ABOUT ME

Mentasti Simone, PhD student in CS and Mechanics

Contacts:

simone.mentasti@polimi.it

Research field:

Human vehicle interaction in autonomous connected cars

Slide and example link:

https://goo.gl/GonArW

Logistics:

The classroom has no power source…
Is it a problem?

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.

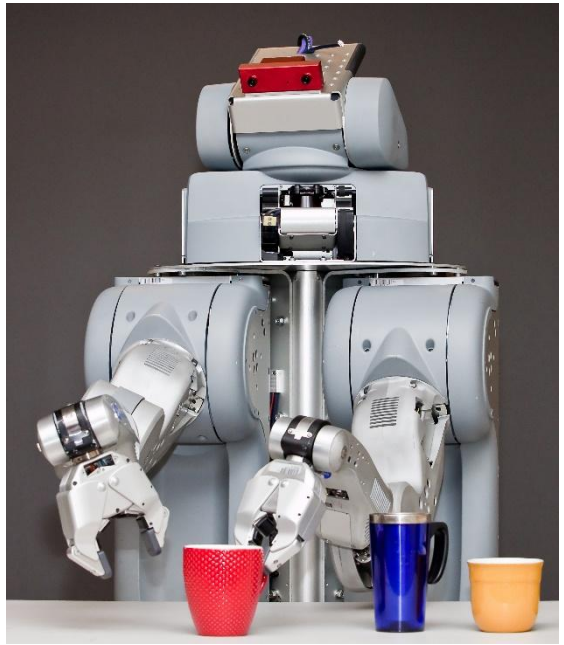
The act of simulating something first requires to develop a model; this model represents the key characteristics or behaviors/functions of the selected physical or abstract system or process.

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



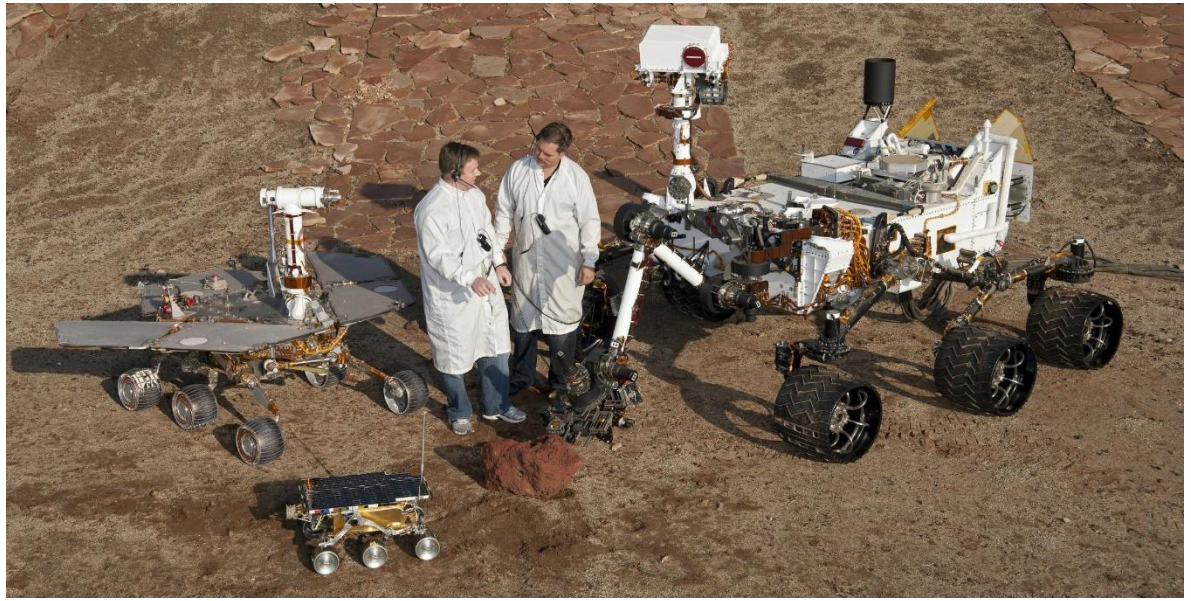
FOR WHAT PURPOSE?

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...

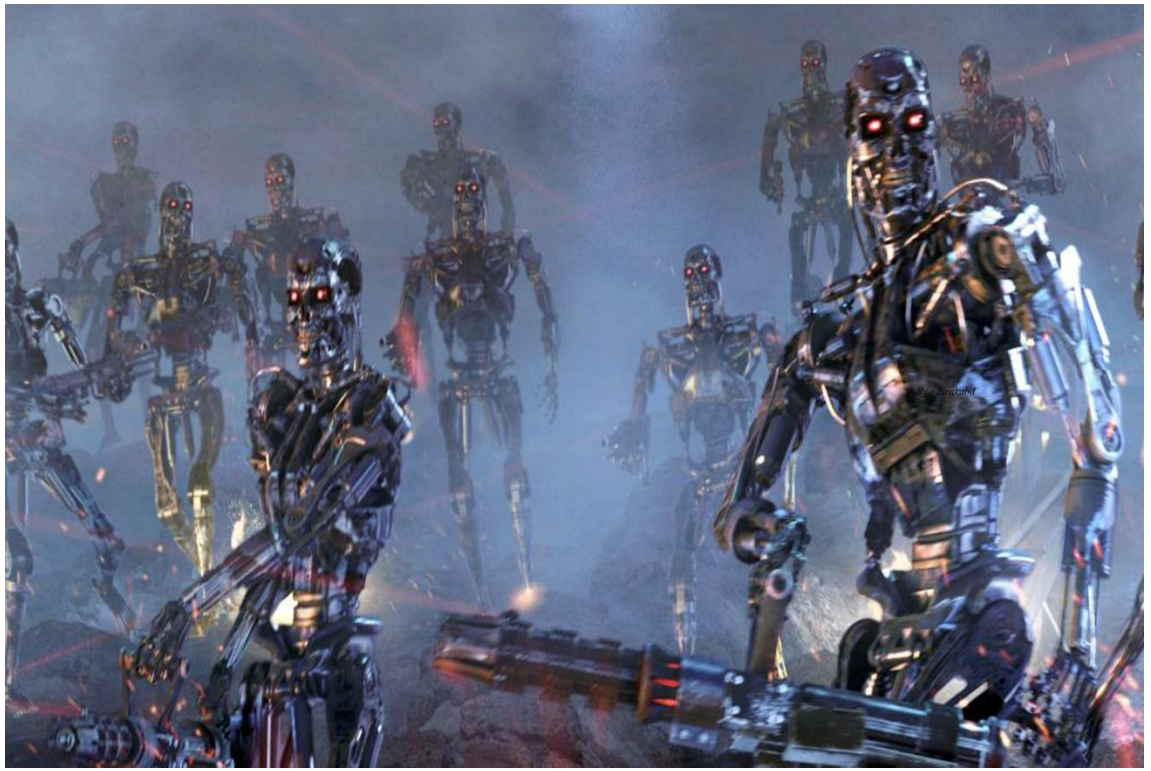
- are small and safe
- can be easily tested in the filed
- require real world interactions

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

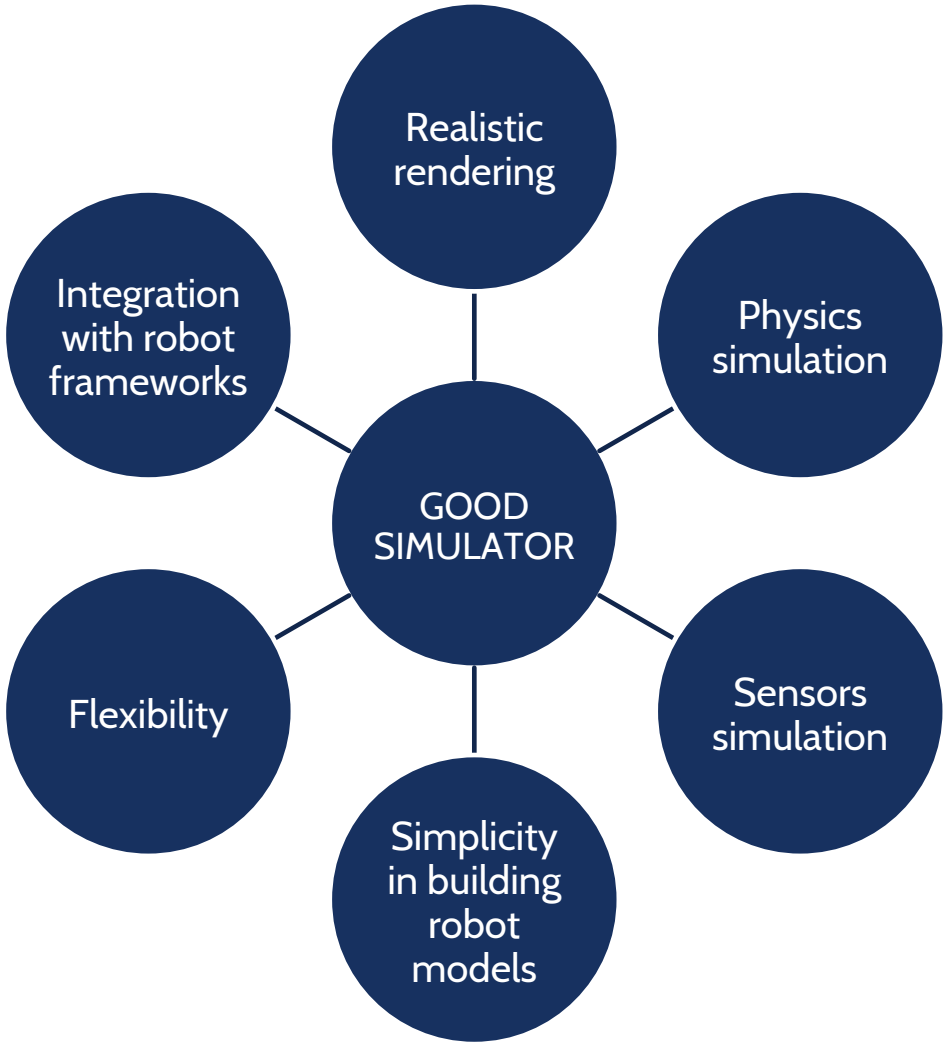
Moreover...

- 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



v-rep

GAZEBO

ROBOT SIMULATORS

Stage

2D (player project)vs 3D (Gazebo)

Different programming languages (C++, Python, LUA, Ruby)

Different modeller (internal like Gazebo/esternal like Morse using Blender)

Different rendering engine (OpenGL, Blender, Java, Ogre)

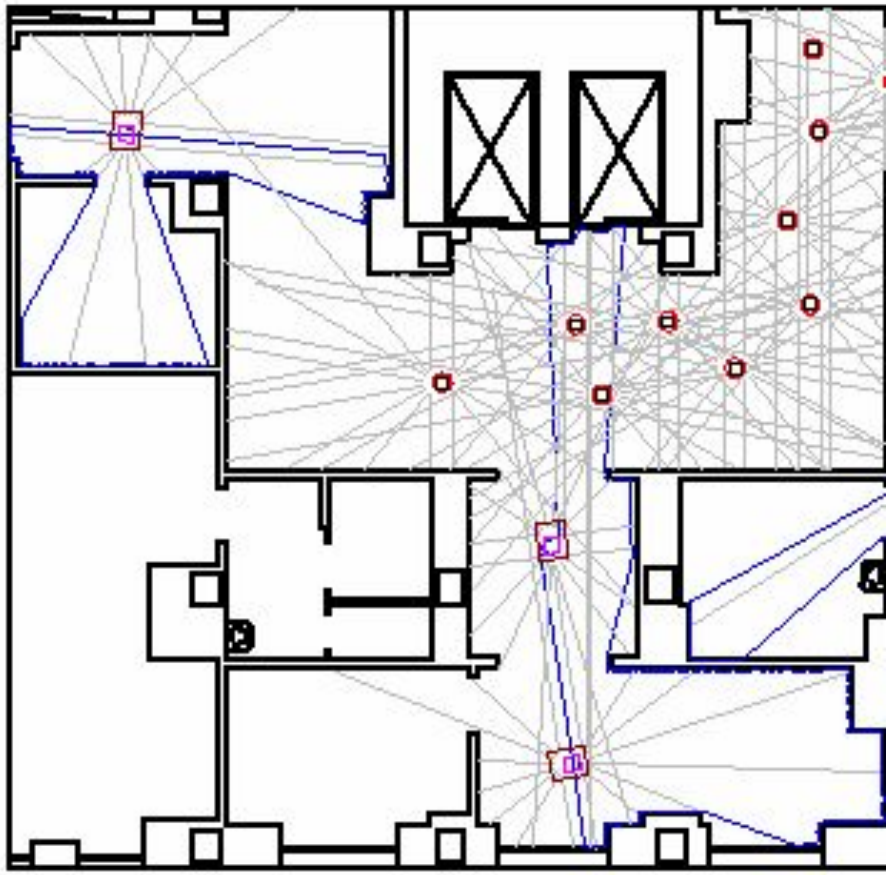
Different sets of sensors (IMU, GPS, Collision, Laser, Cameras, ...)

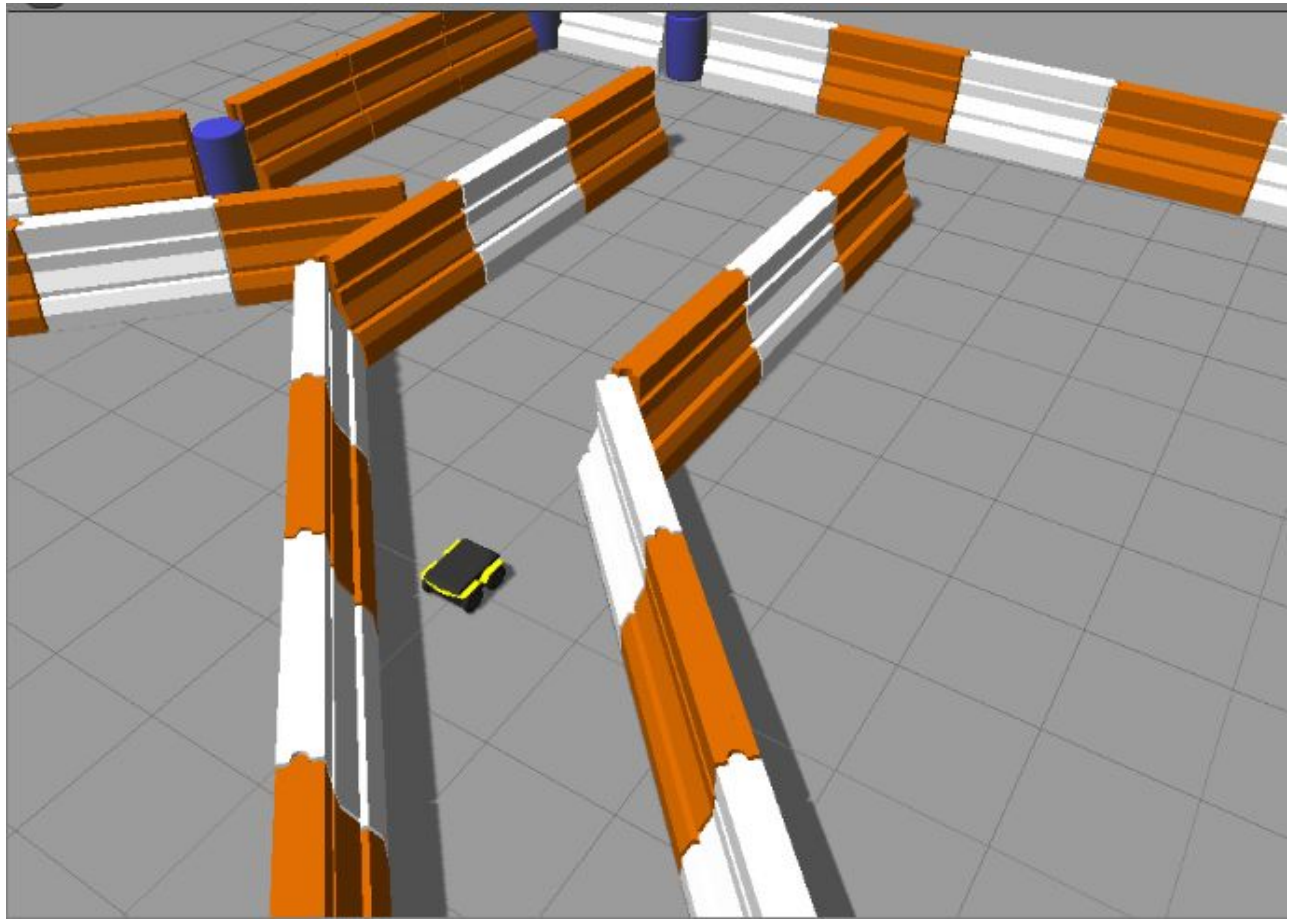
RoboDK
ROBOT DEVELOPMENT KIT

ENERGID
Robotics and Machine Vision

GAZEBO

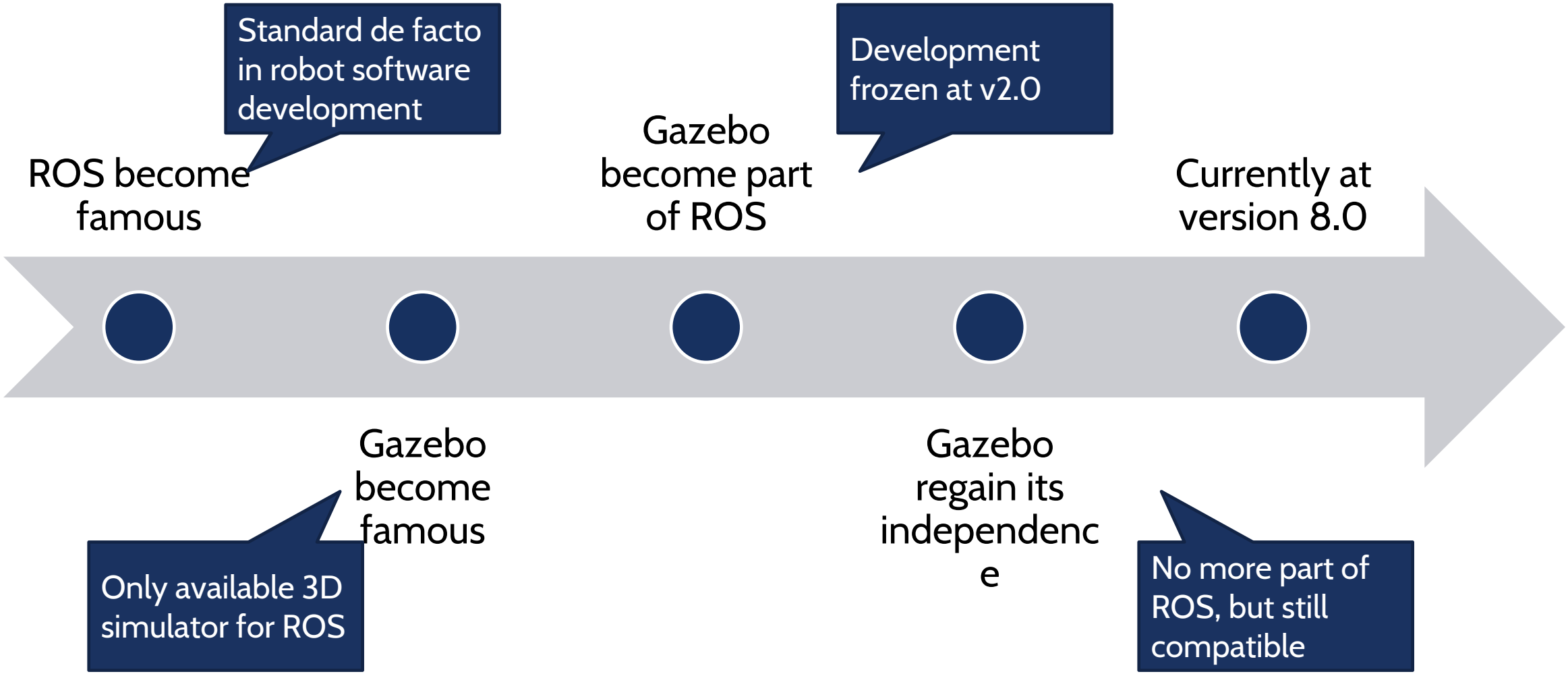
ROBOT SIMULATORS





ROBOT SIMULATORS

BACK IN THE DAY...



WHY GAZEBO?

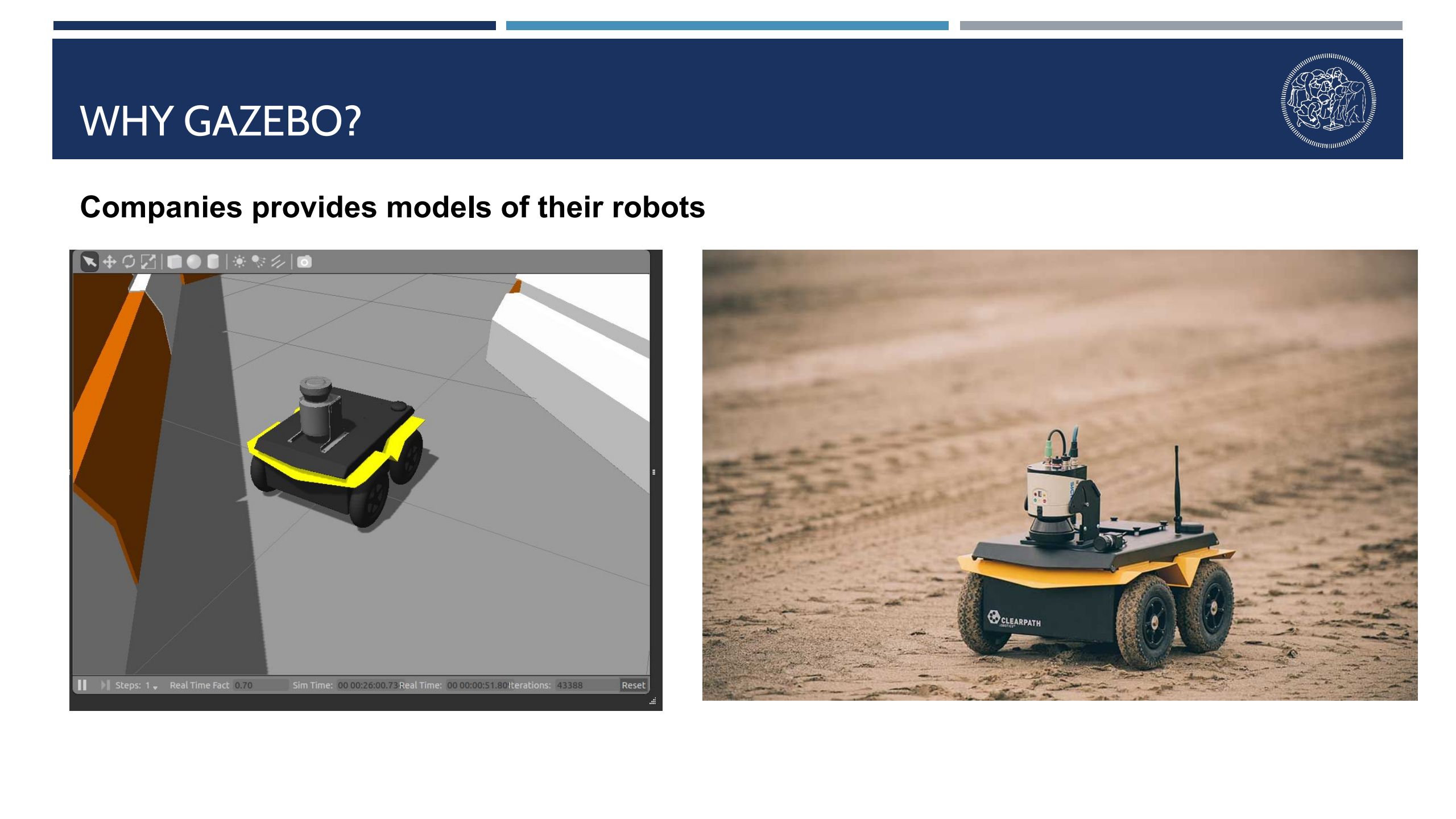
Main features of Gazebo

- 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
- ROS integration

Advanced features

- Remote & cloud simulation
- Open source

WHY GAZEBO?



Companies provides models of their robots

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

SYSTEM REQUIREMENTS

Gazebo is currently best used on Ubuntu.

I strongly suggest a computer with:

- A dedicated GPU
- Any modern CPU
- At least 500MB of free disk space
- Ubuntu Xenial

Versions used in this course:

- Ubuntu **16.04 LTS** (Xenial Xerus) & Gazebo **7.0**

INSTALLATION

In a working installation of Ubuntu 16.04:

$ sudo apt-get update

$ sudo apt-get install gazebo7

To run Gazebo:

$ gazebo

PREDICTABLE QUESTIONS

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

Can I use a different/newer/older version of Gazebo? YES (5.0/6.0/8.0)

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

Can I use a different Linux distribution? YES 

Can I use Windows/OS X? NO 

Can I use a virtual machine? YES

Is the use of the simulator required for the project? YES

I know Gazebo and I hate it! Can I use another simulator? NO

Architecture

Separation of physics and visualization

server: physics and sensor generation (**$ gzserver**)
client: visualization and user interface (**$ gzclient**)

Socket communication

Protobuf provides message passing

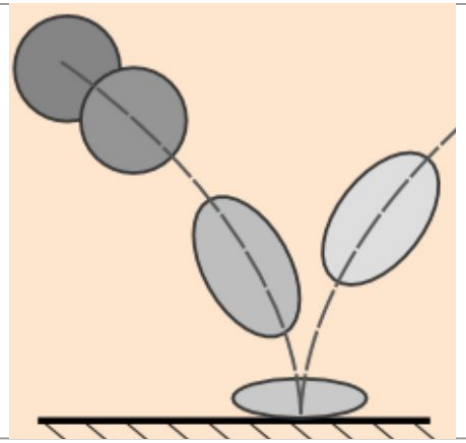
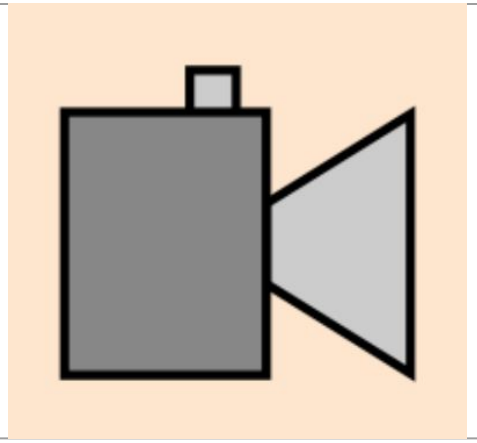
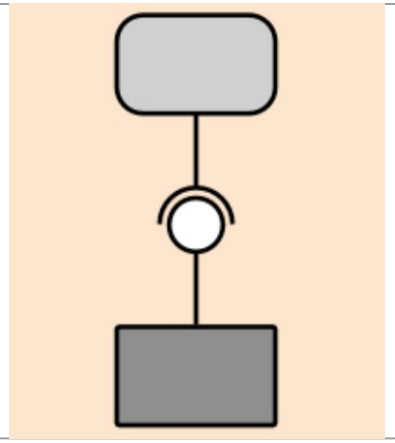
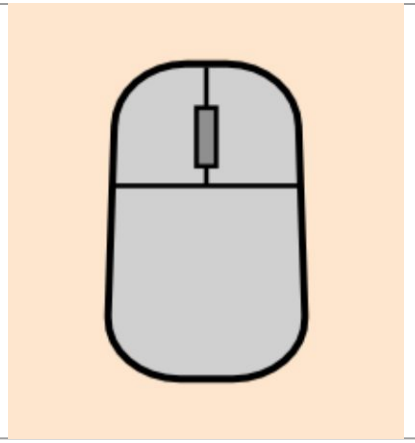
Plugin interface

Control any aspect of simulation

Simulation Description Format (SDF)

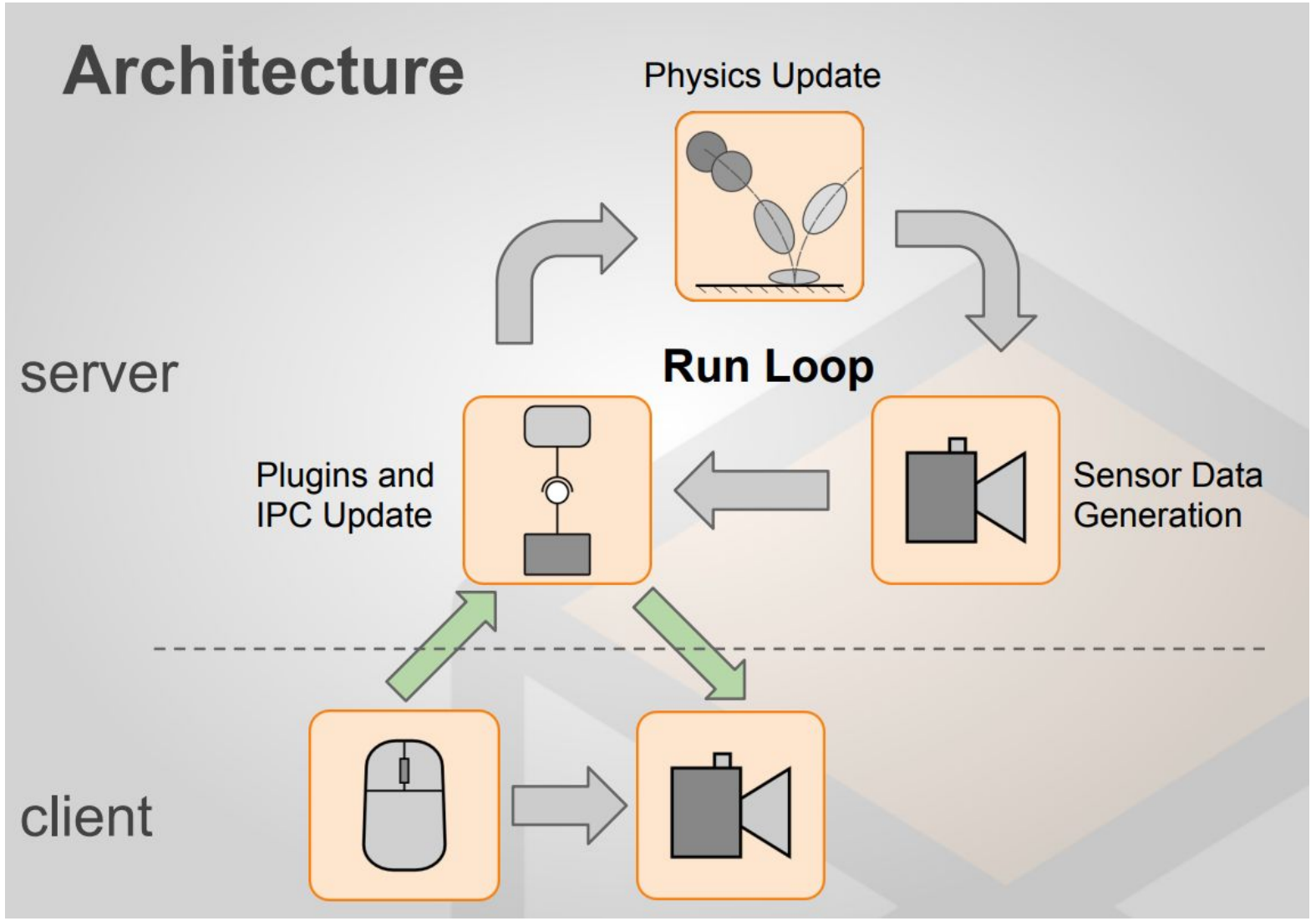
XML based format for worlds and models

Architecture

Physics	Rendering	Interfaces	User Interfaces
			
Rigid Body dinamics	OpenGL	Plugins and IPC	GUI
ODE Bullet ...	OGRE	Google Protobuf Boost ASIO	QT CEGUI

Credits:
Nate Koenig

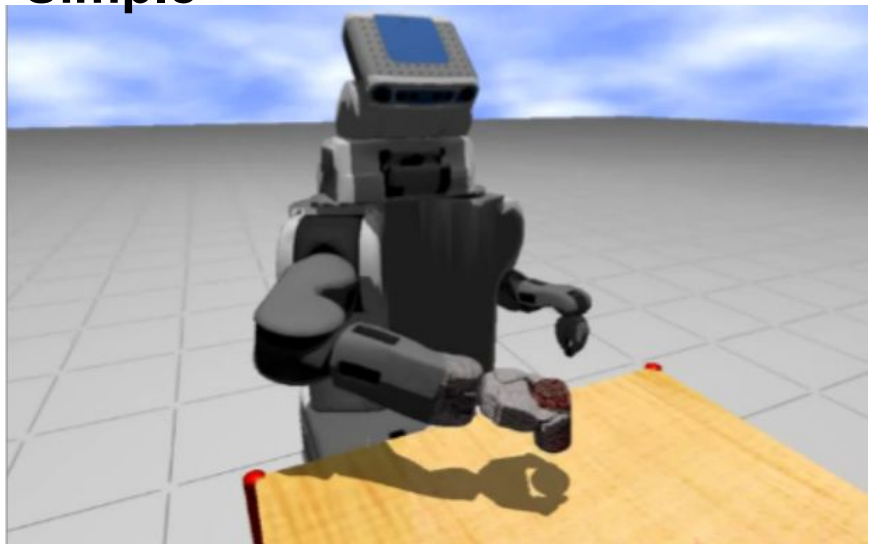
Architecture



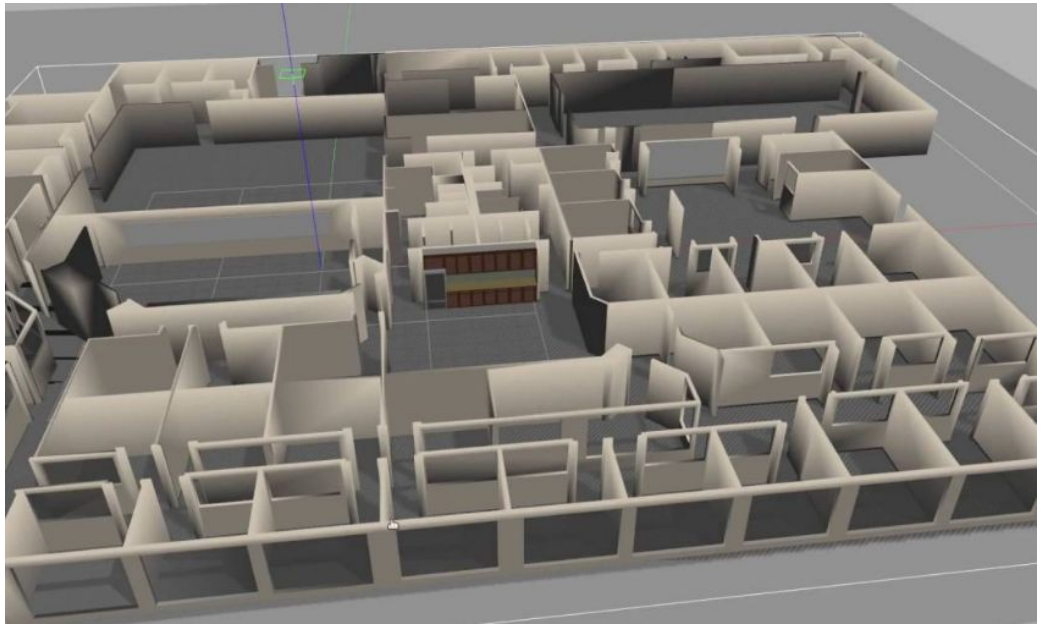
Credits:
Nate Koenig

Enviroments

Simple



Indoor

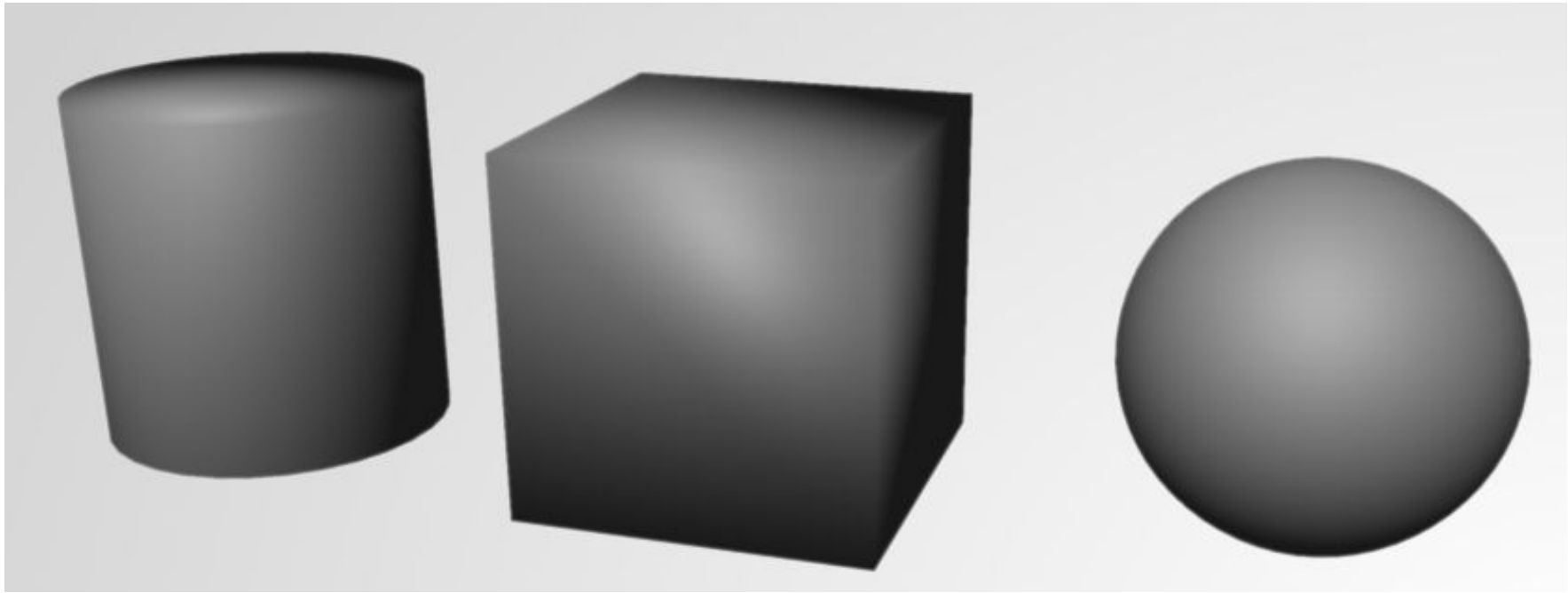


Outdoor

Credits:
Nate Koenig

Creating Enviroments

Built into Gazebo

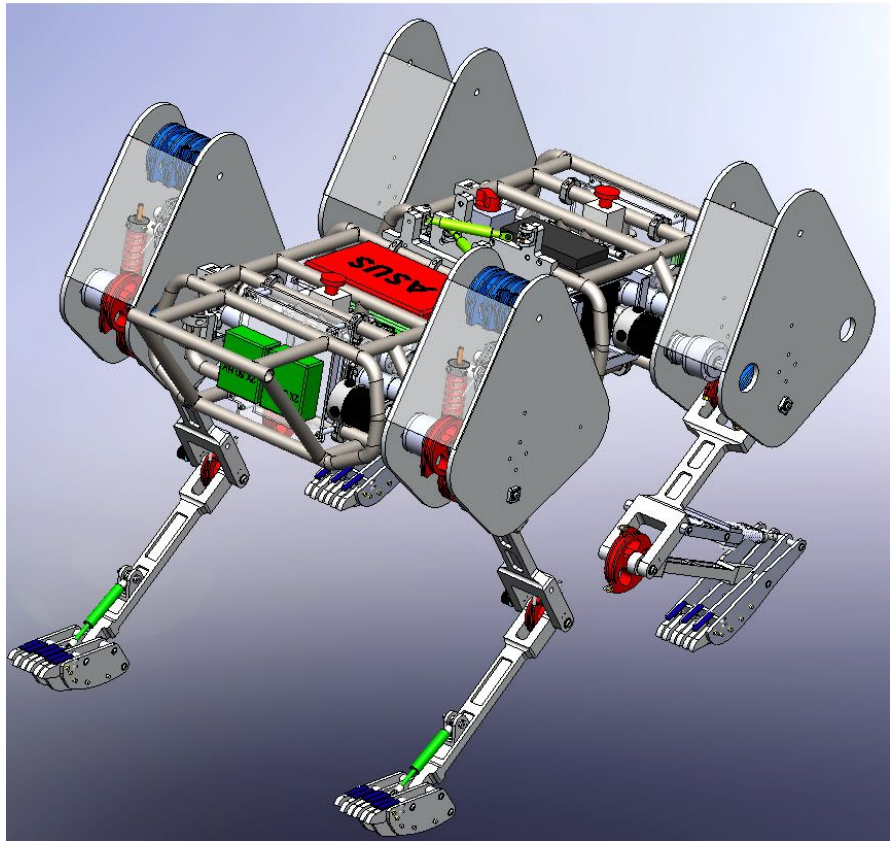


Creating Enviroments

Use external tools

Blender

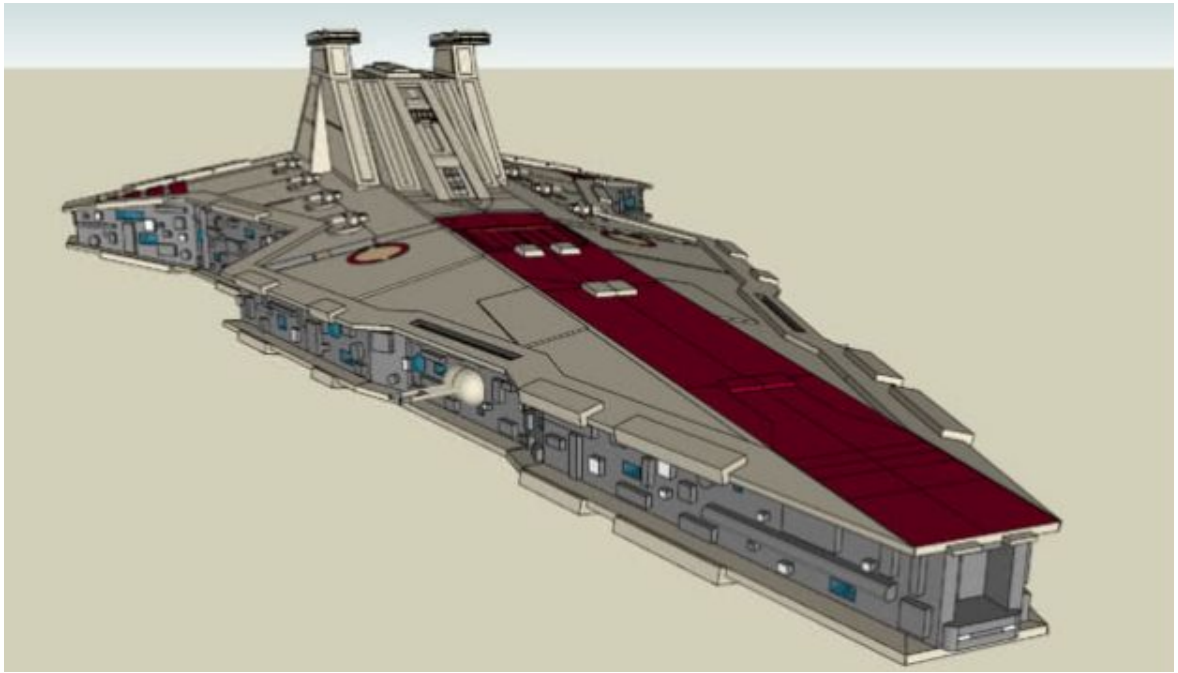
SolidWorks

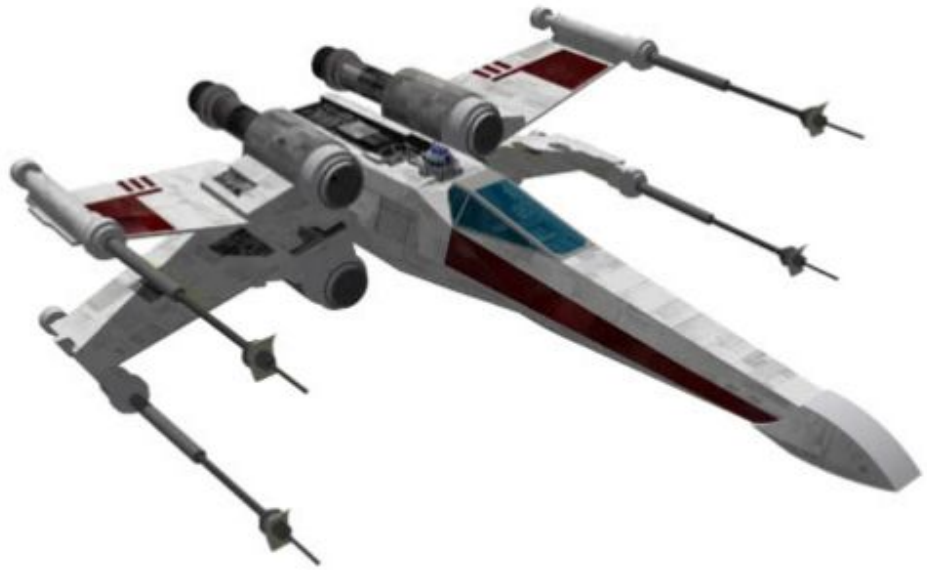


Creating Enviroments

Download from the web

3dwarehouse.sketchup.com





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

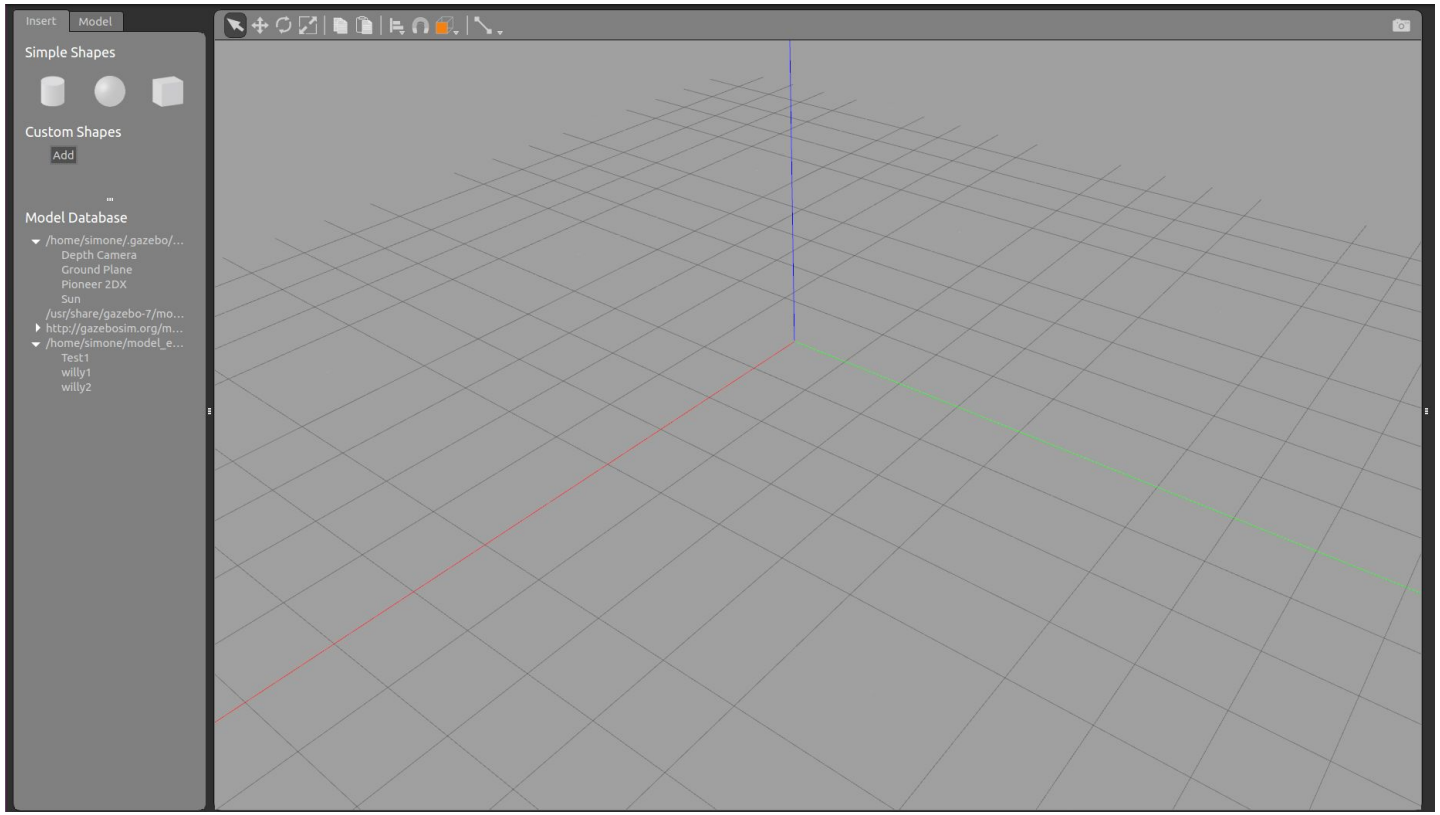
"Compact" description of a complete simulated world

Sounds complex but it's powerful and necessary

Link

https://goo.gl/GonArW

USING THE EDITOR



Why a graphical editor:

- -easier
- -faster (sometimes)
- -visual feedback
- -create good code

Open a terminal and run gazebo:

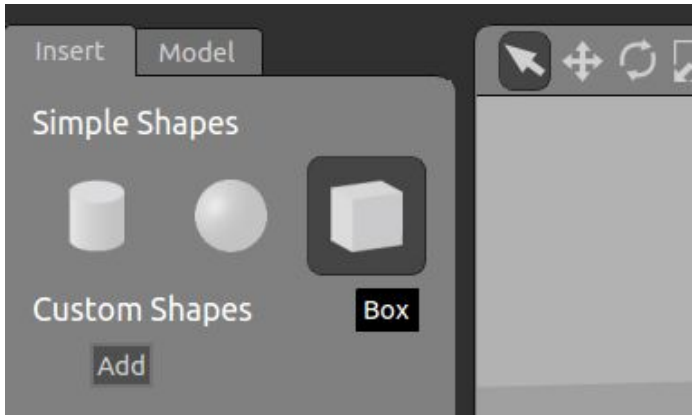
$ gazebo

Open the model editor:

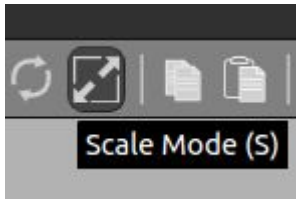
Edit-> Model editor

USING THE EDITOR

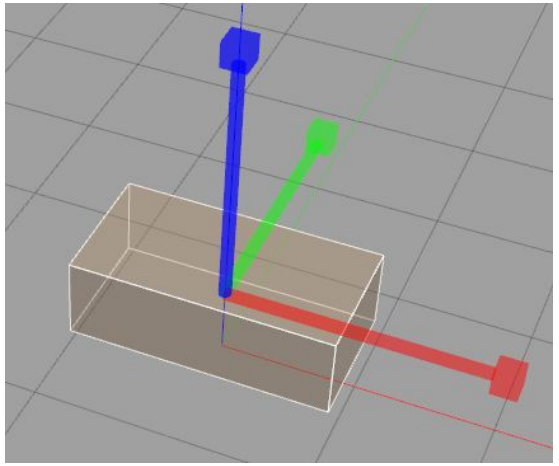
Drag and drop a cube from the left panel to the 3D space



Click on the “Scale Mode” button to edit the box dimensions



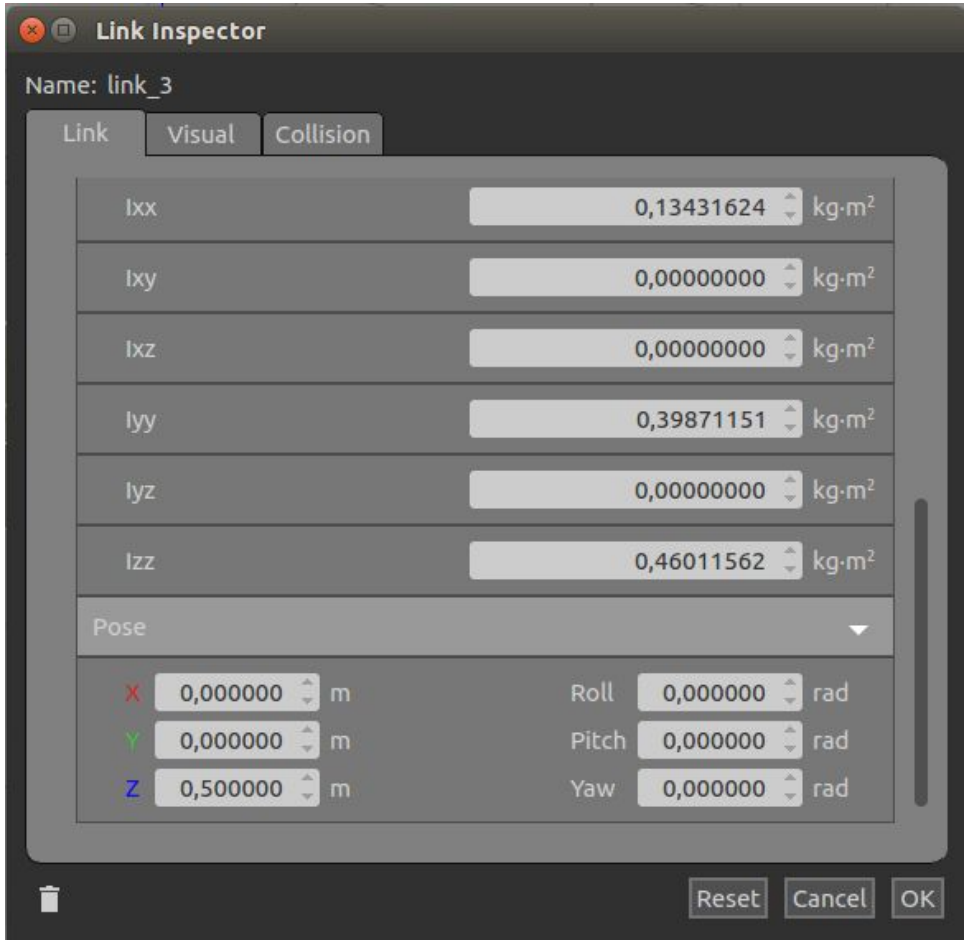
Change the dimensions to 2m long (x axis) and 0.5m high (z axis)



USING THE EDITOR

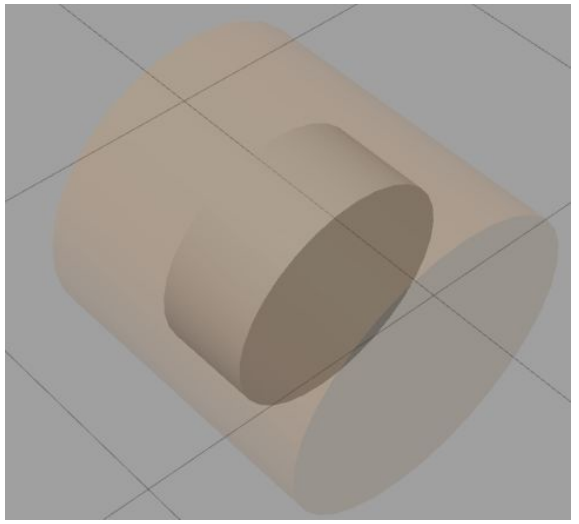
To change the position in a more accurate way open the “Link editor” by double clicking on the object

Change the Z Pose value to 0,4 m



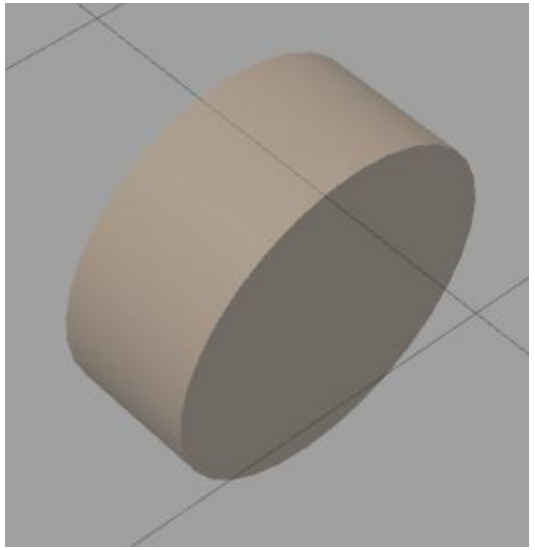
USING THE EDITOR

Create a wheel by dragging a cylinder
Then rotate it, using the Link editor, by 90°
(1,5707 rad) on the x axis.



Now we can resize it, but not using the scale button but with the Link editor.

Under the visual tab change radius to 0.3m and length to 0.25m (this change only the object appearance)



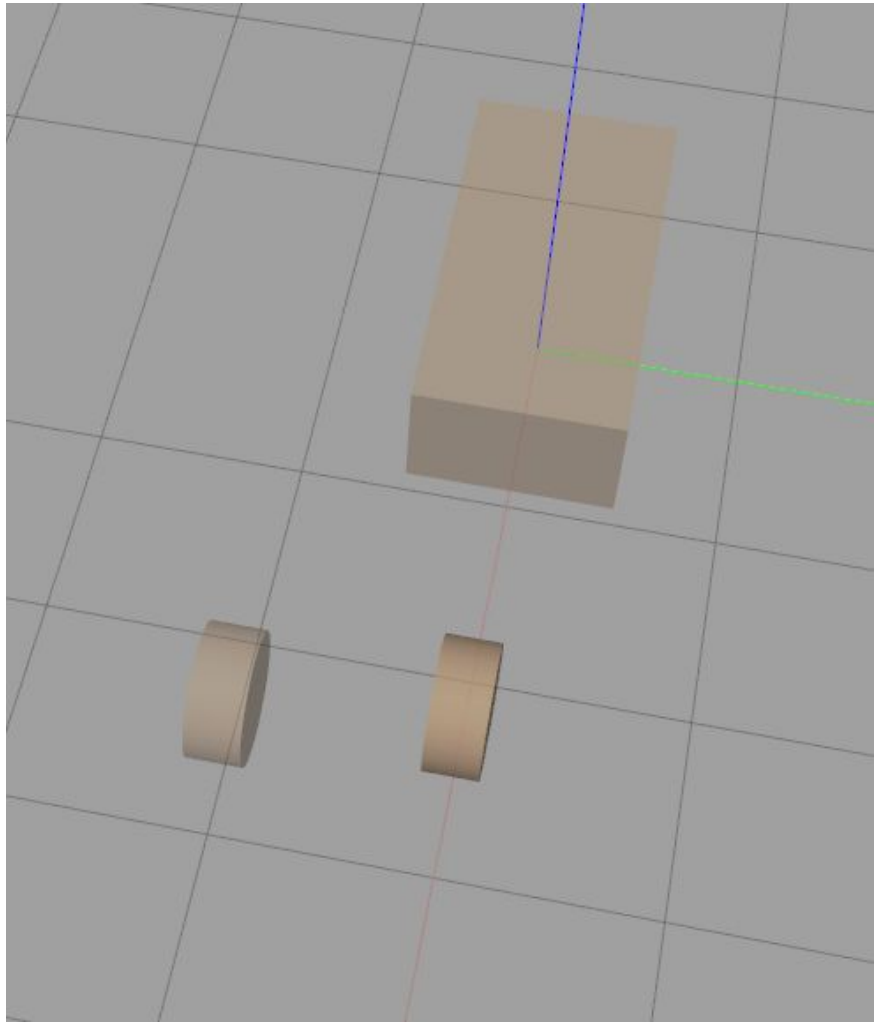
Then make the same changes under the Collision tab (this will change the “behaviour” of the object)

USING THE EDITOR

Copy and paste the “wheel” using the buttons on the top panel

Now we have some components of our car, we have to put them together and define their behaviour

To do this we will use joints

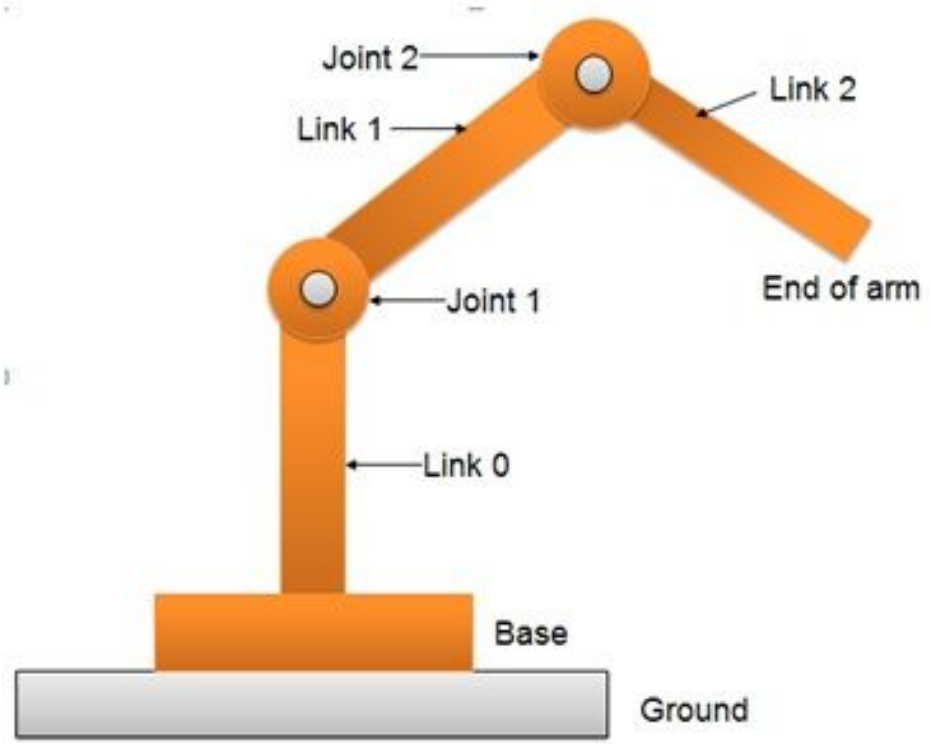


ABOUT JOINTS

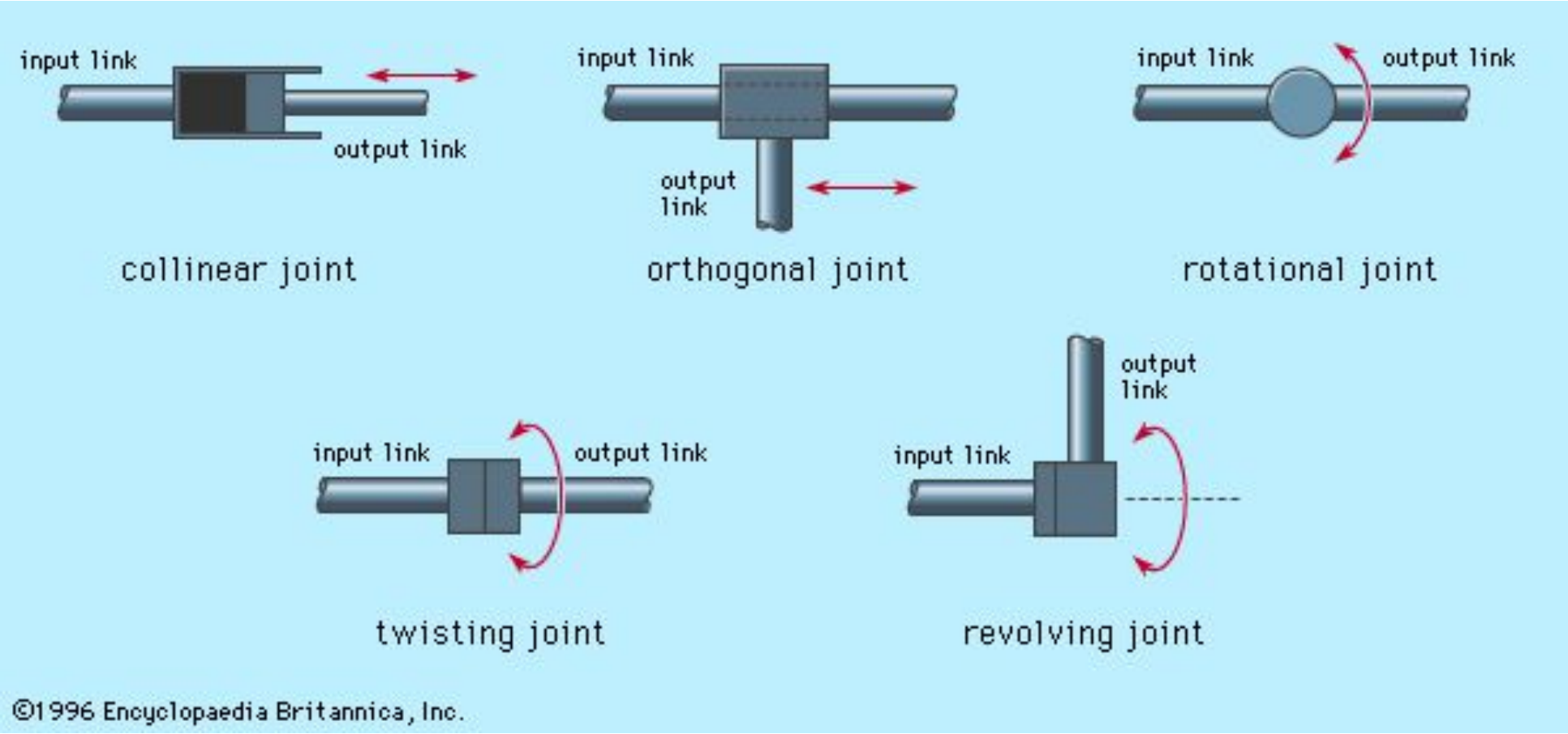
Some definition

"The links are the rigid members connecting the joints"

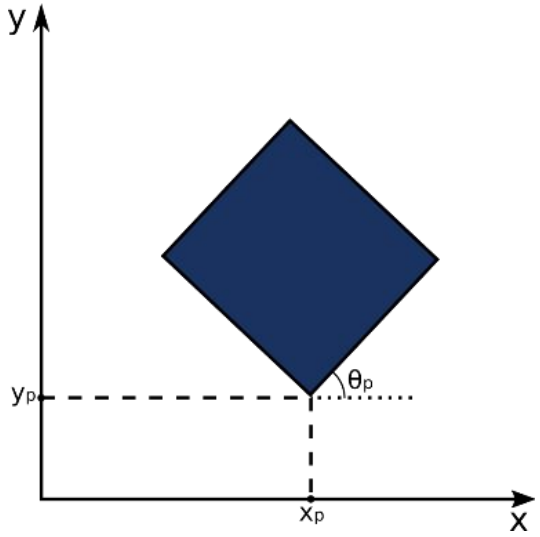
"The joints (also called axes) are the movable components of the robot that cause relative motion between adjacent links"

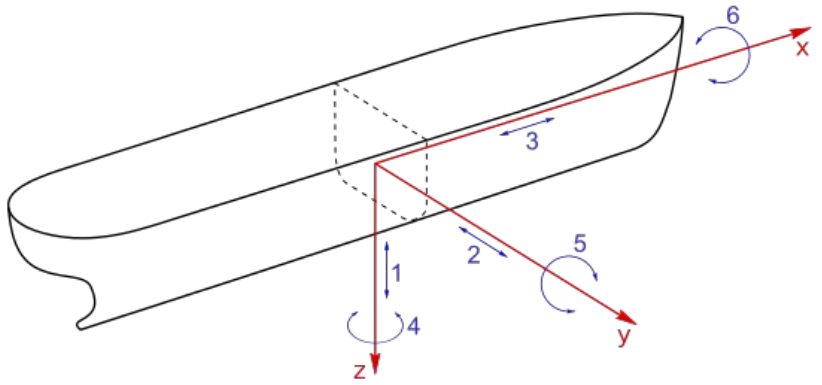


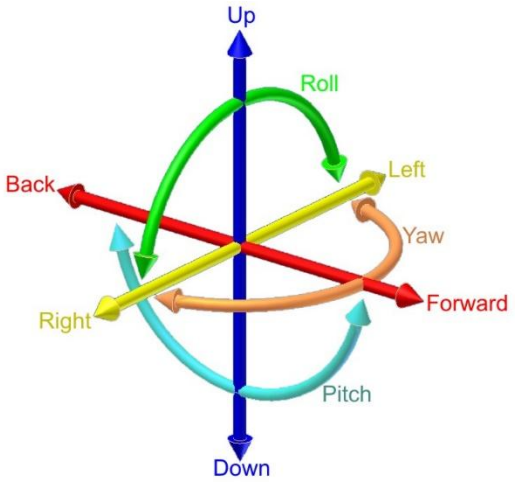
ABOUT JOINTS



ABOUT JOINTS



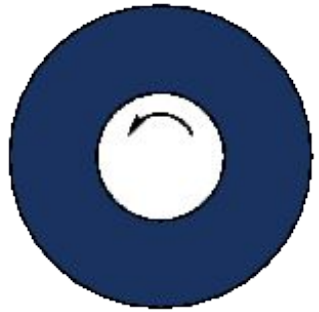




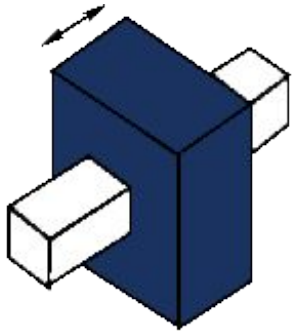
Degree of freedom definition:

"In a mechanical system is the number of independent parameters that define its configuration."

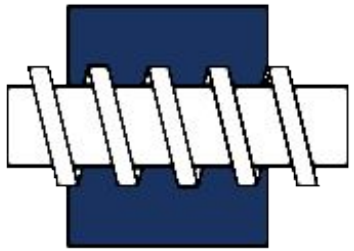
ABOUT JOINTS



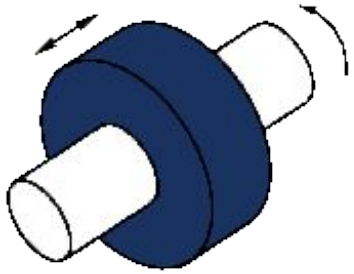
Revolute (1 DoF)



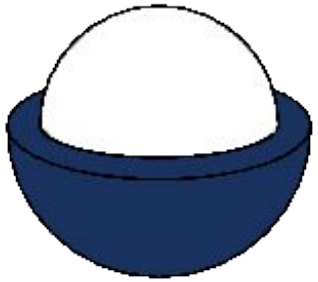
Prismatic (1 DoF)



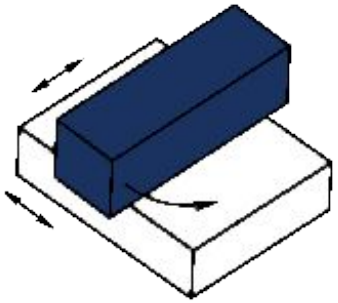
Screw (1 DoF)



Cilindric (2 DoF)



Sphere (3 DoF)



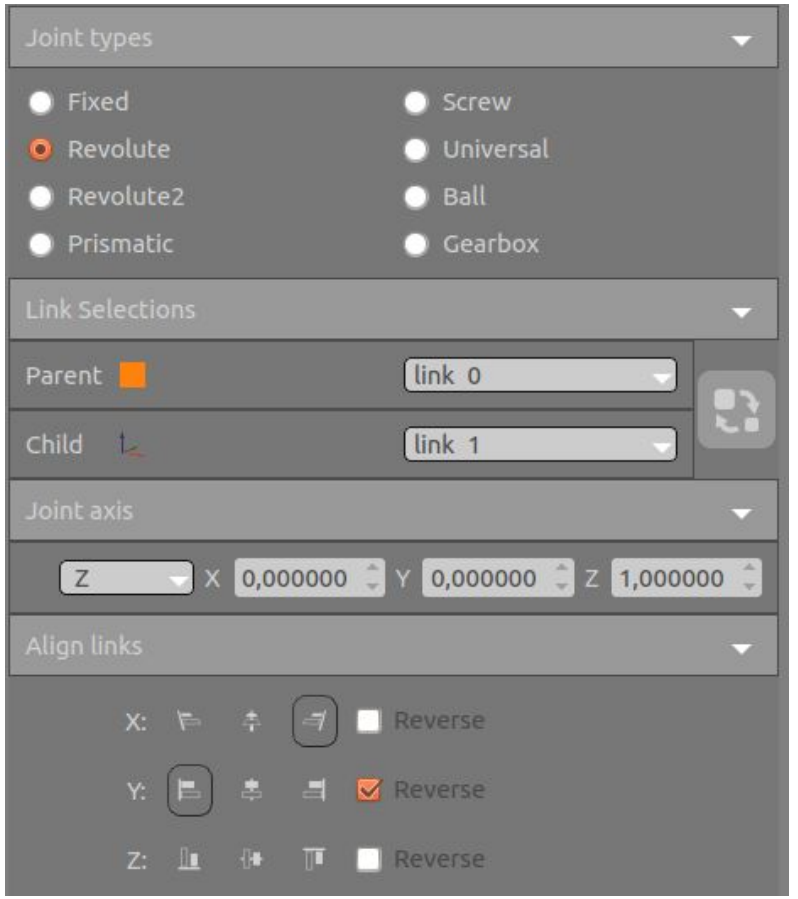
Planar (3 DoF)

USING THE EDITOR

Click on the joint button
Select the joint type, in this case revolute

Then select parent link (in our case the vehicle structure) and the child (one wheel)

Next select the joint axes, to make the wheel correctly spin we select the Z axis (we will see a yellow circle on the axis)

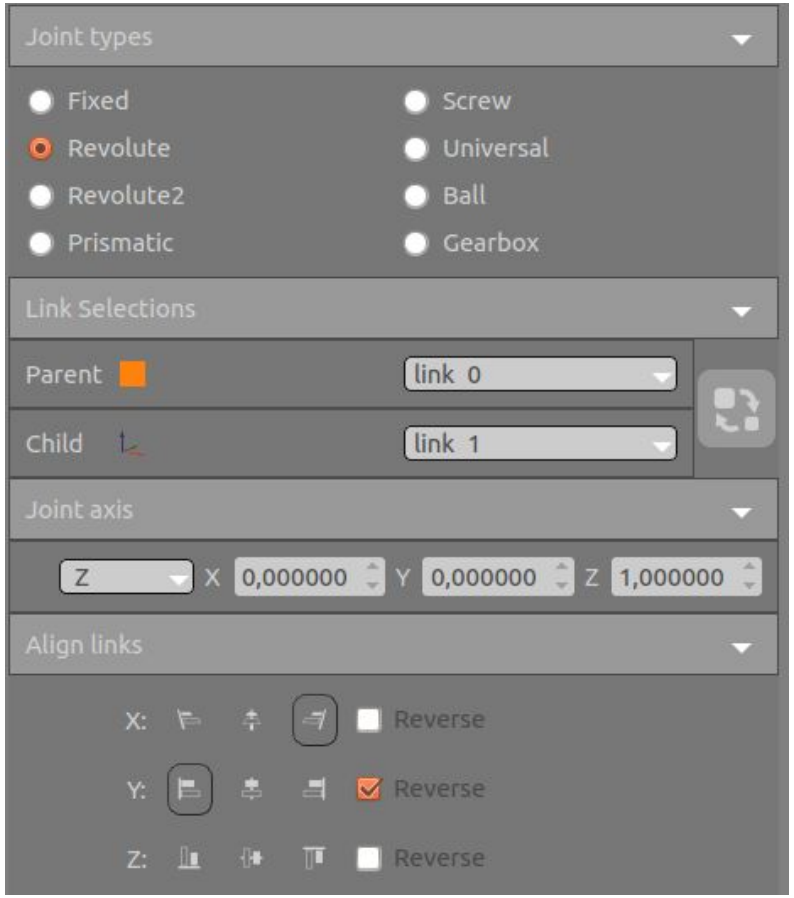


USING THE EDITOR

Last we select the Align links tab to attach the wheel to the body we use:

x axis: align max
y axis: align min and reverse

To create the joint press create

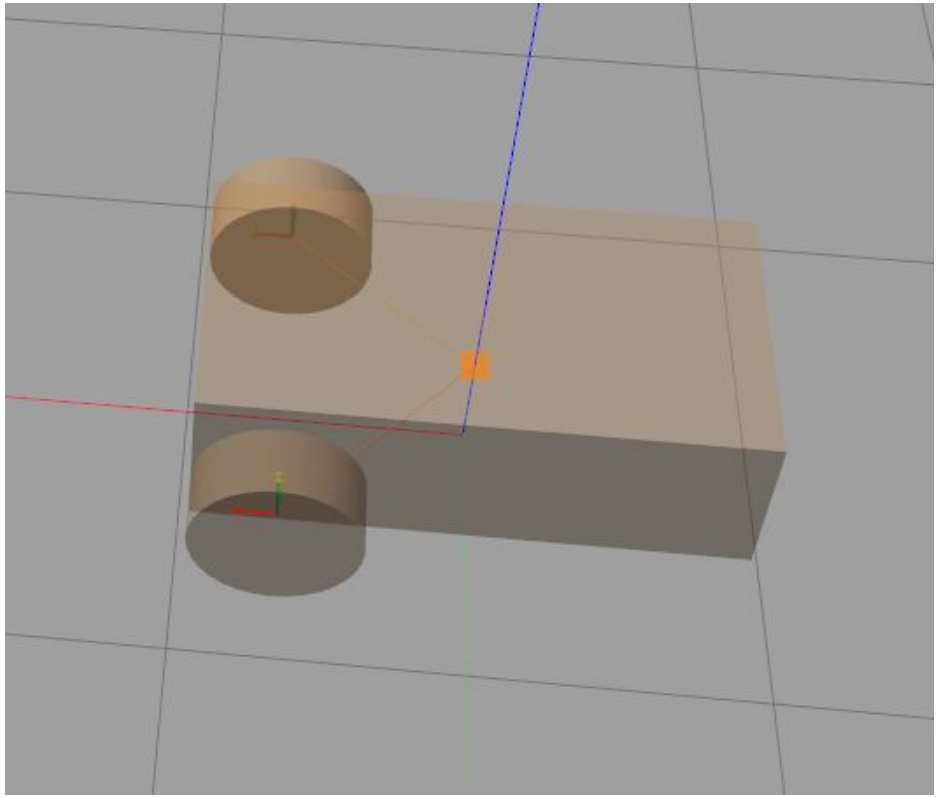


USING THE EDITOR

To position the wheels above the ground we will use again the link inspector and change the Z pose value to 0.3m

For the other wheel we will use similar parameters, but with y align max instead of y align min in the align links tab

Next to create the rear wheel we will use a sphere with 0.2m radius

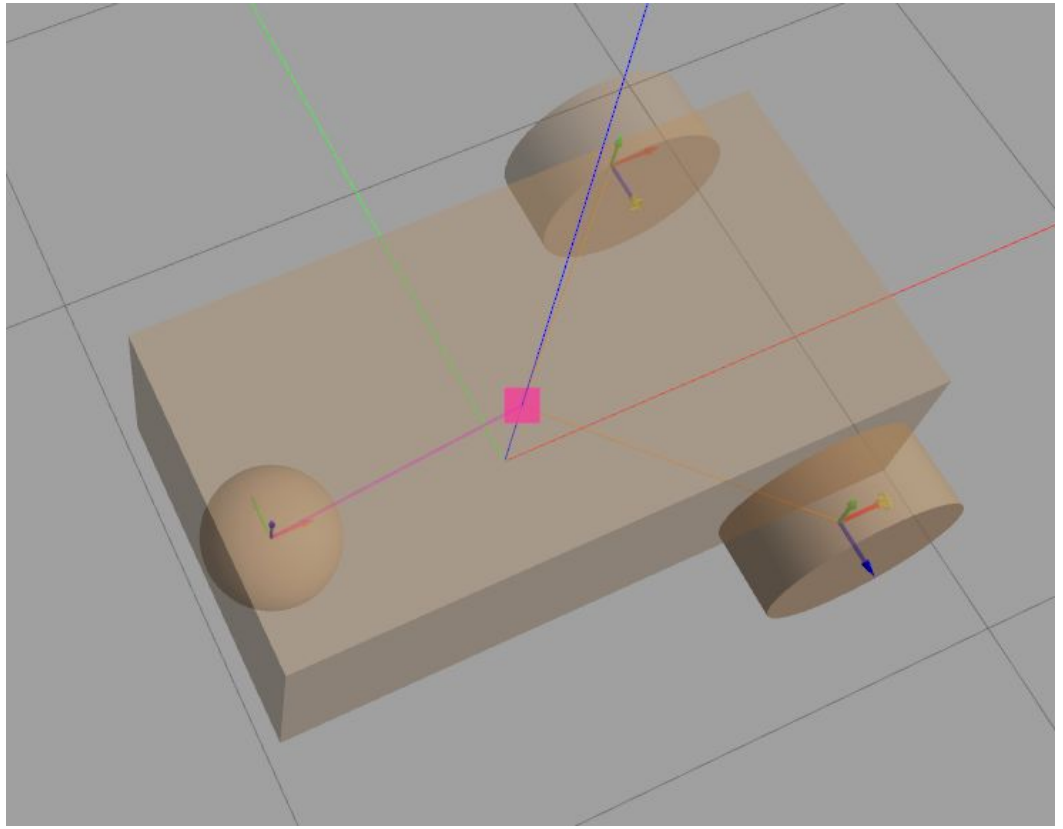


USING THE EDITOR

Then we will apply a joint and position it above the ground as previously shown

The rear will rotate on different direction, so the joint type will not be Revolute but Ball

Last we will change the pose, with z axis value to 0.2 m.



WRITING THE CODE

Two standard, SDF from Gazebo and URDF from ROS

SDF

-xml
-Developed as part of Gazebo
-Describe robots and Scene
-Visual design environment
-SDF file in gazebo can interact with ROS code

URDF

-xml
-Developed as part of ROS
-High ROS integration
-Describe only robots
-No official tools, just rviz to visualize the “robot”
-URDF file can be imported in Gazebo

XML

Markup language:
"*A markup language is a system for annotating a document in a way that is syntactically distinguishable from the text*"

Standard w3c

First version 1998

Created for web

Nowadays used in different fields:
- -web
- -DataBases
- -interprocess communication

```
<?xml version="1.0" encoding="UTF-8"?>
<users>
    <user>
        <name>mario</name>
        <surname>rossi</surname>
        <age>26</age>
    </user>
    <user>
        <name>Giovanni</name>
        <surname>Giusti</surname>
    </user>
</users>
```

XML

Why XML?

Designed to transport data

Designed to be self descriptive

Does not use predefined tags

Is extensible

```
<?xml version="1.0" encoding="UTF-8"?>
<users>
    <user>
        <name>mario</name>
        <surname>rossi</surname>
        <age>26</age>
    </user>
    <user>
        <name>Giovanni</name>
        <surname>Giusti</surname>
    </user>
</users>
```

XML

How it works?

-Prolog (version and encoding)

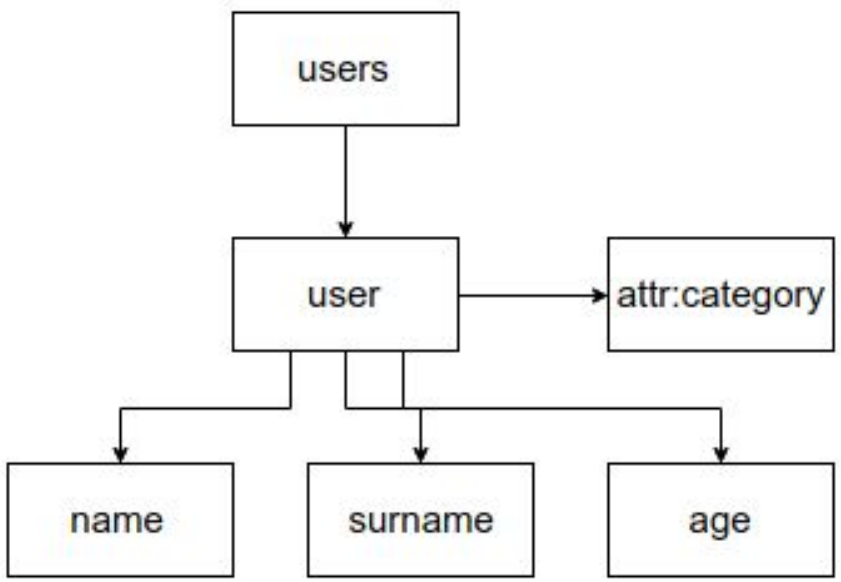
-Tags

-Three structure

-Attribute (optional)

```
<?xml version="1.0" encoding="UTF-8"?>
<users>
    <user category="admin">
        <name>mario</name>
        <surname>rossi</surname>
        <age>26</age>
    </user>
    <user category="simple_user">
        <name>Giovanni</name>
        <surname>Giusti</surname>
    </user>
</users>
```

XML




```
<?xml version="1.0" encoding="UTF-8"?>
<users>
    <user category="admin">
        <name>mario</name>
        <surname>rossi</surname>
        <age>26</age>
    </user>
    <user category="simple_user">
        <name>Giovanni</name>
        <surname>Giusti</surname>
    </user>
</users>
```

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

- A valid SDF file may contain only a single or a list object and act as an “archive”, model can be reused in different world
- A world can contain different model inside the world tag
- A world can include external model file

SDF/MODEL

What is a **model**?

- A container for the elements of the robot (attributes: **name**)
- Composed by links and joints, or other models.
- Use the **include** tag to include previously defined models. Recursion can create really complex structures.

What is a **link**?

- Any rigid element of the robot. Child of the **model** tag.
- It has physical and visual properties and collisions

SDF/MODEL

What is a **joint**?

- Connects two links together with kinematic and dynamics properties
- Various type of joint are available depending on the behavior of the links (revolute, spherical, ...)
- 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.

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 STL files, or obj 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

GENERATED CODE

Open the file generated by Gazebo (model.sdf)

```
<?xml version='1.0'?>
<sdf version='1.6'>
  <model name='Es1'>
    <link name='link_3'>
      <pose frame=''>-0.140499 0 0.1 0 -0 0</pose>
      <inertial>
        <mass>1.17432</mass>
        <inertia>
<pose frame=''>0 0 0 0 -0 0</pose>
      </inertial>
      <gravity>1</gravity>
      <self_collide>0</self_collide>
      <kinematic>0</kinematic>
      <visual name='visual'>
        <pose frame=''>0 0 0 0 -0 0</pose>
        <geometry>
          <box>
            <size>1.92399 1 0.61035</size>
          </box>
        <lighting>1</lighting>
          <script>

<uri>file://media/materials/scripts/gazebo.material</uri>
....
```

File Name

Part Name

Pose

Stuff we didn't choose

Size

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'>model.sdf</sdf>
    <author>
        <name>Builder Bob</name>
        <email>robert.builder@polimi.it</email>
    </author>
    <description>A two wheeled robot.</description>
</model>
```

LET'S SEE AN EXAMPLE

Create the sdf file: **gedit ~/.gazebo/models/willy2/model.sdf**

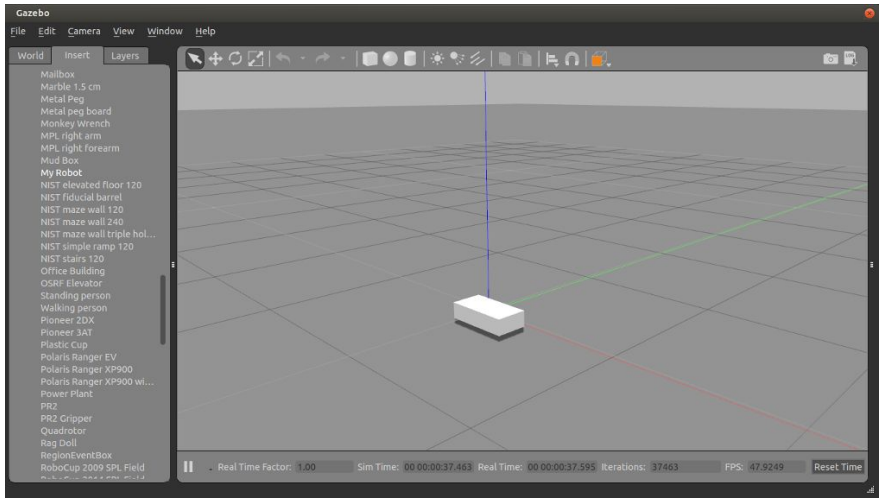
Fill the sdf file:

```
<?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

Add the code (next slide) inside the model tag

BUILDING THE ROBOT

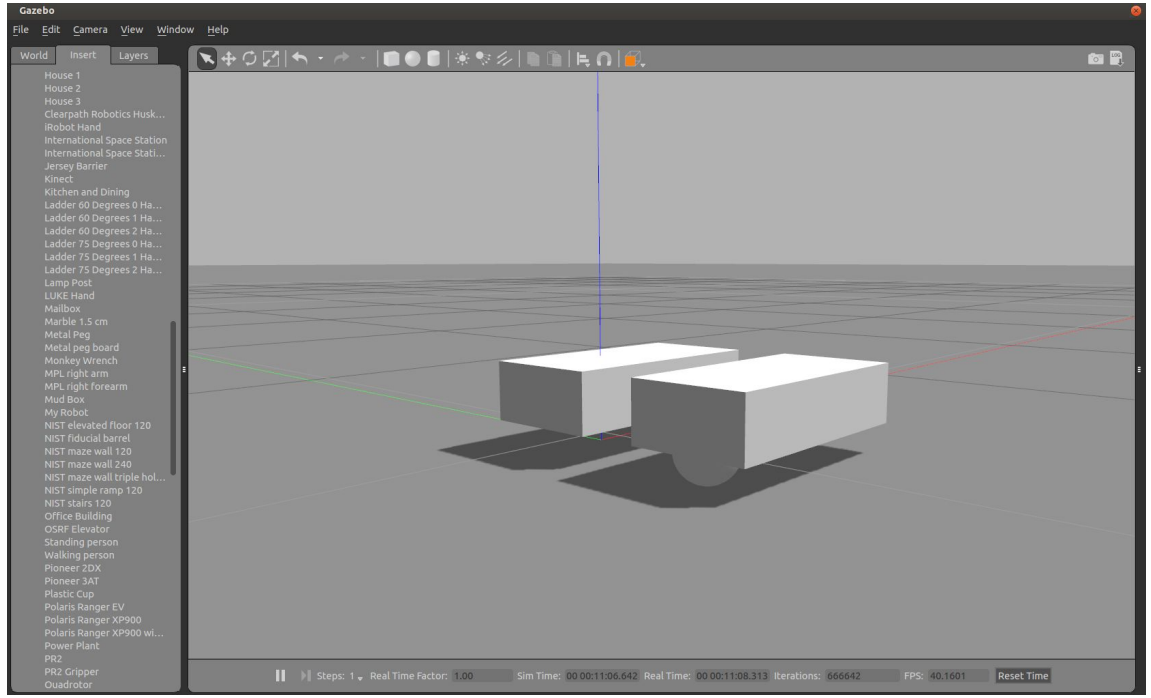
```
<link name='chassis'>
        <pose>0 0 .1 0 0 0</pose>
        <collision name='collision'>
              <geometry>
                    <box>
                          <size>.4 .2 .1</size>
                    </box>
              </geometry>
        </collision>
        <visual name='visual'>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

Insert the code from the next slides inside the link tags



ADDING A CASTER

```
<collision name='caster_collision'>
        <pose>-0.15 0 -0.05 0 0 0</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>
```

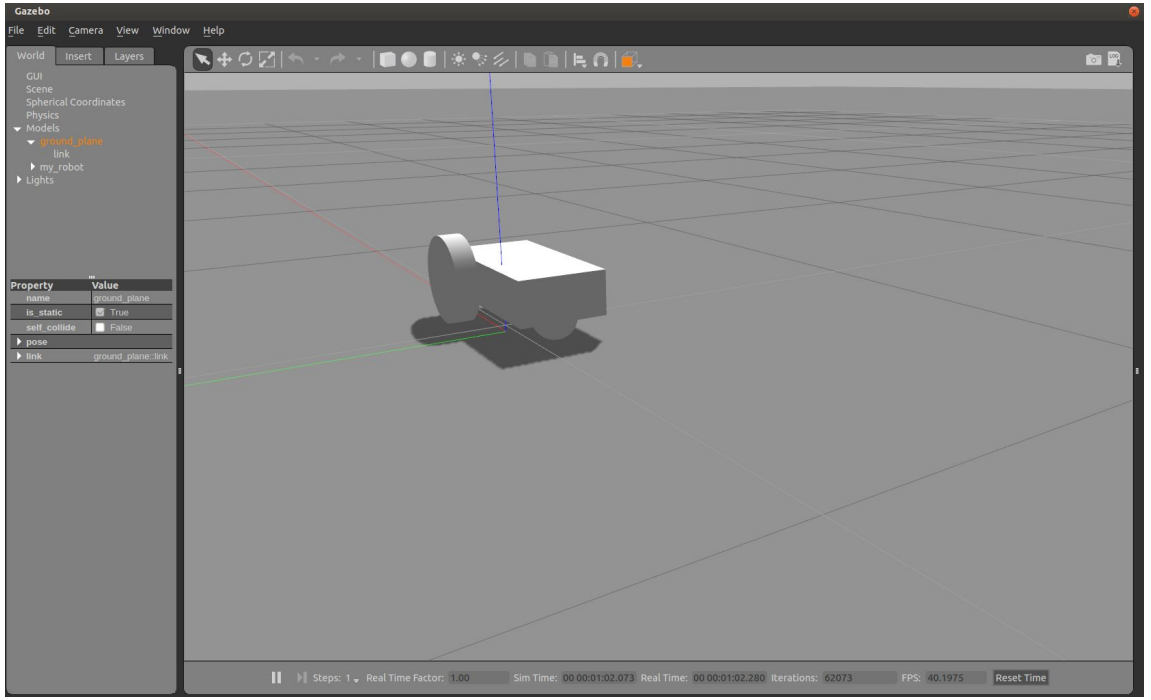
ADDING A CASTER

```
<visual name='caster_visual'>
        <pose>-0.15 0 -0.05 0 0 0</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>
```

ADDING THE WHEELS

```
<link name="right_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>
```

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.

ADDING THE JOINTS

```
<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>
      <parent>chassis</parent>
      <axis>
        <xyz>0 1 0</xyz>
      </axis>
    </joint>
```