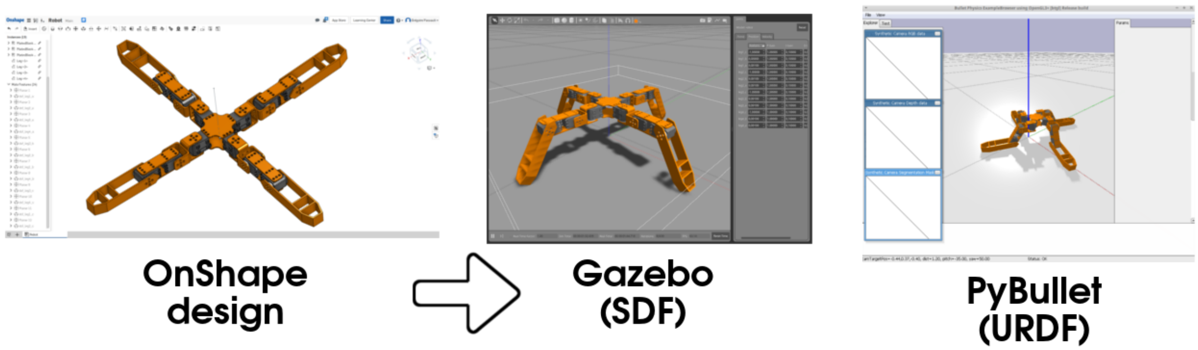
Onshape to robot

Release 1

Mar 14, 2023

Contents:


OnShape
design
Gazebo
(SDF)
PyBullet
(URDF)

CHAPTER 1

What is this ?

`onshape-to-robot` is a tool that allows you to import robots designed in the **Onshape CAD** software to descriptions format like **URDF** or **SDF**, so that you can use them for physics simulation or in your running code (requesting frames, computing dynamics etc.)

- onshape-to-robot GitHub repository
- Robots examples GitHub repository
- onshape-to-robot on pypi

1.1 Installation & requirements

1.1.1 Requirements

You will need an Onshape account and Python 3.

Note: You might also need OpenSCAD for pure shape estimation and meshlab for STLs simplification. Those are not mandatory requirements but will unlock more features:

```
apt-get install openscad meshlab
```

1.1.2 Installation

From pip

Run the following to install onshape-to-robot from pypi:

```
pip install onshape-to-robot
```

From source repository

First, clone the repository:

```
git clone git@github.com:Rhoban/onshape-to-robot.git
```

Install the dependencies (can be in your python3 virtualenv):

```
pip install numpy pybullet requests commentjson colorama numpy-stl
```

You can know use the scripts that are in the root folder of repository (feel free to add it to your `$PATH` and `$PYTHONPATH` to run it from anywhere)

1.1.3 Setting up your API key

To go any further, you will need to obtain API key and secret from the OnShape developer portal

We recommend you to store your API key and secret in environment variables, you can add something like this in your `.bashrc`:

```
// Obtained at https://dev-portal.onshape.com/keys
export ONSHAPE_API=https://cad.onshape.com
export ONSHAPE_ACCESS_KEY=Your_Access_Key
export ONSHAPE_SECRET_KEY=Your_Secret_Key
```

Alternatively, those keys can be stored in the `config.json` file, that will override those parameters (see below). It is however preferred to use environment variables because you can then share safely your *config.json* without sharing your secret keys.

1.2 Commands

Here are the commands provided by *onshape-to-robot*

1.2.1 `onshape-to-robot` - import a robot

This is the main script that run the robot import.

- Check out design time considerations, and how to create *your own config.json* file
- See the examples of robots if you want to try it

Usage:

```
onshape-to-robot [directory containing config.json]
```

1.2.2 `onshape-to-robot-edit-shape` - editing pure shape

This generates a `.scad` file and opens it, allowing you to editing the pure shape approximation of some mesh.

- Check out the *pure shape* page for more information

Usage:

```
onshape-to-robot-edit-shape [stl file]
```

1.2.3 `onshape-to-robot-bullet` - running bullet simulation to test your robot (URDF)

This script starts a bullet simulation with the given robot that was imported with onshape-to-robot.

Usage:

```
onshape-to-robot-bullet [-f] [directory containing robot.urdf]
```

The optional `-f` flag can be passed to fix th robot base (useful for robotics arms for example).

Note also that: * If a joint name ends with `_speed`, it will control its with speed instead of position. * If a joint name ends with `_passive`, it will not be controlled. * If a joint specify its limits, they will be used for side sliders.

1.2.4 `onshape-to-robot-clear-cache` - clearing cache

`onshape-to-robot` uses cache for further call to avoid re-fetching all the data each time. You can use this command to clear the cache.

Usage:

```
.. code-block:: bash
```

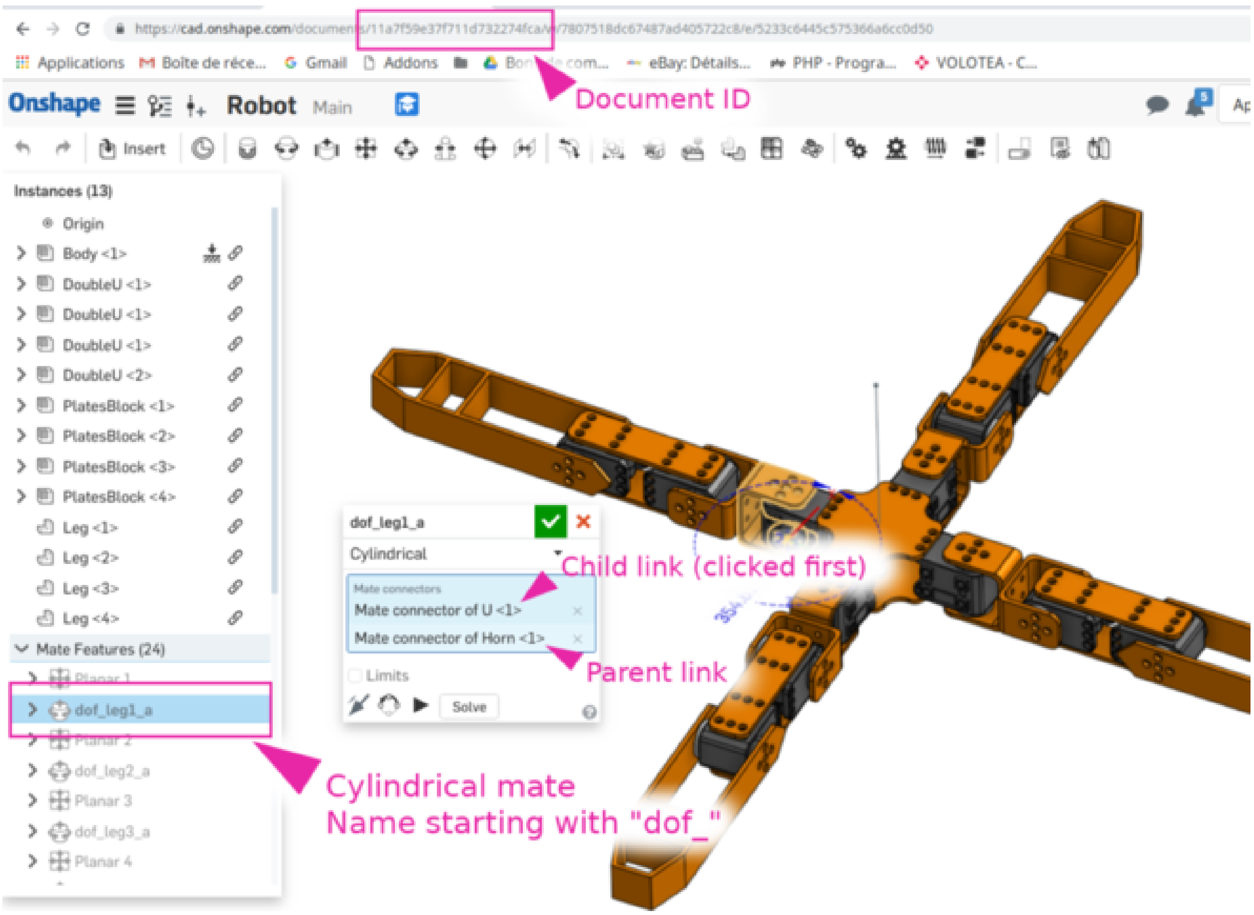
onshape-to-robot-clear-cache

1.3 Design-time considerations

Note: Try to make your robot assembly mostly based on sub pre-assembled components (avoid to have a lot of constraints that are not relevant for the export). In this main assembly, do not use features such as sub-assemblies network.

1.3.1 Specifying degrees of freedom

- Degree of freedoms should be slider, cylindrical or revolute mate connectors named `dof_something`, where `something` will be used to name the joint in the final document * If the mate connector is **cylindrical** or **revolute**, a `revolute` joint will be issued * If the mate connector is a **slider**, a `prismatic` joint will be issued * If the mate connector is **fastened**, a `fixed` joint will be issued
- When doing this connection, click the children joint first. This will be used to find the trunk of the robot (part with children but no parent)



1.3.2 Inverting axis orientation

It is possible to invert the axis for convenience by adding `_inv` at the end of the name. For instance `dof_head_pitch_inv` will result in a joint named `head_pitch` having the axis inverted with the one from the OnShape assembly.

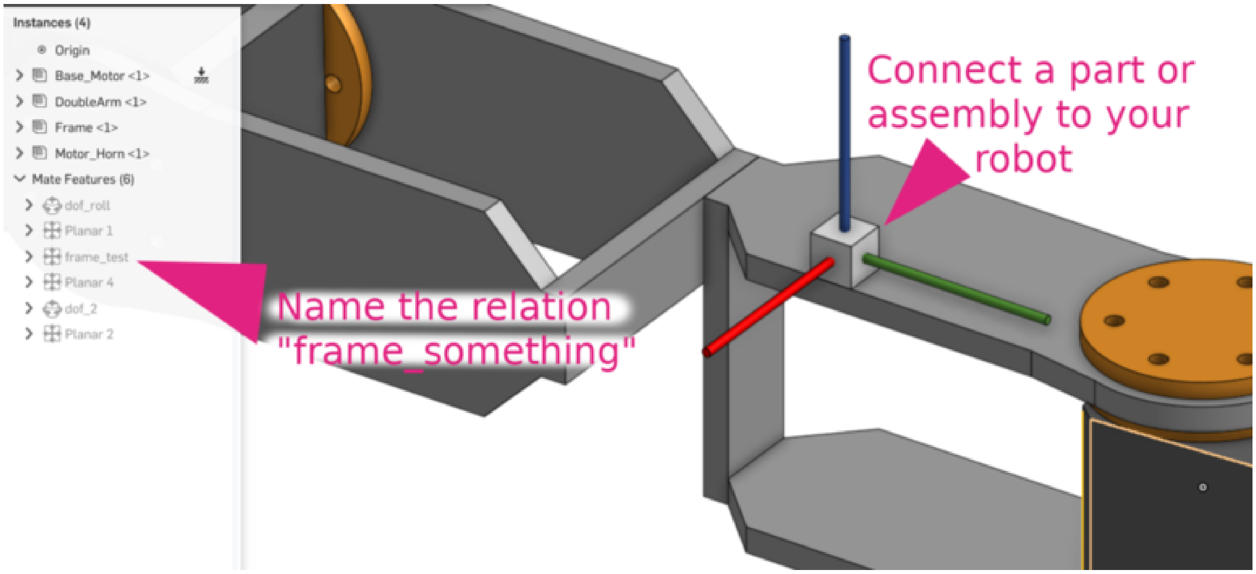
1.3.3 Naming links

If you create a mate connector and name it `link_something`, the link corresponding to the part on which it is attached will be named `something` in the resulting URDF.

1.3.4 Adding custom frames in your model

If you want to track some frames on your robot, you can do the following:

- Connect any part to your robot using mate relations in OnShape
- Name one of these relations `frame_something`, when `something` will be the name of the frame (a link) in the resulting `sdf` or `urdf`

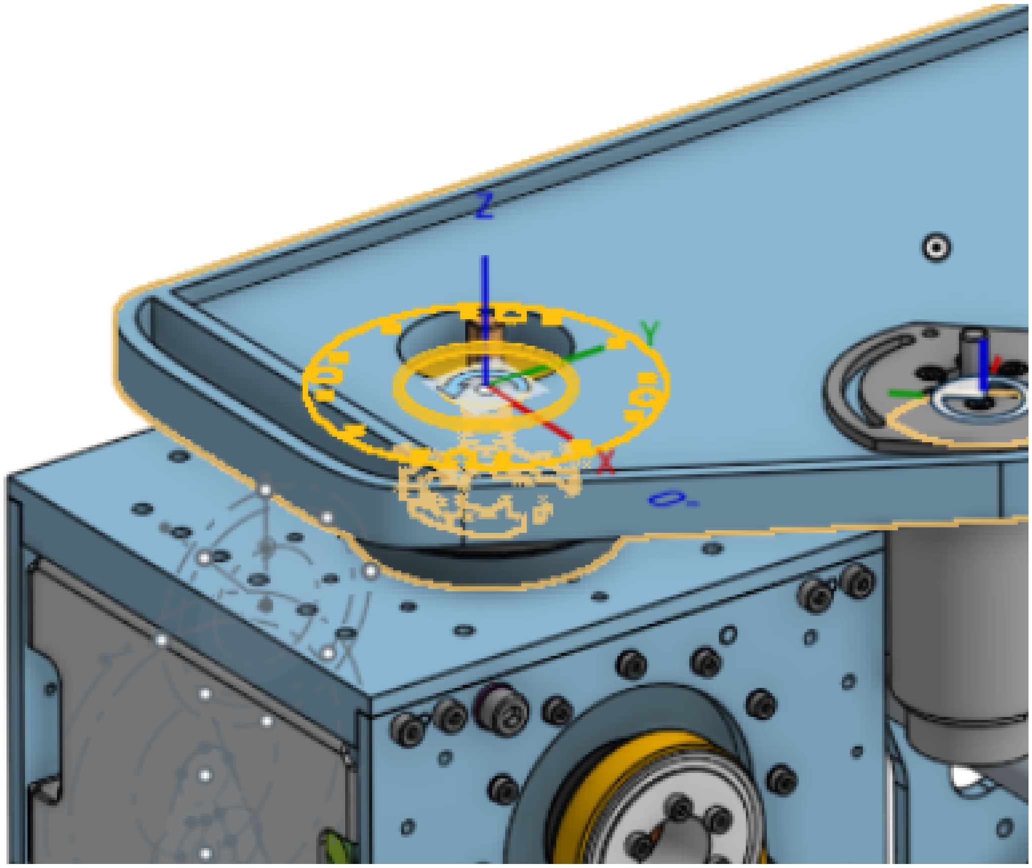


If you want to give it a try, you can use the `onshape-to-robot-bullet` in `urdf` mode, it will output the frames on standard output.

Here is a link of a document that can be used as a frame (note: the center cube is 5mm side, so you might need 2.5mm offset to center it).

1.3.5 Joint frames

Joint frames are the ones you see in OnShape when you click on the joint in the tree on the left. Thus, they are always revolving around the z axis, or translating along the z axis, even if the `_inv` suffix is added.



1.4 Export your own robot (writing config.json)

To export your own robot, first create a directory:

```
mkdir my-robot
```

Then edit `my-robot/config.json`, here is the minimum example:

```
{
    "documentId": "document-id",
    "outputFormat": "urdf"
}
```

The `document-id` is the number (below XXXXXXXXX) you can find in Onshape URL:

```
https://cad.onshape.com/documents/XXXXXXXXX/w/YYYYYYYY/e/ZZZZZZZZ
```

Once this is done, if you properly *installed and setup your API key*, just run:

```
onshape-to-robot my-robot
```

1.4.1 `config.json` entries

Here is the full list of possible entries for this configuration.

`onshape_api`, `onshape_access_key` and `onshape_secret_key`

Those allow you to specify the URL for onshape API (`onshape_api`), and your API key (`onshape_access_key`) and secret key (`onshape_secret_key`).

> **Warning:** Even if this is possible to specify the api key in the `config.json` file, it is very sensitive and we recommend you to set them up as environment variable. See the following section: *Setting up API key section*.

`documentId`

This is the onshape ID of the document to be imported. It can be found in the Onshape URL, just after `document/`.

`outputFormat`

required

This should be either `urdf` or `sdf` to specify which output format is wanted for robot description created by the export.

`assemblyName`

optional

This can be used to specify the name of the assembly (in the Onshape document) to be used for robot export. If none is used, the first assembly found will be used.

`workspaceId`

optional, no default

This argument can be used to use a specific workspace of the document. This can be used for specific branches ofr your robot without making a version. The workspace ID can be found in URL, after the `/w/` part when selecting a specific version in the tree.

`versionId`

optional, no default

This argument can be used to use a specific version of the document instead of the last one. The version ID can be found in URL, after the `/v/` part when selecting a specific version in the tree.

If it is not specified, the very last version will be used for import.

`configuration`

optional, default: "default"

This is the robot configuration string that will be passed to Onshape. An example of format:

```
left_motor_angle=3+radian;enable_yaw=true
```

`drawFrames`

optional, default: false

When *adding custom frames to your model*, the part that is used for positionning the frame is by default excluded from the output description (a dummy link is kept instead). Passing this option to `true` will keep it instead.

`drawCollisions`

optional, default: false

If you use *pure shapes approximations*, the collisions in your description will not be meshes but shapes like boxes, cylinders etc. If you pass this argument to `true`, it will use the same output in the `visual` tag, making the visual similar to what is used for collisions.

This can be used for debugging, but also to lighten the robot visualization if it is complex during experiments and avoiding loading meshes just for visualization.

`useScads`

optional, default: true (needs openscad installed)

If you create *pure shapes approximations* of your parts, you will have `.scad` files sitting in your directory, this flag can be used to disable using them (if `false`, full meshes will be then used for collisions).

`pureShapeDilatation`

optional, default: 0

If you want to use pure shape as safety check for collisions, you can use this parameter to add some extra dilatation to all of them.

`jointMaxEffort` and `jointMaxVelocity`

optional, default: 1 and 20

Those parameters can be used to specify the values that will be included in the `joint` entries.

Alternatively, they can be dictionaries associating named joints to the values.

`dynamics`

optional, default: {}

This `dict` can be used to override the mass and inertia computed by Onshape for a specific part. See *example* below.

`noDynamics`

optional, default: false

This flag can be set if there is no dynamics. In that case all masses and inertia will be set to 0. In pyBullet, this will result in static object (think of some environment for example).

`ignore`

optional, default: []

This can be a list of parts that you want to be ignored during the export.

Note: the dynamics of the part will not be ignored, but the visual and collision aspect will.

`whitelist`

optional, default: None

This can be used as the opposed of `ignore`, to import only some items listed in the configuration (all items not listed in `whitelist` will be ignored if it is not `None`)

`color`

optional, default: None

Can override the color for parts (should be an array: `[r, g, b]` with numbers from 0 to 1)

`packageName`

optional

Prepends a string to the paths of STL files. This is helpful for ROS users as they often need to specify their `robot_description` package.

`addDummyBaseLink`

optional

Adds a `base_link` without inertia as root. This is often necessary for ROS users.

`robotName`

optional

Specifies the robot name.

`additionalXML`

optional

Specifies a file with XML content that is inserted into the URDF/SDF at the end of the file. Useful to add things that can't be modelled in onshape, e.g. simulated sensors.

`useFixedLinks`

optional, default: false

With this option, visual parts will be added through fixed links to each part of the robot. Mostly, this feature is a hack to keep colors properly for rendering in PyBullet (see https://github.com/bulletphysics/bullet3/issues/2650).

`mergeSTLs`

optional, default: "no"

Can be "no", "visual", "collision" or "all".

This can be used to merge STLs file of the same `link` into one unique STL. It is actually better combined with `simplifySTLs`, that can be used to reduce the STL file sizes.

Note: this will only merge visual for visual, see ''mergeSTLsCollisions''

`mergeSTLsCollisions`

optional, default: false

STLs used for collisions will also be merged if this flag is `true`. Note that

`simplifySTLs`

optional, default: "no"

Can be "no", "visual", "collision" or "all".

If this is set, the STL files will be reduced (see `maxSTLSize`). This requires `meshlab` tool (`sudo apt-get install meshlab`).

`maxSTLSize`

optional, default: 3

This is the maximum size (in `M`) of STL files before they are reduced by `simplifySTLs`.

`useCollisionsConfigurations`

optional, default: true

With this option (enabled by default), the collisions=true configuration will be passed when exporting STL meshes (and NOT dynamics), in order to retrieve simplified mesh parts from OnShape.

This is a way to approximate your robot with simpler meshes.

`postImportCommands`

optional, default: []

This is an array of commands that will be executed after the import is done. It can be used to be sure that some processing scripts are run everytime you run onshape-to-robot.

1.4.2 Example `config.json` file

Here is an example of configuration:

```
{
    // You should store those three in environment variables
    "onshape_api": "https://cad.onshape.com",
    "onshape_access_key": "[KEY]",
    "onshape_secret_key": "[SECRET]",

    // Can be found in the URL when editing the assembly
    "documentId": "483c803918afc4d52e2647f0",
    // If not specified, the first assembly will be used
    "assemblyName": "robot",
    // Can be urdf or sdf
    "outputFormat": "urdf",
    // The frames parts are kept in the final file
    "drawFrames": false,
    // Collisions (pure shapes) are also used in the visual section
    "drawCollisions": false,
    // Wether or not the scan for SCAD files (pure shapes) should be done
    "useScads": true,
    // Masses, com and inertias will be zero (can be used if you import a static
    // field for example)
    "noDynamics": false,
    // Should the STLs of the same link be merged?
    "mergeSTLs": "no",
    // Should we simplify STLs files?
    "simplifySTLs": "no",
    // Maximum size (M) of STL files to run simplification (required meshlab)
    "maxSTLSize": 3,

    // Those can be used to configure the joint max efforts and velocity, and
    // overriden for specific joints
    "jointMaxEffort": {
        "default": 1.5,
        "head_pitch": 0.5
    },
    "jointMaxVelocity": 22,

    // This can be used to override the dynamics of some part (suppose it's a compound
    // which dynamics is well specified)
    "dynamics": {
        "motorcase": {
            "mass": 0.5,
            "com": [0, 0.1, 0],
            "inertia": [0.1, 0, 0,
                        0, 0.1, 0,
                        0, 0, 0.1]
        },
        // "fixed" can be used to assign a null mass to the object, which makes it fixed (non-dynamics)
        "base": "fixed"
    },

    // Some parts can be totally ignored during import
    "ignore": [
        "small_screw",
        "small_nut"
    ]
}
```

1.4.3 Testing your robot in simulator

You can then use the `onshape-to-robot-bullet my-robot` command to give a try to your robot.

1.5 Pure shapes approximation

By default, meshes are also used for collision. This is versatile but is computationally expensive, and can be numerically instable.

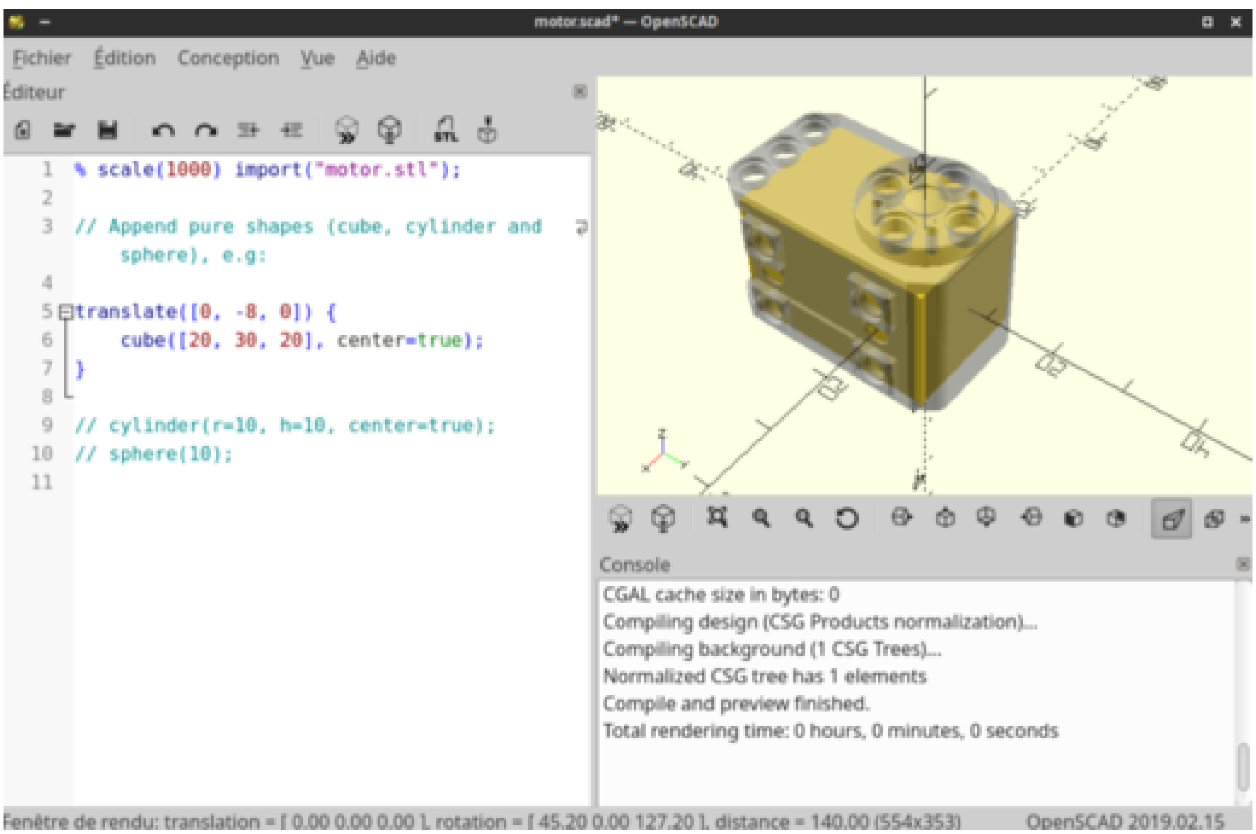
You can approximate those parts with pure shapes (namely boxes, spheres and cylinders). For this, we propose a solution based on `openscad`:

```
apt-get install openscad
```

1.5.1 Approximating a mesh

To do approximation, you need to create a `.scad` file next to `.stl` one. For instance `motor.scad` that will approximate `motor.stl` file.

You can use `onshape-to-robot-edit-shape [stl-file]` that will automatically prepare and run the `.scad` using a template visualizing the `.stl` with transparency, allowing you to edit the pure shapes related:



You will then have to write manually some scad code that approximate your shape.

Note: Only the union of the written shape will be used. When using `onshape-to-robot-edit-shape`, some examples will appear commented. You can use `cube`, `sphere` and `cylinder`, along with `translate` and `rotate` to mainly do t he job. Have a look at examples for more insight.

Then, the pure shapes from your scad will be used when generating the `sdf` or `urdf` file (next time you will run `onshape-to-robot`, it will read your `.scad` files, assuming `useScads` is not `false`).

1.5.2 Part with no collision

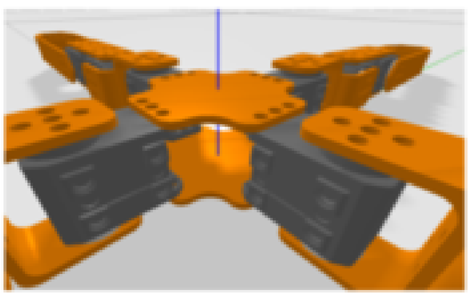
If the `.scad` file is empty, your part will have no collision.

1.5.3 Re-using the mesh

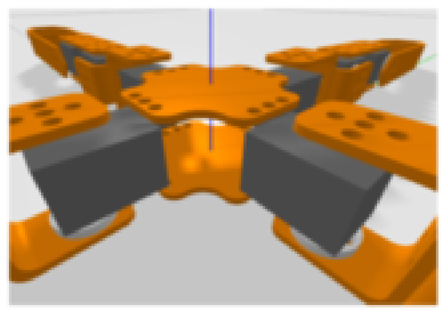
To use the mesh for collision, simply remove the `.scad` that has the same name that the `.stl` file.

1.5.4 Drawing collisions

If you pass *drawCollisions* to *true*, the collisions will also be used for the render, which can be useful to debug:



Default
(meshes)



motor.scad approximation,
drawCollisions=true