Creating the package and Robot Description

So let's get started with Task 1B!

Instead of sharing a downloadable (to make yours (participants) and ours (theme devs) life easy) as we did in Task 0, we shall take a different (slightly more difficult) approach for this task! What's that and Why?

> **Assemble/Setup our package and urdf from scratch!** This time we shall go step by step and create most of the package from scratch! This should develop a clearer understanding of ROS and it will also be good practice/learning for future projects with ROS.
>
> (Parts of the package shall still be shared as a downloadable. Link for that shall come up as one continues reading this document)

Setup

1. Create another workspace inside `eyrc_hb` folder. The name of the workspace should be `hb_task1b_ws` .
2. Create a package using `ros2 pkg create` . The package will have dependency on rospy and gazebo. The name of this package should be `hb_task_1b`
3. Create a file named `hb_bot.urdf.xacro` in a subfolder `./src/hb_task_1b/urdf` .
4. Create a python file, `controller.py` in a subfolder `./src/hb_task_1b/scripts` .
   Note: *Remember how to create files and subolders using `mkdir -p` and `touch` commands?*
5. Finally, check if you have `teleop_twist_keyboard` package installed by running the command:
   `ros2 run teleop_twist_keyboard teleop_twist_keyboard`
   If not installed, run the command:
   `sudo apt-get install ros-humble-teleop-twist-keyboard`

If

the `hb_bot.urdf.xacro` file was ready, i.e.:

- `hb_bot.urdf.xacro` file would have the **definition** of all the **links** and **joints** on the robot, along with the planar move **gazebo plugin** to define a model for actuating and feedback.

Then

we would be able to

- run `ros2 launch hb_task1b_ws gazebo.launch.py` to open gazebo with the bot inside it
  This launch file will be provided by us. (Linking will be found in this document)

- and `rosrun teleop_twist_keyboard teleop_twist_keyboard.py` to control the simulated robot!

But

the `hb_bot.urdf.xacro` is not ready yet... So let's get started with that.

Robot Description

Now for the `hb_bot.urdf.xacro` file, Following is part of the contents of the urdf file:

```
<?xml version="1.0" ?>
<!--
*********************************************************************************************
*
*        ============================================
*                  HB Theme (eYRC 2023-24)
*        ============================================
*
*
*  Filename:            hb_bot.urdf.xacro
*  Created:
*  Last Modified:
*  Author:
*
*********************************************************************************************
-->



<robot name="hb_bot" xmlns:xacro="http://www.ros.org/wiki/xacro">
    <xacro:include filename="$(find holo)/urdf/materials.xacro"/>

    <link name ="footprint_link">
    </link>
    <joint name="footprint_joint" type="fixed">
         <origin
                xyz="0.0 0.0 0.0"
                rpy="0 0 0" />
        <parent link="footprint_link"/>
        <child link="base_link"/>


    </joint>

    <!--Base link-->
    <link name ="base_link">
        <inertial>
            <origin xyz="0 0.0 0.28" rpy="0 0 0" />
            <mass
                value="0.28" />
            <inertia
                ixx="0.011666666666667"
                ixy="0"
                ixz="0"
                iyy="0.011666666666667"
                iyz="0"
                izz="0.011666666666667" />

        </inertial>
         <collision name="collision">
            <origin
                xyz="0 0.0 0.28"
                rpy="0 0 0" />

            <geometry>
                <mesh filename ="file://$(find holo)/meshes/base.dae" scale="0.01  0.01 0
```

```
            </geometry>
        </collision>
        <visual>
            <origin
                xyz="0 0 0.28"
                rpy="0 0 0" />
            <geometry>
                <mesh
                filename ="file://$(find holo)/meshes/base.dae" scale="0.01  0.01 0.01"/>
            </geometry>
        </visual>
    </link>
.
.
<!-- lot more stuff needs to go here -->
.
.
</robot>
```

This is how we'll need to start the robot definition.

```
<?xml version="1.0" ?>
```

This XML declaration just describes some of the most general properties of the document.

```
<robot name="hb_bot">
<!-- Start of Robot Definition
Inside which we shall have all of the remaining definitions, which will end with the last
</robot>
```

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

This element is the definition of the chassis of the robot. A single rigid body.

- **inertial**: To define the inertial properties of the rigid body (chassis)
  - origin
  - mass
  - inertia matrix: this was generated by solidworks and may or may not accurate, but do note it is a diagonal matrix as expected.
- **visual**: To describe only how it LOOKS.
  - origin
  - geometry: Here we enter an STL file (mesh file) for the purpose of visualisation.
- **collision**: To describe its interaction with the world.
  - origin
  - geometry: Here we simplify the shape to simple cylinders to make processing collisions easier for gazebo (while still maintaining a reasonable match with the real shape).

Similarly here is the definition of one more link: the front wheel of a three-omni-wheeled bot:

```
    <!-- right_wheel -->

    <link name ="right_wheel">

        <inertial>
            <origin
                xyz="0 -0.05 0.0"
                rpy="1.57 0 0" />
            <mass
                value="0.060" />
            <inertia
                ixx="1.825e-4"
                ixy="0"
                ixz="0.00000000"
                iyy="1.825e-4"
                iyz="0"
```

```
                izz="1.825e-4" />
        </inertial>
         <collision name="R_collision">
         <origin
                xyz="0 -0.05 0.0"
                rpy="1.57 0 0" />
          <geometry>
            <cylinder length="0.13" radius="0.14"/>
          </geometry>
        </collision>

        <visual>
            <origin
            xyz="0.0 0.0 0.0"
            rpy="0.0 0.0 0.0" />
            <geometry>
                <mesh filename ="file://$(find holo)/meshes/wheel.stl" scale="5 5 5"/>
            </geometry>

        </visual>
    </link>
```

This is almost exactly the same as the chassis link definition. So with no further explanation, Let's add this to the above robot definition.

Now that we have two links in our file, let's add a joint between the two.

```
    <!-- Joint -->
    <joint name ="right_wheel_joint" type="continuous" >
        <origin
                xyz="0.0 0.68 0.18"
                rpy="0 0 0" />
        <parent link="base_link"/>
        <child link="right_wheel"/>
        <axis xyz="0.0 1.0 0.0"/>
         <limit
            effort="5"
            velocity="5" />
    </joint>

    <!-- material properties -->
    <gazebo reference="right_wheel">
        <material>Gazebo/Black</material>
    </gazebo>
```

This element is how we ***define a joint between two links***.

```
 <joint name="right_wheel_joint" type="continuous">
 </joint>
```

the **type = "continuous"** here means to create a simple revolute joint.

The elements within this element are pretty self-explanatory.

The **origin** x,y,z, roll, pitch, yaw, is what defines the position of the joint between the **"parent link"** and the **"child link"**. The **axis** define the axis of the revolute joint being defined. And the **limit** puts some limit on the motion of the joint.

And there you go! we have a robot with one wheel!😄 All we need to do now is add two more wheels to the bot, and of course the gazebo plugin!

...

Or is something missing? yes ofcourse, where are the STL files? We'll need the STL files for the chassis and the wheel.

Here is the download link of all required files for Task 1B:

Task_1B.zip

Following is how the structure of the `hb_task_1b` package should look like after adding these files to your package (repo).

```
Package (hb_task_1b)
    - hb_task_1b
        - __init__.py
    - launch
        - gazebo.launch.py
    - urdf
        - hb_bot.urdf.xacro
        - materials.xacro
    - world
        - gazebo.world
    - scripts
        - controller.py
        - service_node.py
    - meshes
        - base.dae
        - wheel.stl
        - 17eyantra_logo_large e.png
```

You have seen that we haven't mentioned about `my_robot_interfaces` folder right!!!.Yup that's the package which has custom service message. You have to cut that package and paste it outside the `hb_task_1b` package

Following is how the structure of the workspace `hb_task1b_ws` will be :

```
- hb_task1b_ws
    - src
        - my_robot_interfaces
        - hb_task_1b
```

Task

Problem Statement: Add two more wheels **(left wheel and rear wheel)** to the xacro file.

One useful hint: the main element that changes from front wheel to left or right wheel is it's origin. The x and y will change to +/-0.58 and +/-0.36 for the left/right wheels. We'll let you figure out the roll, pitch, and yaw by yourself.

Finally to finish the URDF File:

Let's add the **Gazebo Plugin**

```
<!-- ros_control plugin -->

  <gazebo>
    <plugin name="object_controller" filename="libgazebo_ros_planar_move.so">
      <commandTopic>cmd_vel</commandTopic>
      <odometryTopic>odom</odometryTopic>
      <odometryFrame>odom</odometryFrame>
      <odometryRate>20.0</odometryRate>
      <robotBaseFrame>chasis</robotBaseFrame>
    </plugin>
  </gazebo>
</robot>
```

This plugin allows us control the robot movement through the /cmd_vel topic defined in <commandTopic> and also get the robot pose through the /odom topic defined in <odometryTopic>.

And voila! That's actually it! Just launch the launch file, run the teleop node as described at the start and have fun driving around the Hologlyph Bot!

BONUS: Observe the Holonomic Mode in teleop package. and figure out what teleop is actually doing by using `echo` and `info`. For Ex: `ros2 topic info /cmd_vel`.
