GRAMAZIO
KOHLER
RESEARCH

ETH zürich

Introduction to compas_fab

Robotic fabrication package for the COMPAS framework

Pre-workshop Rob|Arch 2018

Gonzalo Casas, Augusto Gandia

ETH Zürich, September 9th, 2018

Today’s agenda

COMPAS & COMPAS FAB

Introduction to the framework and the robotic fabrication package

PATH PLANNING

Robot kinematics and path planning in architecture

ROS + COMPAS

Simulating and controlling robots

HANDS-ON

Time to hack!


COMPAS

COMPAS ecosystem

frontends

- Blender 
- Grasshopper 
- Rhino 

COMPAS Framework

backends

- Abaqus
- ROS
- V-REP
- Numpy
- . . .

Main compas library

main library

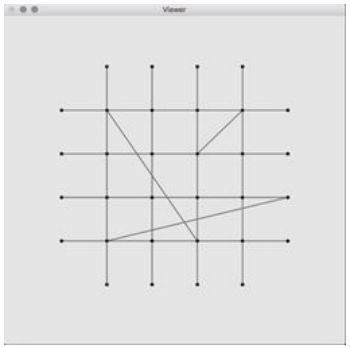
```
compas
```

- com
- datastructures
- files
- geometry
- interop
- numerical
- robots
- topology
- utilities
- viewers

Core functionality independent of CAD

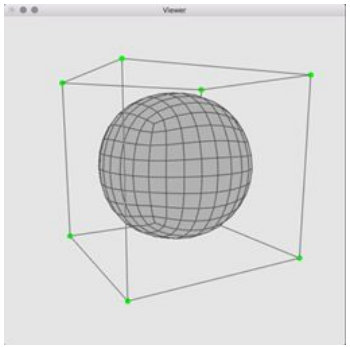
- Python
- Flexible data structures
- Algorithms for AEFC
- Geometry processing
- Easy interop with C/C++
- Numerical computation
- GPU acceleration & JIT compilation
- Plotters & Viewers

Data structures



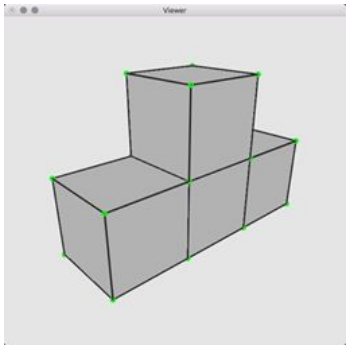
Network

- General networks
- Directed edge graph



Mesh

- Surface meshes
- Half-edge



Volumetric Mesh

- Cellular meshes
- Half-plane

Algorithms

Geometry

- Mesh smoothing
- Face planarization
- Convex hull

Numerical

- Dynamic relaxation
- Force density
- Topology optimization

Topology

- Traversal
- Subdivision
- Triangulation

CAD Integration

main library

`compas`

- `com`
- `datastructures`
- `files`
- `geometry`
- `interop`
- `numerical`
- `robots`
- `topology`
- `utilities`
- `viewers`

`compas_blender`

`compas_ghpython`

`compas_rhino`

CAD interfaces

- Unified integration of data structures
- Unified scripting interface
- Unified UI system

```python
# Rhino
from compas.datastructures import Mesh
from compas_rhino import xdraw_mesh
mesh = Mesh.from_obj('https://u.nu/hypar')

vertices, faces = mesh.to_vertices_and_faces()
xdraw_mesh(vertices, faces)
```

```python
# Blender
from compas.datastructures import Mesh
from compas_blender import xdraw_mesh
mesh = Mesh.from_obj('https://u.nu/hypar')

vertices, faces = mesh.to_vertices_and_faces()
xdraw_mesh(vertices, faces)
```

```python
# Rhino
from compas.datastructures import Mesh
from compas_rhino import xdraw_mesh
mesh = Mesh.from_obj('https://u.nu/hypar')

vertices, faces = mesh.to_vertices_and_faces()
xdraw_mesh(vertices, faces)
```

```python
# Blender
from compas.datastructures import Mesh
from compas_blender import xdraw_mesh
mesh = Mesh.from_obj('https://u.nu/hypar')

vertices, faces = mesh.to_vertices_and_faces()
xdraw_mesh(vertices, faces)
```

Additional packages

main library

`compas`

- `com`
- `datastructures`
- `files`
- `geometry`
- `interop`
- `numerical`
- `robots`
- `topology`
- `utilities`
- `viewers`

packages

`compas_ags`

Algebraic Graphic Statics

`compas_3gs`

3D Graphic Statics

`compas_fab`

Robotic Fabrication

`compas_fea`

Finite Element Analysis

`compas_rbe`

Rigid Block Equilibrium

`compas_bend`

Bending-active Structures

Additional packages

main library

`compas`

- `com`
- `datastructures`
- `files`
- `geometry`
- `interop`
- `numerical`
- `robots`
- `topology`
- `utilities`
- `viewers`

packages

`compas_ags`

Algebraic Graphic Statics

`compas_3gs`

3D Graphic Statics

`compas_fab`

Robotic Fabrication

`compas_fea`

Finite Element Analysis

`compas_rbe`

Rigid Block Equilibrium

`compas_bend`

Bending-active Structures


COMPAS FAB

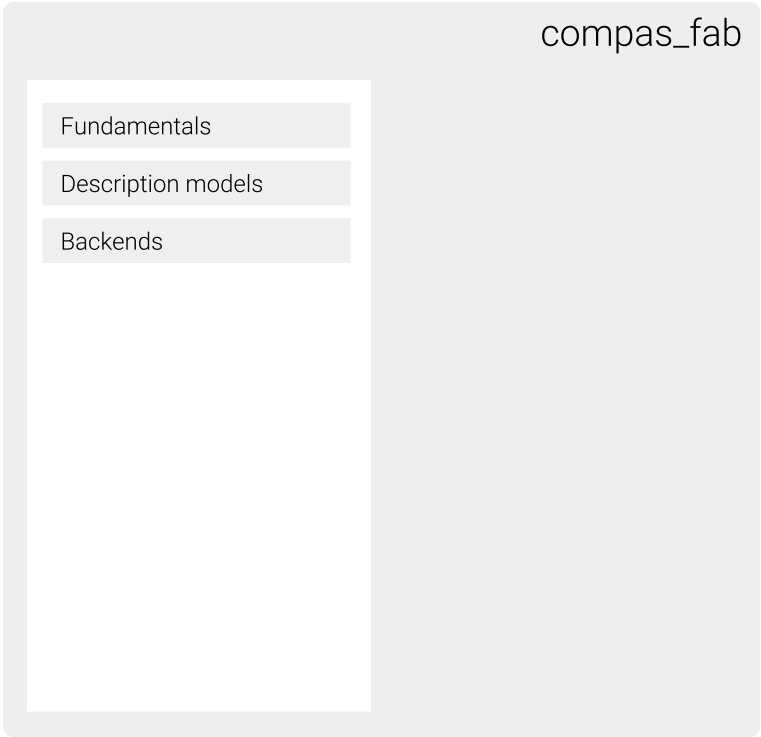
Overview of compas_fab

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

Built upon COMPAS framework.

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

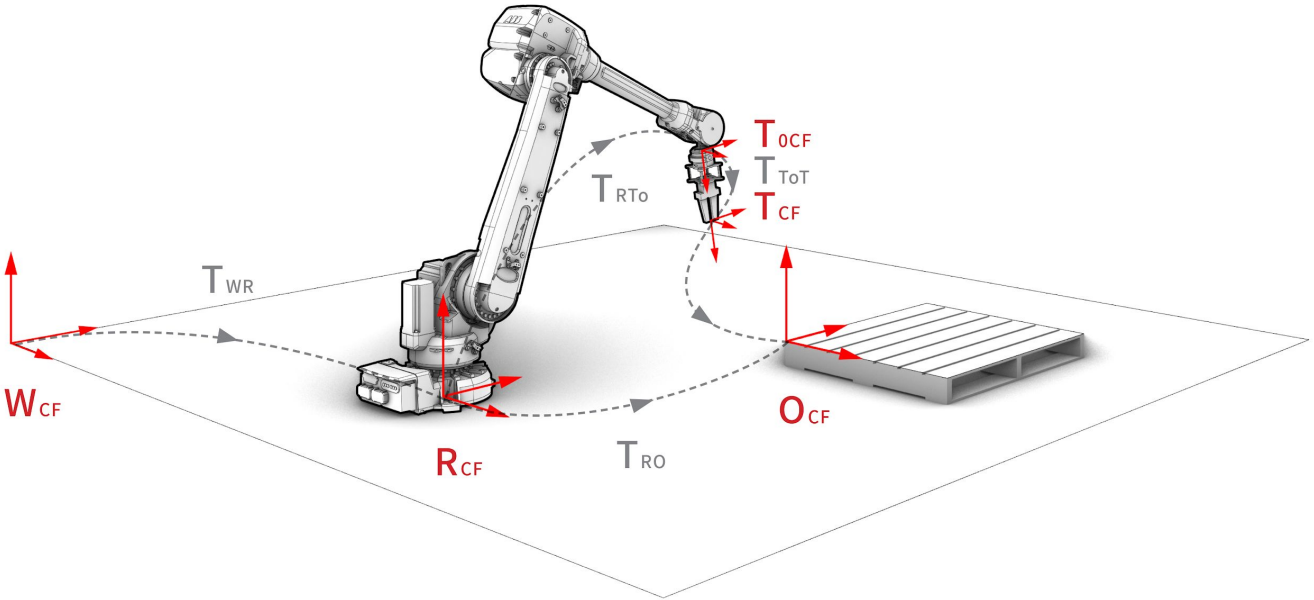
Robotic fabrication



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

Robotic fabrication: fundamentals

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



```python
# Frame example
from compas.geometry import Frame
from compas_ghpython import xdraw_frame

point =  (0.0, 0.0, 63.0)
xaxis =  (1.0, 0.0, 0.0)
yaxis =  (0.0, 1.0, 0.0)

F1 = Frame(point, xaxis, yaxis)
a = [xdraw_frame(F1)]

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

F2 = Frame(point, xaxis, yaxis)
a.append(xdraw_frame(F2))
```

```python
# Transformation example
from compas.geometry.xforms import Transformation
from compas.geometry import transform_points
from compas_ghpython import xdraw_points

P1 = (35., 35., 35.)
a += xdraw_points([{'pos': P1}])

world = Frame.worldXY()
Tw = Transformation.from_frame_to_frame(world, F1)
Pw = transform_points([P1], Tw)
a += xdraw_points([{'pos': Pw[0]}])

T = Transformation.from_frame_to_frame(F1, F2)
P2 = transform_points(Pw, T)
a += xdraw_points([{'pos': P2[0]}])
```

```python
# Frame example
from compas.geometry import Frame
from compas_ghpython import xdraw_frame

point =  (0.0, 0.0, 63.0)
xaxis =  (1.0, 0.0, 0.0)
yaxis =  (0.0, 1.0, 0.0)

F1 = Frame(point, xaxis, yaxis)
a = [xdraw_frame(F1)]

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

F2 = Frame(point, xaxis, yaxis)
a.append(xdraw_frame(F2))
```

```python
# Transformation example
from compas.geometry.xforms import Transformation
from compas.geometry import transform_points
from compas_ghpython import xdraw_points

P1 = (35., 35., 35.)
a += xdraw_points([{'pos': P1}])

world = Frame.worldXY()
Tw = Transformation.from_frame_to_frame(world, F1)
Pw = transform_points([P1], Tw)
a += xdraw_points([{'pos': Pw[0]}])

T = Transformation.from_frame_to_frame(F1, F2)
P2 = transform_points(Pw, T)
a += xdraw_points([{'pos': P2[0]}])
```

```python
# Frame example
from compas.geometry import Frame
from compas_ghpython import xdraw_frame

point =  (0.0, 0.0, 63.0)
xaxis =  (1.0, 0.0, 0.0)
yaxis =  (0.0, 1.0, 0.0)

F1 = Frame(point, xaxis, yaxis)
a = [xdraw_frame(F1)]

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

F2 = Frame(point, xaxis, yaxis)
a.append(xdraw_frame(F2))
```

```python
# Transformation example
from compas.geometry.xforms import Transformation
from compas.geometry import transform_points
from compas_ghpython import xdraw_points

P1 = (35., 35., 35.)
a += xdraw_points([{'pos': P1}])

world = Frame.worldXY()
Tw = Transformation.from_frame_to_frame(world, F1)
Pw = transform_points([P1], Tw)
a += xdraw_points([{'pos': Pw[0]}])

T = Transformation.from_frame_to_frame(F1, F2)
P2 = transform_points(Pw, T)
a += xdraw_points([{'pos': P2[0]}])
```

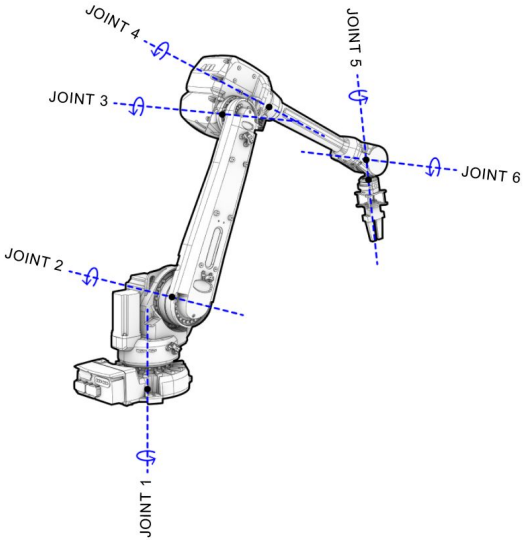
```python
# Frame example
from compas.geometry import Frame
from compas_ghpython import xdraw_frame

point =  (0.0, 0.0, 63.0)
xaxis =  (1.0, 0.0, 0.0)
yaxis =  (0.0, 1.0, 0.0)

F1 = Frame(point, xaxis, yaxis)
a = [xdraw_frame(F1)]

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

F2 = Frame(point, xaxis, yaxis)
a.append(xdraw_frame(F2))
```

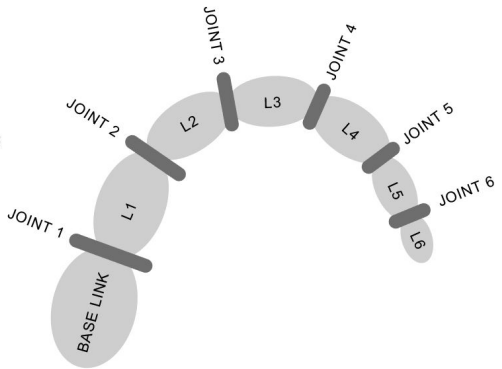
```python
# Transformation example
from compas.geometry.xforms import Transformation
from compas.geometry import transform_points
from compas_ghpython import xdraw_points

P1 = (35., 35., 35.)
a += xdraw_points([{'pos': P1}])

world = Frame.worldXY()
Tw = Transformation.from_frame_to_frame(world, F1)
Pw = transform_points([P1], Tw)
a += xdraw_points([{'pos': Pw[0]}])

T = Transformation.from_frame_to_frame(F1, F2)
P2 = transform_points(Pw, T)
a += xdraw_points([{'pos': P2[0]}])
```

```python
# Frame example
from compas.geometry import Frame
from compas_ghpython import xdraw_frame

point =  (0.0, 0.0, 63.0)
xaxis =  (1.0, 0.0, 0.0)
yaxis =  (0.0, 1.0, 0.0)

F1 = Frame(point, xaxis, yaxis)
a = [xdraw_frame(F1)]

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

F2 = Frame(point, xaxis, yaxis)
a.append(xdraw_frame(F2))
```

```python
# Transformation example
from compas.geometry.xforms import Transformation
from compas.geometry import transform_points
from compas_ghpython import xdraw_points

P1 = (35., 35., 35.)
a += xdraw_points([{'pos': P1}])

world = Frame.worldXY()
Tw = Transformation.from_frame_to_frame(world, F1)
Pw = transform_points([P1], Tw)
a += xdraw_points([{'pos': Pw[0]}])

T = Transformation.from_frame_to_frame(F1, F2)
P2 = transform_points(Pw, T)
a += xdraw_points([{'pos': P2[0]}])
```

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 Robot

Robot.from_urdf_file('/urdf/ur5.urdf')
```

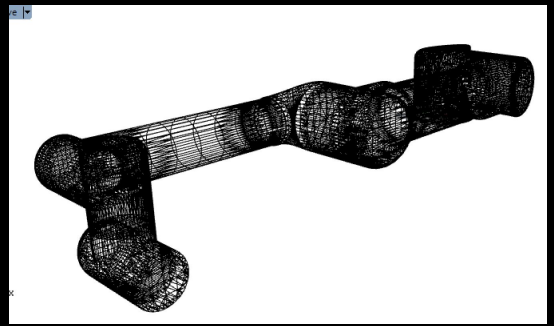
```python
# Rhino
from compas.robots import model
from compas_rhino import xdraw_mesh

r = model.Robot.from_urdf_file('/urdf/ur5.urdf')
r.load_geometry()

for mesh in r.get_visual_meshes():
    v, f = mesh.to_vertices_and_faces()
    xdraw_mesh(v, f)
```

```python
# Blender
from compas.robots import model
from compas_blender import xdraw_mesh

r = model.Robot.from_urdf_file('/urdf/ur5.urdf')
r.load_geometry()

for mesh in r.get_visual_meshes():
    v, f = mesh.to_vertices_and_faces()
    xdraw_mesh(v, f)
```

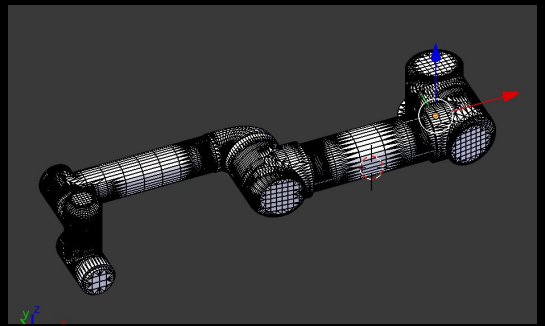
```python
# Rhino
from compas.robots import model
from compas_rhino import xdraw_mesh

r = model.Robot.from_urdf_file('/urdf/ur5.urdf')
r.load_geometry()

for mesh in r.get_visual_meshes():
    v, f = mesh.to_vertices_and_faces()
    xdraw_mesh(v, f)
```

```python
# Blender
from compas.robots import model
from compas_blender import xdraw_mesh

r = model.Robot.from_urdf_file('/urdf/ur5.urdf')
r.load_geometry()

for mesh in r.get_visual_meshes():
    v, f = mesh.to_vertices_and_faces()
    xdraw_mesh(v, f)
```

```python
# Rhino
from compas.robots import model
from compas_rhino import xdraw_mesh

r = model.Robot.from_urdf_file('/urdf/ur5.urdf')
r.load_geometry()

for mesh in r.get_visual_meshes():
    v, f = mesh.to_vertices_and_faces()
    xdraw_mesh(v, f)
```



```python
# Blender
from compas.robots import model
from compas_blender import xdraw_mesh

r = model.Robot.from_urdf_file('/urdf/ur5.urdf')
r.load_geometry()

for mesh in r.get_visual_meshes():
    v, f = mesh.to_vertices_and_faces()
    xdraw_mesh(v, f)
```



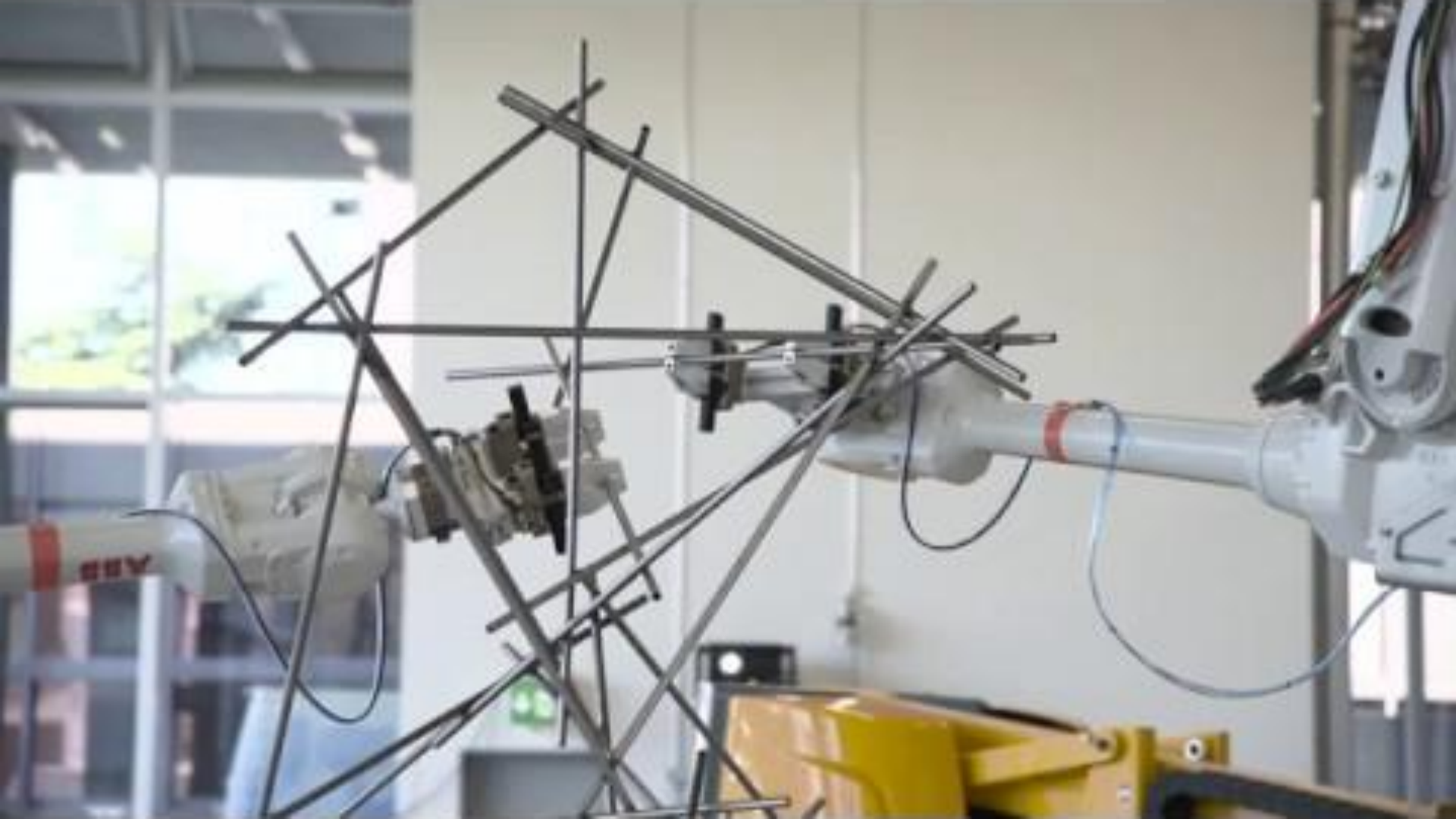
Robotic fabrication: backends

V-REP
Robot simulator

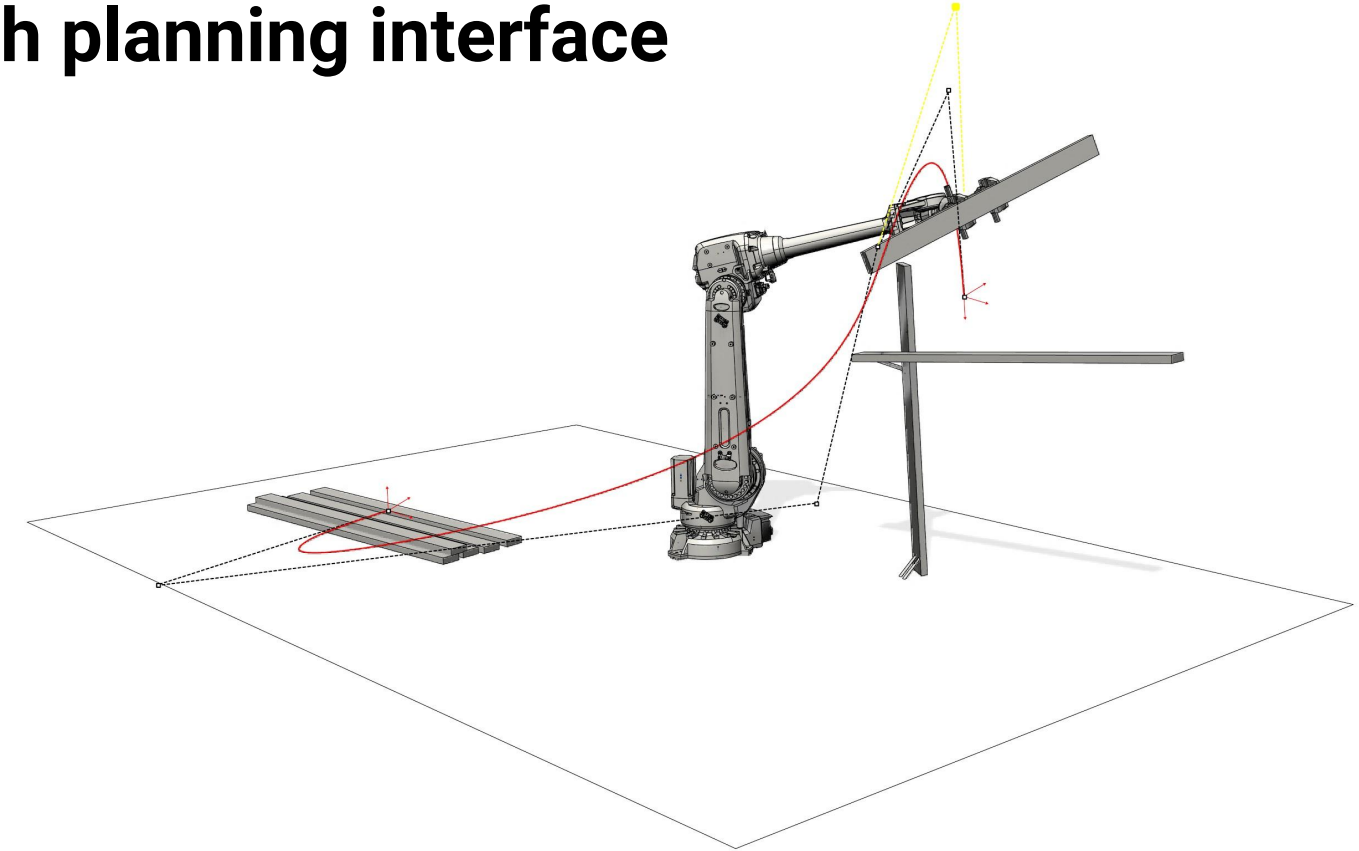
ROS
Robot Operating System

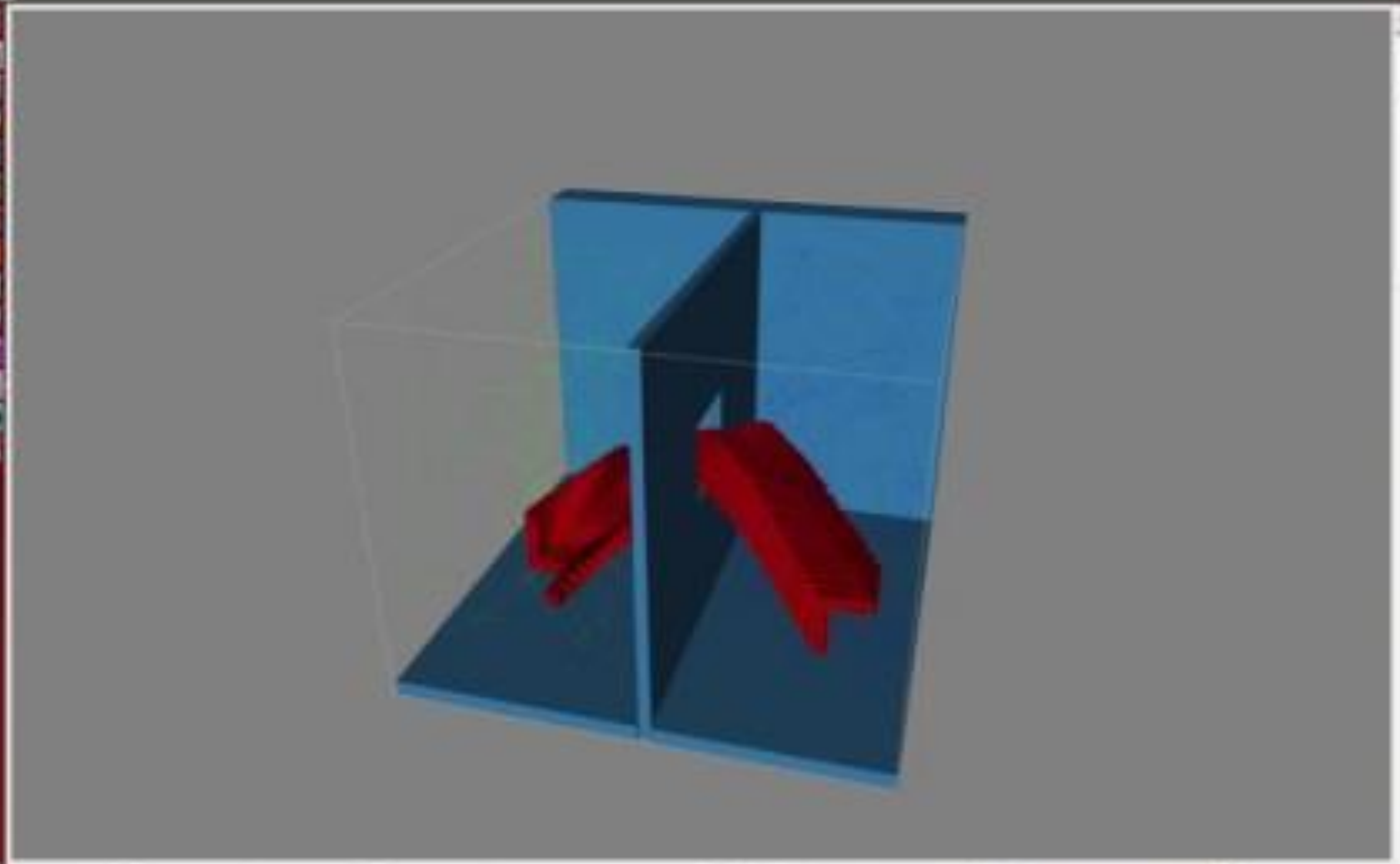
Moveit!
Motion Planning Framework

Path planning

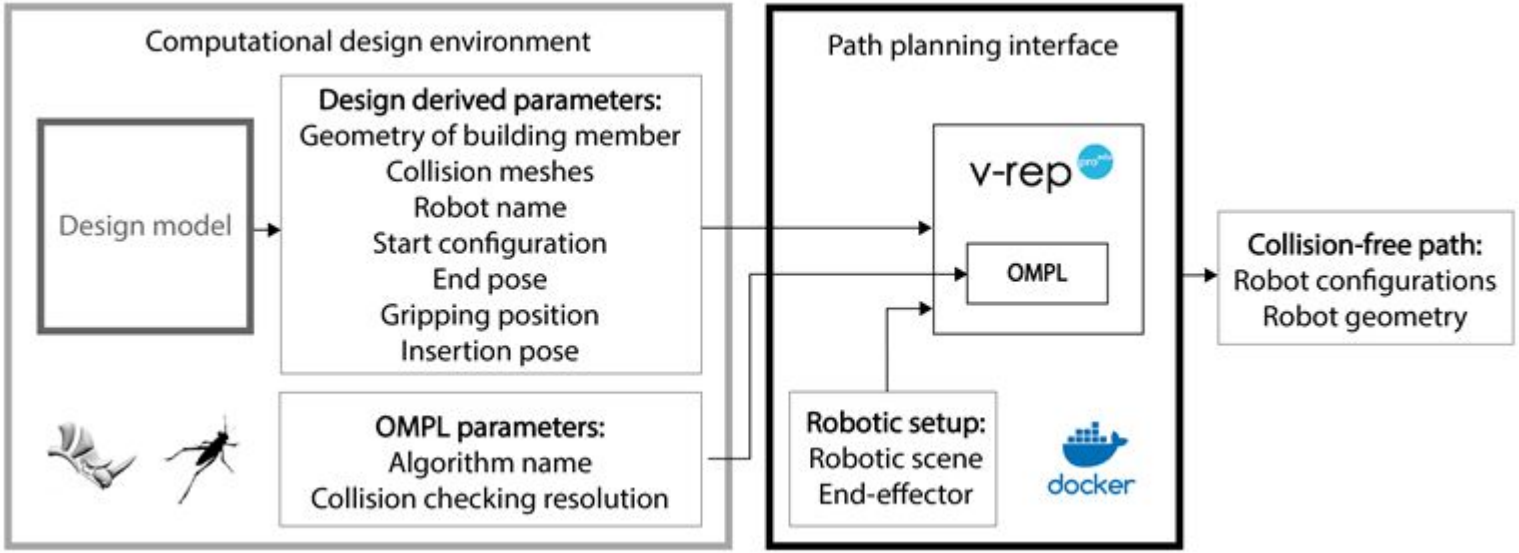


Path planning interface

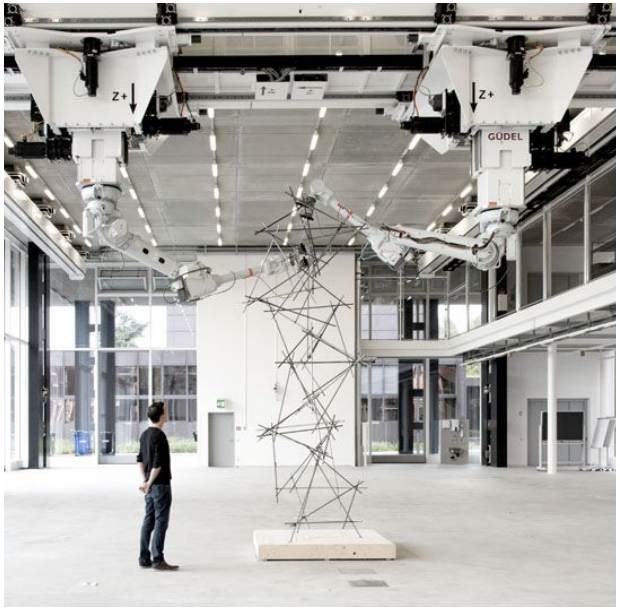




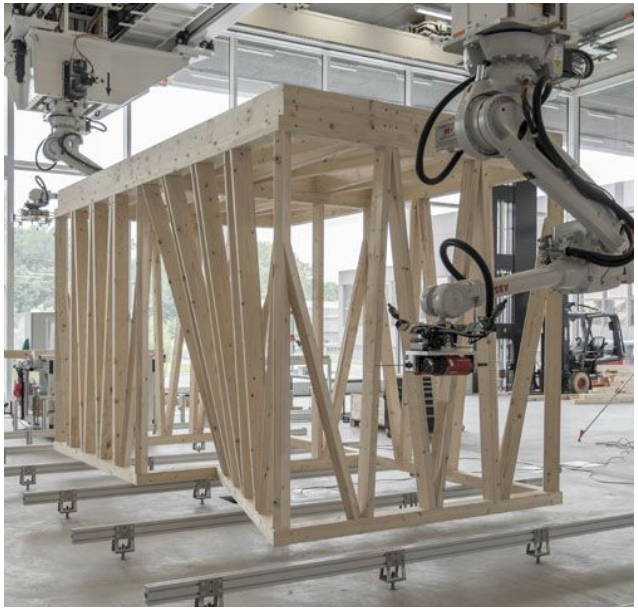
Path planning interface: V-Rep as backend



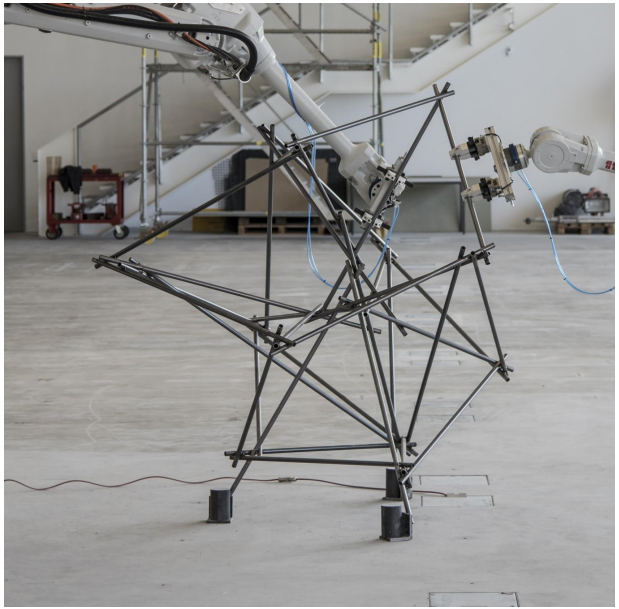
Path planning



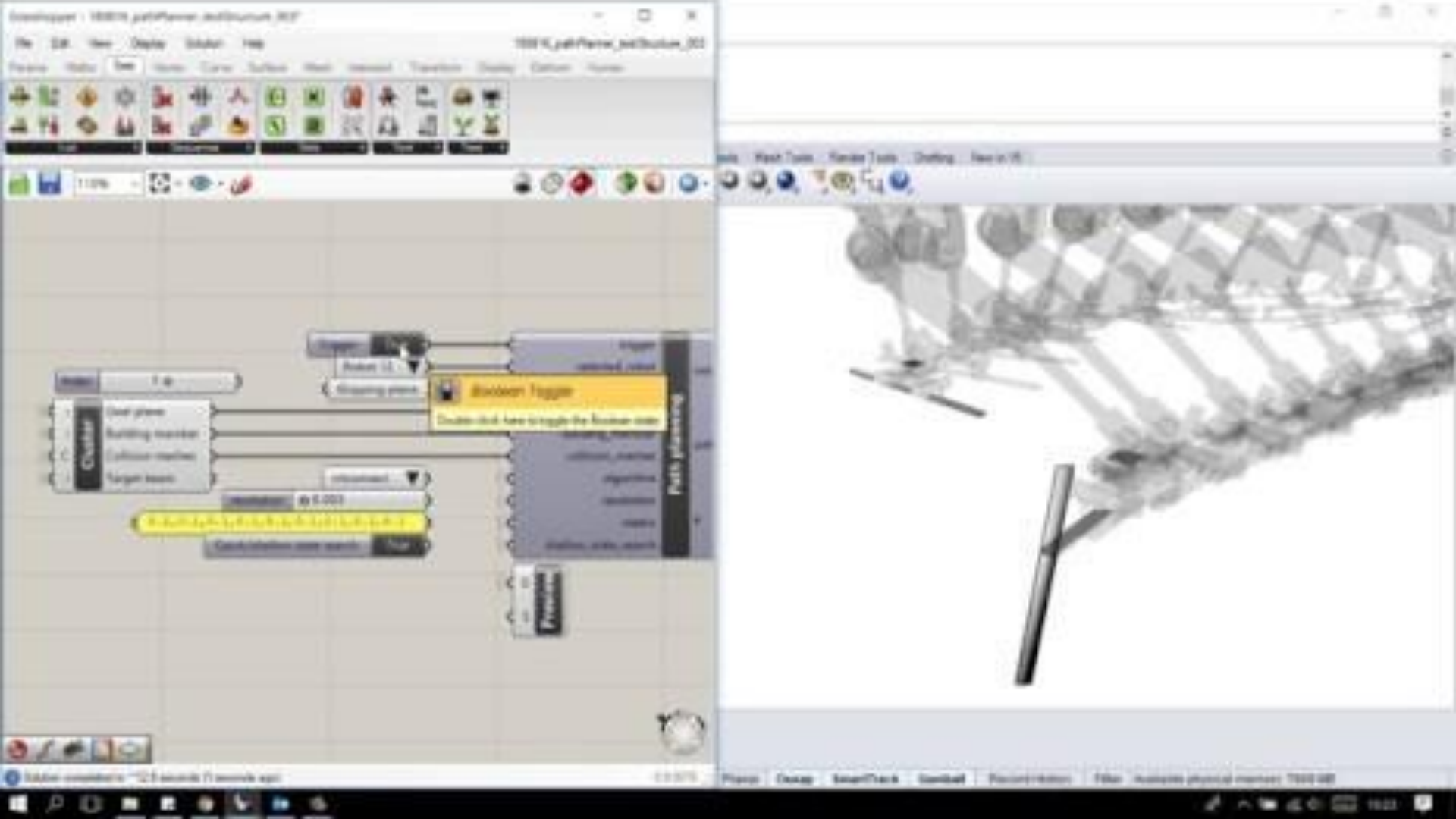
AAG Demonstrator of the “Robotic Lightweight Structures”, 2016



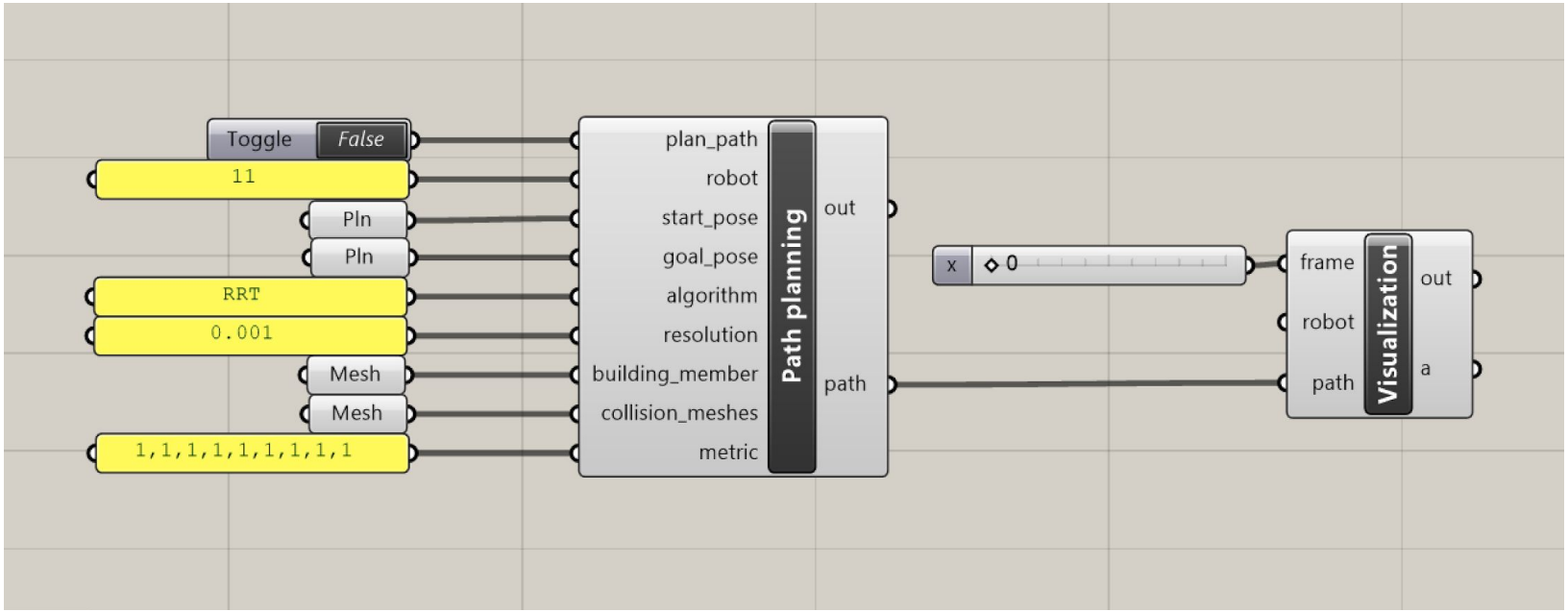
“The Spatial Timber Assemblies”, 2017



Second mockup of the “Robotic Lightweight Structures”, 2017


Boolean Toggle
Double-click here to toggle the Boolean state
Cluster
Path planning
Preview

Path planning: CAD integration



```python
# Complete path planning example
from compas.datastructures import Mesh
from compas.geometry import Frame
from compas_fab.robots import *
from compas_fab.robots import rfl
from compas_fab.backends import VrepClient

# Parameters
start_pose      = Frame((7.453, 2.905, 0.679), (1, 0, 0), (0, -1, 0))
goal_pose       = Frame((5.510, 5.900, 1.810), (0, 0, -1), (0, 1, 0))
algorithm       = 'rrtconnect'
resolution      = 0.02
building_member = Mesh.from_obj('timber_beam.obj')
structure       = [Mesh.from_obj('timber_structure.obj')]
metric          = [0.1] * 9

with VrepClient() as client:
   robot = rfl.Robot('A', client=client)
   simulator.pick_building_member(robot, building_member, start_pose)
   path = simulator.find_path_plan(robot, goal_pose, metric_values=metric, collision_meshes=structure,
                                   algorithm=algorithm, resolution=resolution, shallow_state_search=True)
```

```
# Complete path planning example
from compas.datastructures import Mesh
from compas.geometry import Frame
from compas_fab.robots import *
from compas_fab.robots import rfl
from compas_fab.backends import VrepClient

# Parameters
start_pose      = Frame((7.453, 2.905, 0.679), (1, 0, 0), (0, -1, 0))
goal_pose       = Frame((5.510, 5.900, 1.810), (0, 0, -1), (0, 1, 0))
algorithm       = 'rrtconnect'
resolution      = 0.02
building_member = Mesh.from_obj('timber_beam.obj')
structure       = [Mesh.from_obj('timber_structure.obj')]
metric          = [0.1] * 9

with VrepClient() as client:
   robot = rfl.Robot('A', client=client)
   simulator.pick_building_member(robot, building_member, start_pose)
   path = simulator.find_path_plan(robot, goal_pose, metric_values=metric, collision_meshes=structure,
                                   algorithm=algorithm, resolution=resolution, shallow_state_search=True)
```

```python
# Complete path planning example
from compas.datastructures import Mesh
from compas.geometry import Frame
from compas_fab.robots import *
from compas_fab.robots import rfl
from compas_fab.backends import VrepClient

# Parameters
start_pose      = Frame((7.453, 2.905, 0.679), (1, 0, 0), (0, -1, 0))
goal_pose       = Frame((5.510, 5.900, 1.810), (0, 0, -1), (0, 1, 0))
algorithm       = 'rrtconnect'
resolution      = 0.02
building_member = Mesh.from_obj('timber_beam.obj')
structure       = [Mesh.from_obj('timber_structure.obj')]
metric          = [0.1] * 9

with VrepClient() as client:
   robot = rfl.Robot('A', client=client)
   simulator.pick_building_member(robot, building_member, start_pose)
   path = simulator.find_path_plan(robot, goal_pose, metric_values=metric, collision_meshes=structure,
                                   algorithm=algorithm, resolution=resolution, shallow_state_search=True)
```


ROS
Open Source Robotics Foundation

What is ROS?

`plumbing`

- Process management
- Interprocess communication
- Device drivers

`tools`

- Simulation
- Visualization
- Graphical UI
- Data logging

`capabilities`

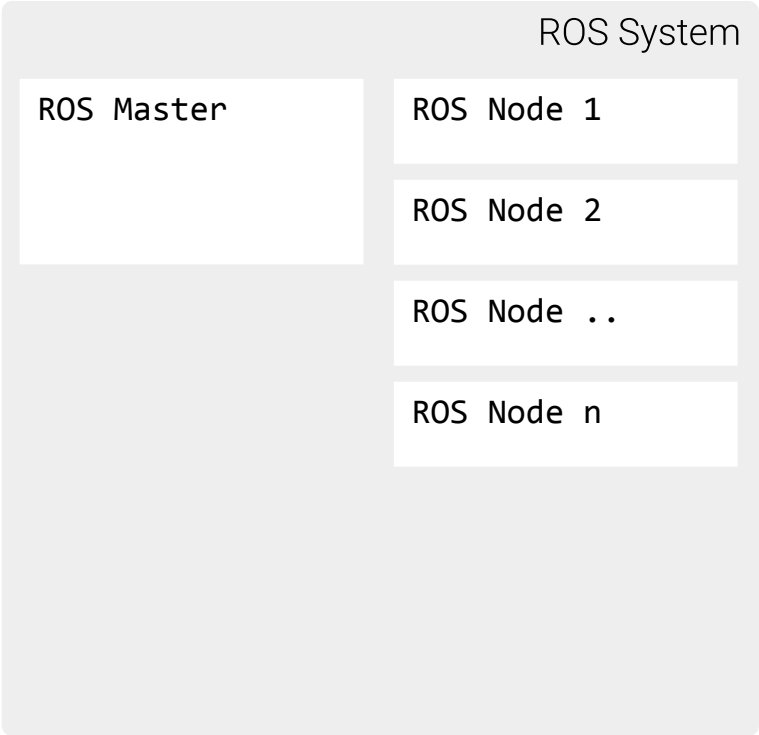
- Control
- Planning
- Perception
- Mapping
- Manipulation

`ecosystem`

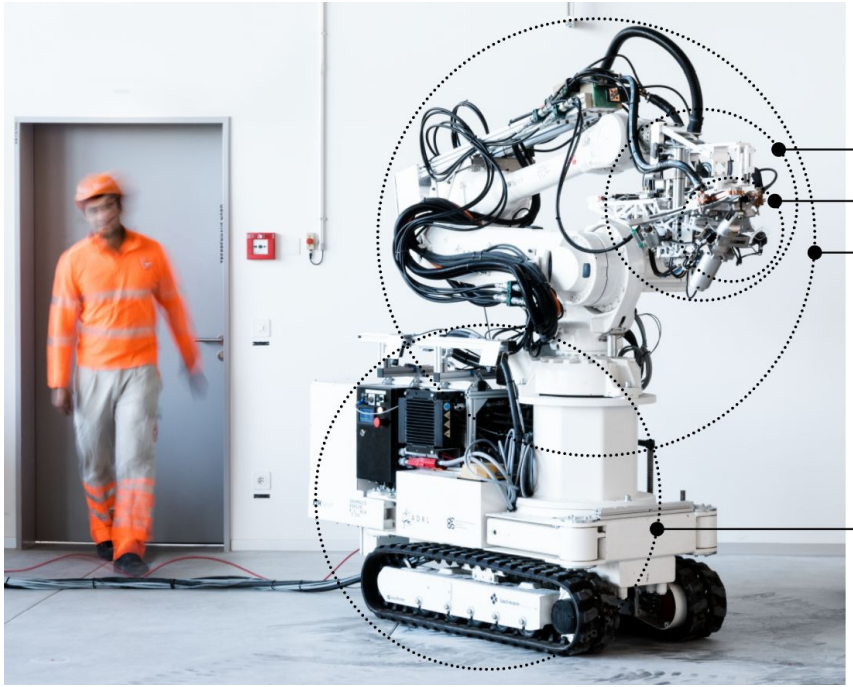
- Package organization
- Software distribution
- Community
- Documentation
- Tutorials

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

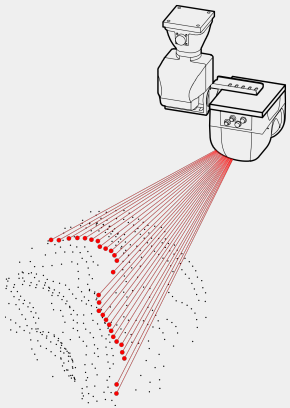
ROS Node 2: Localization

ROS Node 3: Robot Arm

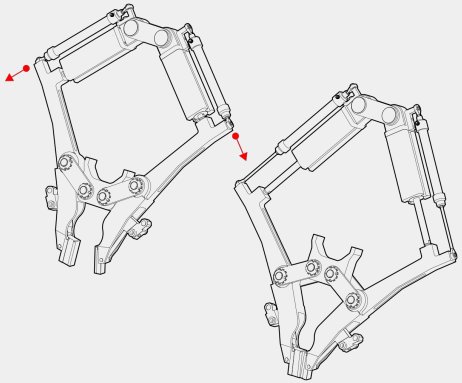
ROS Node 4: Mobile Base

ROS Concepts

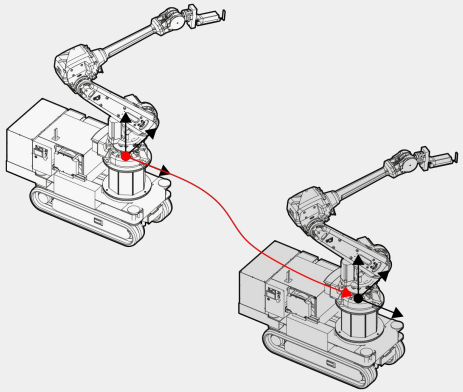
Topics



Services



Actions



ROS Concepts

Topics

- Nodes communicate over topics
- Publish/subscribe model
- One-way data stream
- Example: robot states (joints), sensor data

Services

- Blocking call per request
- Request/response model
- Short trigger or calculation
- Example: calculate path, open gripper

Actions

- Non-blocking requests
- Goal-oriented
- Cancellable (preemptable)
- Actually implemented with topics
- Example: navigation, motion execution

ROS Concepts: protocol

Message types

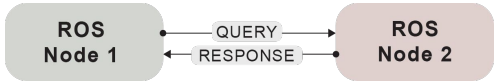
- Dictionary-like data structure
- Defines the type of topics
- Large collection of predefined types

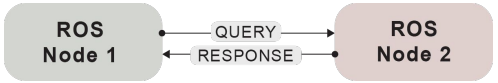
Service types

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

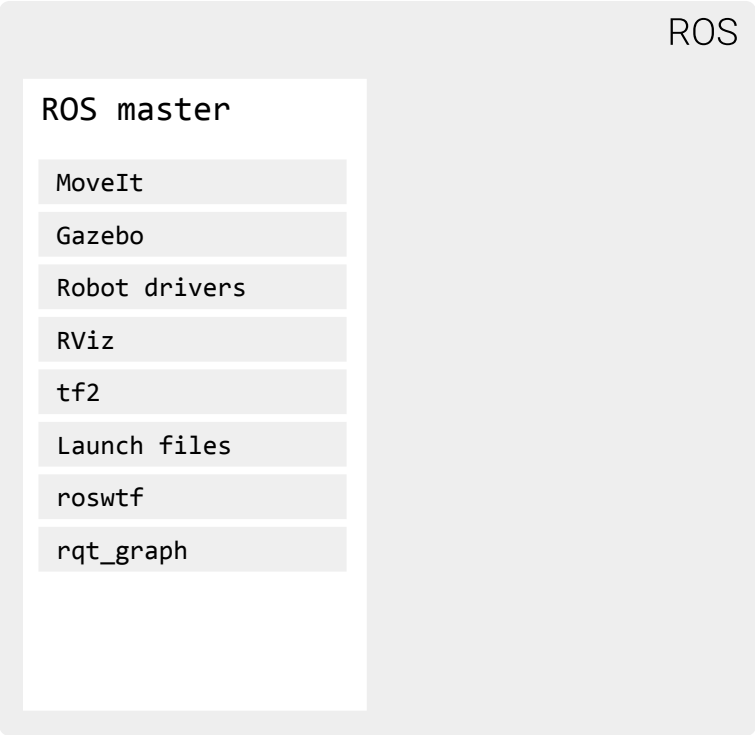
Parameters

- Global parameter values
- Constant
- ROS Parameter server





ROS



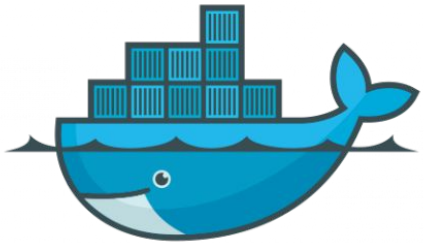
Very extensive tooling available:

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

Ease distribution


python™
Package
Index

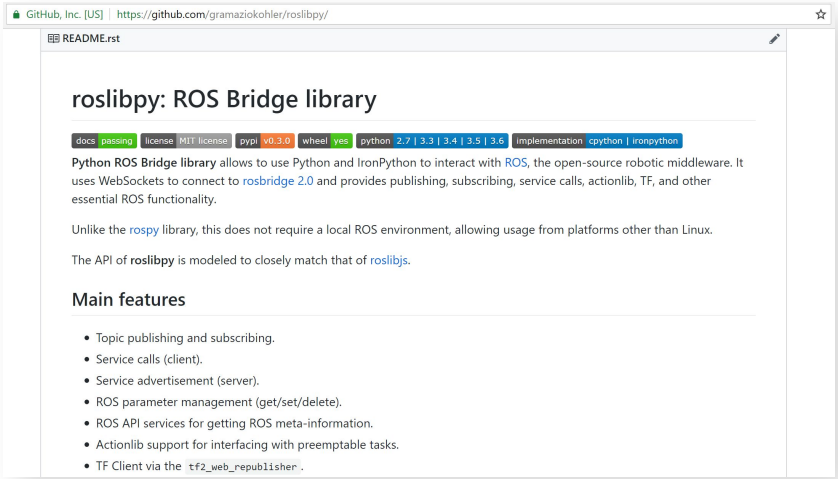

CONDA


docker

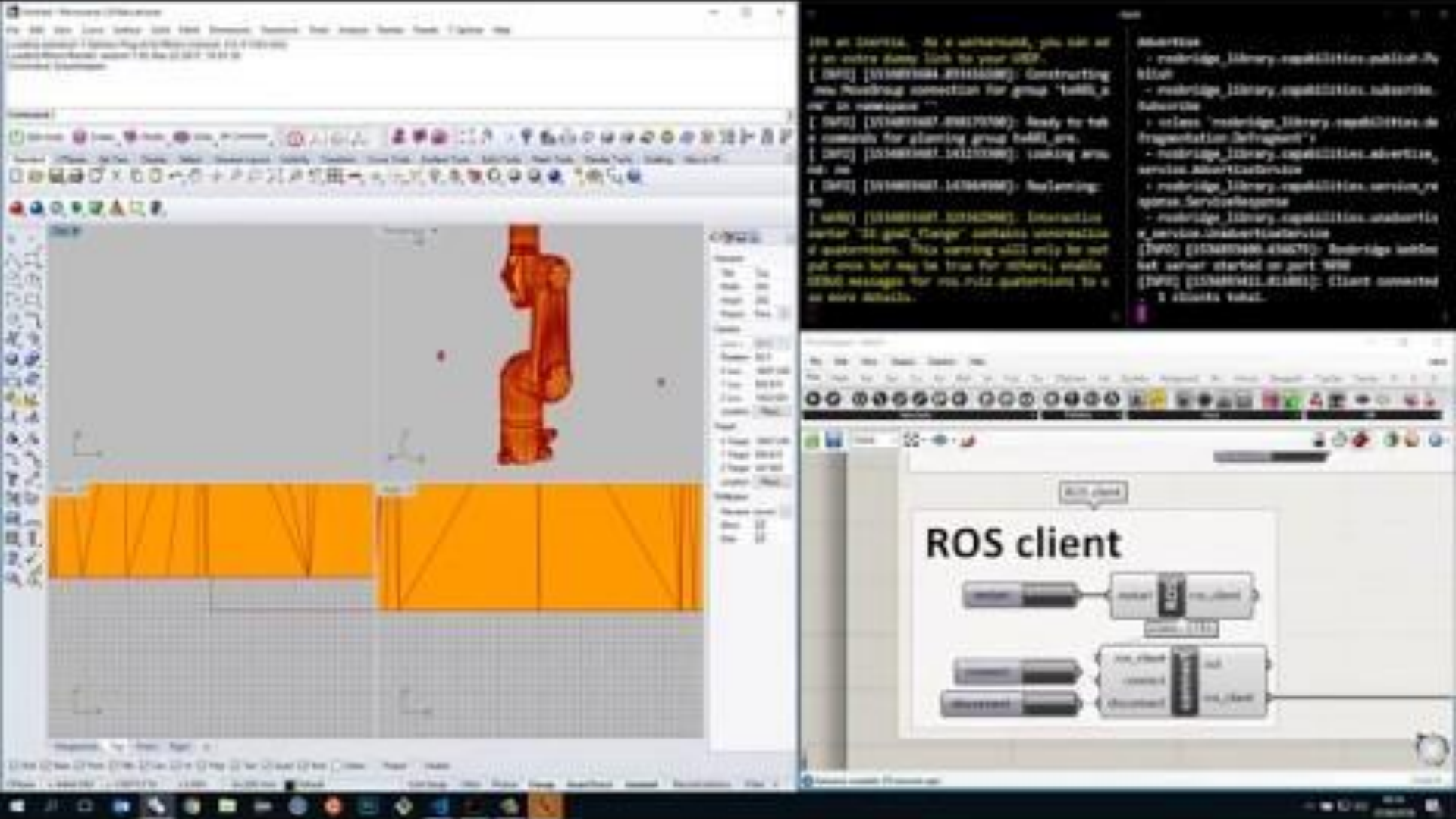
```
docker run -p 19997:19997 -d gramaziokohler/vrep-rfl
```

```
docker-compose up -d
```

ROS from CAD environments



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


ROS client

Hands-on time!

u.nu/lets-code

GRAMAZIO
KOHLER
RESEARCH

ETH zürich