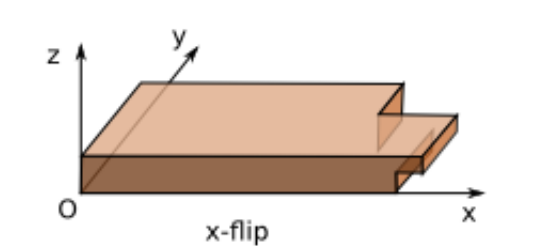
- x is the length direction
- y is the width direction
- z is the height direction
- the origin (O) is one of the corner of the base cuboid
- the main part of the plank has positive coordinates (x,y,z) in this reference frame
- (O,x,y,z) is orthonormal direct.

9.3 Plank flip possibilities

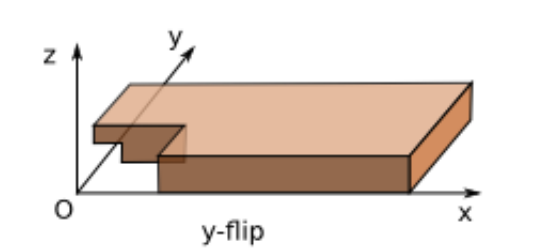
According to the plank reference frame definition, there are four possibilities to place the plank within this reference frame.



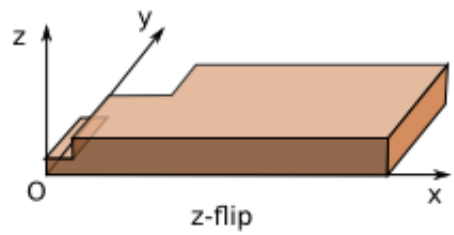
Flip possibility 1



Flip possibility 2



Flip possibility 3



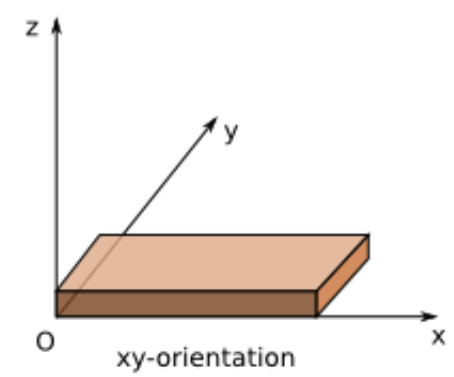
Flip possibility 4

Notice that z-flip is equivalent to the combination of x-flip and y-flip.

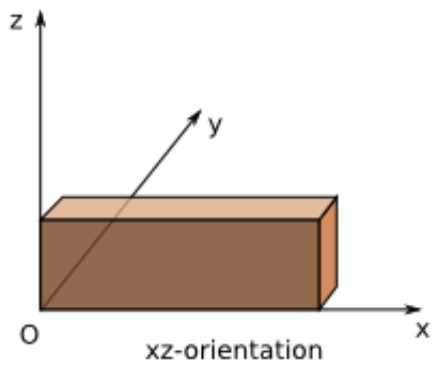
9.4 Plank orientation possibilities

We focus only on cuboid construction. Namely each plank of the construction is parallel to one of the 3 axis X, Y and Z of a given orthogonal reference frame.

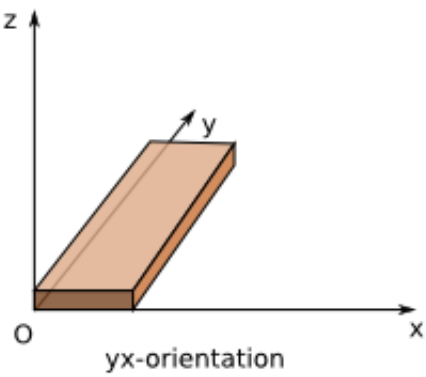
Considering a simple plank (just a rectangular cuboid wihtout cut), the position of the plank is not influenced by flip along x, y and z. In a given reference frame, this plank has six possible orientations in a cuboid construction. An orientation is marked by the length direction axis followed by the width direction axis. With this nomenclature, the six orientations are: 'xy', 'xz', 'yx', 'yz', 'zx' and 'zy'.



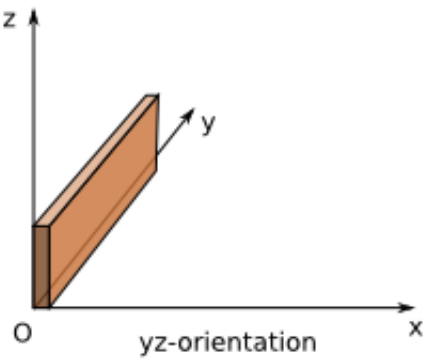
Orientation possibility 1



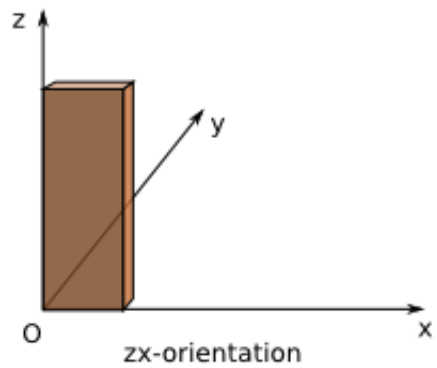
Orientation possibility 2



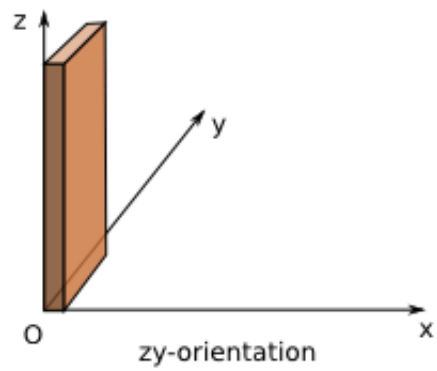
Orientation possibility 3



Orientation possibility 4



Orientation possibility 5



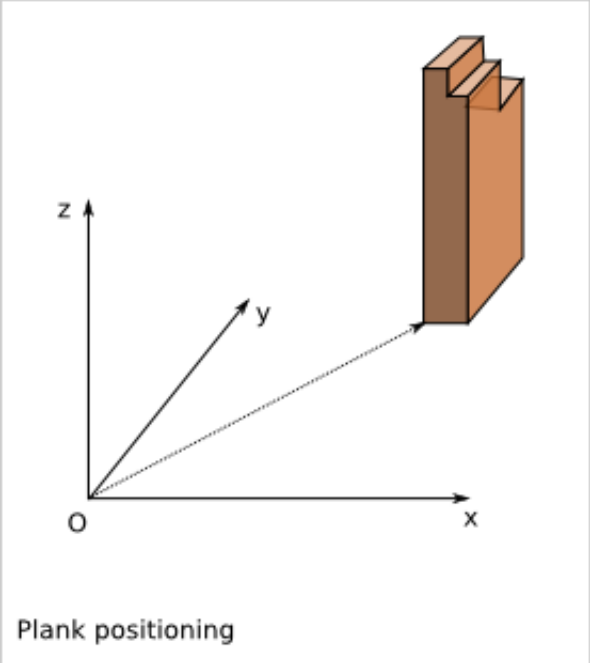
Orientation possibility 6

9.5 Plank position in a cuboid construction

The position of a plank (or assimilated) in a cuboid construction can be defined by three operations:

- flip (identity, x-flip, y-flip, z-flip)
- orientation ('xy', 'xz', 'yx', 'yz', 'zx', 'zy')
- translation (x,y,z)

The function place_plank() realizes those operations. To realize those three operation, the function needs also as argument the length, the width and the height of the plank.



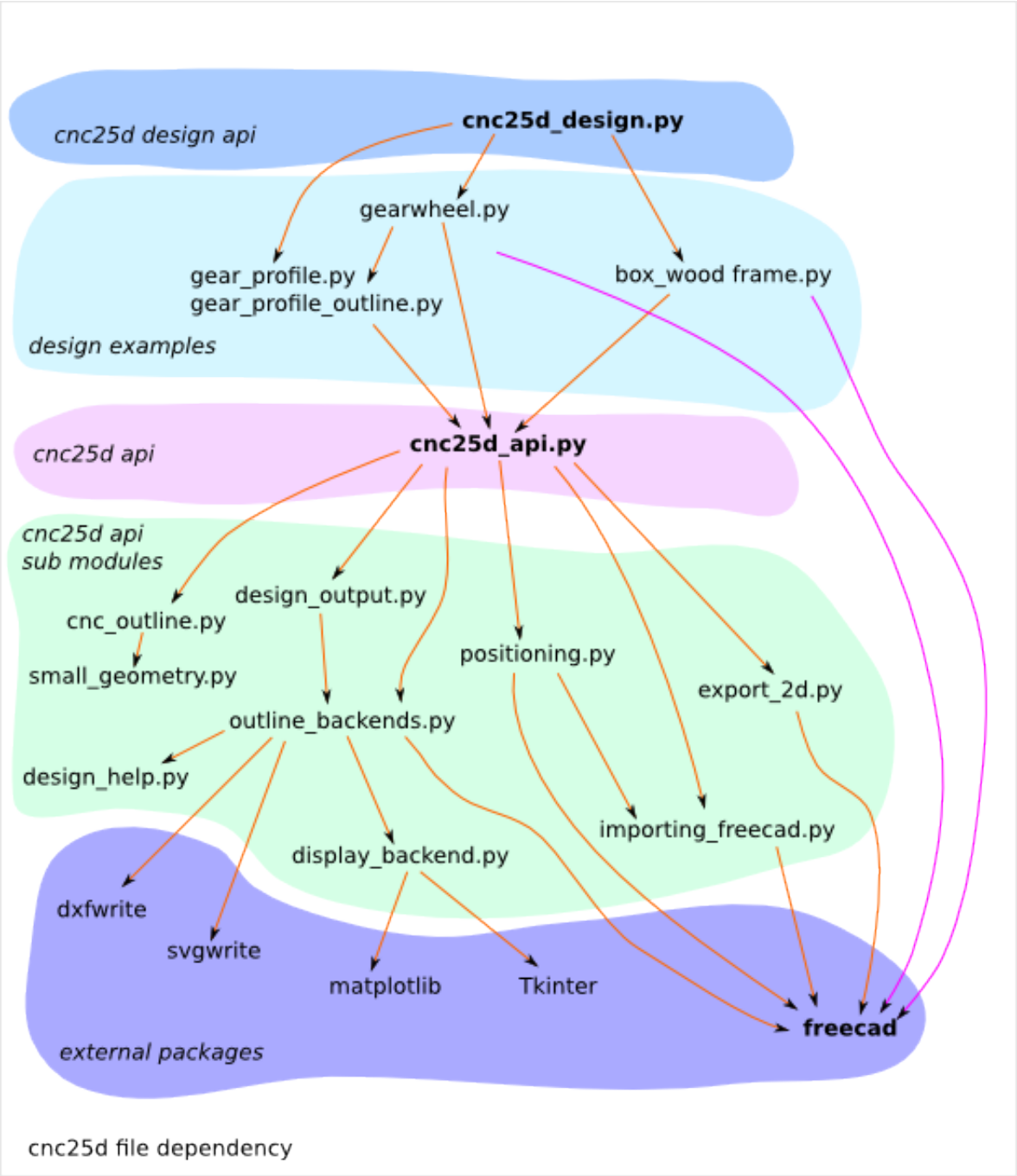
Plank positioning

CHAPTER 10

Cnc25D Internals

10.1 File layout

```
Cnc25D/
  .gitignore
  CHANGES.rst         # Release change notes. Required by PyPI
  LICENSE.txt         # Applicable license
  README.rst          # README used by GitHUb and PyPI
  setup.py            # Python package distribution setup file
  bin/                                  # contains binaries to be installed on the host system duri
    cnc25d_example_generator_src.py     # source code of the cnc25d_example_generator.py script
    cnc25d_example_generator.py         # generated by scr/micropreprocessor.py
  cnc25d/                             # the main package
    __init__.py
    importing_freecad.py              # lets import the FreeCAD libraries
    cnc_outline.py                    # cnc25d API to design parts
    export_2d.py                      # cnc25d API to export DXF or SVG
    box_wood_frame.py                 # box_wood_frame design example
    tests/                        # contains the test files of the cnc25d package
      __init__.py
      importing_cnc25d.py         # modify sys.path to import the cnc25d library
      cnc25d_api_macro.py         # usage example of the cnc25d API. Reused by cnc25d_example_generat
      box_wood_frame_macro.py     # usage example of box_wood_frame. Reused by cnc25d_example_generat
  docs/                               # cnc25d package documentation sources
    box_wood_frame.svg                # SVG draft
    box_wood_frame.txt                # text autmatically extracted from the SVG draft
    cnc25d_api.rst                    # source of the Sphinx generated documentation
    index.rst                         # top file of the Sphinx documentation sources
    conf.py                           # Sphinx configuration
    Makefile                          # make clean html to rebuild the documentation
    images/                           # contains the images used by the Sphinx documentation
      3_axis_cnc.png
  scr/                                          # additional scripts for developers
    micropreprocessor.py                        # lets generate cnc25d_example_generator.py
    note_on_cnc25d_dev.txt                      # notes for developers
```



10.2 Design example generation

The binary script *cnc25d_example_generator.py* just writes example scripts. These example scripts are actually the files *cnc25d/tests/cnc25d_api_macro.py* and *cnc25d/tests/box_wood_frame_macro.py*. The test-macro script must have those lines at the beginning of the script, so it can be excuted in the source repository as well as in the installed environment:

```
try:    # when working with an installed Cnc25D package
  from cnc25d import cnc25d_api
```

```
except: # when working on the source files
  import importing_cnc25d # give access to the cnc25d package
  from cnc25d import cnc25d_api
cnc25d_api.importing_freecad()
```

Because of the Python package workflow, the example scripts can not be copied after the installation and must be embedded in the binary script *cnc25d_example_generator.py* before the creation of the Python package distribution. This is the purpose of the script *scr/micropreprocessor.py*. The file *bin/cnc25d_example_generator_src.py* contains the skeleton of the script *bin/cnc25d_example_generator.py*. The following command include the example scripts to generate the final script *bin/cnc25d_example_generator.py*:

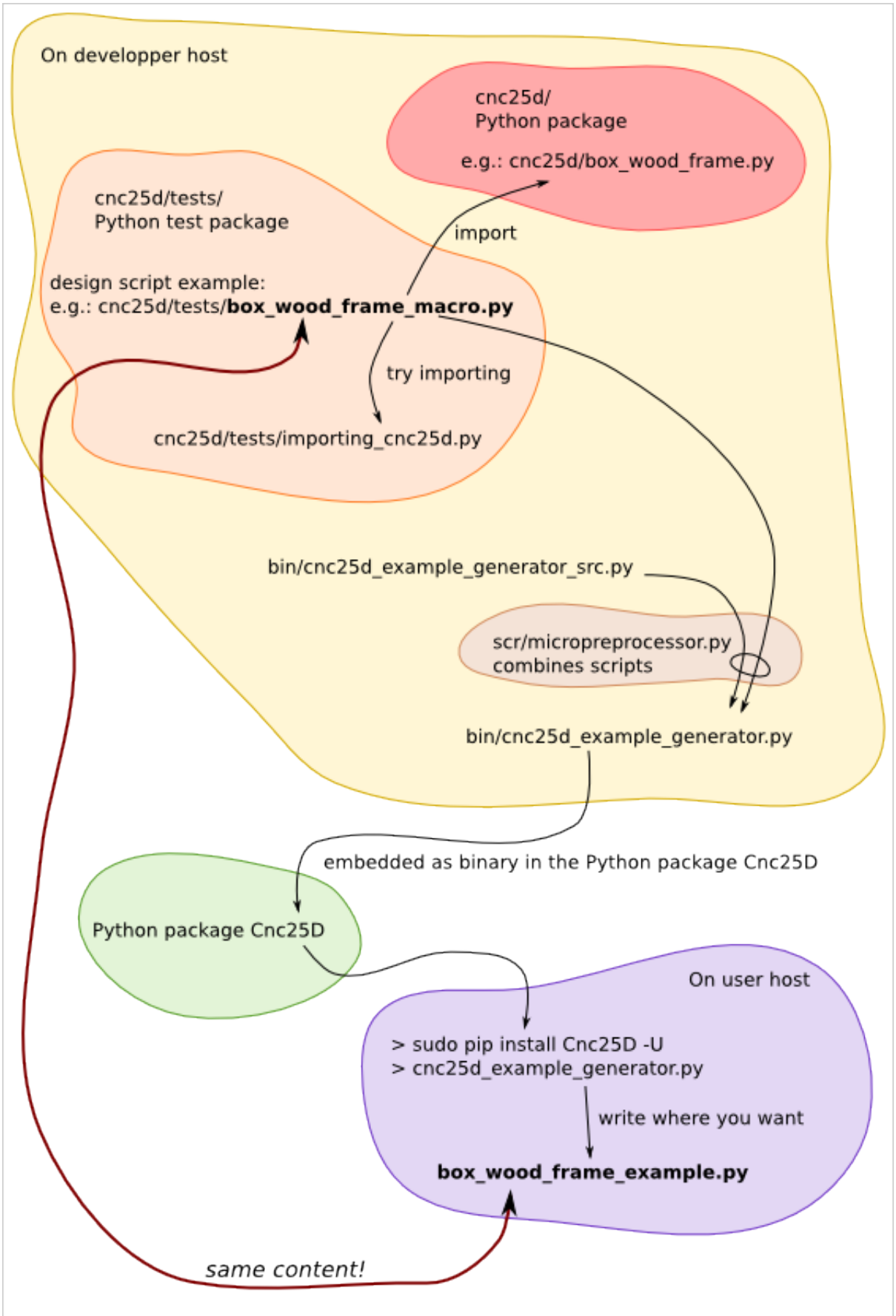
```
> scr/micropreprocessor.py bin/cnc25d_example_generator_src.py
```

The purpose of this workflow is to help the maintenance of the generated example scripts and avoid bugs in their content.

To create a new design example, follow those steps:

- Create the new design example file in the directory *Cnc25D/cnc25d/tests/* with a file name such as *my_new_design_macro.py*
- Check it by executing it
- Add the few lines in the file *Cnc25D/bin/cnc25d_example_generator_src.py* that includes the new script *Cnc25D/cnc25d/tests/my_new_design_macro.py*
- Regenerate *Cnc25D/bin/cnc25d_example_generator.py* with the command:

```
    > scr/micropreprocessor.py bin/cnc25d_example_generator_src.py
```



Design script example generation

10.3 Python package distribution release

10.4 Documentation process

SVG files are edited with Inkscape and are use as draft documents for pictures and texts. If you want to modify one of the *PNG* of the documentation, you can find the vectorial source in one of the *SVG* files. After modifying the *SVG*, save it and export the picture as *PNG* in the directory *docs/images/*.

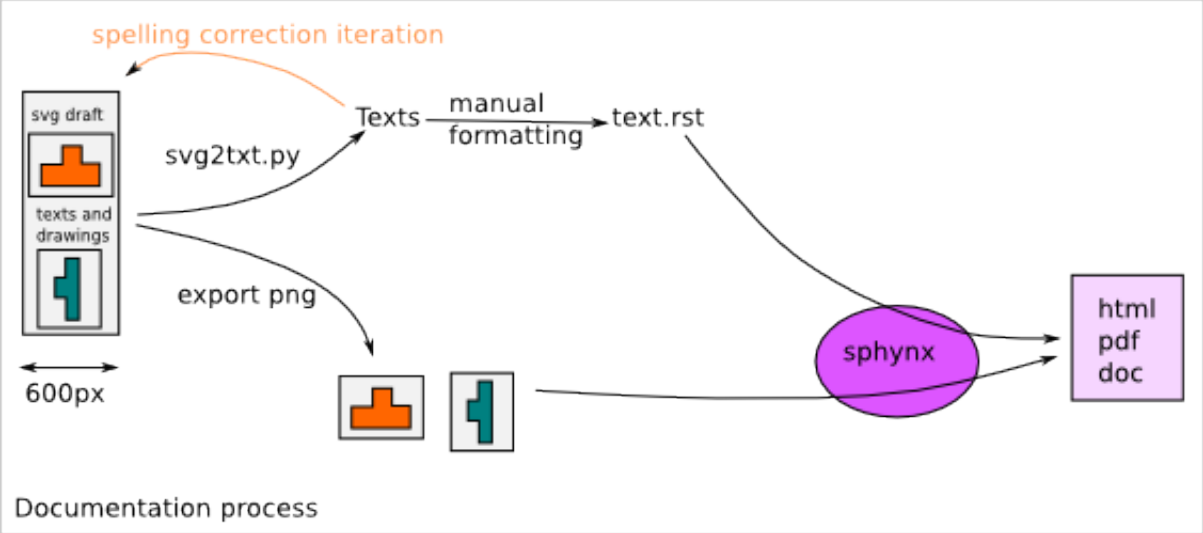
A good practice is to use a *SVG* document with a width of 600 pixels. It helps creating not too large pictures for a nice fitting in *html* and *pdf* documents. Extend the height of the *SVG* document as much as you need it.

Texts can be extracted from the *SVG* files with the command:

```
> scr/svg2txt.py docs/*.svg
```

The generated *txt* files are used for checking spelling and are raw material for the *reStructuredText* files.

The sources of the Sphinx documentation are only the *reStructuredText* files (*.rst) and the *PNG* files (*.png).



CHAPTER 11

Creating a Cnc25D Design

You can use one of the existing Cnc25D Designs or create your own *Cnc25D design* using the Cnc25D API. To create your own *Cnc25D design*, you can use your own *ad-hoc* way like in the Box Wood Frame Design variant *box_wood_frame_ng.py* or use the *recommended* way using the *class bare_design* as explained in this page.

11.1 Design Script Example

ABC is the name of our *Cnc25D design* example.

```
import cnc25d_api
cnc25d_api.importing_freecad()
import Part            # to show-up 3D in FreeCAD
import sys             # to exit on error
import argparse        # to define the ABC_design constraint
import math            # usually useful to calcule point coordinates

def ABC_constraint_constructor(parser):
  """ define the ABC constraint constructor using the argparse description
  """
  parser.add_argument('--length_A', '-a', action='store', type=float, default=10.0,
    help="set the length_A of ABC. Default: 10.0")
  parser.add_argument('--length_B', '-b', action='store', type=float, default=0.0,
    help="set the length_B of ABC. If equal 0.0, set to length_A. Default: 0.0")
  parser.add_argument('--smooth_radius', '--sr', action='store', type=float, default=0.0,
    help="set the smooth-radius of the corners of ABC. Default: 0.0")
  return(parser) # return an argparse object

def ABC_constraint_check(c):
  """ check the ABC constraint c and set the dynamic default values
  """
  # dynamic default values
  if(c['length_B']==0):
    c['length_B'] = c['length_A']
  # check the constraint
  if(c['length_B']<c['length_A']*0.1):
    print("ERR129: Error, length_B {:0.3f} is too small compare to length_A {:0.3f}".format(c['length
    sys.exit(2)
  return(c) # return a dictionary

def ABC_figures(c):
  """ construct the ABC 2D-figure-outlines at the A-format from the constraint c
      It returns a dictionary of figures with outlines in the A-format
```

```
  """
  r_figures = {}
  r_height = {}
  #
  ABC_base_figure = []
  ABC_external_outline_A = [] # the square
  ABC_external_outline_A.append((0.0,0.0, c['smooth_radius']))
  ABC_external_outline_A.append((0.0+c['length_A'], 0.0, c['smooth_radius']))
  ABC_external_outline_A.append((0.0+c['length_A'], 0.0+c['length_B'], c['smooth_radius']))
  ABC_external_outline_A.append((0.0, 0.0+c['length_B'], c['smooth_radius']))
  cnc25d_api.outline_close(ABC_external_outline_A)
  ABC_base_figure.append(ABC_external_outline_A)
  #
  r_figures['ABC_base'] = ABC_base_figure
  r_height['ABC_base'] = c['length_A']
  return((r_figures, r_height)) # return a tuple of two dictionaries

def ABC_3d(c):
  """ construct the ABC-assembly-configuration for 3D-freecad-object from the constraint c
      It returns a dictionary of assembly-configurations
  """
  r_assembly = {}
  r_slice = {}
  #
  simple_abc_assembly = []
  simple_abc_assembly.append(('ABC_base', 0.0, 0.0, c['length_A'], c['length_B'], c['length_A'], 'i',
  #
  size_xyz = (c['length_A'], c['length_B'], c['length_A'])
  zero_xyz = (0.0, 0.0, 0.0)
  slice_x = [ (i+1)/12.0*size_xyz[0] for i in range(10) ]
  slice_y = [ (i+1)/12.0*size_xyz[1] for i in range(10) ]
  slice_z = [ (i+0.1)/12.0*size_xyz[2] for i in range(10) ]
  slice_xyz = (size_xyz[0], size_xyz[1], size_xyz[2], zero_xyz[0], zero_xyz[1], zero_xyz[2], slice_z,
  #
  r_assembly['abc_assembly_conf1'] = simple_abc_assembly
  r_slice['abc_assembly_conf1'] = slice_xyz
  return((r_assembly, r_slice)) # return a tuple of two dictionaries

def ABC_info(c):
  """ create the text info related to the ABC from the constraint c
  """
  r_txt = """
length_A: \t{:0.3f}
length_B:  \t{:0.3f}
smooth_radius: \t{:0.3f}
""".format(c['length_A'], c['length_B'], c['smooth_radius'])
  return(r_txt) # return a string-text

def ABC_self_test():
  """ set the self_tests for the ABC-design
  """
  r_tests = [
    ('default abc', ''),
    ('unregular abc', '--length_A 30.0 --length_B 20.0 --smooth_radius 8.0'),
    ('heigh abc', '--length_A 5.0 --length_B 5.0 --smooth_radius 2.0 --output_file_basename test_outp
  return(r_tests) # return a list of 2-tuples

class ABC(bare_design):
```

```
  """ ABC design
  """
  def __init__(self, constraint={}):
    """ configuration of the ABC design
    """
    self.design_setup(
      s_design_name              = "ABC_design",
      f_constraint_constructor   = ABC_constraint_constructor,
      f_constraint_check         = ABC_constraint_check,
      f_2d_constructor           = ABC_figures,
      d_2d_simulation            = {},
      f_3d_constructor           = ABC_3d,
      f_3d_freecad_constructor   = None,
      f_info                     = cube_info,
      l_display_figure_list      = [],
      s_default_simulation       = "",
      l_2d_figure_file_list      = [],
      l_3d_figure_file_list      = [],
      l_3d_conf_file_list        = [],
      l_3d_freecad_file_list     = None,
      f_cli_return_type          = None,
      l_self_test_list           = ABC_self_test())
    self.apply_constraint(constraint)

if __name__ == "__main__":
  my_abc = ABC()
  my_abc.cli("--length_A 50.0 --length_B 30.0 --output_file_basename test_output/abc.dxf")
  if(cnc25d_api.interpretor_is_freecad()):
    Part.show(my_abc.get_fc_obj_3dconf('abc_assembly_conf1'))
```

11.2 Design Functions

A design is built via several mandatory and optional functions. After defining theses functions, they are bound to a design during the *design setup* phase. The name of the function is irrelevant but their argument list and their returned values are specified in this section. The argument list can be extended with optional arguments if you want to reuse this function in an other context.

11.2.1 ABC_constraint_constructor()

```
# parser = argparse.ArgumentParser()

def ABC_constraint_constructor(parser):
  parser.add_argument('--my_constraint', '--mc', action='store', type=float, default=10.0,
    help="my_constraint to parametrize the design. Default: 10.0")
  return(parser)
```

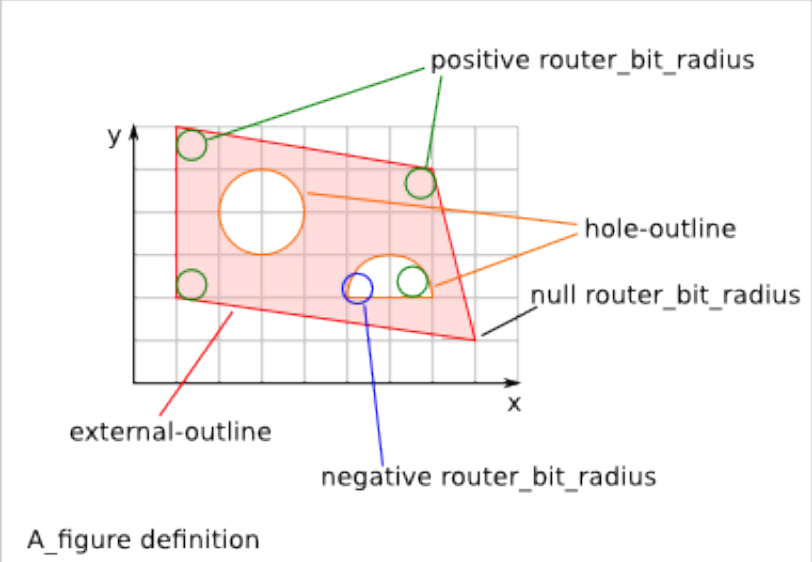
The *ABC_constraint_constructor()* function defines the constraint list of the design. Each constraint is declared with the method *argparse.ArgumentParser().add_argument()*. For one constraint, you can specigy the type (*float*, *integer*, *string* ..), the default value and some explanation. Out of this *parser argument list*, a dictionary of the *design constraint* is created using the longest name of each *ârser argument*.

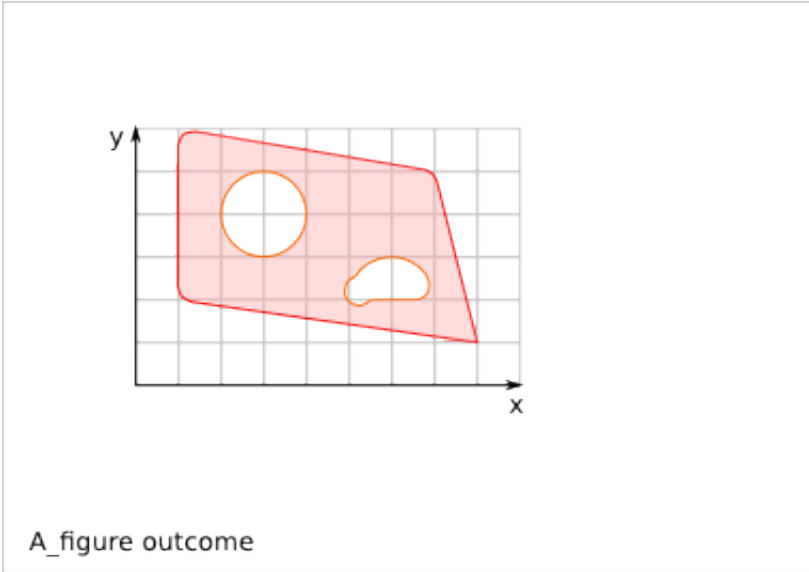
11.2.2 ABC_constraint_check()

```
# c = { 'constraint_A' : 3.0, 'constraint_B' : 3.0 }

def ABC_constraint_check(c):
  c['constraint_C'] = c['constraint_A'] + c['constraint_B']    # create a new entry in the constraint
  if(c['constraint_A']<2):  # a dummy design rule check
    print("Error: constraint_A {:0.3f} must be bigger than 2".format(c['constraint_A']))
    sys.exit(2)
  return(c)
```

The *ABC_constraint_check()* checks the coherence of the values set to the design constraint, completes the constraint dictionary with new values or even modifies the constraint values. Most of the design rule check must occur inside this function. To avoid headache to the users of you design, make sure the constraint default values pass the design rule check.

11.2.3 ABC_figures()






```
def ABC_figures(c)
  r_figures = {}
  r_height = {}
  #
  A_fig = [] # start the figure A_fig. A figure is list of outlines. The first outline is the externa
  quadrilateral = [] # start the outline quadrilateral. An outline can be a circle or a chain of line
  quadrilateral.append((10, 20, 5)) # set the first point of the quadrilateral outline. A positive ro
  quadrilateral.append((80, 10, 0)) # set a line to the second point. The router_bit_radius request
  quadrilateral.append((70, 50, 5)) # set a line to the third point. A positive router_bit_radius of
  quadrilateral.append((10, 60, 5)) # set a line to the fourth point.
  quadrilateral.append((10, 20, 0)) # set a line to the first point. The router_bit_radius request mu
  A_fig.append(quadrilateral) # the outline quadrilateral is added to the figure A_fig. quadrilatera
  hole_circle = (30, 40, 10) # define the outline hole_circle. A circle is an outline exception defi
  A_fig.append(hole_circle) # the outline hole_circle is added to the figure A_fig. hole_circle is a
  other_hole = [] # start the outline other_hole, consisting of a line and an arc.
  other_hole.append((50, 10, -5)) # set the first point of the outline other_hole. A negative router_
  other_hole.append((70, 10, 5)) # set a line to the second point. A positive router_bit_radius of 5
  other_hole.append((60, 20, 50, 10, 0)) # set an arc passing through an intermediate point and going
  A_fig.append(other_hole) # the outline other_hole is added to the figure A_fig. other_hole is a ho
  #
  r_figures['A_figure'] = A_fig
  r_height['A_figure'] = 10.0
  return((r_figures, r_height))
```

The *ABC_figures()* defines the 2D-figures of the design. As we are focusing on 2.5D designs, it is probably the heart of your design. The function must use as argument the *constraint dictionary*, that has already by processed by the previous function *ABC_constraint_check()*. The function must return a tuple of to dictionaries containing the same keys.

The first dictionary contains the 2D-figures, that are from a *Python* point of view a list of list of list.

The second dictionary contains the extrusion height of each figure. These heights are used by the function *write_figure_brep()*. For some figures, like assembly figures, the height might not make any sense. In those cases, set the height to the conventional value 1.0.

To generate the figures and outlines, you can use some function of the Cnc25D API:

- cnc25d_api.outline_shift_x(outline, x_offset, x_coefficient)
- cnc25d_api.outline_shift_y(outline, y_offset, y_coefficient)

- cnc25d_api.outline_shift_xy(outline, x_offset, x_coefficient, y_offset, y_coefficient)
- cnc25d_api.outline_rotate(outline, rotation_center_x, rotation_center_y, rotation_angle)
- cnc25d_api.outline_close(outline)
- cnc25d_api.outline_reverse(outline)
- cnc25d_api.rotate_and_translate_figure(figure, rotation_center_x, rotation_center_y, rotation_angle, translate_x, translate_y)
- cnc25d_api.flip_rotate_and_translate_figure(figure, zero_x, zero_y, size_x, size_y, x_flip, y_flip, rotation_angle, translate_x, translate_y)

For more details, read the chapter Cnc25D API Outline Creation.

11.2.4 ABC_3d()

```
def ABC_3d(c)
  r_assembly = {}
  r_slice = {}
  #
  r_assembly['A_3dconf'] = [('A_figure', 0.0, 0.0, 70, 50, 30, 'i', 'xy', 0, 0, 0)]
  r_slice['A_3dconf'] = (70, 50, 30, 10, 10, 0, [5, 15, 25], [20, 30, 40], [20, 30, 40])
  #
  return((r_assembly, r_slice))
```

The function *ABC_3d()* defines the 3D assembly generated from the extruded 2D-figures. The function must use as argument the *constraint dictionary*, that has already by processed by the previous function *ABC_constraint_check()*. The function must return a tuple of to dictionaries containing the same keys. The first dictionary contains *assembly-3D-configurations*. The second dictionary contains the *slice-configurations*.

An *assembly-3D-configurations* is a list of extruded and placed figures. Each item of the list contains:

- 2D-figure label: defined by the function ABC_figures()
- zero_x, zero_y: the reference coordinates of the 2D-figure
- size_x, size_y: the reference sizes of the 2D-figure
- size-z: the height of extrusion
- i,x,y,z-flip: the flip of the extruded part
- xy,yx,xz,zx,yz,zy-orientation: the orientation of the extruded part
- translation-xyz: the final translation

For more details, read the chapter Plank Positioning Details.

The *slice-configurations* is used by write_assembly_brep() to generate several 2D-cuts of the 3D-assembly. A *slice-configurations* is defined by:

- size-x, size-y, size-z: the reference dimension of the 3D-assembly
- zero-x, zero-y, zero-z: the reference coordinates of the 3D assembly
- slice-xy-list: the list of z-coordinates to cut the assembly in the xy-plan
- slice-xz-list: the list of y-coordinates to cut the assembly in the xz-plan
- slice-yz-list: the list of x-coordinates to cut the assembly in the yz-plan

11.2.5 ABC_3d_freecad_construction(c)

```
def A_freecad_construction(c):
  r_3dobj = Part.makeCompound()
  return(r_3dobj)

def ABC_3d_freecad_construction(c):
  r_fc_obj_f = {}
  r_slice = {}
  #
  r_fc_obj_f['A_3dobj'] = A_freecad_construction
  r_slice['A_3dobj'] = []
  ###
  return((r_fc_obj_f, r_slice))
```

The function *ABC_3d_freecad_construction()* is similar to the function *ABC_3d()* but instead of recording *assembly-3D-configurations*, it points to freecad_construction functions. These functions can access directly to the FreeCAD API. They provide more possibilities than the compact but restricted format *assembly-3D-configurations*.

The function *freecad_construction()* must use as argument the *constraint dictionary* and must return a FreeCAD object.

11.2.6 ABC_info()

```
def ABC_info(c):
  r_txt = """
  constraint_A: {:0.3f}
  """.format(c['constraint_A'])
  return(r_txt)
```

The function *ABC_info()* generates a string that is used as log during the design construction. The function must use as argument the *constraint dictionary* and must return a *string*.

11.2.7 ABC_simulations()

```
def simulation_A(c):
  print("use the cnc25d_api to test what you want")
  return(1)

def ABC_simulations():
  r_sim = {}
  r_sim['sim_A'] = simulation_A
  return(r_sim)
```

The function *ABC_simulations()* generates a dictionary containing pointers to simulation functions. The function doesn't need any argument and return the function pointer dictionary. Actually, the function could be replaced by a function pointer dictionary. For aesthetic, I prefer using a function without argument.

The simulation function must use as argument the *constraint dictionary*. The return value of this function is irrelevant.

11.2.8 ABC_self_test()

```
def ABC_self_test():
  r_tests = [
  ('test_A', '--constraint_A 7.0 --constraint_B 5.0'),
```

```
  ('test_B', '--constraint_A 3.0 --constraint_B 9.0')]
  return(r_tests)
```

The function *ABC_self_test()* generates a list of 2-tuple containing sets of constraint used to test the design in general or corner cases. The function doesn't need any argument and could be replaced by a simple list. Each item of the list is a test case. The two strings of a test-case are the *test-name* and the *constraint-values* at the *CLI (command-line-interface)* format.

11.2.9 ABC_cli_return_type()

```
def ABC_cli_return_type(c):
  return(r_cli)
```

The function *ABC_cli_return_type()* generates the value returned by the method *cli()*. The function must use as argument the *constraint dictionary*. It returns what you want the method *cli()* must return. This function is obsolete and should not be used anymore.

11.3 Design Setup

```
class ABC(bare_design):
  def __init__(self, constraint={}):
    self.design_setup( # function to setup a cnc25d design
      s_design_name              = "ABC_design", # mandatory string, used to enhance information and e
      f_constraint_constructor   = ABC_constraint_constructor, # mandatory function, set the design co
      f_constraint_check         = ABC_constraint_check, # highly recommended function to check the de
      f_2d_constructor           = ABC_figures, # function that generates a dictionary that contains 2
      d_2d_simulation            = ABC_simulations(), # dictionary to functions running simulations
      f_3d_constructor           = ABC_3d, # function that generates a dictionary that contains 3D-ass
      f_3d_freecad_constructor   = ABC_3d_freecad_construction, # function that generates a dictionary
      f_info                     = ABC_info, # function that generates a string
      l_display_figure_list      = [], # list of the 2D-figures to be displayed in a Tk-window
      s_default_simulation       = "", # simulation string name, set the default action to simulation
      l_2d_figure_file_list      = [], # 2D-figures to be written in SVG or DXF files
      l_3d_figure_file_list      = [], # 2D-figures to be written in Brep files
      l_3d_conf_file_list        = [], # 3D-assembly-configurations to be written in Brep files
      l_3d_freecad_file_list     = [], # 3D-freecad-construction to be written in Brep files
      f_cli_return_type          = [], # obsolete function that defines the return value of the metho
      l_self_test_list           = ABC_self_test()) # list of tests to be run to check the design
    self.apply_constraint(constraint) # optional but quiet convenient
```

If you don't want to use one or several settings, set them to *None* or comment the line. Concerning the list, usually an empty list means all available 2D-figures or 3D-assembly. *None* means nothing.

11.4 Design Usage

```
my_abc = ABC(ABC_constraint)
my_abc.outline_display() # display the 2D-figures of the list l_display_figure_list in Tk-windows
my_abc.write_figure_svg("test_output/abc_macro") # write in SVG files the 2D-figures of the list l_2d
my_abc.write_figure_dxf("test_output/abc_macro") # write in DXF files the 2D-figures of the list l_2d
my_abc.write_figure_brep("test_output/abc_macro") # write in Brep files the extruded 2D-figures of th
my_abc.write_assembly_brep("test_output/abc_macro") # write in Brep files the 3D-assembly of the list
my_abc.write_freecad_brep("test_output/abc_macro") # write in Brep files the 3D-assembly of the list
```

```
#my_abc.run_simulation("sim_A") # run the simulation
my_abc.view_design_configuration() # display information of the design setup. Useful when you want to
my_abc.run_self_test("") # run the test case of the list l_self_test_list
my_abc.cli("--output_file_basename test_output/my_abc.dxf") # Warning: all constraint values are rese

if(cnc25d_api.interpretor_is_freecad()): # check if the interpretor is freecad
  Part.show(my_abc.get_fc_obj_3dconf('A_3dconf')) # display the 3D object corresponding to the 3D-ass

my_fig = my_abc.get_A_figure('A_figure') # get the figure A_figure at the A-format
my_fig = my_abc.get_B_figure('A_figure') # get the figure A_figure at the B-format
my_fc_obj = my_abc.get_fc_obj_3dconf('A_3dconf') # get the FreeCAD object generated by the 3D-assembl
my_fc_obj = my_abc.get_fc_obj_function('A_3dobj') # get the FreeCAD object generated by the 3D-freeca
my_txt = my_abc.info() # get text information about the design ABC
my_constraint = my_abc.get_constraint() # get a dictionary containing all set and internal constraint
my_abc.apply_constraint(my_constraint) # change the constraint of the ABC design my_abc with checking
my_abc.apply_external_constraint(my_constraint) # change the constraint of the ABC design my_abc with
```

11.5 Internal Methods

The internal methods can be used in some advanced cases.

```
(figs, heights) = my_abc.apply_2d_constructor() # generates and returns the 2D-figures according to t
(assembly_3dconfs, slice_confs) = my_abc.apply_3d_constructor() # generates and returns the 3D-assemb
(freecad_function_pts, slice_confs) = my_abc.apply_3d_freecad_constructor()  # generates and returns

my_abc.set_design_name(s_design_name) # overwrite the design name
my_abc.set_constraint_constructor(f_constraint_constructor) # overwrite the function that defines the
my_abc.set_constraint_check(f_constraint_check) # overwrite the function that checks the design const
my_abc.set_2d_constructor(f_2d_constructor) # overwrite the function that generates the 2D-figures
my_abc.set_2d_simulation(d_2d_simulation) # overwrite the dictionary that points to the simulation fu
my_abc.set_3d_constructor(f_3d_constructor) # overwrite the function that generates the 3D-assembly-c
my_abc.set_3d_freecad_constructor(f_3d_freecad_constructor) # overwrite the function that points to t
my_abc.set_info(f_info) # overwrite the function that generates the information string
my_abc.set_display_figure_list(l_display_figure_list) # overwrite the list of the displayed 2D-figure
my_abc.set_default_simulation(s_default_simulation) # overwrite the default action as simulation. If
my_abc.set_2d_figure_file_list(l_2d_figure_file_list) # overwrite the list of the 2D-figures to be wr
my_abc.set_3d_figure_file_list(l_3d_figure_file_list) # overwrite the list of the 2D-figures to be wr
my_abc.set_3d_conf_file_list(l_3d_conf_file_list) # overwrite the list of the 3D-assembly-configurati
my_abc.set_3d_freecad_file_list(l_3d_freecad_file_list) # overwrite the list of the 3D-freecad-functi
my_abc.set_cli_return_type(f_cli_return_type) # overwrite the function to generate the return value o
my_abc.set_self_test(l_self_test_list) # overwrite the list of tests
```

CHAPTER 12

Cnc25D Designs

12.1 Cnc25D design introduction

In addition to the Cnc25D API functions, the *Cnc25D Python package* includes also several parametric designs. The design parameters are called *constraints* and are set via a dictionary. Most of the constraints are not mandatory and if you don't set some constraints, their default values are used. Use the files provided by the *cnc25d_example_generator.py* as template to generate one of the *Cnc25D designs*. Depending on the *constraints output_file_basename* and *return_type*, you can generate *.dxf*, *.svg* or *.brep* files or include the *Cnc25D Design- as *Part-object* in your FreeCAD macro. For more information about *how to use the Cnc25D designs* read the section Cnc25D Design Details.

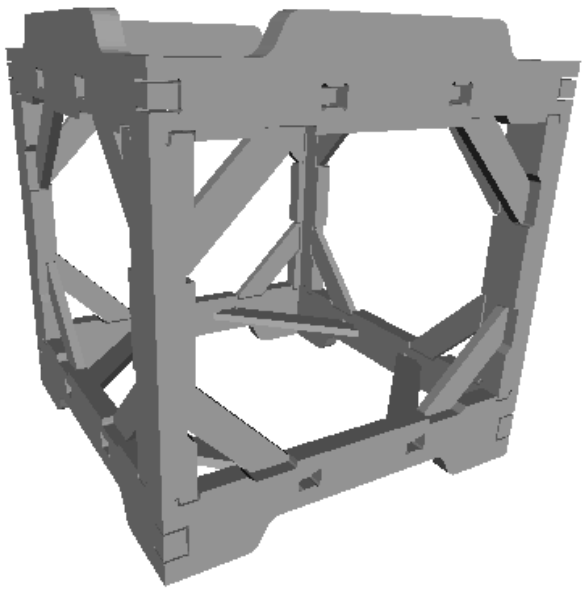
12.2 Cnc25D design list

- Box Wood Frame Design
- Gear Profile Function
- Gearwheel Design
- Gearring Design
- Gearbar Design
- Split-gearwheel Design
- Epicyclic Gearing Design
- Axle Lid Design
- Motor Lid Design
- Bell Design
- Bagel Design
- Bell Bagel Assembly
- Crest Design
- Cross_Cube Design
- Gimbal Design

12.3 Cnc25D design overview

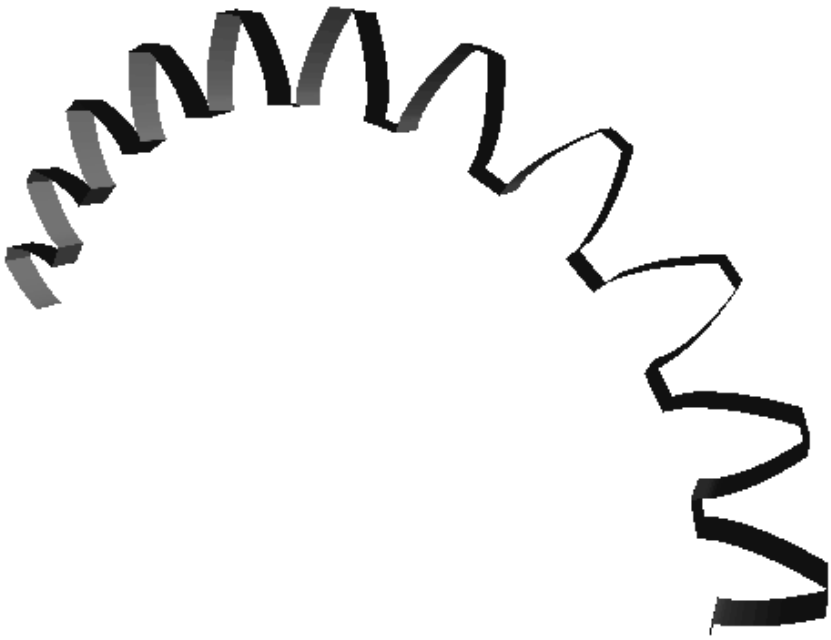
12.3.1 Box_wood_frame

The Box Wood Frame Design is a piece of furniture. Its particularity is that its top-shape and its bottom-shape are complementary. So, you can pile-up your boxes.



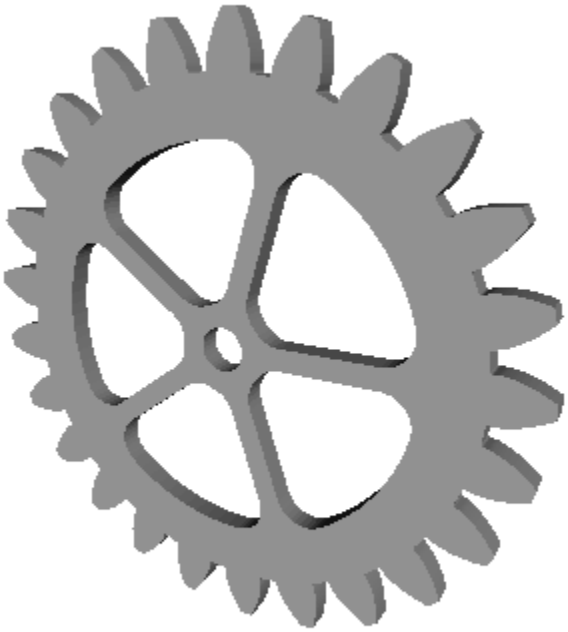
12.3.2 Gear_profile

The Gear Profile Function generates the gear-profile outline. You can also simulate this outline with a second gear-profile to make sure it works as you wish it. The gear-profile itself is not a 3D part but a simple outline. You can use this outline to create a complete 3D part.



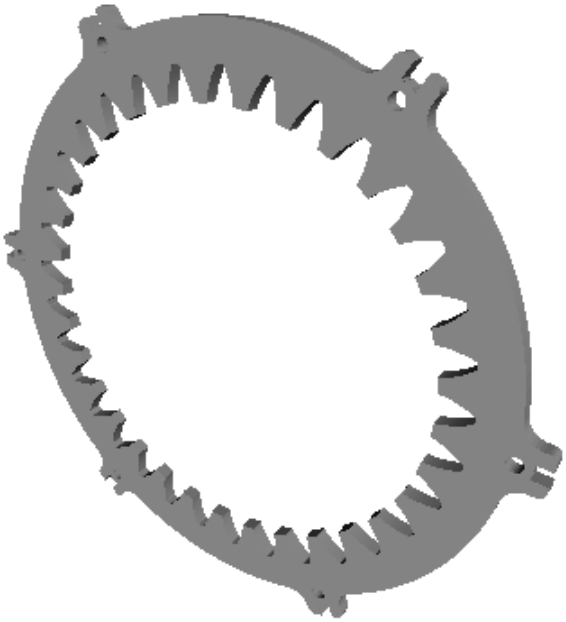
12.3.3 Gearwheel

The Gearwheel Design is a complete gearwheel part (a.k.a. spur). You can specify the number of gear-teeth, the number of legs, the size of the axle and much more.



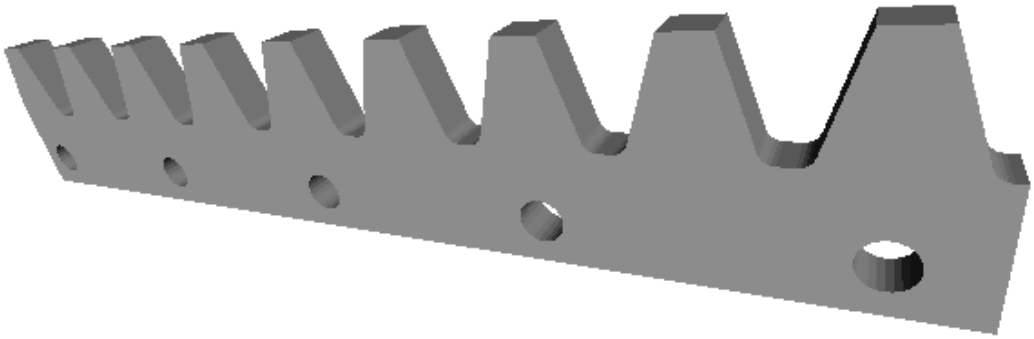
12.3.4 Gearring

The Gearring Design is a complete gearring part (a.k.a. annulus). You can use it to create your epicyclic gear system.



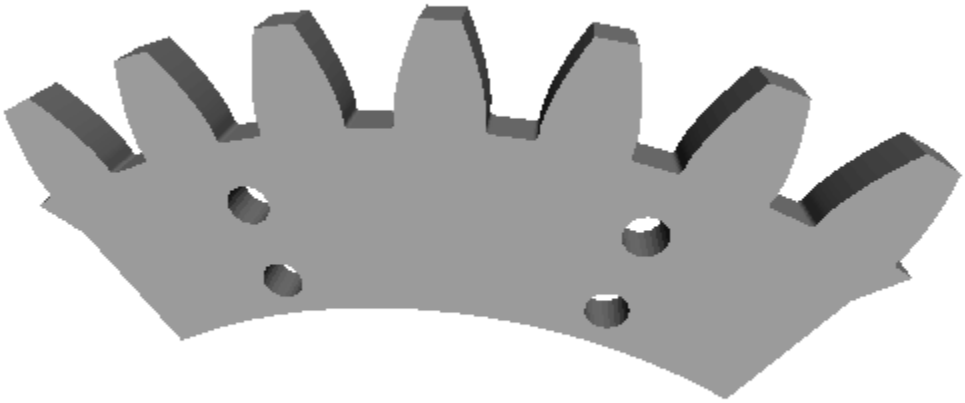
12.3.5 Gearbar

The Gearbar Design is a complete rack part.



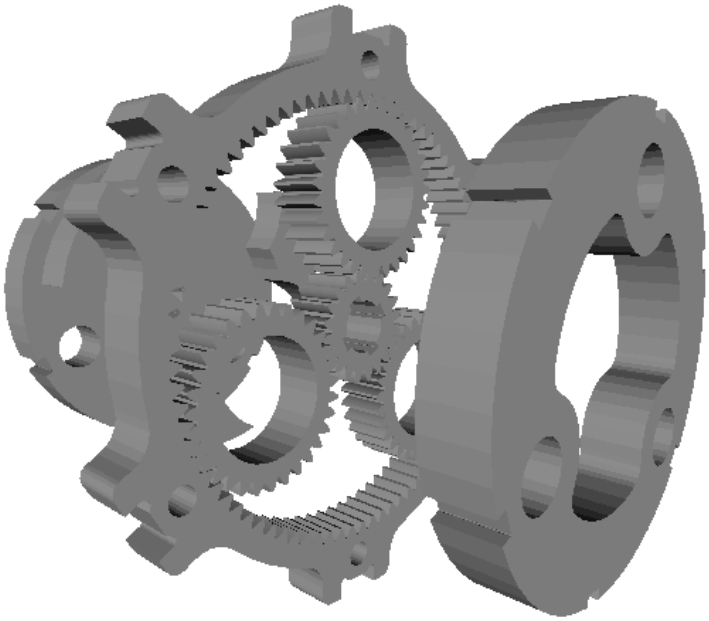
12.3.6 Split_gearwheel

The Split-gearwheel Design generates several 3D parts that can be assembled to create a complete gearwheel. The split gearwheel lets you make large gearwheel by making smaller sub parts and then assembling them.



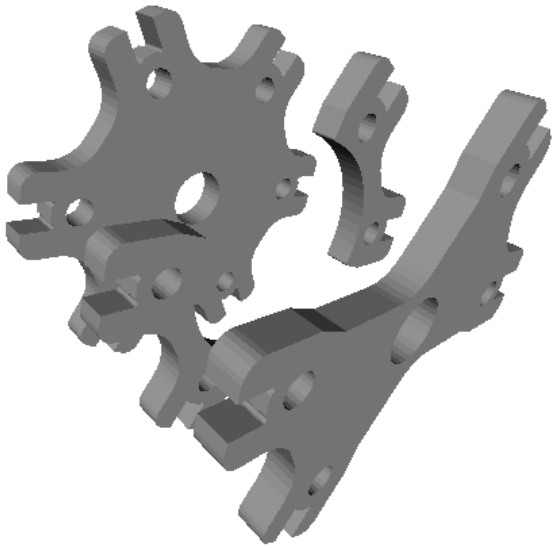
12.3.7 Epicyclic_gearing

The Epicyclic Gearing Design is a complete epicyclic gearing system. You can use it to increase the torque (and decreasing the rotation speed).



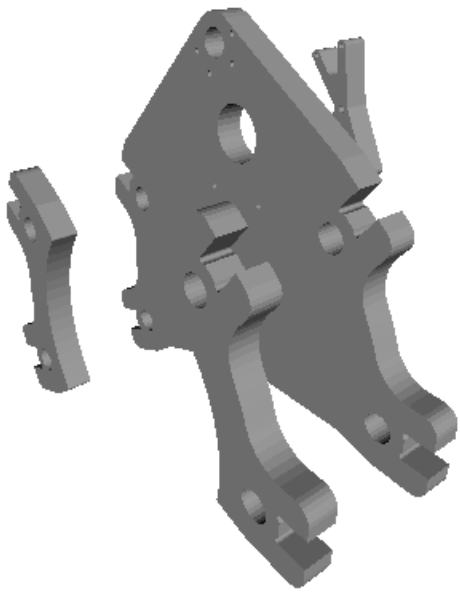
12.3.8 Axle_lid

The Axle Lid Design is a axle-lid design kit. You can use it to complete the epicyclic_gearing design.



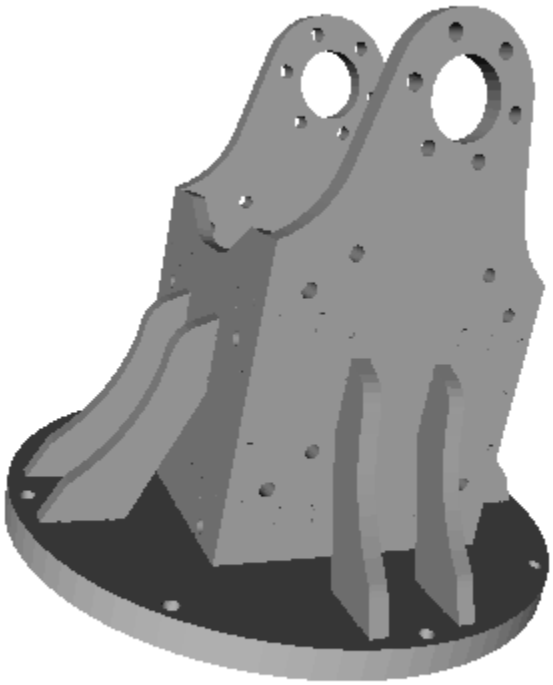
12.3.9 Motor_lid

The Motor Lid Design is an extension of the axle-lid design kit to mount an electrical motor. You can use it to complete the epicyclic_gearing design.



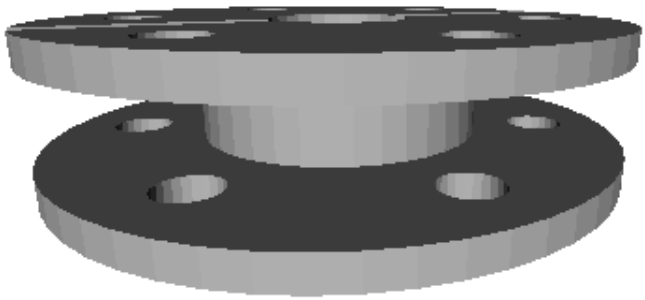
12.3.10 Bell

The Bell Design is the extremity of a *gimbal* system. You can complete is with a *bagel* and a *cross_cube* to get a complete *gimbal* system.



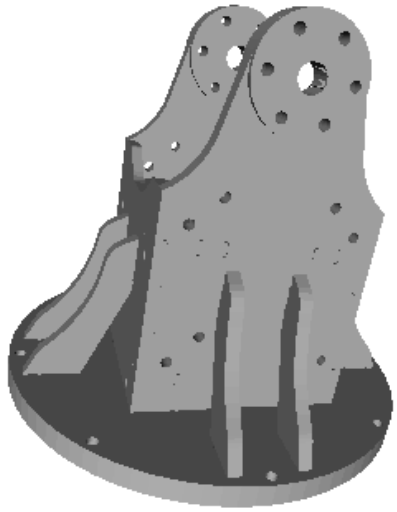
12.3.11 Bagel

The Bagel Design is the axle-guidance of the *bell* piece.



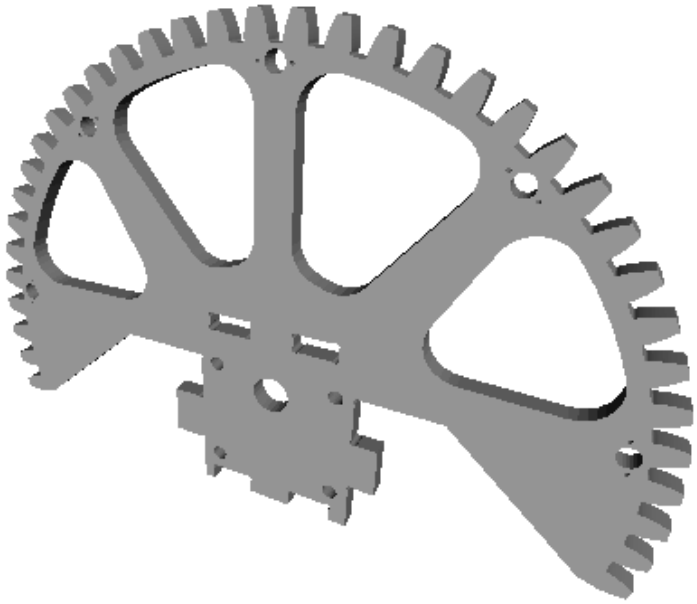
12.3.12 Bell_bagel_assembly

The Bell Bagel Assembly is the assembly of a *bell* piece and two *bagels*.



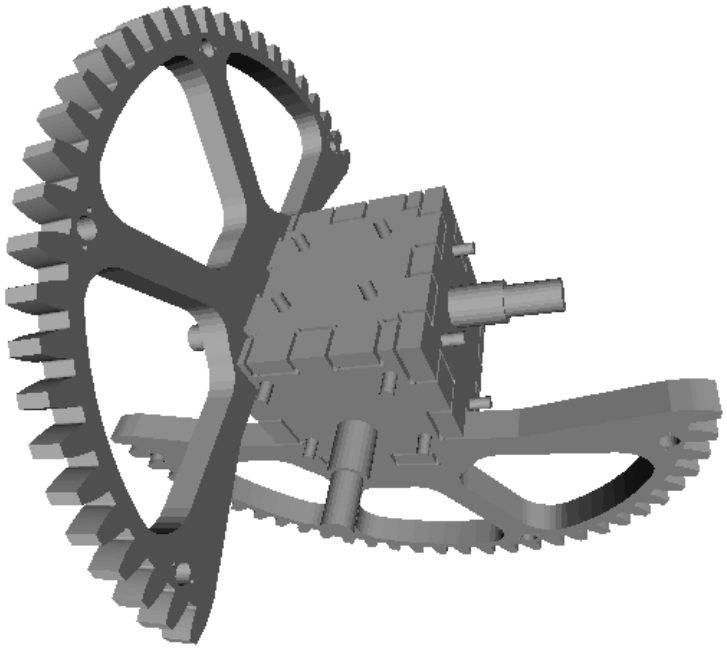
12.3.13 Crest

The Crest Design is an optional part for the *cross_cube* piece.



12.3.14 Cross_cube

The Cross_Cube Design is the *two-axle-join* of a *gimbal* system.



12.3.15 Gimbal

The Gimbal Design is a mechanism with two angles as degree of freedom.

CHAPTER 13

Cnc25D Design Details

13.1 Cnc25D design usage

13.1.1 From the source repository

Using the design module

Go to the *Cnc25D source repository* and execute the *design script* with or without arguments:

```
> cd Cnc25D
> python cnc25D/XYZDesign.py
```

or:

```
> python cnc25D/XYZDesign.py --param_A 50.0 --param_C 30.0
```

Without arguments, the default command line is used.

When you don't use argument, you can also use *freecad* instead of *python*

```
> freecad cnc25D/XYZDesign.py
```

With *freecad*, you can not choose the arguments on the command line because of the conflict with the freecad argument parser. So you have to change the default command line at the end of the design script:

```
if __name__ == "__main__":
  FreeCAD.Console.PrintMessage("XYZDesign.py says hello!\n")
  my_xyz = XYZDesign_cli("--param_A 6.0 --param_B 13.0 --return_type freecad_object".split())  # defau
  try: # depending on xyz_c['return_type'] it might be or not a freecad_object
    Part.show(my_xyz)
    print("freecad_object returned")
  except:
    pass
    #print("return_type is not a freecad-object")
```

The argument *–return_type freecad_object* lets you visualizing the result in FreeCAD.

Using the test-macro

Go to the *Cnc25D source repository* and execute the *test-macro* without argument:

```
> cd Cnc25D
> python cnc25D/tests/XYZDesign_macro.py
```

or:

```
> freecad cnc25D/tests/XYZDesign_macro.py
```

You can use those test-macro scripts as *FreeCAD macro* and run them from the FreeCAD GUI. Make sure the *test-macro script* returns a *freecad_object*:

```
xyz_x['return_type'] = 'freecad_object'
```

13.1.2 From the installed Cnc25D package

After installing the *Cnc25D Python package*, run *cnc25d_example_generator.py* to get the *Cnc25D example scripts*. These *Cnc25D example scripts* are actually a copy of the previous *test-macros*. You can execute them without argument with *python* or *freecad*:

```
> cd where/I/have/generated/the/Cnc25D/example/scripts
> python eg05_XYZDesign_example.py
```

or:

```
> freecad eg05_XYZDesign_example.py
```

Like with the *test-macro script*, make sure the script returns a *freecad_object*. If not, edit your script and set the following constraint:

```
xyz_x['return_type'] = 'freecad_object'
```

Your script can also be used as a *FreeCAD macro* and can be called from the *FreeCAD GUI*.

13.2 Cnc25D design implementation structure

Template of a Cnc25D design script:

```
################################################################
# import
################################################################

import cnc25d_api
cnc25d_api.importing_freecad()
import math
import sys, argparse
import Part

################################################################
# XYZDesign dictionary-constraint-arguments default values
################################################################

def XYZDesign_dictionary_init():
  """ create and initiate a XYZDesign_dictionary with the default value
  """
  r_xyzd = {}
  r_xyzd['param_A'] = 5.0
  r_xyzd['param_B'] = 10.0
```

```
  r_xyzd['param_C'] = 0.0
  r_xyzd['return_type'] = 'int_status' # possible values: 'int_status', 'cnc25d_figure', 'freecad_ob
  # ...
  return(r_xyzd)

################################################################
# XYZDesign argparse
################################################################

def XYZDesign_add_argument(ai_parser):
  """
  Add arguments relative to the XYZDesign
  This function intends to be used by the XYZDesign_cli and XYZDesign_self_test
  """
  r_parser = ai_parser
  r_parser.add_argument('--param_A','--pa', action='store', type=float, default=5.0, dest='sw_param_A
    help="Set the param_A. Default: 5.0")
  r_parser.add_argument('--param_B','--pb', action='store', type=float, default=10.0, dest='sw_param_
    help="Set the param_B. Default: 10.0")
  r_parser.add_argument('--param_C','--pc', action='store', type=float, default=0.0, dest='sw_param_C
    help="Set the param_C. If equal to 0.0, the default value is computed. Default: 0.0")
  # ...
  return(r_parser)

################################################################
# the most important function to be used in other scripts
################################################################

def XYZDesign(ai_constraints):
  """
  The main function of the script.
  It generates a XYZDesign according to the constraint-arguments
  """
  ### check the dictionary-arguments ai_constraints
  xyzdi = XYZDesign_dictionary_init()
  xyz_c = xyzdi.copy()
  xyz_c.update(ai_constraints)
  if(len(xyz_c.viewkeys() & xyzdi.viewkeys()) != len(xyz_c.viewkeys() | xyzdi.viewkeys())): # check
    print("ERR157: Error, xyz_c has too much entries as {:s} or missing entries as {:s}".format(xyz_c
    sys.exit(2)
  ### dynamic default value
  if(ai_constraints['param_C']==0):
    xyz_c['param_C'] = xyz_c['param_B']/5

  ### generate the XYZDesign figure
  # ...

  # display with Tkinter
  if(xyz_c['tkinter_view']):
    print(XYZDesign_parameter_info)
    cnc25d_api.figure_simple_display(xyz_figure, xyz_figure_overlay, XYZDesign_parameter_info)
  # generate output file
  cnc25d_api.generate_output_file(xyz_figure, xyz_c['output_file_basename'], xyz_c['XYZDesign_height

  #### return
  if(xyz_c['return_type']=='int_status'):
    r_xyz = 1
  elif(xyz_c['return_type']=='cnc25d_figure'):
```

```
    r_xyz = xyz_figure
  elif(xyz_c['return_type']=='freecad_object'):
    r_xyz = cnc25d_api.figure_to_freecad_25d_part(xyz_figure, xyz_c['XYZDesign_height'])
  else:
    print("ERR508: Error the return_type {:s} is unknown".format(xyz_c['return_type']))
    sys.exit(2)
  return(r_xyz)

################################################################
# XYZDesign wrapper dance
################################################################

def XYZDesign_argparse_to_dictionary(ai_xyz_args):
  """ convert a XYZDesign_argparse into a XYZDesign_dictionary
  """
  r_xyzd = {}
  r_xyzd['param_A']  = ai_xyz_args.sw_param_A
  r_xyzd['param_B']  = ai_xyz_args.sw_param_B
  r_xyzd['param_C']  = ai_xyz_args.sw_param_c
  #### return
  return(r_xyzd)

def XYZDesign_argparse_wrapper(ai_xyz_args, ai_args_in_txt=""):
  """
  wrapper function of XYZDesign() to call it using the XYZDesign_parser.
  XYZDesign_parser is mostly used for debug and non-regression tests.
  """
  # view the XYZDesign with Tkinter as default action
  tkinter_view = True
  if(ai_xyz_args.sw_simulation_enable or (ai_xyz_args.sw_output_file_basename!='')):
    tkinter_view = False
  # wrapper
  xyzd = XYZDesign_argparse_to_dictionary(ai_xyz_args)
  xyzd['args_in_txt'] = ai_args_in_txt
  xyzd['tkinter_view'] = tkinter_view
  #xyzd['return_type'] = 'int_status'
  r_xyz = XYZDesign(xyzd)
  return(r_xyz)

################################################################
# self test
################################################################

def XYZDesign_self_test():
  """
  This is the non-regression test of XYZDesign.
  """
  test_case_switch = [
    ["Test_A"           , "--param_A 20.0"],
    ["Test B"           , "--param_B 15.0 --param_C 5.0"],
    ["Advanced Test C"  , "--param_A 10.0 --param_B 8.0 --param_C 15.0"]]
  #print("dbg741: len(test_case_switch):", len(test_case_switch))
  XYZDesign_parser = argparse.ArgumentParser(description='Command line interface for the function XYZ
  XYZDesign_parser = XYZDesign_add_argument(XYZDesign_parser)
  XYZDesign_parser = cnc25d_api.generate_output_file_add_argument(XYZDesign_parser, 1)
  for i in range(len(test_case_switch)):
    l_test_switch = test_case_switch[i][1]
    print("{:2d} test case: '{:s}'\nwith switch: {:s}".format(i, test_case_switch[i][0], l_test_switc
```

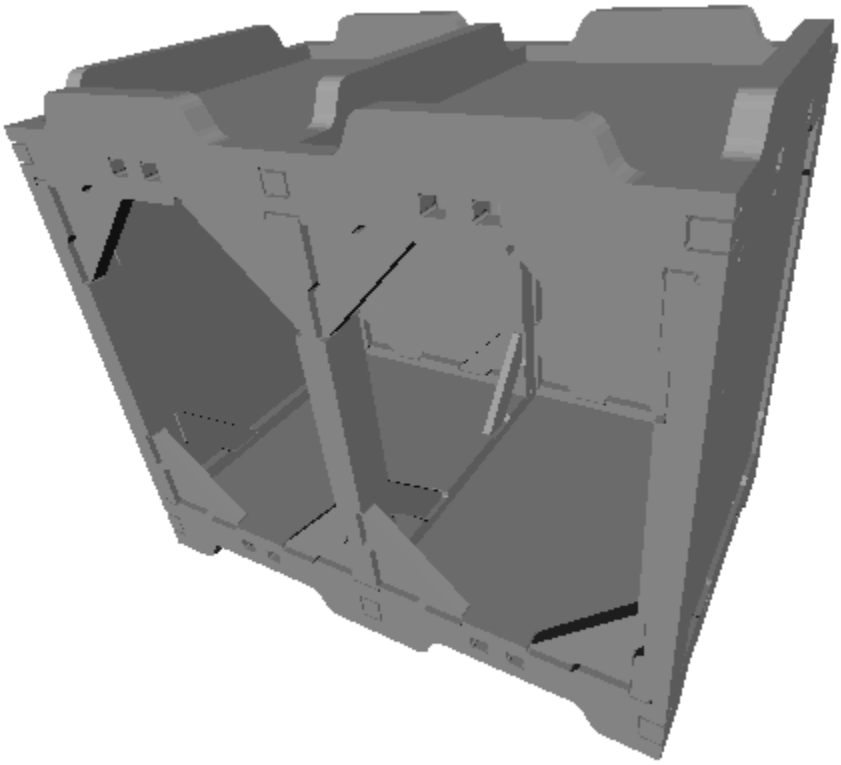
```
    l_args = l_test_switch.split()
    #print("dbg414: l_args:", l_args)
    st_args = XYZDesign_parser.parse_args(l_args)
    r_xyzst = XYZDesign_argparse_wrapper(st_args)
  return(r_xyzst)

################################################################
# XYZDesign command line interface
################################################################

def XYZDesign_cli(ai_args=None):
  """ command line interface of XYZDesign.py when it is used in standalone
  """
  # XYZDesign parser
  XYZDesign_parser = argparse.ArgumentParser(description='Command line interface for the function XYZ
  XYZDesign_parser = XYZDesign_add_argument(XYZDesign_parser)
  XYZDesign_parser = cnc25d_api.generate_output_file_add_argument(XYZDesign_parser, 1)
  # switch for self_test
  XYZDesign_parser.add_argument('--run_test_enable','--rst', action='store_true', default=False, dest
  help='Generate several corner cases of parameter sets.')
  effective_args = cnc25d_api.get_effective_args(ai_args)
  effective_args_in_txt = "XYZDesign arguments: " + ' '.join(effective_args)
  xyz_args = XYZDesign_parser.parse_args(effective_args)
  print("dbg111: start making XYZDesign")
  if(xyz_args.sw_run_self_test):
    r_xyz = XYZDesign_self_test()
  else:
    r_xyz = XYZDesign_argparse_wrapper(xyz_args, effective_args_in_txt)
  print("dbg999: end of script")
  return(r_xyz)

################################################################
# main
################################################################

if __name__ == "__main__":
  FreeCAD.Console.PrintMessage("XYZDesign.py says hello!\n")
  my_xyz = XYZDesign_cli("--param_A 6.0 --param_B 13.0".split())
  try: # depending on xyz_c['return_type'] it might be or not a freecad_object
    Part.show(my_xyz)
    print("freecad_object returned")
  except:
    pass
    #print("return_type is not a freecad-object")
```

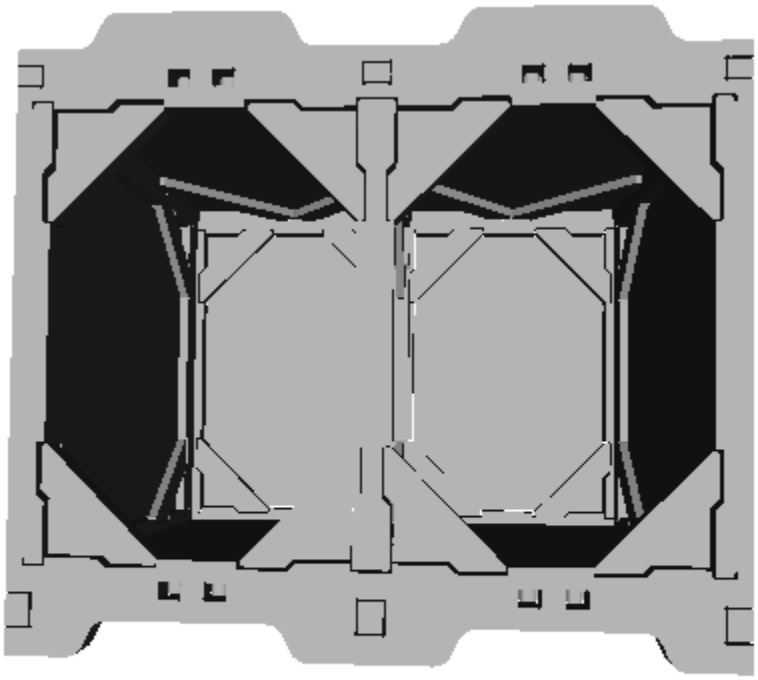
CHAPTER 14

Box Wood Frame Design

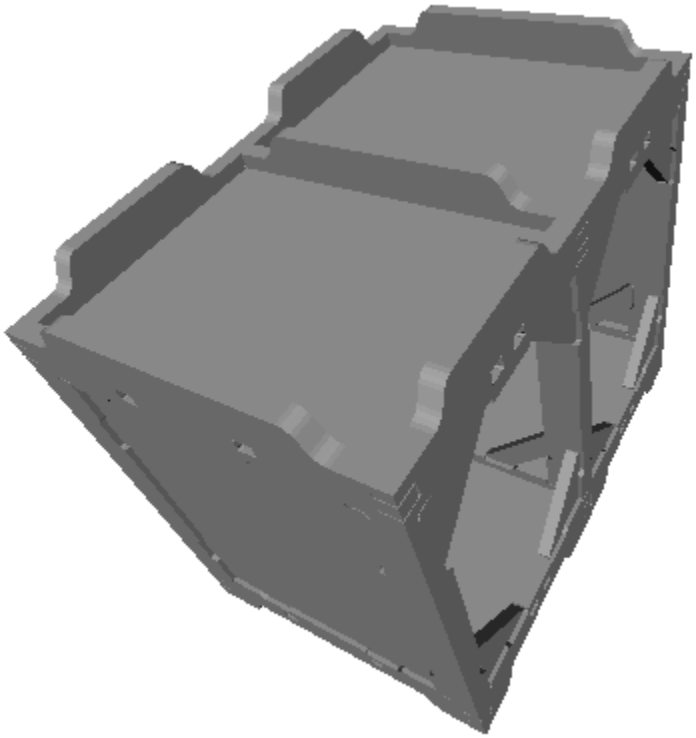
14.1 Box wood frame presentation

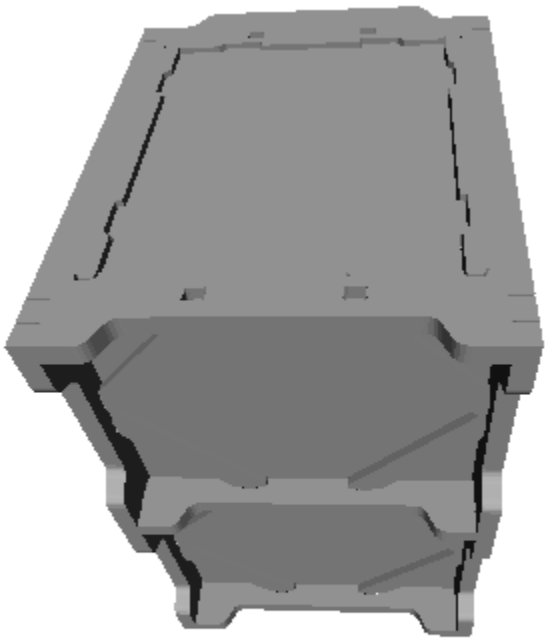
Box wood frame is the name of this piece of furniture:





Its main characteristic is its top and bottom fittings that lets pile-up a *Box wood frame* over an other:





This *pile-up* functionality has several goals:

- split the manufacturing of large wardrobe into several small modules
- make easier the move of furniture
- be part of the structure of *straw houses*.

The *Box wood frame* design uses complex and precise recessed fittings to assemble the planks. So the cuts of the planks must be done with a CNC or with a manual wood router and templates. Then the planks can be glued together.

14.2 Box wood frame creation

After installing FreeCAD and the Python package Cnc25D as described at the paragraph *Cnc25D Installation*, run the executable *cnc25d_example_generator.py* in the directory where you want to create the *Box wood frame*:

```
> cd /directory/where/I/want/to/create/a/box/wood/frame/
> cnc25d_example_generator.py # answer 'y' or 'yes' when it asks you to generate the example  box_woo
> python box_wood_frame_example.py
```

After several minutes of computation, you get plenty of DXF and STL files that let you manufacture a *Box wood frame*. Read the *text_report.txt* file to get further information on your generated *Box wood frame* and on the descriptions of the other generated files. Use LibreCAD to view the DXF files. Use MeshLAB to view the STL files:

```
> librecad bwf37_assembly_with_amplified_cut.dxf
> meshlab     # import bwf36_assembly_with_amplified_cut.stl
> less bwf49_text_report.txt
```

Your *Box wood frame* has been generated with the default parameters. You may want to changes these parameter values to adapt them to your need. Edit the file *box_wood_frame_example.py*, change some parameters values, save your changes and run again:

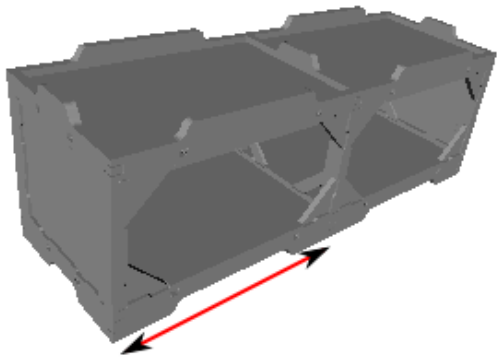
```
> python box_wood_frame_example.py
```

Now you get the *Box wood frame* design files according to your parameters.

14.3 Box wood frame parameters

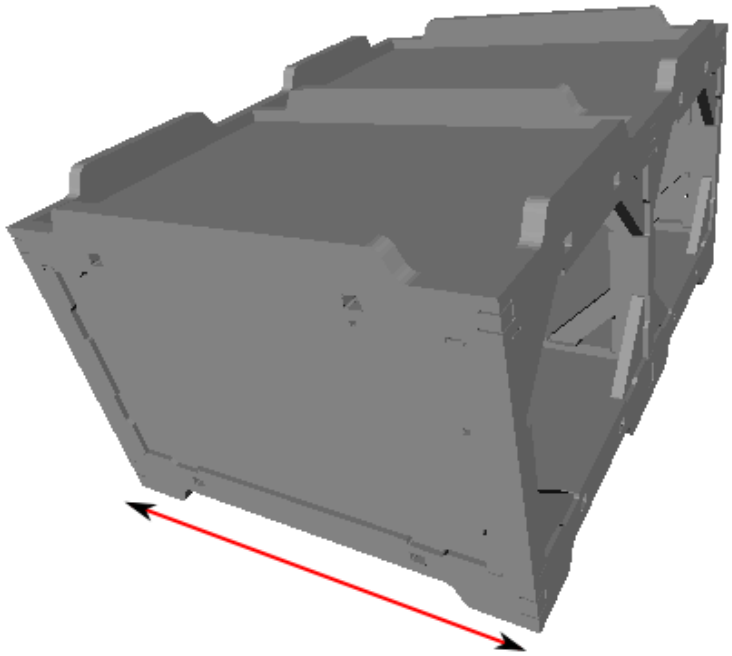
14.3.1 bwf_box_width

bwf_box_width default value : 400.0



14.3.2 bwf_box_depth

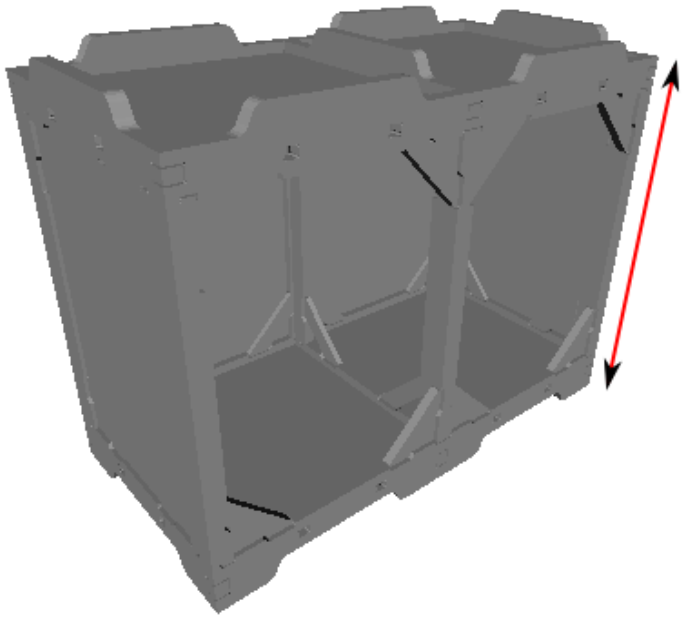
bwf_box_depth default value : 400.0



recommendation: Keep bwf_box_depth = bwf_box_width to get more pile up possibilities.

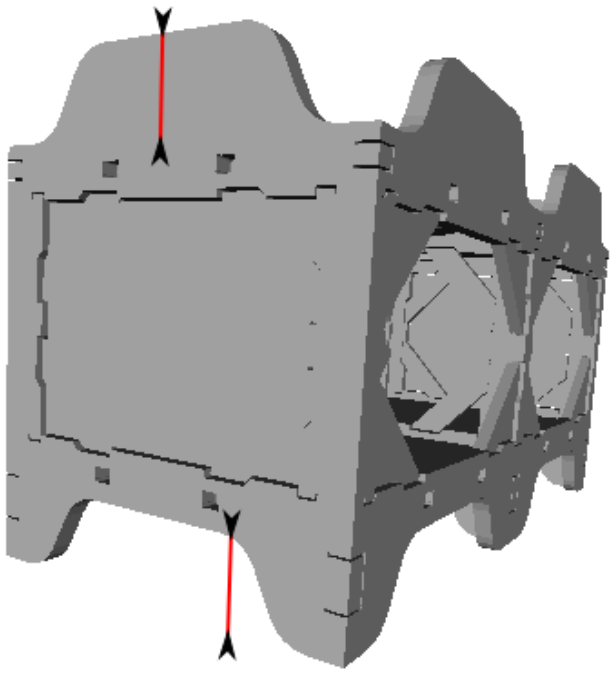
14.3.3 bwf_box_height

bwf_box_height default value : 400.0



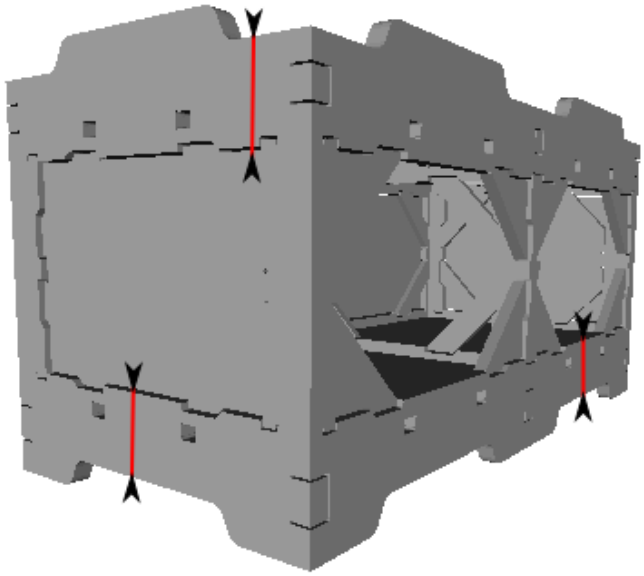
14.3.4 bwf_fitting_height

bwf_fitting_height default value : 30.0



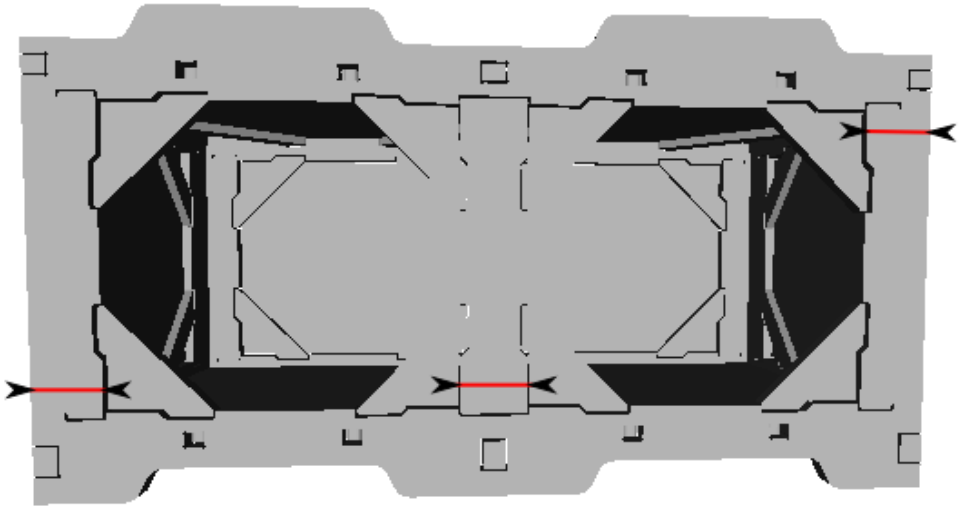
14.3.5 bwf_h_plank_width

bwf_h_plank_width default value : 50.0



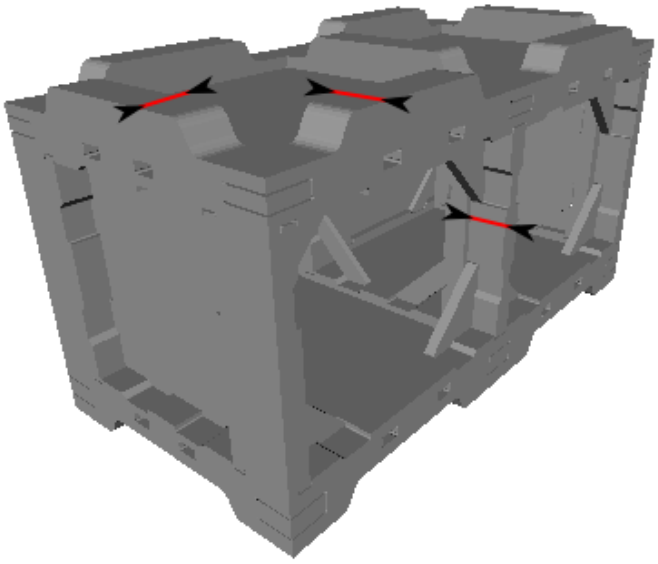
14.3.6 bwf_v_plank_width

bwf_v_plank_width default value : 30.0



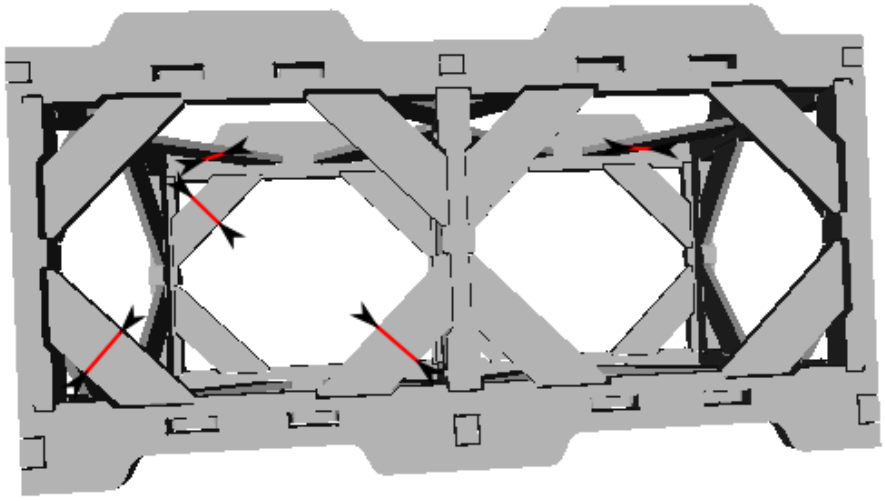
14.3.7 bwf_plank_height

bwf_plank_height default value : 20.0



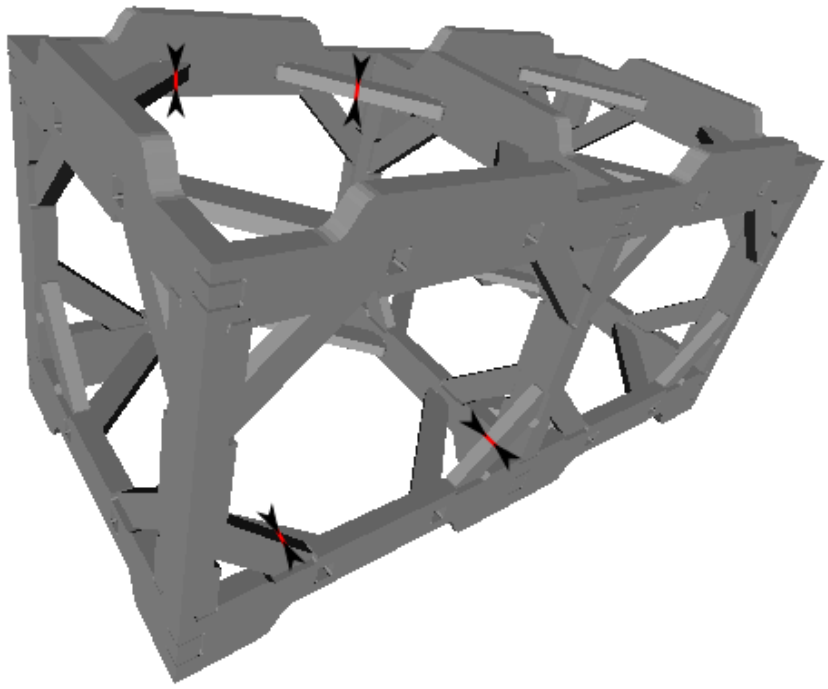
14.3.8 bwf_d_plank_width

bwf_d_plank_width default value : 30.0



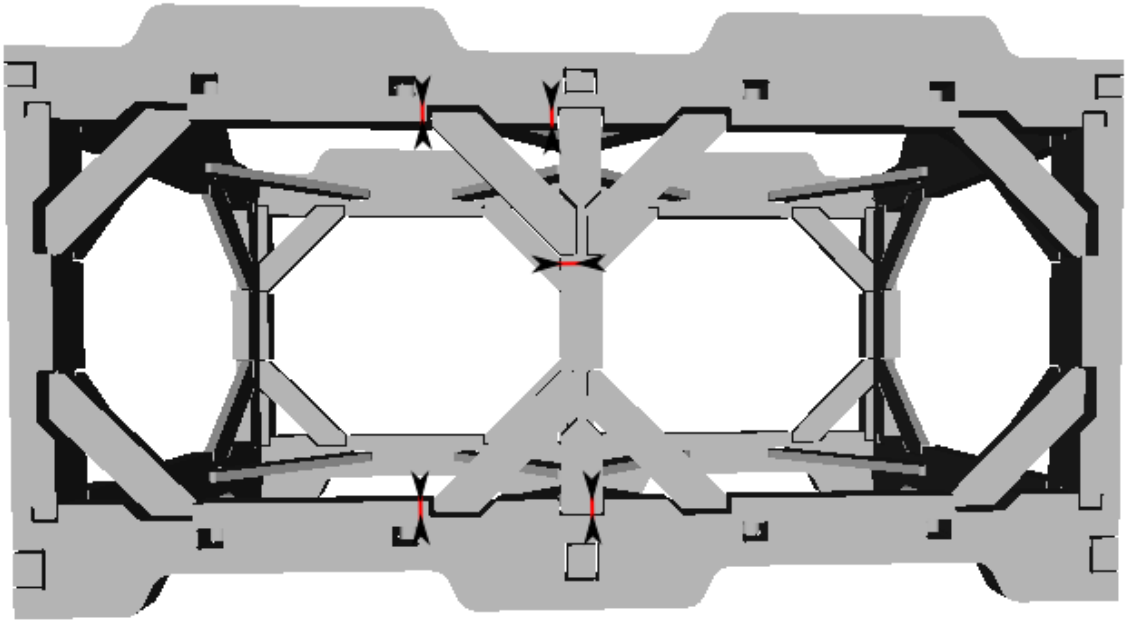
14.3.9 bwf_d_plank_height

bwf_d_plank_height default value : 10.0



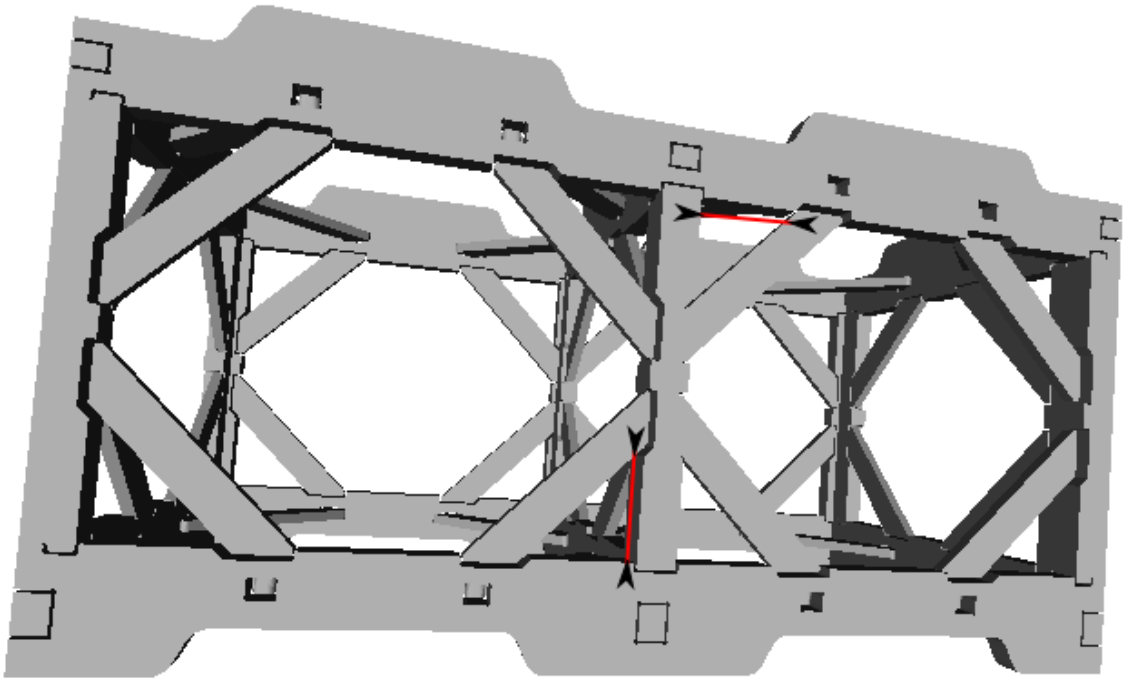
14.3.10 bwf_crenel_depth

bwf_crenel_depth default value : 5.0



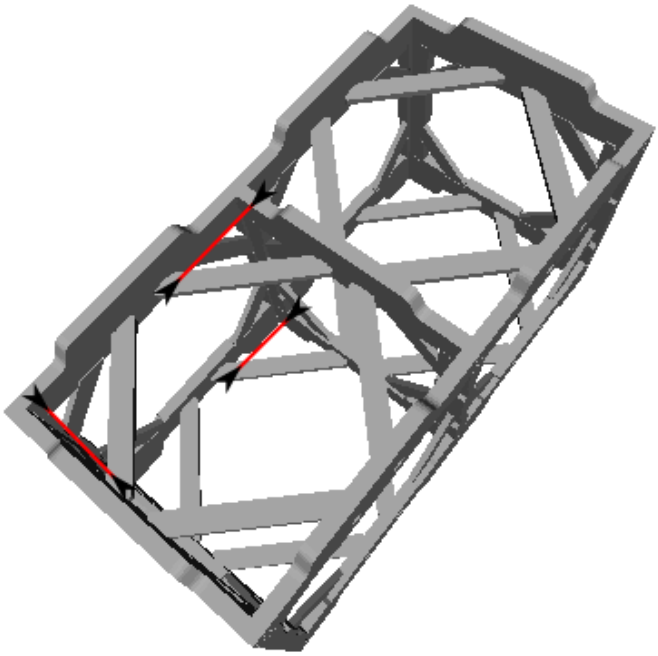
14.3.11 bwf_wall_diagonal_size

bwf_wall_diagonal_size default value : 50.0



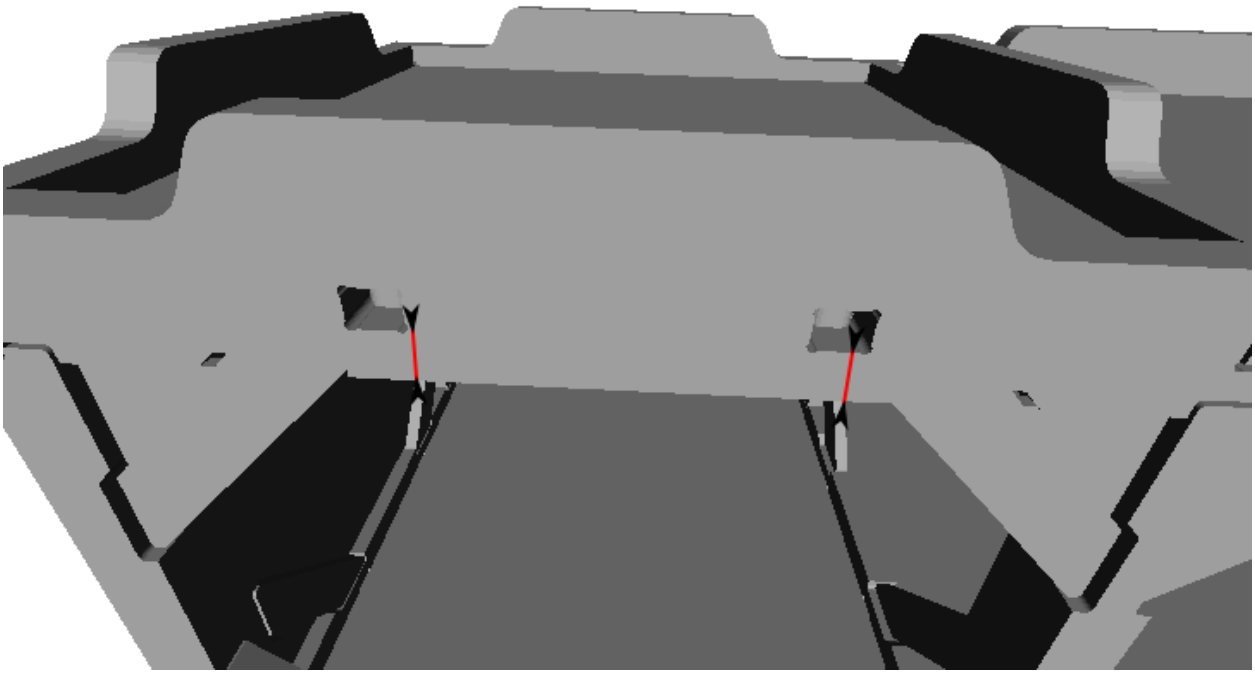
14.3.12 bwf_tobo_diagonal_size

bwf_tobo_diagonal_size default value : 100.0



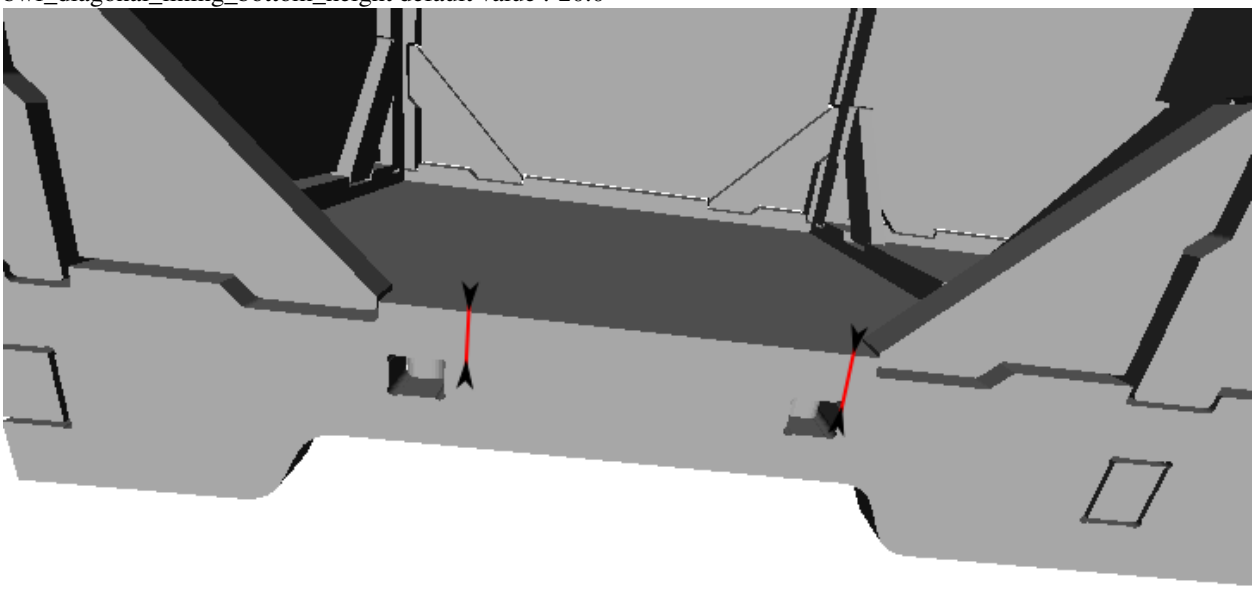
14.3.13 bwf_diagonal_lining_top_height

bwf_diagonal_lining_top_height default value : 20.0



14.3.14 bwf_diagonal_lining_bottom_height

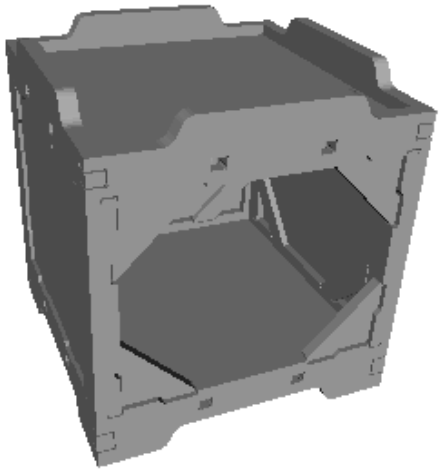
bwf_diagonal_lining_bottom_height default value : 20.0



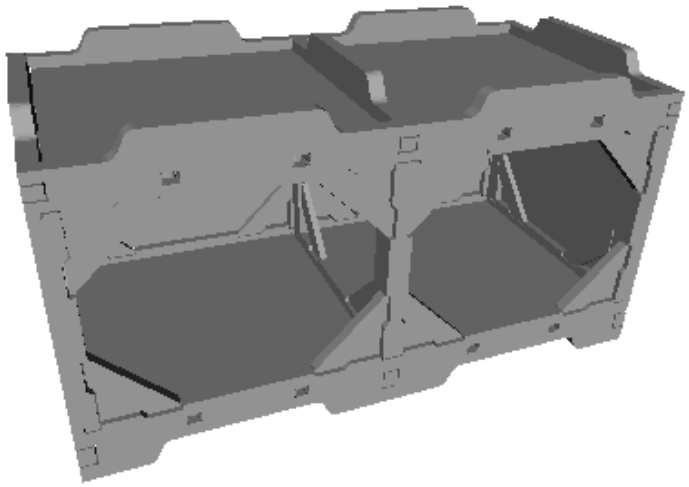
14.3.15 bwf_module_width

bwf_module_width default value : 1

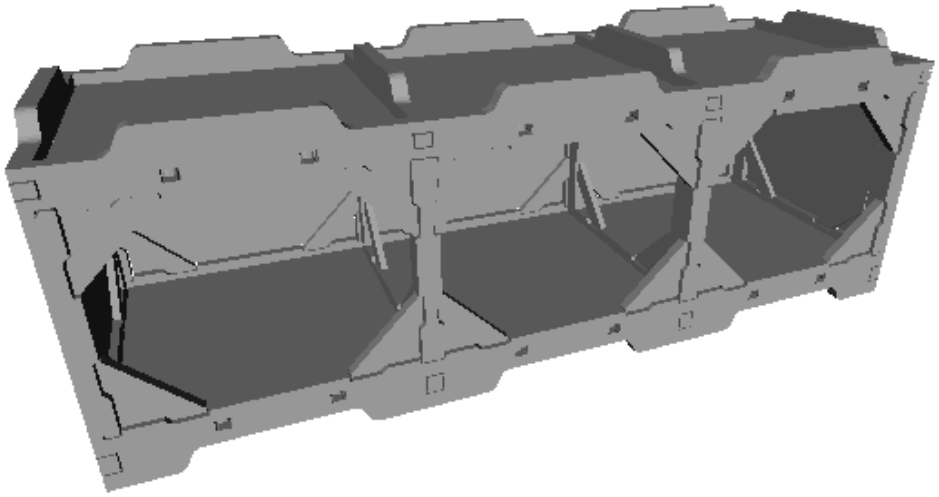
bwf_module_width = 1



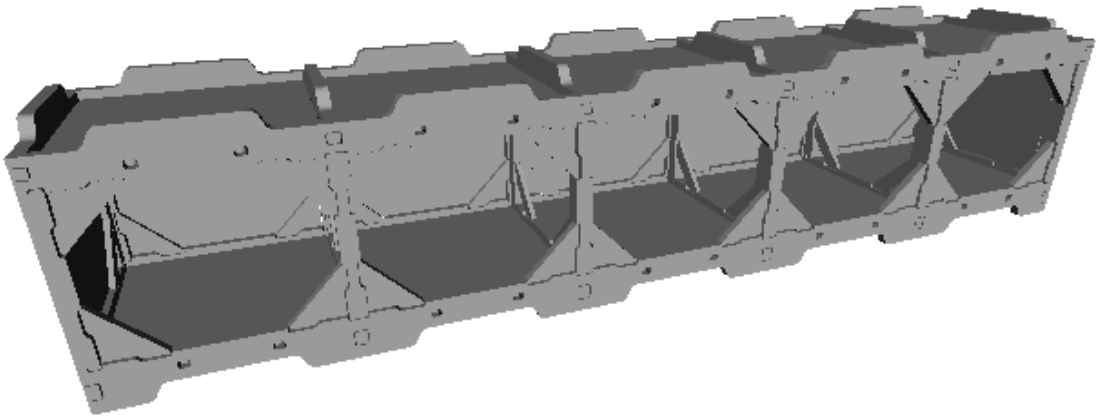
bwf_module_width = 2



bwf_module_width = 3

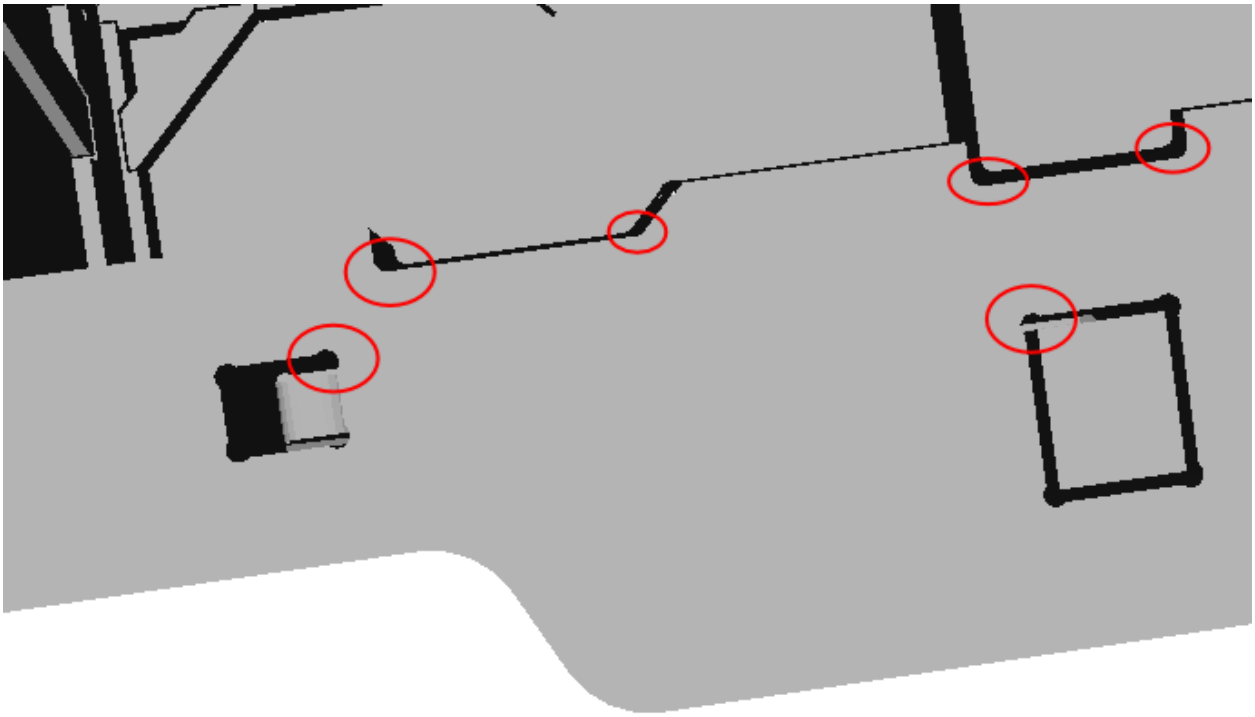


bwf_module_width = 5



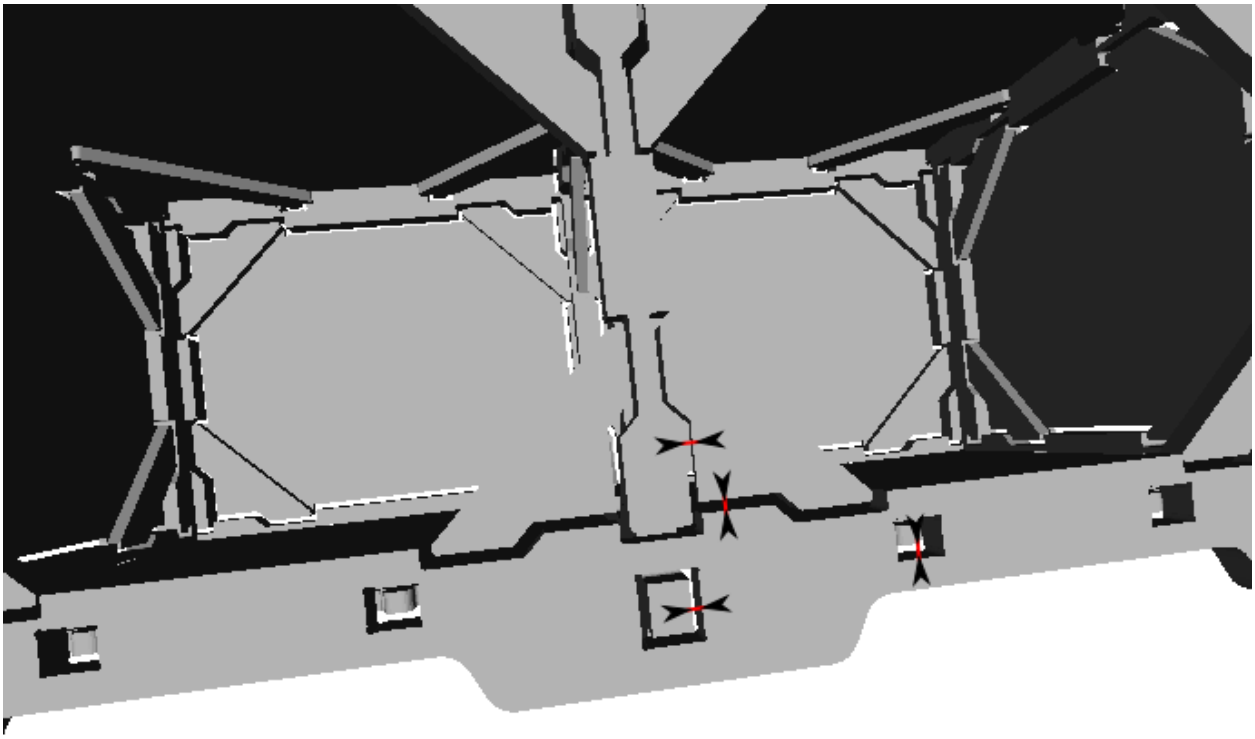
14.3.16 bwf_router_bit_radius

bwf_router_bit_radius default value : 2.0



14.3.17 bwf_cutting_extra

bwf_cutting_extra default value : 2.0



Note: The parameter *bwf_cutting_extra* doesn't affect the cnc cutting plan. It just help to see the junction between the plans.

14.3.18 bwf_slab_thickness

bwf_slab_thickness default value : 5.0

The slabs are the skin of your *box wood frame*. Set the slab thickness to the available plywood thickness of your supplier. Try to keep this relation:

```
bwf_plank_height > bwf_d_plank_height + bwf_slab_thickness
```

14.3.19 bwf_output_file_basename

bwf_output_file_basename default value : “”

Set the parameter *bwf_output_file_basename* to a not-empty string if you want to generate the output files. The *box_wood_frame_example.py* generates many files. These files can be generated in a directory or be identified by a common basename. The generated text file *text_report.txt* described all generated files.

Output file base name example:

```
bwf_output_file_basename = "my_output_dir/"
bwf_output_file_basename = "my_output_dir/my_output_basename"
bwf_output_file_basename = "my_output_basename"
```

14.4 Box wood frame conception

The notes relative to process of conception of the *Box wood frame* are available in the chapter Box Wood Frame Conception Details.

14.5 Box wood frame manufacturing

As you can see in the design files, the outline of the planks are quiet complex. Those many recessed fittings enable a solid assembly. To cut the planks precisely according to design files you have two methods:

- Use a 3-axis CNC
- Use a manual wood router and templates for each type of planks.

Notice that you need a CNC to make the templates.

The CNC method is well adapted when you want just few pieces of *Box wood frame*. The planks are cut in large plywood slabs (long and wide). This increase the final price of a *Box wood frame* module.

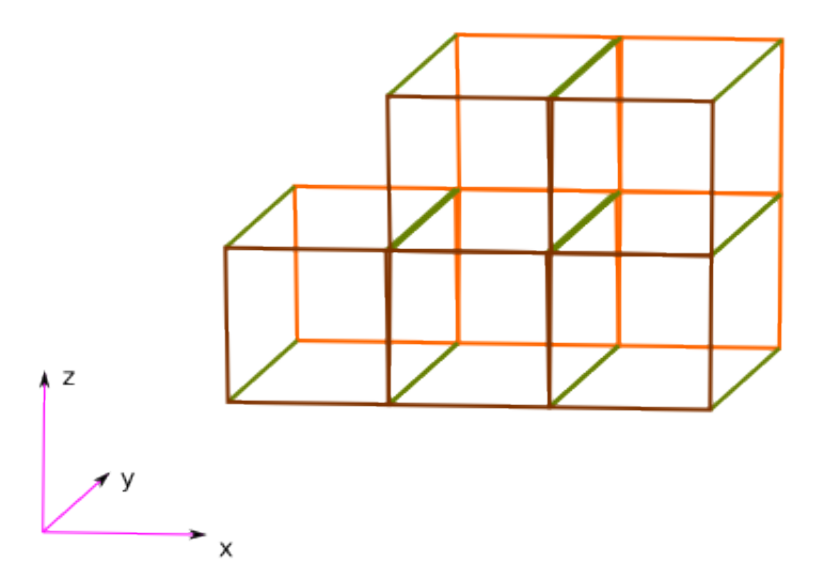
After getting the templates fitting your *Box wood frame* parameters, you can use a manual route to duplicate the planks. As raw material you can use solid wood plank (long and narrow). This is cheaper and provide a stronger assembly.

CHAPTER 15

Box Wood Frame Conception Details

15.1 Design purpose

The Box_Wood_Frame design is a solid and cost effective piece of furniture that can be piled up.



Pile-up of pieces of furniture

The pile-up feature is useful for:

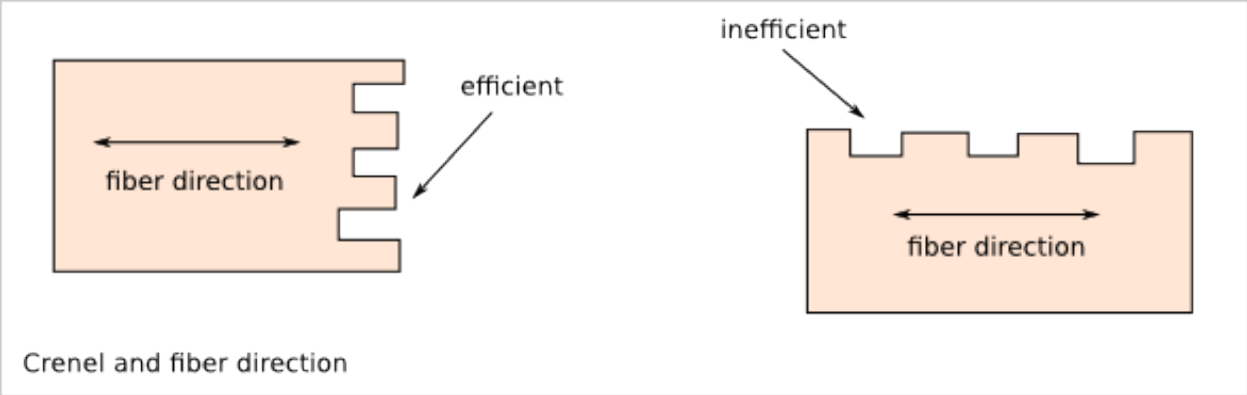
- rearranging interior
- transporting the pieces of furniture
- moving accommodation
- And also, maybe, building straw house
- it lets make big pieces of furniture out of small pieces of material
- the conception is also cut in several small problematics
- the big pieces of furniture can be easily dismount, transport and remount

The frame is made out of solid wood planks. The faces can then be closed with light plywood.

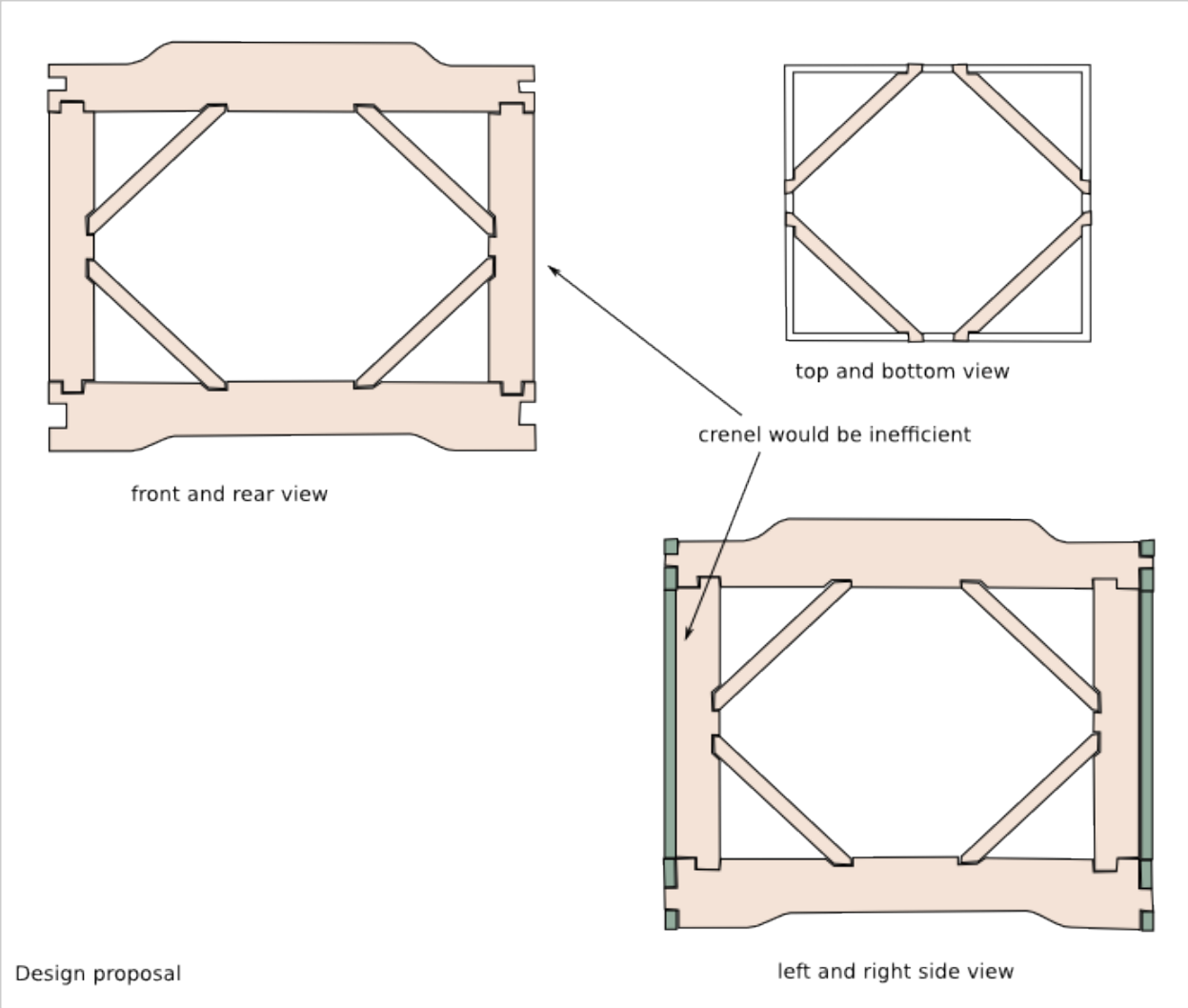
A module corresponds to one or several box of the grid. We focus on the concatenation of box along the x axis. N is the number of concatenated box (along the x axis).

15.2 Construction method

The planks are fixed by crenel and glue.



15.3 Design proposal



Wood frame plank list:

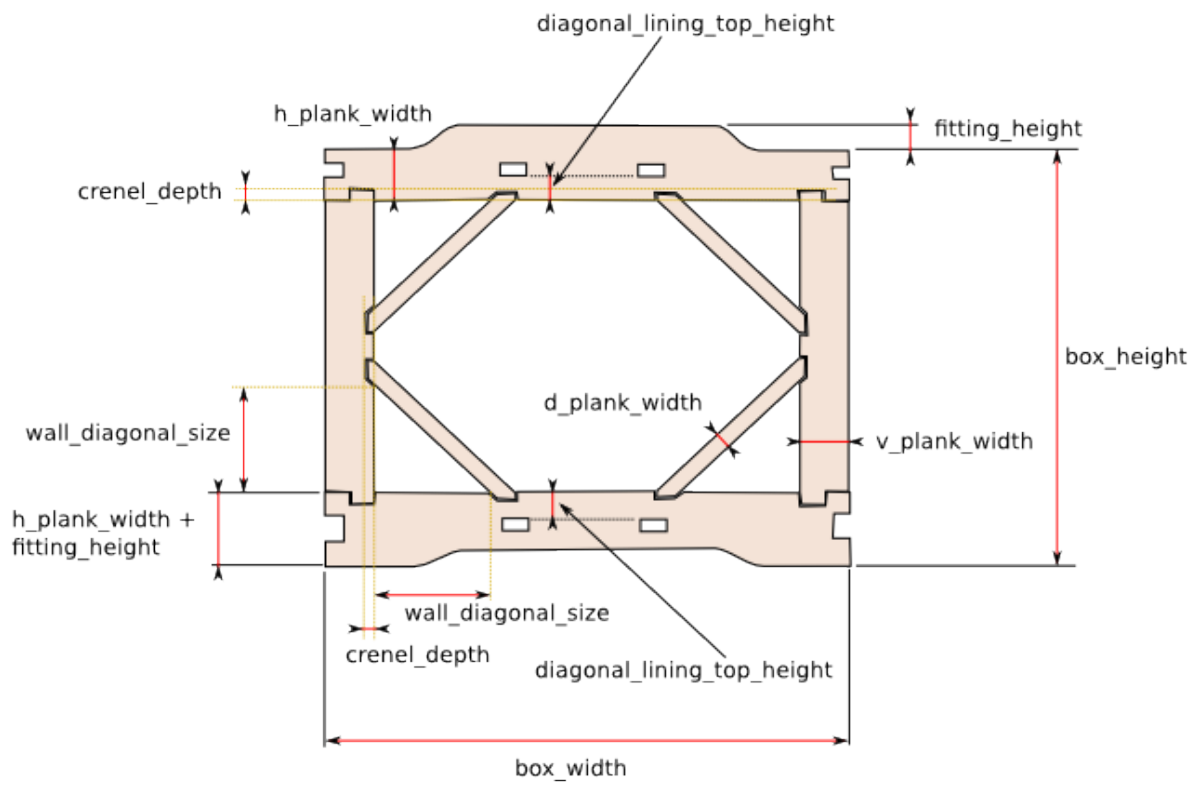
- horizontal plank: 20x60
- vertical plank: 20x30
- diagonal: 10x30 (thiner to give space for the face plywood)

15.4 Box wood frame parameters

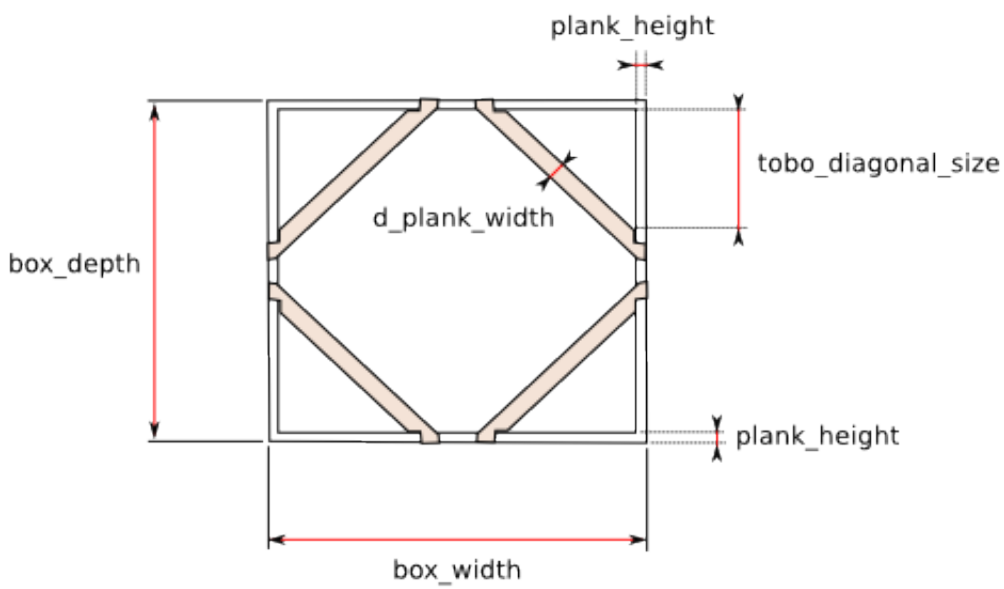
A module is defined by:

- The box dimension (w*d*h): 300*300*300
- The number of aggregated box: N*1*1

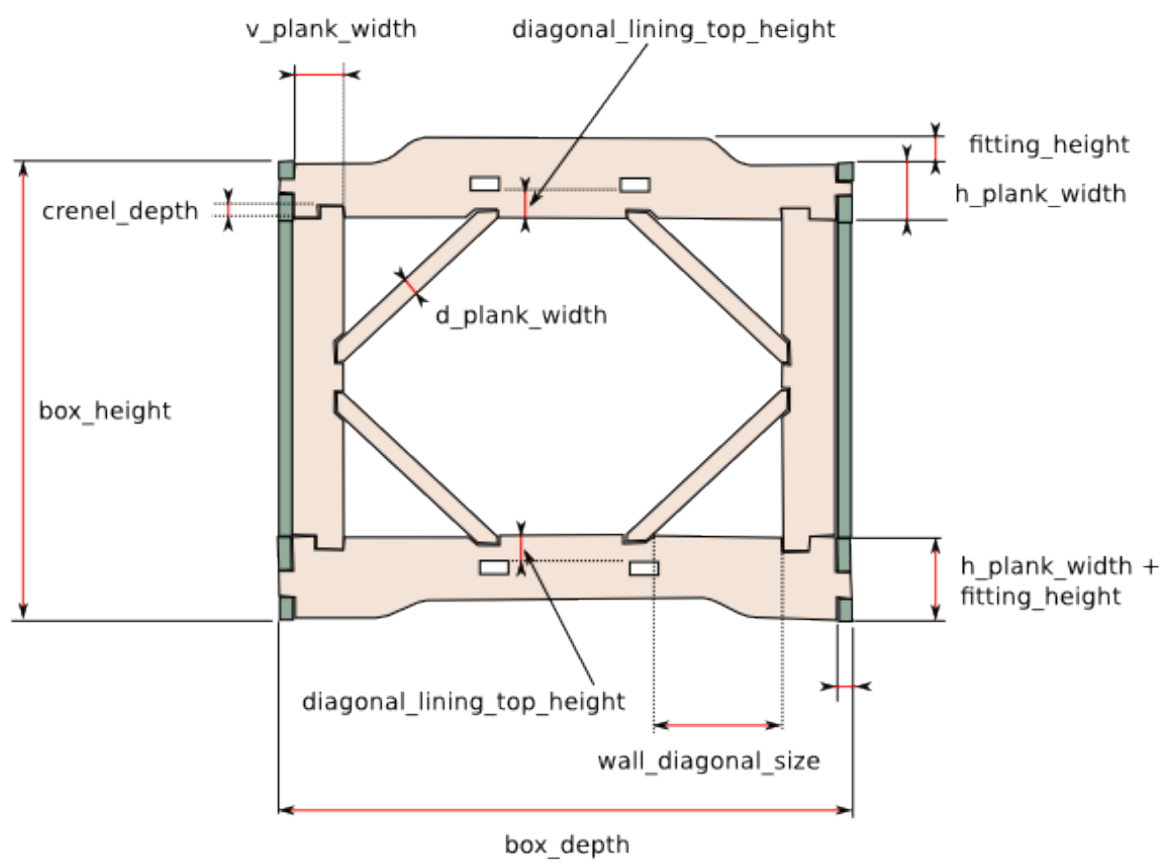
N is the length of the module. It is a number of boxes.



Box wood frame parameters 1



Box wood frame parameters 2



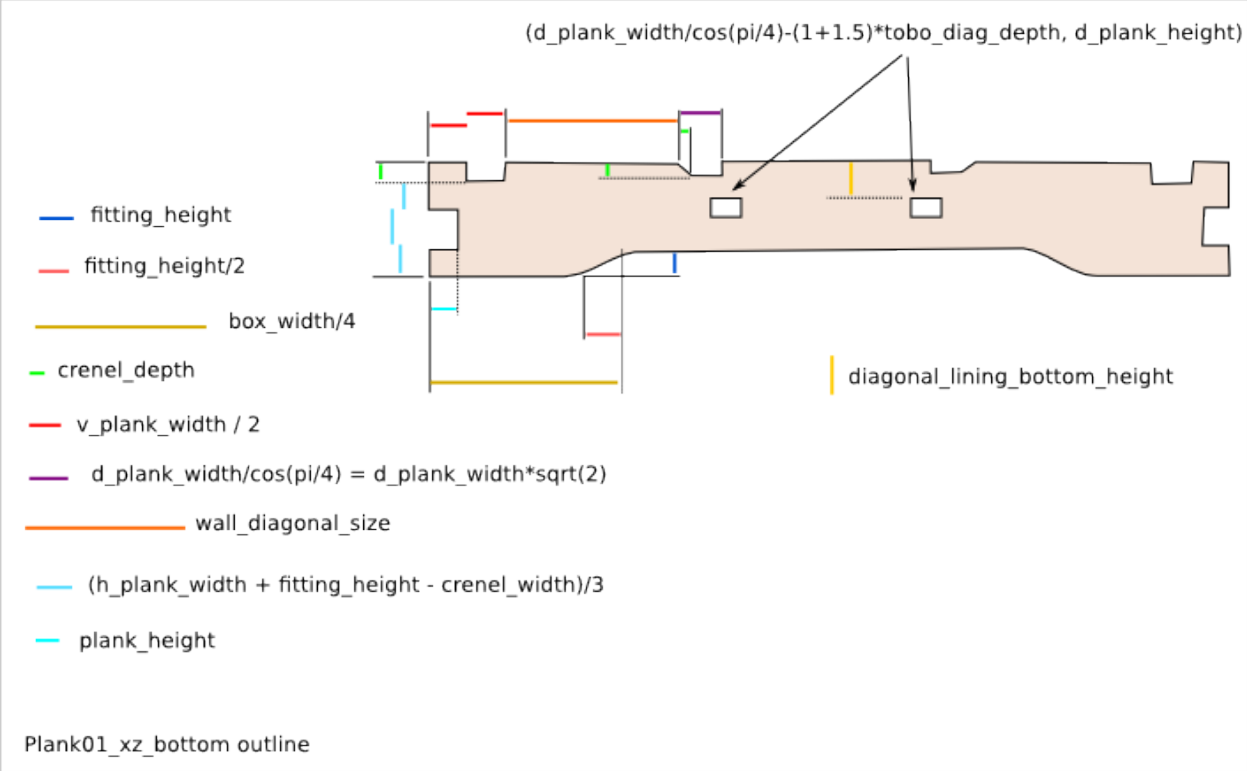
Box wood frame parameters 3

15.5 Plank outline description

Q is the number of required planks to build one module. It can depends on N, the length of the module.

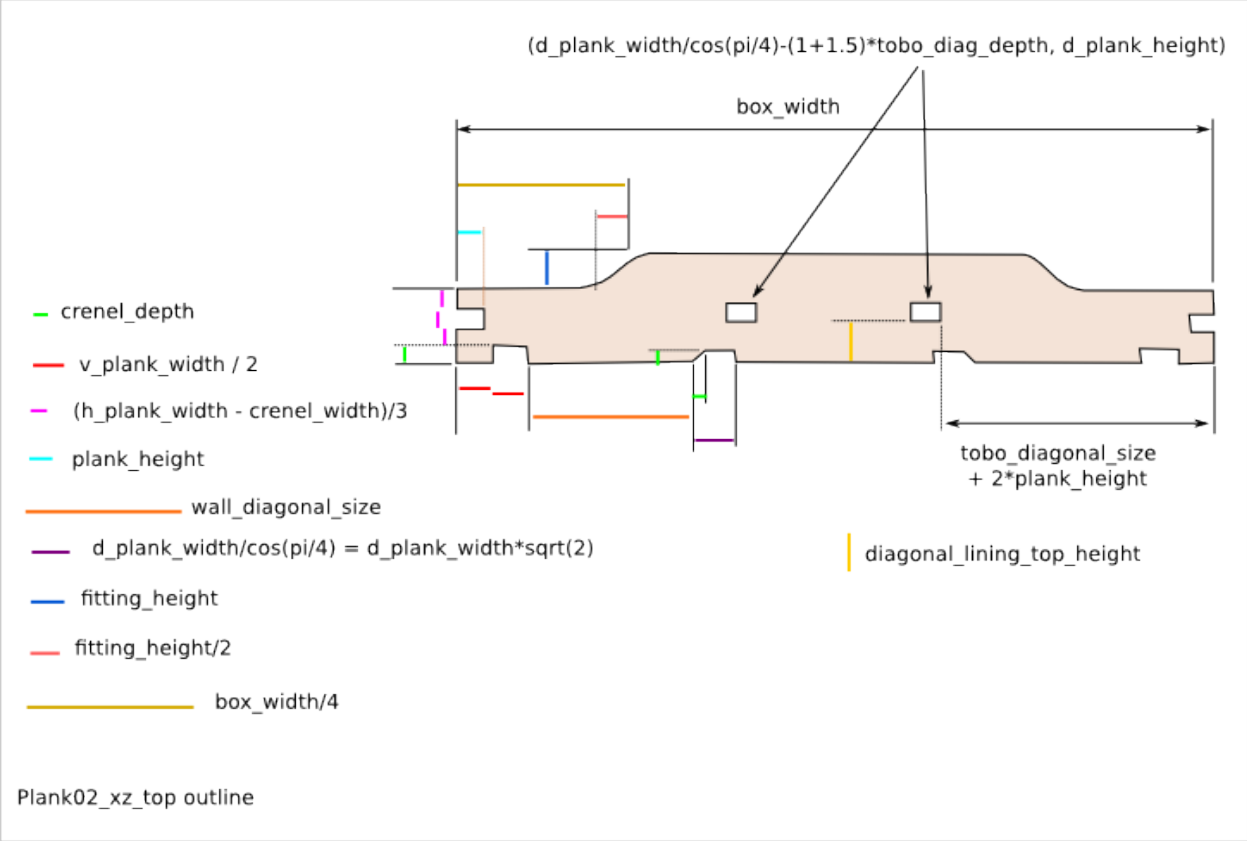
15.5.1 plank01_xz_bottom

Q = 2



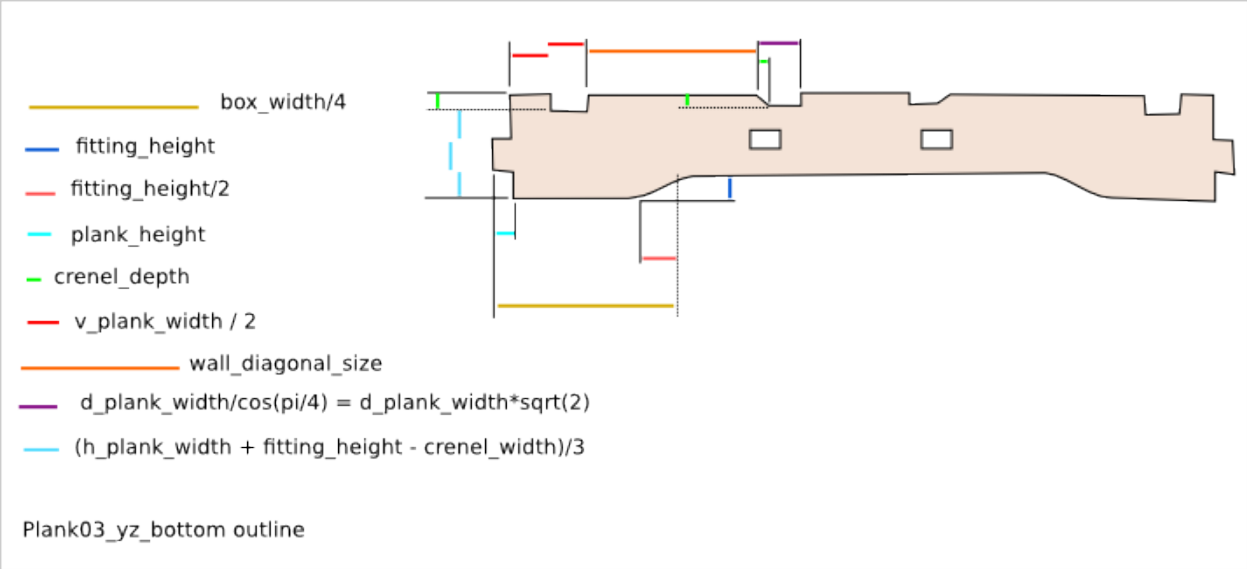
15.5.2 plank02_xz_top

Q = 2



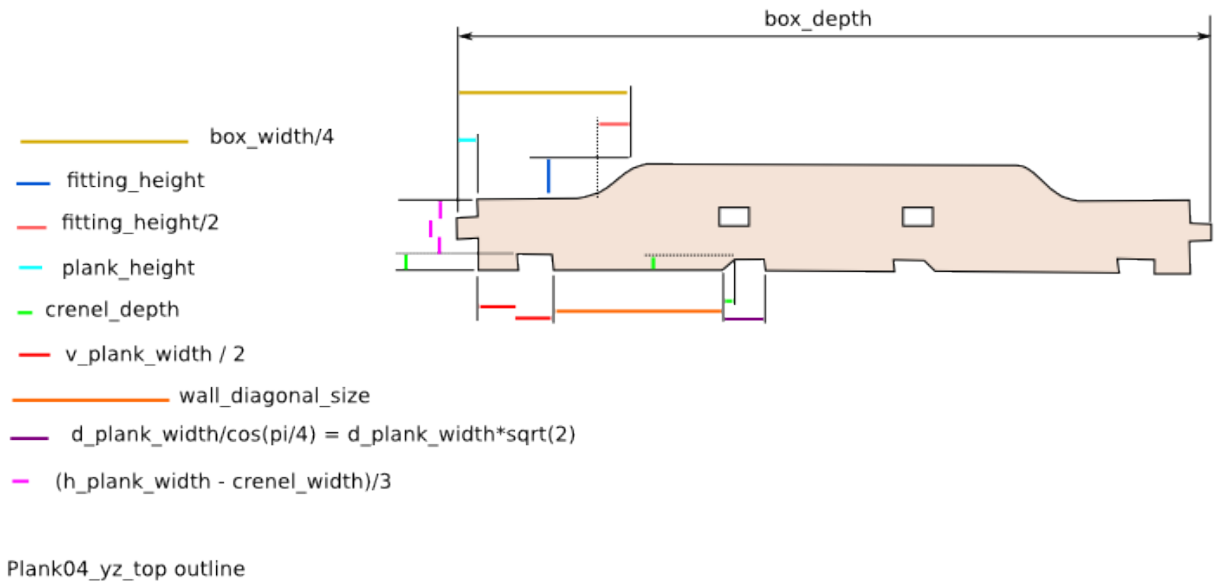
15.5.3 plank03_yz_bottom

Q = 2



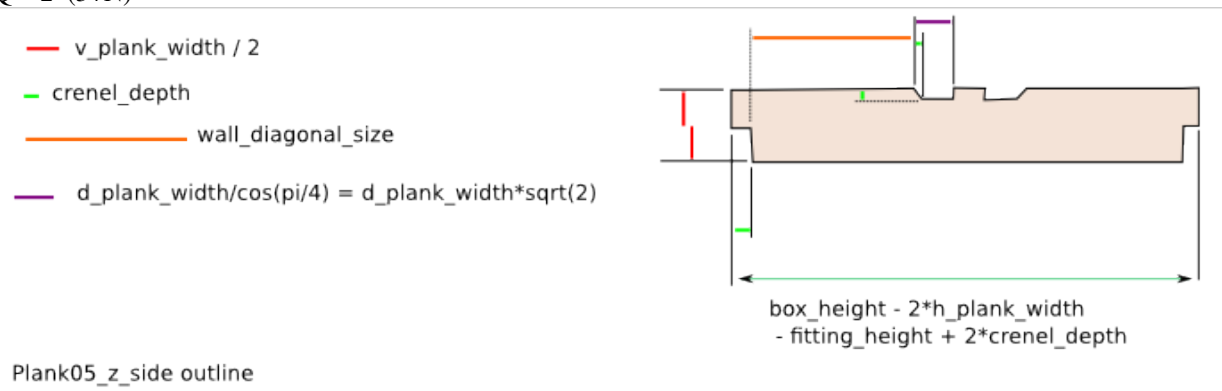
15.5.4 plank04_yz_top

Q = 2



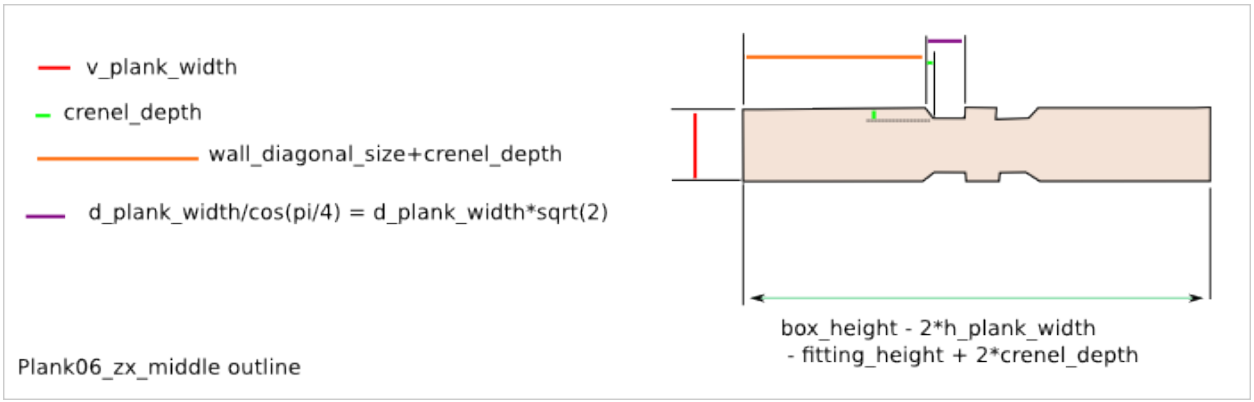
15.5.5 plank05_z_side

Q = 2*(3+N)



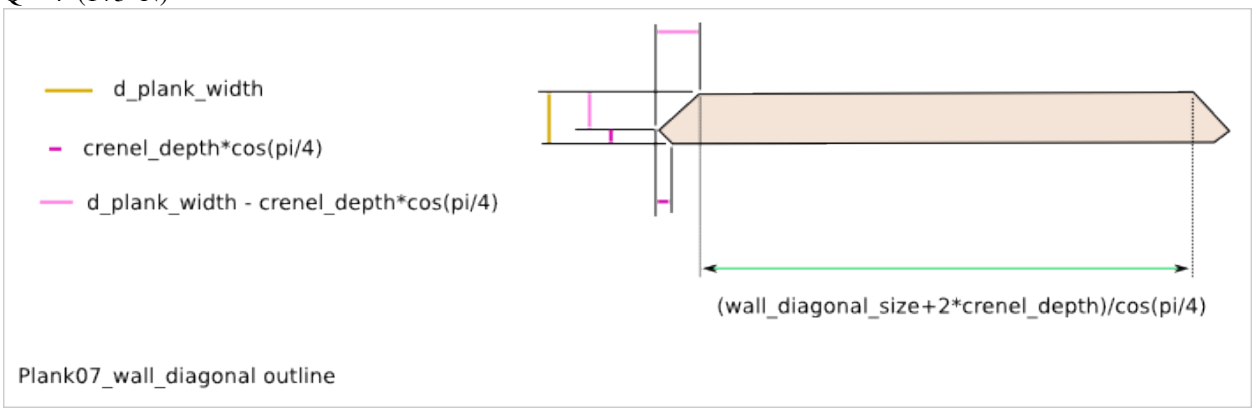
15.5.6 plank06_zx_middle

Q = 2*(N-1)



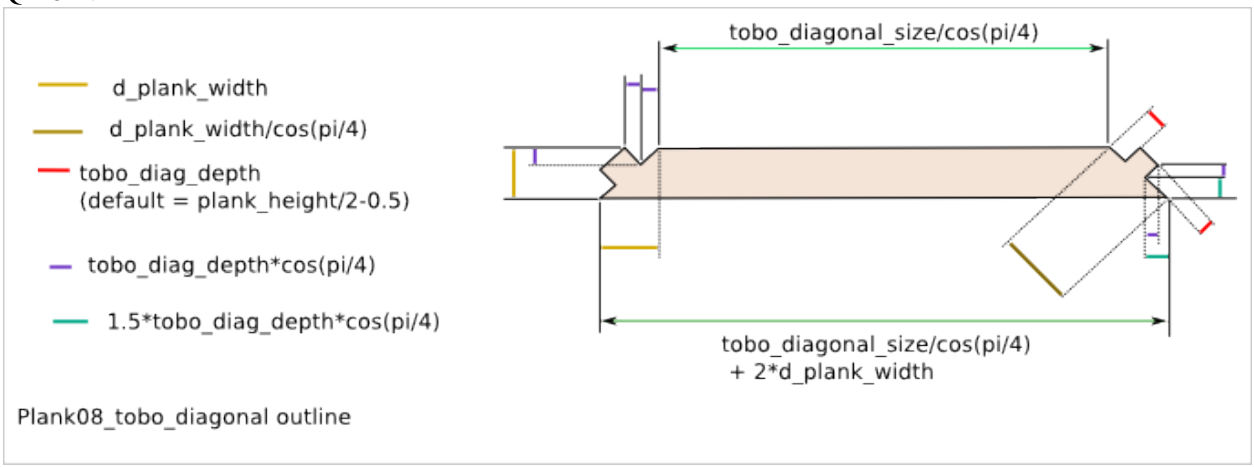
15.5.7 plank07_wall_diagonal

Q = 4*(1+3*N)



15.5.8 plank08_tobo_diagonal

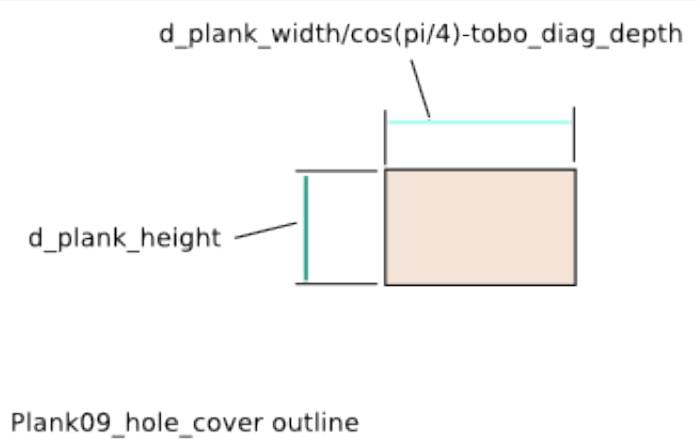
Q = 8*N



15.5.9 hole_cover

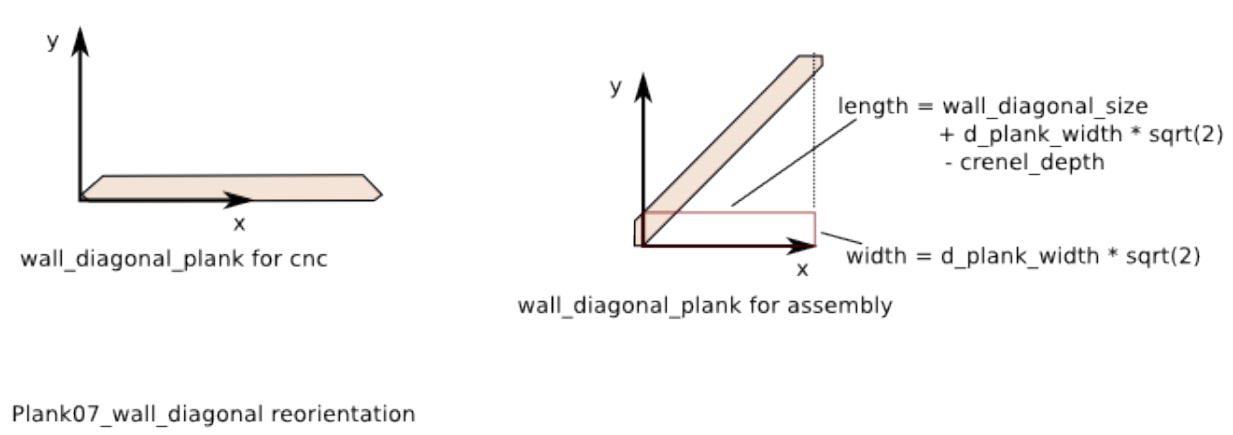
Q = 8*(N+1)

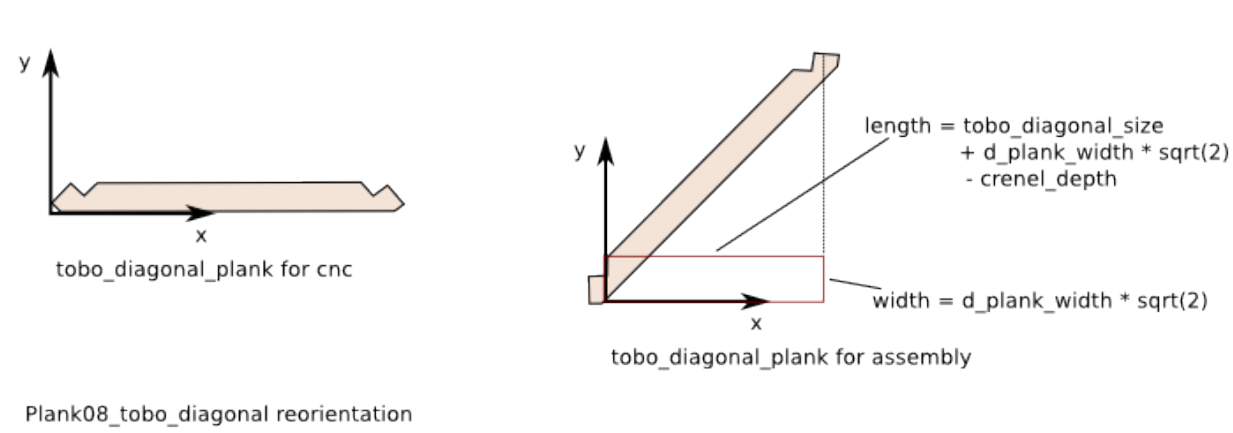
The plank09_hole_cover has an aesthetic functionality.



15.6 Diagonal plank reorientation

The planks are positioned in the cuboid assembly with the place_plank() function. To position the diagonal planks with this function, the diagonal planks must first be rotated of 45 degrees and affected with virtual length and width corresponding to the assimilated straight plank.

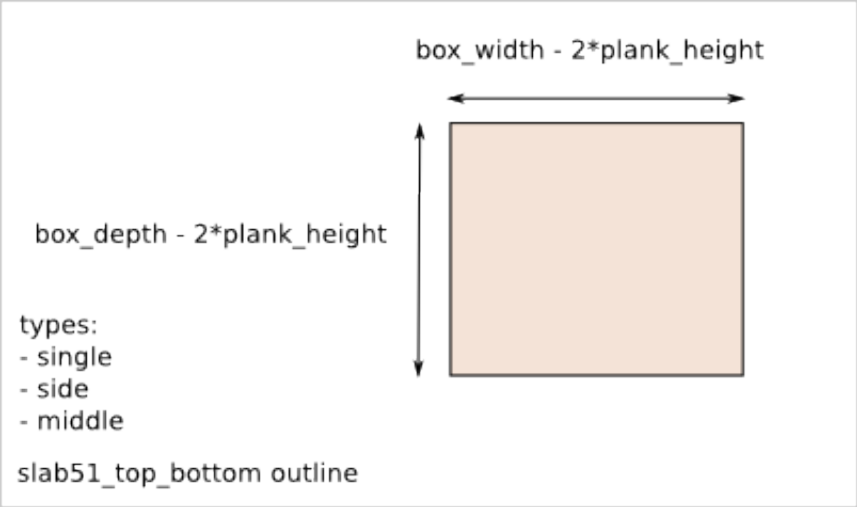




15.7 Slab outline description

15.7.1 slab51_tobo_single

Q = 2 if (N==1) else 0



15.7.2 slab52_tobo_side

Q = 4 if (N>1) else 0

Same outline as slab51_tobo_single except that the length is:

```
box_width - 1.5*plank_height
```

15.7.3 slab53_tobo_middle

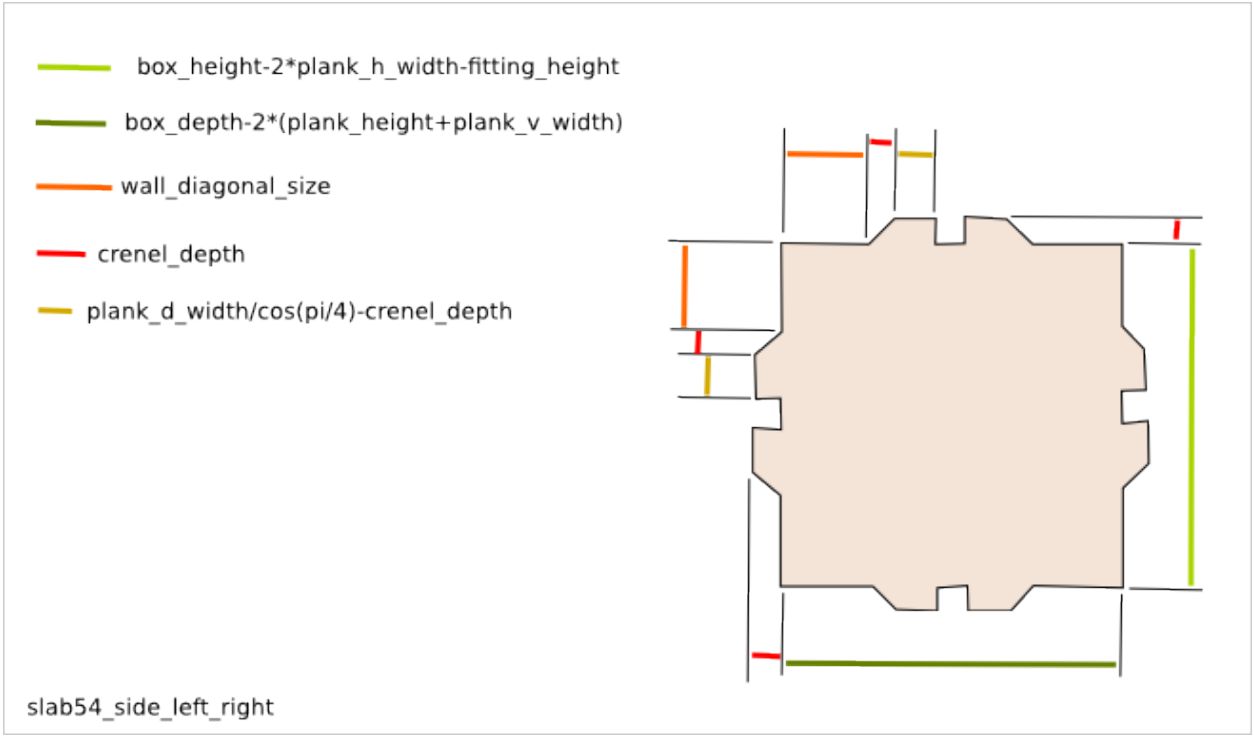
Q = 2*(N-2) if (N>2) else 0

Same outline as slab51_tobo_single except that the length is:

```
box_width - 1.0*plank_height
```

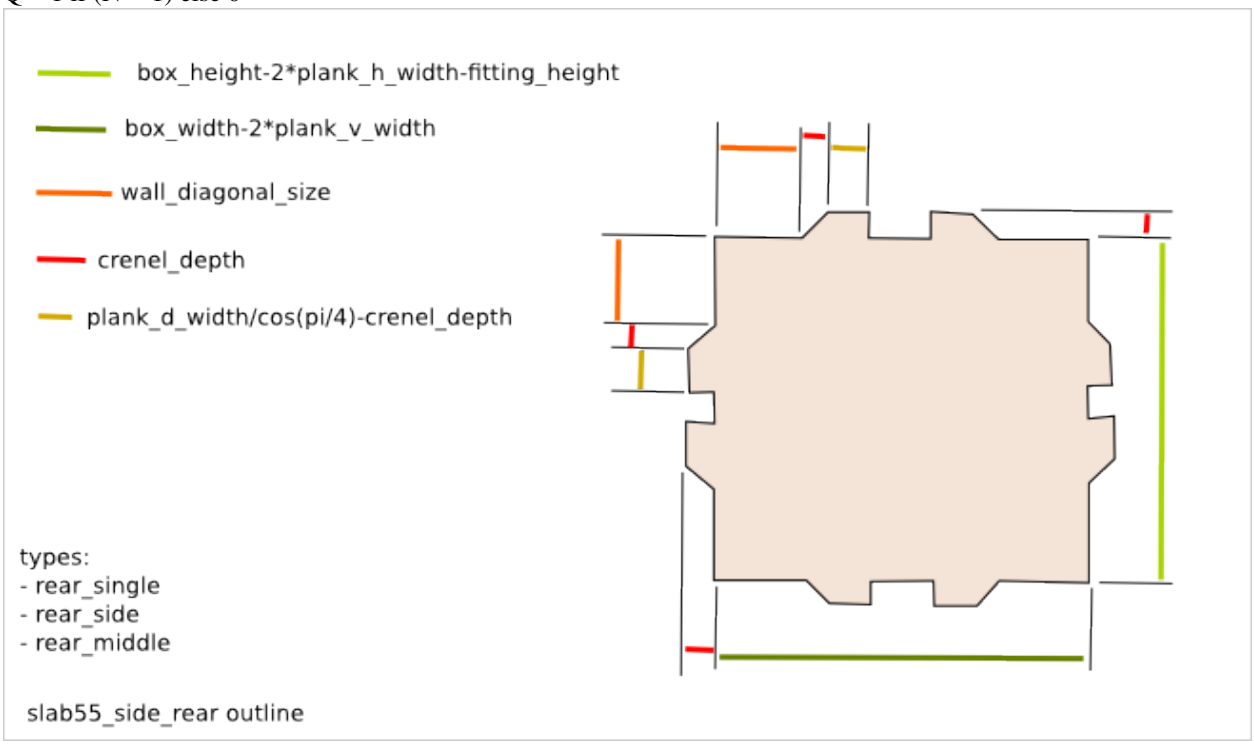
15.7.4 slab54_side_left_right

Q = 2



15.7.5 slab55_side_rear_single

Q = 1 if (N==1) else 0



15.7.6 slab56_side_rear_side

Q = 2 if (N>1) else 0

Same outline as slab55_side_rear_single except that the length is:

```
box_width - 1.5*plank_v_width
```

15.7.7 slab57_side_rear_middle

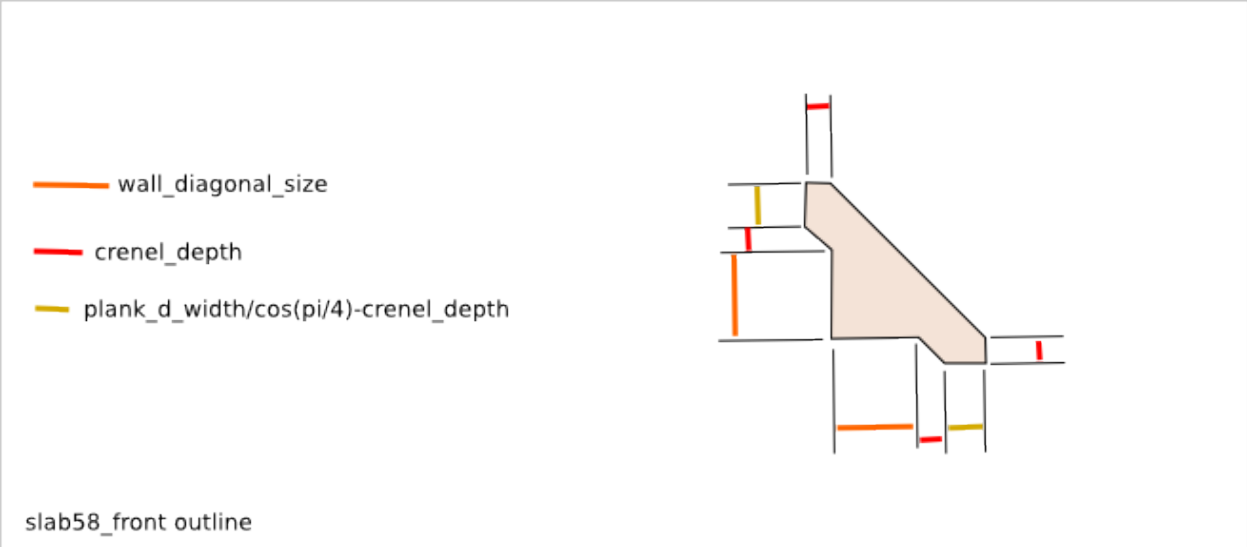
Q = N-2 if (N>2) else 0

Same outline as slab55_side_rear_single except that the length is:

```
box_width - 1.0*plank_v_width
```

15.7.8 slab58_front

Q = 4*N

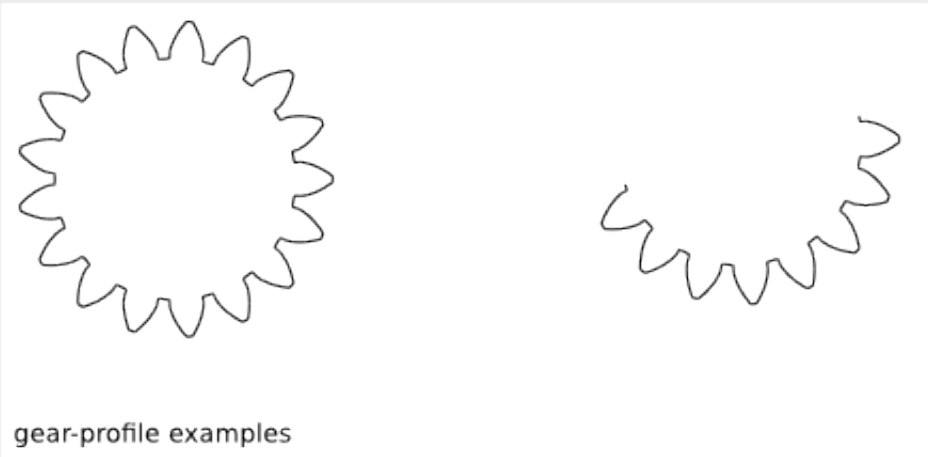


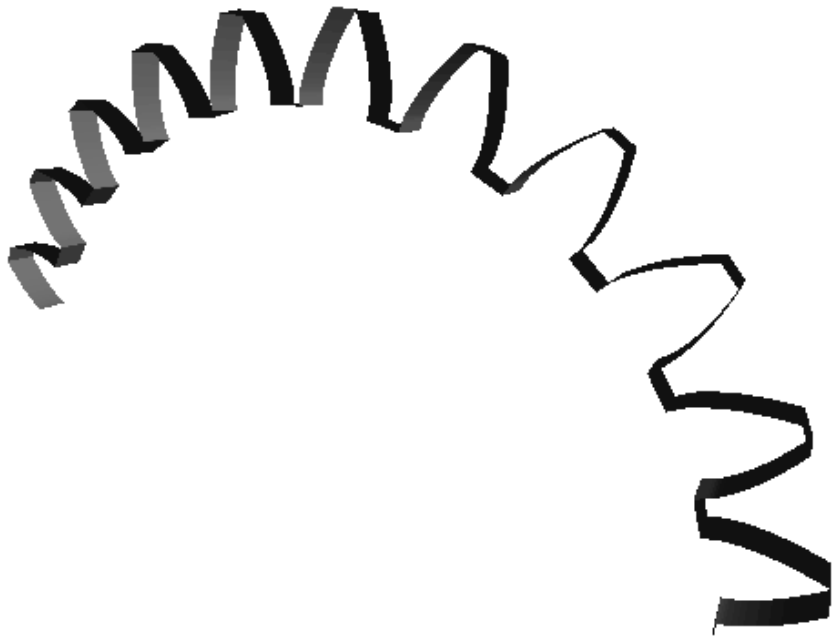
CHAPTER 16

Gear Profile Function

The *gear_profile()* function generates a format-B outline of a gear profile with the following features:

- the gear-tooth-profile ensures a *constant* line of action and a *constant* speed ratio
- the gear-profile (including the gear hollow) is makable by a 3-axis CNC
- a gear system with two parts can be simulated with the Tkinter GUI
- very configurable: asymmetrical teeth are possible
- active tooth profile made out of arcs and with a continuous tangent inclination
- optional portion of gear to make split gearwheel





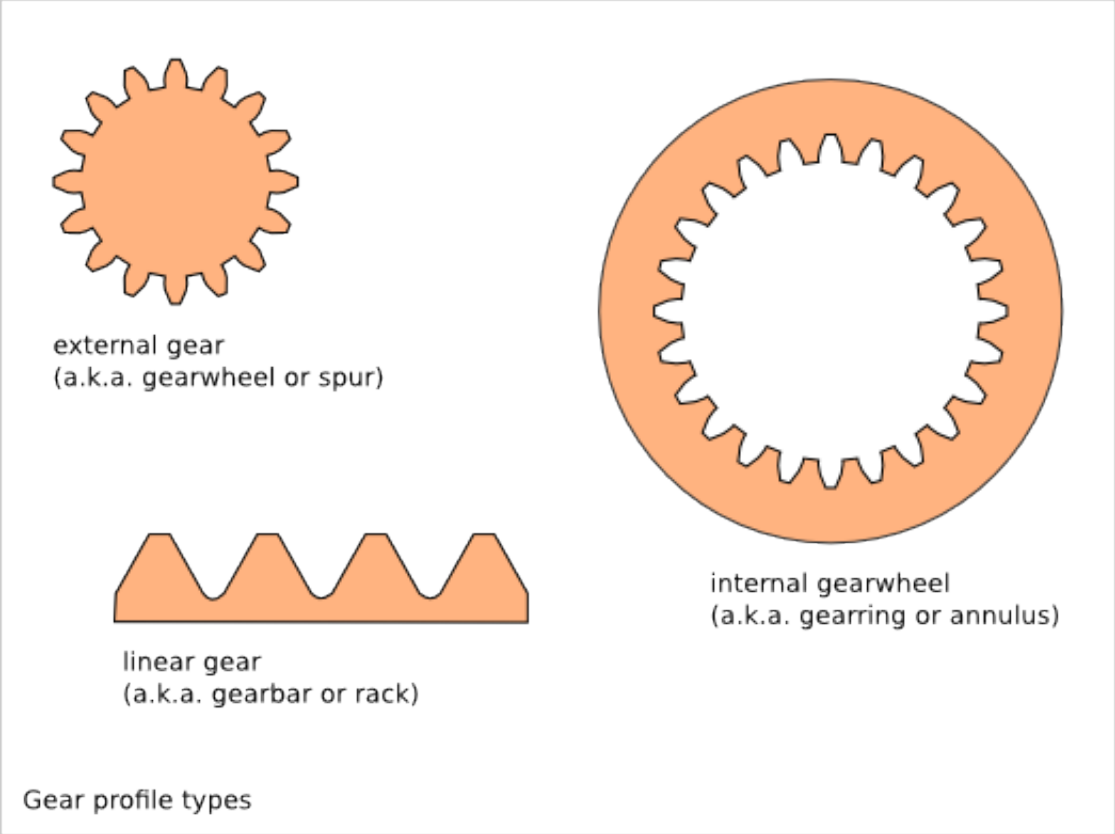
To get an overview of the possible gear_profiles that can be generated by *gear_profile()*, run:

```
> python gear_profile.py --run_self_test
```

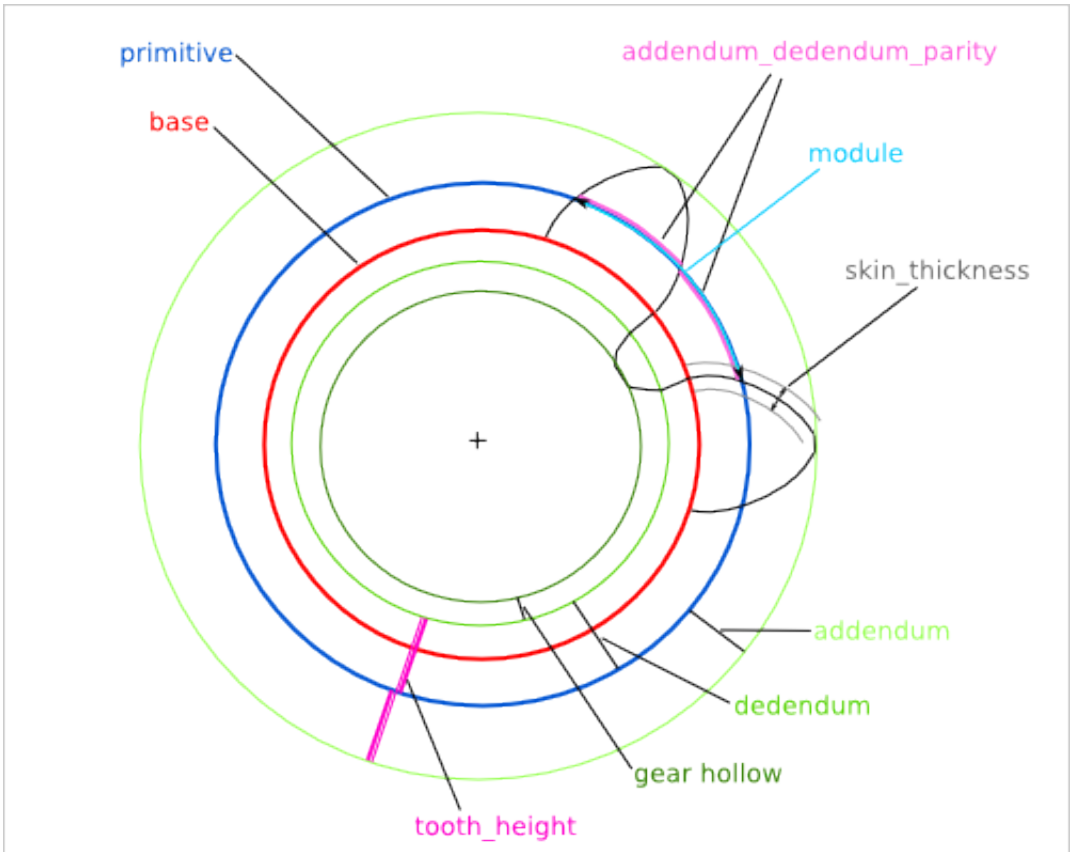
16.1 Gear high-level parameters

The gear *high-level* parameters let describe with a reduce number of integers and floats a complete gear system. Some of these *high-level* are depending on each others.

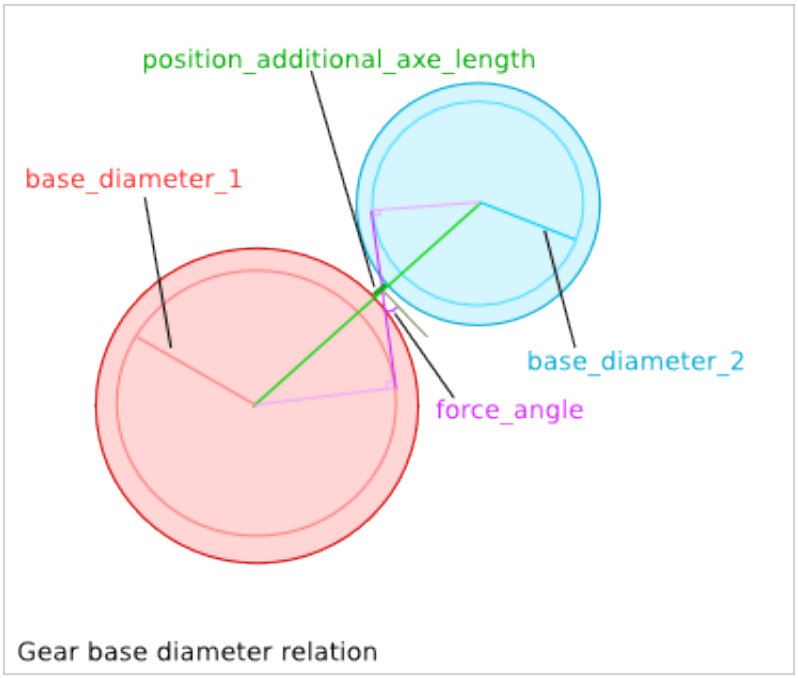
16.1.1 Gear types



16.1.2 Gearwheel high-level parameters

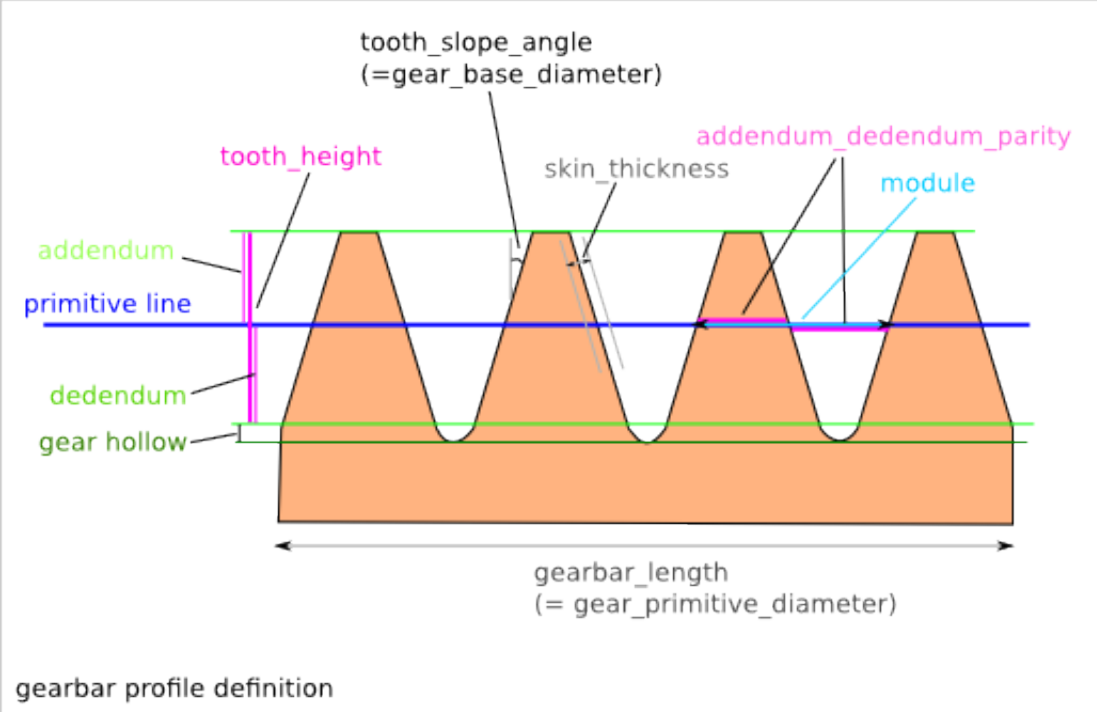


Gearwheel profile definition



Gear base diameter relation

16.1.3 Gearbar high-level parameters



gearbar profile definition

16.2 gear_profile() function arguments list

The arguments of the function *gear_profile()* are not directly the *high-level* gear parameters but *constraints* used to deduce the *high-level* gear parameters.

The switches of the module *gear_profile.py* are directly connected to the function *gear_profile()*. Use the module *gear_profile.py* to experiment the *gear_profile()* arguments. Notice that *-h* and *–run_self_test* are not arguments of *gear_profile()*.

```
usage: gear_profile.py [-h] [--gear_type SW_GEAR_TYPE]
                       [--gear_tooth_nb SW_GEAR_TOOTH_NB]
                       [--gear_module SW_GEAR_MODULE]
                       [--gear_primitive_diameter SW_GEAR_PRIMITIVE_DIAMETER]
                       [--gear_addendum_dedendum_parity SW_GEAR_ADDENDUM_DEDENDUM_PARITY]
                       [--gear_tooth_half_height SW_GEAR_TOOTH_HALF_HEIGHT]
                       [--gear_addendum_height_pourcentage SW_GEAR_ADDENDUM_HEIGHT_POURCENTAGE]
                       [--gear_dedendum_height_pourcentage SW_GEAR_DEDENDUM_HEIGHT_POURCENTAGE]
                       [--gear_hollow_height_pourcentage SW_GEAR_HOLLOW_HEIGHT_POURCENTAGE]
                       [--gear_router_bit_radius SW_GEAR_ROUTER_BIT_RADIUS]
                       [--gear_base_diameter SW_GEAR_BASE_DIAMETER]
                       [--gear_force_angle SW_GEAR_FORCE_ANGLE]
                       [--gear_tooth_resolution SW_GEAR_TOOTH_RESOLUTION]
                       [--gear_skin_thickness SW_GEAR_SKIN_THICKNESS]
                       [--gear_base_diameter_n SW_GEAR_BASE_DIAMETER_N]
                       [--gear_force_angle_n SW_GEAR_FORCE_ANGLE_N]
                       [--gear_tooth_resolution_n SW_GEAR_TOOTH_RESOLUTION_N]
                       [--gear_skin_thickness_n SW_GEAR_SKIN_THICKNESS_N]
                       [--second_gear_type SW_SECOND_GEAR_TYPE]
```

```
                      [--second_gear_tooth_nb SW_SECOND_GEAR_TOOTH_NB]
                      [--second_gear_primitive_diameter SW_SECOND_GEAR_PRIMITIVE_DIAMETER]
                      [--second_gear_addendum_dedendum_parity SW_SECOND_GEAR_ADDENDUM_DEDENDUM_PARITY]
                      [--second_gear_tooth_half_height SW_SECOND_GEAR_TOOTH_HALF_HEIGHT]
                      [--second_gear_addendum_height_pourcentage SW_SECOND_GEAR_ADDENDUM_HEIGHT_POURCE
                      [--second_gear_dedendum_height_pourcentage SW_SECOND_GEAR_DEDENDUM_HEIGHT_POURCE
                      [--second_gear_hollow_height_pourcentage SW_SECOND_GEAR_HOLLOW_HEIGHT_POURCENTAG
                      [--second_gear_router_bit_radius SW_SECOND_GEAR_ROUTER_BIT_RADIUS]
                      [--second_gear_base_diameter SW_SECOND_GEAR_BASE_DIAMETER]
                      [--second_gear_tooth_resolution SW_SECOND_GEAR_TOOTH_RESOLUTION]
                      [--second_gear_skin_thickness SW_SECOND_GEAR_SKIN_THICKNESS]
                      [--second_gear_base_diameter_n SW_SECOND_GEAR_BASE_DIAMETER_N]
                      [--second_gear_tooth_resolution_n SW_SECOND_GEAR_TOOTH_RESOLUTION_N]
                      [--second_gear_skin_thickness_n SW_SECOND_GEAR_SKIN_THICKNESS_N]
                      [--gearbar_slope SW_GEARBAR_SLOPE]
                      [--gearbar_slope_n SW_GEARBAR_SLOPE_N]
                      [--center_position_x SW_CENTER_POSITION_X]
                      [--center_position_y SW_CENTER_POSITION_Y]
                      [--gear_initial_angle SW_GEAR_INITIAL_ANGLE]
                      [--second_gear_position_angle SW_SECOND_GEAR_POSITION_ANGLE]
                      [--second_gear_additional_axis_length SW_SECOND_GEAR_ADDITIONAL_AXIS_LENGTH]
                      [--cut_portion SW_CUT_PORTION SW_CUT_PORTION SW_CUT_PORTION]
                      [--gear_profile_height SW_GEAR_PROFILE_HEIGHT]
                      [--simulation_enable]
                      [--output_file_basename SW_OUTPUT_FILE_BASENAME]
                      [--run_self_test]

Command line interface for the function gear_profile().

optional arguments:
-h, --help            show this help message and exit
--gear_type SW_GEAR_TYPE, --gt SW_GEAR_TYPE
                      Select the type of gear. Possible values: 'e', 'i',
                      'l'. Default: 'e'
--gear_tooth_nb SW_GEAR_TOOTH_NB, --gtn SW_GEAR_TOOTH_NB
                      Set the number of teeth of the first gear_profile.
--gear_module SW_GEAR_MODULE, --gm SW_GEAR_MODULE
                      Set the module of the gear. It influences the
                      gear_profile diameters.
--gear_primitive_diameter SW_GEAR_PRIMITIVE_DIAMETER, --gpd SW_GEAR_PRIMITIVE_DIAMETER
                      If not zero, redefine the gear module to get this
                      primitive diameter of the first gear_profile. Default:
                      0. If gearbar, it redefines the length.
--gear_addendum_dedendum_parity SW_GEAR_ADDENDUM_DEDENDUM_PARITY, --gadp SW_GEAR_ADDENDUM_DEDENDUM_PA
                      Set the addendum / dedendum parity of the first
                      gear_profile. Default: 50.0%
--gear_tooth_half_height SW_GEAR_TOOTH_HALF_HEIGHT, --gthh SW_GEAR_TOOTH_HALF_HEIGHT
                      If not zero, redefine the tooth half height of the
                      first gear_profile. Default: 0.0
--gear_addendum_height_pourcentage SW_GEAR_ADDENDUM_HEIGHT_POURCENTAGE, --gahp SW_GEAR_ADDENDUM_HEIGH
                      Set the addendum height of the first gear_profile in
                      pourcentage of the tooth half height. Default: 100.0%
--gear_dedendum_height_pourcentage SW_GEAR_DEDENDUM_HEIGHT_POURCENTAGE, --gdhp SW_GEAR_DEDENDUM_HEIGH
                      Set the dedendum height of the first gear_profile in
                      pourcentage of the tooth half height. Default: 100.0%
--gear_hollow_height_pourcentage SW_GEAR_HOLLOW_HEIGHT_POURCENTAGE, --ghhp SW_GEAR_HOLLOW_HEIGHT_POUR
                      Set the hollow height of the first gear_profile in
                      pourcentage of the tooth half height. The hollow is a
```

```
                        clear space for the top of the teeth of the other
                        gearwheel. Default: 25.0%
--gear_router_bit_radius SW_GEAR_ROUTER_BIT_RADIUS, --grr SW_GEAR_ROUTER_BIT_RADIUS
                        Set the router_bit radius used to create the gear
                        hollow of the first gear_profile. Default: 0.1
--gear_base_diameter SW_GEAR_BASE_DIAMETER, --gbd SW_GEAR_BASE_DIAMETER
                        If not zero, redefine the base diameter of the first
                        gear involute. Default: 0
--gear_force_angle SW_GEAR_FORCE_ANGLE, --gfa SW_GEAR_FORCE_ANGLE
                        If not zero, redefine the gear_base_diameter to get
                        this force angle at the gear contact. Default: 0.0
--gear_tooth_resolution SW_GEAR_TOOTH_RESOLUTION, --gtr SW_GEAR_TOOTH_RESOLUTION
                        It sets the number of segments of the gear involute.
                        Default: 2
--gear_skin_thickness SW_GEAR_SKIN_THICKNESS, --gst SW_GEAR_SKIN_THICKNESS
                        Add or remove radial thickness on the gear involute.
                        Default: 0.0
--gear_base_diameter_n SW_GEAR_BASE_DIAMETER_N, --gbdn SW_GEAR_BASE_DIAMETER_N
                        If not zero, redefine the base diameter of the first
                        gear negative involute. Default: 0
--gear_force_angle_n SW_GEAR_FORCE_ANGLE_N, --gfan SW_GEAR_FORCE_ANGLE_N
                        If not zero, redefine the negative_gear_base_diameter
                        to get this force angle at the gear contact. Default:
                        0.0
--gear_tooth_resolution_n SW_GEAR_TOOTH_RESOLUTION_N, --gtrn SW_GEAR_TOOTH_RESOLUTION_N
                        If not zero, it sets the number of segments of the
                        gear negative involute. Default: 0
--gear_skin_thickness_n SW_GEAR_SKIN_THICKNESS_N, --gstn SW_GEAR_SKIN_THICKNESS_N
                        If not zero, add or remove radial thickness on the
                        gear negative involute. Default: 0.0
--second_gear_type SW_SECOND_GEAR_TYPE, --sgt SW_SECOND_GEAR_TYPE
                        Select the type of gear. Possible values: 'e', 'i',
                        'l'. Default: 'e'
--second_gear_tooth_nb SW_SECOND_GEAR_TOOTH_NB, --sgtn SW_SECOND_GEAR_TOOTH_NB
                        Set the number of teeth of the second gear_profile.
--second_gear_primitive_diameter SW_SECOND_GEAR_PRIMITIVE_DIAMETER, --sgpd SW_SECOND_GEAR_PRIMITIVE_D
                        If not zero, redefine the gear module to get this
                        primitive diameter of the second gear_profile.
                        Default: 0.0. If gearbar, it redefines the length.
--second_gear_addendum_dedendum_parity SW_SECOND_GEAR_ADDENDUM_DEDENDUM_PARITY, --sgadp SW_SECOND_GEA
                        Overwrite the addendum / dedendum parity of the second
                        gear_profile if different from 0.0. Default: 0.0%
--second_gear_tooth_half_height SW_SECOND_GEAR_TOOTH_HALF_HEIGHT, --sgthh SW_SECOND_GEAR_TOOTH_HALF_H
                        If not zero, redefine the tooth half height of the
                        second gear_profile. Default: 0.0
--second_gear_addendum_height_pourcentage SW_SECOND_GEAR_ADDENDUM_HEIGHT_POURCENTAGE, --sgahp SW_SECO
                        Set the addendum height of the second gear_profile in
                        pourcentage of the tooth half height. Default: 100.0%
--second_gear_dedendum_height_pourcentage SW_SECOND_GEAR_DEDENDUM_HEIGHT_POURCENTAGE, --sgdhp SW_SECO
                        Set the dedendum height of the second gear_profile in
                        pourcentage of the tooth half height. Default: 100.0%
--second_gear_hollow_height_pourcentage SW_SECOND_GEAR_HOLLOW_HEIGHT_POURCENTAGE, --sghhp SW_SECOND_G
                        Set the hollow height of the second gear_profile in
                        pourcentage of the tooth half height. The hollow is a
                        clear space for the top of the teeth of the other
                        gearwheel. Default: 25.0%
--second_gear_router_bit_radius SW_SECOND_GEAR_ROUTER_BIT_RADIUS, --sgrr SW_SECOND_GEAR_ROUTER_BIT_RA
                        If not zero, overwrite the router_bit radius used to
```

```
                        create the gear hollow of the second gear_profile.
                        Default: 0.0
--second_gear_base_diameter SW_SECOND_GEAR_BASE_DIAMETER, --sgbd SW_SECOND_GEAR_BASE_DIAMETER
                        If not zero, redefine the base diameter of the second
                        gear involute. Default: 0.0
--second_gear_tooth_resolution SW_SECOND_GEAR_TOOTH_RESOLUTION, --sgtr SW_SECOND_GEAR_TOOTH_RESOLUTIO
                        If not zero, it sets the number of segments of the
                        second gear involute. Default: 0
--second_gear_skin_thickness SW_SECOND_GEAR_SKIN_THICKNESS, --sgst SW_SECOND_GEAR_SKIN_THICKNESS
                        Add or remove radial thickness on the gear involute.
                        Default: 0.0
--second_gear_base_diameter_n SW_SECOND_GEAR_BASE_DIAMETER_N, --sgbdn SW_SECOND_GEAR_BASE_DIAMETER_N
                        If not zero, redefine the base diameter of the second
                        gear negative involute. Default: 0.0
--second_gear_tooth_resolution_n SW_SECOND_GEAR_TOOTH_RESOLUTION_N, --sgtrn SW_SECOND_GEAR_TOOTH_RESO
                        If not zero, it sets the number of segments of the
                        second gear negative involute. Default: 0
--second_gear_skin_thickness_n SW_SECOND_GEAR_SKIN_THICKNESS_N, --sgstn SW_SECOND_GEAR_SKIN_THICKNESS
                        If not zero, add or remove radial thickness on the
                        gear negative involute. Default: 0.0
--gearbar_slope SW_GEARBAR_SLOPE, --gbs SW_GEARBAR_SLOPE
                        if not zero, set the tooth slope angle for the
                        gearbar. Default 0.0
--gearbar_slope_n SW_GEARBAR_SLOPE_N, --gbsn SW_GEARBAR_SLOPE_N
                        if not zero, set the tooth negative slope angle for
                        the gearbar. Default 0.0
--center_position_x SW_CENTER_POSITION_X, --cpx SW_CENTER_POSITION_X
                        Set the x-position of the first gear_profile center.
                        Default: 0.0
--center_position_y SW_CENTER_POSITION_Y, --cpy SW_CENTER_POSITION_Y
                        Set the y-position of the first gear_profile center.
                        Default: 0.0
--gear_initial_angle SW_GEAR_INITIAL_ANGLE, --gia SW_GEAR_INITIAL_ANGLE
                        Set the gear reference angle (in Radian). Default: 0.0
--second_gear_position_angle SW_SECOND_GEAR_POSITION_ANGLE, --sgpa SW_SECOND_GEAR_POSITION_ANGLE
                        Angle in Radian that sets the postion on the second
                        gear_profile. Default: 0.0
--second_gear_additional_axis_length SW_SECOND_GEAR_ADDITIONAL_AXIS_LENGTH, --sgaal SW_SECOND_GEAR_AD
                        Set an additional value for the inter-axis length
                        between the first and the second gear_profiles.
                        Default: 0.0
--cut_portion SW_CUT_PORTION SW_CUT_PORTION SW_CUT_PORTION, --cp SW_CUT_PORTION SW_CUT_PORTION SW_CUT
                        (N, first_end, last_end) If N>1, cut a portion of N
                        tooth ofthe gear_profile. first_end and last_end
                        defines in details where the profile stop (0: slope-
                        top, 1: top-middle, 2: slope-bottom, 3: hollow-
                        middle). Default: (0,0,0)
--gear_profile_height SW_GEAR_PROFILE_HEIGHT, --gwh SW_GEAR_PROFILE_HEIGHT
                        Set the height of the linear extrusion of the first
                        gear_profile. Default: 1.0
--simulation_enable, --se
                        It display a Tk window where you can observe the gear
                        running. Check with your eyes if the geometry is
                        working.
--output_file_basename SW_OUTPUT_FILE_BASENAME, --ofb SW_OUTPUT_FILE_BASENAME
                        If not the empty_string (the default value), it
                        outputs the (first) gear in file(s) depending on your
                        argument file_extension: .dxf uses mozman dxfwrite,
```

```
                        .svg uses mozman svgwrite, no-extension uses FreeCAD
                        and you get .brep and .dxf
--run_self_test, --rst
                        Generate several corner cases of parameter sets and
                        display the Tk window where you should check the gear
                        running.
```

16.3 From gear_profile() arguments to high-level parameters

16.3.1 Gear type

Gear type possible values:

```
- e : external (a.k.a. gearwheel)
- i : internal (a.k.a. gearring)
- l : linear (a.k.a. gearbar)
```

16.3.2 Gear tooth number (N)

```
N > 2
```

16.3.3 Gear module (m)

Set after those priorities:

```
1. gear-module parameter
2. primitive diameter parameter (m=pd/N)
3. second primitive diameter parameter (m=pd2/N2)
4. the default value (m=1)
```

16.3.4 Gear base diameter (bd)

Set after those priorities:

```
1. gear base diameter parameter
2. second gear base diameter parameter (bd=bd2*N1/N2)
3. gearbar slope angle (bd=pd*cos(sa))
4. force angle parameter (bd=pd*cos(fa))
5. the default value (bd=[dedendum diameter of the smallest gear])
```

When two gears are specified (by setting second_gear_tooth_nb), and the gear base diameter is not constrainted, the dedendum diameter of the smallest gear is used to calculate the gear base diameter.

16.3.5 Gearbar slope angle (sa)

It is only applicable with a gearbar. Because gearbar-gearbar system doesn't exist, the first and the second gear share the parameters *gearbar_slope* and *gearbar_slope_n*.

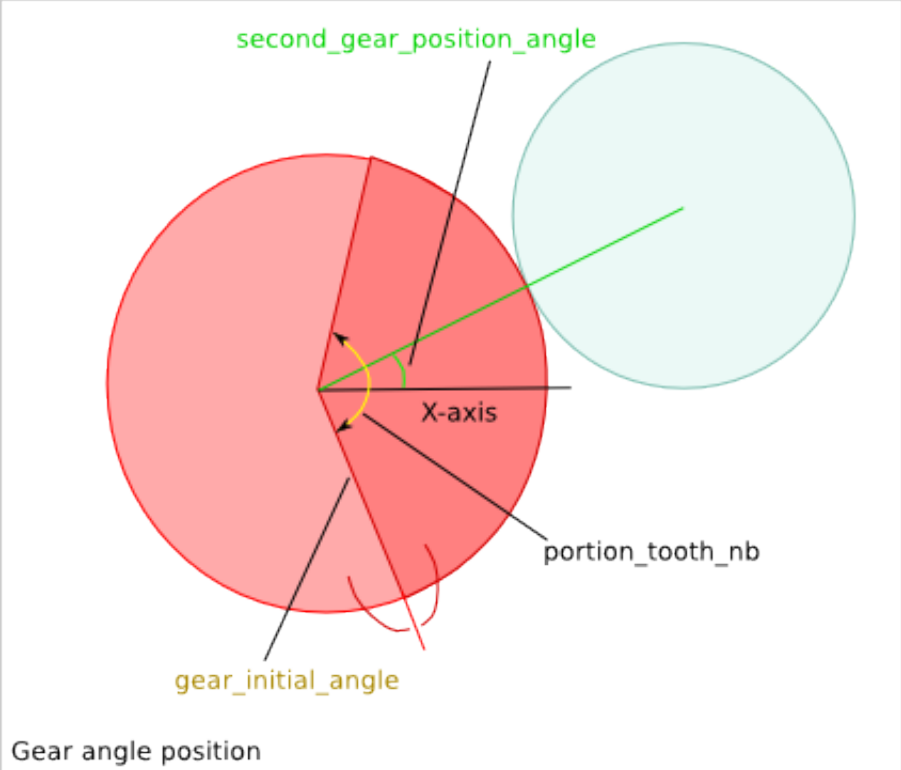
Set after those priorities:

```
1. gearbar_slope parameter
2. force angle parameter (sa=fa)
3. second gear base diameter parameter (sa=acos(bd/pd))
```

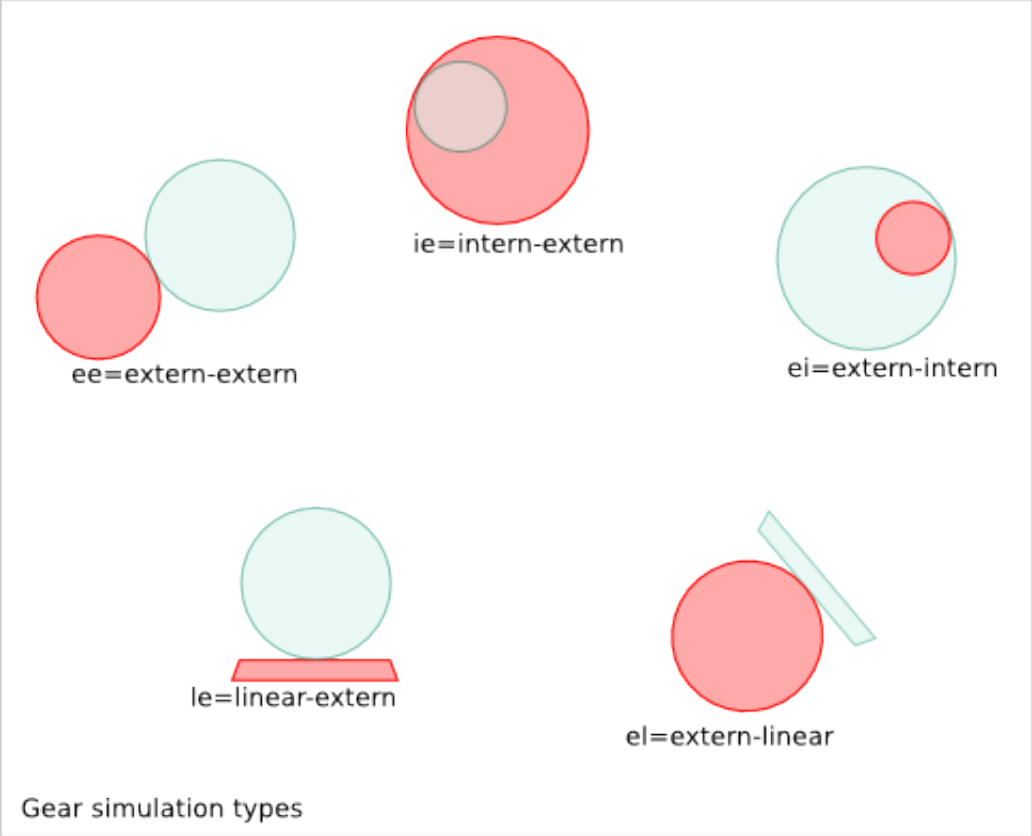
The Gearbar slope has no default value and must be constraint by one of those three possibilities.

16.4 Complement on gear high-level parameters

16.4.1 Gearwheel angle position



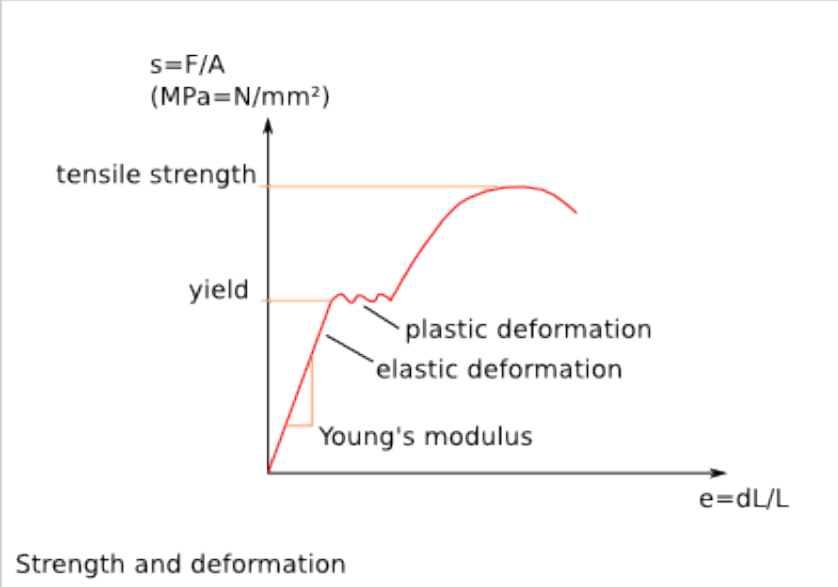
16.4.2 Simluation cases



CHAPTER 17

Gear Guidelines

17.1 Strength and deformation



Strength and deformation

17.2 Gear module

The *gear module* defines the size of a gear tooth. Two gearwheels working together must have the same *gear module*:

```
circular_pitch = Pi * gear_module
```

Per default, the *tooth height* is defined with the *gear module*:

```
addendum_height = gear_module
dedendum_height = gear_module
tooth_height = 2 * gear_module
```

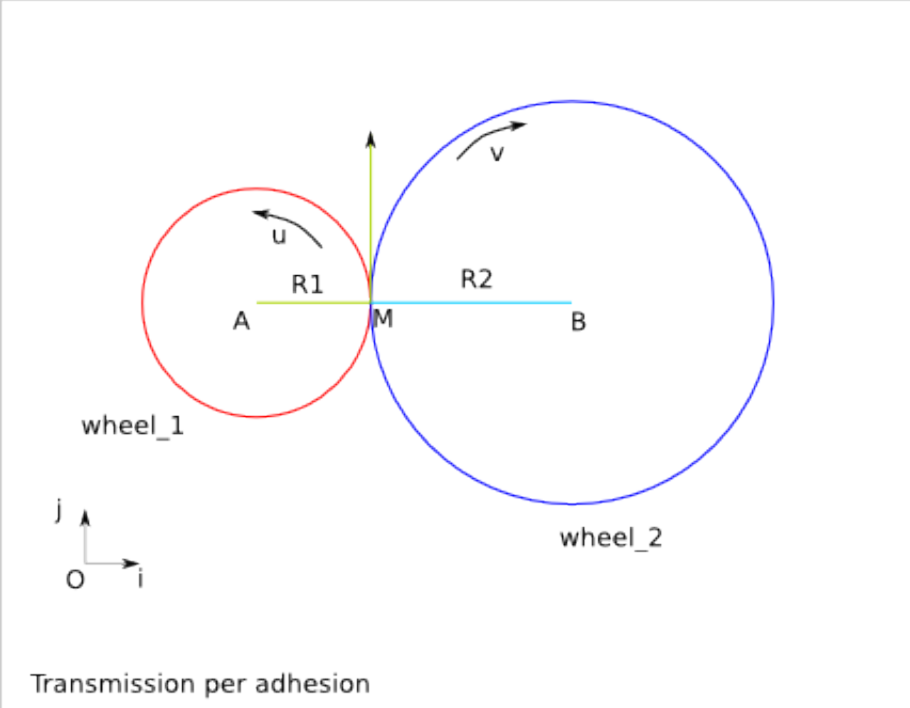
A small *gear module* generates less friction and then provides a better energy transmission efficiency. A large *gear module* supports higher efforts:

```
Module sizing formula in the literaure:
m>2.34*sqrt(T/(k*Rpe))
with:
T = tangential effort on the tooth = F*cos(a)
torque = C = d*F
d = R*cos(a) (R = primitive radius = Z*m/2)
T = F*cos(a) = C/d*cos(a) = C/R
k = tooth width coefficient (usually k=8 or 10)
tooth width = b = k*m
Rpe = Re/s
Re = Yield = elasticity limit
s = security coefficient
```

CHAPTER 18

Gear Profile Theory

18.1 Transmission per adhesion




```
(O,i,j) orthonormal reference frame

wheel_1 rotation speed: u (radian/s)
wheel_2 rotation speed: v (radian/s)
```

speed of M, point of wheel_1:

```
V(M) = u*R1*j
```

speed of N, point of wheel_2:

```
V(N) = v*R2*j
```

Because of the adhesion of the wheel_1 and wheel_2 in M:

```
V(M).j = V(N).j
u*R1 = v*R2
v = u*R1/R2
```

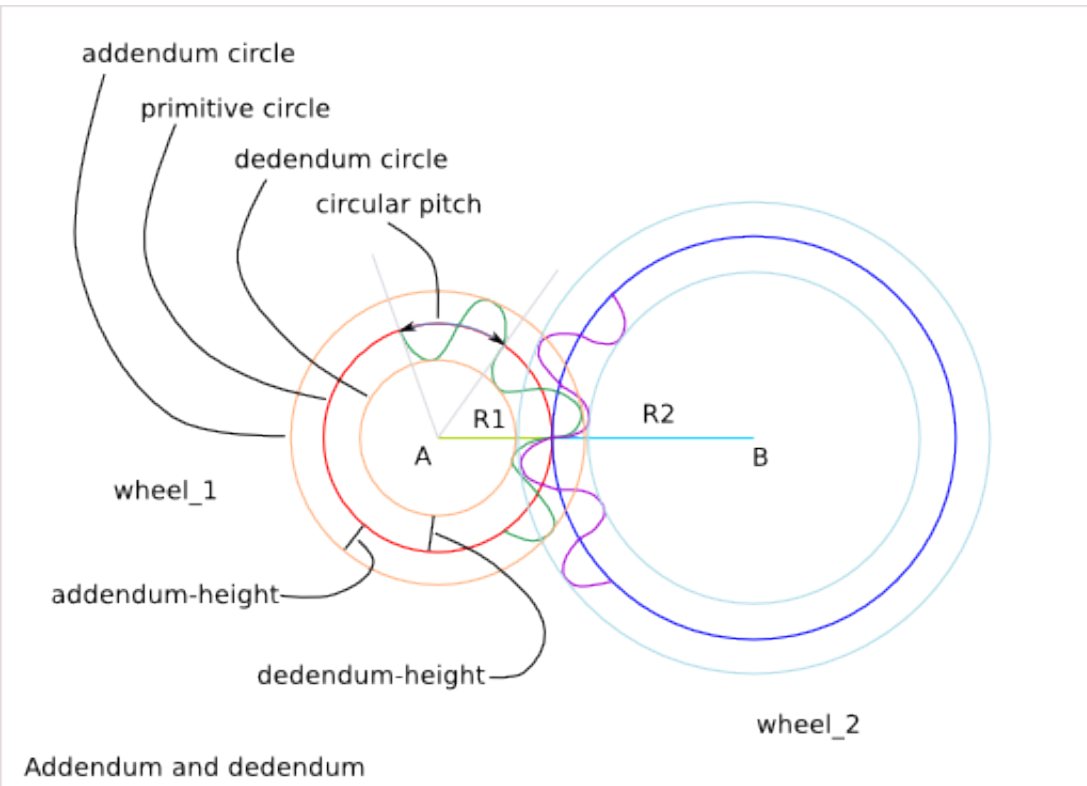
18.1.1 Issue

The maximal torque transmission is limited by the adhesion capacity.

18.1.2 Idea

Create hollows and bums around the wheel to get a contact point force transmission.

18.2 Transmission with teeth



18.2.1 One wheel description

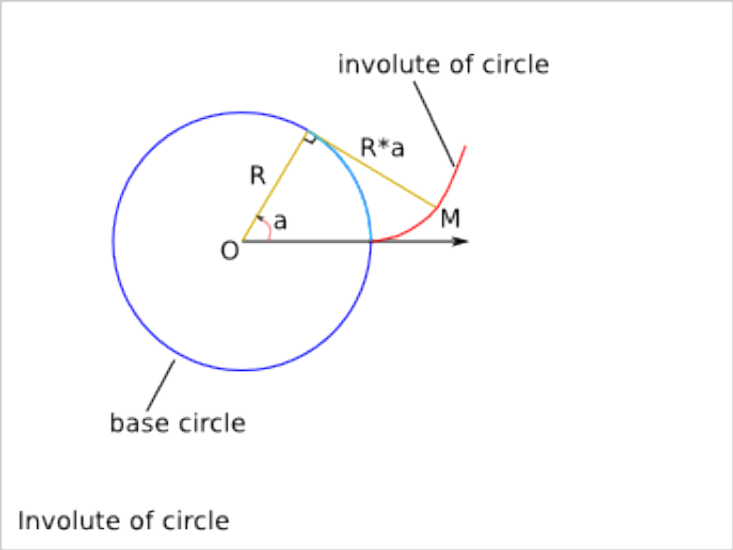
```
angular_pitch = 2*pi/tooth_nb
circular_pitch = angular_pitch * primitive radius
addendum_radius = primitive_radius + addendum_height
dedendum_radius = primitive_radius + dedendum_height
tooth_height = addendum_height + dedendum_height
```

18.2.2 Conditions for working gear

```
circular_pitch_1 = circular_pitch_2
addendum_height_1 < dedendum_height_2
addendum_height_2 < dedendum_height_1
transmission ratio = primitive_radius_1 / primitive_radius_2 = tooth_nb_1 / tooth_nb_2
```

Problematic: How to design the tooth-profile?

18.3 Tooth profile



Cartesian equation:

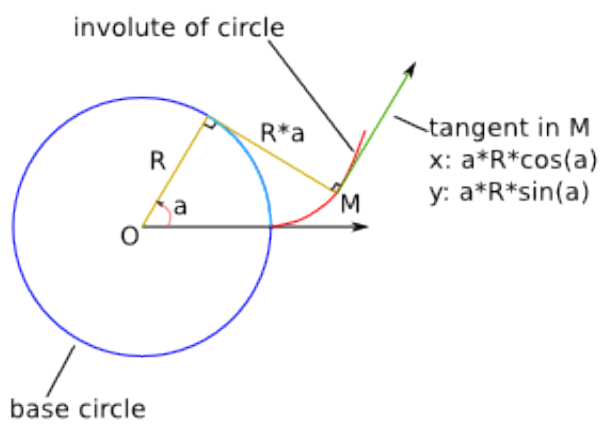
```
Mx(a) = R*cos(a)+a*R*cos(a-pi/2)
My(a) = R*sin(a)+a*R*sin(a-pi/2)
```

Trigonometry formula remind:

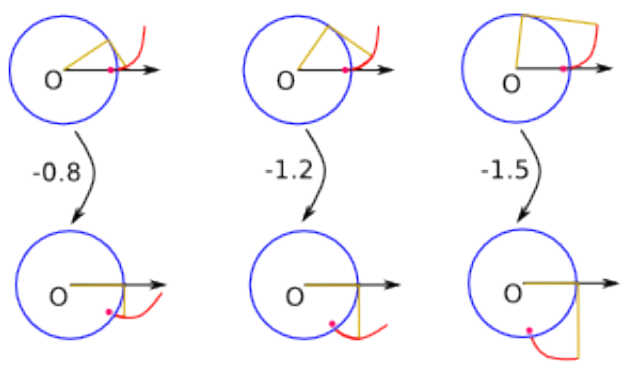
```
cos(-x) = cos(x)
sin(-x) = -sin(x)
cos(pi/2-x) = sin(x)
sin(pi/2-x) = cos(x)
cos(a-pi/2)=cos(pi/2-a)=sin(a)
sin(a-pi/2)=-sin(pi/2-a)=-cos(a)
```

Tangent vector:

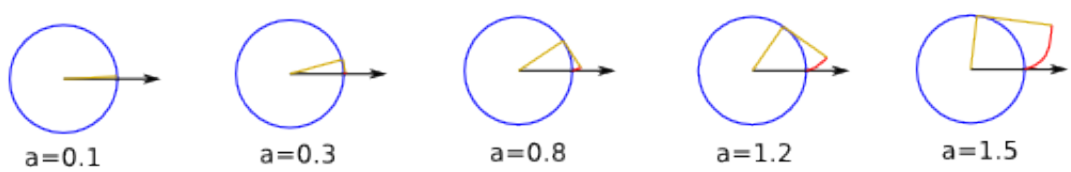
```
Mx'(a) = -R*sin(a)+R*cos(a-pi/2)-a*R*sin(a-pi/2) = -a*R*sin(a-pi/2) = a*R*cos(a)
My'(a) =  R*cos(a)+R*sin(a-pi/2)+a*R*cos(a-pi/2) =  a*R*cos(a-pi/2) = a*R*sin(a)
```



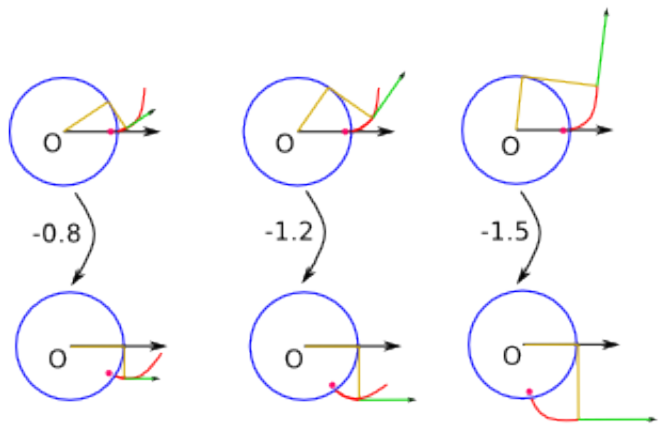
Involute of circle and tangent



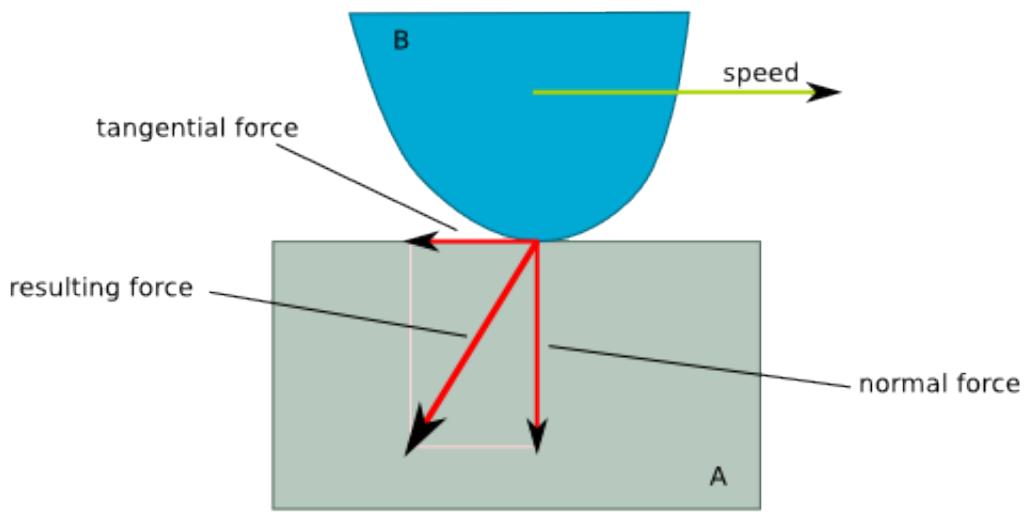
Rotation of center O



Involute of circle evolution

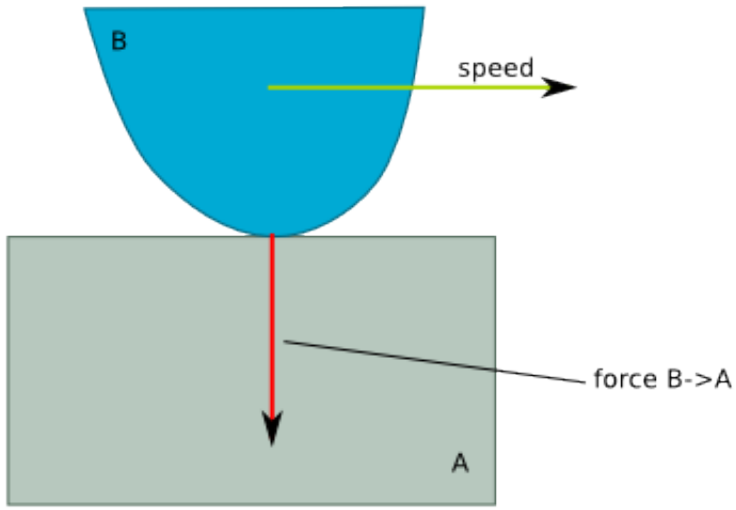


Rotation of center O with tangent



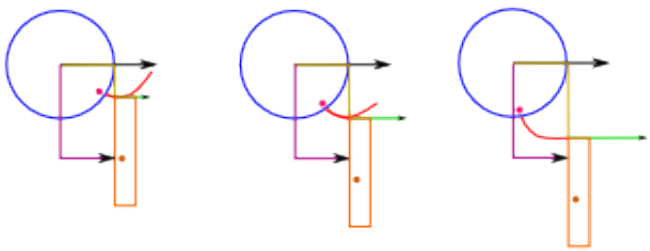
Coulomb's law of friction:
tangential_force = friction_coefficient * normal_force

Realistic ponctual contact

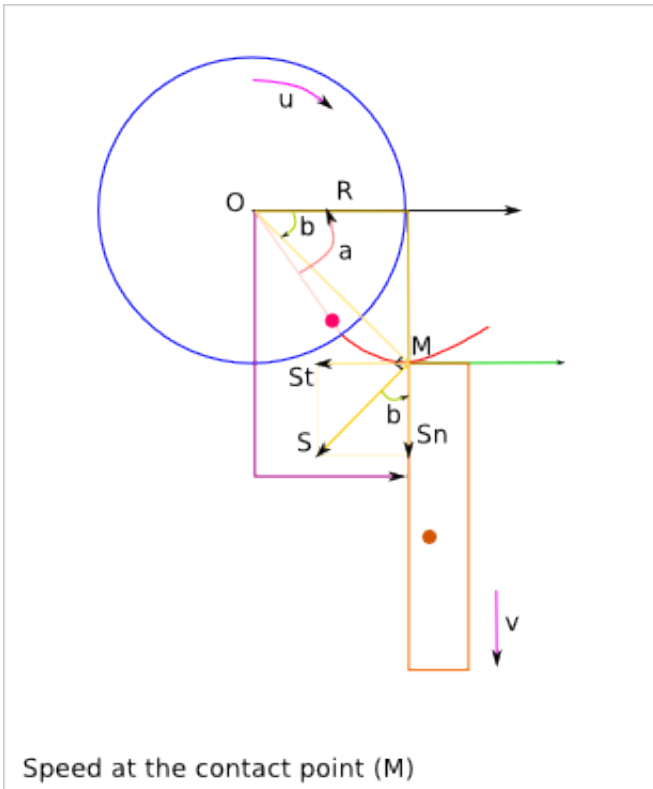


friction_coefficient = 0
The force of B on A is normal to the surface contact

Ideal ponctual contact



Involute of circle translating a bar




```
u: rotation speed of the wheel
v: linear speed of tha bar
u(t) = d/dt(a(t))

OM = sqrt(R² + (a*R)²) = R*sqrt(1+a²)

S = OM*u
Sn = S*cos(b)
St = S*sin(b)

Sn = u*R*sqrt(1+a²)*cos(b)
relation between a(t) and b(t)?
tan(b) = (a*R)/R = a
Sn = u*R*sqrt(1+tan²(b))*cos(b)
```

Trigonometry formula remind:

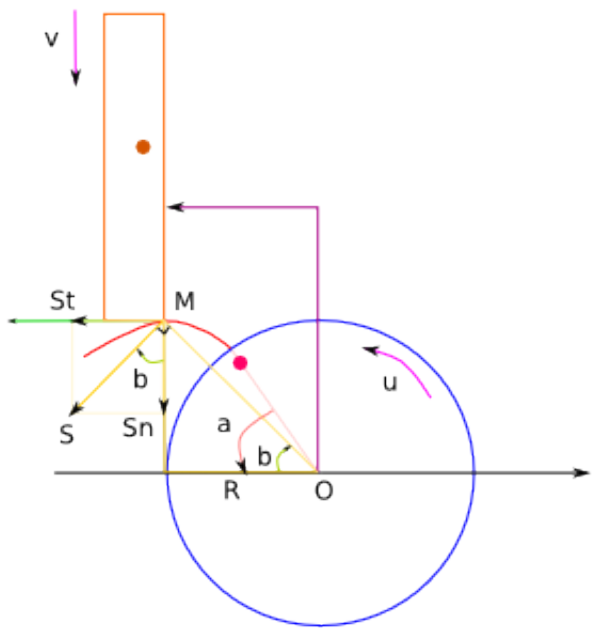
```
1+tan²(x) = (cos²(x)+sin²(x))/cos²(x) = 1/cos²(x)
```

So,:

```
v = Sn = u*R
```

v does not depend on the angle a!

```
St = u*R*sqrt(1+a²)*sin(b) = u*R*tan(b) = u*R*a
```

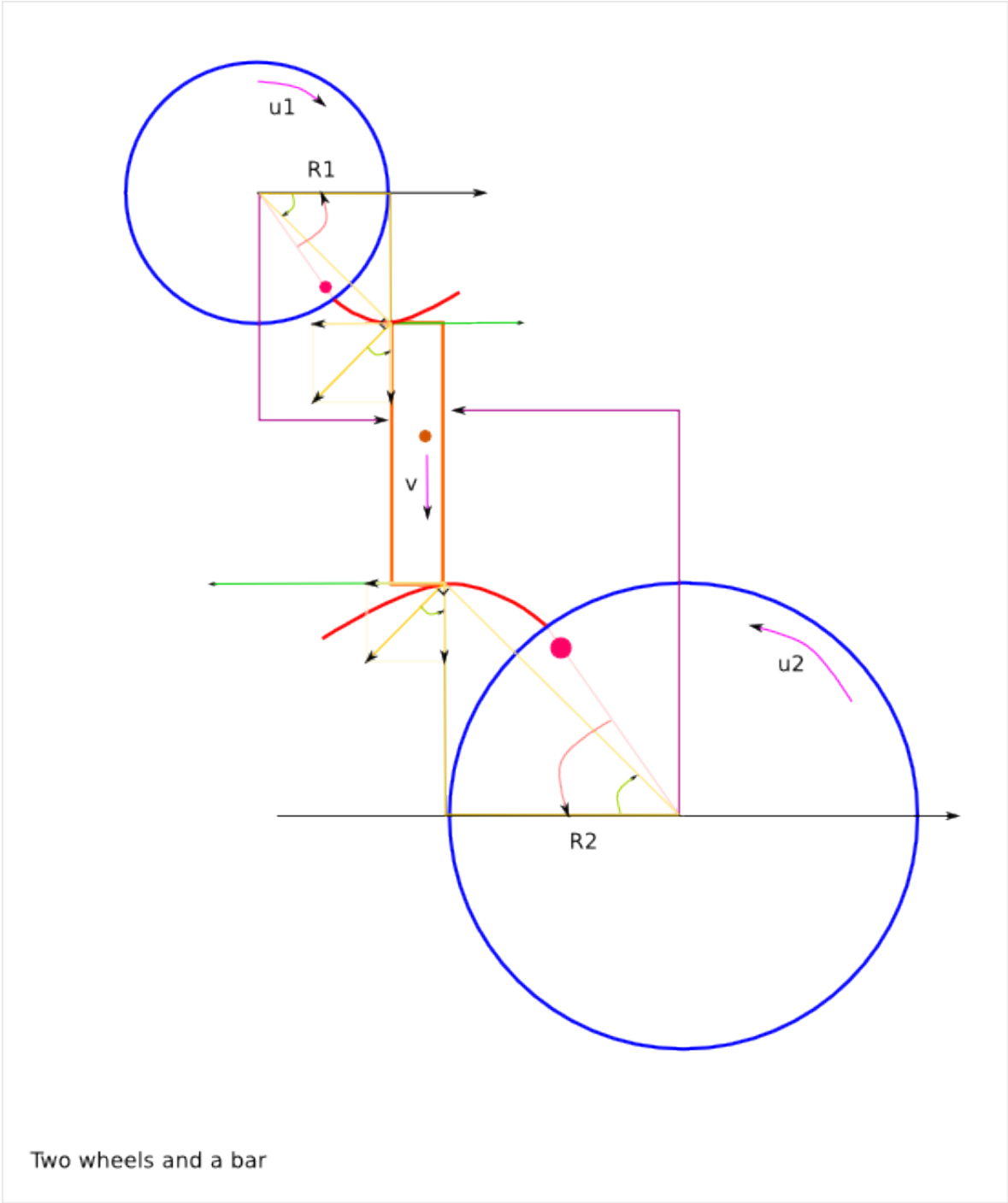


Inversed speed and new point of view

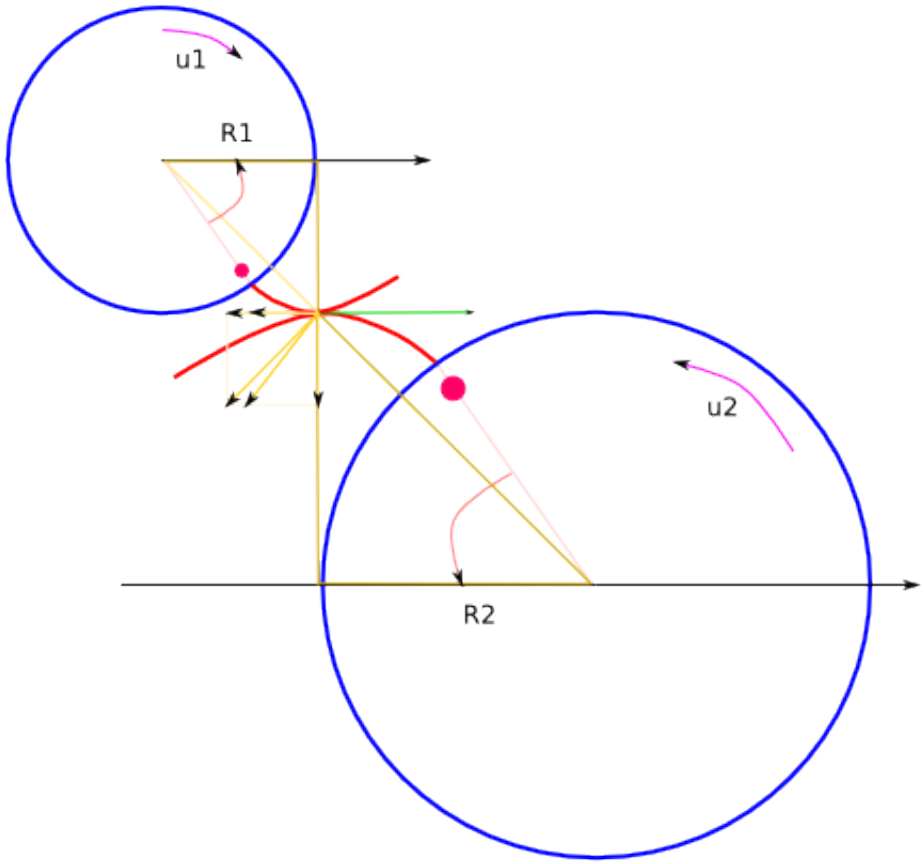
```
u: rotation speed of the wheel
v: linear speed of tha bar
u(t) = d/dt(a(t))

OM = sqrt(R² + (a*R)²) = R*sqrt(1+a²)

S = OM*u
Sn = S*cos(b)
St = S*sin(b)

v = Sn = u*R*sqrt(1+a²)*cos(b)
= u*R*sqrt(1+tan²(b))*cos(b) = u*R
```

v does not depend on the angle a!

```
St = u*R*sqrt(1+a²)*sin(b) = u*R*tan(b) = u*R*a
```

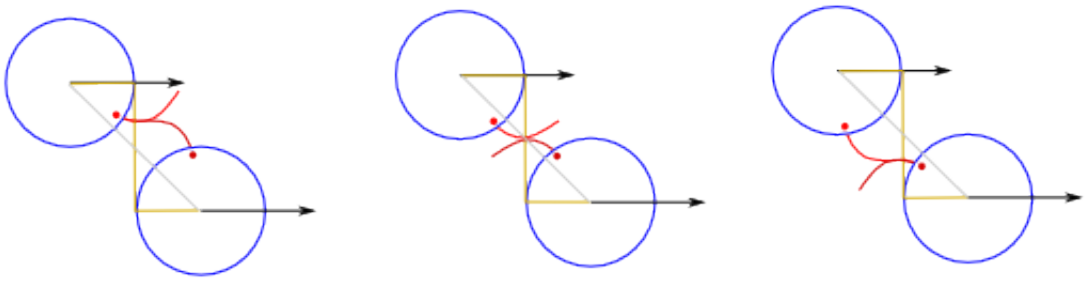



```
v = u1*R1 = u2*R2
So, u2 = u1*R1/R2
```

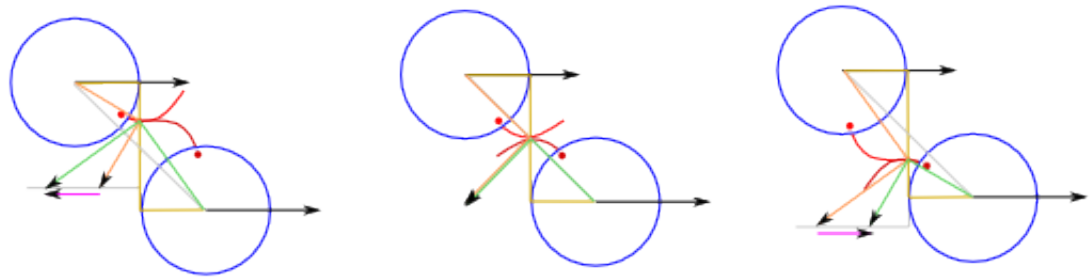


Two wheels

Sn1 = Sn2 because of the contact



Two wheel evolution

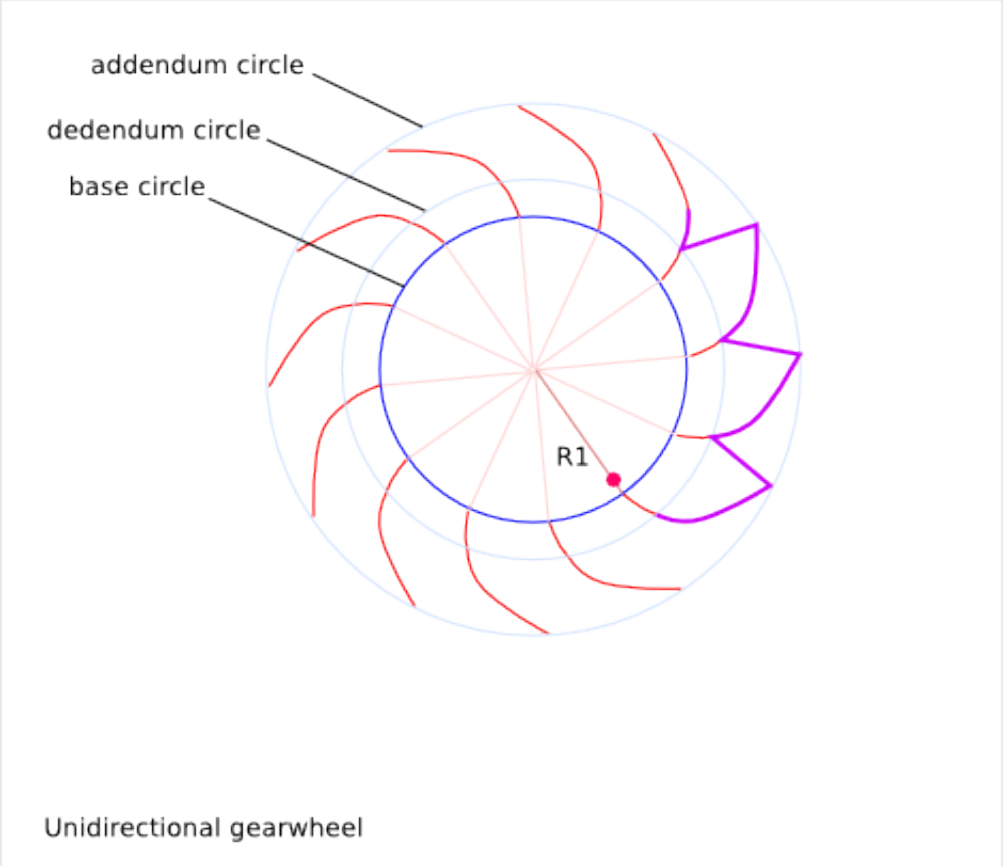


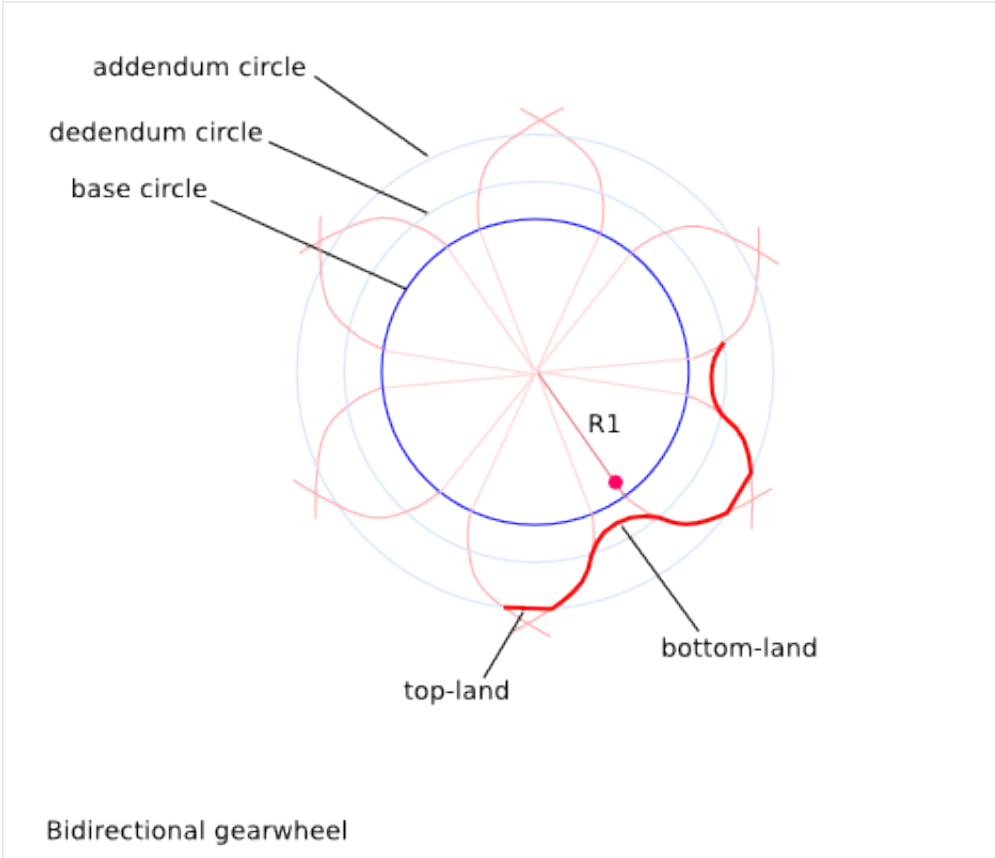
Two wheel evolution with speed vectors

Friction between the two wheels:

```
Sf = St2 - St1 = u2*R2*a2 - u1*R1*a1
= u1*R1*(a2-a1)
But,
a1 = k1-u1*t
a2 = k2+u2*t
Sf = u1*R1*(k1-k2+(u1+u2)*t)
```

18.4 Gear profile construction





18.5 Gear rules

- The base diameter of the two directions can be different
- **The top-land and bottom-land are not critical part of the profile** The top-land can be a straight line. The bottom-land is usually a hollow to help the manufacturing.
- The rotation ratio implies by the involutes-of-circles is:

```
base_radius_1 / base_radius_2
```

The rotation ratio implies by the teeth is:

```
tooth_nb_1 / tooth_nb_2
```

In order to get a continuous transmission without cough, we must ensure that:

```
base_radius_1 / base_radius_2 = tooth_nb_1 / tooth_nb_2
```

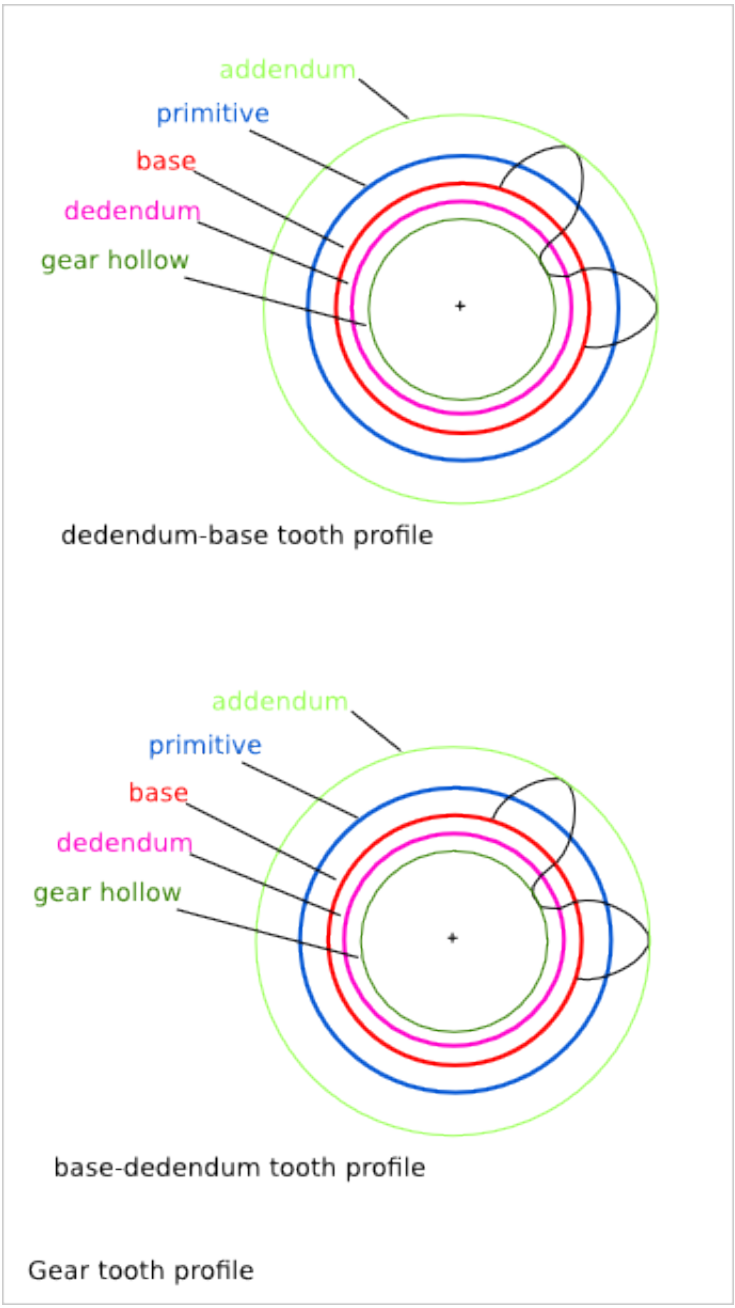
If you use two base circles for the positive rotation and the negative rotation, then:

```
base_radius_positive_1 / base_radius_positive_2 = tooth_nb_1 / tooth_nb_2
base_radius_negative_1 / base_radius_negative_2 = tooth_nb_1 / tooth_nb_2
```

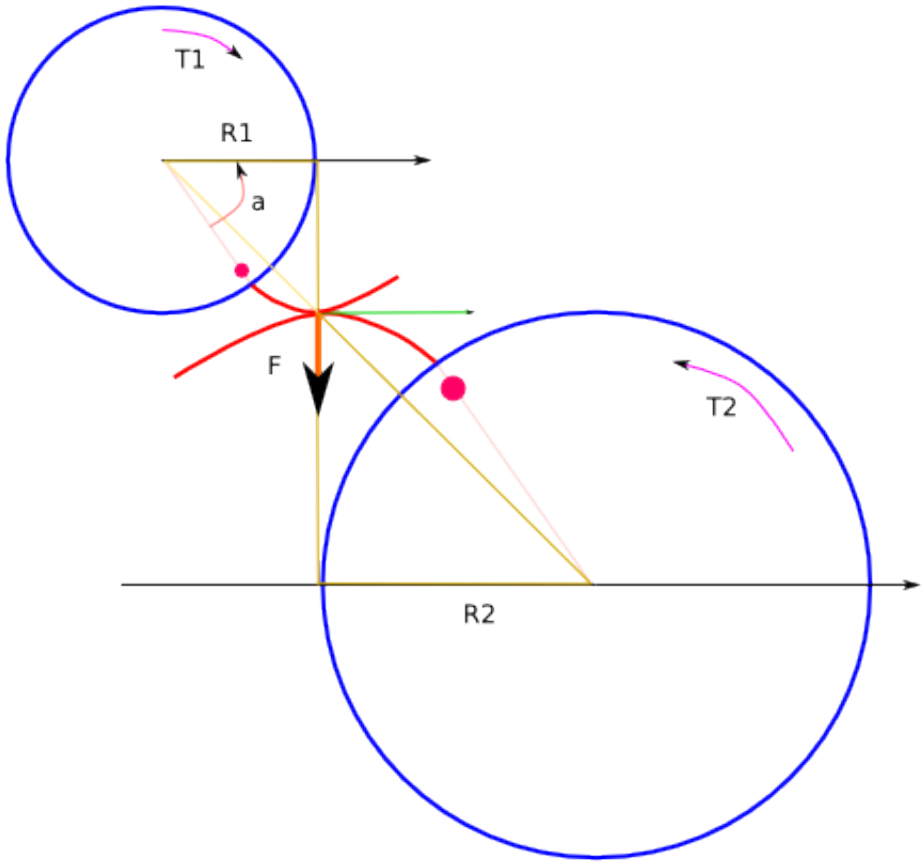
- The position of the positive involute of circle compare to the negative involute of circle is arbitrary and it is usually defined by the addendum-dedendum-ration on the primitive circle. Just make sure the top-land and bottom-land still exist (positive length). The addendum-dedendum-ration of the second wheel must be the complementary.

Do not mix-up the *primitive circle* and the *base circle*. The *primitive circle* helps defining the *addendum* and *dedendum* circles. The *base circle* defines the *involutes of circle*.We have the relation:

```
base_radius < primitive_radius
```



18.6 Torque transmission

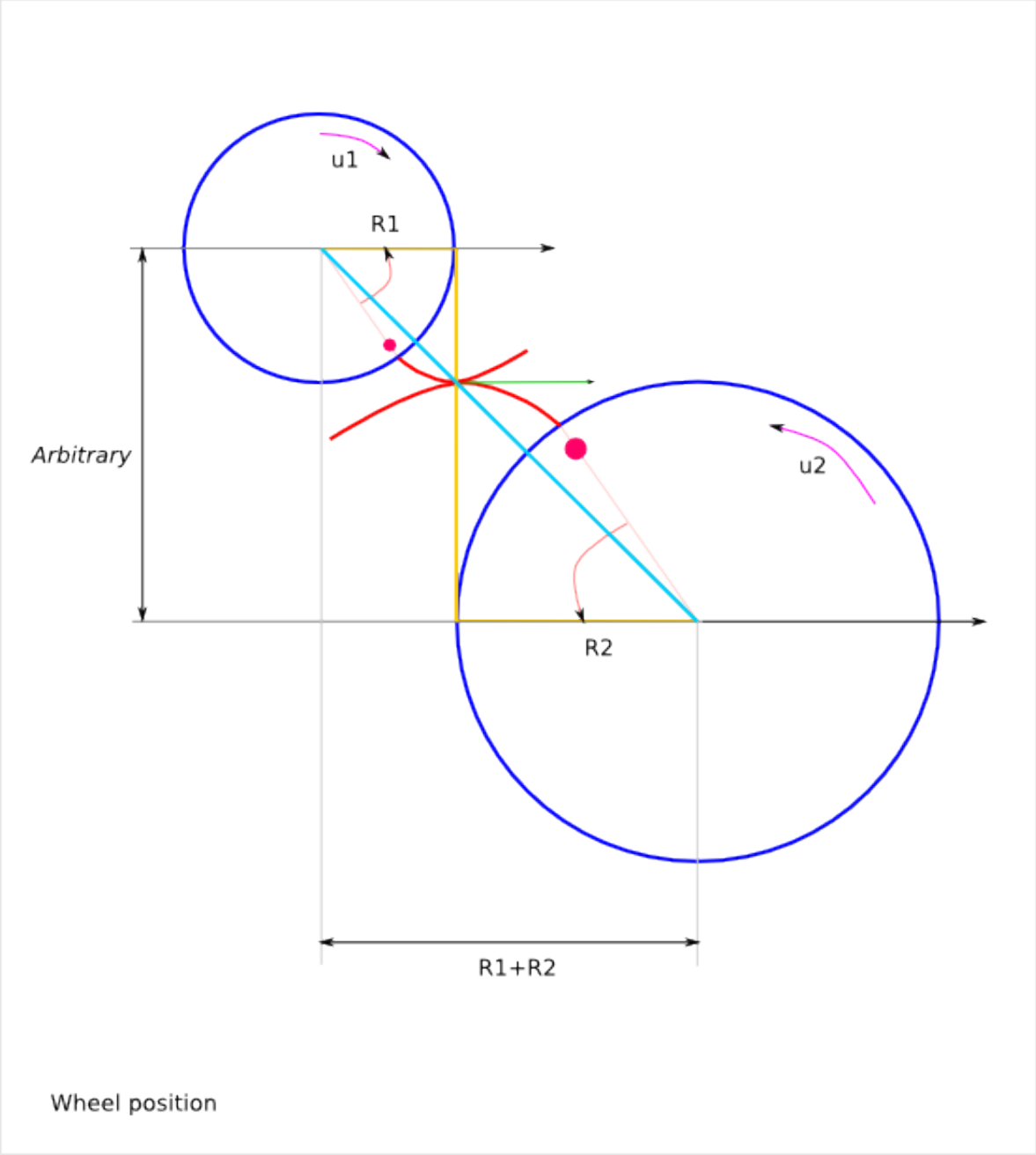


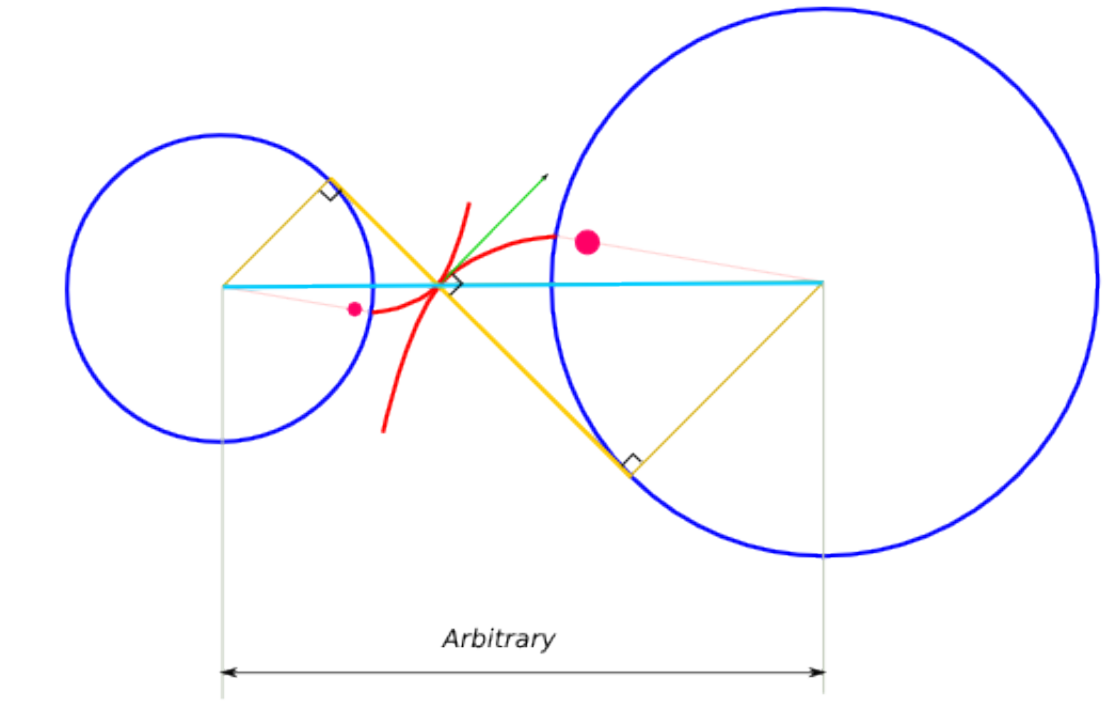
Torque transmission

```
F = T1/R1 = T2/R2
T2 = T1*R2/R1
```

The transmitted torque T2 does not depend on the angle a!

18.7 Gearwheel position





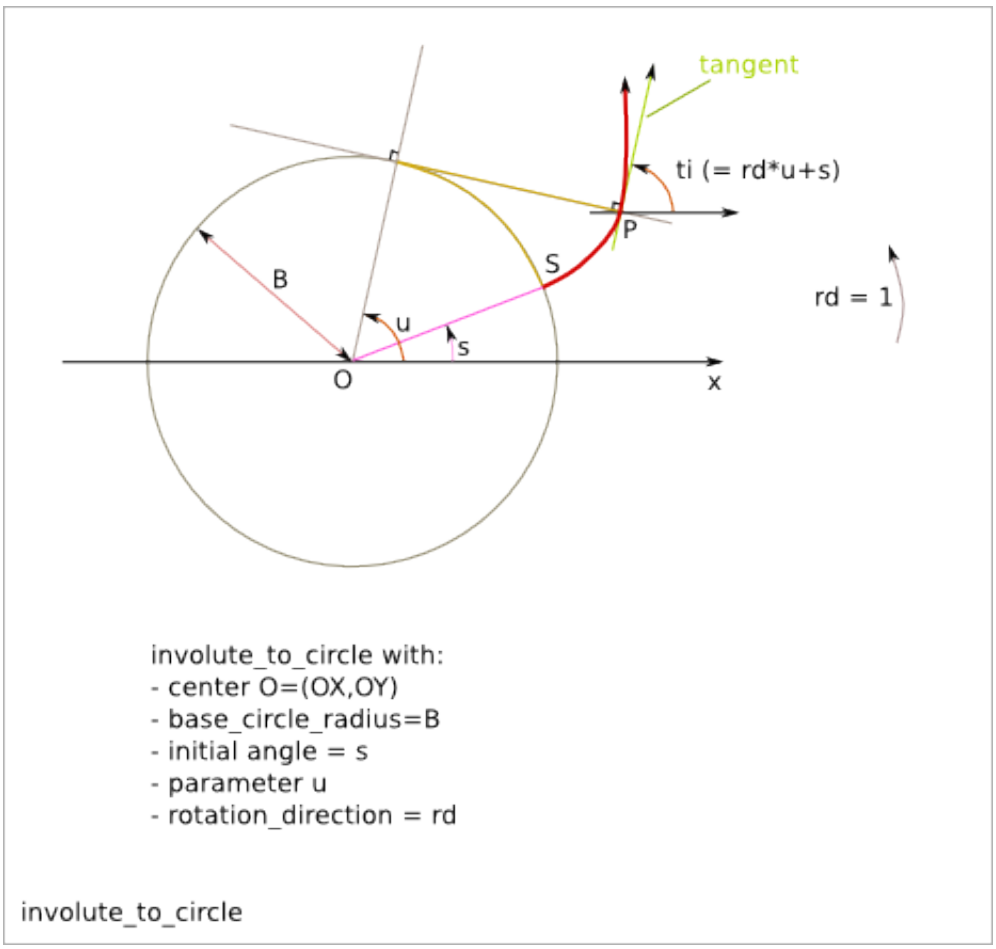
Wheel position with horizontal inter-axis

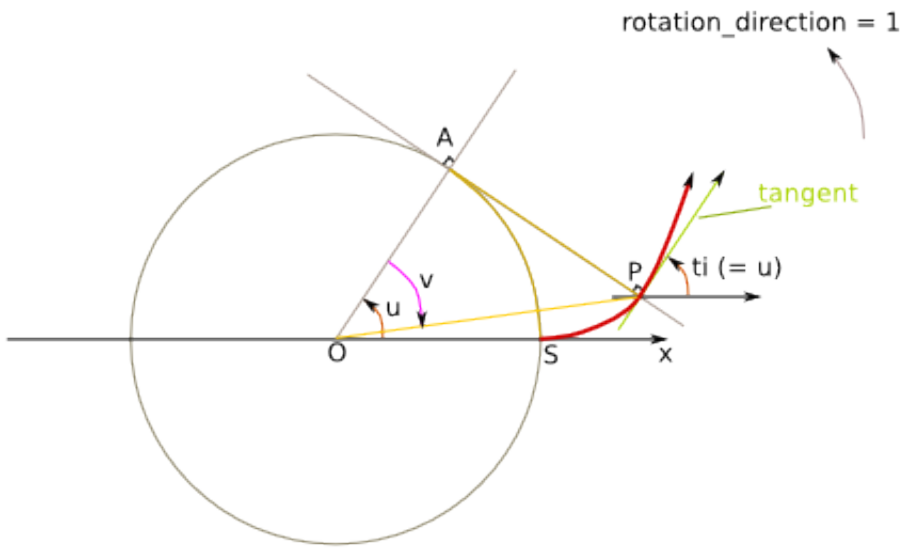
The rotation ration depends only on the two base circle diameters. It does not depend on the inter-axis length. The inter-axis length can be set arbitrary within a reasonable range (addendum and dedendum height constraints).

CHAPTER 19

Gear Profile Details

19.1 Involute of circle

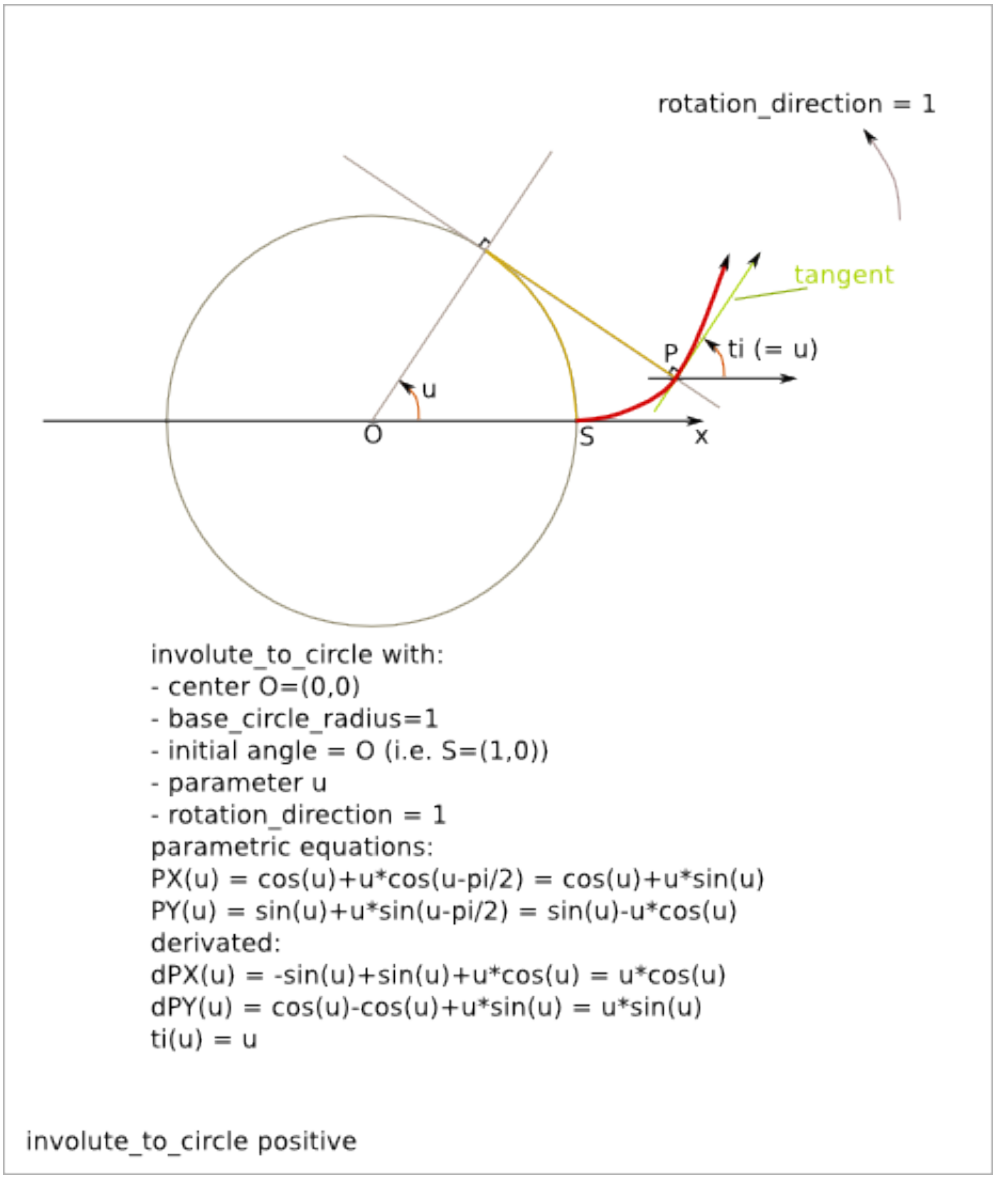


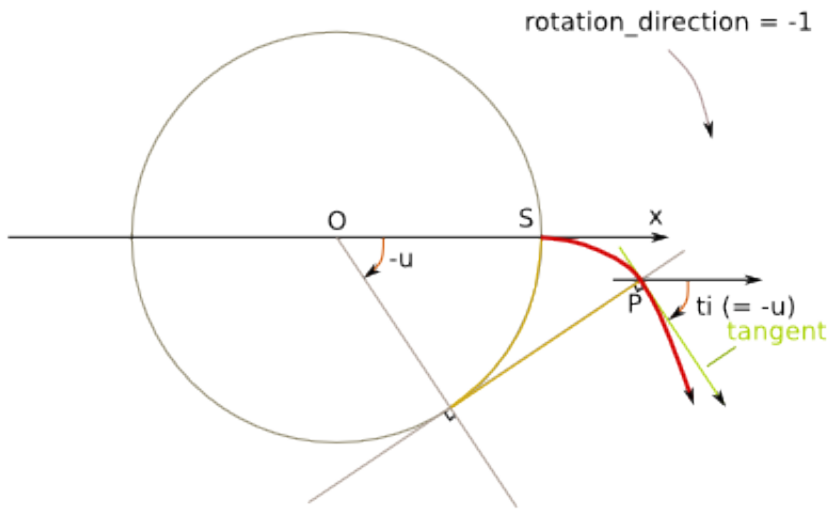


PX(u) = cos(u)+u*cos(u-pi/2) = cos(u)+u*sin(u)
PY(u) = sin(u)+u*sin(u-pi/2) = sin(u)-u*cos(u)
ti(u) = u

v=AOP
tan(v) = AP/OA = u
xOP = u-v = u-atan(u)
OP = OA*sqrt(1+u^2)
u = sqrt(OP^2/OA^2-1)
xOP = s+rd*(u-atan(u))

Involute_to_circle relations





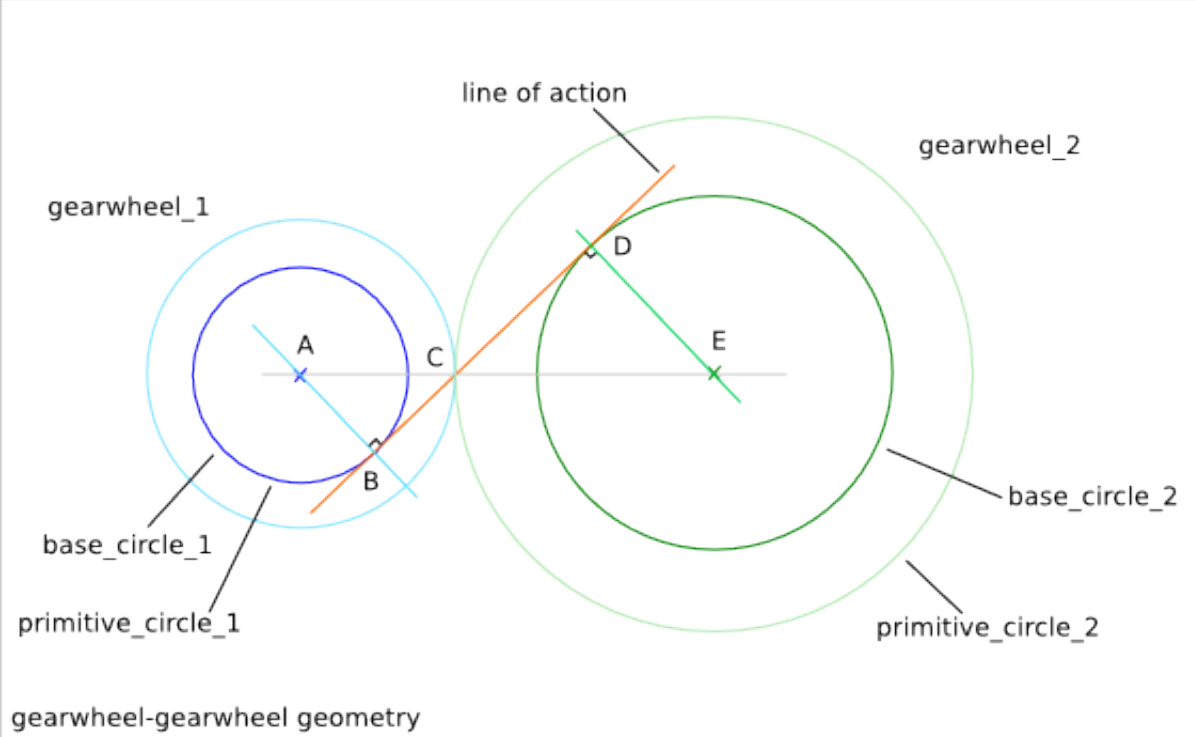
involute_to_circle with:
- center O=(0,0)
- base_circle_radius=1
- initial angle = O (i.e. S=(1,0))
- parameter u
- rotation_direction = -1

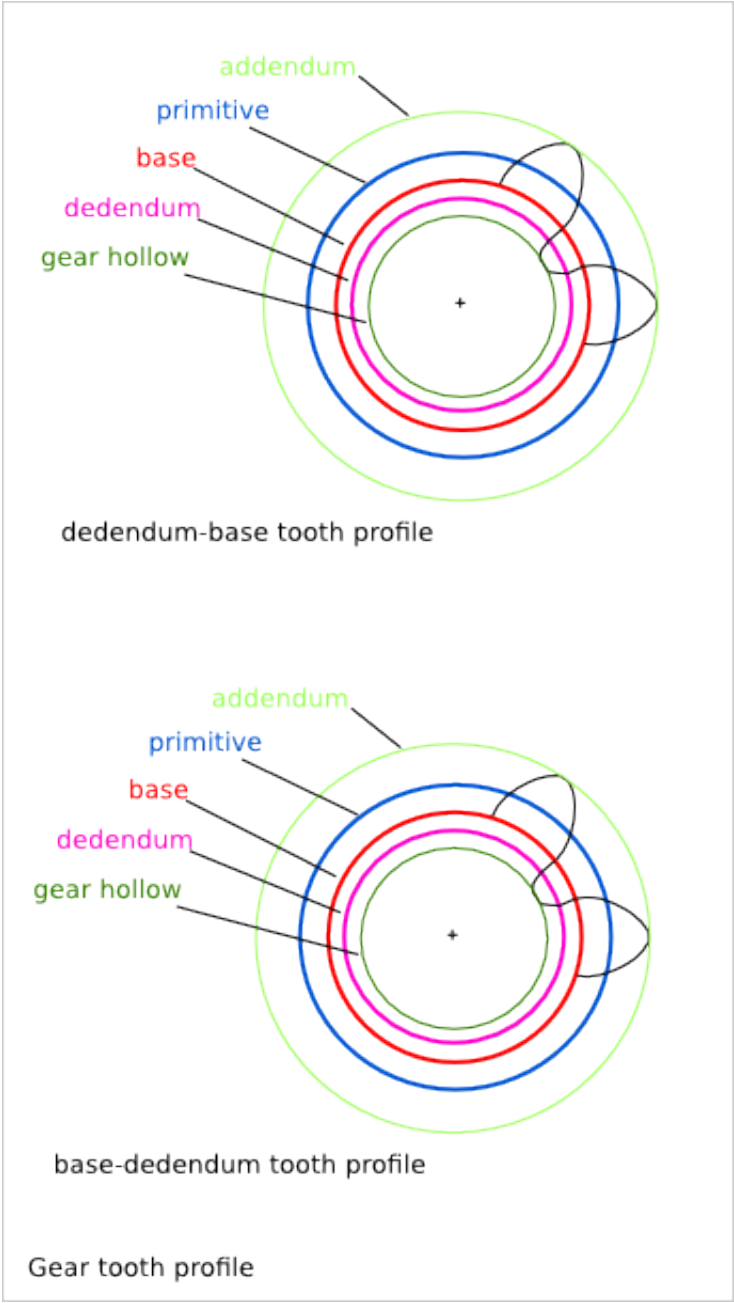
parametric equations:
PX(u) = cos(-u)+u*cos(-u+pi/2) = cos(u)+u*sin(u)
PY(u) = sin(-u)+u*sin(-u+pi/2) = -sin(u)+u*cos(u)
derivated:
dPX(u) = -sin(u)+sin(u)+u*cos(u) = u*cos(u)
dPY(u) = -cos(u)+cos(u)-u*sin(u)) = -u*sin(u)
ti(u) = -u

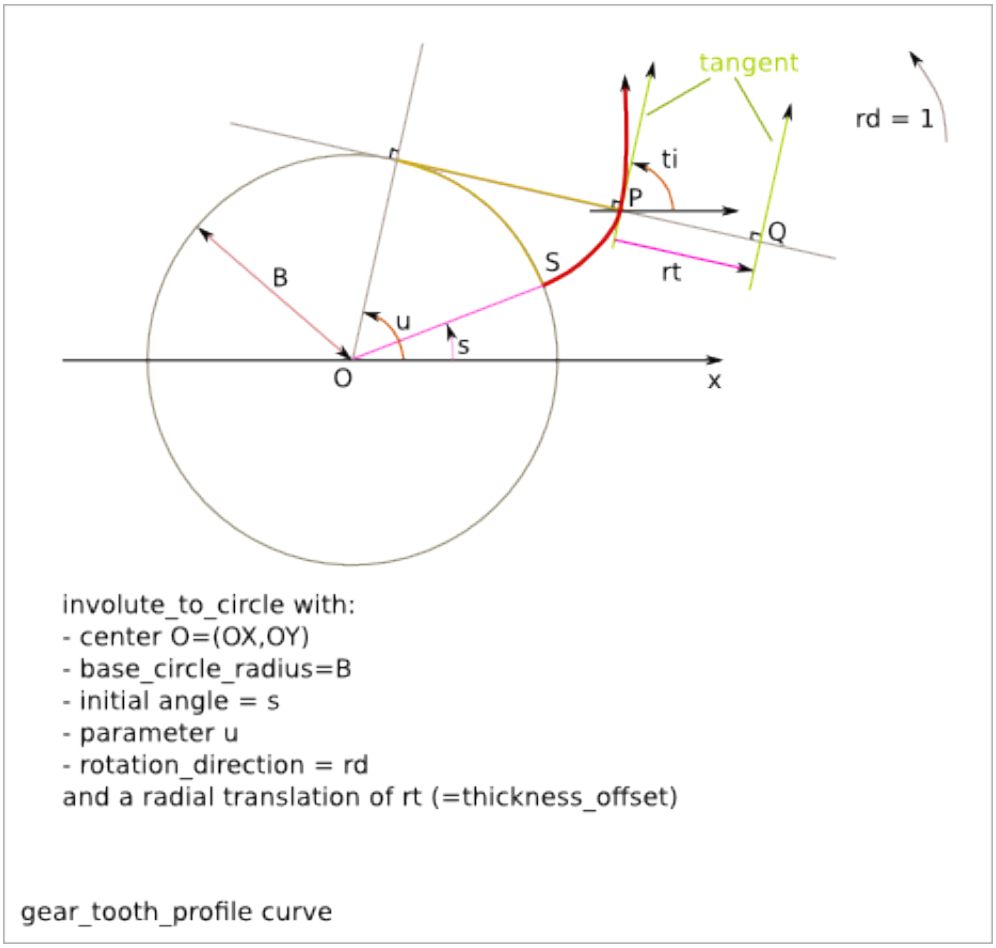
involute_to_circle negative

19.2 Gear outline

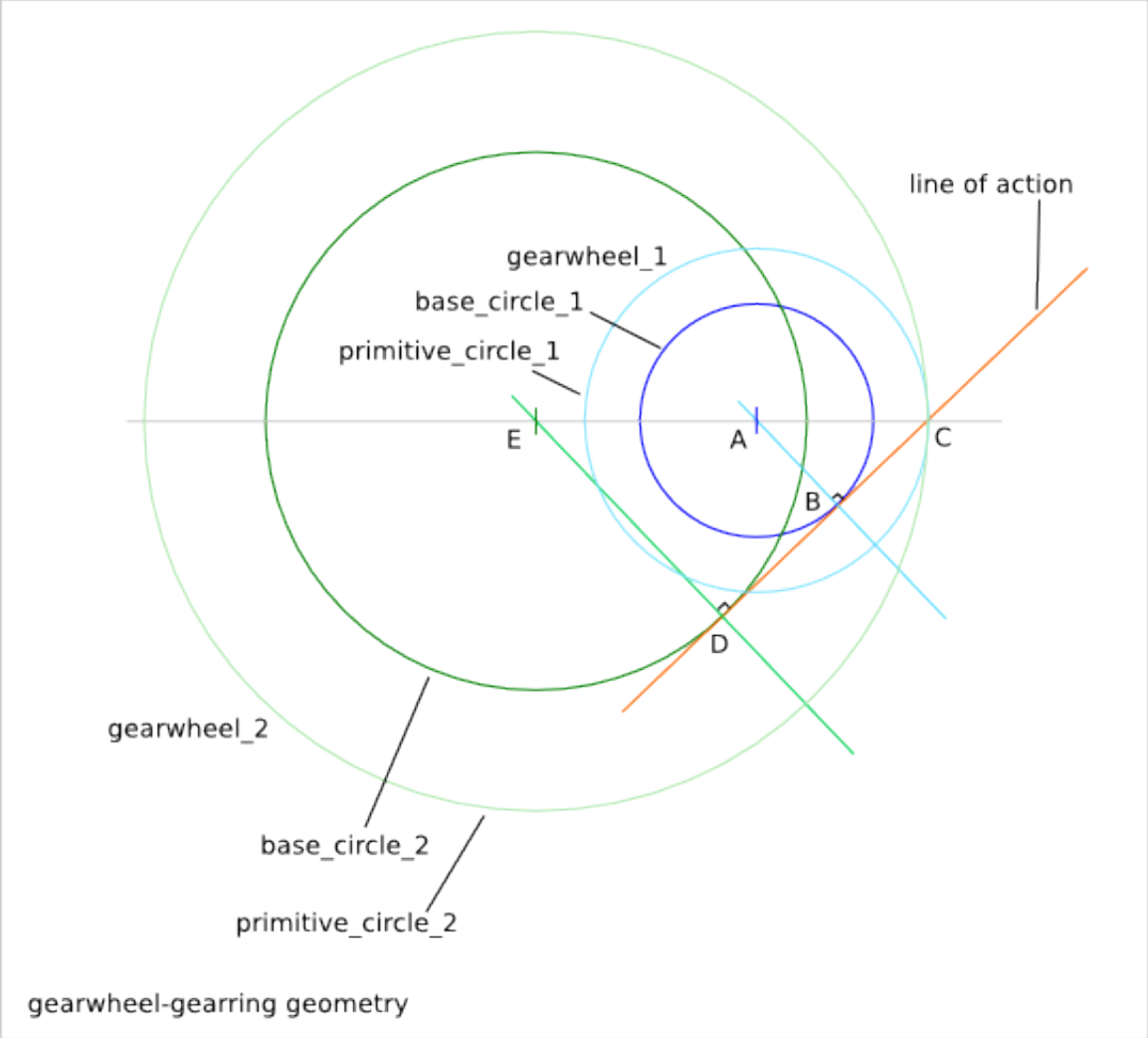
19.2.1 Gearwheel



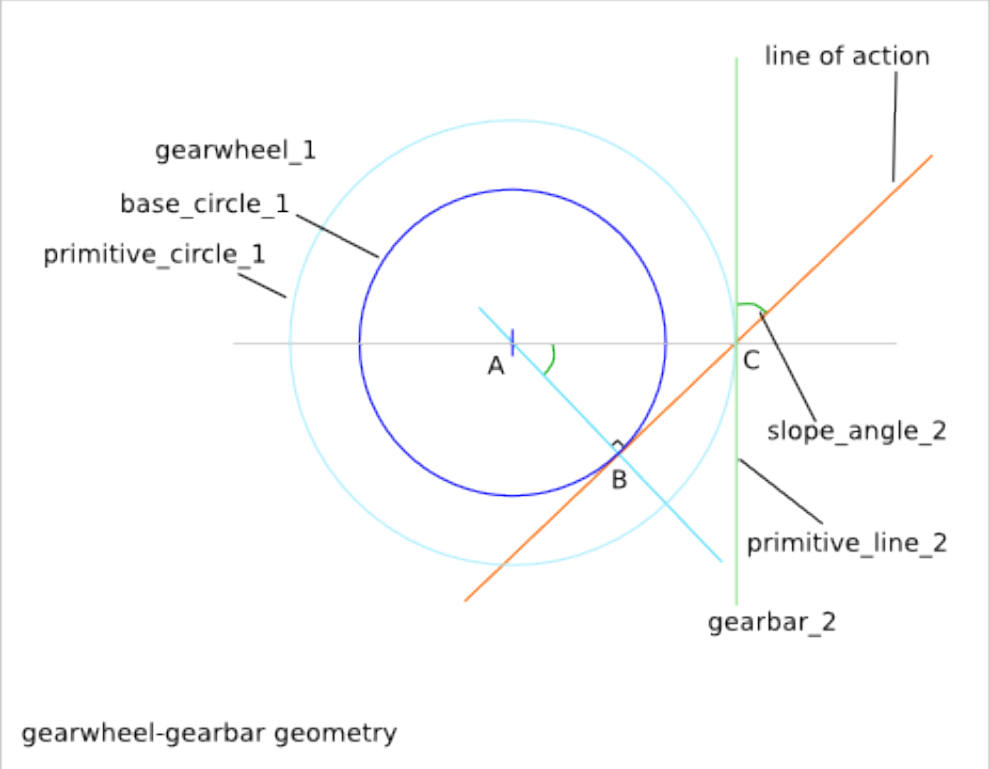


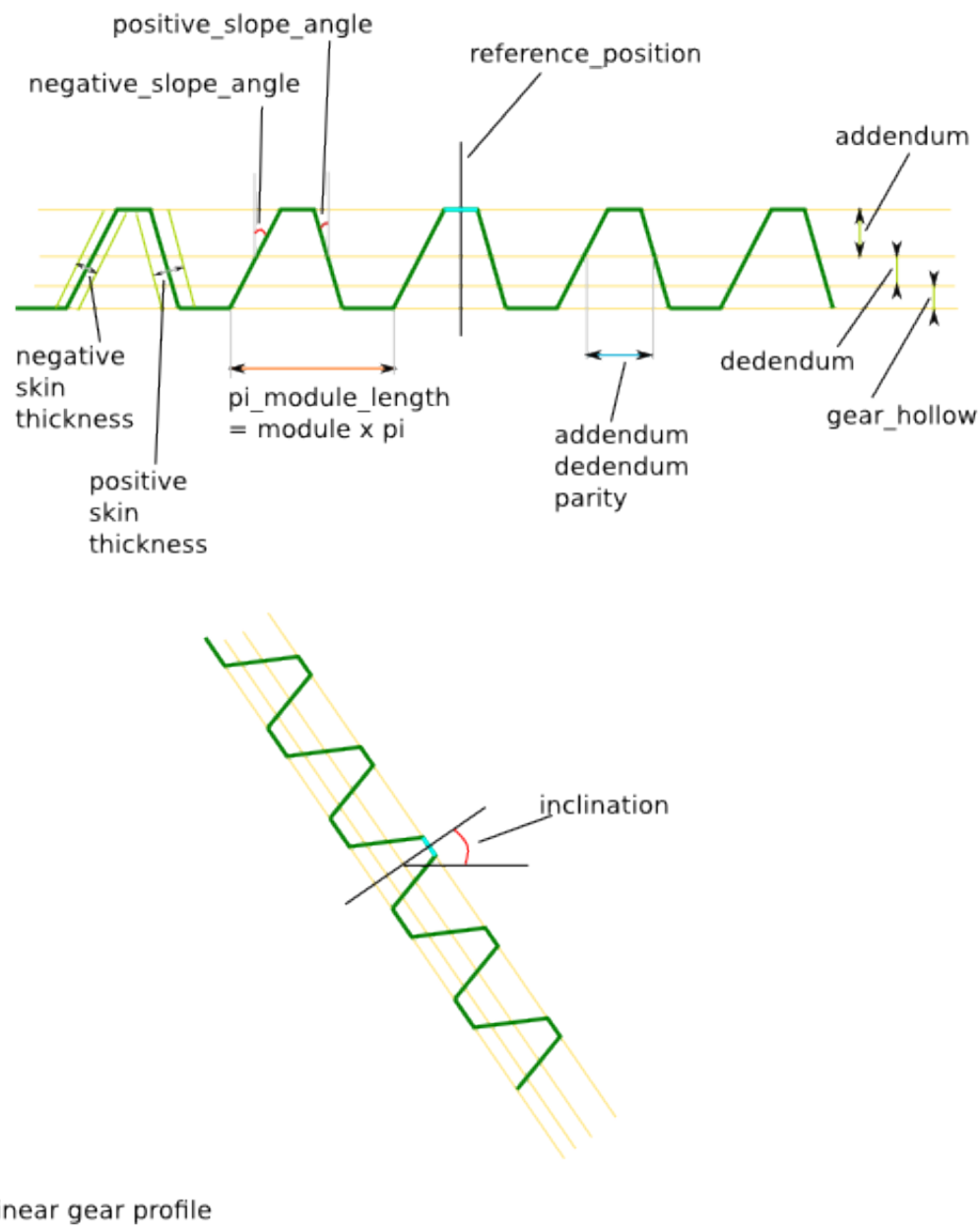


19.2.2 Gearring



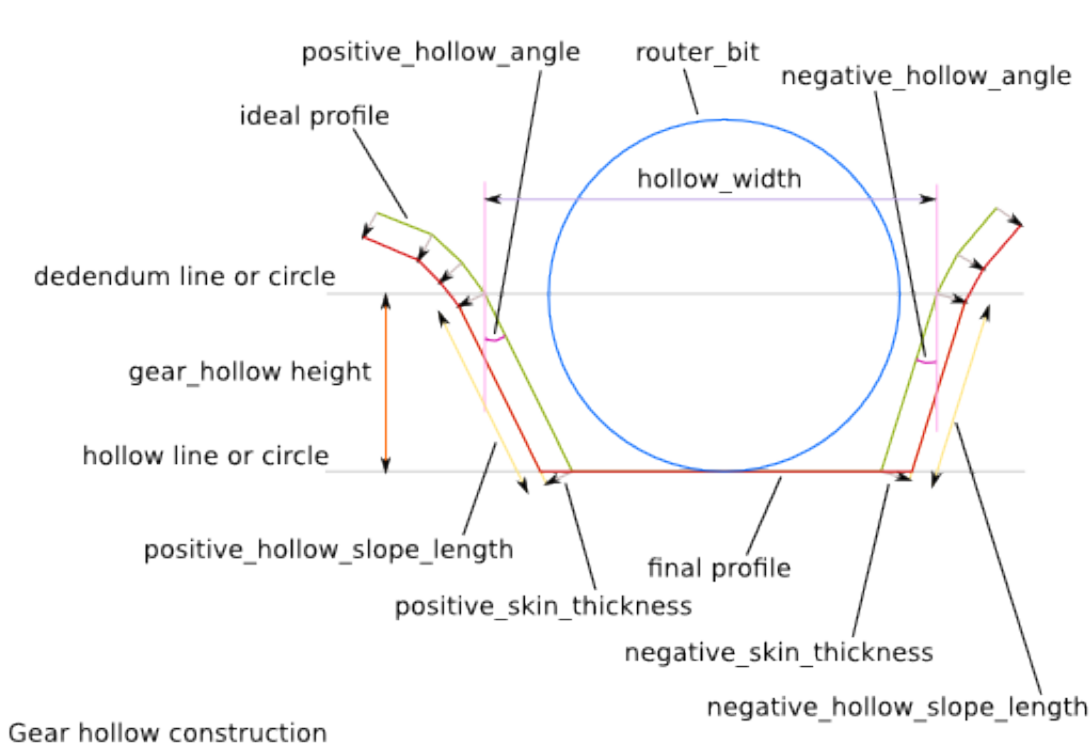
19.2.3 Gearbar



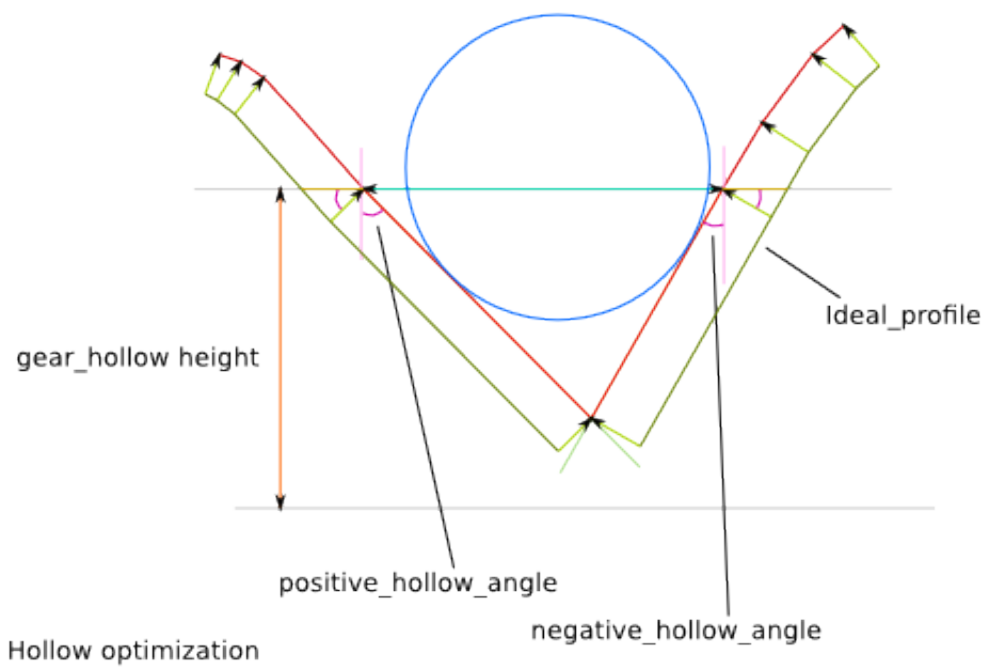


Linear gear profile

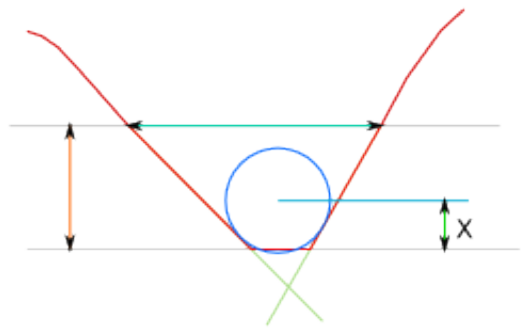
19.2.4 Gear hollow



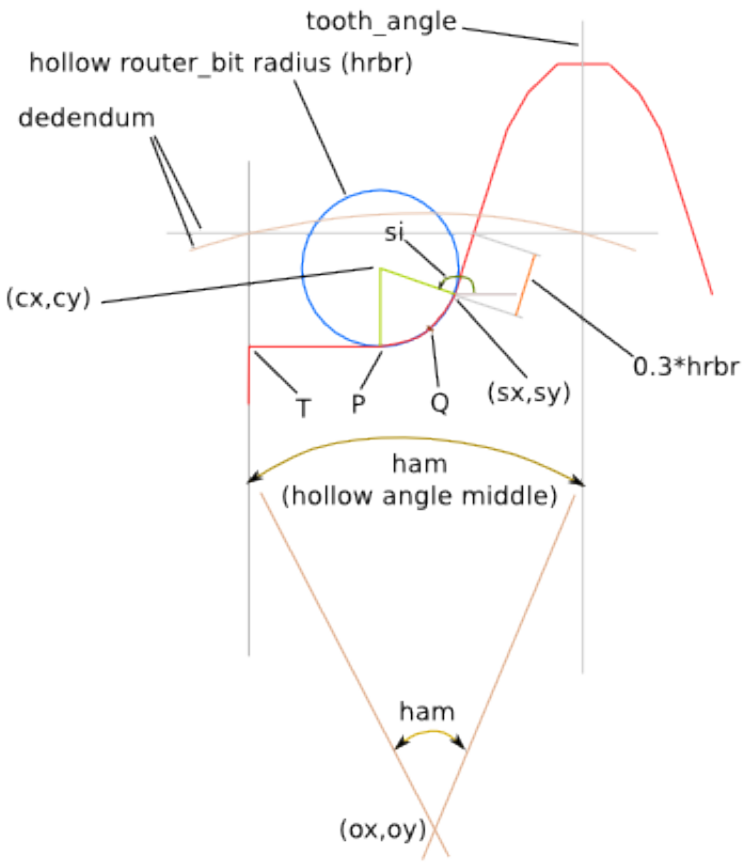
Gear hollow construction



Hollow optimization



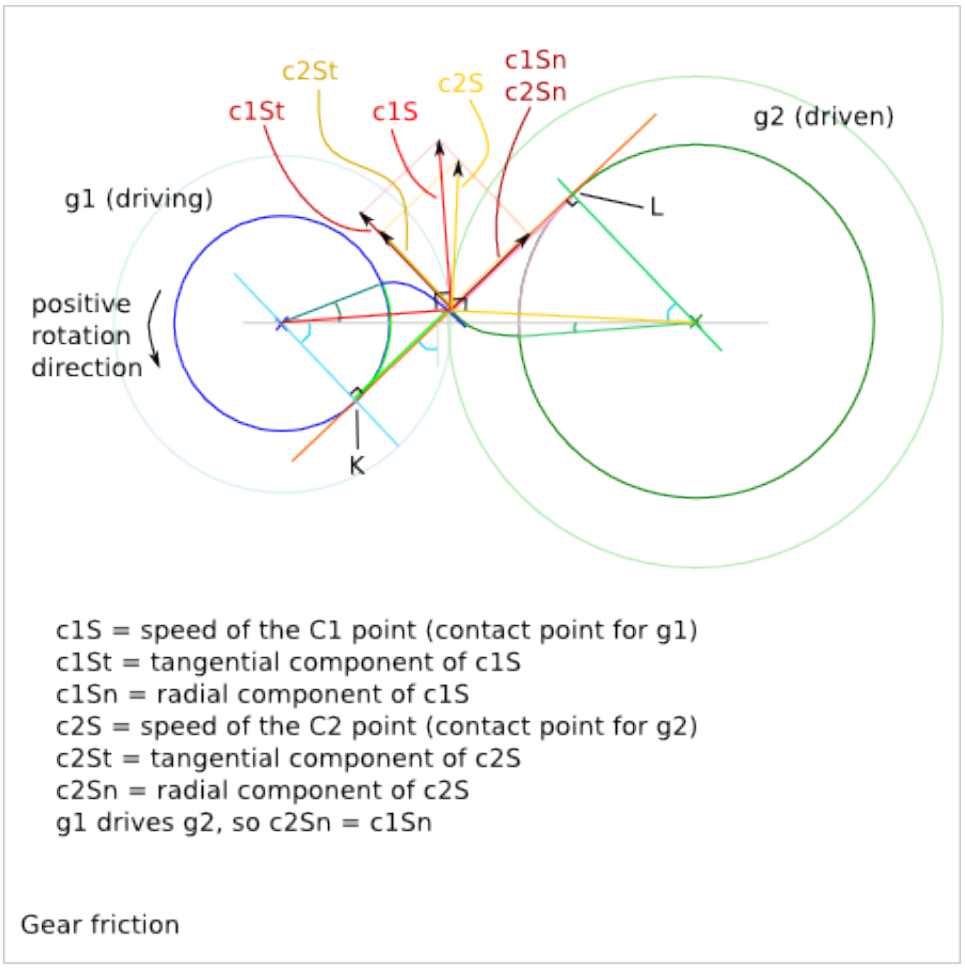
Hollow optimization condition



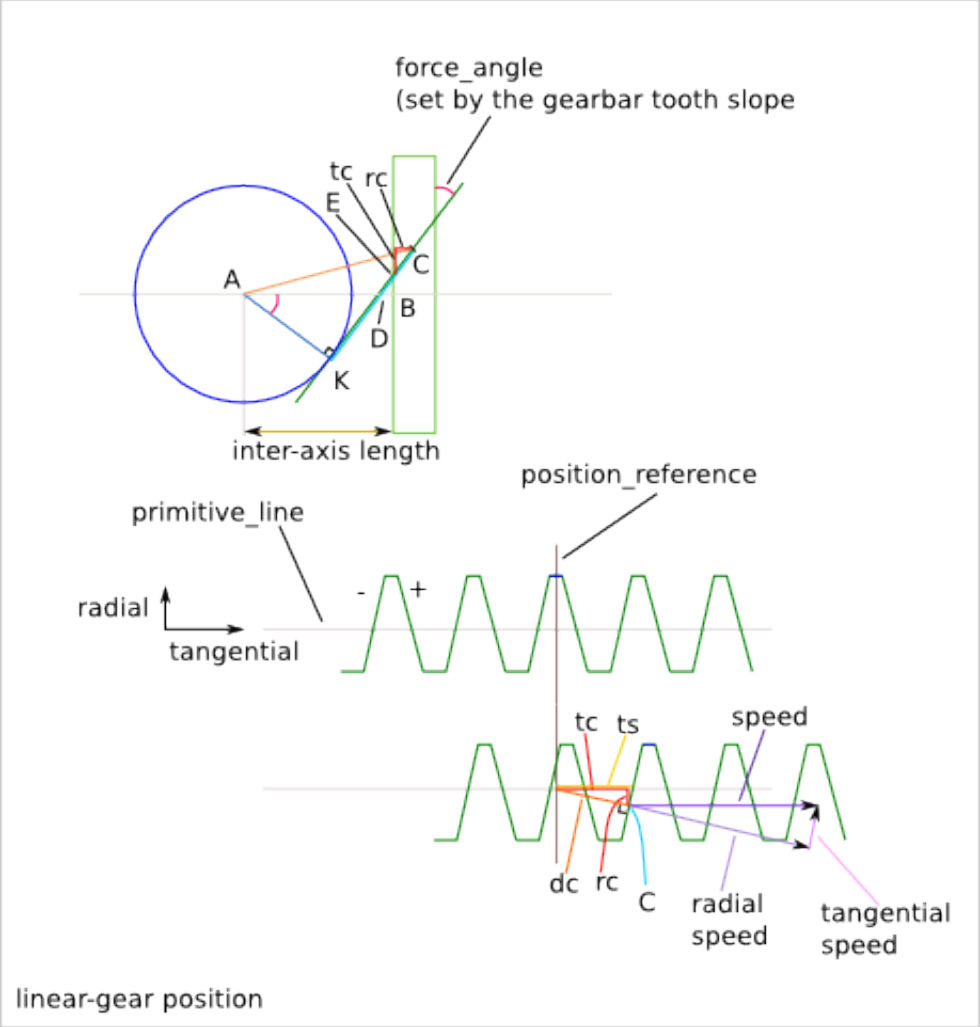
Gear half-hollow (type 3)

19.3 Gear position

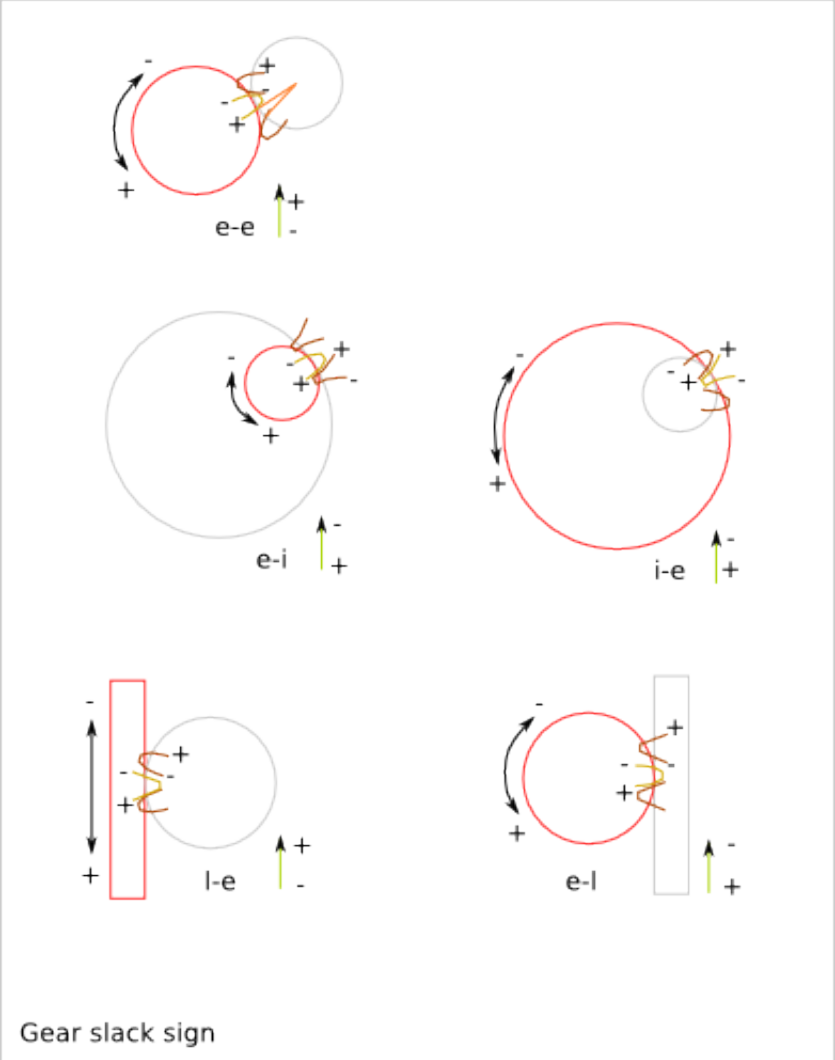
19.3.1 Gearwheel

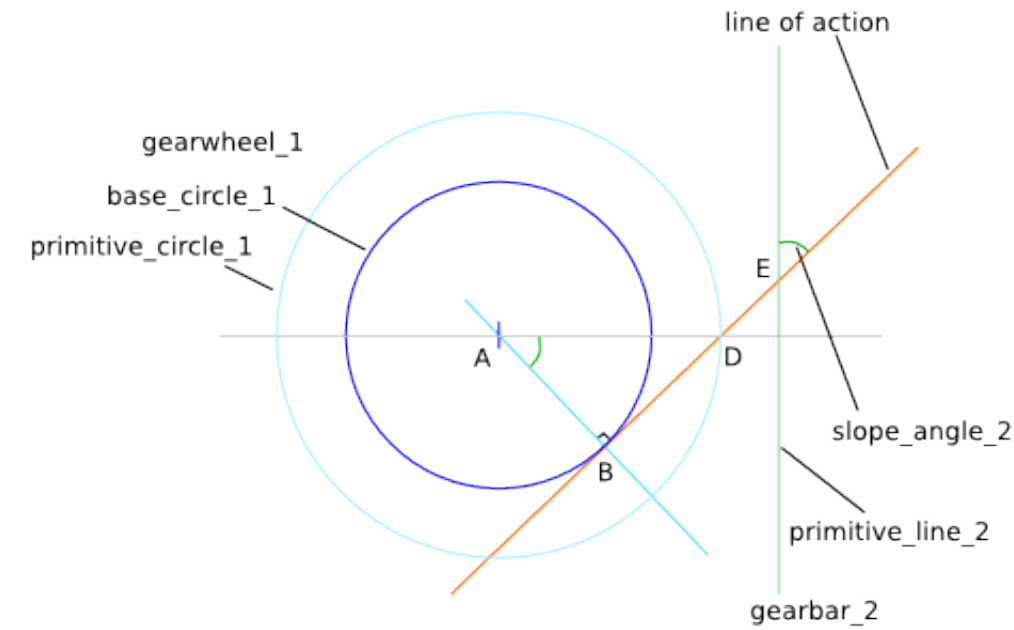


19.3.2 Gearbar

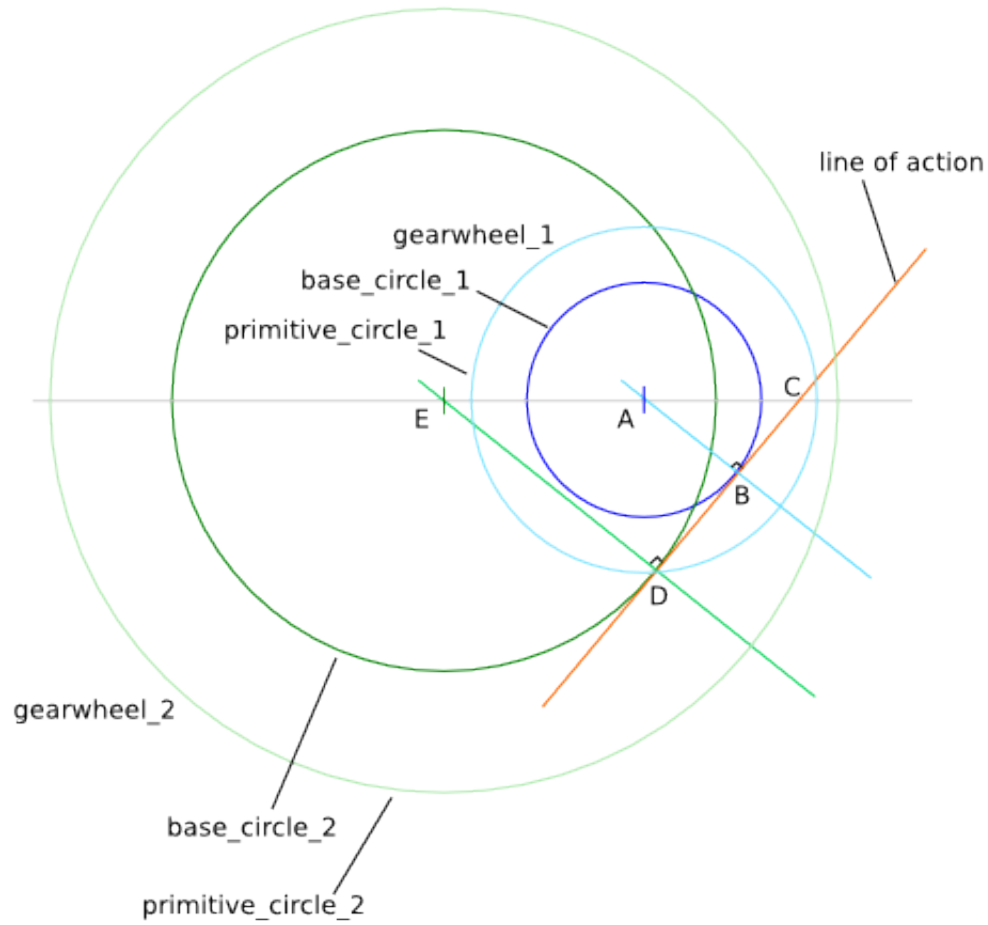


19.3.3 Position with additional inter-axis length

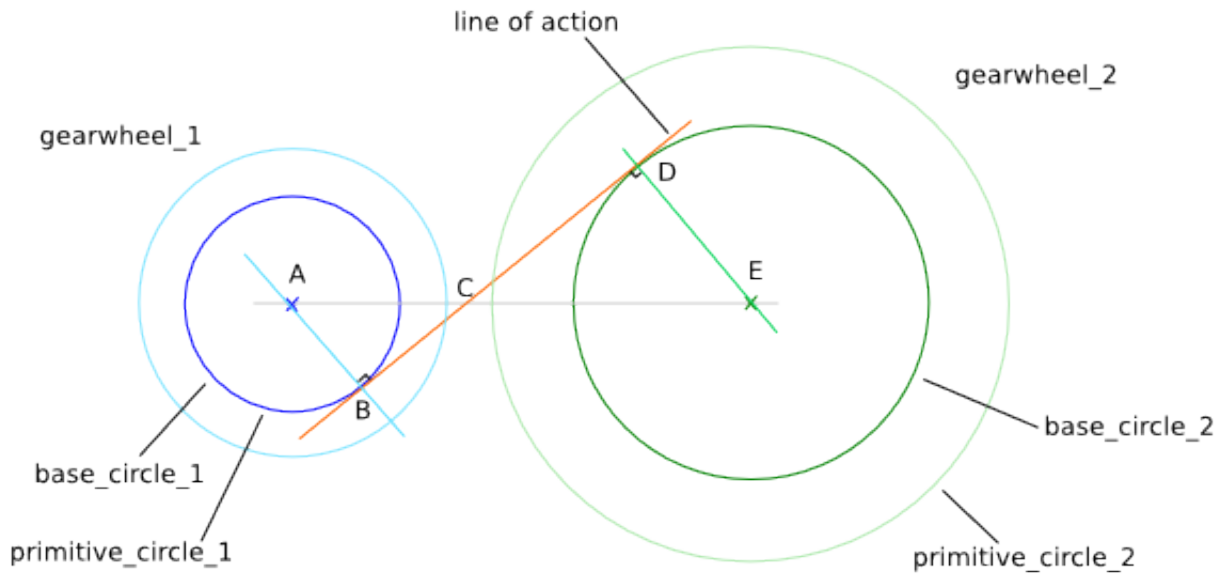




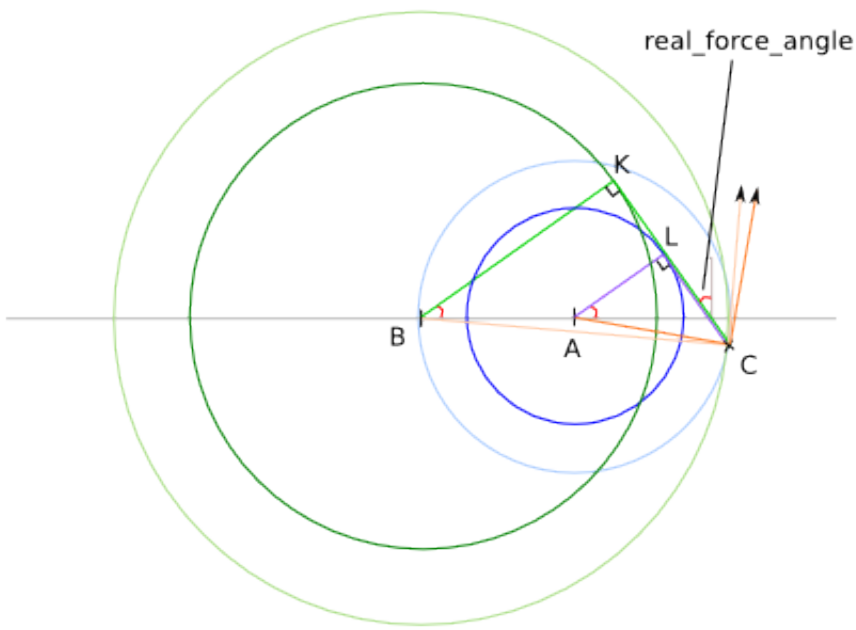
gearwheel-gearbar with additional inter-axis length geometry



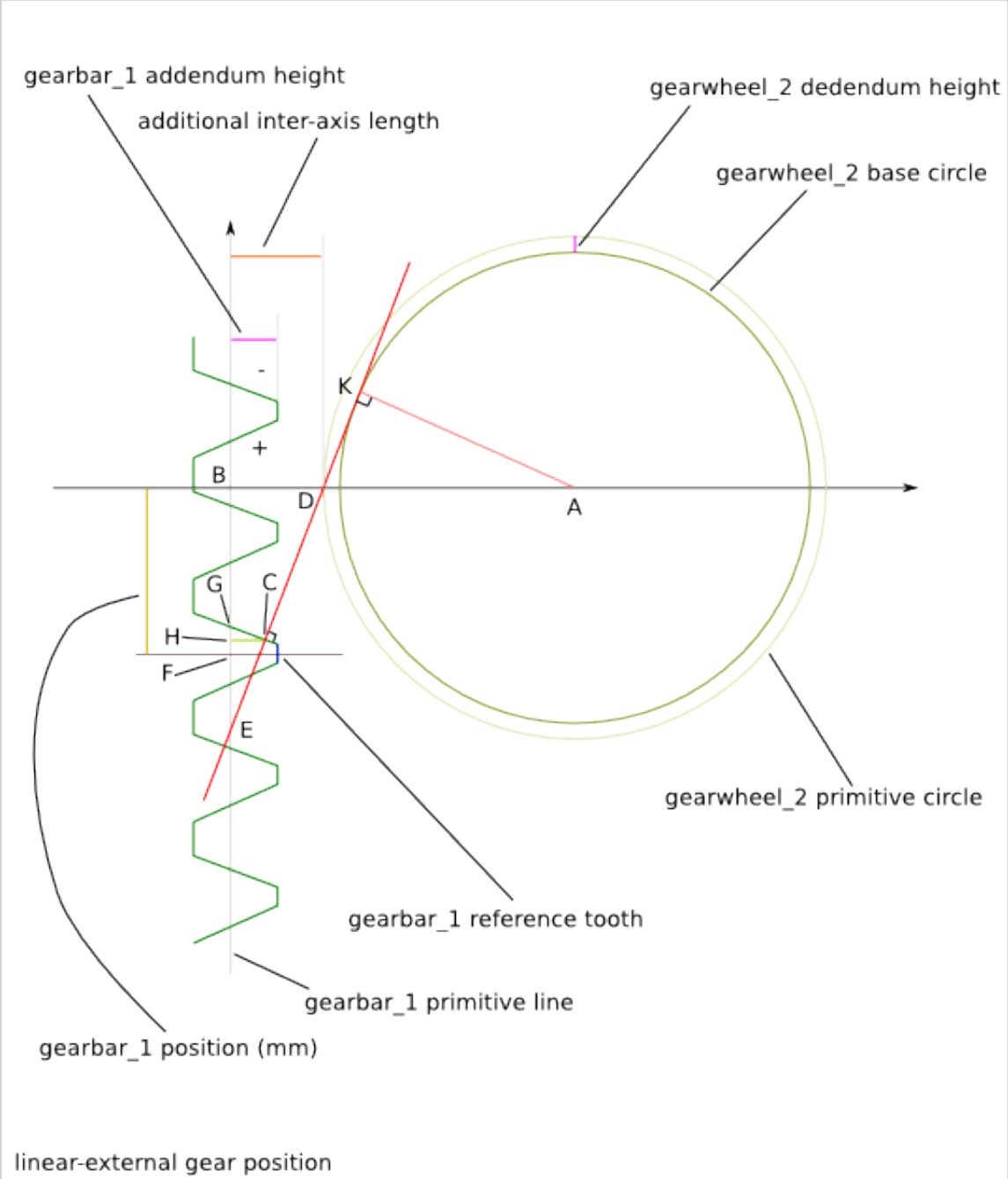
gearwheel-gearring with additional inter-axis length geometry



gearwheel-gearwheel with additional inter-axis length geometry



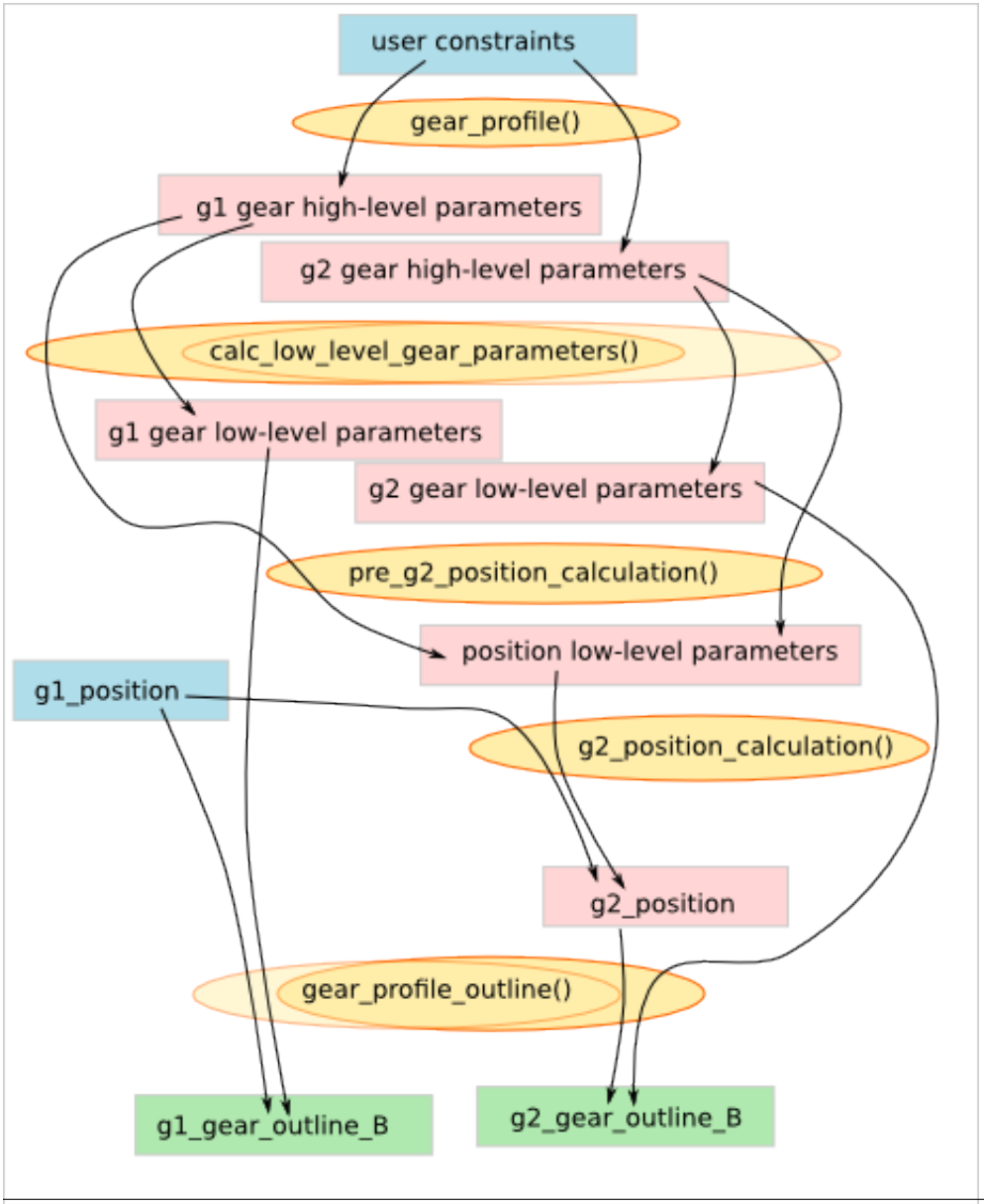
Internal-external gear position



CHAPTER 20

Gear Profile Implementation

20.1 Internal data-flow

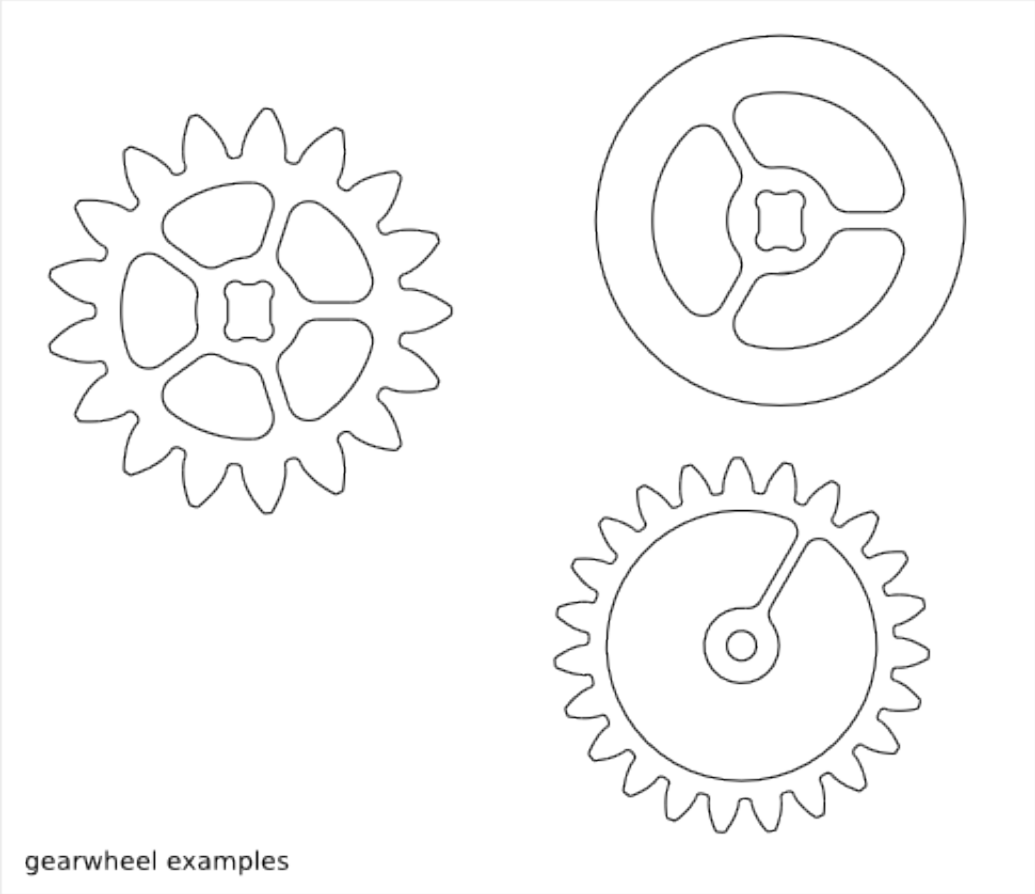


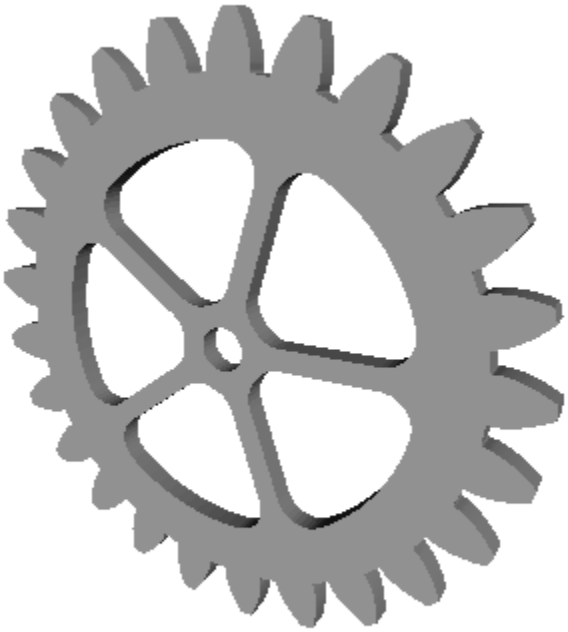
gear_profile data flow

CHAPTER 21

Gearwheel Design

Ready-to-use parametric *gearwheel* design (a.k.a. spur).



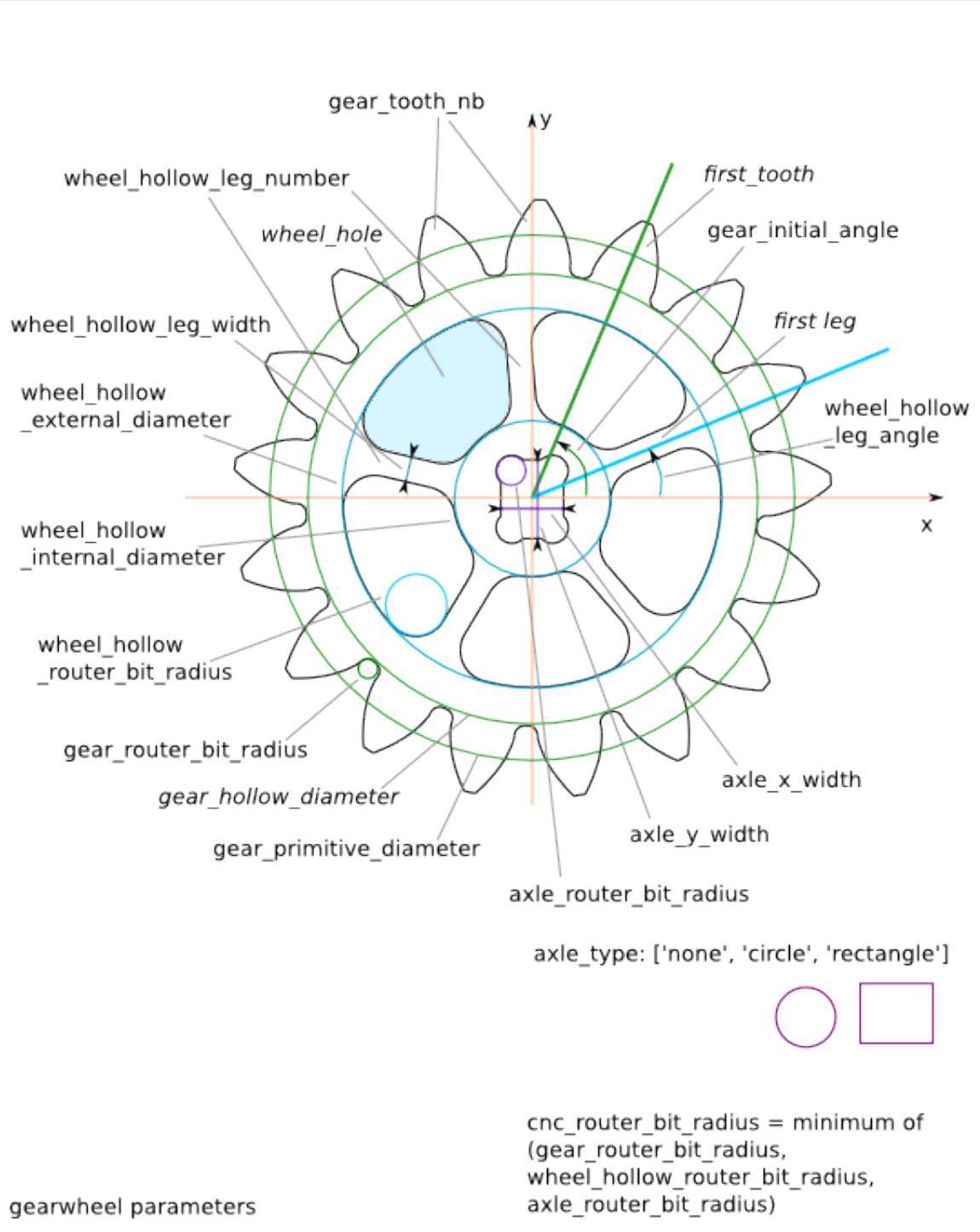


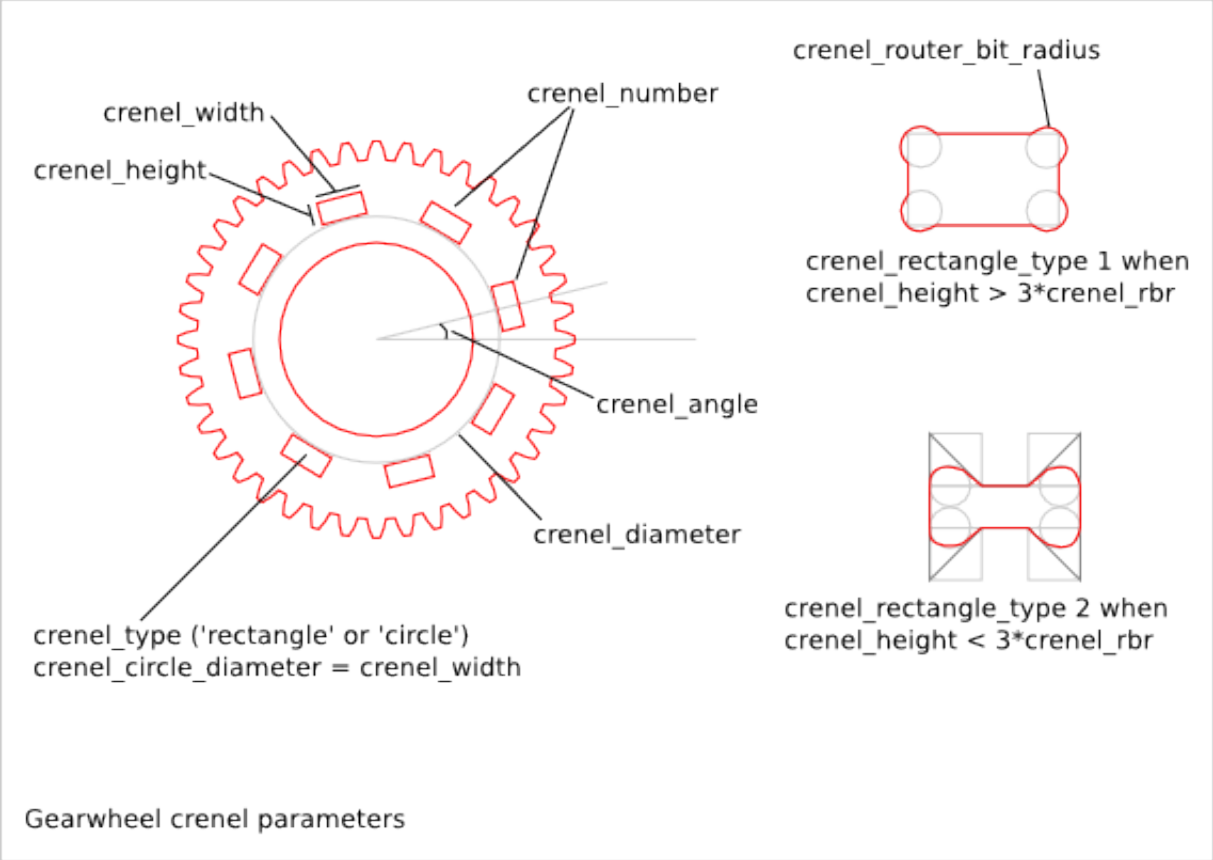
To get an overview of the possible gearwheel designs that can be generated by *gearwheel()*, run:

```
> python gearwheel.py --run_self_test
```

21.1 Gearwheel Parameter List

The parameter relative to the gear-profile are directly inherit from the Gear Profile Function.





21.2 Gearwheel Parameter Dependency

21.2.1 router_bit_radius

Four router_bit radius are defined: *gear_router_bit_radius*, *wheel_hollow_router_bit_radius*, *axle_router_bit_radius* and *cnc_router_bit_radius*. Each set the router_bit radius for different areas except *cnc_router_bit_radius* that set the mimnimum value for the three other router_bit radius. If an other router_bit radius is smaller than *cnc_router_bit_radius*, it is set to *cnc_router_bit_radius*. So, we have the relations:

```
cnc_router_bit_radius < gear_router_bit_radius
cnc_router_bit_radius < wheel_hollow_router_bit_radius
cnc_router_bit_radius < axle_router_bit_radius
```

21.2.2 axle_type

Three possible shapes of axle are possible: *none*, *circle* or *rectangle*. *none* means there is no axle (*axle_x_width* and *axle_y_width* are ignored). For *circle*, the parameter *axle_x_width* is used to set the circle diameter (*axle_y_width* is ignored). *axle_x_width* and *axle_y_width* must be bigger than twice *axle_router_bit_radius*:

```
2*axle_router_bit_radius < axle_x_width
2*axle_router_bit_radius < axle_y_width
```

21.2.3 wheel_hollow_leg_number

wheel_hollow_leg_number sets the number of legs (equal the number of wheel_hollows). If you set zero, no wheel_hollow are created and the other parameters related to the wheel_hollow are ignored. *wheel_hollow_internal_diameter* must be bigger than the axle. *wheel_hollow_external_diameter* must be smaller than the *gear_hollow_diameter* (which is not a parameter but derivated from other gear parameter):

```
axle_x_width < wheel_hollow_internal_diameter
sqrt(axle_x_width²+axle_y_width²) < wheel_hollow_internal_diameter
wheel_hollow_internal_diameter + 4*wheel_hollow_router_bit_radius < wheel_hollow_external_diameter
wheel_hollow_external_diameter < gear_hollow_diameter
```

21.2.4 gear_tooth_nb

gear_tooth_nb sets the number of teeth of the gear_profile. If *gear_tooth_nb* is set to zero, the gear_profile is replaced by a simple circle of diameter *gear_primitive_radius*.

21.2.5 Alignment angles

The rectangle axle is always fixed to the XY-axis. The angle between the first *wheel_hollow leg* (middle of it) and the X-axis is set with *wheel_hollow_leg_angle*. The angle between the first *gear_profile* tooth (middle of the addendum) and the X-axis is set with *gear_initial_angle*.

21.2.6 crenel_mark_nb

crenel_mark_nb lets you modify the first (or the several first) crenel to help you recognizing the first tooth. If the *crenel_type* is set to *rectangle*, the right-angle of the first crenels are rounded. If the *crenel_type* is set to *circle*, the first crenels have a egg-form. If you don't want to mark the first crenel, set *crenel_mark_nb* to *zero*. This feature is useful when you work with small gearwheel and you want to align them easily.

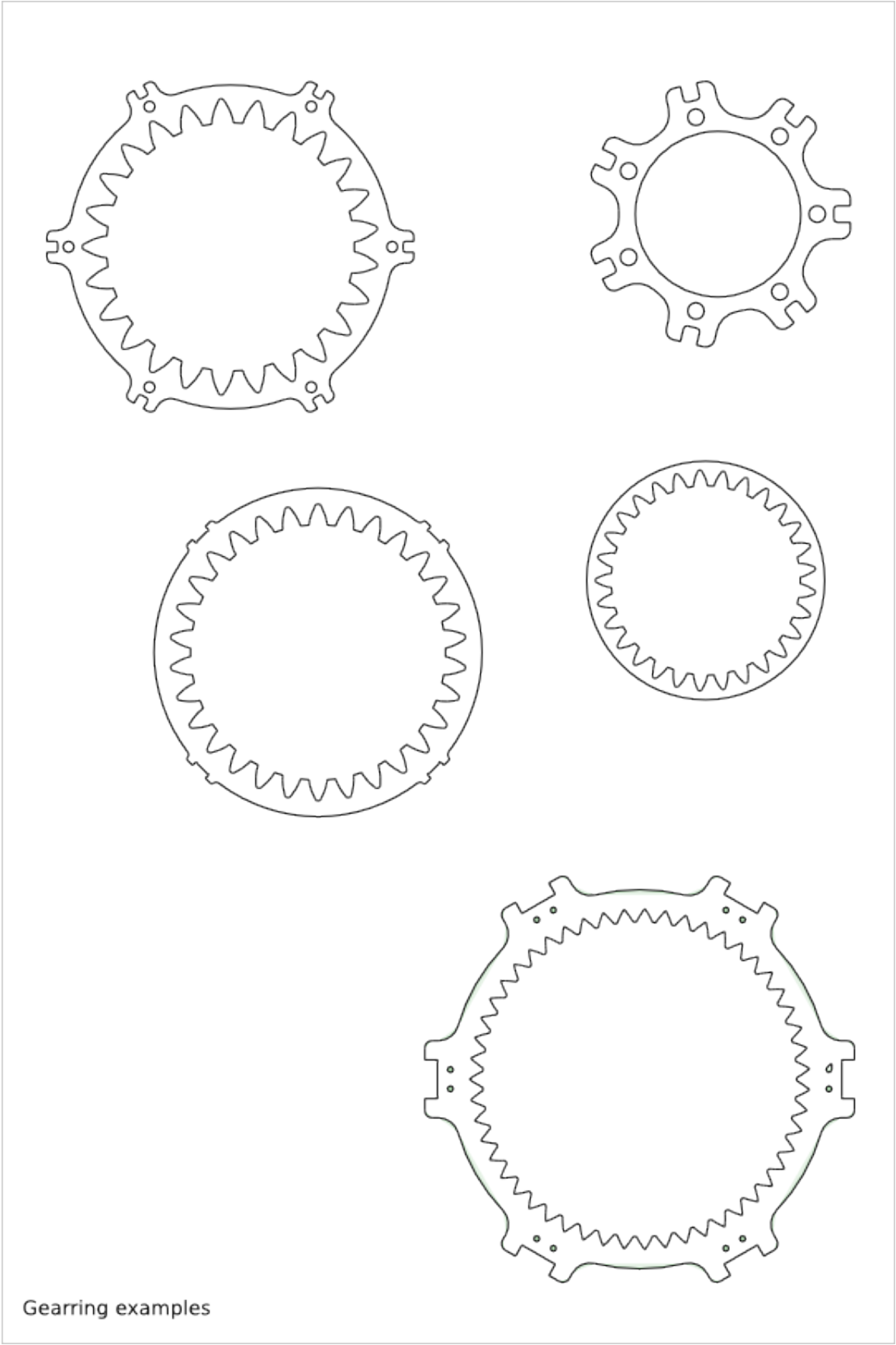
21.2.7 crenel_tooth_align

crenel_tooth_align is an alternative to the parameters *crenel_number* and *crenel_angle*. If *crenel_tooth_align* is set to a positive integer N, crenels are generated just under the gear-teeth, every N teeth. This feature is useful when you have a small space between the gear-teeth and the axle. In this case, material must be optimized by aligning crenel and teeth to avoid weak points (a.k.a. bottle-neck). To use *crenel_tooth_align*, the parameters *crenel_number* and *crenel_angle* must be set to *zero*.

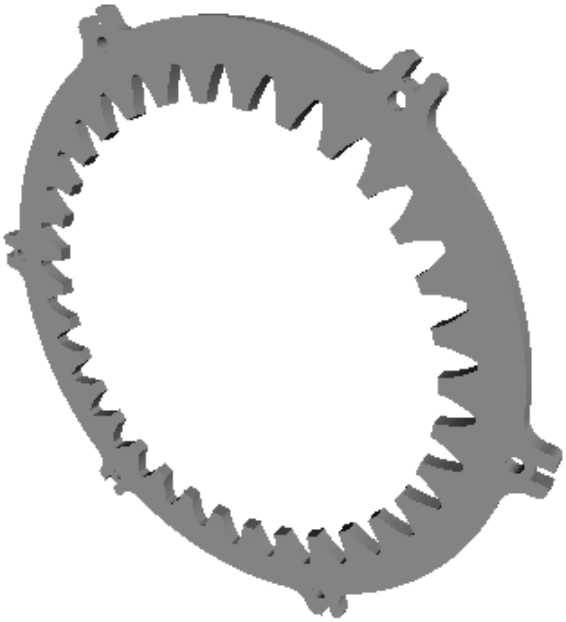
CHAPTER 22

Gearring Design

Ready-to-use parametric *gearring* design (a.k.a. annulus).



Gearring examples

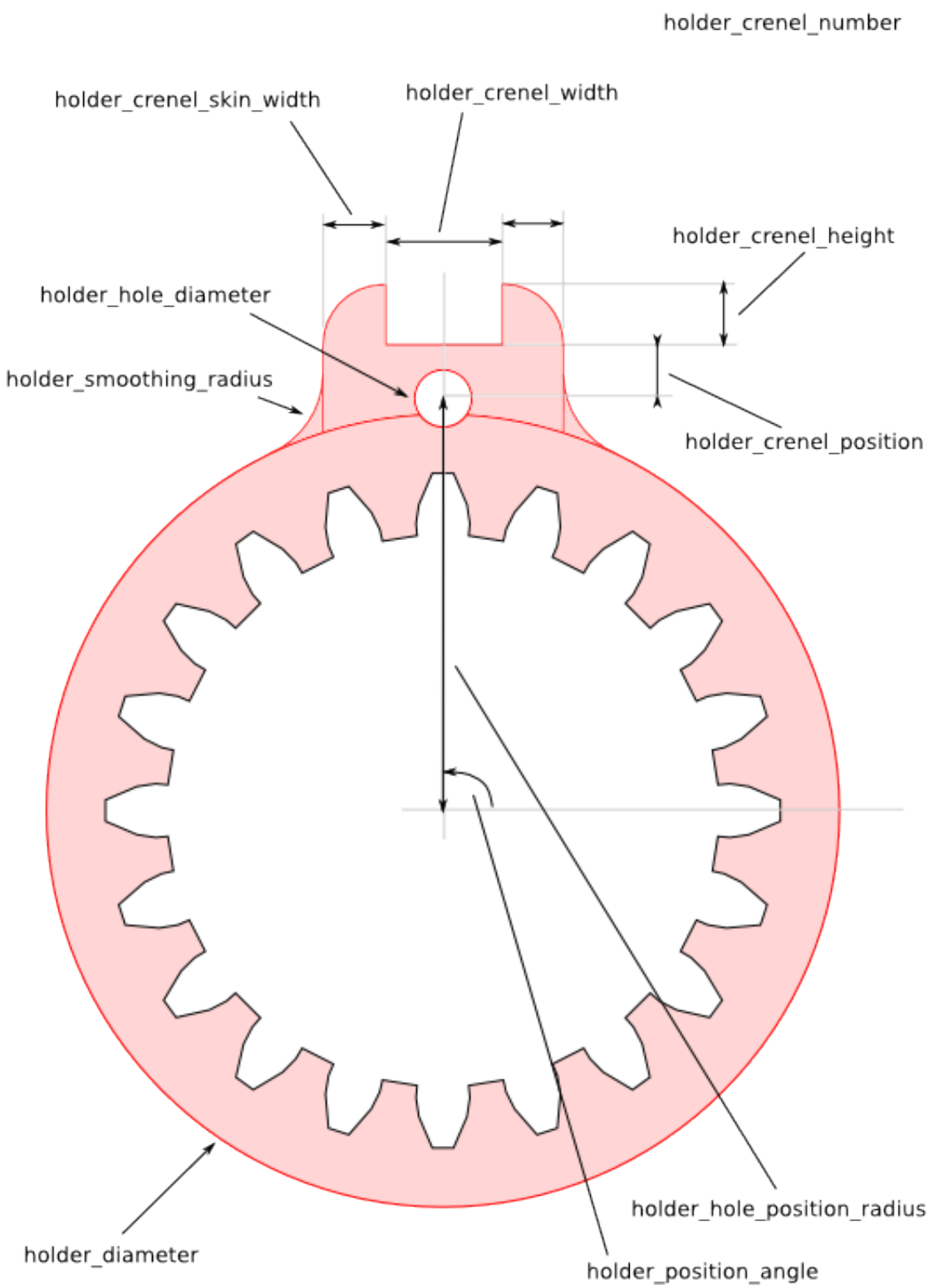


To get an overview of the possible gearring designs that can be generated by *gearring()*, run:

```
> python gearring.py --run_self_test
```

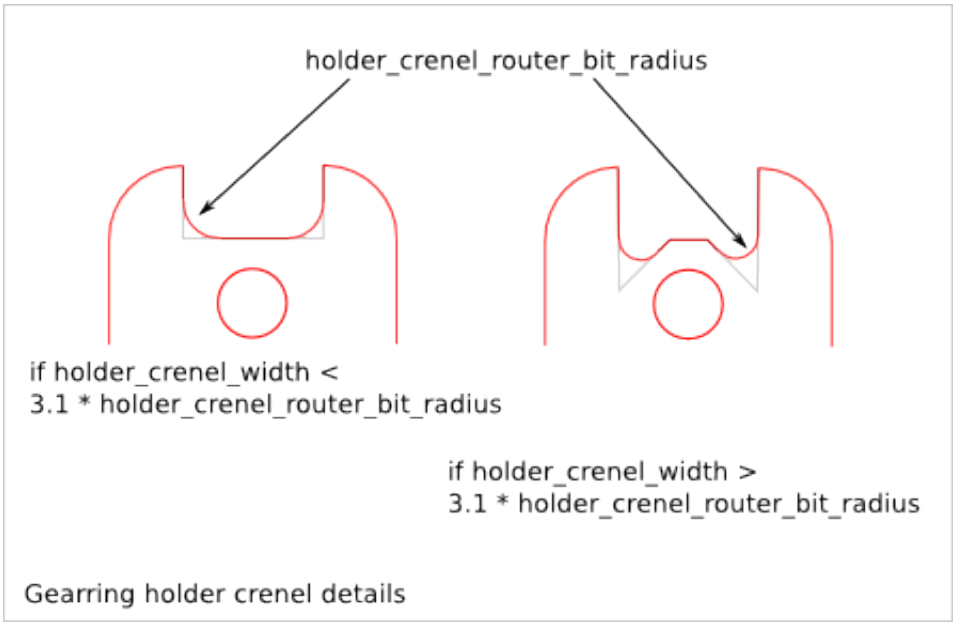
22.1 Gearring Parameter List

The parameter relative to the gear-profile are directly inherit from the Gear Profile Function.

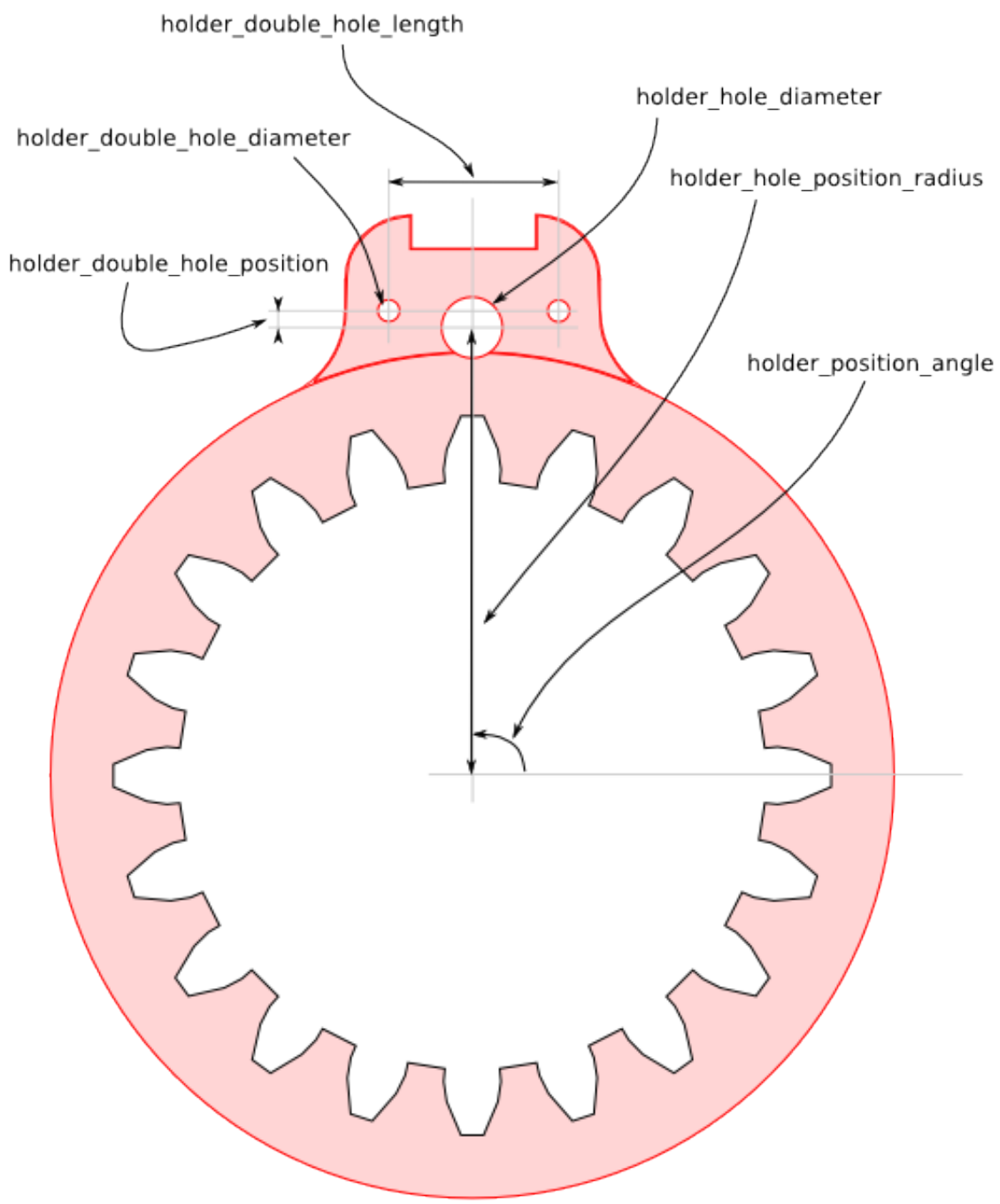


if holder_hole_diameter = 0, no holder hole
if holder_crenel_height = 0, no crenel

Gearring holder parameters



Gearring holder crenel details



Gearring holder_double_hole_length parameter

22.2 Gearring Parameter Dependency

22.2.1 router_bit_radius

Four router_bit radius are defined: *gear_router_bit_radius*, *holder_crenel_router_bit_radius*, *holder_smoothing_radius* and *cnc_router_bit_radius*. Each set the router_bit radius for different areas except *cnc_router_bit_radius* that set the minimum value for the three other router_bit radius. If an other router_bit radius is smaller than *cnc_router_bit_radius*, it is set to *cnc_router_bit_radius*. So, we have the relations:

```
cnc_router_bit_radius < gear_router_bit_radius
cnc_router_bit_radius < holder_crenel_router_bit_radius
cnc_router_bit_radius < holder_smoothing_radius
```

If you leave *holder_smoothing_radius* to 0.0, it will be changed automatically to the biggest possible value.

22.2.2 holder_hole_diameter

holder_hole_diameter sets the diameter of the holder-holes. If *holder_hole_diameter* is set to 0.0, no holder-hole are created.

22.2.3 holder_crenel_number

holder_crenel_number sets the number of holder-crenels (equal to the number of holder-hole). If *holder_crenel_number* is set to zero, no holder-crenel is created and the outline of the gearring is a simple circle.

22.2.4 holder_crenel_width

holder_crenel_width must be bigger than the router_bit diameter:

```
holder_crenel_width > 2 * holder_crenel_router_bit_radius
```

If *holder_crenel_width* is big enough, the crenel bottom shape is changed to get *alternative enlarged* corners.

22.2.5 gear_tooth_nb

gear_tooth_nb sets the number of teeth of the gear_profile. If *gear_tooth_nb* is set to zero, the gear_profile is replaced by a simple circle of diameter *gear_primitive_radius*.

22.2.6 Alignment angles

gear_initial_angle sets the angle between the X-axis and the middle of the addendum of the first tooth. *holder_position_angle* sets the angle between the X-axis and the middle of the first holder-crenel. Use *gear_initial_angle* or *holder_position_angle* or both to ajust the offset angle between the gear-profile anf the gearring-holder.

22.2.7 holder_hole_mark_nb

holder_hole_mark_nb lets you modify the first (or the several first) crenel to help you recognizing the first tooth. The first crenels have a egg-form instead of the circle-form. If you don't want to mark the first crenel, set *crenel_mark_nb* to *zero*. This feature is useful when you need pile up gearring and find easily the first tooth to align them.

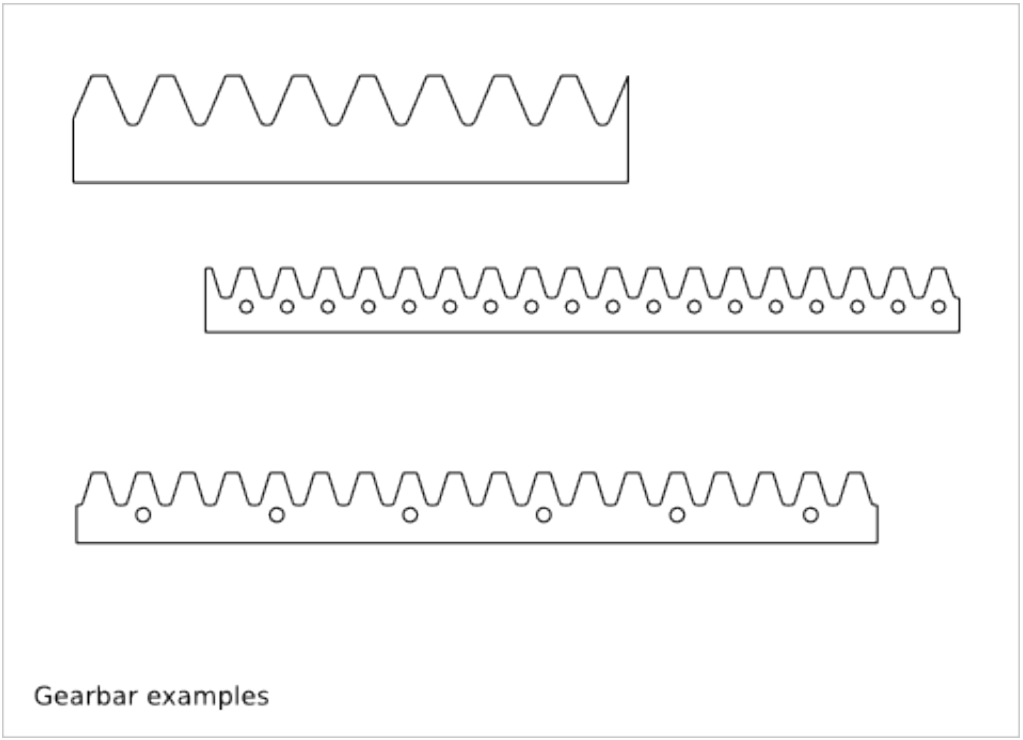
22.2.8 holder_double_hole

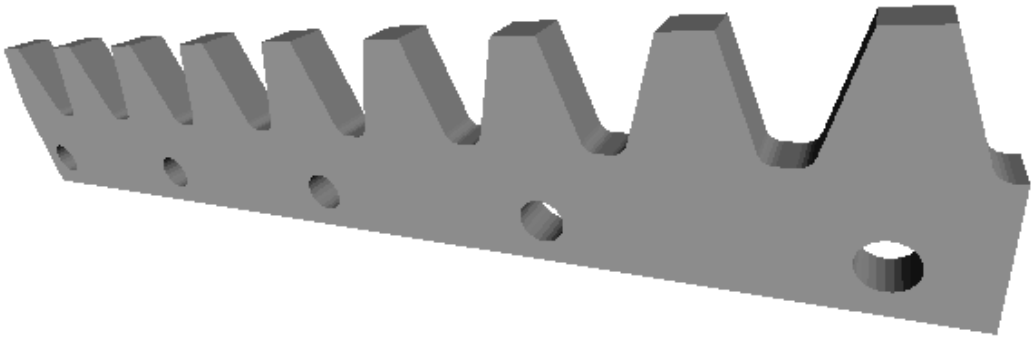
In addition to the *holder_hole*, you can generate the *holder_double_hole* defined by the parameters *holder_double_hole_diameter*, *holder_double_hole_length* and *holder_double_holde_position*. The distance between the two double_holes is set by *holder_double_hole_length*. The radius position is set by *holder_double_holde_position* relative to the *holder_hole_position_radius*. The *holder_double_holes* are useful when you use the crenel-hole with thin steel-rod for alignment and Z-shearing resistance and you want to increase the stability. At the same time, you can use the *holder_holes* to put threaded rods.

CHAPTER 23

Gearbar Design

Ready-to-use parametric *gearbar* design (a.k.a. rack).



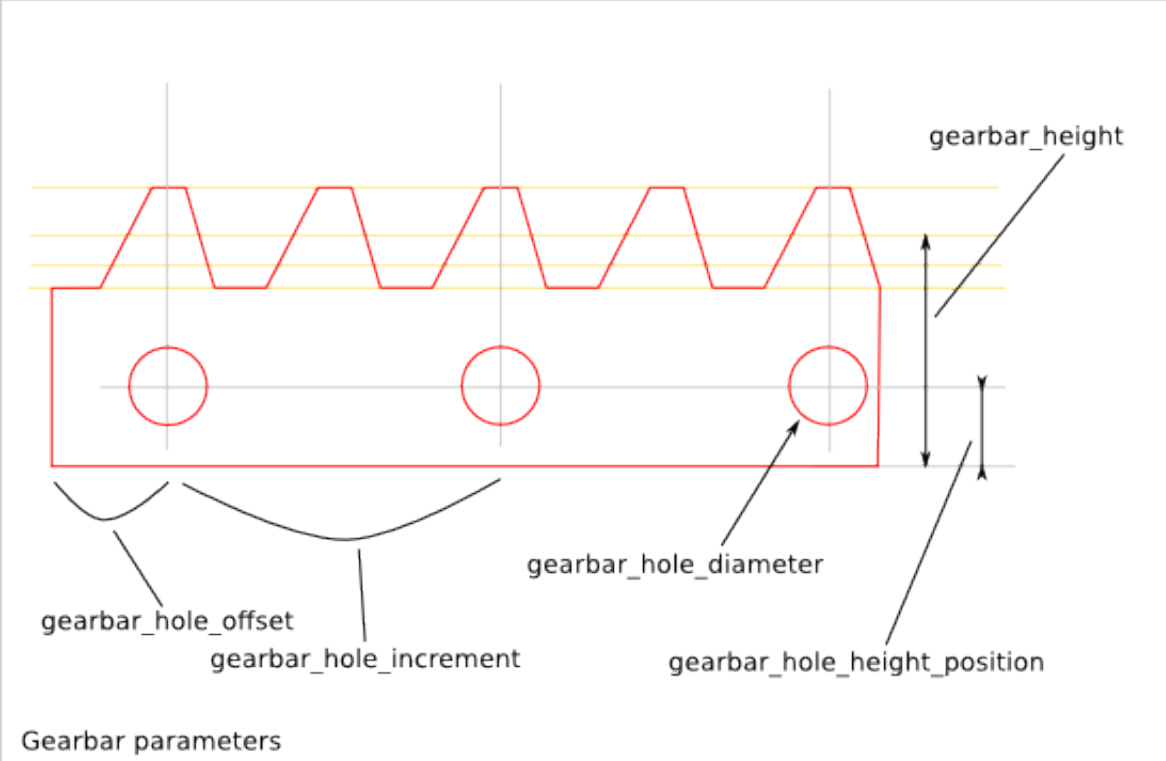


To get an overview of the possible gearbar designs that can be generated by *gearbar()*, run:

```
> python gearbar.py --run_self_test
```

23.1 Gearbar Parameter List

The parameter relative to the gear-profile are directly inherit from the Gear Profile Function.



23.2 Gearbar Parameter Dependency

23.2.1 gearbar_hole_diameter

gearbar_hole_diameter sets the diameter of the gearbar-holes. If *gearbar_hole_diameter* is set to 0.0, no gearbar-hole are created.

23.2.2 gearbar_hole_height_position

gearbar_hole_height_position sets the vertical position of the gearbar-hole centers. *gearbar_hole_height_position* must be placed between the bottom of the gearbar and the gear-profile:

```
gearbar_hole_radius = gearbar_hole_diameter/2
gearbar_hole_height_position > gearbar_hole_radius
gearbar_hole_height_position < minimal_gear_profile_height - gearbar_hole_radius
```

23.2.3 gearbar_hole_offset and gearbar_hole_increment

The abscissas of the centers of the gearbar-holes are always located at the middle of the addendum of a gear-tooth. *gearbar_hole_offset* sets the number of gear-teeth between the left-side of the gearbar to the first gearbar-hole. *gearbar_hole_increment* sets the number of gear-teeth between two consecutive gearbar-holes:

```
gearbar_hole_increment > 0
```

23.2.4 gear_tooth_nb

gear_tooth_nb sets the number of teeth of the gear_profile. If *gear_tooth_nb* is set to zero, the gear_profile is replaced by a simple line of length *gear_primitive_radius*.

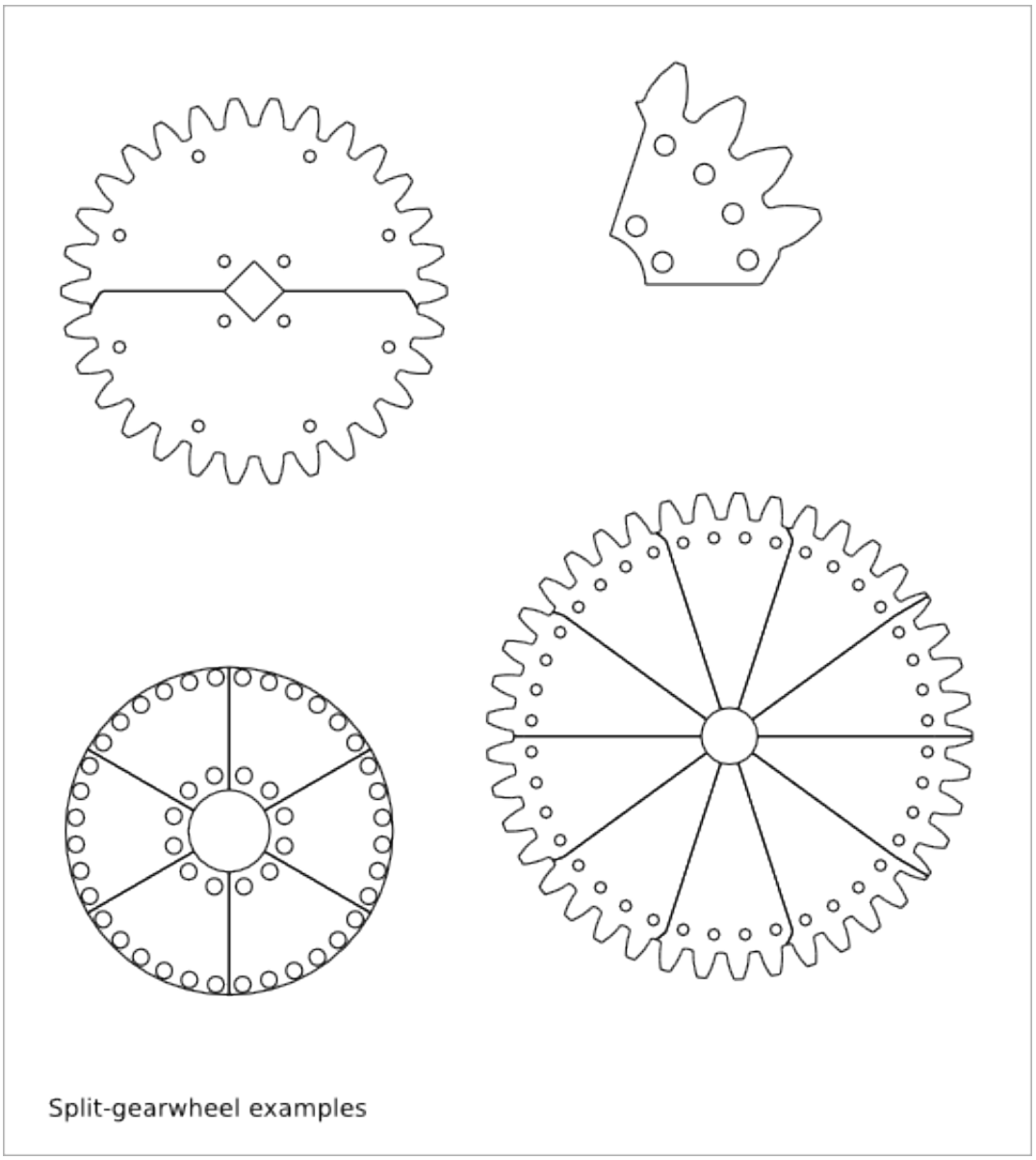
23.2.5 Alignment

gear_initial_angle, *center_position_x*, *center_position_y* and *second_gear_position_angle* are only used for the simulation. The gearbar as a simple display, as a FreeCAD object or as a design file is always placed to get its bottom-left corner at the (0,0) coordinates.

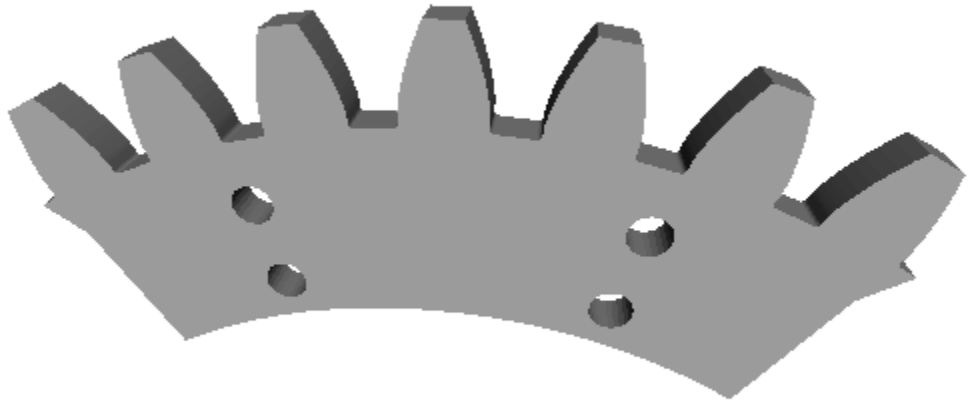
CHAPTER 24

Split-gearwheel Design

Ready-to-use parametric *split-gearwheel* design (i.e. spur that is split for its fabrication).



Split-gearwheel examples

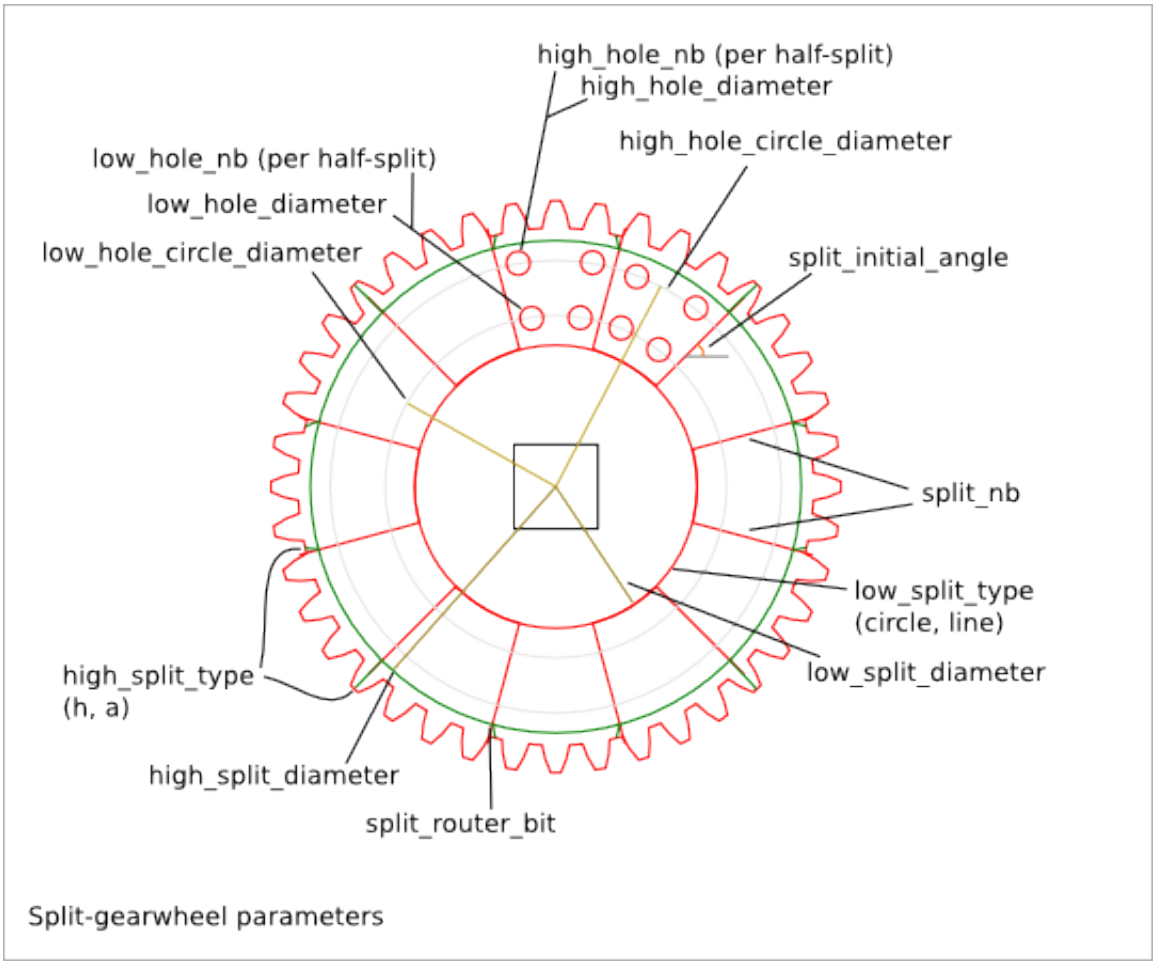


To get an overview of the possible split-gearwheel designs that can be generated by *split_gearwheel()*, run:

```
> python split_gearwheel.py --run_self_test
```

24.1 Split-gearwheel Parameter List

The parameter relative to the gear-profile are directly inherit from the Gear Profile Function.



24.2 Split-gearwheel Parameter Dependency

24.2.1 router_bit_radius

Three router_bit radius are defined: *gear_router_bit_radius*, *split_router_bit_radius*, and *cnc_router_bit_radius*. Each set the router_bit radius for different areas except *cnc_router_bit_radius* that set the mimnimum value for the two other router_bit radius. If an other router_bit radius is smaller than *cnc_router_bit_radius*, it is set to *cnc_router_bit_radius*. So, we have the relations:

```
cnc_router_bit_radius < gear_router_bit_radius
cnc_router_bit_radius < split_router_bit_radius
```

24.2.2 split_nb

split_nb defines in how many parts the gearwheel must be split. The *split_gearwheel()* function generates two sets (A and B) of *split_nb* parts. So you get at the end 2*split_nb parts. The set A (respectively B) makes a complete gearwheel. The set-A-gearwheel and the set-B-gearwheel can be stick together to ensure a better stability. The *low-holes* and *high-holes* ensure a good alignment between the set-A parts and set-B parts. The parameters *low_hole_nb* and *high_hole_nb* define the number of holes per half-split-portion i.e. the common portion between a set-A parts and a set-B part.

24.2.3 low_split_diameter and high_split_diameter

The constraints define 5 circles: *low_split_diameter*, *low_hole_circle_diameter*, *high_hole_circle_diameter*, *high_split_diameter* and *minimal_gear_profile_radius* (inferred from the gear-profile). If *gear_tooth_nb* = 0 then *high_split_diameter* = *minimal_gear_profile_radius*. These five circles are strictly included in each others:

```
low_split_diameter + low_hole_radius < low_hole_circle_diameter
low_hole_circle_diameter + low_hole_radius + high_hole_radius < high_hole_circle_diameter
high_hole_circle_diameter + high_hole_radius < high_split_diameter
high_split_diameter < minimal_gear_profile_radius
```

24.2.4 low_split_type

low_split_type defines the outline at the low-split-circle:

```
circle : the outline is an arc of circle
line   : the outline is composed of two lines
```

24.2.5 high_split_type

high_split_type defines how to join the split radius with the gear-profile. Indeed the number of gear-teeth and the number of split-portion are independant. In most of the case, the gear-hollow doesn't fit exactly the split radius. The split radius stops at the high-split circle. Then, the outline goes straight to the gear-profile. The angle at the high-split circle is smooth with *split_router_bit_radius*. The possible values for *high_split_type* are:

```
'h': the outline goes to the closest gear-hollow middle
'a': the outline goes to the addendum middle if it best fits, otherwise it goes to the closest gear-h
```

24.2.6 gear_tooth_nb

gear_tooth_nb sets the number of teeth of the gear_profile. If *gear_tooth_nb* is set to zero, the gear_profile is replaced by a simple circle of diameter *gear_primitive_radius*.

24.2.7 Alignment angles

gear_initial_angle sets the angle between the X-axis and the middle of the addendum of the first tooth. *split_initial_angle* sets the angle between the X-axis and the first split radius. Use *gear_initial_angle* or *split_initial_angle* or both to ajust the offset angle between the gear-profile anf the split-portion.

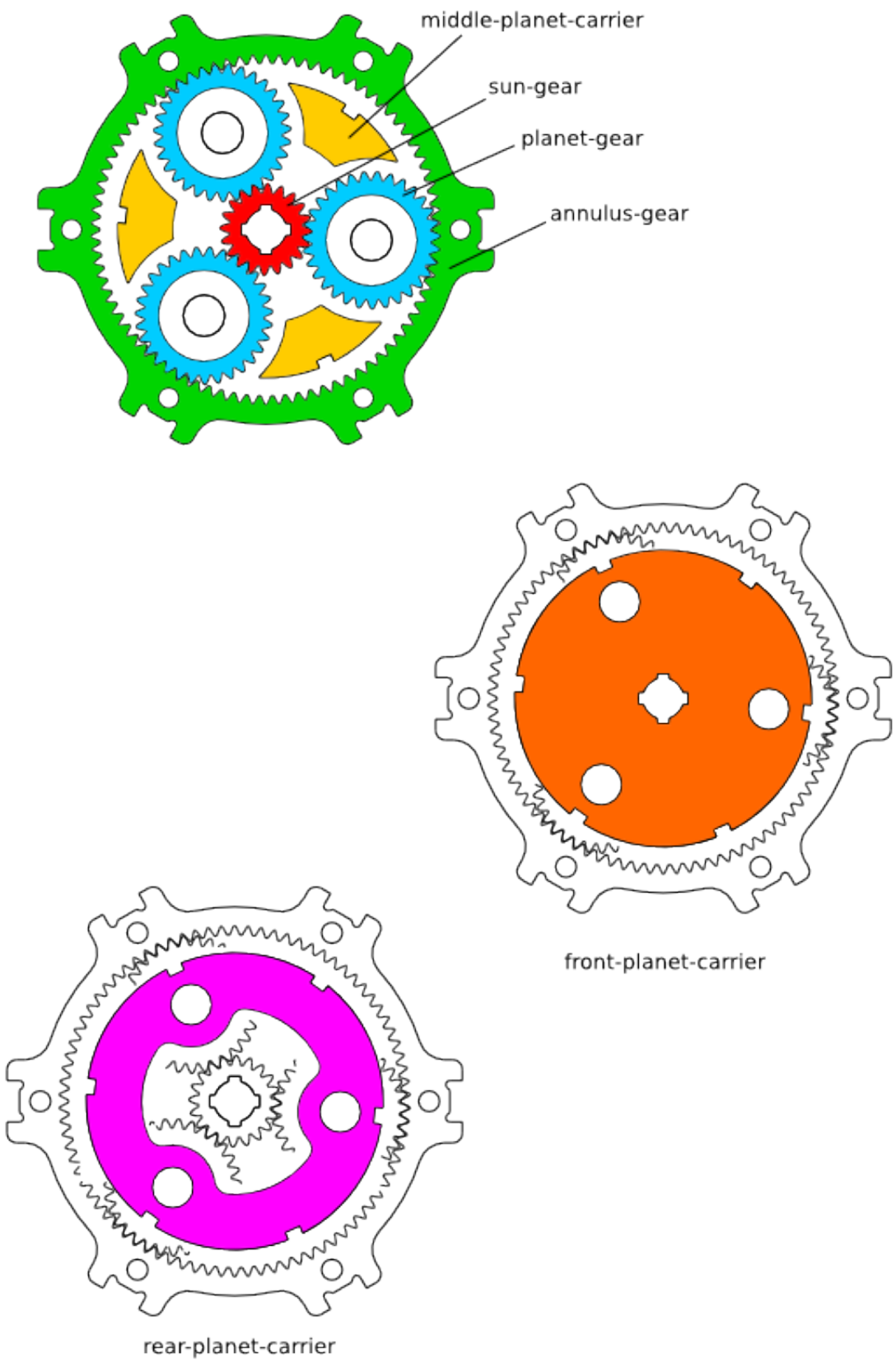
CHAPTER 25

Epicyclic Gearing Design

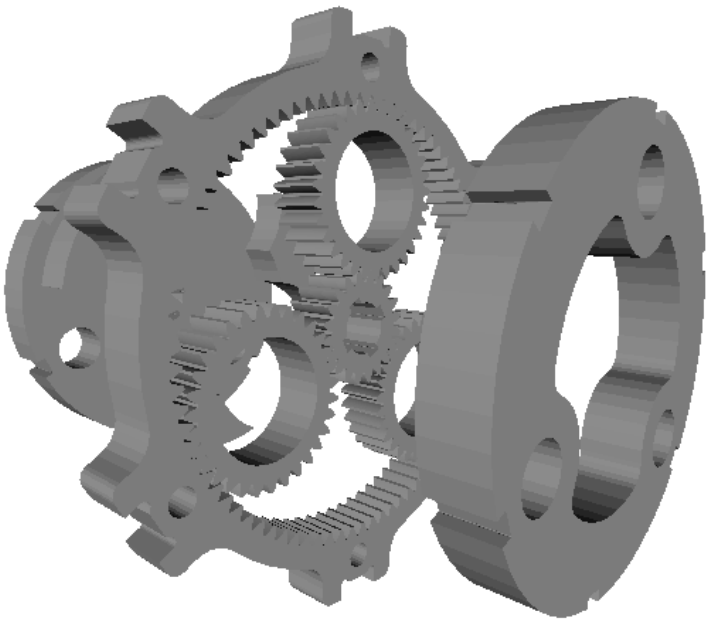
Ready-to-use parametric *epicyclic-gearing* design. Check also the epicyclic-gearing design variants Low_torque_transmission Design and High_torque_transmission Design that might better fit your requirements.

The *epicyclic gearing* is a system made out of several parts:

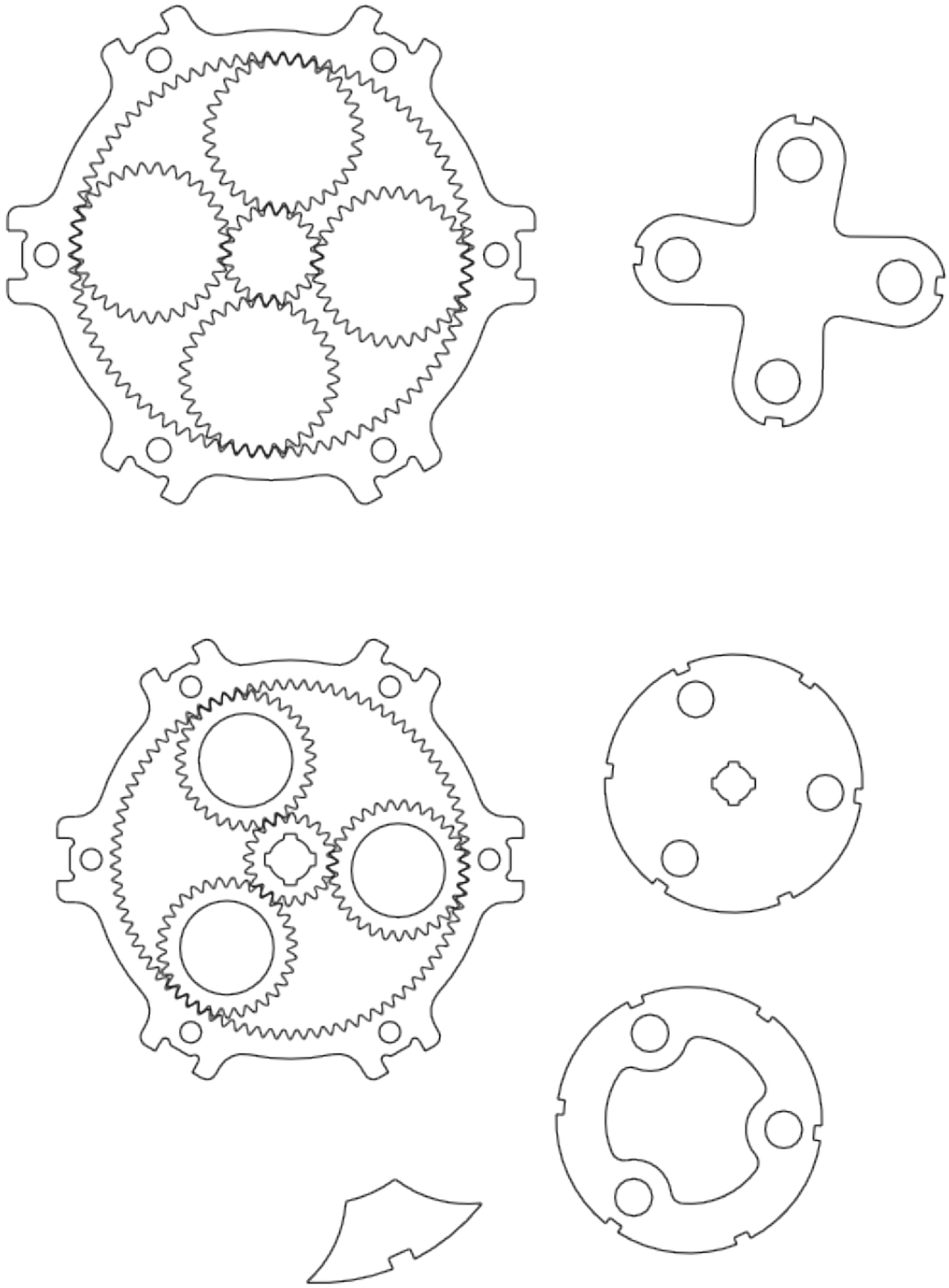
- sun-gear
- planet-gear
- annulus-gear
- planet-carrier (rear, middle and front)



Epicyclic gearing parts



You can generate several configuration of *epicyclic gearing system*:



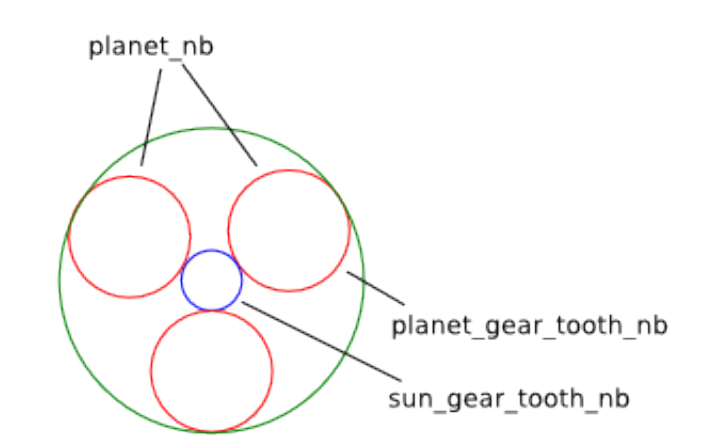
Epicyclic gearing examples

To get an overview of the possible epicyclic-gearing designs that can be generated by *epicyclic_gearing()*, run:

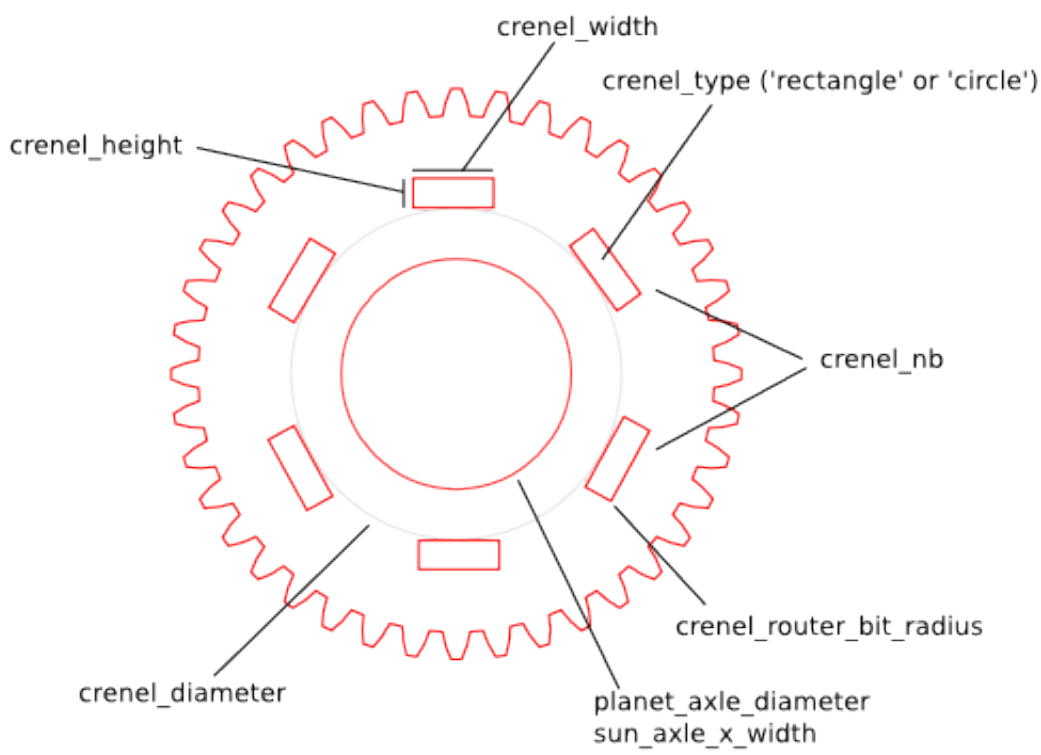
```
> python epicyclic_gearing.py --run_self_test
```

25.1 Epicyclic Gearing Parameter List

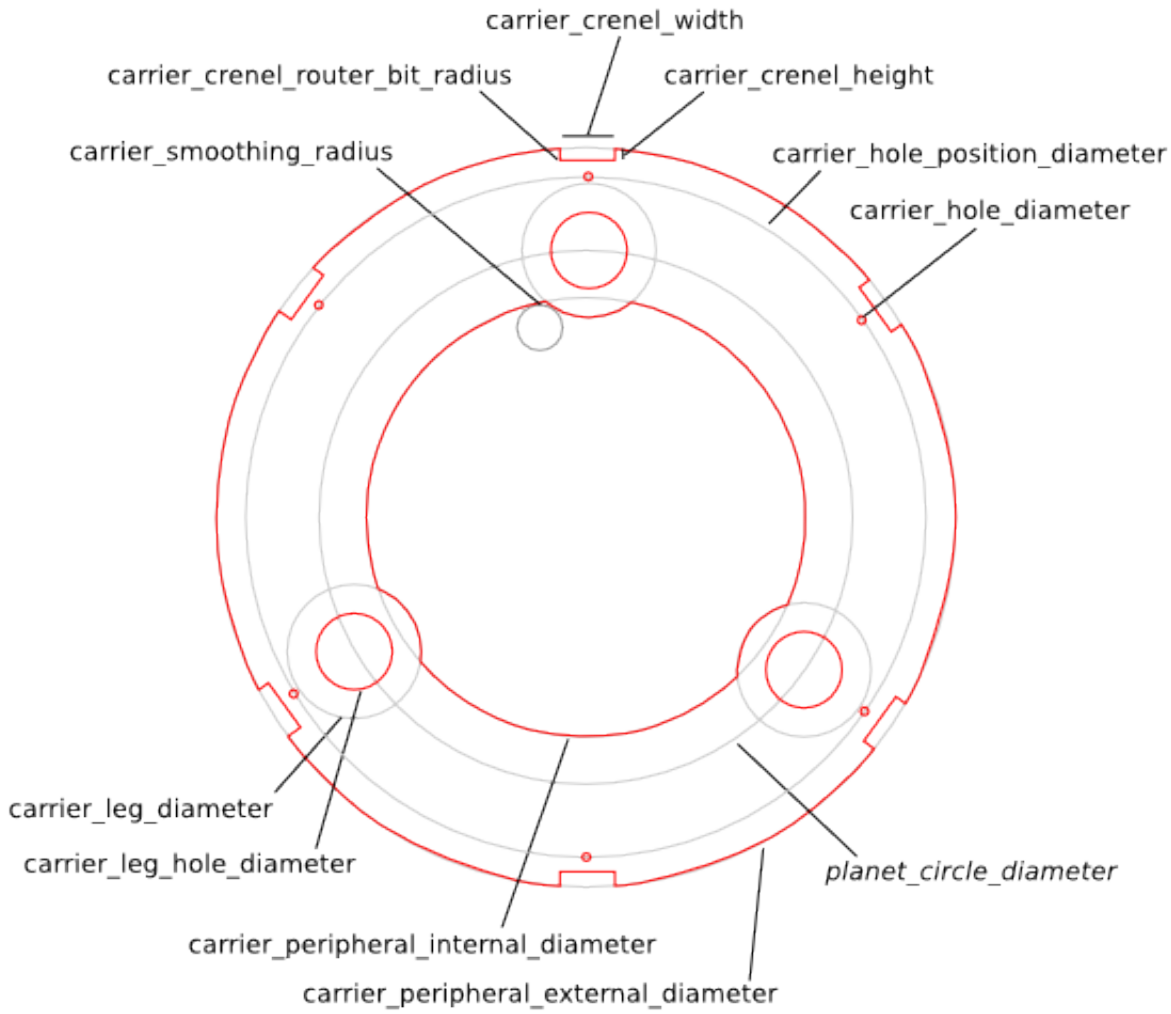
The parameter relative to the gearring are inherit from the Gearring Design.



Epicyclic gearing structure parameters

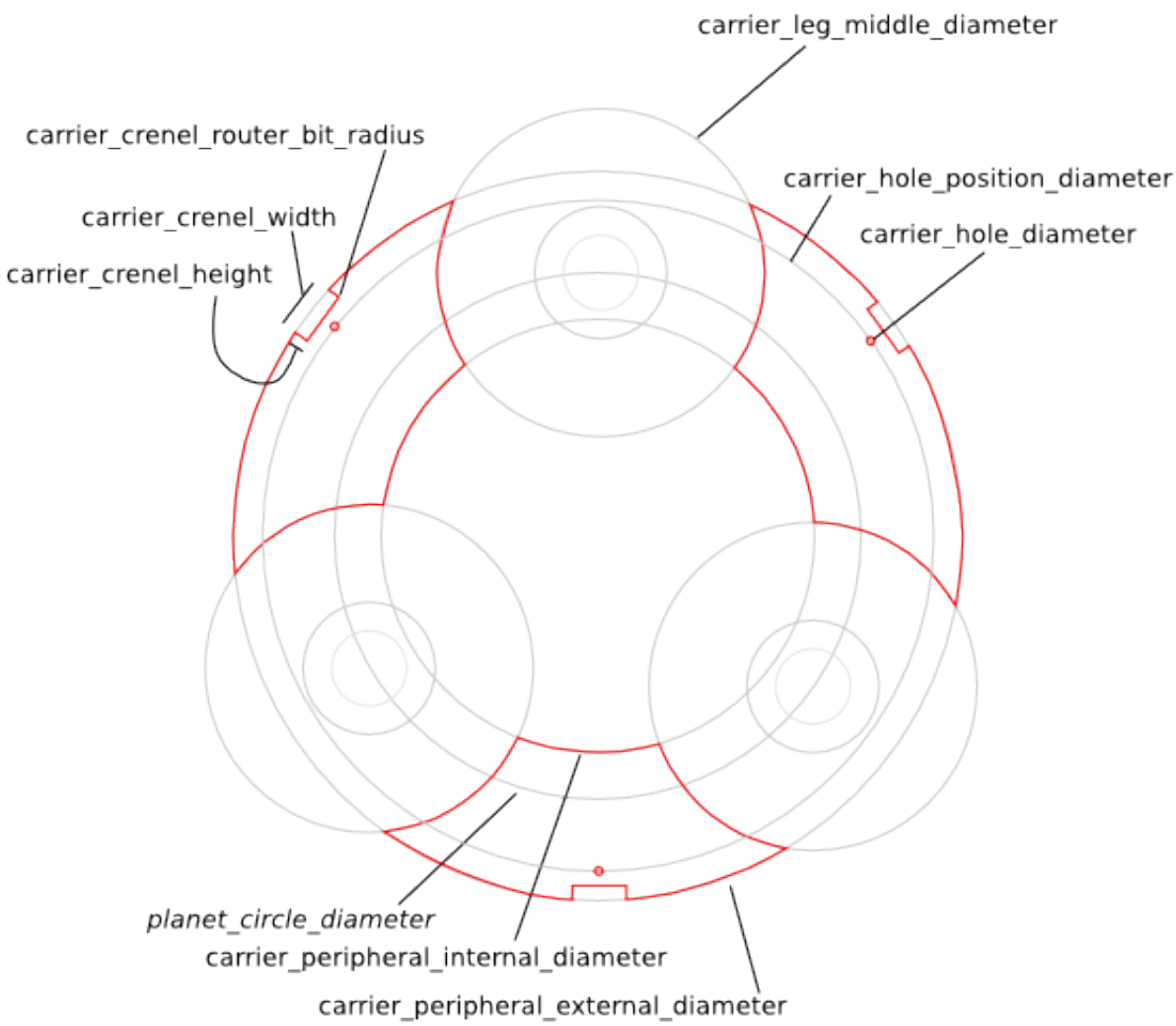


Sun and planet parameters



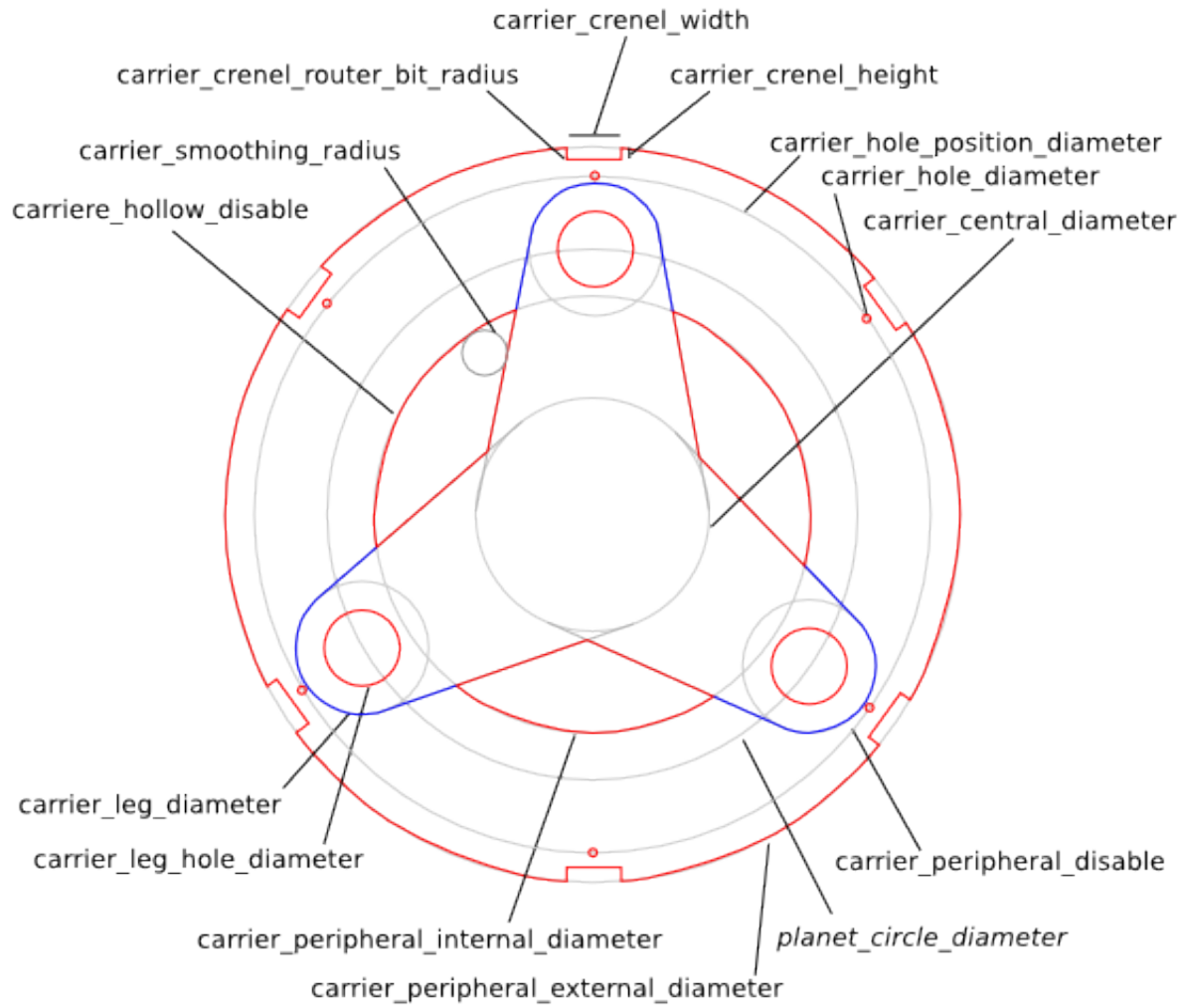
If carrier_peripheral_disable = True, no rear_planet_carrier is generated

Rear planet carrier parameters

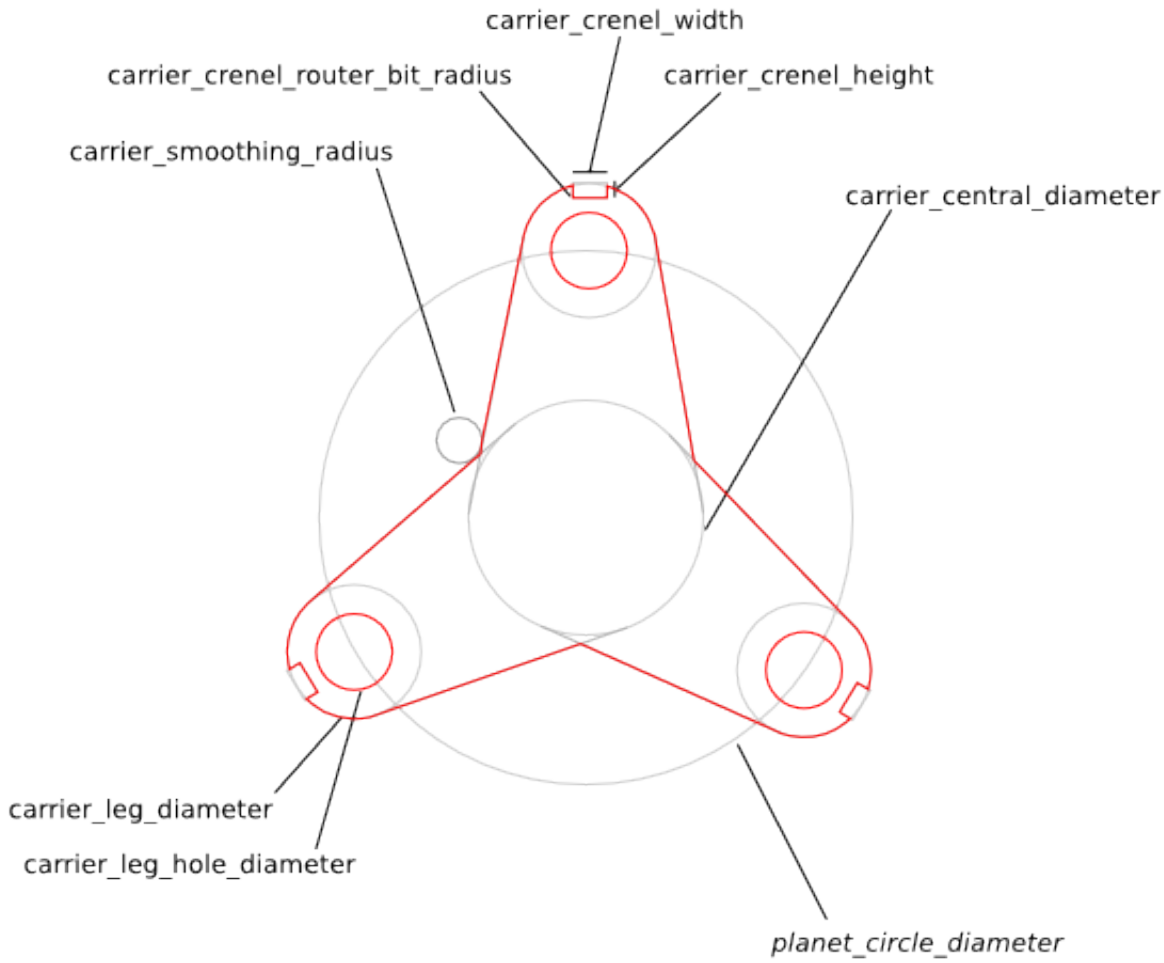


If carrier_peripheral_disable = True, no middle_planet_carrier is generated

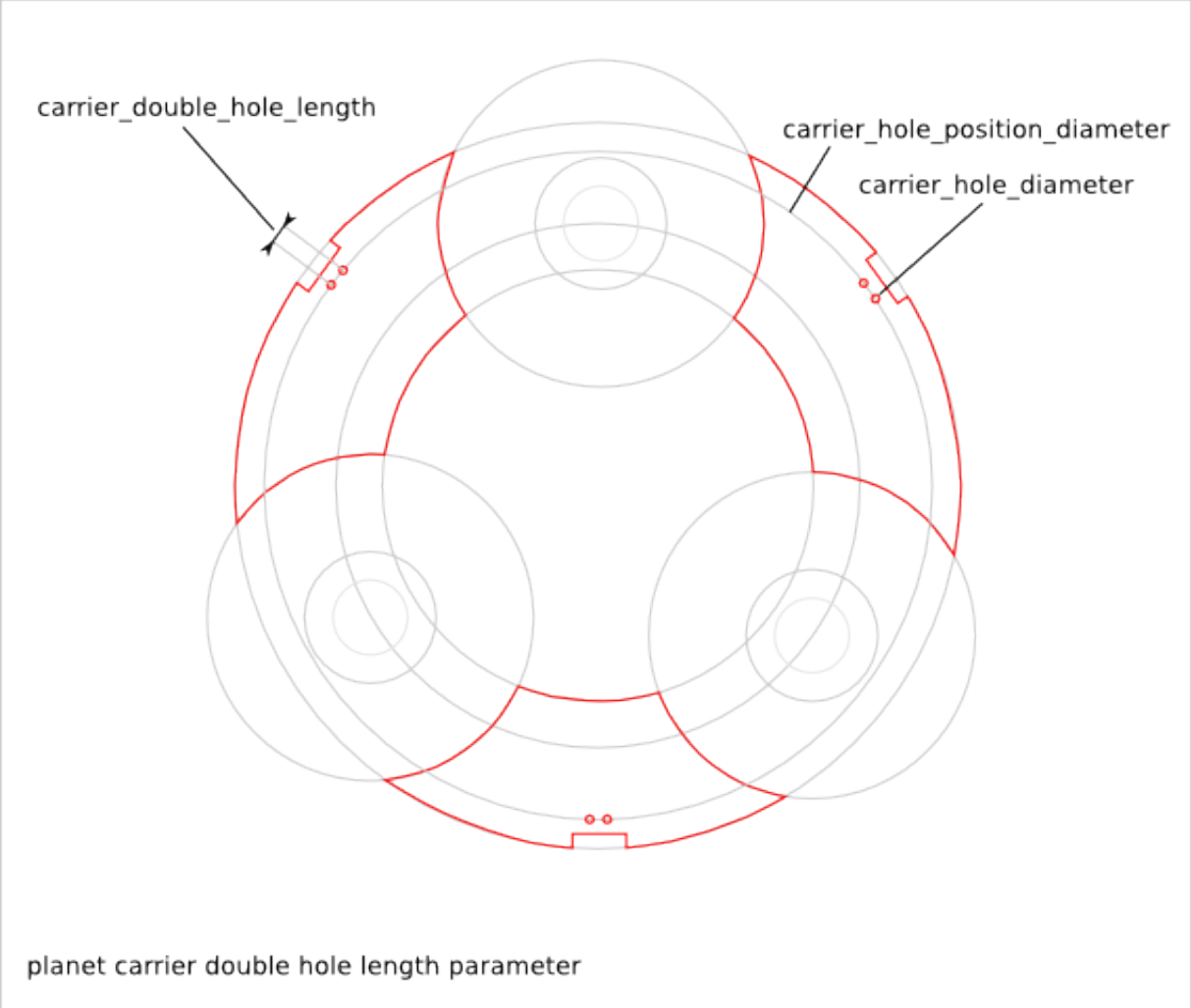
Middle planet carrier parameters



Front planet carrier parameters



Front planet carrier without peripheral



25.2 Epicyclic Gearring Parameter Dependency

25.2.1 router_bit_radius

Six router_bit radius are defined: *gear_router_bit_radius*, *sun_crenel_router_bit_radius*, *planet_crenel_router_bit_radius*, *carrier_crenel_router_bit_radius*, *carrier_smoothing_radius* and *cnc_router_bit_radius*. Each set the router_bit radius for different areas except *cnc_router_bit_radius* that set the minimum value for the five other router_bit radius. If an other router_bit radius is smaller than *cnc_router_bit_radius*, it is set to *cnc_router_bit_radius*. So, we have the relations:

```
cnc_router_bit_radius < gear_router_bit_radius
cnc_router_bit_radius < sun_crenel_router_bit_radius
cnc_router_bit_radius < planet_crenel_router_bit_radius
cnc_router_bit_radius < carrier_crenel_router_bit_radius
cnc_router_bit_radius < carrier_smoothing_radius
```

If you leave *carrier_smoothing_radius* to 0.0, it will be changed automatically to a default larger value.

25.2.2 sun_gear_tooth_nb and planet_gear_tooth_nb

sun_gear_tooth_nb and *planet_gear_tooth_nb* set the number of teeth of the sun-gear and planet-gears. The number of teeth of the annulus-gear is set to:

```
annulus_gear_tooth_nb = sun_gear_tooth_nb + 2 * planet_gear_tooth_nb
```

To get a working epicyclic-gearing, the sum of sun_gear_tooth_nb and annulus_gear_tooth_nb must be divisible by the number of planet-gears:

```
(annulus_gear_tooth_nb + sun_gear_tooth_nb) % planet_nb = 0
equivalent to:
(2*(sun_gear_tooth_nb + planet_gear_tooth_nb)) % planet_nb = 0
```

The transmission ration is equal to:

```
sun_gear_tooth_nb/(sun_gear_tooth_nb + annulus_gear_tooth_nb)
```

25.2.3 planet_nb

planet_nb sets the number of planet-gears. If *planet_nb* is set to 0, the maximal number of planet-gears is chosen.

25.2.4 carrier_peripheral_disable

If *carrier_peripheral_disable* is *True*, no rear-planet-carrier and no middle-planet-carrier are generated. The front-planet-carrier has also an alternative design.

25.2.5 carrier_hollow_disable

If *carrier_hollow_disable* is *True*, hollows are created in the front-planet-carrier. This remove some material to get a lighter system. This option is available only when *carrier_peripheral_disable* is *False*.

25.2.6 carrier_crenel_height

carrier_crenel_height sets the height of the carrier-crenels. If *carrier_crenel_height* is set to 0, the carrier-crenel are not created. The number of carrier-crenels is 2*planet_nb.

25.2.7 planet_axle_diameter and carrier_leg_hole_diameter

planet_axle_diameter and *carrier_leg_hole_diameter* are both related to the diameters of the planet-gear axle. *planet_axle_diameter* sets the diameter of the axle of the planet-gears. *carrier_leg_hole_diameter* sets the diameter of the corresponding coaxial holes in the rear and front planet-carrier. Using two different values for these two parameters can be useful when you want to use a ball bearing system.

25.2.8 sun axle and carrier axle design

The sun axle design is defined with several parameters such as *sun_axle_diameter*, *sun_crenel_diameter*, *sun_crenel_nb*, *sun_crenel_width*, *sun_crenel_height* and *sun_crenel_router_bit_radius*. The design of the axle of the plant-carrier is copied from the sun axle design. So there is no parameters directly related to the planet-carrier axle design. Notice that in case of cascade epicyclic gearing, the planet-carrier of a stage intends to be jammed to the sun-gear of the next stage.

25.2.9 carrier_double_hole_length

The crenel-hole can is replaced by a double-crenel-hole when *carrier_double_hole_length* is set to a float bigger than zero. In this case, two holes are created with a distance of *carrier_double_hole_length*. Double-hole are useful to increase the stability of the planet-carrier.

25.2.10 top_lid parameters

Those parameters are inherit from Axle Lid Design

25.2.11 input and output gearwheels

The *epicyclic-gearing* design can generate the input and the output gearwheels. It is recommended to re-generate those gearwheels with the *gearwheel.py* script to get access to the complete Gearwheel Design parameter list.

25.3 Epicyclic Gearing Recommendations

25.3.1 For laser-cutter

The laser-cutter remove usually more material than the ideal line. This is because of the lase beam width. To get a well adjusted gear system without too much play, we need to compensate this excess of removed material. The parameter *gear_skin_thickness* lets you move the gear-profile-outline in order to compensate the laser beam width. Because the laser remove too much material, you should set *gear_skin_thickness* to a positive values (e.g: 0.75 mm).

If you set a quiet large value to *gear_skin_thickness*, it may happens that the gear-ring (a.k.a. annulus) can not be generated any more because its bottom-land is too small or even negative. In this case, there is a small workaround: modify slightly the lowest part of the dedendum of the gear-ring to make this gear-hollow feasible by using the parameter *gearring_dedendum_to_hollow_pourcentage*. For example, if *gearring_dedendum_to_hollow_pourcentage* is set to 10, 10% of the gear-ring dedendum is changed into the gear-hollow.

gear_skin_thickness does not compensate the height of the gear-teeth. If you think the laser-cutter make the gear-teeth too small, you can increase the value of the parameter *gear_addendum_height_pourcentage*. For example, if you set *gear_addendum_height_pourcentage* to 110, the theoretical (before laser-cutting) gear-tooth-addendum height is set to 1.1*gear_module.

25.3.2 For 3D printing

Usually 3D printed parts are a bit larger than the CAD design. This is because of the extruded wire width. This extra thickness can be compensated with a negative value sets to the parameter *gear_skin_thickness*.

If you set a too large negative value to *gear_skin_thickness*, the top the gear-tooth might not be designable anymore because the top-land will be negative. In this case, you can reduce the height of the gear-tooth addendum with the parameter *gear_addendum_height_pourcentage*. For example, if you set *gear_addendum_height_pourcentage* to 90, the theoretical (without the extra extruded wire width) gear-tooth-addendum height is set to 0.9*gear_module.

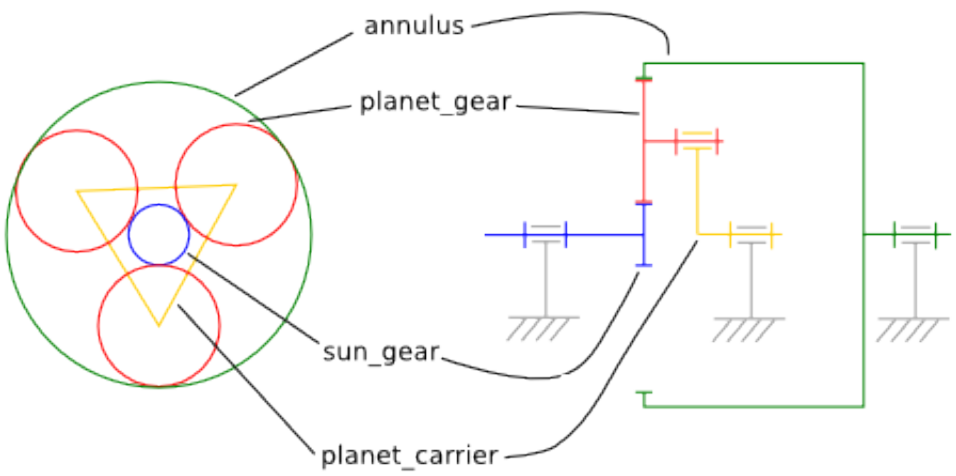
25.3.3 For CNC milling

With CNC, the biggest challenge is the size to the router-bit. *cnc_router_bit_radius* must be equal or bigger than the effective used router-bit radius. If *gear_router_bit_radius* is smaller than *cnc_router_bit_radius*, it is automatically sets to the value of *cnc_router_bit_radius*.

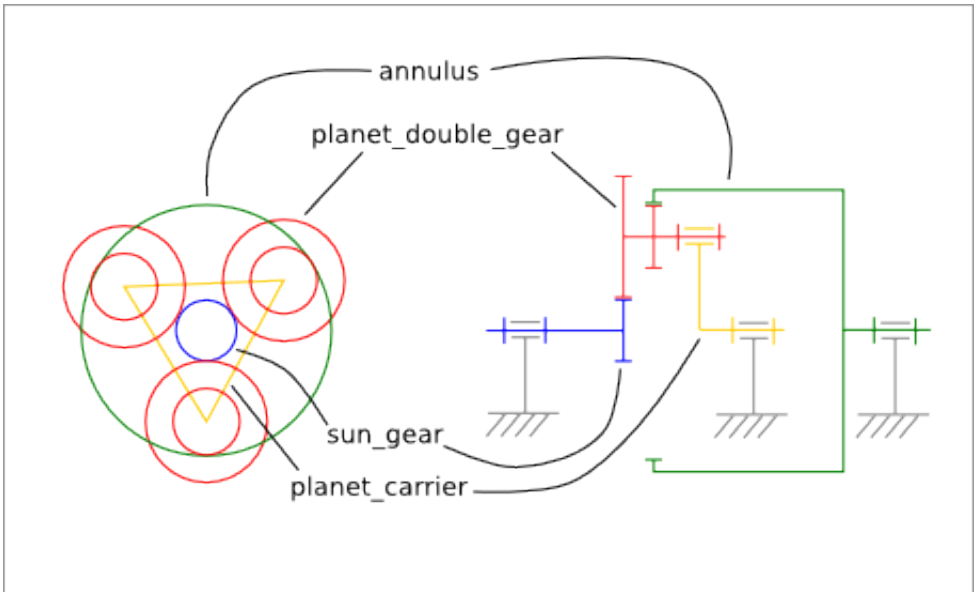
If *gear_router_bit_radius* is too large, it may happens that the gear-ring can not be generated anymore because the gear_router_bit_radius is too large compare to the gear-hollow width. In this case, there is a small workaround: modify slightly the lowest part of the dedendum of the gear-ring to make this gear-hollow feasible by using the parameter *gearring_dedendum_to_hollow_pourcentage*. For example, if *gearring_dedendum_to_hollow_pourcentage* is set to 10, 10% of the gear-ring dedendum is changed into the gear-hollow.

CHAPTER 26

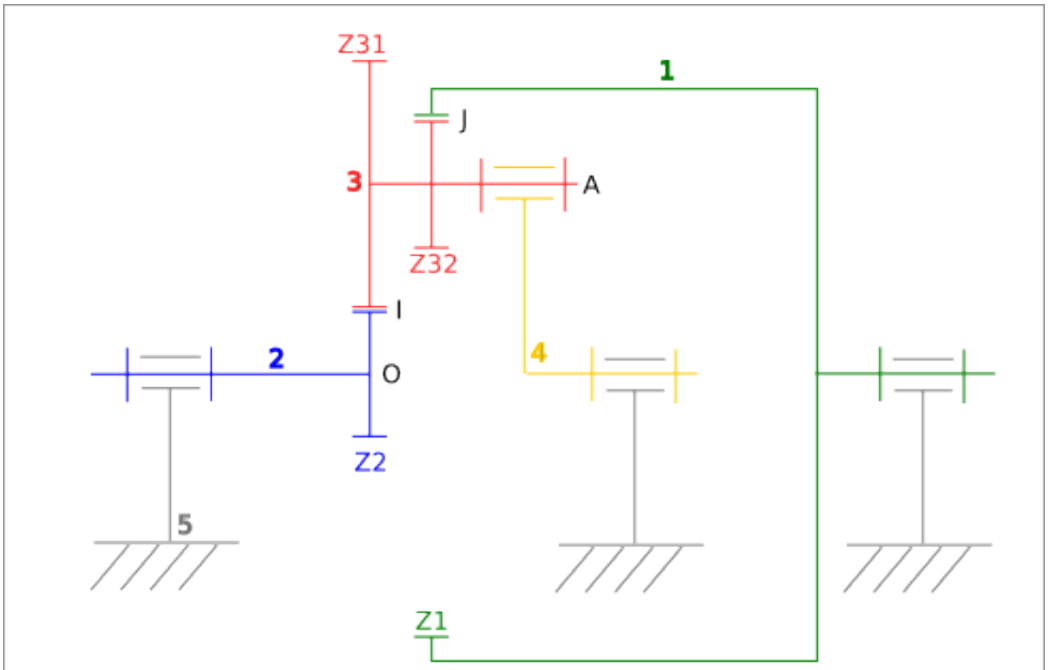
Epicyclic Gearing Details



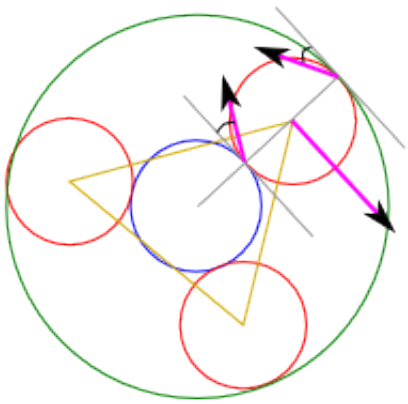
epicyclic gearing with simple gear planets



epicyclic gearing with double gear planets



epicyclic gearing geometry



To = Torque at output
Ti = Torque at input
R2 = radius of the sun-gear
R3 = radius of the planet-gear
N = number of planets
fa = force_angle
F = force on the gear-tooth

F = To/(2*N*(R2+R3)*cos(fa))

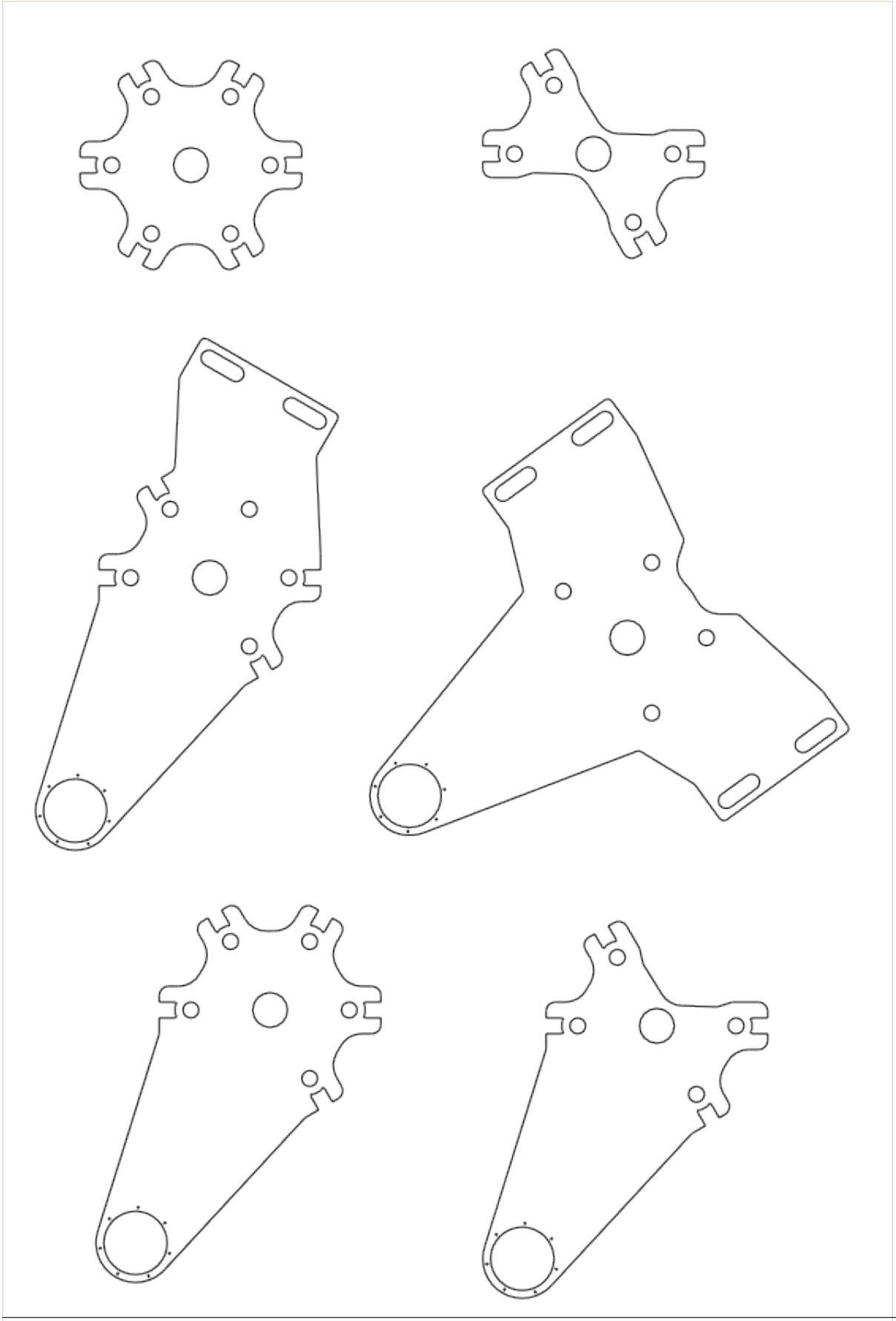
Epicyclic gearing forces

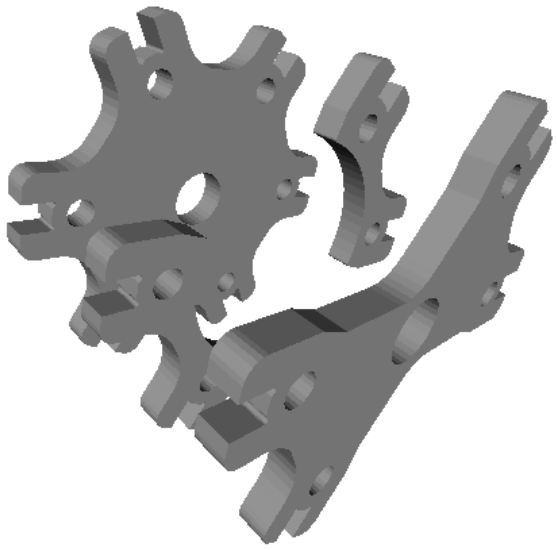
CHAPTER 27

Axle Lid Design

Ready-to-use parametric *axle_lid* design kit. The *axle_lid* is a an assembly of three parts:

- annulus-holder
- middle-axle-lid
- top_axle-lid



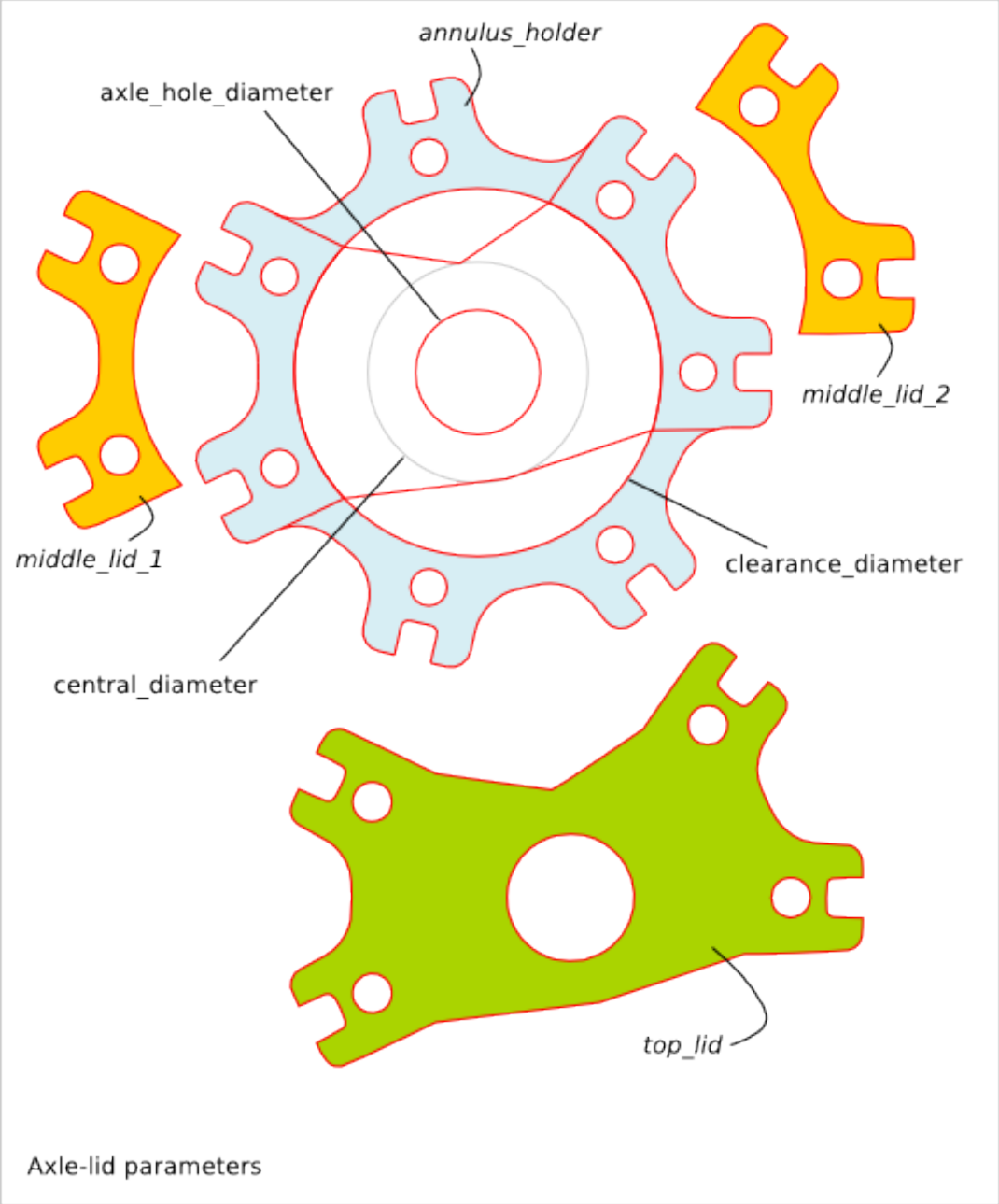


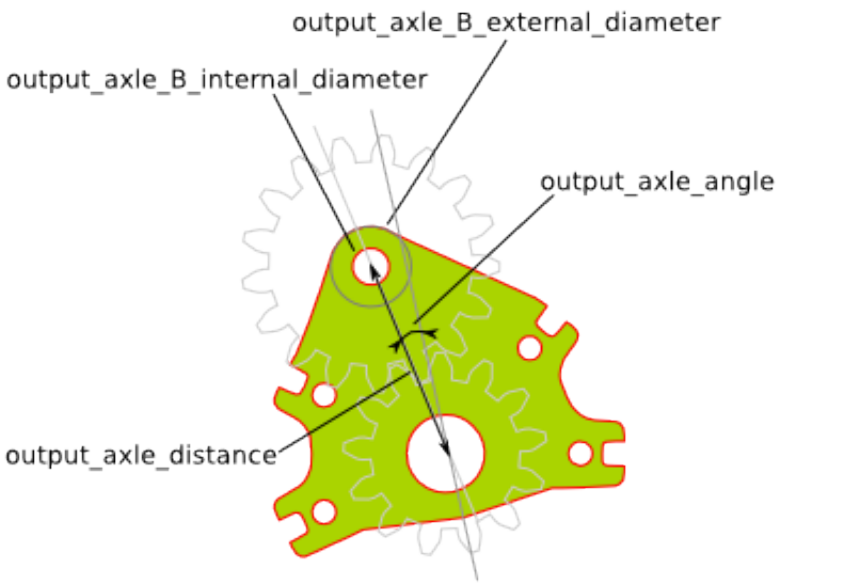
To get an overview of the possible axle_lid designs that can be generated by *axle_lid()*, run:

```
> python axle_lid.py --run_self_test
```

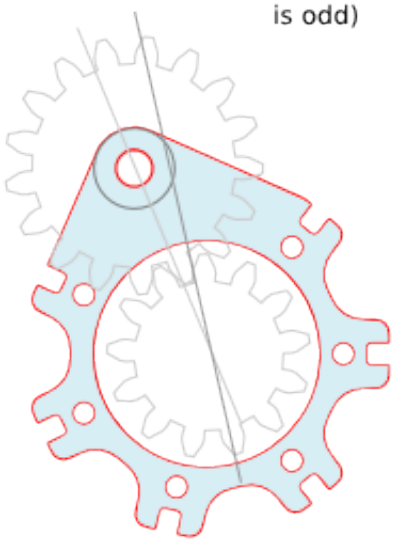
27.1 Axle-lid Parameter List

The parameter relative to the external outline are inherit from the Gearring Design.

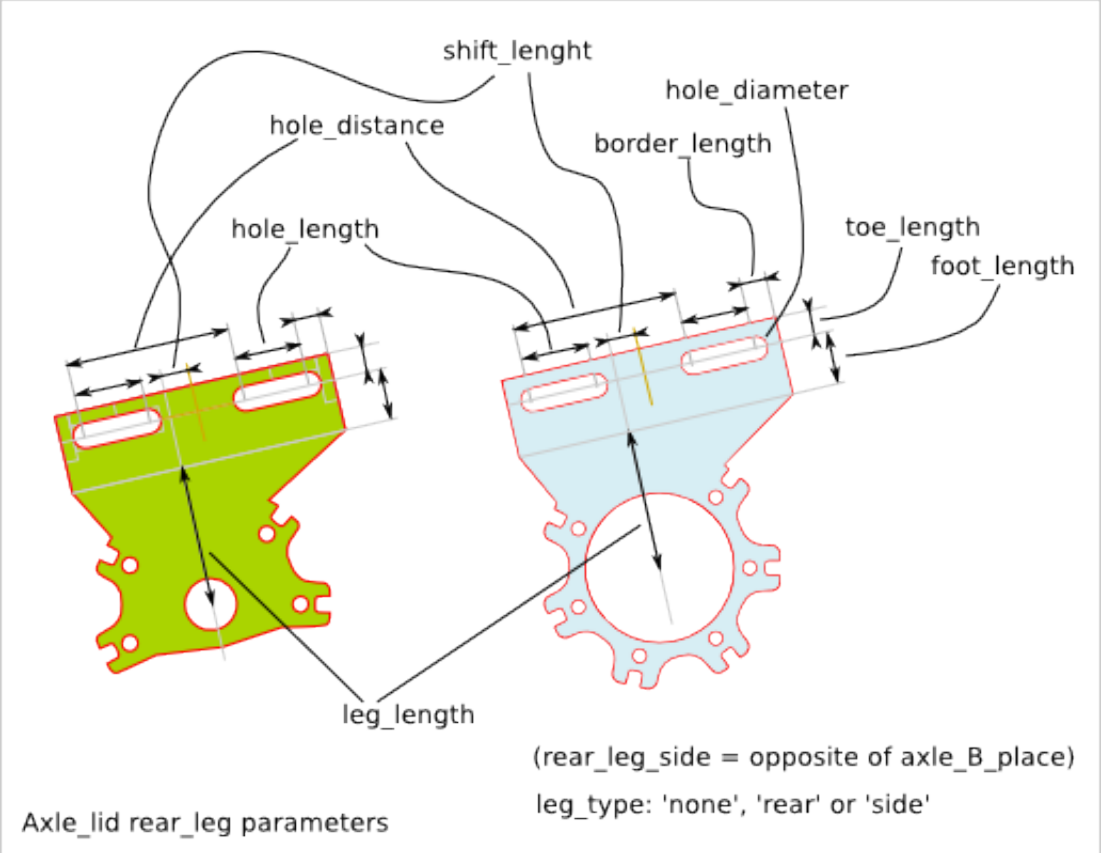


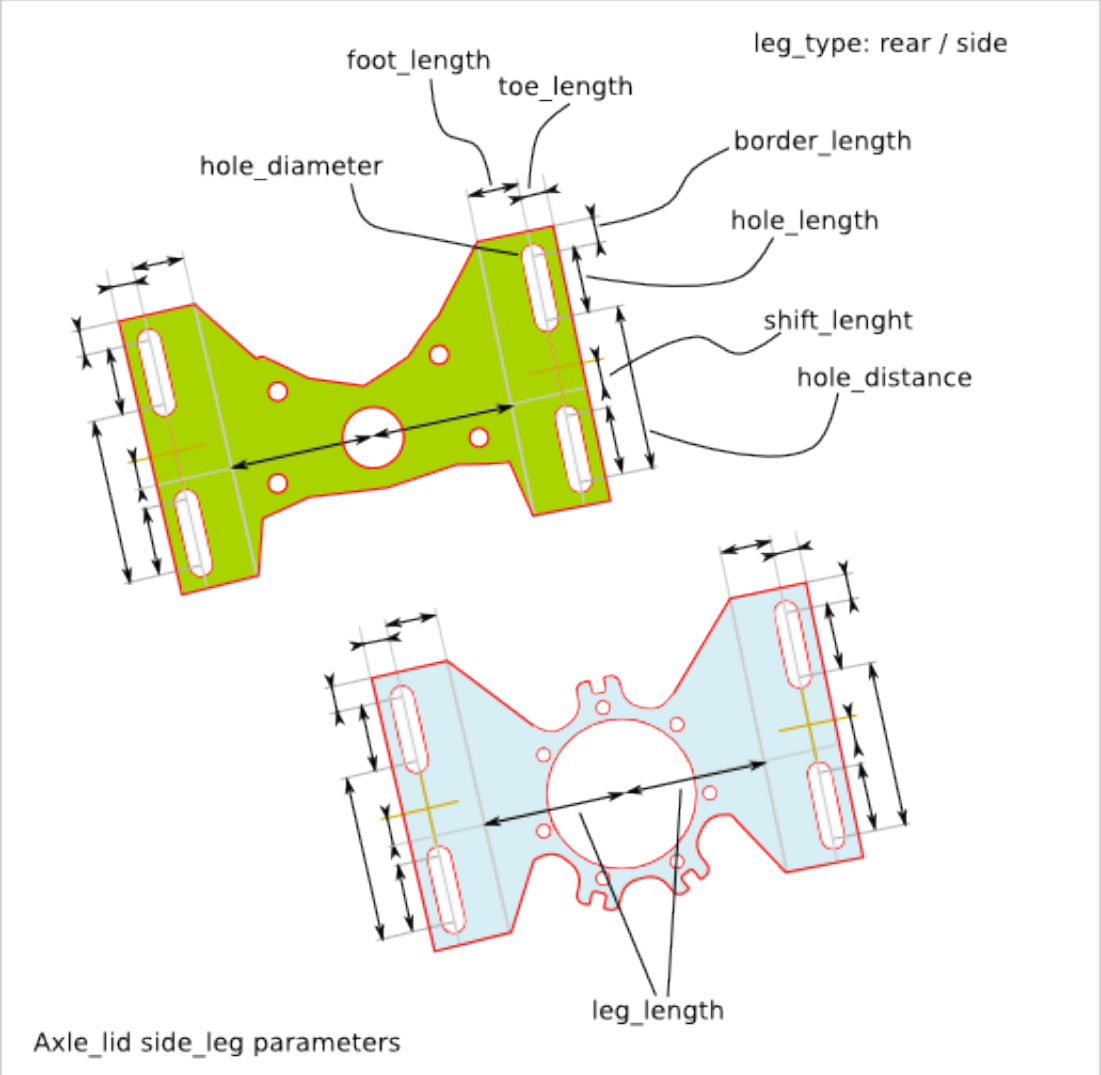


axle_B_place: 'none', 'small' or 'large'
(only useful when the number of crenels
is odd)



Axle_lid axle_B parameters





27.2 Axle-lid Parameter Dependency

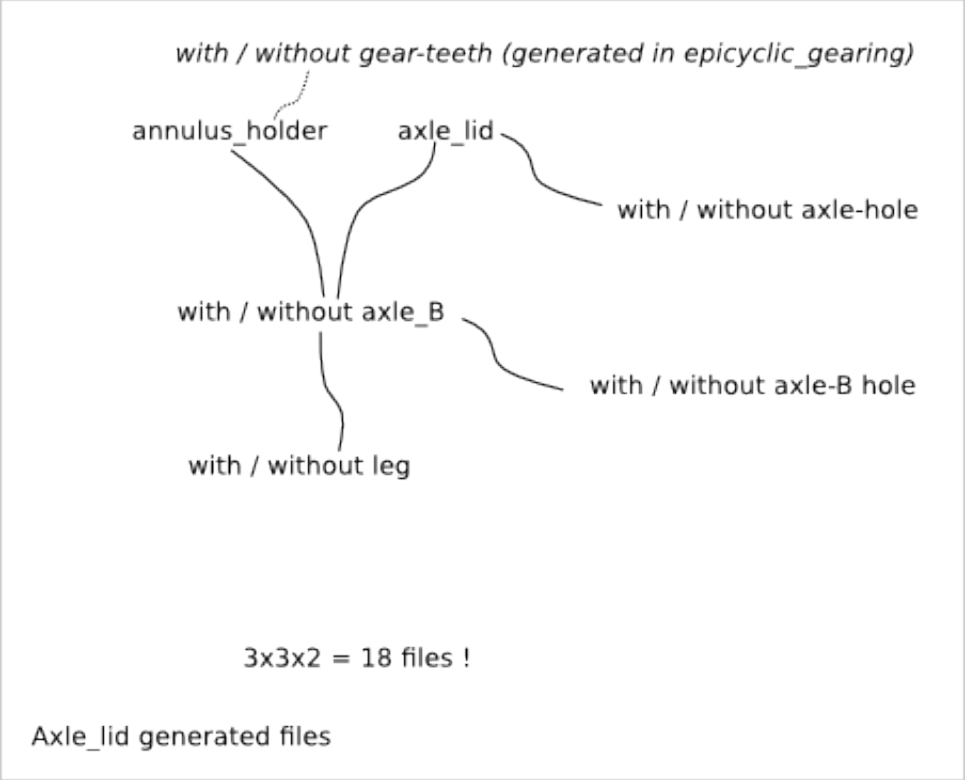
27.2.1 Diameters

The following relations between diameters (or radius) must be respected:

```
cnc_router_bit_radius < axle_hole_diameter/2
axle_hole_diameter    < central_diameter
central_diameter      < clearance_diameter
clearance_diameter    < holder_diameter
```

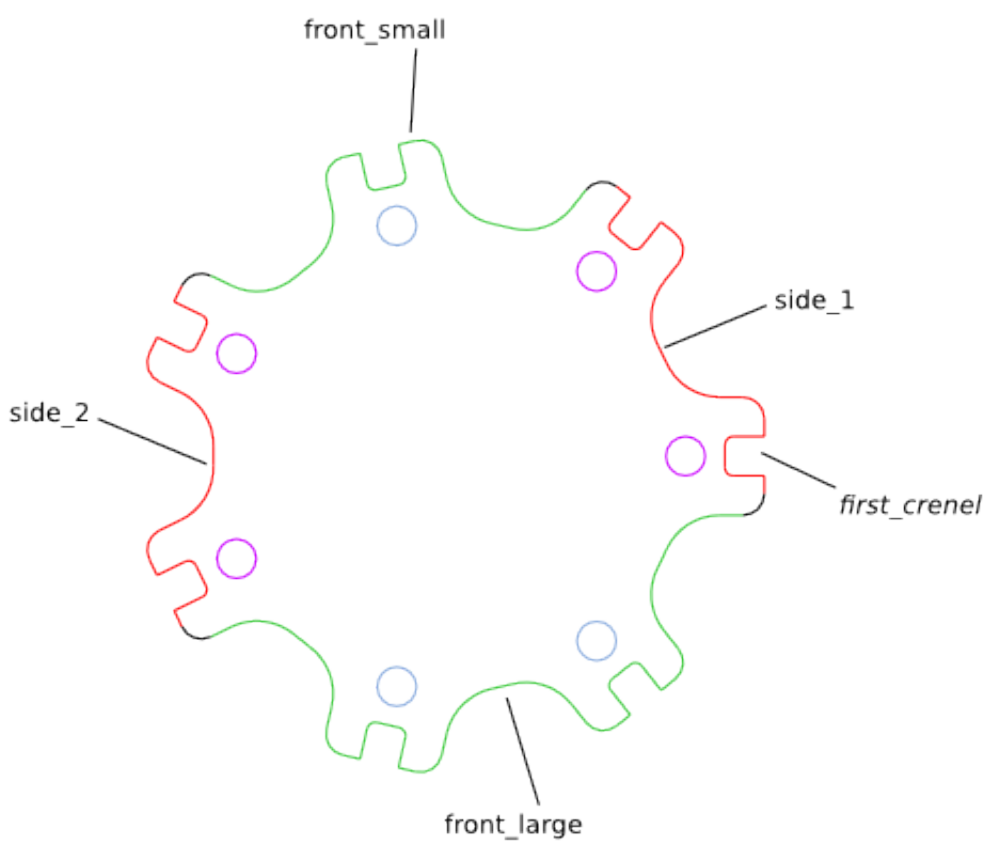
27.2.2 Generated files

For a same set of parameters, you may need several flavour of the design such as a plate with a hole and the same plate without this hole. Instead of adding input parameters to select if the plate must have a hole or not, the both variants are generated. You just need to pick up the file you need.

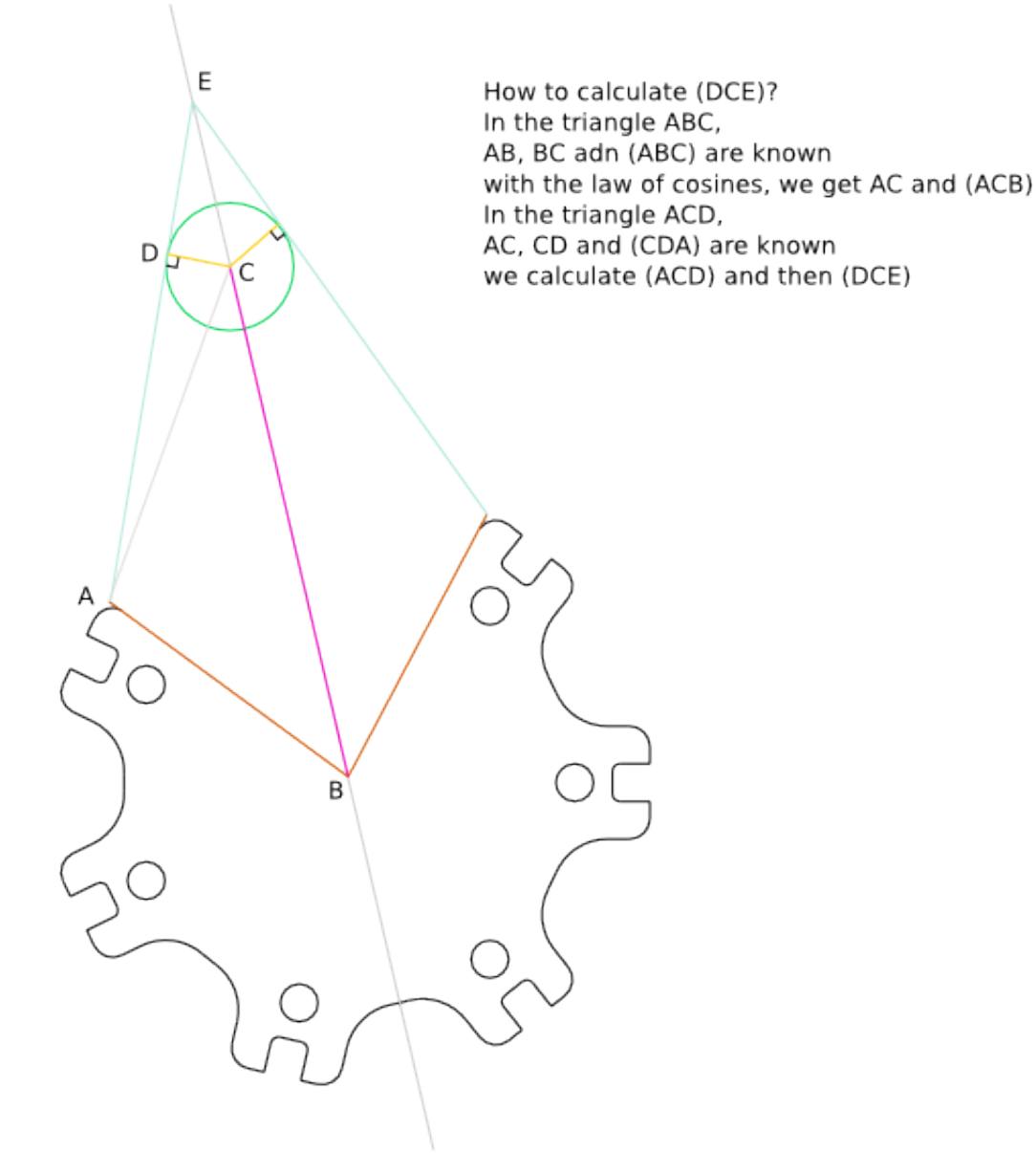


CHAPTER 28

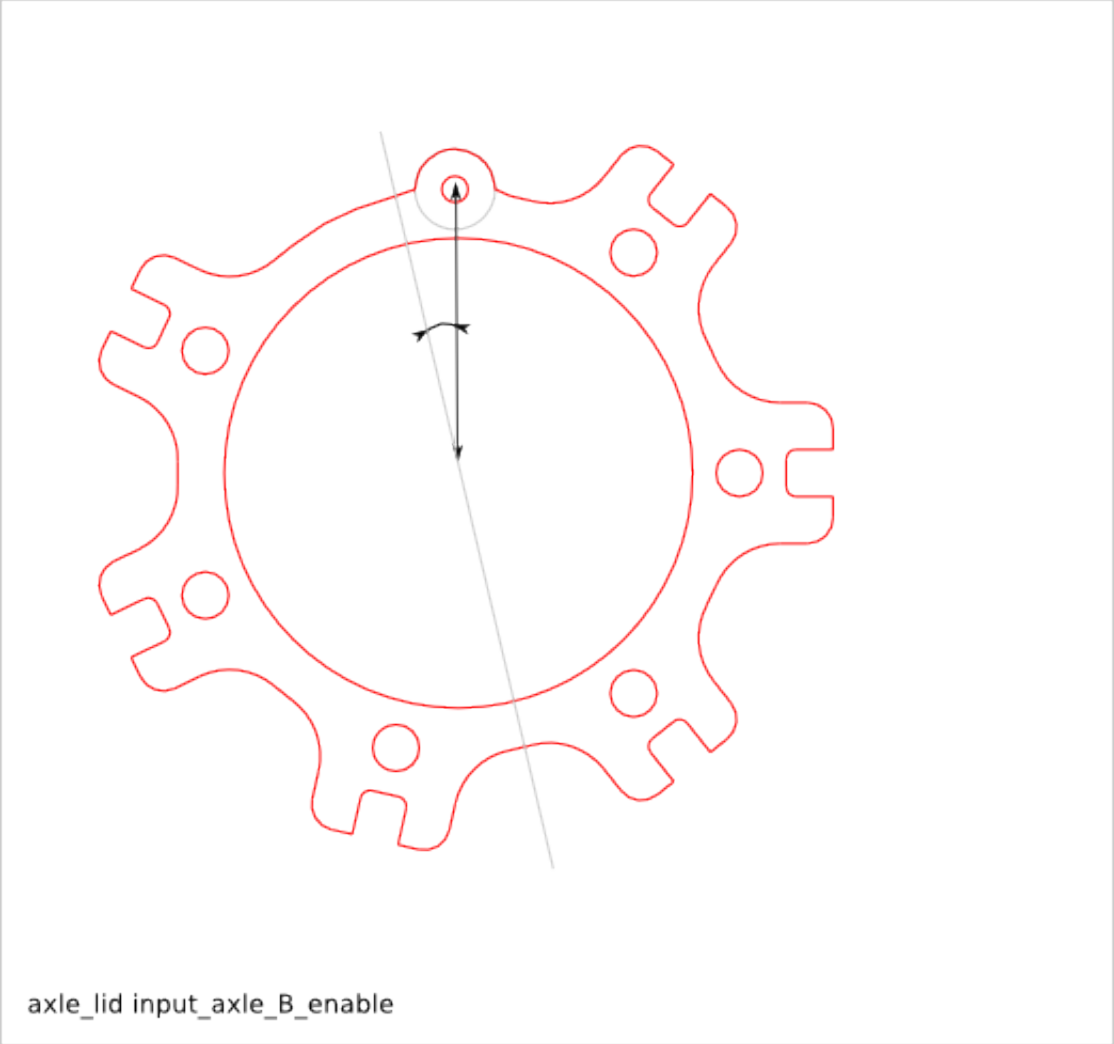
Axle_lid Details



axle_lid holder_cut outlines



axle_lid axle_B external circle tangent

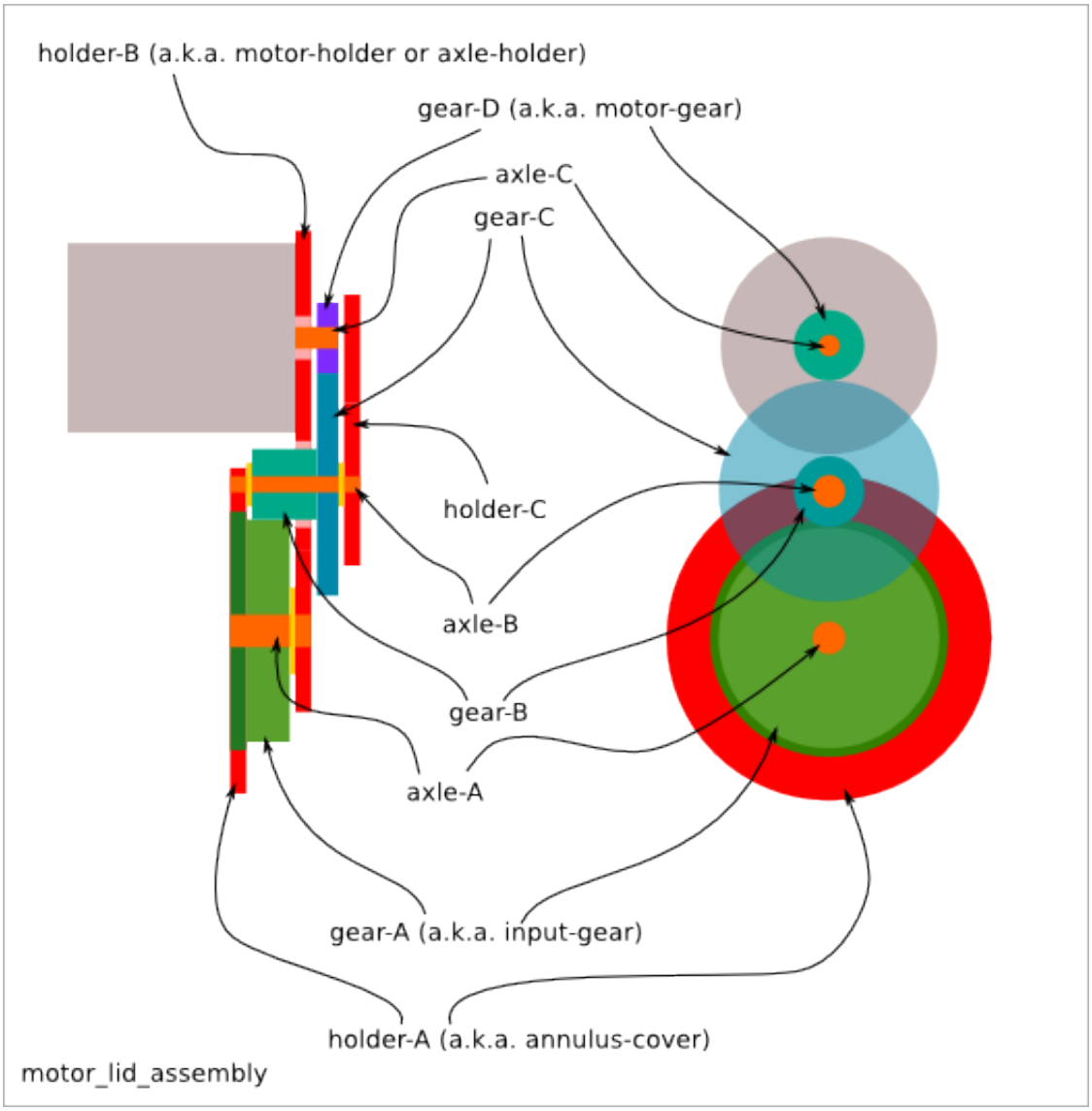


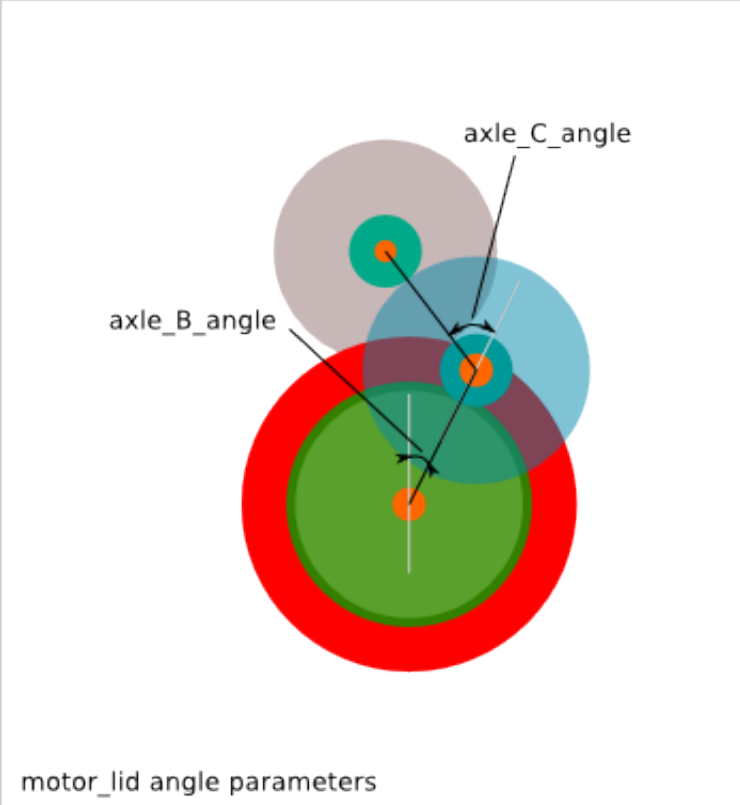
CHAPTER 29

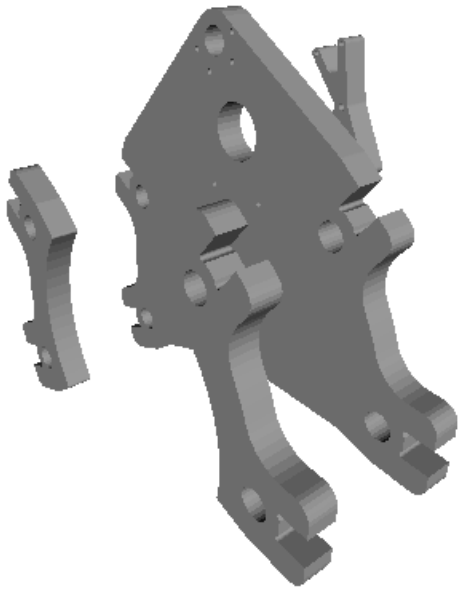
Motor Lid Design

Ready-to-use parametric *motor_lid* assembly. This assembly aims at holding the gear system between an electric motor and the epicyclic-gearing. The *motor_lid* is an assembly of several parts:

- holder-A (a.k.a. annulus-holder)
- holder-B (a.k.a. motor-holder or axle-holder)
- holder-C

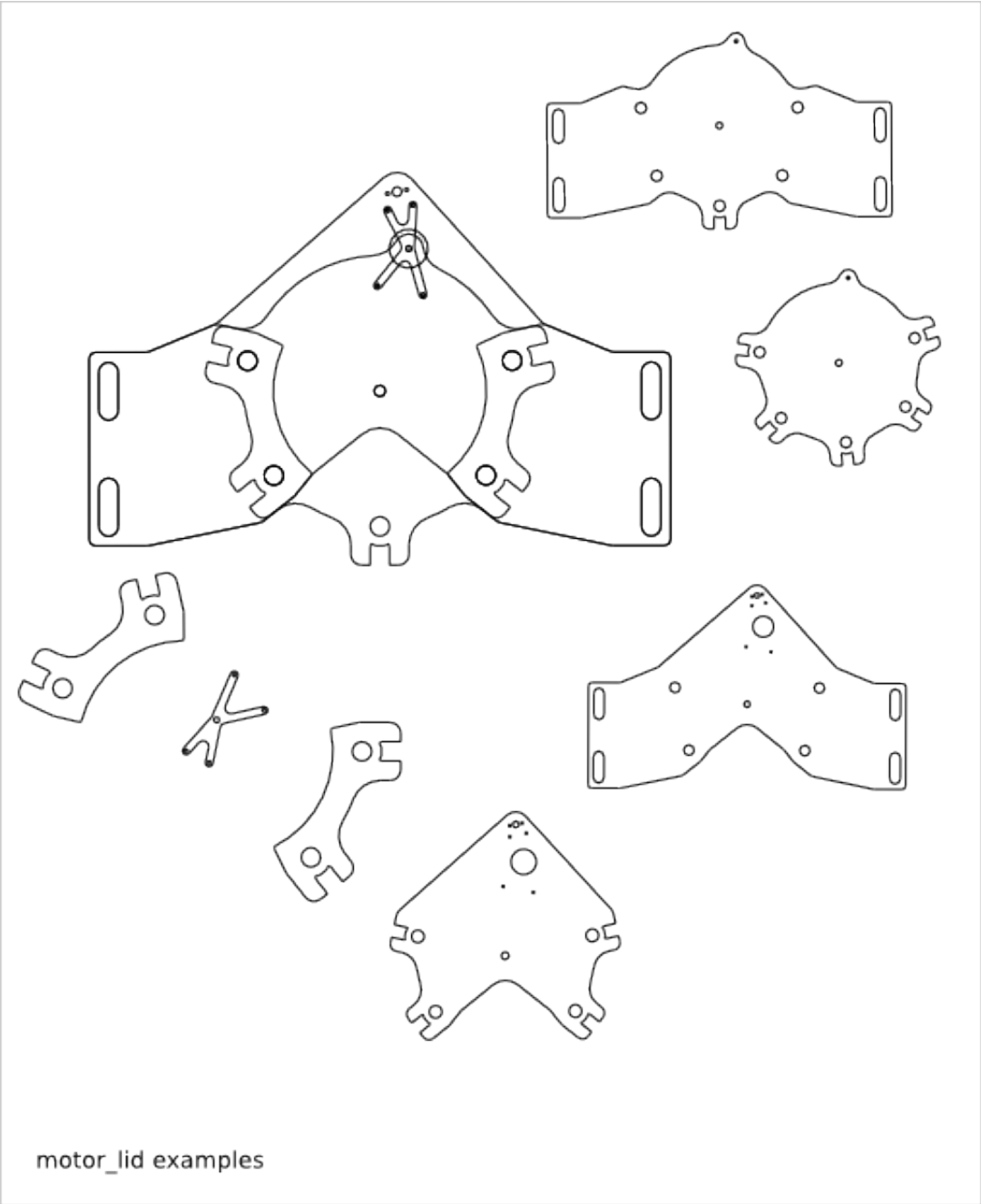






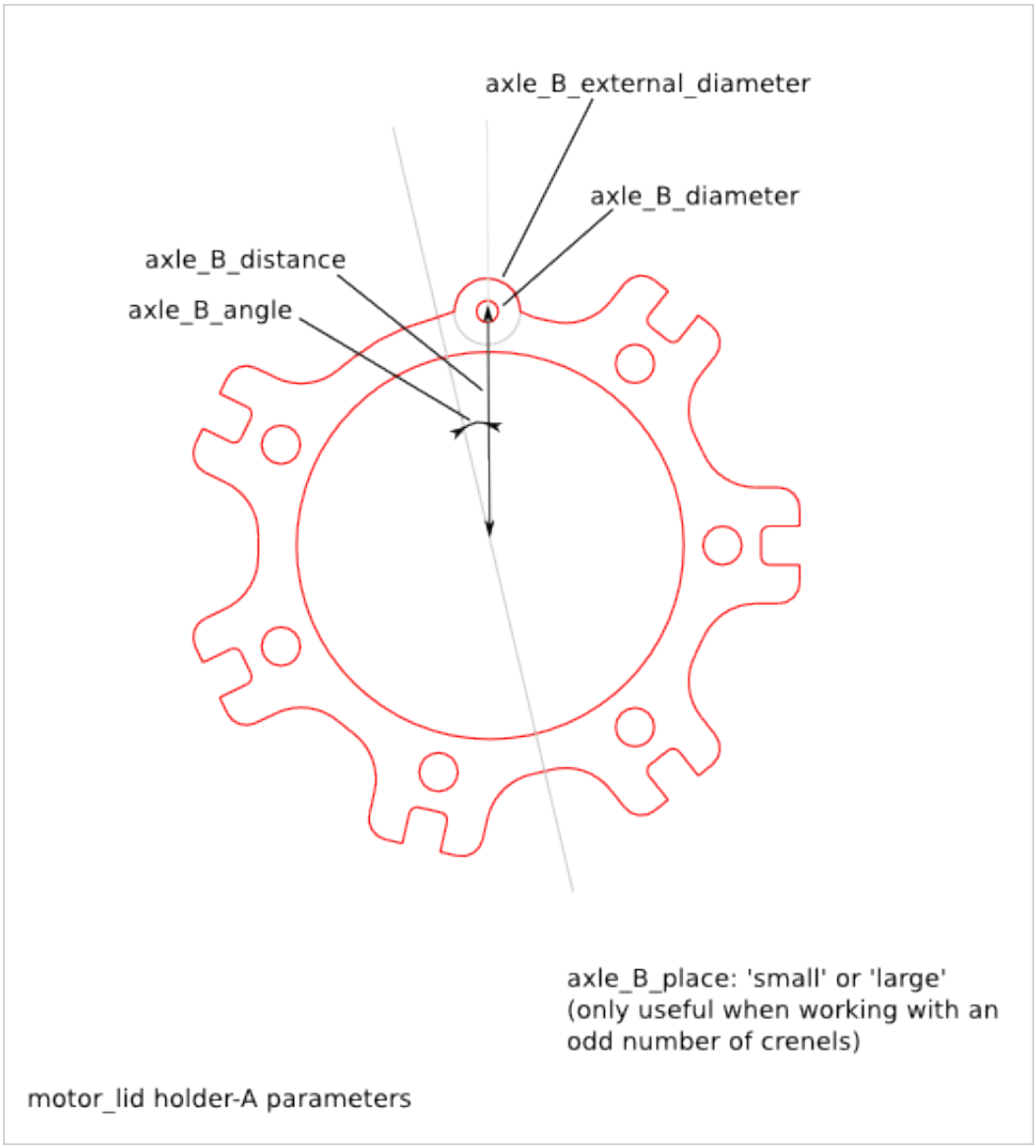
To get an overview of the possible motor_lid designs that can be generated by *motor_lid()*, run:

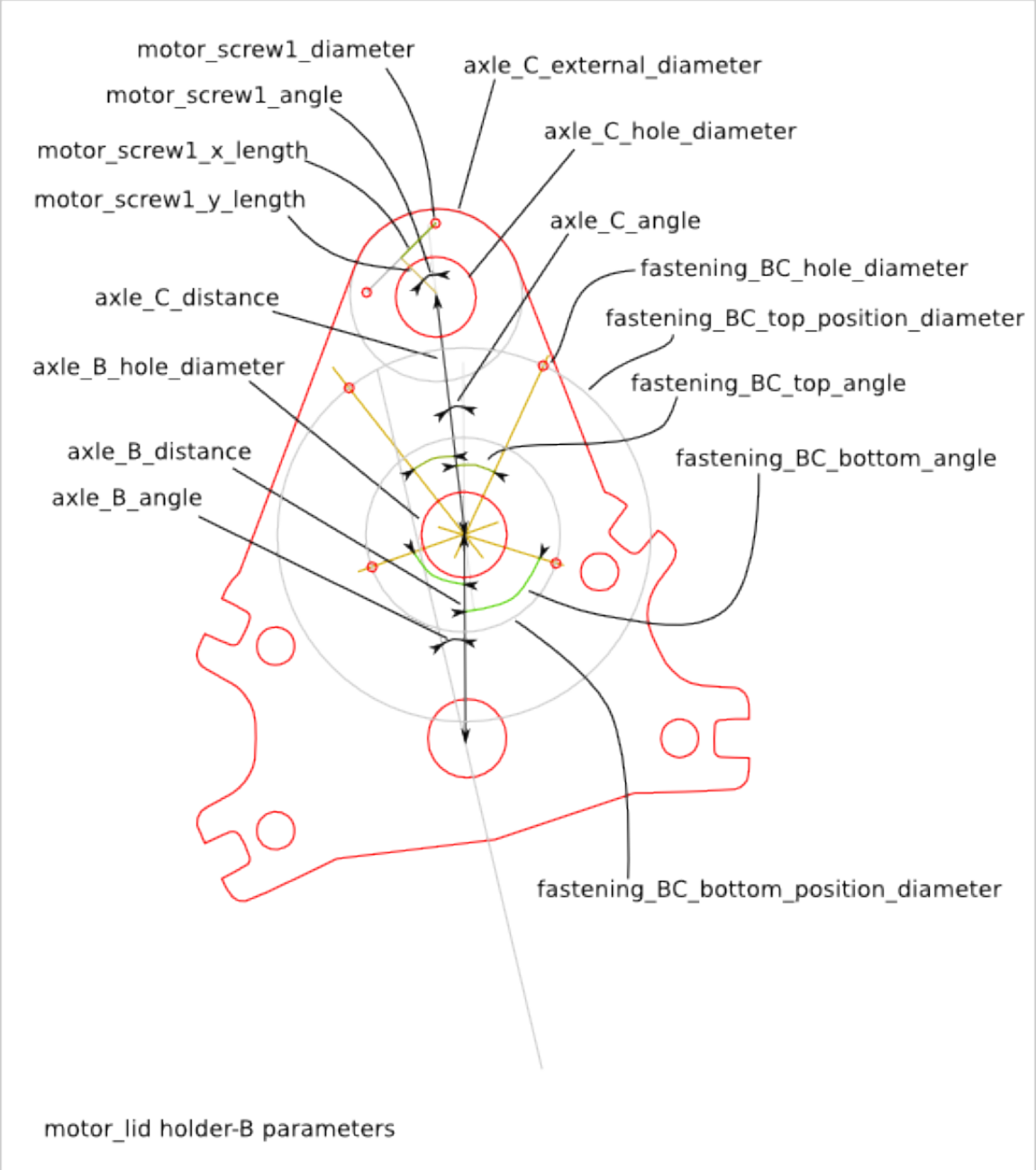
```
> python mostor_lid.py --run_self_test
```

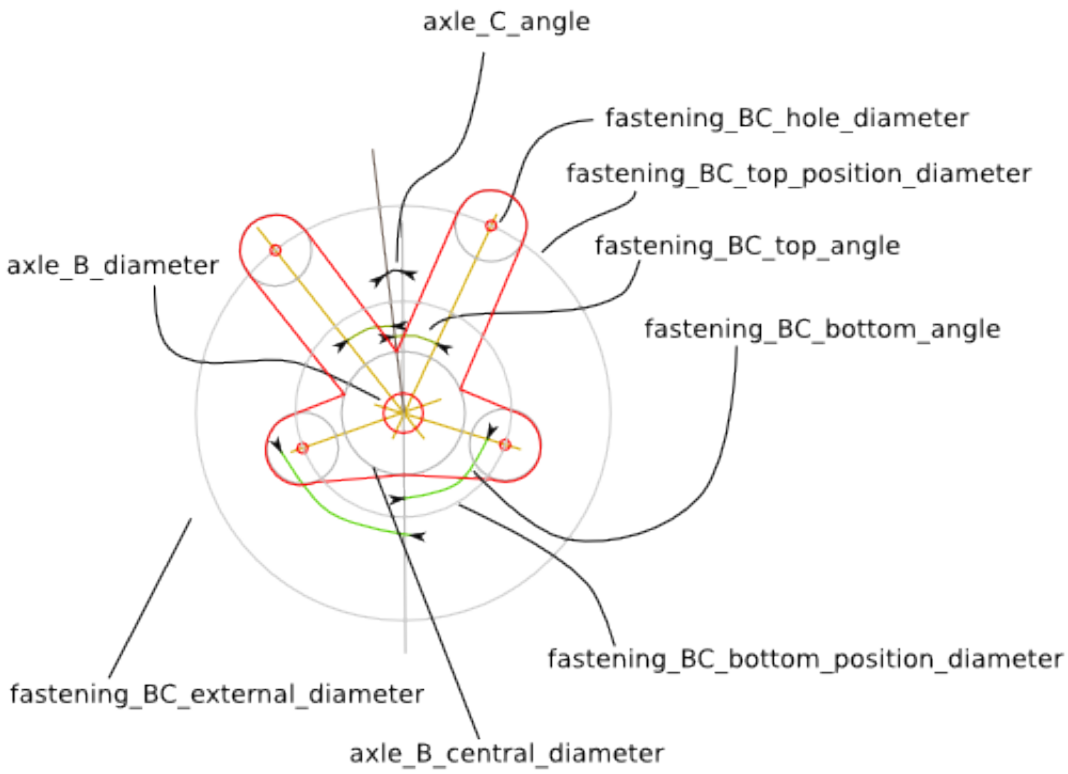


29.1 Motor-lid Parameter List

The parameter relative to the external outline are inherit from the Gearring Design and Axle Lid Design.





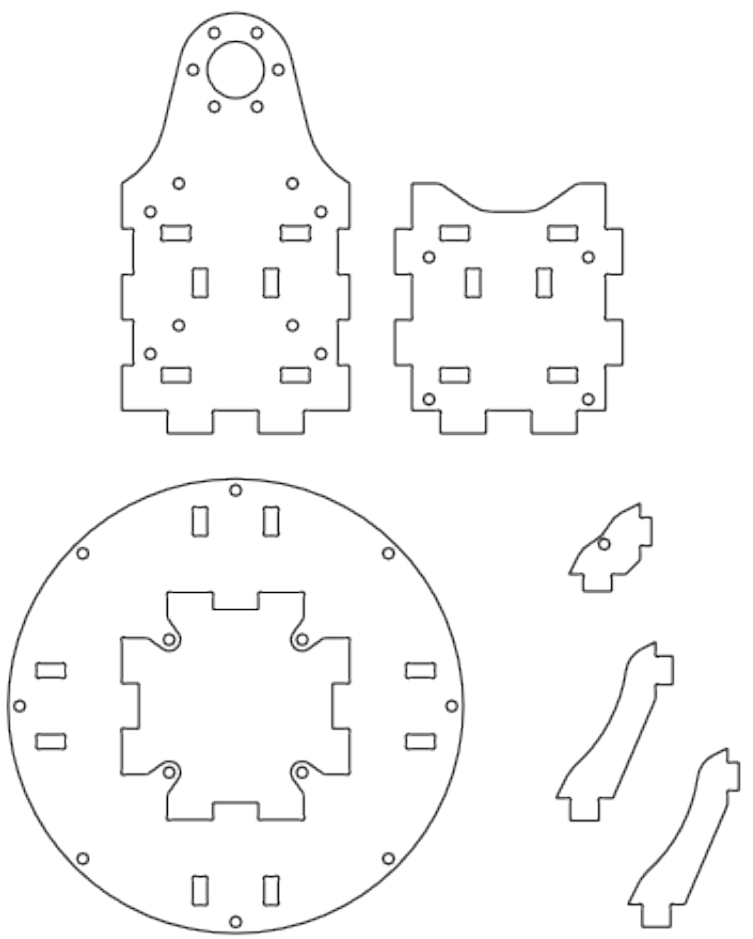


motor_lid holder-C parameters

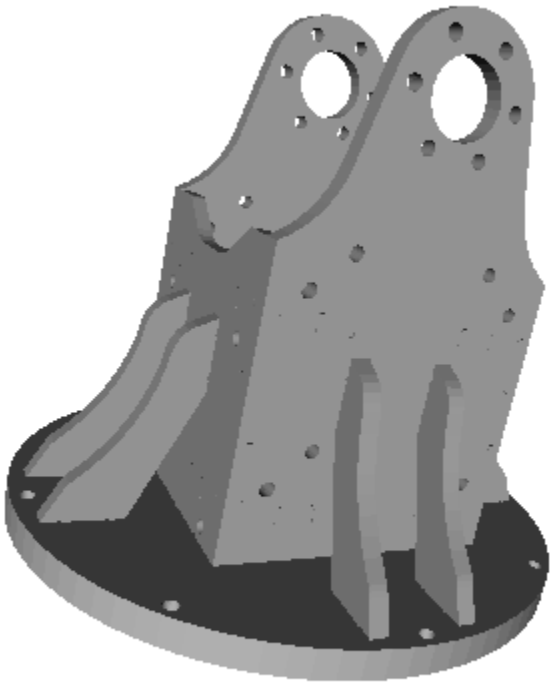
CHAPTER 30

Bell Design

Ready-to-use parametric *bell* design. It is the extremity piece of a *gimbal* assembly. The *bell* piece is composed of several flat parts fixed together.



Bell part example



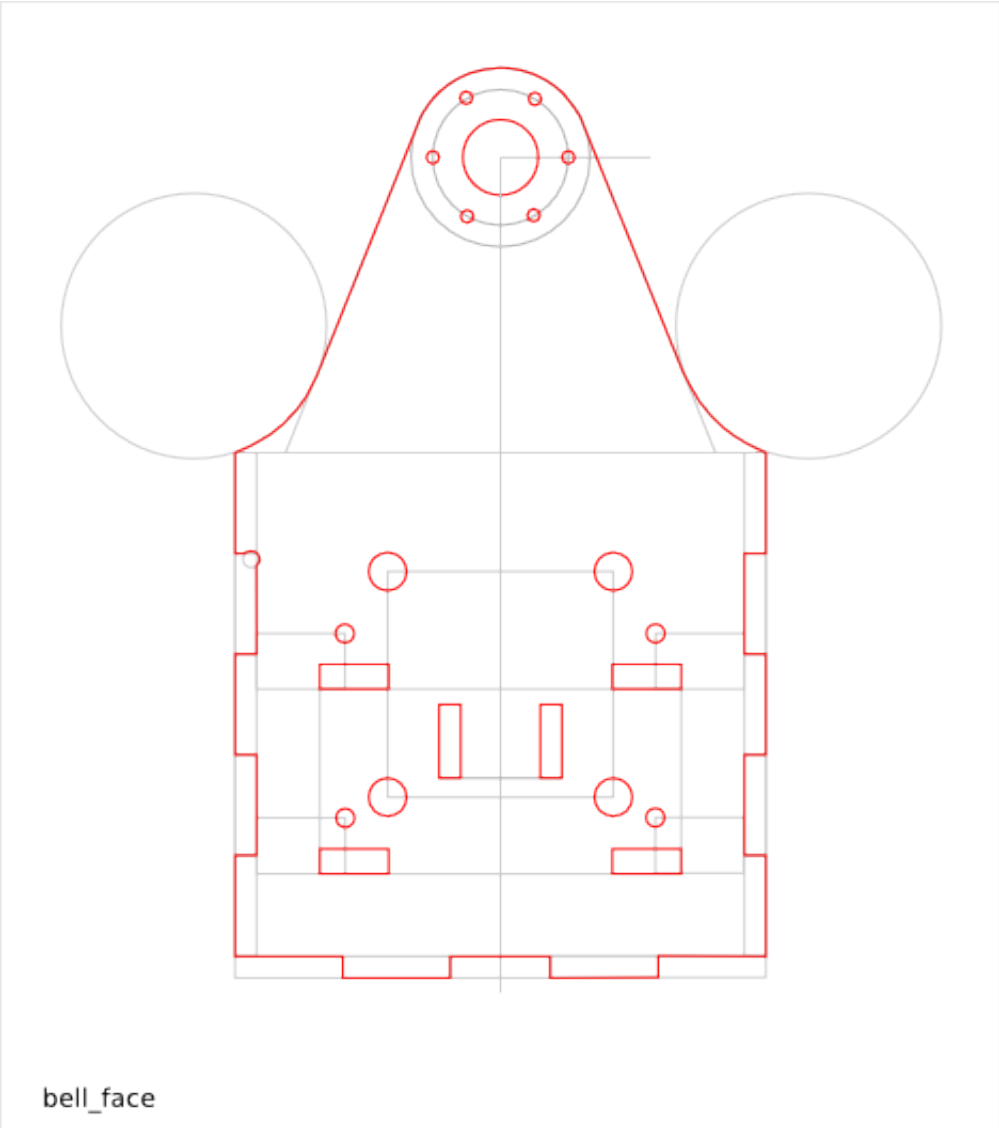
To get an overview of the possible *bell* designs that can be generated by *bell()*, run:

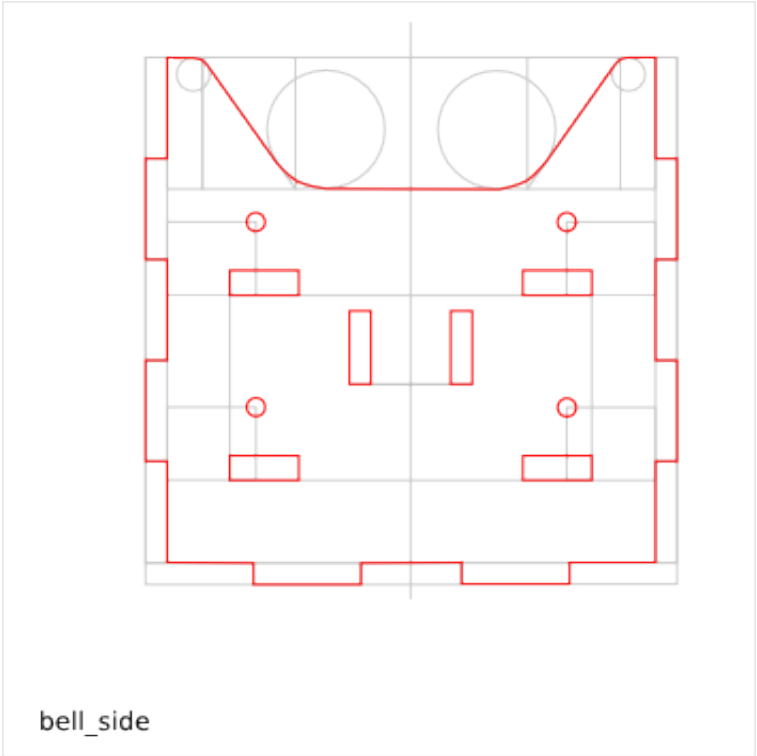
```
> python bell.py --run_self_test
```

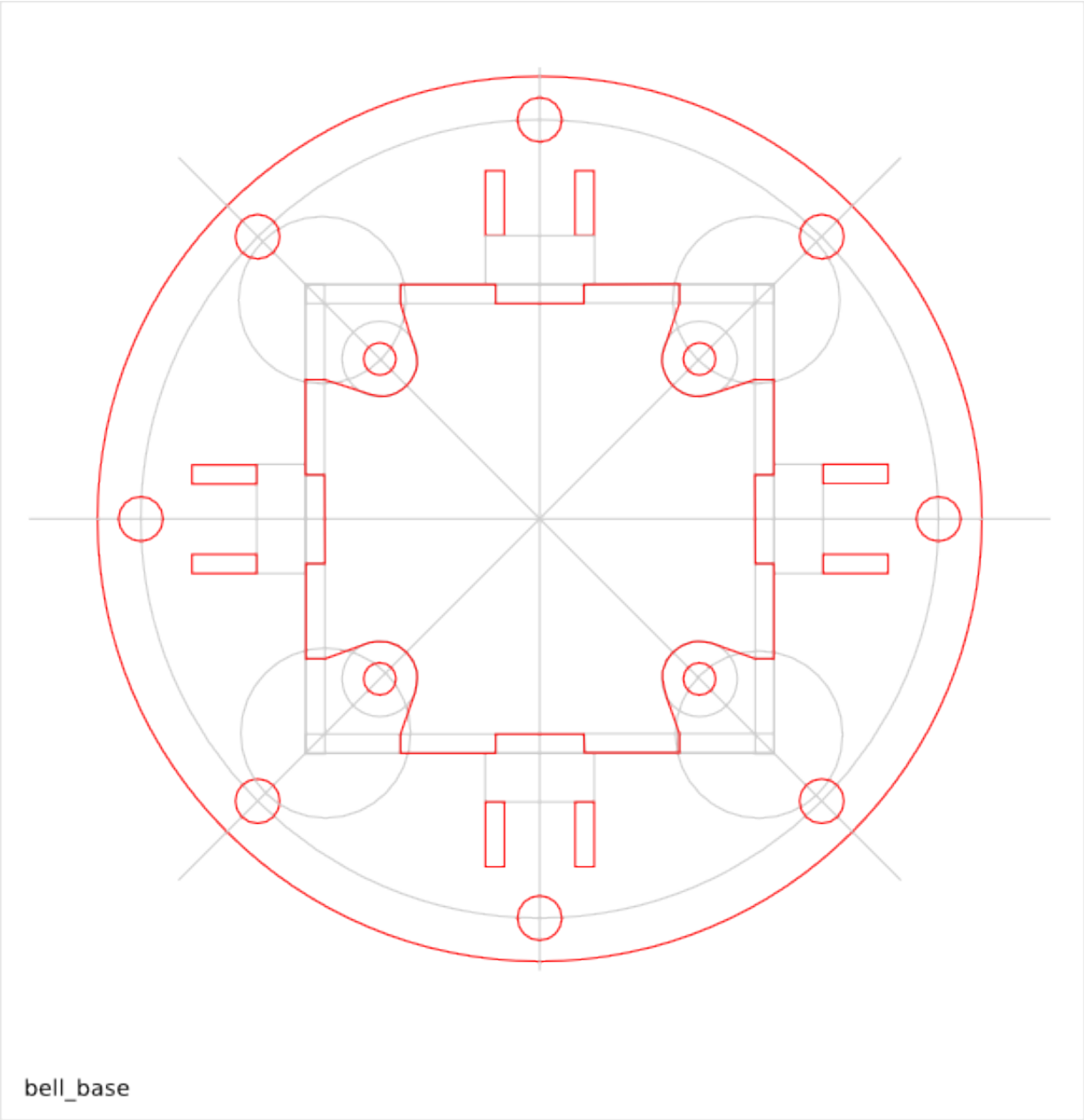
30.1 Bell Parts and Geometry

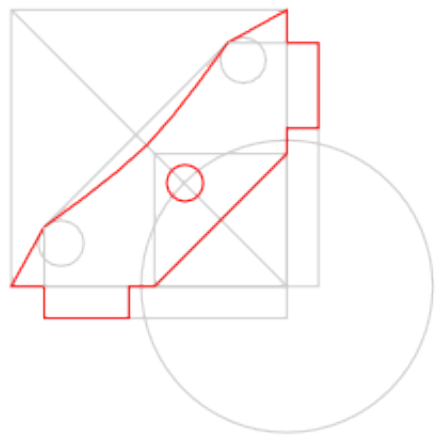
The *bell* is composed out of the following flat parts:

- bell_face x2
- bell_side x2
- bell_base x1
- bell_internal_buttress x8
- bell_external_buttress x8 (alternative: bell_external_buttress_face x4 and bell_external_buttress_side x4)

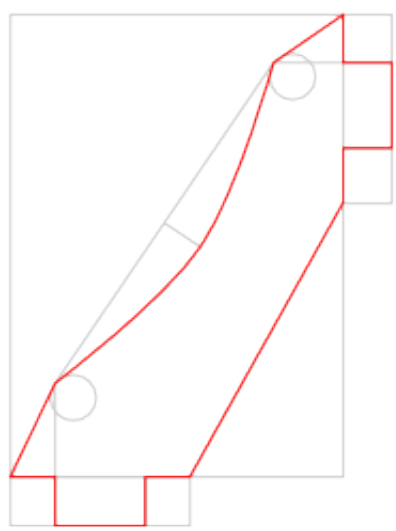






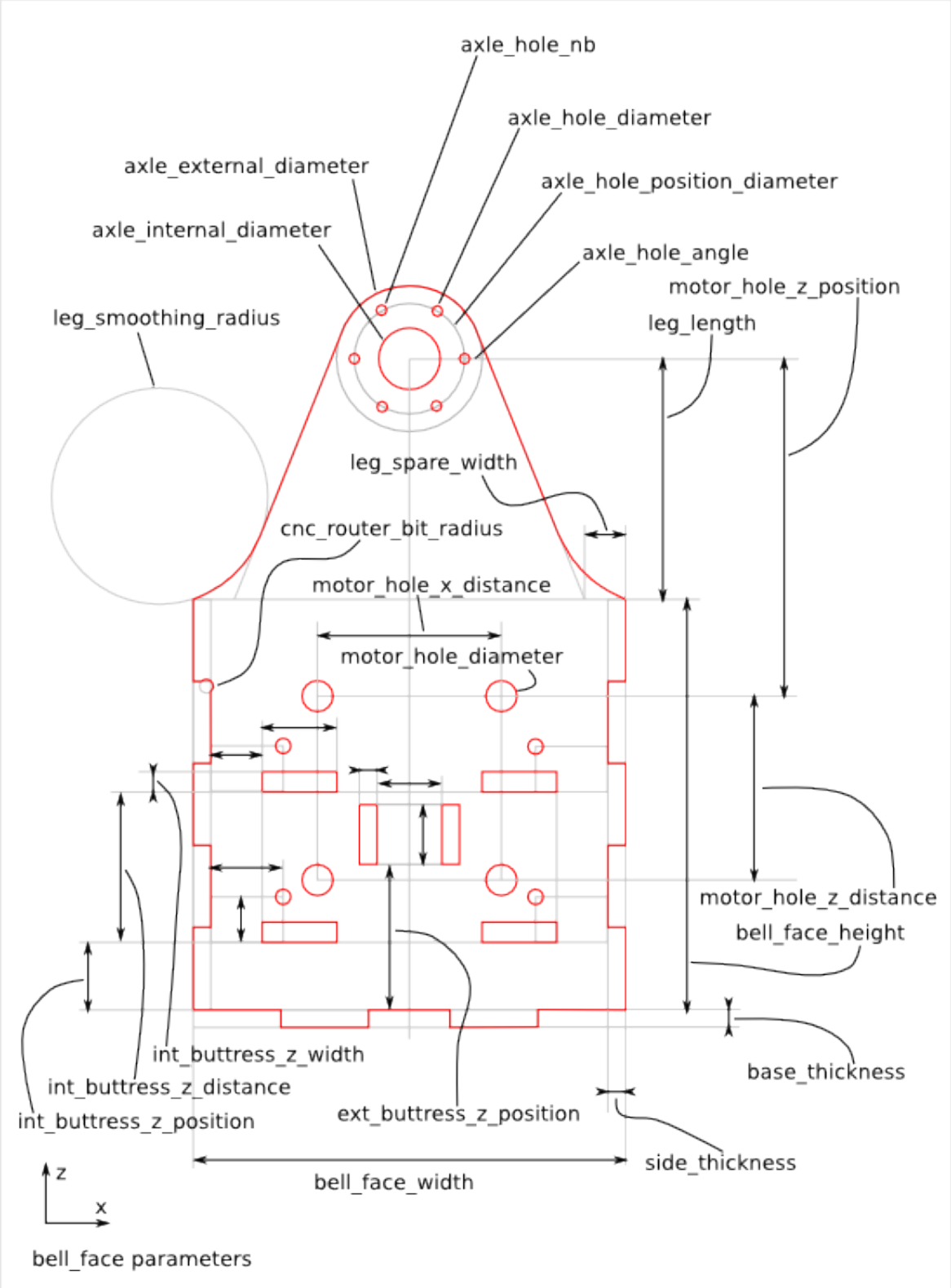


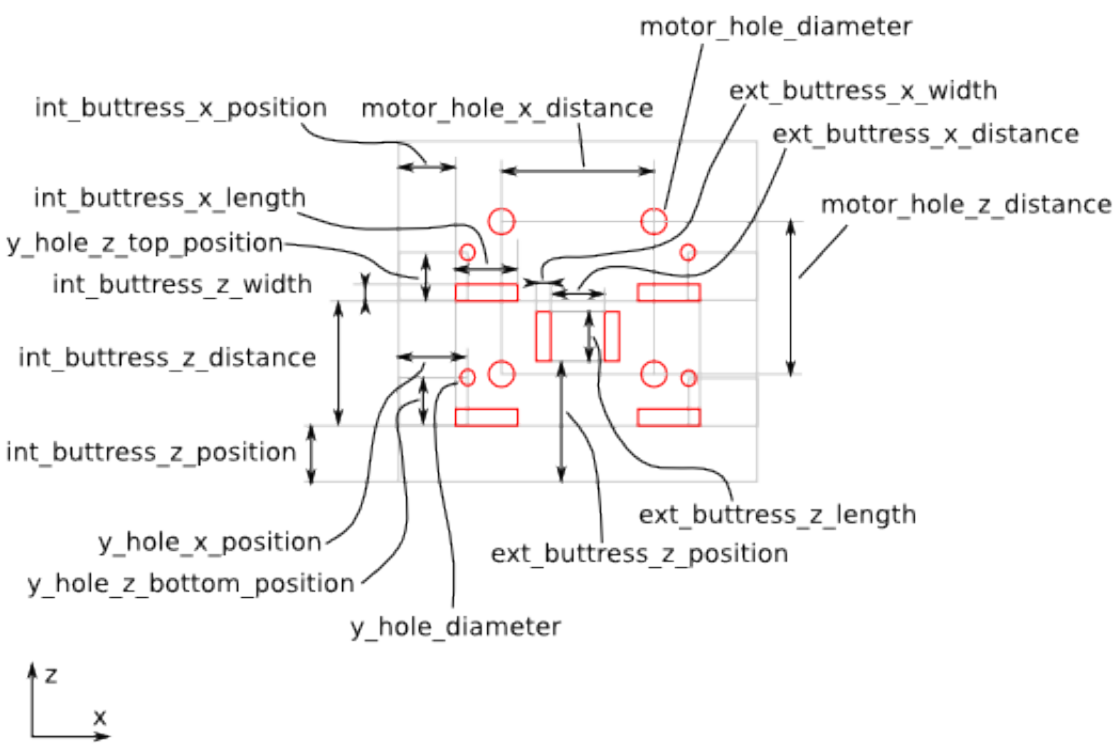
bell_internal_buttress



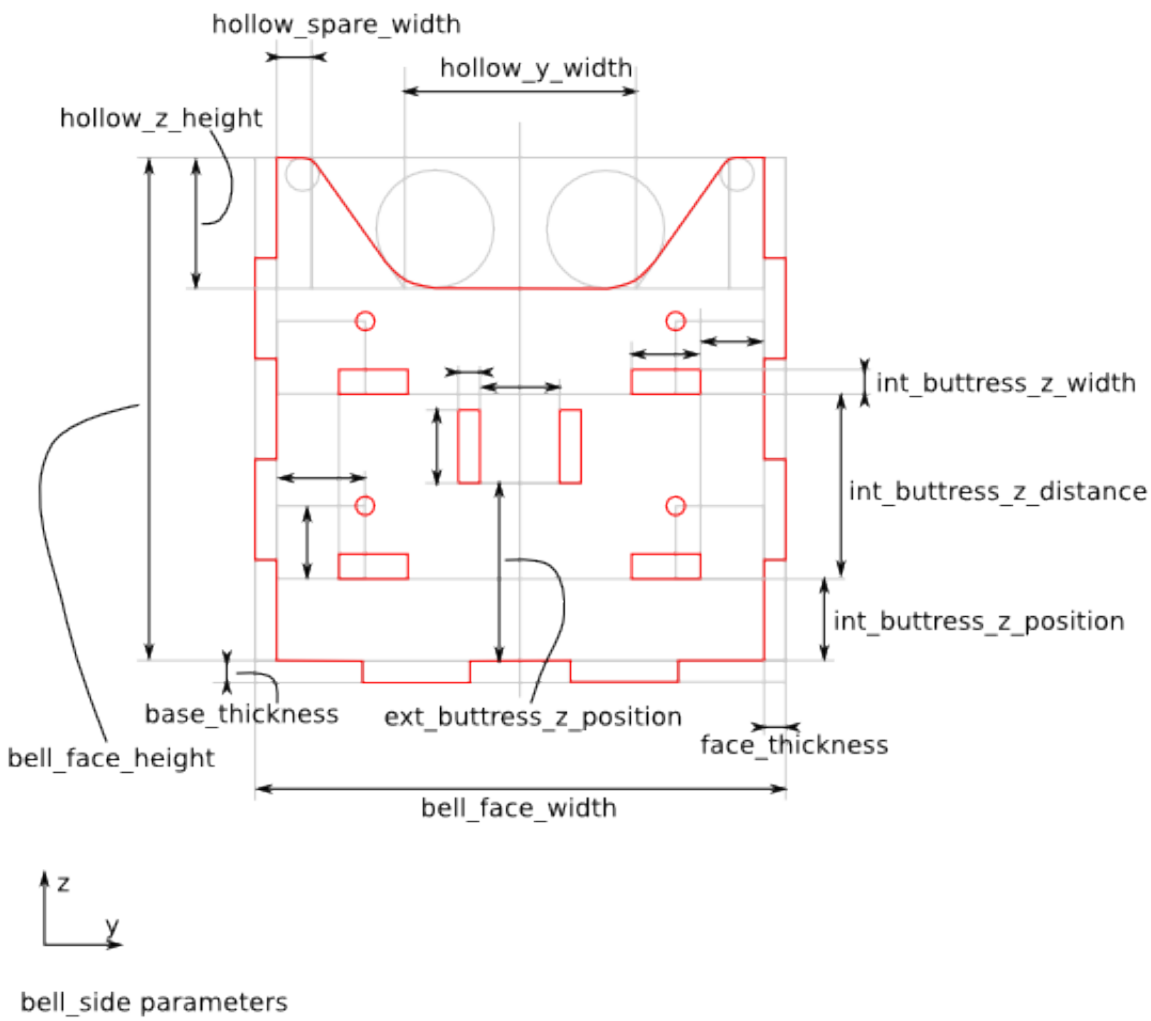
bell_external_buttress

30.2 Bell Parameter List

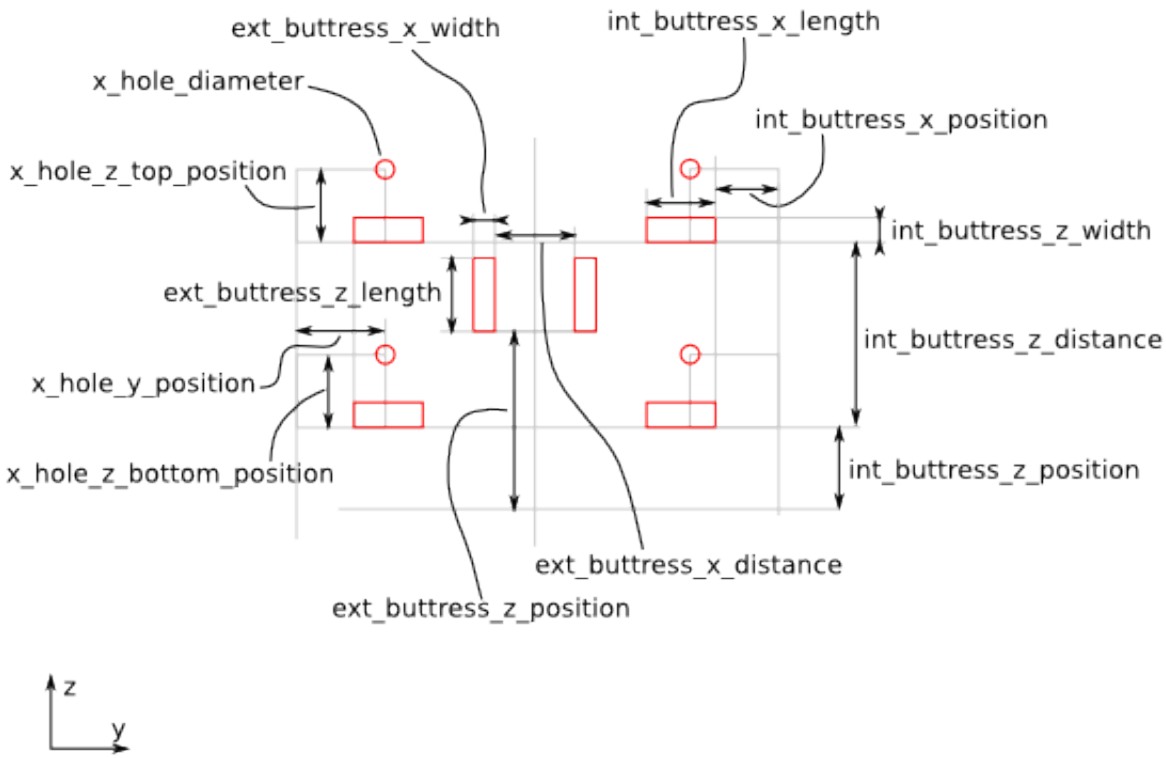




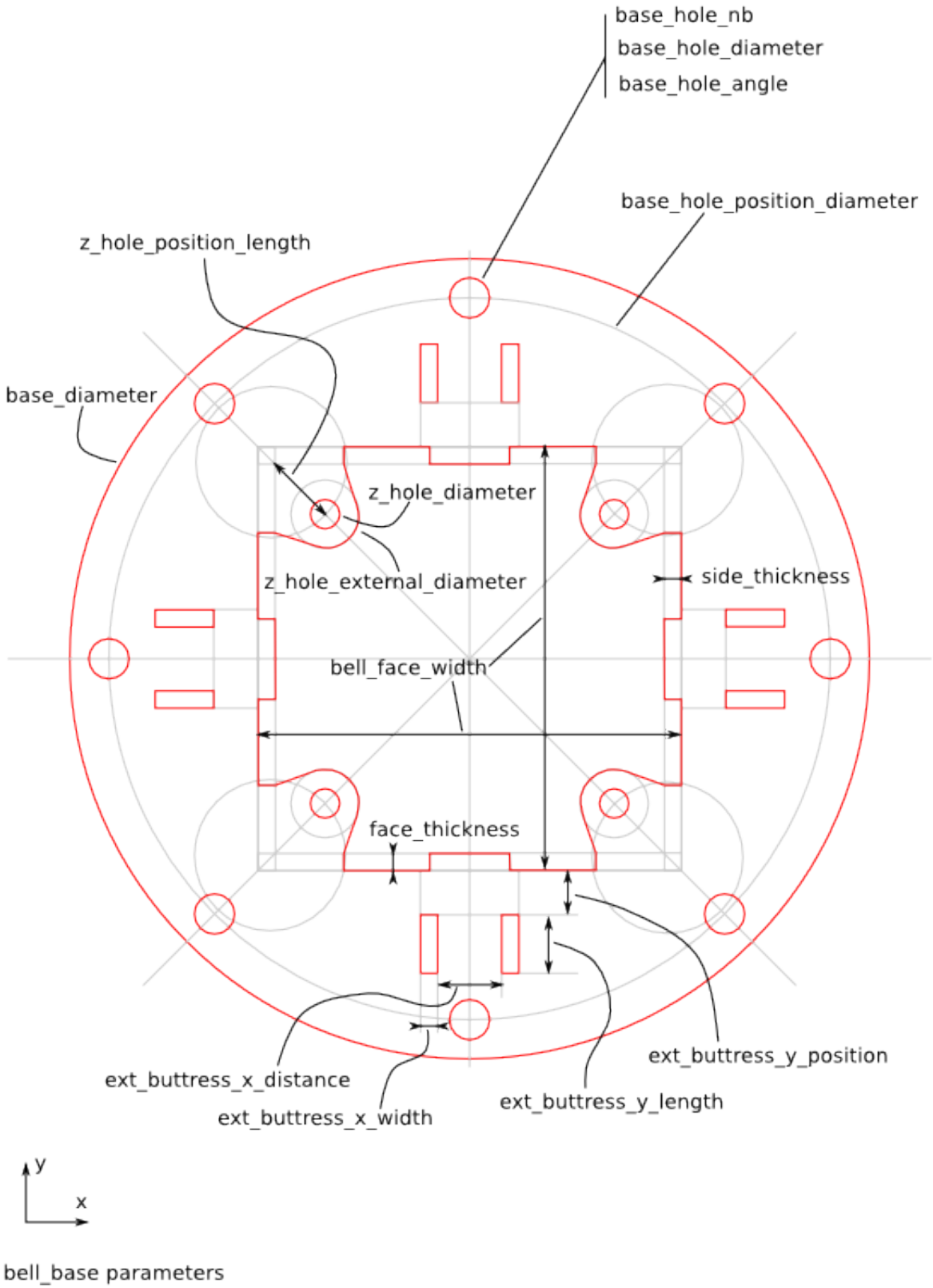
bell_face hole parameters



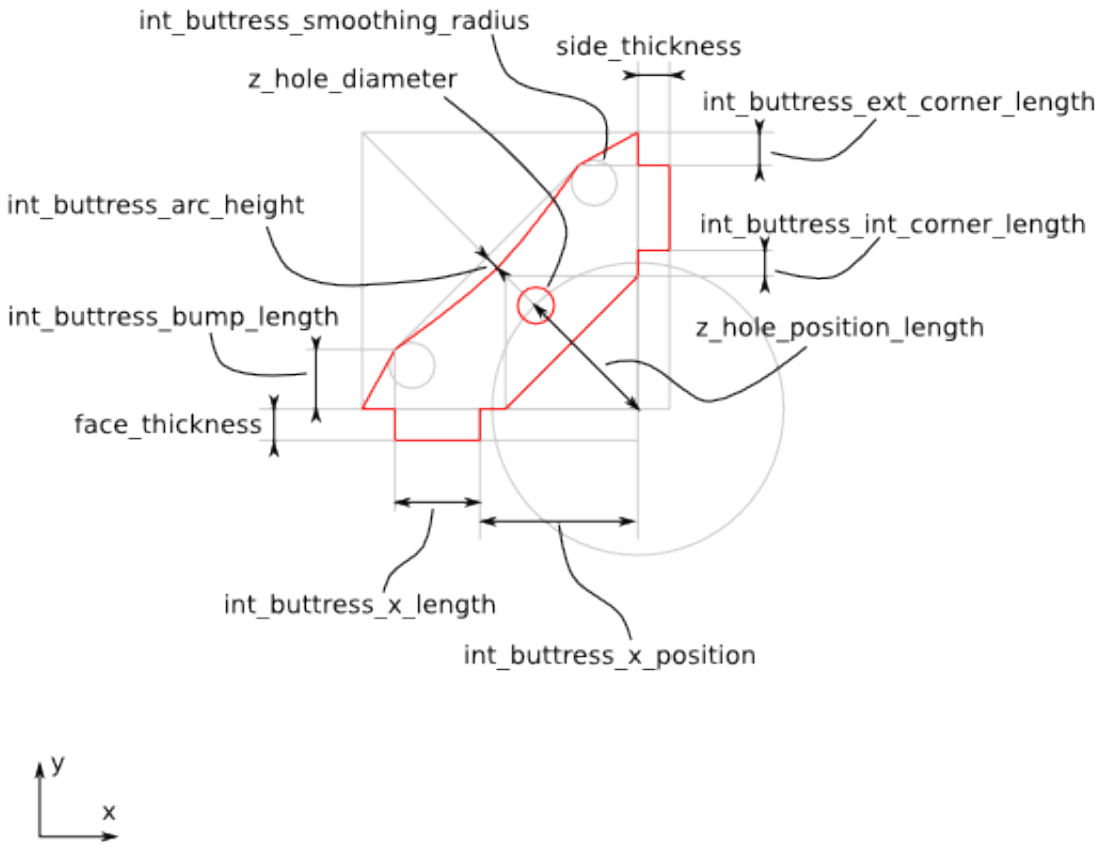
bell_side parameters



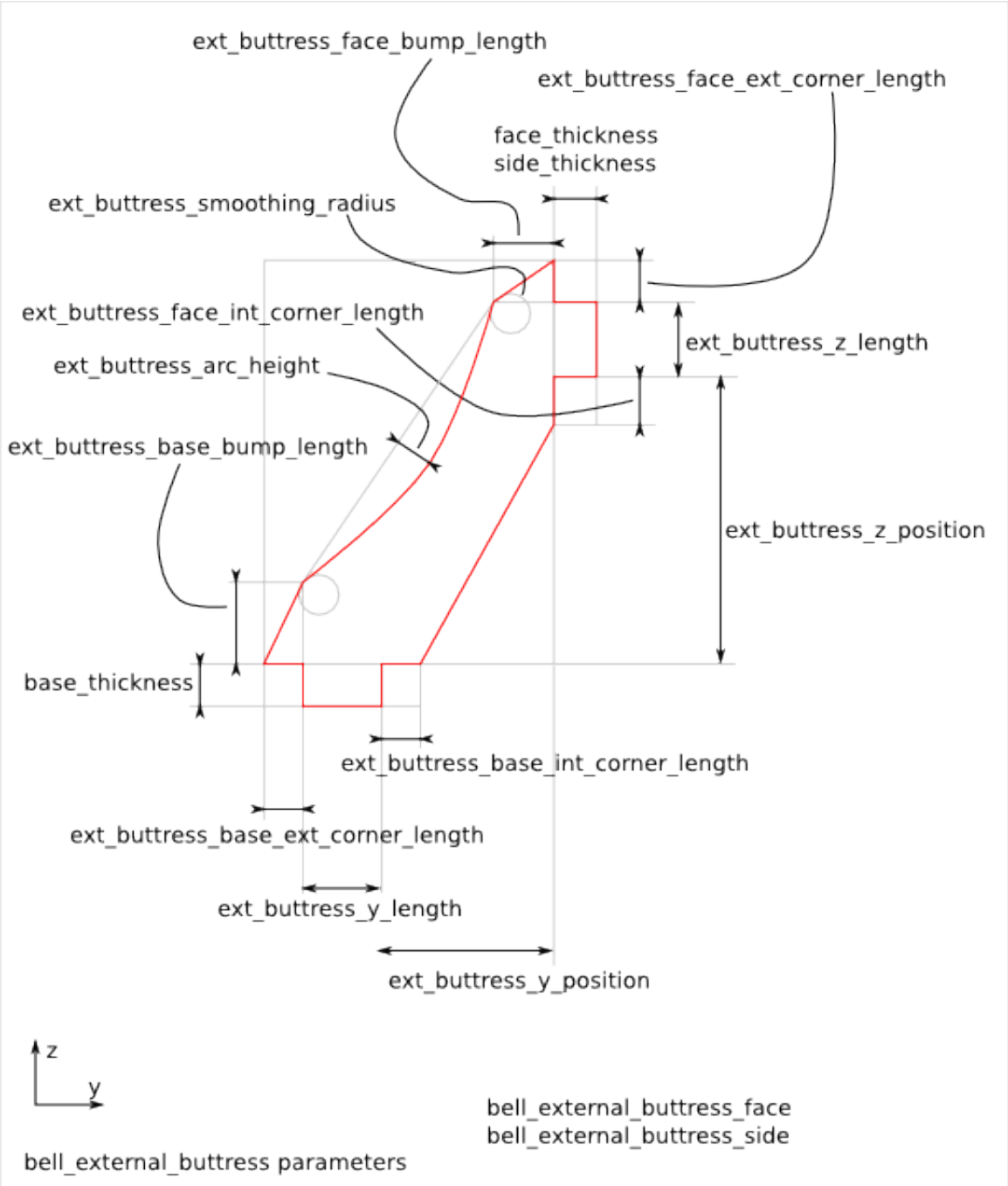
bell_side hole parameters



bell_base parameters



bell_internal_buttress parameters



30.3 Bell Parameter Dependency

30.3.1 router_bit_radius

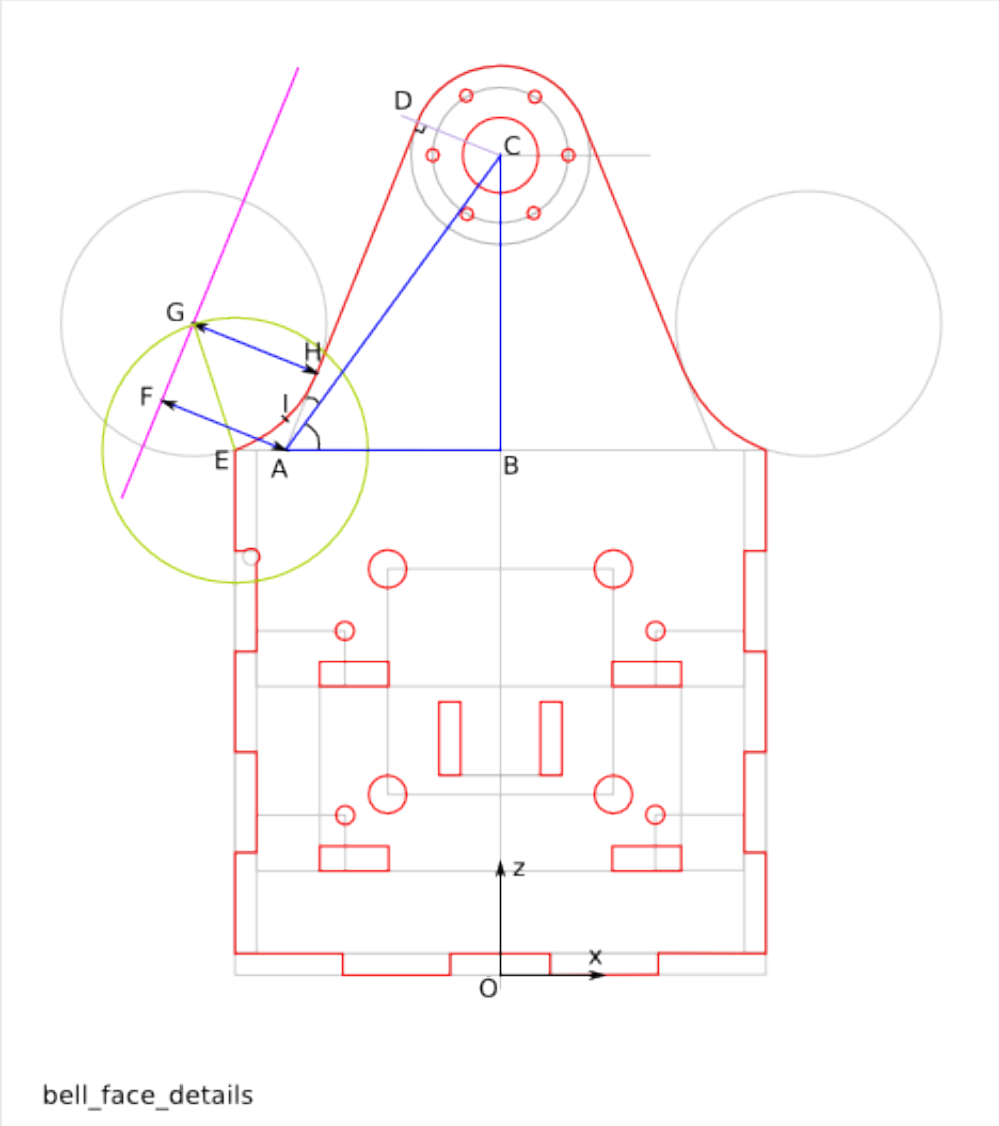
The two parameters *leg_smooth_radius* and *cnc_router_bit_radius* are related to the *router_bit_radius*. The parameter *cnc_router_bit_radius* guarantees the smallest possible *router_bit_radius* value. So, we have the relations:

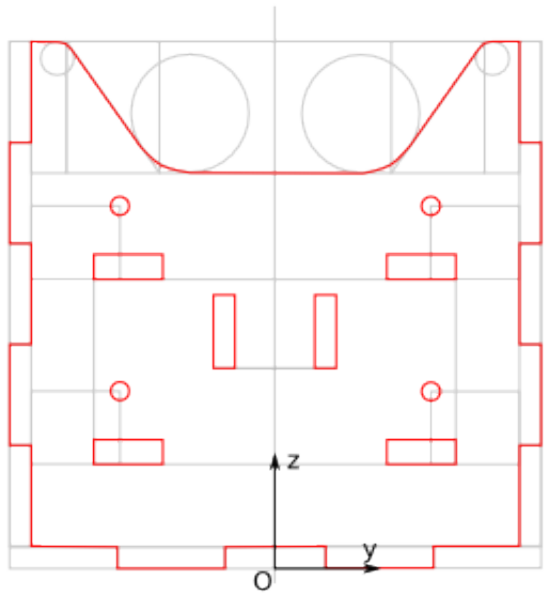
```
cnc_router_bit_radius < leg_smooth_radius
```

CHAPTER 31

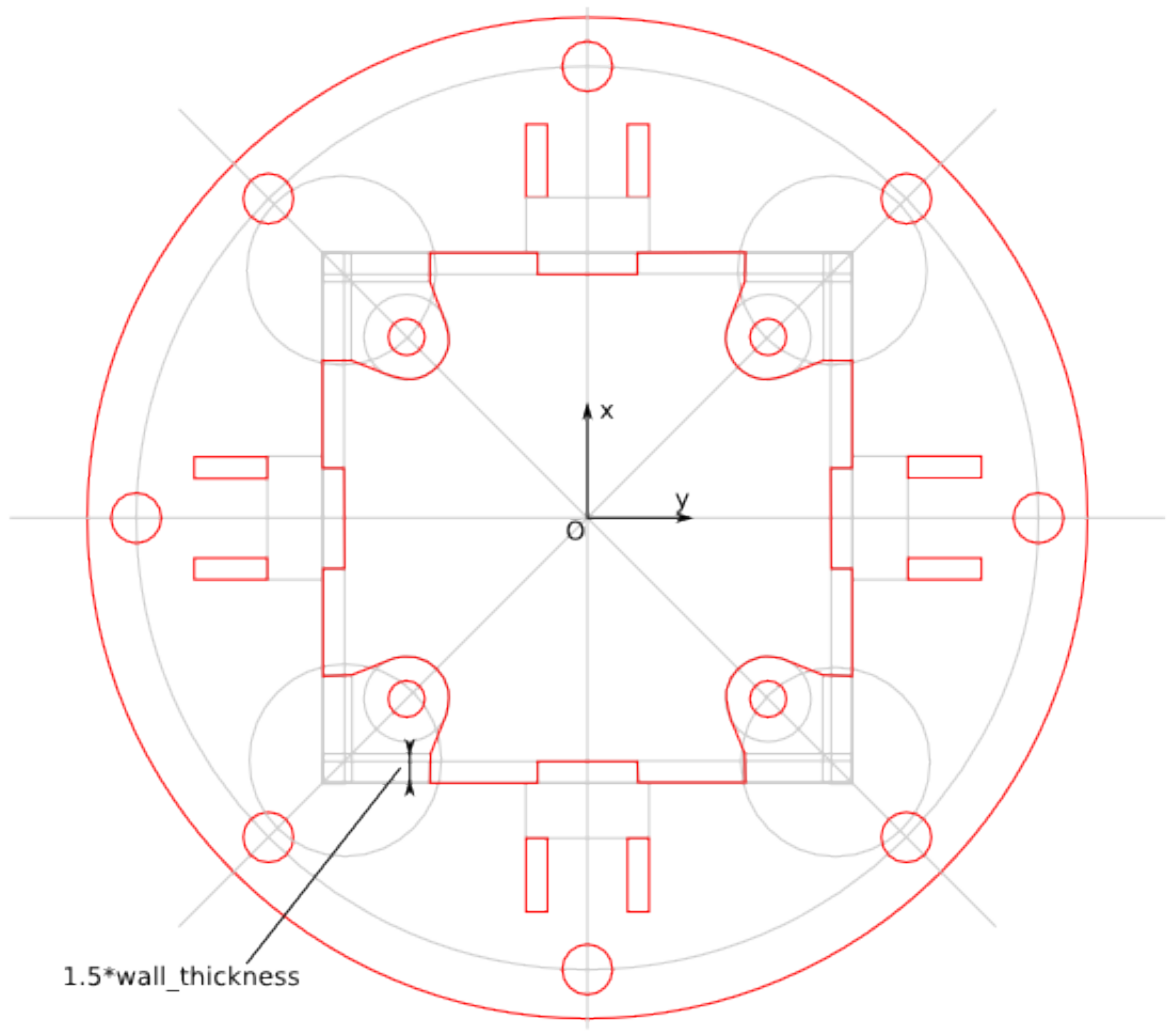
Bell Details

Construction details of the *bell* design.



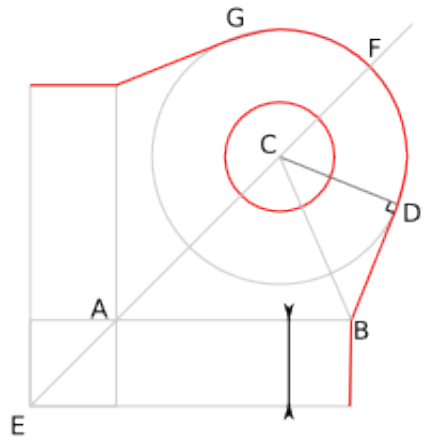


bell_side_details



wall_thickness = max(face_thickness, side_thickness)

bell_base_details

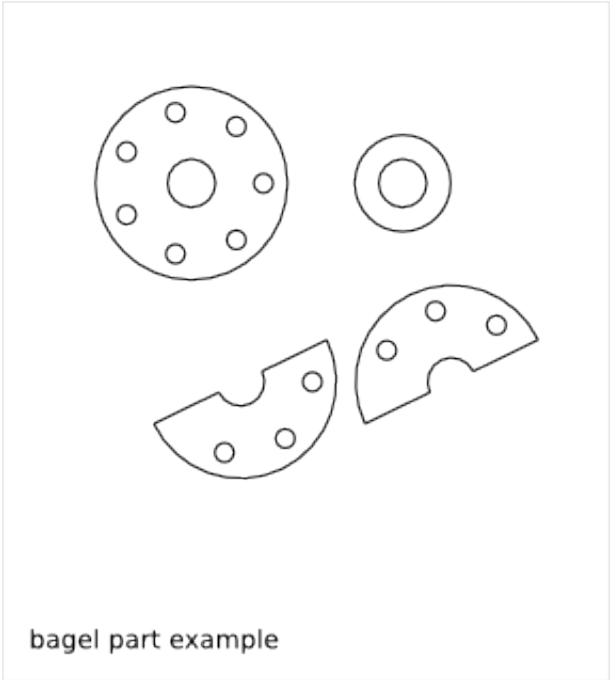


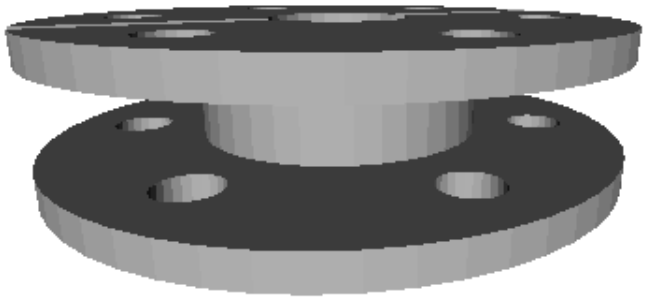
bell_base_main_hole_details

CHAPTER 32

Bagel Design

Ready-to-use parametric *bagel* design. It is the axle-guidance for the *bell* piece. The *bagel* is fixed to the *bell* but is mounted after the *axle* has been set in positon.





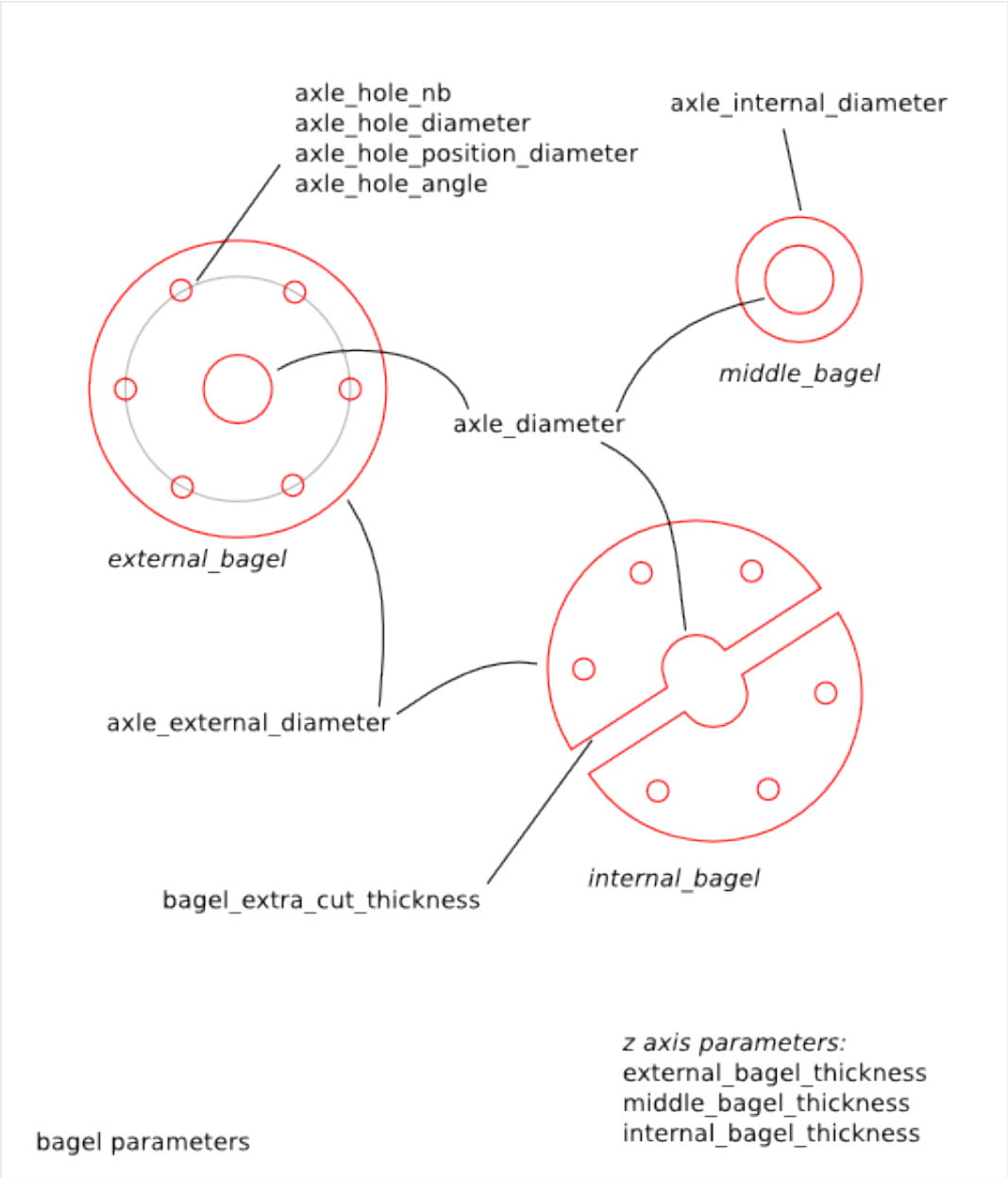
To get an overview of the possible *bagel* designs that can be generated by *bagel()*, run:

```
> python bagel.py --run_self_test
```

32.1 Bagel Parts and Parameters

The *bagel* is composed out of the following flat parts:

- external_bagel
- middle_bagel
- internal_bagel



32.2 Bagel Parameter Dependency

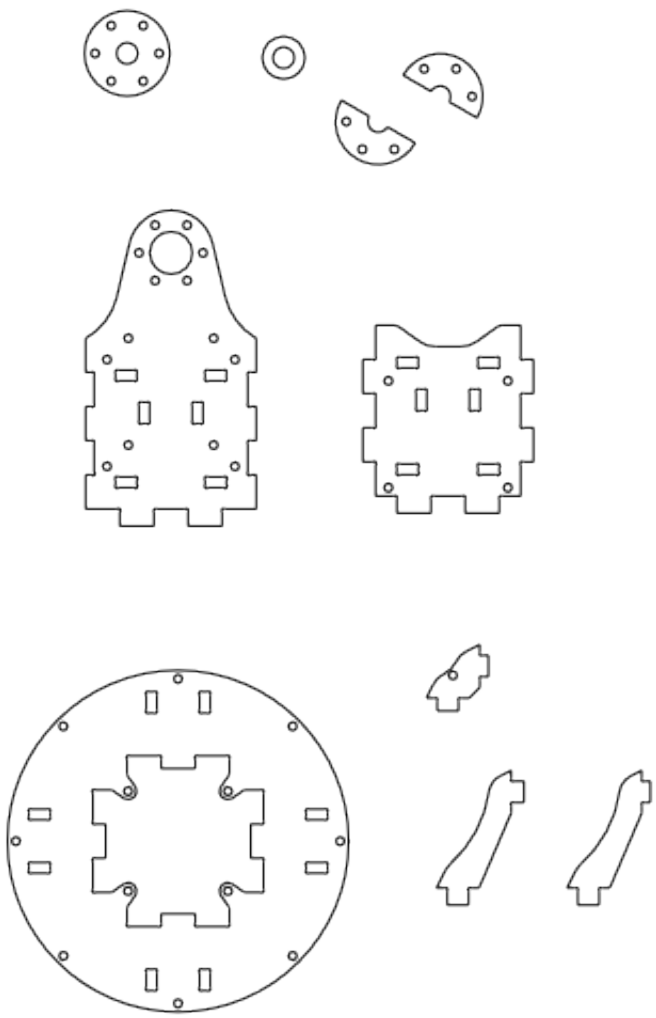
32.2.1 axle_internal_diameter

The *bell* design and the *bagel* design have both the *axle_internal_diameter* parameter. With ideal conditions, these two parameters get the same value. But you might want to but slightly different values to adjust the fit of the *middle_bagel* into the *bell axle internal hole*.

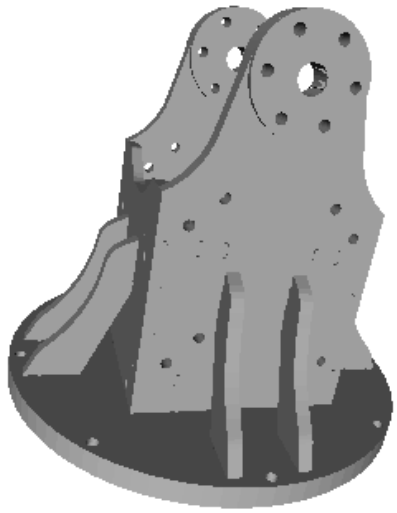
CHAPTER 33

Bell Bagel Assembly

Ready-to-use parametric *bell bagel assembly*. It generates the *bell* and the *bagel* parts.



bell_bagel_assembly part example



To get an overview of the possible *bell_bagel_assembly* designs that can be generated by *bell_bagel_assembly()*, run:

```
> python bell_bagel_assembly.py --run_self_test
```

33.1 Bell-Bagel-Assembly Parameters

The *bell_bagel_assembly* parameters are directly inherited from the Bell Design parameters and the Bagel Design parameters.

33.2 Bell-Bagel-Assembly Parameter Dependency

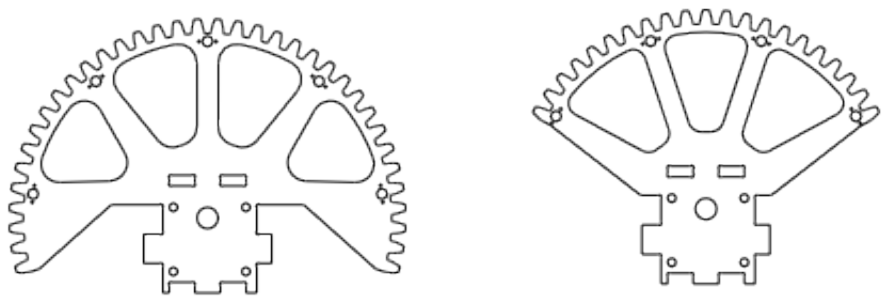
33.2.1 axle_internal_diameter

The *bell* design and the *bagel* design have both the *axle_internal_diameter* parameter, respectly called *axle_internal_diameter* and *bagel_axle_internal_diameter*. With ideal conditions, these two parameters get the same value. But you might want to but slightly different values to adjust the fit of the *middle_bagel* into the *bell axle internal hole*.

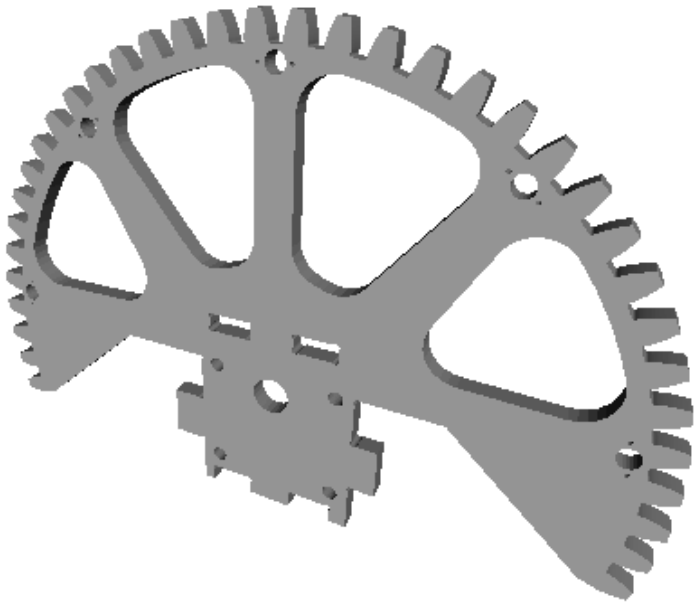
CHAPTER 34

Crest Design

Ready-to-use parametric *crest* design. It is an optional part of the *cross_cube* assembly to get a motorized *gimbal* system.



crest part examples

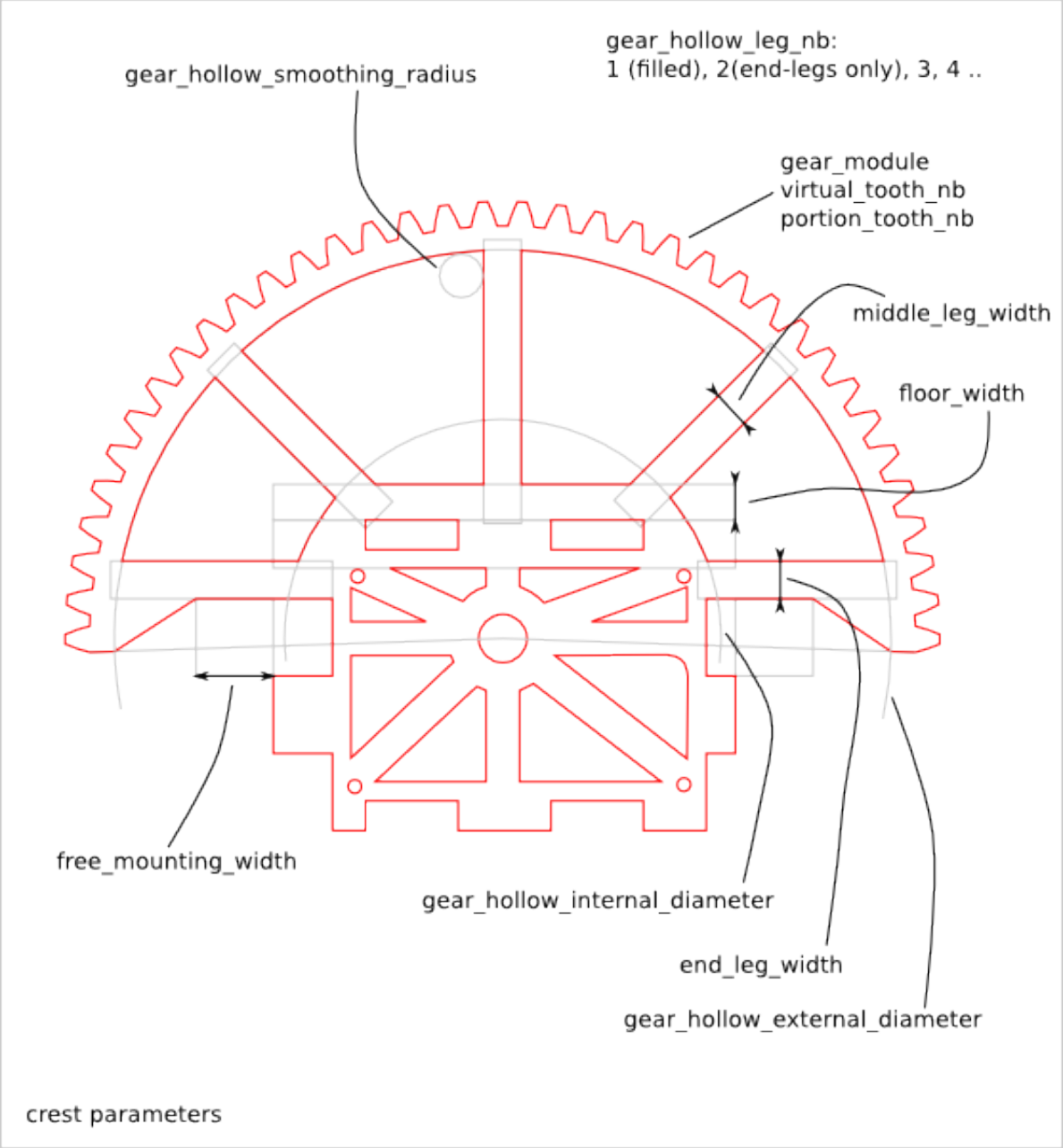


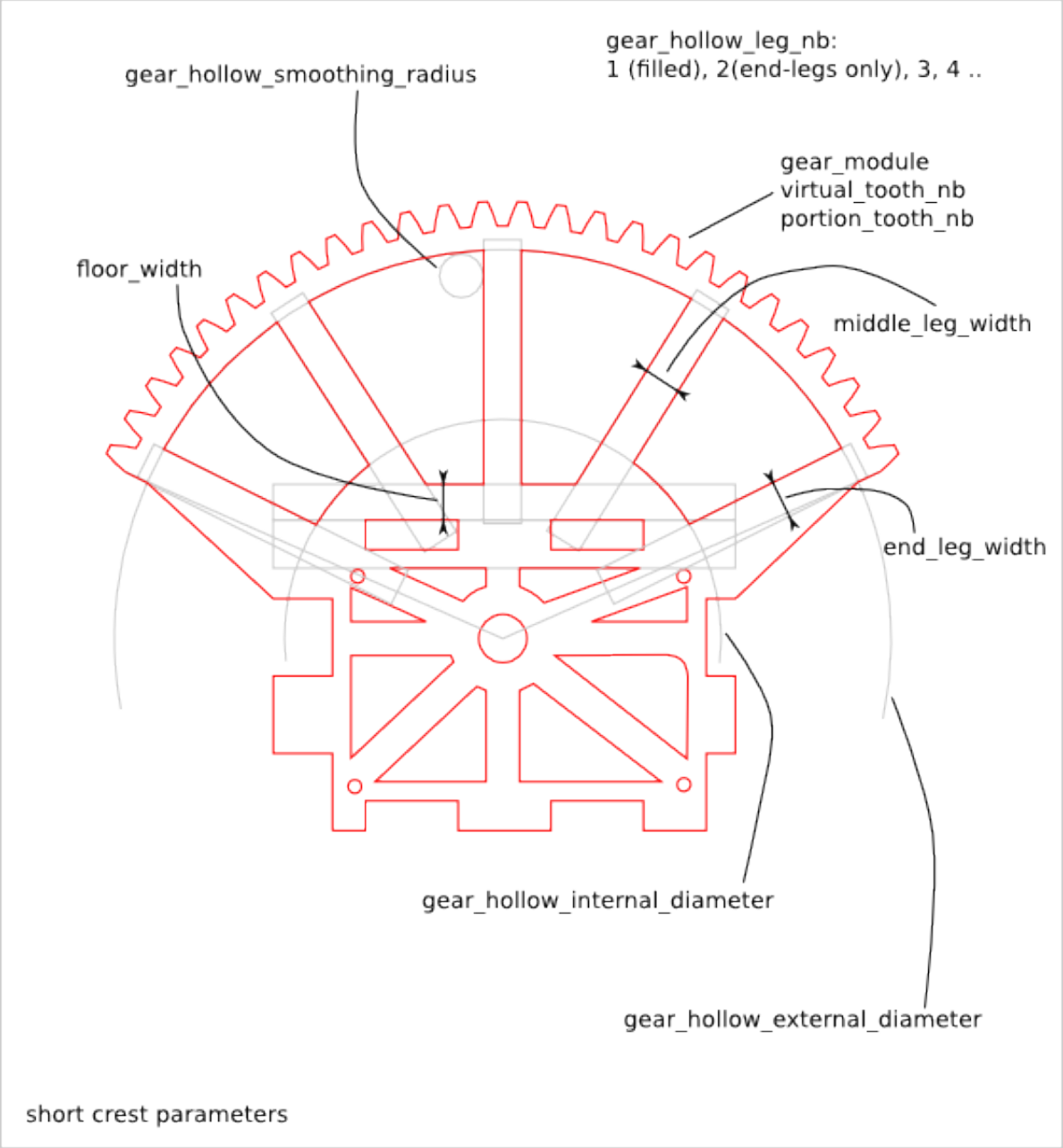
To get an overview of the possible *crest* designs that can be generated by *crest()*, run:

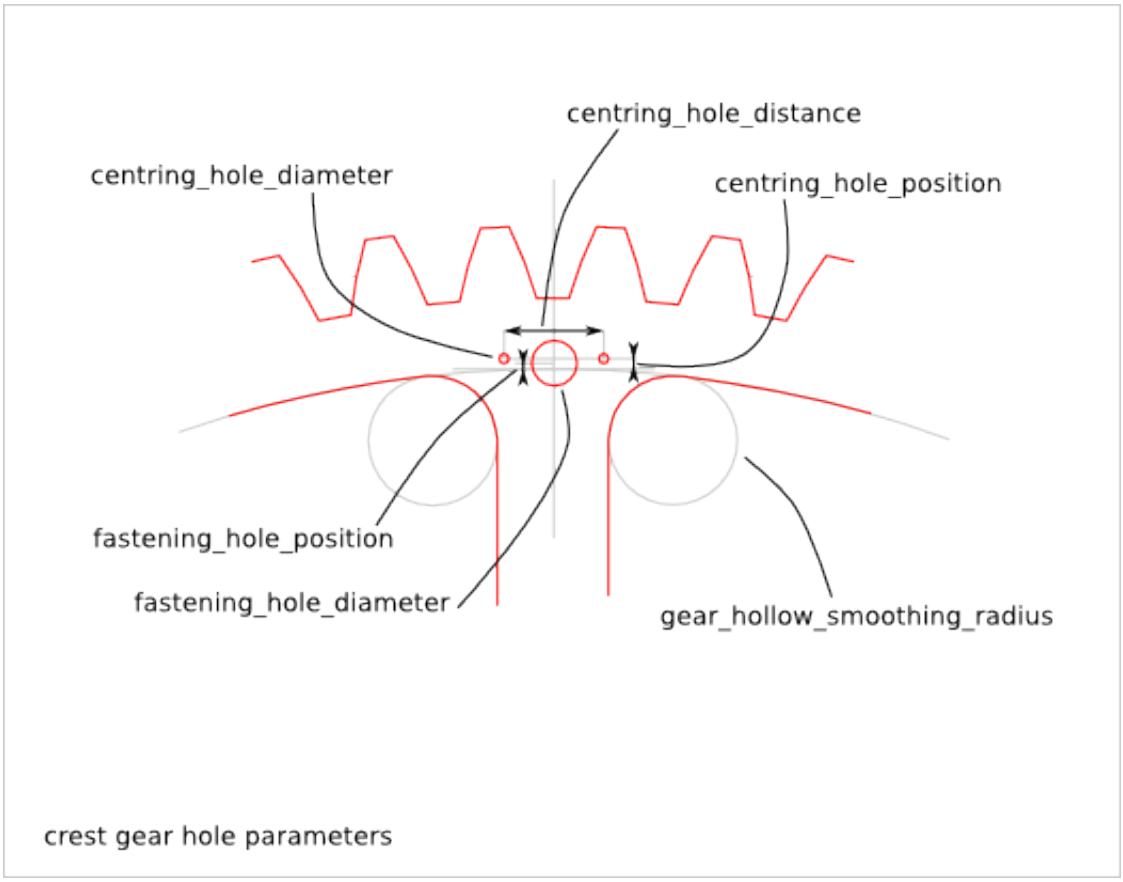
```
> python crest.py --run_self_test
```

34.1 Crest Parameters

The *crest* part inherit several parameters from Cross_Cube Design.







34.2 Crest Parameter Dependency

34.2.1 crest_cnc_router_bit_radius

The following parameter are related to the *router_bit radius*:

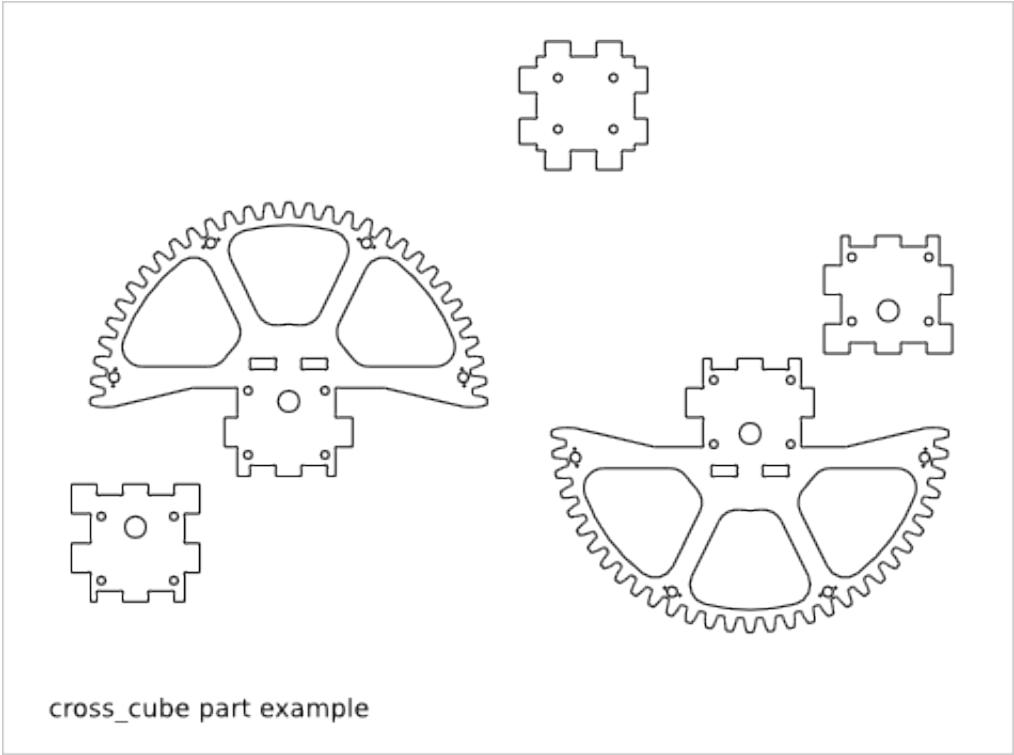
- crest_cnc_router_bit_radius
- gear_cnc_router_bit_radius
- gear_hollow_smoothing_radius
- cross_cube_cnc_router_bit_radius
- face_hollow_smoothing_radius
- top_hollow_smoothing_radius

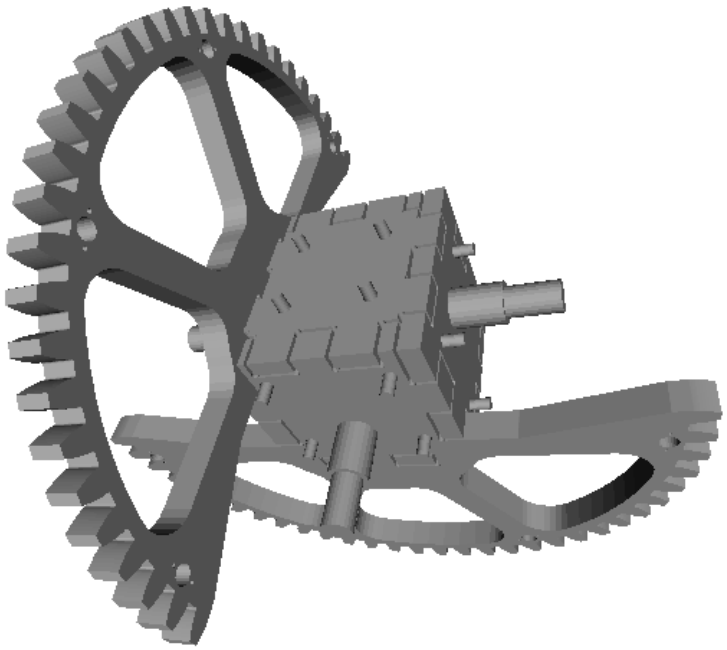
The *crest_cnc_router_bit_radius* parameter guarantees the smallest value for all related *router_bit radius* parameters.

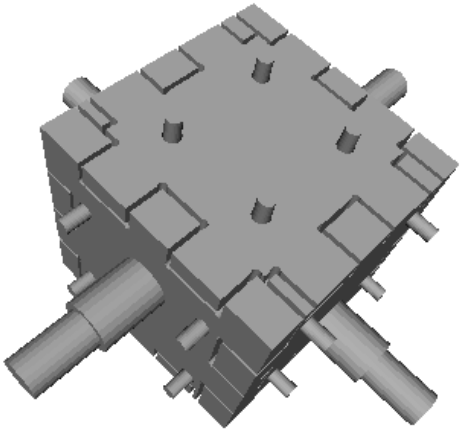
CHAPTER 35

Cross_Cube Design

Ready-to-use parametric *cross_cube* design. It is a cross axle holder for gimbal.







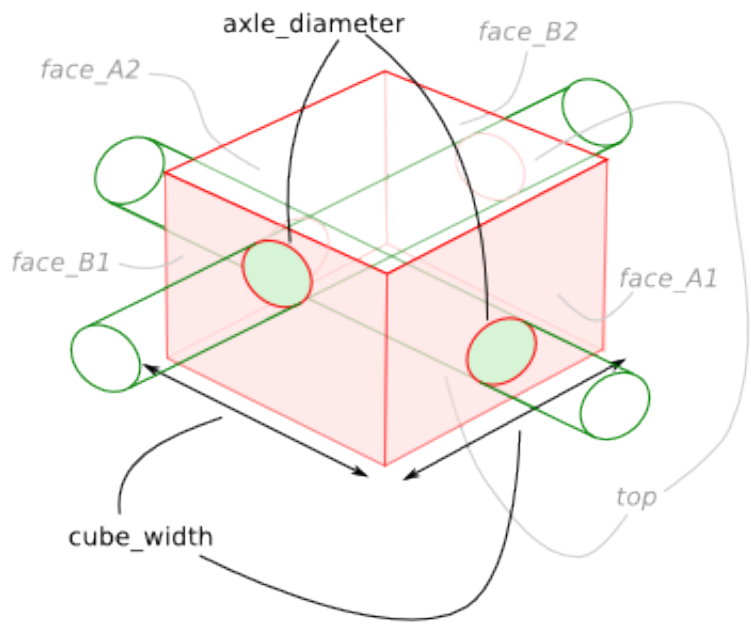
To get an overview of the possible *cross_cube* designs that can be generated by *cross_cube()*, run:

```
> python cross_cube.py --run_self_test
```

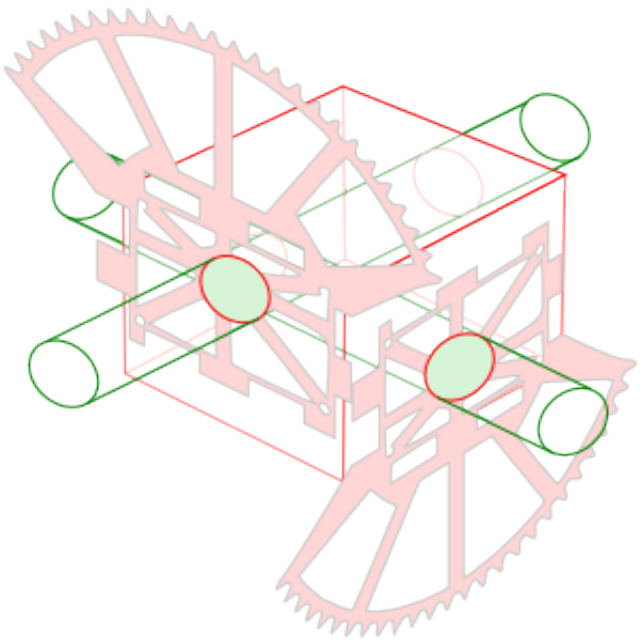
35.1 Cross_Cube Parts and Parameters

The *cross_cube* piece is composed out of the following flat parts:

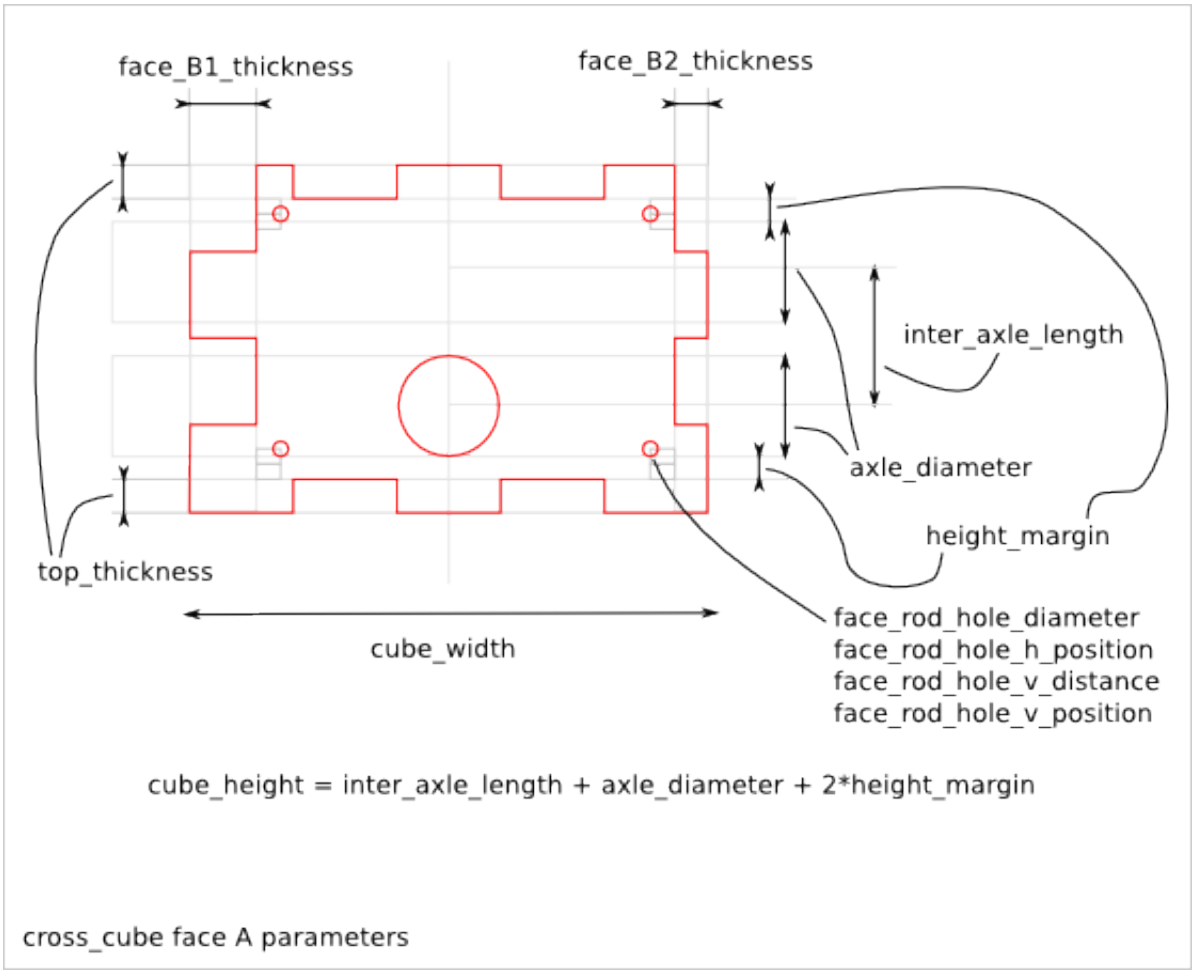
- face_A1
- face_A2
- face_B1
- face_B2
- top

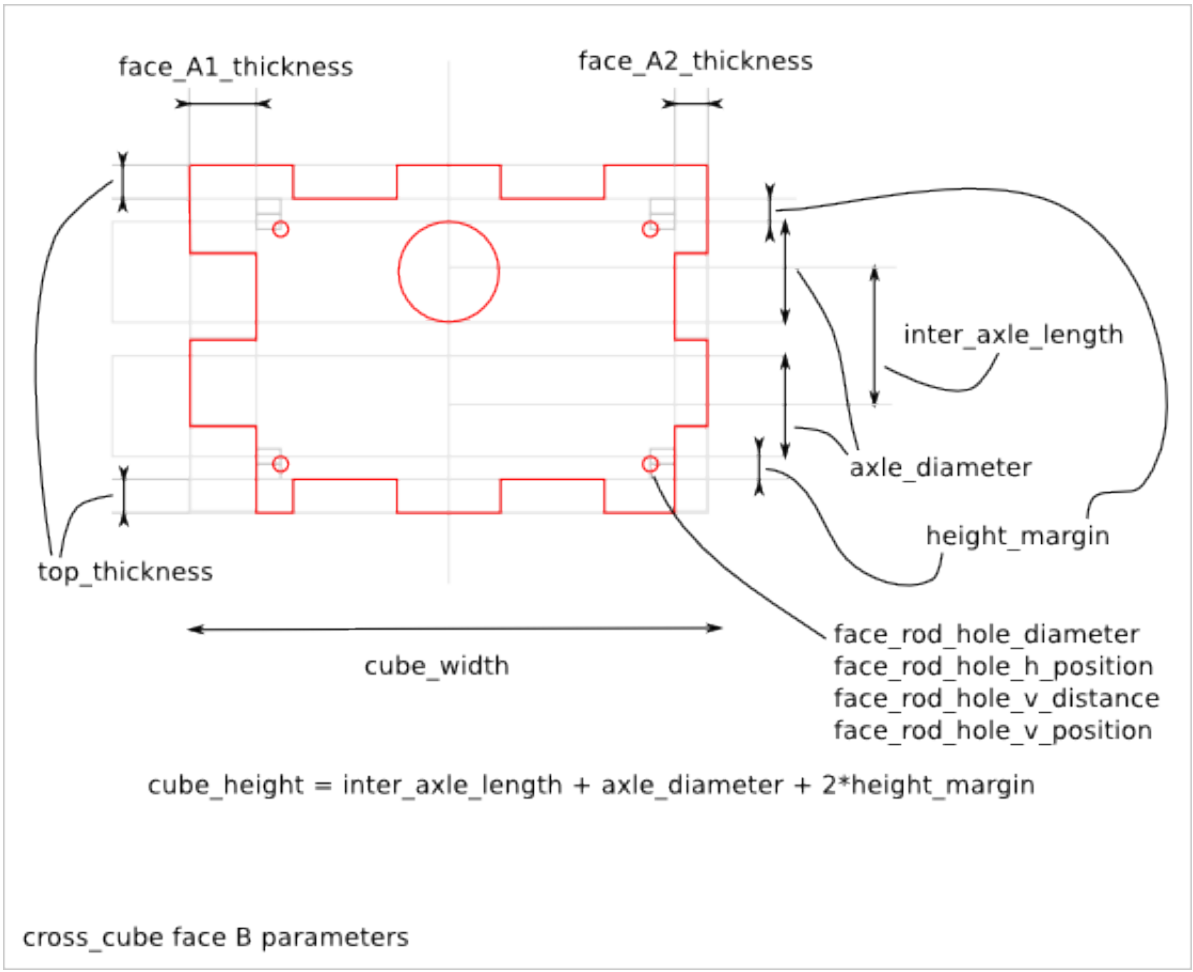


cross cube main parameters

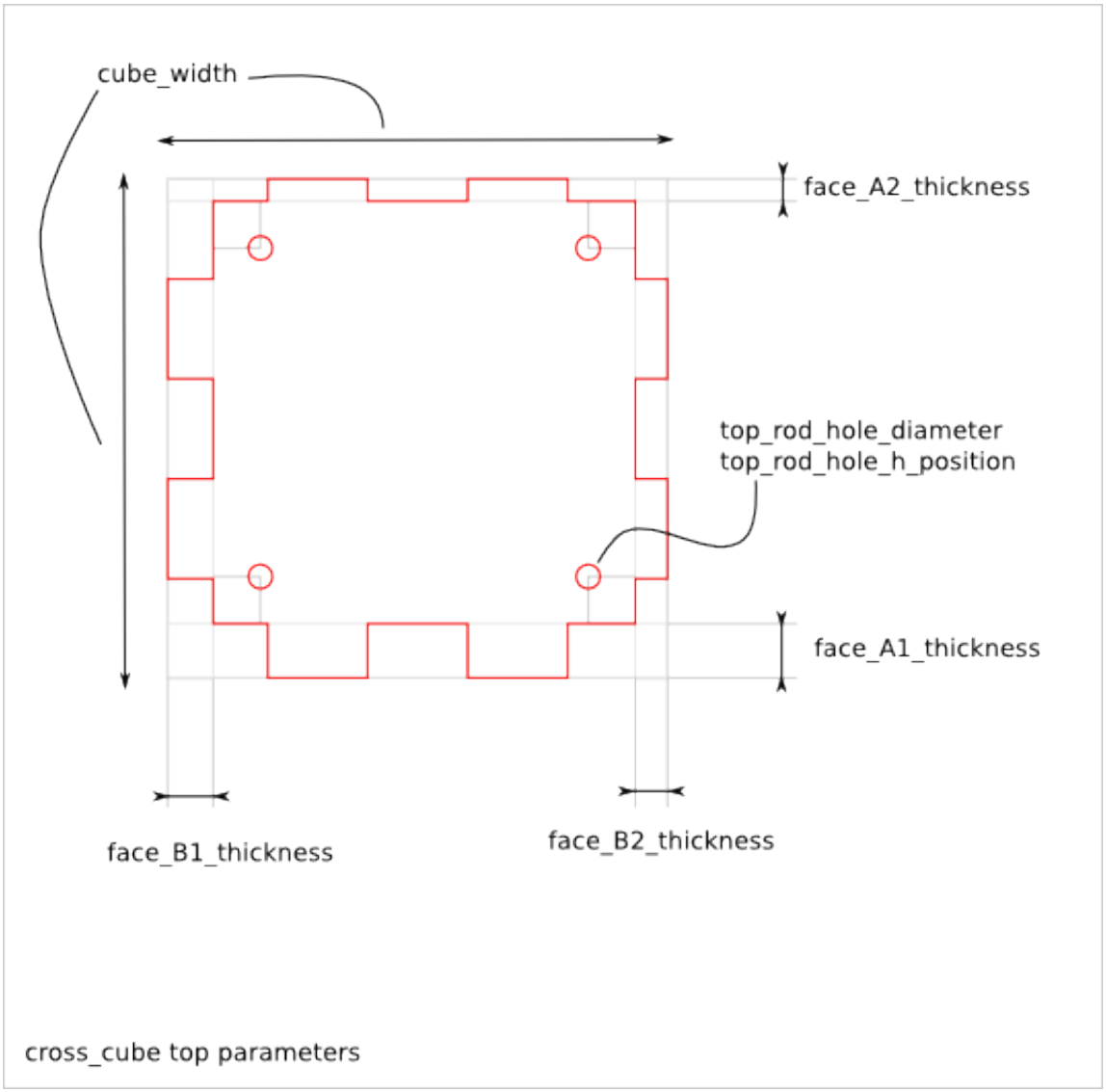


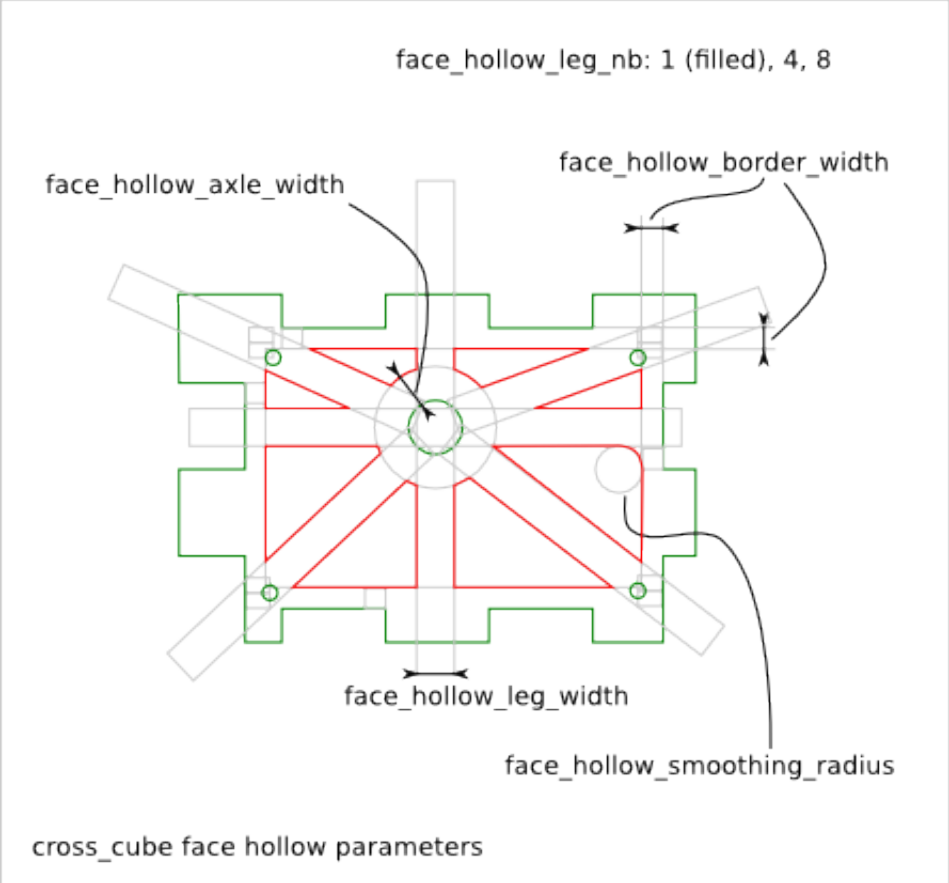
crest cross cube assembly

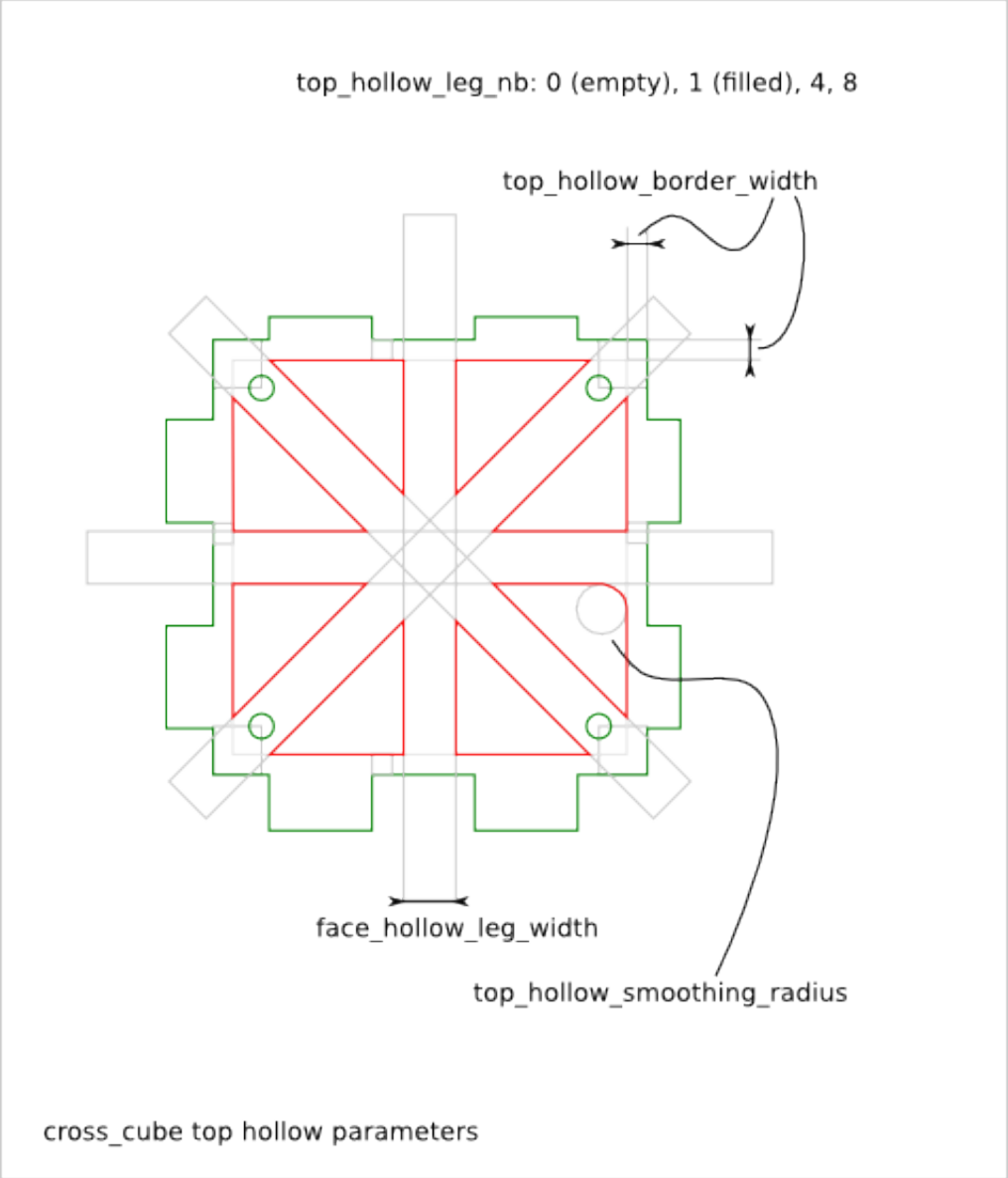


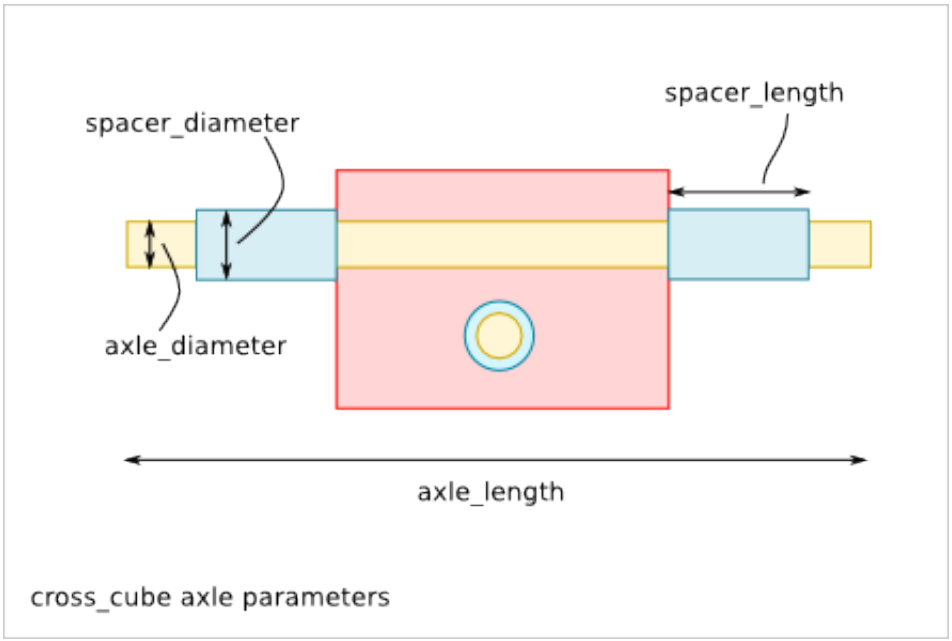


cross_cube face B parameters









35.2 Cross_Cube Parameter Dependency

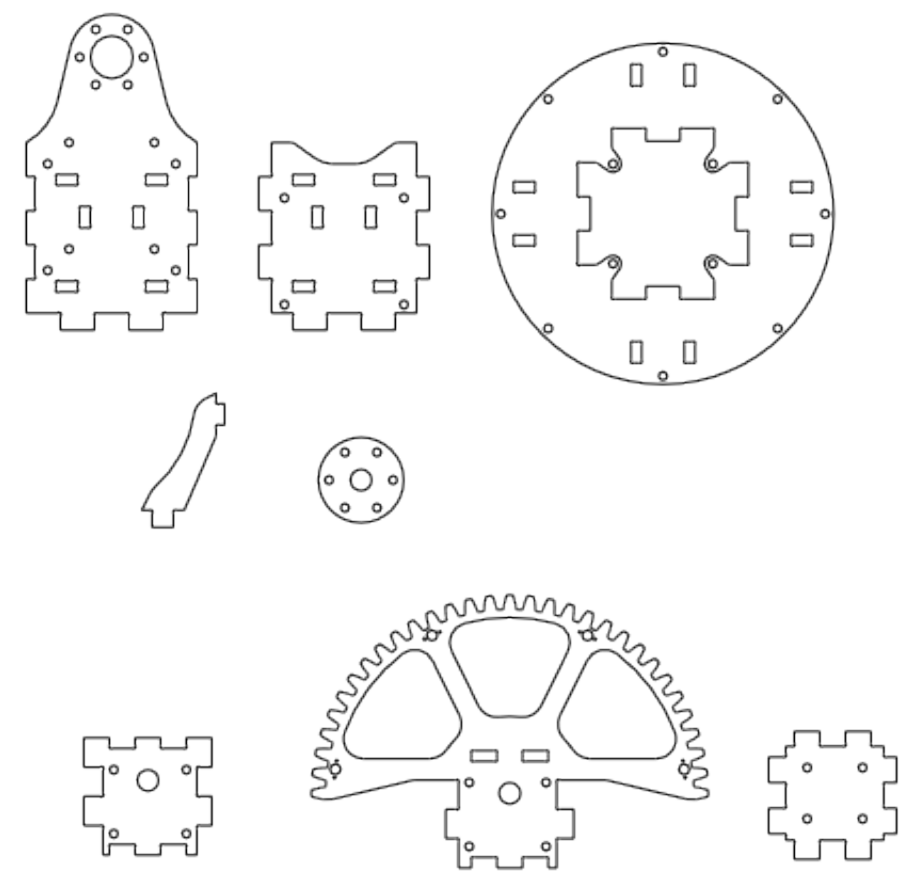
35.2.1 cross_cube_extra_cut_thickness

The *cross_cube_extra_cut_thickness* parameter can be used to compensate the manufacturing process or to check the 3D assembly with FreeCAD. The default value is 0.0.

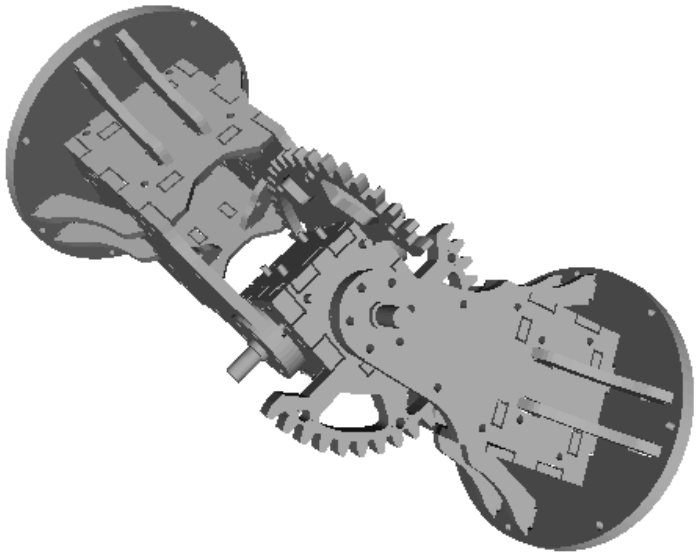
CHAPTER 36

Gimbal Design

Ready-to-use parametric *gimbal* design. It is a mechanism with two degrees of freedom, that let's adjusting the roll-pitch orientation.



gimbal part example

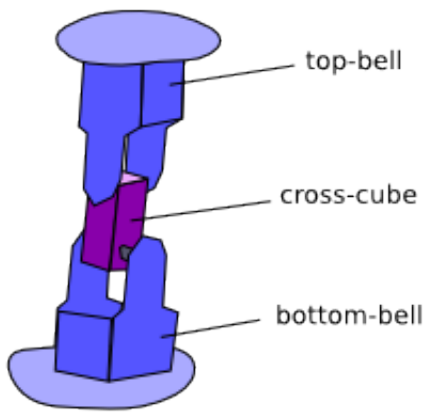


To get an overview of the possible *gimbal* designs that can be generated by *gimbal()*, run:

```
> python gimbal.py --run_self_test
```

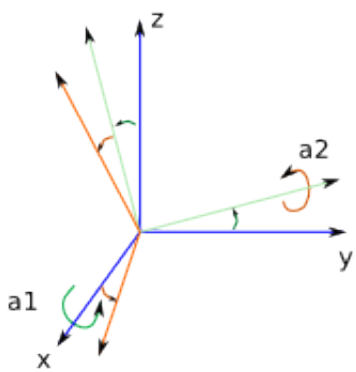
36.1 Gimbal Parameters

The *gimbal* mechanism is composed by two *bell_bagel_assembly* and one *cross_cube* with crests.

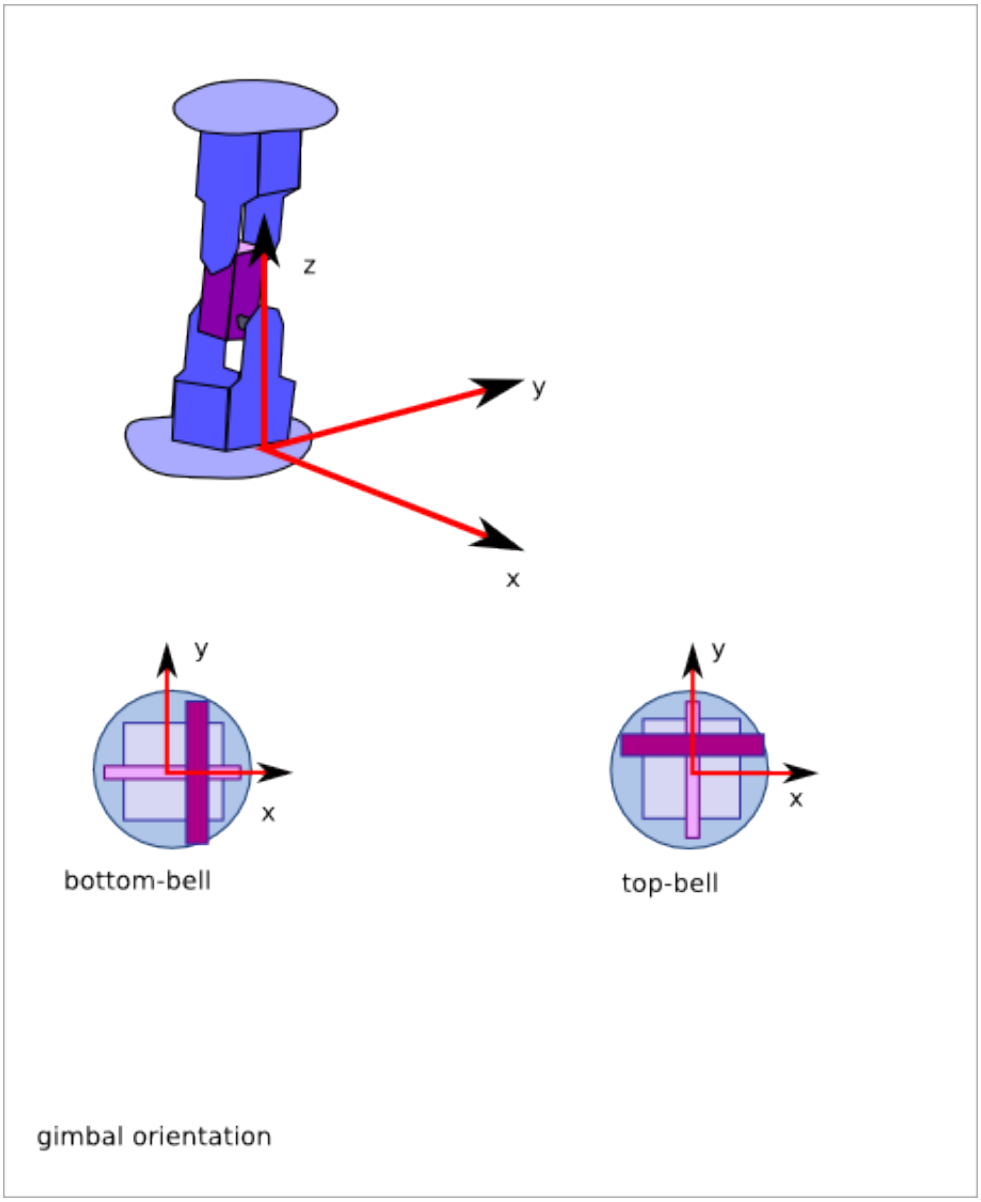


gimbal assembly

The *gimbal* parameters are inherited from Cross_Cube Design and Bell Bagel Assembly. In addition to the *cross_cube* parameters and *bell_bagel* parameters, you have the two *roll-pitch* angles.



roll-pitch angles



36.2 Gimbal Construction

36.2.1 Material

The *bagels* and the *cross_cube* are done with an harder material than the *bell*.

36.2.2 Assembly order

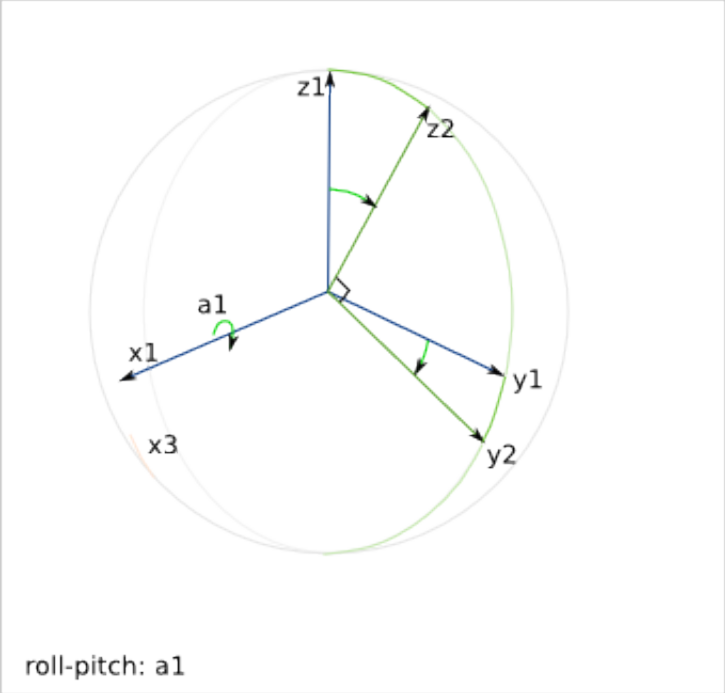
- make the *cross_cube*
- make the *bell* without the *bagels*
- make the *motors*

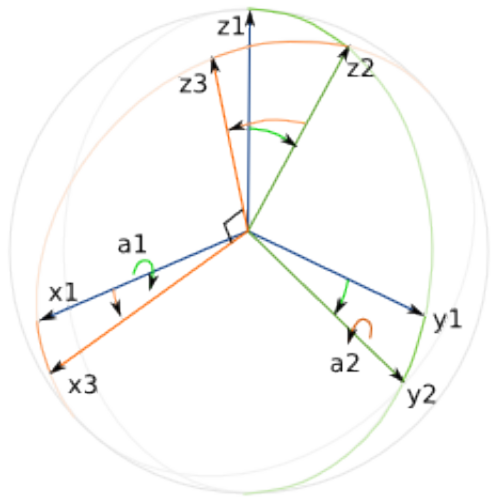
- place the *cross_cube* into the *bell-leg-axle-hole*
- mount the *bagels*
- mount the *motors*

CHAPTER 37

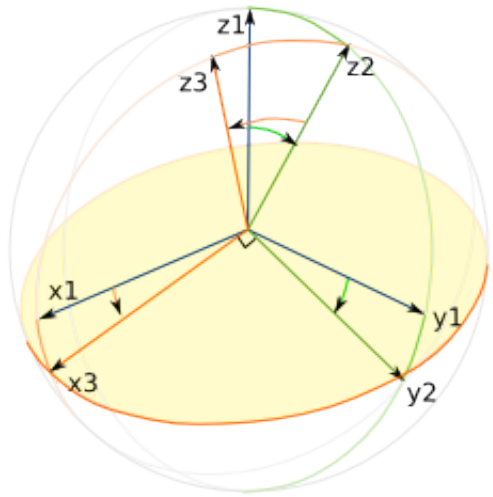
Gimbal Details

37.1 Roll-Pitch angles

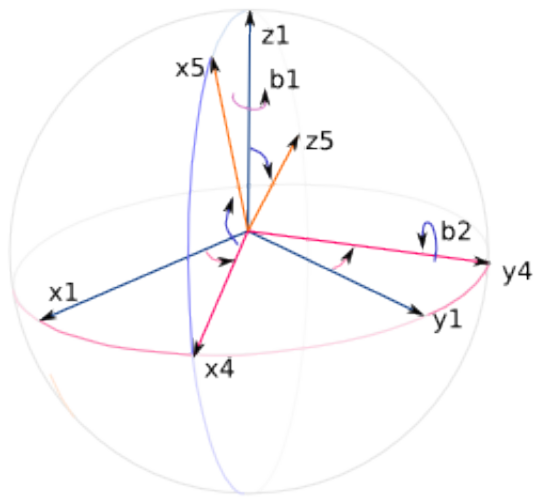




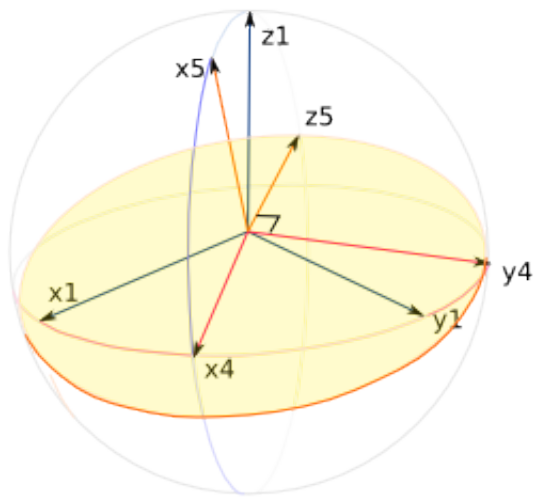
roll-pitch: a1, a2



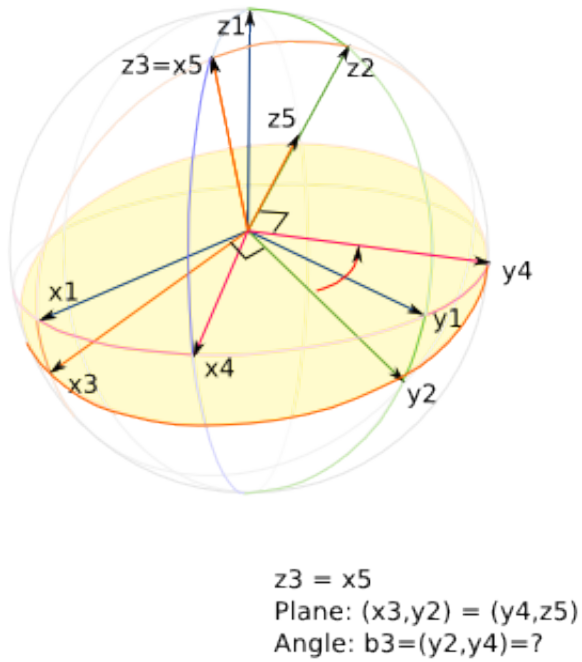
roll-pitch: z3-plane (x3,y2)



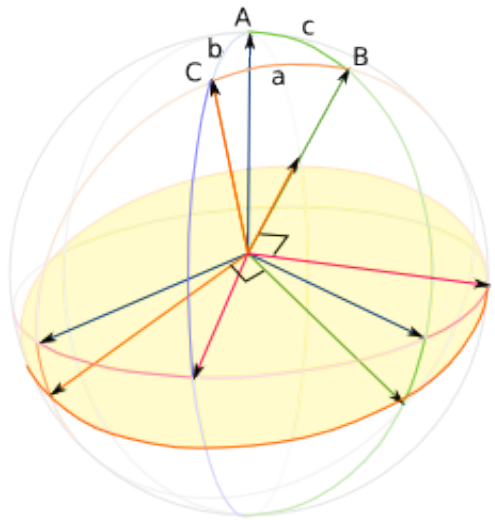
pan-tilt: b1,b2



pan-tilt: x5-plane (y4,z5)



roll-pitch versus pan-tilt



By construction:
c = a1
B = pi/2
a = a2

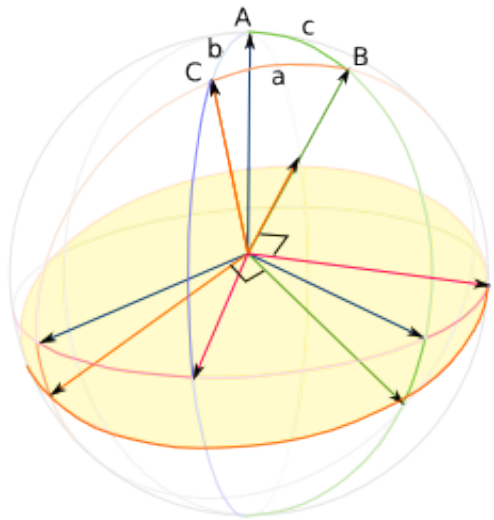
spherical law of cosines:
cos c = cos a cos b + sin a sin b cos C
cos b = cos a cos c + sin a sin c cos B
So,
cos b = cos a2 cos a1

spherical law of sines:
sin a / sin A = sin b / sin B = sin c / sin C

sin A = sin a * sin B / sin b = sin a2 / sin b

b1 = pi/2 - A
b2 = pi/2 - b

roll-pitch to pan-tilt conversion



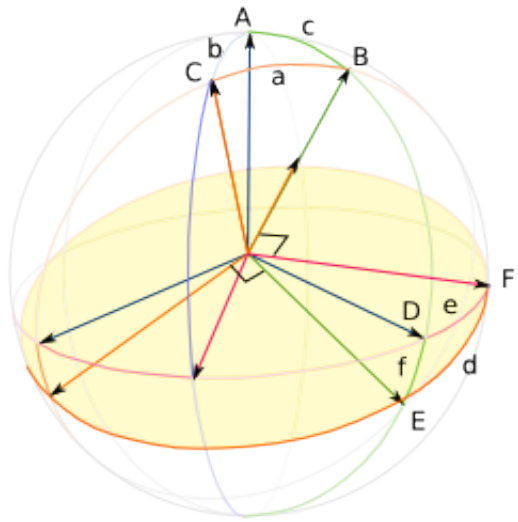
By construction:
b = pi/2-b2
A = pi/2 - b1
B = pi/2

spherical law of sines:
sin a / sin A = sin b / sin B = sin c / sin C
So,
sin a = sin b * sin A / sin B = cos b2 cos b1

dual law of cosines:
cos C = - cos A cos B + sin A sin B cos c
spherical law of cosines:
cos c = cos a cos b + sin a sin b cos C
So,
cos C = - cos A cos B + sin A sin B cos a cos b + sin A sin B sin a sin b cos C
cos C = (sin A sin B cos a cos b - cos A cos B)/(1 - sin A sin B sin a sin b)
cos C = (sin A cos a cos b)/(1 - sin A sin a sin b)
cos C = (cos b1 cos a cos b)/(1 - cos b1 cos b2 cos b1 cos b2)
cos C = (cos b1 cos a sin b2)/$\cos^2$a = cos b1 sin b2 / cos a
and then cos c

a1 = c
a2 = a

pan-tilt to roll-pitch conversion



By construction:
f = a1
D = pi/2
e = b1

spherical law of cosines:
cos d = cos e cos f + sin e sin f cos D
So,
cos d = cos a1 cos b1

angle (y2,y4)

roll-pitch pan-tilt drift angle

CHAPTER 38

Planet_Carrier Design

Ready-to-use parametric *planet_carrier* design. It is composed of the rear and the front planet_carrier. It is used by the Low_torque_transmission Design and High_torque_transmission Design.

To get an overview of the possible *planet_carrier* designs that can be generated by *planet_carrier()*, run:

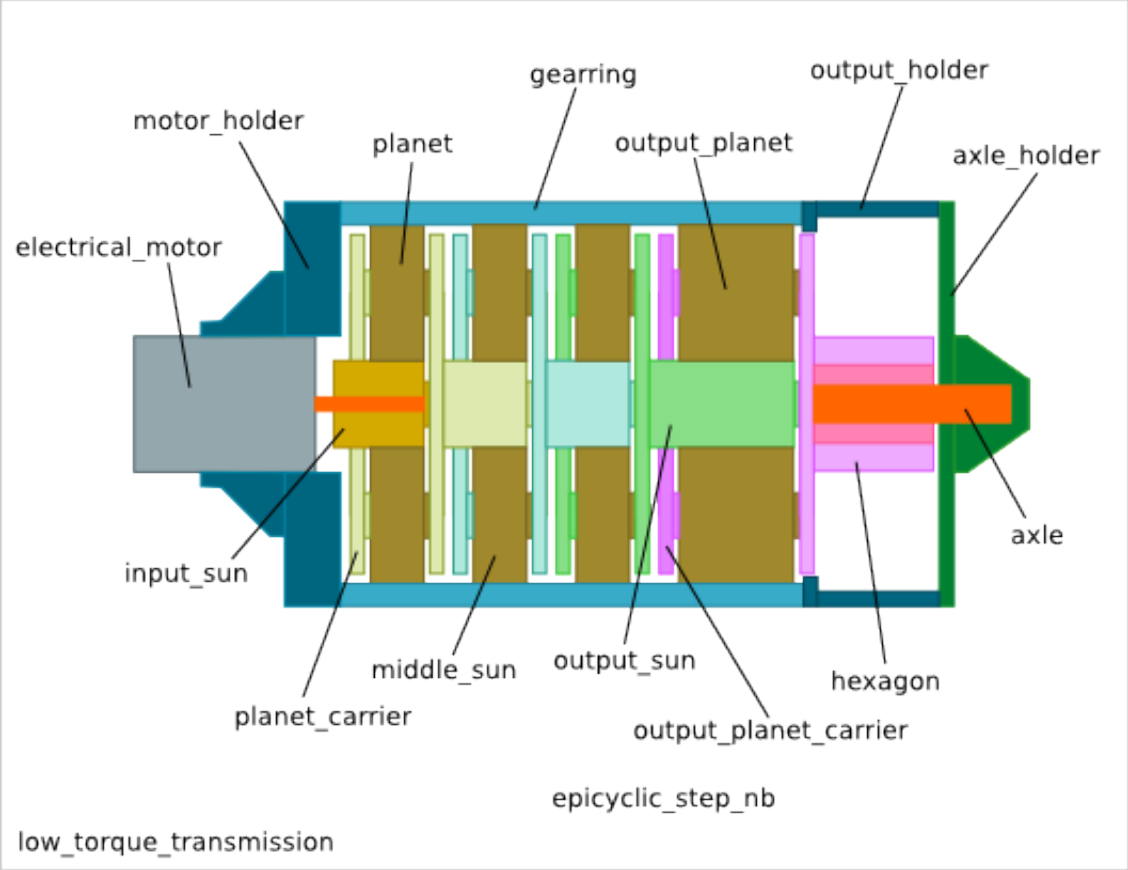
```
> python planet_carrier.py --run_self_test
```

38.1 Planet_Carrier Parameters

38.1.1 Overview

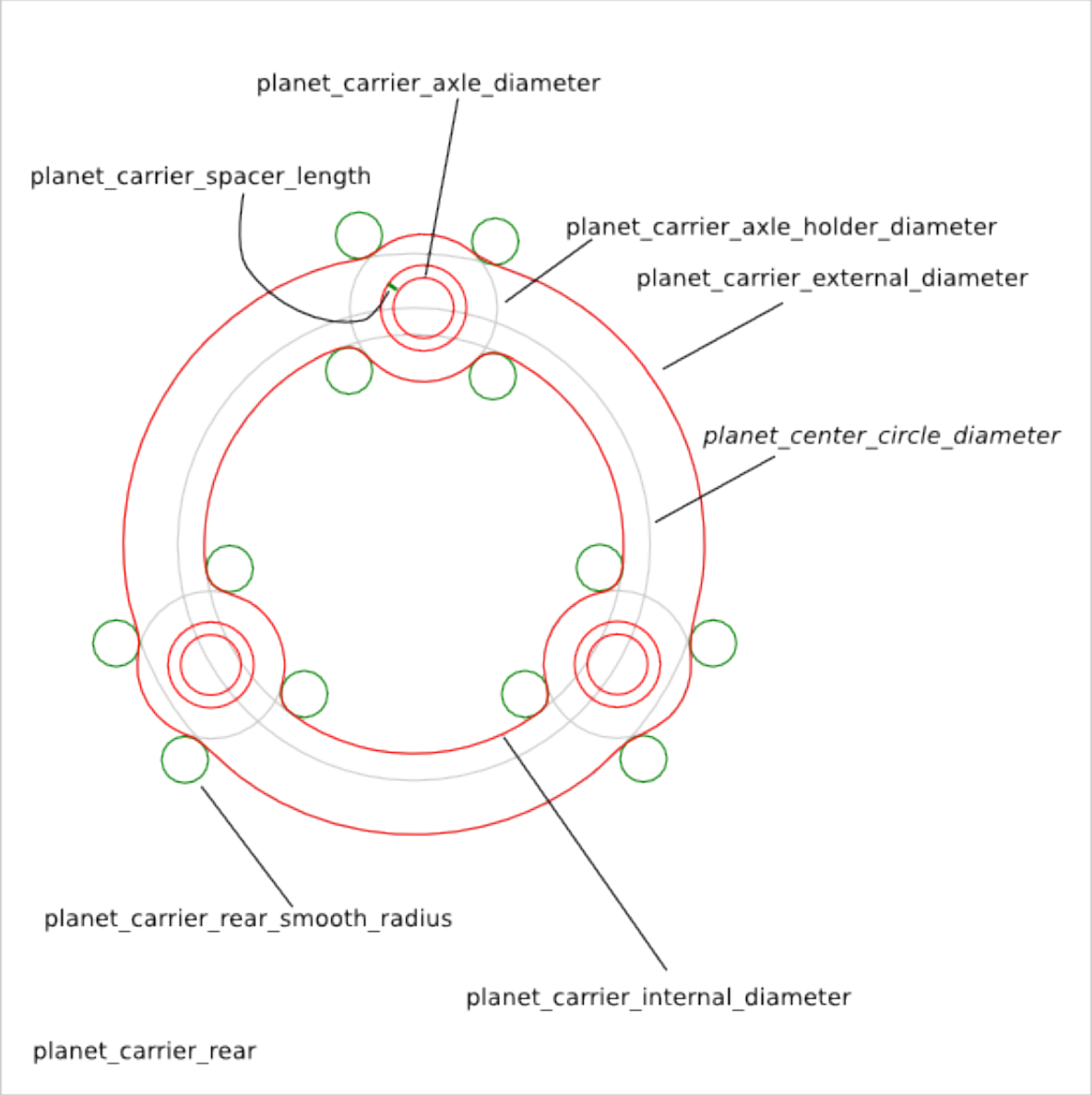
The *planet_carrier* is composed of the following parts:

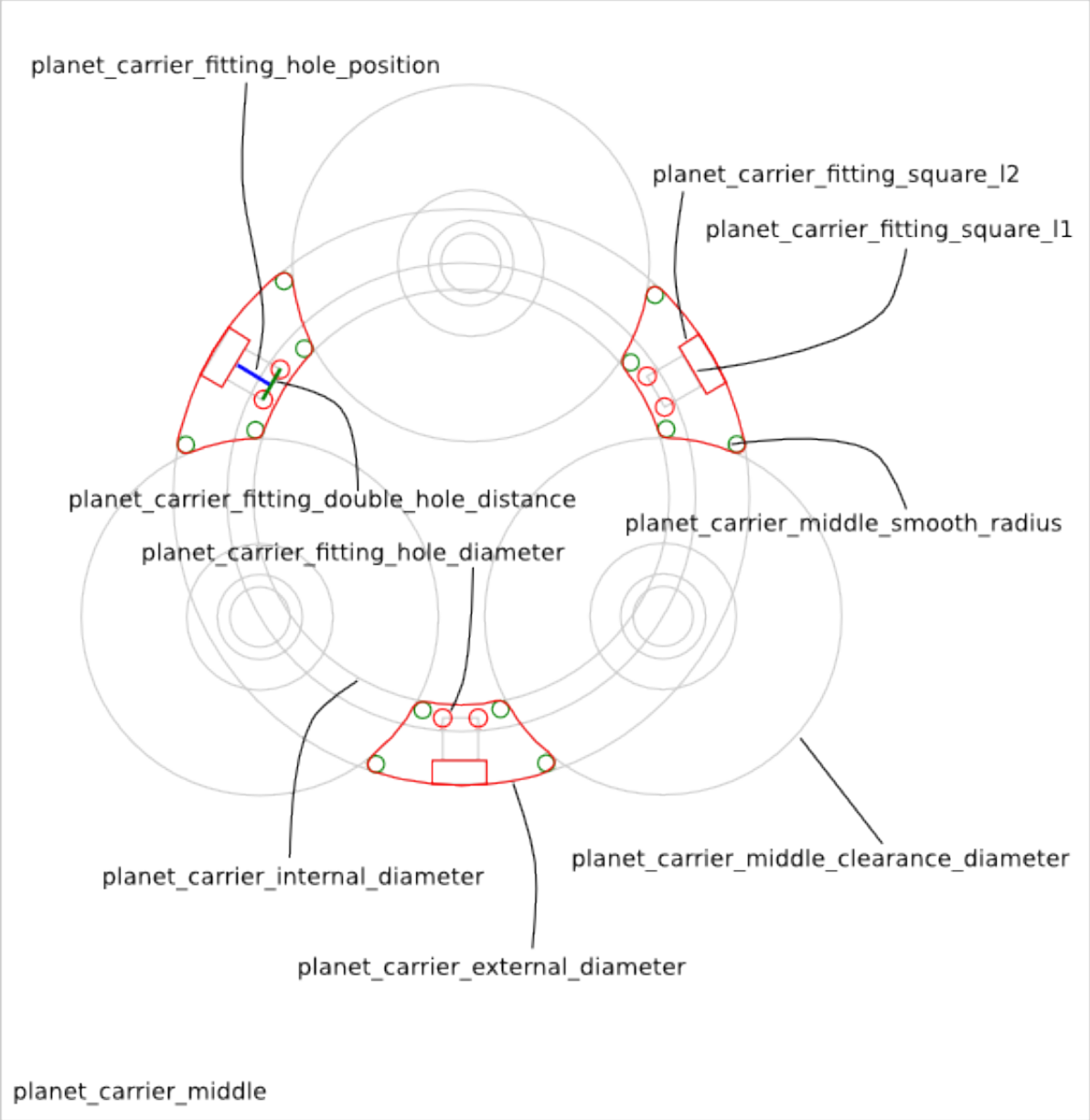
- planet_carrier_rear
- planet_carrier_front

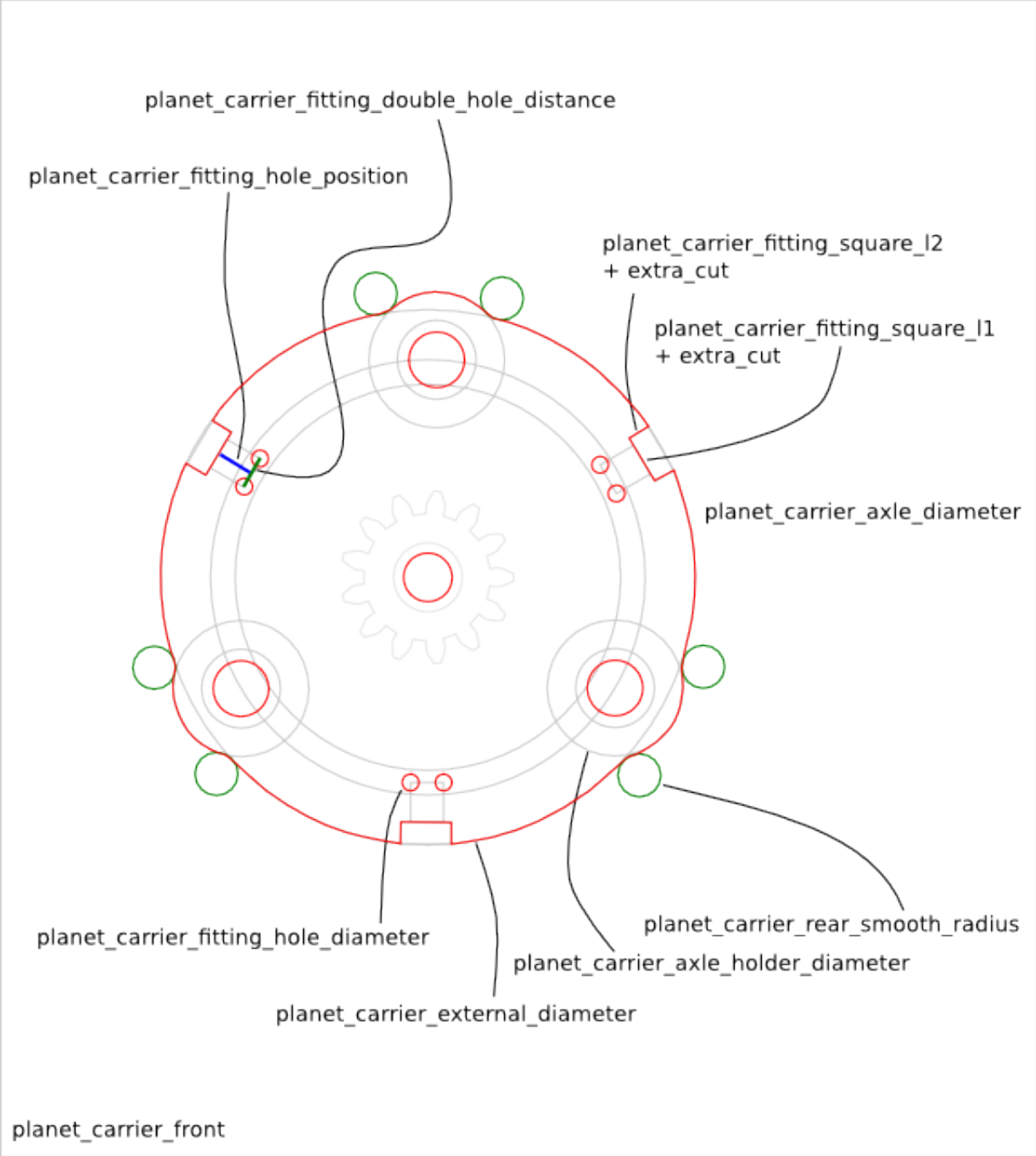


The *planet_carrier_rear* is the fusion of the rear plate the the middle bits. The *planet_carrier_front* is the simple plate that can be fused with a sun-gear in a future design.

38.1.2 Diameters

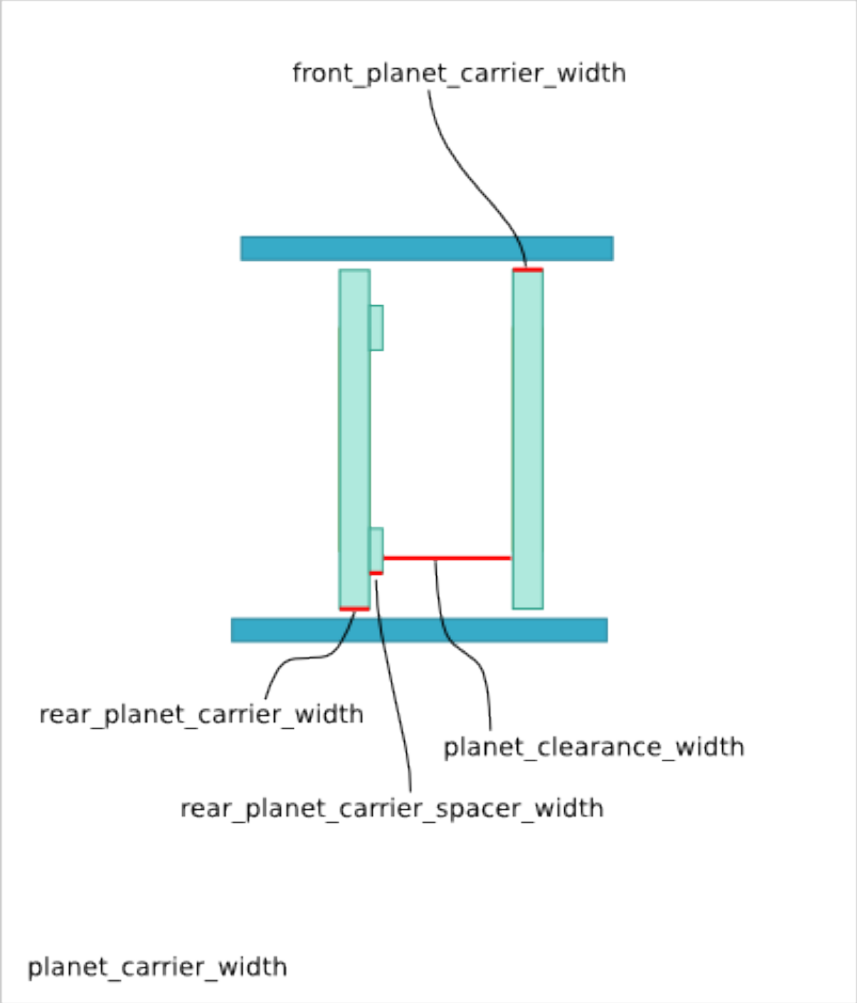






38.1.3 z-direction parameters

The parameters related to the extrusion size in the z-direction:



38.2 Planet_Carrier Parameter Dependency

38.2.1 Diameters

- planet_center_circle_diameter
- planet_carrier_external_diameter
- planet_carrier_internal_diameter

CHAPTER 39

Low_torque_transmission Design

Ready-to-use parametric *low-torque-transmission* design. It is a reduction system based on a train of epicyclic-gearing. The design includes the electric-motor holder for a cylindric or square format. The output is an hexagon on which you can plug a gearwheel. It is a variant of Epicyclic Gearing Design

low_torque_transmission design characteristics:

- train of epicyclic-gearing of n-step (to reach a high reduction ratio)
- same epicyclic profile for the n-step (to get a simple gearring-holder desing)
- same epicyclic width for the n-1 first steps (only the last step might at its yield limits)
- epicyclic with 3 planets (for a more stable planet-holder)
- coaxial electric motor at the input (for a compact and reliable transmission)
- hexagon at the output (for an exchangeable output gearwheel)
- output axle hold on one side only (because on the other side, there is already the motor)

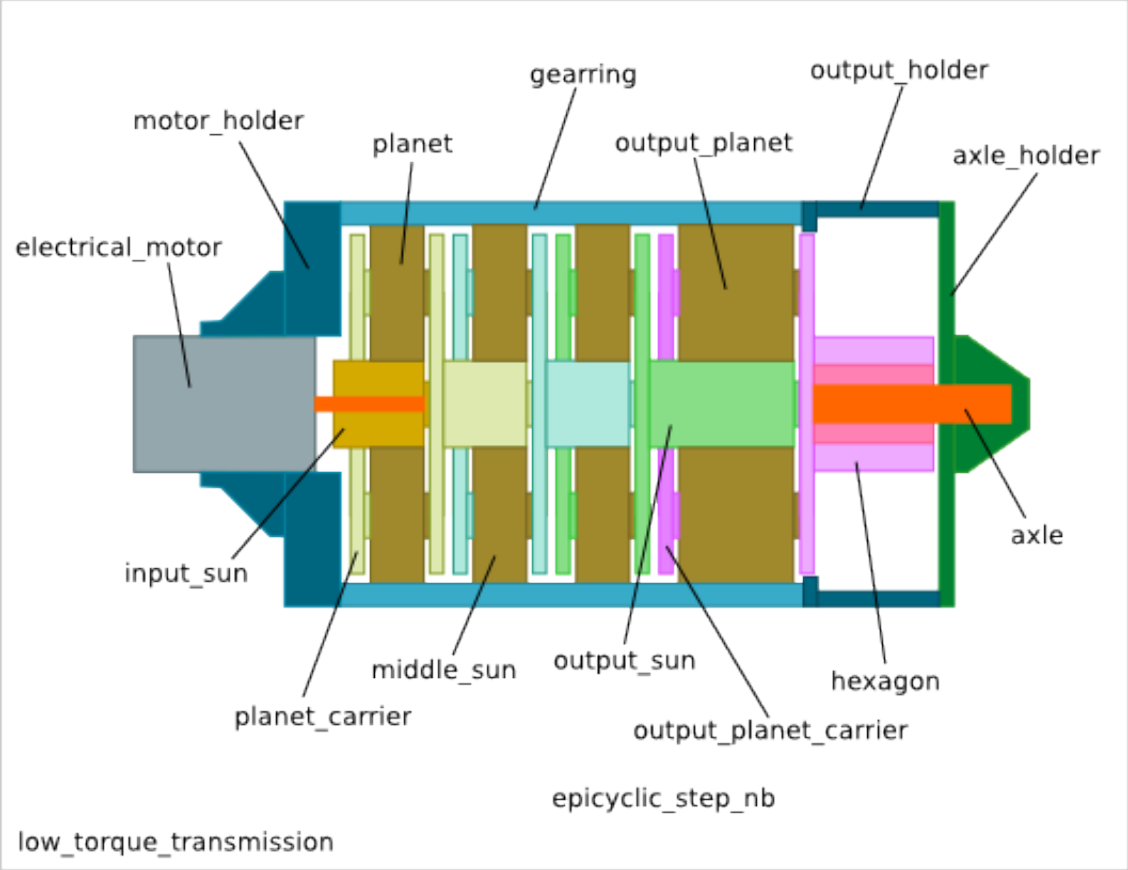
To get an overview of the possible *low_torque_transmission* designs that can be generated by *low_torque_transmission()*, run:

```
> python low_torque_transmission.py --run_self_test
```

39.1 Low_torque_transmission Parameters

39.1.1 Overview

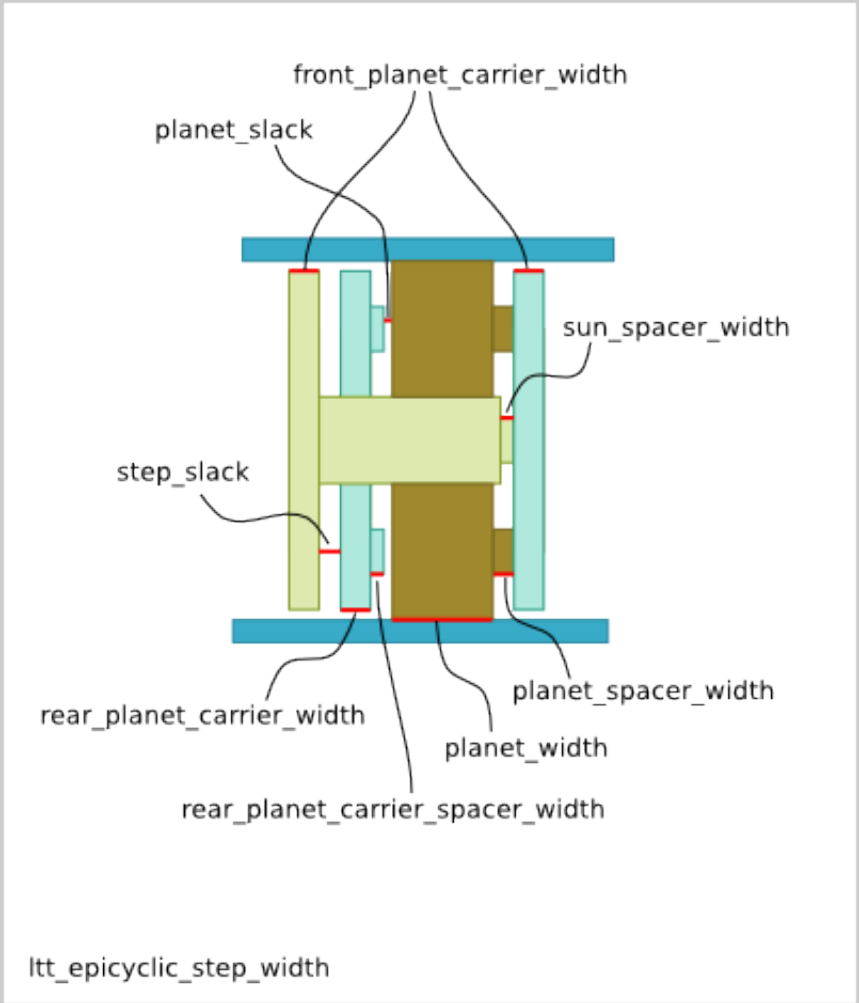
The Low_torque_transmission is composed of the following parts

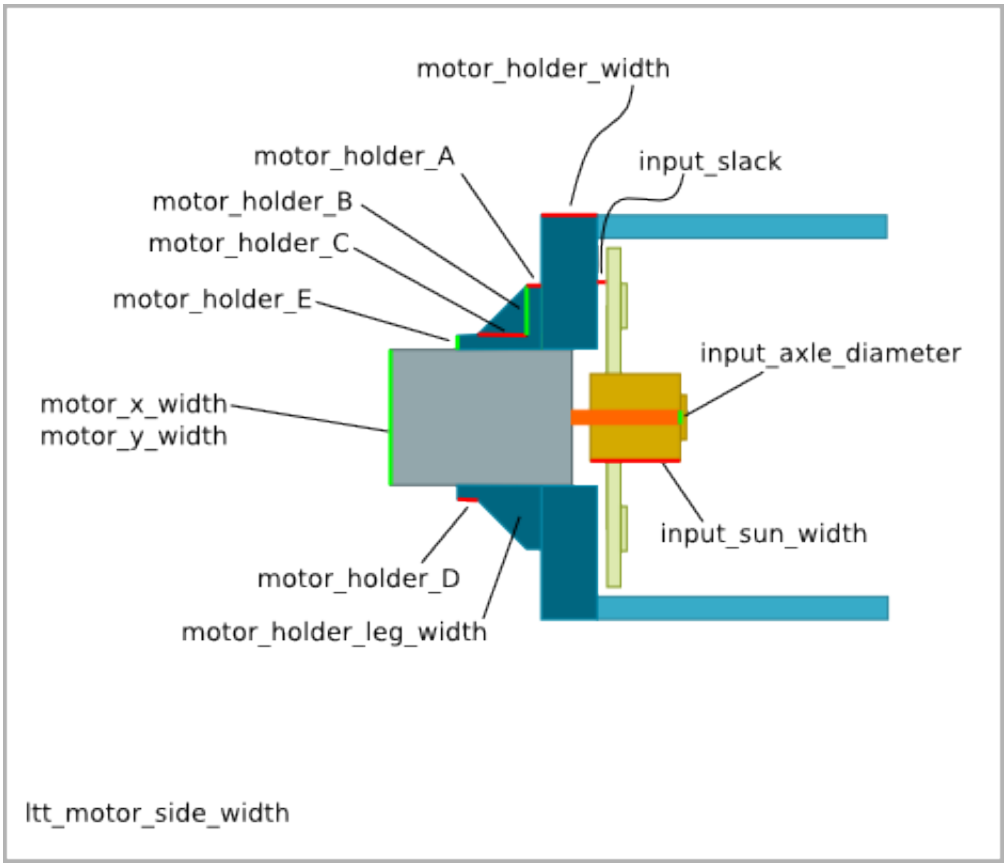


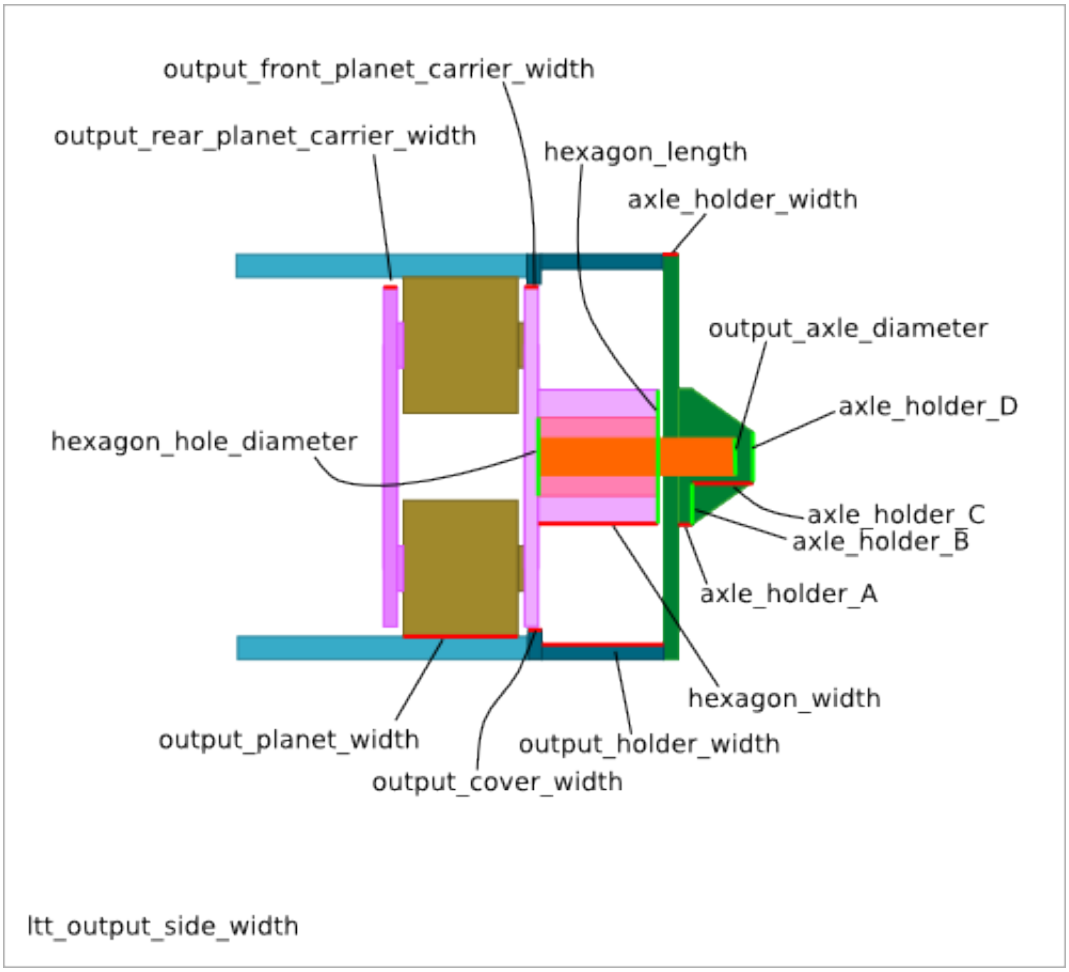
The *low_torque_transmission* inherits the parameters from the Gearring Design. The parameter *epicyclic_step_nb* sets the number of epicyclic-steps.

39.1.2 z-direction parameters

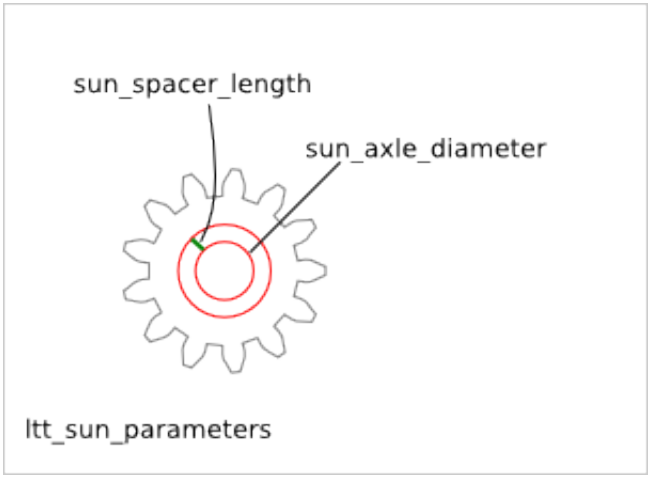
The parameters related to the extrusion size in the z-direction:

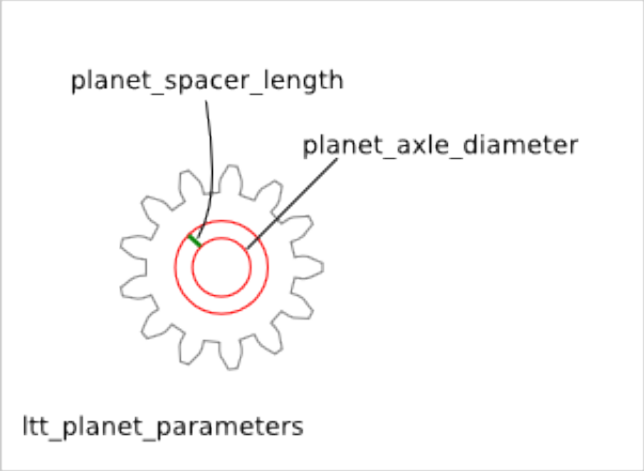




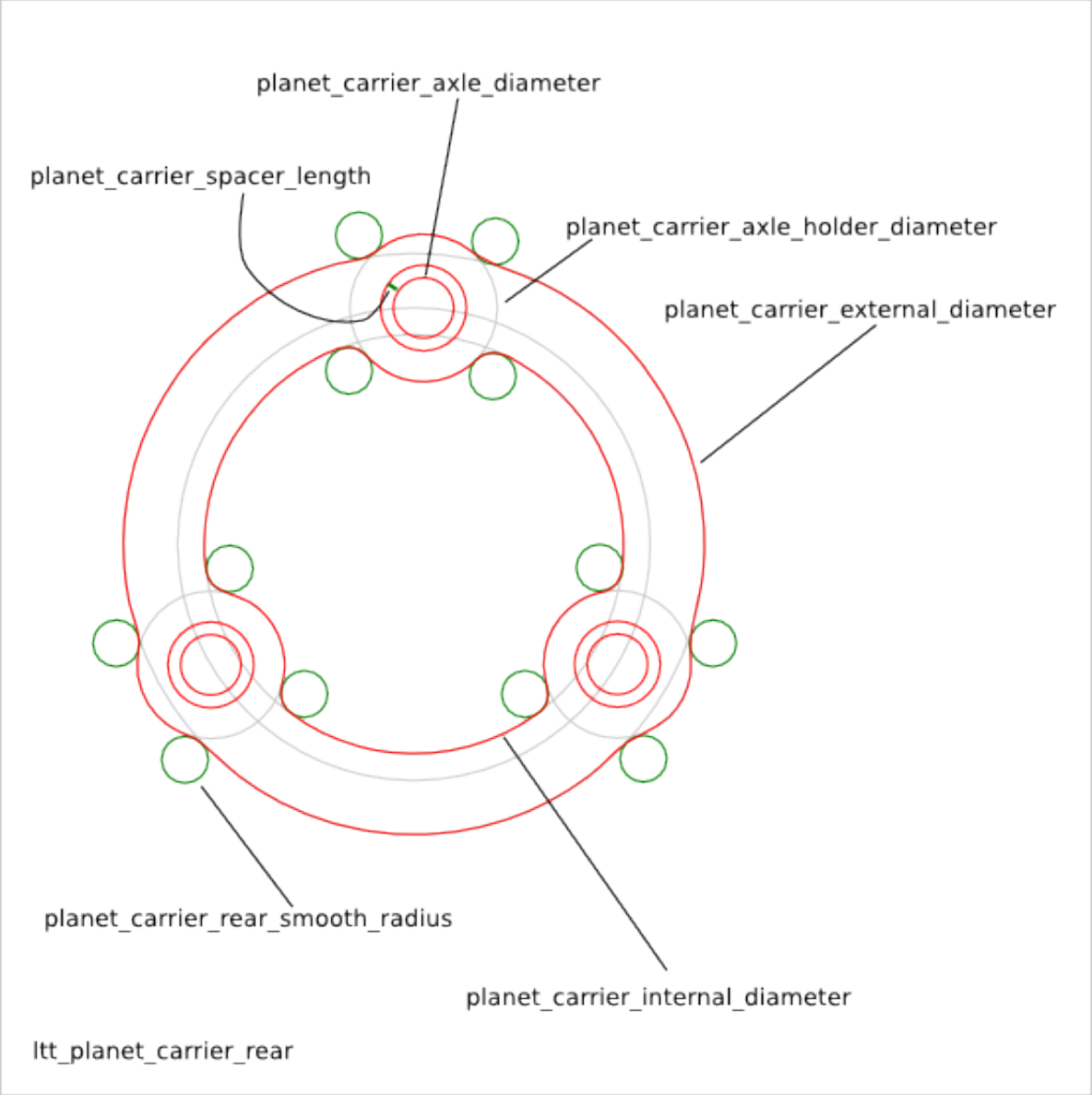


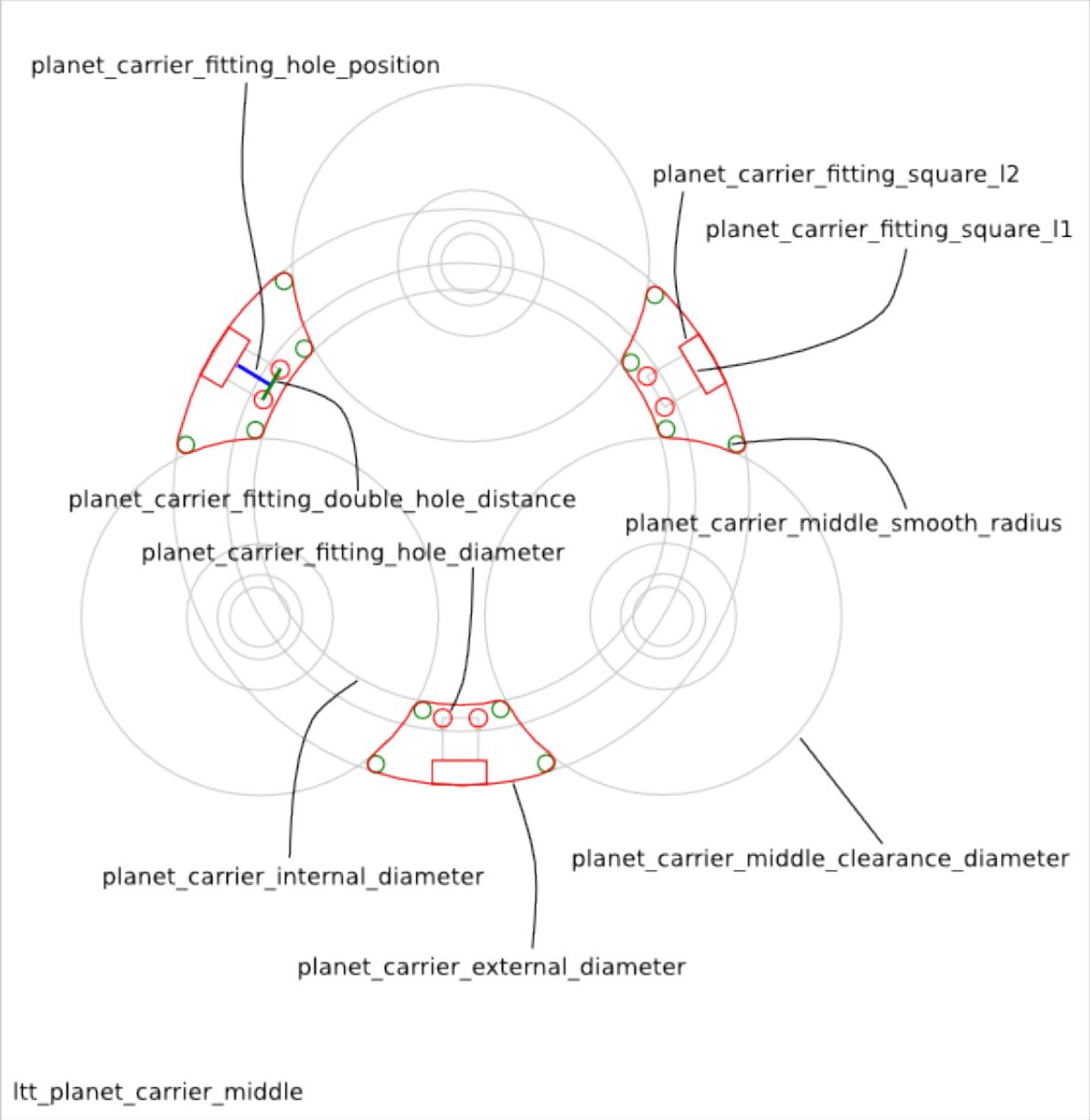
39.1.3 Sun and planet parameters

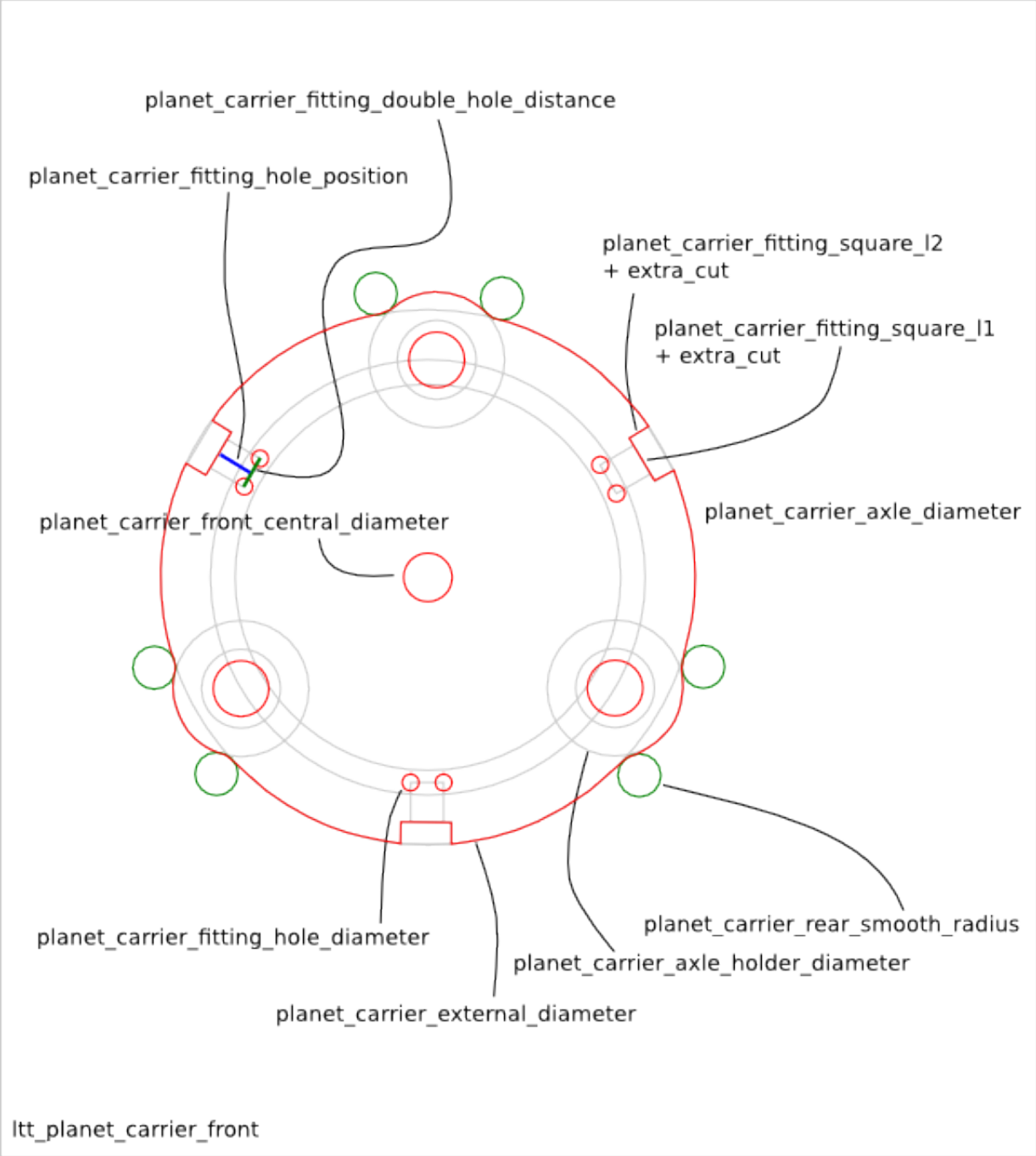




39.1.4 Planet-carrier parameters







39.2 Low_torque_transmission Parameter Dependency

39.2.1 hexagon_width

The *output_hexagon* must into the *output_holder*. But also the *output_front_planet_carrier_width* must be inside the *output-cover to guarantee enough slack between the *output_planet* and the *output_cover*. So we get the relations:

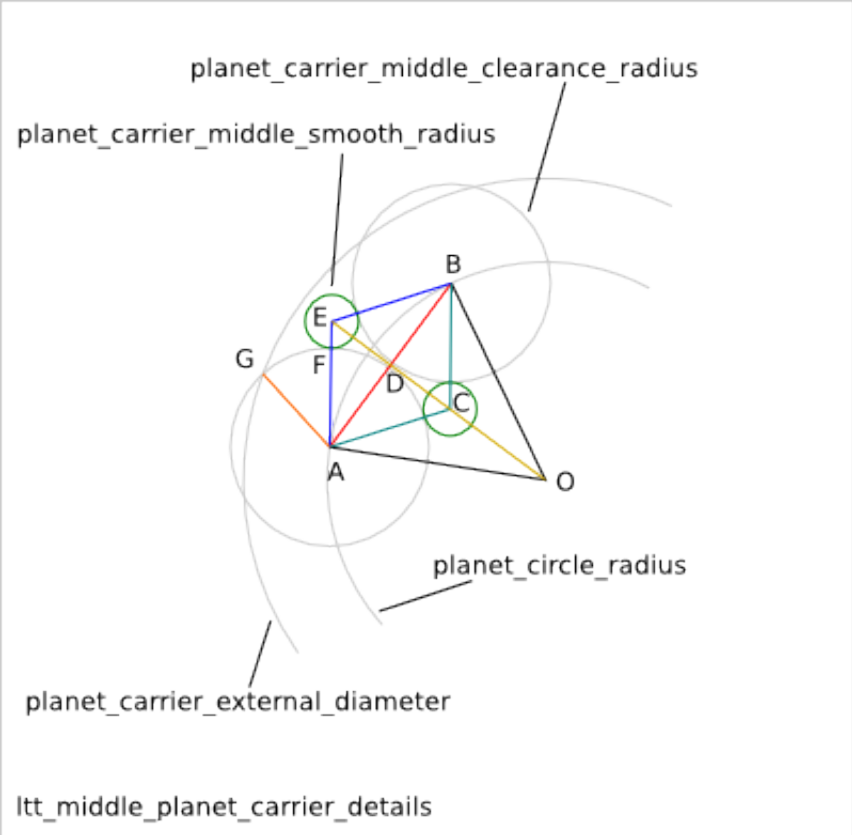
```
output_cover_width + hexagon_width > output_holder_width
hexagon_width < output_holder_width
```

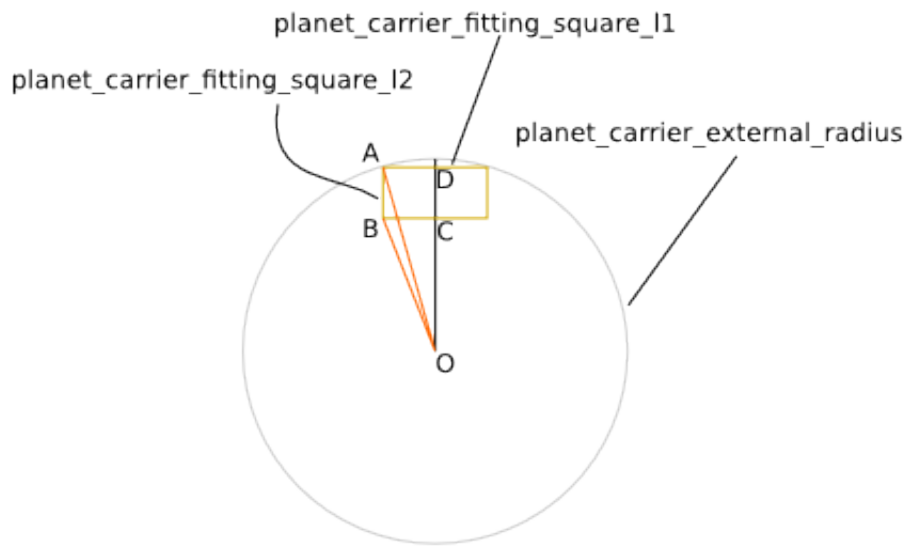
39.2.2 input_slack

The *input_slack* parameter sets some play between the *motor_holder* and the first *rear_planet_carrier*. Notice that this value is affected by the length of the output axle.

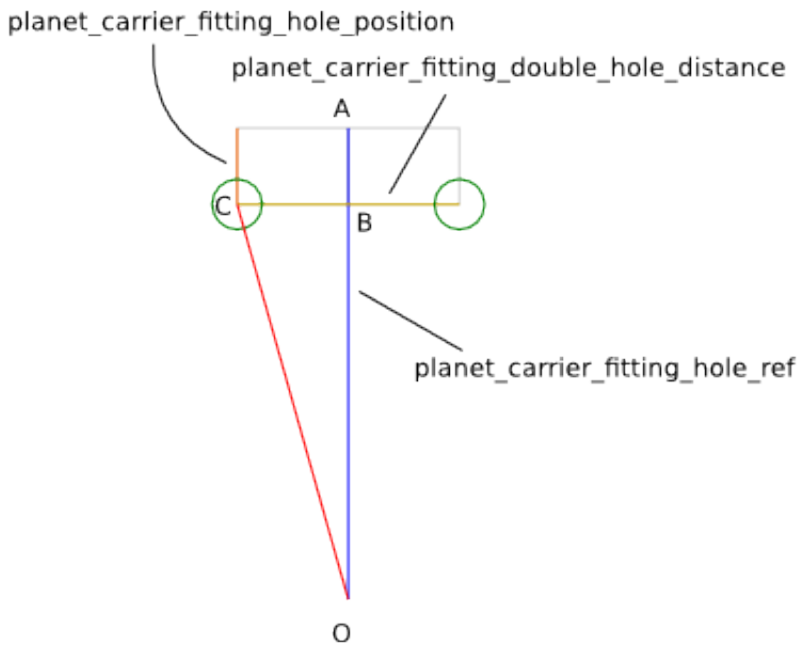
CHAPTER 40

Low_torque_transmission Details





ltt_planet_carrier_fitting_square_details



ltt_planet_carrier_fitting_hole_details

CHAPTER 41

High_torque_transmission Design

Ready-to-use parametric *high-torque-transmission* design. It is a reduction system based on a train of epicyclic-gearing. The design includes an input and an output gearwheel as well as a central axle to support high strength on the input/output gearwheels. It is a variant of Epicyclic Gearing Design.

CHAPTER 42

Indices and tables

- genindex
- modindex
- search