This lecture will be recorded

COMPAS

064-0026-00L: COMPAS II

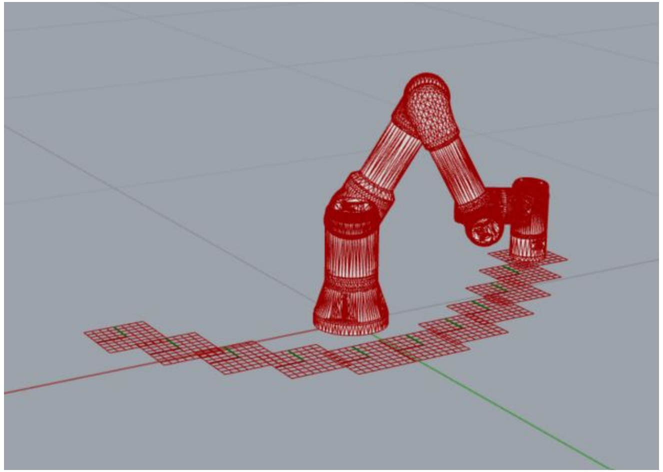
Introduction to Computational Methods for Digital Fabrication in Architecture slides + code

`https://dfab.link/fs2022`

```
(base)  conda env update -f https://dfab.link/fs2022.yml
```

Review of last lecture assignment

1. Start the MoveIt container for a UR3e
2. Use the `RosClient` to load the robot
3. Taking a robot and a list of frames as parameter, calculate a feasible configuration for each of the frames
4. Try to find an optimal `start_configuration` for each so that the motion from one config to the next is minimized
5. Store all found configurations in a JSON file using `compas.json_dump` or `compas.json_dumps`
6. Commit **BOTH** the Python and JSON files



TODAY

planning scene

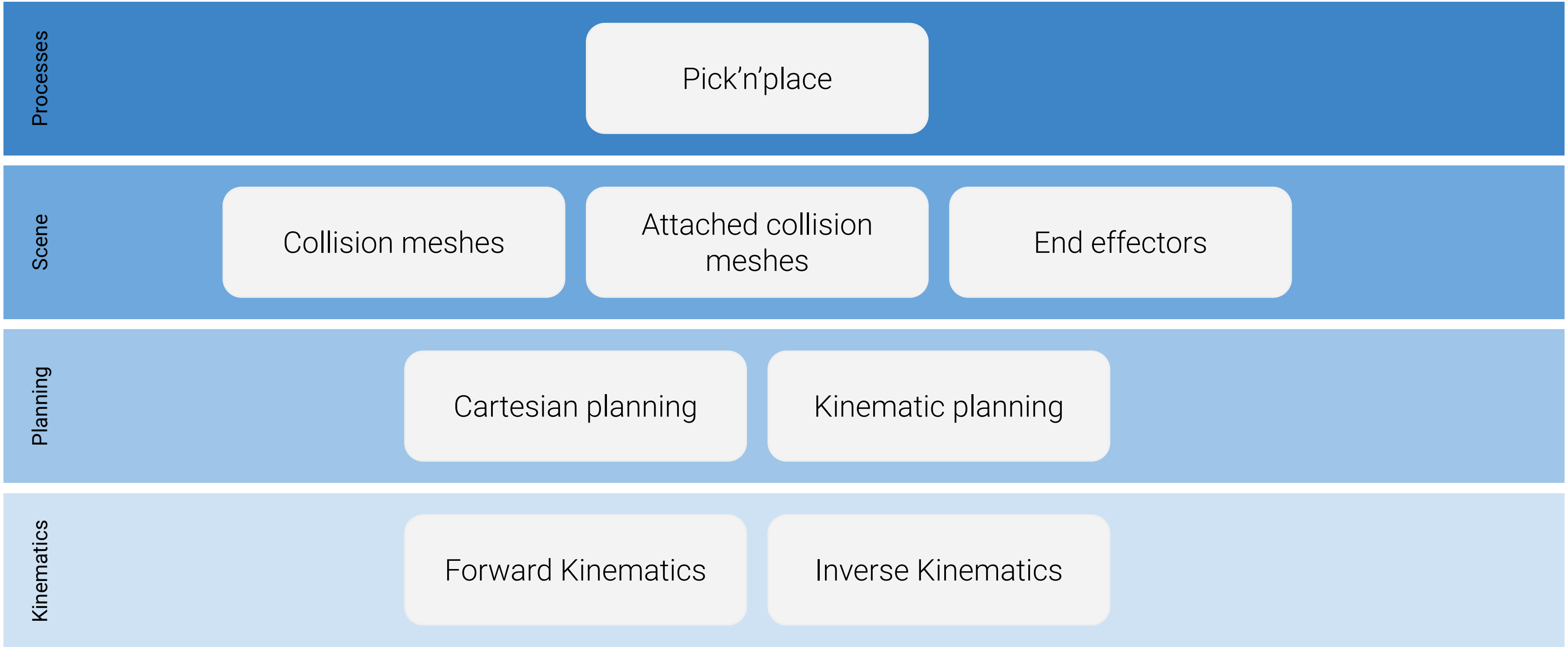
motion planning

pick and place

Understand the **method** to plan a **pick and place process** docker/moveit_noetic

`docker-compose up -d`

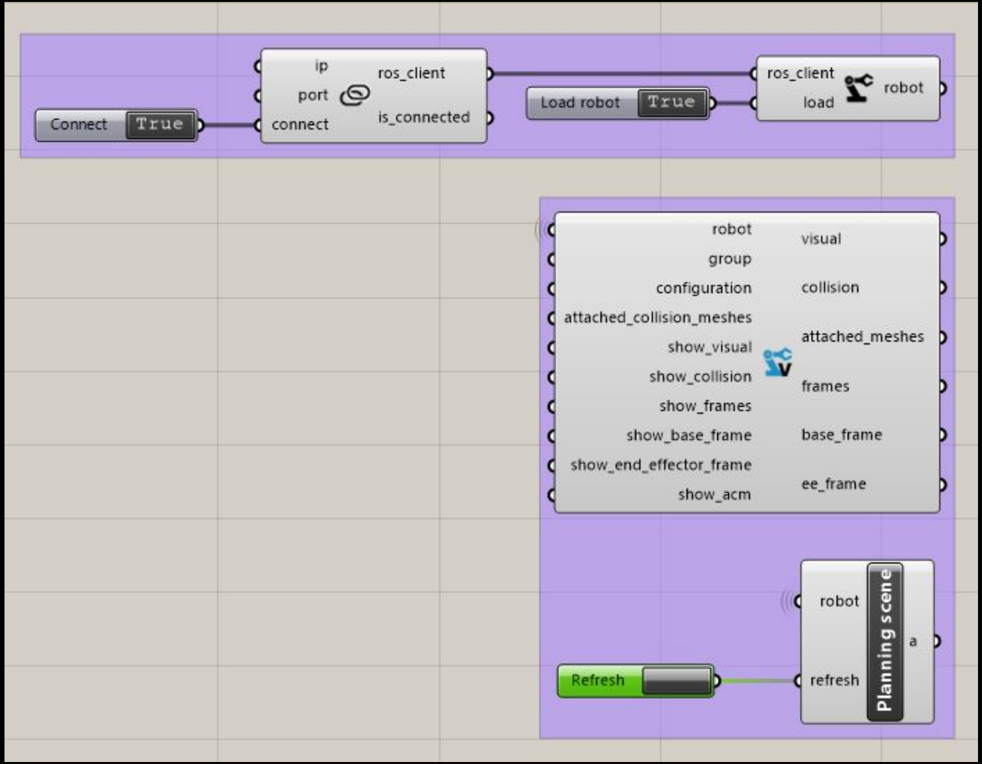
Right-click → Compose Up



Building blocks

planning scene

Load planning scene



Add collision meshes

```
scene = PlanningScene(robot)

box = Box.from_diagonal([(-0.7, -0.7, 0), (.7, .7, -0.02)])
mesh = Mesh.from_shape(box)
cm = CollisionMesh(mesh, "floor")
scene.add_collision_mesh(cm)
```

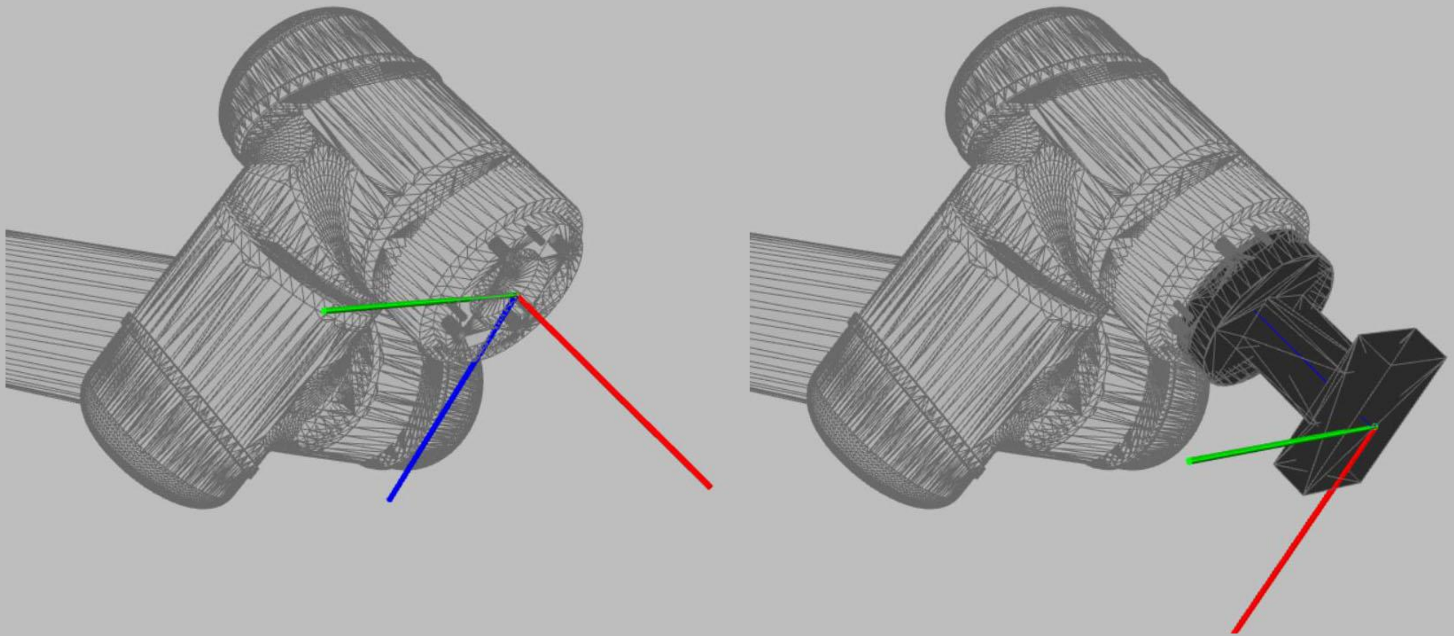
Append collision meshes

```
scene = PlanningScene(robot)

brick = Box.from_diagonal([(-0.006, -0.015, 0), (.006, .015, 0.012)])

for i in range(5):
    mesh = Mesh.from_shape(brick)
    cm = CollisionMesh(mesh, "brick_wall")
    cm.frame.point.y = 0.5
    cm.frame.point.z = brick.zsize * i

    scene.append_collision_mesh(cm)
```

Remove collision meshes

```
scene = PlanningScene(robot)

scene.remove_collision_mesh("brick_wall")
scene.remove_collision_mesh("floor")
```

Attach **tool**



Attach tools

```
# create tool from mesh and frame
mesh = Mesh.from_stl(os.path.join(HERE, "vacuum_gripper.stl"))
tool = Tool(mesh, Frame([0, 0, 0.07], [1, 0, 0], [0, 1, 0]),
link_name="wrist_3_link")

# Attach the tool
robot.attach_tool(tool)

# now we can convert frames at robot's tool tip and flange
frames_tcf = [Frame((-0.3, 0.0, -0.2), (0.2, 0.9, -0.2), (0.8, -0.1, 0.4))]
frames_t0cf = robot.from_tcf_to_t0cf(frames_tcf)
```

Detach tools

```
# Attach the tool
robot.attach_tool(tool)

# Do something useful with the tool...

# Remove the tool
robot.detach_tool()
```

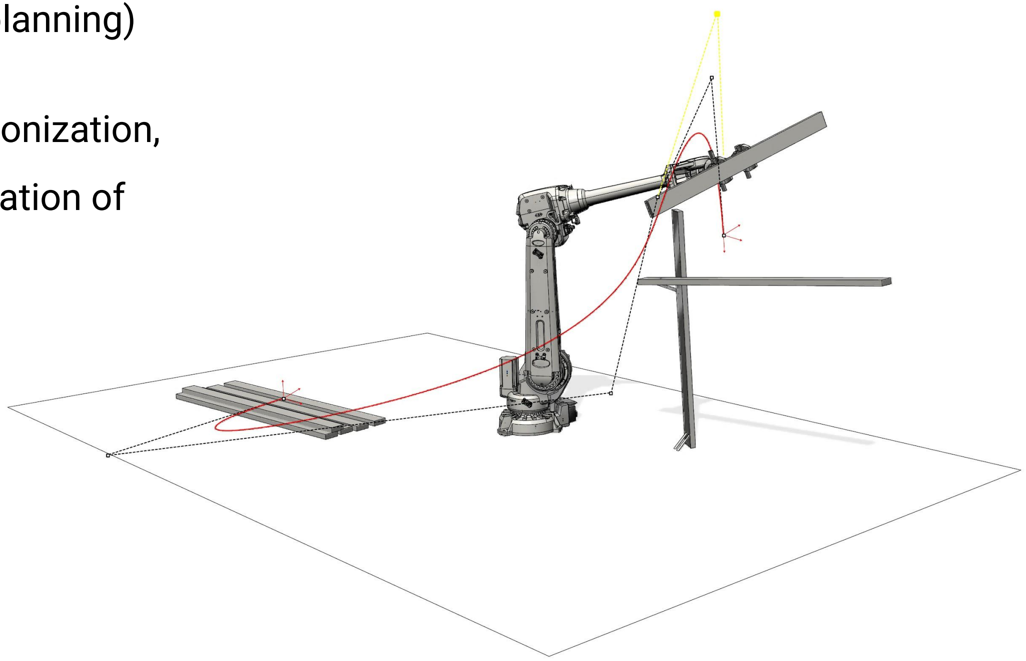
Planning with end-effectors

```
# Attach the tool
robot.attach_tool(tool)

# Adjust frames based on attached tool
frames = robot.from_tcf_to_t0cf(frames)

# Plan as usual
trajectory = robot.plan_cartesian_motion(frames, start_configuration)
```

path planning

Path vs Motion planning

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



Path planning

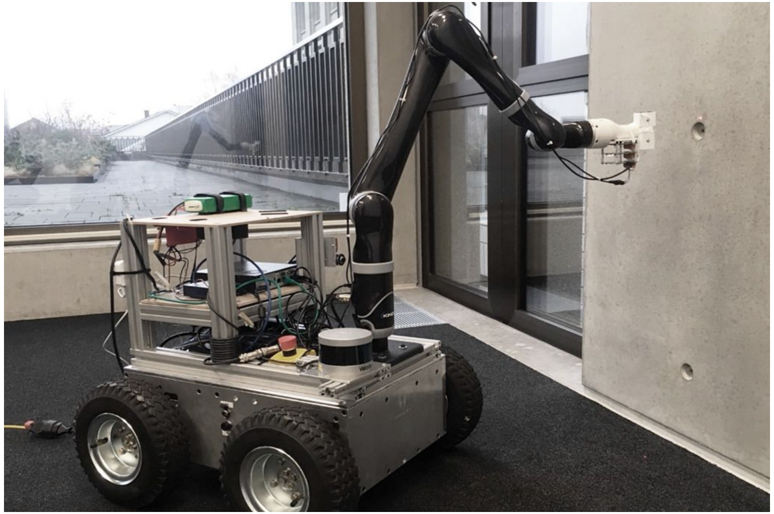
Collision checks

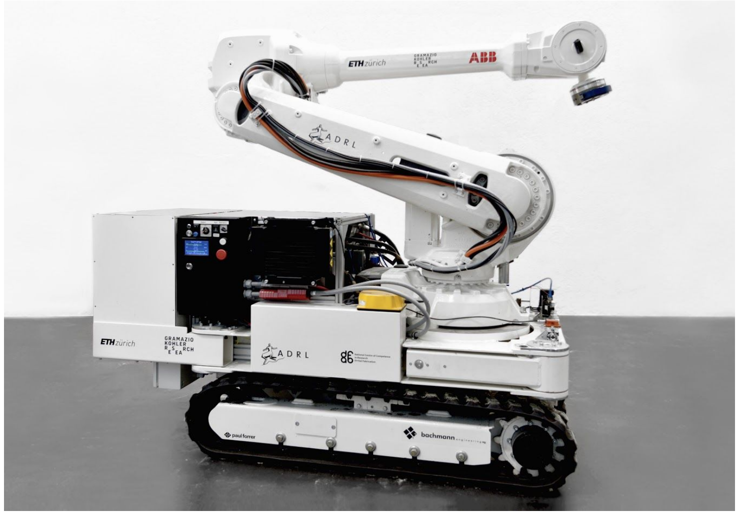
- Intricate positions (spatial assembly)
- Multiple robots working closely

Motion planning

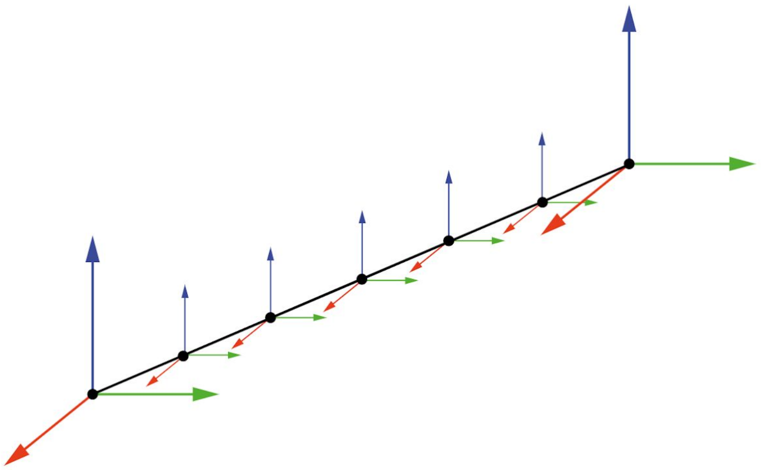
Trajectory checks

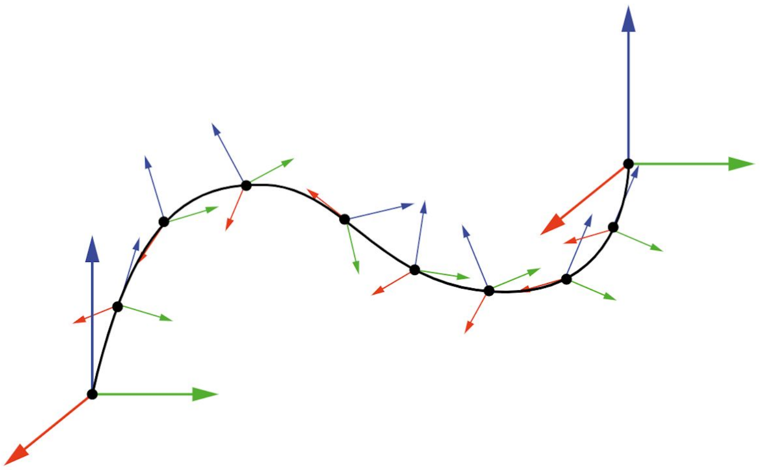
- Synchronization
- Continuous processes





Cartesian motion vs free-space motion





Plan cartesian motion

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

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

    frames = []
    frames.append(Frame((0.3, 0.1, 0.05), (-1, 0, 0), (0, 1, 0)))
    frames.append(Frame((0.4, 0.3, 0.05), (-1, 0, 0), (0, 1, 0)))

    start_configuration = robot.zero_configuration()
    start_configuration.joint_values = (-0.106, 5.351, 2.231, -2.869, 4.712, 1.465)

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

Plan motion

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

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

    frame = Frame((0.4, 0.3, 0.05), (-1, 0, 0), (0, 1, 0))

    start_configuration = robot.zero_configuration()
    start_configuration.joint_values = (-0.106, 5.351, 2.231, -2.869, 4.712, 1.465)

    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)
```

Constraints

```
frame = Frame((0.4, 0.3, 0.05), (-1, 0, 0), (0, 1, 0))
tolerance_position = 0.001
tolerance_axes = [math.radians(1)] * 3

config = robot.zero_configuration()
config.joint_values = (-0.106, 5.351, 2.231, -2.869, 4.712, 1.465)
tol_above = [0.1] * 6
tol_below = [0.1] * 6

# create goal constraints from frame
goal_constraints = robot.constraints_from_frame(frame, tolerance_position, tolerance_axes)

# create goal constraints from frame
goal_constraints = robot.constraints_from_configuration(config, tol_above, tol_below)
```

Available planners

Pipelines

- Open Motion Planning Library (OMPL)
  - Primary/default pipeline of stochastic planners
  - Available planners: http://ompl.kavrakilab.org/planners.html
- Pilz Industrial Motion Planner
  - Deterministic pipeline for industrial arm motion planning.
  - Available planners: PTP, LIN, CIRC
- Others (CHOMP, STOMP, SBPL)
  - MoveIt integration is partial in most cases

⛔ *The cartesian planner is not a planner*

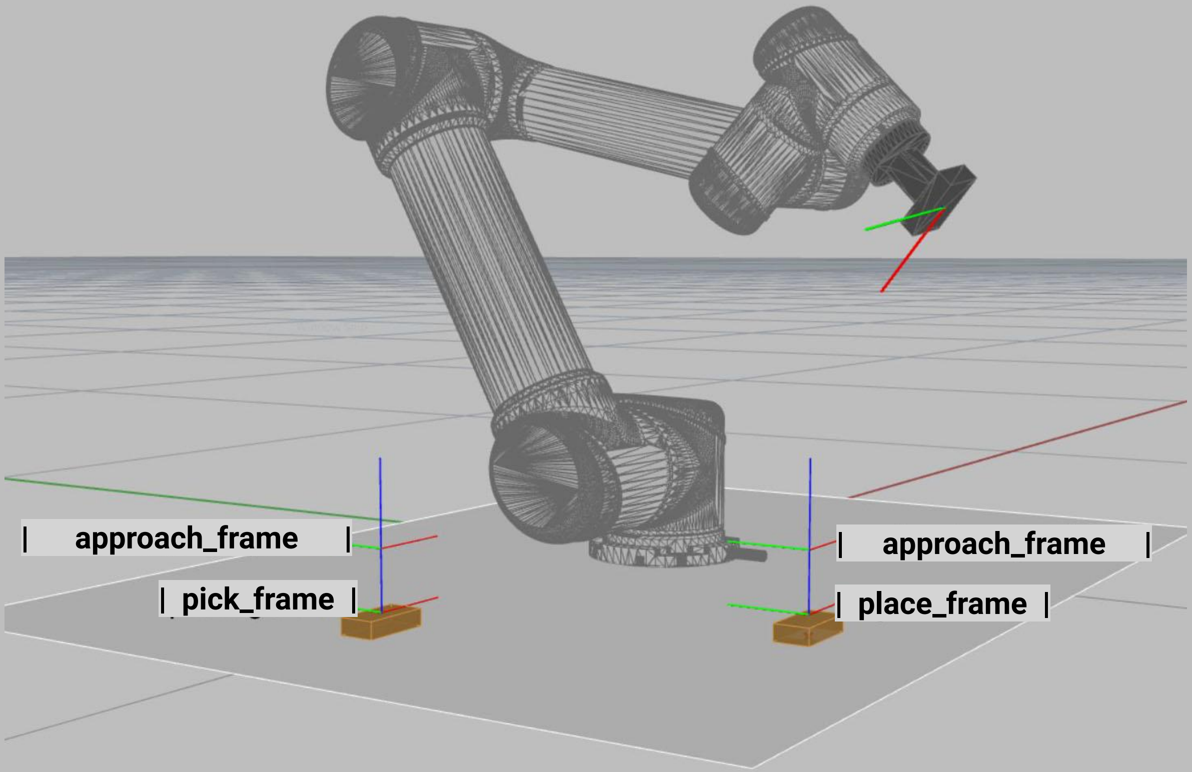
exercise

pick and place

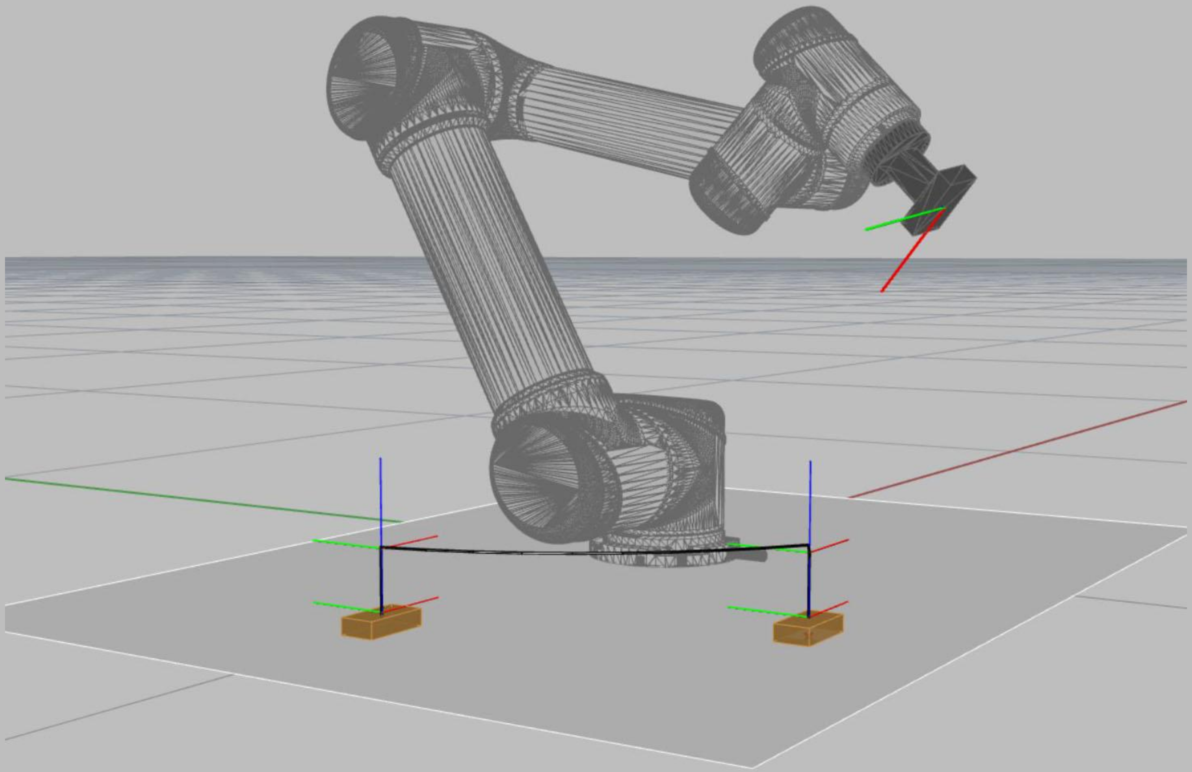
Pick and Place

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

Pick and **Place**

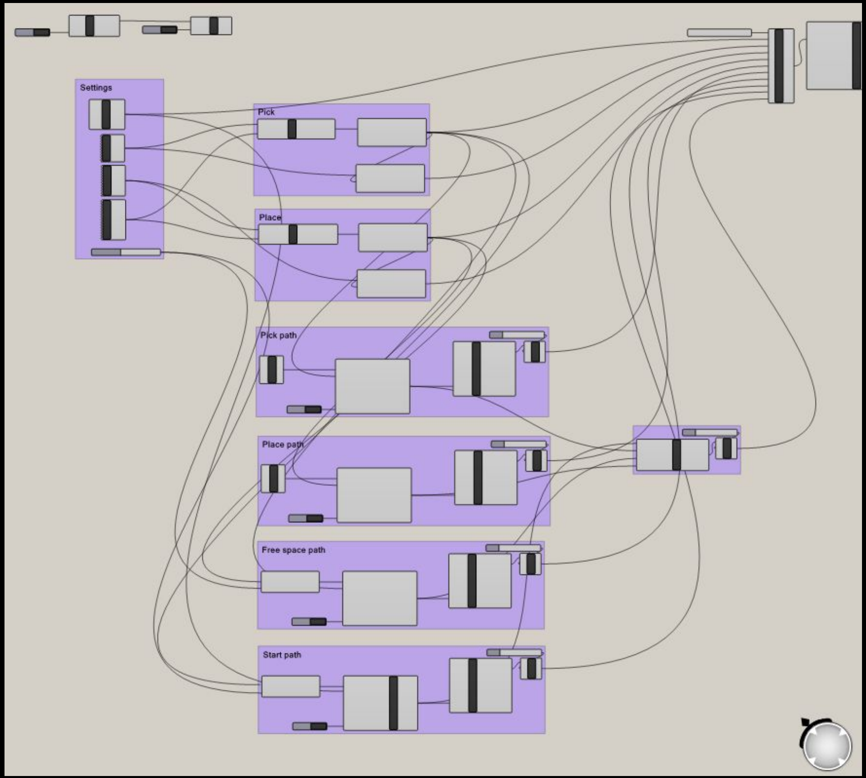


Pick and **Place**



Pick and Place

```python
def get_tool():
    current_folder = os.path.dirname(__file__)
    mesh = Mesh.from_stl(os.path.join(current_folder,
"vacuum_gripper.stl"))
    frame = Frame([0, 0, 0.07], [1, 0, 0], [0, 1, 0])
    tool = Tool(mesh, frame, link_name="wrist_3_link")
    return tool


def get_approach_vector(n):
    return (0, 0, 0.05)


def get_pick_frame(n):
    return Frame((0.3, 0.1, 0.05), (-1, 0, 0), (0, 1, 0))


def get_place_frame(n):
    return Frame((0.4, 0.3, 0.05), (-1, 0, 0), (0, 1, 0))

...
```



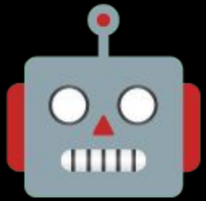
Preparation for next lecture

- No coding assignment, but:
- **Use and experiment** with example 420:
  - To search paths for more than one part
  - To add the current part as attached collision mesh
- **Pair and Answer** the following two questions:
  - What was the most important thing you learned today?
  - What important question(s) remain(s) unanswered?

Next week

- Have the answers to the two questions at hand, we'll discuss them in a round.
- Ask for help if needed: Slack, Forum, Office Hours (Fridays, request via Slack)
- Next lecture:
  - Assembly of discrete elements
  - Modelling assemblies as networks (DAGs)
  - Pick & Place process for assemblies

Thanks!



ETH zürich