Moving the Robot

Estimated time to completion: **20 minutes**

5.6 Move a laser up and down by position

Step 1: Create the new files

Create the new files and test them in the simulation:

▶ Execute in Terminal 1

In [ ]:
```
cd ~/ros2_ws/src
```

In [ ]:
```
touch my_box_bot_gazebo/launch/control.launch.py
```

In [ ]:
```
touch my_box_bot_gazebo/launch/spawn_robot_ros2_control_complete.launch.xml
```

In [ ]:
```
mkdir -p my_box_bot_description/config
```

In [ ]:
```
touch my_box_bot_description/config/controller_position.yaml
```

In [ ]:
```
touch my_box_bot_description/launch/urdf_visualize_control_complete.launch.py
```

In [ ]:

```
cp my_box_bot_description/urdf/box_bot_physcal_control.urdf my_box_bot_description/urdf/box_bot_control_complete.urdf
```

Step 2: Update the URDF model to have a laser model link

Add the following lines at the bottom of it just before the `</robot>` tag, to have a laser model 3D mesh that represents the laser:

+ box_bot_control_complete.urdf

In [ ]:

```
<!-- Laser Position Control-->

<link name="laser_scan_link">
    <visual>
      <origin rpy="0 0 0" xyz="0 0 0"/>
      <geometry>
        <mesh filename="package://my_box_bot_description/meshes/sensors/rplidar.dae" scale="1.0 1.0 1.0"/>
      </geometry>
    </visual>

    <collision>
       <origin rpy="0 0 0" xyz="0 0 0.0204"/>
      <geometry>
        <box size="0.074986 0.074935 0.0408"/>
      </geometry>
    </collision>

    <inertial>

      <mass value="0.01"/>
      <origin rpy="0 0 0" xyz="0 0 0.0204"/>
      <inertia ixx="6.066578520833334e-06" ixy="0" ixz="0" iyy="6.072950163333333e-06" iyz="0" izz="9.365128684166666e-06"/>
    </inertial>
</link>

<joint name="laser_scan_link_joint" type="prismatic">
    <origin rpy="0 0 0" xyz="0.0 0.0 0.05"/>
    <parent link="chassis"/>
    <child link="laser_scan_link"/>
    <axis xyz="0 0 1"/>
    <limit lower="-0.1" upper="0.0" effort="20.0" velocity="2.0"/>
    <dynamics damping="0.1" friction="1.0"/>
</joint>

<link name="laser_scan_frame">
</link>

<joint name="laser_scan_frame_joint" type="fixed">
    <origin rpy="0 0 0" xyz="0 0 0.03"/>
```

```
    <parent link="laser_scan_link"/>
    <child link="laser_scan_frame"/>
    <axis xyz="0 0 0"/>
</joint>
```

Review certain elements:

In [ ]:
```
<link name="laser_scan_link">
    <visual>
      <origin rpy="0 0 0" xyz="0 0 0"/>
      <geometry>
        <mesh filename="package://my_box_bot_description/meshes/sensors/rplidar.dae" scale="1.0 1.0 1.0"/>
      </geometry>
    </visual>

    <collision>
       <origin rpy="0 0 0" xyz="0 0 0.0204"/>
      <geometry>
        <cylinder length="0.0408" radius="0.037493"/>
      </geometry>
    </collision>

    <inertial>

      <mass value="0.01"/>
      <origin rpy="0 0 0" xyz="0 0 0.0204"/>
      <inertia ixx="6.066578520833334e-06" ixy="0" ixz="0" iyy="6.072950163333333e-06" iyz="0" izz="9.365128684166666e-06"/>
    </inertial>
</link>
```

Here, create the **laser_scan_link**. This will visually show the mesh **rplidar.dae** you copied in the previous unit to use the chassis model **cute_cube.dae**.

For collision, add a cylinder shape that fits best the 3D mesh to lower the physics calculations instead of using the mesh directly.

Add a small mass of 0.01 kg.

In [ ]:
```
<joint name="laser_scan_link_joint" type="prismatic">
    <origin rpy="0 0 0" xyz="0.0 0.0 0.05"/>
    <parent link="chassis"/>
    <child link="laser_scan_link"/>
    <axis xyz="0 0 1"/>
    <limit lower="-0.1" upper="0.0" effort="20.0" velocity="2.0"/>
    <dynamics damping="0.1" friction="1.0"/>
</joint>
```

Here, add a **prismatic joint**. Prismatic joints allow translation in one axis. In this case, the axis is the **Z-axis**.

Attach your **laser_scan_link** to the **chassis link** directly.

Set as lower limits -0.1 meters and **0.0 meters** for upper limits.

Set friction and damping to avoid the joint running crazy when moving, simulating a real joint.

As for **effort and velocity**, set these values by experimenting with which ones work the best in the simulation. Unfortunately, Gazebo does not provide tools to easily calculate these values.

In [ ]:
```
<link name="laser_scan_frame">
</link>

<joint name="laser_scan_frame_joint" type="fixed">
    <origin rpy="0 0 0" xyz="0 0 0.03"/>
    <parent link="laser_scan_link"/>
    <child link="laser_scan_frame"/>
    <axis xyz="0 0 0"/>
</joint>
```

Add a simple link to represent the frame where the sensor's center would be. It is a utility for simulating sensor positioning.

Step 3: Add the ros2_control XML tags

Now add the **Gazebo plugin system** that allows you to control the joints and broadcast their state in ROS.

Open up **box_bot_control_complete.urdf** in the IDE and add the following lines at the bottom before the `</robot>` tag.

+ box_bot_control_complete.urdf

In [ ]:
```
<!-- Position Config -->
    <ros2_control name="GazeboSystem" type="system">
      <hardware>
        <plugin>gazebo_ros2_control/GazeboSystem</plugin>
      </hardware>

      <joint name="laser_scan_link_joint">
        <command_interface name="position">
          <param name="min">-0.05</param>
          <param name="max">0.0</param>
        </command_interface>
        <state_interface name="position"/>
        <state_interface name="velocity"/>
        <state_interface name="effort"/>
      </joint>

  </ros2_control>



  <gazebo>
    <plugin filename="libgazebo_ros2_control.so" name="gazebo_ros2_control">
      <parameters>$(find my_box_bot_description)/config/controller_position.yaml</parameters>
      <robot_param_node>/my_robot_state_publisher_node</robot_param_node>
    </plugin>
  </gazebo>
```

Review the code:

In [ ]:

```
<ros2_control name="GazeboSystem" type="system">
      <hardware>
        <plugin>gazebo_ros2_control/GazeboSystem</plugin>
      </hardware>

      <joint name="laser_scan_link_joint">
        <command_interface name="position">
          <param name="min">-0.05</param>
          <param name="max">0.0</param>
        </command_interface>
        <state_interface name="position"/>
        <state_interface name="velocity"/>
        <state_interface name="effort"/>
      </joint>

</ros2_control>
```

- Now, use this **ros2_control** tag in URDFs in ROS2. Here, you can state loads of things related to ROS.
- In this case, you are adding the **gazebo_ros2_control/GazeboSystem** plugin.
- And with that, you state **all the joints you want to be controlled by ROS2 systems**.
- In this case, you have only **laser_scan_link_joint**, using the command interface named **position**.
- State the limits min and max and the state interface options - position, velocity, and effort.

In [ ]:

```
  <gazebo>
    <plugin filename="libgazebo_ros2_control.so" name="gazebo_ros2_control">
      <parameters>$(find my_box_bot_description)/config/controller_position.yaml</parameters>
      <robot_param_node>/my_robot_state_publisher_node</robot_param_node>
    </plugin>
  </gazebo>
```

And finally, the part of the `gazebo_ros2_control`. This is the plugin managing the controllers and moving the joints needed.

- Use the `robot_param_node` tag to specify the **NAME of the robot_state_publisher node**. This is because this plugin needs access to certain `services` whose **NAMES** depend on the name given to the **ROBOT STATE PUBLISHER NODE**.
- In your case, start the **Robot State Publisher node** like this, with the name `my_robot_state_publisher_node`.

In [ ]:
```
robot_state_publisher_node = Node(
    package='robot_state_publisher',
    executable='robot_state_publisher',
    name='my_robot_state_publisher_node',
    emulate_tty=True,
    parameters=[{'use_sim_time': True, 'robot_description': Command(['xacro ', robot_desc_path])}],
    output="screen"
)
```

- This means that your plugin node *(gazebo_ros2_control)* is looking for the services named:
  - `/my_robot_state_publisher_node/describe_parameters`
  - `/my_robot_state_publisher_node/get_parameter_types`
  - `/my_robot_state_publisher_node/get_parameters`
  - `/my_robot_state_publisher_node/list_parameters`
  - `/my_robot_state_publisher_node/set_parameters`
  - `/my_robot_state_publisher_node/set_parameters_atomically`
- By default, if you do not use the tag `robot_param_node` , the services it needs will assume that the `robot_state publisher node name` is **robot_state_publisher**:
  - `/robot_state_publisher/describe_parameters`
  - `/robot_state_publisher/get_parameter_types`
  - `/robot_state_publisher/get_parameters`
  - `/robot_state_publisher/list_parameters`
  - `/robot_state_publisher/set_parameters`
  - `/robot_state_publisher/set_parameters_atomically`

Step 4: Add the configuration file for ros2_control

To know which joints and how they must move, specify that inside the `controller_position.yaml` under the `joints` parameter by listing the names using dashes (`-`). The indentation level of new joints must match the indentation shown for `laser_scan_link_joint` shown below.

controller_position.yaml

In [ ]:

```
controller_manager:
  ros__parameters:
    update_rate: 100  # Hz

    joint_trajectory_controller:
      type: joint_trajectory_controller/JointTrajectoryController

    joint_state_broadcaster:
      type: joint_state_broadcaster/JointStateBroadcaster

joint_trajectory_controller:
  ros__parameters:
    joints:
      - laser_scan_link_joint
    interface_name: position
    command_interfaces:
      - position
    state_interfaces:
      - position
      - velocity
```

Some things to note:

- Set the **controller_manager**:
  - update_rate = 100
  - That you have a controller named **joint_trajectory_controller** of type **joint_trajectory_controller/JointTrajectoryController**
  - That you have a controller named **joint_state_broadcaster** of type **joint_state_broadcaster/JointStateBroadcaster**
- Define the **joint_trajectory_controller** as having:
  - joints = laser_scan_link_joint. You will only control this joint.
  - interface_name = position. This is the name you gave the interface in the URDF.
  - Set both the command and state interfaces, which define how you interact and control the joint. In this case, you are using position values. If you want it to move, tell the joint where to move in the **Z-axis position**, within the upper and lower limits stated.

Step 5: Fill in the code for the new launch file

control.launch.py

In [ ]:

```python
#!/usr/bin/python3
# -*- coding: utf-8 -*-
from launch_ros.actions import Node
from launch import LaunchDescription


# this is the function launch  system will look for


def generate_launch_description():


    spawn_controller = Node(
        package="controller_manager",
        executable="spawner",
        arguments=["joint_state_broadcaster"],
        output="screen",
    )

    spawn_controller_traj = Node(
        package="controller_manager",
        executable="spawner",
        arguments=["joint_trajectory_controller"],
        output="screen",
    )



    # create and return launch description object
    return LaunchDescription(
        [
            spawn_controller,
            spawn_controller_traj,
        ]
    )
```

You need this launch **control.launch.py** because you have to **load the controllers explicitly**. In this case, you are using the **spawner** from **controller_manager** to load:

- joint_trajectory_controller: the one that deals with moving the joints through trajectory messages.
- joint_state_broadcaster: The one that broadcasts the joint states of those particular joints.

These were defined inside the **controller_position.yaml**.

Here is the complete code for the robot description, including controllers:

box_bot_control_complete.urdf

In [ ]:

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

  <material name="red">
      <color rgba="1.0 0.0 0.0 1"/>
  </material>

  <material name="green_light">
      <color rgba="0.0 1.0 0.0 1"/>
  </material>

  <material name="green_dark">
    <color rgba="0.0 0.5 0.0 1"/>
  </material>

  <material name="blue">
      <color rgba="0.0 0.0 1.0 1"/>
  </material>

  <link name="base_link">
  </link>


  <!-- Body -->
  <link name="chassis">
    <visual>
      <geometry>
        <mesh filename="package://my_box_bot_description/meshes/cute_cube.dae" scale="0.1 0.1 0.1"/>
      </geometry>
    </visual>

    <collision>
      <geometry>
        <box size="0.1 0.1 0.1"/>
      </geometry>
    </collision>

    <inertial>
      <mass value="0.5"/>
```

```
      <origin rpy="0 0 0" xyz="0 0 0"/>
      <inertia ixx="0.0008333333333333335" ixy="0" ixz="0" iyy="0.0008333333333333335" iyz="0" izz="0.0008333333333333335"/>
    </inertial>

  </link>

  <joint name="base_link_joint" type="fixed">
    <origin rpy="0 0 0" xyz="0 0 0" />
    <parent link="base_link" />
    <child link="chassis" />
  </joint>

  <!-- Wheel Left -->
  <link name="left_wheel">
      <visual>
        <origin rpy="0 1.5707 1.5707" xyz="0 0 0"/>
        <geometry>
          <cylinder length="0.001" radius="0.035"/>
        </geometry>
        <material name="red"/>
      </visual>

      <collision>
        <origin rpy="0 1.5707 1.5707" xyz="0 0 0"/>
        <geometry>
          <cylinder length="0.001" radius="0.035"/>
        </geometry>
      </collision>

      <inertial>
        <origin rpy="0 1.5707 1.5707" xyz="0 0 0"/>
        <mass value="0.05"/>
        <inertia ixx="1.5316666666666667e-05" ixy="0" ixz="0" iyy="1.5316666666666667e-05" iyz="0" izz="3.0625000000000006e-05"/>
      </inertial>

  </link>

  <gazebo reference="left_wheel">
    <kp>1000000000000000000000000000.0</kp>
    <kd>1000000000000000000000000000.0</kd>
```

```
    <mu1>10.0</mu1>
    <mu2>10.0</mu2>
    <material>Gazebo/Green</material>
  </gazebo>


  <joint name="joint_left_wheel" type="continuous">
    <origin rpy="0 0 0" xyz="0 0.05 -0.025"/>
    <child link="left_wheel"/>
    <parent link="chassis"/>
    <axis rpy="0 0 0" xyz="0 1 0"/>
    <limit effort="10000" velocity="1000"/>
    <joint_properties damping="1.0" friction="1.0"/>
  </joint>

  <!-- Wheel Right -->
  <link name="right_wheel">
      <visual>
        <origin rpy="0 1.5707 1.5707" xyz="0 0 0"/>
        <geometry>
          <cylinder length="0.001" radius="0.035"/>
        </geometry>
        <material name="green"/>
      </visual>

      <collision>
        <origin rpy="0 1.5707 1.5707" xyz="0 0 0"/>
        <geometry>
          <cylinder length="0.001" radius="0.035"/>
        </geometry>
      </collision>

      <inertial>
        <origin rpy="0 1.5707 1.5707" xyz="0 0 0"/>
        <mass value="0.05"/>
        <inertia ixx="1.5316666666666667e-05" ixy="0" ixz="0" iyy="1.5316666666666667e-05" iyz="0" izz="3.0625000000000006e-05"/>
      </inertial>
  </link>

  <gazebo reference="right_wheel">
```

```
    <kp>1000000000000000000000000000.0</kp>
    <kd>1000000000000000000000000000.0</kd>
    <mu1>10.0</mu1>
    <mu2>10.0</mu2>
    <material>Gazebo/Orange</material>
  </gazebo>

  <joint name="joint_right_wheel" type="continuous">
    <origin rpy="0 0 0" xyz="0 -0.05 -0.025"/>
    <child link="right_wheel"/>
    <parent link="chassis"/>
    <axis rpy="0 0 0" xyz="0 1 0"/>
    <limit effort="10000" velocity="1000"/>
    <joint_properties damping="1.0" friction="1.0"/>
  </joint>


  <!-- Caster Wheel Front -->
  <link name="front_yaw_link">
      <visual>
        <origin rpy="0 1.5707 1.5707" xyz="0 0 0"/>
        <geometry>
          <cylinder length="0.001" radius="0.0045000000000000005"/>
        </geometry>
        <material name="blue"/>
      </visual>

      <collision>
        <origin rpy="0 1.5707 1.5707" xyz="0 0 0"/>
        <geometry>
          <cylinder length="0.001" radius="0.0045000000000000005"/>
        </geometry>
      </collision>

      <inertial>
          <origin rpy="0 1.5707 1.5707" xyz="0 0 0"/>
          <mass value="0.001"/>
          <inertia ixx="5.145833333333334e-09" ixy="0" ixz="0" iyy="5.145833333333334e-09" iyz="0" izz="1.0125000000000003e-08"/>
      </inertial>
```

```xml
  </link>

  <joint name="front_yaw_joint" type="continuous">
    <origin rpy="0 0 0" xyz="0.04 0 -0.05" />
    <parent link="chassis" />
    <child link="front_yaw_link" />
    <axis xyz="0 0 1" />
    <limit effort="1000.0" velocity="100.0" />
    <dynamics damping="0.0" friction="0.1"/>
  </joint>

    <gazebo reference="front_yaw_link">
        <material>Gazebo/Blue</material>
    </gazebo>



  <link name="front_roll_link">
      <visual>
        <origin rpy="0 1.5707 1.5707" xyz="0 0 0"/>
        <geometry>
          <cylinder length="0.001" radius="0.0045000000000000005"/>
        </geometry>
        <material name="red"/>
      </visual>

      <collision>
        <origin rpy="0 1.5707 1.5707" xyz="0 0 0"/>
        <geometry>
          <cylinder length="0.001" radius="0.0045000000000000005"/>
        </geometry>
      </collision>

      <inertial>
          <origin rpy="0 1.5707 1.5707" xyz="0 0 0"/>
          <mass value="0.001"/>
          <inertia ixx="5.145833333333334e-09" ixy="0" ixz="0" iyy="5.145833333333334e-09" iyz="0" izz="1.0125000000000003e-08"/>
      </inertial>
  </link>
```

```
<joint name="front_roll_joint" type="continuous">
  <origin rpy="0 0 0" xyz="0 0 0" />
  <parent link="front_yaw_link" />
  <child link="front_roll_link" />
  <axis xyz="1 0 0" />
  <limit effort="1000.0" velocity="100.0" />
  <dynamics damping="0.0" friction="0.1"/>
</joint>

  <gazebo reference="front_roll_link">
      <material>Gazebo/Red</material>
  </gazebo>


<link name="front_pitch_link">
  <visual>
    <origin rpy="0 1.5707 1.5707" xyz="0 0 0"/>
    <geometry>
      <sphere radius="0.010"/>
    </geometry>
    <material name="green_dark"/>
  </visual>

  <collision>
    <origin rpy="0 1.5707 1.5707" xyz="0 0 0"/>
    <geometry>
      <sphere radius="0.010"/>
    </geometry>
  </collision>

  <inertial>
      <origin rpy="0 1.5707 1.5707" xyz="0 0 0"/>
      <mass value="0.001"/>
      <inertia ixx="4e-08" ixy="0" ixz="0" iyy="4e-08" iyz="0" izz="4e-08"/>
  </inertial>
</link>

<gazebo reference="front_pitch_link">
  <kp>1000000000000000000000000000.0</kp>
  <kd>1000000000000000000000000000.0</kd>
```

```
    <mu1>0.5</mu1>
    <mu2>0.5</mu2>
    <material>Gazebo/Purple</material>
  </gazebo>

  <joint name="front_pitch_joint" type="continuous">
    <origin rpy="0 0 0" xyz="0 0 0" />
    <parent link="front_roll_link" />
    <child link="front_pitch_link" />
    <axis xyz="0 1 0" />
    <limit effort="1000.0" velocity="100.0" />
    <dynamics damping="0.0" friction="0.1"/>
  </joint>

<!-- Caster Wheel Back -->
  <link name="back_yaw_link">
    <visual>
        <origin rpy="0 1.5707 1.5707" xyz="0 0 0"/>
        <geometry>
          <cylinder length="0.001" radius="0.0045000000000000005"/>
        </geometry>
        <material name="blue"/>
      </visual>

      <collision>
        <origin rpy="0 1.5707 1.5707" xyz="0 0 0"/>
        <geometry>
          <cylinder length="0.001" radius="0.0045000000000000005"/>
        </geometry>
      </collision>

      <inertial>
          <origin rpy="0 1.5707 1.5707" xyz="0 0 0"/>
          <mass value="0.001"/>
          <inertia ixx="5.145833333333334e-09" ixy="0" ixz="0" iyy="5.145833333333334e-09" iyz="0" izz="1.0125000000000003e-08"/>
      </inertial>
  </link>

  <joint name="back_yaw_joint" type="continuous">
    <origin rpy="0 0 0" xyz="-0.04 0 -0.05" />
```

```xml
    <parent link="chassis" />
    <child link="back_yaw_link" />
    <axis xyz="0 0 1" />
    <limit effort="1000.0" velocity="100.0" />
    <dynamics damping="0.0" friction="0.1"/>
  </joint>

    <gazebo reference="back_yaw_link">
        <material>Gazebo/Blue</material>
    </gazebo>



  <link name="back_roll_link">
      <visual>
        <origin rpy="0 1.5707 1.5707" xyz="0 0 0"/>
        <geometry>
          <cylinder length="0.001" radius="0.0045000000000000005"/>
        </geometry>
        <material name="red"/>
      </visual>

      <collision>
        <origin rpy="0 1.5707 1.5707" xyz="0 0 0"/>
        <geometry>
          <cylinder length="0.001" radius="0.0045000000000000005"/>
        </geometry>
      </collision>

      <inertial>
          <origin rpy="0 1.5707 1.5707" xyz="0 0 0"/>
          <mass value="0.001"/>
          <inertia ixx="5.145833333333334e-09" ixy="0" ixz="0" iyy="5.145833333333334e-09" iyz="0" izz="1.0125000000000003e-08"/>
      </inertial>
  </link>

  <joint name="back_roll_joint" type="continuous">
    <origin rpy="0 0 0" xyz="0 0 0" />
    <parent link="back_yaw_link" />
    <child link="back_roll_link" />
```

```xml
    <axis xyz="1 0 0" />
    <limit effort="1000.0" velocity="100.0" />
    <dynamics damping="0.0" friction="0.1"/>
  </joint>

    <gazebo reference="back_roll_link">
        <material>Gazebo/Red</material>
    </gazebo>



  <link name="back_pitch_link">
    <visual>
      <origin rpy="0 1.5707 1.5707" xyz="0 0 0"/>
      <geometry>
        <sphere radius="0.010"/>
      </geometry>
      <material name="green_light"/>
    </visual>

    <collision>
      <origin rpy="0 1.5707 1.5707" xyz="0 0 0"/>
      <geometry>
        <sphere radius="0.010"/>
      </geometry>
    </collision>

    <inertial>
        <origin rpy="0 1.5707 1.5707" xyz="0 0 0"/>
        <mass value="0.001"/>
        <inertia ixx="4e-08" ixy="0" ixz="0" iyy="4e-08" iyz="0" izz="4e-08"/>
    </inertial>
  </link>

  <gazebo reference="back_pitch_link">
    <kp>1000000000000000000000000000.0</kp>
    <kd>1000000000000000000000000000.0</kd>
    <mu1>0.5</mu1>
    <mu2>0.5</mu2>
    <material>Gazebo/Yellow</material>
```

```xml
</gazebo>

<joint name="back_pitch_joint" type="continuous">
  <origin rpy="0 0 0" xyz="0 0 0" />
  <parent link="back_roll_link" />
  <child link="back_pitch_link" />
  <axis xyz="0 1 0" />
  <limit effort="1000.0" velocity="100.0" />
  <dynamics damping="0.0" friction="0.1"/>
</joint>

<!-- PLUGINS -->

<!-- JOINT PUBLISHER -->
<gazebo>
  <plugin name="box_bot_joint_state" filename="libgazebo_ros_joint_state_publisher.so">
    <ros>
        <remapping>~/out:=joint_states</remapping>
    </ros>
    <update_rate>30</update_rate>

    <joint_name>joint_left_wheel</joint_name>
    <joint_name>joint_right_wheel</joint_name>
    <joint_name>front_yaw_joint</joint_name>
    <joint_name>back_yaw_joint</joint_name>
    <joint_name>front_roll_joint</joint_name>
    <joint_name>back_roll_joint</joint_name>
    <joint_name>front_pitch_joint</joint_name>
    <joint_name>back_pitch_joint</joint_name>

  </plugin>
</gazebo>

<!-- Differential drive -->
<gazebo>
  <plugin filename="libgazebo_ros_diff_drive.so" name="differential_drive_controller">

    <!-- wheels -->
    <left_joint>joint_left_wheel</left_joint>
    <right_joint>joint_right_wheel</right_joint>
```

```
    <!-- kinematics -->
    <wheel_separation>0.1</wheel_separation>
    <wheel_diameter>0.07</wheel_diameter>

    <!-- limits -->
    <max_wheel_torque>1.0</max_wheel_torque>
    <max_wheel_acceleration>2.0</max_wheel_acceleration>

    <!-- output -->
    <publish_odom>true</publish_odom>
    <publish_odom_tf>true</publish_odom_tf>

    <odometry_frame>odom</odometry_frame>
    <robot_base_frame>base_link</robot_base_frame>

  </plugin>
</gazebo>


  <!-- Laser Position Control-->

<link name="laser_scan_link">
  <visual>
    <origin rpy="0 0 0" xyz="0 0 0"/>
    <geometry>
      <mesh filename="package://my_box_bot_description/meshes/sensors/rplidar.dae" scale="1.0 1.0 1.0"/>
    </geometry>
  </visual>

  <collision>
     <origin rpy="0 0 0" xyz="0 0 0.0204"/>
    <geometry>
      <cylinder length="0.0408" radius="0.037493"/>
    </geometry>
  </collision>

  <inertial>

    <mass value="0.01"/>
```

```xml
      <origin rpy="0 0 0" xyz="0 0 0.0204"/>
      <inertia ixx="6.066578520833334e-06" ixy="0" ixz="0" iyy="6.072950163333333e-06" iyz="0" izz="9.365128684166666e-06"/>
    </inertial>
  </link>

  <joint name="laser_scan_link_joint" type="prismatic">
    <origin rpy="0 0 0" xyz="0.0 0.0 0.05"/>
    <parent link="chassis"/>
    <child link="laser_scan_link"/>
    <axis xyz="0 0 1"/>
    <limit lower="-0.1" upper="0.0" effort="20.0" velocity="2.0"/>
    <dynamics damping="0.1" friction="1.0"/>
  </joint>

  <link name="laser_scan_frame">
  </link>

  <joint name="laser_scan_frame_joint" type="fixed">
    <origin rpy="0 0 0" xyz="0 0 0.03"/>
    <parent link="laser_scan_link"/>
    <child link="laser_scan_frame"/>
    <axis xyz="0 0 0"/>
  </joint>

  <!-- Position Config -->
  <ros2_control name="GazeboSystem" type="system">
      <hardware>
        <plugin>gazebo_ros2_control/GazeboSystem</plugin>
      </hardware>

      <joint name="laser_scan_link_joint">
        <command_interface name="position">
          <param name="min">-0.05</param>
          <param name="max">0.0</param>
        </command_interface>
        <state_interface name="position"/>
        <state_interface name="velocity"/>
        <state_interface name="effort"/>
      </joint>
```

```
  </ros2_control>


  <gazebo>
    <plugin filename="libgazebo_ros2_control.so" name="gazebo_ros2_control">
      <parameters>$(find my_box_bot_description)/config/controller_position.yaml</parameters>
      <robot_param_node>/my_robot_state_publisher_node</robot_param_node>
    </plugin>
  </gazebo>


</robot>
```

What you have to have in your launch file is:

urdf_visualize_control_complete.launch.py

In [ ]:

```python
import os

from ament_index_python.packages import get_package_share_directory
from launch import LaunchDescription
from launch.substitutions import Command
from launch_ros.actions import Node

# this is the function launch  system will look for
def generate_launch_description():

    ####### DATA INPUT ##########
    urdf_file = 'box_bot_control_complete.urdf'
    #xacro_file = "box_bot.xacro"
    package_description = "my_box_bot_description"

    ####### DATA INPUT END ##########
    print("Fetching URDF ==>")
    robot_desc_path = os.path.join(get_package_share_directory(package_description), "urdf", urdf_file)

    # Robot State Publisher

    robot_state_publisher_node = Node(
        package='robot_state_publisher',
        executable='robot_state_publisher',
        name='my_robot_state_publisher_node',
        emulate_tty=True,
        parameters=[{'use_sim_time': True, 'robot_description': Command(['xacro ', robot_desc_path])}],
        output="screen"
    )

    # RVIZ Configuration
    rviz_config_dir = os.path.join(get_package_share_directory(package_description), 'rviz', 'urdf_vis.rviz')


    rviz_node = Node(
            package='rviz2',
            executable='rviz2',
            output='screen',
            name='rviz_node',
```

```
            parameters=[{'use_sim_time': True}],
            arguments=['-d', rviz_config_dir])

    # create and return launch description object
    return LaunchDescription(
        [
            robot_state_publisher_node,
            rviz_node
        ]
    )
```

Note the **name of the Robot State Publisher node**, `my_robot_state_publisher_node` :

In [ ]:
```
robot_state_publisher_node = Node(
        package='robot_state_publisher',
        executable='robot_state_publisher',
        name='my_robot_state_publisher_node',
        emulate_tty=True,
        parameters=[{'use_sim_time': True, 'robot_description': Command(['xacro ', robot_desc_path])}],
        output="screen"
    )
```

This is how you define the main entry point file that starts all other components:

spawn_robot_ros2_control_complete.launch.xml

In [ ]:
```
<?xml version='1.0' ?>
<launch>
    <!-- Publish URDF file in robot_description topic -->
    <include file="$(find-pkg-share my_box_bot_description)/launch/urdf_visualize_control_complete.launch.py"/>
    <!-- Read robot_description and spawn in gazebo running sim -->
    <include file="$(find-pkg-share my_box_bot_gazebo)/launch/spawn_robot_description.launch.py"/>
    <!-- Load the controllers -->
    <include file="$(find-pkg-share my_box_bot_gazebo)/launch/control.launch.py"/>
</launch>
```

controller_position.yaml

In [ ]:
```
controller_manager:
  ros__parameters:
    update_rate: 100  # Hz

    joint_trajectory_controller:
      type: joint_trajectory_controller/JointTrajectoryController

    joint_state_broadcaster:
      type: joint_state_broadcaster/JointStateBroadcaster

joint_trajectory_controller:
  ros__parameters:
    joints:
      - laser_scan_link_joint
    interface_name: position
    command_interfaces:
      - position
    state_interfaces:
      - position
      - velocity
```

And add the following to **my_box_bot_description/CMakeLists.txt** before the ament_package() call:

CMakeLists.txt

In [ ]:
```
...
install(
  DIRECTORY
    urdf
    rviz
    launch
    meshes
    config
  DESTINATION
    share/${PROJECT_NAME}/
)
...
```

Launch and see what happens:

▶ **Execute in Terminal 1**

In [ ]:
```
cd ~/ros2_ws
```

In [ ]:
```
colcon build
```

In [ ]:
```
source install/setup.bash
```

In [ ]:
```
ros2 launch my_box_bot_gazebo start_world.launch.py
```

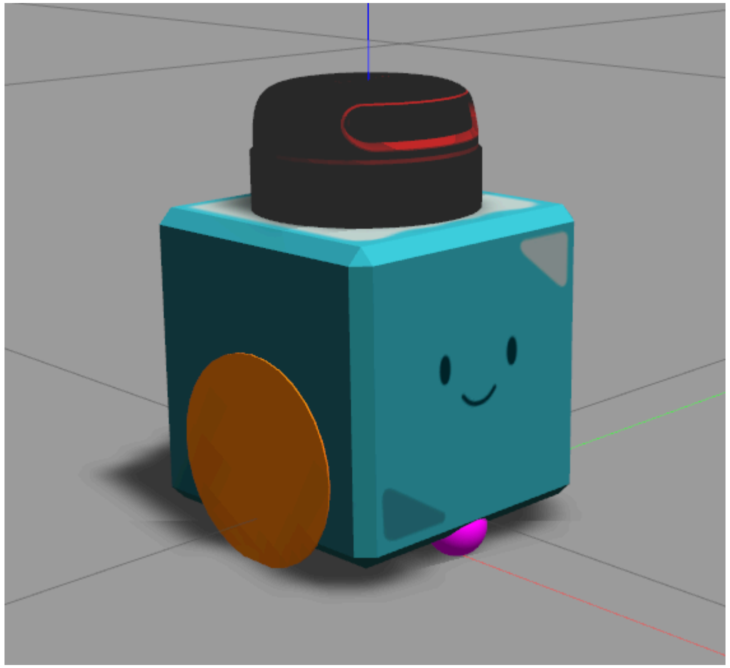
▶ **Execute in Terminal 2**

In [ ]:
```
cd ~/ros2_ws
```

In [ ]:
```
source install/setup.bash
```

In [ ]:
```
ros2 launch my_box_bot_gazebo spawn_robot_ros2_control_complete.launch.xml
```

Expected result:



You should see your robot with an **RPlidar** on top. But how can you move it? There are several ways, but the most comfortable and versatile is to create a Python script that handles everything, especially for trajectory messages needed here.

16/11/2023