ABOUT ME

Gianluca Bardaro, PhD student in Robotics

Contacts:

- gianluca.bardaro@polimi.it
- 02 2399 **3565**

Research field:

- Formal approach to robot development
- Robot and robot architecture models
- Robot simulation

GAZEBOSIM AND SDF

ROBOTICS

POLITECNICO
MILANO 1863

WHAT IS A SIMULATION

Simulation is the imitation of the operation of a real-world process or system over time.

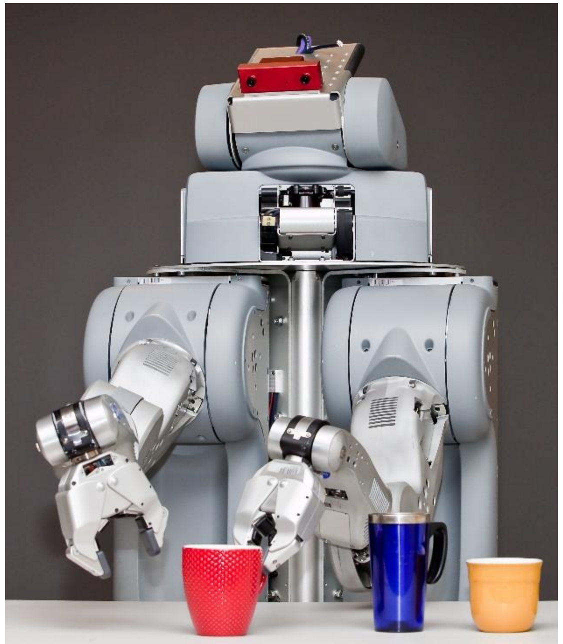
The act of simulating something first requires that a model be developed; this model represents the key characteristics or behaviors/functions of the selected physical or abstract system or process.

The model represents the system itself, whereas the simulation represents the operation of the system over time.

FOR WHAT PURPOSE?

Robots…

- are small and safe
- can be easily tested in the filed
- require real world interactions



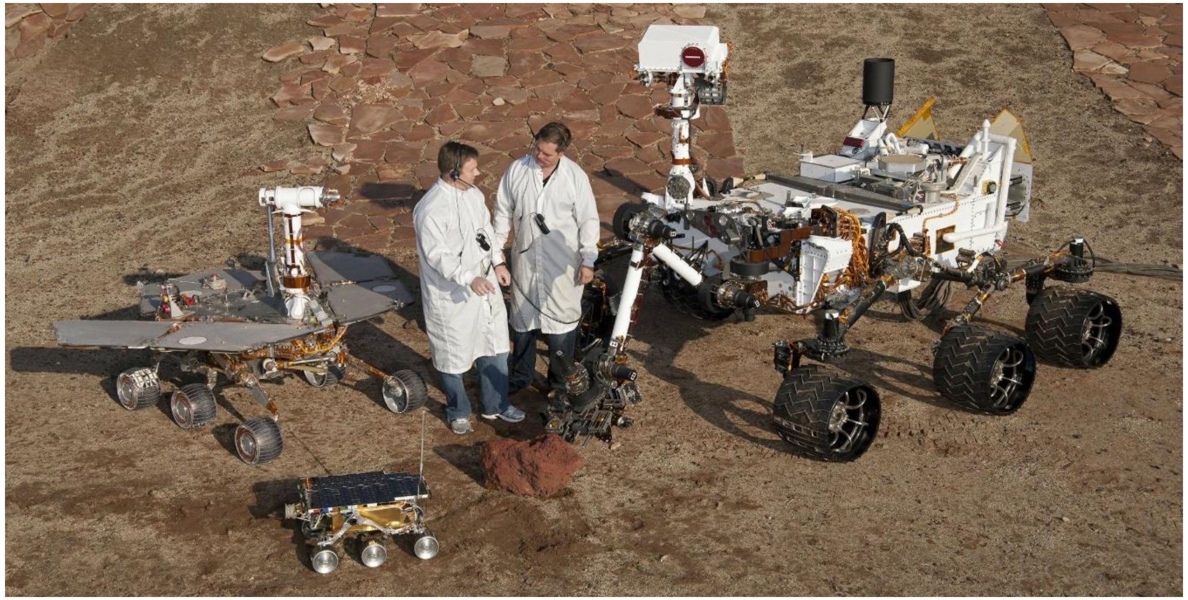
FOR WHAT PURPOSE?

Robots…

- are small and safe
- can be easily tested in the filed
- require real world interactions

But robots…

- can be big and dangerous
- need to be tested in some specific conditions
- have a behavior based on software which is prone to bugs



FOR WHAT PURPOSE?

Robots…

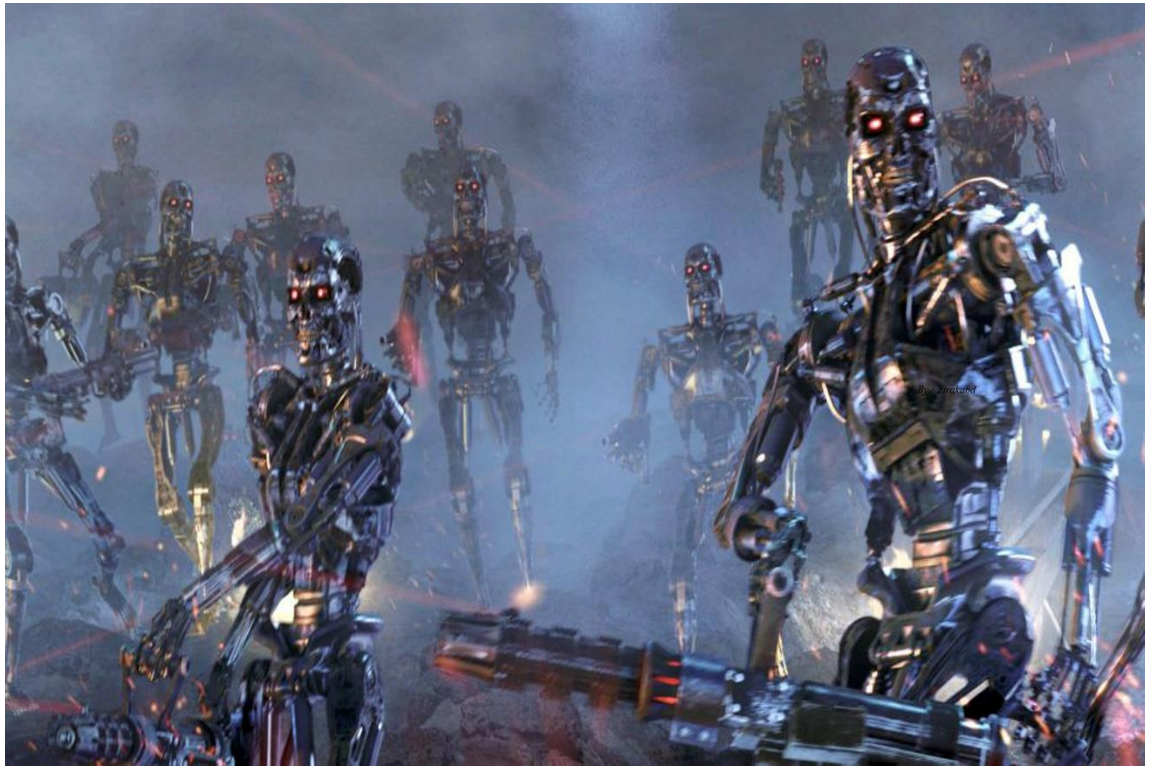
- are small and safe
- can be easily tested in the filed
- require real world interactions

But robots…

- can be big and dangerous
- need to be tested in some specific conditions
- have a behavior based on software which is prone to bugs

Moreover…

- as engineers we know that everything should be based on a well detailed project and should be tested and verified before any real application



Remember to test and simulate, it can save your life!

ROBOT SIMULATORS

Important features:

- Real world interaction
- Physics simulation
- Sensors simulation
- Simplicity in building robot models
- Flexibility
- Integration with robot frameworks

ROBOT SIMULATORS

GAZEBO

WHY GAZEBO?

A bit of history…

- ROS become the standard de facto in robot software development
- Gazebo used to be THE simulator to work with ROS…because it was the only one
- ROS swallowed Gazebo and froze its development
- Gazebo detach itself from ROS and regain its independence maintaining a good integration with ROS
- Now at version 7.0 it has some interesting features

Main features of Gazebo

- Dynamic simulation based on various physics engines (ODE, Bullet, Simbody and DART)
- Sensors (with noise) simulation
- Plugin to customize robots, sensors and the environment
- Realistic rendering of the environment and the robots
- Library of robot models and model editor
- ROS integration

Advanced features

- Remote & cloud simulation
- Open source

SYSTEM REQUIREMENTS AND INSTALLATION

Gazebo is currently best used on Ubuntu, a flavor of Linux. You will need a computer that has:

- A dedicated GPU
- A CPU that is at least an Intel i5, or equivalent
- At least 500MB of free disk space
- Ubuntu Trusty or later installed

Versions used in this course:

Ubuntu 15.04 (Vivid Vervet) & Gazebo 7.0

Assuming you already have a working installation of Ubuntu, download this:

http://osrf-distributions.s3.amazonaws.com/gazebo/gazebo7_install.sh

Go in the download folder, open a terminal there and write the command:

```
sh gazebo7_install.sh
```

That's it, to run gazebo open a terminal and write:

```
gazebo
```

PREDICTABLE QUESTIONS

What kind of existing knowledge do I need to use Gazebo?

Can I use a different/newer/older version of Gazebo?

Can I use a different/newer/older version of Ubuntu?

Can I use a different Linux distribution?

Can I use Windows/OS X?

Can I use a virtual machine?

Is the use of the simulator required for the project?

I know Gazebo and I hate it! Can I use another simulator?

CUSTOMIZATION

What kind of customization are we looking for in a simulator?

- Modifying existing robot or sensor models
- Building our own robot or sensor models
- Modifying the behavior of existing robot models
- Controlling and defining a behavior for our own robot models
- Creating specific environment compatible with our experiments
- Integrating the simulation with the user and the robot architecture

CREATING AND MODIFYING A MODEL

Using the model editor

Newer versions of Gazebo provide tools to create and modify models directly form the user interface

Create object and change their shape or position using graphical tools

Nice little windows to customize physical and geometrical parameters

Easily connect two object with a joint

Let's see it in action!

Using simulation description format (SDF)

SDF is an evolution of the unified robot description format (URDF)

An XML file format that describes environments, objects and robots for robotic simulation

Hierarchical and well defined

"Compact" description of a complete simulated world

Sounds complex and boring, but it's powerful and necessary

SIMULATION DESCRIPTION FORMAT

As any XML file is composed by tags, but differently from some XML files the structure is quite simple

Tag structure:

- **sdf**
  - **world**
  - **model**
  - **actor**
  - **light**

```
<?xml version='1.0'?>
<sdf version='1.6'>
  <world name='default'>
    ...
  </world>
</sdf>
```

```
<?xml version='1.0'?>
<sdf version='1.6'>
  <actor name='act'>
    ...
  </actor>
</sdf>
```

```
<?xml version='1.0'?>
<sdf version='1.6'>
  <model name='model'>
    ...
  </model>
</sdf>
```

```
<?xml version='1.0'?>
<sdf version='1.6'>
  <light name='light'>
    ...
  </light>
</sdf>
```

SDF/ WORLD

The **`world`** represent everything inside the simulation ready to be simulated

- Most important available child tags are: **`scene`**, **`light`**, **`model`**, **`actor`**, **`plugin`**, **`gui`**, **`include`**
- Physics related child tags: **`physics`**, **`gravity`**, **`magnetic_field`**, **`spherical_coordinates`**
- More child tags: **`audio`**, **`atmosphere`**, **`wind`**, **`road`**, **`state`**, **`population`**

`sdf/(light, model, actor)` VS **`world/(light, model, actor)`**

- SDF is an evolution and a substitute of URDF, so it must maintain the same functionalities
- A valid SDF file may contain only a single or a list object and act as an "archive"
- Object defined outside the tag **`world`** can be used with the tag **`include`**

SDF/MODEL

What is a **`model`**?

- It's just a container for the element of the robot, its only attribute is the **`name`**
- It's composed by links and joints, or even by other models. It is possible to use the **`include`** tag to include previously defined models. Given this recursive structure a model may become really complex.

What is a **`link`**?

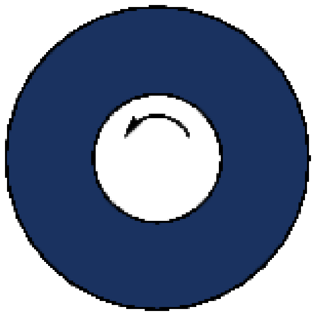
- It's any rigid element of the robot and there is no limit on the number of links, must be child of the **`model`**
- It has physical and visual properties and collisions

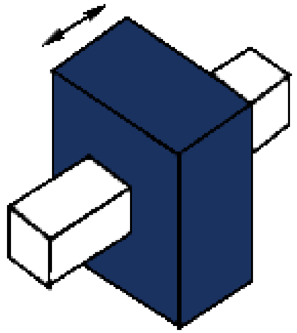
What is a **`joint`**?

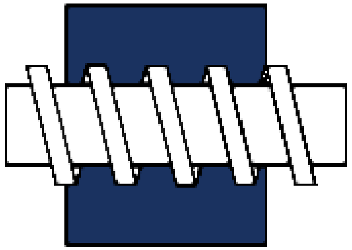
- It is used to connect two links together with kinematic and dynamics properties
- Various type of joint are available depending on the behavior of the links (revolute, spherical, prismatic, …)
- Always defined between a parent link and a child link

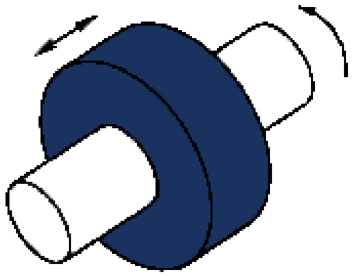
`pose` and **`frame`** are two key elements of each of these component. Together they define the position and orientation of each element with respect to another. The correct use of reference frame can vastly simplify the construction of any complex robot.

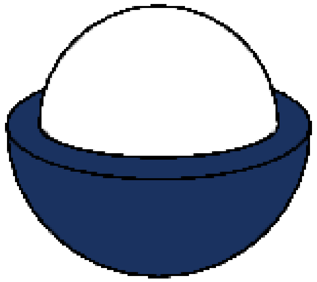
ABOUT JOINTS

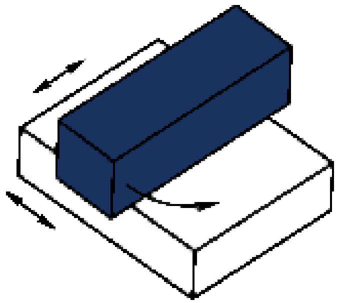

Revolute (1 DoF)


Prismatic (1 DoF)


Screw (1 DoF)


Cilindric (2 DoF)


Sphere (3 DoF)


Planar (3 DoF)

MORE ABOUT MODELS

Models have complex structures may include various component to improve they appearance and behavior.

A specific folder structure is used to define a model:

- `.gazebo/models/my_model`: our model folder inside the main Gazebo folder
  - `model.config`: Meta-data about the model
  - `model.sdf`: SDF description of the model
  - `meshes`: a directory for all COLLADA and STL files
  - `materials/texture` & `material/scripts`: texture images and material scripts
  - `plugins`: a directory for all the code used to define the behavior of the model

SDF DEFINITION

Looks pretty simple, is this all?! Of course not

You can find the complete description of SDF here:

http://sdformat.org/spec

LET'S SEE AN EXAMPLE

Create a model directory

```
mkdir -p ~/.gazebo/models/willy2
```

Create the configuration file

```
gedit ~/.gazebo/models/willy2/model.config
```

Fill the configuration file

```
<?xml version="1.0"?>
<model>
  <name>willy2</name>
  <version>1.0</version>
  <sdf version='1.4'>willy2.sdf</sdf>

  <author>
   <name>Builder Bob</name>
   <email>robert.builder@polimi.it</email>
  </author>

  <description>
    A two wheeled robot.
  </description>
</model>
```

Create the sdf file

```
gedit ~/.gazebo/models/willy2/model.sdf
```

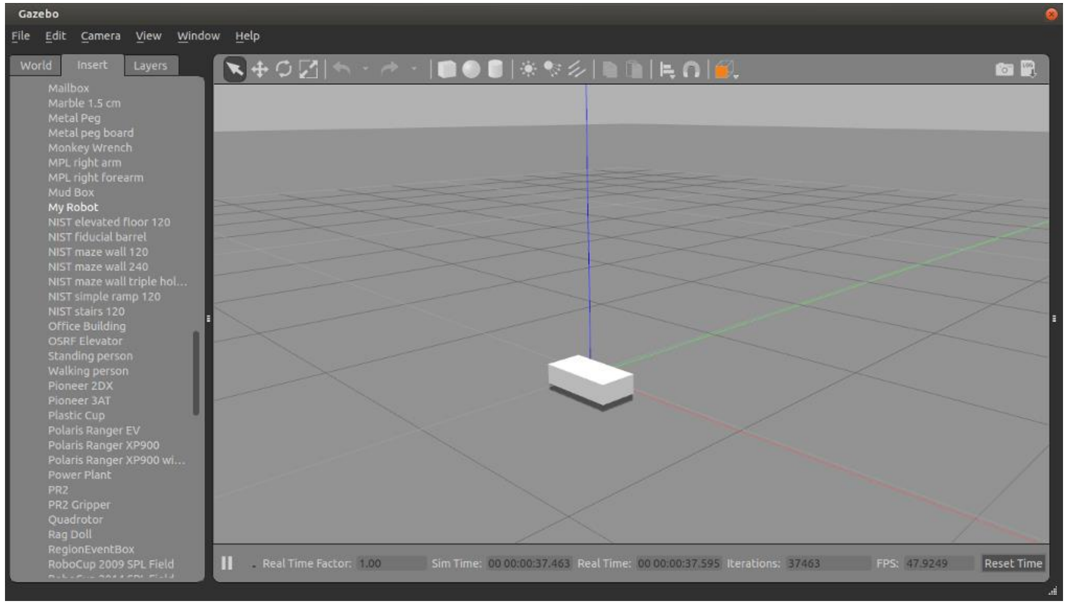
Fill the sdf file

```
<?xml version='1.0'?>
<sdf version='1.4'>
  <model name="my_robot">
  </model>
</sdf>
```

BUILDING THE ROBOT

It's important to build the robot progressively, start with a simple base and add up the other elements

The result we want it's something like this:



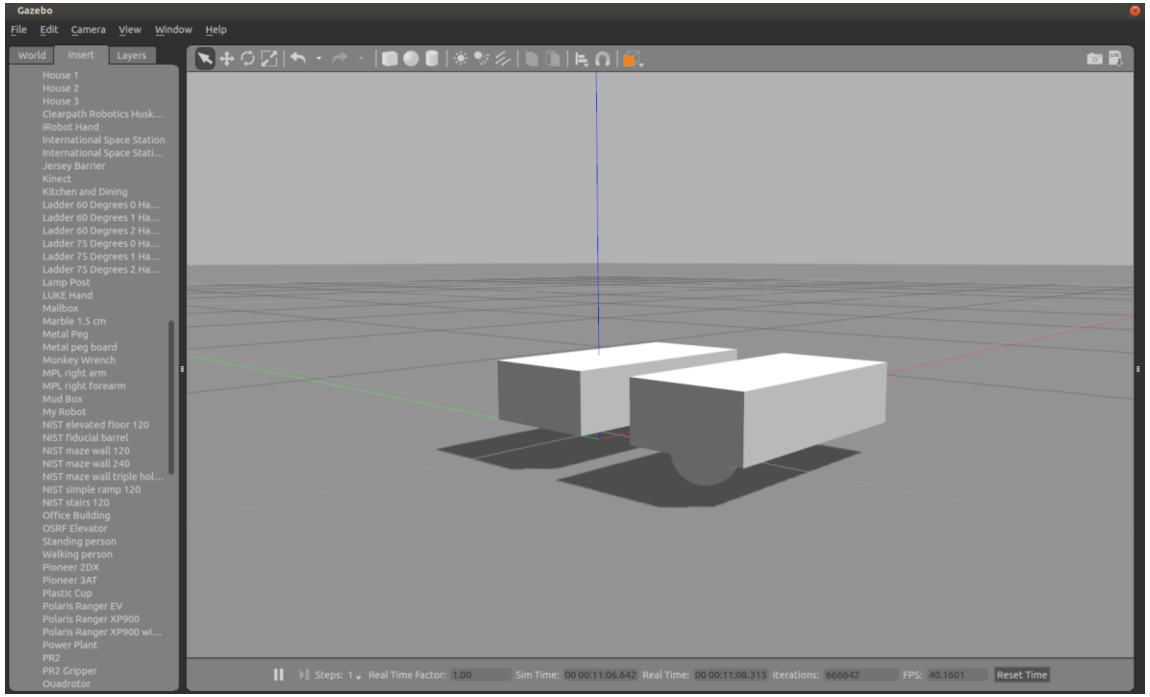
For this we need only a simple `link` shaped like a box

```
<link name='chassis'>
<pose>0 0 .1 0 0 0</pose>
  <collision name='collision'>
    <geometry>
      <box>
        <size>.4 .2 .1</size>
      </box>
    </geometry>
  </collision>
  <visual name='visual'>
    <geometry>
      <box>
        <size>.4 .2 .1</size>
      </box>
    </geometry>
  </visual>
</link>
```

ADDING A CASTER

A caster is a simple wheel with no constraint, it's not connected to the body of the robot using a joint, it's used only to sustain the weight.

Since there is no joint we can add it to the base using a second `collision` without defining a new link.



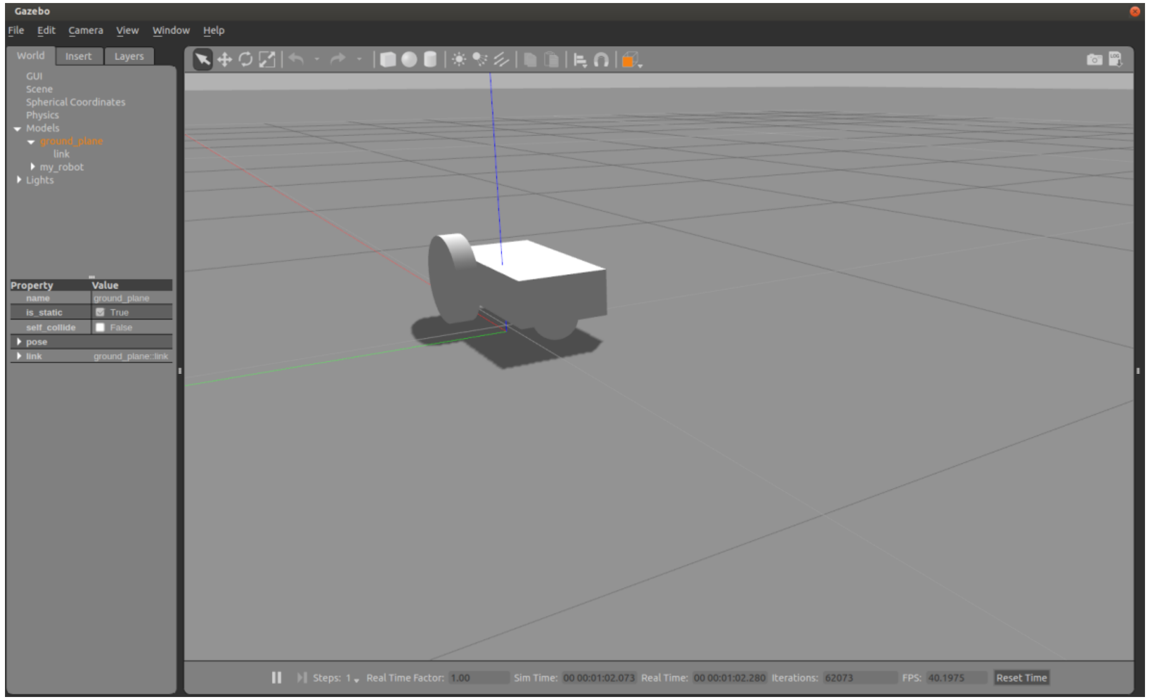
ADDING A CASTER

```
<collision name='caster_collision'>
  <pose>-0.15 0 -0.05000</pose>
  <geometry>
    <sphere>
      <radius>.05</radius>
    </sphere>
  </geometry>

  <surface>
    <friction>
      <ode>
        <mu>0</mu>
        <mu2>0</mu2>
        <slip1>1.0</slip1>
        <slip2>1.0</slip2>
      </ode>
    </friction>
  </surface>
</collision>

<visual name='caster_visual'>
  <pose>-0.15 0 -0.05000</pose>
  <geometry>
    <sphere>
      <radius>.05</radius>
    </sphere>
  </geometry>
</visual>
```

ADDING THE WHEELS

The two wheels are real wheels, not like the caster. They are the source of the movement of the robot and they will be controlled.

The wheels are defined as links and are connected to the body of the robot using joints.



ADDING THE WHEELS

```
<link name="left_wheel">
  <pose>0.1 0.13 0.1 0 1.5707 1.5707</pose>
  <collision name="collision">
    <geometry>
      <cylinder>
        <radius>.1</radius>
        <length>.05</length>
      </cylinder>
    </geometry>
  </collision>
  <visual name="visual">
    <geometry>
      <cylinder>
        <radius>.1</radius>
        <length>.05</length>
      </cylinder>
    </geometry>
  </visual>
</link>

<link name="right_wheel">
    <pose>0.1 -0.13 0.1 0 1.5707 1.5707</pose>
    ...
</link>
```

ADDING THE JOINTS

We use joints to connect the wheels to the chassis.

Since the wheels are constrained in any direction of movement except for the rotation around an axis we use a revolute joint.

```
<joint type="revolute" name="left_wheel_hinge">
  <pose>0 0 -0.03 0 0 0</pose>
  <child>left_wheel</child>
  <parent>chassis</parent>
  <axis>
    <xyz>0 1 0</xyz>
  </axis>
</joint>
<joint type="revolute" name="right_wheel_hinge">
  <pose>0 0 0.03 0 0 0</pose>
  <child>right_wheel</child>
  ...
</joint>
```

THE ROBOT IS COMPLETE