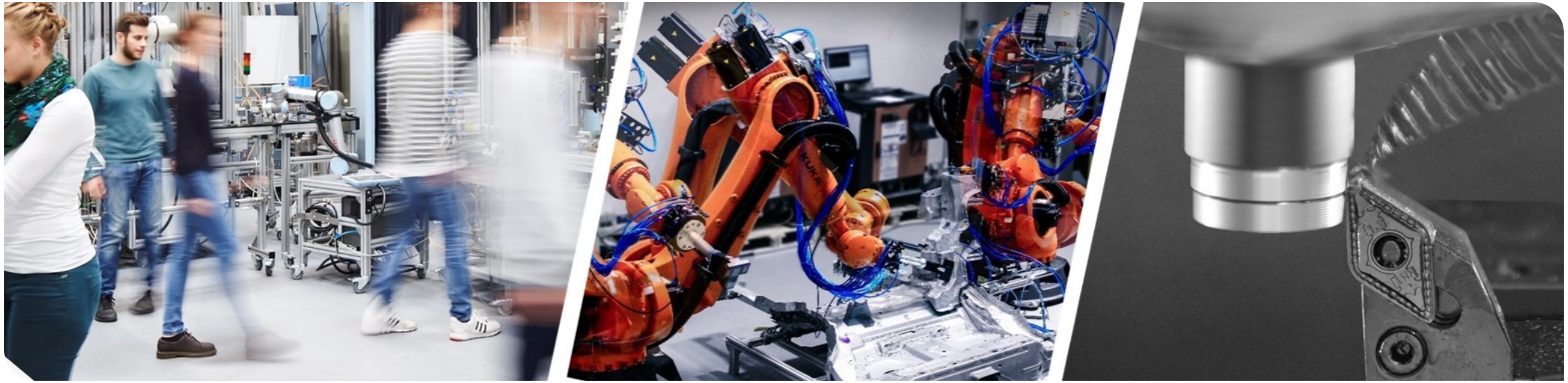
Karlsruher Institut für Technologie

Institut für Produktionstechnik

Pybullet Industrial: Roboter Simulationen alleine reichen nicht!



Unterschiede zwischen klassischer Robotik und Produktionstechnischer Robotik

Pybullet Industrial

Roboter in der Klassischen Robotik Forschung

Roboter in der Produktionstechnik Forschung

https://www.mybotshop.de/Franka-Emika-Franka-Research-3

Toolchain Gap

Pybullet Industrial

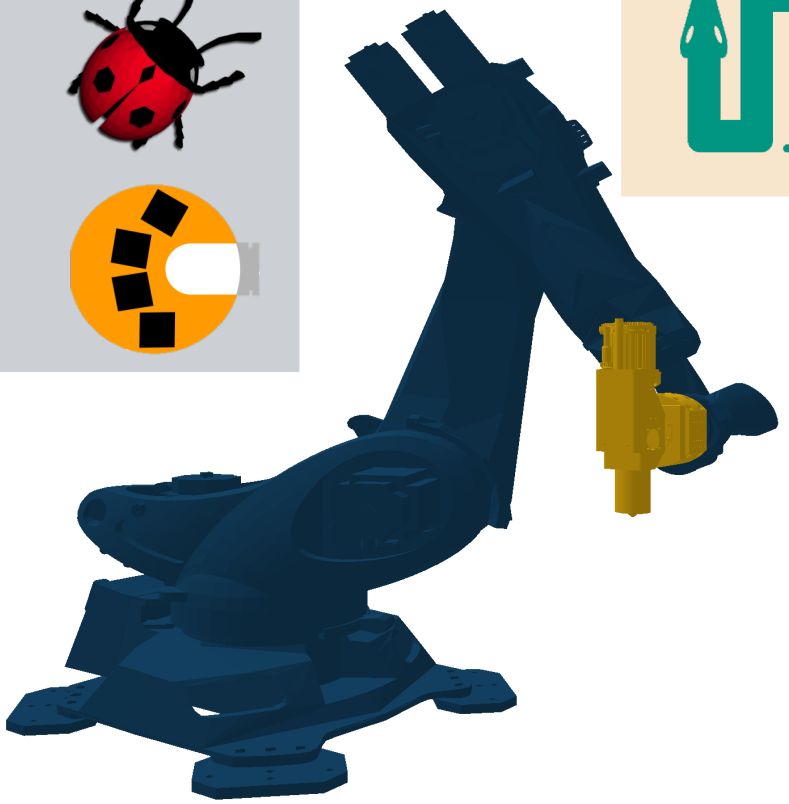
Roboter Mehrkörpersimulation

GAZEBO

NVIDIA ISAAC

End Effektor Prozesssimulation

Pybullet Industrial



Unterstütze Prozesse

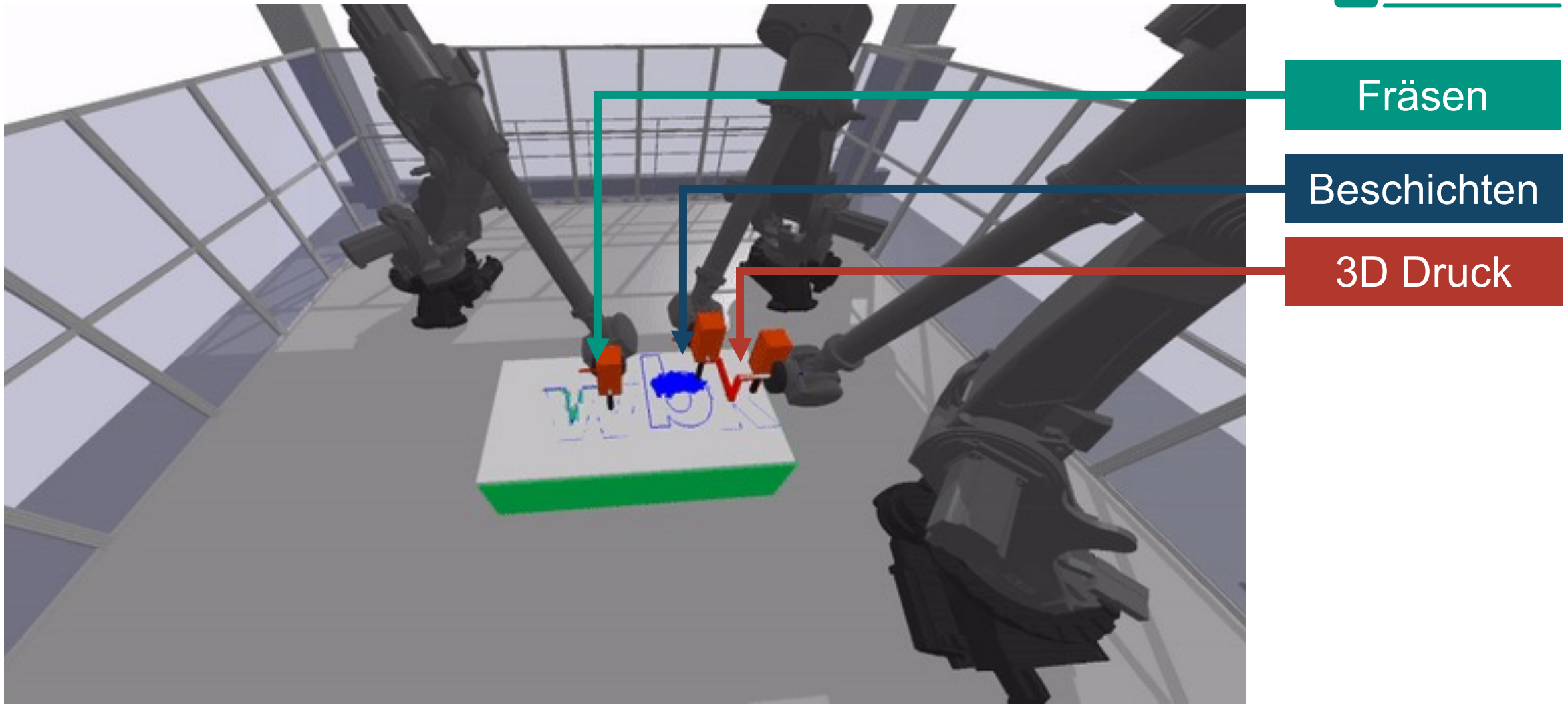
Material hinzufügen
- Schweißen
- Kleben
- 3D-Drucken
- Beschichten
- Etc...

Material entfernen
- Fräsen
- Bohren
- Schneiden
- Etc ...

Material bewegen
- Greifen
- Halten
- Transportieren
- Etc...

Beispielprozesse

Pybullet Industrial



How it Works

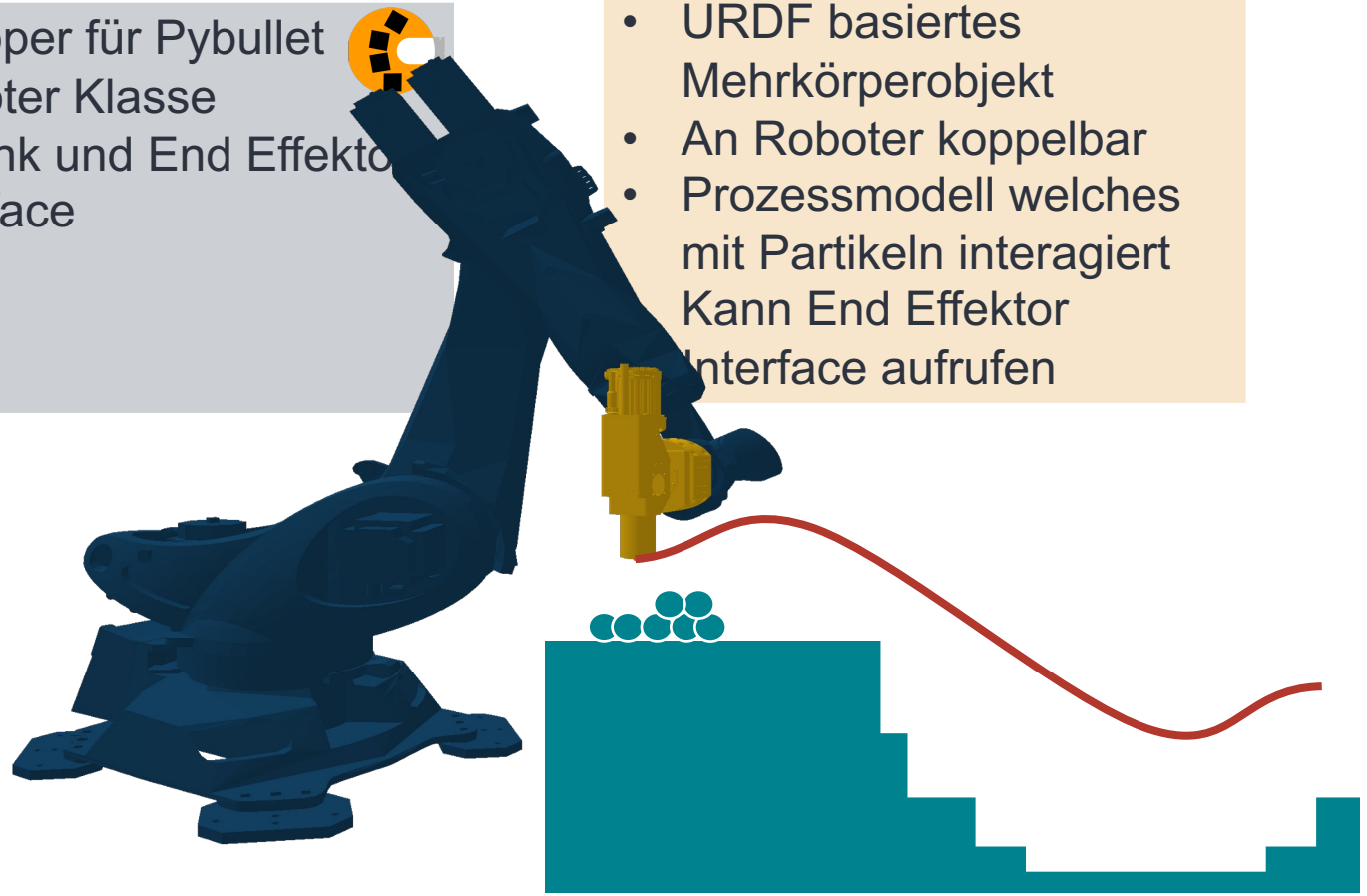
Pybullet Industrial

RobotBase

- Wrapper für Pybullet Roboter Klasse
- Gelenk und End Effekto Interface

EndeffectorTool

- URDF basiertes Mehrkörperobjekt
- An Roboter koppelbar
- Prozessmodell welches mit Partikeln interagiert Kann End Effektor Interface aufrufen



Particle

- Ausgedehnte Körper mit definierbaren Physikalischen Eigenschaften

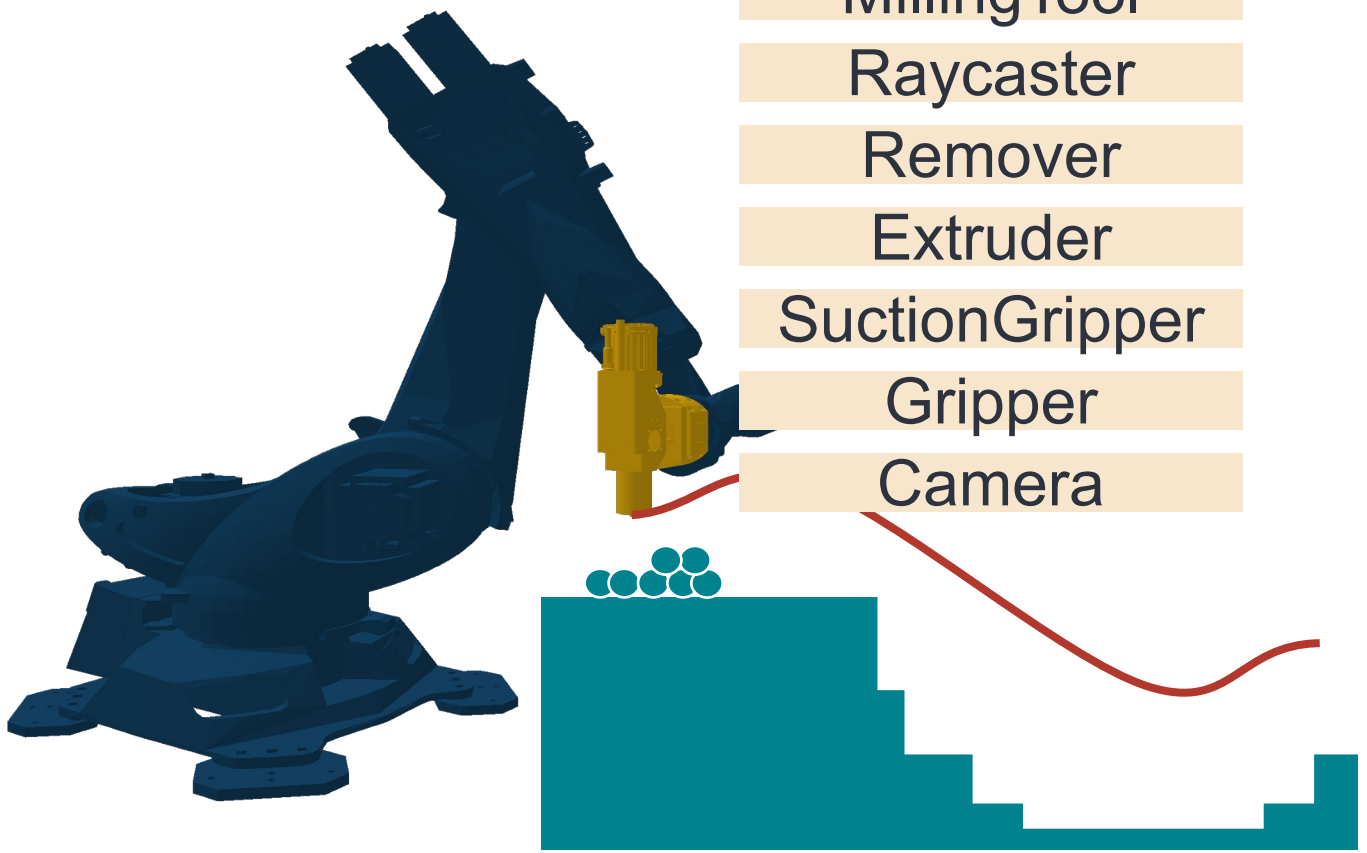
ToolPath

- Toolpfadbeschreibung hinsichtlich Position, Orientierung und Toolaktivierung

Klassenübersicht

Pybullet Industrial

RobotBase



EndeffectorTool

- MillingTool
- Raycaster
- Remover
- Extruder
- SuctionGripper
- Gripper
- Camera

Particle

- MetalVoxel
- Paint
- Plastic

ToolPath

Code Beispiel: Fräsen

Pybullet Industrial

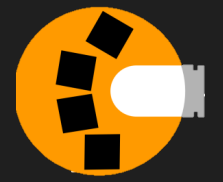
```python
import os
import numpy as np
import pybullet as p

import pybullet_industrial as pi

dirname = os.path.dirname(__file__)
parentDir = os.path.dirname(dirname)
urdf_file1 = os.path.join(parentDir, 'examples','robot_descriptions', 'milling_head.urdf')
urdf_file2 = os.path.join(dirname,'robot_descriptions', 'kuka_robot.urdf')
physics_client = p.connect(p.GUI, options='--background_color_red=1 ' +
                                          '--background_color_green=1 ' +
                                          '--background_color_blue=1')
p.setPhysicsEngineParameter(numSolverIterations=5000)
p.configureDebugVisualizer(p.COV_ENABLE_GUI, 0)

milling_properties = {'diameter': 0.1,'rotation speed': 2*np.pi/5,'number of teeth': 5,'height': 0.1,'number of rays': 1}
robot = pi.RobotBase(urdf_file2, [0, 0, 0], [0, 0, 0, 1])
milling_tool = pi.MillingTool(urdf_file1, [0,0,0], [0, 0, 0, 1], milling_properties)
milling_tool.couple(robot, 'link6')
```

Pybullet Aufsetzten



Roboter und End Effektor Deklarieren

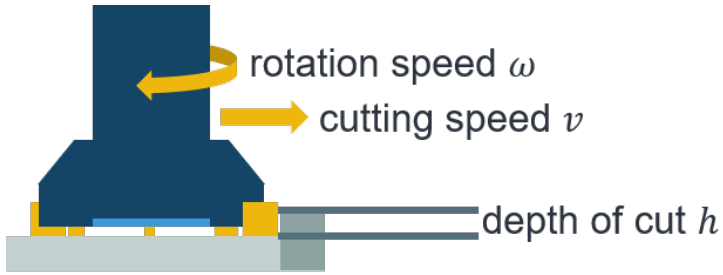
```
24
25  target_orientation = p.getQuaternionFromEuler([0, 0, 0])
26  test_path = pi.build_box_path([1.9, 0.0, 0.53], [0.5, 0.6], 0.1, [0, 0, 0, 1], 100)
27
28  for _ in range(20):
29      milling_tool.set_tool_pose(*test_path.get_start_pose())
30      for _ in range(50):
31          p.stepSimulation()
32
33  p.configureDebugVisualizer(p.COV_ENABLE_RENDERING, 0)
34  pi.spawn_material_block([1.9, -0.5, 0.53],[1, 1, 1/20],pi.MetalVoxel,{'particle size': 1/20})
35  p.configureDebugVisualizer(p.COV_ENABLE_RENDERING, 1)
36
37  for positions, orientations, tool_path in test_path:
38      milling_tool.set_tool_pose(positions, orientations)
39      milling_tool.mill()
40      for _ in range(30):
41          p.stepSimulation()
42
```

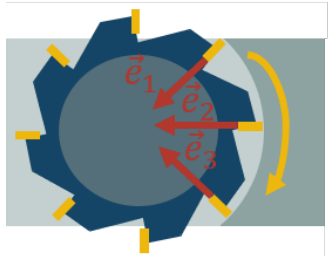
Toolpfad Definieren

Partikel Laden

Werkzeug Bewegen und Fräsen

Code Beispiel: Fräsen





Material specific force F_m
Chip thickness exponent d_m
Force direction $\vec{e}_i$
Number of teeth t

Total cutting force:

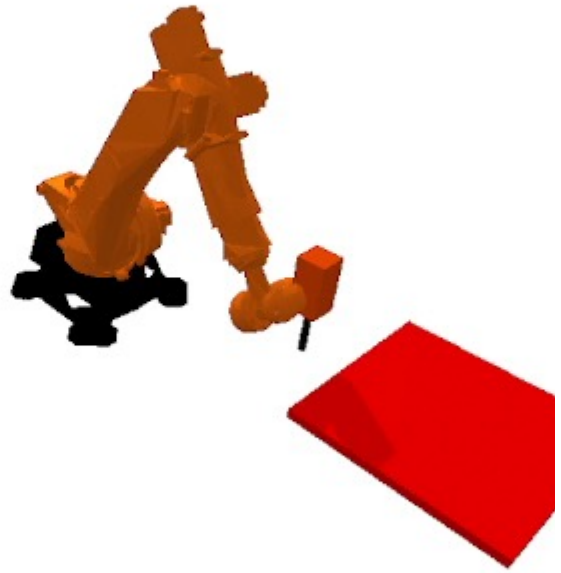
$$\vec{F} = \sum_{i=1}^{t} \vec{F}_i$$

Cutting force per tooth:

$$\vec{F}_i = \frac{F_m}{d^{d_m-1}} h_i \vec{e}_i$$

With chip thickness d :

$$d = \frac{v}{\omega t}$$



Simulation an ROS anschließen

Pybullet Industrial

```python
class PyBulletNode(Node):
    def __init__(self):
        super().__init__('pybullet_simulation_node')
        self.physicsClient = p.connect(p.GUI)

        urdf_path = "path/to/your/robot/urdf.urdf"
        self.robot = pi.RobotBase(urdf_path, [0, 0, 0], [0, 0, 0, 1])

        # -- setup the rest of your simulation here --

        self.joint_state_publisher = self.create_publisher(JointState, 'joint_states', 10)
        self.desired_joint_state_subscriber = self.create_subscription(
            JointState, 'desired_joint_states', self.handle_desired_joint_states, 10)

        self.update_interval = 0.01
        self.timer = self.create_timer(self.update_interval, self.update_simulation)
        self.step_counter = 0

    def handle_desired_joint_states(self, msg: JointState):
        target_positions = {name: position for name, position in zip(msg.name, msg.position)}
        self.robot.set_joint_position(target_positions)

```

```
        self.joint_state_publisher = self.create_publisher(JointState, 'joint_states', 10)
        self.desired_joint_state_subscriber = self.create_subscription(
            JointState, 'desired_joint_states', self.handle_desired_joint_states, 10)

        self.update_interval = 0.01
        self.timer = self.create_timer(self.update_interval, self.update_simulation)
        self.step_counter = 0

    def handle_desired_joint_states(self, msg: JointState):
        target_positions = {name: position for name, position in zip(msg.name, msg.position)}
        self.robot.set_joint_position(target_positions)

    def update_simulation(self):
        p.stepSimulation()
        self.step_counter += 1

        current_joint_state = self.robot.get_joint_state()
        joint_state_msg = JointState()
        sim_time_sec = int(self.step_counter * self.update_interval)
        sim_time_nsec = int((self.step_counter * self.update_interval - sim_time_sec) * 1e9)
        joint_state_msg.header = Header(stamp=rclpy.time.Time(seconds=sim_time_sec, nanoseconds=sim_time_nsec))

        joint_state_msg.name = list(current_joint_state.keys())
        joint_state_msg.position = [state['position'] for state in current_joint_state.values()]
        joint_state_msg.velocity = [state['velocity'] for state in current_joint_state.values()]
        joint_state_msg.effort = [state['torque'] for state in current_joint_state.values()]

        self.joint_state_publisher.publish(joint_state_msg)
```

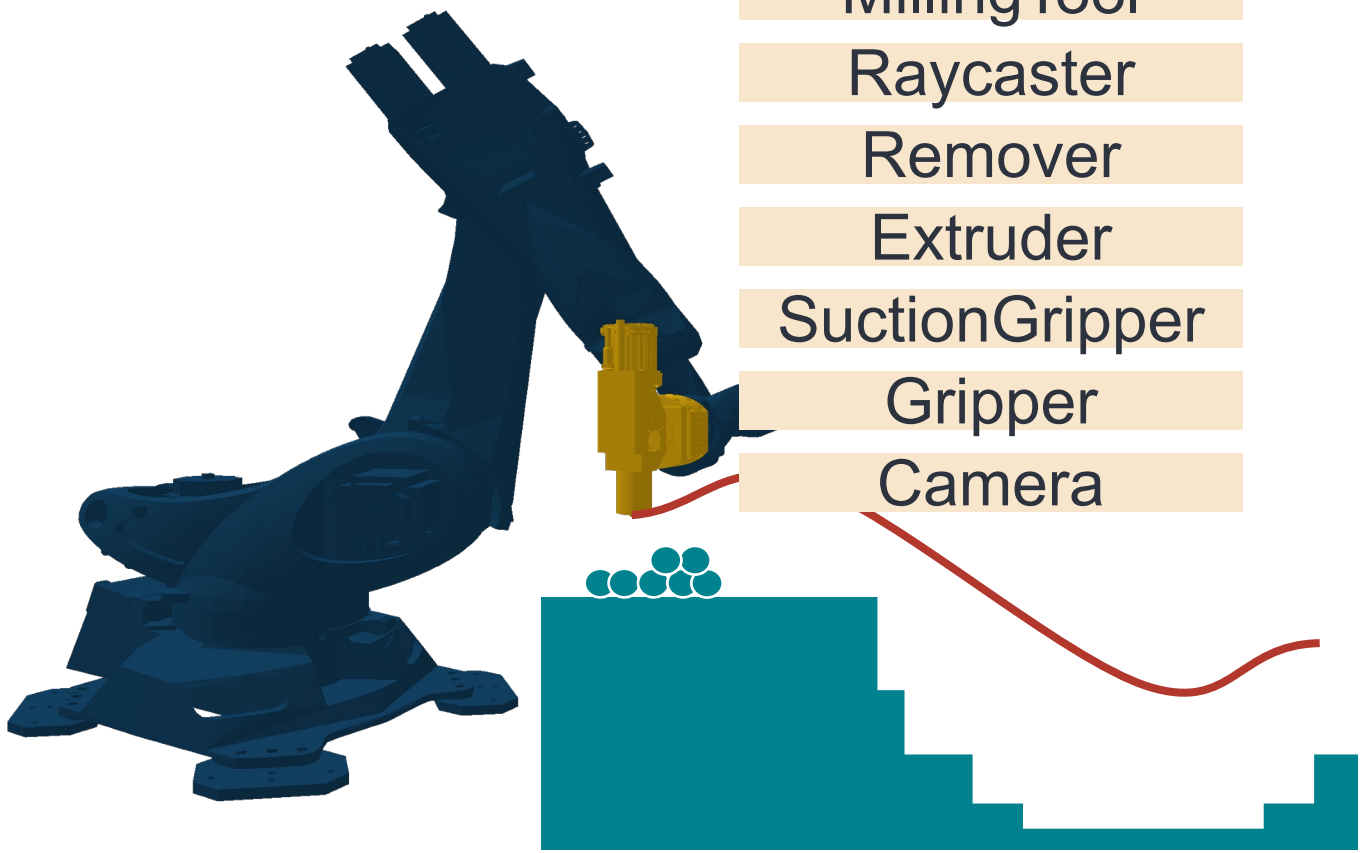
Einschub: VR

Pybullet Industrial

Klassenübersicht

Pybullet Industrial

RobotBase



EndeffectorTool

- MillingTool
- Raycaster
- Remover
- Extruder
- SuctionGripper
- Gripper
- Camera

Particle

- MetalVoxel
- Paint
- Plastic

ToolPath

GCodeProcessor

G-Code Processor

Pybullet Industrial

%Fahre direkt zur gegebenen Pose
G0 X1.9 Y-0.5 Z1.5 A-1.5707 B0 C0

%Interpoliere linear zu den neuen Koordinaten
G1 X2.7
G1 Z1.3

%Wechsle Werkzeug
T1

%Aktiviere und Deaktiviere Werkzeug
M10
M11

```python
m_commands = {"10": [lambda: actuate_gripper(gripper, 1)],"11": [lambda: actuate_gripper(gripper, 0)]}
t_commands = {"0": [lambda: decouple_endeffector(gripper)],"1": [lambda: couple_endeffector(gripper, robot, 'link6')]}

dirname = os.path.dirname(__file__)
textfile = os.path.join(dirname, 'Gcodes', 'gcode_G123.txt')
with open(textfile, encoding='utf-8') as f:
    gcode_input = f.read()

gcode_object = pi.GCodeProcessor(gcode_input, robot,endeffector_list,m_commands, t_commands)

gcode_iterator = iter(gcode_object)
```

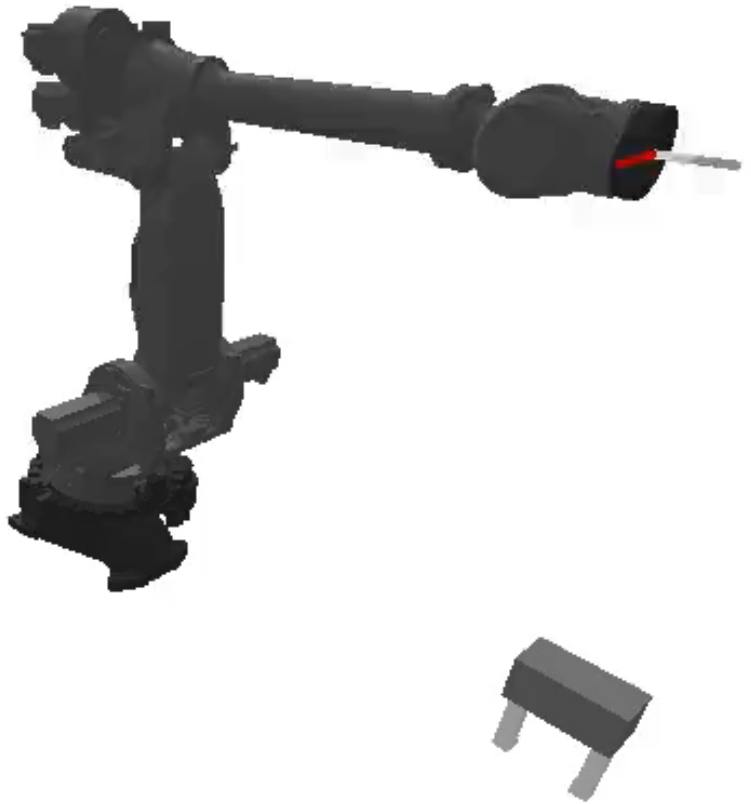
% with "%" it is possible to place comments in the G-Code

%Go to default position and adjust the orientation
G0 X1.9 Y-0.5 Z1.5 A-1.5707 B0 C0
%Approach Gripper
G1 X2.7
G1 Z1.3

%Toolchange (if programmed)
T1
%Move up and adjust orientation
G1 Z1.5 A3.1415 B0 C0
%Go back to the default position
G1 X1.9 Y-0.5

%Close the gripper

Pybullet Industrial



G-Code zu ROS-Schnittstelle

```python
cli_couple = node.create_client(CoupleEndeffector, 'couple_endeffector')
cli_decouple = node.create_client(DecoupleEndeffector, 'decouple_endeffector')

def couple_endeffector_ros(gripper_name):
    req = CoupleEndeffector.Request()
    req.gripper_name = gripper_name
    future = cli_couple.call_async(req)
    rclpy.spin_until_future_complete(node, future)
    return future.result().success

def decouple_endeffector_ros(gripper_name):
    req = DecoupleEndeffector.Request()
    req.gripper_name = gripper_name
    future = cli_decouple.call_async(req)
    rclpy.spin_until_future_complete(node, future)
    return future.result().success

# Update your GCode command mappings
m_commands = {"10": [lambda: actuate_gripper(gripper, 1)], "11": [lambda: actuate_gripper(gripper, 0)]}
t_commands = {"0": [lambda: decouple_endeffector_ros('gripper_name')], "1": [lambda: couple_endeffector_ros('gripper_name')]}
```

Pybullet Industrial

CAM Computer Aided Manufacturing Software

G-Code

Pybullet Industrial

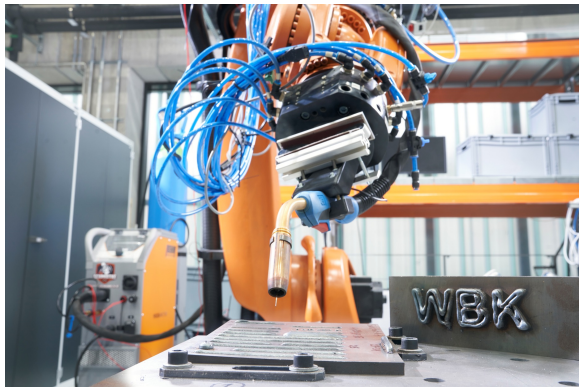
ROS Schnittstellen

ROS

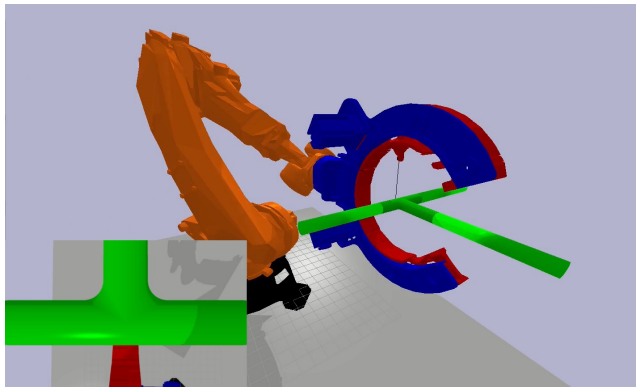
Wie man helfen kann

Pybullet Industrial

Parametrisierte Werkzeuge zur Verfügung stellen



Neue spezialisierte Werkzeuge für andere Fertigungsprozesse entwickeln



Pybullet Industrial weiter entwickeln

Pybullet Industrial

- Planungsinterface
- Mehr Sensoren
- …

Fragen?

Pybullet Industrial

Pybullet Industrial

PowerPoint Slides