

A Python library for scripting and rapid-prototyping of Gazebo simulations

Musa Morena Marcusso Manhães

**Bosch Corporate Research – Renningen, Germany
Robotics Systems Department (CR/AER)**

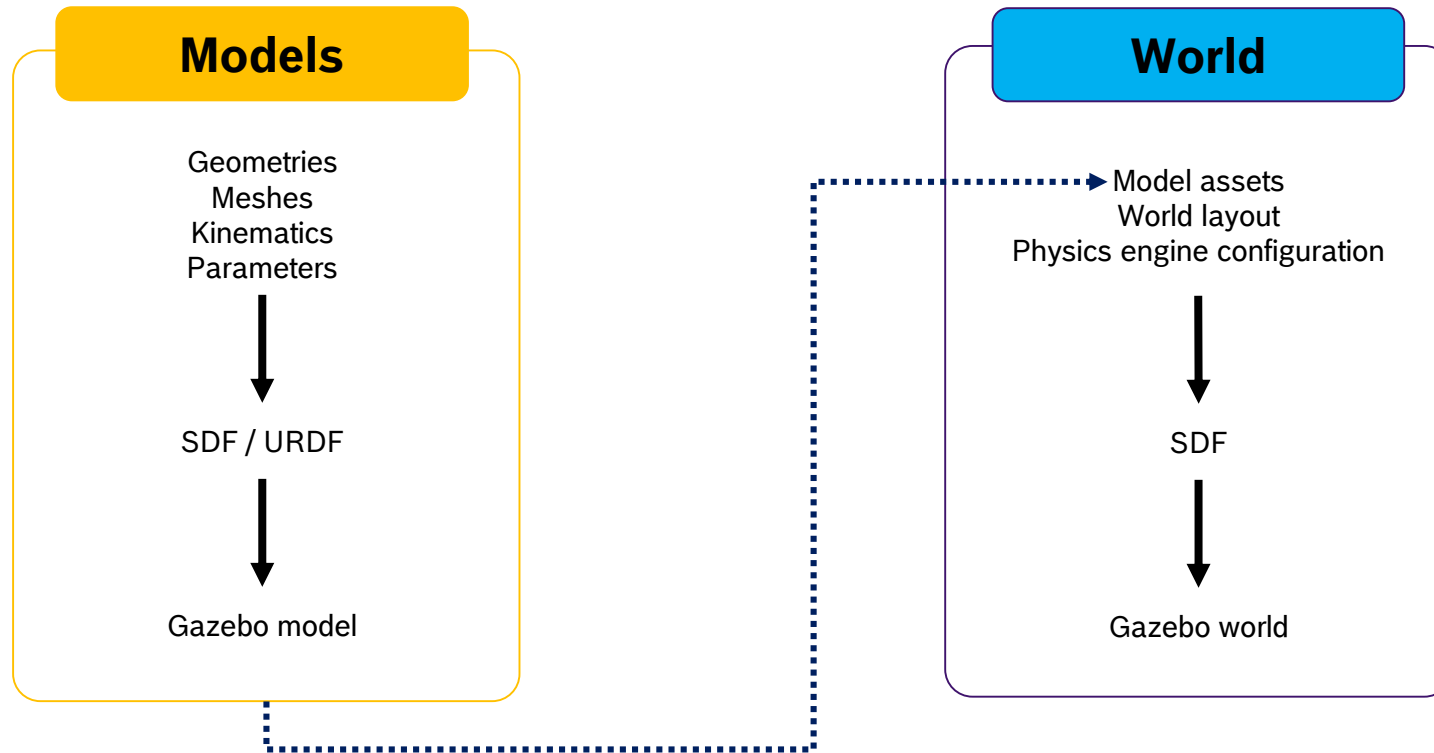
7th ROS-Industrial Conference – Stuttgart, Germany



INTRODUCTION

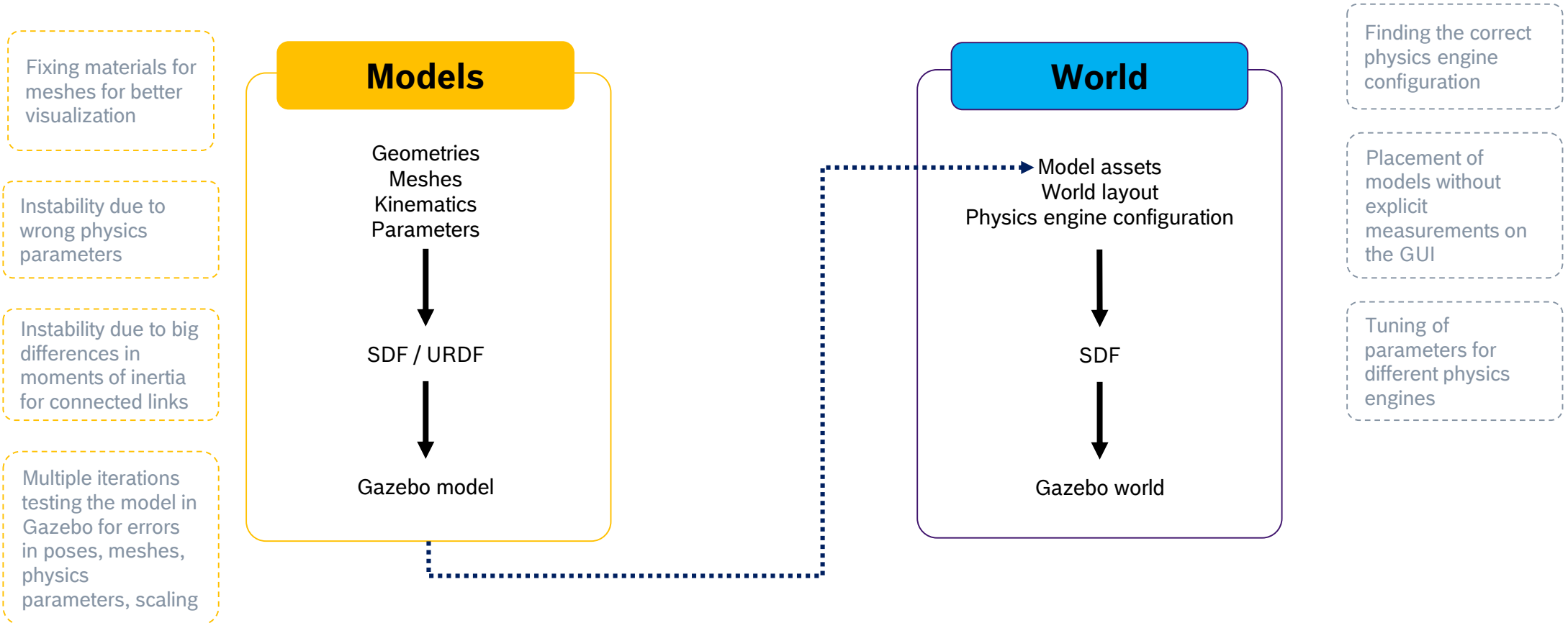
Introduction

The ideal process for the development of a Gazebo simulation



Introduction

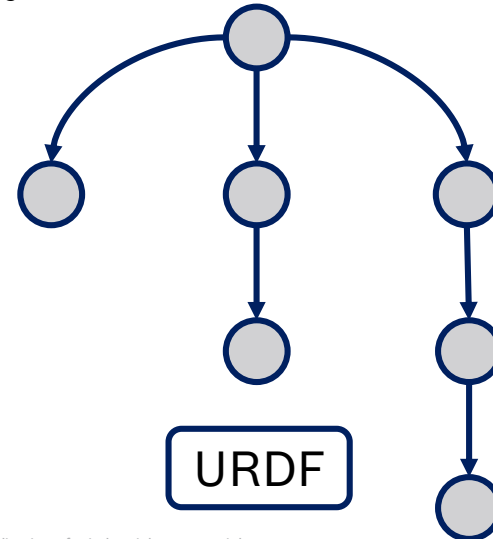
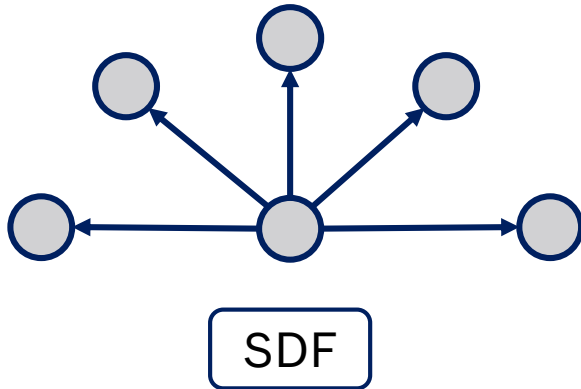
The actual process for the development of a Gazebo simulation



Introduction

Application-dependent difficulties

- ▶ Generation variations of worlds and models (e.g. object placement, model geometry, physics engine configuration)
- ▶ Scripting of world layouts and event-based actions
- ▶ SDF allows more control of the model and its parametrization regarding physics, but most of robot descriptions are written in URDF and don't use SDF to its full potential
- ▶ The differences between SDF and URDF morphology

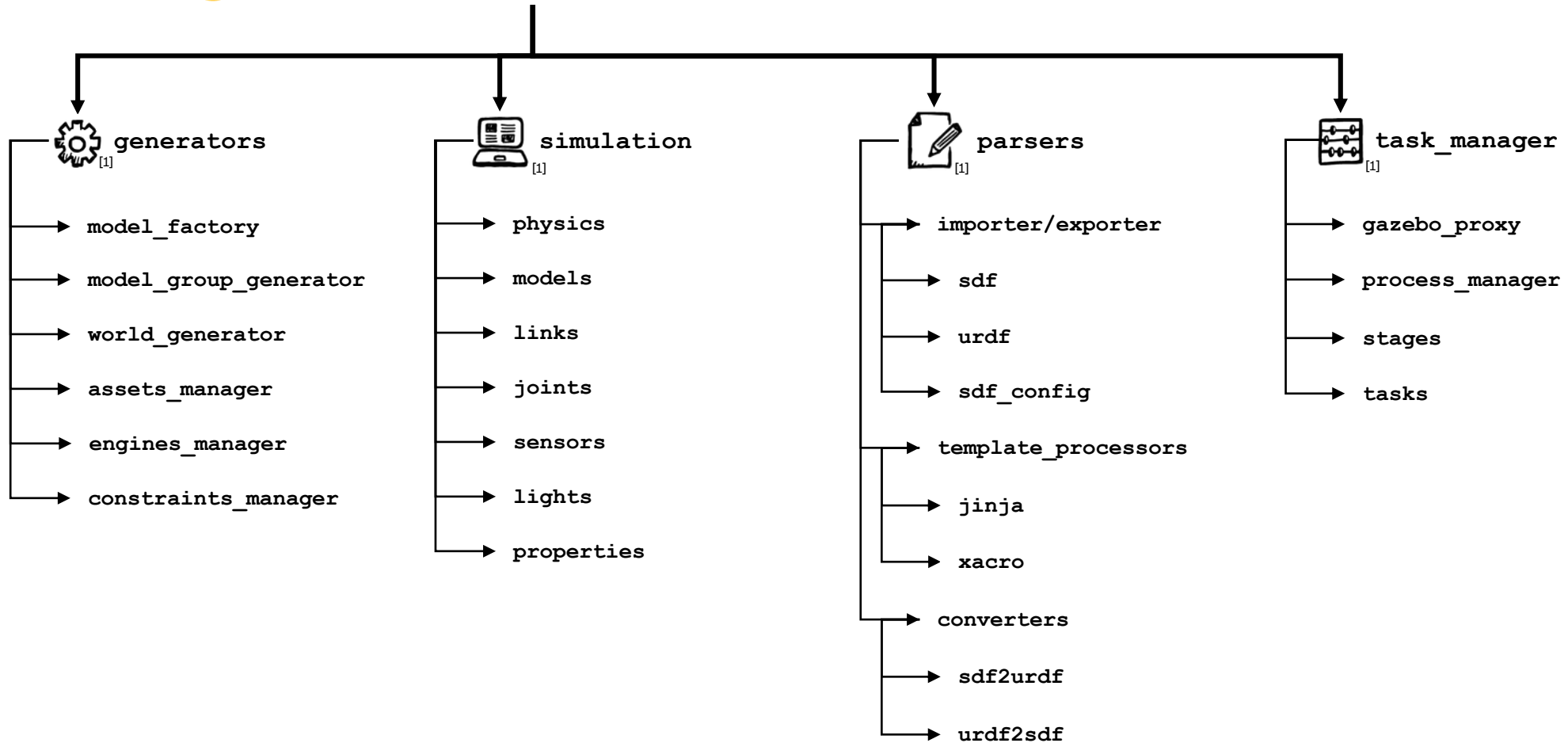


Introduction

Approach: Procedural Generation

- ▶ Technique from gaming development
- ▶ Rapid-prototyping of simulation scenarios
- ▶ Abstractions to simulation entities
- ▶ Allow scripting of Gazebo simulations (generation of models, setting/accessing parameters in runtime, interacting with simulation via script)
- ▶ Extend templating options for robot descriptions
- ▶ Improve conversion between URDF and SDF formats for better use of Gazebo's features

pcg_gazebo_pkgs



[1] Source: <https://www.iconfinder.com/iconsets/brainy-mixed>

License: <https://creativecommons.org/licenses/by/3.0/>

FEATURES

- List all static Gazebo models found on the resources path
- Load the Gazebo model using its tag name
- Visualize visual and collision geometries
- Access and edit SDF parameters without editing the file

JupyterLab - Mozilla Firefox

localhost:8888/lab

File Edit View Run Kernel Tabs Settings Help

02 - Inspection of Gazebo models Python 3

Inspection of Gazebo models

List all Gazebo models

The library loads all Gazebo models in the

- catkin workspace
- `$HOME/.gazebo/models`
- `/usr/share/gazebo-$GAZEBO_VERSION/models`

```
[ ]: import warnings
warnings.filterwarnings('ignore')

from pcg_gazebo.simulation import get_gazebo_model_names

print('List of models:')
for tag in get_gazebo_model_names():
    print('\t - {}'.format(tag))
```

Loading Gazebo models already in the database

Using the tag for the Gazebo model, the SDF file can be loaded as a `SimulationObject`.

```
[ ]: from pcg_gazebo.simulation import SimulationModel

model = SimulationModel.from_gazebo_model('gsa_sofa_manduris')
```

Visualize the visual and collision geometries of the model using `pyglet`.

```
[ ]: model.show(mesh_type='collision')

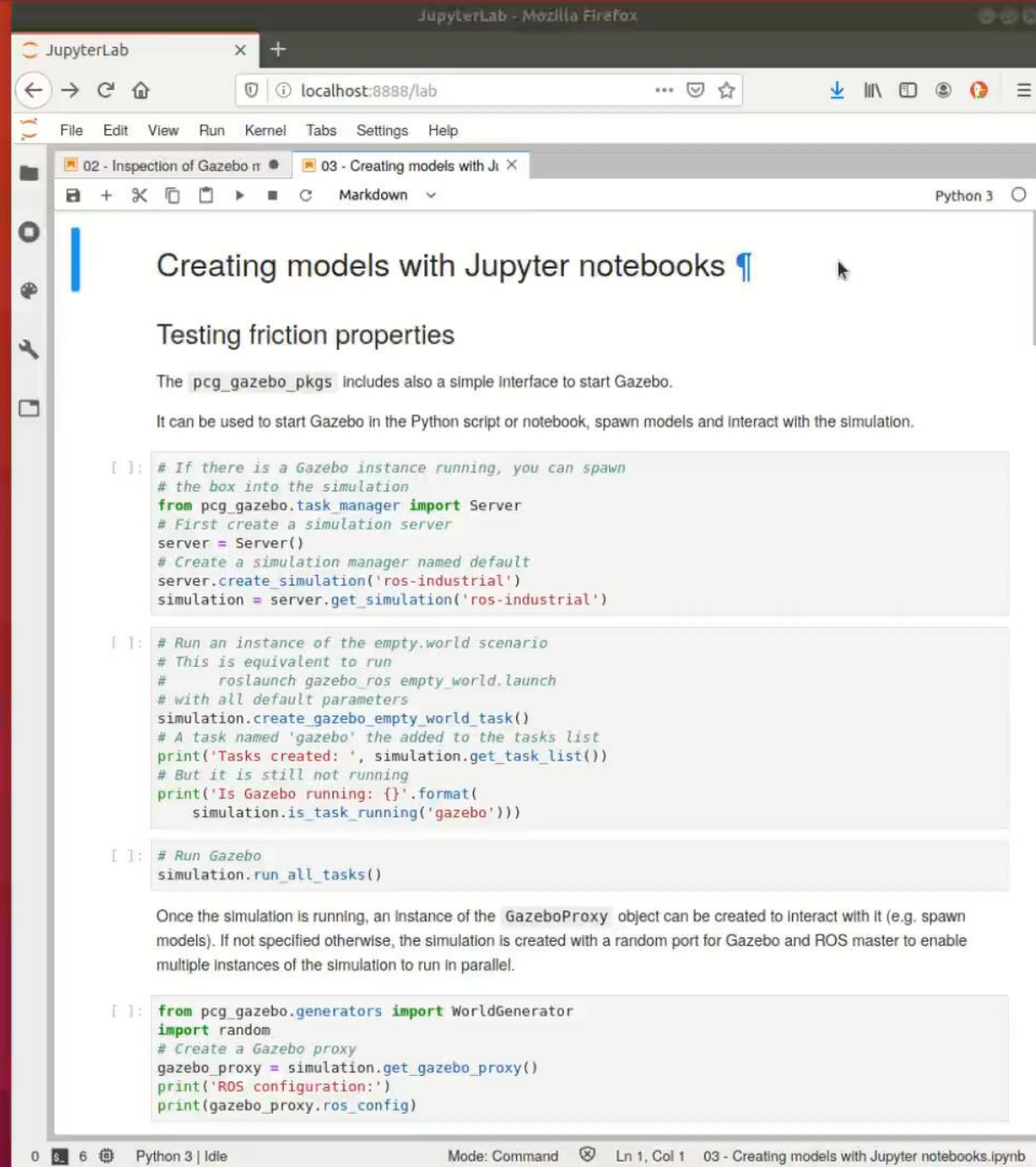
[ ]: model.show(mesh_type='visual')
```

Inspect the SDF elements

Load another Gazebo model and inspect the SDF elements directly.

0 6 Python 3 | Idle Mode: Command Ln 1, Col 1 02 - Inspection of Gazebo models.ipynb

- Open Gazebo inside the notebook
- Create box model and spawn it in Gazebo with 5 different friction coefficients
- Apply force to each box



The screenshot shows a JupyterLab interface in a Mozilla Firefox browser. The notebook is titled '03 - Creating models with Jupyter notebooks'. The code is written in Python 3 and is organized into three cells. The first cell imports the 'Server' class from 'pcg_gazebo.task_manager' and creates a simulation server. The second cell runs an empty world scenario and prints the task list. The third cell runs the Gazebo simulation and prints the ROS configuration.

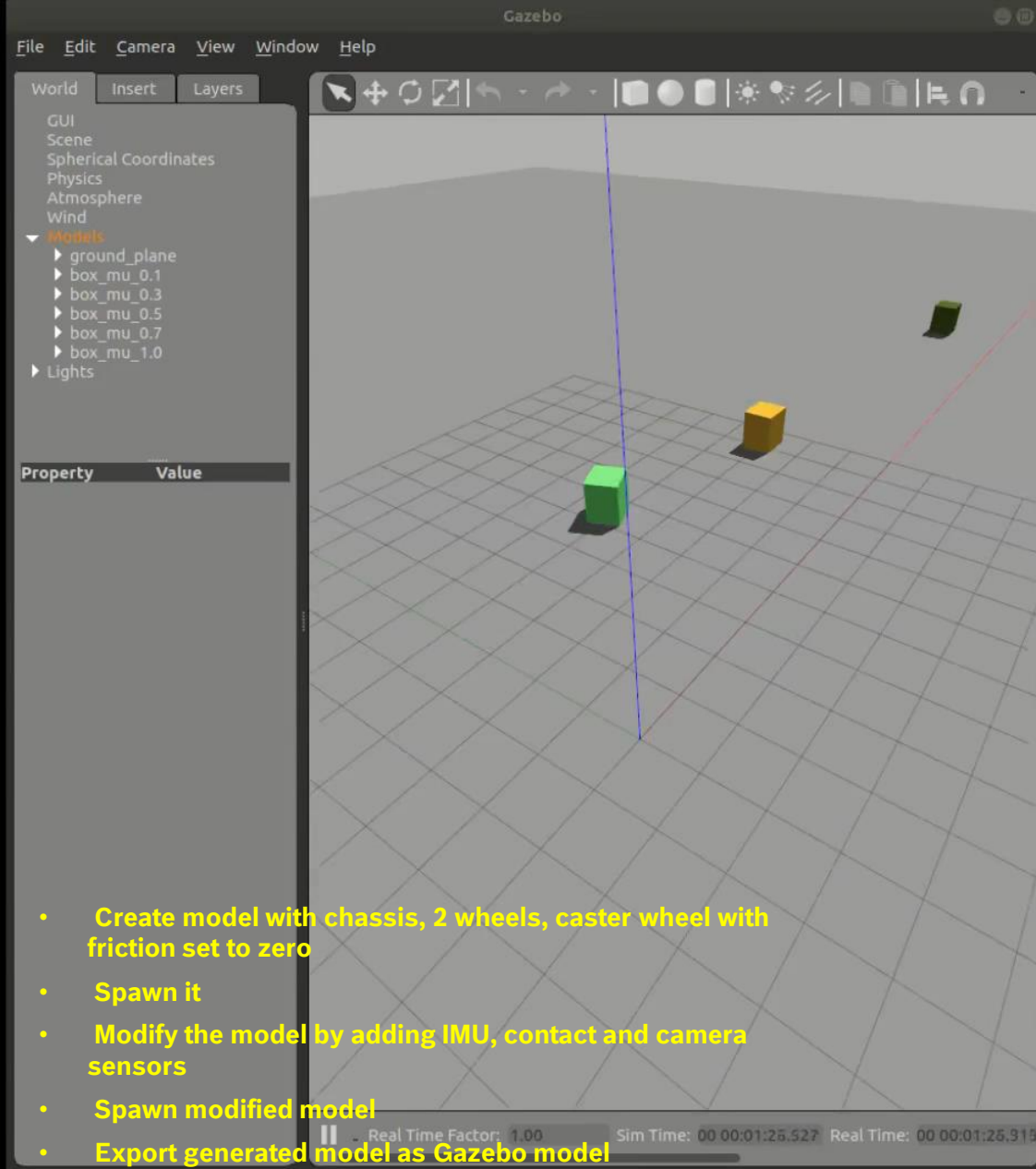
```
[ ]: # If there is a Gazebo instance running, you can spawn
# the box into the simulation
from pcg_gazebo.task_manager import Server
# First create a simulation server
server = Server()
# Create a simulation manager named default
server.create_simulation('ros-industrial')
simulation = server.get_simulation('ros-industrial')

[ ]: # Run an instance of the empty.world scenario
# This is equivalent to run
#   roslaunch gazebo_ros empty_world.launch
# with all default parameters
simulation.create_gazebo_empty_world_task()
# A task named 'gazebo' the added to the tasks list
print('Tasks created: ', simulation.get_task_list())
# But it is still not running
print('Is Gazebo running: {}'.format(
    simulation.is_task_running('gazebo')))

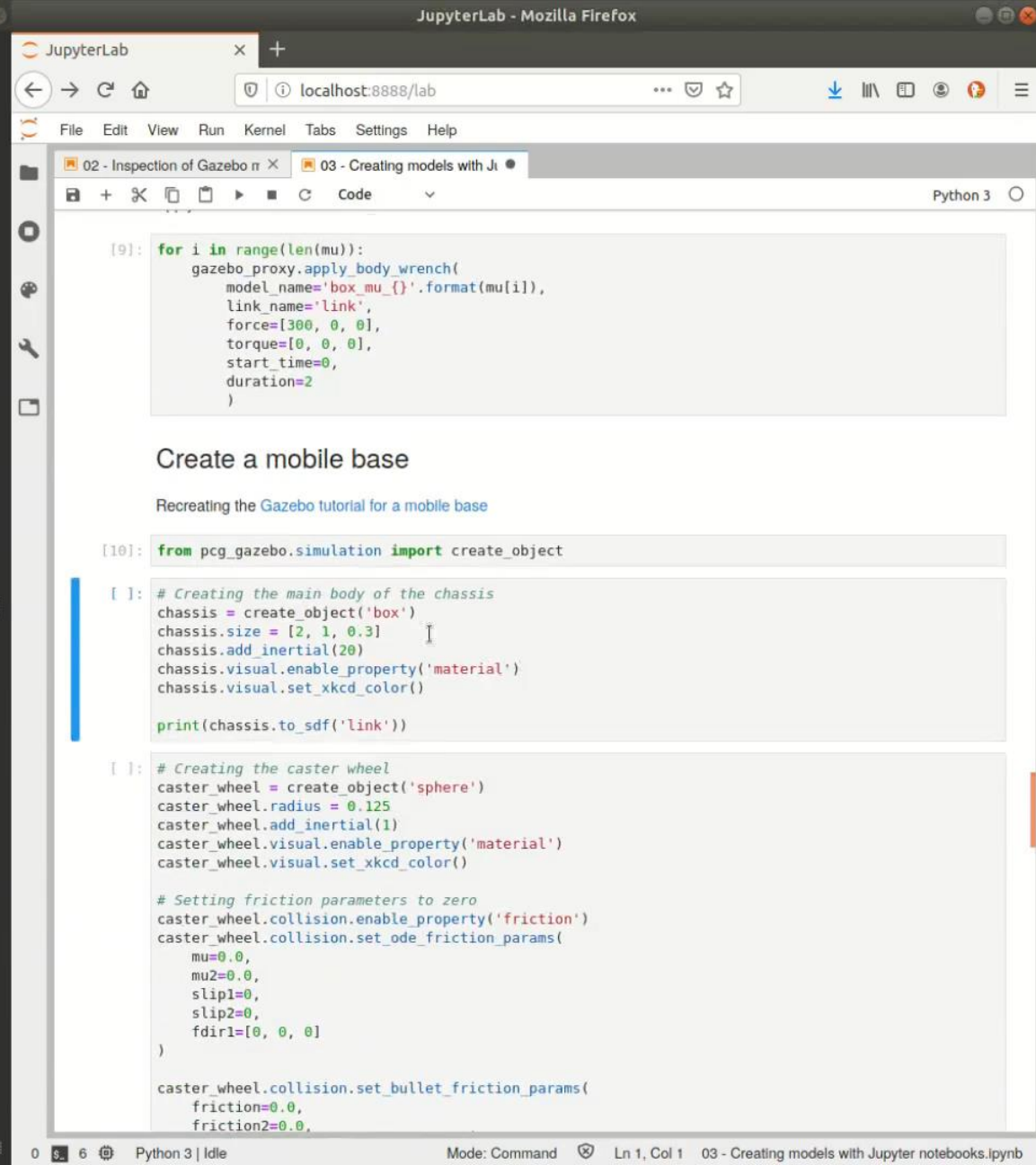
[ ]: # Run Gazebo
simulation.run_all_tasks()

Once the simulation is running, an instance of the GazeboProxy object can be created to interact with it (e.g. spawn models). If not specified otherwise, the simulation is created with a random port for Gazebo and ROS master to enable multiple instances of the simulation to run in parallel.

[ ]: from pcg_gazebo.generators import WorldGenerator
import random
# Create a Gazebo proxy
gazebo_proxy = simulation.get_gazebo_proxy()
print('ROS configuration:')
print(gazebo_proxy.ros_config)
```



- Create model with chassis, 2 wheels, caster wheel with friction set to zero
- Spawn it
- Modify the model by adding IMU, contact and camera sensors
- Spawn modified model
- Export generated model as Gazebo model



- Use model factory to create single-link models (box, sphere, cylinder, from mesh, from extruded polygon)
- Use lambda functions to dynamically generate the parameters

JupyterLab - Mozilla Firefox

localhost:8888/lab

File Edit View Run Kernel Tabs Settings Help

04 - Model factory.ipynb Python 3

Model factory

Single-link models can be easily generated using a few parameters.

```
[ ]: from pcg_gazebo.generators.creators import create_models_from_config
from pcg_gazebo.task_manager import Server

# Start an empty world Gazebo simulation
server = Server()
server.create_simulation('default')
simulation = server.get_simulation('default')
simulation.create_gazebo_empty_world_task()
print(simulation.get_task_list())
print('Is Gazebo running: {}'.format(
    simulation.is_task_running('gazebo')))
simulation.run_all_tasks()

# Create a Gazebo proxy
gazebo_proxy = simulation.get_gazebo_proxy()
```

```
[ ]: import random

def create_and_spawn(config, pos=None):
    models = create_models_from_config(config)
    for model in models:
        model.spawn(
            gazebo_proxy=gazebo_proxy,
            robot_namespace=model.name,
            pos=pos if pos is not None else [
                20 * random.random() - 10,
                20 * random.random() - 10,
                2 * random.random()
            ])
```

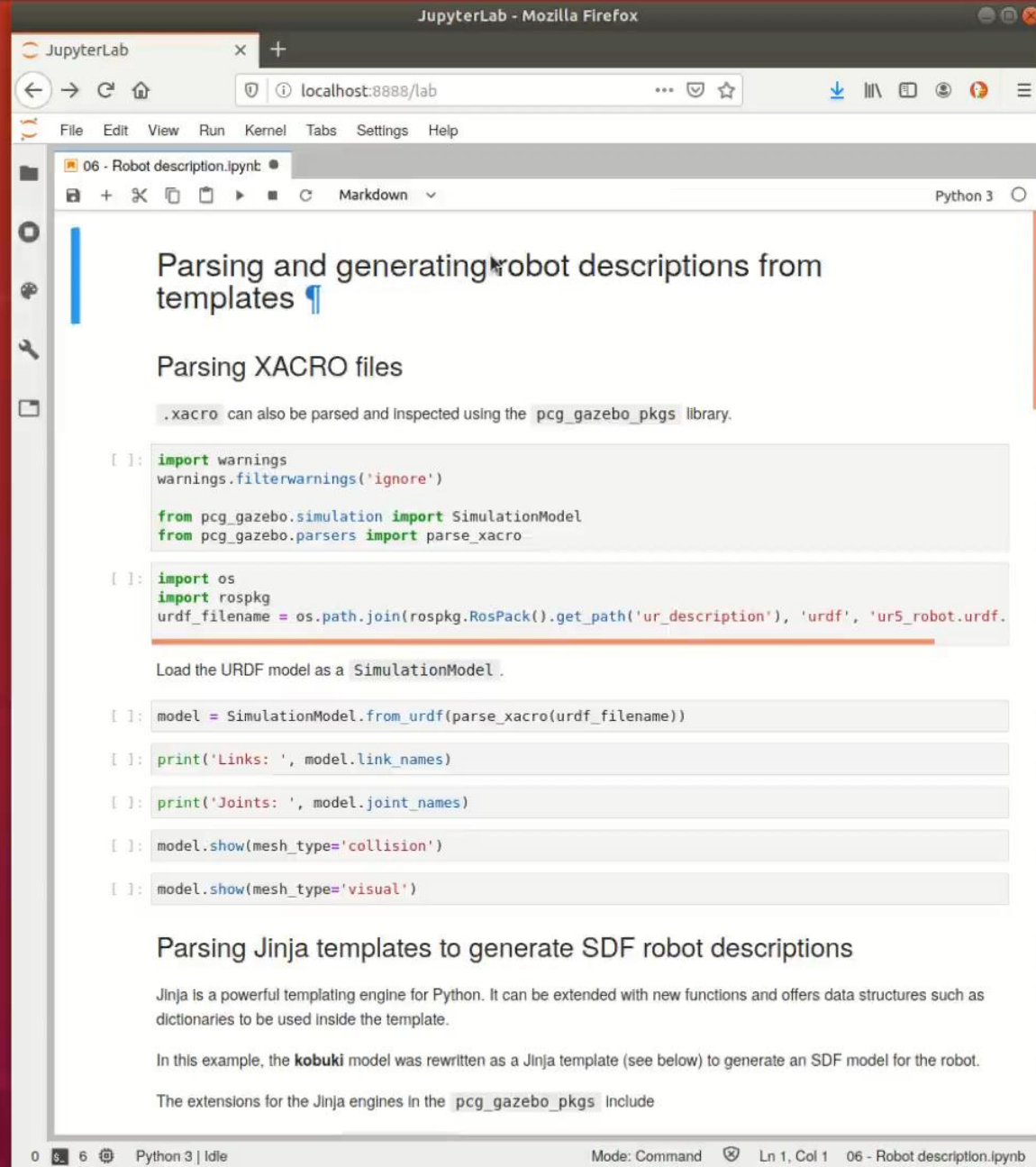
Extruded models

```
[ ]: from pcg_gazebo.generators.shapes import random_rectangle, \
    random_points_to_triangulation, random_rectangles

config = [
    dict(
        type='extrude',
        args=dict(
```

0 6 Python 3 | Idle Mode: Command Ln 1, Col 1 04 - Model factory.ipynb

- Load xacro files and its generated URDF model
- Parse Jinja templates with pcg extensions for generation of model and world configurations
- Call ROS processes from the notebook to interact with the robot



JupyterLab - Mozilla Firefox

localhost:8888/lab

File Edit View Run Kernel Tabs Settings Help

06 - Robot description.ipynb

Markdown Python 3

Parsing and generating robot descriptions from templates

Parsing XACRO files

.xacro can also be parsed and inspected using the `pcg_gazebo_pkgs` library.

```
[ ]: import warnings
warnings.filterwarnings('ignore')

from pcg_gazebo.simulation import SimulationModel
from pcg_gazebo.parsers import parse_xacro

[ ]: import os
import rospkg
urdf_filename = os.path.join(rospkg.RosPack().get_path('ur_description'), 'urdf', 'ur5_robot.urdf.
```

Load the URDF model as a `SimulationModel`.

```
[ ]: model = SimulationModel.from_urdf(parse_xacro(urdf_filename))

[ ]: print('Links: ', model.link_names)

[ ]: print('Joints: ', model.joint_names)

[ ]: model.show(mesh_type='collision')

[ ]: model.show(mesh_type='visual')
```

Parsing Jinja templates to generate SDF robot descriptions

Jinja is a powerful templating engine for Python. It can be extended with new functions and offers data structures such as dictionaries to be used inside the template.

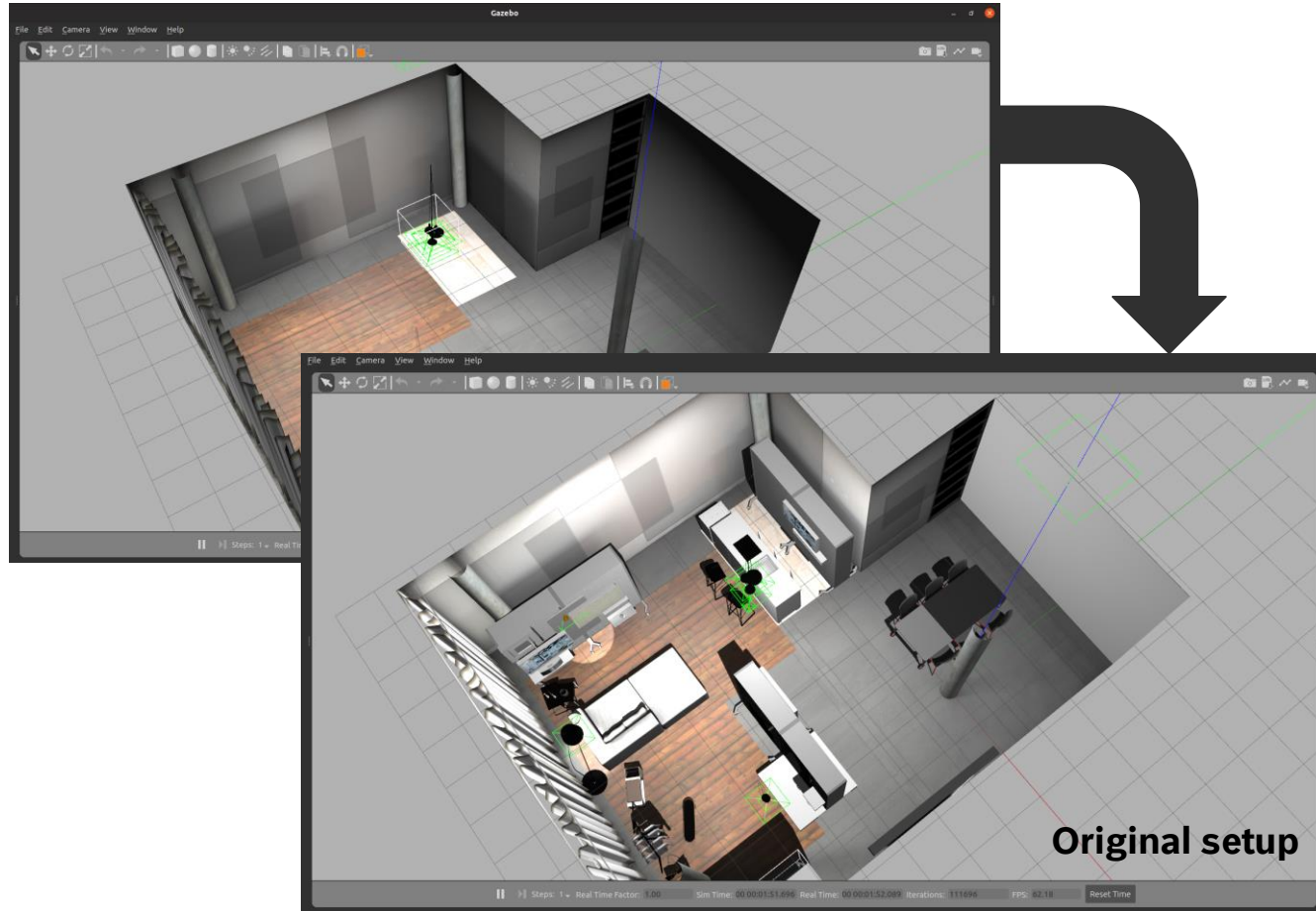
In this example, the **kobuki** model was rewritten as a Jinja template (see below) to generate an SDF model for the robot.

The extensions for the Jinja engines in the `pcg_gazebo_pkgs` include

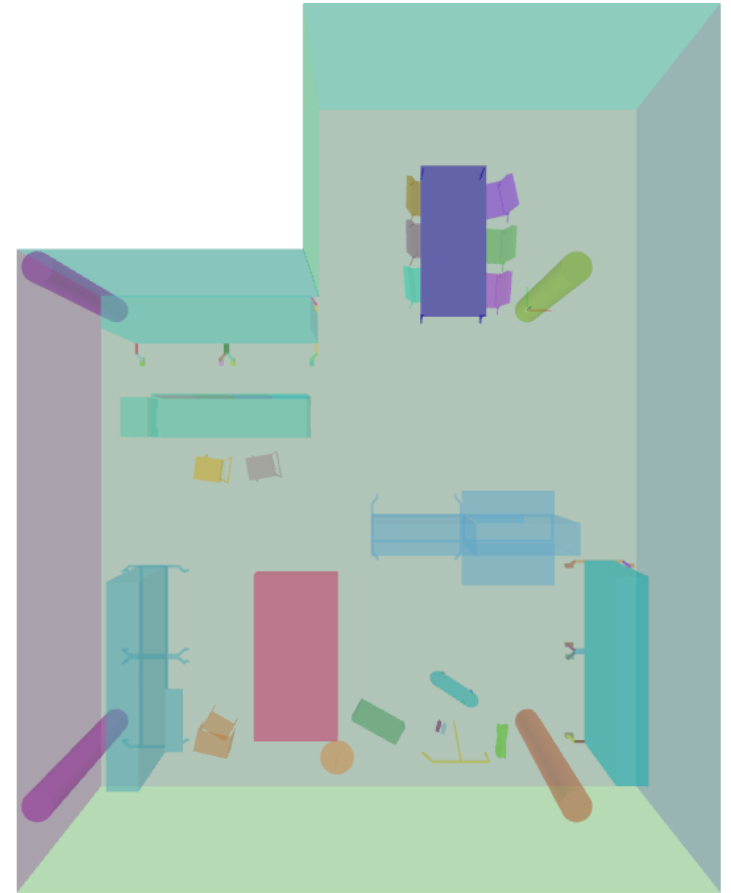
0 6 Python 3 | Idle Mode: Command Ln 1, Col 1 06 - Robot description.ipynb

Features

World generation

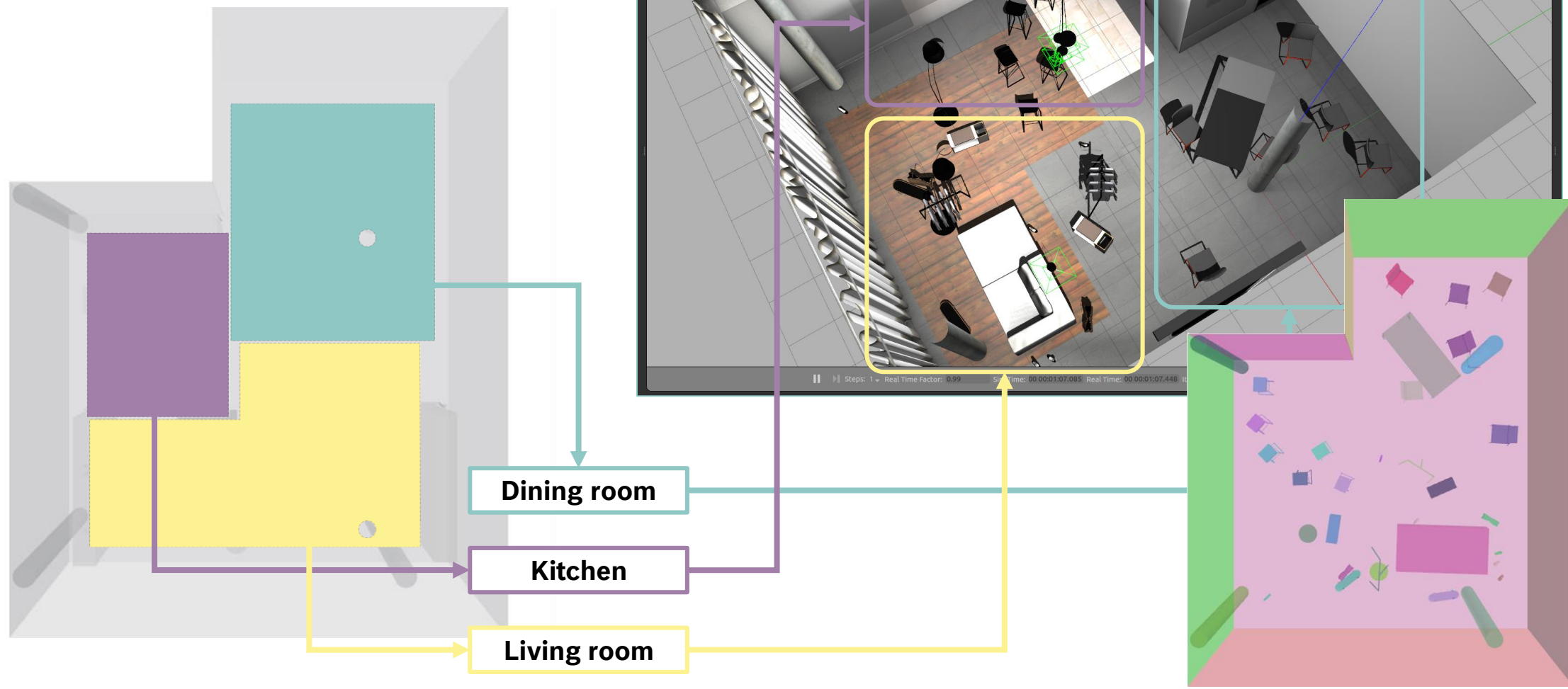


Collision geometries



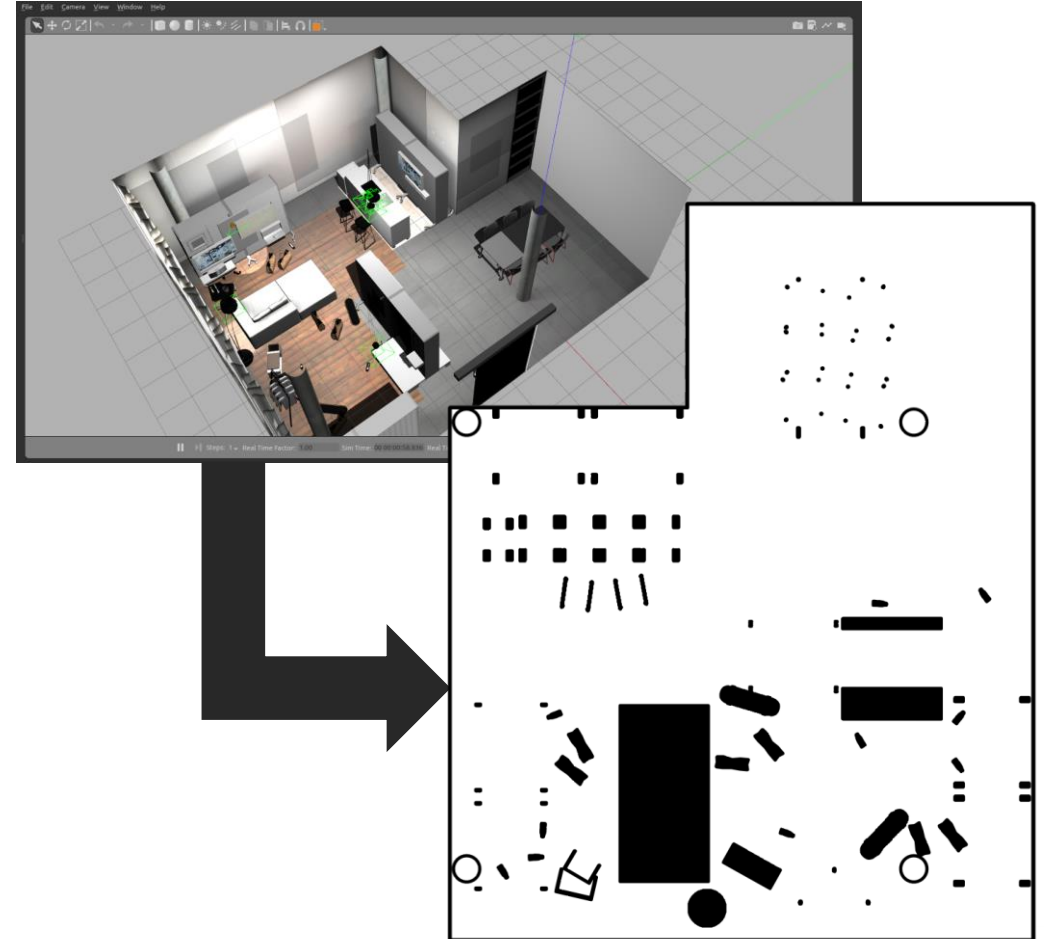
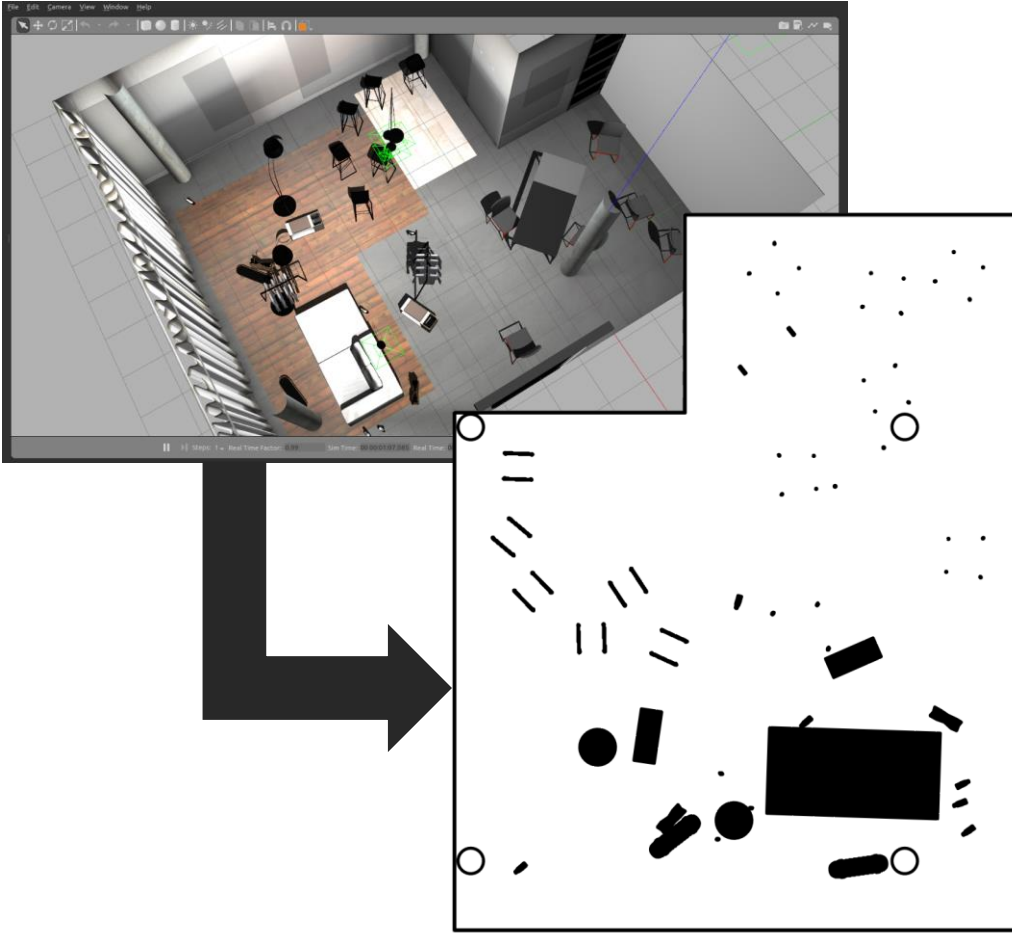
Features

Pose randomization



Features

Generating grid maps from Gazebo worlds via ray tracing



CONCLUSION

Conclusion

- ▶ `pcg_gazebo_pkgs` can be used for testing simulation scenarios without editing XML files
- ▶ Scripting can be used to generate assets and interact with the simulation
- ▶ Dynamic model and world generation allows generation of large number of assets with small effort for testing robotics systems solutions in various contexts
- ▶ Python libraries can be used on the simulation building process, along with Jupyter notebooks
- ▶ Model editing and inspection
- ▶ `sdf2urdf` and `urdf2sdf` give more possibilities of ways to represent the robot description
- ▶ Package available at https://github.com/boschresearch/pcg_gazebo_pkgs under Apache-2.0 license

THANK YOU

CONTACT

Musa Morena Marcusso Manhães (musa.marcusso@de.bosch.com)

REPOSITORY

https://github.com/boschresearch/pcg_gazebo_pkgs