COMPAS

Swinburne University of Technology

Robotic Fabrication with COMPAS FAB

AGENDA

intro

kinematics

planning

scene & end effectors

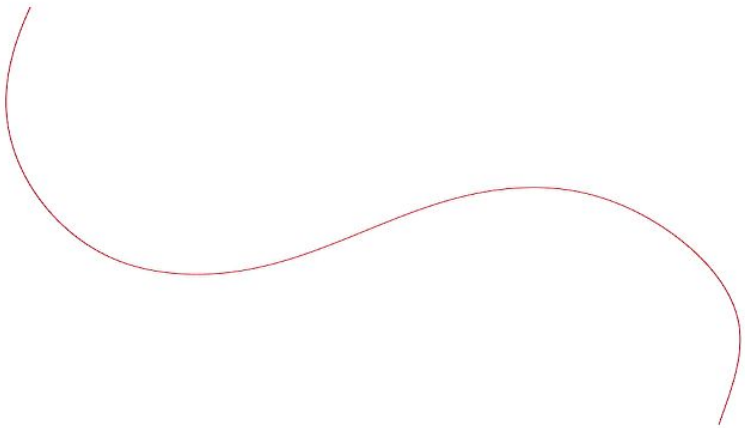
pick and place

assemblies

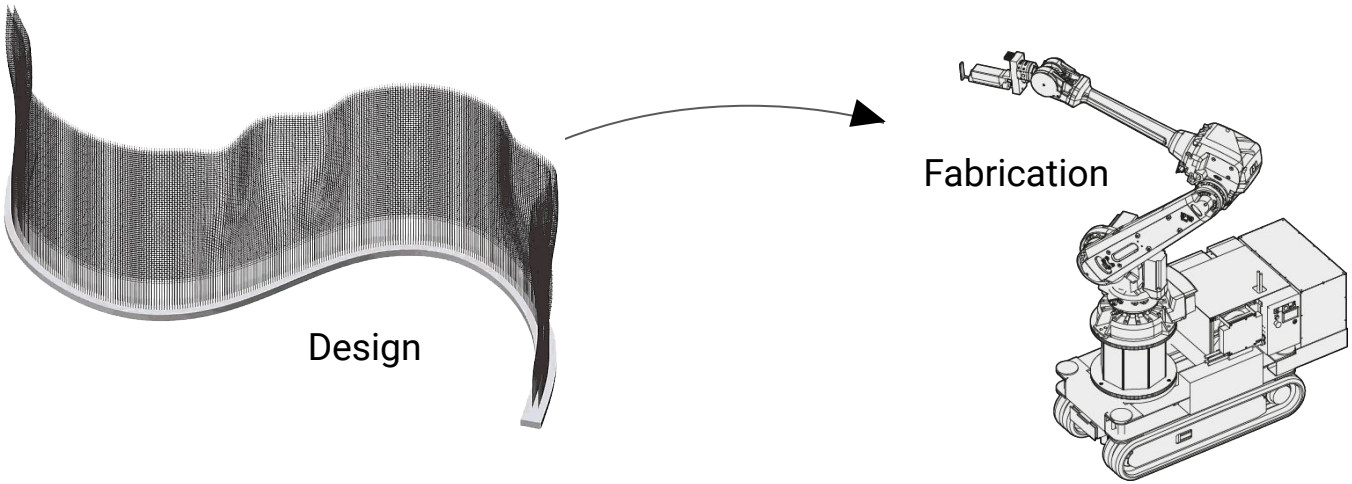
robot control

intro

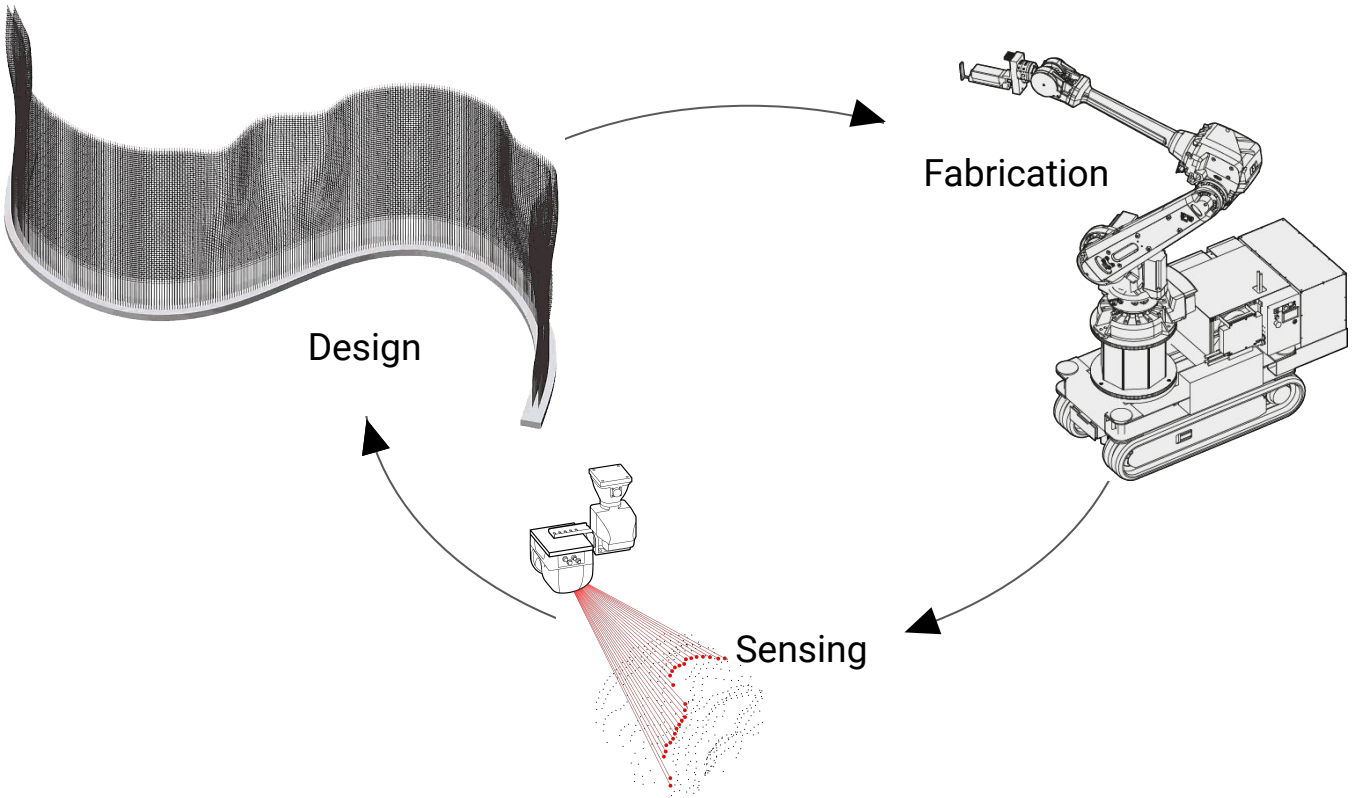
Computational design



Design to fabrication



Design to fabrication



Interdisciplinarity

DFAB House

HiLo Roof Prototype

MeshMould

KnitCandela

Data Driven Acoustic Design

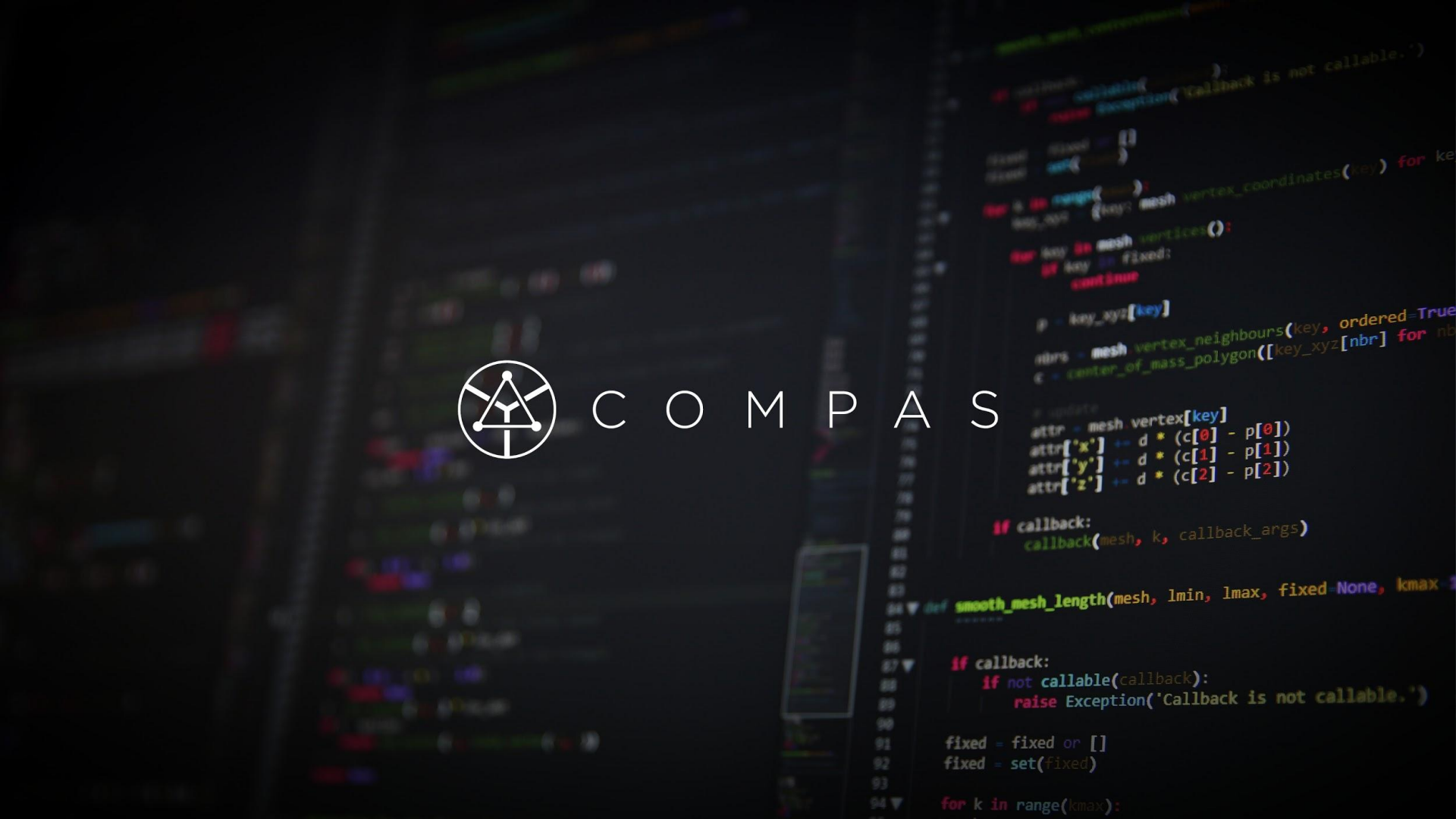
Concrete Choreography

HiLo Roof

Adaptive Detailing

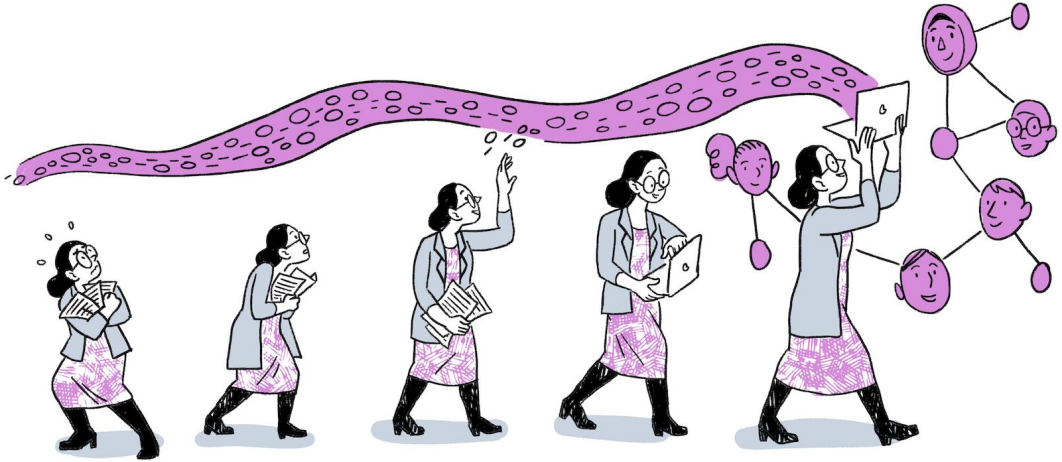
WEF Floor

COMPAS


if callback:
if not callable(callback):
raise Exception('Callback is not callable.')
fixed = fixed or []
fixed = set(fixed)
for k in range(kmax):
smooth_mesh_length(mesh, lmin, lmax, fixed=None, kmax
if callback:
callback(mesh, k, callback_args)
attr = mesh.vertex[key]
attr['x'] += d * (c[0] - p[0])
attr['y'] += d * (c[1] - p[1])
attr['z'] += d * (c[2] - p[2])
continue
p = key_xyz[key]
c = center_of_mass_polygon([key_xyz[nbr]

Reproducible Research is work that **can be independently recreated** from the **same data** and the **same code** that the original team used.

The Turing Way, https://github.com/alan-turing-institute/the-turing-way



EVOLVING TOWARDS AN ERA OF OPEN RESEARCH

Scriberia

The Turing Way project illustration by Scriberia. Original version on Zenodo. http://doi.org/10.5281/zenodo.3695300.

ETH zürich

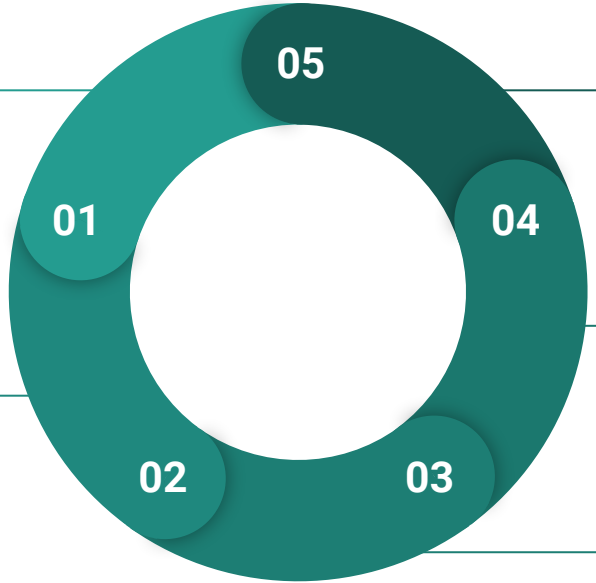
Elements of open research

Open Data

Documenting and sharing research data openly for re-use.

Open Source Software

Documenting research code and routines, and making them freely accessible and available.



Open Notebooks

An emerging practice, documenting and sharing the experimental process of trial and error.

Open Access

Making all published outputs freely accessible for maximum use and impact.

Open Hardware

Documenting designs, materials, and other relevant information related to hardware, and making them freely accessible and available.

The Turing Way project illustration by Scriberia. Original version on Zenodo. http://doi.org/10.5281/zenodo.3695300

getting started

Installation

https://tinyurl.com/swinburne21

gramaziokohler / workshop_swinburne_2021

Unwatch 3 | Star 0 | Fork 0

Code | Issues | Pull requests | Actions | Projects | Wiki | Security | Insights | Settings

main | 1 branch | 0 tags | Go to file | Add file | Code

gonzalocasas Add code examples — b704a09 — minutes ago — 4 commits

.github/workflows	Initial commit	11 hours ago
docker	Add docker examples	41 minutes ago
examples	Add code examples	19 minutes ago
images	Initial commit	11 hours ago
.editorconfig	Initial commit	11 hours ago
.gitignore	Initial commit	11 hours ago
LICENSE	Initial commit	11 hours ago
README.md	Update install details	11 hours ago
environment.yml	Initial commit	11 hours ago
setup.cfg	Initial commit	11 hours ago

README.md

About

No description, website, or topics provided.

Readme

MIT License

Releases

No releases published

Create a new release

Packages

No packages published

Publish your first package

⌘ + SPACE

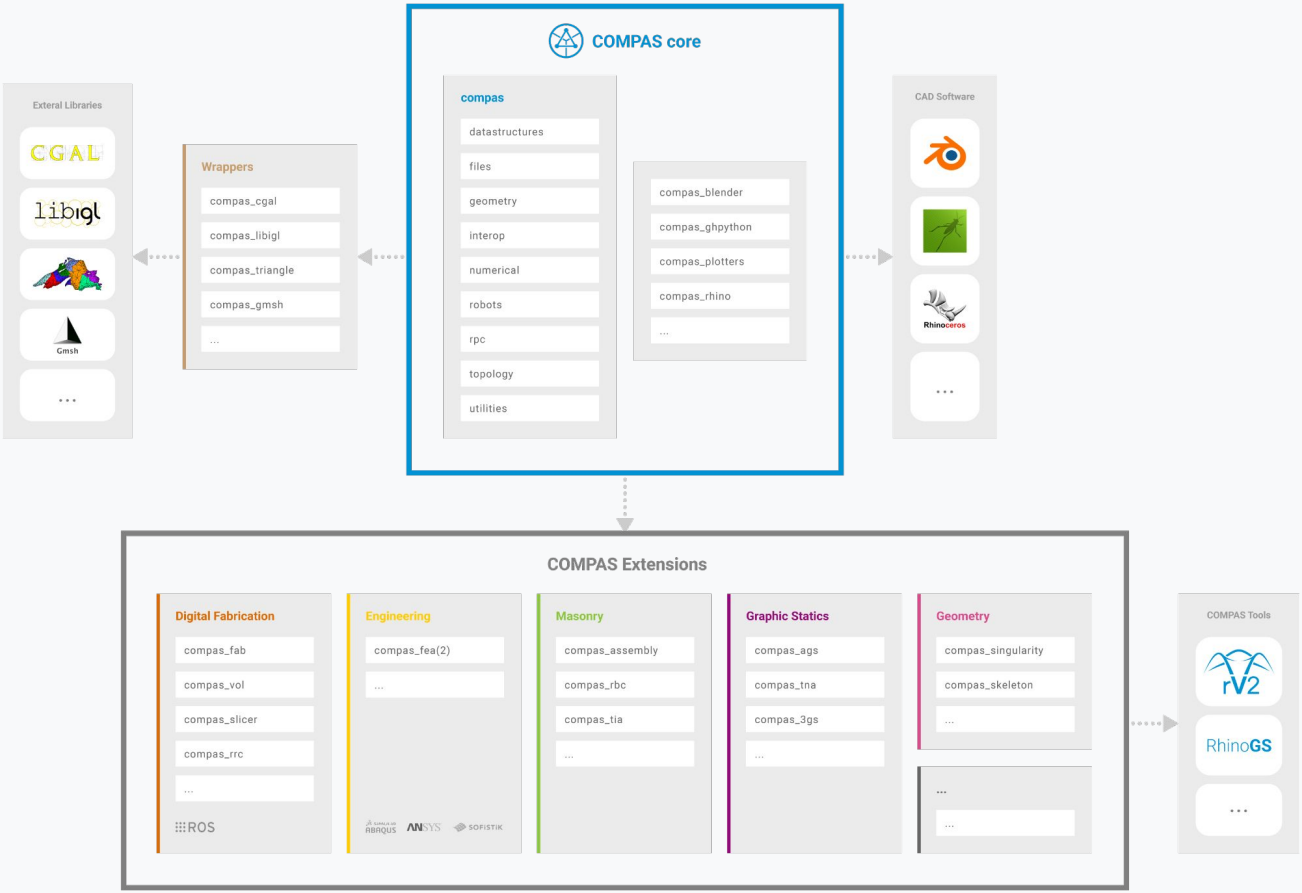
Terminal

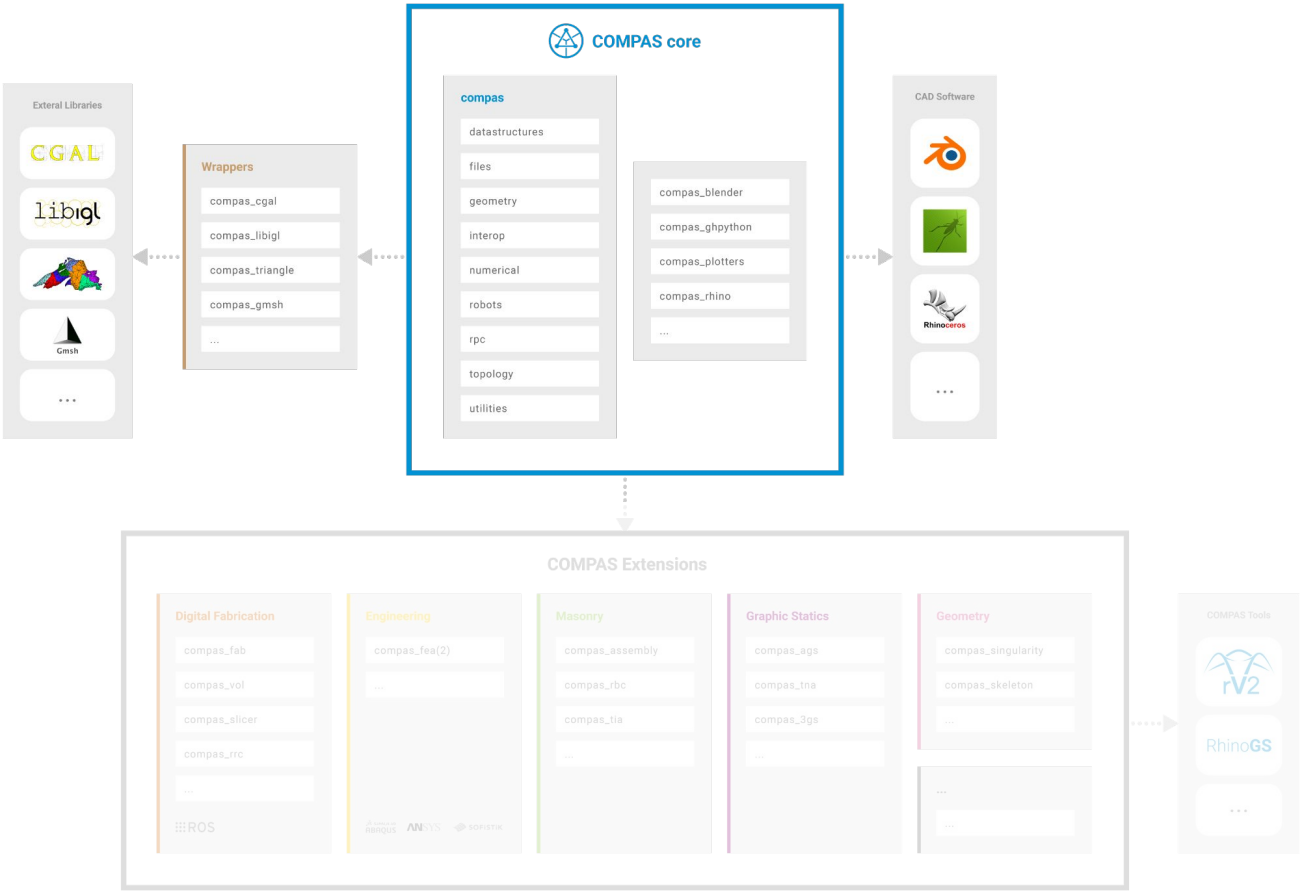
Anaconda Prompt

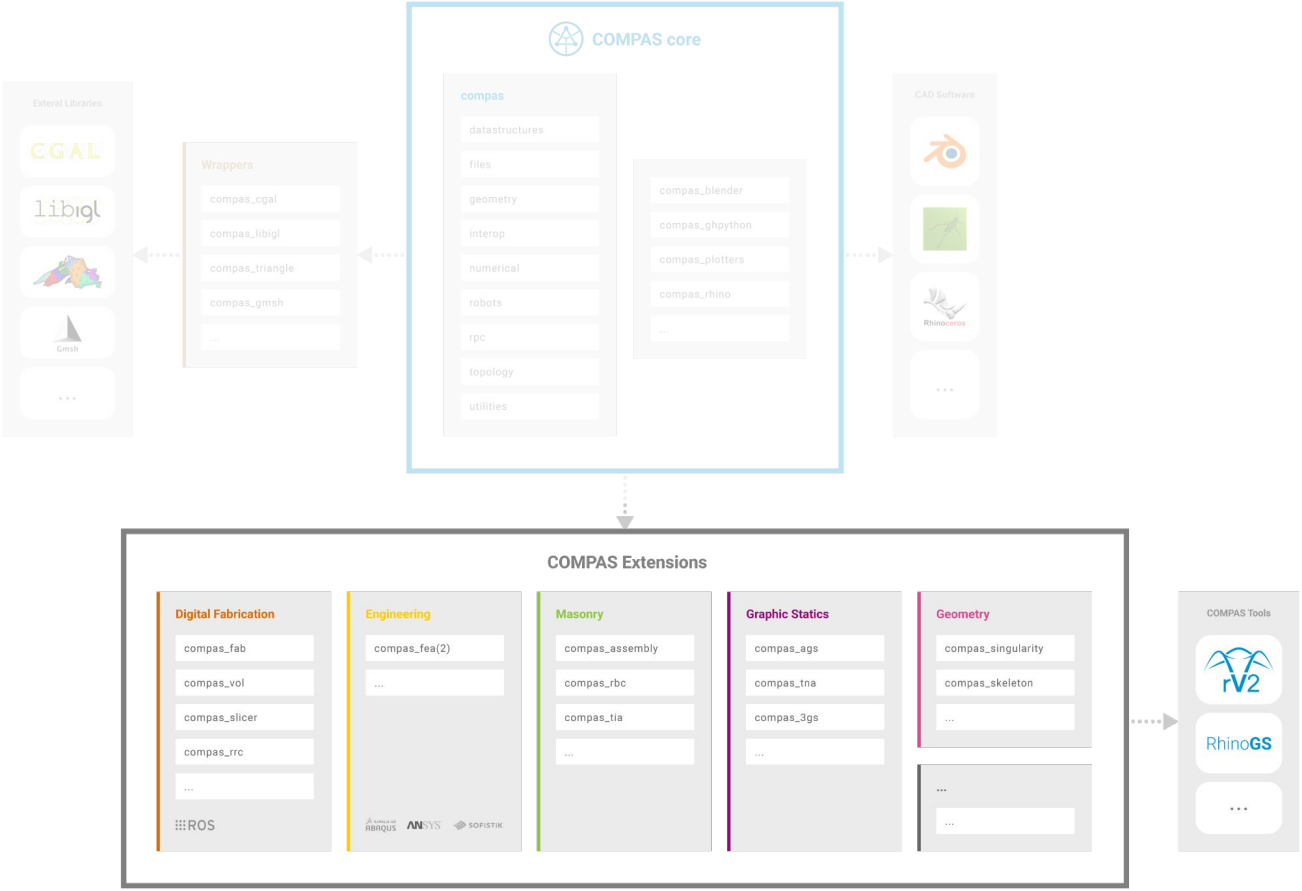
```
(base)  conda config --add channels conda-forge
(base)  cd path/to/workshop_swinburne_2021
(base)  conda env create -f environment.yml
(base)  conda activate swinburne
(swinburne) conda install python.app
```

Create environment

```
(base)  conda config --add channels conda-forge
(base)  cd path/to/workshop_swinburne_2021
(base)  conda env create -f environment.yml
(base)  conda activate swinburne
(swinburne) conda install python.app
```


COMPAS core
compas
datastructures
files
geometry
interop
numerical
robots
rpc
topology
utilities
compas_blender
compas_ghpython
compas_plotters
compas_rhino
...
Exteral Libraries
CGAL
libigl
Gmsh
...
Wrappers
compas_cgal
compas_libigl
compas_triangle
compas_gmsh
...
CAD Software
Rhinoceros
...
COMPAS Extensions
Digital Fabrication
compas_fab
compas_vol
compas_slicer
compas_rrc
...
ROS
Engineering
compas_fea(2)
...
ABAQUS
ANSYS
SOFISTIK
Masonry
compas_assembly
compas_rbc
compas_tia
...
Graphic Statics
compas_ags
compas_tna
compas_3gs
...
Geometry
compas_singularity
compas_skeleton
...
...
...
COMPAS Tools
rV2
RhinoGS
...


COMPAS core
Exteral Libraries
CGAL
libigl
Gmsh
...
Wrappers
compas_cgal
compas_libigl
compas_triangle
compas_gmsh
...
compas
datastructures
files
geometry
interop
numerical
robots
rpc
topology
utilities
compas_blender
compas_ghpython
compas_plotters
compas_rhino
...
CAD Software
Rhinoceros
...
COMPAS Extensions
Digital Fabrication
compas_fab
compas_vol
compas_slicer
compas_rrc
...
ROS
Engineering
compas_fea(2)
...
ABAQUS
ANSYS
SOFISTIK
Masonry
compas_assembly
compas_rbc
compas_tia
...
Graphic Statics
compas_ags
compas_tna
compas_3gs
...
Geometry
compas_singularity
compas_skeleton
...
...
...
COMPAS Tools
rV2
RhinoGS
...


External Libraries
CGAL
libigl
Gmsh
...
Wrappers
compas_cgal
compas_libigl
compas_triangle
compas_gmsh
...
COMPAS core
compas
datastructures
files
geometry
interop
numerical
robots
rpc
topology
utilities
compas_blender
compas_ghpython
compas_plotters
compas_rhino
...
CAD Software
Rhinoceros
...
COMPAS Extensions
Digital Fabrication
compas_fab
compas_vol
compas_slicer
compas_rrc
...
ROS
Engineering
compas_fea(2)
...
ABAQUS
ANSYS
SOFISTIK
Masonry
compas_assembly
compas_rbc
compas_tia
...
Graphic Statics
compas_ags
compas_tna
compas_3gs
...
Geometry
compas_singularity
compas_skeleton
...
...
...
COMPAS Tools
rV2
RhinoGS
...

COMPAS core

- compas.data
- compas.datastructures
- compas.files
- compas.geometry
- compas.numerical
- compas.plugins
- compas.robots
- compas.rpc
- compas.scene
- compas.topology
- compas.utilities
- compas_plotters
- compas_rhino
- compas_ghpython
- compas_blender

COMPAS core extensions

- compas_cgal
- compas_cloud
- compas_dashboard
- compas_gmsh
- compas_libigl
- compas_occ
- compas_triangle
- compas_view2

Data modelling & Assessment

- compas_ifc
- compas_lca

Digital Fabrication

- compas_fab
- compas_slicer
- compas_rrc
- compas_vol
- compas_timber

Engineering

- compas_fea(2)

Command Prompt - python

```
(swinburne) C:\Users\gcasas>python
Python 3.8.10 | packaged by conda-forge | (default, May 11 2021, 06:25:23) [MSC v.1916 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> import compas
>>> from compas.datastructures import Mesh
>>> mesh = Mesh.from_obj(compas.get('faces.obj'))
>>> print(mesh.summary())
Mesh summary
============
- vertices: 36
- edges: 60
- faces: 25
>>> 
```

dr_numpy.py — compas-dev

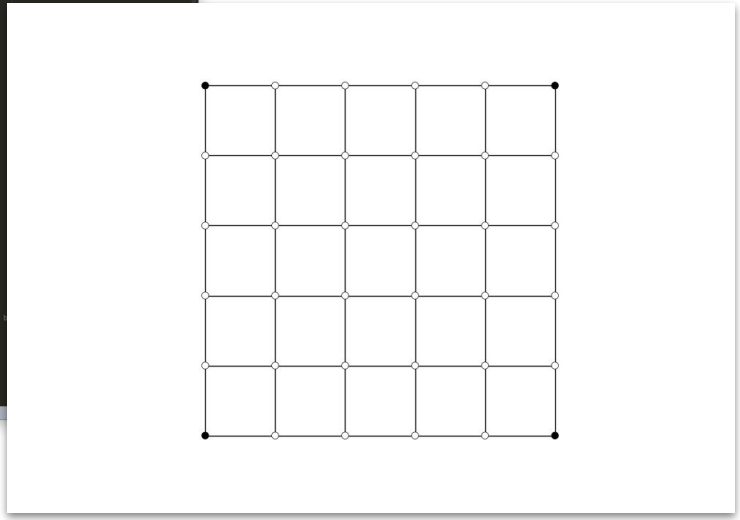
FOLDERS

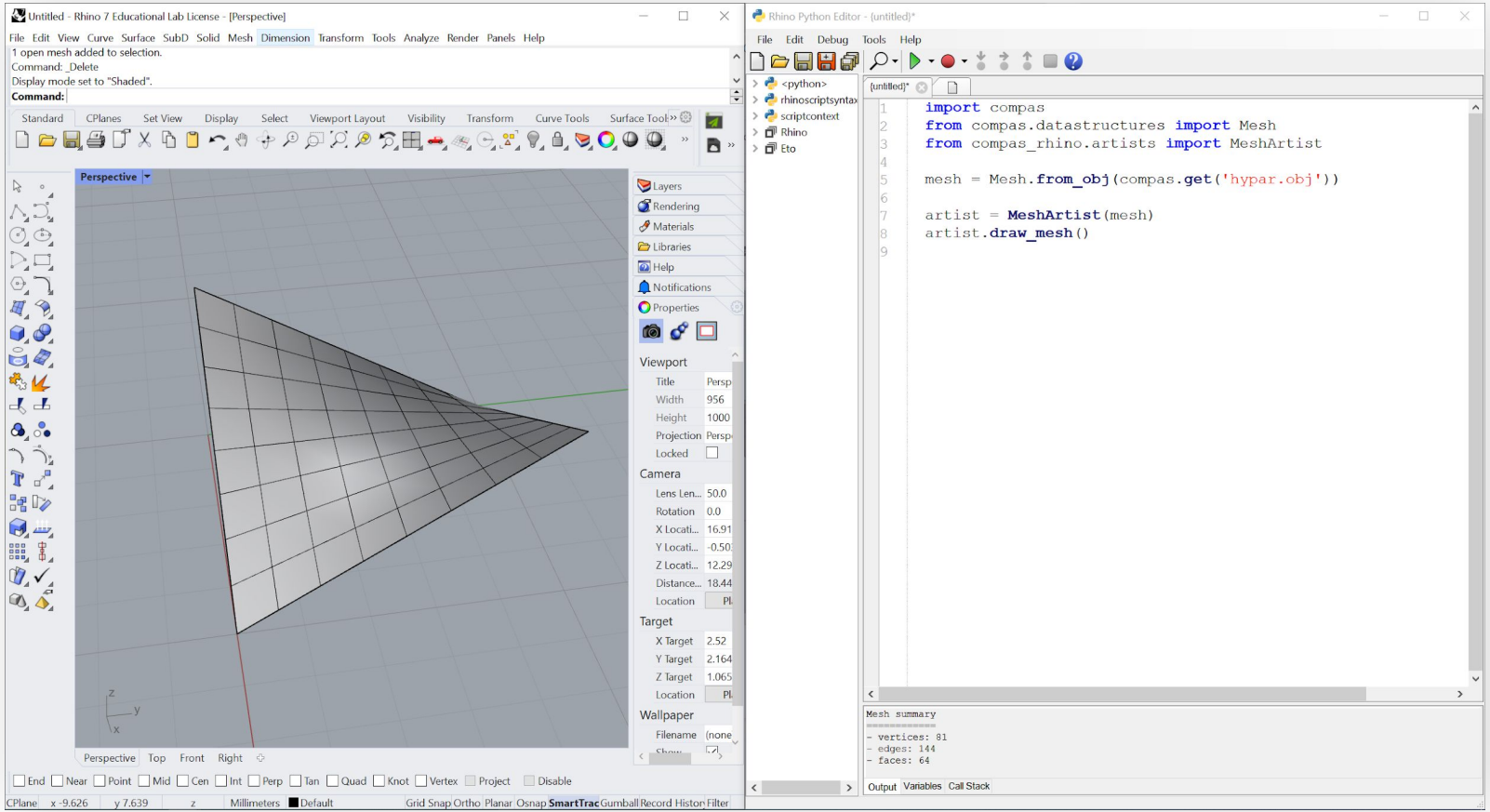
- compas-dev
  - compas
    - build
    - data
    - dist
    - docs
    - libs
    - src
      - compas
        - __pycache__
        - com
        - datastructures
        - files
        - geometry
        - interop
        - numerical
          - __pycache__
          - _alglib
          - alglib
          - algorithms
          - descent
          - devo
          - dr
          - drx
            - __pycache__
            - __init__.py
            - drx_numpy.py
          - fd
          - ga
          - lma
          - mma
          - pca
          - solvers
          - topop
          - __init__.py
          - linalg.py
          - matrices.py
          - operators.py
          - utilities.py
        - plotters
        - robots
        - rpc
        - topology
        - utilities
        - viewers
        - __init__.py
        - _os.py
    - COMPAS.egg-info
    - compas_blender
    - compas_ghpython
    - compas_hpc
    - compas_revit
    - compas_rhino
    - temp
    - tests
    - .bumpversion.cfg
    - .editorconfig
    - .gitignore
    - .travis.yml
    - CONTRIBUTORS.md
    - LICENSE

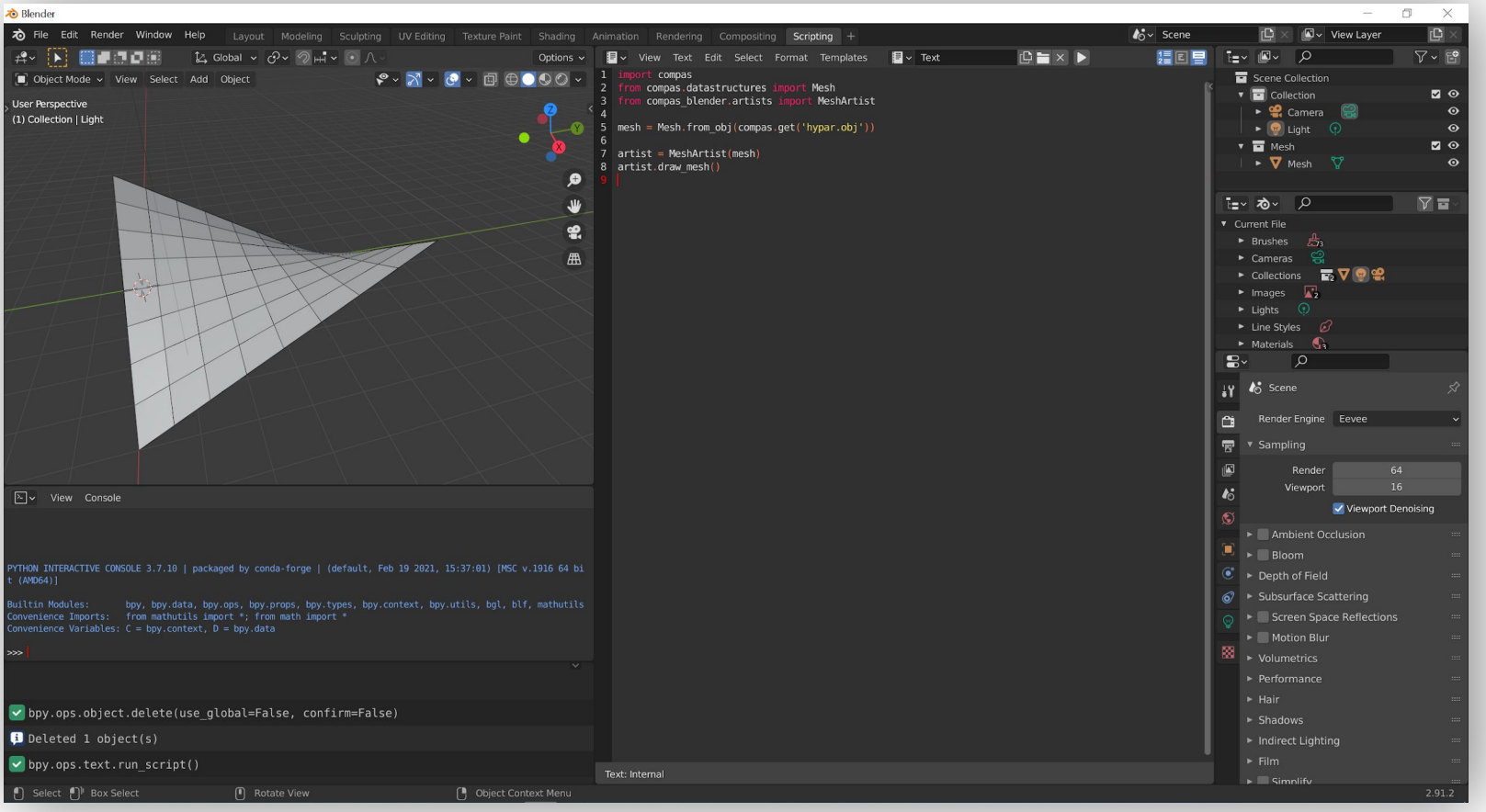
dr_numpy.py

```python
from __future__ import absolute_import
from __future__ import division
from __future__ import print_function

import compas

try:
    from numpy import array
    from numpy import isnan
    from numpy import isinf
    from numpy import ones
    from numpy import zeros
    from scipy.linalg import norm
    from scipy.sparse import diags

except ImportError:
    compas.raise_if_not_ironpython()

from compas.numerical import connectivity_matrix
from compas.numerical import normrow


__all__ = ['dr_numpy']


K = [
    [0.0],
    [0.5, 0.5],
    [0.5, 0.0, 0.5],
    [1.0, 0.0, 0.0, 1.0],
]


class Coeff():
    def __init__(self, c):
        self.c = c
        self.a = (1 - c * 0.5) / (1 + c * 0.5)
        self.b = 0.5 * (1 + self.a)


def dr_numpy(vertices, edges, fixed, loads, qpre, fpre, lpre, linit, E, radius,
             callback=None, callback_args=None, **kwargs):
    """Implementation of the dynamic relaxation method for form findong and analysis
    of articulated networks of axial-force members.

    Parameters
    ----------
    vertices : list
        XYZ coordinates of the vertices.
    edges : list
        Connectivity of the vertices.
    fixed : list
        Indices of the fixed vertices.
    loads : list
        XYZ components of the loads on the vertices.
    qpre : list
        Prescribed force densities in the edges.
    fpre : list
        Prescribed forces in the edges.
    lpre : list
        Prescribed lengths of the edges.
    linit : list
        Initial length of the edges.
    E : list
        Stiffness of the edges.
    radius : list
        Radius of the edges.
    callback : callable, optional
        User-defined function that is called at every iteration.
    callback_args : tuple, optional
        Additional arguments passed to the callback.

    Notes
    -----
    For more info, see [1]_.

    References
    ----------
    .. [1] De Laet L., Veenendaal D., Van Mele T., Mollaert M. and Block P.,
           «Bending incorporated: designing tension structures by integrating b
           Proceedings of Tensinet Symposium 2013,Istanbul, Turkey, 2013.

    Examples
    --------
    .. plot::
        :include-source:

        import compas
        from compas.datastructures import Network
```

Git branch: master, index: ✓, working: 10✱, Line 33, Column 1




Untitled - Rhino 7 Educational Lab License - [Perspective]
File Edit View Curve Surface SubD Solid Mesh Dimension Transform Tools Analyze Render Panels Help
1 open mesh added to selection.
Command: _Delete
Display mode set to "Shaded".
Command:
Standard CPlanes Set View Display Select Viewport Layout Visibility Transform Curve Tools Surface Tools
Perspective
Layers
Rendering
Materials
Libraries
Help
Notifications
Properties
Viewport
Title Persp
Width 956
Height 1000
Projection Persp
Locked
Camera
Lens Len... 50.0
Rotation 0.0
X Locati... 16.91
Y Locati... -0.50
Z Locati... 12.29
Distance... 18.44
Location
Target
X Target 2.52
Y Target 2.164
Z Target 1.065
Location
Wallpaper
Filename (none
Perspective Top Front Right
End Near Point Mid Cen Int Perp Tan Quad Knot Vertex Project Disable
CPlane x -9.626 y 7.639 z Millimeters Default Grid Snap Ortho Planar Osnap SmartTrack Gumball Record Histor Filter
Rhino Python Editor - (untitled)*
File Edit Debug Tools Help
<python>
rhinoscriptsyntax
scriptcontext
Rhino
Eto
(untitled)*
import compas
from compas.datastructures import Mesh
from compas_rhino.artists import MeshArtist
mesh = Mesh.from_obj(compas.get('hypar.obj'))
artist = MeshArtist(mesh)
artist.draw_mesh()
Mesh summary
- vertices: 81
- edges: 144
- faces: 64
Output Variables Call Stack

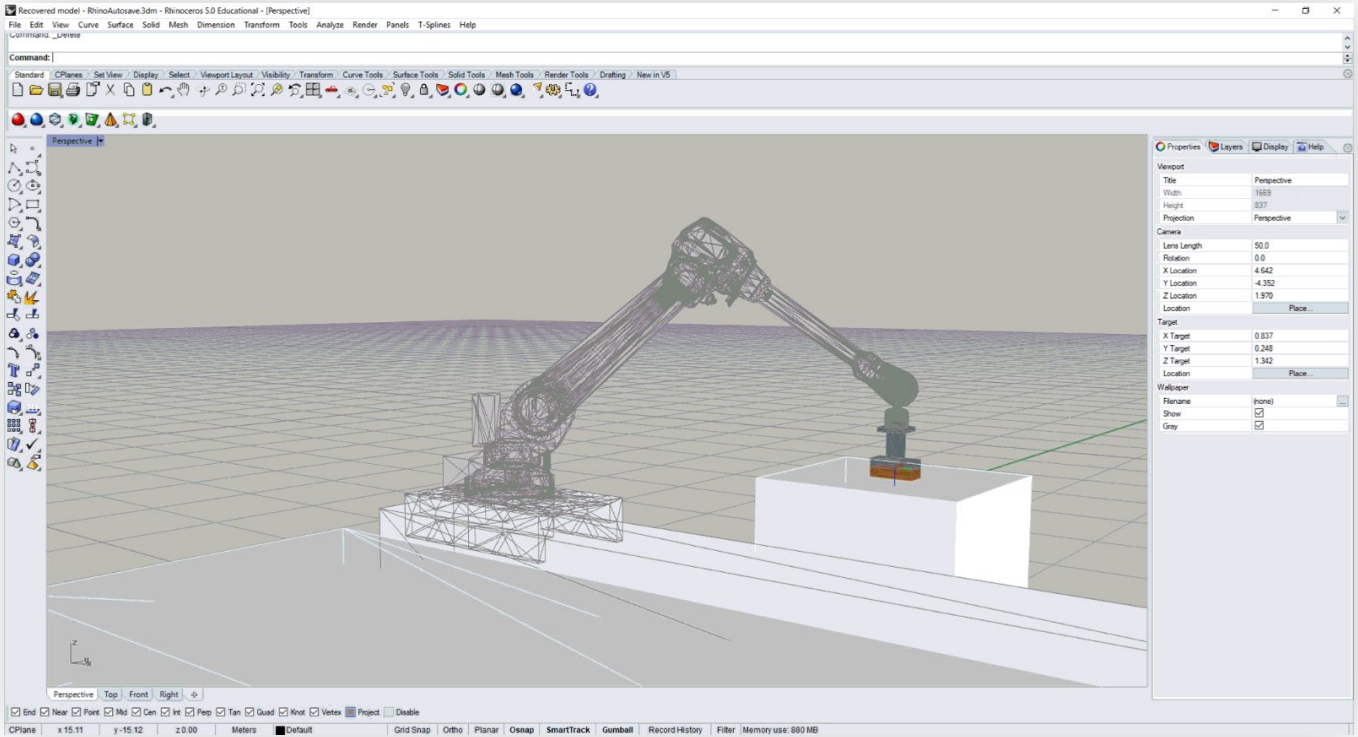

Blender
File Edit Render Window Help
Layout Modeling Sculpting UV Editing Texture Paint Shading Animation Rendering Compositing Scripting
Scene
View Layer
Global
Options
Object Mode View Select Add Object
User Perspective
(1) Collection | Light
View Console
PYTHON INTERACTIVE CONSOLE 3.7.10 | packaged by conda-forge | (default, Feb 19 2021, 15:37:01) [MSC v.1916 64 bit (AMD64)]
Builtin Modules: bpy, bpy.data, bpy.ops, bpy.props, bpy.types, bpy.context, bpy.utils, bgl, blf, mathutils
Convenience Imports: from mathutils import *; from math import *
Convenience Variables: C = bpy.context, D = bpy.data
>>>
bpy.ops.object.delete(use_global=False, confirm=False)
Deleted 1 object(s)
bpy.ops.text.run_script()
View Text Edit Select Format Templates
Text
import compas
from compas.datastructures import Mesh
from compas_blender.artists import MeshArtist
mesh = Mesh.from_obj(compas.get('hypar.obj'))
artist = MeshArtist(mesh)
artist.draw_mesh()
Text: Internal
Scene Collection
Collection
Camera
Light
Mesh
Mesh
Current File
Brushes
Cameras
Collections
Images
Lights
Line Styles
Materials
Scene
Render Engine Eevee
Sampling
Render 64
Viewport 16
Viewport Denoising
Ambient Occlusion
Bloom
Depth of Field
Subsurface Scattering
Screen Space Reflections
Motion Blur
Volumetrics
Performance
Hair
Shadows
Indirect Lighting
Film
Simplify
Select Box Select Rotate View Object Context Menu
2.91.2

```python
# Rhino
import compas
from compas.datastructures import Mesh
from compas_rhino.artists import MeshArtist

mesh = Mesh.from_obj(compas.get('hypar.obj'))

artist = MeshArtist(mesh)
artist.draw_mesh()
```

```python
# Blender
import compas
from compas.datastructures import Mesh
from compas_blender.artists import MeshArtist

mesh = Mesh.from_obj(compas.get('hypar.obj'))

artist = MeshArtist(mesh)
artist.draw_mesh()
```

```python
# Rhino
import compas
from compas.datastructures import Mesh
from compas_rhino.artists import MeshArtist

mesh = Mesh.from_obj(compas.get('hypar.obj'))

artist = MeshArtist(mesh)
artist.draw_mesh()
```

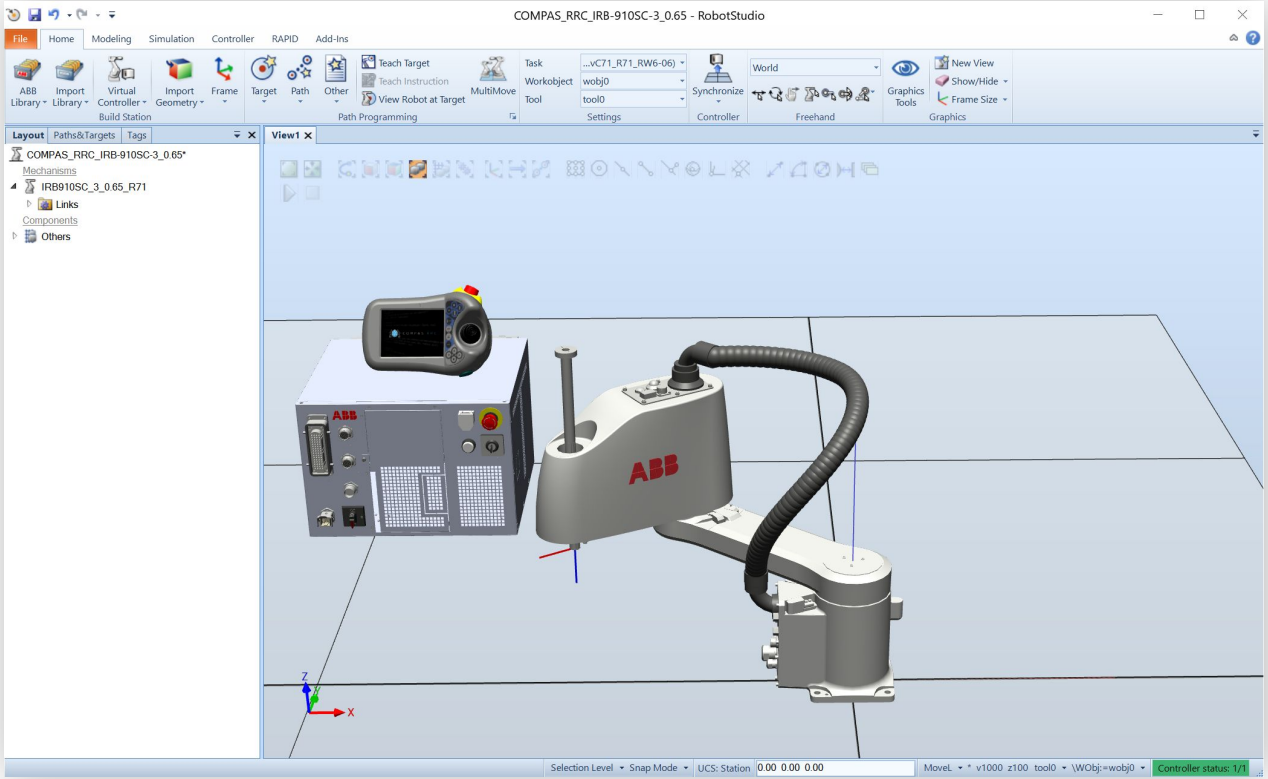
```python
# Blender
import compas
from compas.datastructures import Mesh
from compas_blender.artists import MeshArtist

mesh = Mesh.from_obj(compas.get('hypar.obj'))

artist = MeshArtist(mesh)
artist.draw_mesh()
```



COMPAS FAB

Robotic fabrication package

ETH zürich



COMPAS RRC

Robot control for ABB robots

Installation

https://tinyurl.com/swinburne21

Activate

```
(base)  conda config --add channels conda-forge
(base)  cd path/to/workshop_swinburne_2021
(base)  conda env create -f environment.yml
(base)  conda activate swinburne
(swinburne) conda install python.app
```

Only on Mac

```
(base)  conda config --add channels conda-forge
(base)  cd path/to/workshop_swinburne_2021
(base)  conda env create -f environment.yml
(base)  conda activate swinburne
(swinburne) conda install python.app
```

Restart Rhino

```
(swinburne) python -m compas_rhino.install
(swinburne) python -m compas_rhino.install -v 7.0
```

primitives and datastructures

Primitives & Shapes

compas.geometry

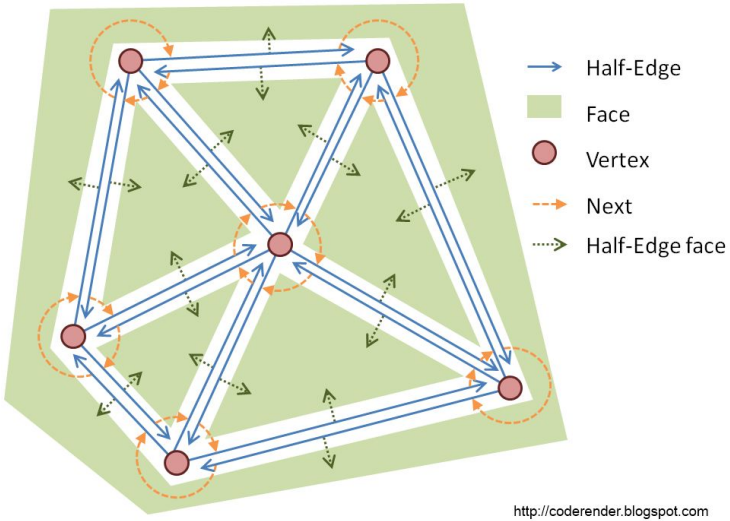
- point, vector, line, plane, circle, polyline, polygon, frame
- transformation, translation, rotation, scale, reflection, projection, shear
- box, sphere, polyhedron, torus, cone, capsule

Object	Python	COMPAS
point	`point = [0, 0, 0]`	`point = Point(0, 0, 0)`
vector	`vector = [0, 0, 1]`	`vector = Vector(0, 0, 1)`
line	`line = [0, 0, 0], [1, 0, 0]`	`line = Line(point, point)`
plane	`plane = [0, 0, 0], [0, 0, 1]`	`plane = Plane(point, vector)`
circle	`circle = ([0, 0, 0], [0, 0, 1]), 1.0`	`circle = Circle(plane, radius)`
polyline	`polyline = [0, 0, 0], [1, 0, 0], [1, 1, 0], [0, 0, 0]`	`polyline = Polyline(points)`
polygon	`polygon = [0, 0, 0], [1, 0, 0], [1, 1, 0]`	`polygon = Polygon(points)`
frame	`frame = [0, 0, 0], [1, 0, 0], [0, 1, 0]`	`frame = Frame(point, xaxis, xyp`

Mesh

compas.datastructures

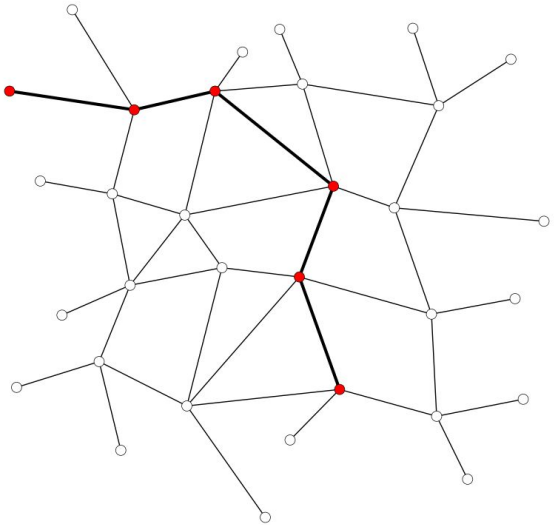
- **CAD-agnostic**
- half-edge data structure
- support for n-sided polygonal faces
- open or closed, polygonal surfaces
- custom attributes at mesh, vertex, edge and face



Network

compas.datastructures

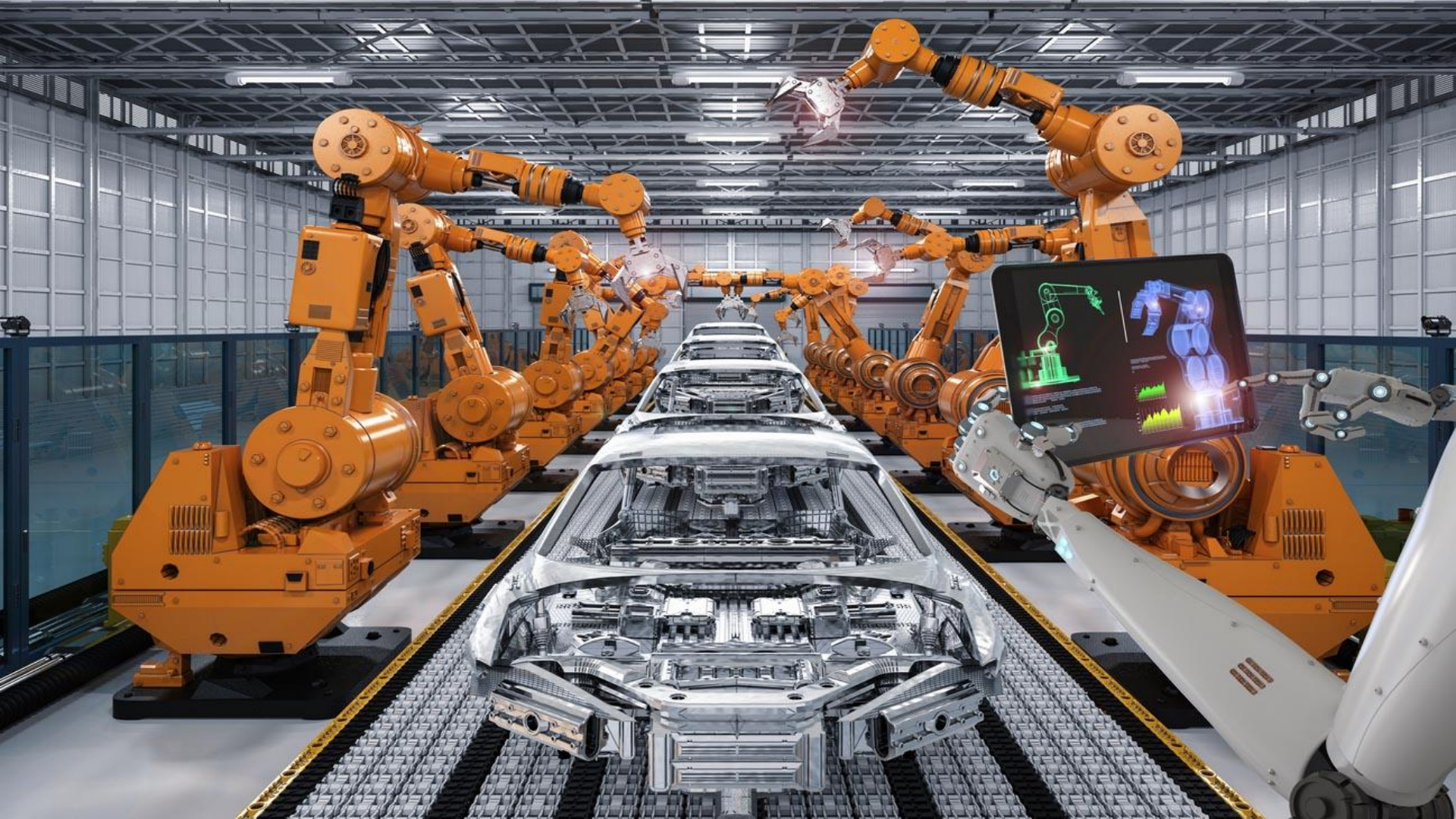
- directed edge graph data structure
- graph: topological
- network: geometric implementation of graph
- custom attributes at network, node and edge
- networkx support



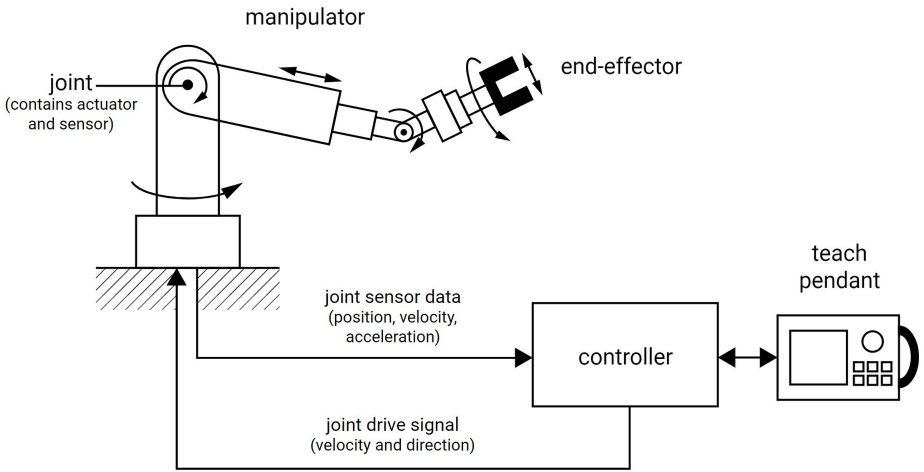
Remote Procedure Calls

compas.rpc

industrial robots

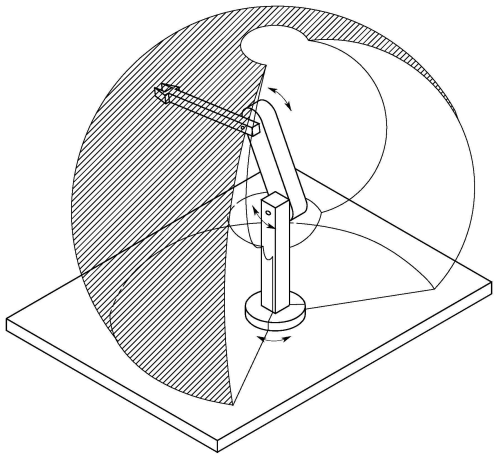


Basic robot components

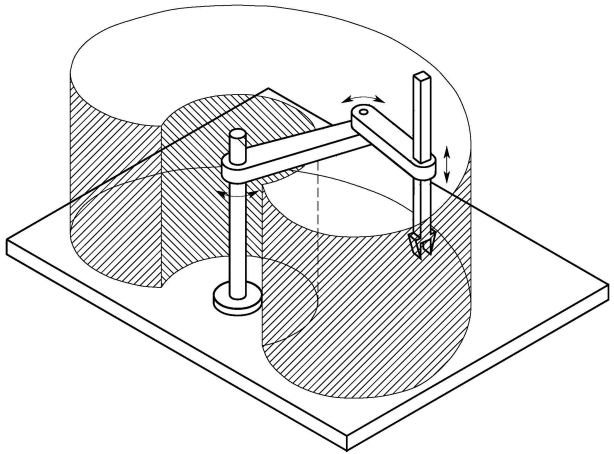


- manipulator
- end-effector
- actuators
- sensors
- controller
- teach pendant

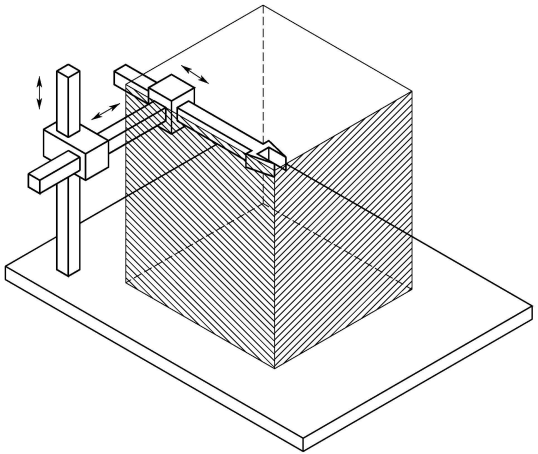
Robot workspace



Anthropomorphic manipulator

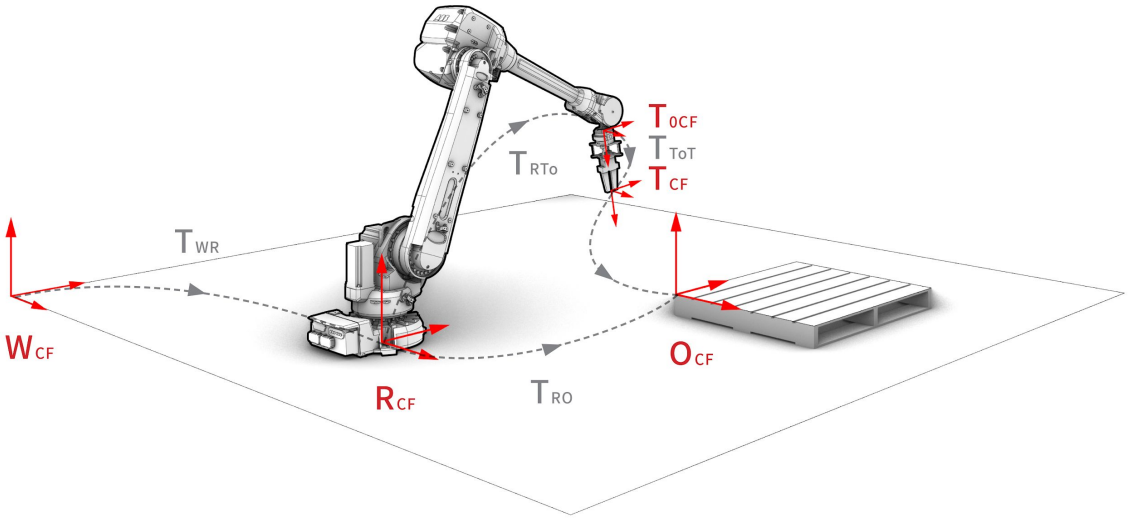


SCARA manipulator



Cartesian manipulator

Robot coordinate frames



World (WCF)

Robot (RCF)

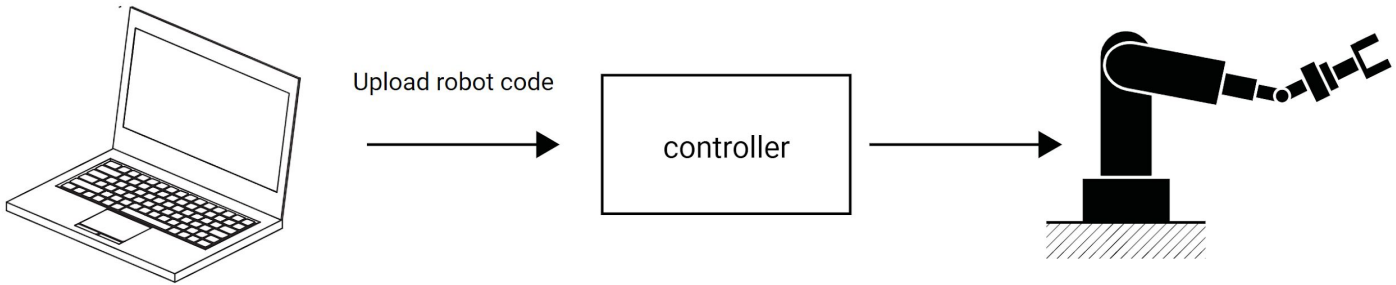
Tool0 (T0CF)

Tool (TCF)

Object (OCF)

Robot control

- Offline control
- Online control
  a. real-time
  b. non-real-time

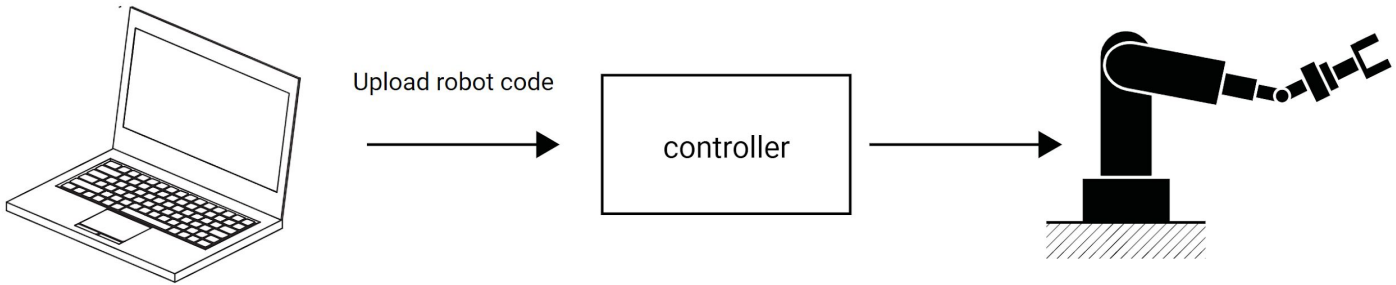


Robot control

- Offline control
- Online control
  a. real-time
  b. non-real-time

Vendor-specific language and tools:

Vendor	Language	Software
ABB	RAPID	ABB RobotStudio
Staubli	VAL3	Staubli Robotics Suite
UR	URScript	URSim
KUKA	KRL	KUKA WorkVisual

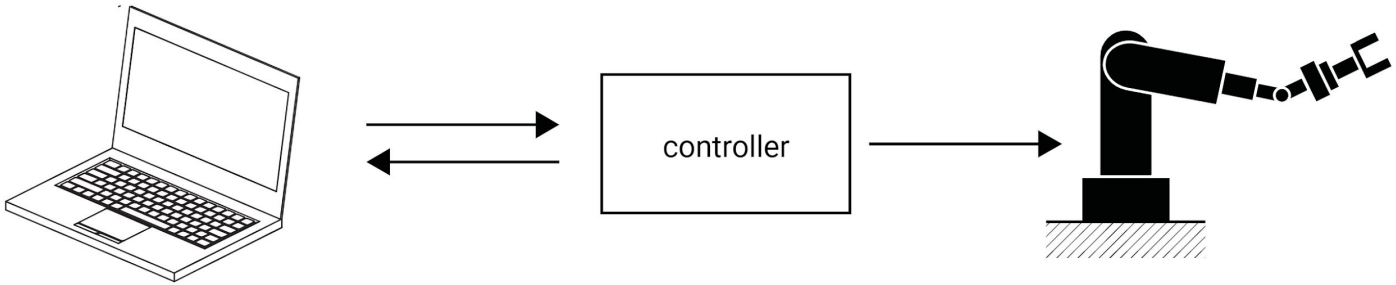


Robot control

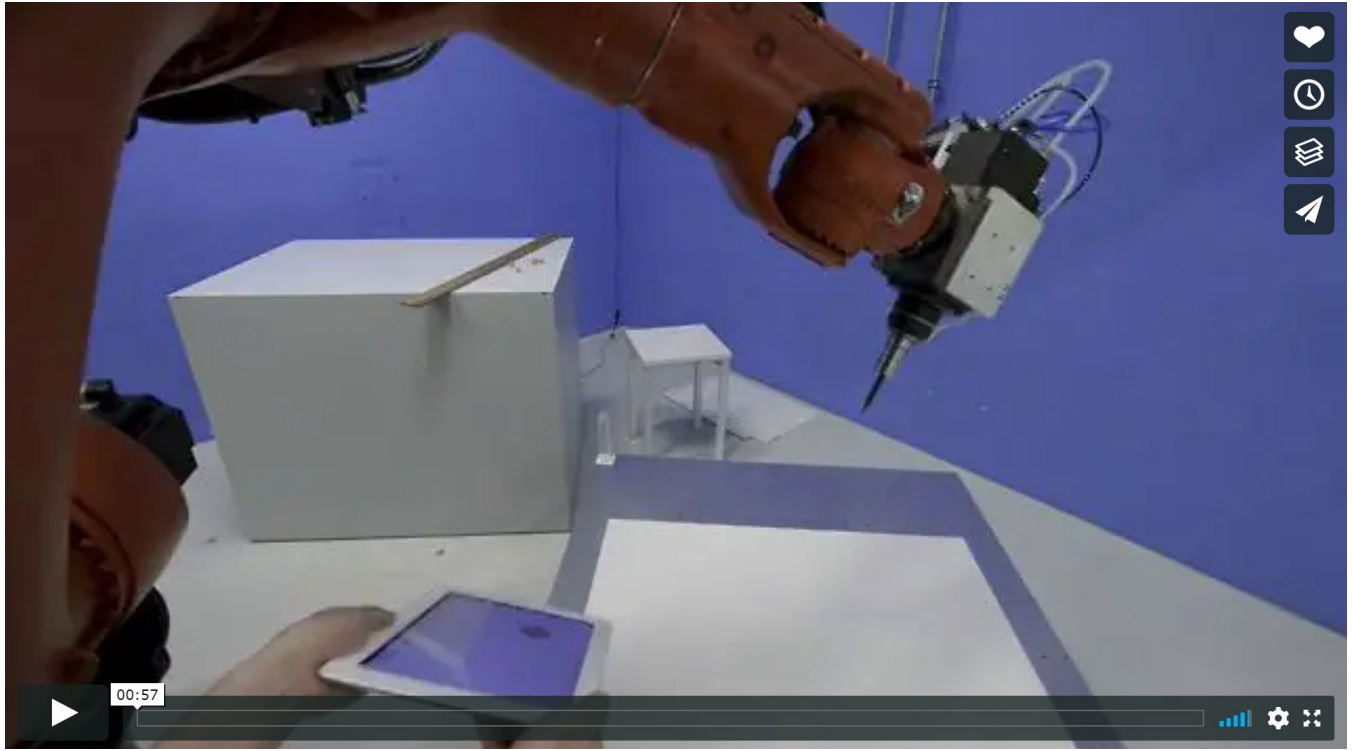
- Offline control
- Online control
  a. real-time
  b. non-real-time

Robot control

- Offline control
- Online control
  a. real-time
  b. non-real-time



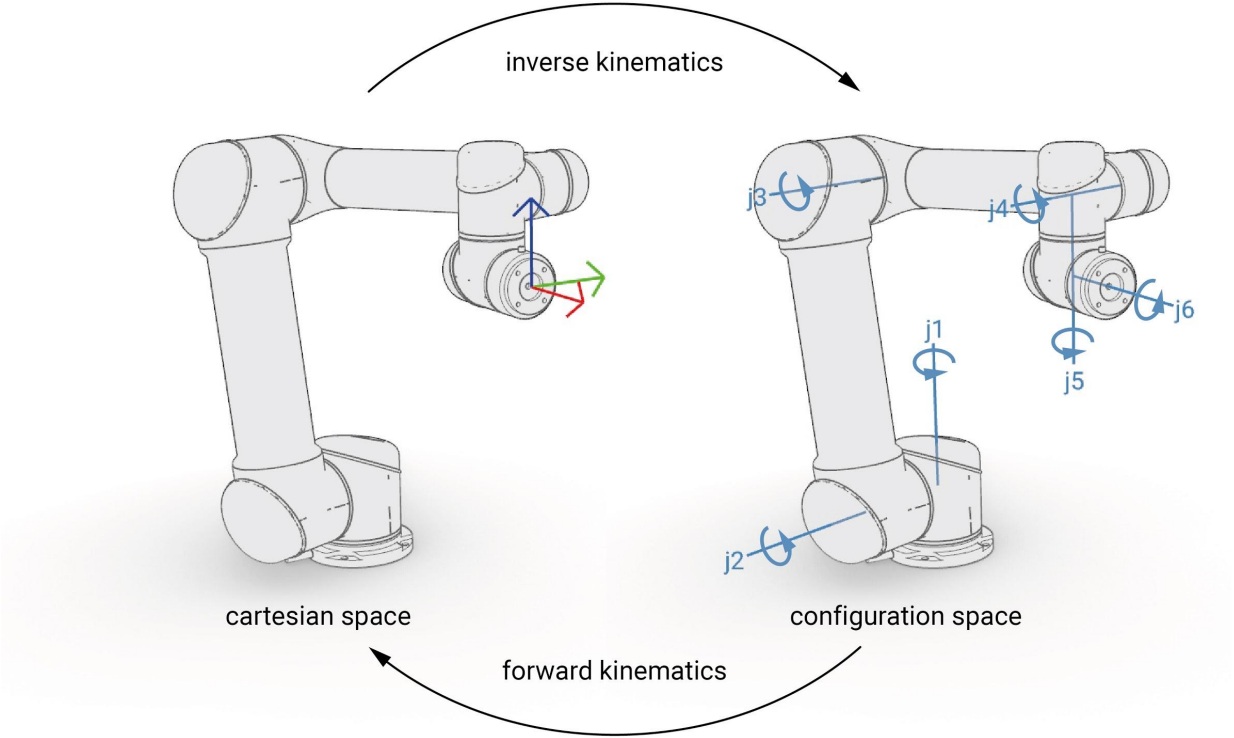
Example: online non-real-time control



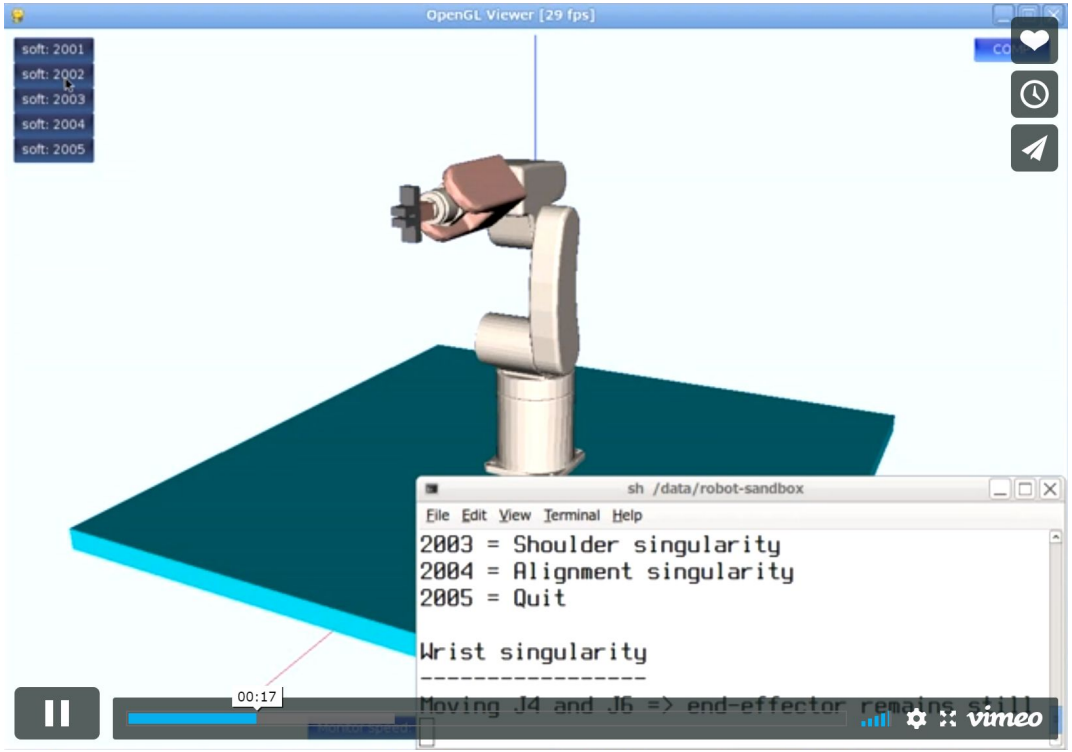
Robot control

Control robots directly from within parametric design environment (Grasshopper, Dynamo, ...)

Robot positioning

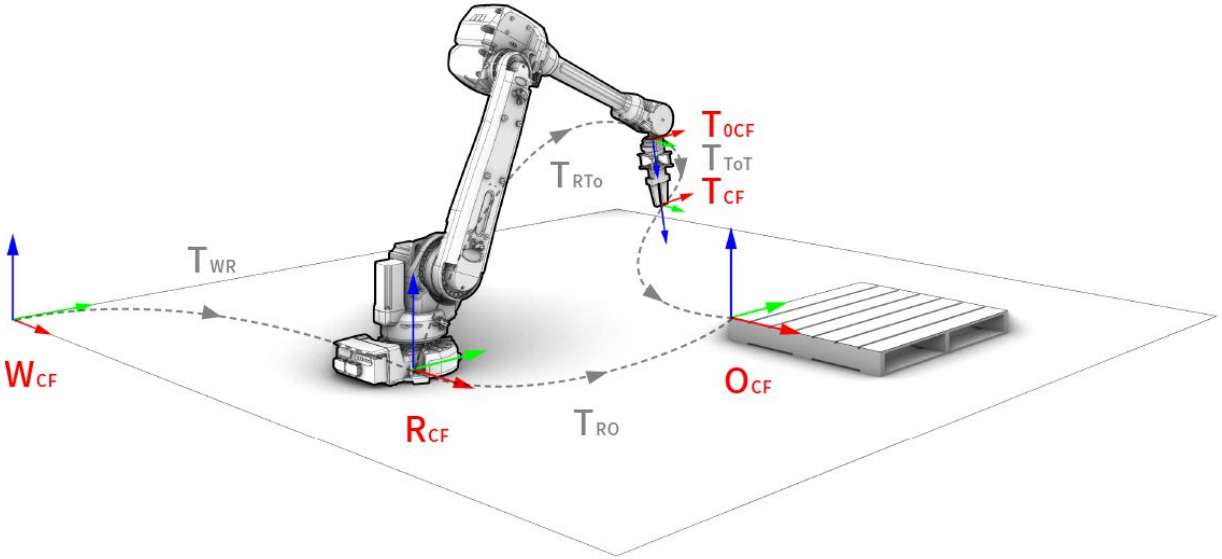


Robot singularities



robotic fundamentals

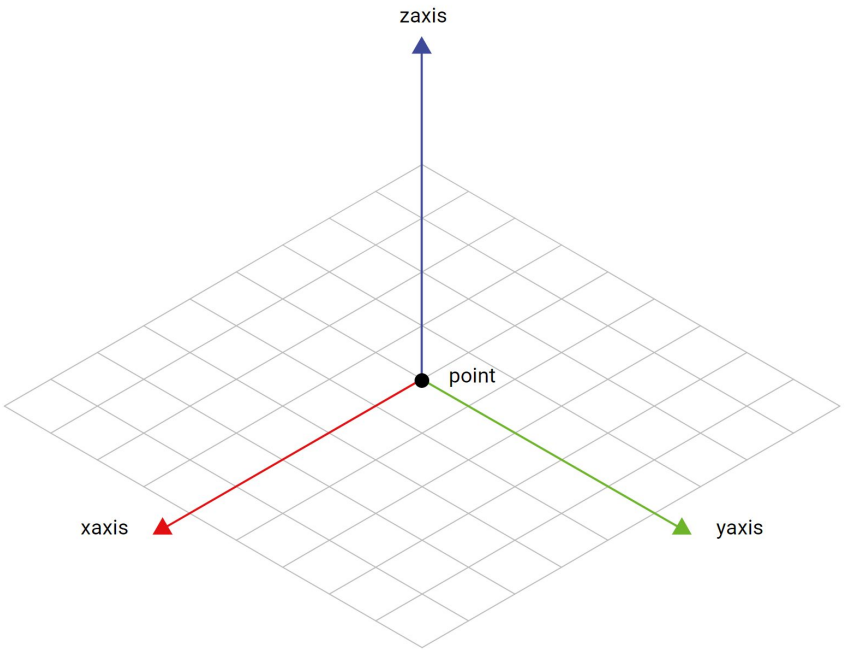
Frame and Transformation



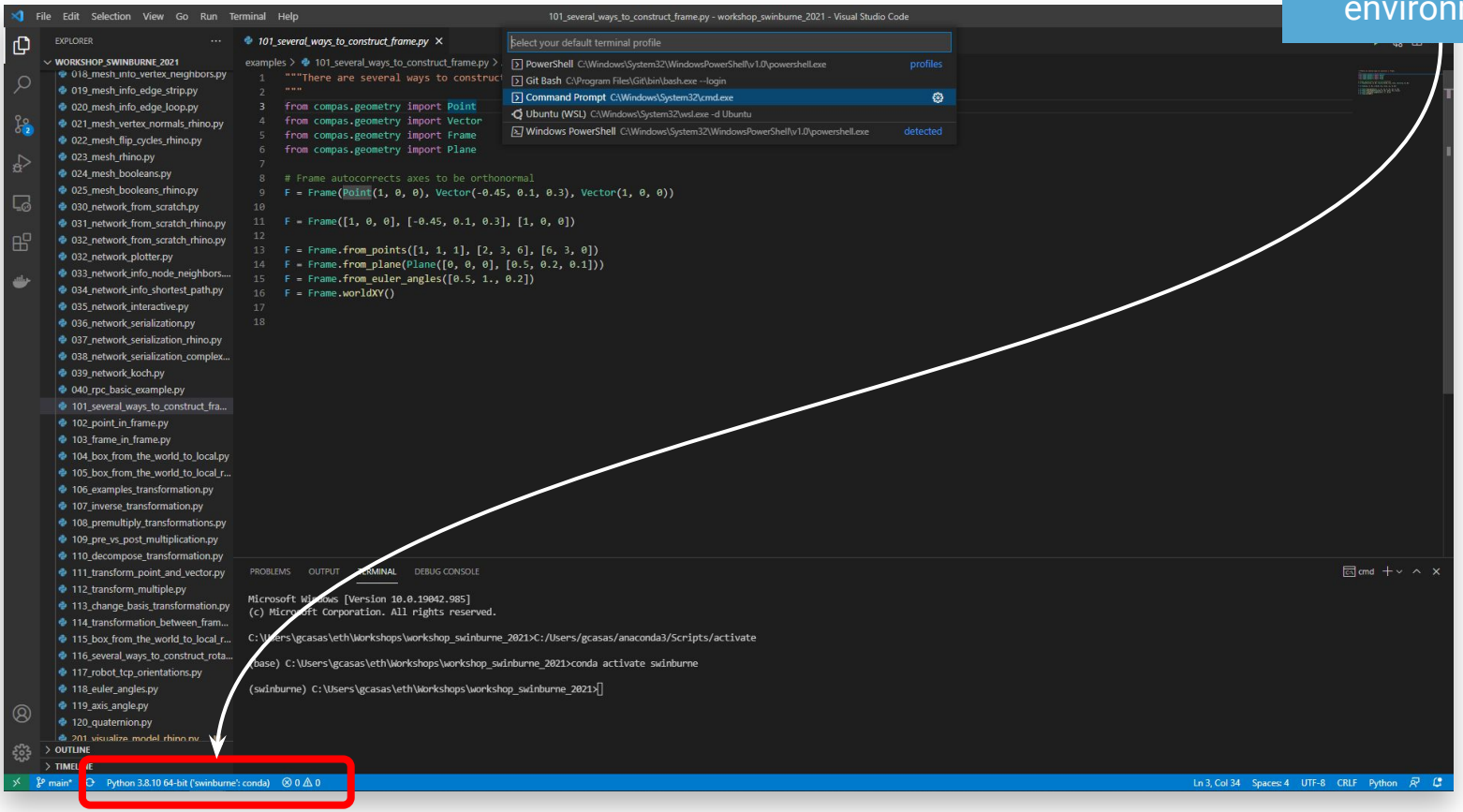
Frame

A frame is defined by a base point and two orthonormal base vectors (xaxis, yaxis), which specify the normal (zaxis).

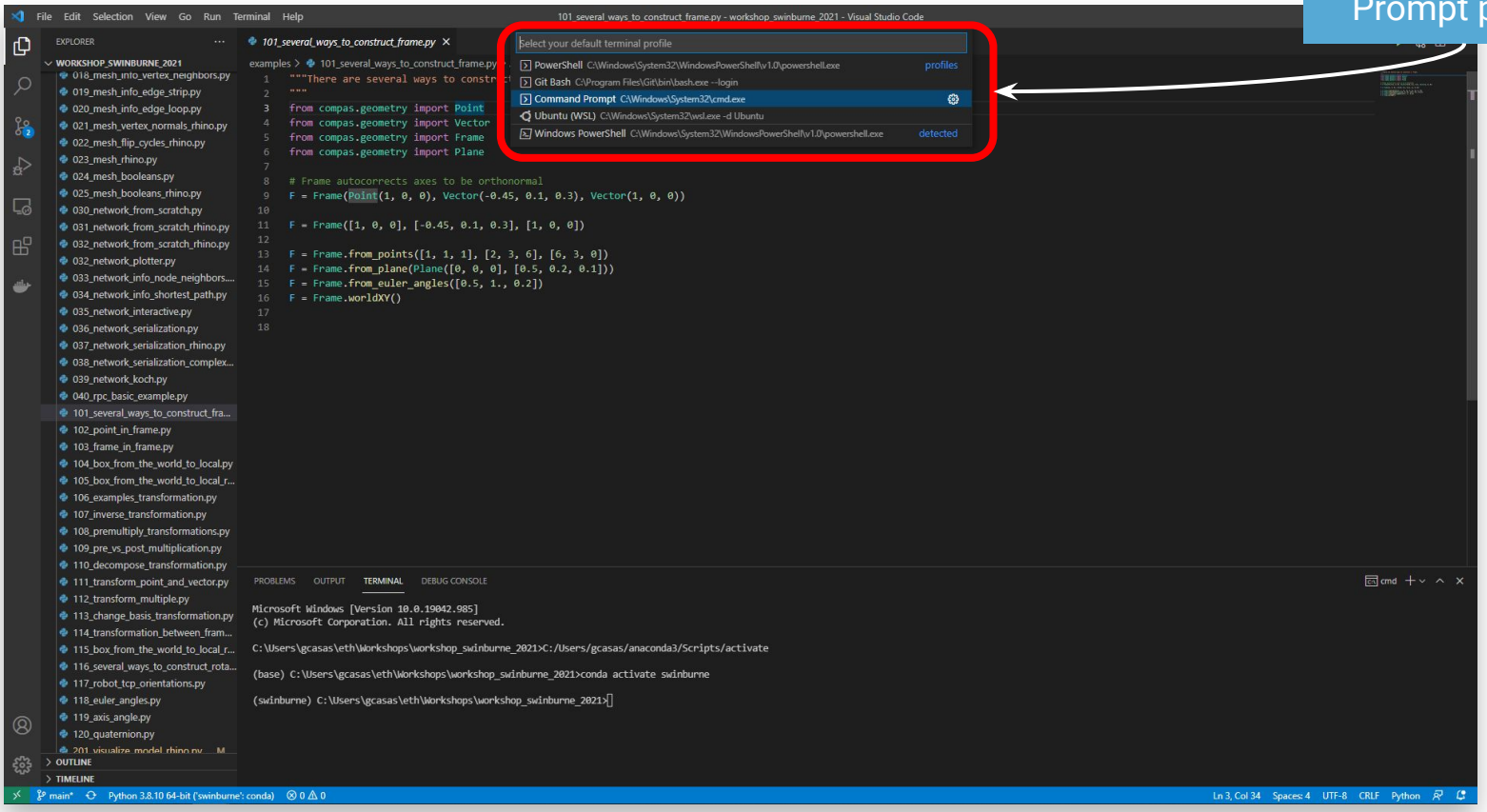
It describes location and orientation in a (right-handed) cartesian coordinate system.



Select environment



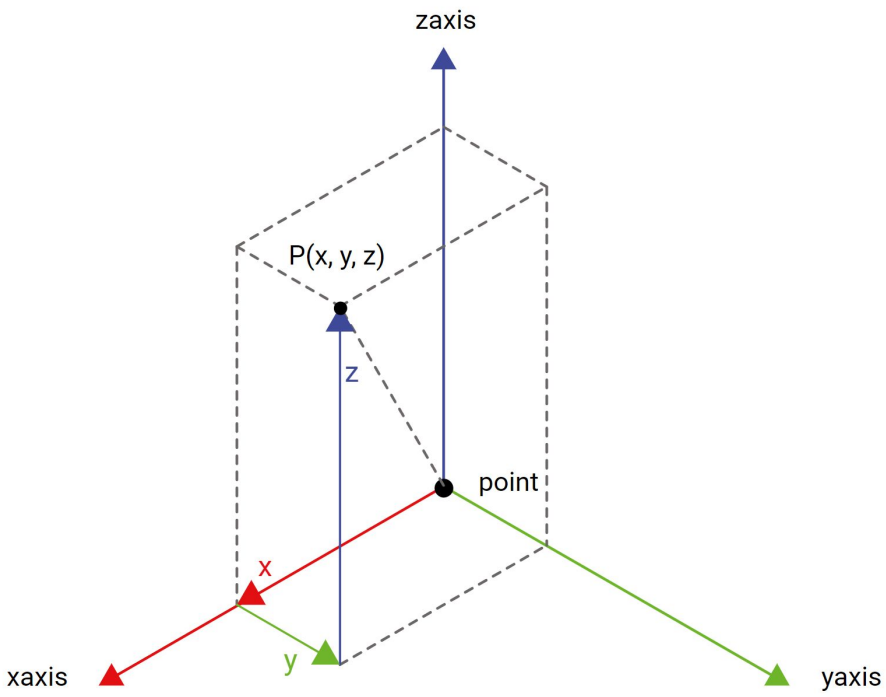
Select Command Prompt profile



```python
"""There are several ways to construct a `Frame`.
"""
from compas.geometry import Point
from compas.geometry import Vector
from compas.geometry import Frame
from compas.geometry import Plane

# Frame autocorrects axes to be orthonormal
F = Frame(Point(1, 0, 0), Vector(-0.45, 0.1, 0.3), Vector(1, 0, 0))

F = Frame([1, 0, 0], [-0.45, 0.1, 0.3], [1, 0, 0])

F = Frame.from_points([1, 1, 1], [2, 3, 6], [6, 3, 0])
F = Frame.from_plane(Plane([0, 0, 0], [0.5, 0.2, 0.1]))
F = Frame.from_euler_angles([0.5, 1., 0.2])
F = Frame.worldXY()
```

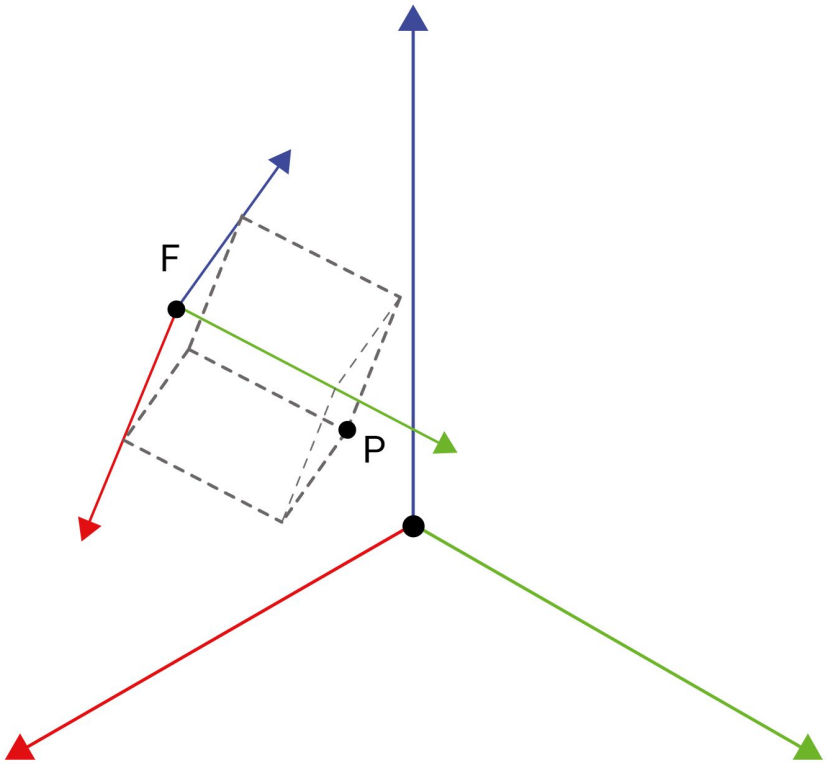
```python
"""There are several ways to construct a `Frame`.
"""
from compas.geometry import Point
from compas.geometry import Vector
from compas.geometry import Frame
from compas.geometry import Plane

# Frame autocorrects axes to be orthonormal
F = Frame(Point(1, 0, 0), Vector(-0.45, 0.1, 0.3), Vector(1, 0, 0))

F = Frame([1, 0, 0], [-0.45, 0.1, 0.3], [1, 0, 0])

F = Frame.from_points([1, 1, 1], [2, 3, 6], [6, 3, 0])
F = Frame.from_plane(Plane([0, 0, 0], [0.5, 0.2, 0.1]))
F = Frame.from_euler_angles([0.5, 1., 0.2])
F = Frame.worldXY()
```

Frame as a cartesian coordinate system



```python
"""Example: 'point in frame'
"""
from compas.geometry import *

point = Point(146.00, 150.00, 161.50)
xaxis = Vector(0.9767, 0.0010, -0.214)
yaxis = Vector(0.1002, 0.8818, 0.4609)

F = Frame(point, xaxis, yaxis)  # coordinate system F
P = Point(35., 35., 35.)  # point in F (local coordinates)

P_ = F.to_world_coordinates(P)  # point in global (world) coordinates
print("The point's world coordinates: {!r}".format(P_))

P2 = F.to_local_coordinates(P_)
print("The point's local coordinates: {!r}".format(P2))  # should equal P
print(allclose(P2, P))
```

Frame in frame



```python
"""Example: 'frame in frame'
"""
from compas.geometry import *

point = Point(146.00, 150.00, 161.50)
xaxis = Vector(0.9767, 0.0010, -0.214)
yaxis = Vector(0.1002, 0.8818, 0.4609)
F0 = Frame(point, xaxis, yaxis) # coordinate system F0

point = Point(35., 35., 35.)
xaxis = Vector(0.604, 0.430, 0.671)
yaxis = Vector(-0.631, 0.772, 0.074)
f_lcf = Frame(point, xaxis, yaxis) # frame f_lcf in F0 (local coordinates)

# frame in global (world) coordinate system
f_wcf = F0.to_world_coordinates(f_lcf)
print(repr(f_wcf))

f_lcf2 = F0.to_local_coordinates(f_wcf) # world coords back to local coords
print(repr(f_lcf2)) # should equal f_lcf
print(f_lcf == f_lcf2)
```

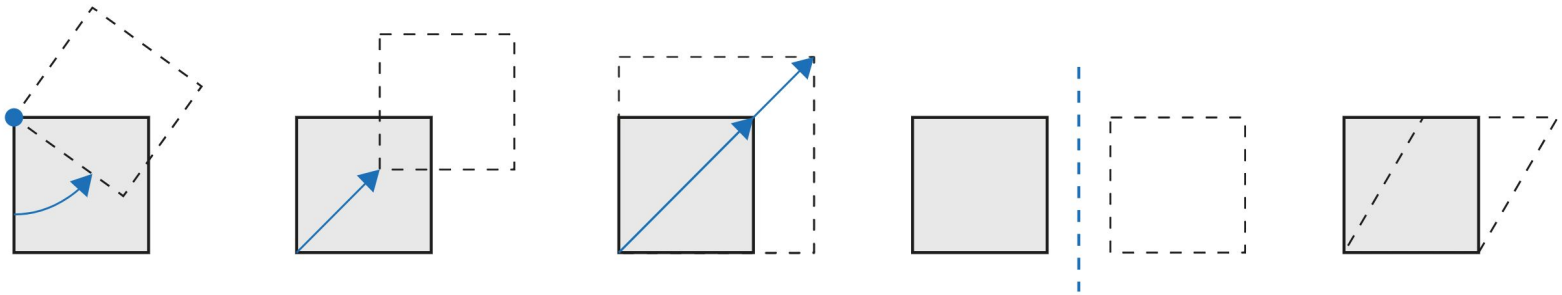
Transformation

Transformations allow you to move, rotate, and scale objects. They are also used to express the coordinates of an object in one frame in the coordinate system of another.

Transformation

Transformations allow you to move, rotate, and scale objects. They are also used to express the coordinates of an object in one frame in the coordinate system of another.

Most transformations preserve the parallel relationship among the parts of the geometry. For example collinear points remain collinear after the transformation. Also points on one plane stay coplanar after transformation. This type of transformation is called an *affine transformation* and concerns transformations such as `Rotation`, `Translation`, `Scale`, `Reflection`, `Shear` and orthogonal and parallel `Projection`. Only perspective `Projection` is not *affine*.



A generic transformation matrix looks like this:

$$\mathbf{T} = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix}$$

Homogenization

This is one of the reasons for distinguishing between points and vectors!

To transform points and vectors, i.e. multiply them with the transformation matrix, we need to homogenize them first. This means representing a 3-vector (x, y, z) as a 4-vector (x, y, z, 1) for points, or (x, y, z, 0) for vectors.

Points:

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix} \times \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Vectors:

$$\begin{bmatrix} x' \\ y' \\ z' \\ 0 \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix} \times \begin{bmatrix} x \\ y \\ z \\ 0 \end{bmatrix}$$

Different types of transformations may take more specific forms:

The `Translation` just uses the first 3 coefficients of the 4th column,

$$\mathbf{T} = \begin{bmatrix} 1 & 0 & 0 & a_{14} \\ 0 & 1 & 0 & a_{24} \\ 0 & 0 & 1 & a_{34} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Different types of transformations may take more specific forms:

The `Translation` just uses the first 3 coefficients of the 4th column,

$$\mathbf{T} = \begin{bmatrix} 1 & 0 & 0 & a_{14} \\ 0 & 1 & 0 & a_{24} \\ 0 & 0 & 1 & a_{34} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Point:

$$\begin{bmatrix} 1 & 0 & 0 & a_{14} \\ 0 & 1 & 0 & a_{24} \\ 0 & 0 & 1 & a_{34} \\ 0 & 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} x + a_{14} \\ y + a_{24} \\ z + a_{34} \\ 1 \end{bmatrix}$$

Different types of transformations may take more specific forms:

The `Translation` just uses the first 3 coefficients of the 4th column,

$$\mathbf{T} = \begin{bmatrix} 1 & 0 & 0 & a_{14} \\ 0 & 1 & 0 & a_{24} \\ 0 & 0 & 1 & a_{34} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Vector:

$$\begin{bmatrix} 1 & 0 & 0 & a_{14} \\ 0 & 1 & 0 & a_{24} \\ 0 & 0 & 1 & a_{34} \\ 0 & 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} x \\ y \\ z \\ 0 \end{bmatrix} = \begin{bmatrix} x \\ y \\ z \\ 0 \end{bmatrix}$$

`Scale` uses only the first 3 values of the diagonal,

$$\mathbf{T} = \begin{bmatrix} a_{11} & 0 & 0 & 0 \\ 0 & a_{22} & 0 & 0 \\ 0 & 0 & a_{33} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

and `Rotation` just uses the upper left 3x3 coefficients.

$$\mathbf{T} = \begin{bmatrix} a_{11} & a_{12} & a_{13} & 0 \\ a_{21} & a_{22} & a_{23} & 0 \\ a_{31} & a_{32} & a_{33} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

```python
"""Transformation examples
"""
from compas.geometry import *

axis, angle = [0.2, 0.4, 0.1], 0.3
R = Rotation.from_axis_and_angle(axis, angle)
print("Rotation:\n", R)

translation_vector = [5, 3, 1]
T = Translation.from_vector(translation_vector)
print("Translation:\n", T)

scale_factors = [0.1, 0.3, 0.4]
S = Scale.from_factors(scale_factors)
print("Scale:\n", S)

point, normal = [0.3, 0.2, 1], [0.3, 0.1, 1]
R = Reflection.from_plane((point, normal))
print("Reflection:\n", R)
```

Inverse transformation

For most of the transformations we work with, there exists an inverse transformation T^{-1} which "undos" the effect of T.

$T \times T^{-1} = I$, with I as identity matrix.

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \frac{1}{2} & 0 & 0 \\ 0 & 0 & \frac{1}{3} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

```python
"""Example: Transform a point and invert the transformation
"""
from compas.geometry import *
from math import pi

p = Point(3, 4, 5)
T = Rotation.from_axis_and_angle([2, 2, 2], pi/4)

p.transform(T)  # transform Point p with T

Tinv = T.inverse()  # create inverse Transformation to T

p.transform(Tinv)  # transform Point p with inverse Transformation

# check if p has the same values as in the beginning
print(p)  # == (3, 4, 5)

# Btw, what is the result of multiplying T with Tinv?
print(T * Tinv)
```

Concatenation

The concatenation of several Transformations are simple matrix multiplications. But matrices are not commutative ($\mathbf{A}\times\mathbf{B}\neq\mathbf{B}\times\mathbf{A}$), so it is important to consider the order of multiplication.

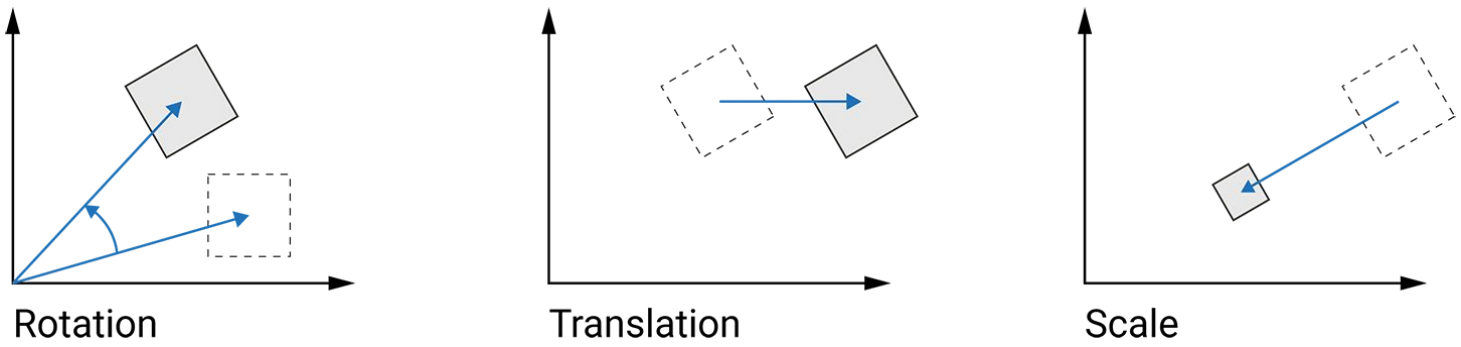
$$\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}$$

Concatenation

If you transform an object first with transformation $\mathbf{A}$, then with transformation $\mathbf{B}$, followed by transformation $\mathbf{C}$, you get the same result as transforming the object with only one transformation $\mathbf{M}$, which is calculated by $\mathbf{M}=\mathbf{C}\times\mathbf{B}\times\mathbf{A}$. (Transformations applied from right to left.)

Think of transformations with respect to the global (world) coordinate system.

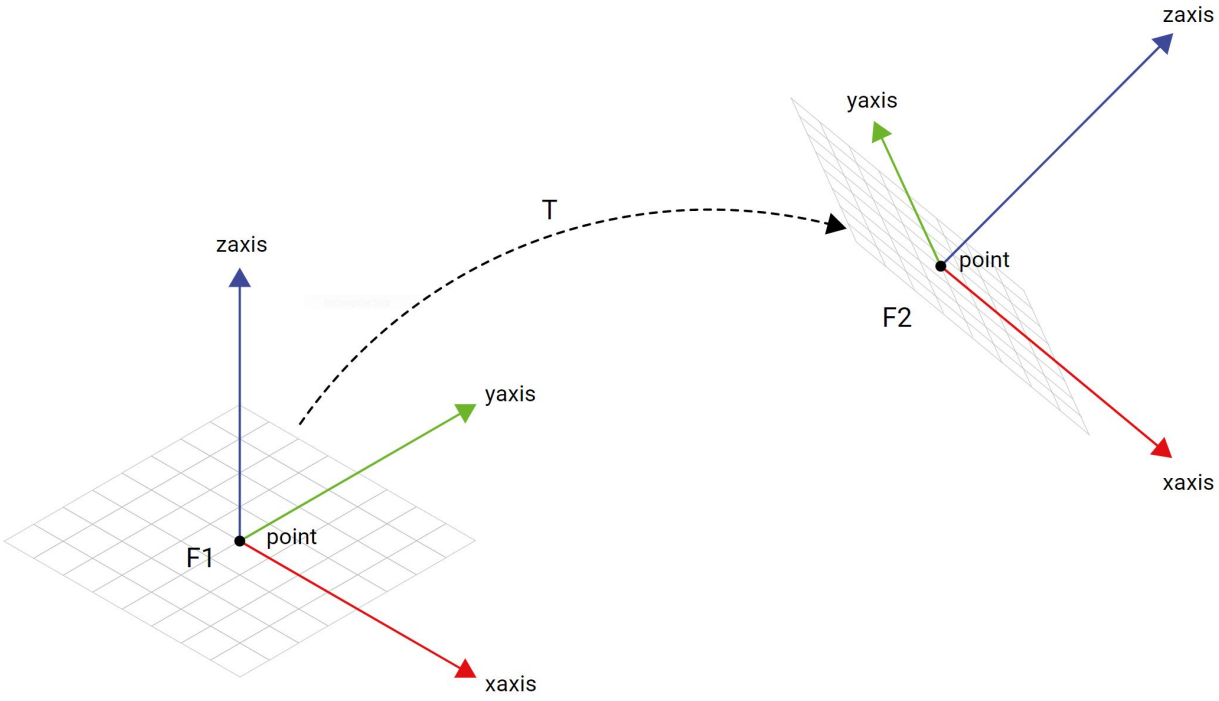


```python
"""Example: Multiply transformations
"""
from compas.geometry import *

p = Point(1, 1, 1)
A = Translation.from_vector([1, 2, 3])  # create Translation
B = Rotation.from_axis_and_angle([-0.8, 0.35, 0.5], 2.2)  # create Rotation
C = Scale.from_factors([0.1, 0.3, 0.4])  # create Scale

# Transform p1 one by one
p1 = p.copy()
p1.transform(A)
p1.transform(B)
p1.transform(C)

# Transform with only one concatenated matrix
p2 = p.copy()
p2.transform(C * B * A)
```

```python
"""Example: multiplication
"""
import math
from compas.geometry import Translation
from compas.geometry import Rotation
from compas.geometry import Scale

R = Rotation.from_axis_and_angle([0, 0, 1], math.radians(30))
T = Translation.from_vector([2, 0, 0])
S = Scale.from_factors([0.5] * 3)
X1 = S * T * R
X2 = R * T * S

print(X1)
print(X2)
print(X1 == X2)
```

```python
"""Example: transform point and vector
"""
from compas.geometry import Point, Vector, Rotation

# create Rotation around point with axis and angle
point = Point(1.0, 0.0, 0.0)
axis, angle = [-0.248, -0.786, -0.566], 2.78
R = Rotation.from_axis_and_angle(axis, angle, point=point)

# apply Transformation to Point
p = Point(1, 1, 1)
p.transform(R)

# apply Transformation to Vector
v = Vector(1, 1, 1)
v.transform(R)

# should not be the same!
print(p)
print(v)
```

Frame and Transformation



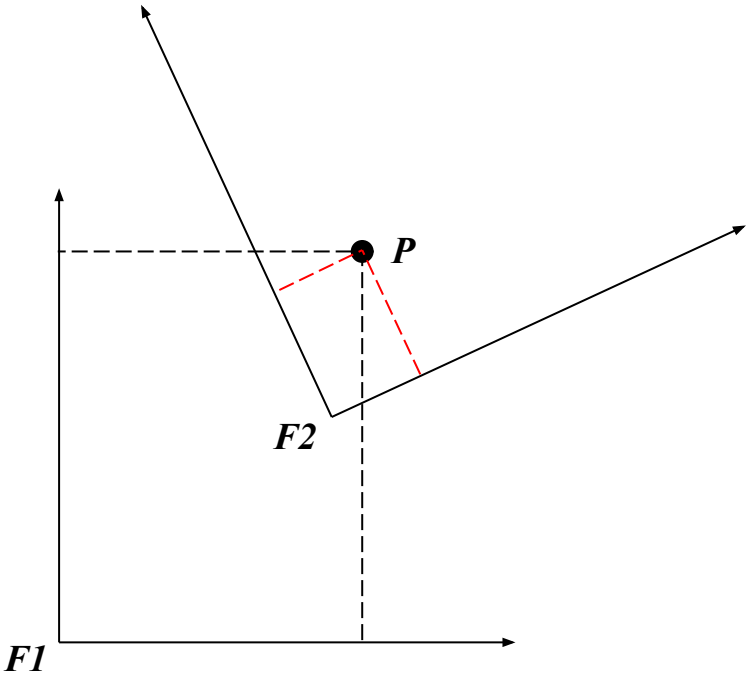
Change-basis transformation and transformation between frames

`Transformation.from_change_of_basis(f1, f2)` and

`Transformation.from_frame_to_frame(f1, f2)`

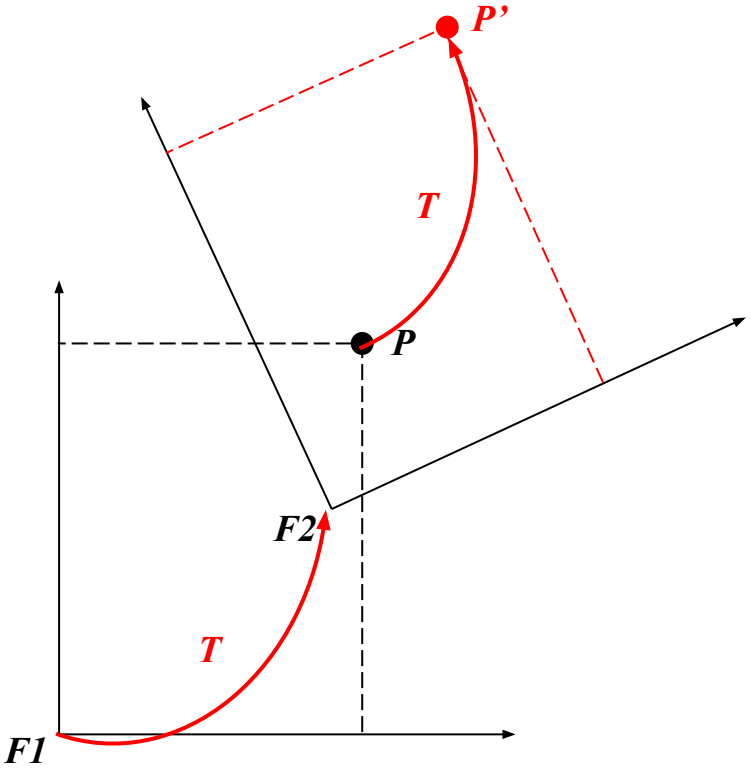
A change-basis transformation allows to remap geometry of one coordinate system in another, i.e. represent the same coordinates in two different frames, whereas the `Transformation` between two frames allows to transform geometry from one coordinate system into the other one.

Change-basis transformation



```python
"""Change-basis transformation.
"""
from compas.geometry import Point
from compas.geometry import Frame
from compas.geometry import Transformation

F1 = Frame.worldXY()
F2 = Frame([1.5, 1, 0], [0.68, 0.68, 0.27], [-0.67, 0.73, -0.15])
P = Point(2, 2, 2)  # local point in F1

# change-basis transformation between two frames F1 and F2.
T = Transformation.from_change_of_basis(F1, F2)

# Represent geometry (point P) in another coordinate frame
print(P.transformed(T))
# You can also use the following
print(F2.to_local_coordinates(P))
```

Transformation between frames

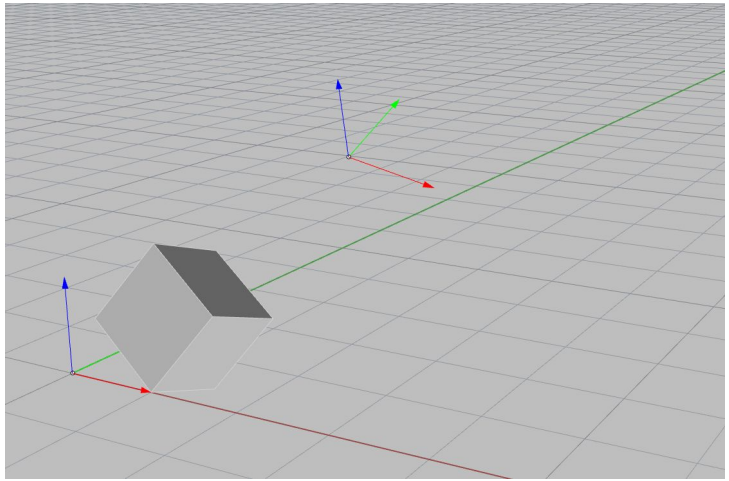


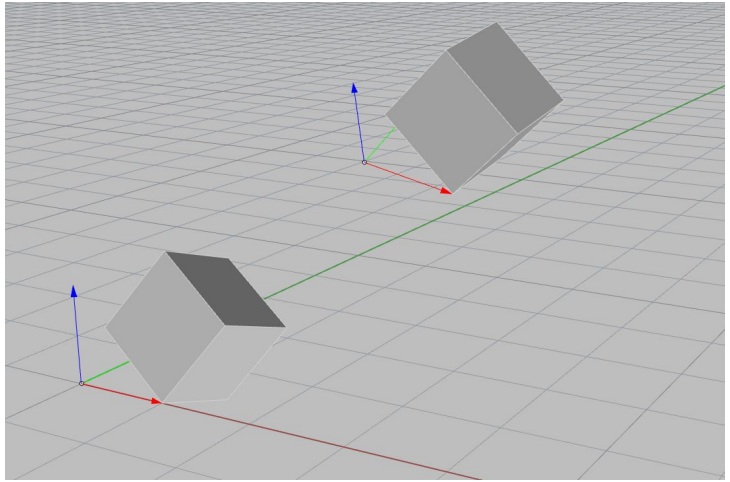
```python
"""Transformation between two frames.
"""
from compas.geometry import Point
from compas.geometry import Frame
from compas.geometry import Transformation

F1 = Frame.worldXY()
F2 = Frame([1.5, 1, 0], [0.68, 0.68, 0.27], [-0.67, 0.73, -0.15])
P = Point(2, 2, 2)  # local point in F1

# transformation between 2 frames F1, F2
T = Transformation.from_frame_to_frame(F1, F2)

# Transform geometry (=point P) into another coordinate frame
print(P.transformed(T))
```

Example

1. Bring a box from the world coordinate system into another coordinate system

2. Draw frames and boxes in Rhino





```python
"""Example: Bring a box from the world coordinate system into another coordinate
system and view in Rhino.
"""
from compas.geometry import Frame
from compas.geometry import Transformation
from compas.geometry import Box
from compas_rhino.artists import FrameArtist
from compas_rhino.artists import BoxArtist

# Box in the world coordinate system
frame = Frame([1, 0, 0], [-0.45, 0.1, 0.3], [1, 0, 0])
width, length, height = 1, 1, 1
box = Box(frame, width, length, height)

# Frame F representing a coordinate system
F = Frame([2, 2, 2], [0.978, 0.010, -0.210], [0.090, 0.882, 0.463])

# Get transformation between frame F and world
# T = ...
# Apply transformation on box.
# box_transformed = ...
```

`transform()` vs `transformed()`

- Make a copy of box and transform it (don't change box):

  ```
  box_transformed = box.transformed(T)
  ```

- Transform box (returns `None`):

  ```
  box.transform(T)
  ```

`transform()` vs `transformed()`

- Make a copy of box and transform it (don't change box):

  ```
  box_transformed = box.transformed(T)
  ```

- Transform box (returns `None`):

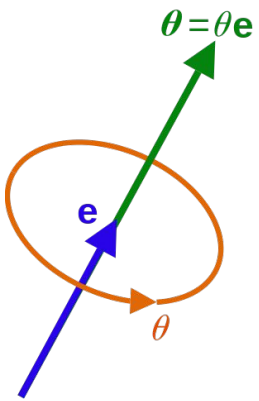
  ```
  box.transform(T)
  ```

Depending on the size of the object, copying takes time. So consider carefully if you really need a copy.

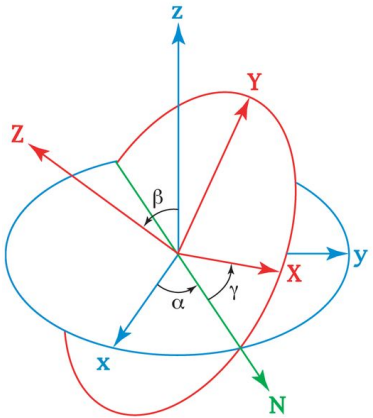
Rotation and orientation

A `Rotation` is a circular movement of an object around a point of rotation. A three-dimensional object can always be rotated around an infinite number of imaginary lines called *rotation axes*.

There are many ways to represent an orientation of an object



Axis–Angle



Euler angles

$$w + x\,\mathbf{i} + y\,\mathbf{j} + z\,\mathbf{k}$$

Unit quaternions

```python
"""There are several ways to construct a `Rotation`.
"""
import math
from compas.geometry import Frame
from compas.geometry import Rotation

R = Rotation.from_axis_and_angle([1, 0, 0], math.radians(30))
R = Rotation.from_axis_and_angle([1, 0, 0], math.radians(30), point=[1, 0, 0])
R = Rotation.from_basis_vectors([0.68, 0.68, 0.27], [-0.67, 0.73, -0.15])
R = Rotation.from_frame(Frame([1, 1, 1], [0.68, 0.68, 0.27], [-0.67, 0.73, -0.15]))
R = Rotation.from_axis_angle_vector([-0.043, -0.254, 0.617])
R = Rotation.from_quaternion([0.945, -0.021, -0.125, 0.303])
R = Rotation.from_euler_angles([1.4, 0.5, 2.3], static=True, axes='xyz')

print(R)
```

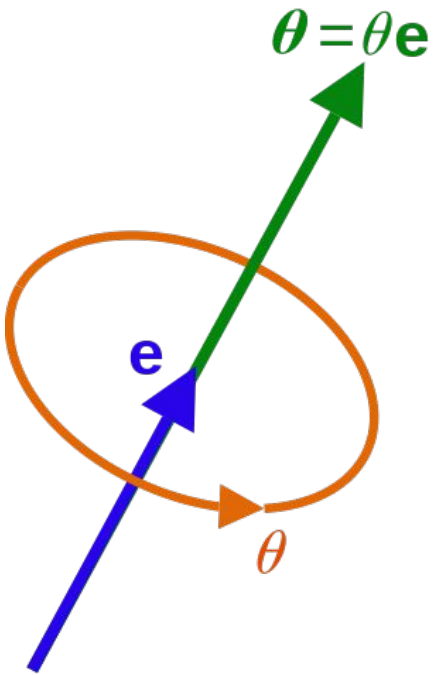
```python
"""Example: Different robot vendors use different conventions to describe TCP
orientation."""

from compas.geometry import Point
from compas.geometry import Vector
from compas.geometry import Frame

point = Point(0.0, 0.0, 63.0)
xaxis = Vector(0.68, 0.68, 0.27)
yaxis = Vector(-0.67, 0.73, -0.15)
F = Frame(point, xaxis, yaxis)

print(F.quaternion)  # ABB
print(F.euler_angles(static=False, axes='xyz'))  # Staubli
print(F.euler_angles(static=False, axes='zyx'))  # KUKA
print(F.axis_angle_vector)  # UR
```

```python
"""Example: Rotations from euler angles, rotate an object based on 3 euler angles.
"""
from compas.geometry import Rotation

alpha, beta, gamma = -0.156, -0.274, 0.785 # euler angles
static, axes = True, 'xyz'

# Version 1: Create Rotation from angles
R1 = Rotation.from_euler_angles([alpha, beta, gamma], static, axes)

# Version 2: Concatenate 3 Rotations
xaxis, yaxis, zaxis = [1, 0, 0], [0, 1, 0], [0, 0, 1]
Rx = Rotation.from_axis_and_angle(xaxis, alpha)
Ry = Rotation.from_axis_and_angle(yaxis, beta)
Rz = Rotation.from_axis_and_angle(zaxis, gamma)

if static:
    R2 = Rz * Ry * Rx
else:
    R2 = Rx * Ry * Rz
```

Axis-angle representation

The axis–angle representation of a rotation parameterizes a rotation in a three-dimensional Euclidean space by two quantities:

- **a unit vector e** (indicating the direction of an axis of rotation)
- **angle θ** (describing the magnitude of the rotation about the axis)



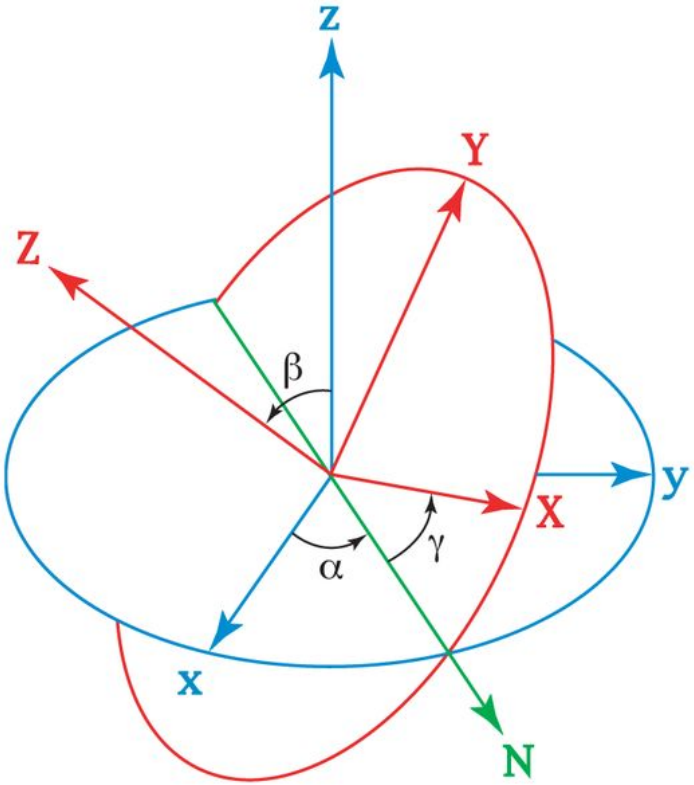
```python
"""Example: Create a rotation from and axis and an angle.
"""
from compas.geometry import Vector
from compas.geometry import Rotation

aav = Vector(-0.043, -0.254, 0.617)
angle, axis = aav.unitized(), aav.length
print(angle, axis)

R = Rotation.from_axis_angle_vector(aav)
axis, angle = R.axis_and_angle
print(axis, angle)
```

Euler angles

The Euler angles are three angles introduced by Leonhard Euler to describe the orientation of a rigid body with respect to a fixed coordinate system.

The three elemental rotations may be

- extrinsic (`static=True`, rotations about the axes xyz of the original coordinate system, which is assumed to remain motionless), or
- intrinsic (`static=False`, rotations about the axes of the rotating coordinate system XYZ, solidary with the moving body, which changes its orientation after each elemental rotation).



Unit Quaternion

In mathematics, the quaternions comprise a number system that extends the complex numbers.

Quaternions are generally represented in the form:

$$w + x\,\mathbf{i} + y\,\mathbf{j} + z\,\mathbf{k}$$

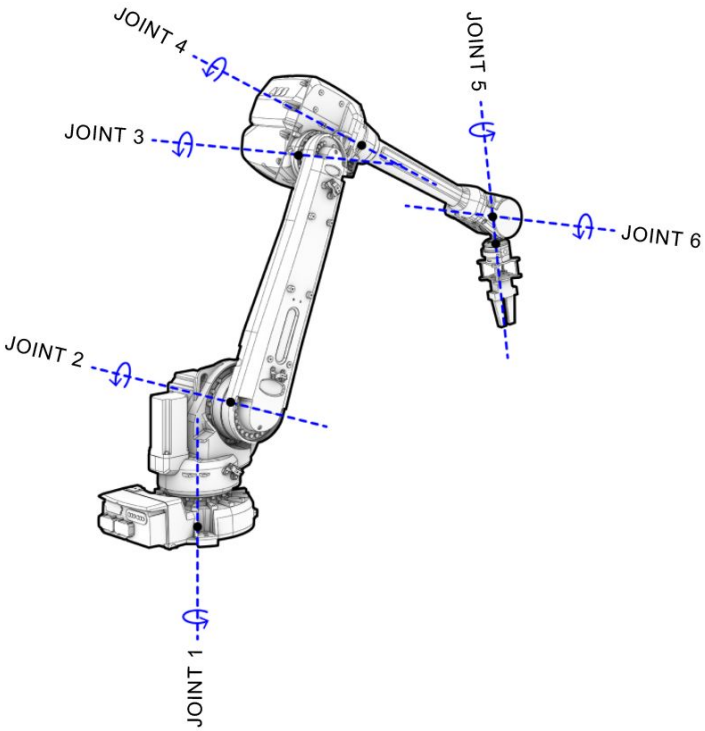
where x, y, z, and w are real numbers, and $\mathbf{i}$, $\mathbf{j}$, and $\mathbf{k}$ are the fundamental quaternion units.

Unit quaternions, also known as versors, provide a convenient mathematical notation for representing orientations and rotations of objects in three dimensions, and are a variation on the axis-angle representation. Compared to Euler angles they are simpler to compose and compared to rotation matrices they are more compact, more numerically stable, and more efficient.

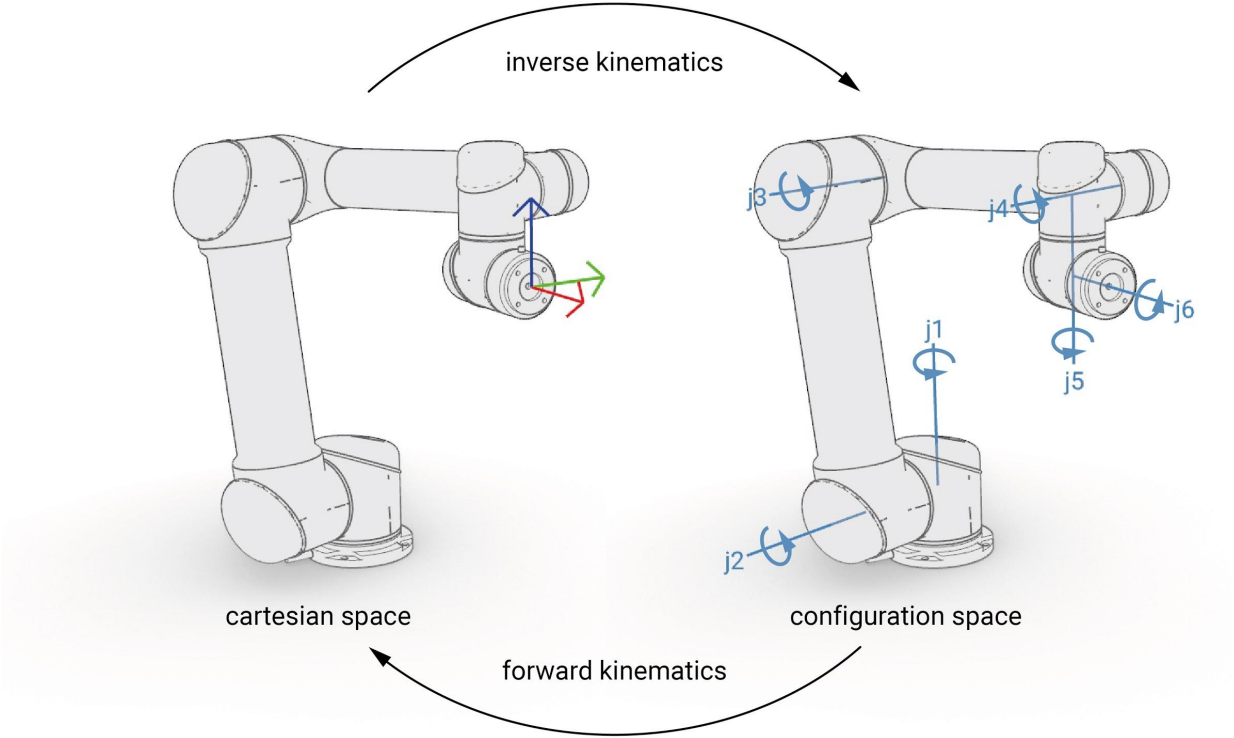
```python
"""Example: Create a rotation from a quaternion.
"""
from compas.geometry import Rotation
from compas.geometry import Quaternion

q = Quaternion(0.918958, -0.020197, -0.151477, 0.363544)
print(q.is_unit)
R = Rotation.from_quaternion(q)
print(R.quaternion == q)
```

kinematics

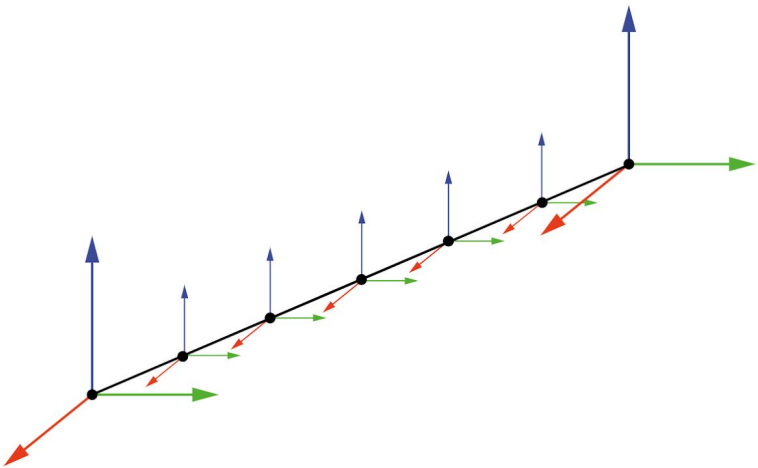
Configuration

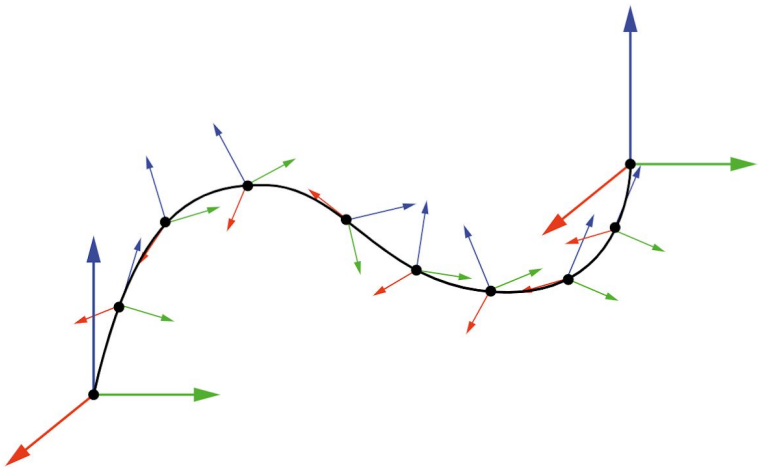


Joint vs Cartesian space



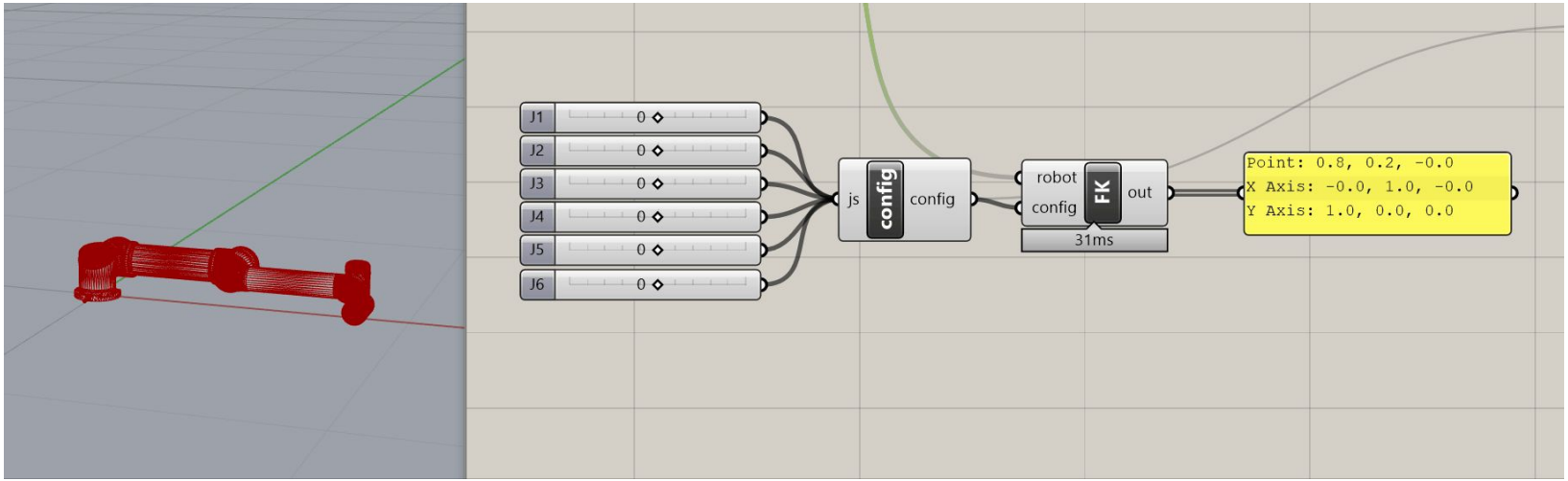
Motion in Joint vs Cartesian space





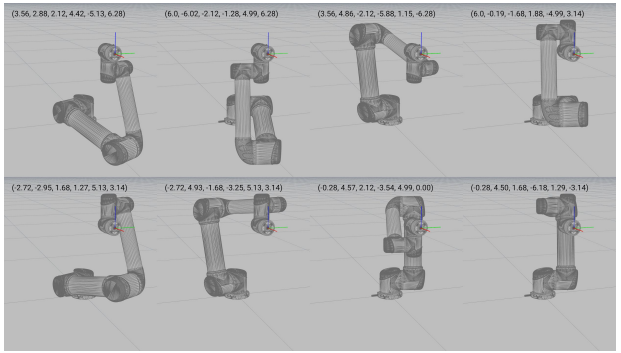
Forward Kinematics

- **From a configuration, get frame of tip (or other link)**
- Fast to solve for serial manipulators



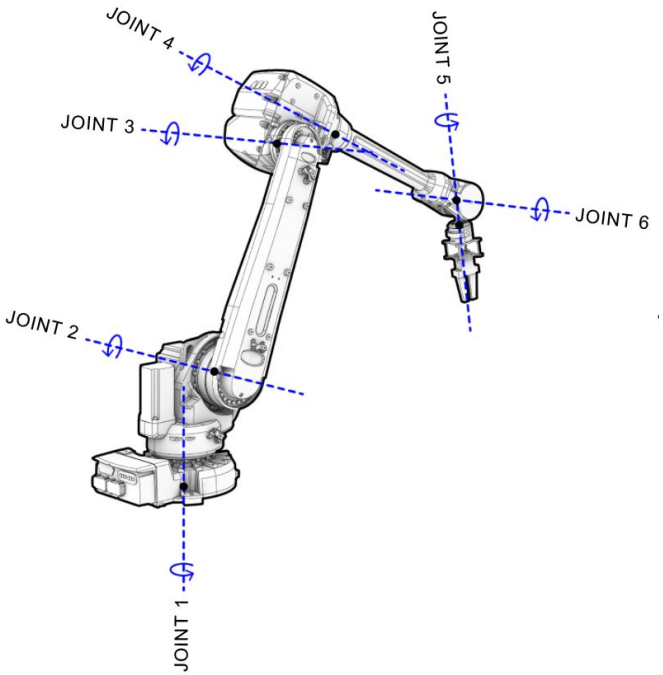
Inverse Kinematics

- **From a frame, get a feasible configuration**
- Analytical approach
  - *Pro:* Computes all solutions and determines if no solution exists
  - *Pro:* Very fast to compute
  - *Pro:* No need to define initial guesses
  - *Con:* Deriving equations is difficult or tedious
  - *Con:* Different derivation for different kinematic structures
  - *Con:* Only works for non-redundant robots
- Numerical approach
  - Based on an initial guess, calculate a sequence of configurations such that the error approaches 0.
  - *Pro:* Works on most robots including redundant ones
  - *Con:* Computes only one solution at a time
  - *Con:* Slower to compute
  - *Con:* Does not indicate if no solution exists



break

robot models



URDF format

Tree structure

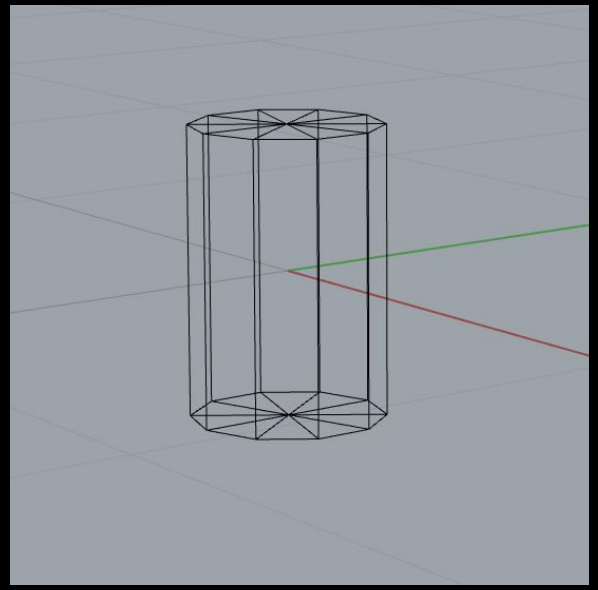
Open source

Visualize model

```python
from compas_rhino.artists import RobotModelArtist
from compas.robots import RobotModel

model = RobotModel.from_urdf_file('models/01_myfirst.urdf')

artist = RobotModelArtist(model, layer='COMPAS::Robot Viz')
artist.clear_layer()
artist.draw_visual()
```

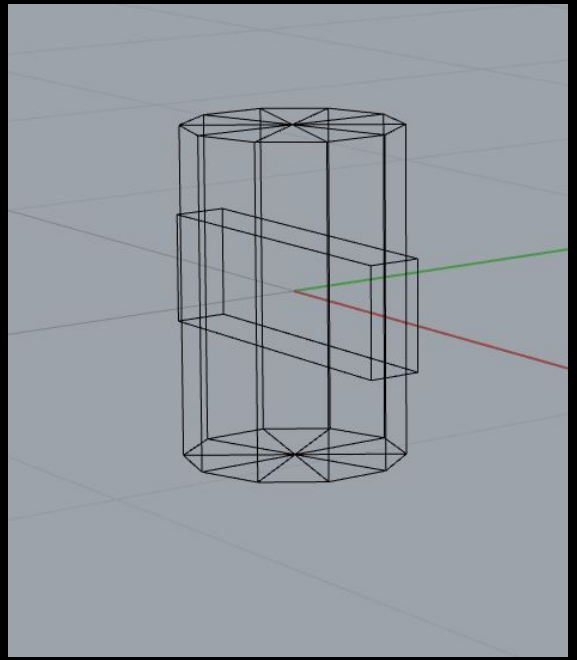
```xml
<?xml version="1.0"?>
<robot name="myfirst">
 <link name="base_link">
   <visual>
     <geometry>
       <cylinder length="0.6" radius="0.2"/>
     </geometry>
   </visual>
 </link>
</robot>
```

```xml
<?xml version="1.0"?>
<robot name="myfirst">
 <link name="base_link">
   <visual>
     <geometry>
       <cylinder length="0.6" radius="0.2"/>
     </geometry>
   </visual>
 </link>
</robot>
```



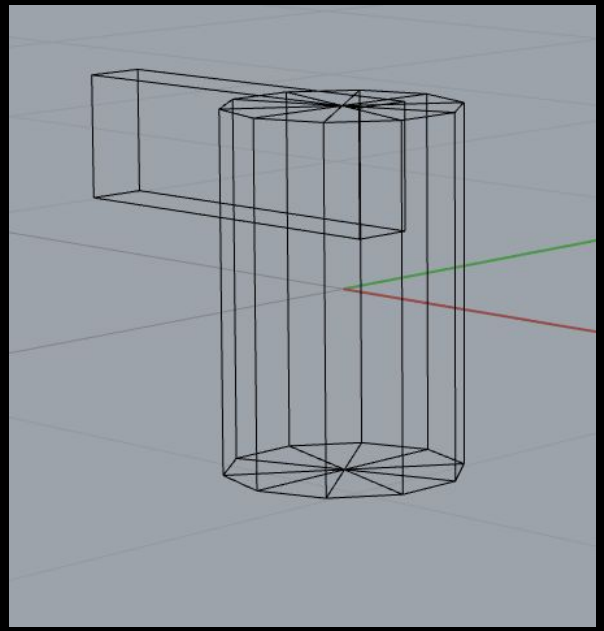
```xml
<?xml version="1.0"?>
<robot name="multipleshapes">

 <link name="base_link">
   <visual>
     <geometry>
       <cylinder length="0.6" radius="0.2"/>
     </geometry>
   </visual>
 </link>

 <link name="right_leg">
   <visual>
     <geometry>
       <box size="0.6 0.1 0.2"/>
     </geometry>
   </visual>
 </link>

 <joint name="base_to_right_leg" type="fixed">
   <parent link="base_link"/>
   <child link="right_leg"/>
 </joint>

</robot>
```

```xml
<?xml version="1.0"?>
<robot name="multipleshapes">

 <link name="base_link">
   <visual>
     <geometry>
       <cylinder length="0.6" radius="0.2"/>
     </geometry>
   </visual>
 </link>

 <link name="right_leg">
   <visual>
     <geometry>
       <box size="0.6 0.1 0.2"/>
     </geometry>
   </visual>
 </link>

 <joint name="base_to_right_leg" type="fixed">
   <parent link="base_link"/>
   <child link="right_leg"/>
 </joint>

</robot>
```



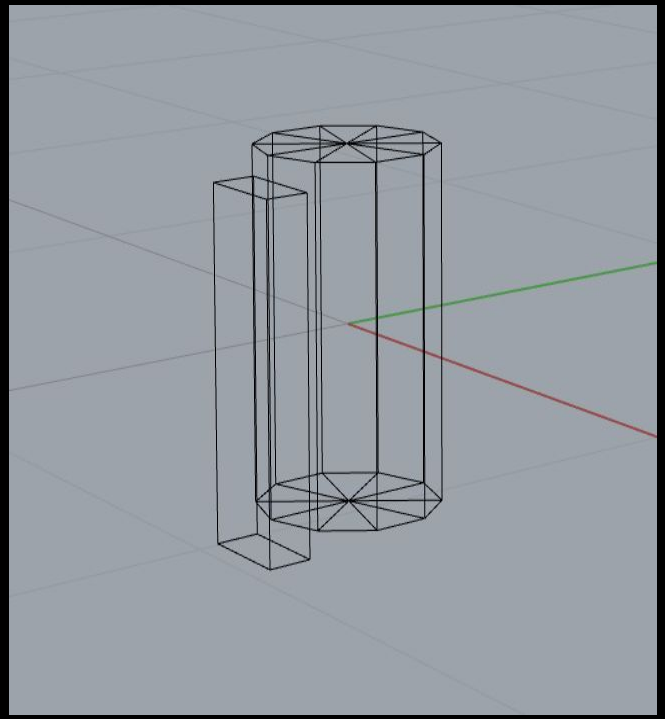
```xml
<?xml version="1.0"?>
<robot name="origins">

 <link name="base_link">
   <visual>
     <geometry>
       <cylinder length="0.6" radius="0.2"/>
     </geometry>
   </visual>
 </link>

 <link name="right_leg">
   <visual>
     <geometry>
       <box size="0.6 0.1 0.2"/>
     </geometry>
   </visual>
 </link>

 <joint name="base_to_right_leg" type="fixed">
   <parent link="base_link"/>
   <child link="right_leg"/>
   <origin xyz="0 -0.22 0.25"/>
 </joint>

</robot>
```

```xml
<?xml version="1.0"?>
<robot name="origins">

 <link name="base_link">
   <visual>
     <geometry>
       <cylinder length="0.6" radius="0.2"/>
     </geometry>
   </visual>
 </link>

 <link name="right_leg">
   <visual>
     <geometry>
       <box size="0.6 0.1 0.2"/>
     </geometry>
   </visual>
 </link>

 <joint name="base_to_right_leg" type="fixed">
   <parent link="base_link"/>
   <child link="right_leg"/>
   <origin xyz="0 -0.22 0.25"/>
 </joint>

</robot>
```

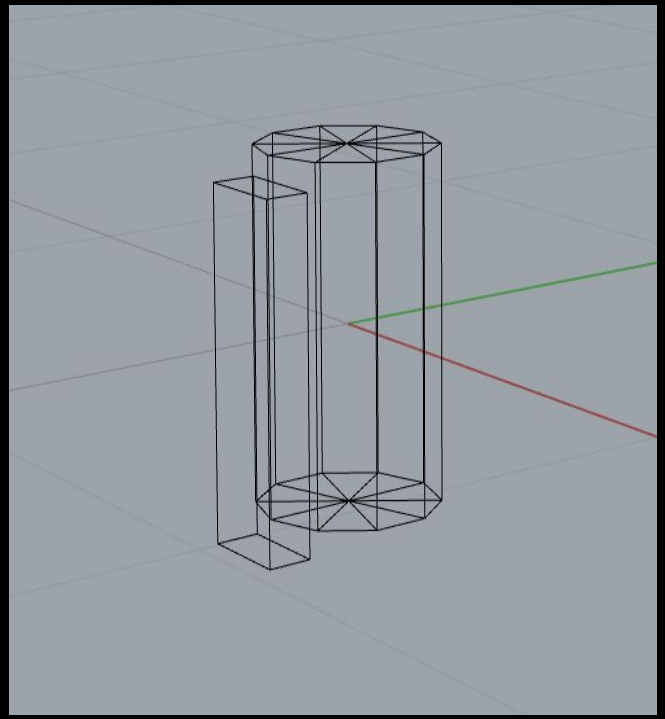


```xml
<?xml version="1.0"?>
<robot name="origins">

 <link name="base_link">
   <visual>
     <geometry>
       <cylinder length="0.6" radius="0.2"/>
     </geometry>
   </visual>
 </link>

 <link name="right_leg">
   <visual>
     <geometry>
       <box size="0.6 0.1 0.2"/>
     </geometry>
     <origin rpy="0 1.57075 0" xyz="0 0 -0.3"/>
   </visual>
 </link>

 <joint name="base_to_right_leg" type="fixed">
   <parent link="base_link"/>
   <child link="right_leg"/>
   <origin xyz="0 -0.22 0.25"/>
 </joint>

</robot>
```

```xml
<?xml version="1.0"?>
<robot name="origins">

 <link name="base_link">
   <visual>
     <geometry>
       <cylinder length="0.6" radius="0.2"/>
     </geometry>
   </visual>
 </link>

 <link name="right_leg">
   <visual>
     <geometry>
       <box size="0.6 0.1 0.2"/>
     </geometry>
     <origin rpy="0 1.57075 0" xyz="0 0 -0.3"/>
   </visual>
 </link>

 <joint name="base_to_right_leg" type="fixed">
   <parent link="base_link"/>
   <child link="right_leg"/>
   <origin xyz="0 -0.22 0.25"/>
 </joint>

</robot>
```



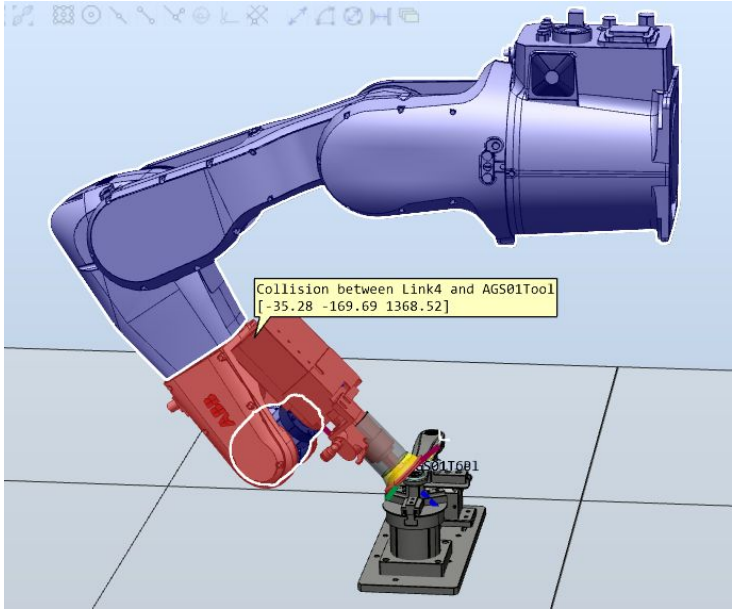
```xml
<?xml version="1.0"?>
<robot name="origins">

 <link name="base_link">
   <visual>
     <geometry>
       <mesh filename="package://basic/cylinder.obj"/>
     </geometry>
   </visual>
 </link>

 <link name="right_leg">
   <visual>
     <geometry>
       <mesh filename="package://basic/box.obj"/>
     </geometry>
     <origin rpy="0 1.57075 0" xyz="0 0 -0.3"/>
   </visual>
 </link>

 <joint name="base_to_right_leg" type="fixed">
   <parent link="base_link"/>
   <child link="right_leg"/>
   <origin xyz="0 -0.22 0.25"/>
 </joint>

</robot>
```



Visualize model with meshes

```
from compas_rhino.artists import RobotModelArtist
from compas.robots import RobotModel
from compas.robots import LocalPackageMeshLoader

model = RobotModel.from_urdf_file('models/05_origins_meshes.urdf')

loader = LocalPackageMeshLoader('models', 'basic')
model.load_geometry(loader)

artist = RobotModelArtist(model, layer='COMPAS::Robot Viz')
artist.clear_layer()
artist.draw_visual()
```

Collision checking



Source: https://forums.robotstudio.com/discussion/10611/how-to-generate-collision-free-path-with-powerpacs

Use different visual/collision geometry

Use bounding volumes

Use primitives

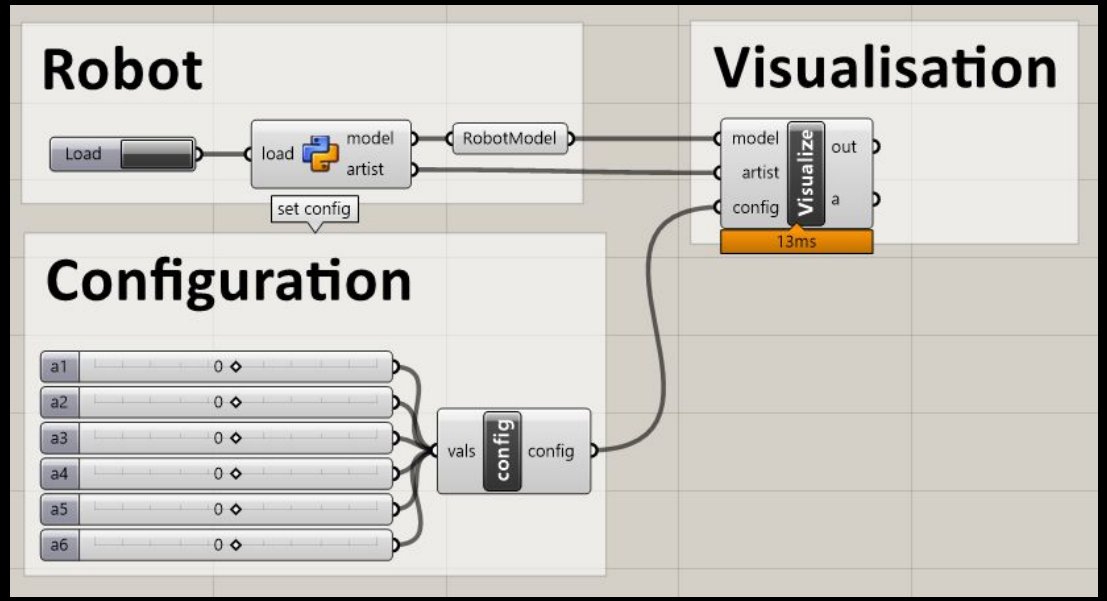
Load local model

```
from compas.robots import LocalPackageMeshLoader
from compas.robots import RobotModel
from compas_rhino.artists import RobotModelArtist

compas.PRECISION = '12f'

loader = LocalPackageMeshLoader('models', 'ur_description')
model = RobotModel.from_urdf_file(loader.load_urdf('ur5.urdf'))
model.load_geometry(loader)

artist = RobotModelArtist(model, layer='COMPAS::Robot Viz')
artist.clear_layer()
artist.draw_visual()
```

Load Github model

```
import compas
from compas.robots import GithubPackageMeshLoader
from compas.robots import RobotModel

compas.PRECISION = '12f'

r = 'ros-industrial/abb'
p = 'abb_irb6600_support'
b = 'kinetic-devel'

github = GithubPackageMeshLoader(r, p, b)
urdf = github.load_urdf('irb6640.urdf')

model = RobotModel.from_urdf_file(urdf)
model.load_geometry(github)
```

Grasshopper



Loading external models

```
# Set high precision to import meshes defined in meters
compas.PRECISION = '12f'

# Load robot and its geometry
with RosClient('localhost') as ros:
    robot = ros.load_robot(load_geometry=True)
    print(robot.model)
```

Building your own robot

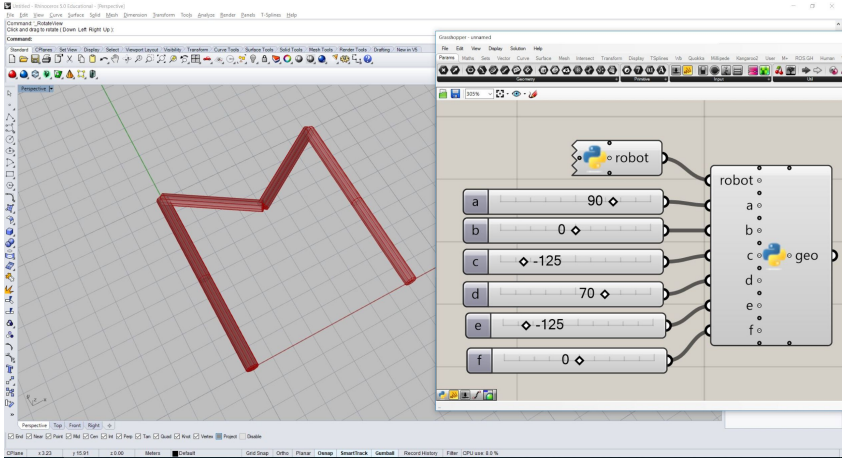
```
model = RobotModel('ur10e',
              joints=[
                  Joint('shoulder_pan_joint', 'revolute', parent='base_link', child='shoulder_link'),
                  Joint('shoulder_lift_joint', 'revolute', parent='shoulder_link', child='upper_arm_link'),
                  Joint('elbow_joint', 'revolute', parent='upper_arm_link', child='forearm_link'),
                  Joint('wrist_1_joint', 'revolute', parent='forearm_link', child='wrist_1_link'),
                  Joint('wrist_2_joint', 'revolute', parent='wrist_1_link', child='wrist_2_link'),
                  Joint('wrist_3_joint', 'revolute', parent='wrist_2_link', child='wrist_3_link'),
              ], links=[
                  Link('base_link'),
                  Link('shoulder_link'),
                  Link('upper_arm_link'),
                  Link('forearm_link'),
                  Link('wrist_1_link'),
                  Link('wrist_2_link'),
                  Link('wrist_3_link'),
              ])
print(model)
```

Building your own robot

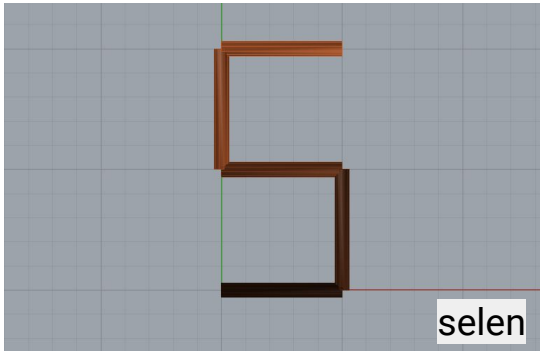
```
# create robot model
model = RobotModel("robot", links=[], joints=[])

# add links
link0 = model.add_link("world")
link1 = model.add_link("link1", visual_mesh=mesh1,)
link2 = model.add_link("link2", visual_mesh=mesh2,)

# add the joints between the links
model.add_joint("joint1", Joint.REVOLUTE, link0, link1, origin1, axis1)
model.add_joint("joint2", Joint.REVOLUTE, link1, link2, origin2, axis2)
```

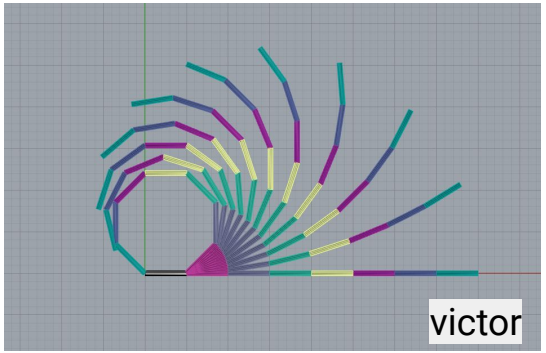
Exercise

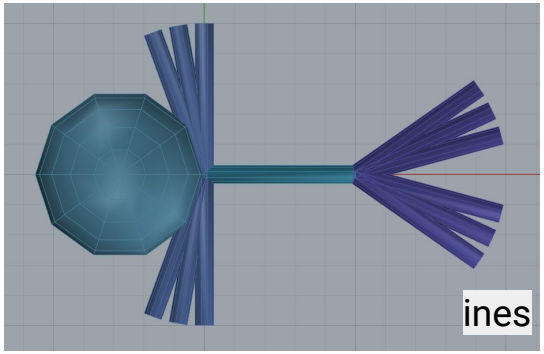
1. Build your own robot with a certain number n of links and n - 1 configurable joints.
2. Create a `Configuration` with certain values and the correct joint types.
3. Create a `RobotModelArtist` of your preference (e.g. `compas_ghpython` or `compas_rhino`)
4. Use the artist to `update` the robot with the created configuration, such that it configures into the letter of your choice (or any other identifiable figure).

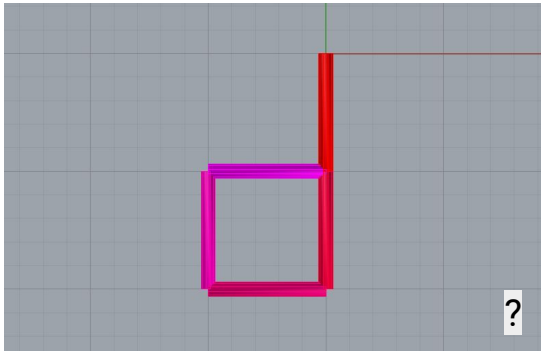


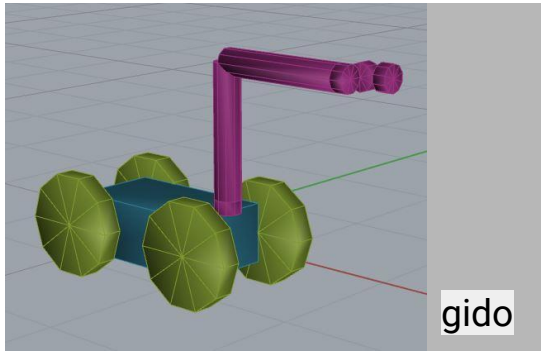
Robot gallery











Day 1: ✅

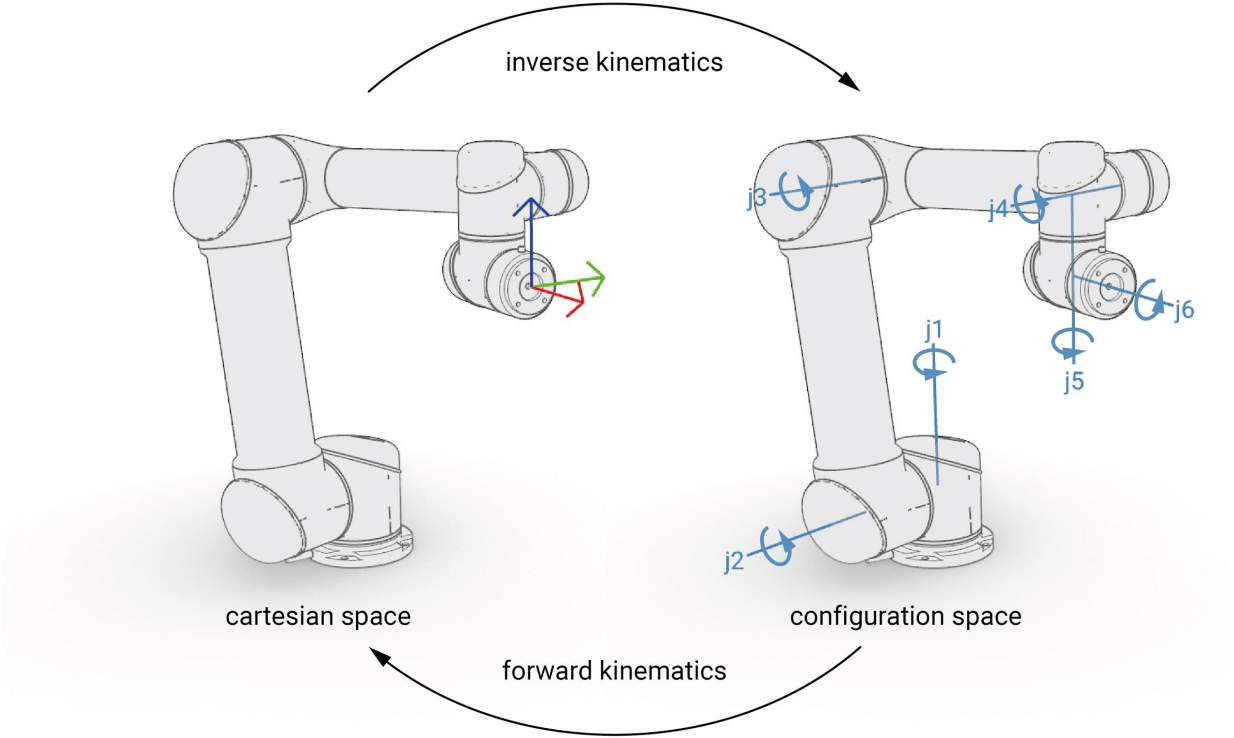
Questions?

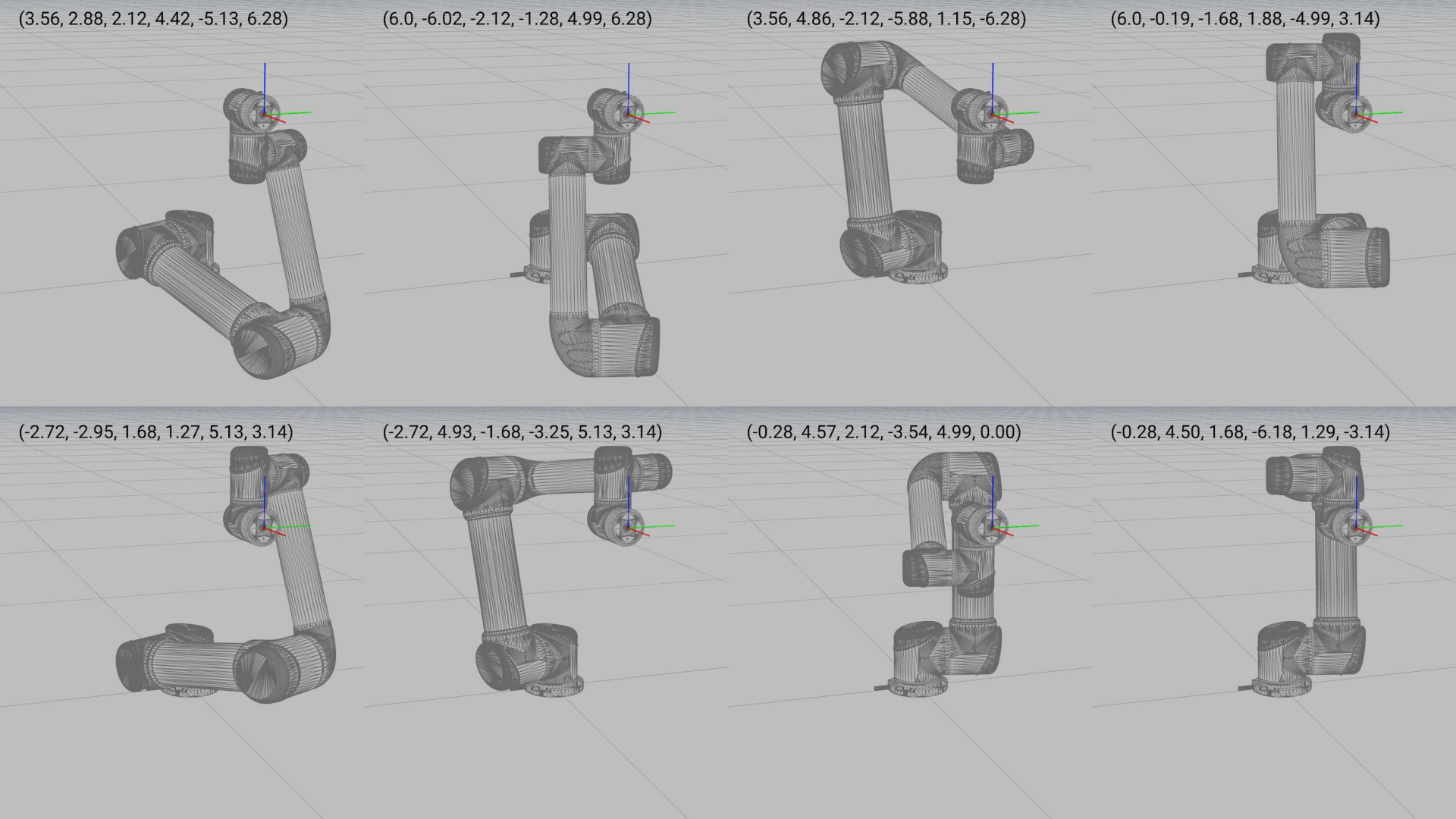
Update

https://tinyurl.com/swinburne21

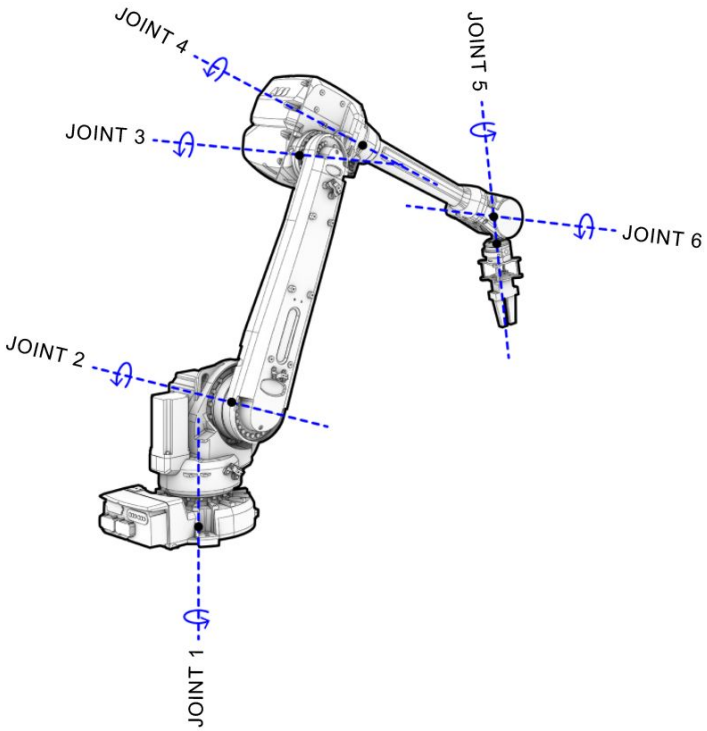
forward kinematics

Joint vs Cartesian space



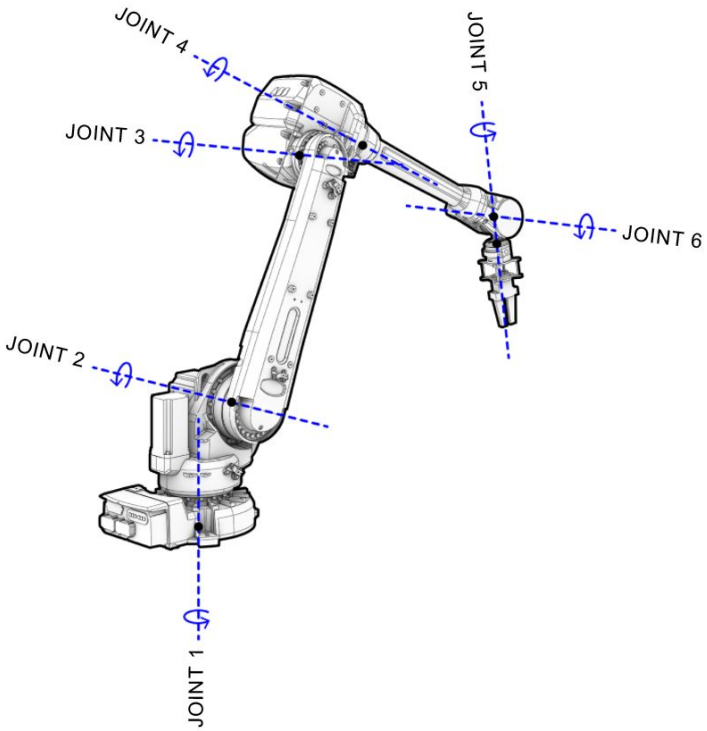

(3.56, 2.88, 2.12, 4.42, -5.13, 6.28)
(6.0, -6.02, -2.12, -1.28, 4.99, 6.28)
(3.56, 4.86, -2.12, -5.88, 1.15, -6.28)
(6.0, -0.19, -1.68, 1.88, -4.99, 3.14)
(-2.72, -2.95, 1.68, 1.27, 5.13, 3.14)
(-2.72, 4.93, -1.68, -3.25, 5.13, 3.14)
(-0.28, 4.57, 2.12, -3.54, 4.99, 0.00)
(-0.28, 4.50, 1.68, -6.18, 1.29, -3.14)

Configuration




```python
from compas.robots import Joint

print(Joint.REVOLUTE)
print(Joint.PRISMATIC)
print(Joint.FIXED)
```

Configuration




```
from math import pi
from compas_fab.robots import Configuration

values = [1.5, pi]
types = [Joint.PRISMATIC, Joint.REVOLUTE]
config = Configuration(values, types)

config =
Configuration.from_revolute_values([pi/2, 0., 0.,
pi/2, pi, 0])

config =
Configuration.from_prismatic_and_revolute_values(
[8.312], [pi/2, 0., 0., 0., 2*pi, 0.8])
```

Forward Kinematics

```
from compas.robots import LocalPackageMeshLoader
from compas.robots import RobotModel

loader = LocalPackageMeshLoader('models', 'ur_description')
model = RobotModel.from_urdf_file(loader.load_urdf('ur5.urdf'))

joints = dict(shoulder_pan_joint=0, shoulder_lift_joint=0, elbow_joint=0,
              wrist_1_joint=0, wrist_2_joint=0, wrist_3_joint=0)

frame = model.forward_kinematics(joints)
```

inverse kinematics

without backend?

Inverse Kinematics

```
from ur_kinematics import inverse_kinematics_ur5

loader = LocalPackageMeshLoader('models', 'ur_description')
model = RobotModel.from_urdf_file(loader.load_urdf('ur5.urdf'))

sols = inverse_kinematics_ur5(frame)
```

robot backends: ros

Robotic backends

`V-REP`
Robot simulator

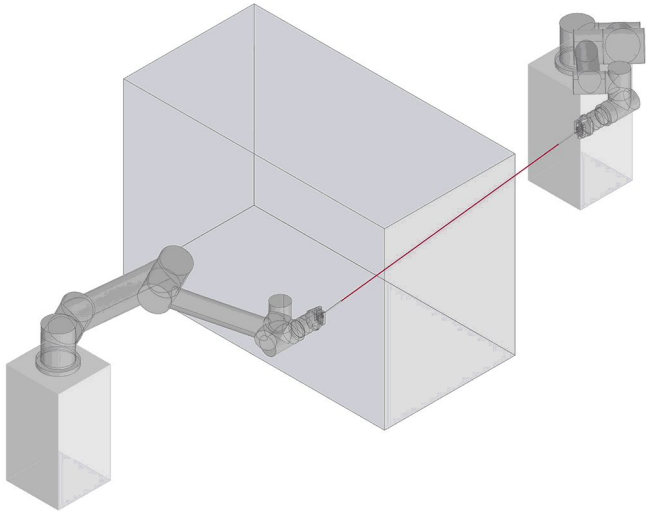
`ROS + MoveIt!`
Robot Operating System

`Pybullet`
Physics library

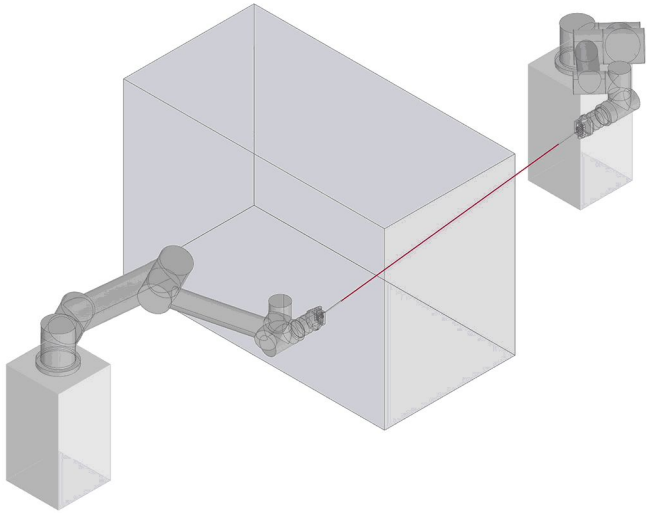
ROS

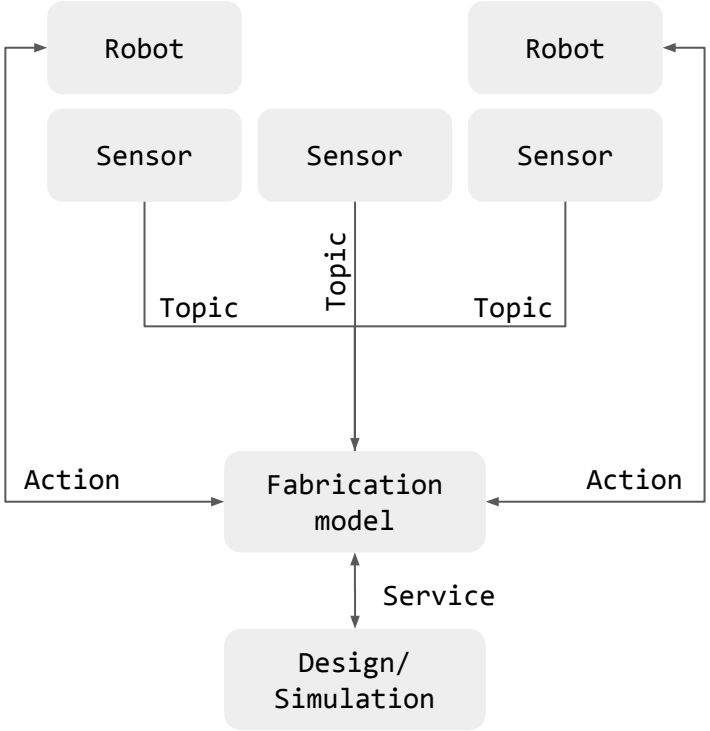
Open Source Robotics Foundation

Fabrication processes

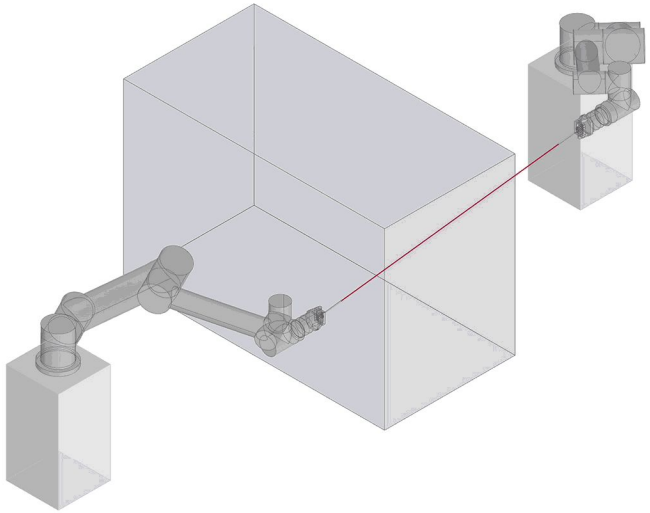


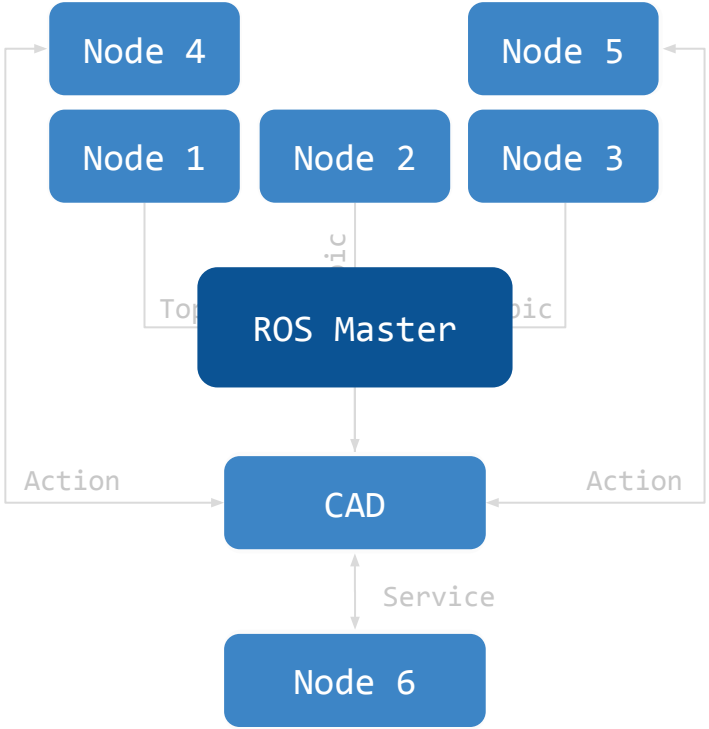
Fabrication processes



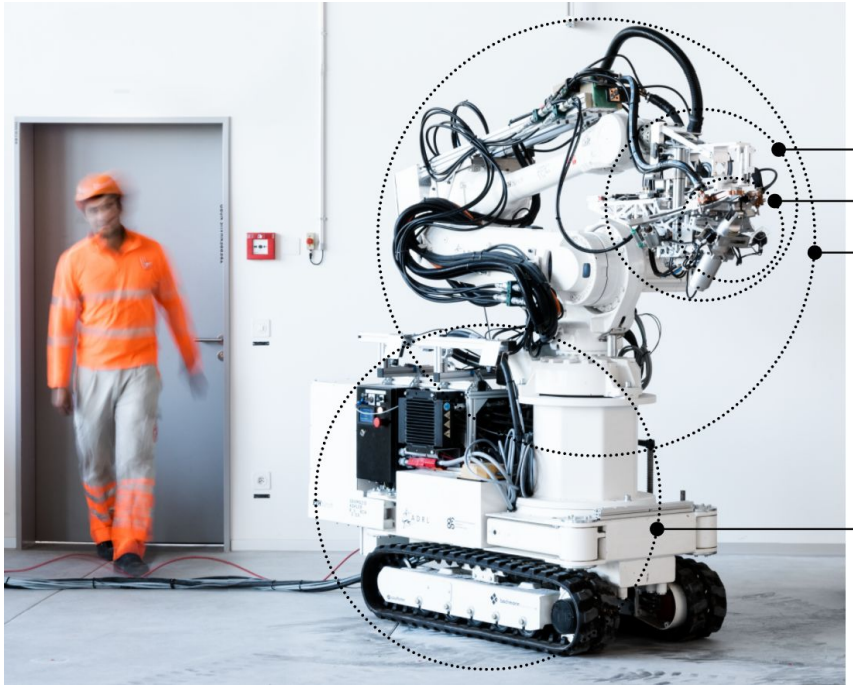


Fabrication processes





What is ROS?



ROS Node 1: End Effector

ROS Node 2: Localization

ROS Node 3: Robot Arm

ROS Node 4: Mobile Base

What is ROS?

plumbing

- Process management
- Interprocess communication
- Device drivers

tools

- Simulation
- Visualization
- Graphical UI
- Data logging

capabilities

- Control
- Planning
- Perception
- Mapping
- Manipulation

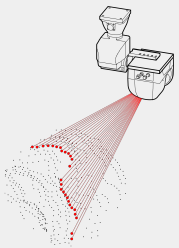
ecosystem

- Package organization
- Software distribution
- Community
- Documentation
- Tutorials

ROS Concepts

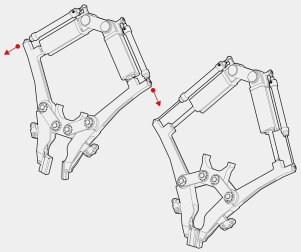
Topics

- Nodes communicate over topics
- **Publish/subscribe** model
- **One-way** data stream
- e.g: *robot states, sensor data*



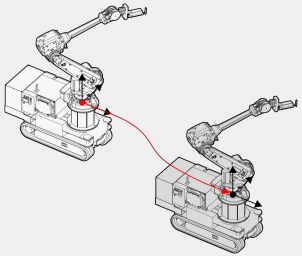
Services

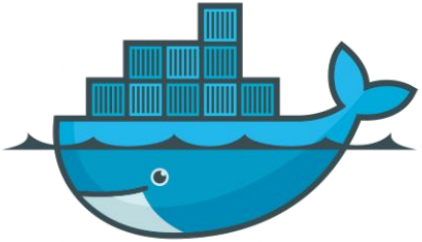
- **Blocking** call per request
- Request/response model
- Short trigger or calculation
- e.g: *calculate path, open gripper*

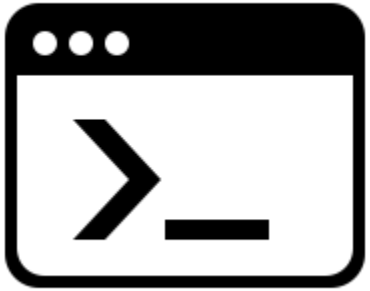


Actions

- **Non-blocking** requests
- Goal-oriented and cancellable
- Implemented with topics
- e.g: *navigation, motion execution*




docker



`docker-compose up -d`

Right-click → Compose Up

docker/first_steps

Verify connection to ROS

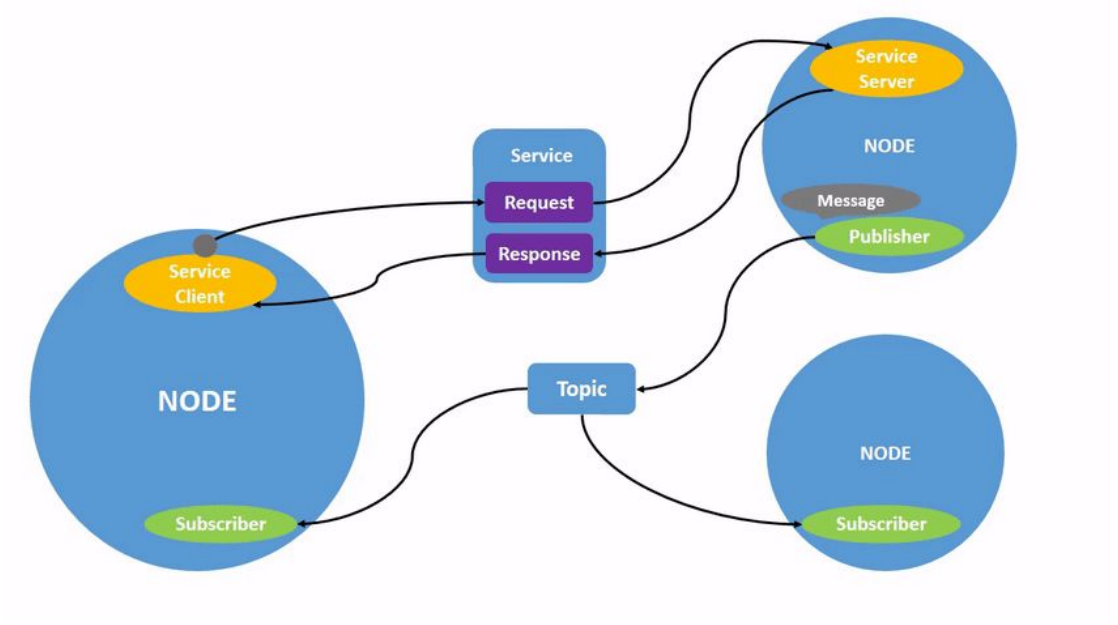
```python
from compas_fab.backends import RosClient

with RosClient('localhost') as client:
    print('Connected:', client.is_connected)
```

Verify connection to ROS

```
from compas_fab.backends import RosClient

with RosClient('192.168.99.100') as client:
    print('Connected:', client.is_connected)
```

interconnecting nodes

ROS communication model



Source: https://docs.ros.org/en/foxy/Tutorials/Understanding-ROS2-Nodes.html

Subscribe to topics

Subscribe to topic: **connect**

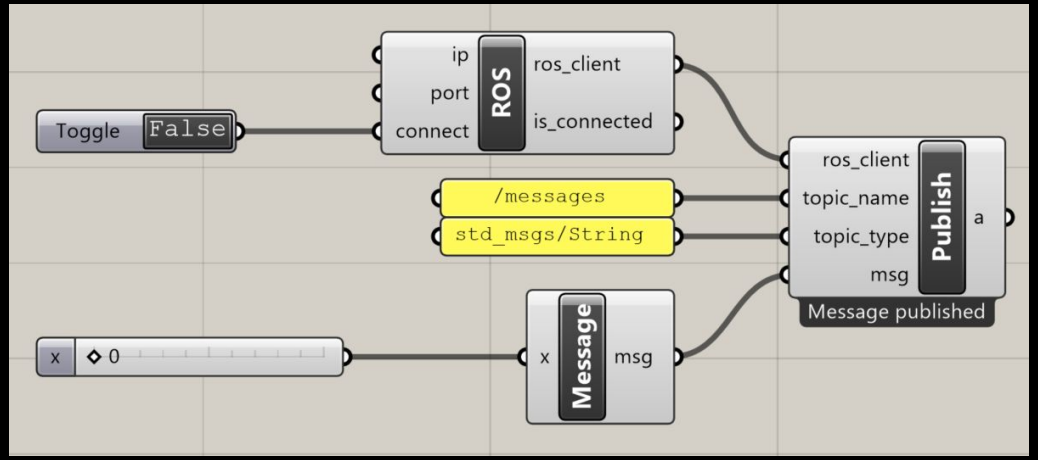
```
def receive_message(message):
    print(Received: ' + message['data'])

with RosClient('localhost') as client:
    listener = Topic(client, '/messages', 'std_msgs/String')
    listener.subscribe(receive_message)

    while client.is_connected:
        time.sleep(1)
```

Subscribe to topic: **subscribe**

```
def receive_message(message):
    print(Received: ' + message['data'])

with RosClient('localhost') as client:
    listener = Topic(client, '/messages', 'std_msgs/String')
    listener.subscribe(receive_message)

    while client.is_connected:
        time.sleep(1)
```

Publish to topics

Publish to topic **from Grasshopper**



Publish to topic: **connect**

```
with RosClient('localhost') as client:
    talker = Topic(client, '/messages', 'std_msgs/String')
    talker.advertise()

    while client.is_connected:
        talker.publish({'data': 'Hello'})
        time.sleep(1)
```

Publish to topic: **advertise**

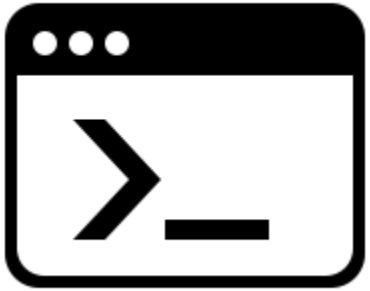
```
with RosClient('localhost') as client:
    talker = Topic(client, '/messages', 'std_msgs/String')
    talker.advertise()

    while client.is_connected:
        talker.publish({'data': 'Hello'})
        time.sleep(1)
```

Publish to topic: **publish**

```
with RosClient('localhost') as client:
    talker = Topic(client, '/messages', 'std_msgs/String')
    talker.advertise()

    while client.is_connected:
        talker.publish({'data': 'Hello'})
        time.sleep(1)
```

planning with moveit

docker/ur5-planner



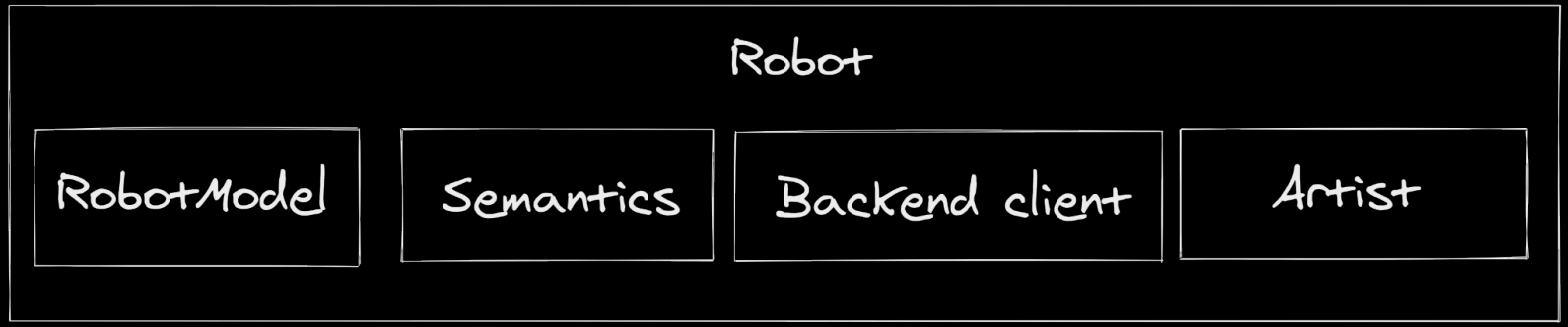
`docker-compose up -d`

Right-click → Compose Up

Loading ROS robot

```
# Load robot and its geometry
with RosClient('localhost') as ros:
    robot = ros.load_robot(load_geometry=True)
    robot.info()
```

compas_fab.robots



`compas.robots.RobotModel`

Describes the kinematics, linkage geometry and dynamics of a robot cell.

`compas_fab.robots.Robot`

Integrates a robot model, additional semantic information for planning, a backend client and artist instance.

Planning

MoveIt!

Kinematics

Motion planning

Planning scene

Forward Kinematics

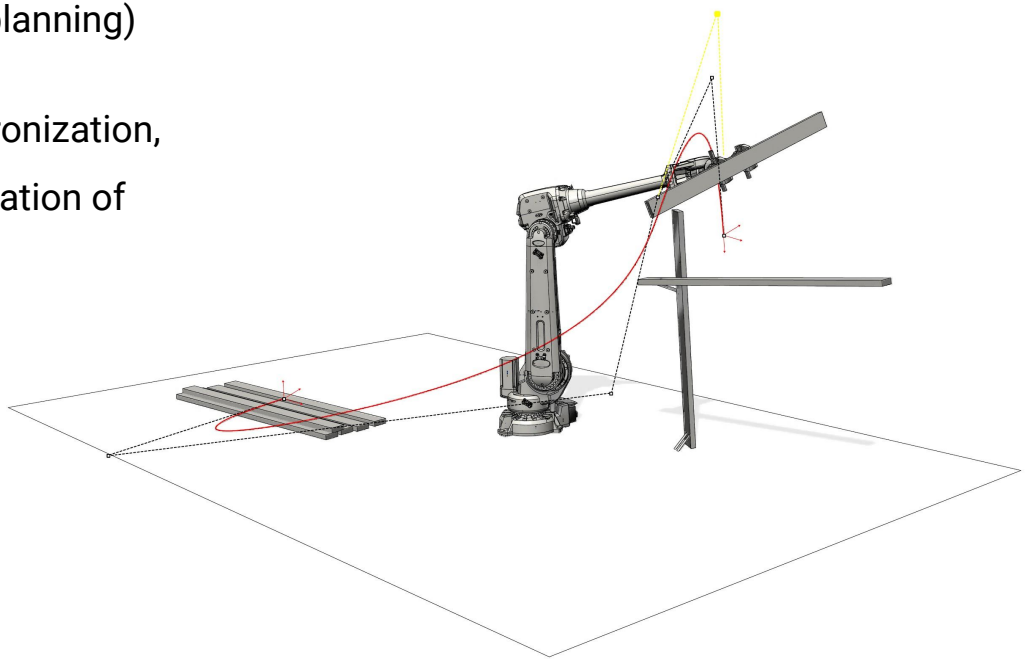
```python
from compas_fab.backends import RosClient
from compas_fab.robots import Configuration

with RosClient() as client:
    robot = client.load_robot()

    configuration = Configuration.from_revolute_values([-3.14, 0.0, 0.0, 0.0, 0.0, 0.0])

    frame_WCF = robot.forward_kinematics(configuration)
```

Inverse Kinematics

```python
from compas.geometry import Frame
from compas_fab.backends import RosClient

with RosClient() as client:
    robot = client.load_robot()

    frame_WCF = Frame([0.3, 0.1, 0.5], [1, 0, 0], [0, 1, 0])
    start_configuration = robot.zero_configuration()

    configuration = robot.inverse_kinematics(frame_WCF, start_configuration)
```

Motion planning

- Collision checking (= path planning)
- Trajectory checking (synchronization, consider speed and acceleration of moving objects)



Motion planning

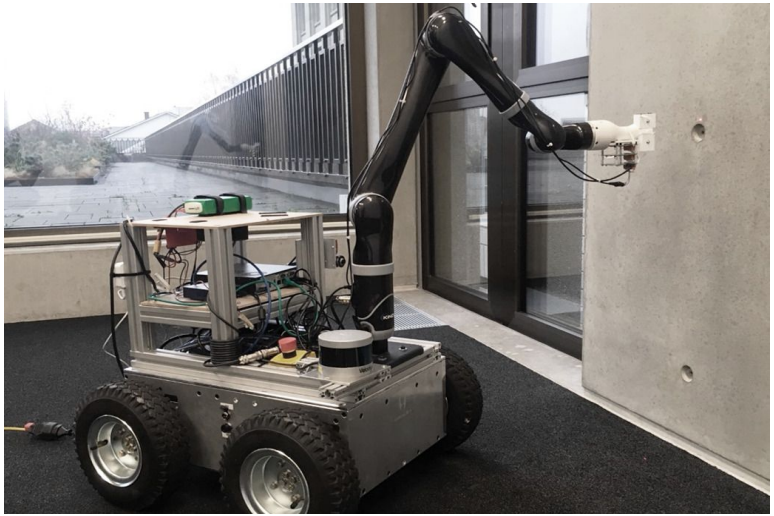
Collision checks

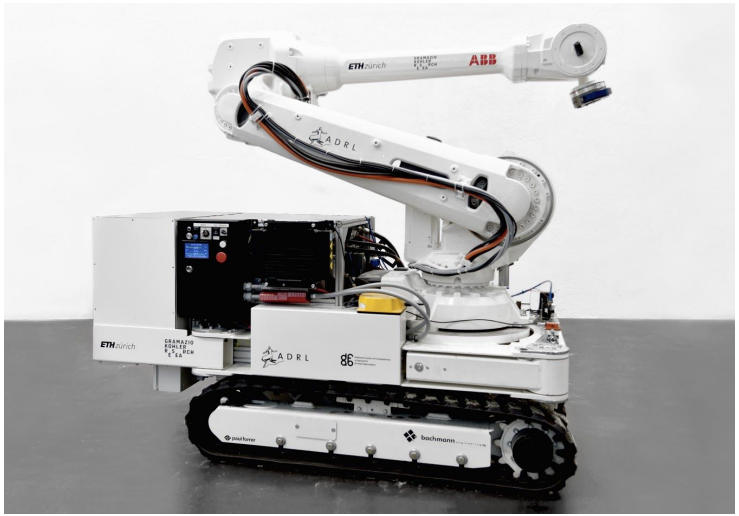
- Intricate positions (spatial assembly)
- Multiple robots working closely

Motion planning

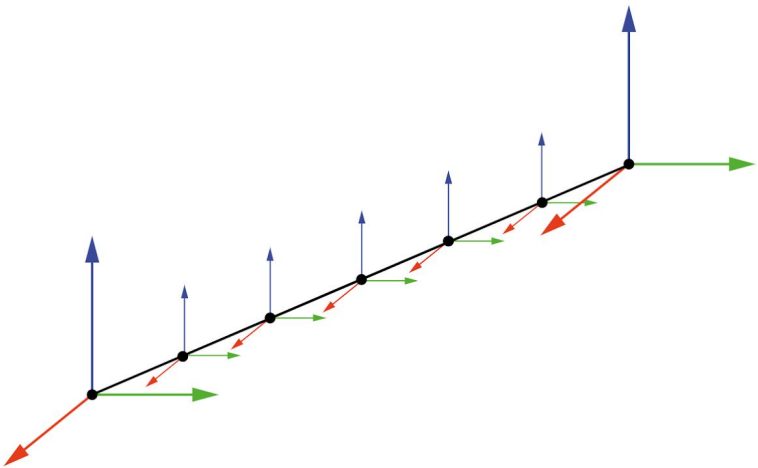
Trajectory checks

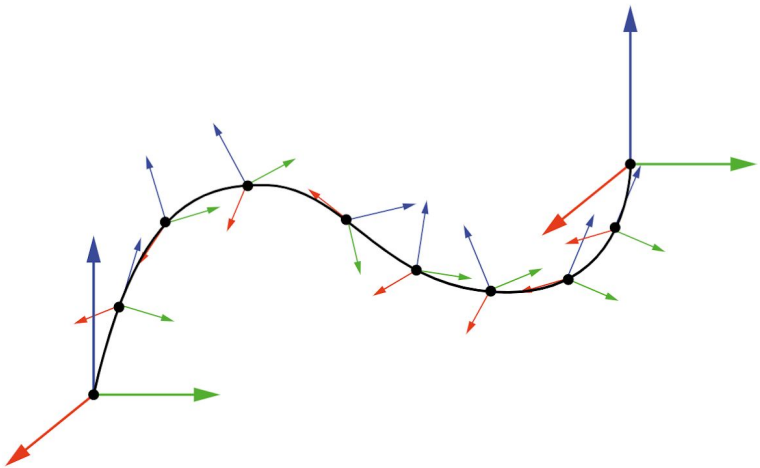
- Synchronization
- Continuous processes





Cartesian motion vs free-space motion





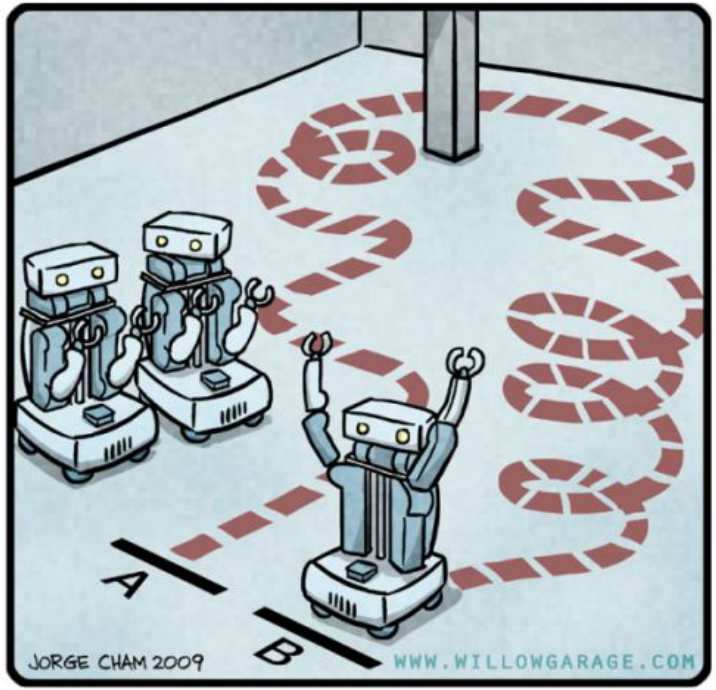
Plan cartesian motion

```python
from compas.geometry import Frame
from compas_fab.backends import RosClient
from compas_fab.robots import Configuration

with RosClient() as client:
    robot = client.load_robot()

    frames = []
    frames.append(Frame((0.29, 0.39, 0.50), (0, 1, 0), (0, 0, 1)))
    frames.append(Frame((0.51, 0.28, 0.40), (0, 1, 0), (0, 0, 1)))

    start_configuration = Configuration.from_revolute_values((3.14, 0.0, 0.0, 0.0, 0.0, 0.0))

    trajectory = robot.plan_cartesian_motion(frames, start_configuration)
```

Plan motion

```python
from compas.geometry import Frame
from compas_fab.backends import RosClient
from compas_fab.robots import Configuration

with RosClient() as client:
    robot = client.load_robot()

    frame = Frame([0.4, 0.3, 0.4], [0, 1, 0], [0, 0, 1])
    start_configuration = Configuration.from_revolute_values((3.14, 0.0, 0.0, 0.0, 0.0, 0.0))

    goal_constraints = robot.constraints_from_frame(frame,
                                                    tolerance_position=0.001,
                                                    tolerance_axes=[0.01, 0.01, 0.01])

    trajectory = robot.plan_motion(goal_constraints, start_configuration)
```

R.O.B.O.T. Comics



"HIS PATH-PLANNING MAY BE SUB-OPTIMAL, BUT IT'S GOT FLAIR."

Defining constraints

- **`JointConstraint`**
  Constrains the value of a joint to be within a certain bound.

- **`OrientationConstraint`**
  Constrains a link to be within a certain orientation.

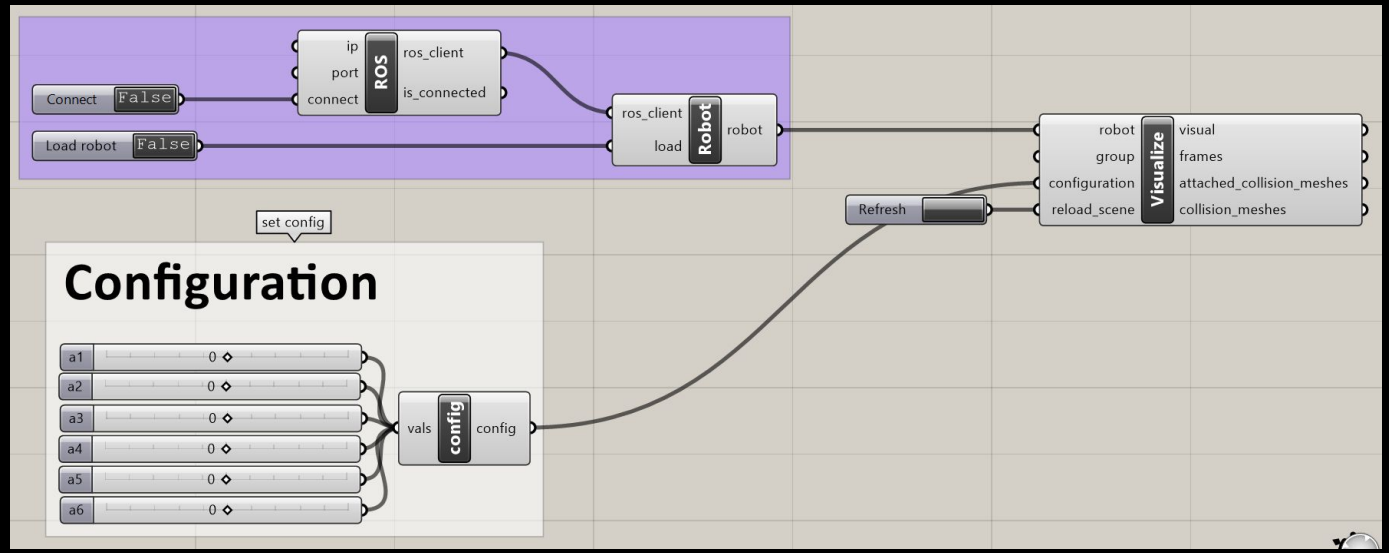
- **`PositionConstraint`**
  Constrains a link to be within a certain bounding volume.

Constraints

```
frame = Frame([0.4, 0.3, 0.4], [0, 1, 0], [0, 0, 1])
tolerance_position = 0.001
tolerance_axes = [math.radians(1)] * 3

start_configuration = Configuration.from_revolute_values([-0.042, 4.295, 0, -3.327, 4.755, 0.])
group = robot.main_group_name

# create goal constraints from frame
goal_constraints = robot.constraints_from_frame(frame,
                                                Tolerance_position,
                                                Tolerance_axes,
                                                group)
```

scene and end effectors

Load planning scene in Grasshopper



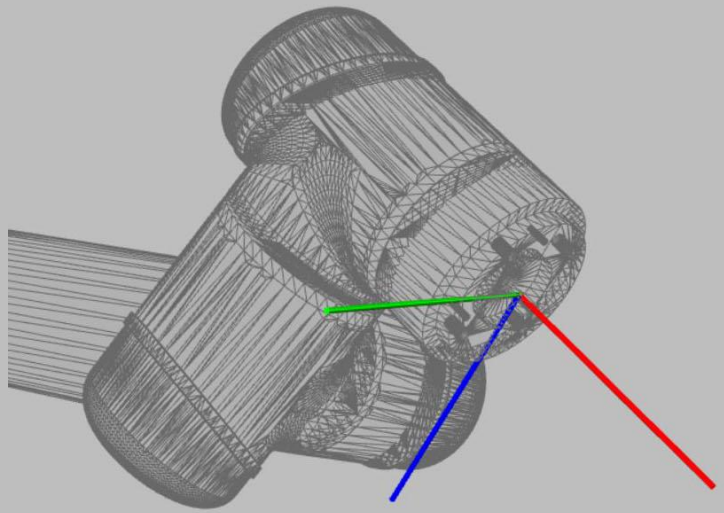
Planning scene operations

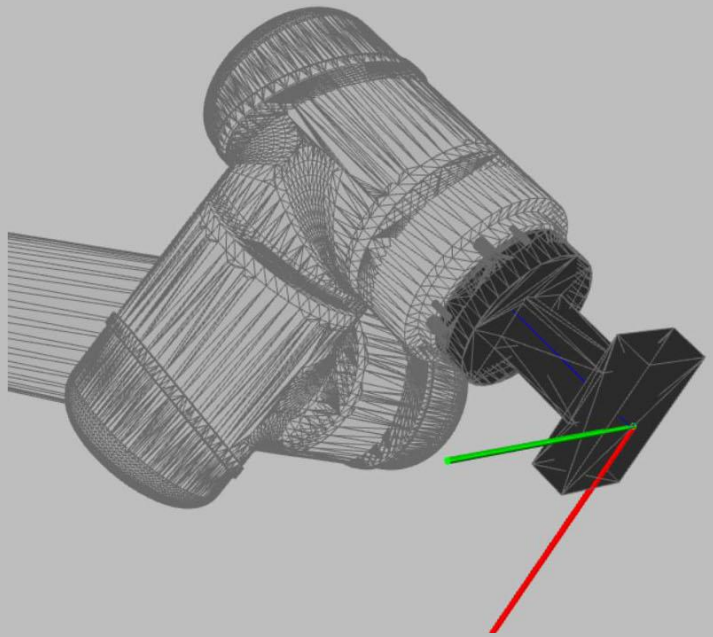
Add/append/remove collision meshes (i.e. obstacles) and add/remove attached collision meshes (i.e. meshes attached to the end effector).

- **Usage:**

```
scene = compas_fab.robots.PlanningScene(robot)
scene.add_collision_mesh(..)
scene.remove_collision_mesh(..)
scene.append_collision_mesh(..)

scene.add_attached_collision_mesh(..)
scene.remove_attached_collision_mesh(..)
scene.attach_collision_mesh_to_robot_end_effector(..)
```

Attach **tool**





Attach/detach tools

```
mesh = Mesh.from_stl(os.path.join(HERE, 'vacuum_gripper.stl'))
frame = Frame([0.07, 0, 0], [0, 0, 1], [0, 1, 0])
tool = Tool(mesh, frame)

with RosClient('localhost') as client:
    robot = client.load_robot()
    scene = PlanningScene(robot)

    # Attach the tool
    robot.attach_tool(tool)
    scene.add_attached_tool()

    # now we can convert frames at robot's tool tip and flange
    frames_tcf = [Frame((-0.309, -0.046, -0.266), (0.276, 0.926, -0.256), (0.879, -0.136, 0.456))]
    frames_tcf0 = robot.from_tcf_to_t0cf(frames_tcf)
```

Planning with end-effectors

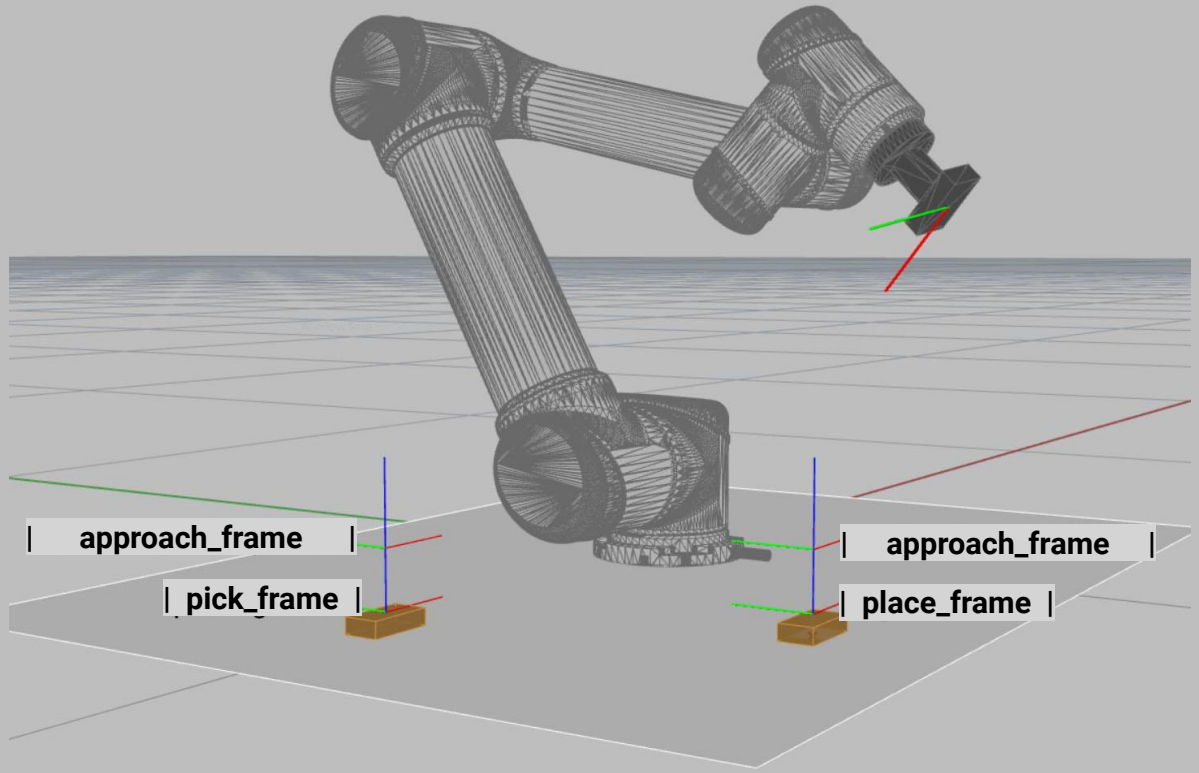
```
with RosClient('localhost') as client:
    robot = client.load_robot()
    scene = PlanningScene(robot)

    # 1. Set tool
    robot.attach_tool(tool)
    scene.add_attached_tool()

    # 2. Define start configuration
    start_configuration = Configuration.from_revolute_values([-5.961, 4.407, -2.265, 5.712, 1.571, -2.820])

    # 3. Define frames
    picking_frame = Frame([-0.43, 0, element_height], [1, 0, 0], [0, 1, 0])
    approach_picking_frame = Frame([-0.43, 0, element_height + approach_distance], [1, 0, 0], [0, 1, 0])
    frames = [picking_frame, approach_picking_frame]

    # 4. Convert frames to tool0_frames
    frames_tool0 = robot.from_tcf_to_t0cf(frames)
    trajectory = robot.plan_cartesian_motion(frames_tool0, start_configuration, options=dict(max_step=0.01))
```

pick and place

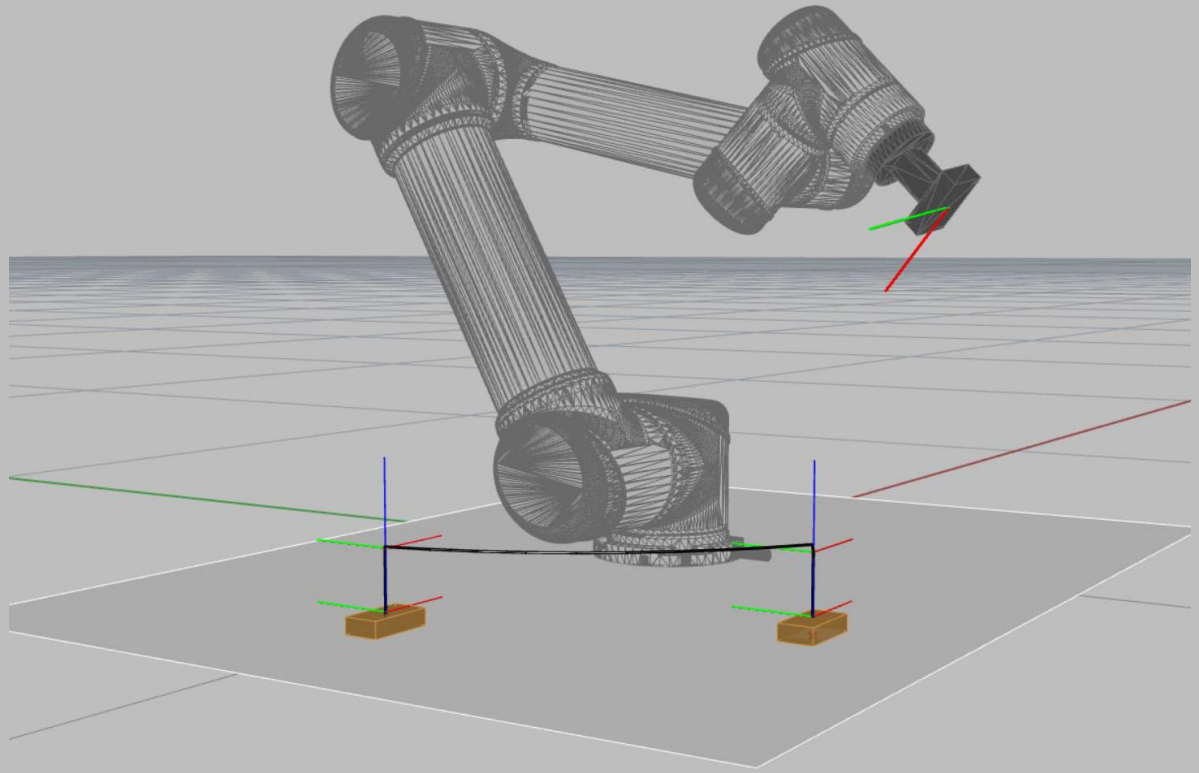
Pick and **Place**

1. Pick (cartesian plan)
2. Move (free-space plan)
3. Place (cartesian plan)

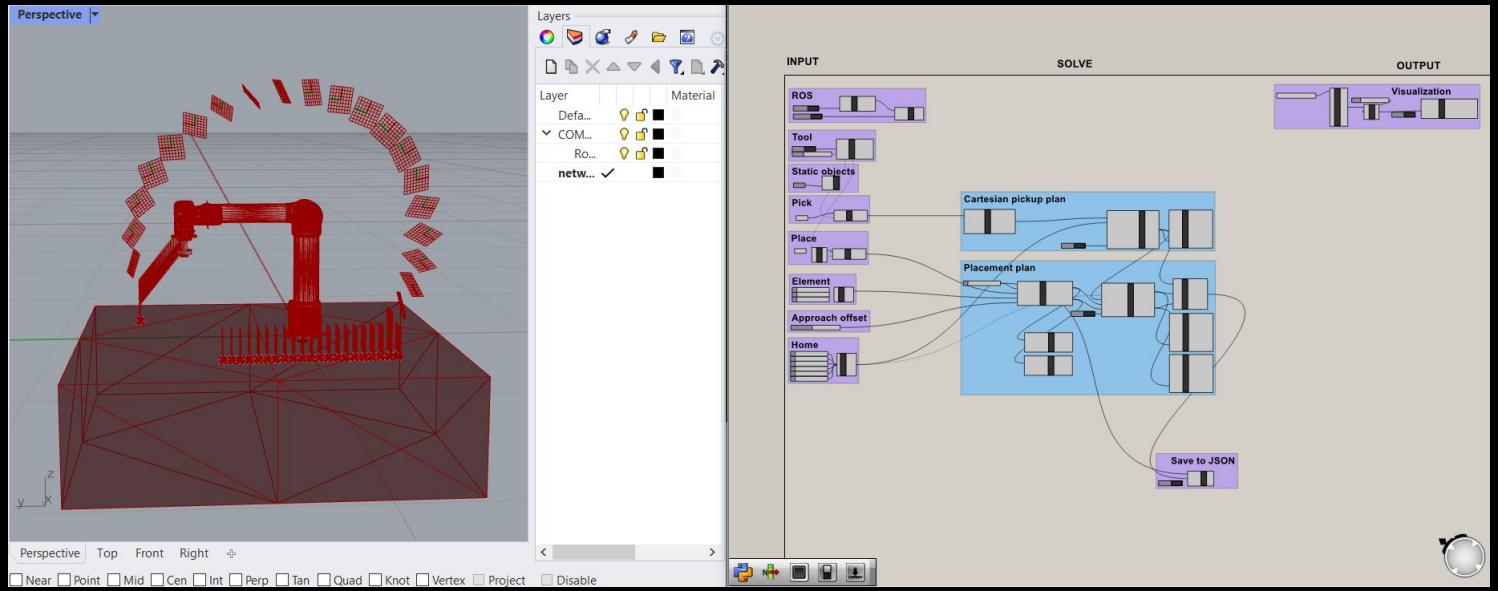
Pick and **Place**



Pick and **Place**

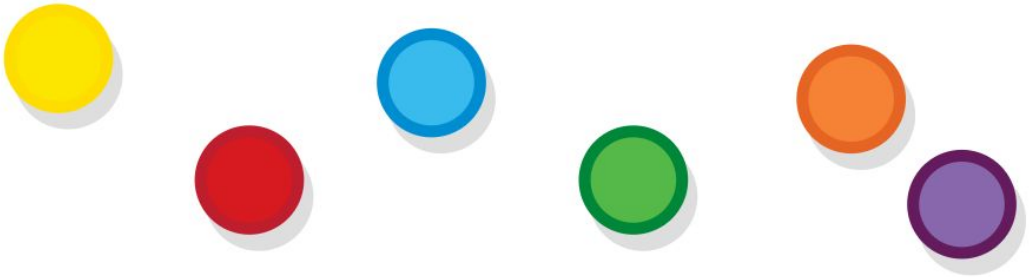


Pick and Place

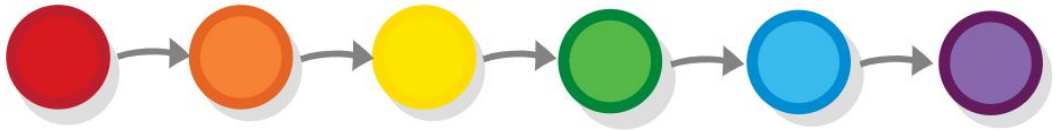


graphs

Sets

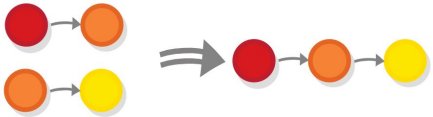


Linear order



```python
@functools.total_ordering
class BoxComparer(object):
    def __init__(self, box, *args):
        self.box = box

    def __eq__(self, other):
        return self.box.data == other.box.data

    def __lt__(self, other):
        return self.box.dimensions < other.box.dimensions
```

Reflexivity
Each object has to be bigger or equal to itself.



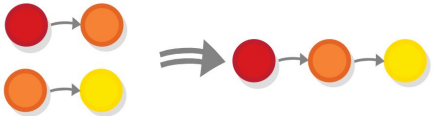
Transitivity
If A is bigger than B, and B is bigger than C, then A is bigger than C.

Antisymmetry
The order function cannot give contradictory results for the opposite pair.
Eg. `x <= y` and `y <= x` only iff `x == y`

Totality
All elements should be comparable to each other

Source: Boris Marinov, https://github.com/boris-marinov/category-theory-illustrated

Reflexivity
Each object has to be bigger or equal to itself.



Transitivity
If A is bigger than B, and B is bigger than C, then A is bigger than C.

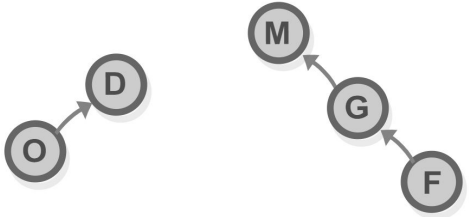
Antisymmetry
The order function cannot give contradictory results for the opposite pair.
Eg. `x <= y` and `y <= x` only iff `x == y`

Totality
All elements should be comparable to each other

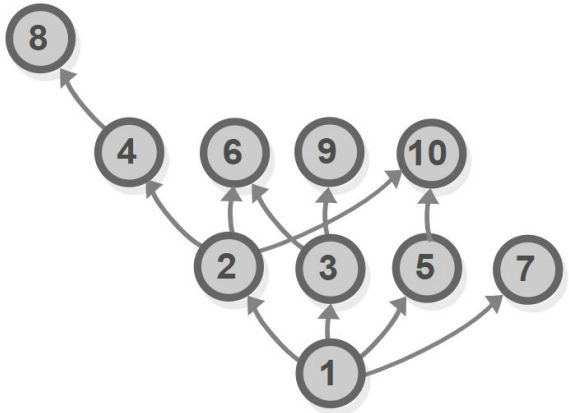
Source: Boris Marinov, https://github.com/boris-marinov/category-theory-illustrated

ETH zürich

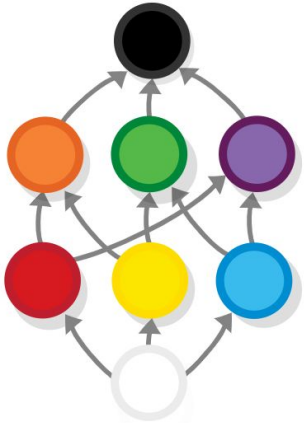
Partial order



Linearly-ordered subsets



Partial order



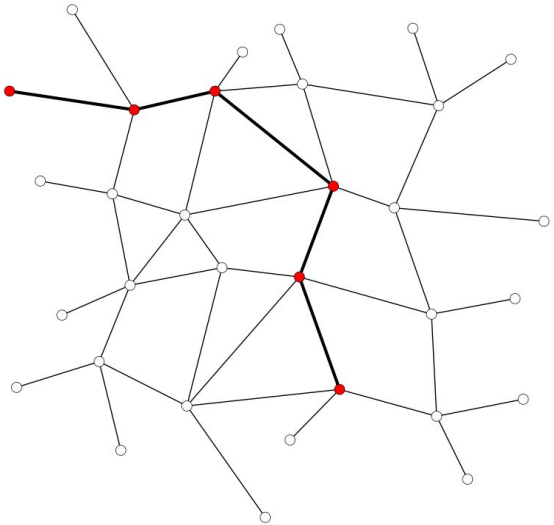
Lattice

Source: Boris Marinov, https://github.com/boris-marinov/category-theory-illustrated

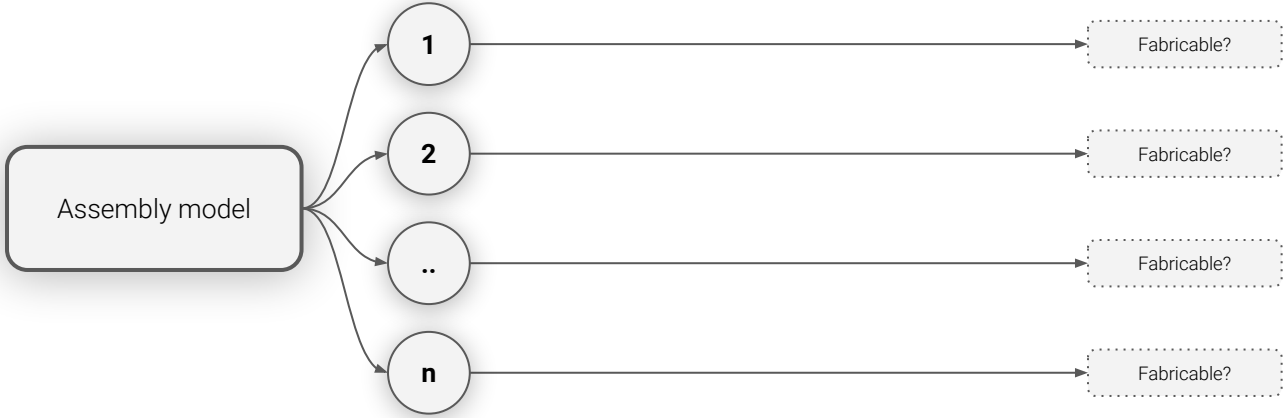
Network

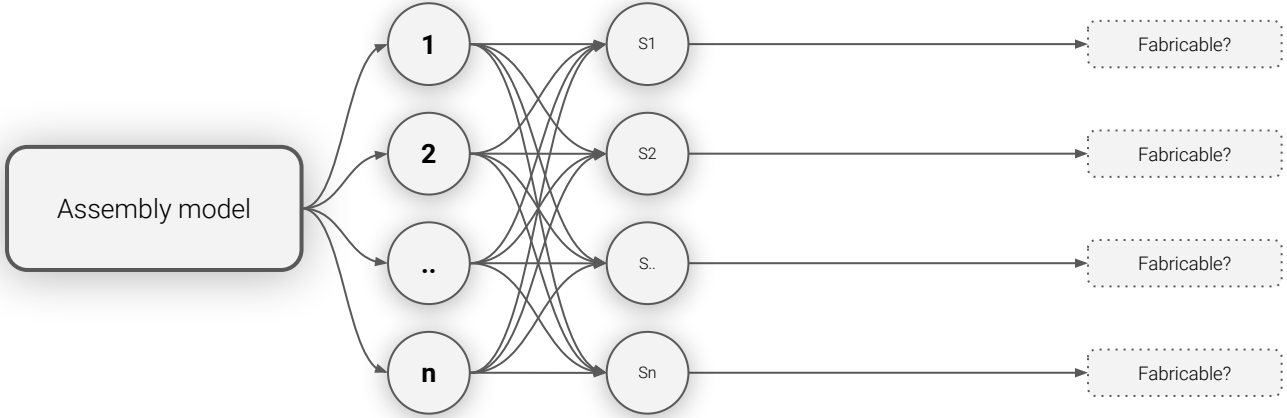
compas.datastructures

- directed edge graph data structure
- graph: topological
- network: geometric implementation of graph
- edge, node, degree, neighbors
- custom attributes
- networkx lossless conversion



```python
network = Network()

s = network.add_node(x=11, y=30, z=0, color=(000, 000, 000), text='black')

o = network.add_node(x=1., y=20, z=0, color=(255, 128, 000), text='orange')
g = network.add_node(x=11, y=20, z=0, color=(000, 255, 000), text='green')
p = network.add_node(x=21, y=20, z=0, color=(128, 000, 128), text='purple')

r = network.add_node(x=1., y=10, z=0, color=(255, 000, 000), text='red')
y = network.add_node(x=11, y=10, z=0, color=(255, 255, 000), text='yellow')
b = network.add_node(x=21, y=10, z=0, color=(000, 000, 255), text='blue')

w = network.add_node(x=11, y=00, z=0, color=(255, 255, 255), text='white')

network.add_edge(w, r)
network.add_edge(w, y)
network.add_edge(w, b)

network.add_edge(r, o)
network.add_edge(r, p)

# [..]
```

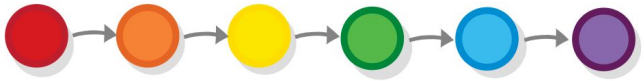

Assembly model
1
2
..
n
Fabricable?
Fabricable?
Fabricable?
Fabricable?


Assembly model
1
2
..
n
S1
S2
S..
Sn
Fabricable?
Fabricable?
Fabricable?
Fabricable?

Sequence types

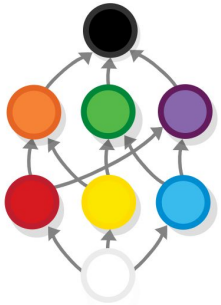
Total orders (fully linear sequences)

- Simple to describe
- Work for simple processes



Partial orders (e.g. dependency graph).

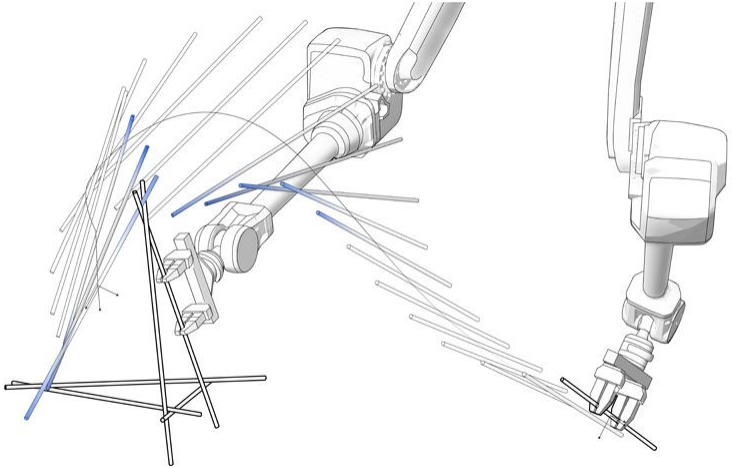
- Allow to express more advanced process (e.g. multiple robots in parallel)
- More involved to describe
- Broader selection of algorithms available



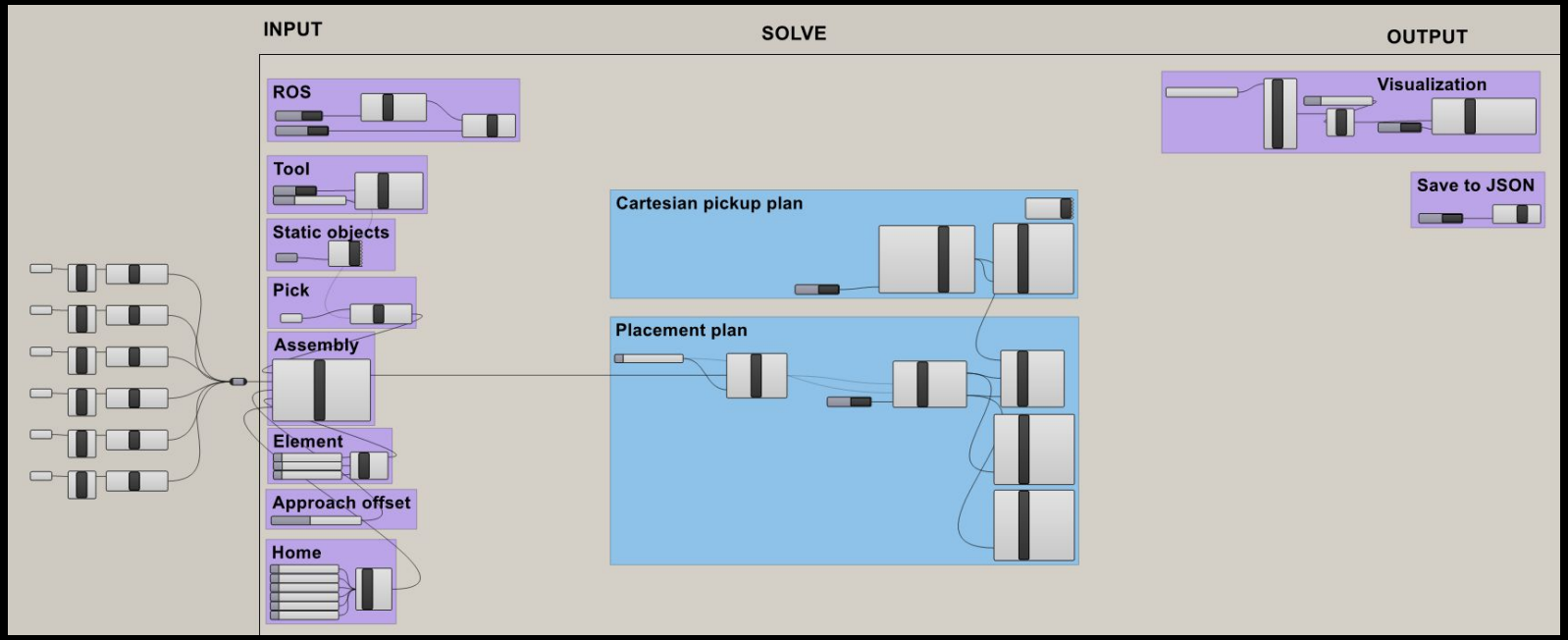
Impact of building sequence

Sequence affects fabricability in multiple ways:

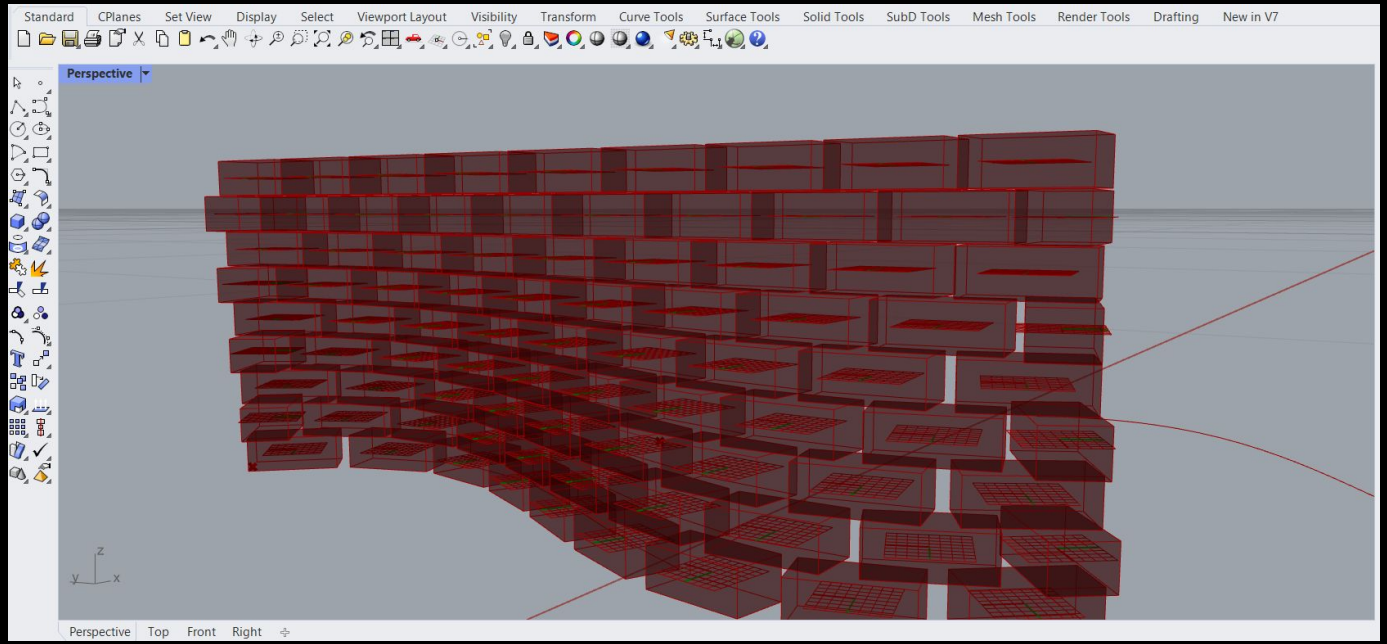
- Stability during fabrication
- Tolerance build-up
- Robotic accessibility
- Material behavior



Network-based Assembly planner



Tomorrow!


Standard
CPlanes
Set View
Display
Select
Viewport Layout
Visibility
Transform
Curve Tools
Surface Tools
Solid Tools
SubD Tools
Mesh Tools
Render Tools
Drafting
New in V7
Perspective
Perspective
Top
Front
Right

Day 2: ✅

Overview

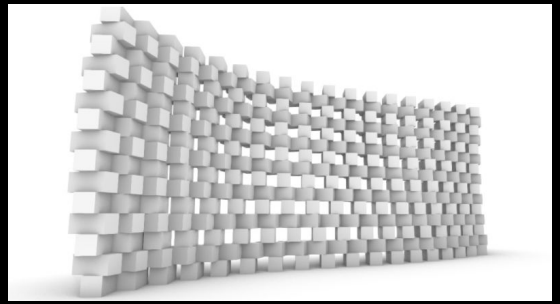
- ✅ COMPAS
- ✅ Robotic fundamentals
- ✅ Building blocks
- ✅ Processes

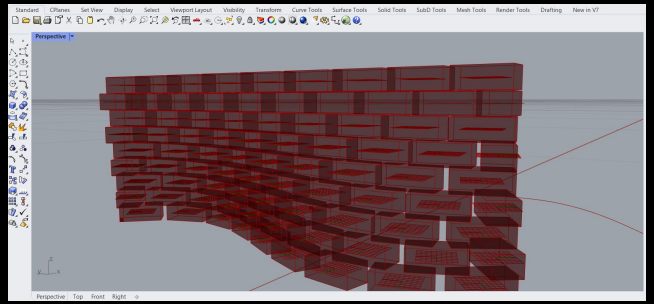
- ⬜ Robot control

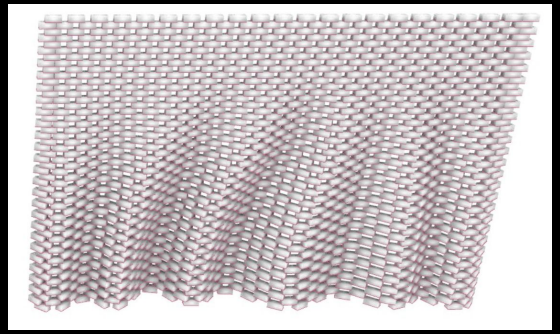
- ⬜ Build assembly
- ⬜ ProtoLab KUKA planner

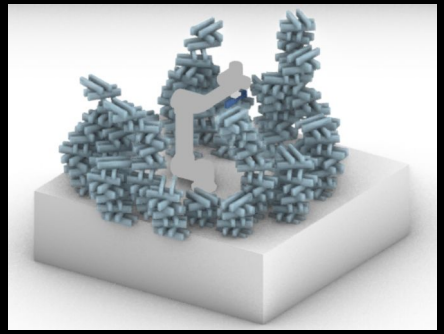
Q&A

Walls and voting 🧱 🥇

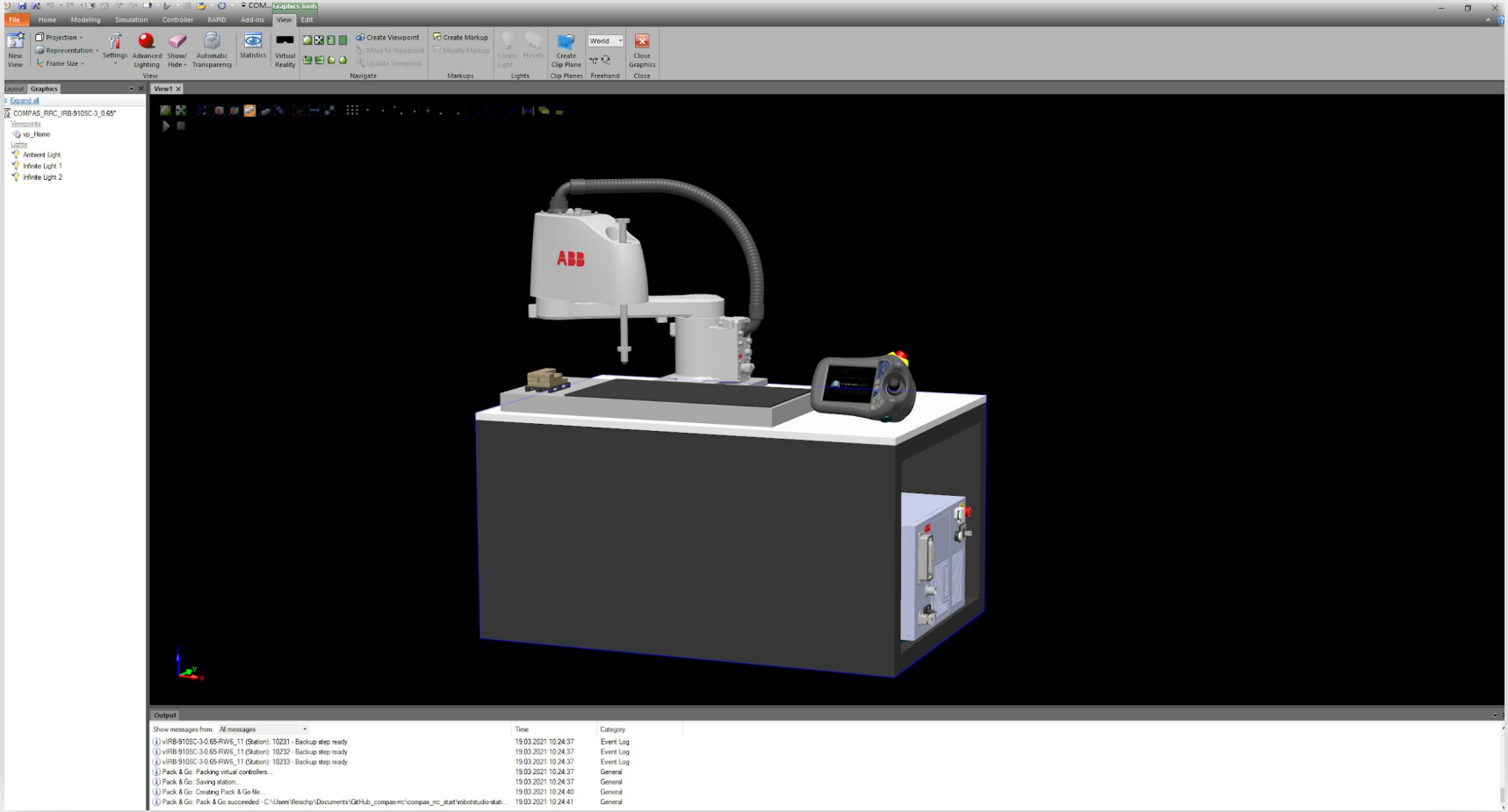








robot control


File
Home
Modeling
Simulation
Controller
RAPID
Add-Ins
View
Edit
Graphics Tools
New View
Projection
Representation
Frame Size
Settings
Advanced Lighting
Show/ Hide
Automatic Transparency
Statistics
Virtual Reality
Create Viewpoint
Move to Viewpoint
Update Viewpoint
Navigate
Create Markup
Modify Markup
Markups
Create Light
Presets
Lights
Create Clip Plane
Clip Planes
World
Freehand
Close Graphics
Close
Layout
Graphics
Expand all
COMPAS_RRC_IRB-910SC-3_0.65*
Viewpoints
vp_Home
Lights
Ambient Light
Infinite Light 1
Infinite Light 2
View 1
ABB
Output
Show messages from: All messages
Time
Category
vIRB-910SC-3-0.65-RW6_11 (Station): 10231 - Backup step ready 19.03.2021 10:24:37 Event Log
vIRB-910SC-3-0.65-RW6_11 (Station): 10232 - Backup step ready 19.03.2021 10:24:37 Event Log
vIRB-910SC-3-0.65-RW6_11 (Station): 10233 - Backup step ready 19.03.2021 10:24:37 Event Log
Pack & Go: Packing virtual controllers... 19.03.2021 10:24:37 General
Pack & Go: Saving station... 19.03.2021 10:24:37 General
Pack & Go: Creating Pack & Go file... 19.03.2021 10:24:40 General
Pack & Go: Pack & Go succeeded - C:\Users\fleischp\Documents\GitHub_compas-rrc\compas_rrc_start\robotstudio-stati... 19.03.2021 10:24:41 General

Search or jump to... / Pull requests Issues Marketplace Explore

compas-rrc / compas_rrc_start Private

Watch 2 Star 0 Fork 0

Code Issues Pull requests Actions Projects Wiki Security Insights Settings

main 1 branch 0 tags Go to file Add file Code

fleischp pack and go updated with black background b12dcbd 15 minutes ago 40 commits

.vscode	pack and go updated with black background	15 minutes ago
Python	first brick example created (new pack&go, new viewer and new python c...	3 days ago
docker	first structure done	11 days ago
robotstudio-extras	names changed agording to the course	11 days ago
robotstudio-stations	pack and go updated with black background	15 minutes ago
robotstudio-viewers	first brick example created (new pack&go, new viewer and new python c...	3 days ago
README.md	Update README.md	6 days ago

README.md

Welcome to COMPAS RRC

> Online control for ABB robots over a simple-to-use Python interface.

Requirements

- Windows 10 Pro 64-bit
- CPU 2 GHz
- Memory 8 GB
- Disk 10 GB SSD
- Screen 1920x1080

Installation

Anaconda

- Anaconda 3: Anaconda is an open source scientific Python distribution. With this tool, we can easily create a Python environment. Install Anaconda using default options.

Docker

- Docker Desktop: Docker is a virtualization platform. We use it to run Linux containers for ROS on Windows machines.
- After installation, it is required to enable "Virtualization" on the BIOS of the computer. Usually this requires rebooting your computer and pressing a vendor-specific key (`F2`, `F4`, `Del`, are typical options) to enter the BIOS.

About

No description, website, or topics provided.

Readme

Releases

No releases published
Create a new release

Packages

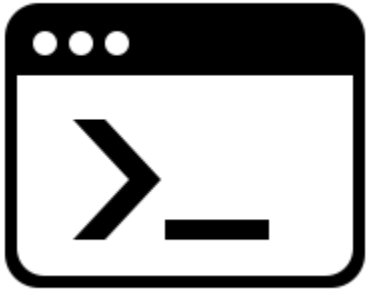
No packages published
Publish your first package

Contributors 2

fleischp Philippe Fleischmann
gonzalocasas Gonzalo Casas

Languages

Python 100.0%



`docker-compose up -d`

Right-click → Compose Up

`docker/irb910sc-planner`

Planner & RRC driver for ABB IRB910SC

Traditional programming (offline control)

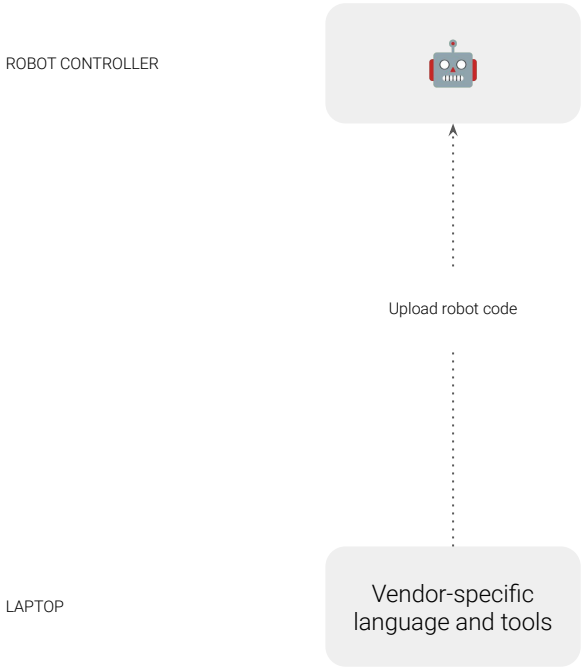


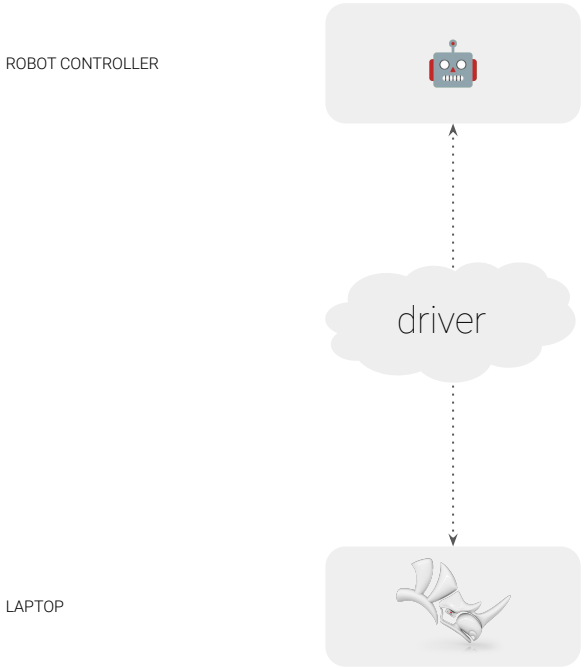
ABB: RAPID

Stäubli: VAL3

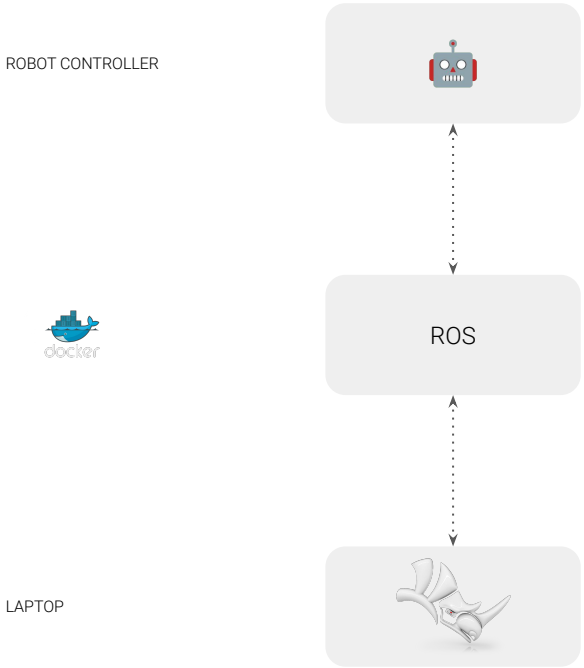
UR: URScript

Kuka: KRL

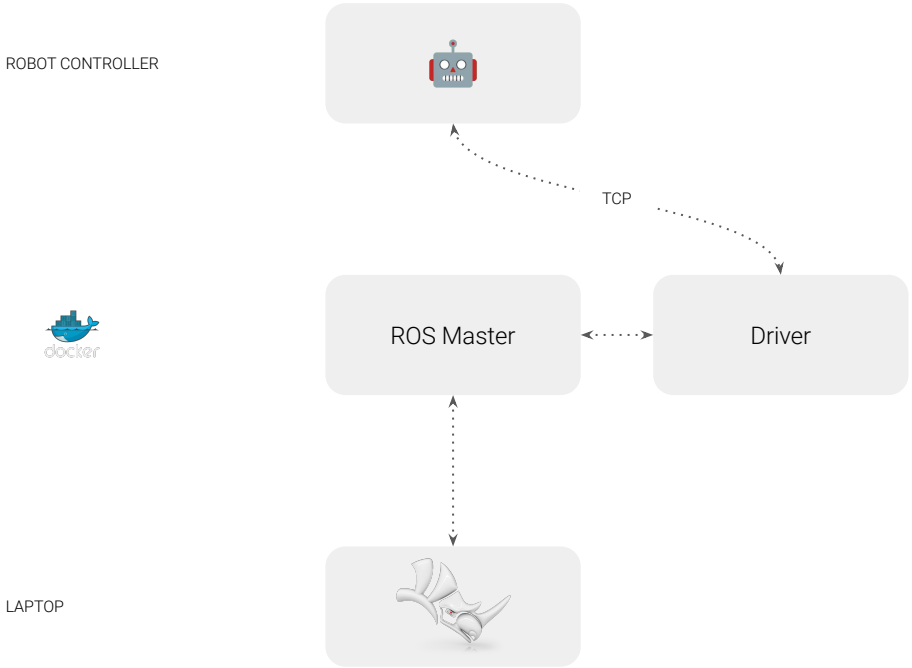
Our goal: online control



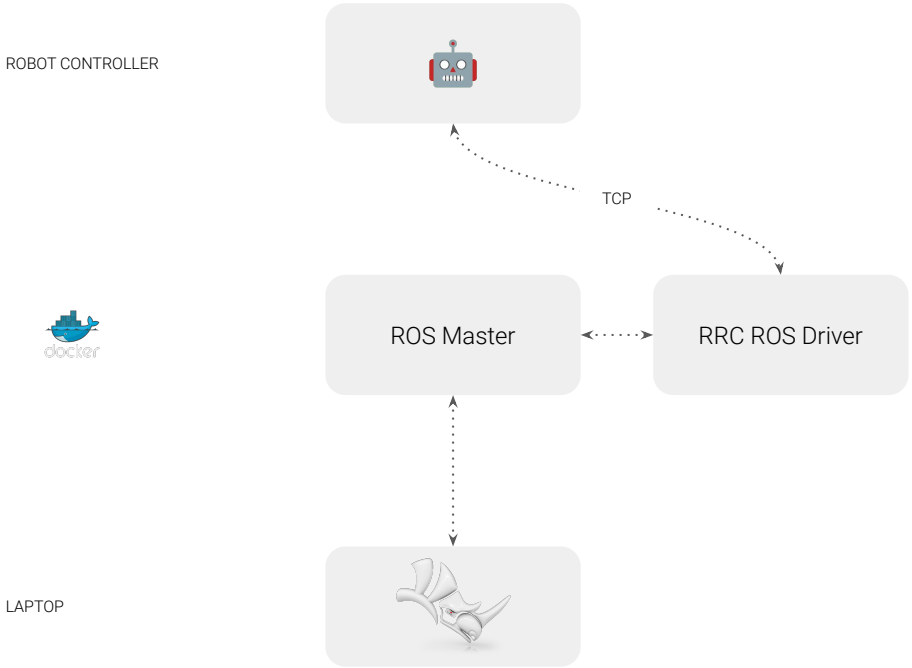
Overview: all the moving parts



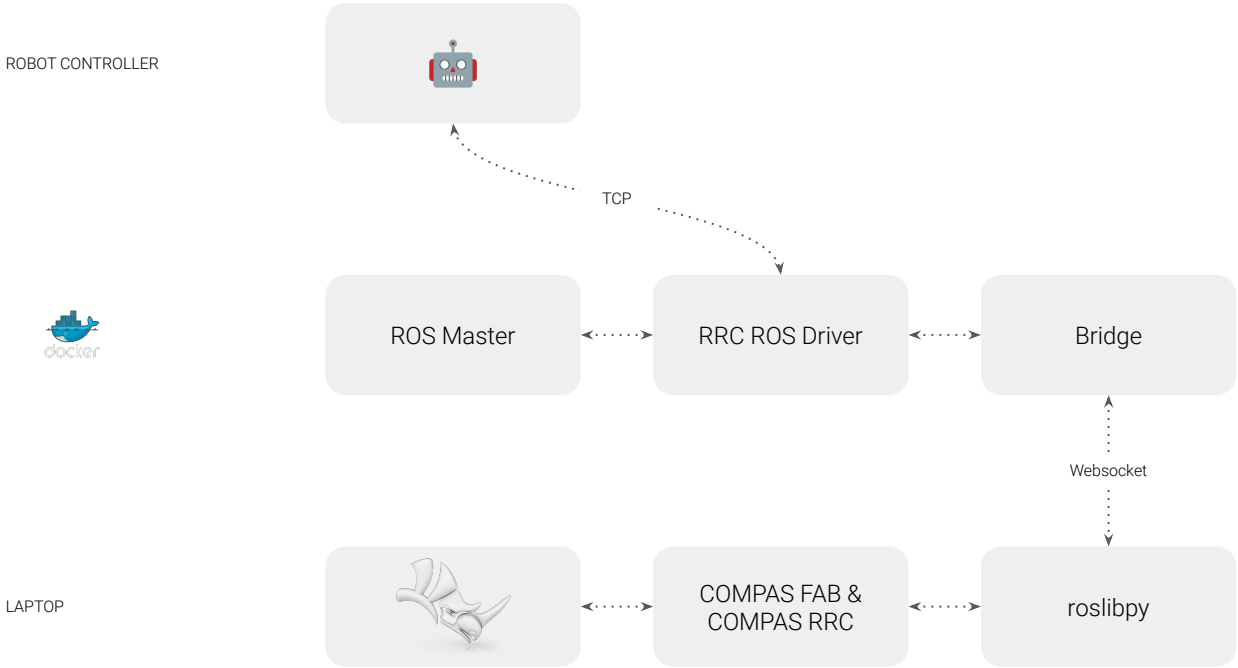
Overview: all the moving parts



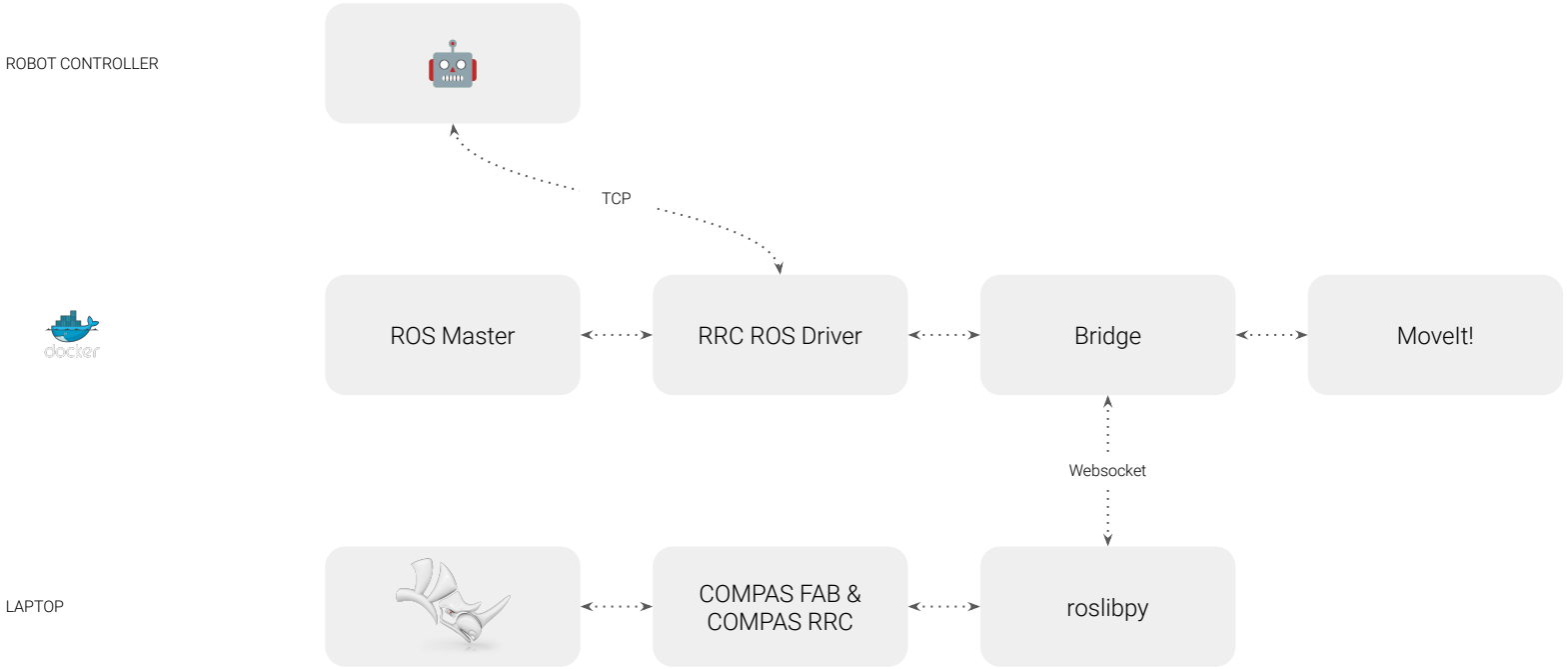
Overview: all the moving parts



Overview: all the moving parts



Overview: all the moving parts



Control

COMPAS RRC

Offline control

Online real-time control

Online non-real-time control

COMPAS RRC

COMPAS RRC

3D Concrete Printing

- Data streaming

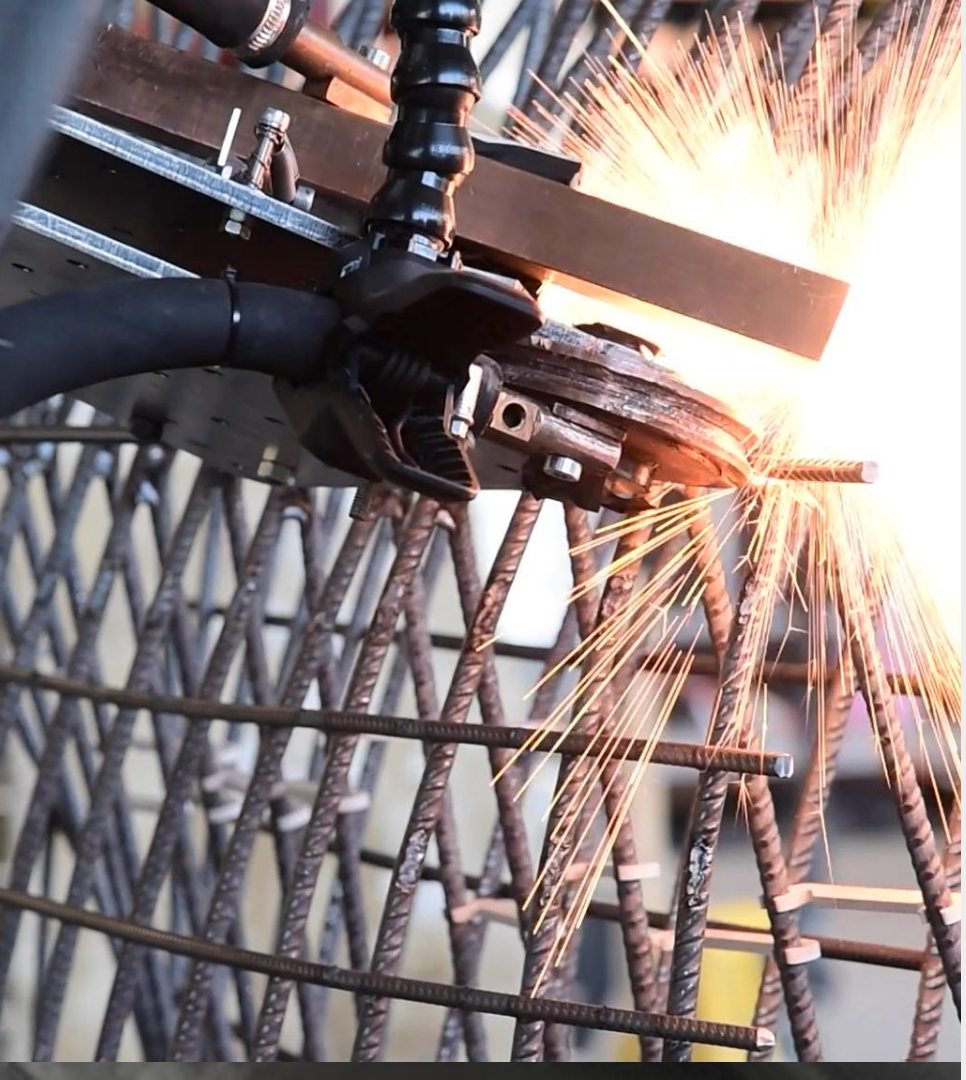
- More than 300’000 points
- More than 4 hours of printing

ETH zürich

WAAM

- Process Feedback

- Integrated Fronius welder
- Arc Weld PowerPac
- SmartTac



Mesh Mould

- Advanced Processing
- Slice positioning with
  - COMPAS RRC
- Rebar welding with
  - Externally Guided Motion

ETH zürich

COMPAS RRC

Features

Live communication

Multi-Tasking

Multi-Move

Multi-Controller

Multi-Location

COMPAS RRC

Communication

Send ~75ms

Send and Wait ~150ms

Send and Wait in the Future

Send and Subscribe ~75ms

COMPAS RRC

Instructions

Motions

Signals

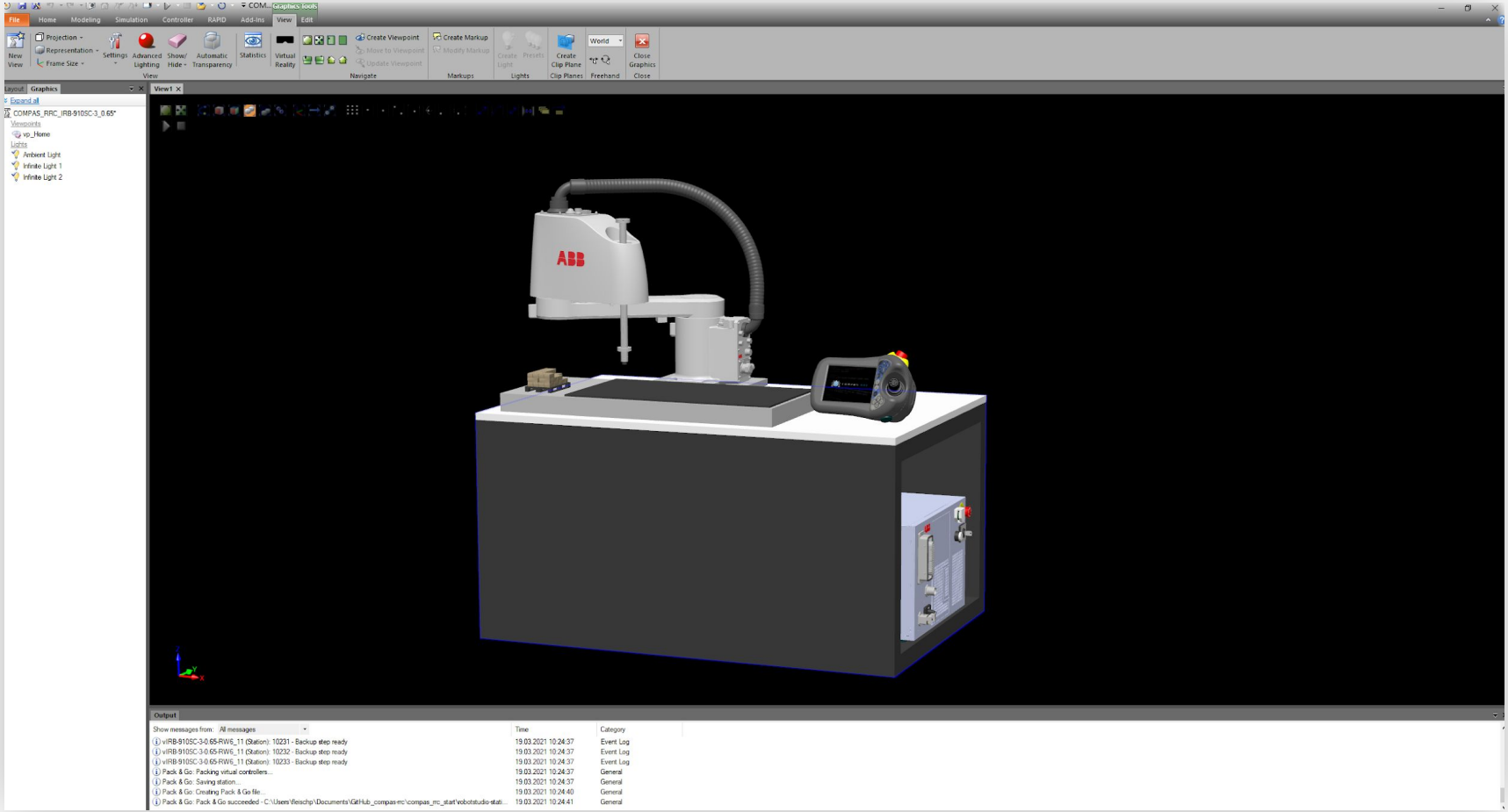
Basics

Utilities

Custom

ETH zürich

instructions overview


File
Home
Modeling
Simulation
Controller
RAPID
Add-Ins
View
Edit
Graphics Tools
New View
Projection
Representation
Frame Size
Settings
Advanced Lighting
Show/ Hide
Automatic Transparency
Statistics
Virtual Reality
Create Viewpoint
Move to Viewpoint
Update Viewpoint
Navigate
Create Markup
Modify Markup
Markups
Create Light
Presets
Lights
Create Clip Plane
Clip Planes
World
Freehand
Close Graphics
Close
Layout
Graphics
Expand all
COMPAS_RRC_IRB-910SC-3_0.65*
Viewpoints
vp_Home
Lights
Ambient Light
Infinite Light 1
Infinite Light 2
View 1
ABB
Output
Show messages from: All messages
Time
Category
vIRB-910SC-3-0.65-RW6_11 (Station): 10231 - Backup step ready 19.03.2021 10:24:37 Event Log
vIRB-910SC-3-0.65-RW6_11 (Station): 10232 - Backup step ready 19.03.2021 10:24:37 Event Log
vIRB-910SC-3-0.65-RW6_11 (Station): 10233 - Backup step ready 19.03.2021 10:24:37 Event Log
Pack & Go: Packing virtual controllers... 19.03.2021 10:24:37 General
Pack & Go: Saving station... 19.03.2021 10:24:37 General
Pack & Go: Creating Pack & Go file... 19.03.2021 10:24:40 General
Pack & Go: Pack & Go succeeded - C:\Users\fleischp\Documents\GitHub_compas-rrc\compas_rrc_start\robotstudio-stati... 19.03.2021 10:24:41 General

Hello world

```
ros = rrc.RosClient()
abb = rrc.AbbClient(ros, '/rob1')
done = abb.send_and_wait(rrc.PrintText('Welcome to COMPAS_RRC'))
```

Send instruction (non-blocking)

```
# Send an instruction without waiting for any kind of feedback
abb.send(rrc.PrintText('Hello.'))
```

Send instruction with feedback (blocking)

```
# Send and wait
done = abb.send_and_wait(rrc.PrintText('Sent with feedback.'))
```

Send instruction with feedback (non-blocking)

```
# Send and defer waiting
future = abb.send(rrc.PrintText('feedback',feedback_level=rrc.FeedbackLevel.DONE))

# Here you can do other stuff [..]

# Wait for feedback
done = future.result(timeout=3.0)
```

COMPAS RRC

Basics

Set Tool

Set WorkObject

Set Max Speed

Set Acceleration

COMPAS RRC

Motion

Get Frame
Move to Frame

Get Joints
Move to Joints

Get Robtarget
Move to Robtarget

ETH zürich

COMPAS RRC

Utilities

Wait Time (Delay)

Stop (Pause)

Stop Watch

COMPAS RRC

Utilities

No-op (Ping)

Print Text

COMPAS RRC

Utilities

Custom Instructions

COMPAS RRC

IO signals

Read analog
Set analog

Read digital
Set digital
Pulse digital

Read group signal
Set group signal

Work objects

```
# Define pick positions
frame_on_pick = Frame(Point(50, 50, 50), Vector(0, -1, 0), Vector(-1, 0, 0))
frame_on_place = Frame(Point(50, 50, 50), Vector(0, -1, 0), Vector(-1, 0, 0))

# Move to frame on pickup pallet (work object)
abb.send(rrc.SetWorkObject('ob_RRC_Brick_Pallet'))
abb.send_and_wait(rrc.MoveToFrame(frame_on_pick, speed, rrc.Zone.FINE))

# Move to frame on place (work object)
abb.send(rrc.SetWorkObject('ob_RRC_Build_Space'))
abb.send_and_wait(rrc.MoveToFrame(frame_on_place, speed, rrc.Zone.FINE))
```

Pick & place example

```
# Create a new brick
done = abb.send_and_wait(rrc.PulseDigital('doNewBrick',0.2))
# [..]

# Vacuum on
abb.send(rrc.SetDigital('doVacuumOn',1))

# Motion
abb.send(rrc.MoveToFrame(pre_place_position, speed, rrc.Zone.Z10))
abb.send(rrc.MoveToFrame(place_position, speed, rrc.Zone.FINE))

# Vacuum off
abb.send(rrc.SetDigital('doVacuumOn',0))
```

wrapping up

COMPAS

Interdisciplinary research and transfer

Open source & Open Research

Primary datastructures: Mesh & Network

Processes

Prototypical Pick & Place

Assembly of totally ordered elements

Assembly as graphs

Robotic fundamentals

Frames, Transformation and Configuration

Robot control

Control types and RRC instruction set

Building blocks for robotics

Robot models

FK, IK, Kinematic and Cartesian Planning

ROS and MoveIt

Outlook

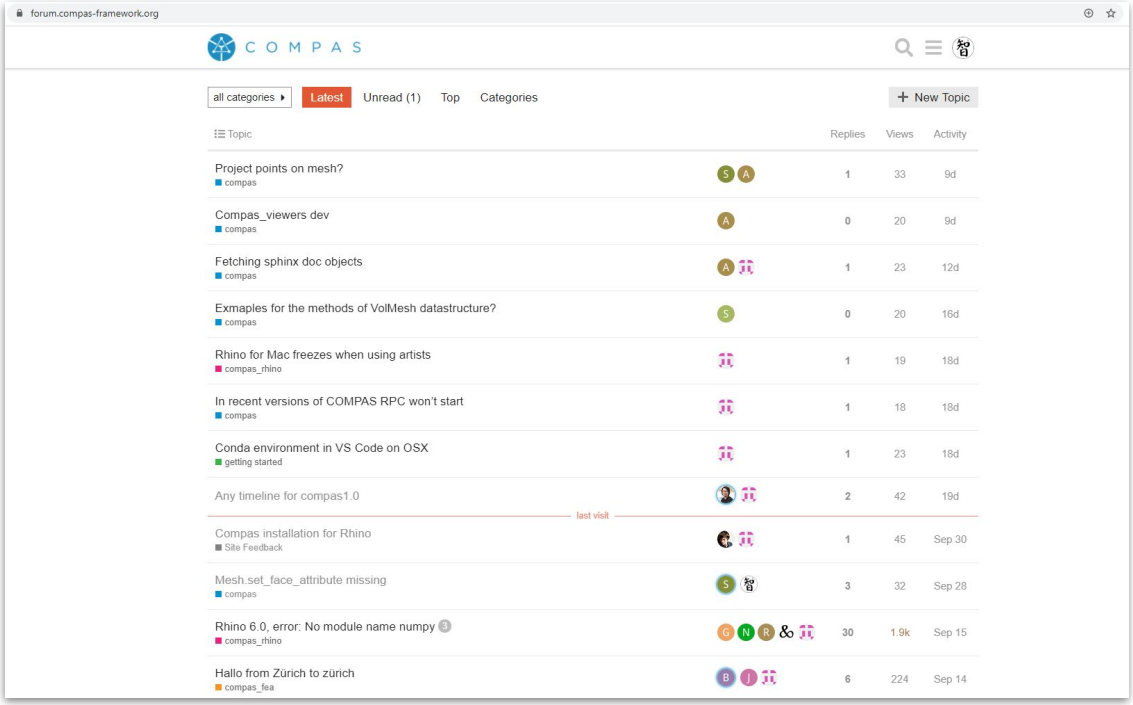
ProtoLab planning container 🤖

More GH components 🦗

Better input and output representations

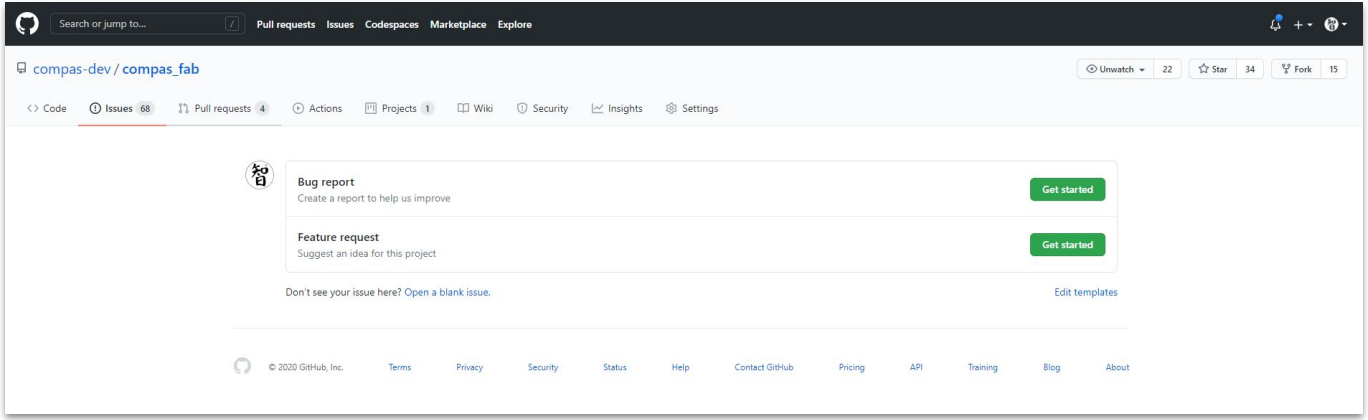
XR integration… and much more! 🚀

aka. how to send us feedback and get involved



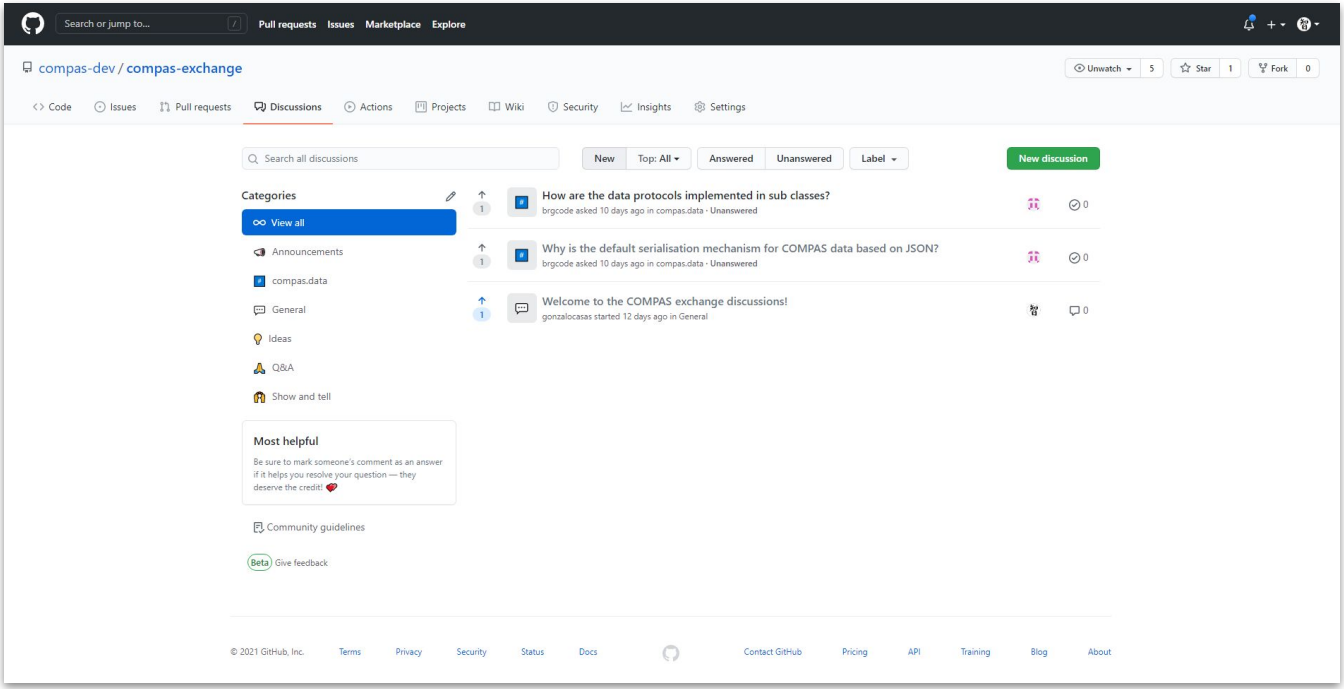
COMPAS forum

https://forum.compas-framework.org/



Github issues

https://github.com/compas-dev/compas_fab/issues

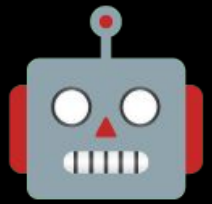


COMPAS exchange

https://github.com/compas-dev/compas-exchange

Twitter

https://twitter.com/compas_dev

Pull Requests
WELCOME
OPEN SOURCE

Thanks!