LR_3_Prep

Тема: **Создание моделей роботов и объектов окружения**

Создание первого urdf файла

Чтобы создать первую модель, нам понадобится пакет в системе catkin. В дальнейшем мы будем пользоваться некоторыми вспомогательыми иструментами VS Code, поэтому настоятельно рекомендуется его установка.

1. Создаем новый пакет в рабочем пространстве catkin

```
cd catkin_ws/src
```

```
catkin_create_pkg my_robot_model rospy std_msgs roscpp
```

```
catkin build
```

2. Внутри папки нашего пакета создаем подпапки scripts, launch и urdf.

   ```
   cd my_robot_model
   ```

   ```
   mkdir scripts launch urdf
   ```

3. Создадим свою первую модель в формате urdf, для этого переходим в папку urdf и созадем xml файл с расширением .urdf

   ```
   cd urdf
   ```

   ```
   touch my_first_model.urdf
   ```

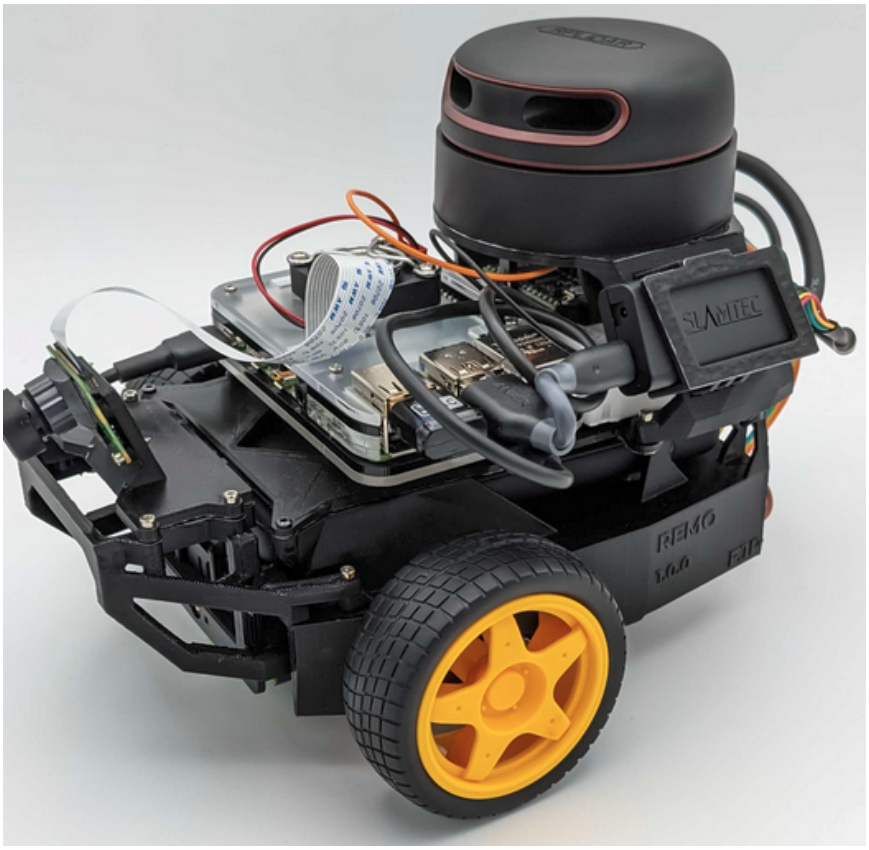
   ```
   code my_first_model.urdf
   ```

Убедитесь, что у вас в Visual Studio Code установлено расширение ROS от **Microsoft. Также вам необходимо установить расширение ROS Snippets(Liews Wuttipat)** .
Запустить визуализатор можно при помощи CTRL+Shift+P и там выбрать из списка URDF Viewer.
Также рекомендуется запускать VS Code из рабочего окружения catkin, например:

```
cd catkin_ws
code .
```

После чего убедитесь, что VS Code видит ваш проект как ROS (внизу окна показывет версию ROS и прочую информацию), это может быть необходимо для корректной работы расширений в VS Code.

4. Приступим непосредственно к созданию модели робота. В качестве тренировки мы сделаем модель мобильного робота дифференциальным приводом, показаный на рисунке ниже.



В первую очередь мы должы определить, какие элеметы нашего робота имеют относительную подвижность. Как видим из изображения, два колеса являются приводыми и имеют вращательную степень подвижности вокруг одной оси. Третье колесо (его не видно на изображении) имеет пассивно-сферический вид, т.е. может свободно вращаться сразу вокруг двух осей. Также необходимо определить, какой элемент является базовым, базовый элемент - тот элемент, к которому все остальные элементы присоединяются при помощи шарниров, образуя при этом древовидную структуру.

- Создадим первый элемент - базовый, это будет корпус платформы для нашего примера.

```
<?xml version='1.0'?>
<robot name="robot" xmlns:xacro="http://www.ros.org/wiki/xacro">
  <origin xyz="0.0 0.0 0.0" rpy="0.0 0.0 0.0"/>

  <link name='base_link'>

    <visual name='base_visual'>
      <origin xyz="0.0 0.0 0.0" rpy="0.0 0.0 0.0"/>
      <geometry>
        <box size="0.2 0.6 0.6"/>
      </geometry>
      <!-- <material name="">
        <color rgb="1.0 0.0 0.0">
      </material> -->
    </visual>

    <!-- <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> -->

    <!-- <collision>
     <origin xyz="0 0 0" rpy="0 0 0"/>
     <geometry>
       <cylinder radius="1" length="0.5"/>
     </geometry>
   </collision> -->
  </link>
  <link name='wheel_left'>
    <visual name='wheel_left_visual'>
      <origin xyz="0.2 0 0" rpy=" 0 0 0"/>
      <geometry>
        <cylinder length="0.7" radius="0.1"/>
      </geometry>
      <material name="gray"/>
    </visual>
  </link>

  <joint type="fixed" name="wheel_joint">
    <origin xyz="0.15 0.0 0" rpy="0 0 0"/>
    <child link="wheel_left"/>
    <parent link="base_link"/>
  </joint>

</robot>
```

У шарнира, который соединяет колесо с базой необходимо установить type="continuous” поскольку мы не хотим задавать какие либо огрничения на его вращение.

- Необходимо скопировать и переименовать звено wheel_left в звено wheel_right
- Также необходимо настроить положение и ориентацию колес.

5. Давайте теперь откроем модель робота в rviz, для этого создадим launch файл и пропишем внутри:
   a. Загрузку модели робота в robot_description сервера парметров ROS
   b. Запуск RViz

```
<launch>
<arg name="gui" default="False" />
<param name="robot_description" textfile="$(find my_robot_model)/urdf/my_first_model.urdf" />
<param name="use_gui" value="$(arg gui)"/>

<node name="rviz" pkg="rviz" type="rviz" />

</launch>
```

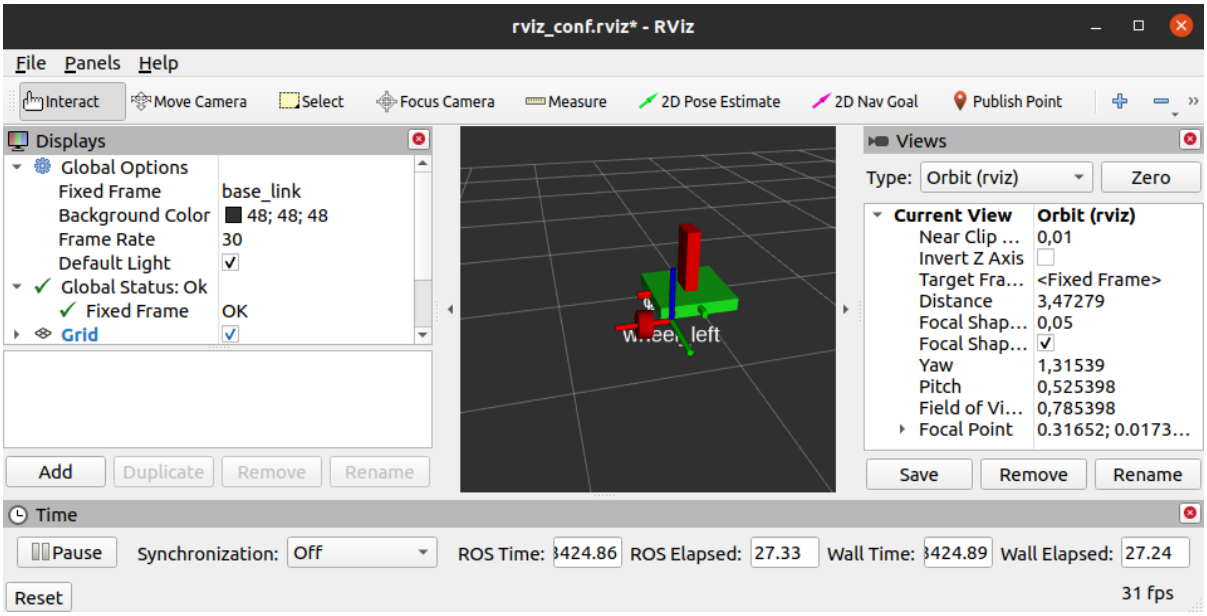
И запустии его

`roslaunch my_robot_model robot.launch`

6. Чтобы можно было управлять шарнирами, нам необходимо подключить два пакета: **joint_state_publisher** и **robot_state_publisher**, сделаем это в launch файле добавив следующие строки.

```
<node name="joint_state_publisher" pkg="joint_state_publisher" type="joint_state_publisher" />
<node name="robot_state_publisher" pkg="robot_state_publisher" type="state_publisher" />
```

7. После подключения пакетов мы сможем визуализировать все системы координат в RViz, для этого нажмем **Add** a new display и выберем **TF.**



8. Запустим графический итерфейс для joint_state_publisher чтобы попробовать повращать шарниры колеса, но сначала нужно установить данный пакет под названием joint_state_publisher_gui воспользовавшись командой:

   `sudo apt install ros-noetic-joint-state-publisher-gui`

   Подключение Это можно сделать и через терминал, но мы сделаем это через launch файл, указав дополнительную строку.

   ```
   <node name="joint_state_publisher_gui" pkg="joint_state_publisher_gui" type="joint_state_publisher_gui" />
   ```

   Также необходимо закомментировать joint_state_publisher так как одновременно два пакета не должны публиковать в одни и те же топики.

9. Проверим корректность расположения осей вращения и звеньев мобильного робота и внесем необходимые изменения, чтобы модель соответствоала роботу, изображенному на картинке выше.

Создание модели в xacro

Чтобы наша модель была более гибкой и универсальной мы можем воспользоваться возможностями xacro.

1. Создадим новый файл под названием diff_robot_model.urdf.xacro, скоипруем туда описание нашей модели и укажем дополнительный парметр в теге robot

   ```
   <robot xmlns:xacro="http://www.ros.org/wiki/xacro"  name="my_robot">
   ```

2. Также для работы xacro нам необходимо изменить launch файл, нас интересует строка загрузки robot_decription в сервервер параметров, его необходимо исправить как показано ниже

   ```
   <param name="robot_description" command="xacro '$(find my_robot_model)/urdf/diff_robot_model.urdf.xacro'" />
   ```

3. Создадим также свойство в нашем файле

   ```
   <xacro:property name="side" value="right" />
   ```

   Теперь мы можем его подставить в разметке, чтобы обеспечить быстрое редактирование (как у звена, так и у шарнира)

   Например,

   ```
   <link name='wheel_${side}'>
       <visual name='wheel_${side}_visual'>
   ```

   > Задание на 5 минут:
   > Создайте свойство с наванием **wheel_diameter** и в описании модели замените диаметры цилиндров на значение свойства. Проверьте в RViz результат.

4. Создадим новый файл с названием materials.xacro и зададим внутри наши цвета, которые мы хотим использовать в модели.

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

  <material name="black">
    <color rgba="0.0 0.0 0.0 1.0"/>
  </material>

  <material name="blue">
```

```
    <color rgba="0.0 0.0 0.8 1.0"/>
  </material>

  <material name="green">
    <color rgba="${30/255} ${255/255} ${30/255} 1.0"/>
  </material>
  <material name="red">
    <color rgba="0.8 0.0 0.0 1.0"/>
  </material>
</robot>
```

Теперь в основном файле .xacro мы можем подключить этот файл следующим образом

```
<xacro:include filename="$(find my_robot_model)/urdf/materials.xacro" />
```

И использовать данные цвета для нашей модели робота.

```
<material name="red"/> <!-- внутри visual -->
```

5. Использование **xacro:macro** позволяет избегать повторения фрахментов разметки, например, колесо повторяется для левой и правой стороны, при этом в фрагменте меняется только имя элемента и положение. Создадим отдельный файл, в котором будет содержаться описание колеса и будем его копии испортировать с автоматической подстройкой под нужную сторону.
6. Создаем новый файл wheel.xacro

```
<xacro:macro name="wheel" params="side x y z">
    <link name='wheel_${side}'>
      <visual name='wheel_${side}_visual'>
        <origin xyz="0.2 0 0" rpy=" 1.5 0.0 1.5"/>
        <geometry>
          <cylinder length="0.1" radius="0.1"/>
        </geometry>
        <material name="red"/>
      </visual>
    </link>
    <joint name="wheel_{side}_joint" type="continuous">
      <origin xyz="${x} ${y} ${z}" rpy="0 0 0"/>
      <child link="wheel_${side}"/>
      <parent link="base_link"/>
    </joint>
  </xacro:macro>
```

7. И в файле модели укажем вызов macro с необходимыми нам парметрами (парметры очень похожи на свойства, с единственным отличием - их значения нам надо передавать при вызове)

```
<xacro:wheel side="left" x="0.25" y="0.4" z="0.0" />
```

Использование 3д моделей сложной формы

Чтобы наши модели были более реалистичными, мы можем подключить любой stl файл внутри блока geometry

```
<mesh filename="package://my_robot_model/models/base_frame.stl" scale="1.0 1.0 1.0">
```

Первый запуск симуляции

Чтобы смоделирвоать движение мобильного робота нам необходимо чтобы у каждого звена были блоки **collision** и **inertial**. Если они отсутствуют, то их необходимо добавить. Также массы объектов должны быть больше 0.

Для зупуска Gazebo пропишем следующий код в launch файле

```
<include file="$(find gazebo_ros)/launch/empty_world.launch">
    <arg name="world_name" value="$(find turtlebot3_gazebo)/worlds/empty.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="debug" value="false"/>
  </include>


  <node name="my_robot_spawn" pkg="gazebo_ros" type="spawn_model" output="screen"
   args="-urdf -param robot_description -model my_robot_model" />
```

Цвета объектов в urdf не поддерживаются в Gazebo, поэтому нам необходимо указать специальные теги для симулятора.

Задание

1. Добавить двухзвенный манипулятор к подвижной платформе.
2. Добавить ограничения к шарнирам манипулятора, чтобы он не мог сталкиваться со вторым звеном

Ссылки на ROS Wiki:

http://wiki.ros.org/urdf/Tutorials/Building a Visual Robot Model with URDF from Scratch