Robotics and Intelligent Systems Lab

Week 7

Francesco Maurelli

Fall 2022

ROS Time

ROS Time

- Normally, ROS uses the PC's system clock as time source (wall time)
- For simulations or playback of logged data, it is convenient to work with a simulated time (pause, slow-down etc.)
- To work with a simulated clock:
  - Set the `/use_sim_time` parameter

```
$ rosparam set use_sim_time true
```

  - Publish the time on the topic `/clock` from
    - Gazebo (enabled by default)
    - ROS bag (use option –clock)

ROS Time

- A Time is a specific moment (e.g. ”today at 5pm”)
- A Duration is a period of time (e.g. ”5 hours”). Durations can be negative.
- Time and Duration have an identical representation

```
int32 sec
int32 nsec
```

ROS Time in roscpp

- To take advantage of the simulated time, you should always use the ROS Time APIs:
  - `ros::Time`

```
ros::Time begin = ros::Time::now();
double secs = begin.toSec();
```

  - `ros::Duration`

```
ros::Duration duration(0.5); // 0.5s
```

  - `ros::Rate`

```
ros::Rate rate(10); // 10Hz
```

- If wall time is required, use `ros::WallTime`, `ros::WallDuration`, and `ros::WallRate`

Time and Duration Arithmetic

- Like other primitive types, you can perform arithmetic operations on Times and Durations
  - duration + duration = duration (1 hour + 1 hour = 2 hours)
  - duration - duration = duration (2 hours - 1 hour = 1 hour)
  - time + duration = time (Today + 1 day = tomorrow)
  - time - time = duration (Today - tomorrow = -1 day)
  - time + time is undefined (Today + tomorrow = error)

```
ros::Duration two_hours = ros::Duration(60*60) + ros::Duration(60*60);
ros::Duration one_hour = ros::Duration(2*60*60) - ros::Duration(60*60);
ros::Time tomorrow = ros::Time::now() + ros::Duration(24*60*60);
ros::Duration negative_one_day = ros::Time::now() - tomorrow;
```

Sleeping and Rates

- Sleep for the amount of time specified by the duration

```
ros::Duration(0.5).sleep(); // sleep for half a second
```

- ros::Rate convenience class which makes a best effort at maintaining a particular rate for a loop

```
ros::Rate r(10); // 10 hz
while (ros::ok())
{
  ...  do some work ...
  r.sleep();
}
```

ROS Time in rospy

- Get the current time as either a rospy.Time instance. `rospy.Time.now()` and `rospy.get_rostime()` are equivalent.

```
now = rospy.get_rostime()
rospy.loginfo("Current time %i %i", now.secs, now.nsecs)

seconds = rospy.get_time() # Get the current time in float seconds.
```

- Creating a new Time instance. secs and nsecs are optional and default to zero.

```
epoch = rospy.Time() # secs=nsecs=0
t = rospy.Time(10) # t.secs=10
t = rospy.Time(12345, 6789)
t = rospy.Time.from_sec(123456.789)
```

Time and Duration in rospy

- Time and Duration instances can be converted to seconds as well as nanoseconds for easy use with non-ROS libraries.

```
t = rospy.Time.from_sec(time.time())
seconds = t.to_sec() #floating point
nanoseconds = t.to_nsec()

d = rospy.Duration.from_sec(60.1) # One minute and one tenth of a second
seconds = d.to_sec() #floating point
nanoseconds = d.to_nsec()
```

Time and Duration Arithmetic in rospy

- Same logic as in roscpp

```
two_hours = rospy.Duration(60*60) + rospy.Duration(60*60)
one_hour = rospy.Duration(2*60*60) - rospy.Duration(60*60)
tomorrow = rospy.Time.now() + rospy.Duration(24*60*60)
negative_one_day = rospy.Time.now() - tomorrow
```

Sleeping and Rates in rospy

- `rospy.sleep(duration)`. `duration` can either be a rospy.Duration or seconds (float).

```
rospy.sleep(10.)  # sleep for 10 seconds

d = rospy.Duration(10, 0) # sleep for duration
rospy.sleep(d)
```

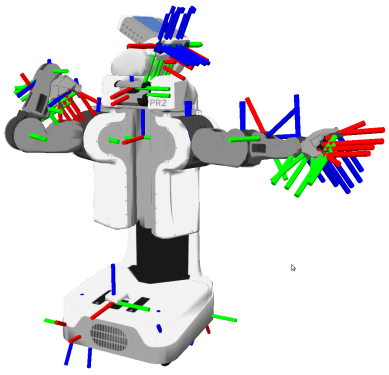
- `rospy.Rate` makes a best effort at maintaining a particular rate for a loop.

```
r = rospy.Rate(10) # 10hz
while not rospy.is_shutdown():
  pub.publish("hello")
  r.sleep()
```

ROS Transformations

TF2 Transformation System

- A tool which lets the user keep track of multiple coordinate frames over time.
- Maintains the relationship between coordinate frames in a tree structure buffered in time.
- It lets the user transform points, vectors, etc between any two coordinate frames at any desired point in time.
- Implemented as publisher/subscriber model on the /tf and /tf_static topics



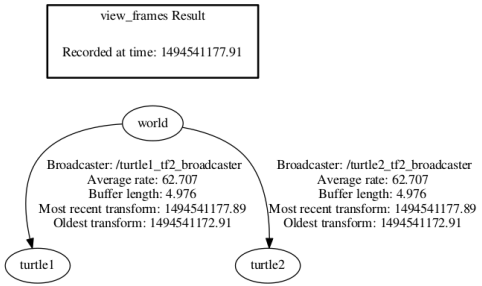
source: wiki.ros.org/tf2

Transformation Tree

- TF uses a buffer to listen to all broadcasted transforms
- Query for specific transforms from the transform tree

tf2_msgs/TFMessage.msg

```
geometry_msgs/TransformStamped[] transforms
  std_msgs/Header header
    uint32 seq
    time stamp
    string frame_idstring
  child_frame_idgeometry_msgs/Transform transform
    geometry_msgs/Vector3 translation
    geometry_msgs/Quaternion rotation
```



source: wiki.ros.org/tf2

TF Tools

- Print information about the current tree

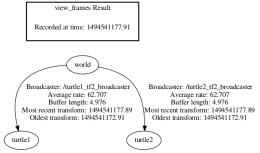
```
$ rosrun tf tf_monitor
```

- Print information about the transform between two frames

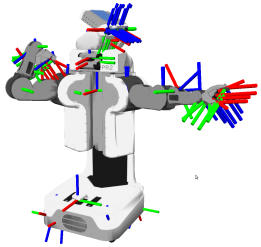
```
$ rosrun tf tf_echo
source_frame
target_frame
```

- Viewing frames (note: `tf view_frames` is deprecated)

```
$ sudo apt-get install
ros-noetic-tf2-tools
$ rosrun tf2_tools
view_frames.py
```



- RViz: 3d visualization of frames



Learning TF

- First let us create a new package called `learning_tf2`

```
$ cd catkin_ws/src
$ catkin_create_pkg learning_tf2 tf2 tf2_ros roscpp rospy turtlesim
```

- Go to the package we just created and create a folder called nodes

```
$ roscd learning_tf2
$ mkdir nodes
```

- Then create a new file nodes/turtle_tf2_broadcaster.py

```
$ touch nodes/turtle_tf2_broadcaster.py
```

Creating a tf Broadcaster

```python
#!/usr/bin/env python
import rospy

# Because of transformations
import tf_conversions

import tf2_ros
import geometry_msgs.msg
import turtlesim.msg


def handle_turtle_pose(msg, turtlename):
    br = tf2_ros.TransformBroadcaster()
    t = geometry_msgs.msg.TransformStamped()

    t.header.stamp = rospy.Time.now()
    t.header.frame_id = "world"
    t.child_frame_id = turtlename
    t.transform.translation.x = msg.x
    t.transform.translation.y = msg.y
    t.transform.translation.z = 0.0
    q = tf_conversions.transformations.quaternion_from_euler(0, 0, msg.theta)
    t.transform.rotation.x = q[0]
    t.transform.rotation.y = q[1]
    t.transform.rotation.z = q[2]
    t.transform.rotation.w = q[3]

    br.sendTransform(t)

if __name__ == '__main__':
    rospy.init_node('tf2_turtle_broadcaster')
    turtlename = rospy.get_param('~turtle')
    rospy.Subscriber('/%s/pose' % turtlename,
                     turtlesim.msg.Pose,
                     handle_turtle_pose,
                     turtlename)
    rospy.spin()
```

Running a tf Broadcaster

- Now create a launch file for this demo

```
<launch>
  <node pkg="turtlesim" type="turtlesim_node" name="sim"/>
  <node pkg="turtlesim" type="turtle_teleop_key" name="teleop" output="screen"/>

  <node name="turtle1_tf2_broadcaster" pkg="learning_tf2" type="turtle_tf2_broadcaster.py" respawn="false"
output="screen">
    <param name="turtle" type="string" value="turtle1"/>
  </node>
  <node name="turtle2_tf2_broadcaster" pkg="learning_tf2" type="turtle_tf2_broadcaster.py" respawn="false"
output="screen">
    <param name="turtle" type="string" value="turtle2"/>
  </node>

</launch>
```

- Don't forget to make the node executable and add it to the CMakeLists.txt and build your package

Inspecting a tf Broadcaster

- Now you're ready to start your own turtle broadcaster demo

```
$ roslaunch learning_tf2 start_demo.launch
```

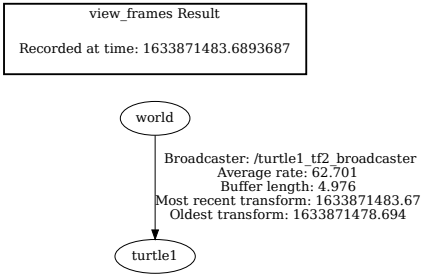
- To inspect the transformations, use the tf_echo tool to check if the turtle pose is actually getting broadcasted

```
▶ rosrun tf tf_echo /world /turtle1
At time 1633870997.990
- Translation: [3.510, 4.539, 0.000]
- Rotation: in Quaternion [0.000, 0.000, 0.976, -0.219]
            in RPY (radian) [0.000, 0.000, -2.699]
            in RPY (degree) [0.000, 0.000, -154.652]
At time 1633870998.678
- Translation: [3.510, 4.539, 0.000]
- Rotation: in Quaternion [0.000, 0.000, 0.976, -0.219]
            in RPY (radian) [0.000, 0.000, -2.699]
            in RPY (degree) [0.000, 0.000, -154.652]
```

Inspecting a tf Broadcaster

- Now try to generated the tf tree

```
$ rosrun tf2_tools view_frames.py
```



Creating a tf Listener

- To create a TF listener a buffer is first needed to fill up

```
tfBuffer = tf2_ros.Buffer()
listener = tf2_ros.TransformListener(tfBuffer)
```

- To look up the transformations, use

```
trans = tfBuffer.lookup_transform(target_frame, source_frame, time)
```

- to get the latest transformation use `rospy.Time()`
- If the transformation is not found, an exception will be returned (LookupException, ConnectivityException, ExtrapolationException)

Creating a tf Listener

```python
#!/usr/bin/env python
import rospy

import math
import tf2_ros
import geometry_msgs.msg
import turtlesim.srv

if __name__ == '__main__':
    rospy.init_node('tf2_turtle_listener')

    tfBuffer = tf2_ros.Buffer()
    listener = tf2_ros.TransformListener(tfBuffer)

    rospy.wait_for_service('spawn')
    spawner = rospy.ServiceProxy('spawn', turtlesim.srv.Spawn)
    turtle_name = rospy.get_param('turtle', 'turtle2') # default param
    spawner(4, 2, 0, turtle_name)

    turtle_vel = rospy.Publisher('%s/cmd_vel' % turtle_name, geometry_msgs.msg.Twist, queue_size=1)

    rate = rospy.Rate(10.0)
    while not rospy.is_shutdown():
        try:
            trans = tfBuffer.lookup_transform(turtle_name, 'turtle1', rospy.Time())
        except (tf2_ros.LookupException, tf2_ros.ConnectivityException, tf2_ros.ExtrapolationException):
            rate.sleep()
            continue

        msg = geometry_msgs.msg.Twist()

        msg.angular.z = 4 * math.atan2(trans.transform.translation.y, trans.transform.translation.x)
        msg.linear.x = 0.5 * math.sqrt(trans.transform.translation.x ** 2 + trans.transform.translation.y ** 2)

        turtle_vel.publish(msg)

        rate.sleep()
```

Running a tf Listener

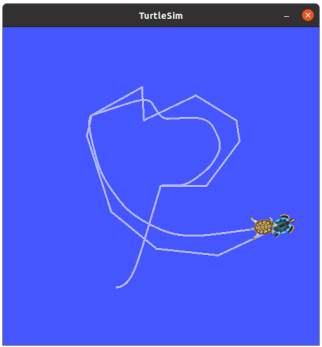
- Add the listener to your CMakeLists.txt and build the package
- Now add the following line to the start_demo.launch

```
<node pkg="learning_tf2" type="turtle_tf2_listener.py" name="listener" output="screen"/>
```

Running a tf Listener

- Add the listener to your CMakeLists.txt and build the package
- Now add the following line to the start_demo.launch

```
<node pkg="learning_tf2" type="turtle_tf2_listener.py" name="listener" output="screen"/>
```

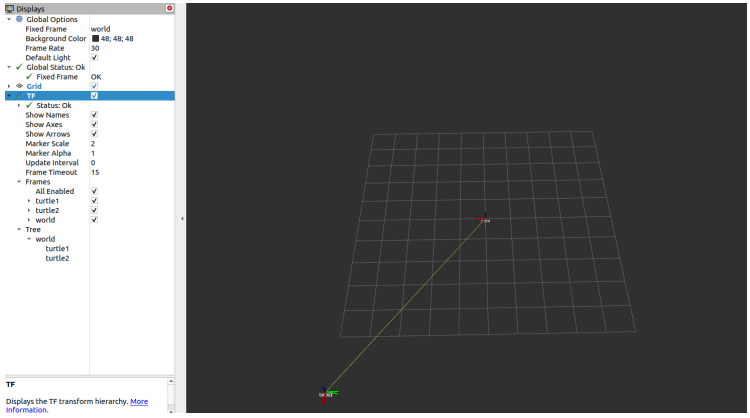


Running a tf Listener

- Try running rviz and add a tf plugin

Running a tf Listener

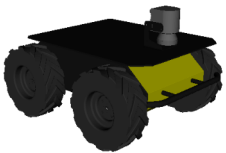
- Try running rviz and add a tf plugin



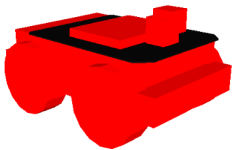
Robot Models

Unified Robot Description Format (URDF)

- Defines an XML format for representing a robot model
  - Kinematic and dynamic description
  - Visual representation
  - Collision model
- URDF generation can be be scripted with XACRO
  (http://wiki.ros.org/xacro)

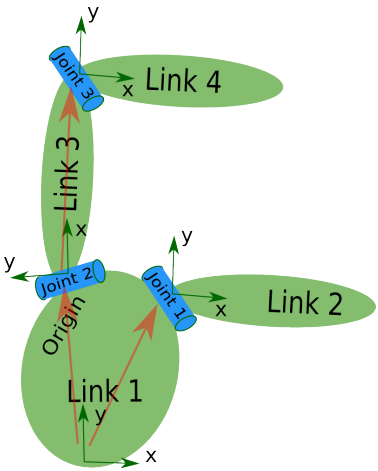


Mesh for visuals



Primitives for collision

Unified Robot Description Format (URDF)

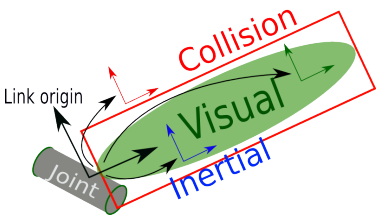


- The description of a robot consists of a set of link elements, and a set of joint elements connecting the links together.

```
<robot name="my_robot">
 <link> ...  </link>
 <link> ...  </link>
 <link> ...  </link>

 <joint> ....  </joint>
 <joint> ....  </joint>
 <joint> ....  </joint>
</robot>
```

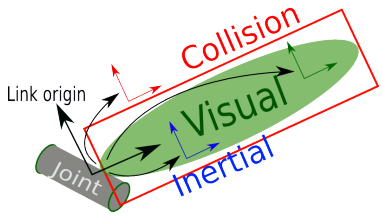
link Element

- A link element is mainly composed of 3 sub-elements
  - `<inertial>`: defines the mass and inertia of the link
  - `<visual>`: provides a visual representation of the link
  - `<collision>`: defines the collision properties of the link
- `<visual>` and `<collision>` can either be primitive shapes or meshes
- Multiple collision bodies can be used for one link



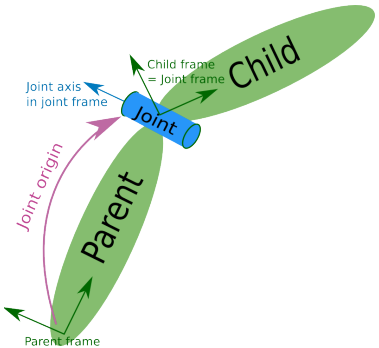
link Element

```
<link name="my_link">
  <inertial>
    <origin xyz="0 0 0.5" rpy="0 0 0"/>
    <mass value="1"/>
    <inertia ixx="100" ixy="0" ixz="0" iyy="100" iyz="0" izz="100" />
  </inertial>

  <visual>
    <origin xyz="0 0 0" rpy="0 0 0" />
    <geometry>
      <box size="1 1 1" />
    </geometry>
    <material name="Cyan">
      <color rgba="0 1.0 1.0 1.0"/>
    </material>
  </visual>

  <collision>
    <origin xyz="0 0 0" rpy="0 0 0"/>
    <geometry>
      <cylinder radius="1" length="0.5"/>
    </geometry>
  </collision>
</link>
```



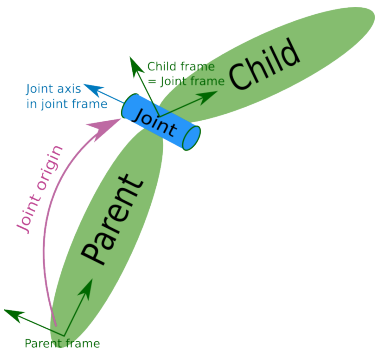
joint Element

- A joint element describes the kinematics and dynamics of the joint and also specifies the safety limits of the join
- Attributes
  - `name`: defines the name of the joint
  - `type`: revolute, prismatic, fixed, ...
- Elements
  - `<origin>`: the transform from the parent link to the child link
  - `<parent>`: parent link
  - `<child>`: child link
  - `<axis>`: x,y,z
  - `<limit>`: upper, lower
  - `<dynamics>`: damping, friction, ...



joint Element

```
<joint name="my_joint" type="revolute">
  <origin xyz="0 0 1" rpy="0 0 3.1416"/>
  <parent link="link1"/>
  <child link="link2"/>

  <dynamics damping="0.0" friction="0.0"/>
  <limit effort="30" velocity="1.0" lower="-2.2" upper="0.7" />
</joint>
```

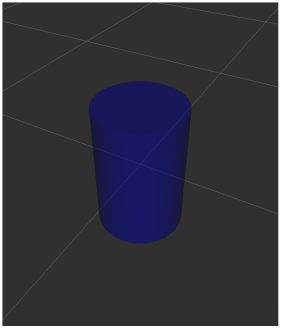


Building a Robot

```
<?xml version="1.0"?>
<robot name="my_robot">

 <material name="blue">
  <color rgba="0 0 0.8 1"/>
 </material>

 <link name="base_link">
  <inertial>
   <mass value="5" />
   <origin xyz="0 0 0" rpy="0 0 0" />
   <inertia ixx="0.2" ixy="0" ixz="0" iyy="0.2" iyz="0" izz="0.1" />
  </inertial>
  <visual>
   <geometry>
    <cylinder length="0.6" radius="0.2"/>
   </geometry>
   <material name="blue"/>
  </visual>
  <collision>
   <geometry>
    <cylinder length="0.6" radius="0.2"/>
   </geometry>
  </collision>
 </link>
</robot>
```

- Display robot in rviz using

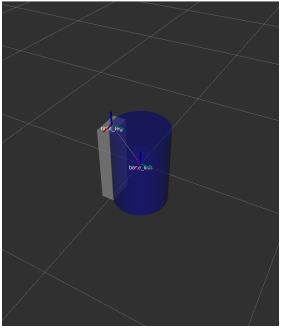
```
$ roslaunch urdf_tutorial
display.launch model:=<path to
file>
```



Building a Robot

Add a right leg

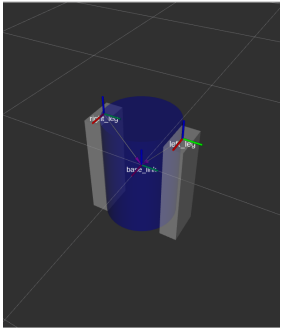
```
<link name="right_leg">
  <visual>
    <geometry>
      <box size="0.6 0.1 0.2"/>
    </geometry>
    <origin rpy="0 1.57075 0" xyz="0 0 -0.3"/>
    <material name="white"/>
  </visual>
  <collision>
    <geometry>
      <box size="0.6 0.1 0.2"/>
    </geometry>
    <origin rpy="0 1.57075 0" xyz="0 0 -0.3"/>  </collision>
</link>

<joint name="base_to_right_leg" type="fixed">
  <parent link="base_link"/>
  <child link="right_leg"/>
  <origin xyz="0 -0.22 0.25"/>
</joint>
```



Building a Robot

Add a left leg

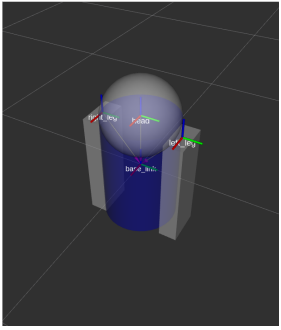
```
<link name="left_leg">
  <visual>
    <geometry>
      <box size="0.6 0.1 0.2"/>
    </geometry>
    <origin rpy="0 1.57075 0" xyz="0 0 -0.3"/>
    <material name="white"/>
  </visual>
  <collision>
    <geometry>
      <box size="0.6 0.1 0.2"/>
    </geometry>
    <origin rpy="0 1.57075 0" xyz="0 0 -0.3"/>  </collision>
</link>

<joint name="base_toleft_leg" type="fixed">
  <parent link="base_link"/>
  <child link="left_leg"/>
  <origin xyz="0 0.22 0.25"/>
</joint>
```



Building a Robot

Add a head

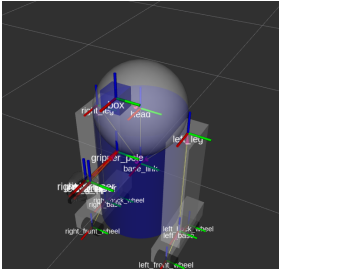
```
<link name="head">
  <visual>
    <geometry>
      <sphere radius="0.2"/>
    </geometry>
    <material name="white"/>
  </visual>
</link>

<joint name="head_swivel" type="fixed">
  <parent link="base_link"/>
  <child link="head"/>
  <origin xyz="0 0 0.3"/>
</joint>
```



Building a Robot

Add a few more components

```
<?xml version="1.0"?>
<robot name="my_robot">

  <material name="blue">
    <color rgba="0 0 0.8 1"/>
  </material>
  <material name="black">
    <color rgba="0 0 0 1"/>
  </material>
  <material name="white">
    <color rgba="1 1 1 1"/>
  </material>

  <link name="base_link">
    <inertial>
      <mass value="5" />
      <origin xyz="0 0 0" rpy="0 0 0" />
      <inertia ixx="0.004" ixy="0.001" ixz="0" iyy="0.006" iyz="0" izz="0.007" />
    </inertial>
    <visual>
      <geometry>
        <cylinder length="0.6" radius="0.2"/>
      </geometry>
      <material name="blue"/>
    </visual>
    <collision>
      <geometry>
        <cylinder length="0.6" radius="0.2"/>
      </geometry>
    </collision>
  </link>

  <link name="right_leg">
    <visual>
      <geometry>
        <box size="0.6 0.1 0.2"/>
      </geometry>
      <origin rpy="0 1.57075 0" xyz="0 0 -0.3"/>
      <material name="white"/>
    </visual>
    <collision>
      <geometry>
        <box size="0.6 0.1 0.2"/>
      </geometry>
      <origin rpy="0 1.57075 0" xyz="0 0 -0.3"/>
    </collision>
  </link>

  <joint name="base_to_right_leg" type="fixed">
    <parent link="base_link"/>
    <child link="right_leg"/>
    <origin xyz="0 -0.22 0.25"/>
  </joint>

  <link name="right_base">
    <visual>
      <geometry>
        <box size="0.4 0.1 0.1"/>
      </geometry>
      <material name="white"/>
    </visual>
    <collision>
      <geometry>
        <box size="0.4 0.1 0.1"/>
      </geometry>
    </collision>
  </link>

  <joint name="right_base_joint" type="fixed">
    <parent link="right_leg"/>
    <child link="right_base"/>
    <origin xyz="0 0 -0.6"/>
  </joint>

  <link name="right_front_wheel">
    <visual>
      <origin rpy="1.57075 0 0" xyz="0.0 0 0"/>
      <geometry>
        <cylinder length="0.1" radius="0.035"/>
      </geometry>
      <material name="black"/>
    </visual>
    <collision>
      <origin rpy="1.57075 0 0" xyz="0.0 0 0"/>
      <geometry>
        <cylinder length="0.1" radius="0.035"/>
      </geometry>
    </collision>
  </link>
  <joint name="right_front_wheel_joint" type="fixed">
    <parent link="right_base"/>
    <child link="right_front_wheel"/>
    <origin rpy="0 0 0" xyz="0.133333333333 0 -0.085"/>
  </joint>

  <link name="right_back_wheel">
    <visual>
      <origin rpy="1.57075 0 0" xyz="0.0 0 0"/>
      <geometry>
        <cylinder length="0.1" radius="0.035"/>
      </geometry>
      <material name="black"/>
    </visual>
    <collision>
      <origin rpy="1.57075 0 0" xyz="0.0 0 0"/>
      <geometry>
        <cylinder length="0.1" radius="0.035"/>
      </geometry>
    </collision>
  </link>
  <joint name="right_back_wheel_joint" type="fixed">
    <parent link="right_base"/>
    <child link="right_back_wheel"/>
    <origin rpy="0 0 0" xyz="-0.133333333333 0 -0.085"/>
  </joint>

  <link name="left_leg">
    <visual>
      <geometry>
        <box size="0.6 0.1 0.2"/>
      </geometry>
      <origin rpy="0 1.57075 0" xyz="0 0 -0.3"/>
      <material name="white"/>
    </visual>
  </link>

  <joint name="base_to_left_leg" type="fixed">
    <parent link="base_link"/>
    <child link="left_leg"/>
    <origin xyz="0 0.22 0.25"/>
  </joint>

  <link name="left_base">
    <visual>
      <geometry>
        <box size="0.4 0.1 0.1"/>
      </geometry>
      <material name="white"/>
    </visual>
    <collision>
      <geometry>
        <box size="0.4 0.1 0.1"/>
      </geometry>
    </collision>
  </link>

  <joint name="left_base_joint" type="fixed">
    <parent link="left_leg"/>
    <child link="left_base"/>
    <origin xyz="0 0 -0.6"/>
  </joint>

  <link name="left_front_wheel">
    <visual>
      <origin rpy="1.57075 0 0" xyz="0.0 0 0"/>
      <geometry>
        <cylinder length="0.1" radius="0.035"/>
      </geometry>
      <material name="black"/>
    </visual>
    <collision>
      <origin rpy="1.57075 0 0" xyz="0.0 0 0"/>
      <geometry>
        <cylinder length="0.1" radius="0.035"/>
      </geometry>
    </collision>
  </link>
  <joint name="left_front_wheel_joint" type="fixed">
    <parent link="left_base"/>
    <child link="left_front_wheel"/>
    <origin rpy="0 0 0" xyz="0.133333333333 0 -0.085"/>
  </joint>

  <link name="left_back_wheel">
    <visual>
      <origin rpy="1.57075 0 0" xyz="0.0 0 0"/>
      <geometry>
        <cylinder length="0.1" radius="0.035"/>
      </geometry>
      <material name="black"/>
    </visual>
    <collision>
      <origin rpy="1.57075 0 0" xyz="0.0 0 0"/>
      <geometry>
        <cylinder length="0.1" radius="0.035"/>
      </geometry>
    </collision>
  </link>
  <joint name="left_back_wheel_joint" type="fixed">
    <parent link="left_base"/>
    <child link="left_back_wheel"/>
    <origin rpy="0 0 0" xyz="-0.133333333333 0 -0.085"/>
  </joint>

  <joint name="gripper_extension" type="fixed">
    <parent link="base_link"/>
    <child link="gripper_pole"/>
    <origin rpy="0 0 0" xyz="0.19 0 0.2"/>
  </joint>

  <link name="gripper_pole">
    <visual>
      <geometry>
        <cylinder length="0.2" radius="0.01"/>
      </geometry>
      <origin rpy="0 1.57075 0" xyz="0.1 0 0"/>
    </visual>
    <collision>
      <geometry>
        <cylinder length="0.2" radius="0.01"/>
      </geometry>
      <origin rpy="0 1.57075 0" xyz="0.1 0 0"/>
    </collision>
  </link>

  <joint name="left_gripper_joint" type="fixed">
    <origin rpy="0 0 0" xyz="0.2 0.01 0"/>
    <parent link="gripper_pole"/>
```



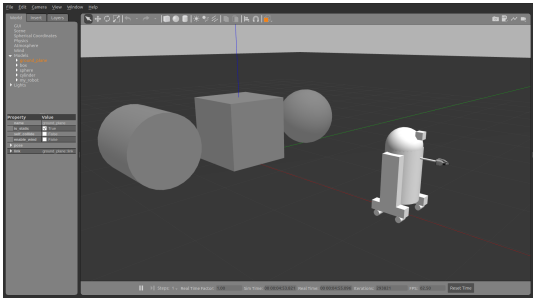
Gazebo

Gazebo Simulator

- Simulate 3d rigid-body dynamics
- Simulate a variety of sensors including noise
- 3d visualization and user interaction
- Includes a database of many robots and environments (Gazebo worlds)
- Provides a ROS interface
- Extensible with plugins

run gazebo

```
$ rosrun gazebo_ros gazebo
```



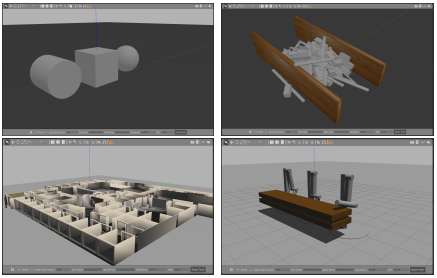
roslaunch & World Models

- To start an empty Gazebo world similar to the `rosrun` command

```
$ roslaunch gazebo_ros empty_world.launch
```

- roslaunch arguments
  - `paused`: Start Gazebo in a paused state
  - `use_sim_time`: use Gazebo's simulation time, published over the ROS topic/clock
  - `gui`: Launch the user interface window of Gazebo (default true)
  - `verbose`: print errors and warnings to the terminal

- Other demo worlds `willowgarage_world.launch`, `mud_world.launch`, `shapes_world.launch`, `rubble_world.launch`



World Files

- Notice in mud_world.launch a simple jointed mechanism is launched. A ”worlds/mud.world” is given as a value for the argument ”world_name”

```
<launch>

 <include file="$(find gazebo_ros)/launch/empty_world.launch">
  <arg name="world_name" value="worlds/mud.world"/>

  <arg name="paused" value="false"/>
  <arg name="use_sim_time" value="true"/>
  <arg name="gui" value="true"/>
  <arg name="headless" value="false"/>
  <arg name="recording" value="false"/>
  <arg name="debug" value="false"/>
 </include>

</launch>
```

World Files

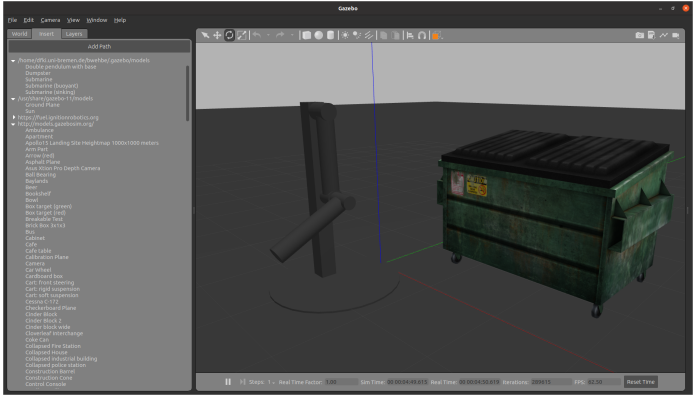
- If we continue to investigate ”worlds/mud.world” you will find the following (found under `/usr/share/gazebo-<version>/`)

```
<sdf version="1.4">
  <world name="default">
    <include>
      <uri>model://sun</uri>
    </include>
    <include>
      <uri>model://ground_plane</uri>
    </include>
    <include>
      <uri>model://double_pendulum_with_base</uri>
      <name>pendulum_thick_mud</name>
      <pose>-2.0 0 0 0 0 0</pose>
    </include>
    ...
  </world>
</sdf>
```

- All Gazebo default models can be found under
  https://github.com/osrf/gazebo_models

Getting Objects from Database

- If you open gazebo and select the Insert tab in the to access the model database. (it might take some time)



Simulation Description Format (SDF)

- SDF is an extension of URDF
- It defines an XML format to describe
  - Environments (lighting, gravity etc.)
  - Objects (static and dynamic)
  - Sensors
  - Robots
- SDF is the standard format for Gazebo
- Gazebo converts a URDF to SDF automatically

more on http://sdformat.org

Creating a World File

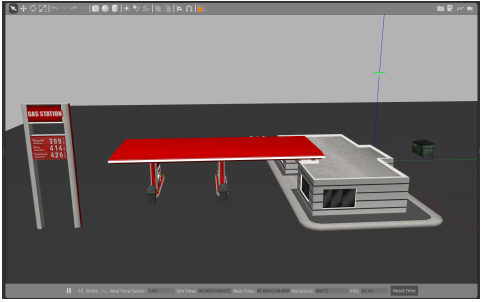
- In your package add a `worlds/my_robot.world`

```
<sdf version="1.4">
  <world name="default">
    <include>
      <uri>model://ground_plane</uri>
    </include>
    <include>
      <uri>model://sun</uri>
    </include>
    <include>
      <uri>model://dumpster</uri>
      <name>dumpster</name>
      <pose>-2.0 7.0 0 0 0 0</pose>
    </include>
    <include>
      <uri>model://gas_station</uri>
      <name>gas_station</name>
      <pose>7.0 -7.0 0 0 0 0</pose>
    </include>
  </world>
</sdf>
```

Launch a World File

- Add a `launch/my_robot.launch`

```
<launch>
  <include file="$(find gazebo_ros)/launch/empty_world.launch">
    <arg name="world_name" value="$(find my_robot)/worlds/my_robot.world"/>
    <!-- more default parameters can be changed here -->
  </include>
</launch>
```



Spawn a Robot

- Spawning a robot in a world

```
$ rosrun gazebo_ros spawn_model -file `rospack find my_robot`/urdf/my_robot.urdf -urdf -x 13 -y -13 -z 1.2
-model my_robot
```

Spawn a Robot

- Do this in a launch file

```
<node name="spawn_urdf" pkg="gazebo_ros" type="spawn_model" args="-file $(find my_robot)/urdf/my_robot.urdf
-urdf -x 13 -y -13 -z 1.2 -model my_robot
```

- Spawning Xacro files

```
<!-- Convert an xacro and put on parameter server -->
<param name="robot_description" command="$(find xacro)/xacro.py $(find
my_robot_description)/robots/my_robot.urdf.xacro" />

<!-- Spawn a robot into Gazebo -->
<node name="spawn_urdf" pkg="gazebo_ros" type="spawn_model" args="-param robot_description -urdf -model
my_robot" />
```

Good Practices

Package organization

```
../catkin_ws/src
    /MYROBOT_description
        package.xml
        CMakeLists.txt
        /urdf
            MYROBOT.urdf
        /meshes
            mesh1.dae
            mesh2.dae
            ...
        /materials
        /cad
    /MYROBOT_gazebo
        /launch
            MYROBOT.launch
        /worlds
            MYROBOT.world
        /models
            world_object1.dae
            world_object2.stl
            world_object3.urdf
        /materials
        /plugins
```

Model_description

```
../catkin_ws/src
    /MYROBOT_description
        package.xml
        CMakeLists.txt
        model.config
        /urdf
            MYROBOT.urdf
        /meshes
            mesh1.dae
            mesh2.dae
            ...
        /materials
        /plugins
        /cad
```

model.config

```
<?xml version="1.0"?>
<model>
  <name>MYROBOT</name>
  <version>1.0</version>
  <sdf>urdf/MYROBOT.urdf</sdf>
  <author>
    <name>My name</name>
    <email>name@email.address</email>
  </author>
  <description>
    A description of the model
  </description>
</model>
```

Useful Links

Useful Links

- Wiki: http://wiki.ros.org/
- API: https://wiki.ros.org/APIs
- Debugging tools: https://wiki.ros.org/roswtf
- Navigation stack: https://wiki.ros.org/navigation
- Control stack: https://wiki.ros.org/ros_control
- Motion planning: https://moveit.ros.org/
- Point Cloud Library : https://wiki.ros.org/pcl
- Transformations: https://wiki.ros.org/tf2
- Xacro: https://wiki.ros.org/xacro
- URDF: https://wiki.ros.org/urdf
- SDF: http://sdformat.org
- Gazebo: http://gazebosim.org/tutorials