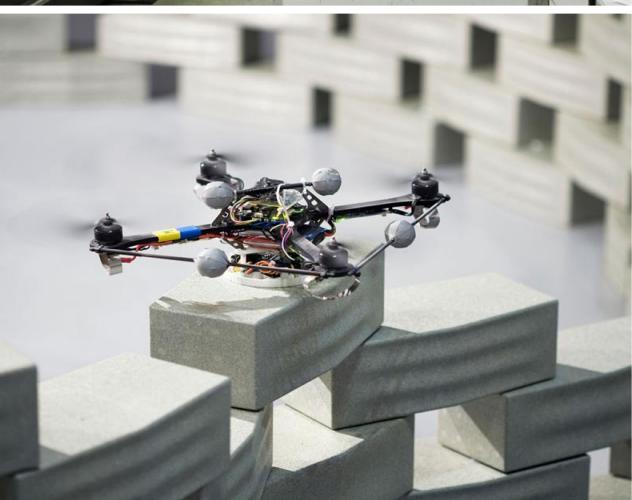
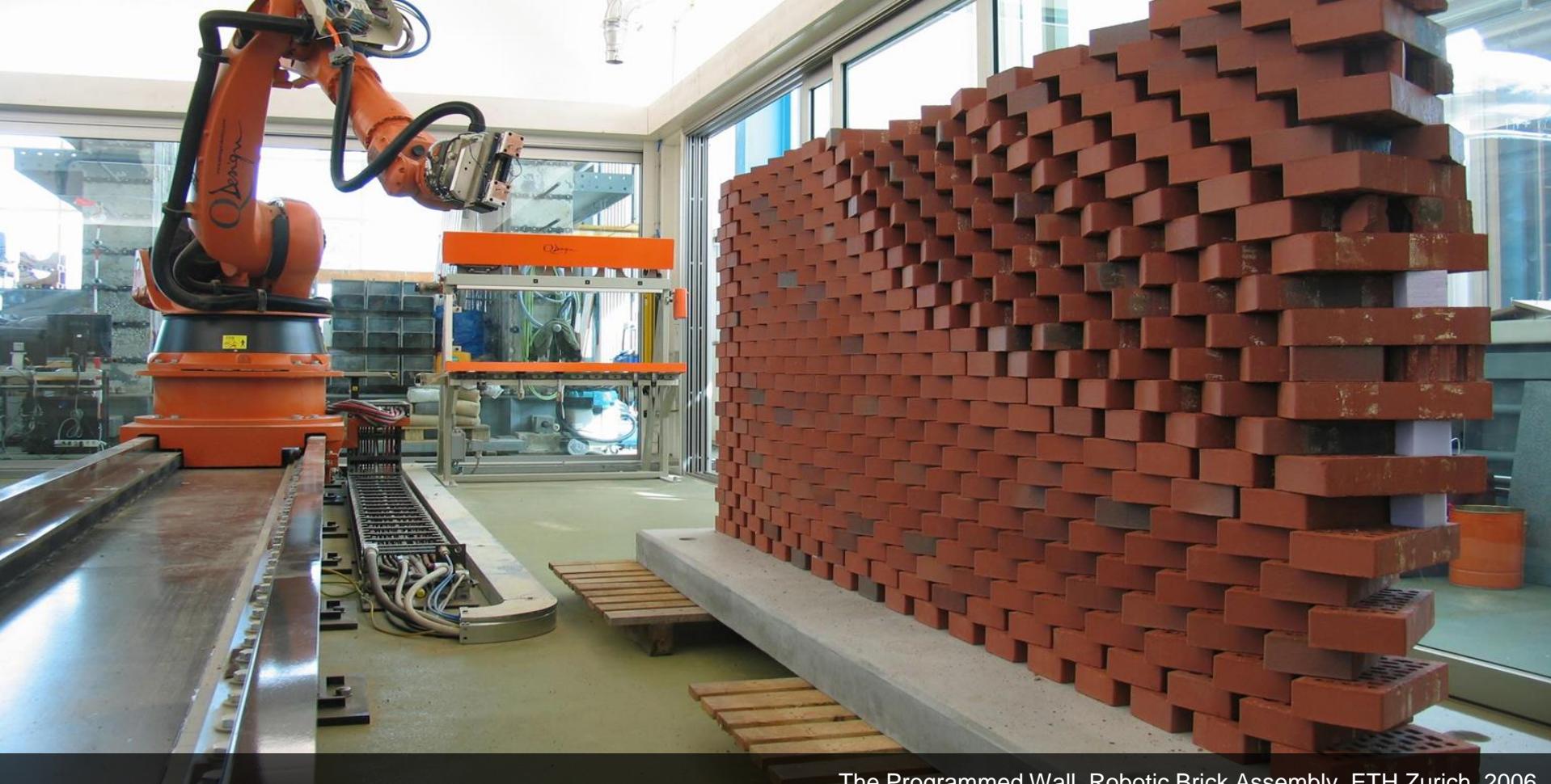


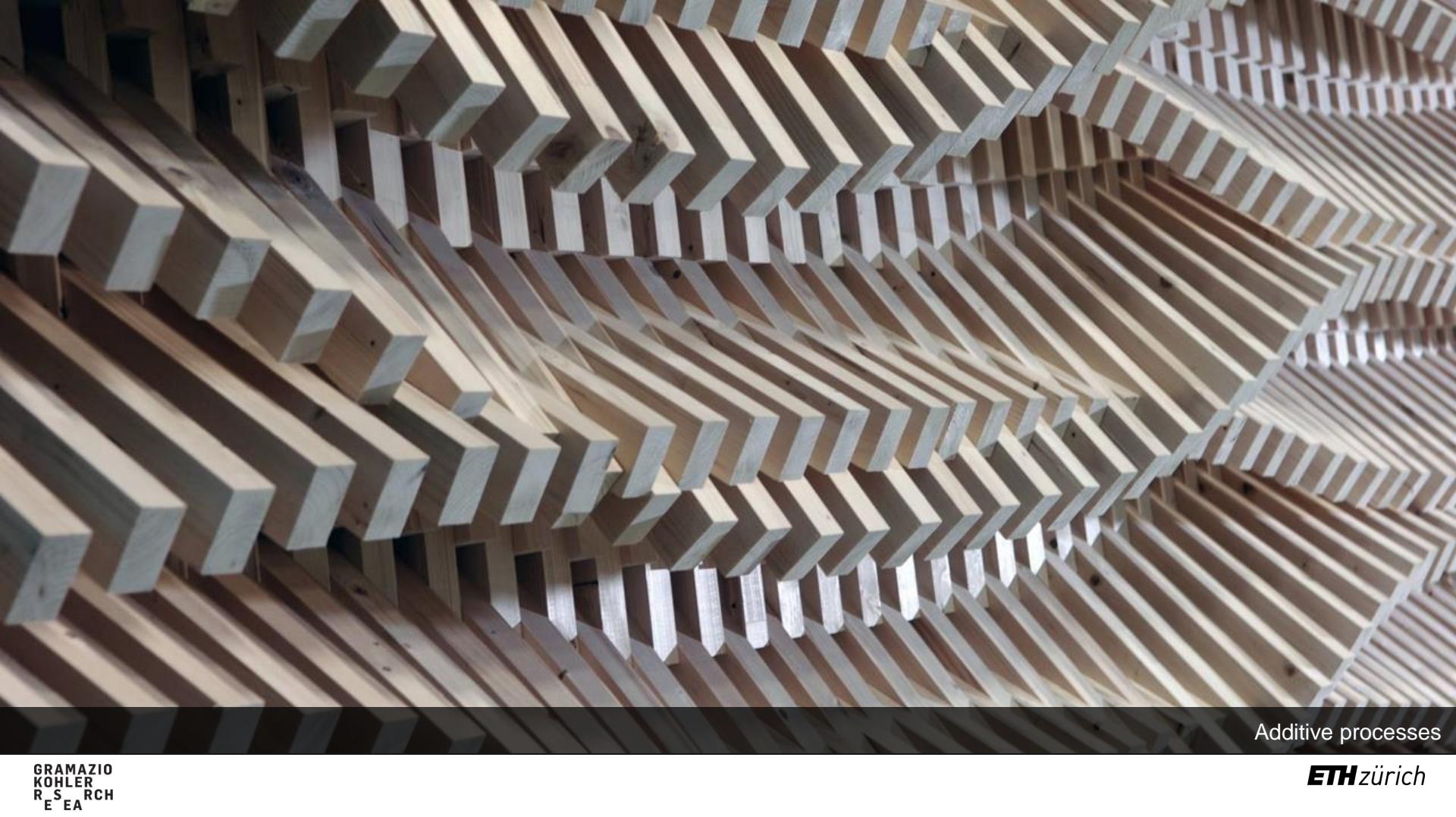
**GRAMAZIO
KOHLER
R S R C H
E E A**

***ETH* zürich**





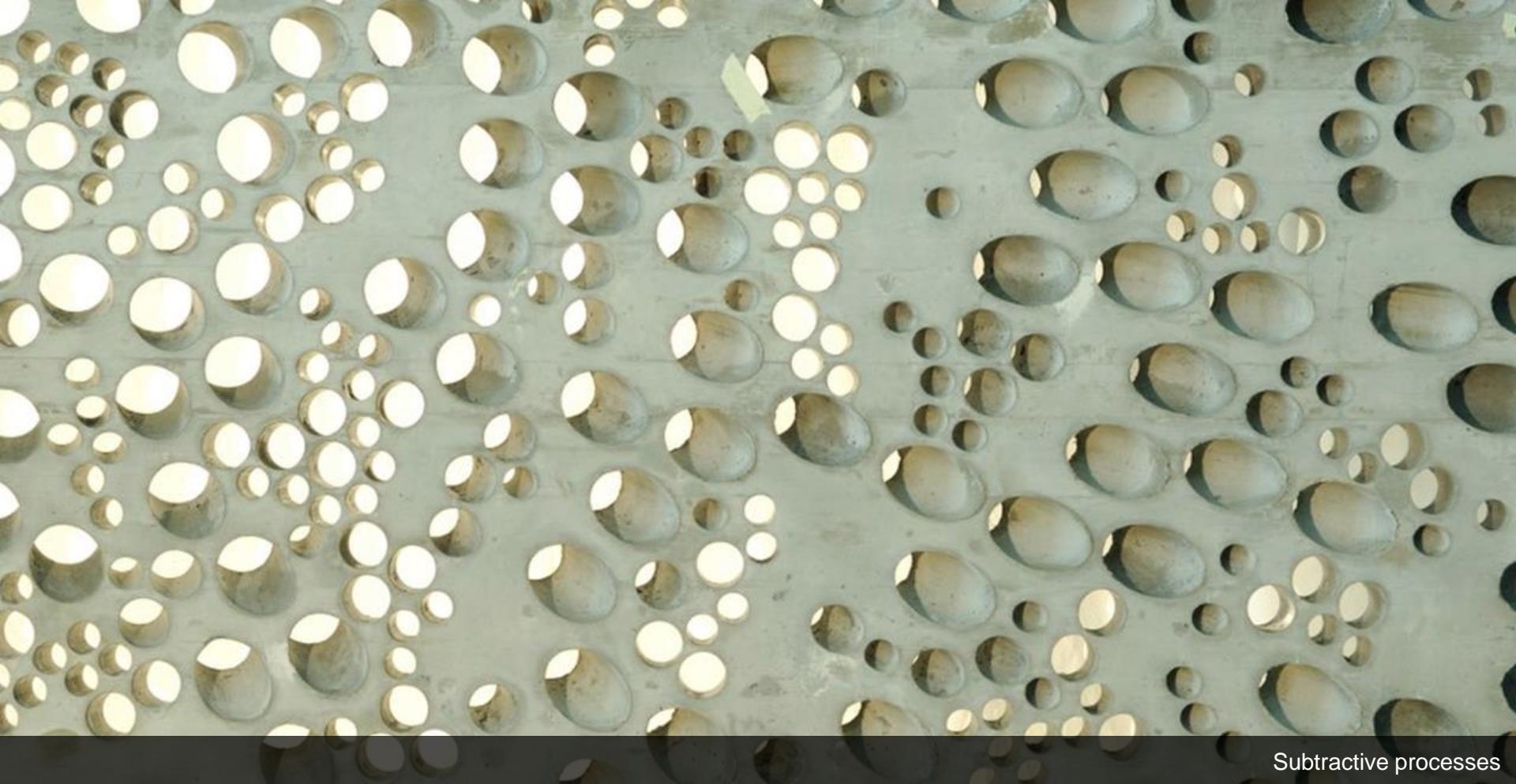
The Programmed Wall, Robotic Brick Assembly, ETH Zurich, 2006



Additive processes

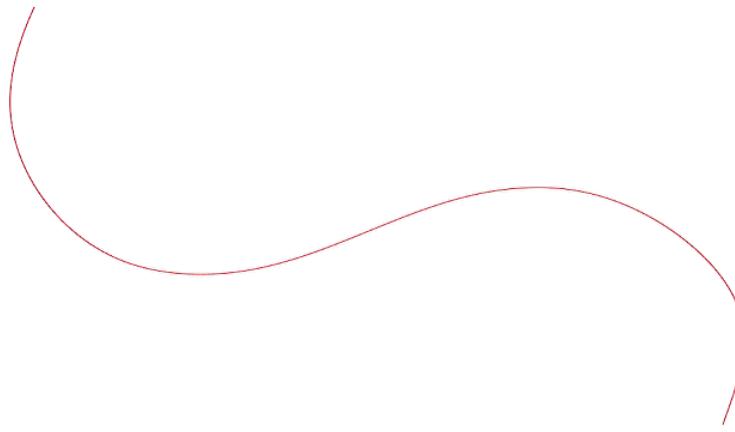


Formative processes



Subtractive processes

Computational Design





C O M P A S S

F A B

robots']):

ound at kwargs['rob

ose(settings['star

to_data()

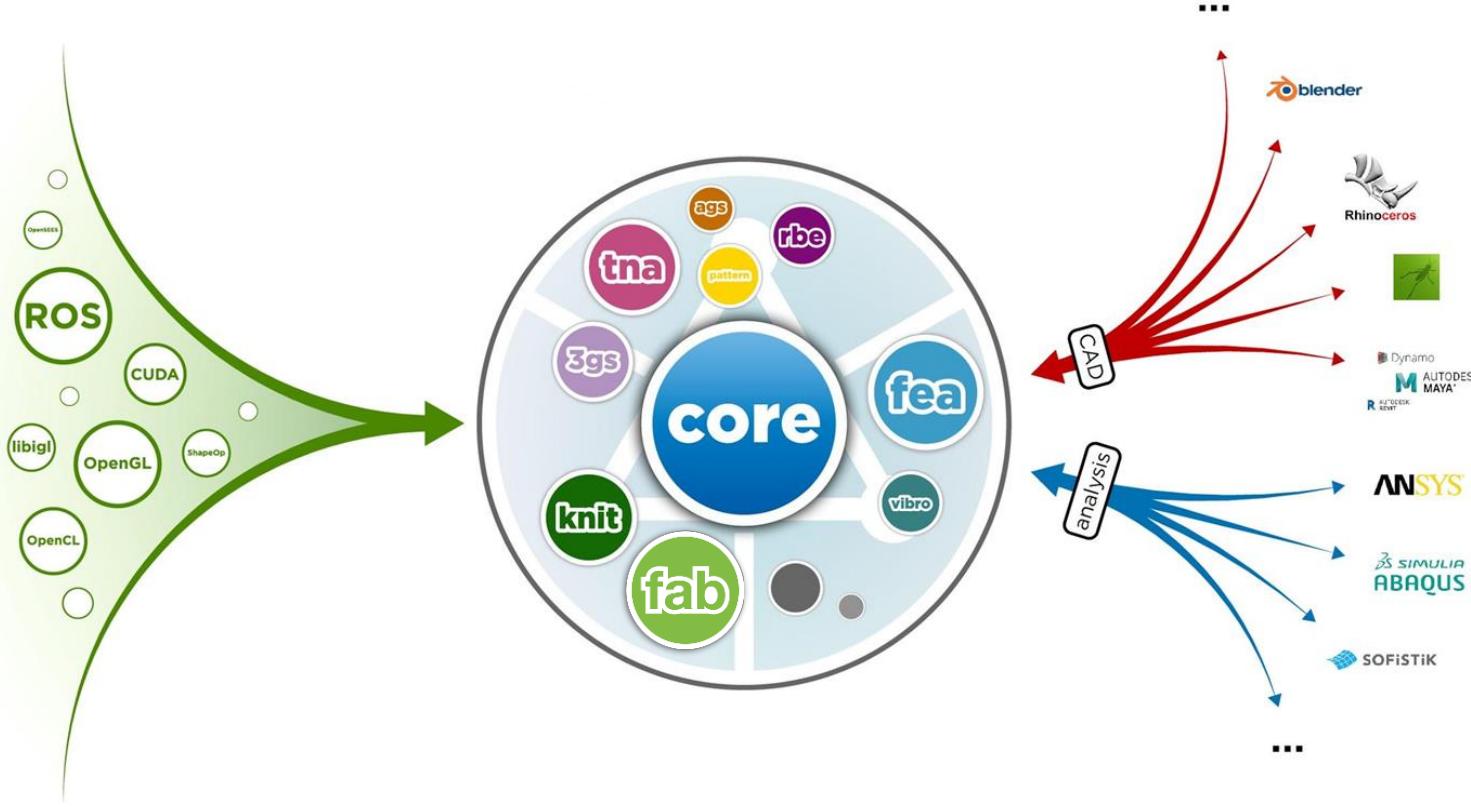
robot

get_config

_or_pose

(settings[

'goal']) = goal



```
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

```
=====
```

```
====
```

```
>>> □
```

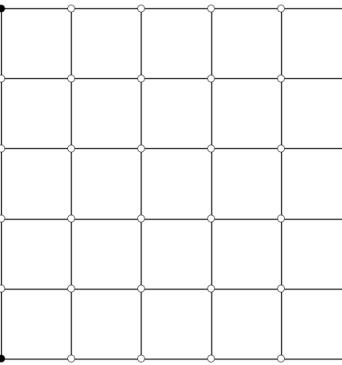
dr_numpy.py — compas-dev

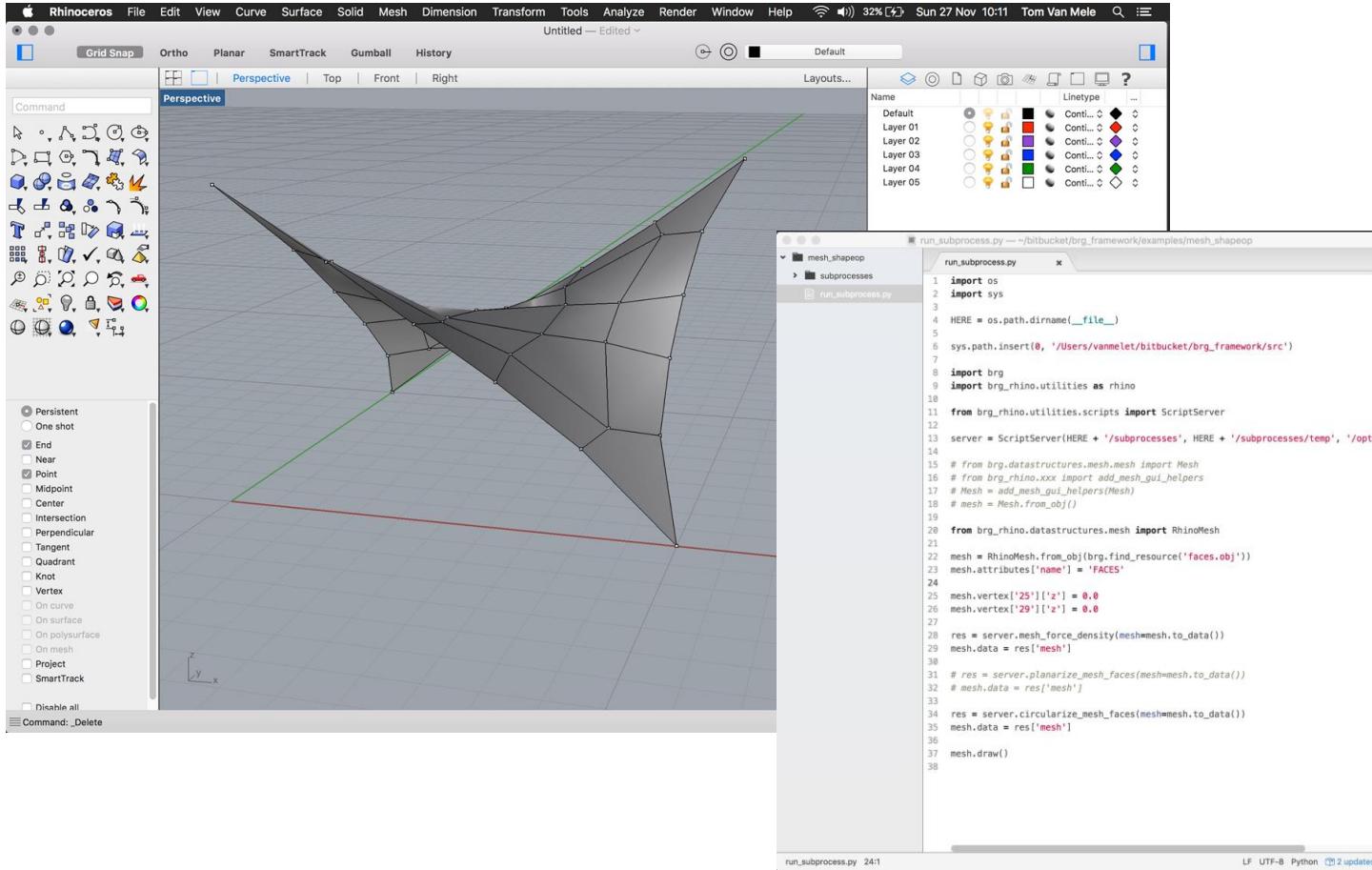
```

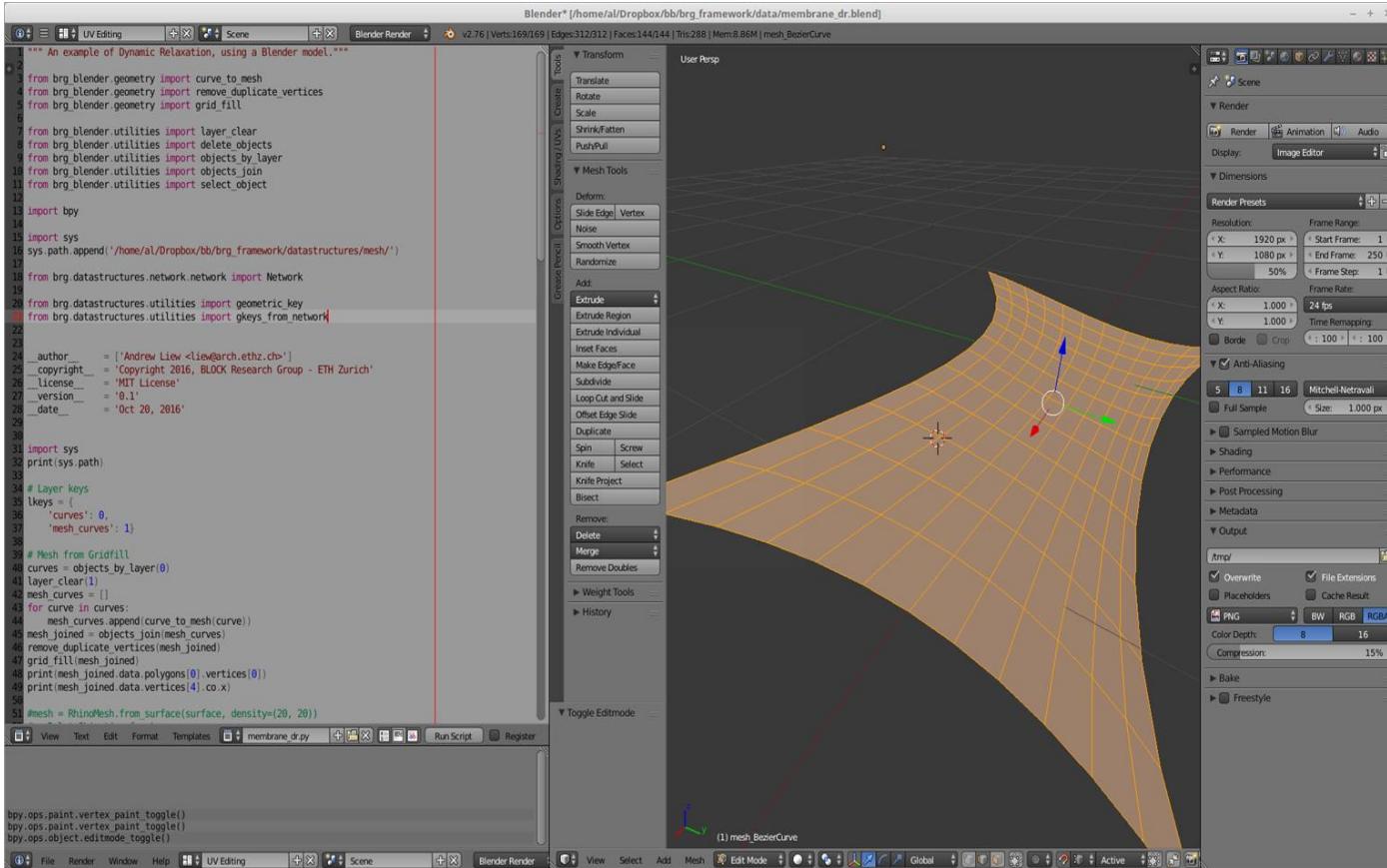
FOLDERS
├── __compas-dev
│   ├── __compas
│   ├── build
│   ├── data
│   ├── dist
│   ├── docs
│   ├── libs
│   └── src
│       ├── __compas
│       │   ├── __pycache__
│       │   ├── __com
│       │   ├── __datastructures
│       │   ├── __files
│       │   ├── __geometry
│       │   ├── __Interop
│       │   ├── __numerical
│       │   │   ├── __pycache__
│       │   │   ├── __algib
│       │   │   ├── __algib
│       │   ├── __algorithms
│       │   ├── __descent
│       │   ├── __devo
│       │   ├── __dr
│       │   ├── __drx
│       │   │   ├── __pycache__
│       │   │   ├── __init__.py
│       │   │   └── dr_numpy.py
│       │   ├── __fd
│       │   ├── __ga
│       │   ├── __ima
│       │   ├── __mma
│       │   ├── __pca
│       │   ├── __solvers
│       │   └── __topopt
│       ├── __init__.py
│       ├── __inalg.py
│       ├── __matrices.py
│       ├── __operators.py
│       ├── __utilities.py
│       ├── __plotters
│       ├── __robots
│       ├── __rpc
│       ├── __topology
│       ├── __utilities
│       ├── __viewers
│       └── __init__.py
│           ├── __os.py
│           └── __compas-gsg-info
│               ├── COMPAS.gsg-info
│               ├── __compas_gsgender
│               ├── __compas_gsgpython
│               ├── __compas_hpc
│               ├── __compas_revit
│               ├── __compas_rhino
│               ├── __temp
│               └── __tests
│                   ├── __umpversion.cfg
│                   ├── __editconfig
│                   ├── __gitignore
│                   ├── __travyaml
│                   └── __CONTRIBUTORS.md
└── LICENSE

dr_numpy.py
1  from __future__ import absolute_import
2  from __future__ import division
3  from __future__ import print_function
4
5  import compas
6
7  try:
8      from numpy import array
9      from numpy import meshgrid
10     from numpy import isnan
11     from numpy import ones
12     from numpy import zeros
13     from numpy.linalg import norm
14     from scipy.sparse import diags
15
16 except ImportError:
17     compas.raise_if_not_ipython()
18
19 from compas.numerical import connectivity_matrix
20 from compas.numerical import normrow
21
22
23 __all__ = ['dr_numpy']
24
25
26 K = [
27     [0.4, 0.5, 0.5],
28     [0.5, 0.5, 0.5],
29     [0.5, 0.5, 0.5],
30     [0.5, 0.5, 0.5]
31 ]
32
33
34 class Coeff():
35     def __init__(self, c):
36         self.c = c
37         self.a = (1 - c + 0.5) / (1 + c + 0.5)
38         self.b = 0.5 / (1 + self.a)
39
40
41 def dr_numpy(vertices, edges, fixed, loads, arms, force, force_type, limit, E, radius,
42             callback=None, callback_args=None, **kwargs):
43     """Implementation of the dynamic-relaxation method for form finding and analysis
44     of articulated networks of axial-force members.
45
46     Parameters
47
48     vertices : list
49         XYZ coordinates of the vertices.
50     edges : list
51         Connectivity of the vertices.
52     fixed : list
53         Indices of the fixed vertices.
54     loads : list
55         XYZ components of the loads on the vertices.
56     arms : list
57         Prescribed force densities in the edges.
58     force : list
59         Prescribed forces in the edges.
60     force_type : list
61         Prescribed lengths of the edges.
62     limit : float
63         Initial length of the edges.
64     E : float
65         Stiffnesses of the edges.
66     radius : list
67         Radius of the edges.
68     callback : callable, optional
69         User-defined function that is called at every iteration.
70     callback_args : tuple, optional
71         Additional arguments passed to the callback.
72
73 Notes
74 -----
75 For more info, see [1]_.
76
77 References
78 -----
79 .. [1] De Lant L., Veenendaal D., Van Hee T., Mulder M. and Block P.,
80     (submitting) Incorporating Designing Tension structures by Integrating S
81     Proceedings of Tensionit Symposium 2013, Istanbul, Turkey, 2013.
82
83 Examples
84 -----
85 .. code-block:: python
86
87     import compas
88     From compas.datastructures import Network

```







Fundamentals

Robot models

Backends

```
# 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()
```

```
# 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()
```

```
# 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()
```

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

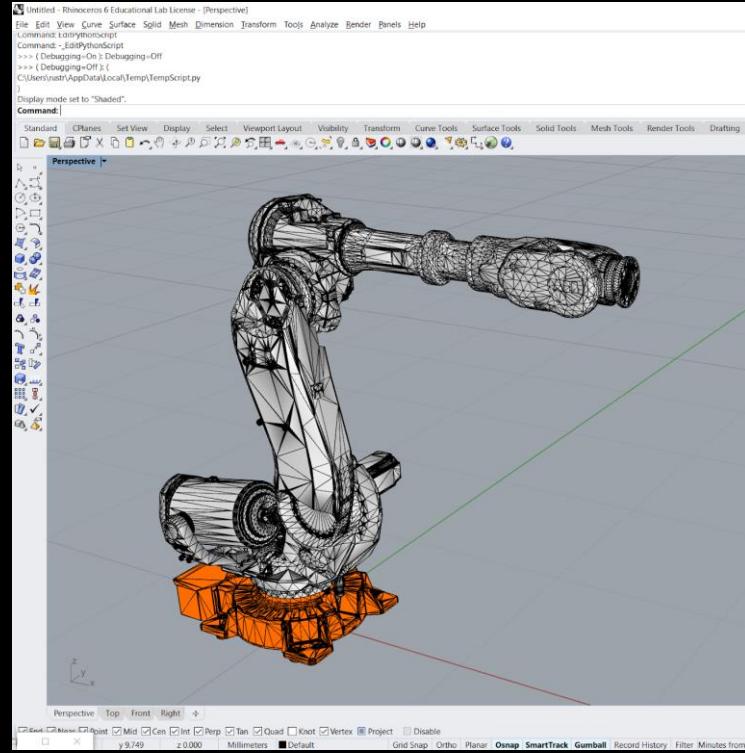
r = 'ros-industrial/abb'
p = 'abb_irb6600_support'
b = 'kinetic-devel'

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

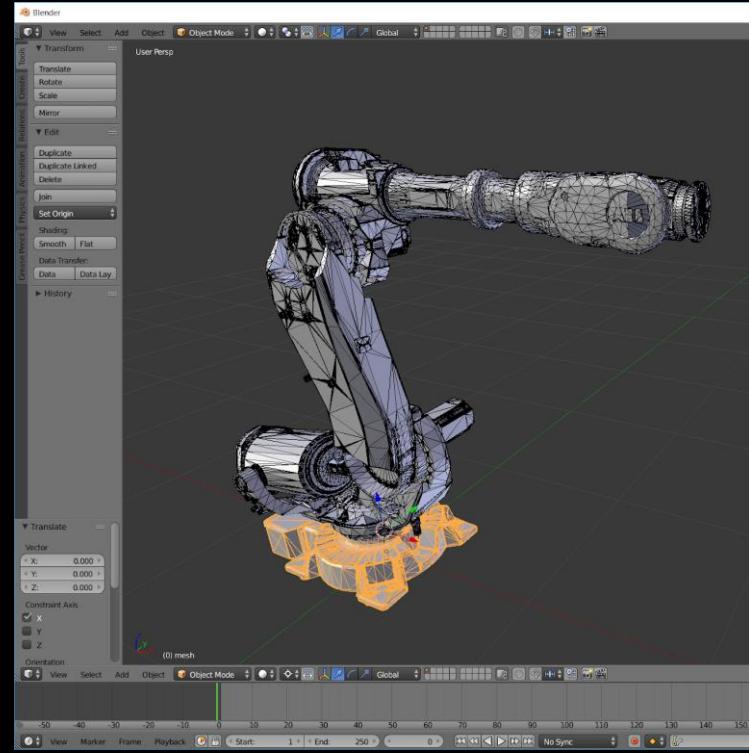
robot = Robot.from_urdf_file(urdf)
robot.load_geometry(github)

RobotArtist(robot).draw_visual()
```

Rhino



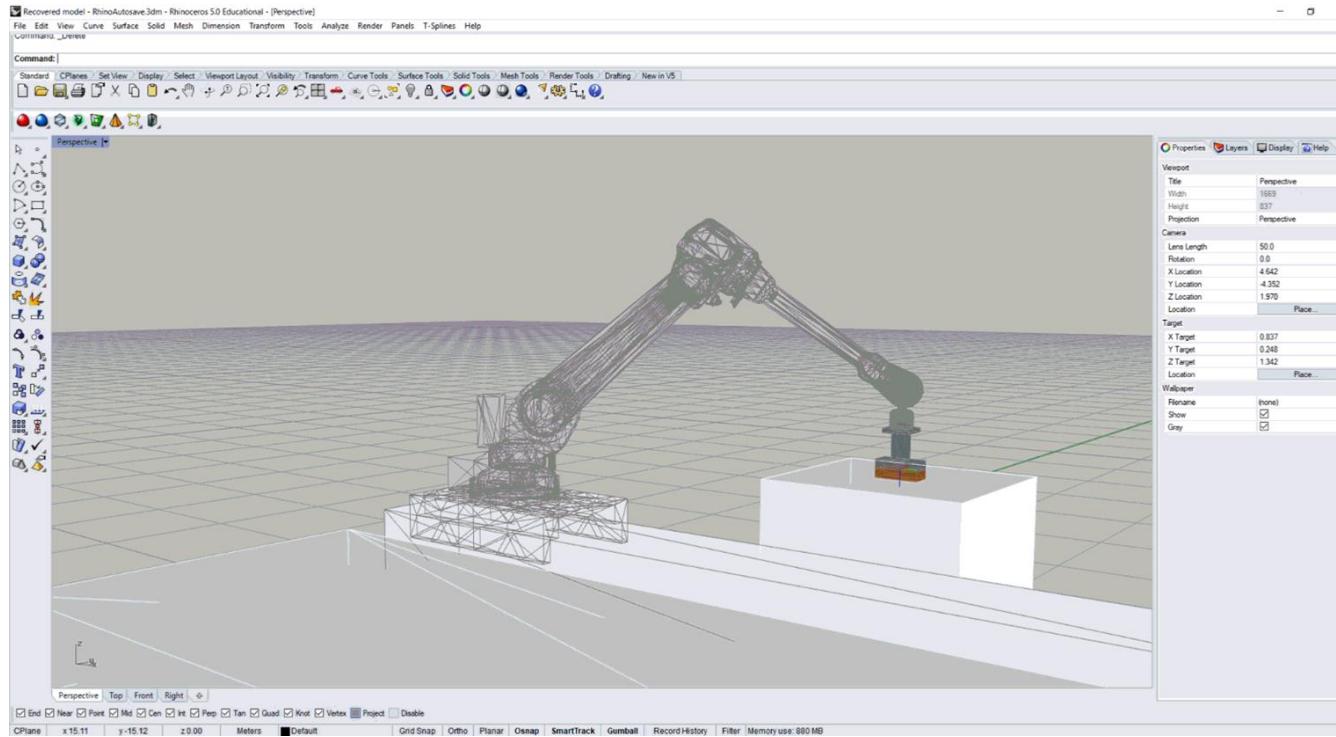
Blender

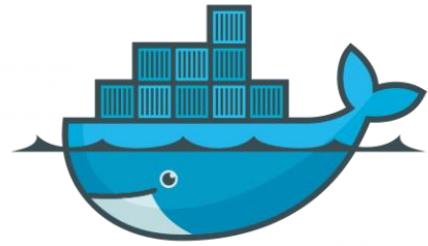


ROS

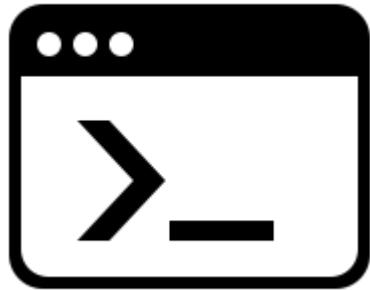


> MoveIt!

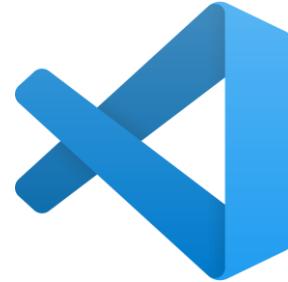




docker

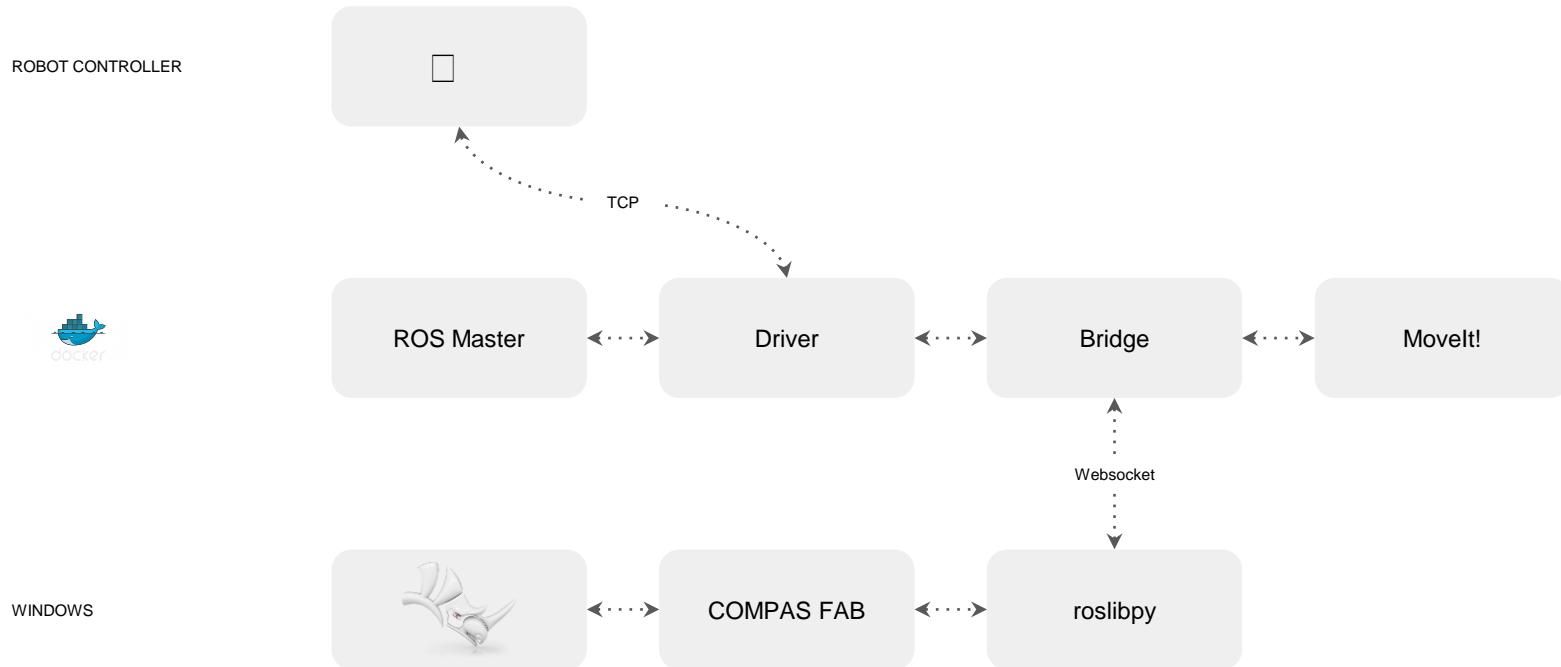


`docker-compose up -d`



Right-click → Compose Up

Overview: all the moving parts



Forward Kinematics

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

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

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

frame_RCF = robot.forward_kinematics(configuration)
```

Inverse Kinematics

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

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

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

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

Plan cartesian motion

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

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

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

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

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

Plan motion

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

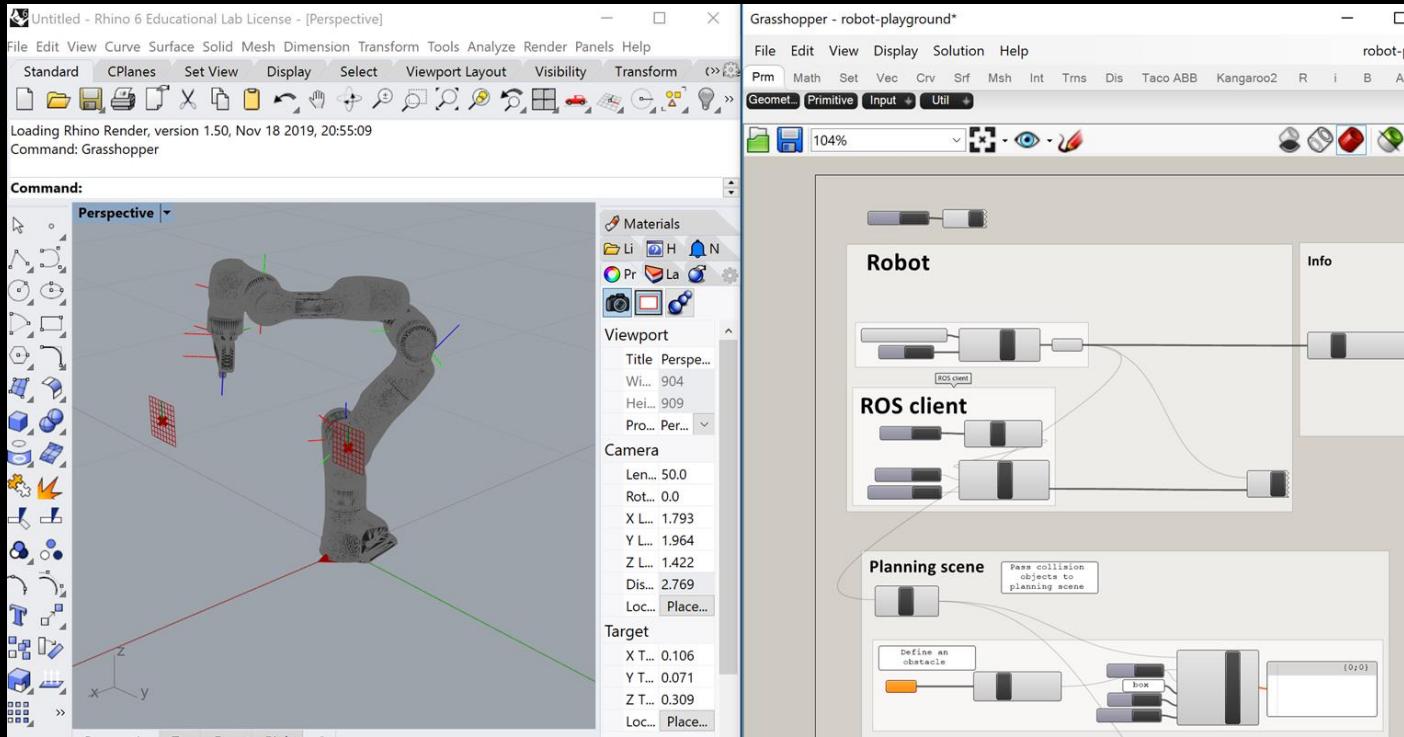
with RosClient() as client:
    robot = client.load_robot()

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

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

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

Rhino + Grasshopper + ROS



Thanks!