National Centre of Competence
in Research
Digital Fabrication

COMPAS Workshop: Robotic Assembly

Tom van Mele, Romana Rust, Gonzalo Casas

ETH Zürich, January 29th, 2019

BRG
GRAMAZIO
KOHLER
RESEARCH

Today’s agenda

COMPAS PROGRAMMING

Introduction to COMPAS, Python for COMPAS and datastructures

ASSEMBLY DESIGN & EQUILIBRIUM

Building an assembly in COMPAS and computing equilibrium

ROBOTIC FABRICATION

Robot fundamentals, models and introduction and usage of ROS

PLANNING & SIMULATION

Calculating path plans and simulating assembly process with ROS and MoveIt

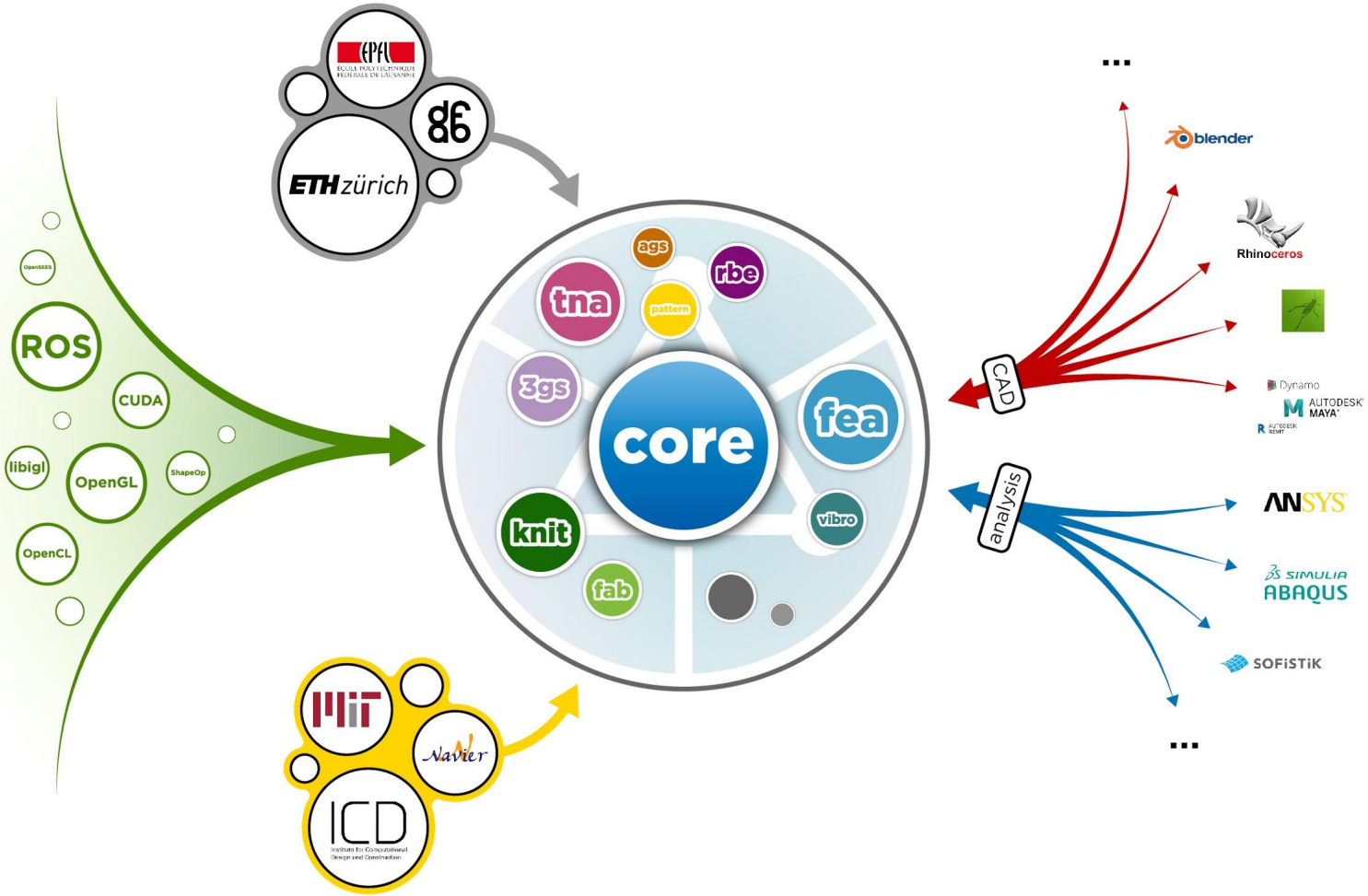
u.nu/rob-ws

COMPAS PROGRAMMING

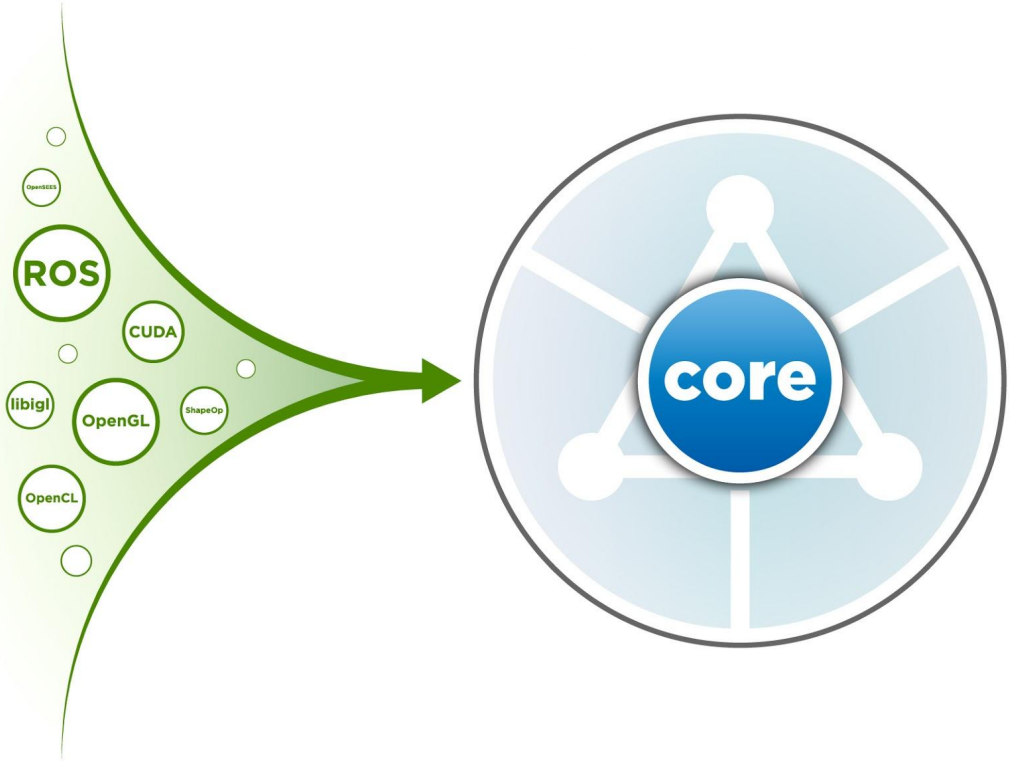
COMPAS concepts, Python for COMPAS, and Datastructures

COMPAS

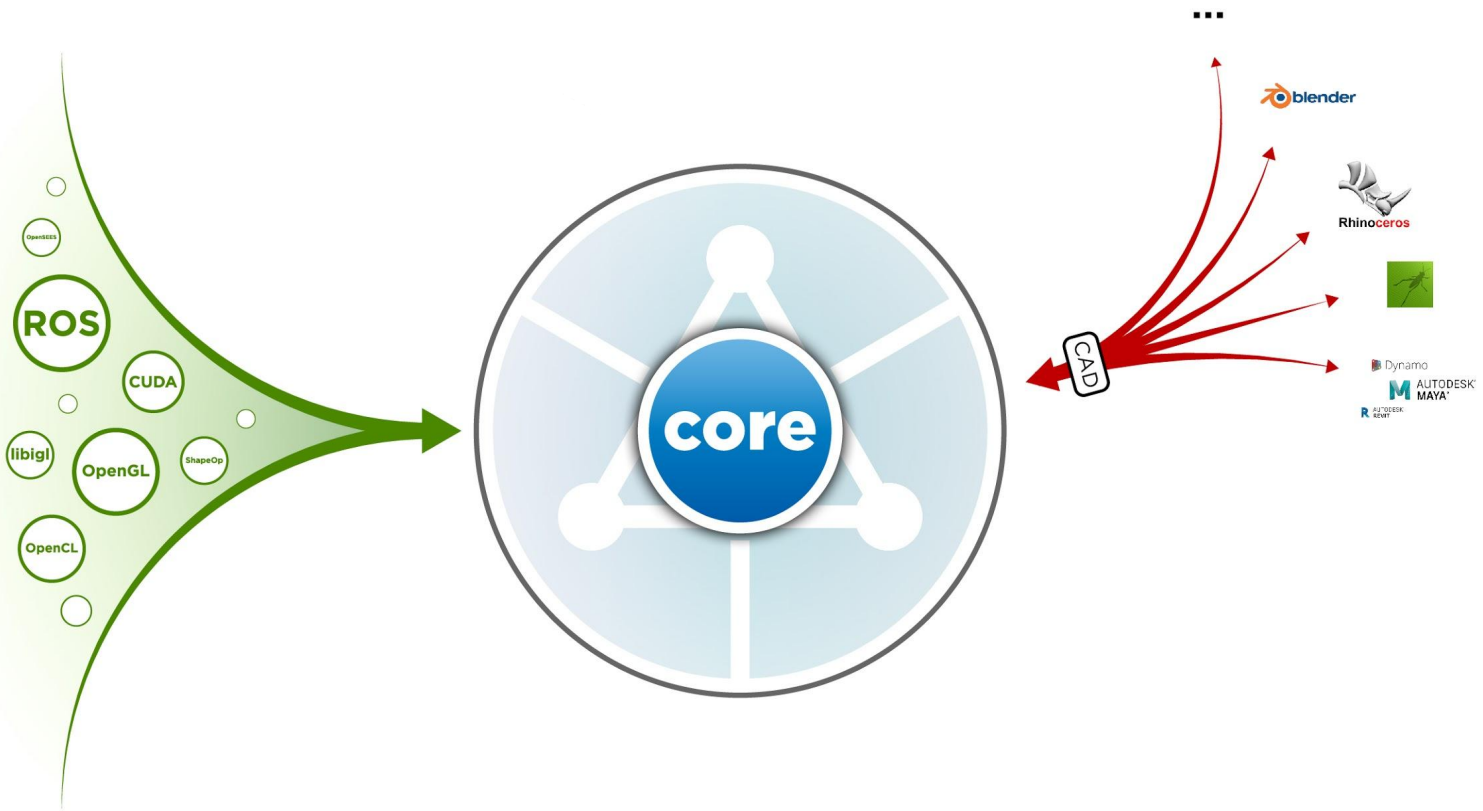
if callback:
callback(mesh, k, callback_args)
def smooth_mesh_length(mesh, lmin, lmax, fixed=None, kmax
if callback:
if not callable(callback):
raise Exception('Callback is not callable.')
fixed = fixed or []
fixed = set(fixed)
for k in range(kmax):

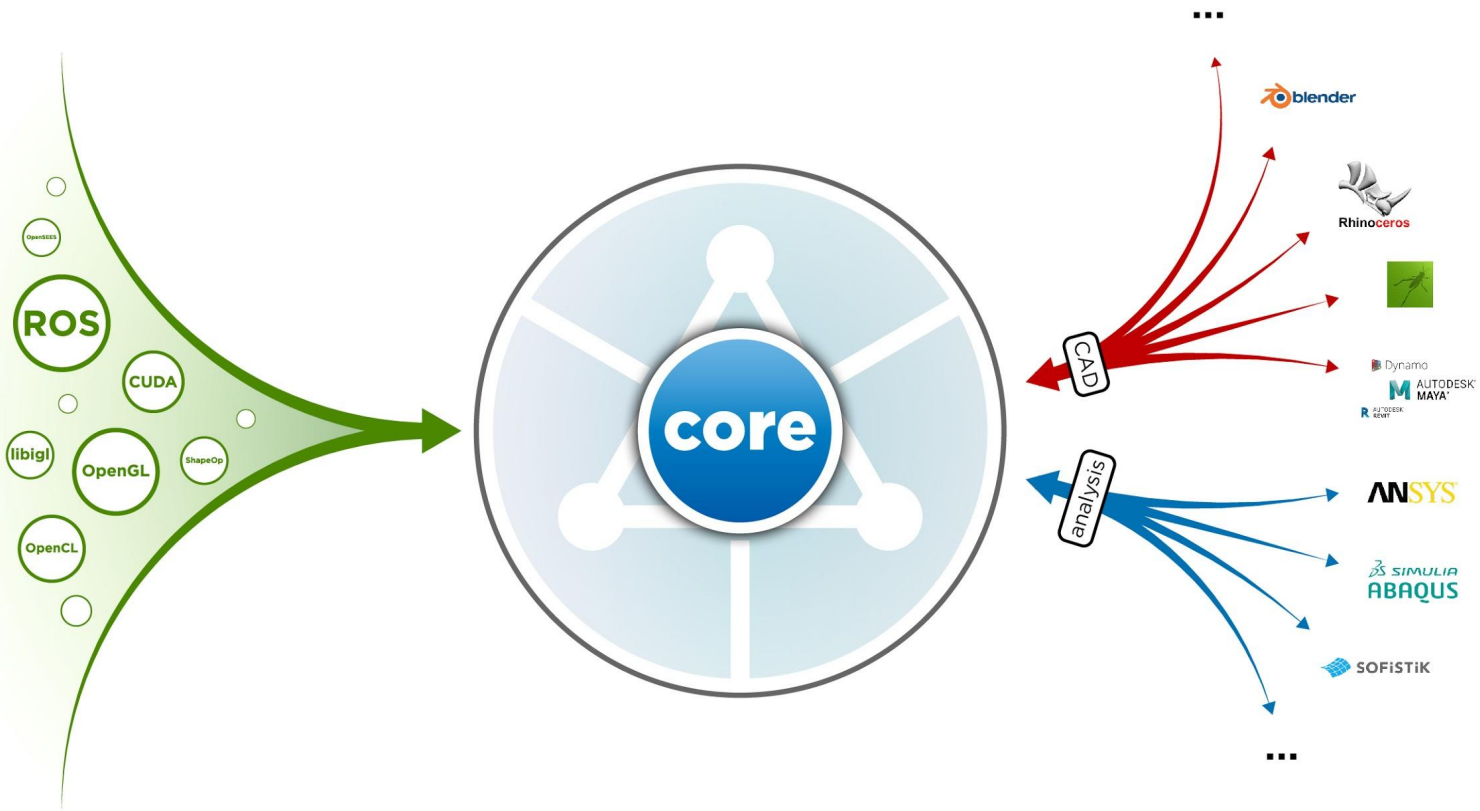

EPFL
ETH zürich
gg
OpenSEES
ROS
CUDA
libigl
OpenGL
ShapeOp
OpenCL
ags
tna
rbe
pattern
3gs
core
fea
knit
vibro
fab
MIT
Navier
ICD
...
blender
Rhinoceros
CAD
Dynamo
AUTODESK MAYA
analysis
ANSYS
SIMULIA ABAQUS
SOFiSTiK
...

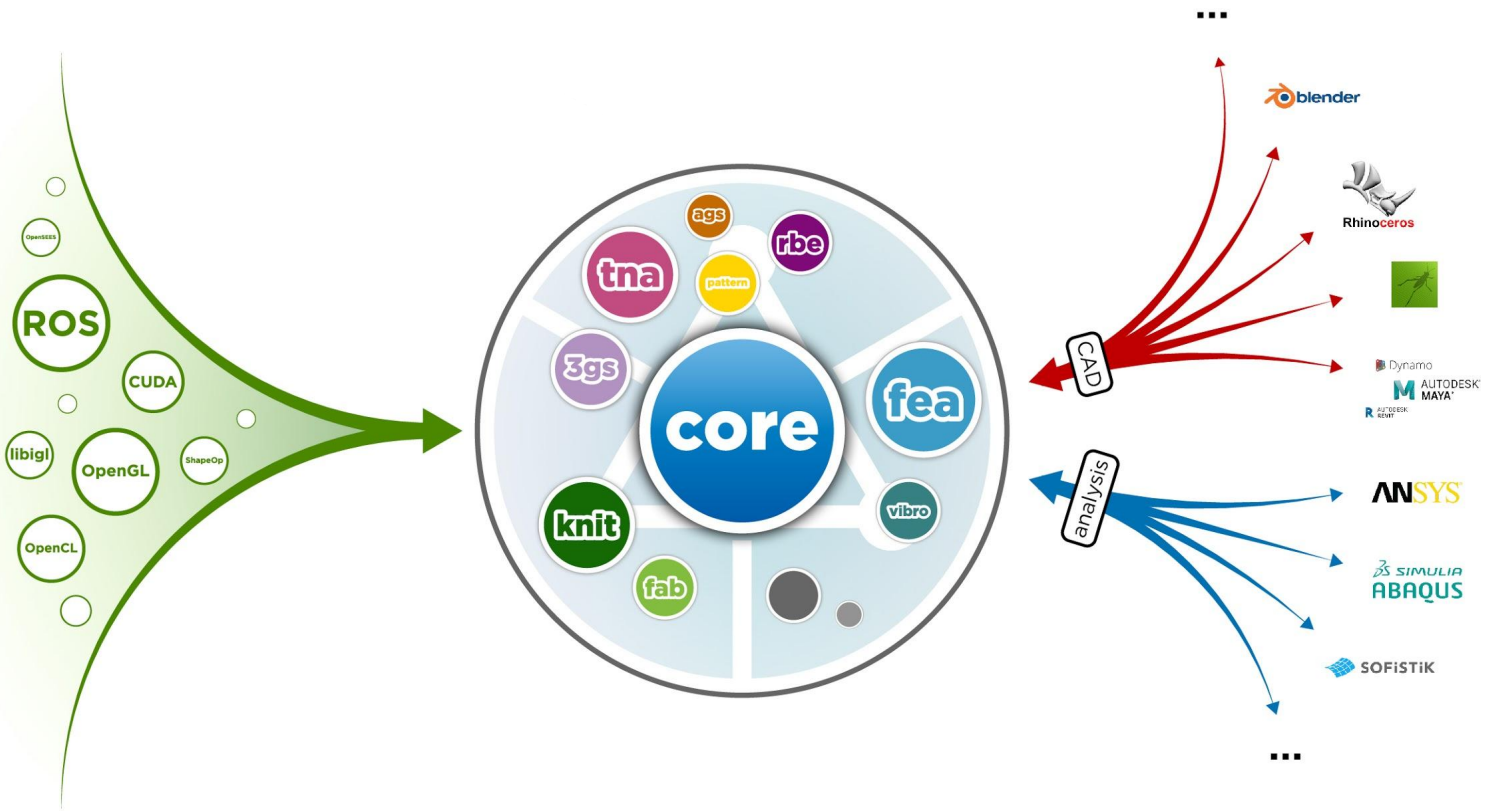
- CPython
- Flexible data structures
- Algorithms & utilities for AEC
- Geometry processing (independent of CAD)
- Numerical backends (NumPy, SciPy, Alglib)
- Visualisation tools

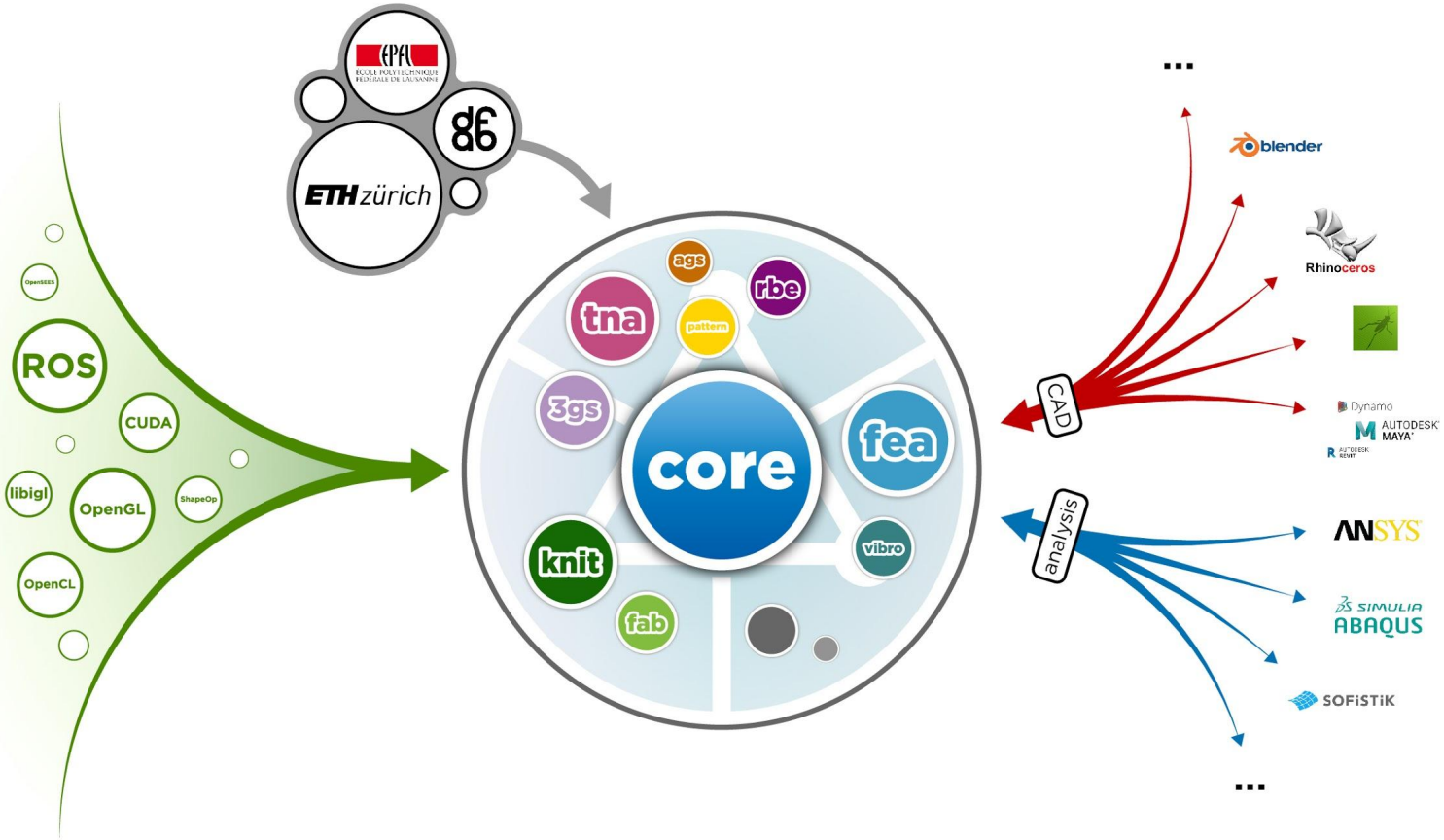


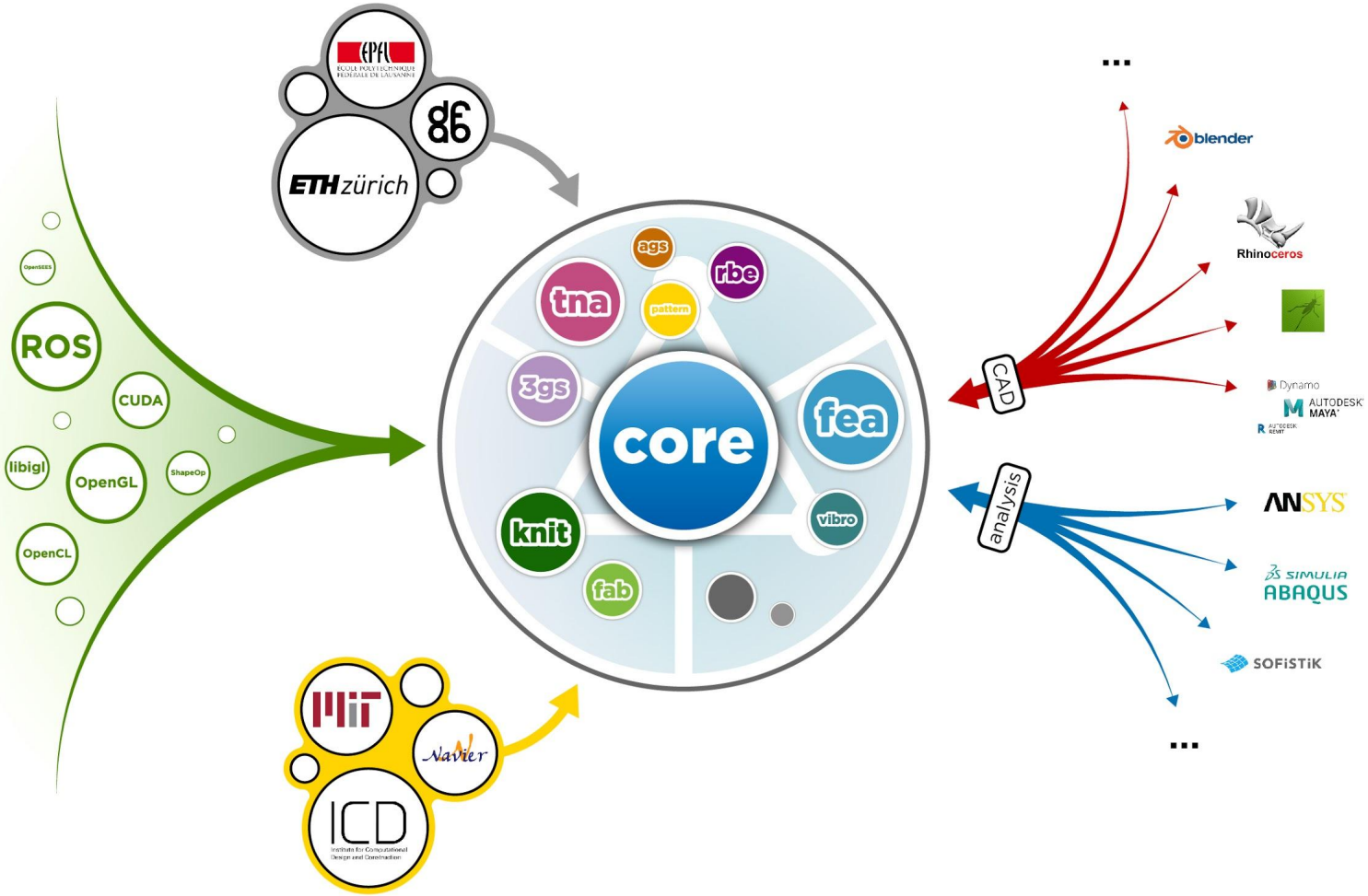
- Easy interop with C/C++
- Integration of 3rd party libs/tools

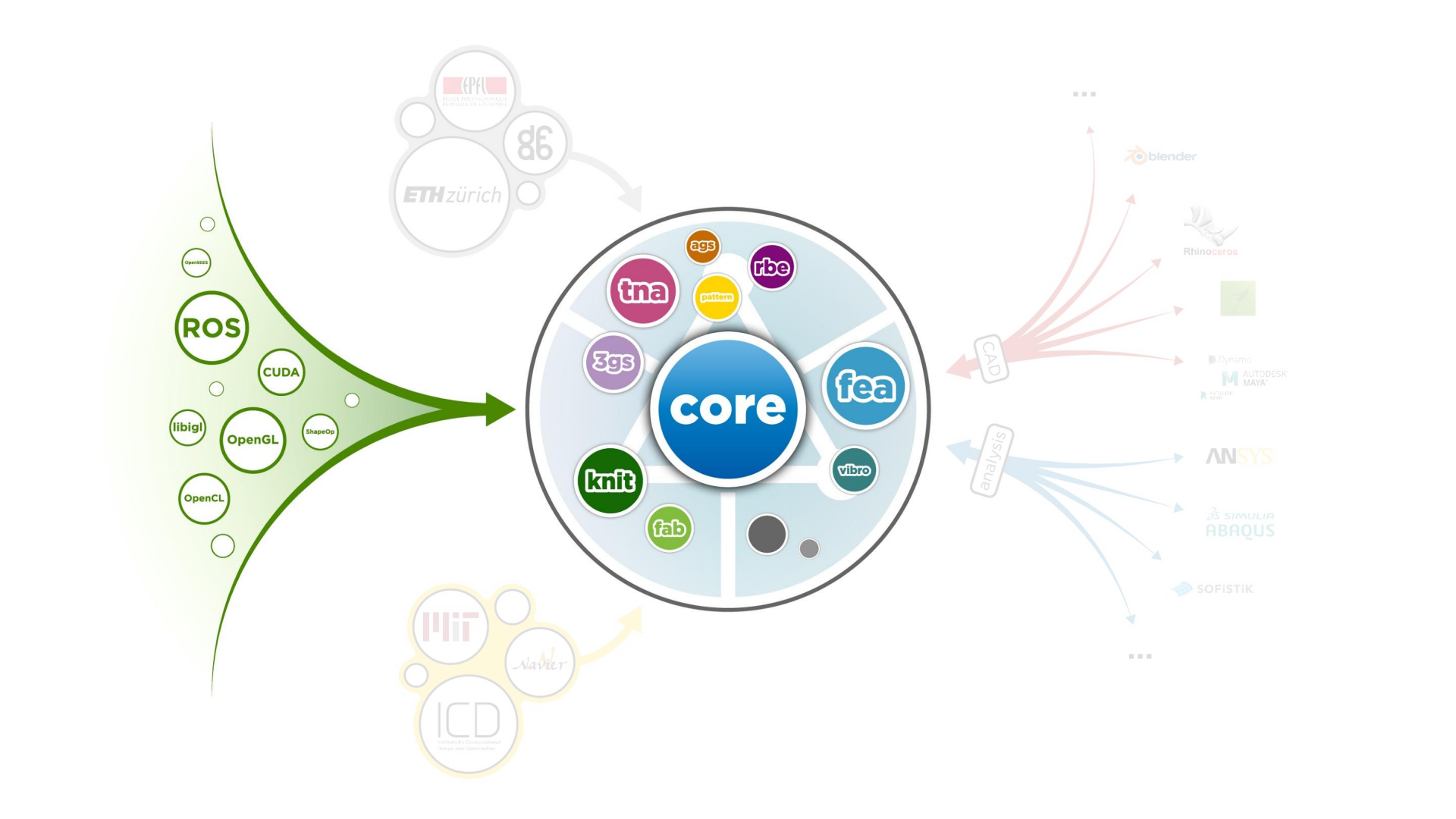

OpenSEES
ROS
CUDA
libigl
OpenGL
ShapeOp
OpenCL
core
CAD
...
blender
Rhinoceros
Dynamo
AUTODESK MAYA


OpenSEES
ROS
CUDA
libigl
OpenGL
ShapeOp
OpenCL
core
CAD
analysis
...
blender
Rhinoceros
Dynamo
AUTODESK MAYA
AUTODESK REVIT
ANSYS
SIMULIA ABAQUS
SOFiSTiK
...


OpenSEES
ROS
CUDA
libigl
OpenGL
ShapeOp
OpenCL
ags
tna
rbe
pattern
3gs
core
fea
knit
vibro
fab
CAD
analysis
...
blender
Rhinoceros
Dynamo
AUTODESK MAYA
AUTODESK REVIT
ANSYS
SIMULIA ABAQUS
SOFiSTiK
...


EPFL
ETH zürich
ROS
CUDA
libigl
OpenGL
ShapeOp
OpenCL
ags
tna
rbe
pattern
3gs
core
fea
knit
vibro
fab
CAD
analysis
...
blender
Rhinoceros
Dynamo
AUTODESK MAYA
ANSYS
SIMULIA ABAQUS
SOFiSTiK
...


EPFL
ETH zürich
gg
OpenSEES
ROS
CUDA
libigl
OpenGL
ShapeOp
OpenCL
ags
tna
rbe
pattern
3gs
core
fea
knit
vibro
fab
MIT
Navier
ICD
...
blender
Rhinoceros
CAD
Dynamo
AUTODESK MAYA
analysis
ANSYS
SIMULIA ABAQUS
SOFiSTiK
...


ETH zürich
OpenSEES
ROS
CUDA
libigl
OpenGL
ShapeOp
OpenCL
ags
tna
rbe
pattern
3gs
core
fea
knit
vibro
fab
CAD
analysis
...
blender
Rhinoceros
Dynamo
AUTODESK MAYA
ANSYS
SIMULIA ABAQUS
SOFiSTiK
...
MIT
Navier
ICD

vanmelet — python — 77×30

```
arch-hix-dock-453:~ vanmelet$ python
Python 3.6.3 |Anaconda custom (64-bit)| (default, Oct  6 2017, 12:04:38)
[GCC 4.2.1 Compatible Clang 4.0.1 (tags/RELEASE_401/final)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> import compas
>>> from compas.datastructures import Mesh
>>> mesh = Mesh.from_obj(compas.get('faces.obj'))
>>> mesh.summary()

================================================================================
===
Mesh summary
================================================================================
===

- name: Mesh
- vertices: 36
- edges: 60
- faces: 25
- vertex degree: 2/4
- face degree: 2/4

================================================================================
===

>>>
```

dr_numpy.py — compas-dev

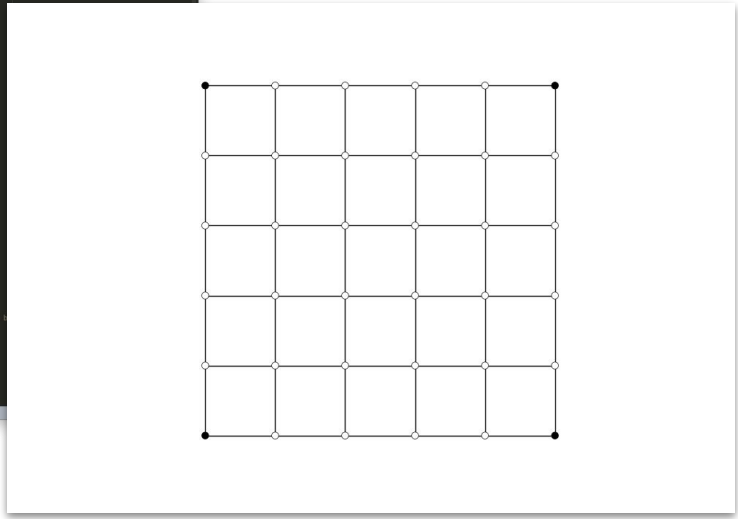
FOLDERS

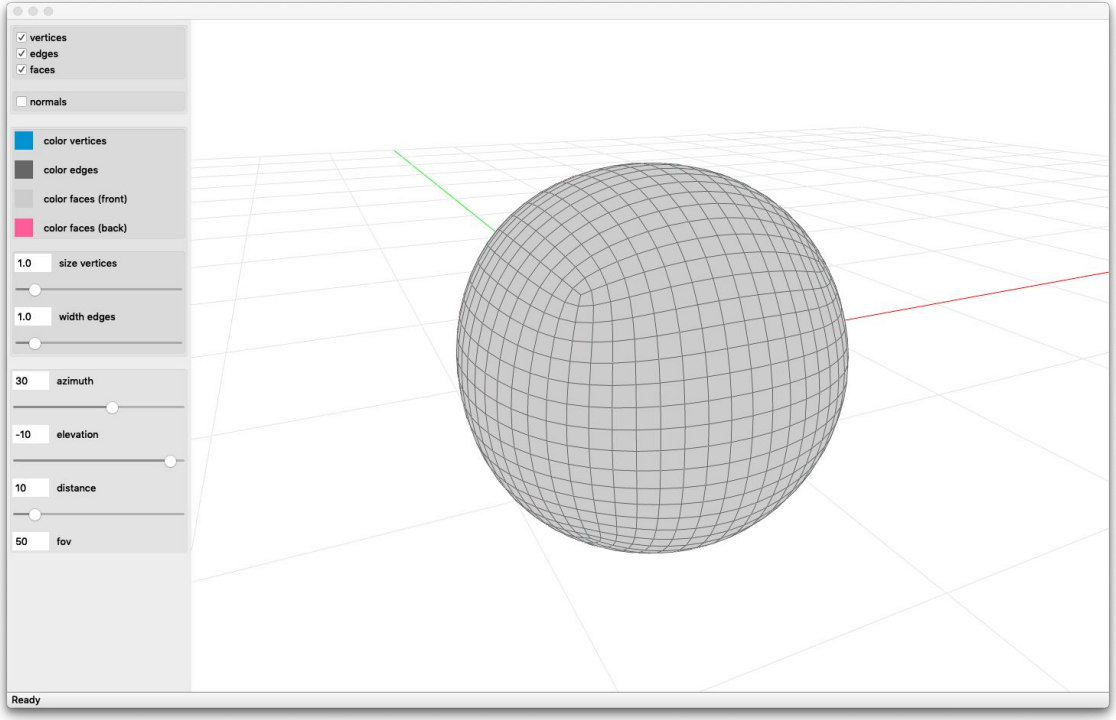
- compas-dev
  - compas
    - build
    - data
    - dist
    - docs
    - libs
    - src
      - compas
        - __pycache__
        - com
        - datastructures
        - files
        - geometry
        - interop
        - numerical
          - __pycache__
          - _alglib
          - alglib
          - 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

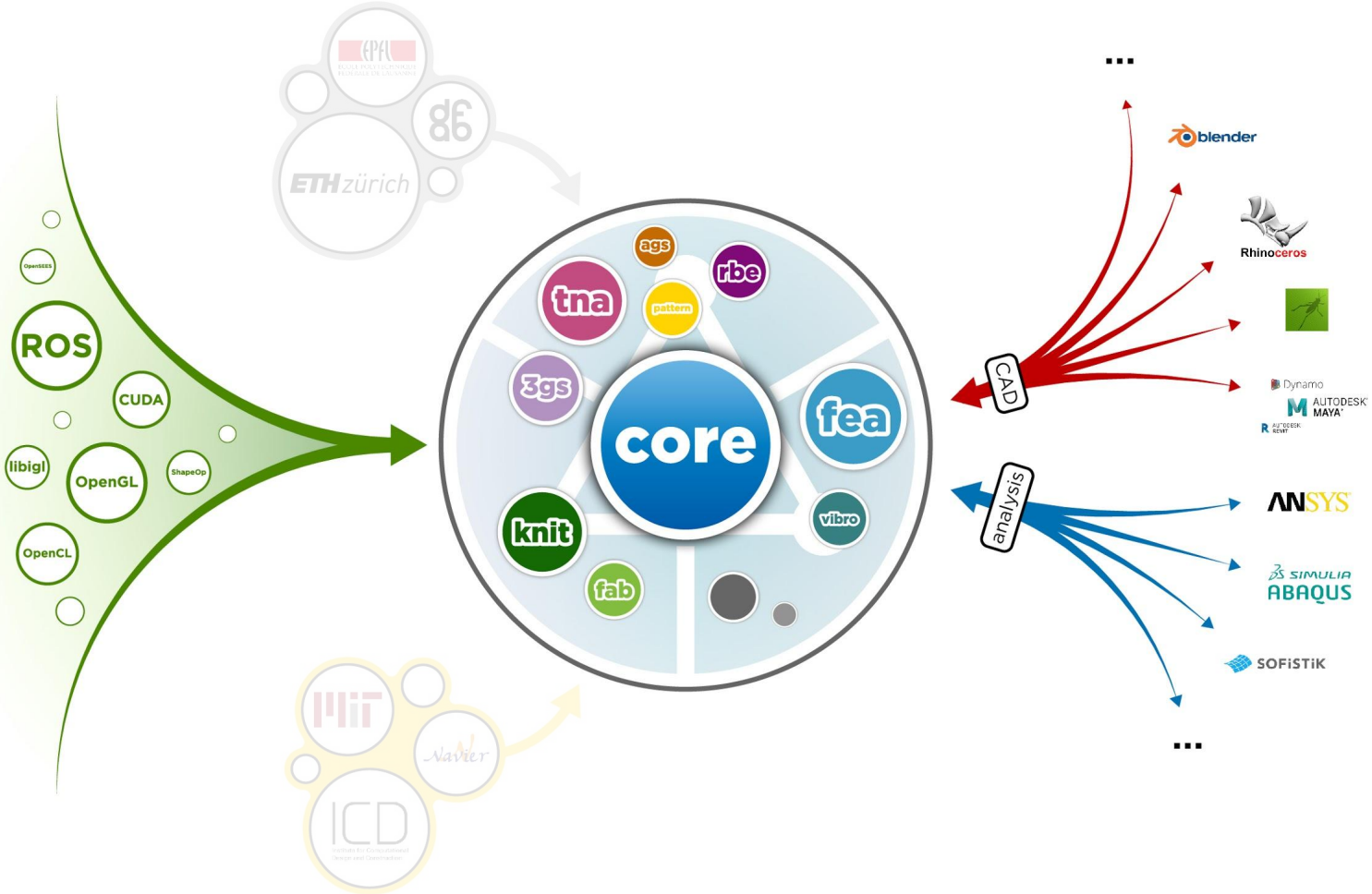
dr_numpy.py

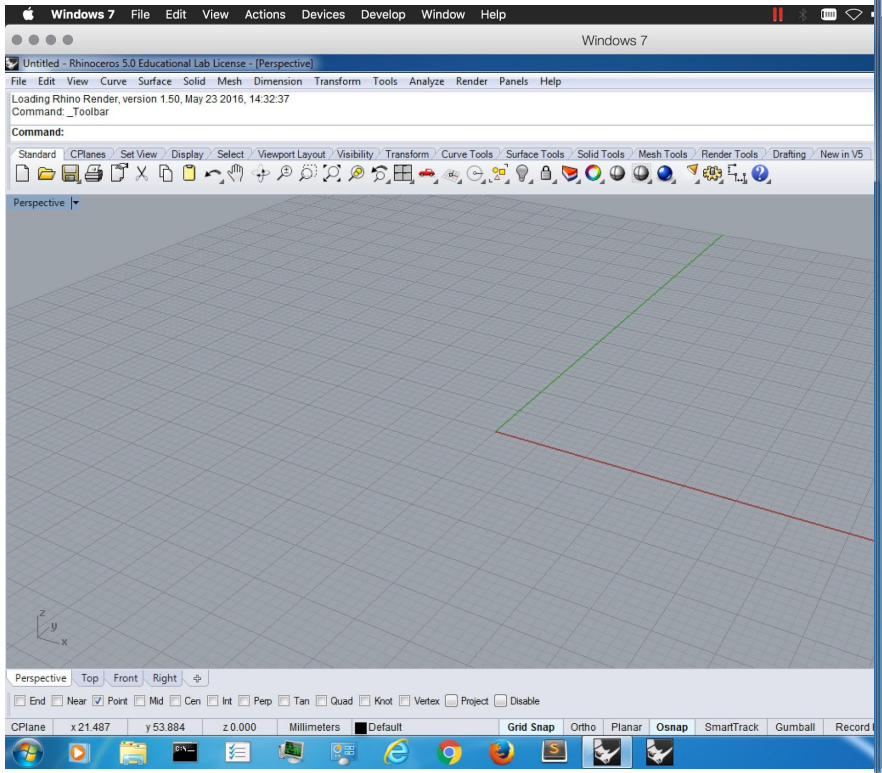
```python
from __future__ import absolute_import
from __future__ import division
from __future__ import print_function

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
```

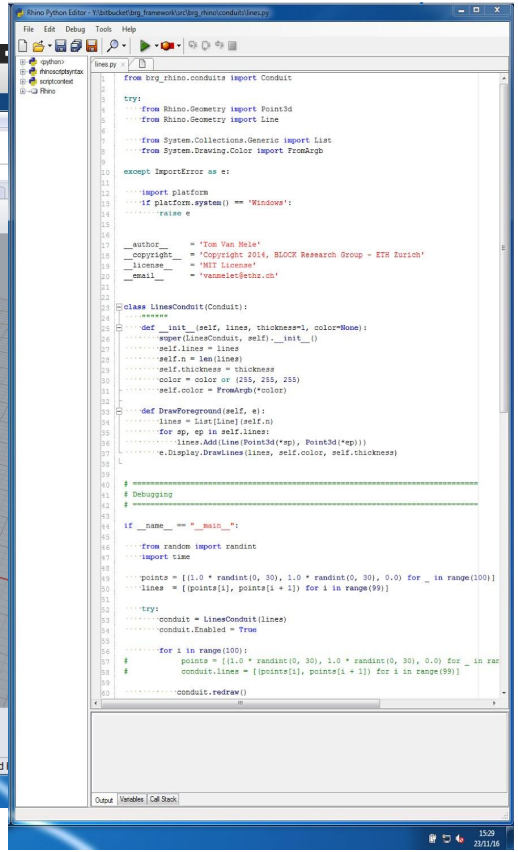
Git branch: master, index: ✓, working: 10✱, Line 33, Column 1

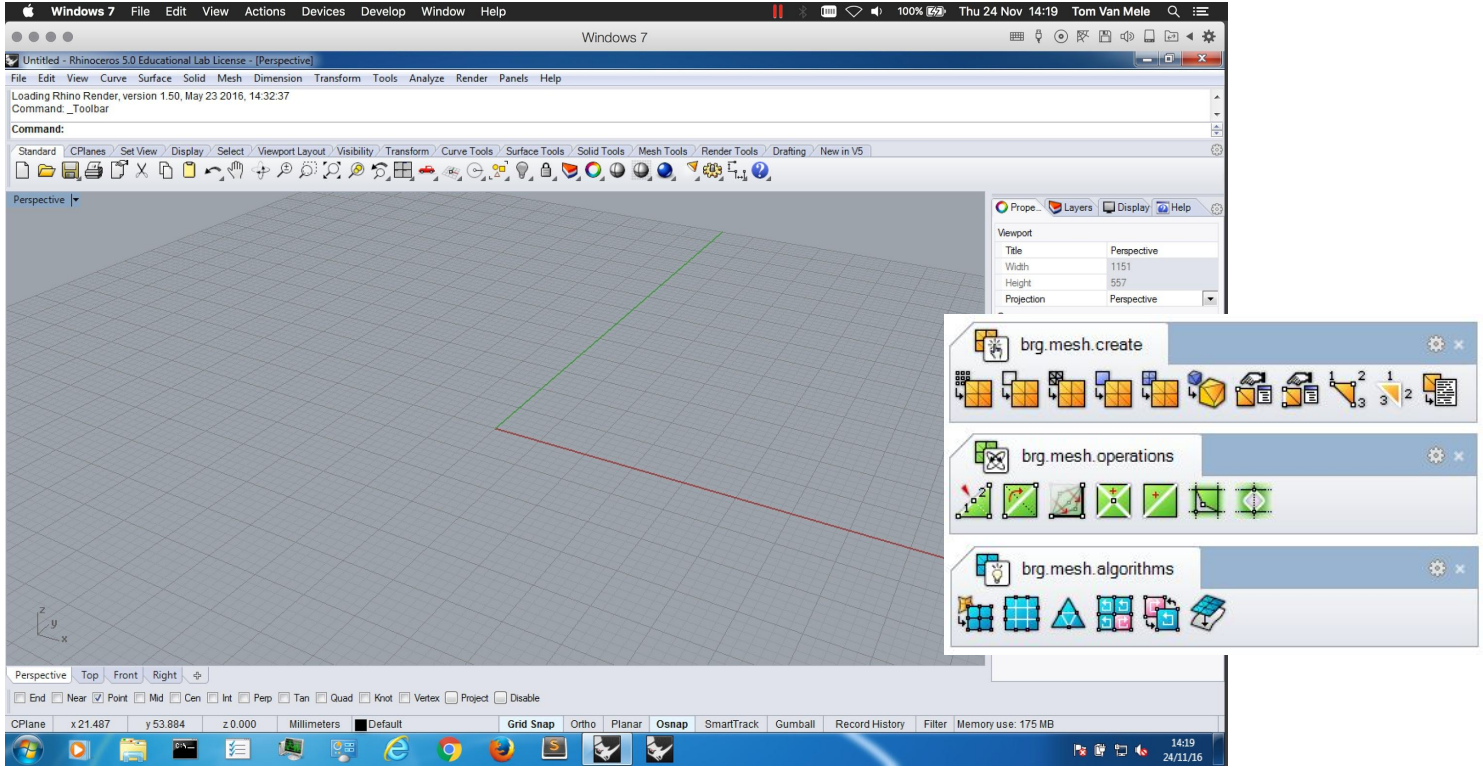


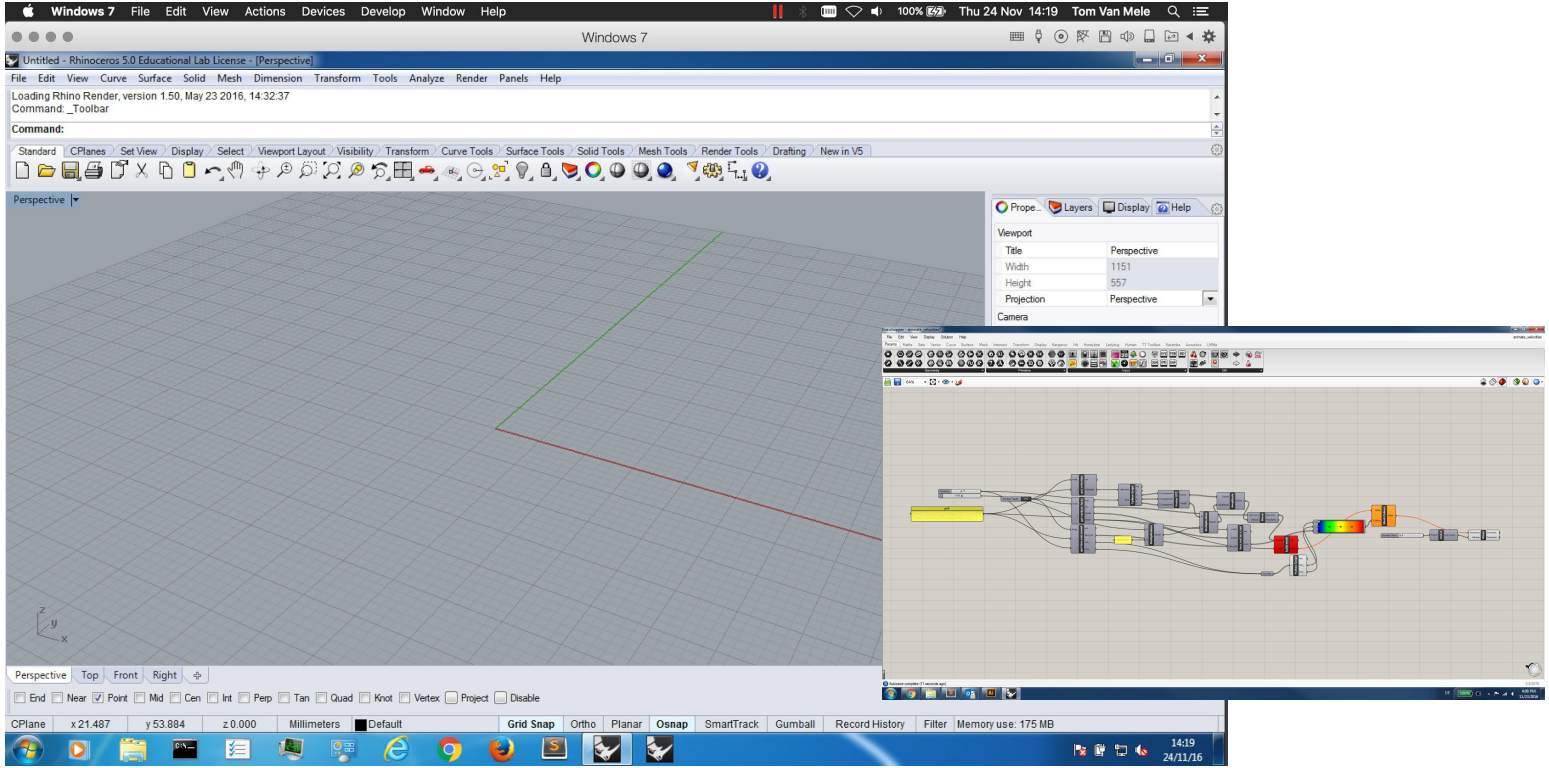

vertices
edges
faces
normals
color vertices
color edges
color faces (front)
color faces (back)
1.0
size vertices
1.0
width edges
30
azimuth
-10
elevation
10
distance
50
fov
Ready

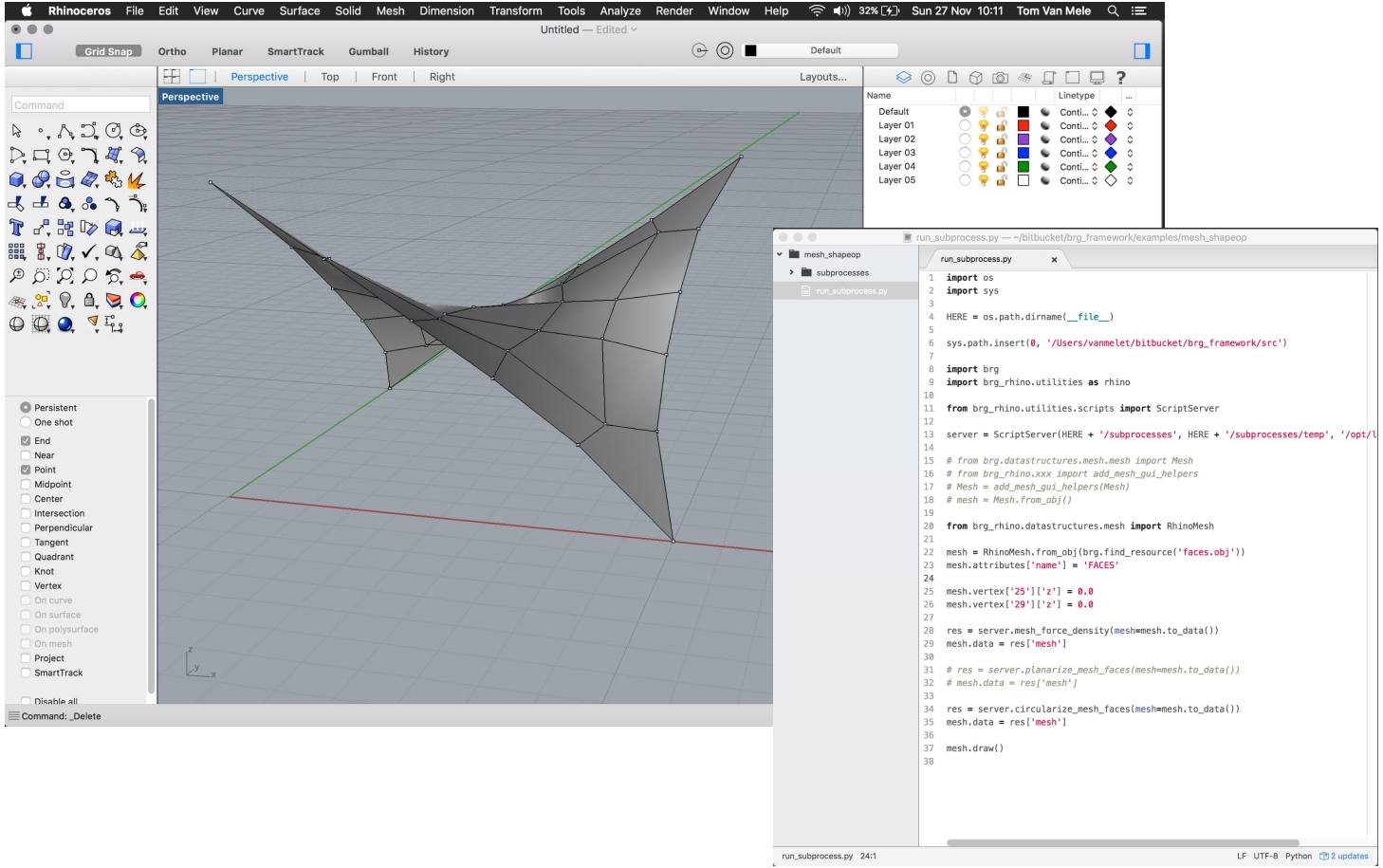

ETH zürich
OpenSEES
ROS
CUDA
libigl
OpenGL
ShapeOp
OpenCL
ags
rbe
tna
pattern
3gs
core
fea
knit
vibro
fab
CAD
analysis
...
blender
Rhinoceros
Dynamo
AUTODESK MAYA
AUTODESK REVIT
ANSYS
SIMULIA ABAQUS
SOFiSTiK
...
MIT
Navier
ICD


Windows 7 File Edit View Actions Devices Develop Window Help
Windows 7
Untitled - Rhinoceros 5.0 Educational Lab License - [Perspective]
File Edit View Curve Surface Solid Mesh Dimension Transform Tools Analyze Render Panels Help
Loading Rhino Render, version 1.50, May 23 2016, 14:32:37
Command: _Toolbar
Command:
Standard CPlanes Set View Display Select Viewport Layout Visibility Transform Curve Tools Surface Tools Solid Tools Mesh Tools Render Tools Drafting New in V5
Perspective
Perspective Top Front Right
End Near Point Mid Cen Int Perp Tan Quad Knot Vertex Project Disable
CPlane x 21.487 y 53.884 z 0.000 Millimeters Default Grid Snap Ortho Planar Osnap SmartTrack Gumball


Rhino Python Editor - V:\bitbucket\brg_framework\src\brg_rhino\conduits\lines.py
File Edit Debug Tools Help
lines.py
<python>
rhinoscriptsyntax
scriptcontext
Rhino
from brg_rhino.conduits import Conduit
try:
from Rhino.Geometry import Point3d
from Rhino.Geometry import Line
from System.Collections.Generic import List
from System.Drawing.Color import FromArgb
except ImportError as e:
import platform
if platform.system() == 'Windows':
raise e
__author__ = 'Tom Van Mele'
__copyright__ = 'Copyright 2014, BLOCK Research Group - ETH Zurich'
__license__ = 'MIT License'
__email__ = 'vanmelet@ethz.ch'
class LinesConduit(Conduit):
"""
def __init__(self, lines, thickness=1, color=None):
super(LinesConduit, self).__init__()
self.lines = lines
self.n = len(lines)
self.thickness = thickness
color = color or (255, 255, 255)
self.color = FromArgb(*color)
def DrawForeground(self, e):
lines = List[Line](self.n)
for sp, ep in self.lines:
lines.Add(Line(Point3d(*sp), Point3d(*ep)))
e.Display.DrawLines(lines, self.color, self.thickness)
# ==========
# Debugging
# ==========
if __name__ == "__main__":
from random import randint
import time
points = [(1.0 * randint(0, 30), 1.0 * randint(0, 30), 0.0) for _ in range(100)]
lines = [(points[i], points[i + 1]) for i in range(99)]
try:
conduit = LinesConduit(lines)
conduit.Enabled = True
for i in range(100):
points = [(1.0 * randint(0, 30), 1.0 * randint(0, 30), 0.0) for _ in ran
conduit.lines = [(points[i], points[i + 1]) for i in range(99)]
conduit.redraw()
Output Variables Call Stack


Windows 7
File Edit View Actions Devices Develop Window Help
Thu 24 Nov 14:19 Tom Van Mele
Untitled - Rhinoceros 5.0 Educational Lab License - [Perspective]
File Edit View Curve Surface Solid Mesh Dimension Transform Tools Analyze Render Panels Help
Loading Rhino Render, version 1.50, May 23 2016, 14:32:37
Command: _Toolbar
Command:
Standard CPlanes Set View Display Select Viewport Layout Visibility Transform Curve Tools Surface Tools Solid Tools Mesh Tools Render Tools Drafting New in V5
Perspective
Viewport
Title Perspective
Width 1151
Height 557
Projection Perspective
brg.mesh.create
brg.mesh.operations
brg.mesh.algorithms
Perspective Top Front Right
End Near Point Mid Cen Int Perp Tan Quad Knot Vertex Project Disable
CPlane x 21.487 y 53.884 z 0.000 Millimeters Default
Grid Snap Ortho Planar Osnap SmartTrack Gumball Record History Filter Memory use: 175 MB
14:19
24/11/16

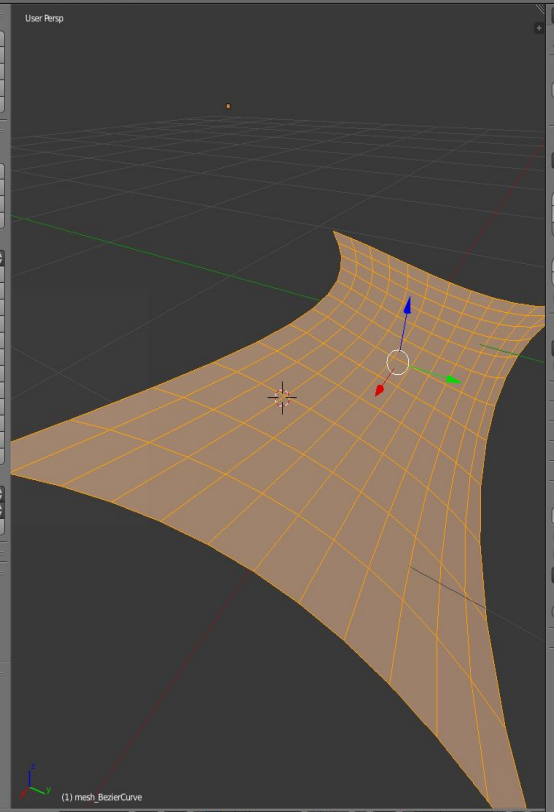

Windows 7 File Edit View Actions Devices Develop Window Help
Thu 24 Nov 14:19 Tom Van Mele
Windows 7
Untitled - Rhinoceros 5.0 Educational Lab License - [Perspective]
File Edit View Curve Surface Solid Mesh Dimension Transform Tools Analyze Render Panels Help
Loading Rhino Render, version 1.50, May 23 2016, 14:32:37
Command: _Toolbar
Command:
Standard CPlanes Set View Display Select Viewport Layout Visibility Transform Curve Tools Surface Tools Solid Tools Mesh Tools Render Tools Drafting New in V5
Perspective
Prope... Layers Display Help
Viewport
Title Perspective
Width 1151
Height 557
Projection Perspective
Camera
Perspective Top Front Right
End Near Point Mid Cen Int Perp Tan Quad Knot Vertex Project Disable
CPlane x 21.487 y 53.884 z 0.000 Millimeters Default Grid Snap Ortho Planar Osnap SmartTrack Gumball Record History Filter Memory use: 175 MB
14:19
24/11/16


Rhinoceros File Edit View Curve Surface Solid Mesh Dimension Transform Tools Analyze Render Window Help
Sun 27 Nov 10:11 Tom Van Mele
Untitled — Edited
Grid Snap Ortho Planar SmartTrack Gumball History
Default
Perspective Top Front Right Layouts...
Perspective
Command
Persistent
One shot
End
Near
Point
Midpoint
Center
Intersection
Perpendicular
Tangent
Quadrant
Knot
Vertex
On curve
On surface
On polysurface
On mesh
Project
SmartTrack
Disable all
Command: _Delete
Name
Linetype
Default
Layer 01
Layer 02
Layer 03
Layer 04
Layer 05
run_subprocess.py — ~/bitbucket/brg_framework/examples/mesh_shapeop
mesh_shapeop
subprocesses
run_subprocess.py
import os
import sys
HERE = os.path.dirname(__file__)
sys.path.insert(0, '/Users/vanmelet/bitbucket/brg_framework/src')
import brg
import brg_rhino.utilities as rhino
from brg_rhino.utilities.scripts import ScriptServer
server = ScriptServer(HERE + '/subprocesses', HERE + '/subprocesses/temp', '/opt/l
# from brg.datastructures.mesh.mesh import Mesh
# from brg_rhino.xxx import add_mesh_gui_helpers
# Mesh = add_mesh_gui_helpers(Mesh)
# mesh = Mesh.from_obj()
from brg_rhino.datastructures.mesh import RhinoMesh
mesh = RhinoMesh.from_obj(brg.find_resource('faces.obj'))
mesh.attributes['name'] = 'FACES'
mesh.vertex['25']['z'] = 0.0
mesh.vertex['29']['z'] = 0.0
res = server.mesh_force_density(mesh=mesh.to_data())
mesh.data = res['mesh']
# res = server.planarize_mesh_faces(mesh=mesh.to_data())
# mesh.data = res['mesh']
res = server.circularize_mesh_faces(mesh=mesh.to_data())
mesh.data = res['mesh']
mesh.draw()
run_subprocess.py 24:1
LF UTF-8 Python 2 updates

Blender* [/home/al/Dropbox/bb/brg_framework/data/membrane_dr.blend]

```python
""" An example of Dynamic Relaxation, using a Blender model."""

from brg_blender.geometry import curve_to_mesh
from brg_blender.geometry import remove_duplicate_vertices
from brg_blender.geometry import grid_fill

from brg_blender.utilities import layer_clear
from brg_blender.utilities import delete_objects
from brg_blender.utilities import objects_by_layer
from brg_blender.utilities import objects_join
from brg_blender.utilities import select_object

import bpy

import sys
sys.path.append('/home/al/Dropbox/bb/brg_framework/datastructures/mesh/')

from brg.datastructures.network.network import Network

from brg.datastructures.utilities import geometric_key
from brg.datastructures.utilities import gkeys_from_network


__author__      = ['Andrew Liew <liew@arch.ethz.ch>']
__copyright__   = 'Copyright 2016, BLOCK Research Group - ETH Zurich'
__license__     = 'MIT License'
__version__     = '0.1'
__date__        = 'Oct 20, 2016'


import sys
print(sys.path)

# Layer keys
lkeys = {
    'curves': 0,
    'mesh_curves': 1}

# Mesh from Gridfill
curves = objects_by_layer(0)
layer_clear(1)
mesh_curves = []
for curve in curves:
    mesh_curves.append(curve_to_mesh(curve))
mesh_joined = objects_join(mesh_curves)
remove_duplicate_vertices(mesh_joined)
grid_fill(mesh_joined)
print(mesh_joined.data.polygons[0].vertices[0])
print(mesh_joined.data.vertices[4].co.x)

#mesh = RhinoMesh.from_surface(surface, density=(20, 20))
```

```
bpy.ops.paint.vertex_paint_toggle()
bpy.ops.paint.vertex_paint_toggle()
bpy.ops.object.editmode_toggle()
```



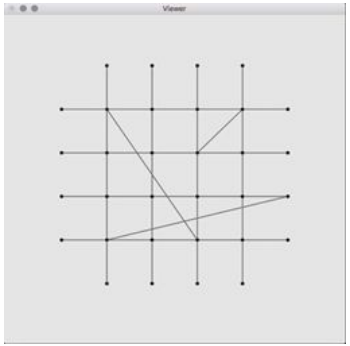
```python
# Rhino
from compas.datastructures import Mesh
from compas_rhino.artists import MeshArtist

mesh = Mesh.from_obj('https://u.nu/hypar')

artist = MeshArtist(mesh)

artist.draw()
```

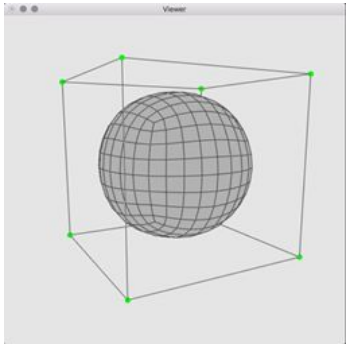
```python
# Blender
from compas.datastructures import Mesh
from compas_blender.artists import MeshArtist

mesh = Mesh.from_obj('https://u.nu/hypar')

artist = MeshArtist(mesh)

artist.draw()
```

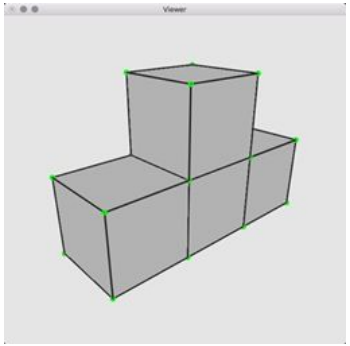
```python
# Rhino
from compas.datastructures import Mesh
from compas_rhino.artists import MeshArtist

mesh = Mesh.from_obj('https://u.nu/hypar')

artist = MeshArtist(mesh)

artist.draw()
```

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

mesh = Mesh.from_obj('https://u.nu/hypar')

artist = MeshArtist(mesh)

artist.draw()
```

Data structures







Network

- General networks
- Directed edge graph

Mesh

- Surface meshes
- Half-edge

Volumetric Mesh

- Cellular meshes
- Half-plane

Algorithms

Geometry

- Smoothing
- Planarization
- Convex hull
- Triangulation
- Transformations
- Bounding boxes
- Isolines
- ...

Numerical

- Dynamic relaxation
- Force density
- Topology optimization
- Genetic algorithms
- Principal component analysis
- Differential evolution
- ...

Topology

- Traversal
- Graph colouring
- Planarity
- Conway operators
- ...

Getting started

python™ Package Index

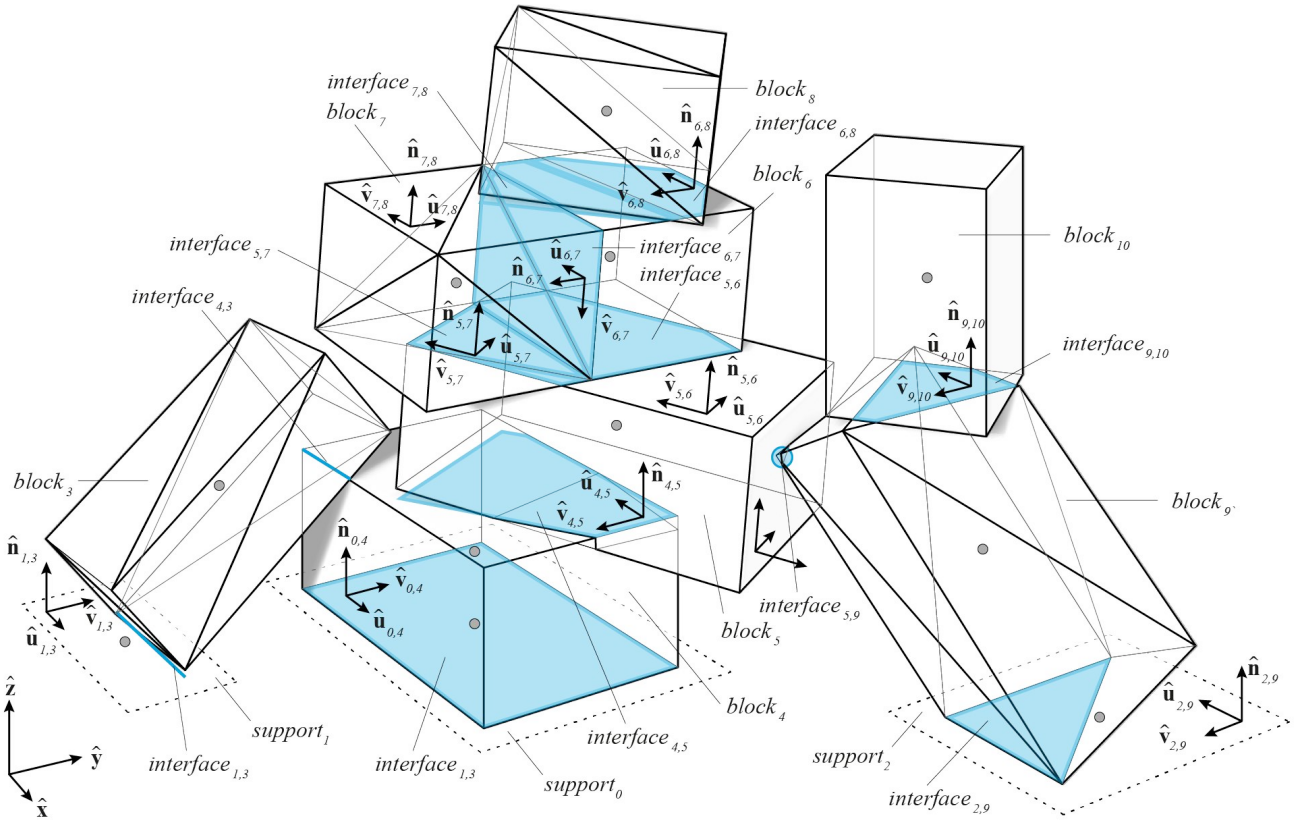
CONDA

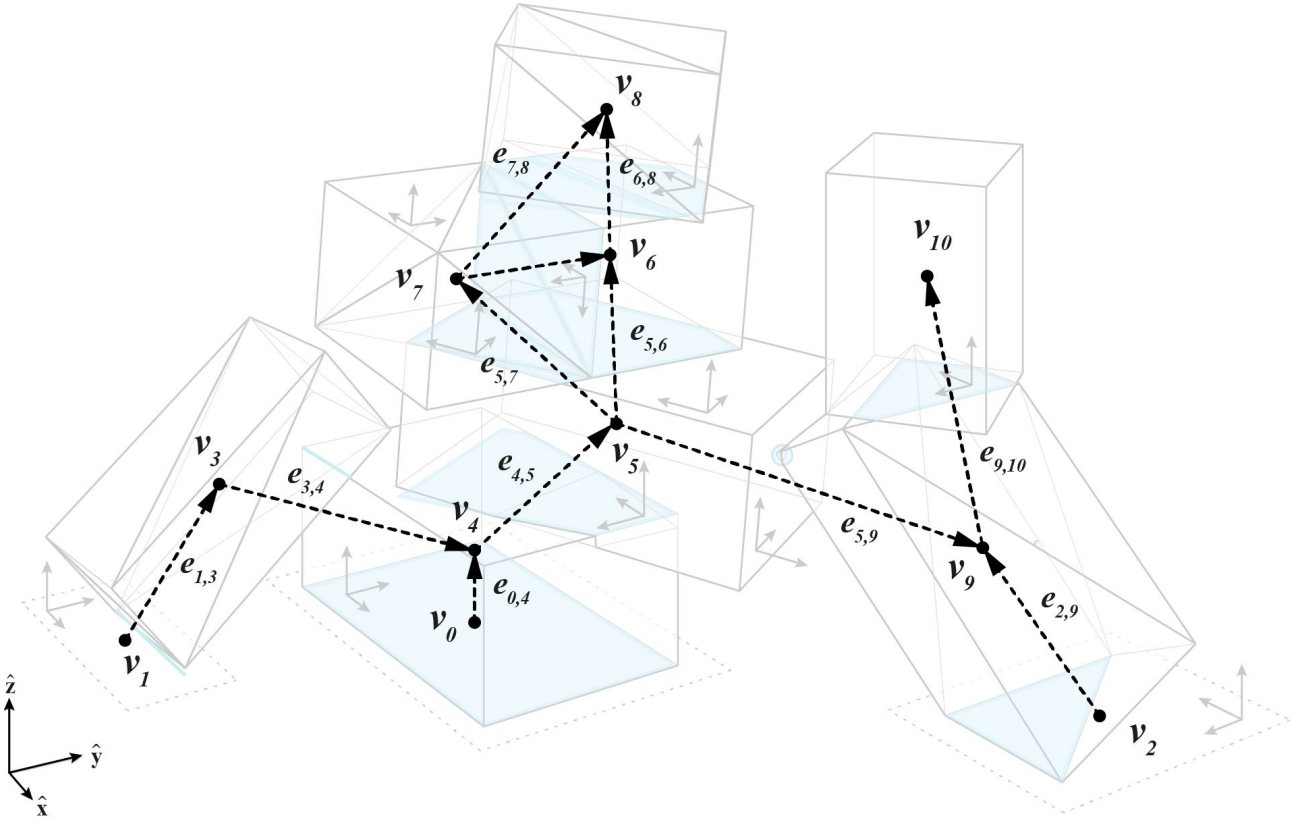
```
conda install compas
```

ASSEMBLY DESIGN & EQUILIBRIUM

Assembly Design, and Equilibrium

compas_ags Algebraic Graph Statics	**compas_tna** Thrust Network Analysis	**compas_3gs** 3D Graphic Statics	**compas_fea** Finite Element Analysis	**compas_fab** Robotic Fabrication
compas_assembly Block Assemblies	**compas_knit** Spatial Knitting	**compas_dem** Discrete Element Modeling (3DEC)	**compas_dr6** 6DOF Dynamic Relaxation	**compas_rbe** Rigid Block Equilibrium
compas_ml Machine Learning	**compas_bend** Bending Active Structures	**compas_rocking** Collapse Mechanisms	**compas_ctrl** Active On-site Control	**compas_pattern** Mesh Topology Finding


interface 7,8
block 7
block 8
interface 6,8
block 6
interface 5,7
interface 6,7
interface 5,6
interface 4,3
block 10
interface 9,10
block 3
block 9
block 5
interface 5,9
block 4
support 1
interface 1,3
interface 1,3
support 0
interface 4,5
support 2
interface 2,9
$\hat{\mathbf{n}}_{7,8}$ $\hat{\mathbf{u}}_{7,8}$ $\hat{\mathbf{v}}_{7,8}$
$\hat{\mathbf{n}}_{6,8}$ $\hat{\mathbf{u}}_{6,8}$ $\hat{\mathbf{v}}_{6,8}$
$\hat{\mathbf{n}}_{6,7}$ $\hat{\mathbf{u}}_{6,7}$ $\hat{\mathbf{v}}_{6,7}$
$\hat{\mathbf{n}}_{5,7}$ $\hat{\mathbf{u}}_{5,7}$ $\hat{\mathbf{v}}_{5,7}$
$\hat{\mathbf{n}}_{5,6}$ $\hat{\mathbf{u}}_{5,6}$ $\hat{\mathbf{v}}_{5,6}$
$\hat{\mathbf{n}}_{9,10}$ $\hat{\mathbf{u}}_{9,10}$ $\hat{\mathbf{v}}_{9,10}$
$\hat{\mathbf{n}}_{4,5}$ $\hat{\mathbf{u}}_{4,5}$ $\hat{\mathbf{v}}_{4,5}$
$\hat{\mathbf{n}}_{0,4}$ $\hat{\mathbf{u}}_{0,4}$ $\hat{\mathbf{v}}_{0,4}$
$\hat{\mathbf{n}}_{1,3}$ $\hat{\mathbf{u}}_{1,3}$ $\hat{\mathbf{v}}_{1,3}$
$\hat{\mathbf{n}}_{2,9}$ $\hat{\mathbf{u}}_{2,9}$ $\hat{\mathbf{v}}_{2,9}$
$\hat{\mathbf{z}}$ $\hat{\mathbf{y}}$ $\hat{\mathbf{x}}$


v_8
$e_{7,8}$
$e_{6,8}$
v_{10}
v_6
v_7
$e_{5,6}$
$e_{5,7}$
v_3
$e_{4,5}$
v_5
$e_{9,10}$
$e_{3,4}$
v_4
$e_{5,9}$
$e_{1,3}$
$e_{0,4}$
v_9
$e_{2,9}$
v_0
v_1
v_2
$\hat{z}$
$\hat{y}$
$\hat{x}$

Creating your own package

ROBOTIC FABRICATION

Robotic Fabrication, and ROS Concepts

COMPAS FAB

compas_ags Algebraic Graph Statics	**compas_tna** Thrust Network Analysis	**compas_3gs** 3D Graphic Statics	**compas_fea** Finite Element Analysis	**compas_fab** Robotic Fabrication
compas_assembly Block Assemblies	**compas_knit** Spatial Knitting	**compas_dem** Discrete Element Modeling (3DEC)	**compas_dr6** 6DOF Dynamic Relaxation	**compas_rbe** Rigid Block Equilibrium
compas_ml Machine Learning	**compas_bend** Bending Active Structures	**compas_rocking** Collapse Mechanisms	**compas_ctrl** Active On-site Control	**compas_pattern** Mesh Topology Finding

Overview

Open source, python package to facilitate the **planning and execution of robotic fabrication processes**.

Provides interfaces to existing software libraries and tools available in the field of robotics and makes them accessible from within the parametric design environment.

Robotic fabrication

compas_fab

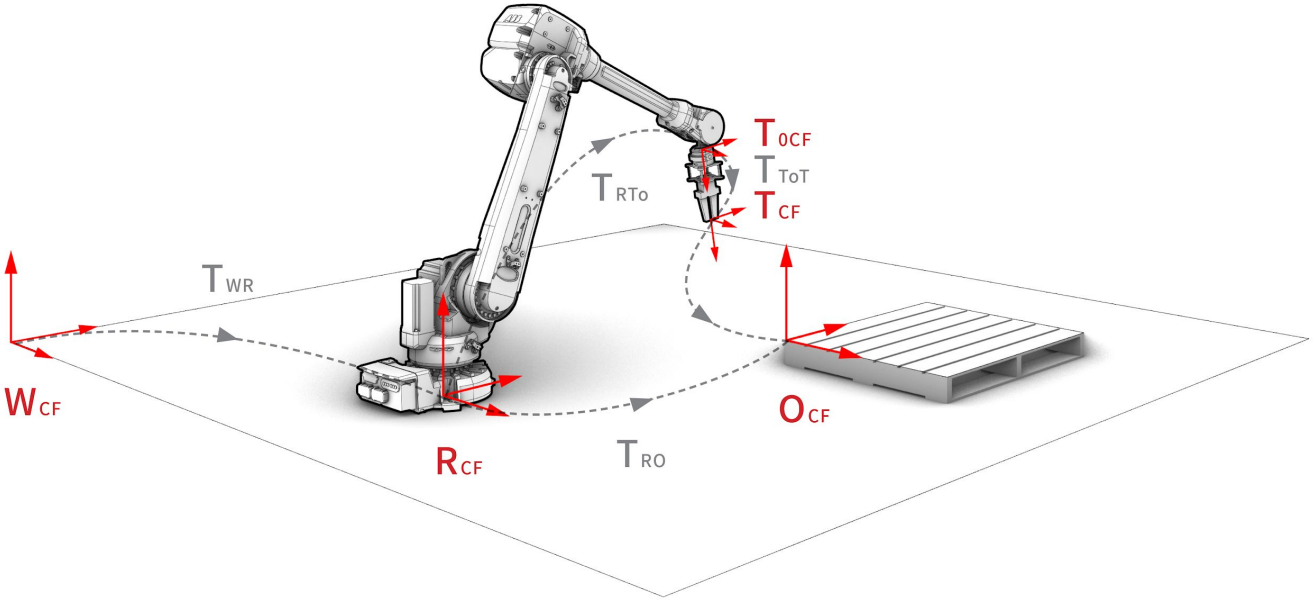
- Fundamentals
- Description models
- Backends

- Planning & execution tools
- CAD-independent
- Robotic fabrication process modeling
- Mediates between tools & libraries
- Multiple robotic backends

```
conda install compas_fab
```

Robotic fabrication: fundamentals

Frame and **Transformation** are two fundamental classes provided by the library to describe position/orientation and coordinate systems.



```python
from compas.geometry import Point
from compas.geometry import Vector
from compas.geometry import Frame
from compas.geometry import Transformation

# Robot coordinate frame
RCF = Frame(Point(1., -1., 0.),
            Vector(0.707, -0.707, 0.),
            Vector(0.707, 0.707, 0.))
# Object coordinate frame
OCF = Frame(Point(3., -3., 0.),
            Vector(0.456, 0.890, 0.),
            Vector(-0.890, 0.456, 0.))
# Frame defined in OCF
frame_OCF = Frame(Point(0.1, 0.1, 0.1)
                  Vector(1, 0, 0), Vector(0, 1, 0))
# Frame expressed in WCF
frame_WCF =
OCF.represent_frame_in_global_coordinates(frame_OCF)
# Frame expressed in RCF
frame_RCF =
RCF.represent_frame_in_local_coordinates(frame_WCF)
```

Better example !!

```python
# Tool center frame, defined in the T0CF
TCF = Frame(Point(0., 0., 0.3),
            Vector(1, 0, 0.),
            Vector(0, 1, 0.))
# The transformation from TCF to T0CF
TC2T0 =
Transformation.from_frame_to_frame(Frame.worldXY(), TCF)

frame_WCF_T0 = frame_WCF.transformed(TC2T0)

# Frame defined in RCF
frame_RCF_T0 =
RCF.represent_frame_in_local_coordinates(frame_WCF_T0)
```

```python
from compas.geometry import Point
from compas.geometry import Vector
from compas.geometry import Frame
from compas.geometry import Transformation

# Robot coordinate frame
RCF = Frame(Point(1., -1., 0.),
            Vector(0.707, -0.707, 0.),
            Vector(0.707, 0.707, 0.))
# Object coordinate frame
OCF = Frame(Point(3., -3., 0.),
            Vector(0.456, 0.890, 0.),
            Vector(-0.890, 0.456, 0.))
# Frame defined in OCF
frame_OCF = Frame(Point(0.1, 0.1, 0.1)
                  Vector(1, 0, 0), Vector(0, 1, 0))
# Frame expressed in WCF
frame_WCF =
OCF.represent_frame_in_global_coordinates(frame_OCF)
# Frame expressed in RCF
frame_RCF =
RCF.represent_frame_in_local_coordinates(frame_WCF)
```

Better example !!

```python
# Tool center frame, defined in the T0CF
TCF = Frame(Point(0., 0., 0.3),
            Vector(1, 0, 0.),
            Vector(0, 1, 0.))
# The transformation from TCF to T0CF
TC2T0 =
Transformation.from_frame_to_frame(Frame.worldXY(), TCF)

frame_WCF_T0 = frame_WCF.transformed(TC2T0)

# Frame defined in RCF
frame_RCF_T0 =
RCF.represent_frame_in_local_coordinates(frame_WCF_T0)
```

```python
from compas.geometry import Point
from compas.geometry import Vector
from compas.geometry import Frame
from compas.geometry import Transformation

# Robot coordinate frame
RCF = Frame(Point(1., -1., 0.),
            Vector(0.707, -0.707, 0.),
            Vector(0.707, 0.707, 0.))
# Object coordinate frame
OCF = Frame(Point(3., -3., 0.),
            Vector(0.456, 0.890, 0.),
            Vector(-0.890, 0.456, 0.))
# Frame defined in OCF
frame_OCF = Frame(Point(0.1, 0.1, 0.1)
                  Vector(1, 0, 0), Vector(0, 1, 0))
# Frame expressed in WCF
frame_WCF =
OCF.represent_frame_in_global_coordinates(frame_OCF)
# Frame expressed in RCF
frame_RCF =
RCF.represent_frame_in_local_coordinates(frame_WCF)
```

Better example !!

```python
# Tool center frame, defined in the T0CF
TCF = Frame(Point(0., 0., 0.3),
            Vector(1, 0, 0.),
            Vector(0, 1, 0.))
# The transformation from TCF to T0CF
TC2T0 =
Transformation.from_frame_to_frame(Frame.worldXY(), TCF)

frame_WCF_T0 = frame_WCF.transformed(TC2T0)

# Frame defined in RCF
frame_RCF_T0 =
RCF.represent_frame_in_local_coordinates(frame_WCF_T0)
```

Exercise

Better example !!

```
from compas.geometry import Point, Vector, Frame, Transformation

F1 = Frame((1, -1, 0),(0.707, -0.707, 0.), (0.707, 0.707, 0.))
F2 = Frame((3, -3, 0), (0.456, 0.890, 0.), (-0.890, 0.456, 0.))
frame_WCF = Frame((1.141, -1, 0.1),(0.707, -0.707, 0.), (0.707, 0.707, 0.))
```

Exercise

```
from compas.geometry import Point, Vector, Frame, Transformation

F1 = Frame((1, -1, 0),(0.707, -0.707, 0.), (0.707, 0.707, 0.))
F2 = Frame((3, -3, 0), (0.456, 0.890, 0.), (-0.890, 0.456, 0.))
frame_WCF = Frame((1.141, -1, 0.1),(0.707, -0.707, 0.), (0.707, 0.707, 0.))

# Represent frame_WCF in F1

# If this frame was in F2, what would it world coordinates be?
```

Exercise

```
from compas.geometry import Point, Vector, Frame, Transformation

F1 = Frame((1, -1, 0),(0.707, -0.707, 0.), (0.707, 0.707, 0.))
F2 = Frame((3, -3, 0), (0.456, 0.890, 0.), (-0.890, 0.456, 0.))
frame_WCF = Frame((1.141, -1, 0.1),(0.707, -0.707, 0.), (0.707, 0.707, 0.))

# Represent frame_WCF in F1
frame_F1 = F1.represent_frame_in_local_coordinates(frame_WCF)
# If this frame was in F2, what would it world coordinates be?
```

Exercise

```python
from compas.geometry import Point, Vector, Frame, Transformation

F1 = Frame((1, -1, 0),(0.707, -0.707, 0.), (0.707, 0.707, 0.))
F2 = Frame((3, -3, 0), (0.456, 0.890, 0.), (-0.890, 0.456, 0.))
frame_WCF = Frame((1.141, -1, 0.1),(0.707, -0.707, 0.), (0.707, 0.707, 0.))

# Represent frame_WCF in F1
frame_F1 = F1.represent_frame_in_local_coordinates(frame_WCF)
# If this frame was in F2, what would it world coordinates be?
frame_WCF_F2 = F2.represent_frame_in_global_coordinates(frame_F1)
print("frame_WCF_F2", frame_WCF_F2)
```

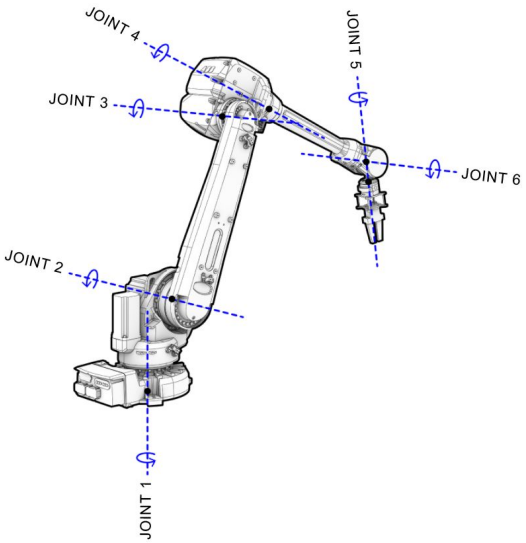
Exercise

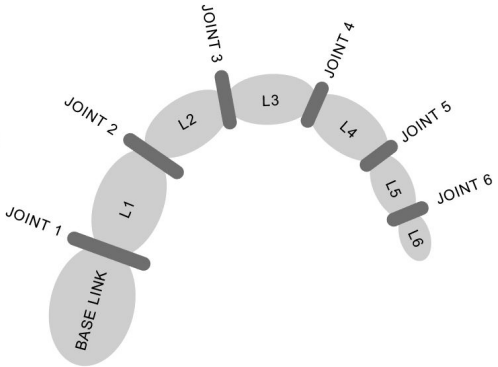
```python
from compas.geometry import Point, Vector, Frame, Transformation

F1 = Frame((1, -1, 0),(0.707, -0.707, 0.), (0.707, 0.707, 0.))
F2 = Frame((3, -3, 0), (0.456, 0.890, 0.), (-0.890, 0.456, 0.))
frame_WCF = Frame((1.141, -1, 0.1),(0.707, -0.707, 0.), (0.707, 0.707, 0.))

# Represent frame_WCF in F1
frame_F1 = F1.represent_frame_in_local_coordinates(frame_WCF)
# If this frame was in F2, what would it world coordinates be?
frame_WCF_F2 = F2.represent_frame_in_global_coordinates(frame_F1)
print("frame_WCF_F2", frame_WCF_F2)

# Alternative solution
T1 = Transformation.from_frame_to_frame(F1, Frame.worldXY())
T2 = Transformation.from_frame(F2)
print("frame_WCF_F2", frame_WCF.transformed(T2 * T1))
```

Orientation specs

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

Robotic fabrication: models





- Links connected by Joints
- Coordinate frames at joints
- Tree structure
- URDF
  - Kinematics & Dynamics
  - Visual representation
  - Collision representation
  - Semantics

Robotic fabrication: models

compas.robots

- Robot
  - Link
    - Inertial
    - Visual
    - Collision
  - Joint
    - Origin
    - Parent
    - Child
    - Dynamics

```python
# Load robot model
from compas.robots import RobotModel

RobotModel.from_urdf_file('ur5.urdf')
```

```xml
<?xml version="1.0" ?>
<robot name="rfl" xmlns:xacro="http://wiki.ros.org/xacro">
  <link name="rail">
    <inertial>
      <origin xyz="0.0 0.0 0.0"/>
      <mass value="120"/>
      <inertia ixx="-4.98" ixy="-0.0" ixz="2.7" iyy="-5.95" iyz="-0.0" izz="0.828"/>
    </inertial>
    <visual>
      <geometry>
        <mesh filename="package://rfl/meshes/visual/rail.stl"/>
      </geometry>
    </visual>
    <collision>
      <geometry>
        <mesh filename="package://rfl/meshes/collision/rail.stl"/>
      </geometry>
    </collision>
  </link>
<joint name="gantry_joint" type="prismatic">
    <parent link="rail"/>
    <child link="gantry"/>
    <origin rpy="0 0 0 " xyz="0 0 0"/>
    <axis xyz="1 0 0"/>
    <limit effort="100000" lower="0.0" upper="37.206" velocity="2.618"/>
    <dynamics damping="0.2" friction="0"/>
  </joint>
...
```

```python
# Rhino
from compas.robots import *
from compas_fab.rhino import RobotArtist

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)

RobotArtist(model).draw_visual()
```

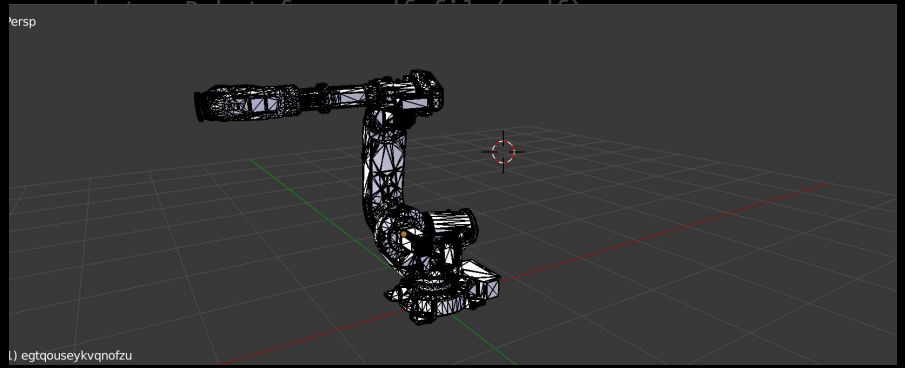
```python
# Blender
from compas.robots import *
from compas_fab.blender import RobotArtist

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)

RobotArtist(model).draw_visual()
```

```python
# Rhino
from compas.robots import *
from compas_fab.rhino import RobotArtist

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

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

robot = Robot.from_urdf_file(urdf)
robot.load_geometry(github)

RobotArtist(robot).draw_visual()
```

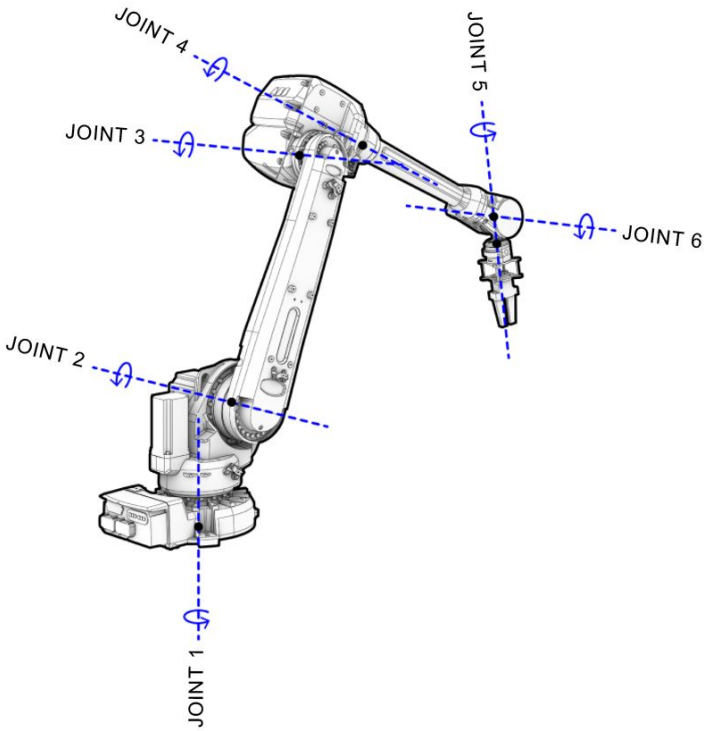
```python
# Blender
from compas.robots import *
from compas_fab.blender import RobotArtist

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

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

robot = Robot.from_urdf_file(urdf)
robot.load_geometry(github)

RobotArtist(robot).draw_visual()
```

```python
# Rhino
from compas.robots import *
from compas_fab.rhino import RobotArtist

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

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

```python
# Blender
from compas.robots import *
from compas_fab.blender import RobotArtist

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

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

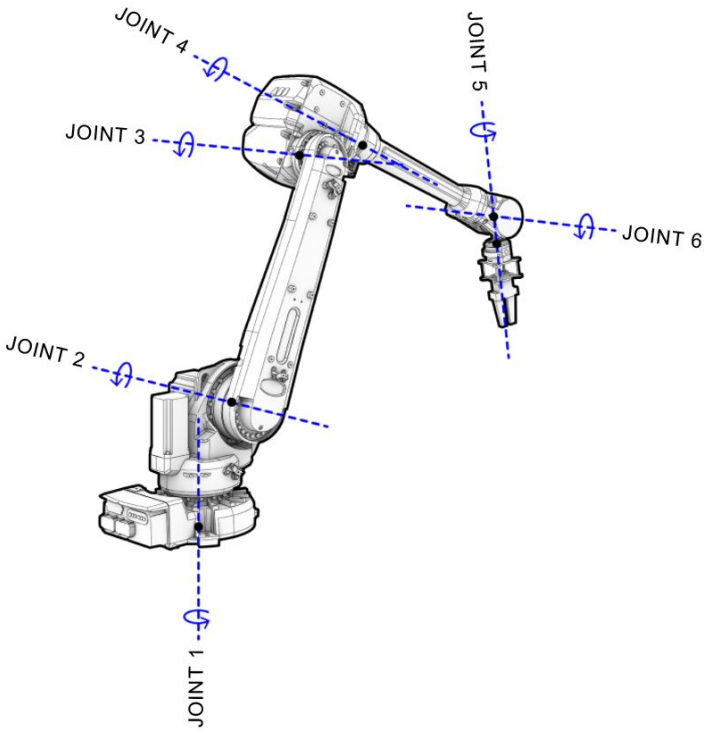


Configuration



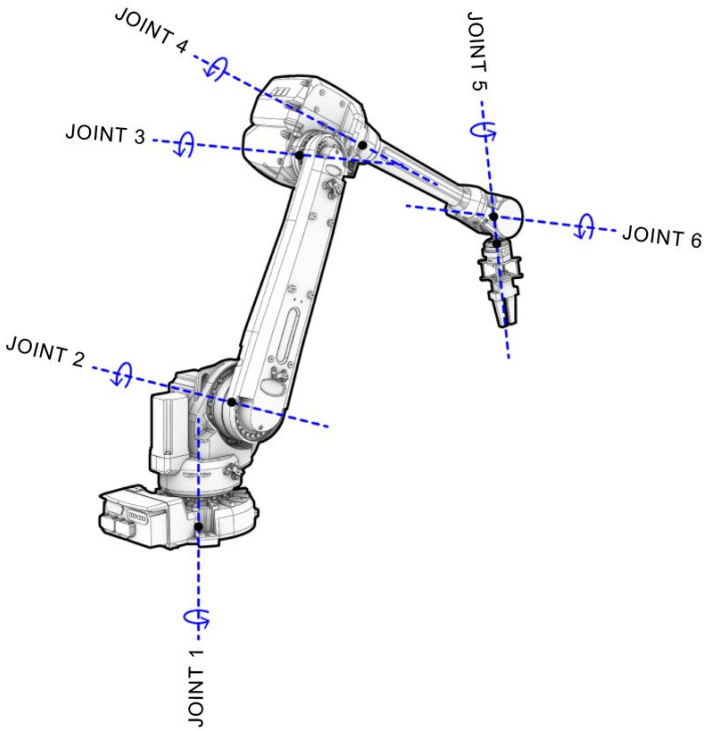

```python
from compas.robots import Joint

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_value
s([8.312], [pi/2, 0., 0., 0., 2*pi, 0.8])
```

Configuration




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

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

Robotic fabrication: backends

V-REP
Robot simulator

ROS
Robot Operating System

Moveit!
Motion Planning Framework

ROS

Open Source Robotics Foundation

What is ROS?

plumbing

- Process management
- Interprocess communication
- Device drivers

tools

- Simulation
- Visualization
- Graphical UI
- Data logging

capabilities

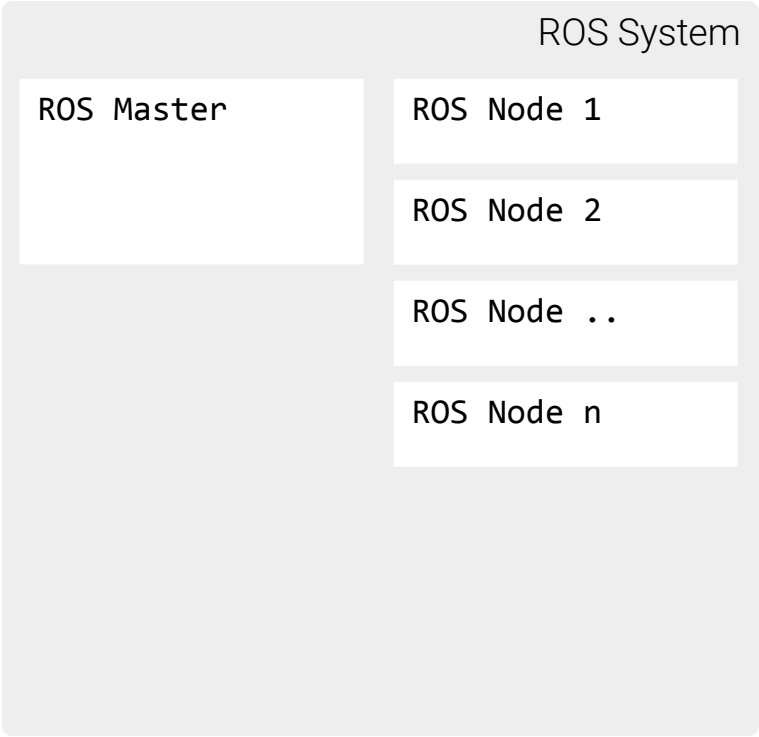
- Control
- Planning
- Perception
- Mapping
- Manipulation

ecosystem

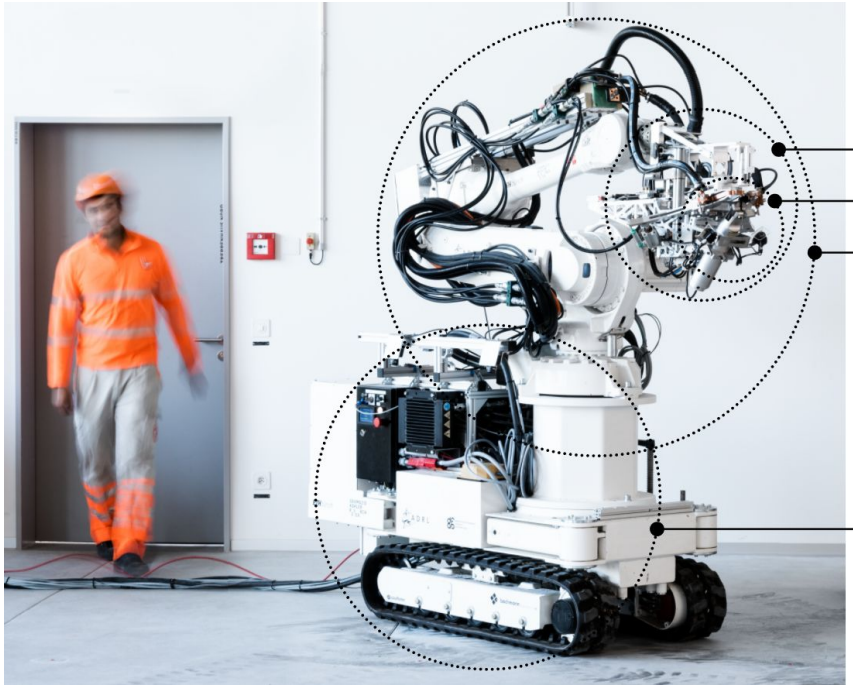
- Package organization
- Software distribution
- Community
- Documentation
- Tutorials

What is ROS?

ROS is an open source middleware for robot software development providing services such as hardware abstraction, low-level device control, message-passing, package management, etc.



ROS System



ROS Node 1: End Effector

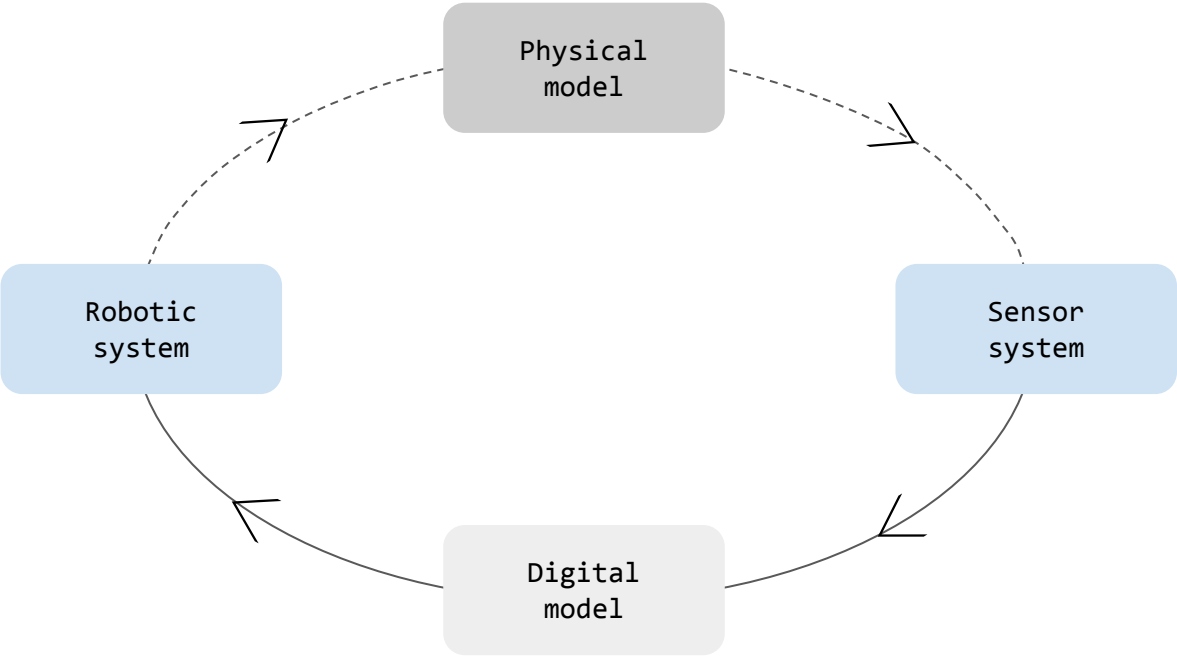
ROS Node 2: Localization

ROS Node 3: Robot Arm

ROS Node 4: Mobile Base

Shared vocabulary

Why ROS?


Physical model
Sensor system
Digital model
Robotic system

ROS Concepts

Communication models

Blocking vs Non-blocking (async)

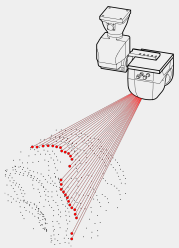
Publish/Subscribe

Request/Response

ROS Concepts

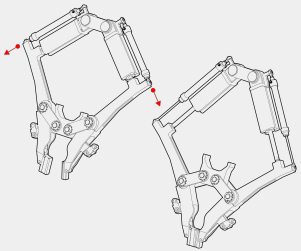
Topics

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



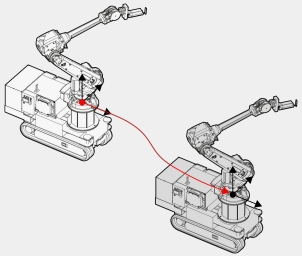
Services

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

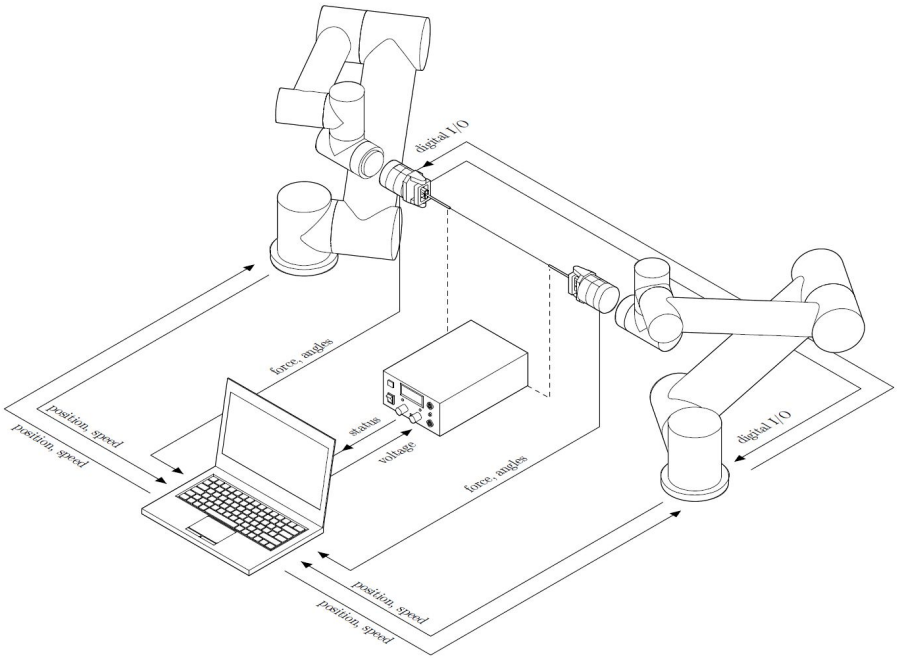


Actions

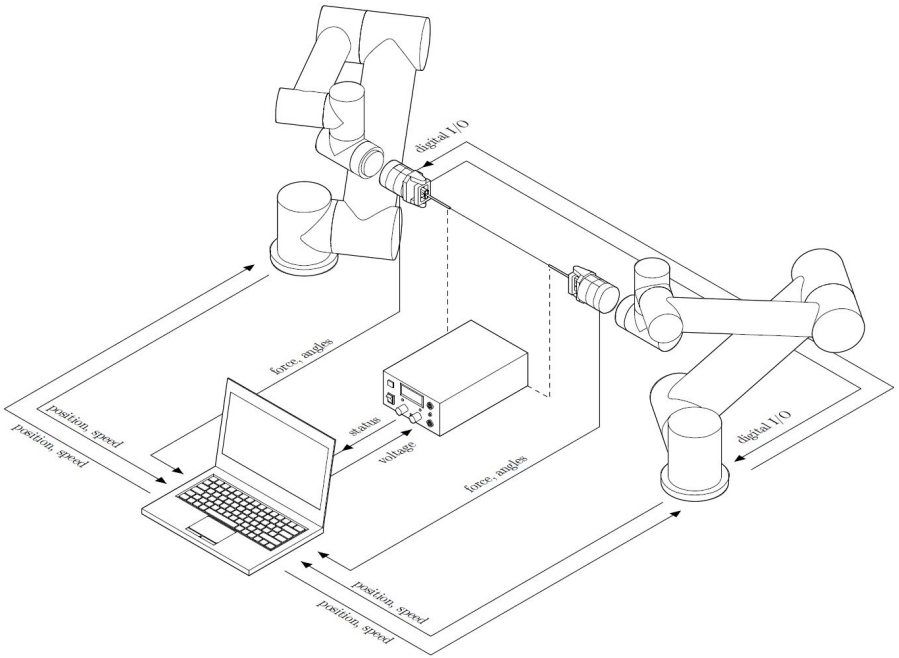
- **Non-blocking** requests
- Goal-oriented and cancellable
- Implemented with topics
- e.g: *navigation, motion execution*

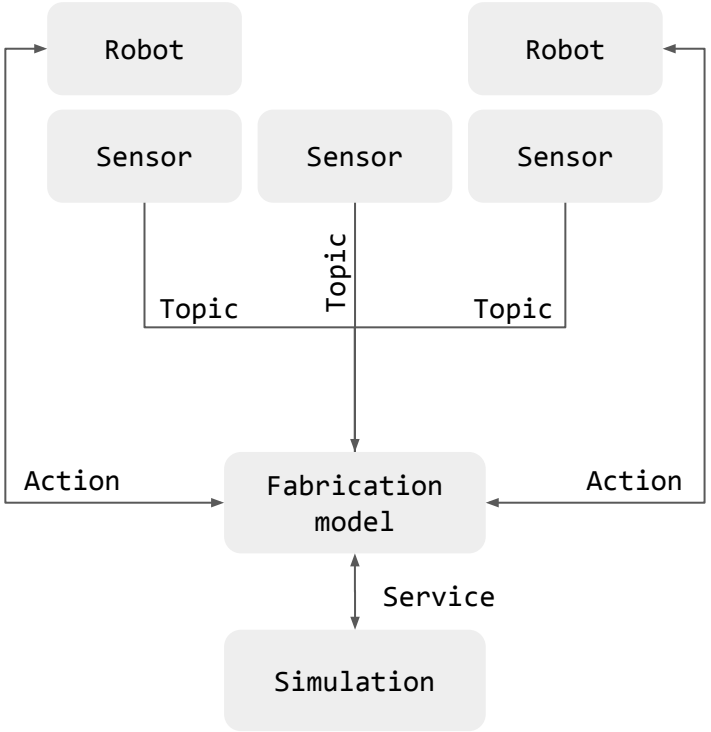


ROS fabrication example



ROS fabrication example





Exercise: ROS topics

```
client = RosClient()
client.run()

topic = Topic(client, '/messages',
              'std_msgs/String')
topic.advertise()

while True:
   topic.publish(dict(data='Hello world'))
   time.sleep(1)
```

Exercise: ROS topics

```
client = RosClient()
client.run()

topic = Topic(client, '/messages',
              'std_msgs/String')
topic.advertise()

while True:
   topic.publish(dict(data='Hello world'))
   time.sleep(1)
```

Exercise: ROS topics

```
client = RosClient()
client.run()

topic = Topic(client, '/messages',
              'std_msgs/String')
topic.advertise()

while True:
   topic.publish(dict(data='Hello world'))
   time.sleep(1)
```

Exercise: ROS topics

```python
import time
from roslibpy import Topic
from compas_fab.backends import RosClient

client = RosClient()
client.run()

topic = Topic(client, '/messages',
              'std_msgs/String')
topic.advertise()

while True:
   topic.publish(dict(data='Hello world'))
   time.sleep(1)

client.terminate()
```

```python
import time
from roslibpy import Topic
from compas_fab.backends import RosClient

client = RosClient()
client.run()

topic = Topic(client, '/messages',
              'std_msgs/String')
topic.subscribe(print)

while True:
   time.sleep(1)


client.terminate()
```

ROS Protocol

Message types

Service types

Parameters

ROS Protocol: Message types

- Dictionary-like data structure
- Defines the type of topics
- Large collection of predefined types

ROS Protocol: Message types

- Dictionary-like data structure
- Defines the type of topics
- Large collection of predefined types

std_msgs/String Message

File: std_msgs/String.msg

Raw Message Definition

```
string data
```

Compact Message Definition

```
string data
```

http://docs.ros.org/api/std_msgs/html/msg/String.html

ROS Protocol: Message types

- Dictionary-like data structure
- Defines the type of topics
- Large collection of predefined types

```
>>> from compas_fab.backends.ros import String
>>> str_msg = String('hello world')
>>> print(str_msg.msg)
{'data': 'hello world'}
```

ROS Protocol: Message types

- Dictionary-like data structure
- Defines the type of topics
- Large collection of predefined types

geometry_msgs/Pose Message

File: geometry_msgs/Pose.msg

Raw Message Definition

```
# A representation of pose in free space, composed of position and orientation.
Point position
Quaternion orientation
```

Compact Message Definition

```
geometry_msgs/Point position
geometry_msgs/Quaternion orientation
```

http://docs.ros.org/kinetic/api/geometry_msgs/html/msg/Pose.html

ROS Protocol: Message types

- Dictionary-like data structure
- Defines the type of topics
- Large collection of predefined types

```
>>> from compas.geometry import Frame
>>> from compas_fab.backends.ros import Pose
>>> frame = Frame.worldXY()
>>> pose = Pose.from_frame(frame)
>>> print(pose.msg)
{'position': {'x': 0.0, 'y': 0.0, 'z': 0.0},
'orientation': {'x': 0.0, 'y': 0.0, 'z': 0.0,
'w': 1.0}}
```

ROS Protocol: Service types

- Similar to message types
- Define request and response types
- Example: service sum() takes two integers and returns one integer

ROS Protocol: Service types

- Similar to message types
- Define request and response types
- Example: service sum() takes two integers and returns one integer

moveit_msgs/GetPositionIK Service

File: moveit_msgs/GetPositionIK.srv

Raw Message Definition

```
# A service call to carry out an inverse kinematics computation
# The inverse kinematics request
PositionIKRequest ik_request

---

# The returned solution
# (in the same order as the list of joints specified in the IKRequest message)
RobotState solution

MoveItErrorCodes error_code
```

Compact Message Definition

```
moveit_msgs/PositionIKRequest ik_request
---
moveit_msgs/RobotState solution
moveit_msgs/MoveItErrorCodes error_code
```

http://docs.ros.org/kinetic/api/moveit_msgs/html/srv/GetPositionIK.html

ROS Protocol: Service types

IK request:


```
{'ik_request': {'group_name': 'manipulator',
'robot_state': {'joint_state': {'header': {'seq': 0,
'stamp': {'secs': 0.0, 'nsecs': 0.0}, 'frame_id':
'base_link'}, 'name': ['shoulder_pan_joint',
'shoulder_lift_joint', 'elbow_joint', 'wrist_1_joint',
'wrist_2_joint', 'wrist_3_joint'], 'position': [0, 0,
0, 0, 0, 0], 'velocity': [], 'effort': []},
'multi_dof_joint_state': {'header': {'seq': 0, 'stamp':
{'secs': 0.0, 'nsecs': 0.0}, 'frame_id': 'base_link'},
'joint_names': [], 'transforms': [], 'twist': [],
'wrench': []}, 'attached_collision_objects': [],
'is_diff': False}, 'constraints': {'name': '',
'joint_constraints': [], 'position_constraints': [],
'orientation_constraints': [],
'visibility_constraints': []}, 'avoid_collisions':
True, 'pose_stamped': {'header': {'seq': 0, 'stamp':
{'secs': 0.0, 'nsecs': 0.0}, 'frame_id': 'base_link'},
'pose': {'position': {'x': 0.3, 'y': 0.1, 'z': 0.5},
'orientation': {'x':
0.0, 'y': 0.0, 'z': 0.0, 'w': 1.0}}}, 'timeout': 1.0,
'attempts': 8}}
```


IK response:


```
{'solution': {'joint_state': {'header': {'seq': 0,
'stamp': {'secs': 0, 'nsecs': 0}, 'frame_id':
'/world'}, 'name': ['shoulder_pan_joint',
'shoulder_lift_joint', 'elbow_joint', 'wrist_1_joint',
'wrist_2_joint', 'wrist_3_joint'], 'position':
[6.240728287192657, -6.2498831197072455,
4.109588503042129, 5.281887273844756,
4.754846002166679, -6.2831853], 'velocity': [],
'effort': []}, 'multi_dof_joint_state': {'header':
{'stamp': {'secs': 0, 'nsecs': 0}, 'frame_id':
'/world', 'seq': 0}, 'joint_names': [], 'transforms':
[], 'twist': [], 'wrench': []},
'attached_collision_objects': [], 'is_diff': False},
'error_code': {'val': 1}}
```

ROS Protocol: Parameters

- Global parameter values
- Constant
- ROS Parameter server

ROS Concepts: protocol

TODO:
Reorganize so that it does not look like its connected with last page
Show message types

Message types

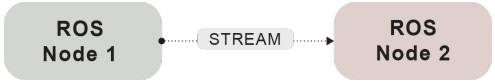
- Dictionary-like data structure
- Defines the type of topics
- Large collection of predefined types

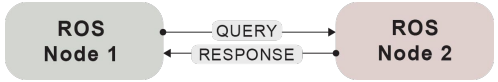
Service types

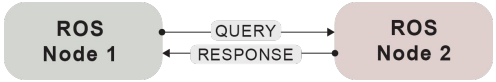
- Similar to message types
- Define request and response types
- Example: service sum() takes two integers and returns one integer

Parameters

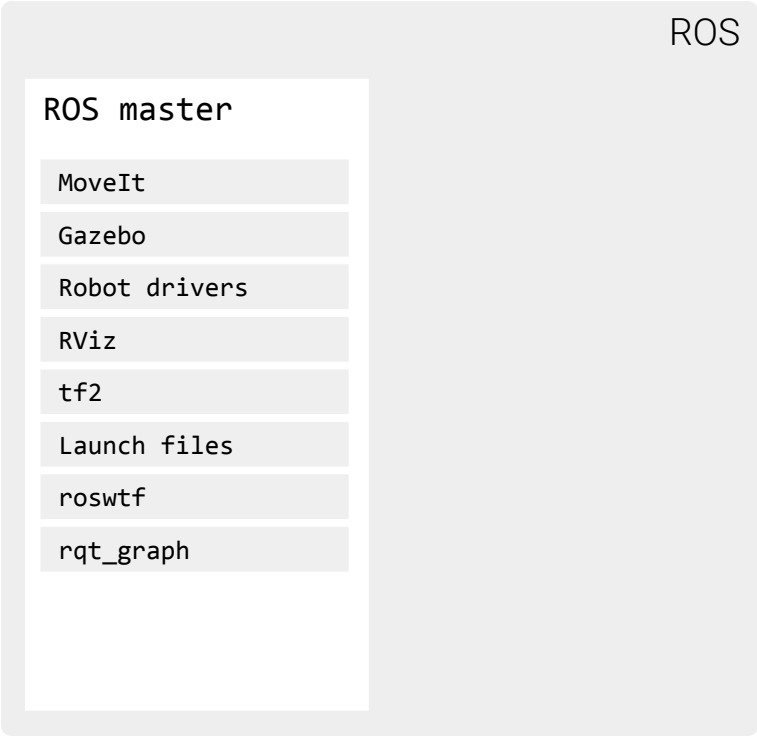
- Global parameter values
- Constant
- ROS Parameter server







ROS



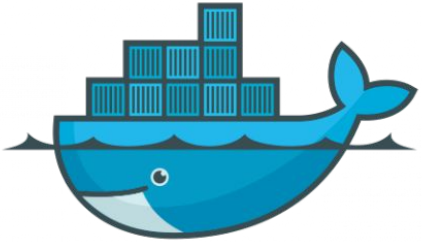
Very extensive tooling available:

- Motion planning framework
- Dynamics simulations
- Hardware abstraction
- Visualizations
- Transformation library

Ease distribution

python™ Package Index

CONDA


docker

```
docker-compose up
```

ROS in a box

`Basic`

Basic set of ROS nodes to play with.

`UR5`

UR5 model with nodes for path planning using fake controllers

`ABB Track`

ABB Linear Axis model with nodes for path planning using fake controllers

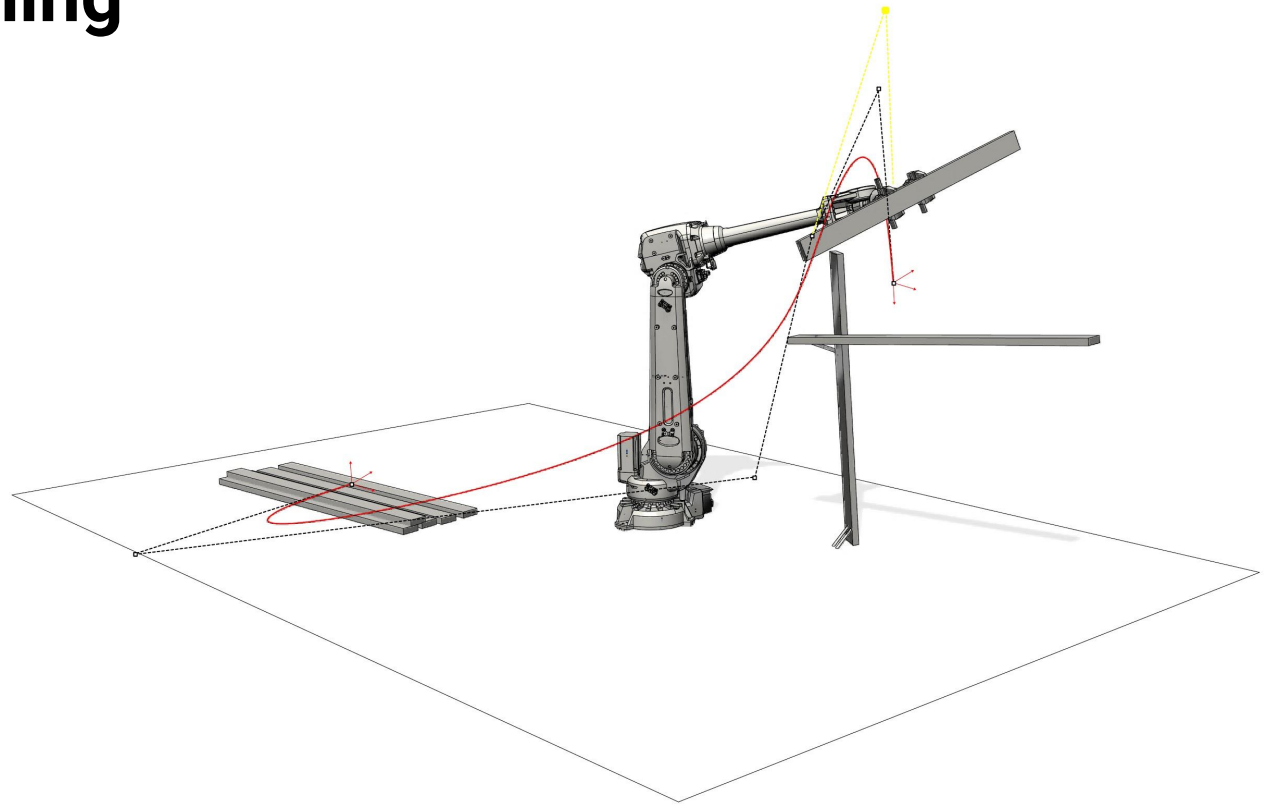
`ABB Track Real`

ABB Linear Axis model with nodes for path planning and driver for real control

PLANNING & SIMULATION

Path Planning, and Assembly Process Simulation

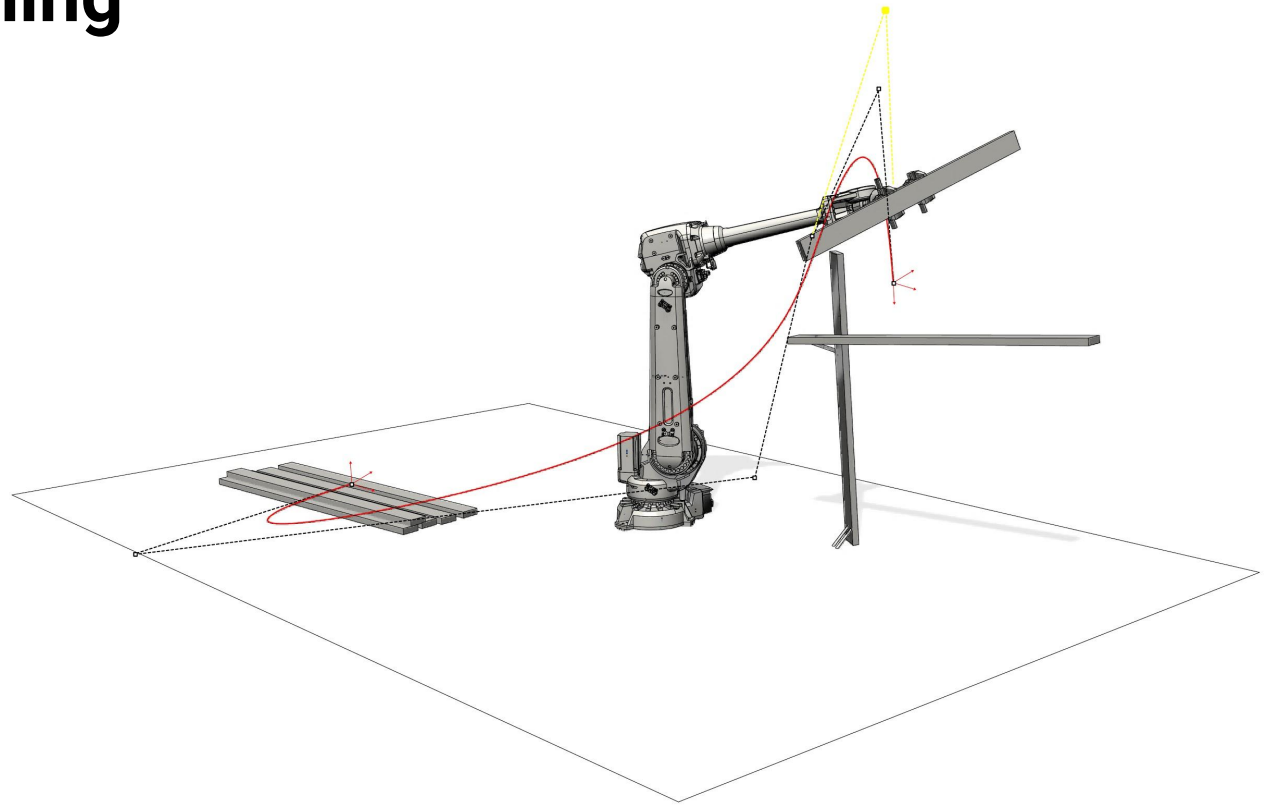
Path planning



Path planning

Collision checks

Trajectory checks



Path planning

Collision checks

- **Intricate positions (spatial assembly)**

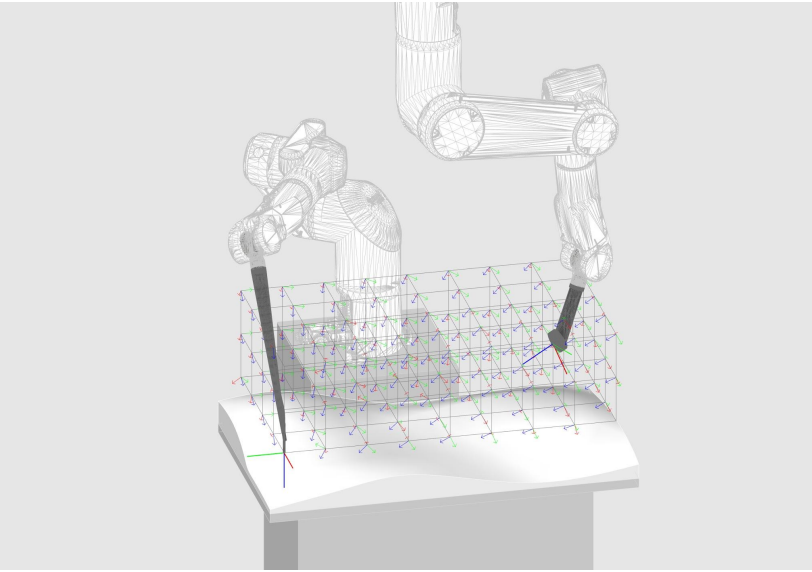
Trajectory checks

Path planning

Collision checks

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

Trajectory checks



Path planning

Collision checks

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

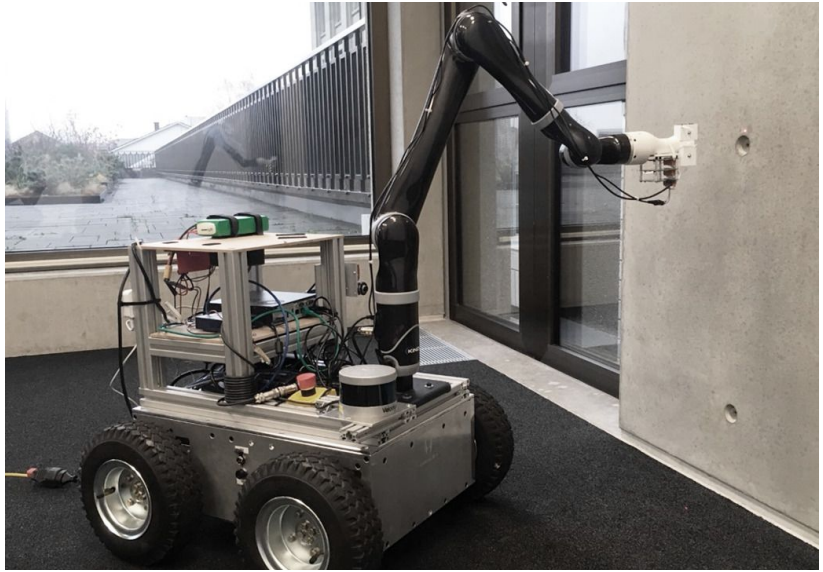
Trajectory checks

Path planning

Collision checks

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

Trajectory checks



WACO, “Construction Robotics”

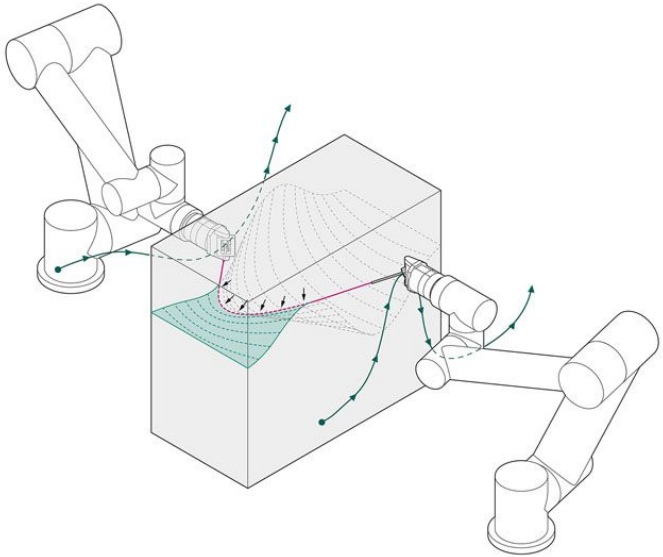
Path planning

Collision checks

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

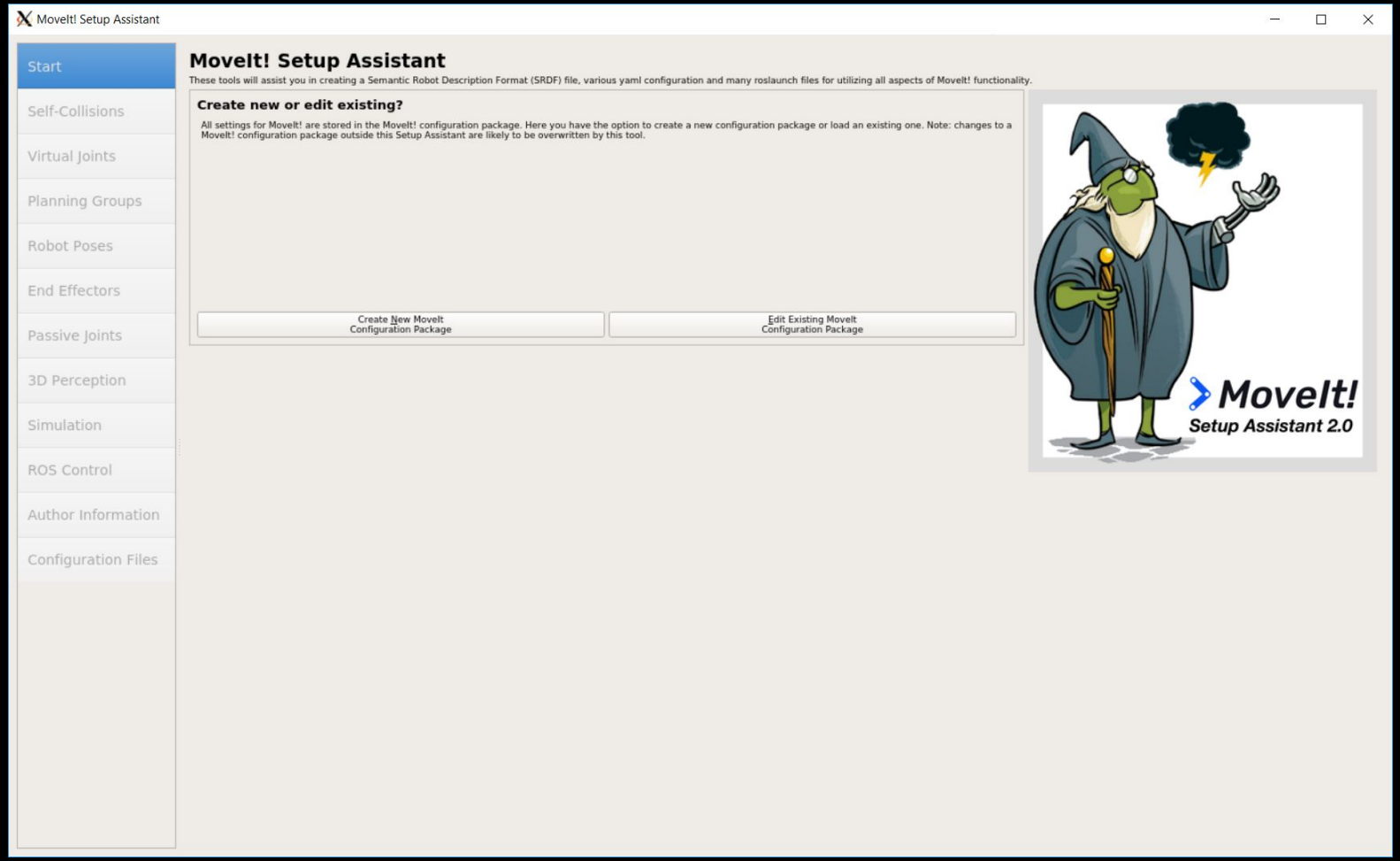
Trajectory checks

- **Continuous processes**



MoveIt!

- ROS' default motion planning library
- Works with different planners: OMPL (Open Motion Planning Library)
- Collision checking through FCL (Flexible Collision Library)
- Kinematics plugins (default KDL: Kinematics and Dynamics Library)
- Trajectory processing routine


MoveIt! Setup Assistant
Start
Self-Collisions
Virtual Joints
Planning Groups
Robot Poses
End Effectors
Passive Joints
3D Perception
Simulation
ROS Control
Author Information
Configuration Files
MoveIt! Setup Assistant
These tools will assist you in creating a Semantic Robot Description Format (SRDF) file, various yaml configuration and many roslaunch files for utilizing all aspects of MoveIt! functionality.
Create new or edit existing?
All settings for MoveIt! are stored in the MoveIt! configuration package. Here you have the option to create a new configuration package or load an existing one. Note: changes to a MoveIt! configuration package outside this Setup Assistant are likely to be overwritten by this tool.
Create New MoveIt Configuration Package
Edit Existing MoveIt Configuration Package
MoveIt!
Setup Assistant 2.0

MoveIt! Setup Assistant

- Start
- Self-Collisions
- Virtual Joints
- Planning Groups
- Robot Poses
- End Effectors
- Passive Joints
- 3D Perception
- Simulation
- ROS Control
- Author Information
- Configuration Files

Optimize Self-Collision Checking

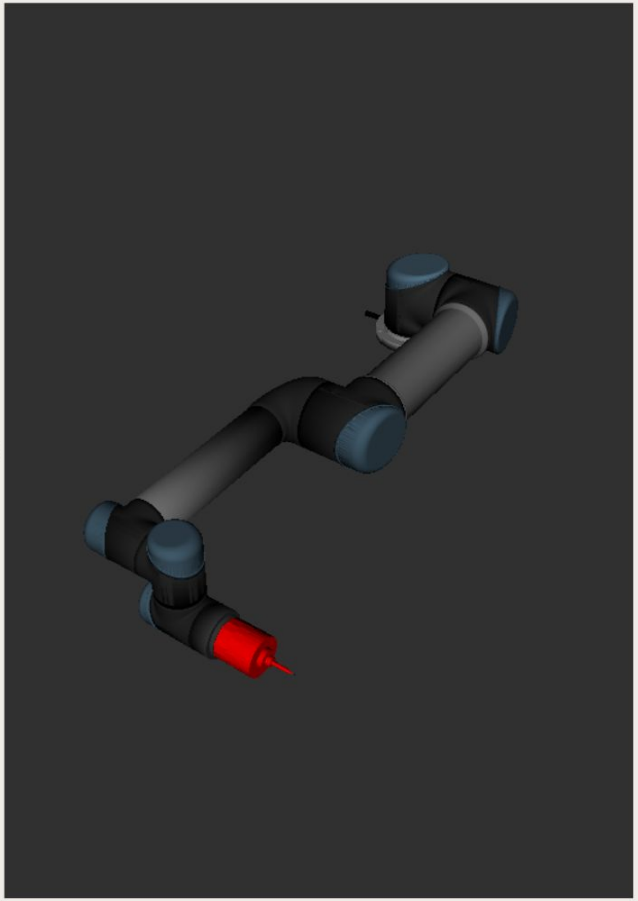
This searches for pairs of robot links that can safely be disabled from collision checking, decreasing motion planning time. These pairs are disabled when they are always in collision, never in collision, in collision in the robot's default position, or when the links are adjacent to each other on the kinematic chain. Sampling density specifies how many random robot positions to check for self collision.

Sampling Density: Low High 10000

Min. collisions for "always"-colliding pairs: 95% Generate Collision Matrix

	base_link	shoulder_link	upper_arm_link	forearm_link	wrist_1_link	wrist_2_link	wrist_3_link	ee_link	measurement_tool
base_link		✓							
shoulder_link	✓		✓						
upper_arm_link		✓		✓					
forearm_link			✓		✓				
wrist_1_link				✓		✓	✓	✓	✓
wrist_2_link					✓		✓	✓	✓
wrist_3_link					✓	✓		✓	✓
ee_link					✓	✓	✓		✓
measurement_tool					✓	✓	✓	✓	

link name filter linear view matrix view Revert



MoveIt! Setup Assistant

- Start
- Self-Collisions
- Virtual Joints
- Planning Groups
- Robot Poses
- End Effectors
- Passive Joints
- 3D Perception
- Simulation
- ROS Control
- Author Information
- Configuration Files

Define Planning Groups

Create and edit 'joint model' groups for your robot based on joint collections, link collections, kinematic chains or subgroups. A planning group defines the set of (joint, link) pairs considered for planning and collision checking. Define individual groups for each subset of the robot you want to plan for.Note: when adding a link to the group, its parent joint is added too and vice versa.

Edit Planning Group 'manipulator'

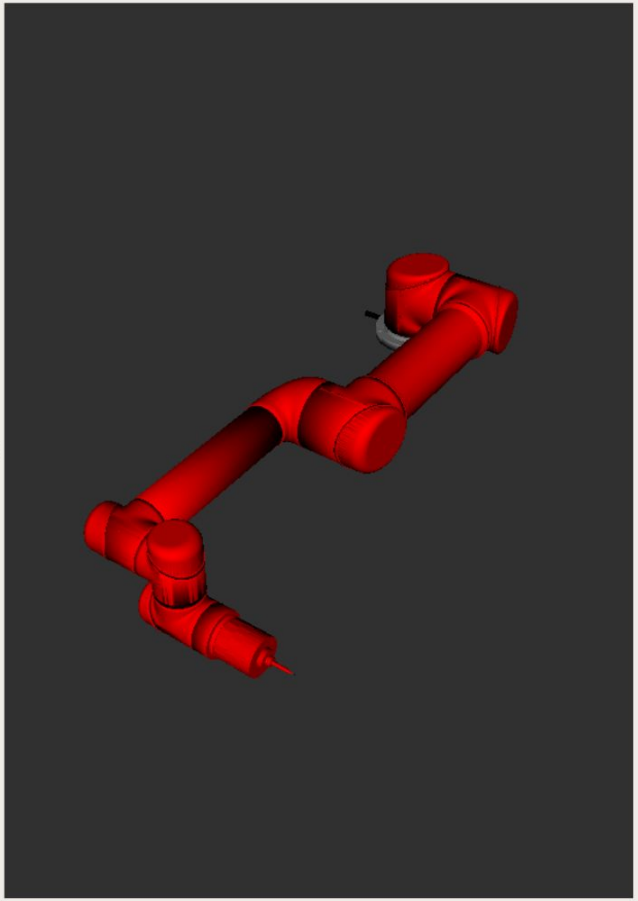
Kinematics

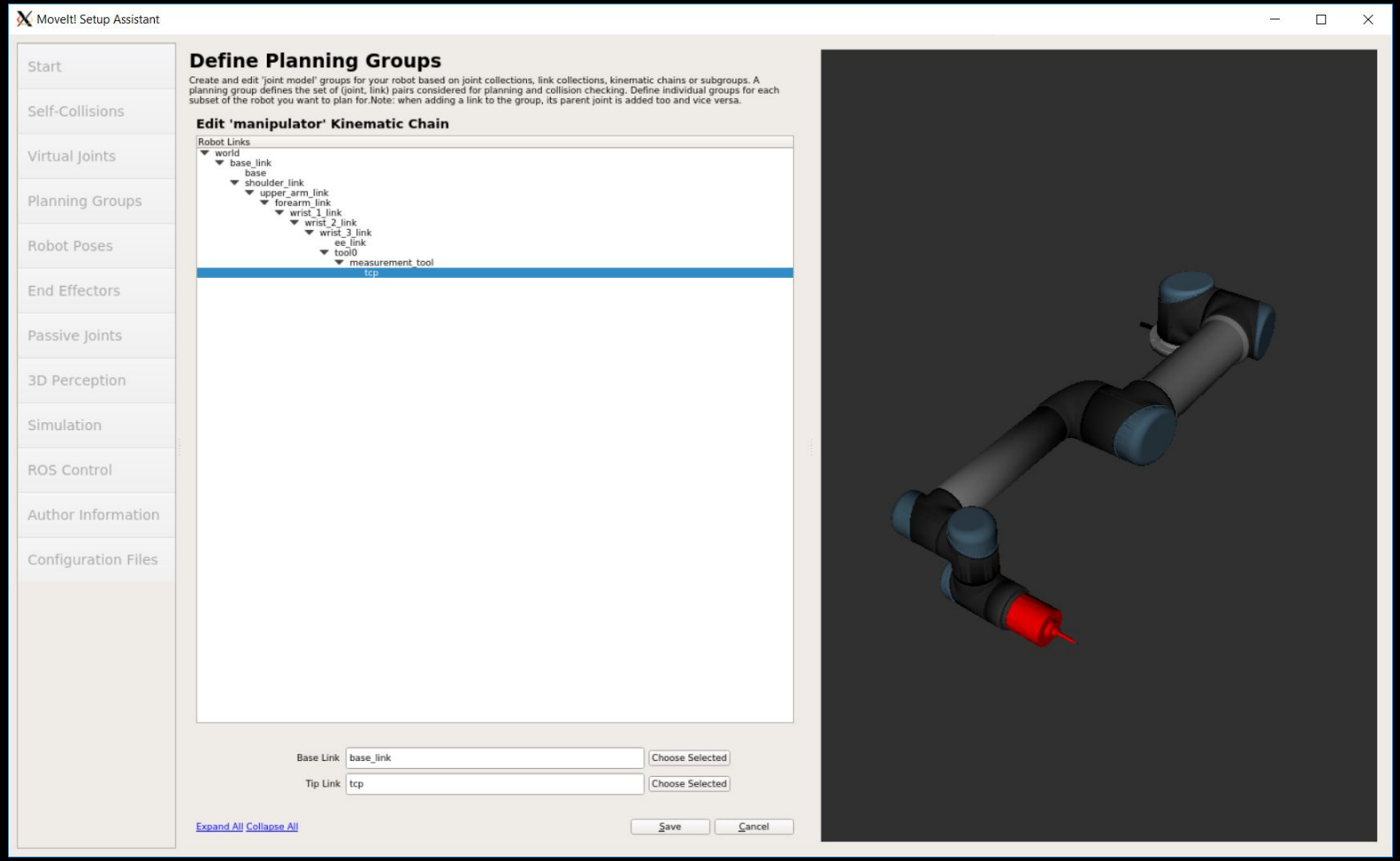
Group Name:	manipulator
Kinematic Solver:	ur_kinematics/UR5KinematicsPlugin
Kin. Search Resolution:	0.005
Kin. Search Timeout (sec):	0.005
Kin. Solver Attempts:	3

OMPL Planning

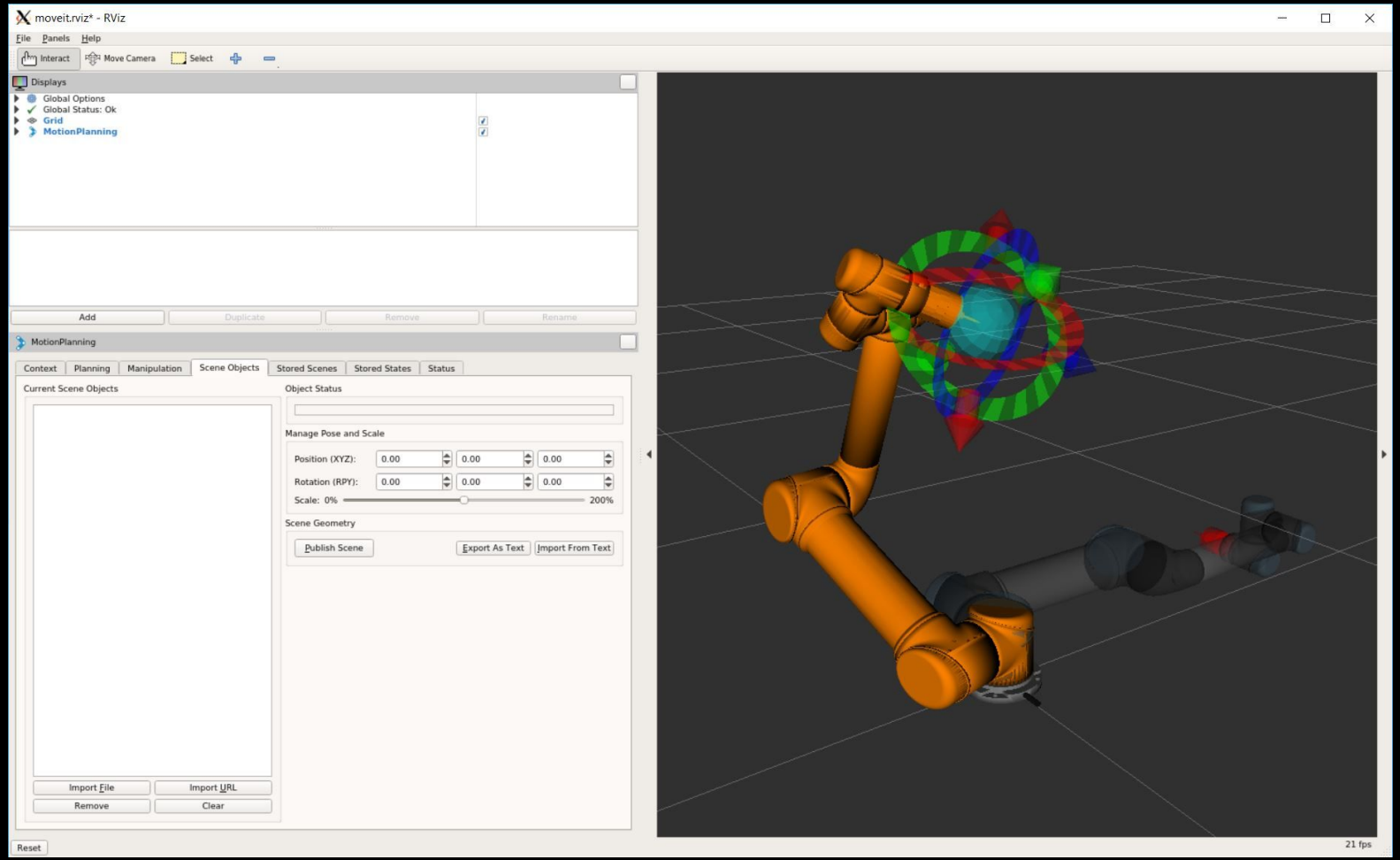
Group Default Planner:	None

Delete Group Save Cancel



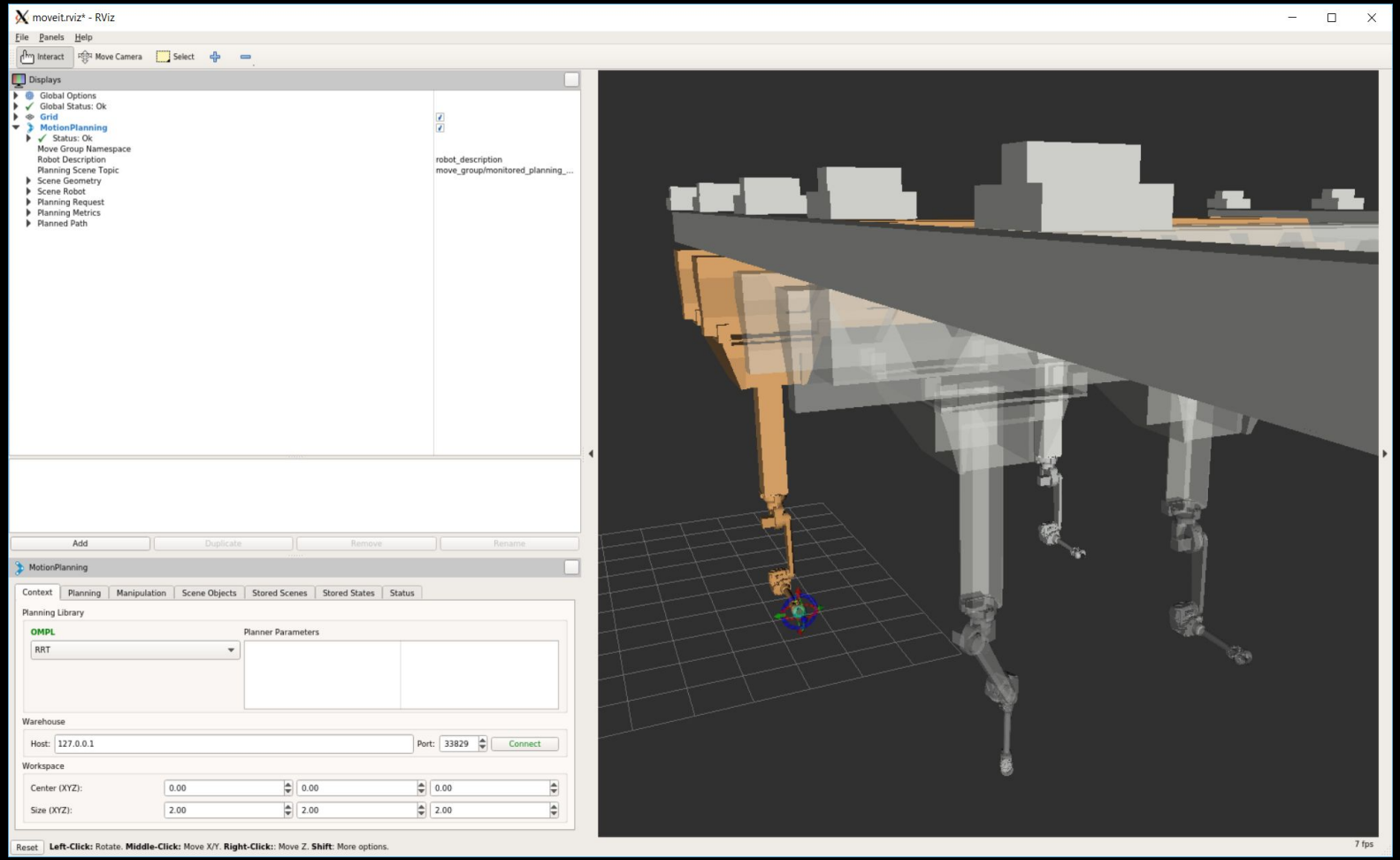

MoveIt! Setup Assistant
Start
Self-Collisions
Virtual Joints
Planning Groups
Robot Poses
End Effectors
Passive Joints
3D Perception
Simulation
ROS Control
Author Information
Configuration Files
Define Planning Groups
Create and edit 'joint model' groups for your robot based on joint collections, link collections, kinematic chains or subgroups. A planning group defines the set of (joint, link) pairs considered for planning and collision checking. Define individual groups for each subset of the robot you want to plan for.Note: when adding a link to the group, its parent joint is added too and vice versa.
Edit 'manipulator' Kinematic Chain
Robot Links
world
base_link
base
shoulder_link
upper_arm_link
forearm_link
wrist_1_link
wrist_2_link
wrist_3_link
ee_link
tool0
measurement_tool
tcp
Base Link
base_link
Choose Selected
Tip Link
tcp
Choose Selected
Expand All
Collapse All
Save
Cancel

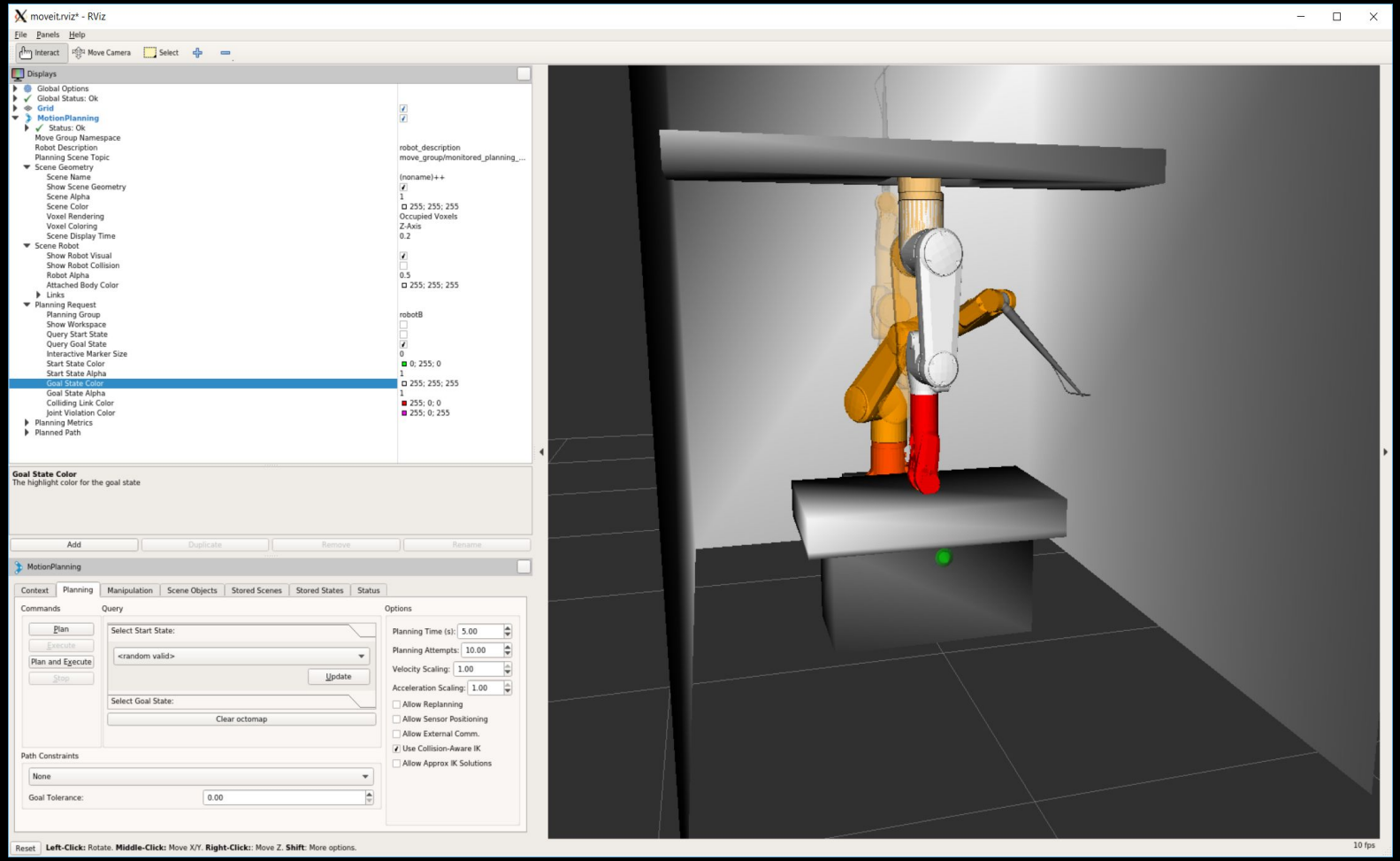
```
user@ITA:~$ roslaunch ur5_moveit_config demo.launch
```


moveit.rviz* - RViz
File
Panels
Help
Interact
Move Camera
Select
Displays
Global Options
Global Status: Ok
Grid
MotionPlanning
Add
Duplicate
Remove
Rename
MotionPlanning
Context
Planning
Manipulation
Scene Objects
Stored Scenes
Stored States
Status
Current Scene Objects
Object Status
Manage Pose and Scale
Position (XYZ):
0.00
0.00
0.00
Rotation (RPY):
0.00
0.00
0.00
Scale: 0%
200%
Scene Geometry
Publish Scene
Export As Text
Import From Text
Import File
Import URL
Remove
Clear
Reset
21 fps

```
user@ITA:~$ roslaunch ur5_moveit_config demo.launch
```

```
user@ITA:~$ rosservice list

/compute_fk
/compute_ik
/compute_cartesian_path
/plan_kinematic_path
/set_planner_params
/apply_planning_scene
...
```


moveit.rviz* - RViz
File
Panels
Help
Interact
Move Camera
Select
Displays
Global Options
Global Status: Ok
Grid
MotionPlanning
Status: Ok
Move Group Namespace
Robot Description
robot_description
Planning Scene Topic
move_group/monitored_planning_...
Scene Geometry
Scene Robot
Planning Request
Planning Metrics
Planned Path
Add
Duplicate
Remove
Rename
MotionPlanning
Context
Planning
Manipulation
Scene Objects
Stored Scenes
Stored States
Status
Planning Library
OMPL
RRT
Planner Parameters
Warehouse
Host:
127.0.0.1
Port:
33829
Connect
Workspace
Center (XYZ):
0.00
0.00
0.00
Size (XYZ):
2.00
2.00
2.00
Reset
Left-Click: Rotate. Middle-Click: Move X/Y. Right-Click:: Move Z. Shift: More options.
7 fps


moveit.rviz* - RViz
File Panels Help
Interact
Move Camera
Select
Displays
Global Options
Global Status: Ok
Grid
MotionPlanning
Status: Ok
Move Group Namespace
Robot Description
Planning Scene Topic
Scene Geometry
Scene Name
Show Scene Geometry
Scene Alpha
Scene Color
Voxel Rendering
Voxel Coloring
Scene Display Time
Scene Robot
Show Robot Visual
Show Robot Collision
Robot Alpha
Attached Body Color
Links
Planning Request
Planning Group
Show Workspace
Query Start State
Query Goal State
Interactive Marker Size
Start State Color
Start State Alpha
Goal State Color
Goal State Alpha
Colliding Link Color
Joint Violation Color
Planning Metrics
Planned Path
robot_description
move_group/monitored_planning_...
(noname)++
255; 255; 255
Occupied Voxels
Z-Axis
0.2
0.5
255; 255; 255
robotB
0
0; 255; 0
1
255; 255; 255
1
255; 0; 0
255; 0; 255
Goal State Color
The highlight color for the goal state
Add
Duplicate
Remove
Rename
MotionPlanning
Context
Planning
Manipulation
Scene Objects
Stored Scenes
Stored States
Status
Commands
Plan
Execute
Plan and Execute
Stop
Query
Select Start State:
<random valid>
Update
Select Goal State:
Clear octomap
Path Constraints
None
Goal Tolerance:
0.00
Options
Planning Time (s):
5.00
Planning Attempts:
10.00
Velocity Scaling:
1.00
Acceleration Scaling:
1.00
Allow Replanning
Allow Sensor Positioning
Allow External Comm.
Use Collision-Aware IK
Allow Approx IK Solutions
Reset
Left-Click: Rotate. Middle-Click: Move X/Y. Right-Click:: Move Z. Shift: More options.
10 fps

```xml
<?xml version="1.0" ?>
<robot name="ur5_with_measurement_tool">
   <group name="manipulator">
       <chain base_link="base_link" tip_link="tcp" />
   </group>
   <group_state name="home" group="manipulator">
       <joint name="elbow_joint" value="1.5707" />
       <joint name="shoulder_lift_joint" value="-1.5707" />
       <joint name="shoulder_pan_joint" value="0" />
       <joint name="wrist_1_joint" value="-1.5707" />
       <joint name="wrist_2_joint" value="-1.5707" />
       <joint name="wrist_3_joint" value="0" />
   </group_state>
   <end_effector name="measurement_tool" parent_link="tool0"
group="measurement_tool" />
   [...]
</robot>
```

```xml
<?xml version="1.0" ?>
<robot name="ur5_with_measurement_tool">
   <group name="manipulator">
       <chain base_link="base_link" tip_link="tcp" />
   </group>
   <group_state name="home" group="manipulator">
       <joint name="elbow_joint" value="1.5707" />
       <joint name="shoulder_lift_joint" value="-1.5707" />
       <joint name="shoulder_pan_joint" value="0" />
       <joint name="wrist_1_joint" value="-1.5707" />
       <joint name="wrist_2_joint" value="-1.5707" />
       <joint name="wrist_3_joint" value="0" />
   </group_state>
   <end_effector name="measurement_tool" parent_link="tool0"
group="measurement_tool" />
   [...]
</robot>
```

```
user@ITA:~$ cd workspace\robotic_assembly_workshop\docker\ros-systems\ros-ur5
```

```
user@ITA:~$ docker-compose up
```

Robot class in compas_fab

TODO !!

-model
-artist
-semantic
-client

```python
import os
import compas
from compas.robots import RobotModel
from compas_fab.robots import Robot
from compas_fab.robots import RobotSemantics
from compas_fab.backends import RosClient

compas.PRECISION = '12f'
HERE = os.path.dirname(__file__)

package = 'ur_description'
path = os.path.join(HERE, "robot_description")

urdf_filename = os.path.join(path, package, "urdf", "ur5.urdf")
srdf_filename = os.path.join(path, package, "ur5.srdf")

model = RobotModel.from_urdf_file(urdf_filename)
artist = None
semantics = RobotSemantics.from_srdf_file(srdf_filename, model)
client = RosClient('127.0.0.1', 9090)
client.run()

robot = Robot(model, artist, semantics, client)
```

```
>>> robot.info()

The robot's name is 'ur5'.
The planning groups are: ['manipulator', 'endeffector']
The main planning group is 'manipulator'.
The end-effector's name is 'ee_link'.
The base link's name is 'base_link'
The base_frame is: Frame(Point(0.0, 0.0, 0.0), Vector(1.0, 0.0, -0.0), Vector(0.0, 1.0,
0.0))
The robot's joints are:
        * 'shoulder_pan_joint' is of type 'revolute' and has limits [6.283, -6.283]
        * 'shoulder_lift_joint' is of type 'revolute' and has limits [6.283, -6.283]
        * 'elbow_joint' is of type 'revolute' and has limits [6.283, -6.283]
        * 'wrist_1_joint' is of type 'revolute' and has limits [6.283, -6.283]
        * 'wrist_2_joint' is of type 'revolute' and has limits [6.283, -6.283]
        * 'wrist_3_joint' is of type 'revolute' and has limits [6.283, -6.283]
```

Inverse kinematic

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

robot.client = RosClient('127.0.0.1', 9090)
robot.client.run()

frame = Frame([0.3, 0.1, 0.5], [1, 0, 0], [0, 1, 0])
start_configuration = Configuration.from_revolute_values([0] * 6)
group = "manipulator" # or robot.main_group_name

response = robot.inverse_kinematics(frame,
                                    start_configuration,
                                    group,
                                    constraints=None,
                                    attempts=50)
print(response.configuration)

robot.client.close()
robot.client.terminate()
```

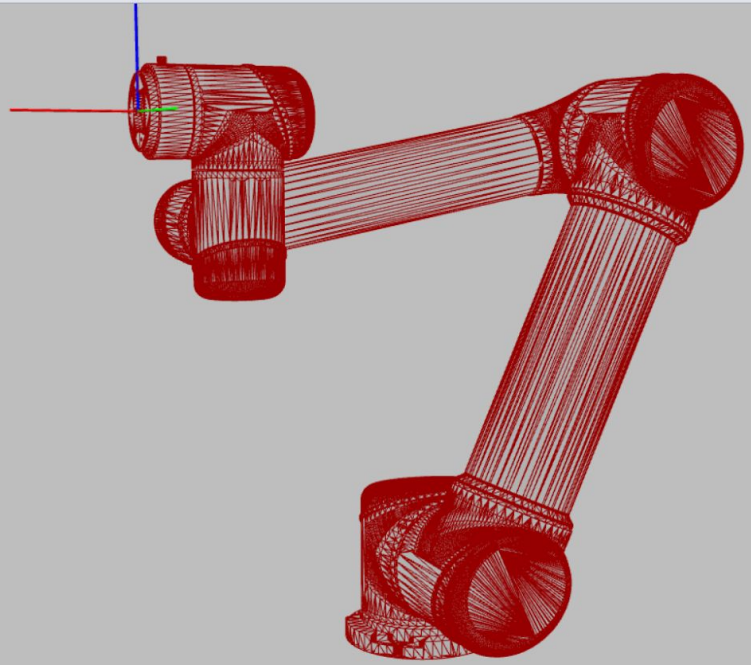
Untitled - Rhinoceros 5.0 Educational - [Perspective]

File Edit View Curve Surface Solid Mesh Dimension Transform Tools Analyze Render Panels T-Splines Help

```
Command: _Export
Meshing... Press Esc to cancel
File successfully written as C:\Users\rustr\workspace\robot_description\ur_description\meshes\ur5\visual\upperarm.obj
1 mesh added to selection.
7 meshes added to selection.
Command: _Delete
Autosaving file as C:\Users\rustr\AppData\Roaming\McNeel\Rhinoceros\5.0\AutoSave\RhinoAutosave.3dm
Autosave completed successfully
Grasshopper User Object removed: C:\Users\rustr\AppData\Roaming\Grasshopper\UserObjects\Count through list.ghuser
Command:
```

Standard | CPlanes | Set View | Display | Select | Viewport Layout | Visibility | Transform | Curve Tools | Surface Tools | Solid Tools | Mesh Tools | Render Tools | Drafting | New in V5

Perspective



Properties | Layers | Display

Viewport	
Title	Perspective
Width	1669
Height	730
Projection	Perspective
Camera	
Lens Length	50.0
Rotation	0.0
X Location	625.063
Y Location	1259.182
Z Location	471.598
Location	Place
Target	
X Target	119.418
Y Target	199.110
Z Target	304.440
Location	Place
Wallpaper	
Filename	(none)
Show	☑
Gray	☑

Perspective | Top | Front | Right

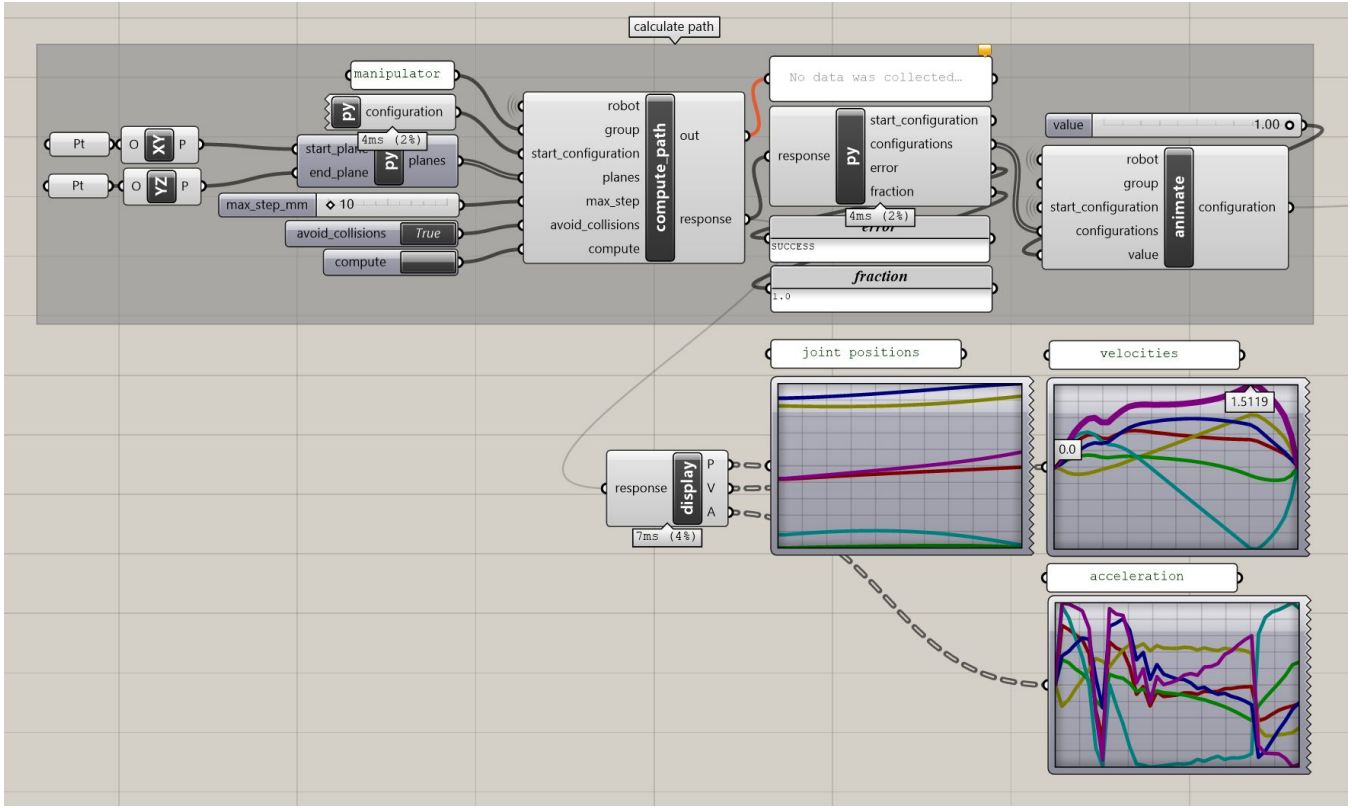
☑ End ☑ Near ☑ Point ☑ Mid ☑ Cen ☑ Int ☑ Perp ☑ Tan ☑ Quad ☑ Knot ☑ Vertex ■ Project ☐ Disable

CPlane | x -201.23 | y 196.66 | z 0.00 | Meters | Default | Grid Snap | Ortho | Planar | Osnap | SmartTrack | Gumball | Record History | Filter | Memory use: 696 MB

Cartesian path

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

robot.client = RosClient('127.0.0.1', 9090)
robot.client.run()
frames = []
frames.append(Frame([0.3, 0.1, 0.5], [1, 0, 0], [0, 1, 0]))
frames.append(Frame([0.4, 0.3, 0.4], [0, 1, 0], [0, 0, 1]))
start_configuration = Configuration.from_revolute_values([-0.042, 4.295, -4.110, -3.327, 4.755, 0.])
group = "manipulator" # or robot.main_group_name

response = robot.compute_cartesian_path(frames,
                                        start_configuration,
                                        max_step=0.01,
                                        avoid_collisions=True,
                                        group=group,
                                        path_constraints=None)

print("Computed cartesian path with %d configurations, " % len(response.configurations))
print("following %d%% of requested trajectory." % (response.fraction * 100))
print("Executing this path at full speed would take approx. %.3f seconds." %
response.solution.joint_trajectory.points[-1].time_from_start.seconds())
print(response.configurations)
```

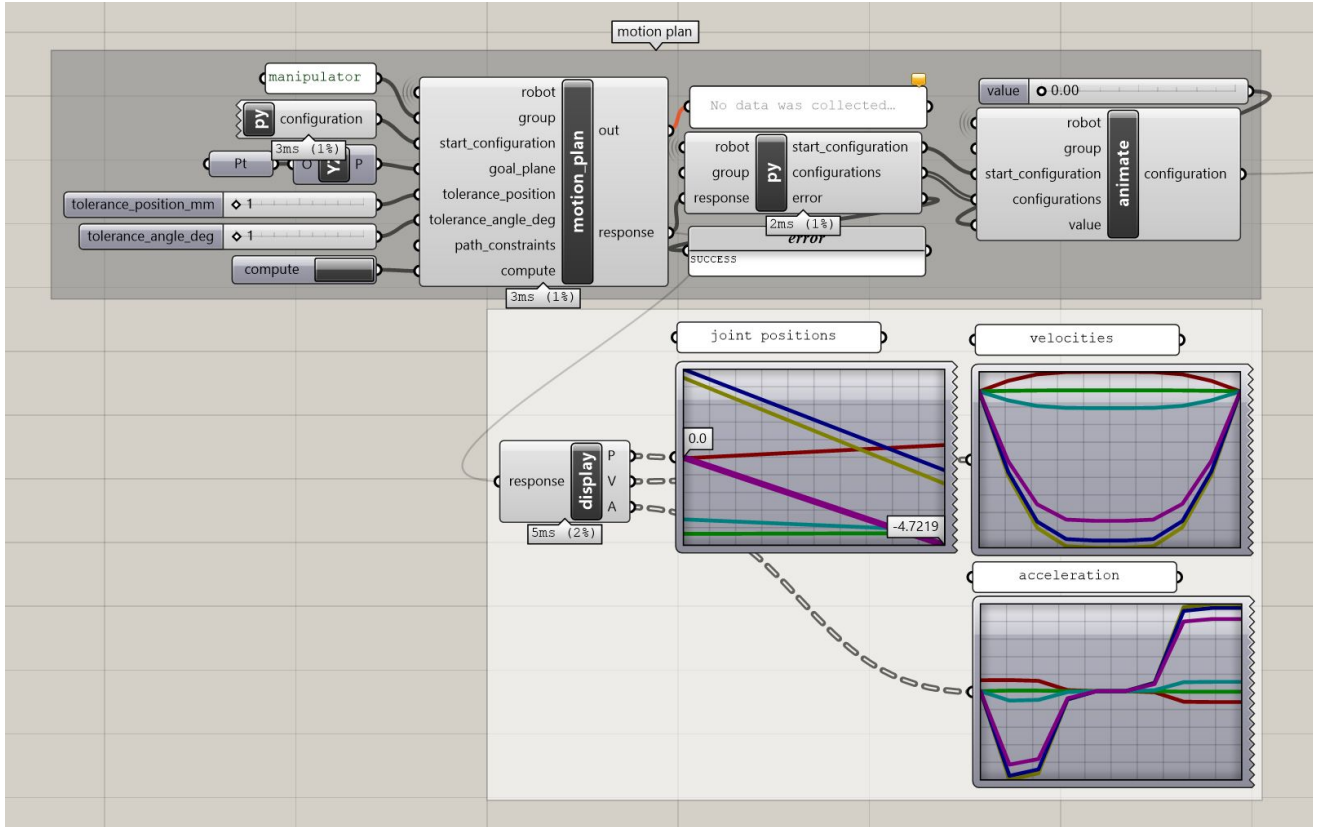
Cartesian path

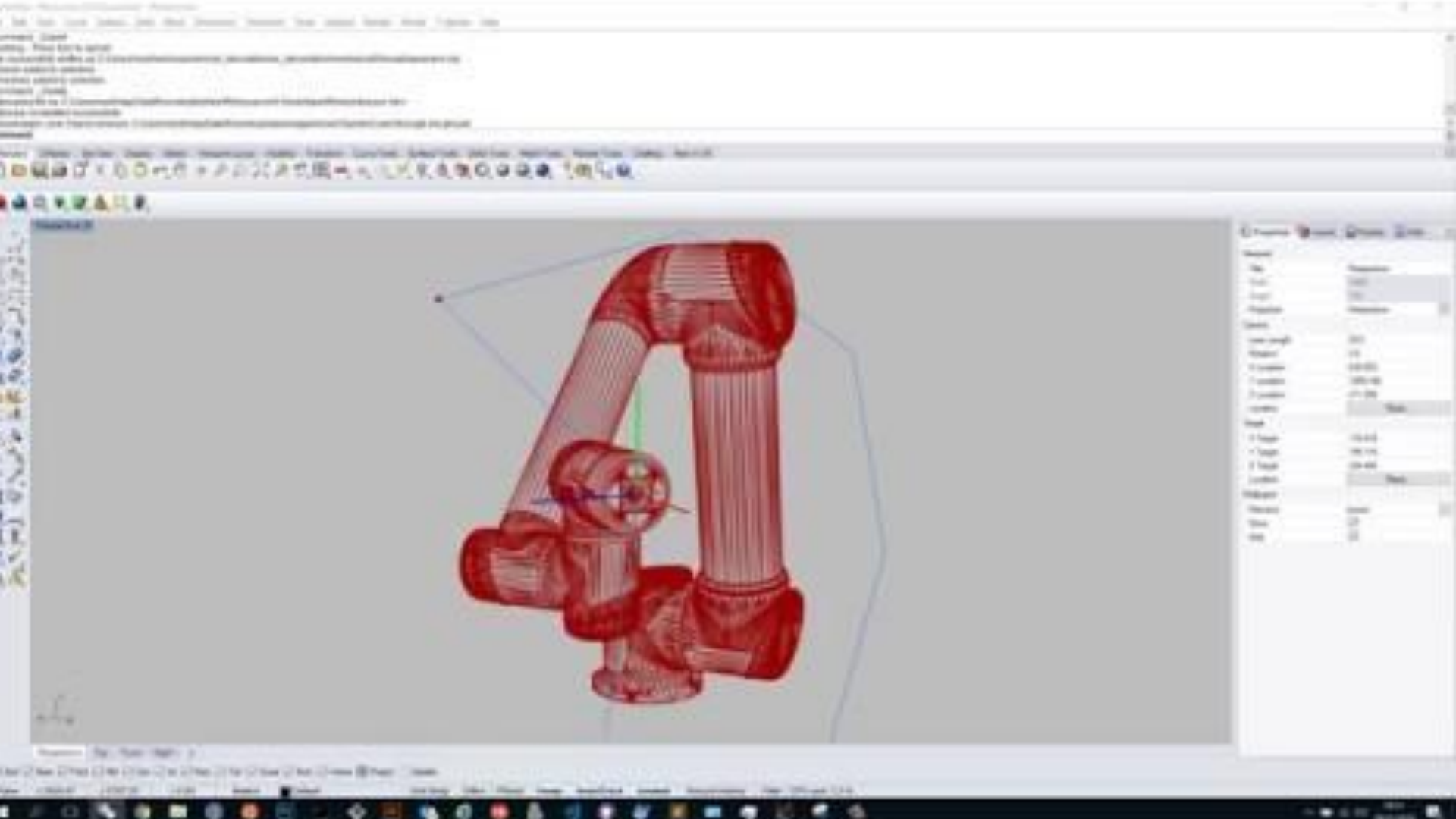


Kinematic path

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

robot.client = RosClient('127.0.0.1', 9090)
robot.client.run()
goal_frame = Frame([0.4, 0.3, 0.4], [0, 1, 0], [0, 0, 1])
start_configuration = Configuration.from_revolute_values([-0.042, 4.295, -4.110, -3.327, 4.755, 0.])
group = "manipulator"  # or robot.main_group_name

response = robot.motion_plan_goal_frame(goal_frame,
                                        start_configuration,
                                        tolerance_position=0.001,
                                        tolerance_angle=math.radians(1),
                                        group=group,
                                        path_constraints=None,
                                        planner_id='RRT',
                                        num_planning_attempts=20,
                                        allowed_planning_time=8.)

print("Computed kinematic path with %d configurations." % len(response.configurations))
print("Executing this path at full speed would take approx. %.3f seconds." %
response.trajectory.joint_trajectory.points[-1].time_from_start.seconds())
```

Kinematic path





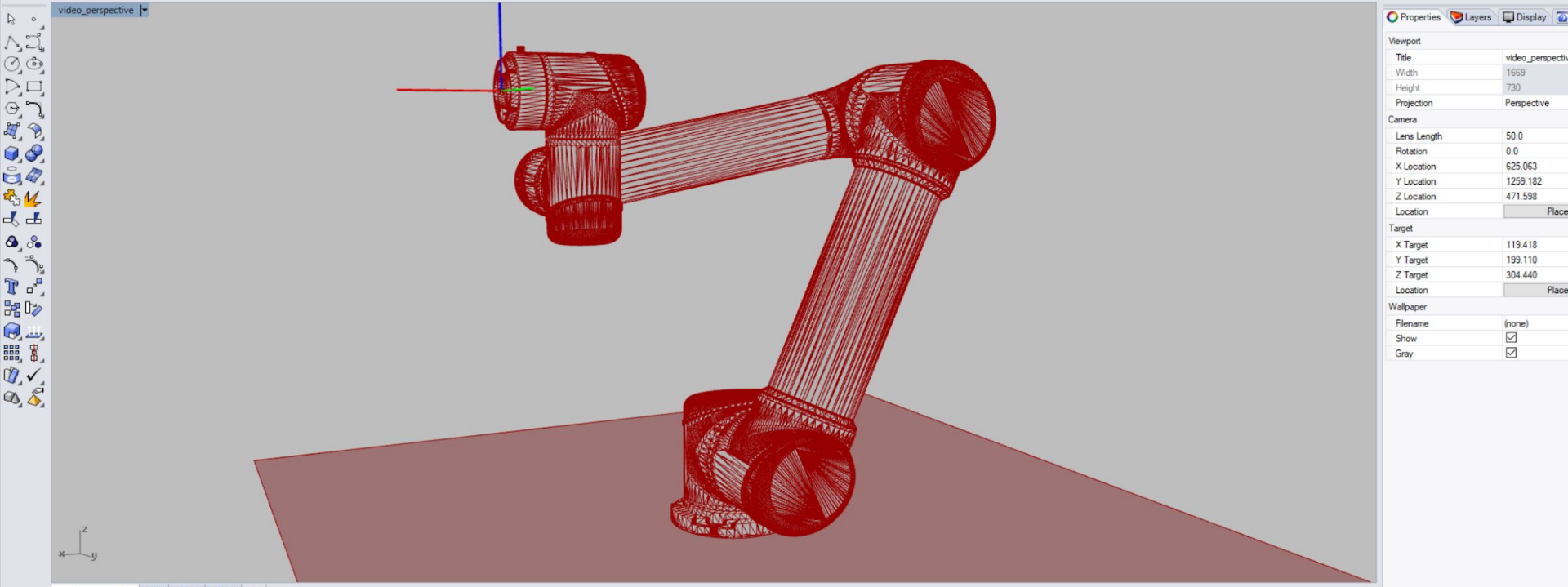
Collision object

Untitled - Rhinoceros 5.0 Educational - [video_perspective]

File Edit View Curve Surface Solid Mesh Dimension Transform Tools Analyze Render Panels T-Splines Help

```
1 curve, 1 surface added to selection.
Command: Move
Point to move from ( Vertical=No ):
Point to move to: 0,0,0
1 curve added to selection.
Command: _Delete
1 surface added to selection.
1 surface added to selection.
Command: _Delete
Command:
```

Standard | CPlanes | Set View | Display | Select | Viewport Layout | Visibility | Transform | Curve Tools | Surface Tools | Solid Tools | Mesh Tools | Render Tools | Drafting | New in V5

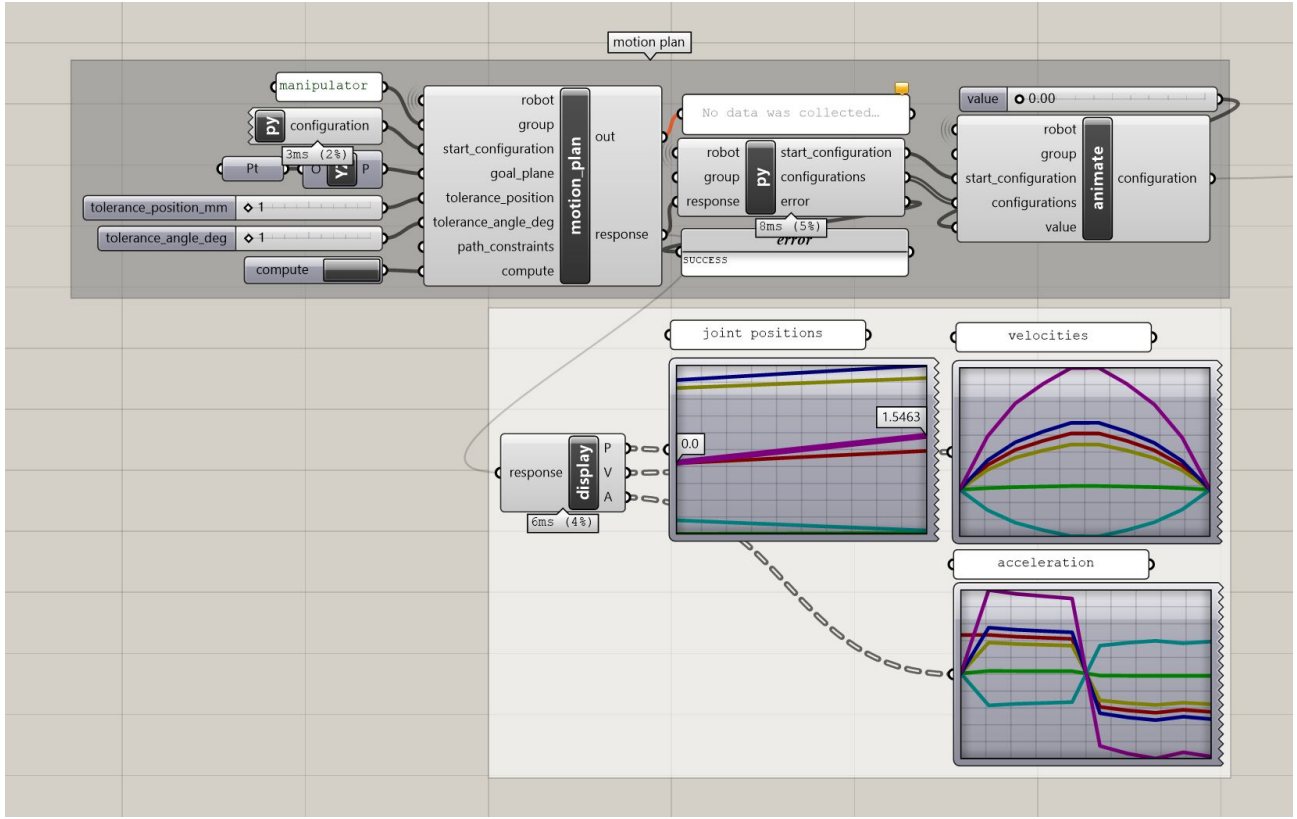


End Near Point Mid Cen Int Perp Tan Quad Knot Vertex Project Disable

CPlane x 726.70 y -434.59 z 0.00 Meters Default Grid Snap Ortho Planar Osnap SmartTrack Gumball Record History Filter Absolute tolerance: 0.01

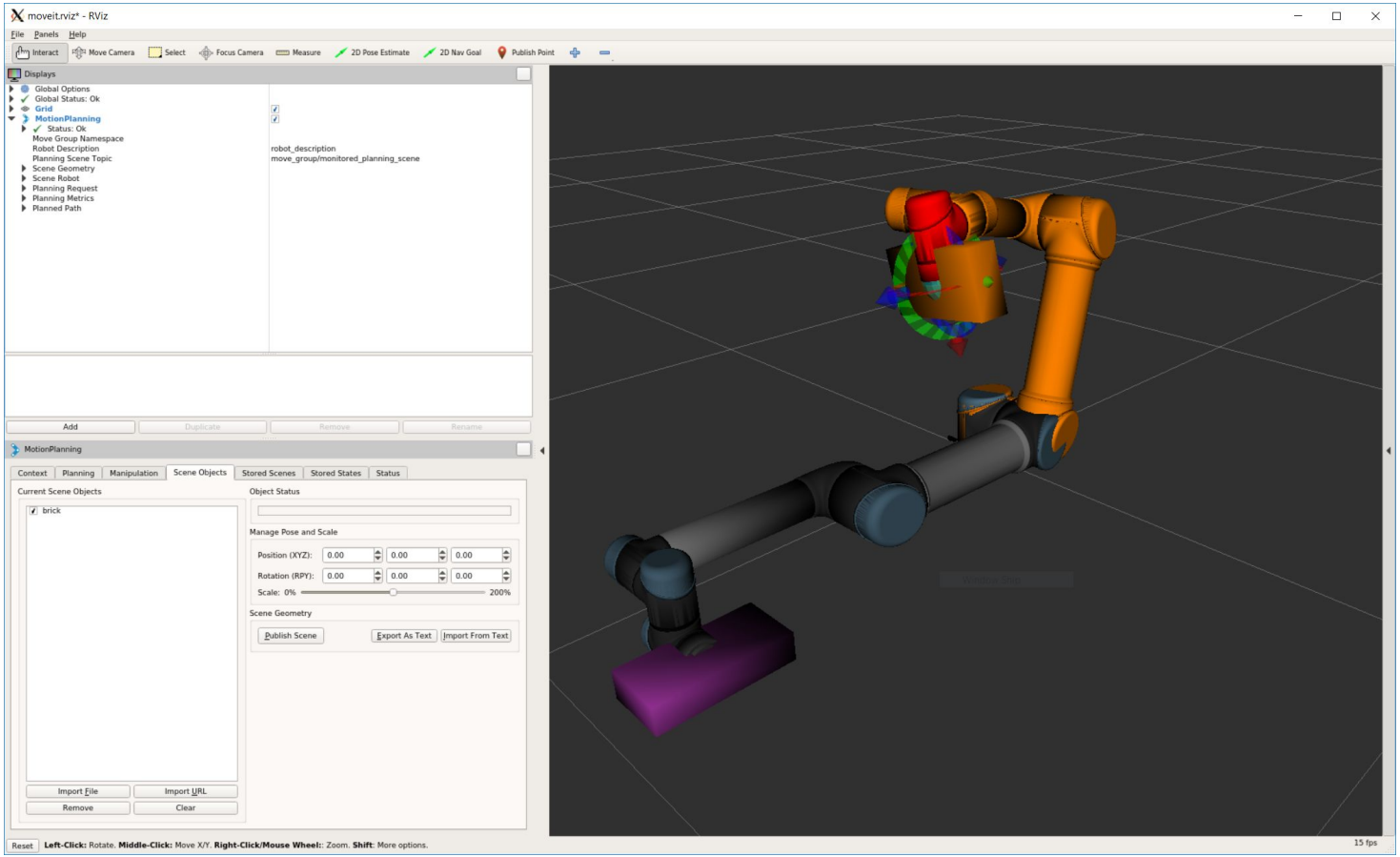
```python
from compas.datastructures import Mesh
from compas_fab.backends import RosClient
from compas_rhino.helpers import mesh_from_guid

client = RosClient('127.0.0.1', 9090)
robot.client = client
robot.client.run()

compas_mesh = mesh_from_guid(Mesh, mesh)
robot.add_collision_mesh_to_planning_scene("floor", compas_mesh)

robot.client.terminate()
```

Kinematic path



Attached collision object

```python
import os
import time

from compas.datastructures import Mesh
from compas_fab.backends import RosClient

robot.client = RosClient('127.0.0.1', 9090)
robot.client.run()

group = robot.main_group_name
brick = Mesh.from_obj(os.path.join(os.path.dirname(__file__), "brick.obj"))

robot.add_attached_collision_mesh('brick', brick, group)
time.sleep(2)
```


moveit.rviz* - RViz
File
Panels
Help
Interact
Move Camera
Select
Focus Camera
Measure
2D Pose Estimate
2D Nav Goal
Publish Point
Displays
Global Options
Global Status: Ok
Grid
MotionPlanning
Status: Ok
Move Group Namespace
Robot Description
robot_description
Planning Scene Topic
move_group/monitored_planning_scene
Scene Geometry
Scene Robot
Planning Request
Planning Metrics
Planned Path
Add
Duplicate
Remove
Rename
MotionPlanning
Context
Planning
Manipulation
Scene Objects
Stored Scenes
Stored States
Status
Current Scene Objects
brick
Object Status
Manage Pose and Scale
Position (XYZ):
Rotation (RPY):
Scale: 0%
200%
Scene Geometry
Publish Scene
Export As Text
Import From Text
Import File
Import URL
Remove
Clear
Reset
Left-Click: Rotate. Middle-Click: Move X/Y. Right-Click/Mouse Wheel:: Zoom. Shift: More options.
15 fps

```python
import os, math
from compas.geometry import Frame
from compas.datastructures import Mesh
from compas_fab.robots import Configuration
from compas_fab.backends import RosClient

robot.client = RosClient('127.0.0.1', 9090)
robot.client.run()
goal_frame = Frame([0.20, 0.38, 0.32], [0, 1, 0], [0, 0, 1])
start_configuration = Configuration.from_revolute_values([-0.042, 4.295, -4.110, -3.327, 4.755, 0.])
group = robot.main_group_name
# Create attached collision object
brick = Mesh.from_obj(os.path.join(os.path.dirname(__file__), "brick.obj"))
aco = robot.create_collision_mesh_attached_to_end_effector('brick', brick, group)

response = robot.motion_plan_goal_frame(goal_frame,
                                        start_configuration,
                                        tolerance_position=0.001,
                                        tolerance_angle=math.radians(1),
                                        group=group,
                                        path_constraints=None,
                                        planner_id='RRT',
                                        num_planning_attempts=20,
                                        allowed_planning_time=8.,
                                        attached_collision_object=aco)
```

```python
import os, math
from compas.geometry import Frame
from compas.datastructures import Mesh
from compas_fab.robots import Configuration
from compas_fab.backends import RosClient

robot.client = RosClient('127.0.0.1', 9090)
robot.client.run()
goal_frame = Frame([0.20, 0.38, 0.32], [0, 1, 0], [0, 0, 1])
start_configuration = Configuration.from_revolute_values([-0.042, 4.295, -4.110, -3.327, 4.755, 0.])
group = robot.main_group_name
# Create attached collision object
brick = Mesh.from_obj(os.path.join(os.path.dirname(__file__), "brick.obj"))
aco = robot.create_collision_mesh_attached_to_end_effector('brick', brick, group)

response = robot.motion_plan_goal_frame(goal_frame,
                                        start_configuration,
                                        tolerance_position=0.001,
                                        tolerance_angle=math.radians(1),
                                        group=group,
                                        path_constraints=None,
                                        planner_id='RRT',
                                        num_planning_attempts=20,
                                        allowed_planning_time=8.,
                                        attached_collision_object=aco)
```

Trajectory execution

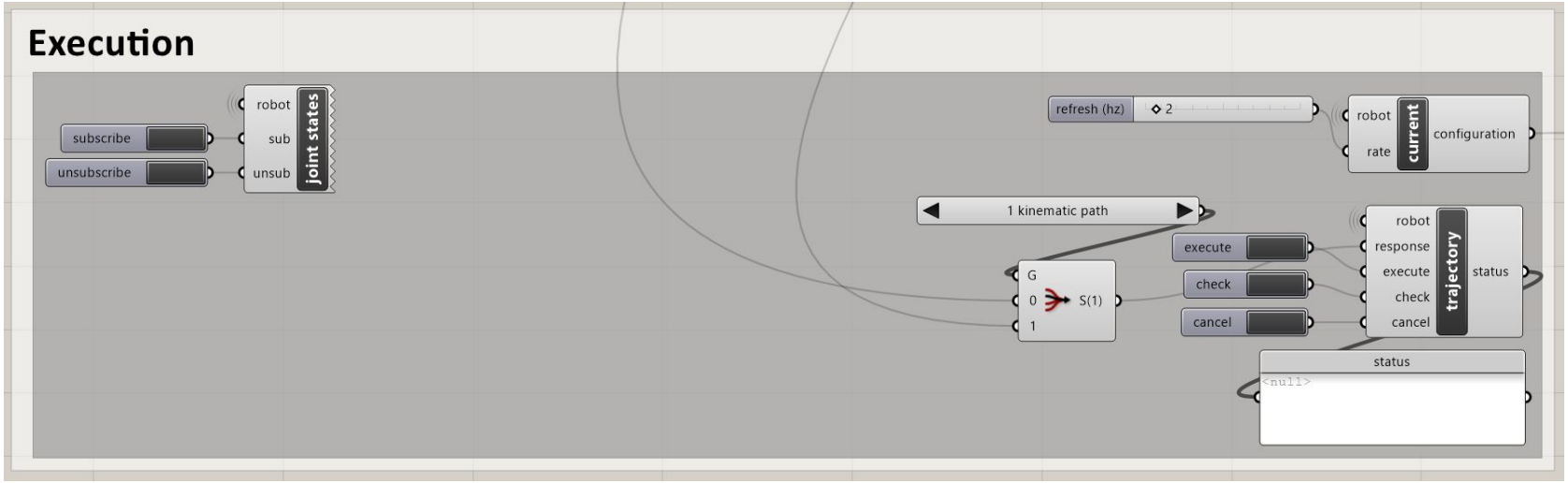
The fire-and-forget version

```
joint_trajectory = response.trajectory.joint_trajectory

robot.client.follow_joint_trajectory(joint_trajectory)
```

The respectable citizen's version

```
joint_trajectory = response.trajectory.joint_trajectory

def completion(result):
   print('Done')

def failure(e):
   print('ERROR', e)

token = robot.client.follow_joint_trajectory(joint_trajectory,
                                             completion,
                                             failure)
```

Execution


subscribe
unsubscribe
robot
sub
unsub
joint states
refresh (hz)
2
robot
rate
current
configuration
1 kinematic path
G
0
1
S(1)
execute
check
cancel
robot
response
execute
check
cancel
trajectory
status
status
<null>

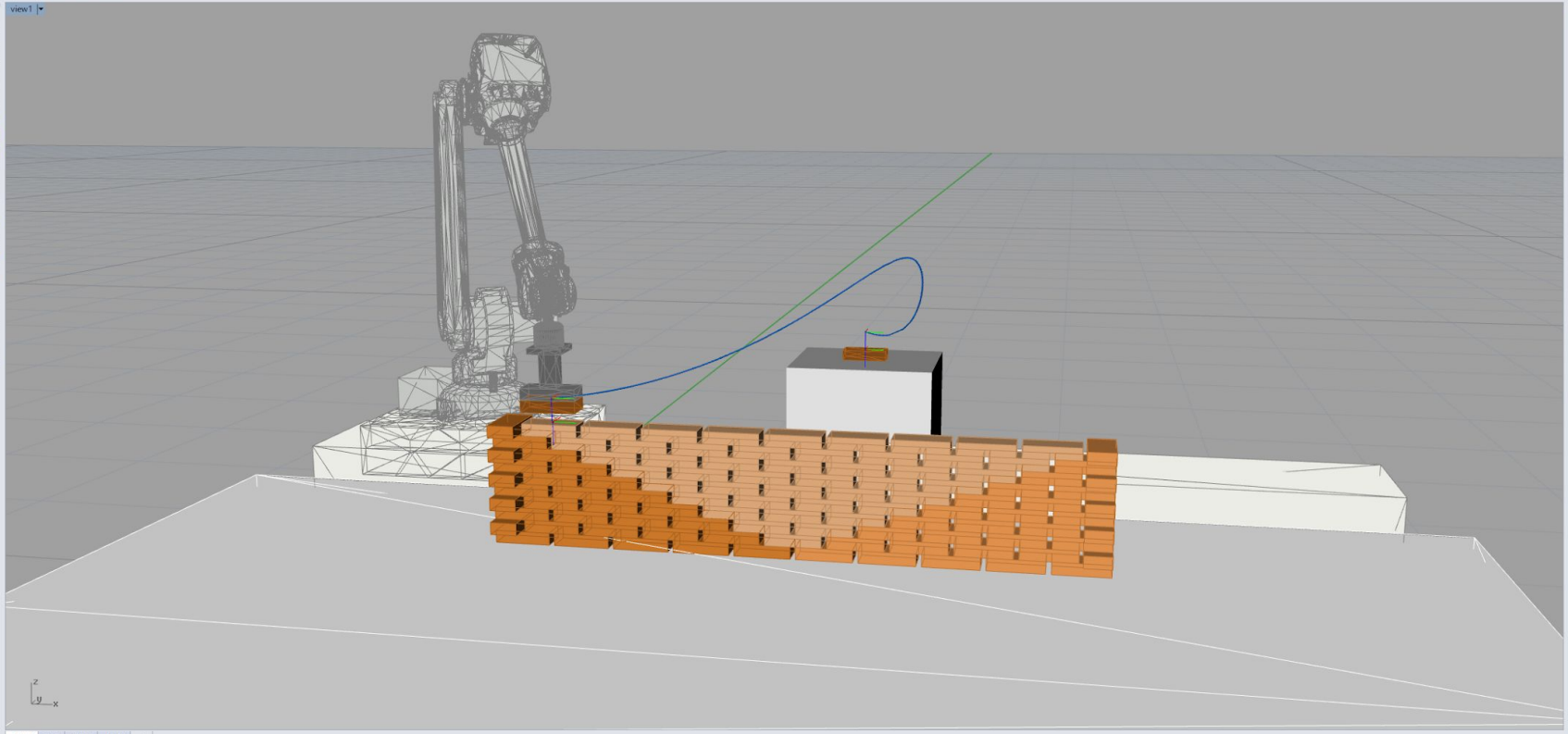
Plan paths for brick assembly

Untitled - Rhinoceros 5.0 Educational - [view1]

File Edit View Curve Surface Solid Mesh Dimension Transform Tools Analyze Render Panels T-Splines Help

```
45 polysurfaces added to selection.
Command: _Delete
Command: _Delete
Command: _Undo
Undoing Delete
25 polysurfaces, 5 extrusions added to selection.
Command: _Delete
Command: NamedView
Command:
Command:
```

Standard | CPlanes | Set View | Display | Select | Viewport Layout | Visibility | Transform | Curve Tools | Surface Tools | Solid Tools | Mesh Tools | Render Tools | Drafting | New in V5



Properties | Layers | Display

Viewport	
Title	view1
Width	2181
Height	1018
Projection	Perspective
Camera	
Lens Length	50.0
Rotation	0.0
X Location	3.050
Y Location	-6.824
Z Location	2.460
Location	Place
Target	
X Target	1.786
Y Target	-0.197
Z Target	1.209
Location	Place
Wallpaper	
Filename	(none)
Show	☑
Gray	☑

view1 | Top | Front | Right

☑ End ☑ Near ☑ Point ☑ Mid ☑ Cen ☑ Int ☑ Perp ☑ Tan ☑ Quad ☑ Knot ☑ Vertex ■ Project ☐ Disable

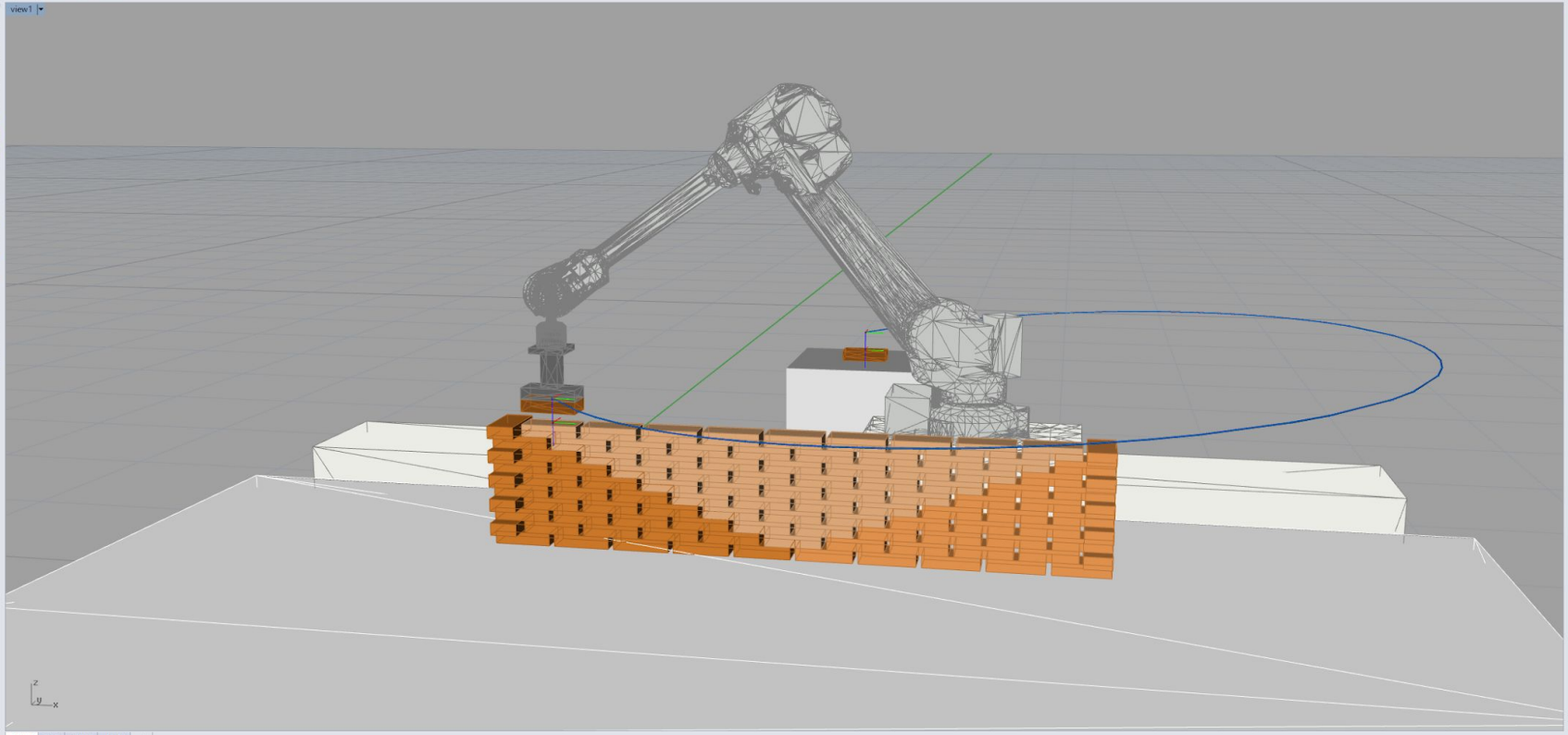
CPlane | x -22.76 | y 183.40 | z 0.00 | Meters | Default | Grid Snap | Ortho | Planar | Osnap | SmartTrack | Gumball | Record History | Filter | CPU use: 1.6 %

Untitled - Rhinoceros 5.0 Educational - [view1]

File Edit View Curve Surface Solid Mesh Dimension Transform Tools Analyze Render Panels T-Splines Help

```
Command: _Delete
Command: _Undo
Undoing Delete
25 polysurfaces, 5 extrusions added to selection.
Command: _Delete
Command: NamedView
Command:
Command: NamedView
Command:
Command:
```

Standard | CPlanes | Set View | Display | Select | Viewport Layout | Visibility | Transform | Curve Tools | Surface Tools | Solid Tools | Mesh Tools | Render Tools | Drafting | New in V5



Properties | Layers | Display

Viewport	
Title	view1
Width	2181
Height	1018
Projection	Perspective
Camera	
Lens Length	50.0
Rotation	0.0
X Location	3.050
Y Location	-6.824
Z Location	2.460
Location	Place
Target	
X Target	1.786
Y Target	-0.197
Z Target	1.209
Location	Place
Wallpaper	
Filename	(none)
Show	☑
Gray	☑

view1 | Top | Front | Right

☑ End ☑ Near ☑ Point ☑ Mid ☑ Cen ☑ Int ☑ Perp ☑ Tan ☑ Quad ☑ Knot ☑ Vertex ☐ Project ☐ Disable

CPlane | x -2.60 | y 22.31 | z 0.00 | Meters | Default | Grid Snap | Ortho | Planar | Osnap | SmartTrack | Gumball | Record History | Filter | Memory use: 1028 MB

ROS from CAD environments

GitHub, Inc. [US] | https://github.com/gramaziokohler/roslibpy/

README.rst

roslibpy: ROS Bridge library

docs passing | license MIT license | pypi v0.3.0 | wheel yes | python 2.7 | 3.3 | 3.4 | 3.5 | 3.6 | implementation cpython | ironpython

Python ROS Bridge library allows to use Python and IronPython to interact with ROS, the open-source robotic middleware. It uses WebSockets to connect to rosbridge 2.0 and provides publishing, subscribing, service calls, actionlib, TF, and other essential ROS functionality.

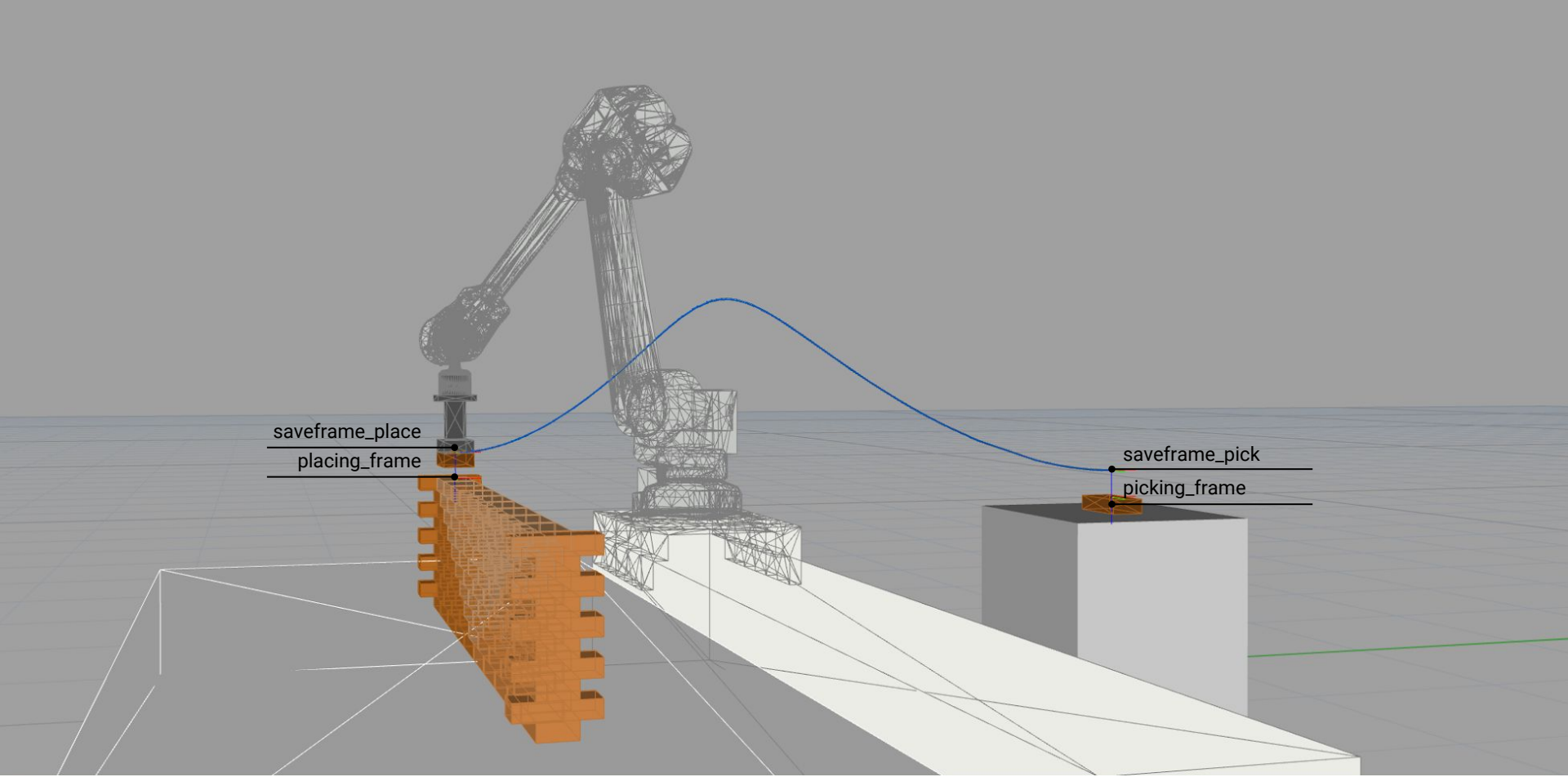
Unlike the rospy library, this does not require a local ROS environment, allowing usage from platforms other than Linux.

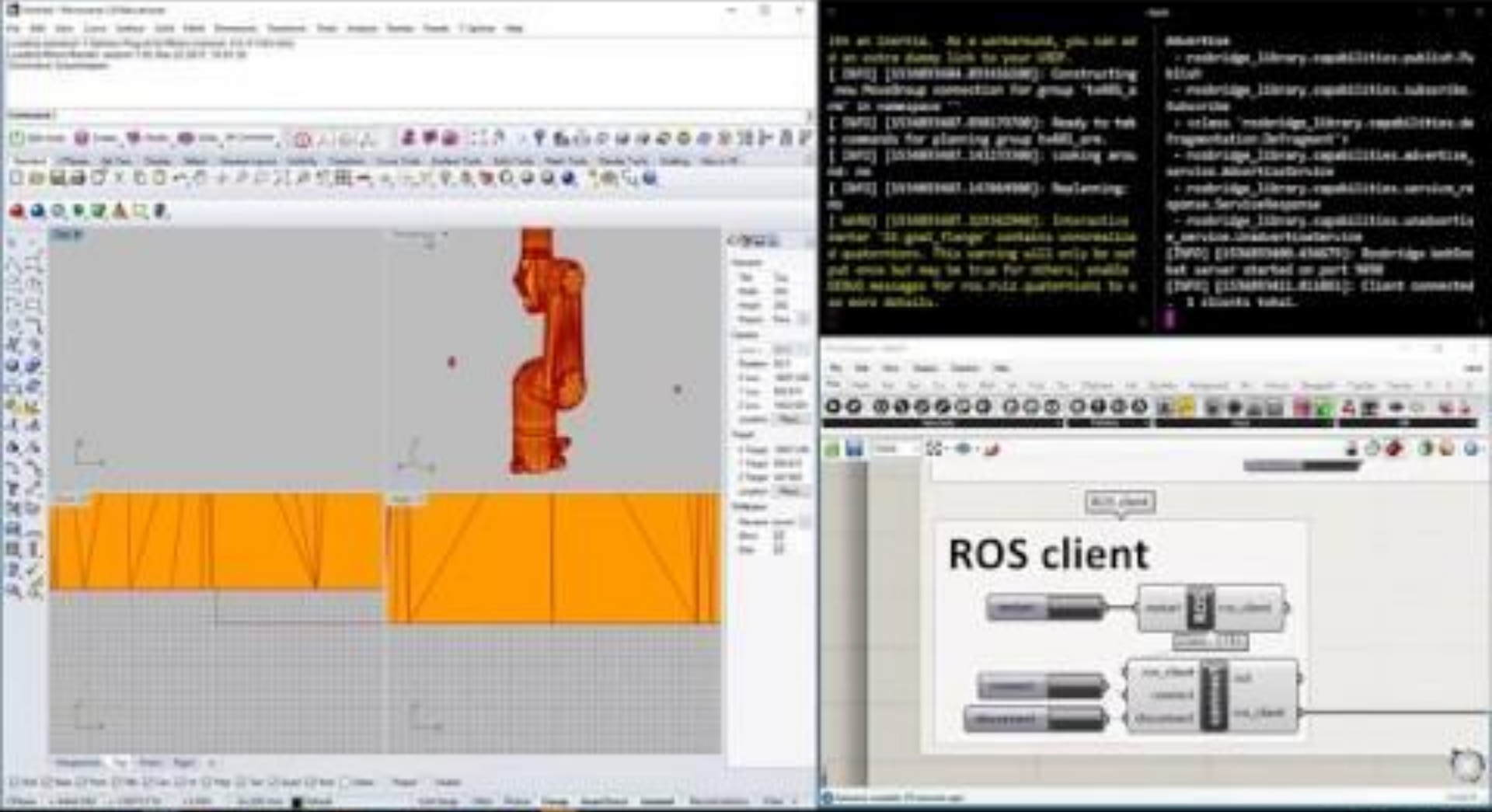
The API of **roslibpy** is modeled to closely match that of roslibjs.

Main features

- Topic publishing and subscribing.
- Service calls (client).
- Service advertisement (server).
- ROS parameter management (get/set/delete).
- ROS API services for getting ROS meta-information.
- Actionlib support for interfacing with preemptable tasks.
- TF Client via the `tf2_web_republisher`.

- Based on **roslibpy** and **rosbridge 2.0**
- Supports CPython and IronPython
- Provides essential ROS functionality
- Topics, Service, Action, TF, etc.


saveframe_place
placing_frame
saveframe_pick
picking_frame


ROS client

National Centre of Competence
in Research
Digital Fabrication