pcg_gazebo_pkgs

A PYTHON LIBRARY FOR SCRIPTING AND RAPID-PROTOTYPING OF SIMULATED GAZEBO MODELS AND WORLDS

Musa Morena Marcusso Manhães

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

ROSCon 2019 - Macau

BOSCH

INTRODUCTION

BOSCH

Procedural Generation for Gazebo
Motivation

- **Problems:**
  - How to do rapid-prototyping of simulations in Gazebo (specially for non-simulation experts)
  - How to quickly design and create new and/or multiple simulations with different
    - Physics engine parameters [1]
    - Robot components and structures [1]
    - Objects placements [1]

- **Idea:** *Procedural Generation for Gazebo*
  - Technique from gaming development

- **Approach**
  - Use scripting language
  - Create abstractions of Gazebo model, worlds and configuration data structures (e.g. physics engine configuration)
  - Use of SDF and URDF formats to import/export model and world descriptions
  - Algorithmically generate of scenes, models and parameter configuration sets for a simulation


[1]

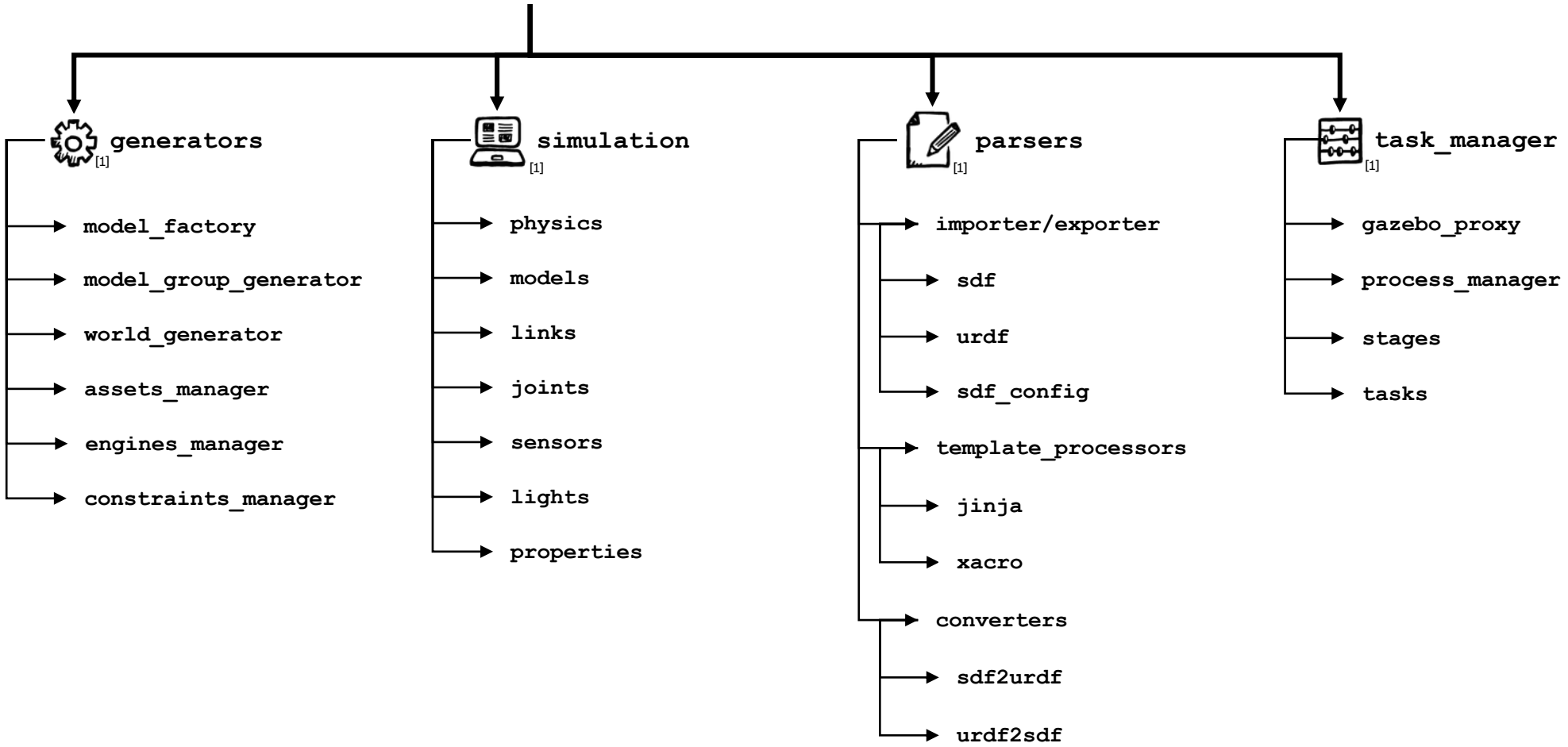
[1] Source: https://www.iconfinder.com/iconsets/brainy-mixed
License: https://creativecommons.org/licenses/by/3.0/

Procedural Generation for Gazebo
Detailed goals

- Rapid-prototyping of simulation scenarios
  - Create abstractions to simulation entities (e.g. models, sensors, physics engine parameters) that can be easily manipulated
  - Allow scripting of Gazebo simulations (generation of models, setting/accessing parameters in runtime, interacting with simulation via script)
    - Easy factory functions to create simulation models to populate scenarios (e.g. geometric primitives, derived from meshes)
    - Also useful in e.g. machine learning algorithms, reinforcement learning, optimization
  - Integration of scripted simulations in automated simulation-based tests
    - Generation of simulation scenarios for testing of, for example, motion planners and SLAM algorithms

- More control over the robot descriptions
  - XML parsers for SDF and URDF
  - 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/

BOSCH

FEATURES

Inspection of Gazebo models
Visualizing models already in the database

- The library loads the paths of all static Gazebo models in the
  - current catkin workspace
  - **`$HOME/.gazebo/models`**
  - **`/usr/share/gazebo-$GAZEBO_VERSION/models`**
- They can be loaded using their tag into a **`SimulationModel`** object

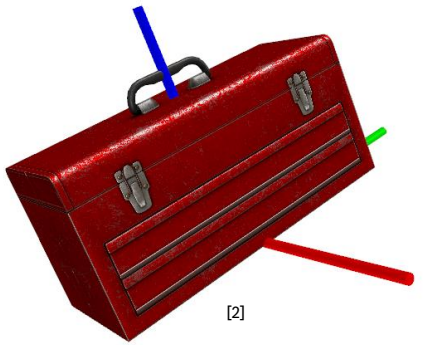
```
In [1]: from pcg_gazebo.simulation import SimulationModel

In [2]: model = SimulationModel.from_gazebo_model('object_jansport_backpack_red')

In [3]: model.show()
```

/usr/local/bin/ipython

[1]

```
In [1]: from pcg_gazebo.simulation import SimulationModel

In [2]: model = SimulationModel.from_gazebo_model('object_toolbox')

In [3]: model.show()
```

/usr/local/bin/ipython



[2]

[1] Source: https://app.ignitionrobotics.org/chapulina/fuel/models/JanSport%20Backpack%20Red
License: Creative Commons – Public Domain
[2] Source: https://app.ignitionrobotics.org/openrobotics/fuel/models/Toolbox
License: Creative Commons – Public Domain

BOSCH

Inspection of Gazebo models

Inspecting SDF from a `SimulationModel` object

```
In [4]: sdf = model.to_sdf()

In [5]: print(sdf)
<model name="toolbox">
  <pose frame="">0 0 0.126548 0 -0 0</pose>
  <static>0</static>
  <self_collide>0</self_collide>
  <allow_auto_disable>0</allow_auto_disable>
  <link name="toolbox">
    <collision name="collision">
      <geometry>
        <mesh>
          <uri>model://object_toolbox/meshes/Toolbox.dae</uri>
          <scale>1 1 1</scale>
        </mesh>
      </geometry>
      <pose frame="">2.59092e-17 1.34051e-17 -0.126548 0 -0 0</pose>
      <max_contacts>10</max_contacts>
    </collision>
    <visual name="visual">
      <geometry>
        <mesh>
          <uri>model://object_toolbox/meshes/Toolbox.dae</uri>
          <scale>1 1 1</scale>
        </mesh>
      </geometry>
      <pose frame="">2.59092e-17 1.34051e-17 -0.126548 0 -0 0</pose>
      <cast_shadows>1</cast_shadows>
      <transparency>0.0</transparency>
    </visual>
    <inertial>
      <inertia>
        <ixx>0.08738482705732964</ixx>
        <iyy>0.034240830541697964</iyy>
        <izz>0.07690599862062936</izz>
        <ixy>0.0</ixy>
        <ixz>0.0</ixz>
        <iyz>0.0</iyz>
      </inertia>
      <pose frame="">0 0 0 0 -0 0</pose>
      <mass>3.0</mass>
    </inertial>
    <pose frame="">0 0 0 0 -0 0</pose>
  </link>
</model>
```

`SimulationModel` objects can be converted into a SDF object

```
In [10]: sdf.links
Out[10]: [<pcg_gazebo.parsers.sdf.link.Link at 0x7fb5cc080c18>]

In [11]: sdf.links[0].inertial.mass.value
Out[11]: 3.0

In [12]: sdf.links[0].visuals[0].geometry.mesh.uri.value
Out[12]: 'model://object_toolbox/meshes/Toolbox.dae'

In [13]: print(sdf.links[0].collisions[0])
<collision name="collision">
  <geometry>
    <mesh>
      <uri>model://object_toolbox/meshes/Toolbox.dae</uri>
      <scale>1 1 1</scale>
    </mesh>
  </geometry>
  <pose frame="">2.59092e-17 1.34051e-17 -0.126548 0 -0 0</pose>
  <max_contacts>10</max_contacts>
</collision>
```

The SDF elements can be accessed via their Python object abstraction

```
In [14]: sdf.links[0].inertial.mass.value = 1

In [15]: print(sdf.links[0].inertial)
<inertial>
  <inertia>
    <ixx>0.08738482705732964</ixx>
    <iyy>0.034240830541697964</iyy>
    <izz>0.07690599862062936</izz>
    <ixy>0.0</ixy>
    <ixz>0.0</ixz>
    <iyz>0.0</iyz>
  </inertia>
  <pose frame="">0 0 0 0 -0 0</pose>
  <mass>1.0</mass>
</inertial>


In [16]: sdf.export_xml('/tmp/new_toolbox.sdf')
```

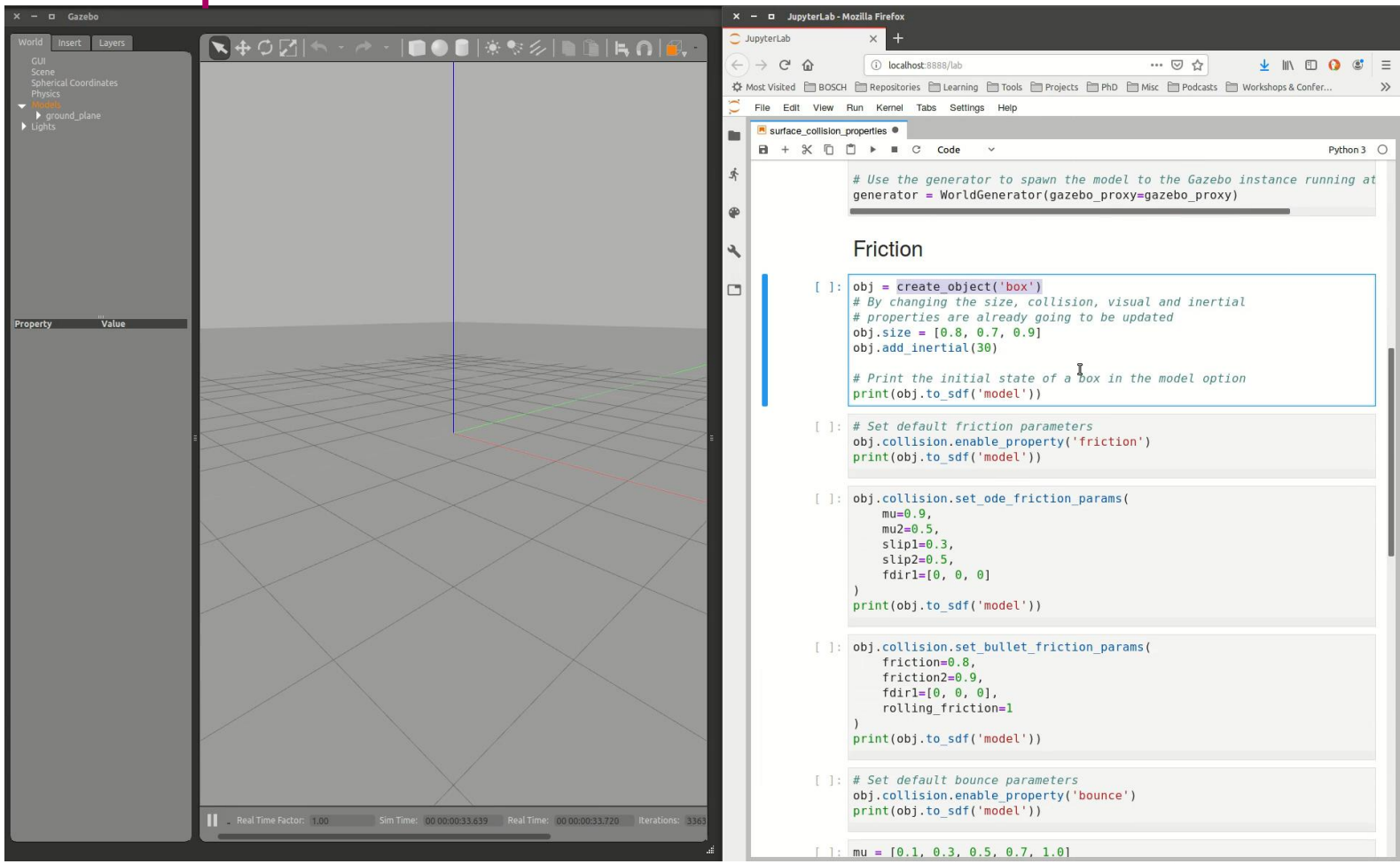
The SDF elements also be altered and the object again exported into a XML file

BOSCH

Creating models using Jupyter notebooks

Experimenting with friction parameters

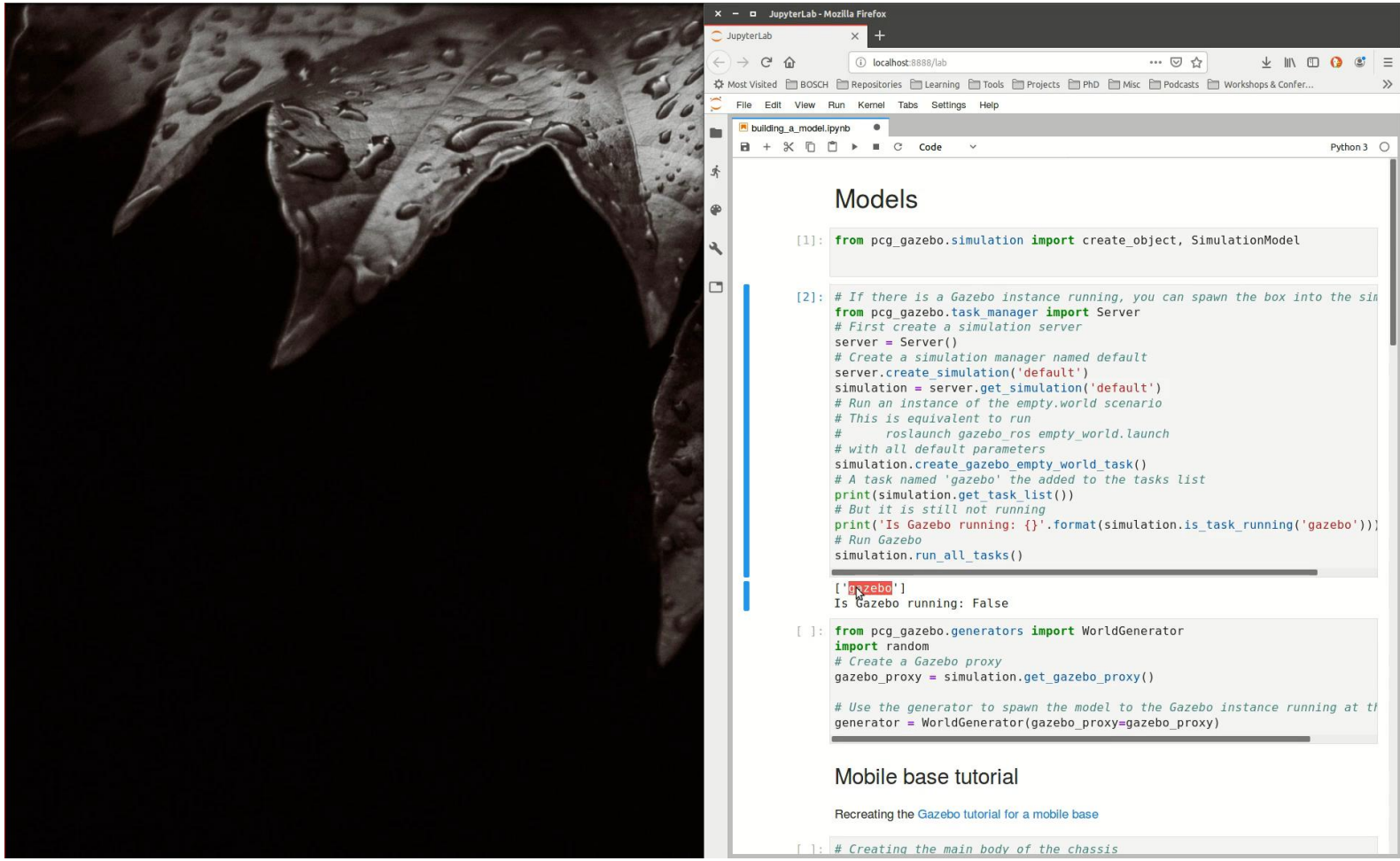
- Create instances of the same **box** model with different friction coefficients
- Inspect the resulting SDF file
- Spawn to Gazebo
- Apply wrench to each model



BOSCH

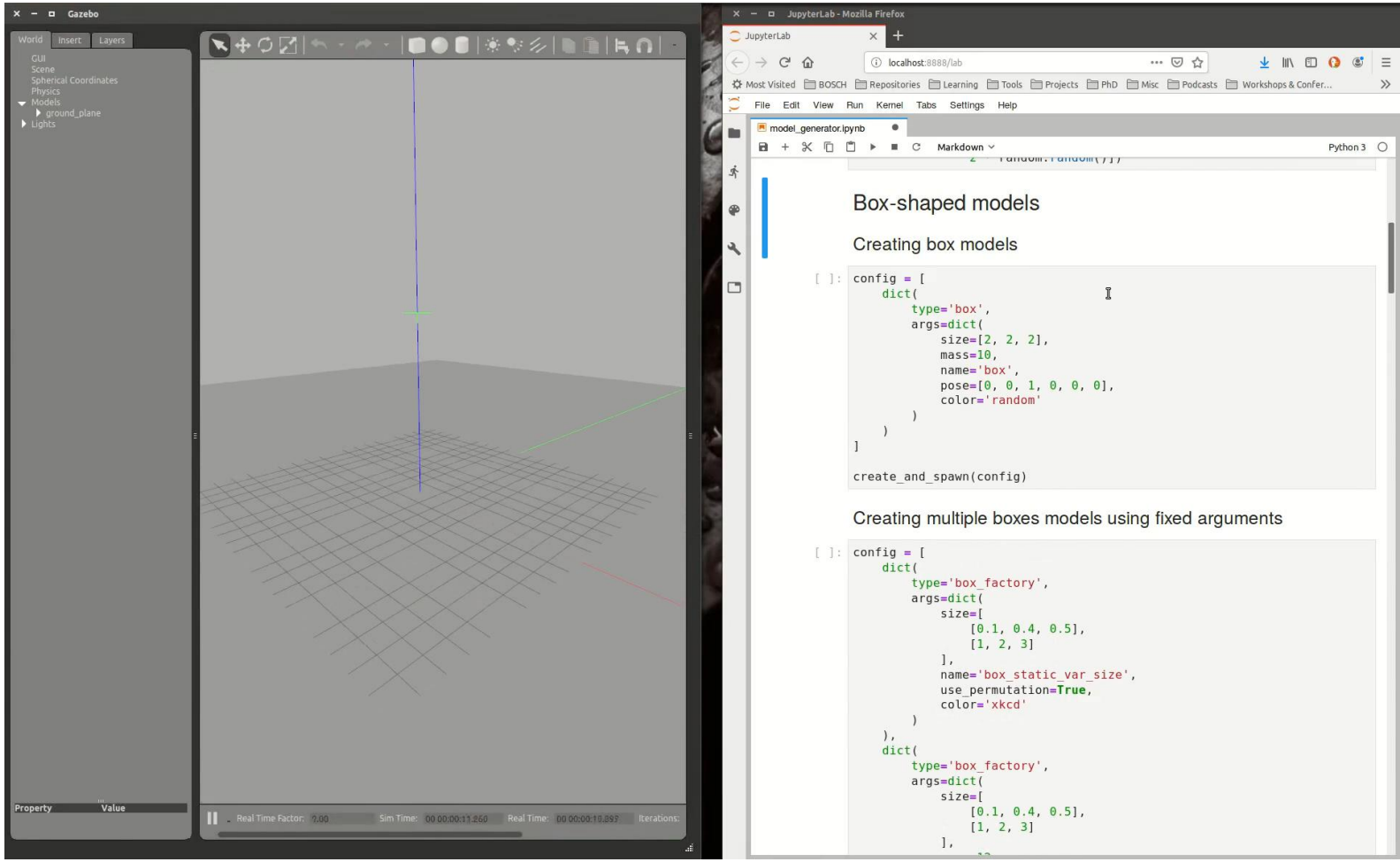
Creating models using Jupyter notebooks
Mobile base tutorial

- Start a Gazebo
- Create mobile base links
  - Chassis
  - Wheels
  - Caster wheel
- Set friction coefficients for caster wheels to zero for both **`bullet`** and **`ode`**
- For links from geometry primitives, the inertial tensors will be computed using the dimensions of the geometry
- Create joints
- Spawn to Gazebo

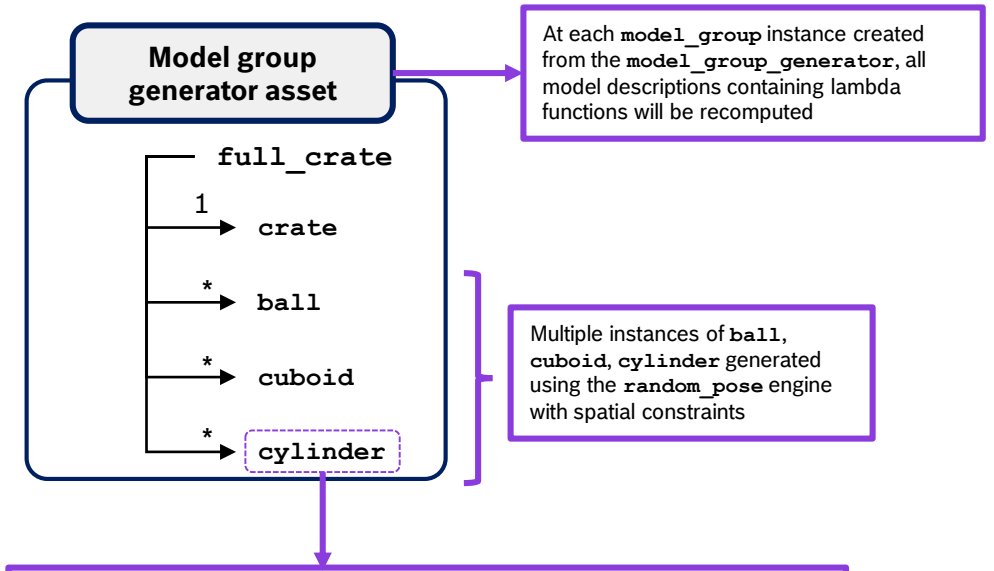


Model factory
Using model factory functions

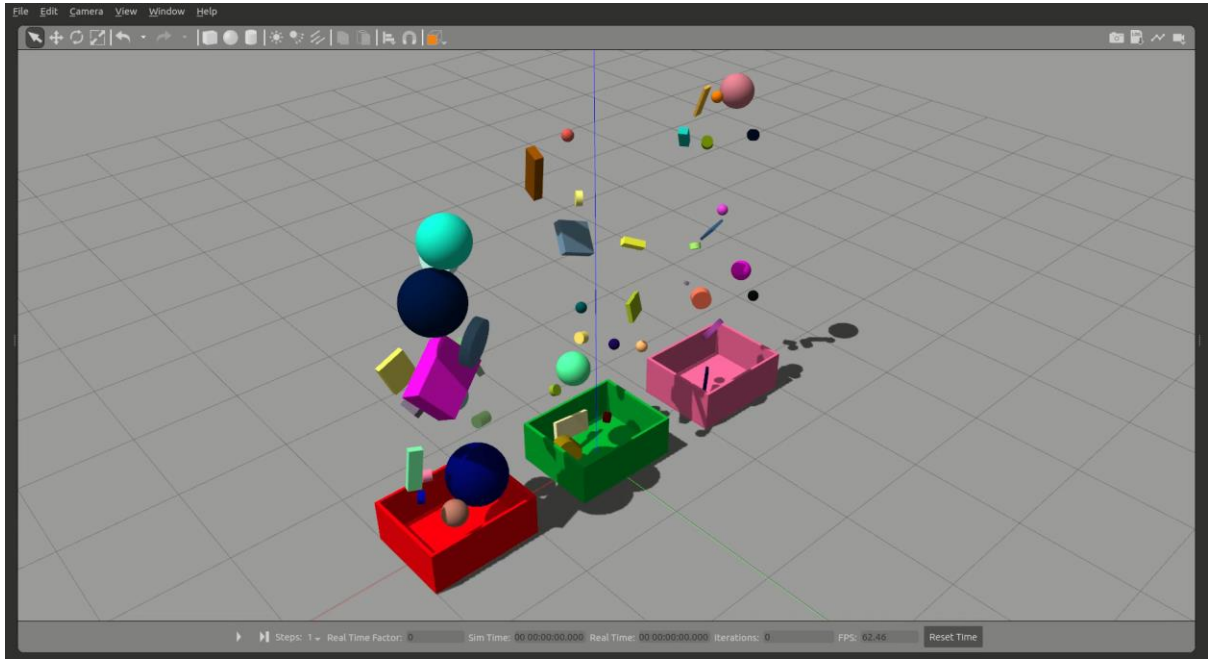
- Complete single-link models can be creaters using the **`pcg_gazebo.creators`** submodule
- Primitives **`box`**, **`sphere`**, **`cylinder`** are available and generated data includes
  - Inertial tensor based on the geometry's dimensions
  - Visual and collision information
- **`extrude`** allows the generation of models from meshes extruded from **`shapely`** geometry objects
- **`mesh`** creators allow the computation of approximated geometries using the input mesh that can be used as simplified collision geometry and input for computation of the moments of inertia
- Models can be parametrized with fixed value inputs or lambda functions



Model group generators
Generating sets of dynamically generated models




```
- tag: cylinder
  description:
    type: cylinder
    args:
      length: "max(0.05, 0.2 *__import__('numpy').random.random())"
      radius: "max(0.05, 0.2 *__import__('numpy').random.random())"
      name: cuboid
      mass: "max(0.01, __import__('numpy').random.random())"
      color: xkcd
```



- **ball**, **cuboid** and **cylinder** properties represented by lambda functions
- New model properties are generated each time an instance of the model is created

SDF Jinja Templates

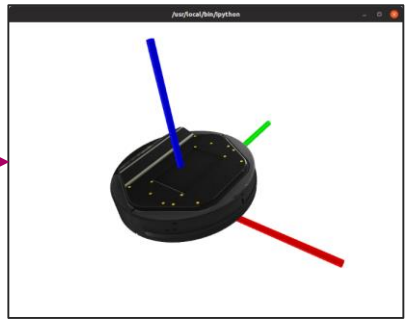
Templating robot descriptions

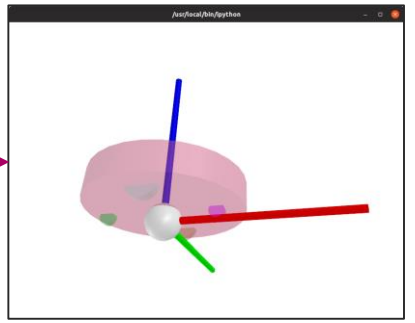
```
{% set base_link = 'base_link' %}

{% if namespace|length > 0 %}
    {% set base_link = [namespace, base_link]|join('/') %}
{% endif %}

<link name="{{ base_link }}">
    <pose>0 0 0 0 0 0</pose>
    <inertial>
        <pose>0 0 0 0 0 0</pose>
        <mass>{{ parameters.base_link_mass }}</mass>
        {{ inertias.solid_cylinder_inertia(
            parameters.base_link_mass,
            parameters.base_link_radius,
            parameters.base_link_length,
            "z") }}
    </inertial>

    <visual name="visual">
        <pose>0 0 0 0 0 0</pose>
        <geometry>
            <mesh>
                <uri>$(find pcg_kobuki_description)/meshes/kobuki/main_body.obj</uri>
                <scale>1 1 1</scale>
            </mesh>
        </geometry>
    </visual>

    <collision name="collision">
        <pose>0 0 0.005737 0 0 0</pose>
        <geometry>
            <cylinder>
                <radius>0.175</radius>
                <length>0.06553</length>
            </cylinder>
        </geometry>

        <surface>
            {{ physics.collision_friction_properties(
                sdf_version=sdf_version|default(1.6, true),
                mu=base_link_friction|default(0.3, true),
                mu2=base_link_friction2|default(0.3, true),
                friction=base_link_friction|default(0.3, true),
                friction2=base_link_friction2|default(0.3, true)
                ) }}
        </surface>
    </collision>
```

`kobuki` robot description written as a SDF Jinja template

Processing the Jinja template, generate SDF and instantiate a **`SimulationModel`** object

```
In [1]: from pcg_gazebo.simulation import SimulationModel

In [2]: from pcg_gazebo.parsers import parse_sdf

In [3]: from pcg_gazebo.utils import process_jinja_template

In [4]: output_xml = process_jinja_template('kobuki.sdf.jinja', parameters=dict(namespace='kobuki', mode='default'))

In [5]: sdf = parse_sdf(output_xml)

In [6]: model = SimulationModel.from_sdf(sdf.models[0])

In [7]: model.show(mesh_type='visual')

In [8]: model.show(mesh_type='collision')
```





Creating a static Gazebo model with metadata in the **`$HOME/.gazebo/models`** folder

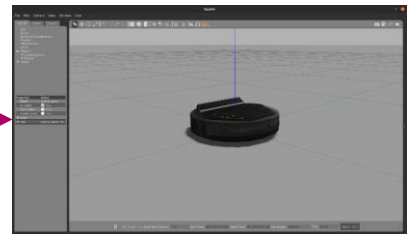
```
In [9]: model.to_gazebo_model()
Out[9]: '/home/[redacted]/.gazebo/models/kobuki'

In [10]: cd /home/[redacted]/.gazebo/models/kobuki/
/home/[redacted]/.gazebo/models/kobuki

In [11]: ls
meshes/  model.config  model.sdf
```

Exporting the URDF file for the **`kobuki`** robot

```
In [12]: urdf = model.to_urdf()

In [13]: urdf.export_xml('kobuki.urdf')
```

Spawn the model to a running instance of Gazebo

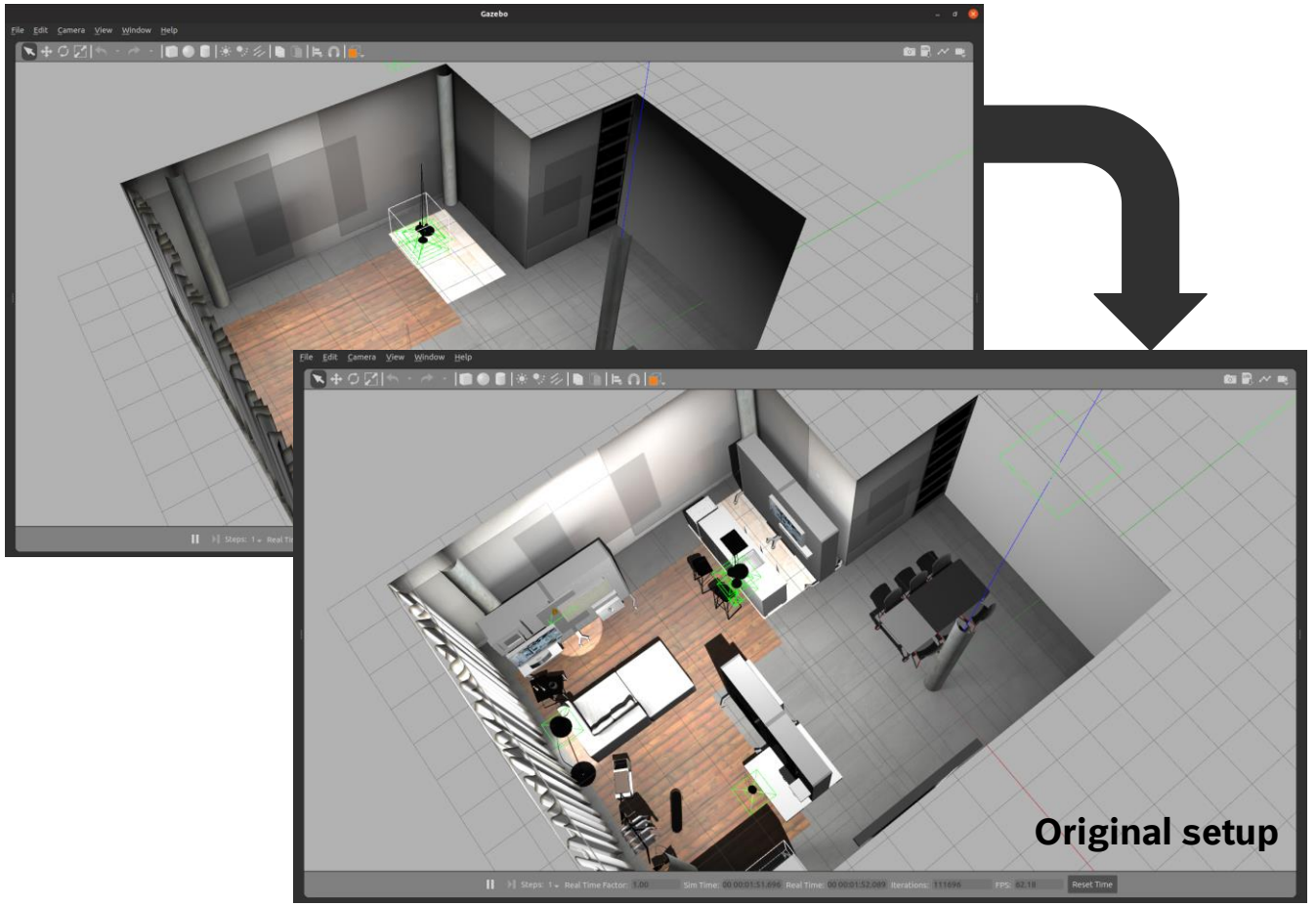
```
In [14]: model.spawn()
Out[14]: True
```



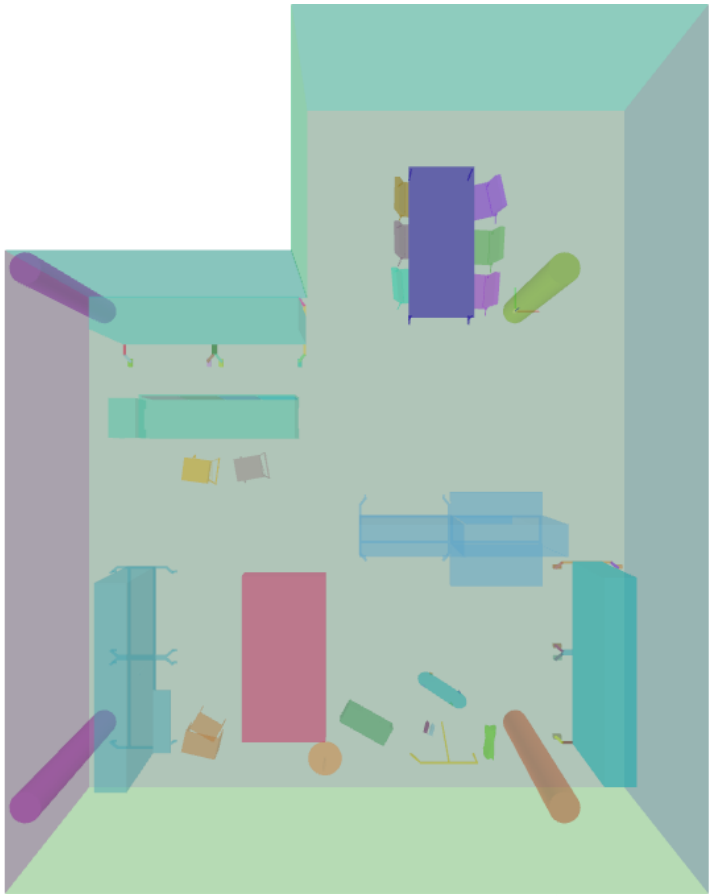
Musa Morena Marcusso Manhães | 31.10.2019
ROSCon 2019 Macau

BOSCH

World generators
Generating world configurations

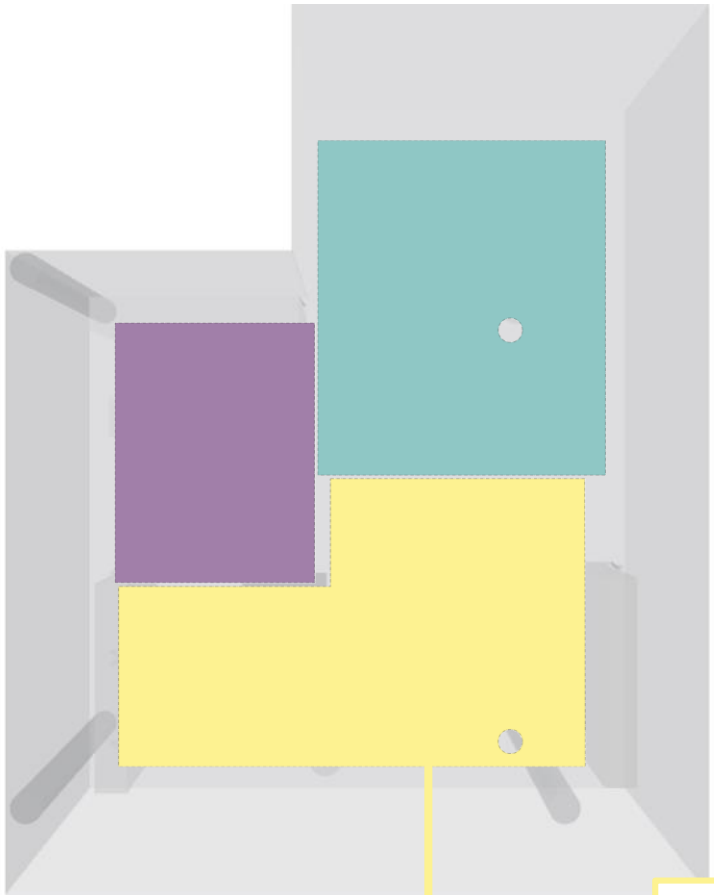


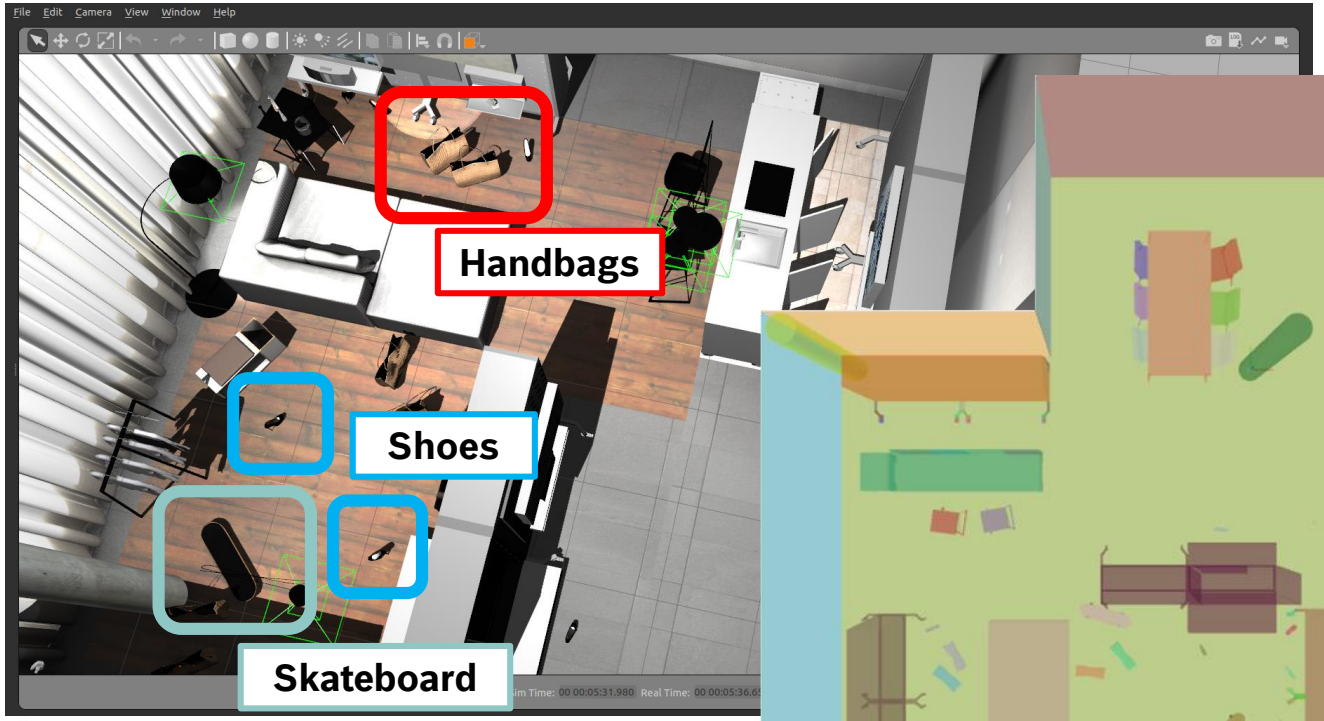
Collision geometries



World generators

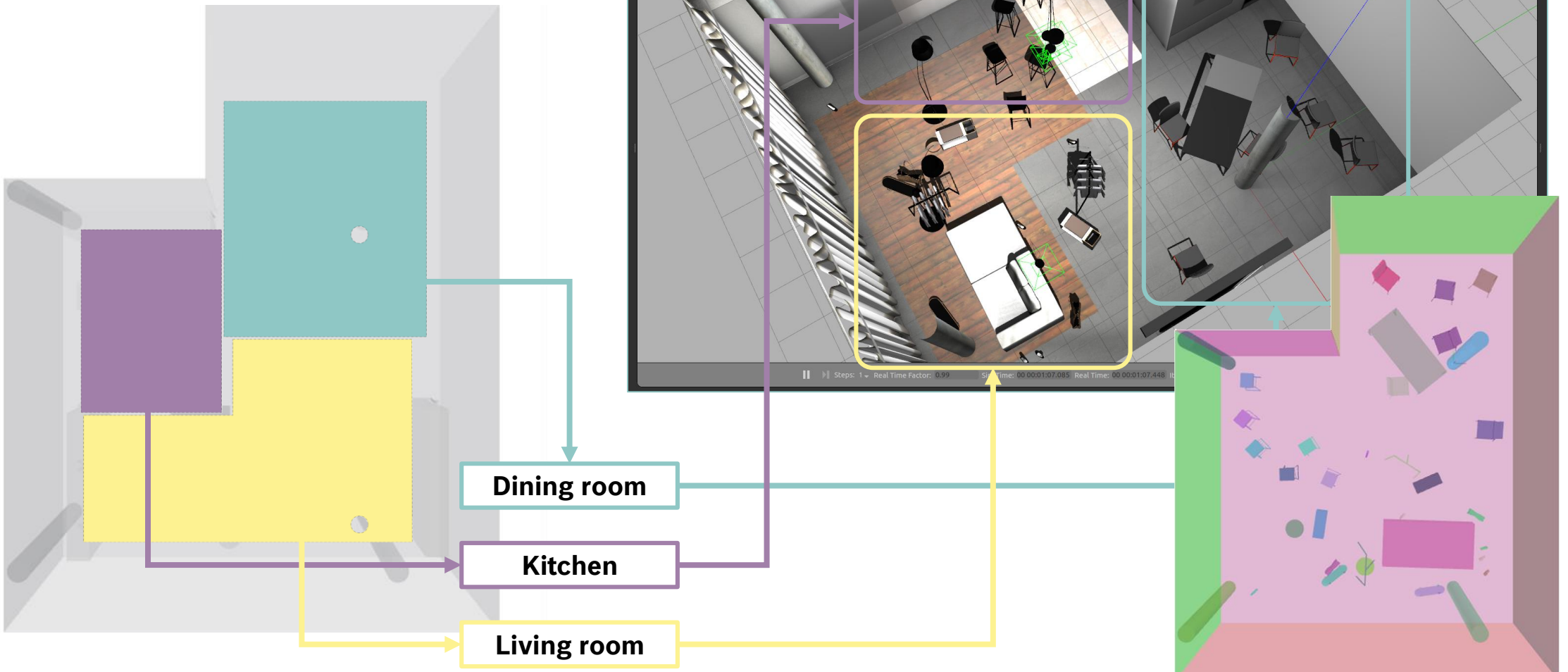
Workspaces and pose randomization with collision checking





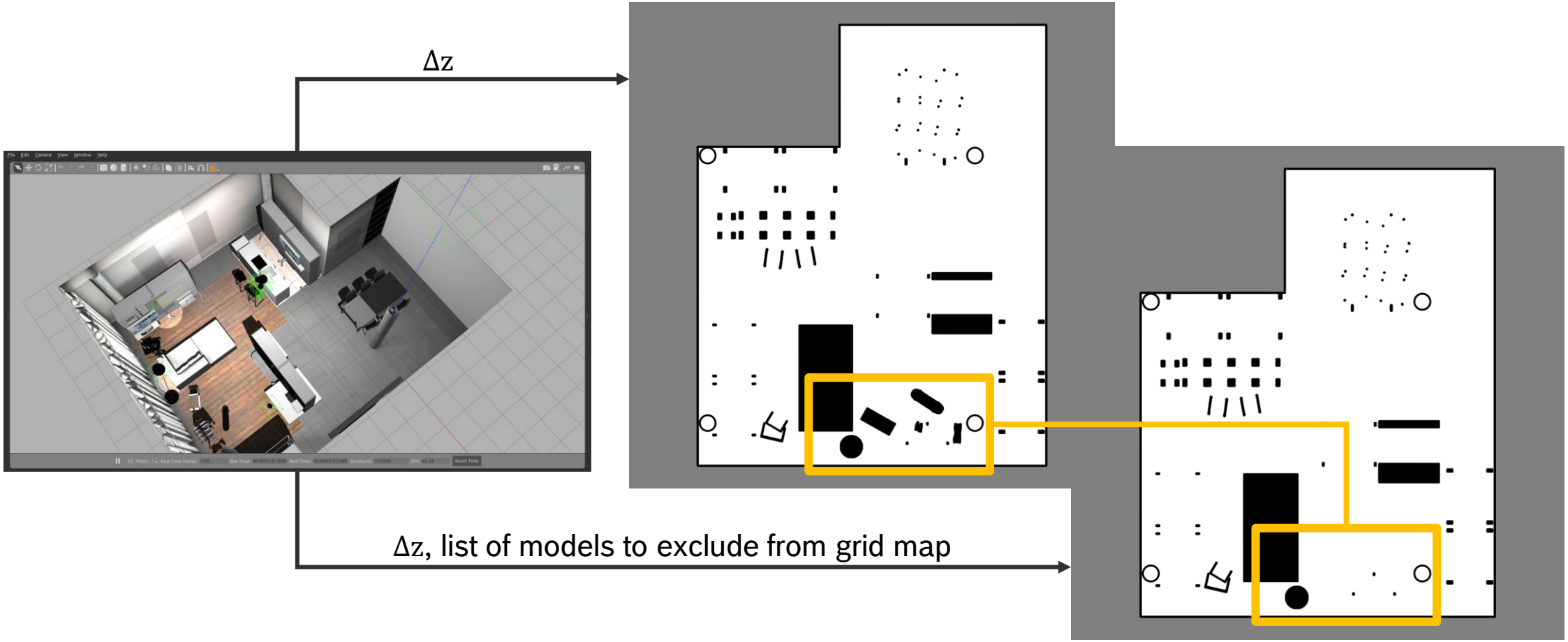
BOSCH

World generators
Pose randomization



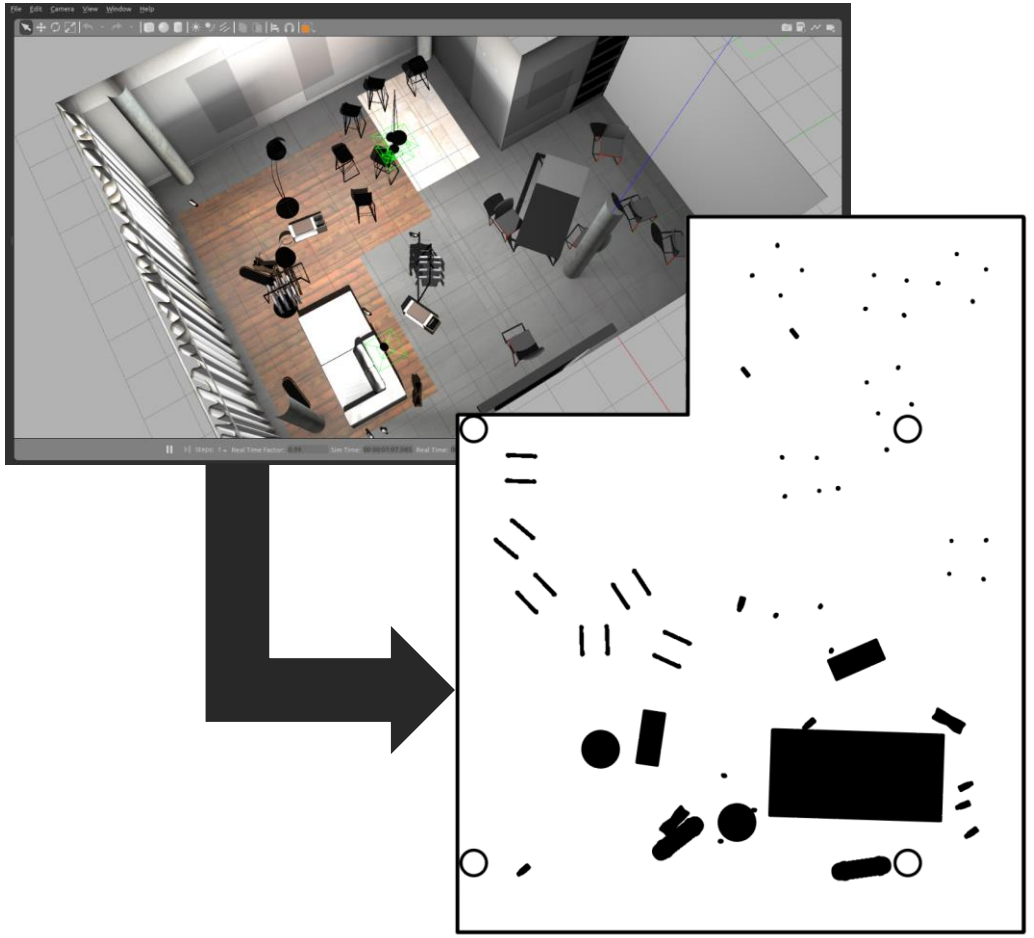
World generators

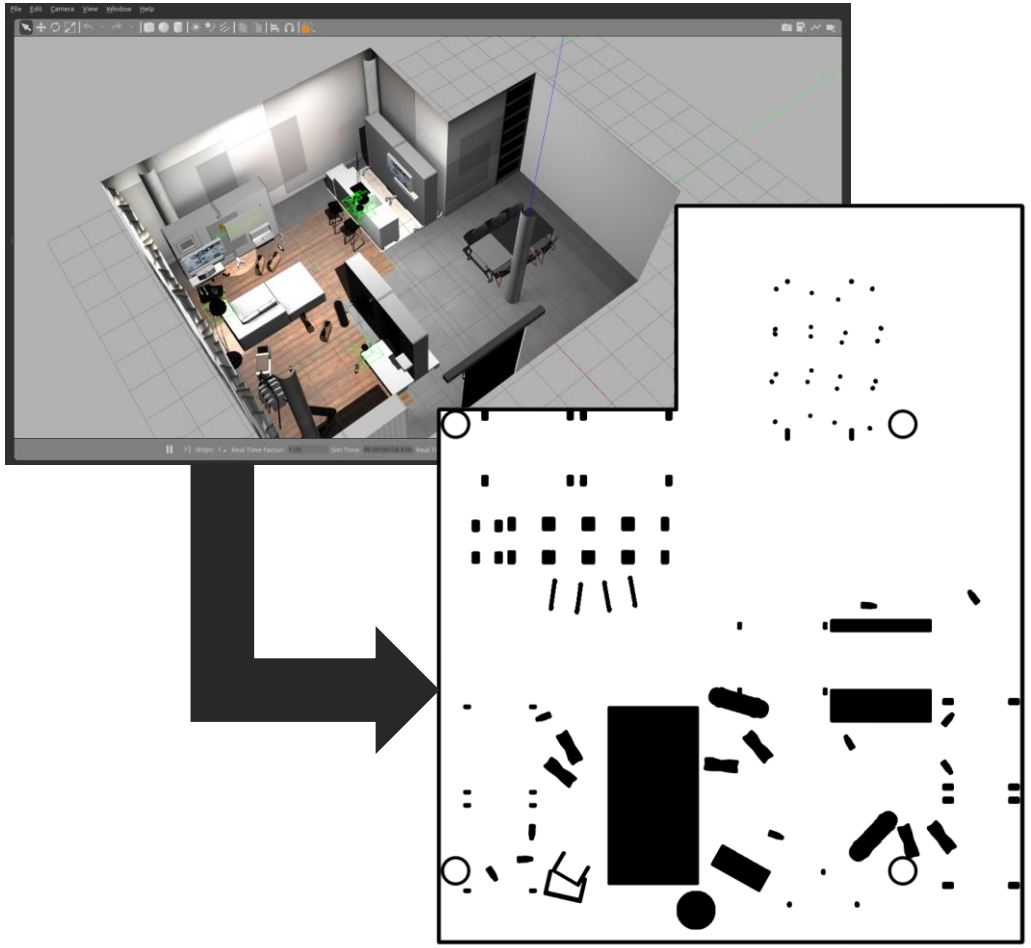
Generating grid maps from Gazebo worlds via ray tracing



World generators

Generating grid maps from Gazebo worlds via ray tracing





BOSCH

TL;DR

TL;DR

- **`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

BOSCH