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
INTRODUCTION
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
    Robot components and structures
    Objects placements

► 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

License: https://creativecommons.org/licenses/by/3.0/
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
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
Inspection of Gazebo models

Inspecting SDF from a `SimulationModel` object

SimulationModel objects can be converted into a SDF object

The SDF elements can be accessed via their Python object abstraction

The SDF elements also be altered and the object again exported into a XML file
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
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 created 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**: cylinder
- **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`

- **full_crate**
  - **1** 
    - **crate**
  - **ball**
  - **cuboid**
  - **cylinder**

At each model_group instance created from the model_group_generator, all model descriptions containing lambda functions will be recomputed.

Multiple instances of ball, cuboid, cylinder generated using the random_pose engine with spatial constraints.

- **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

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

```
1. from pcg_gazebo_simulation import SimulationModel
2. from pcg_gazebo.parsers import parse_sdf
3. from pcg_gazebo.utils import process_jinja_template

4. output_xml = process_jinja_template(robot.sdf.jinja, parameters={
   'namespace': 'kobuki', 'node': 'default'})
5. sdf = parse_sdf(output_xml)
6. model = SimulationModel.from_sdf(sdf.models[0])
7. model.show(mesh_type='visual')
8. model.show(mesh_type='collision')
```

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

```
9. model.to_gazebo_model()
10. cd ${HOME}/gazebo/models/kobuki/
11. ls -l mex_gazebo_models/Models/Abdul
```

Exporting the URDF file for the `kobuki` robot

```
12. urdf = model.to_urdf()
13. urdf.export_xml('kobuki.urdf')
```

Spawn the model to a running instance of Gazebo

```
14. model.spawn()
15. True
```
World generators
Generating world configurations

Collision geometries

Original setup
World generators

Workspaces and pose randomization with collision checking

Living room

Handbags

Shoes

Skateboard
World generators
Pose randomization
World generators
Generating grid maps from Gazebo worlds via ray tracing

$$\Delta z$$, list of models to exclude from grid map
World generators

Generating grid maps from Gazebo worlds via ray tracing
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](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