

GAZEBO



RE510

2024. 03. 25. ~ 2024. 03. 29.

MORIN lab

Professor : Jinwhan Kim

TA : Donghyun Kim

What is Gazebo?

Gazebo

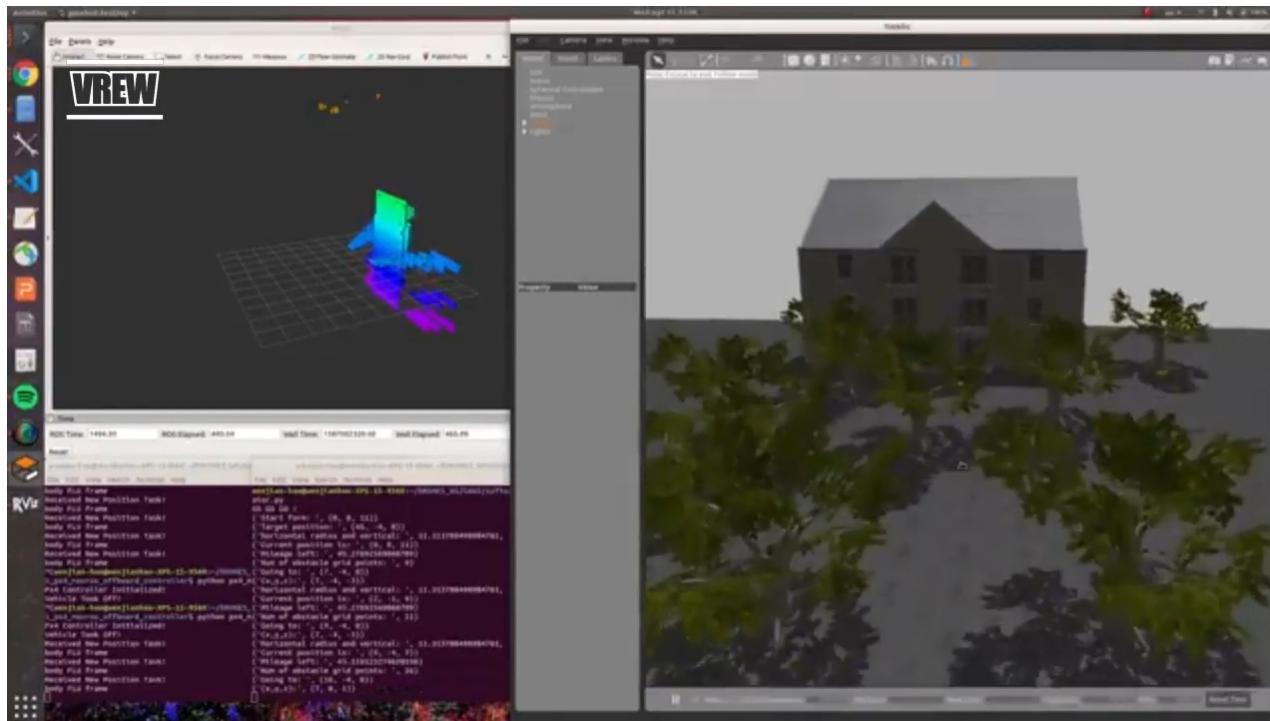


TABLE of CONTENTS

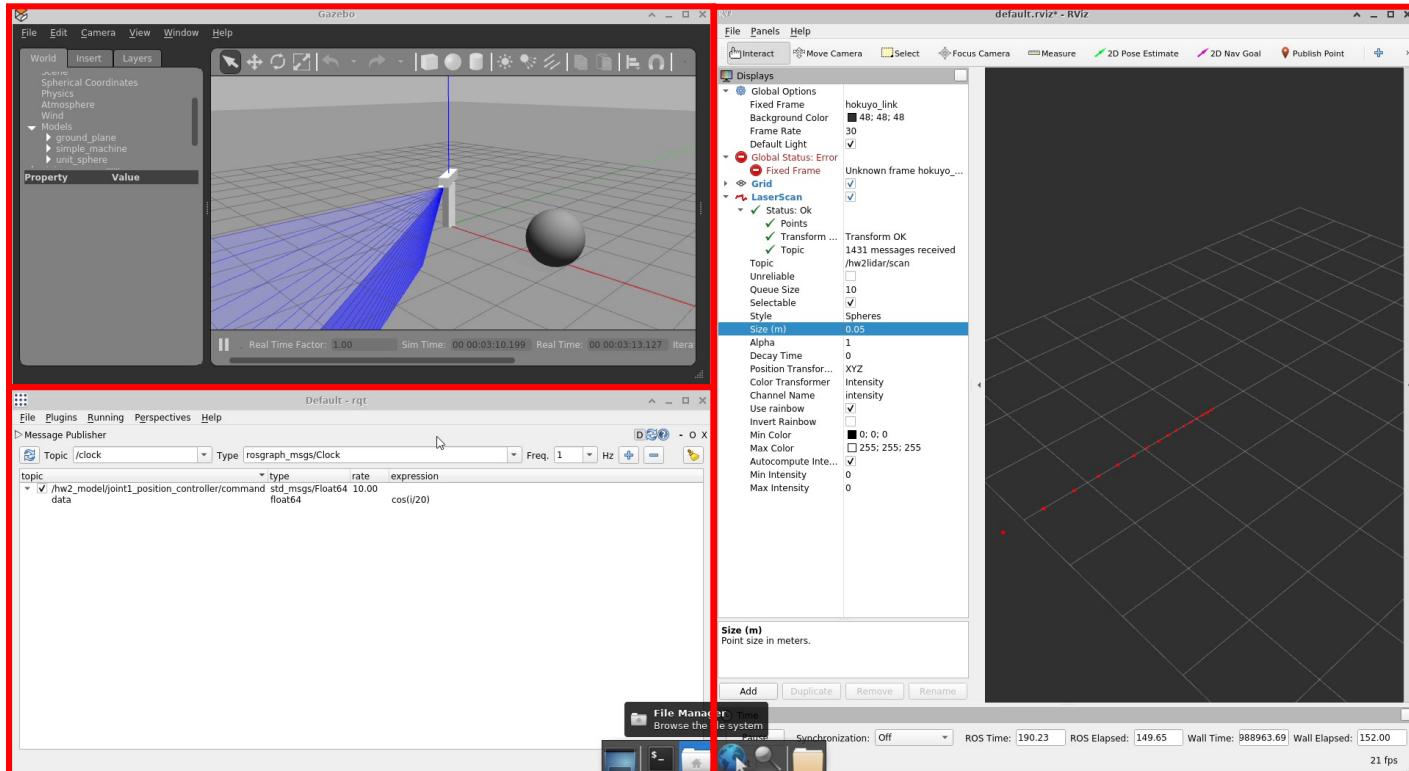
- 1. Build a simple machine by URDF**
 - 1. ROS control**
 - 2. Add sensors**
 - 3. Connect to ROS**
- 2. Build a simple machine by Gazebo GUI**
 - 1. ROS control**
 - 2. Add sensors**
 - 3. Connect to ROS**
- 3. Assignment #2**

Build a simple machine by URDF

Build a simple machine by URDF

Simple lidar

Simple lidar in gazebo



rqt

Build a simple machine by URDF

Package structure

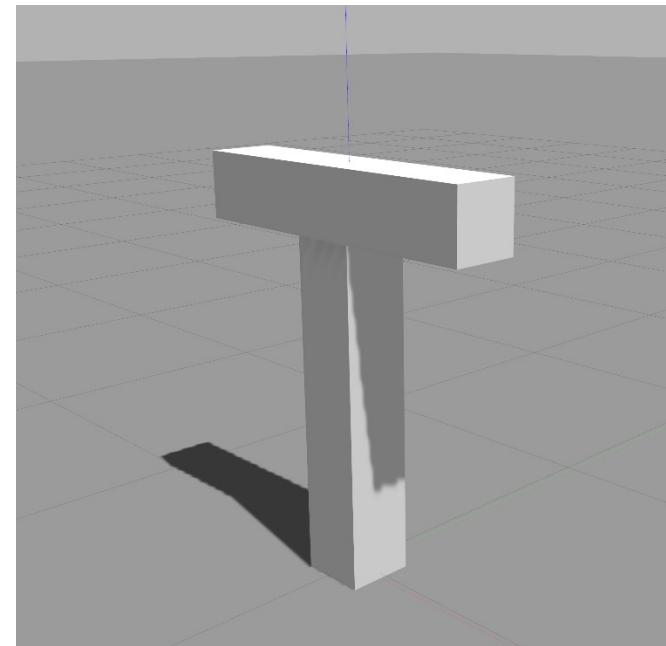
Tutorial package

- CMakeLists.txt
- package.xml
- launch
 - spawn.launch — spawn simple lidar in gazebo
 - function.launch — activate controller, state publisher etc
 - param.yaml — can easily adjust parameter
- model
 - model.xacro

Build a simple machine by URDF

Create a simple model

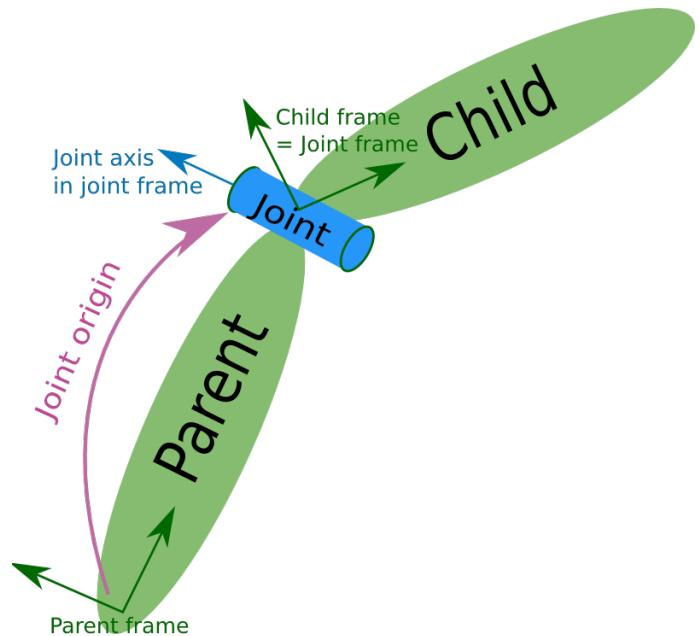
- ▶ <link name="world"/>
 - ▶ Open <link> and close in one line by ending with / before >
- ▶ <link name='link_0'>
- ▶ <collision>
 - ▶ Sets the dimension that come into contact between other objects
- ▶ <visual>
 - ▶ Sets the dimension of the appearance of the link
- ▶ <inertial>
 - ▶ Sets the physical properties such as mass and moment of inertia of the object
- ▶ <joint name="joint_0" type="fixed">
 - ▶ Open <joint> and set name and types
 - ▶ Can be closed by </joint>
 - ▶ <joint> requires parent and child links
- ▶ <parent link="world"/>
- ▶ <child link="link_0"/>



Build a simple machine by URDF

Create a simple model

- ▶ Create another link (`link_1`)
- ▶ Generate a joint between the two links
 - ▶ `<joint name='joint_1' type='continuous'>`
- ▶ Set the parent and child links
- ▶ You can also use different types such as
 - ▶ Revolute : a hinge joint with limited range
 - ▶ Prismatic : a sliding joint with limited range
 - ▶ Fixed : immovable
- ▶ We will add some damping in the joint
 - ▶ `<dynamics damping="0.7"/>`
- ▶ To be more specific with the model, add more parameters
 - ▶ Refer to <http://wiki.ros.org/urdf/XML/joint> for possible parameters
- ▶ Don't forget to add this at the end of the file!
 - ▶ `</robot>`



Build a simple machine by URDF

Create a launch file for spawning the model in Gazebo

- ▶ Use rosrun to see your mechanism using Gazebo
- ▶ First create a package
- ▶ Name the package tutorial

```
$ cd ~/catkin_ws/src
$ catkin_create_pkg tutorial controller_manager joint_state_controller robot_state_publisher
$ cd tutorial
$ mkdir config
$ mkdir launch
```

Necessary dependencies to control our machine later

- ▶ Create a launch file (spawn.launch)

```
<launch>
  <param name="robot_description" command="$(find xacro)/xacro --inorder '[DIRECTORY OF YOUR XACRO FILE]/[NAME OF YOUR XACRO FILE].xacro" />
  <node name="urdf_spawner" pkg="gazebo_ros" type="spawn_model" respawn="false" output="screen" args="-urdf -model [NAME THAT WILL APPEAR IN GAZEBO] -param robot_description"/>
</launch>
```

Build a simple machine by URDF

Open Gazebo

- ▶ There are two ways of starting Gazebo

- ▶ One

```
$ gazebo
```

- ▶ Two (through ROS)

```
$ roscore
```

```
$ rosrun gazebo_ros gazebo
```

- ▶ Strongly recommend that you use the second method

- ▶ Lets you use the information about the simulation environment when using ROS

- ▶ Terminate Gazebo

```
$ killall -9 gazebo & killall -9 gzserver & killall -9 gzclient
```

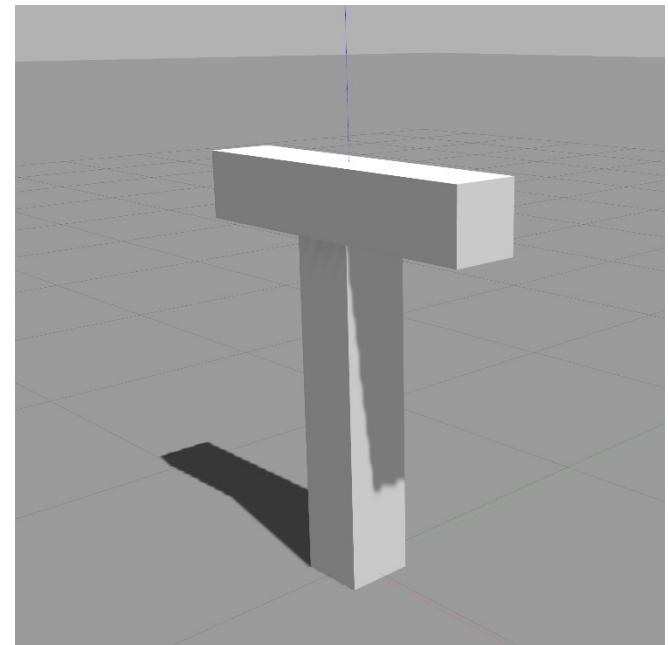
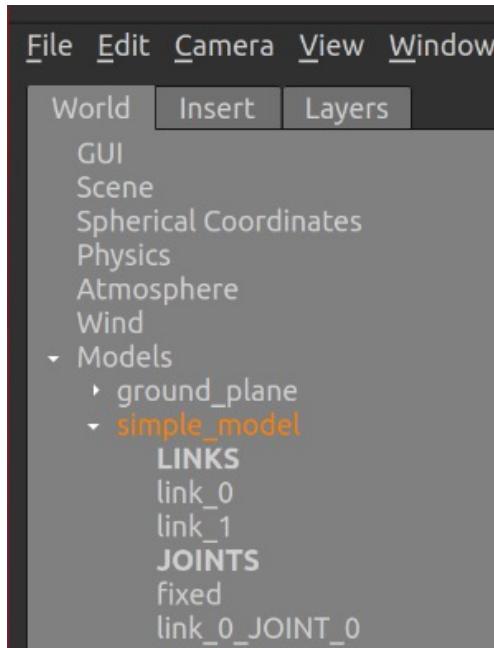
Build a simple machine by URDF

View it in Gazebo

- ▶ Run the launch file!

`$ rosrun tutorial spawn.launch`

- ▶ Two links connected by a continuous joint
- ▶ We can see the links and the joints that makes up our simple model



ROS control

Adding a virtual actuator

- ▶ Now let's add a virtual motor on the joint and move it
- ▶ Go back to our model.xacro file and the following before the end

```
<transmission name="tran_1">  
  <type>transmission_interface/SimpleTransmission</type>  
  <joint name="joint_1">  
    <hardwareInterface>hardware_interface/EffortJointInterface</hardwareInterface>  
  </joint>  
  <actuator name="motor_1">  
    <hardwareInterface>hardware_interface/EffortJointInterface</hardwareInterface>  
    <mechanicalReduction>1</mechanicalReduction>  
  </actuator>  
</transmission>
```

- ▶ As of now, there is only SimpleTransmission type and EffortJointInterface interface implemented for Gazebo
- ▶ Specify the joint that you want to move

ROS control

Adding a Plugin

- ▶ We have specified to which joint an actuator will be added
- ▶ To control this actuator, we also need to add a plugin
 - ▶ Plugin lets you add a lot of functions to your models such as sensors and actuators
 - ▶ It also lets you publish or subscribe ROS messages
 - ▶ It is possible to design the plugins, but there are many useful plugins available online
- ▶ We will be using Gazebo ROS Control for actuation

```
<gazebo>
  <plugin name="gazebo_ros_control" filename="libgazebo_ros_control.so">
    <robotNamespace>/simple_model</robotNamespace>
  </plugin>
</gazebo>
```
- ▶ Make sure the <robotNamespace> is properly set

ROS control

Back to ROS

- ▶ Create param.yaml file
- ▶ Here, specify the robot name, controller settings, and controller tuning parameters

```
simple_model:  
  # Publish all joint states -----  
  joint_state_controller:  
    type:joint_state_controller/JointStateController  
    publish_rate: 50  
  
  # Position Controllers -----  
  joint1_position_controller:  
    type:effort_controllers/JointPositionController  
    joint:joint_1  
    pid:{p: 100.0, i: 0.01, d: 10.0}
```

ROS control

Back to ROS

- ▶ Different types of controllers can be implemented
 - ▶ effort_controllers/JointEffortController – Desired force or torque command
 - ▶ effort_controllers/JointPositionController – Desired position command
 - ▶ effort_controllers/JointVelocityController – Desired velocity command
- ▶ Install package manually

```
$ sudo apt-get install ros-noetic-joint-state-controller
```

```
$ sudo apt-get install ros-noetic-effort-controllers
```

```
$ sudo apt-get install ros-noetic-position-controllers
```

ROS control

Launch ROS control

- ▶ Now, the controllers and the motors are set
- ▶ Go to launch folder in tutorialpackage
- ▶ Create a launch file to start ROS Control (function.launch)
- ▶ Load the controller configuration

```
<rosparam file="$(find tutorial)/config/param.yaml" command="load"/>
```

- ▶ Load the controllers

```
<node name="controller_spawner" pkg="controller_manager" type="spawner" respawn="false" output="screen"  
ns="/simple_model" args="joint1_position_controller joint_state_controller"/>
```

Make sure to also have joint_state_controller

- ▶ Convert joint states to TF transforms

```
<node name="robot_state_publisher" pkg="robot_state_publisher" type="robot_state_publisher" respawn="false"  
output="screen">
```

- ▶ Remap the topic name

```
<remap from="/joint_states" to="/simple_model/joint_states" />
```

ROS control

Launch ROS control

- ▶ rosrun tutorial function.launch
- ▶ Check if the controllers are successfully working

```
>Loading controller: joint_state_controller
Controller Spawner: Loaded controllers: joint1_position_controller, joint_state_controller
Started controllers: joint1_position_controller, joint_state_controller
```

ROS control

Launch ROS control

- You can see the states of the joints through
 - `rostopic echo /simple_model/joint_states` topic

```
header:
  seq: 13926
  stamp:
    secs: 296
    nsecs: 891000000
    frame_id: ''
name:
  - link_0_JOINT_0
position: [-0.7208045373568108]
velocity: [-1.3903108362940253]
effort: [12.580823820427245]
---
```

- Or rqt
- Go to Plugins -> Topics -> Topic Monitor

<input checked="" type="checkbox"/> /simple_model/joint_states	sensor_msgs/JointState	3.71KB/s	49.85	(4.440892098500626e-14,)
↳ header	float64[]			
name	std_msgs/Header			
position	string[]			[link_0_JOINT_0]
velocity	float64[]			(-0.1982488578045869,)
	float64[]			(2.4670063897680337e-14,)

ROS control

Launch ROS control

- ▶ Give command input through the topic /command
 - ▶ Terminal


```
$ rostopic pub -l /simple_model/joint1_position_controller/command std_msgs/Float64 "data: 1.5"
```
 - ▶ Later, you can write nodes that send much more complex topic messages to control your robot
- ▶ You can also use RQT to publish to a topic
 - ▶ Go to Plugins -> Topics -> Message Publisher



Fill out the topic name
and the message type

Plus adds the publisher
Minus removes it

ROS control

Use RQT to publish command input

- In expression, write the command input you want
 - Being a position controller for a revolving joint, the input is in radians
 - i in rqt publisher is the time step

topic	type	rate	expression
/simple_model/joint1_position_controller/command data	std_msgs/Float64	10.00	float64 $\cos(i/50)$

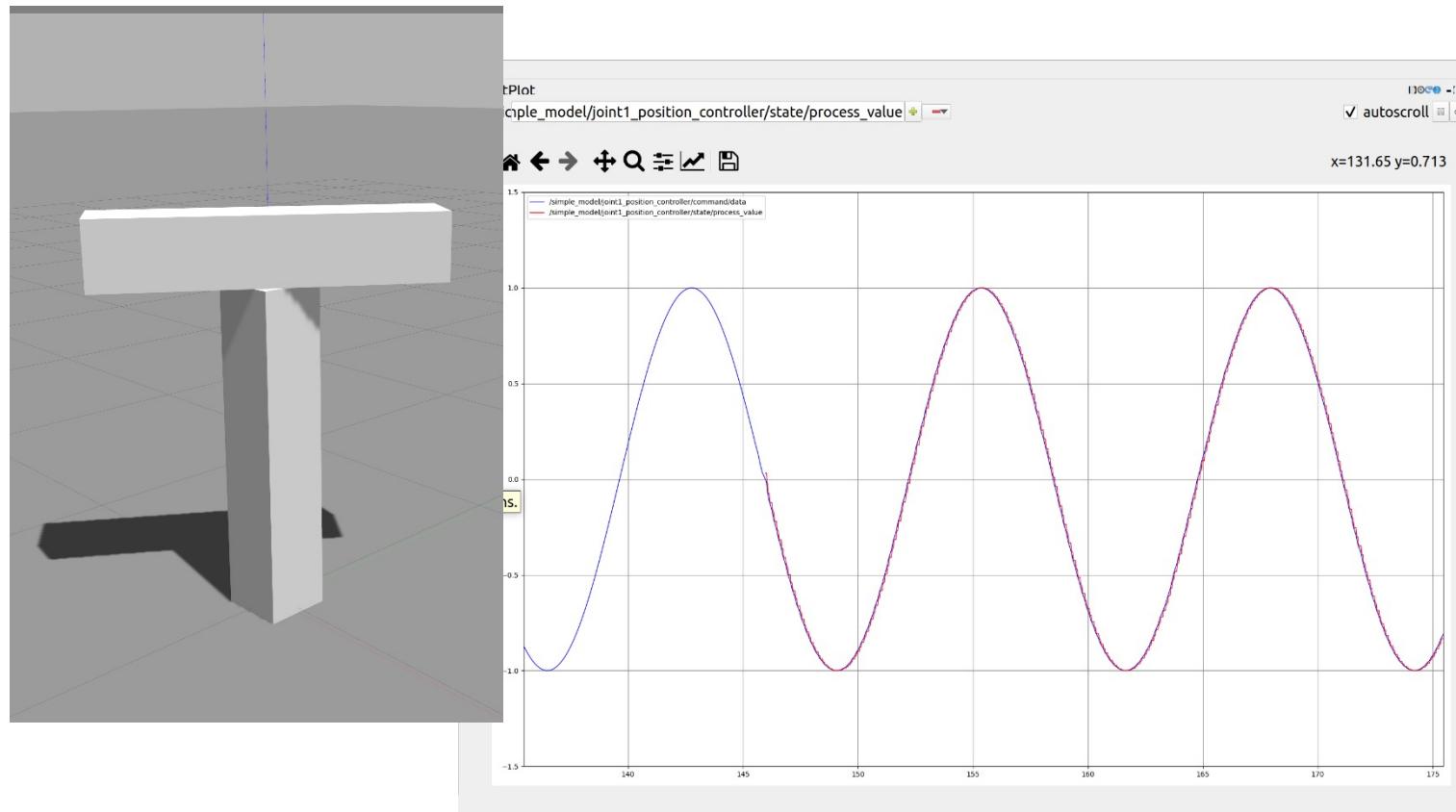
Box should be checked to
publish the message

Command input

ROS control

Visualizing the controller performance

- ▶ We can see how well the controller does with respect to the reference input
 - ▶ Go to Plugins -> Visualization -> Plot



Add sensors

Add 'ray' link and joint

```

<link name='ray'>
  <inertial>
    <origin xyz="0 0 0.05" rpy="0 0 0"/>
    <mass value="0.001"/>
    <inertia
      ixx="0.0166667" ixy="0.0" ixz="0.0"
      iyy="0.0166667" iyz="0.0"
      izz="0.0166667"/>
  </inertial>
  <visual>
    <origin rpy="0 0 0" xyz = "0 0 0.05"/>
    <geometry>
      <cylinder length="0.1" radius = "0.04"/>
    </geometry>
  </visual>
  <collision>
    <geometry>
      <cylinder length="0.1" radius = "0.04"/>
    </geometry>
  </collision>
</link>

<joint name="link_1_ray" type="fixed">
  <parent link = "link_1"/>
  <child link="ray"/>
  <origin rpy="1.5707 0 0" xyz = "0.5 0 0"/>
</joint>

```

Add sensors

Add 'ray' link and joint

```

<gazebo reference="ray">
  <sensor type="ray" name="lidar">
    <pose>0 0 0 1.5707 0 0</pose>  Horizontal becomes vertical
    <visualize>true</visualize>  Enable visualization in GUI
    <update_rate>30</update_rate>
    <ray>
      <scan>  Horizontal beam
        <horizontal>  Number of beams
          <samples>32</samples>
          <resolution>1</resolution>
          <min_angle>-0.53529248</min_angle>
          <max_angle>0.18622663</max_angle>
        </horizontal>
      </scan>
      <range>
        <min>0.05</min>
        <max>70</max>
        <resolution>0.02</resolution>
      </range>
    </ray>
  </sensor>
</gazebo>

```

Connect to ROS

Just after </ray>

```
<plugin name = "gazebo_ros_lidar_publisher" filename = "libgazebo_ros_laser.so">
<topicName>/simple_lidar/scan</topicName>
<frameName>lidar_link</frameName>
</plugin>
```

- ▶ You can see the pointcloud form through RViz
 - ▶ Add by either display type or topic
 - ▶ Change the fixed frame to lidar_link

Build a simple machine by Gazebo GUI

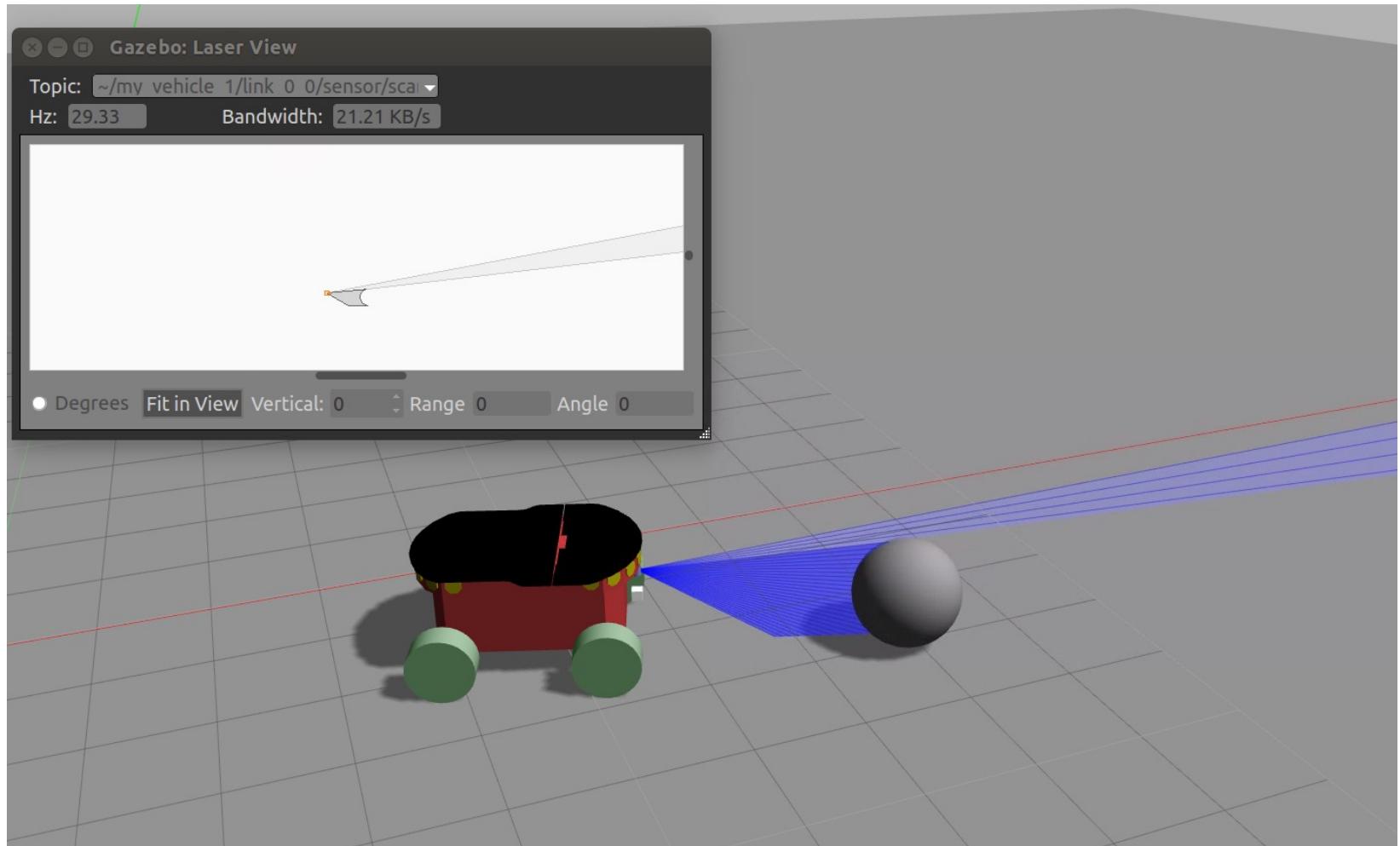
Build a simple machine by Gazebo GUI

Model editor Gazebo GUI

- ▶ Last chapter, we learned how to create a model by writing the model line by line
- ▶ It is also possible to use GUI in Gazebo to build models as well
- ▶ Go to Model Editor

Build a simple machine by Gazebo GUI

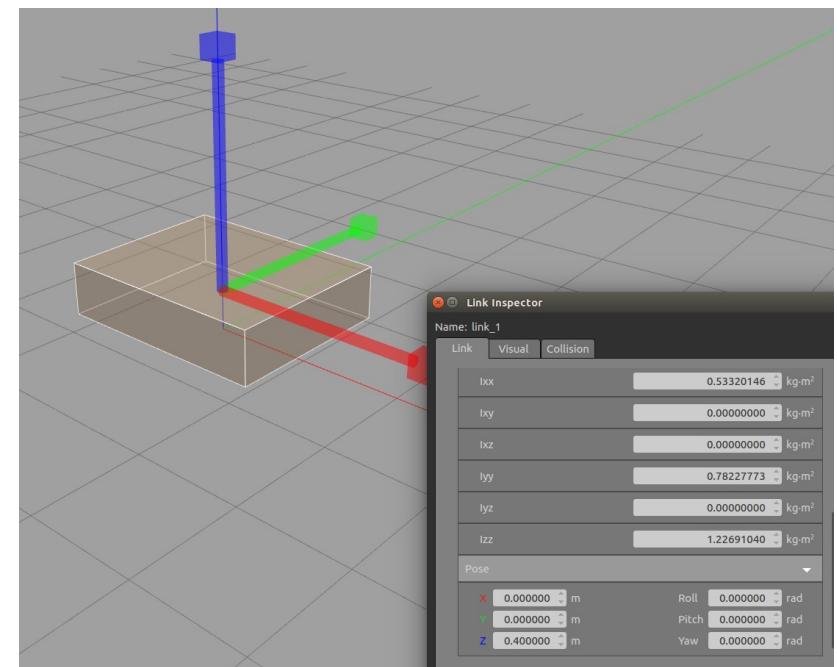
Simple vehicle with lidar



Build a simple machine by Gazebo GUI

Creating a vehicle

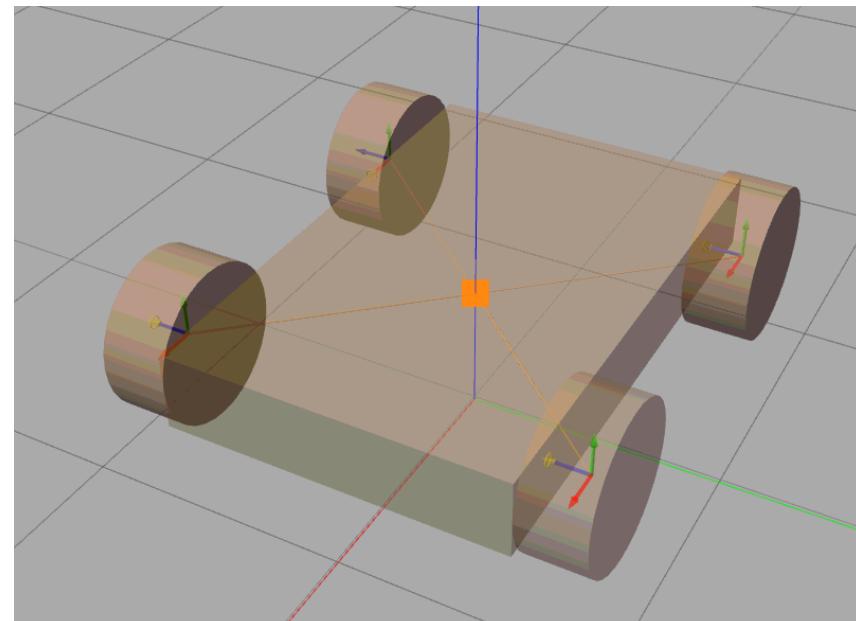
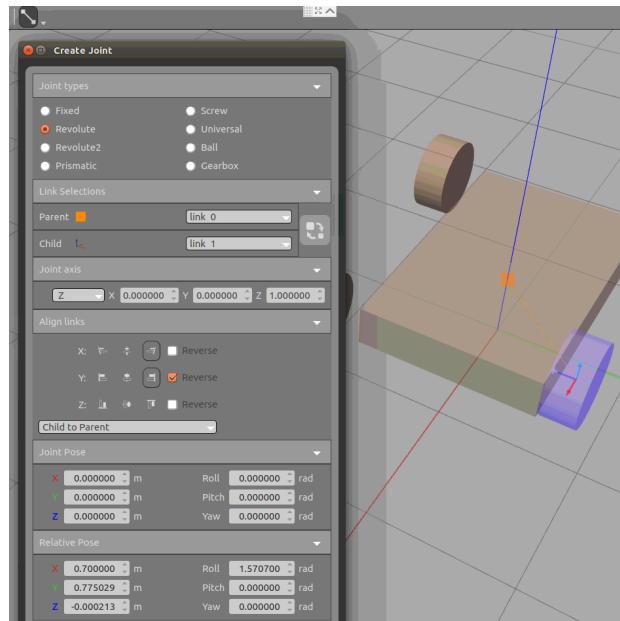
- ▶ Insert Box
- ▶ Use Scale mode to modify the size of the box ($x = 2m$, $y = 1.3m$ $z = 0.3m$)
- ▶ Open Link inspector -> set Pose z : 0.5m
- ▶ You can set the size of the box using Link inspector
- modify the values in both Visual tab, and Collision tab



Build a simple machine by Gazebo GUI

Creating a vehicle

- ▶ Insert Cylinder
- ▶ Open Link inspector -> set Link-Pose : Roll = 1.5707 rad
- ▶ set Visual, Collision - Geometry : r = 0.3m, length = 0.25m
- ▶ Copy and paste
- ▶ Create Joint
 - Revolute, Parent : link (box), Child : link_0(wheel), Joint axis : -Z, Align



Build a simple machine by Gazebo GUI

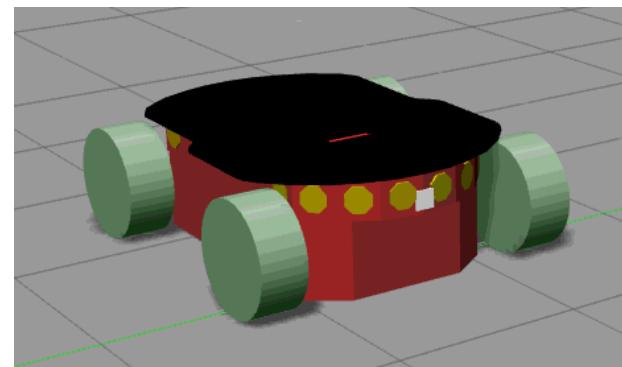
Insert mesh

- ▶ We have to edit the Gazebo SDF file.
- ▶ Find the visual tag of main link (body link) of the vehicle
- ▶ Comment out the original visual code using `<!-- ... -->`
- ▶ Instead, write :

```

<pose>0 0 0.3 0 0 0</pose>
<geometry>
  <mesh>
    <uri>model://pioneer2dx/meshes/chassis.dae</uri>
    <scale>4 4 4</scale>
  </mesh>
</geometry>

```



ROS control

Controlling the vehicle

- ▶ We will use Skid steering Drive plugin (http://gazebosim.org/tutorials?tut=ros_gzplugins)

```
<plugin name='skid_steer_drive_controller' filename='libgazebo_ros_skid_steer_drive.so'>
  <updateRate>5.0</updateRate>
  <robotNamespace>/</robotNamespace>
  <leftFrontJoint>front_left_wheel_joint</leftFrontJoint>
  <rightFrontJoint>front_right_wheel_joint</rightFrontJoint>
  <leftRearJoint>back_left_wheel_joint</leftRearJoint>
  <rightRearJoint>back_right_wheel_joint</rightRearJoint>
  <wheelSeparation>1.5</wheelSeparation>
  <wheelDiameter>0.6</wheelDiameter>
  <robotBaseFrame>base_link</robotBaseFrame>
  <torque>20</torque>
  <broadcastTF>false</broadcastTF>
</plugin>
```

Add sensors

Add a ray sensor

- ▶ Edit model my_vehicle
- ▶ Insert cylinder Link-pose : $x = 0.8\text{m}$, $z = 0.4\text{m}$
Visual, Collision : $R = 0.04\text{m}$, Length = 0.1m
- ▶ Create joint – fixed (Parent : link)
- ▶ Save model as ‘my_vehicle’ and exit the ‘model editor’
- ▶ Go to model.sdf, find the ray sensor link

Add sensors

Just after </collision> and before </sensor>

```

<sensor type='ray' name='lidar'>
  <pose>0 0 0 1.5707 0 0 </pose>
  <visualize>true</visualize>
  <update_rate>30</update_rate>
  <ray>
    <scan>
      <horizontal>
        <samples>32</samples>
        <resolution>1</resolution>
        <min_angle>-0.53529248</min_angle>
        <max_angle>0.18622663</max_angle>
      </horizontal>
    </scan>
    <range>
      <min>0.05</min>
      <max>70</max>
      <resolution>0.02</resolution>
    </range>
  </ray>
</sensor>

```

Horizontal becomes vertical
Enable visualization in GUI

Horizontal beam
Number of beams

http://gazebosim.org/tutorials?cat=guided_i&tut=guided_i_l

Connect to ROS

Just after </ray>

```
<plugin name='gazebo_ros_lidar_publisher' filename='libgazebo_ros_laser.so'>
<topicName>/my_vehicle/laser/scan</topicName>
<frameName>lidar_link</frameName>
</plugin>
```

- ▶ You can see the pointcloud form through RViz
 - ▶ Add by either display type or topic
 - ▶ Change the fixed frame to lidar_link

Build a simple machine

Difference between URDF and using Gazebo gui

- ▶ The file created is in sdf format instead of xacro
- ▶ Xacro / Universal Robotic Description Format (URDF)
 - ▶ Established format for describing robot structure in ROS
 - ▶ In Gazebo, URDF is converted to and read as SDF
 - ▶ Lacks friction and other properties
 - ▶ Cannot specify things that are not robot such as lights
- ▶ Simulation Description Format (SDF)
 - ▶ Devised by Gazebo to meet simulation needs
 - ▶ Can specify the pose of the robot within a world
 - ▶ Solves the shortcomings of URDF/xacro
- ▶ Later, you can combine the mobile robot(sdf) and the actuator(xacro) by converting the sdf file into xacro format and transferring the robot model code to the actuator model code
- ▶ This is necessary because ROS Control requires transmission tag that is not available in SDF

Assignment #2

Assignment # 2

Make your own mobile robot

- It is an assignment that I want to help with your future research.
(Free Format: Bicycle, Tricycle, Car, or even if the four-legged robot)
- Create the model file / launch file
- Submit a compressed package and a short video of the screen, which must include a gazebo window. Capturing rqt, rviz, and terminal window is an option.
- It doesn't matter to you submit the robot you made in class (you will only get basic scores)
- Explain the Xacro file or SDF file in the report and write why it is valid.
- If you copy someone else's, you'll get zero points and punishment.

Thank you!
