This lecture will be recorded

COMPAS

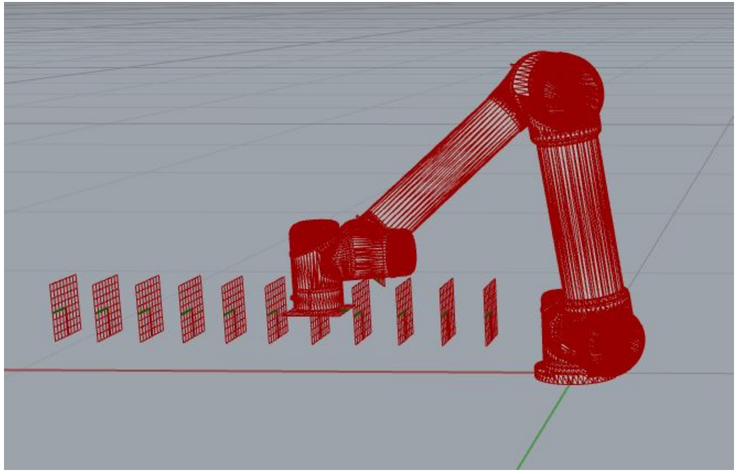
064-0026-00L: COMPAS II

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

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

Review of last lecture 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 and commit the file in your submission



TODAY

cartesian and kinematic planning
planning scene
end effectors & pick and place

Today’s goal

Understand the **method** to plan a **pick and place process**

cartesian and kinematic planning

planning scene

end effectors & pick and place

`docker-compose up -d`

Right-click → Compose Up

`docker/ur5-planner`

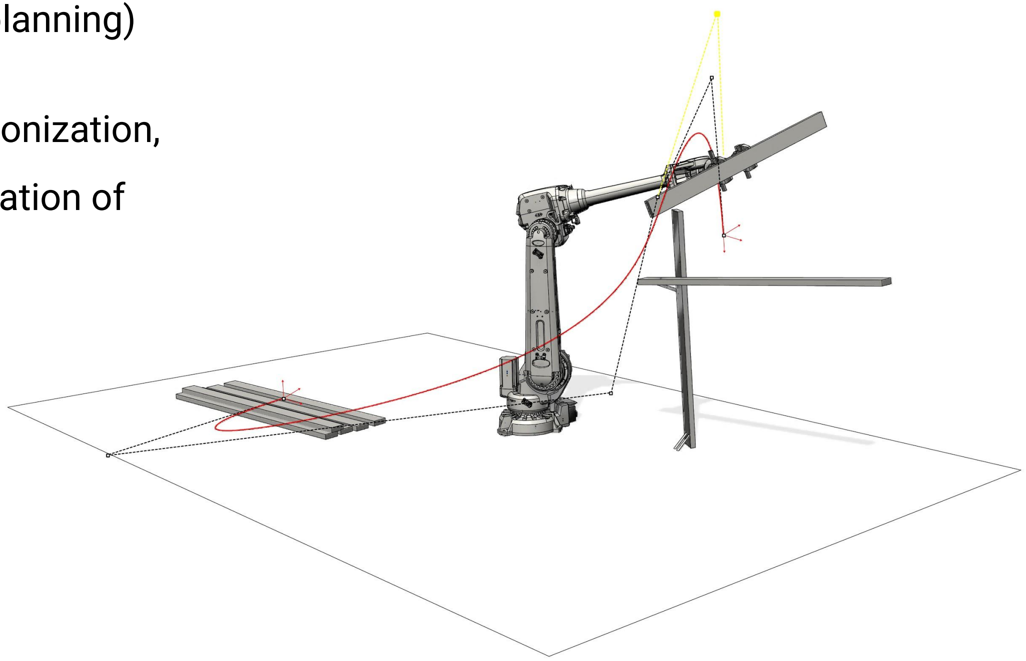
Lightweight MoveIt UR5 planner without any user interface

`docker/moveit`

MoveIt UR5 planner with user interface via browser (noVNC) http://localhost:8080/vnc.html?resize=scale&autoconnect=true

Motion planning

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



Motion planning

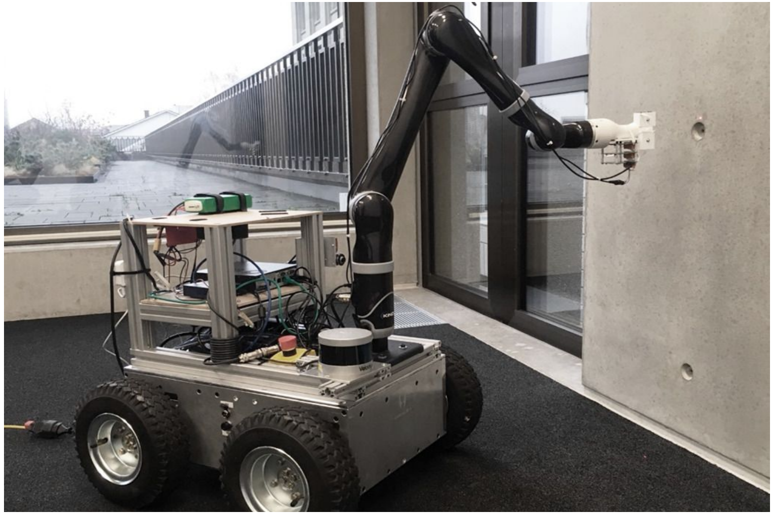
Collision checks

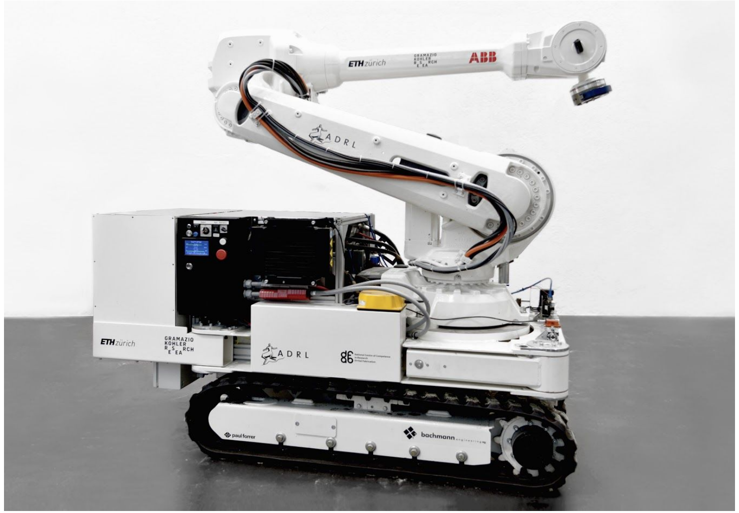
- Intricate positions (spatial assembly)
- Multiple robots working closely

Motion planning

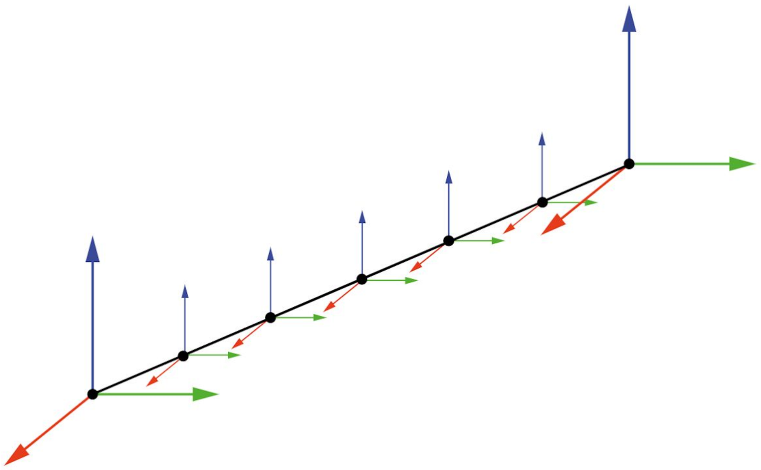
Trajectory checks

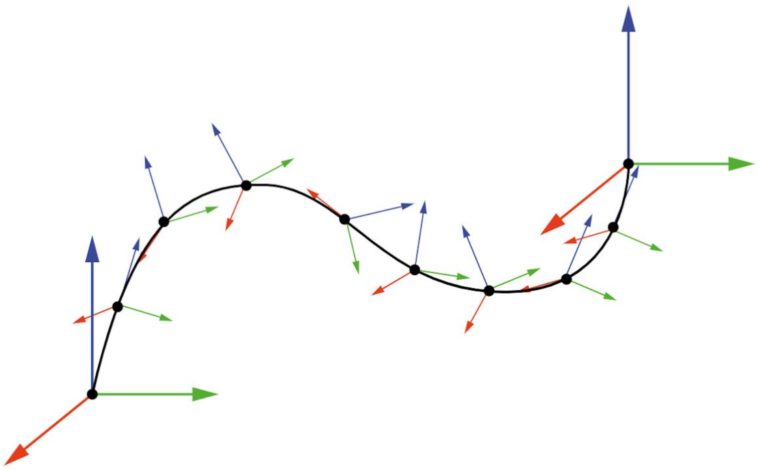
- Synchronization
- Continuous processes





Cartesian motion vs free-space motion





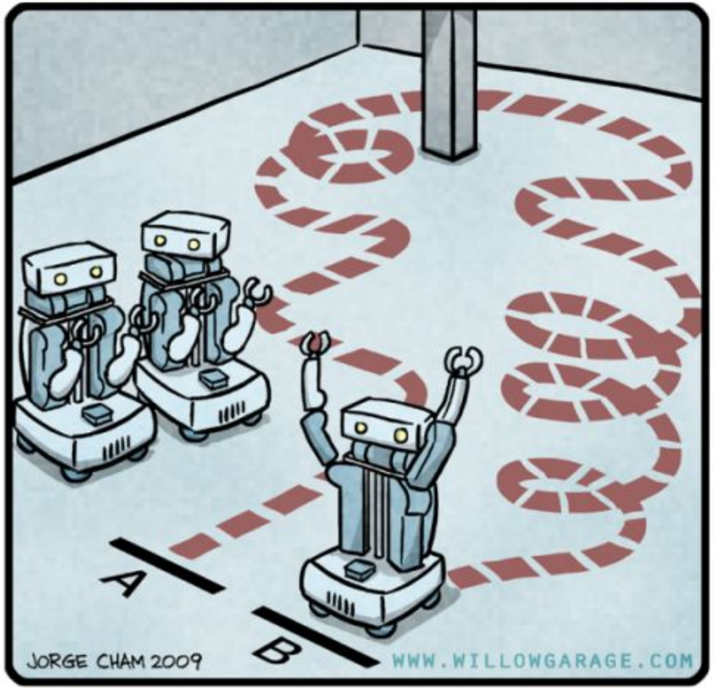
Plan cartesian motion

```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)
```

cartesian and kinematic planning

planning scene

end effectors & build elements

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:**

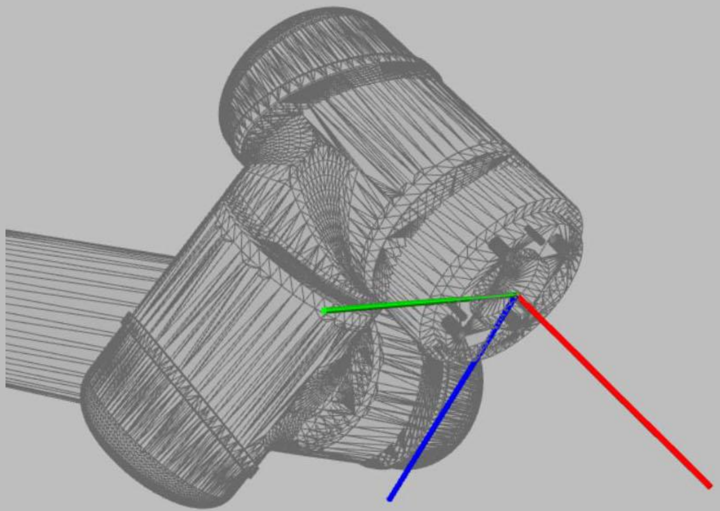
```
scene = compas_fab.robots.PlanningScene(robot)
scene.add_collision_mesh(..)
scene.remove_collision_mesh(..)
scene.append_collision_mesh(..)

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

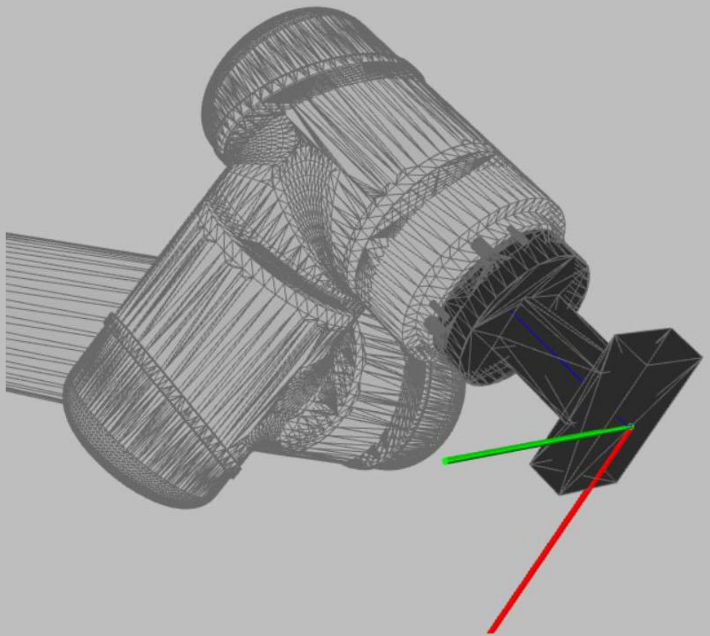
cartesian and kinematic planning

planning scene

end effectors & pick and place

Attach **tool**





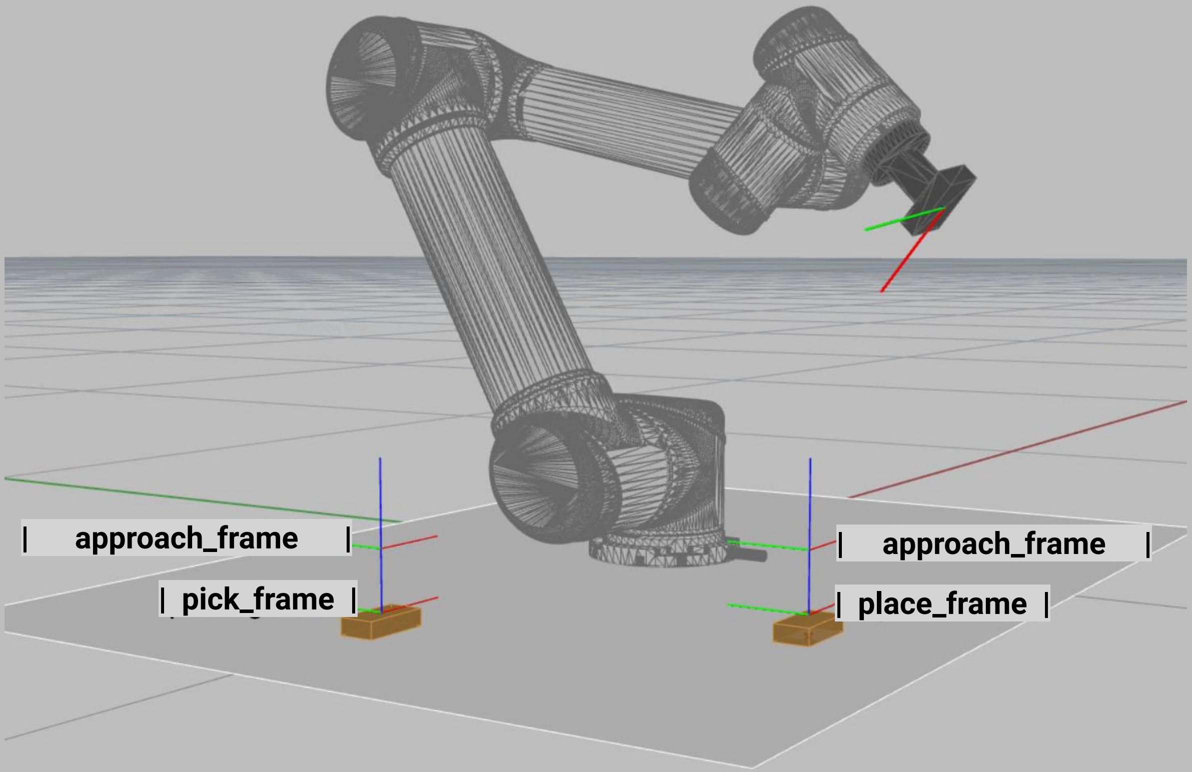
Attach tools

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

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

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

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

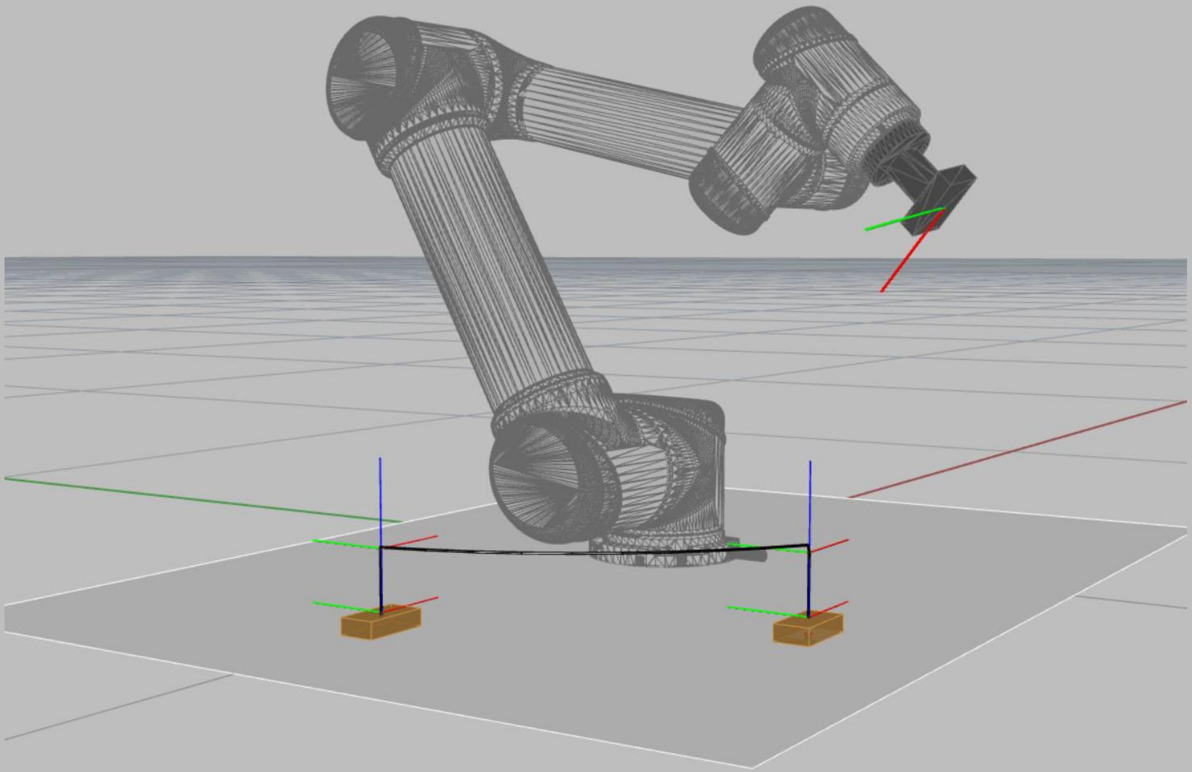
Pick and **Place**

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

Pick and **Place**



Pick and **Place**



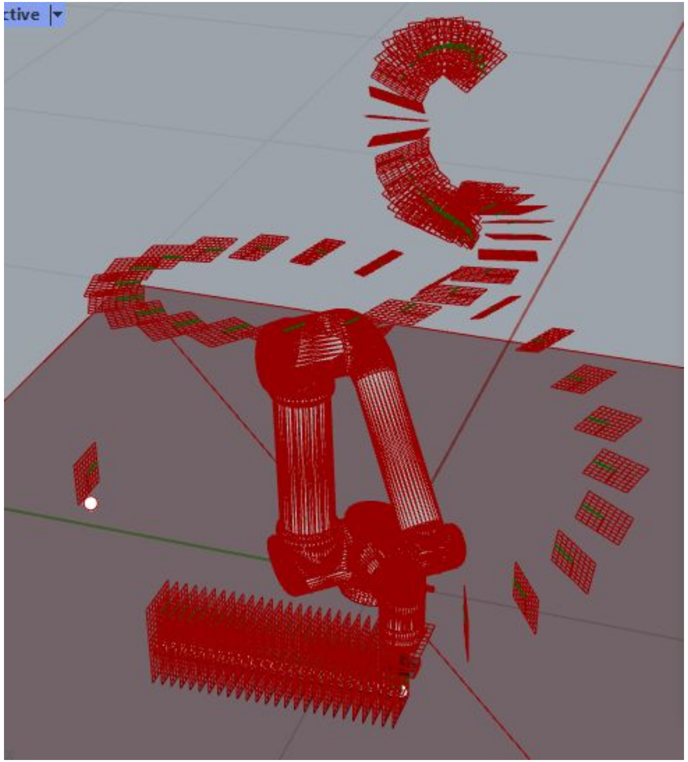
Hands-on session

Assignment

1. Continuation from hands-on exercise
2. Based on the work done during lecture on the example `15_pick_and_place.ghx` explore different sequences of place frames
3. Store all the trajectories (pick+move+place) for at least 8 elements in a JSON file, use `compas.json_dump` to keep data type information

More about serialization:

https://compas.dev/compas/latest/tutorial/serialization.html



Next week

- Assignment submission due: Wed 14th April, 9AM.
- Ask for help if needed: Slack, Forum, Office Hours (Fridays, request via Slack)
- Next lecture in **two weeks**:
    - Assembly of discrete elements
    - Modelling assemblies as networks (DAGs)
    - Pick & Place process for assemblies

Thanks!