This lecture will be recorded

ETH zürich

COMPAS

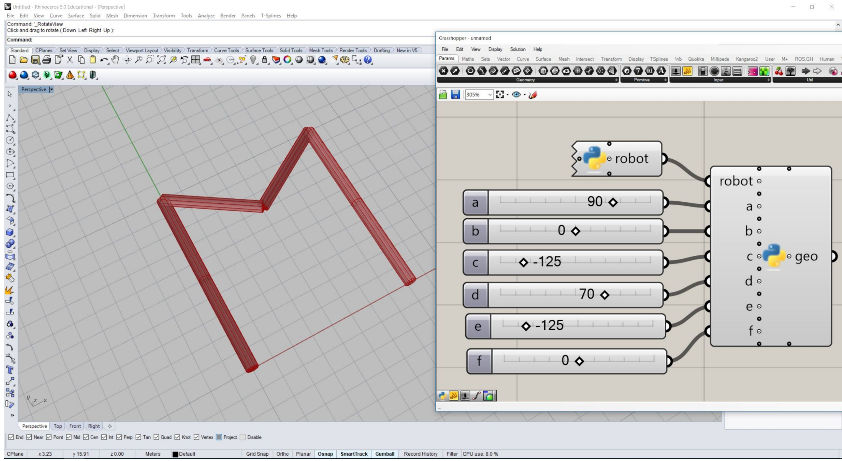
064-0026-00L: COMPAS II

Introduction to Computational Methods for Digital Fabrication in Architecture slides and code

`https://tiny.cc/compas-ii`

Review of last week's assignment

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



TODAY

robot backends: ros intro

interconnecting nodes

planning with moveit

Today's goal

Understand **basics of ROS and MoveIt** for planning

robot backends: ros intro

interconnecting nodes

planning with moveit

Robotic backends

`V-REP`
Robot simulator

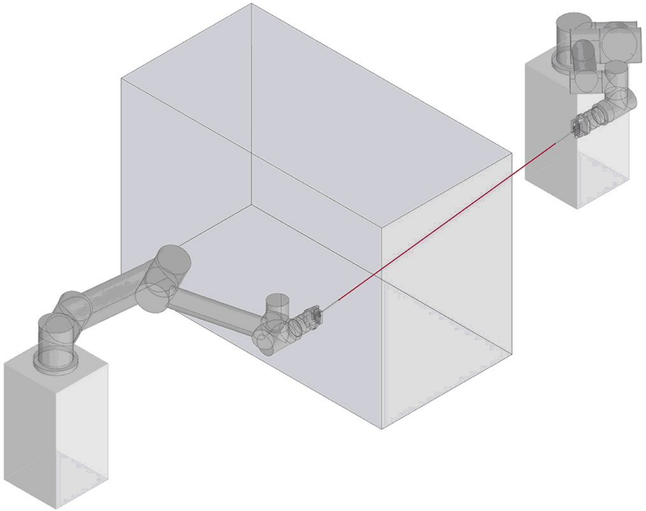
`ROS + MoveIt!`
Robot Operating System

`Pybullet`
Physics library

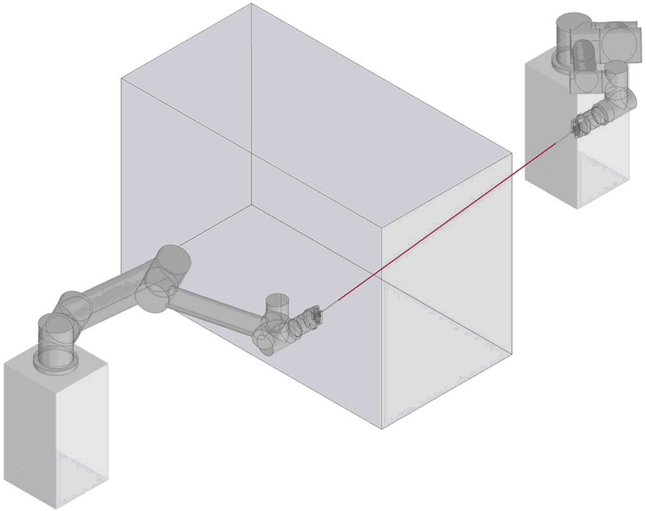
ROS

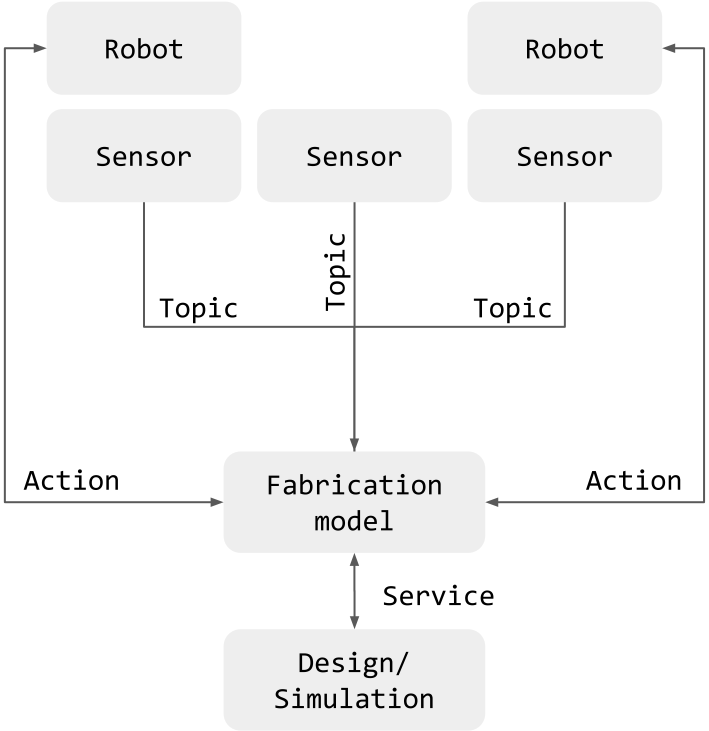
Open Source Robotics Foundation

Fabrication processes

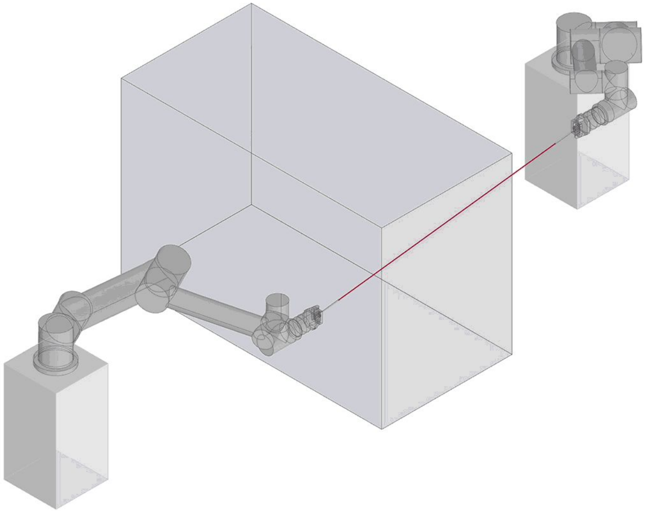


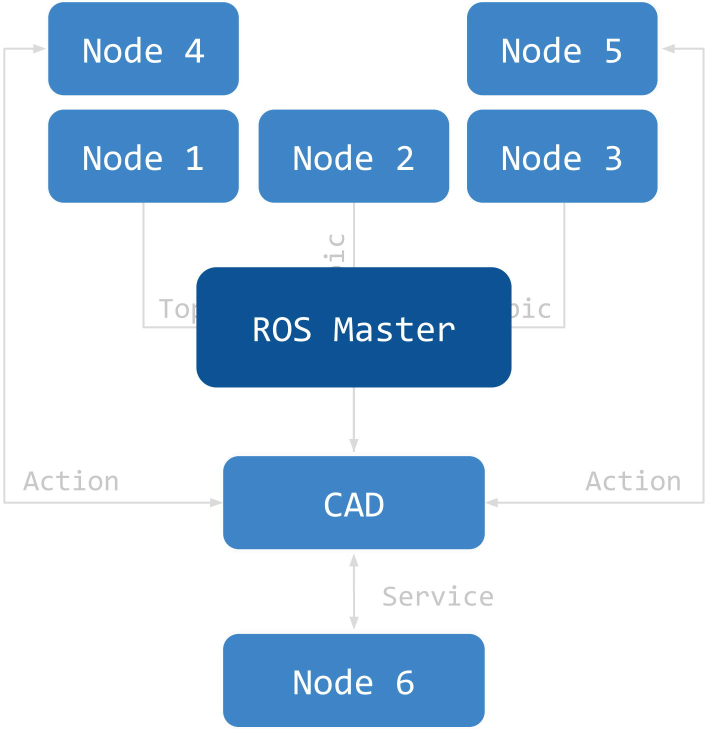
Fabrication processes



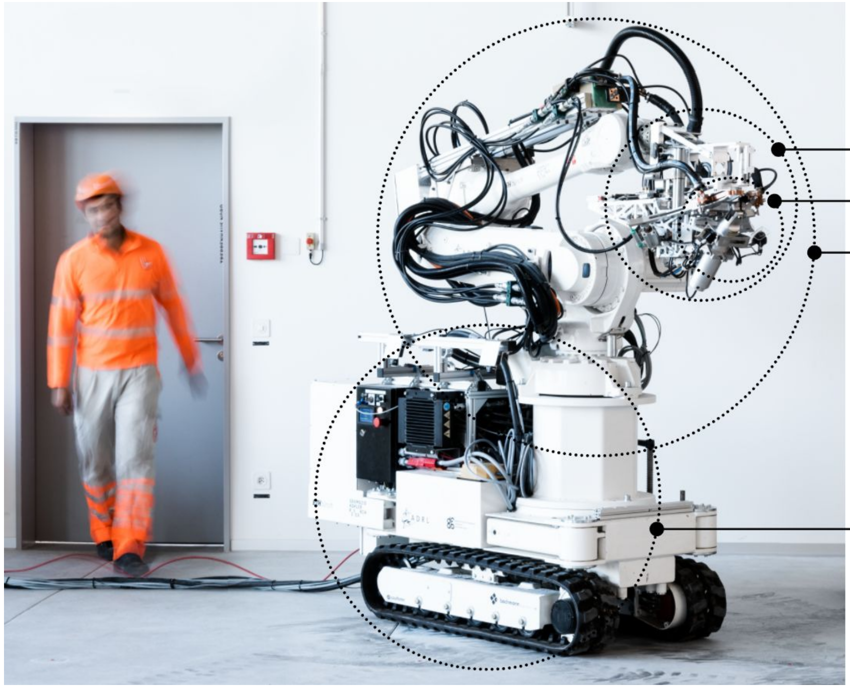


Fabrication processes





What is ROS?



ROS Node 1: End Effector

ROS Node 2: Localization

ROS Node 3: Robot Arm

ROS Node 4: Mobile Base

What is ROS?

`plumbing`

- Process management
- Interprocess communication
- Device drivers

`tools`

- Simulation
- Visualization
- Graphical UI
- Data logging

`capabilities`

- Control
- Planning
- Perception
- Mapping
- Manipulation

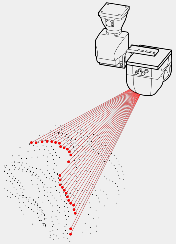
`ecosystem`

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

ROS Concepts

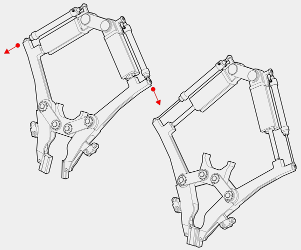
`Topics`

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



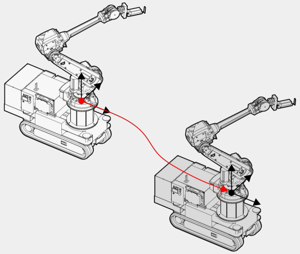
`Services`

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



`Actions`

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



docker

`docker-compose up -d`

Right-click → Compose Up

docker/first_steps

Verify connection to ROS

```
from compas_fab.backends import RosClient

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

Verify connection to ROS

```
from compas_fab.backends import RosClient

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

robot backends: ros intro

interconnecting nodes

planning with moveit

Publish to topics

Publish to topic: **connect**

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

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

Publish to topic: **advertise**

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

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

Publish to topic: **publish**

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

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

Subscribe to topics

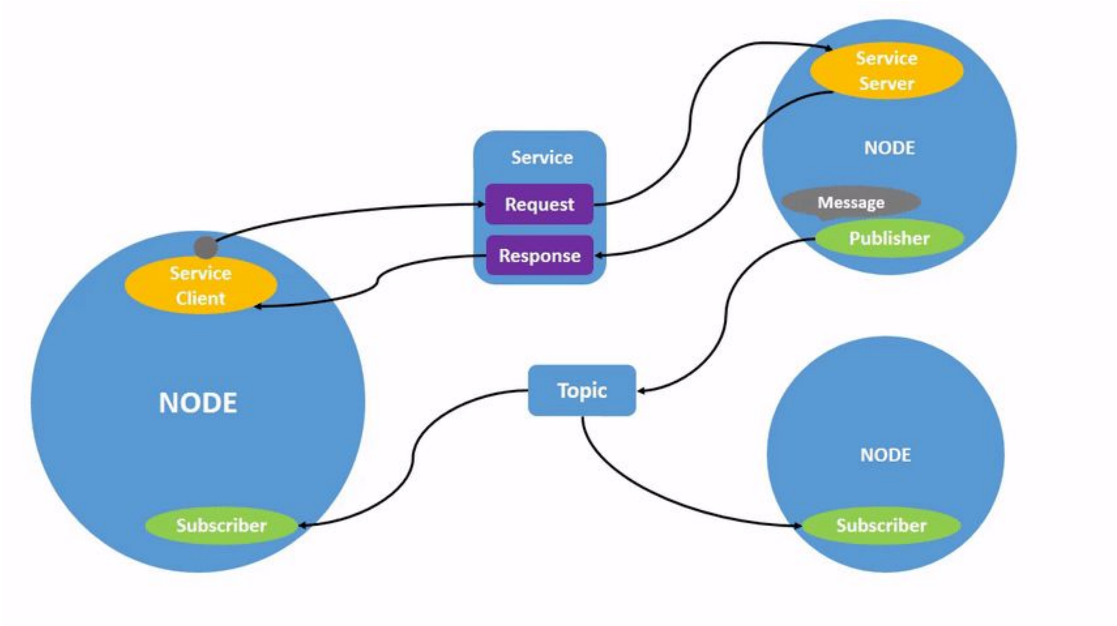
Subscribe to topic: **connect**

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

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

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

Subscribe to topic: **subscribe**

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

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

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

ROS communication model



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

Demo

robot backends: ros intro

interconnecting nodes

planning with moveit

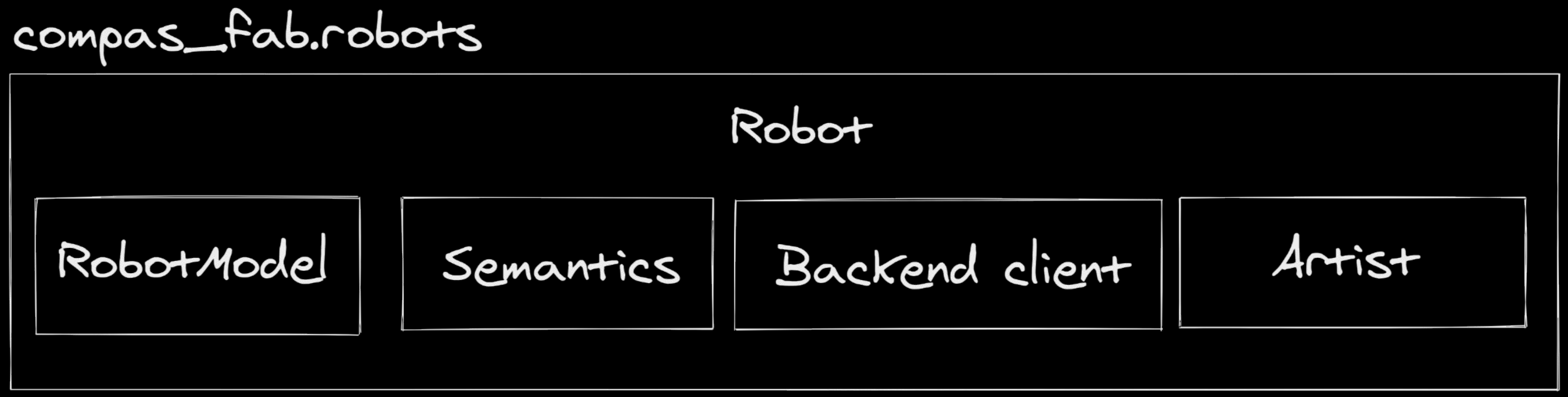
docker/moveit

`docker-compose up -d`

Right-click → Compose Up

Loading ROS robot

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


compas_fab.robots
Robot
RobotModel
Semantics
Backend client
Artist

`compas.robots.RobotModel`

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

`compas_fab.robots.Robot`

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

Planning

MoveIt!

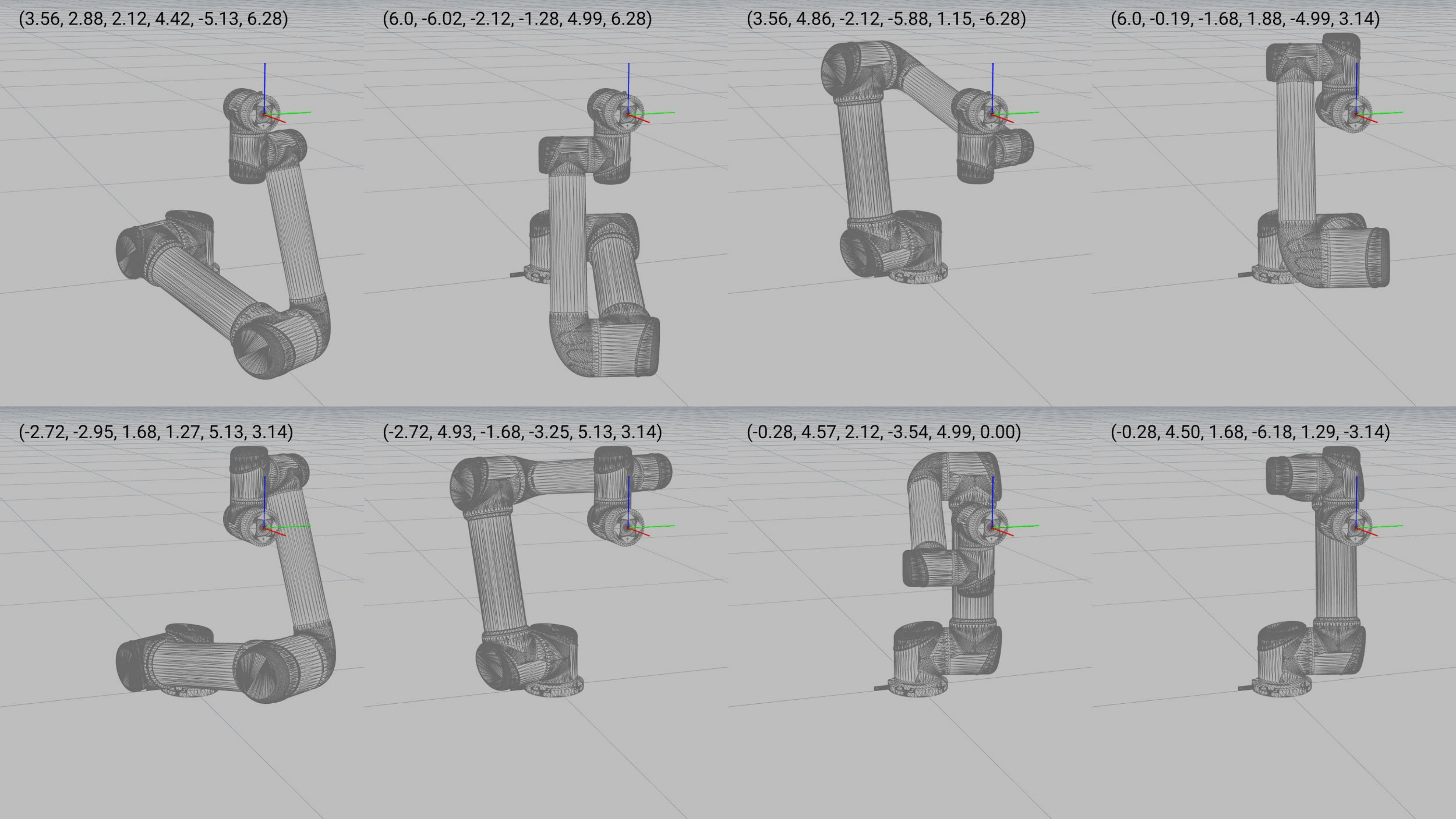
Kinematics

Motion planning

Planning scene

http://localhost:8080/vnc.html?resize=scale&autoconnect=true

http://192.168.99.100:8080/vnc.html?resize=scale&autoconnect=true


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

Forward Kinematics

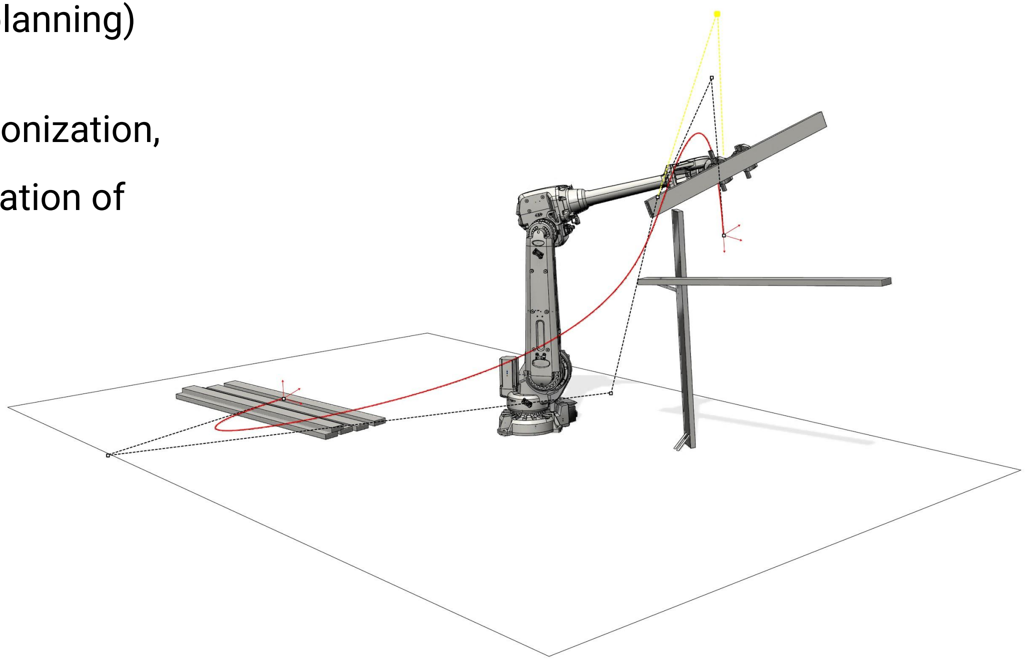
```
from compas_fab.backends import RosClient
from compas_fab.robots import Configuration

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_WCF = 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.zero_configuration()

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

Motion planning

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



Motion planning

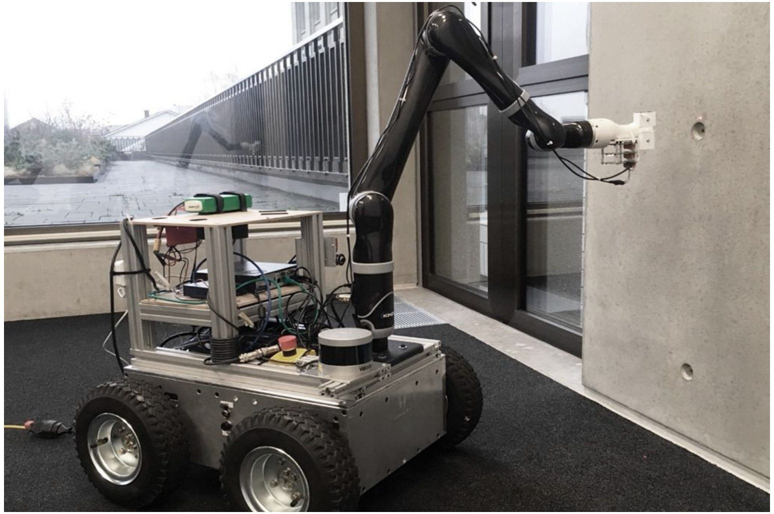
Collision checks

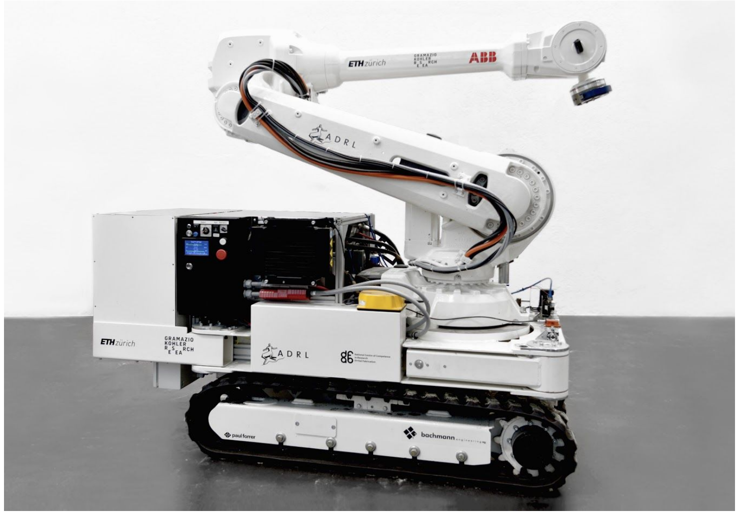
- Intricate positions (spatial assembly)
- Multiple robots working closely

Motion planning

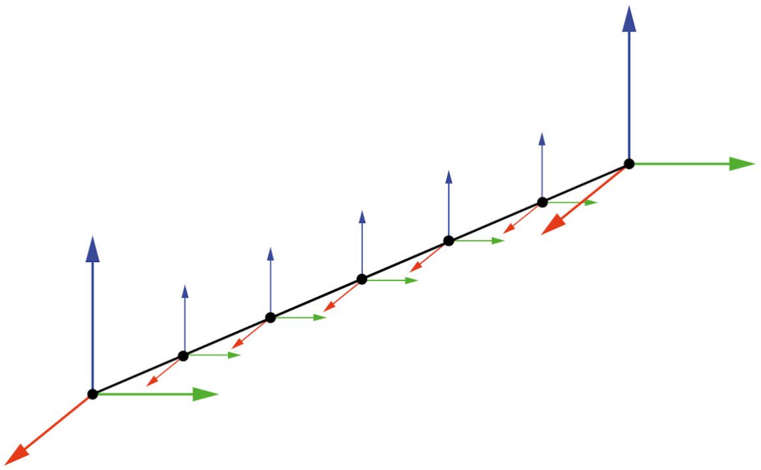
Trajectory checks

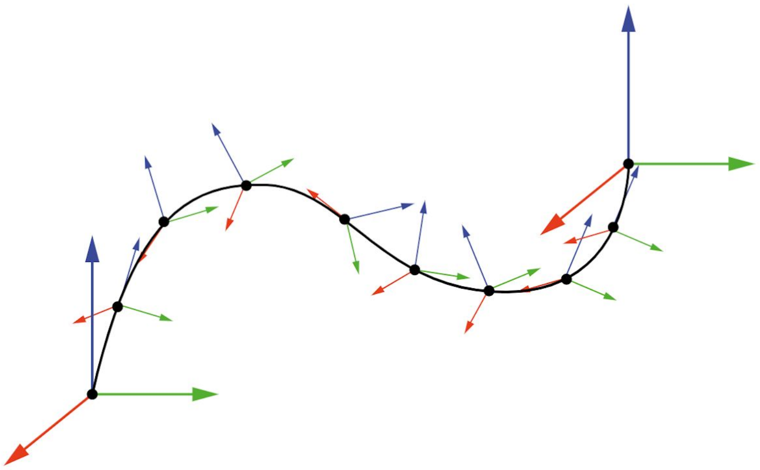
- Synchronization
- Continuous processes





Cartesian motion vs free-space motion





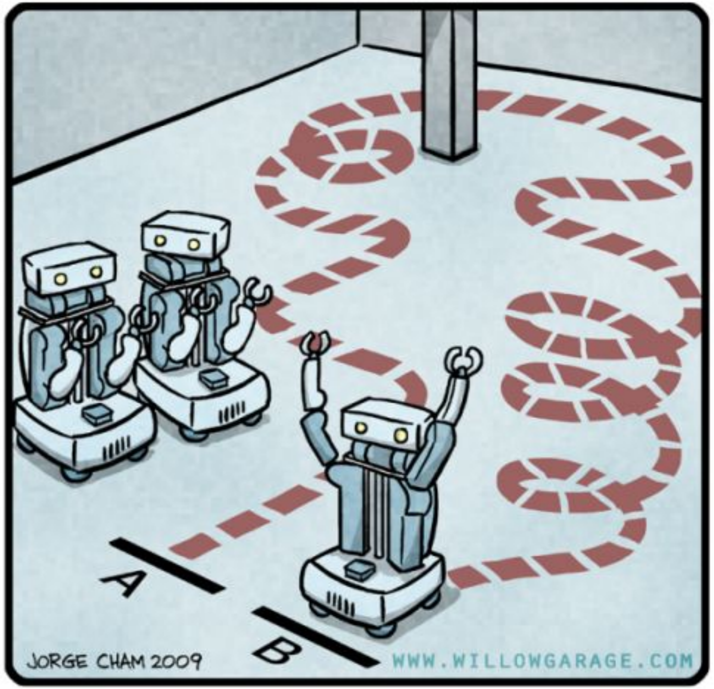
Plan cartesian motion

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

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

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

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

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

Plan motion

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

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

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

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

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

R.O.B.O.T. Comics



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

Defining constraints

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

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

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

Constraints

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

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

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

Planning scene operations

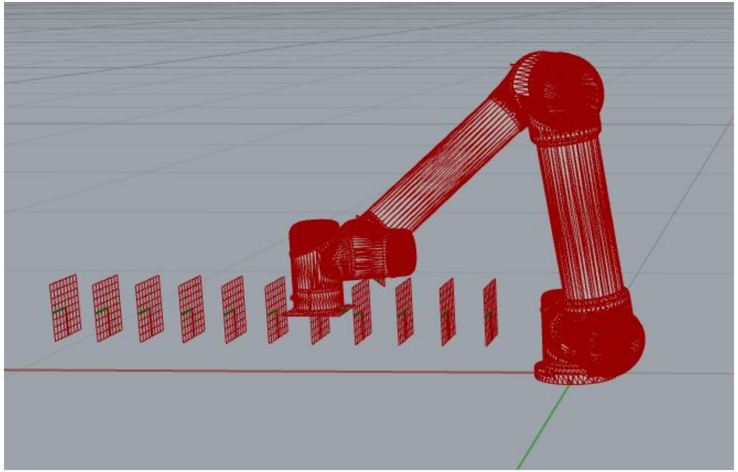
Add/append/remove collision meshes (i.e. obstacles) and add/remove attached collision meshes (i.e. meshes attached to the end effector).

- **Usage:**

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

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

Assignment

1. Setup the MoveIt container for a UR5 on your laptop
2. Use the `RosClient` to load the robot
3. Take as input a list of brick positions as list of frames and use `inverse_kinematics` function to calculate viable configurations for each frame
4. Store the results in a JSON file



docker/ur5-planner

Next week

- Assignment submission due: Wed 24th March, 9AM.
- Ask for help if needed: Slack, Forum, Office Hours (Fridays, request via Slack)
- Next week:
  - Path planning using MoveIt
  - Customizing end effectors
  - Pick & Place process for discrete assemblies

Thanks!