ROS-I Basic Training "Mobility"
ROS MoveIt! Tutorial

Instructor: Nicolas Limpert

August 9, 2017

FH AACHEN
UNIVERSITY OF APPLIED SCIENCES

Contents

1 Introduction

This tutorial explains the basics of how to describe industrial robots in ROS and how to use the created kinematic models. You will learn about the Unified Robot Description Format (URDF). The tutorial will inform about how to create a robot model, how to visualize your robot and begin with easy planning for your model.

- Lines beginning with $ are terminal commands
- Lines beginning with # indicate the syntax of the commands
- The symbol ↪ represents a line break.

2 Create an URDF file

A URDF file is generated in the XML format. URDFs contain information about the robot links and joints. For detailed information about this format check the following link: http://wiki.ros.org/urdf/XML Figure 1 shows a simple kinematics that can be built using URDF.

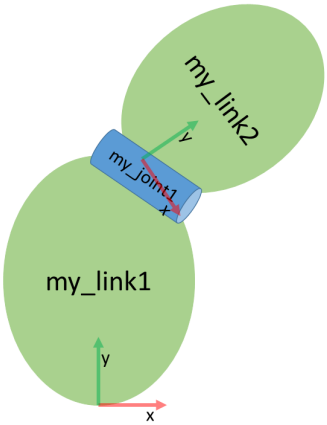

Figure 1: Serial kinematics sketch

It is a simple kinematics existing of two links connected via a rotating joint. Check the example URDF file for this kinematics in appendix 1 attached at the end of this tutorial.

The file describes the link and joint elements. "my_link1" is described within the tags <visual>, which is just for the optics and <collision>, which is responsible for the physics of the model. The visual part is described as a red cylinder with a radius of 0.4 m and a length of 0.6 m. This part can as well be described by using CAD data files of a real robot in the collada (.dae) or stereolithography (.stl) format including texture information. The collision part will be used in simulations like Gazebo describing

the hit box of the link when interfering with other objects. It is also possible for collision to use CAD data files instead of creating simple shapes.

2.1 Write the YouBot URDF file

This knowledge is sufficient to generate the URDF file for the YouBot. The following picture (s. fig. **??**) displays a screenshot of the YouBot's arm including the frames as shown by tf within RViz. Within this tutorial we want to add the links and joints needed to create the YouBot arm. This will also include integration of visuals and collisions as defined as .dae-files.

1. Open the file called "`arm.urdf.xacro`" in youbot_description/urdf/youbot_arm.

2. Make sure that the file's header looks like the following:

```
<?xml version="1.0"?>
<robot  xmlns:sensor="http://playerstage.sourceforge.net/
  ↪ gazebo/xmlschema/#sensor"
       xmlns:controller="http://playerstage.sourceforge.net/
         ↪ gazebo/xmlschema/#controller"
       xmlns:interface="http://playerstage.sourceforge.net/
         ↪ gazebo/xmlschema/#interface"
       xmlns:xacro="http://ros.org/wiki/xacro">

       <xacro:include filename="$(find youbot_description)/
         ↪ urdf/youbot_arm/arm.gazebo.xacro" />
       <xacro:include filename="$(find youbot_description)/
         ↪ urdf/youbot_arm/arm.transmission.xacro" />
       <xacro:include filename="$(find youbot_description)/
         ↪ urdf/youbot_arm/limits.urdf.xacro" />

       <xacro:macro name="youbot_arm" params="parent name *
         ↪ origin">
```

3. Insert the arm's joints by making use of the template as shown in listing 1. Table 1 displays the joints to be inserted.

4. Insert the arm's links by making use of the template as shown in listing 2. Table 2 displays the links to be inserted. Notice that "armn" stands for "arm" with the number n as its postfix.

5. Make sure to uncomment the lines 19 and 22 in the file youbot.urdf.xacro located in `youbot_description/robots`.
   From

```
<!-- youbot arm -->
<!--xacro:include filename="$(find youbot_description)/urdf/
  ↪ youbot_arm/arm.urdf.xacro"/-->

<!-- youbot gripper -->
<!--xacro:include filename="$(find youbot_description)/urdf/
  ↪ youbot_gripper/gripper.urdf.xacro" /-->
```

   To

```
<!-- youbot arm -->
<xacro:include filename="$(find youbot_description)/urdf/
  ↪ youbot_arm/arm.urdf.xacro"/>

<!-- youbot gripper -->
<xacro:include filename="$(find youbot_description)/urdf/
  ↪ youbot_gripper/gripper.urdf.xacro" />
```

When you're done you can proceed with creating a MoveIt! config for your particular URDF in the next section.

3 Your URDF in MoveIt!

The generated URDF file can now be used from several ROS tools, e.g. as robot model in the Gazebo Simulation environment, as visualization element in rviz or as a kinematic model for MoveIt!. MoveIt! is the most widely used Open Source software for motion planning, manipulation, 3D perception, kinematic, control and navigation.
It provides an easy-to-use platform for developing advanced robotics applications, evaluating new robot designs and building integrated robotics products.

3.1 Introduction to the MoveIt! Setup Assistant

Figure 2: MoveIt! Setup Assistant start screen

The MoveIt! Setup Assistant is a powerful graphical tool to configure any robot you want to use with MoveIt!.
The main task of the Setup Assistant consists in generating the Semantic Robot Description Format (SRDF), which is used in the ROS node move_group. Beside the SRDF there are created many configuration files e.g. for joint limits, kinematics and motion planning.
Check http://wiki.ros.org/srdf for further information.
The only information the tool needs is the Unified Robot Description Format (URDF) of the robot, e.g. the file that you have created recently.

3.2 Configure your robot with the MoveIt! Setup Assistant

Before you start the MoveIt! Setup Assistant make sure that you have installed the youbot-manipulation package. Among others, this serves as a kinematics solver specially used for the Youbot:

```
$ git clone https://github.com/svenschneider/youbot-manipulation.
  ↪ git
```

To start the Setup Assistant you need the following command:

```
$ roslaunch moveit_setup_assistant setup_assistant.launch
```

A graphical user interface comparable to figure 2 should appear.

- Select the Create New MoveIt! Configuration Package button and browse for the file called youbot.urdf.xacro in `youbot_description/robots/youbot.urdf.xacro`. This file includes the URDF file you edited in task 2.1, your window should look like shown in figure 3:

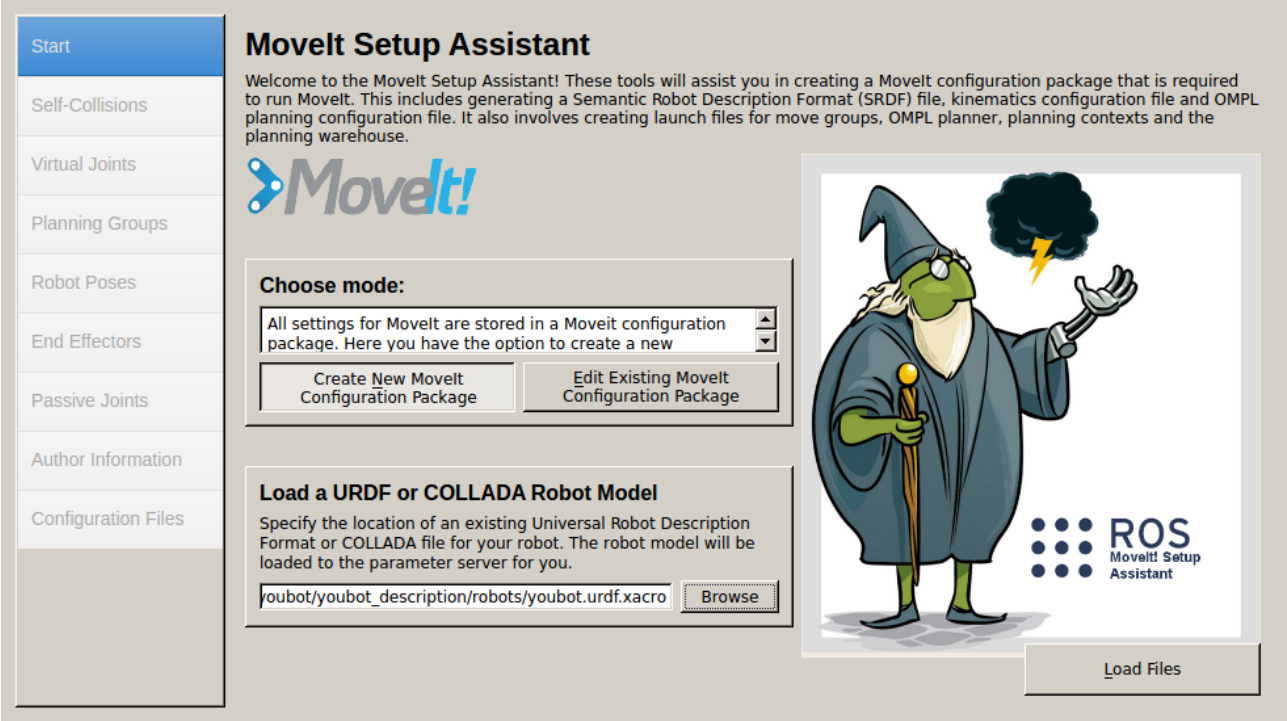

Figure 3: Choose mode and select URDF

- Click on the Load Files button and after a few seconds the Setup Assistant will present you a model of the robot on the right side of the window.
- Afterwards click on the Self Collision pane selector to generate a matrix for pair of links, which are not necessary to be checked for collision every time at planning process. Because they are either always in collision or never in collision.
  - Change the Sampling Density to 60000
  - Select the Regenerate Default Collision Matrix button
  - After a few seconds the MoveIt! Setup Assistant will present you the results of the computation:

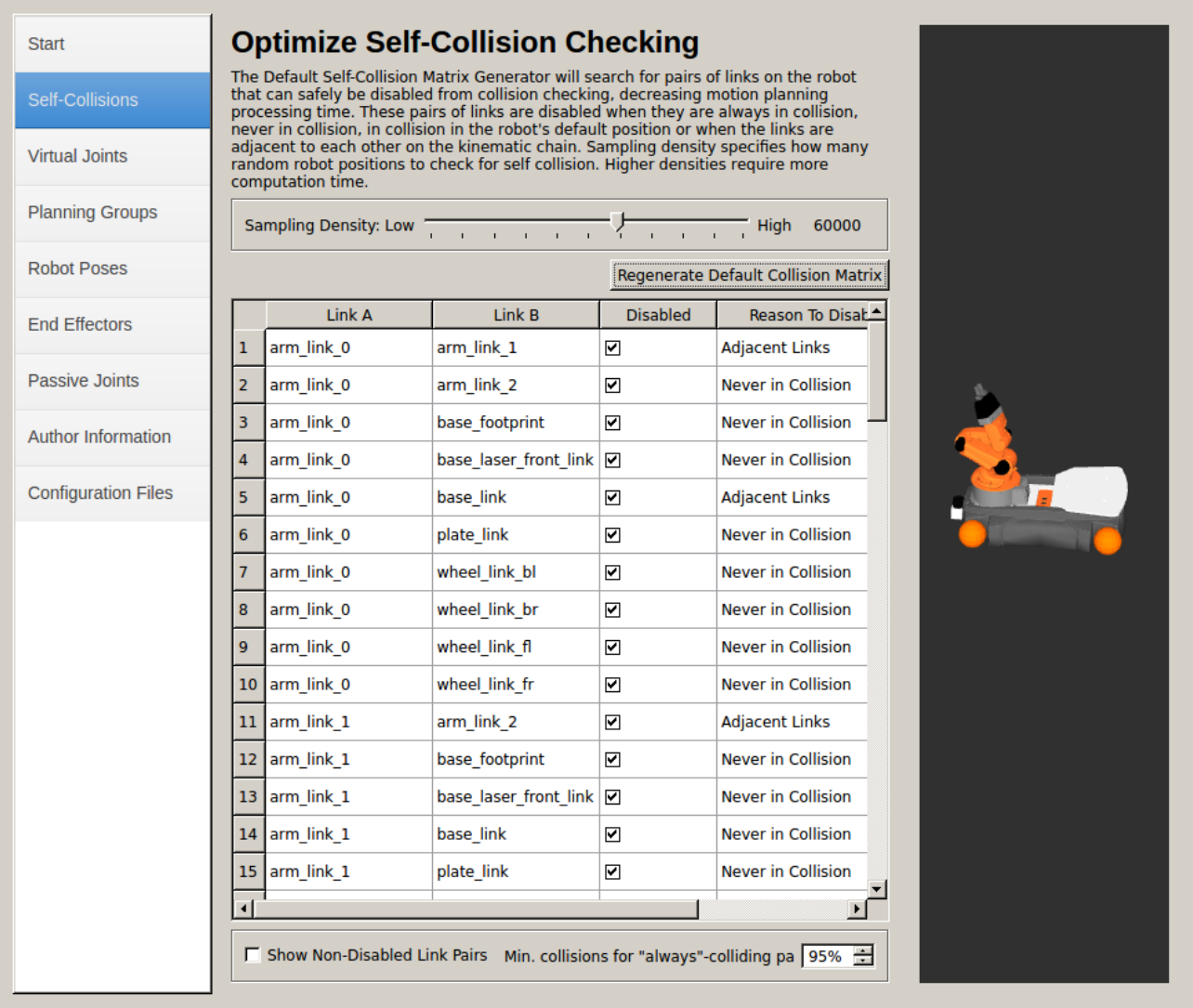

Figure 4: Results for the Default Collision Matrix

- Now add a virtual joint to attach your robot to the world.
  - Select the Virtual Joint pane selector
  - Click on the Add Virtual Joint button

- Set the joint name to "odom"
- Set the child link as "base_footprint" and the parent link as "odom"
- Set the joint type as "fixed"
- Now save the data and your screen should look like this:

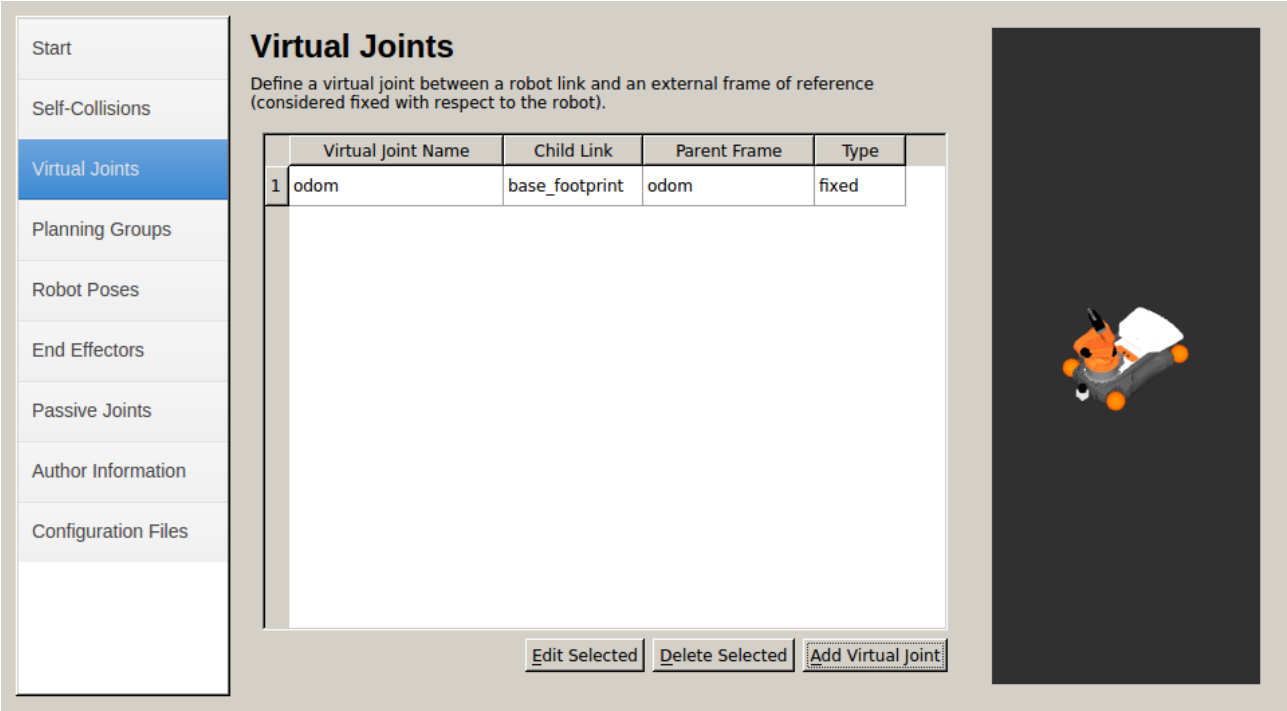

Figure 5: Virtual Joints

- In the next step you will define your planning group for which MoveIt! later on tries to compute a path.
  - Click on the Planning Groups pane selector
  - Click on Add Group
  - Set the Group Name as "arm_1"

- Choose youbot_arm_kinematics_moveit::KinematicsPlugin as Kinematic Solver
- Set Kin. Search Resolution to 0.1
- Set Kin. Solver Attempts to 1
- Click on the "Add Joints" Button
- Select the joints arm_joint_{1-5} as shown in the following screenshot and move them to "Selected Joints" by clicking the > Button:

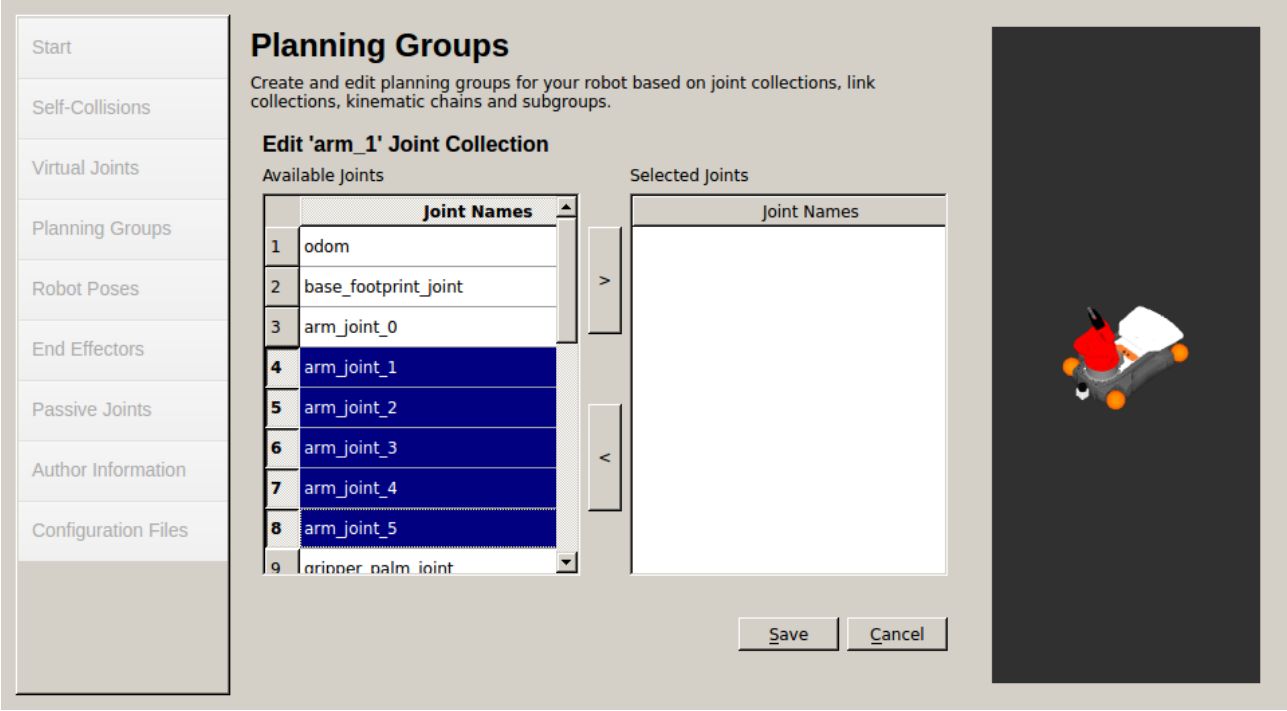

Figure 6: Joints 1-5 to be added

- Click "Save"
- Add another group called arm_1_gripper the same way but without a Kinematic Solver plugin and the joints "gripper_finger_joint_l" and "gripper_finger_joint_r". Your screen should afterwards look like the following:

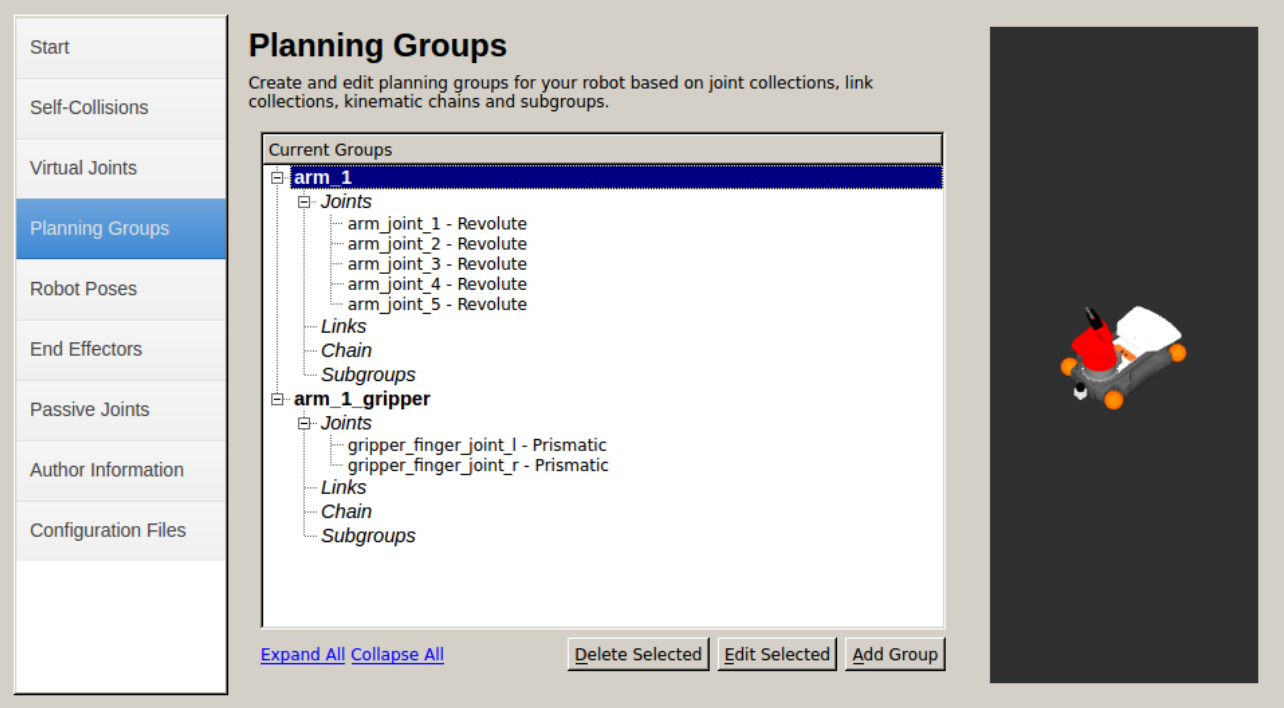

Figure 7: Both kinematic chains added

- The Setup Assistant includes the option to add poses. It is helpful to define poses that will be used often in later developing process e.g. a home pose for the robot.
  - Select Robot Poses
  - Click the Add Pose button
  - Choose a name for the pose and move the joints to a position that you like
  - Save the Pose
- Select the tab "End Effectors"
- Click "Add End Effector" and setup the fields as shown in figure 8.

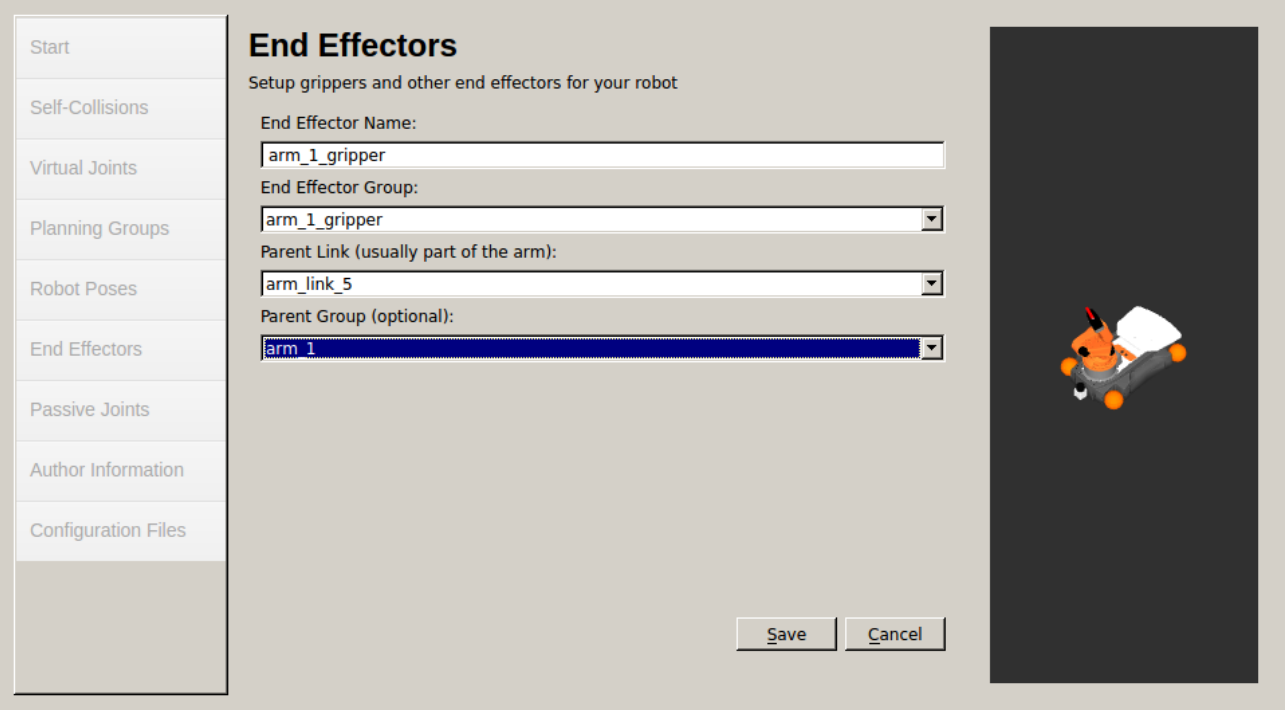

Figure 8: End Effector setup

- Save the End Effector setup and switch to "Author Information"
- To generate the package you need to provide Author information so enter the required information
- Click on "Configuration Files", click "Browse" and select a path inside your workspace as shown in figure 9.

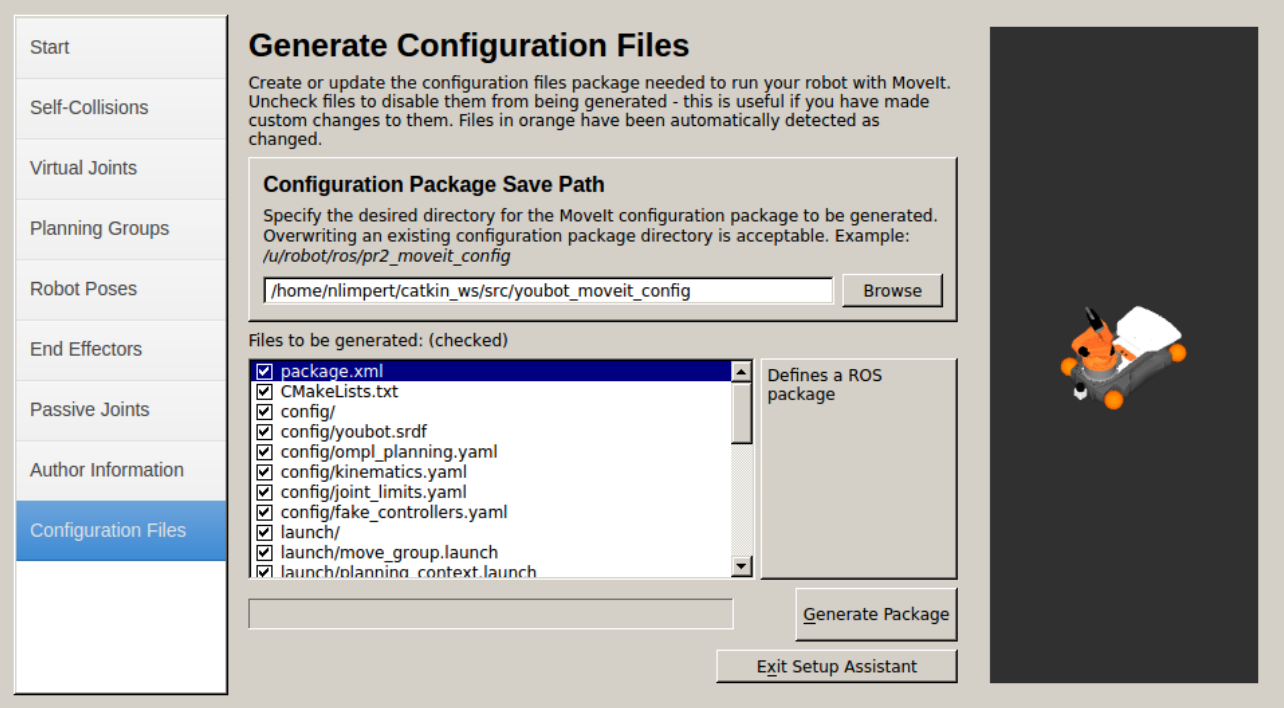

Figure 9: Save configuration files

- Click "Generate Package"
- If the process completes successfully you can try out your newly created MoveIt! configuration as described in the section 4.1

4 MoveIt! Usage

This section makes actual use of your newly created MoveIt! Package.

4.1 demo.launch

You can try out your newly created MoveIt! package by launching the following:

```
$ roslaunch youbot_moveit_config demo.launch
```

This is useful if you want to observe wether joint and link setups actually work. This can also be used to test whether the currently setup joint limits work as expected.

4.2 move_group.launch on the simulated robot

Setting up the MoveIt! config to be used on the simulated robot requires some small further steps. MoveIt! has to be aware of the particular controllers to be used. Therefor we have to do the following:

- Create a file called "controllers.yaml" in youbot_moveit_config/config with the following contents:

```
controller_list:
  - name: arm_1/arm_controller
    action_ns: follow_joint_trajectory
    type: FollowJointTrajectory
    default: true
    joints:
      - arm_joint_1
      - arm_joint_2
      - arm_joint_3
      - arm_joint_4
      - arm_joint_5
```

- Afterwards modify the file youbot_moveit_config/launch/youbot_moveit_controller_manager.launch.xml to contain the following:

```
<launch>

  <!-- Set the param that trajectory_execution_manager needs
    ↪ to find the controller plugin -->
```

```
  <param name="moveit_controller_manager" value="
    ↪ moveit_simple_controller_manager/
    ↪ MoveItSimpleControllerManager"/>

  <!-- The rest of the params are specific to this plugin -->
  <rosparam file="$(find youbot_moveit_config)/config/
    ↪ controllers.yaml"/>

</launch>
```

That's it - you can now launch your simulation, including MoveIt! features and afterwards a predefined Rviz configuration in 3 different terminals:

```
$ roslaunch youbot_gazebo_robot youbot.launch
```

```
$ roslaunch youbot_moveit_config move_group.launch
```

```
$ roslaunch youbot_moveit_config moveit_rviz.launch
```

You should see RViz which should look like shown in figure 10

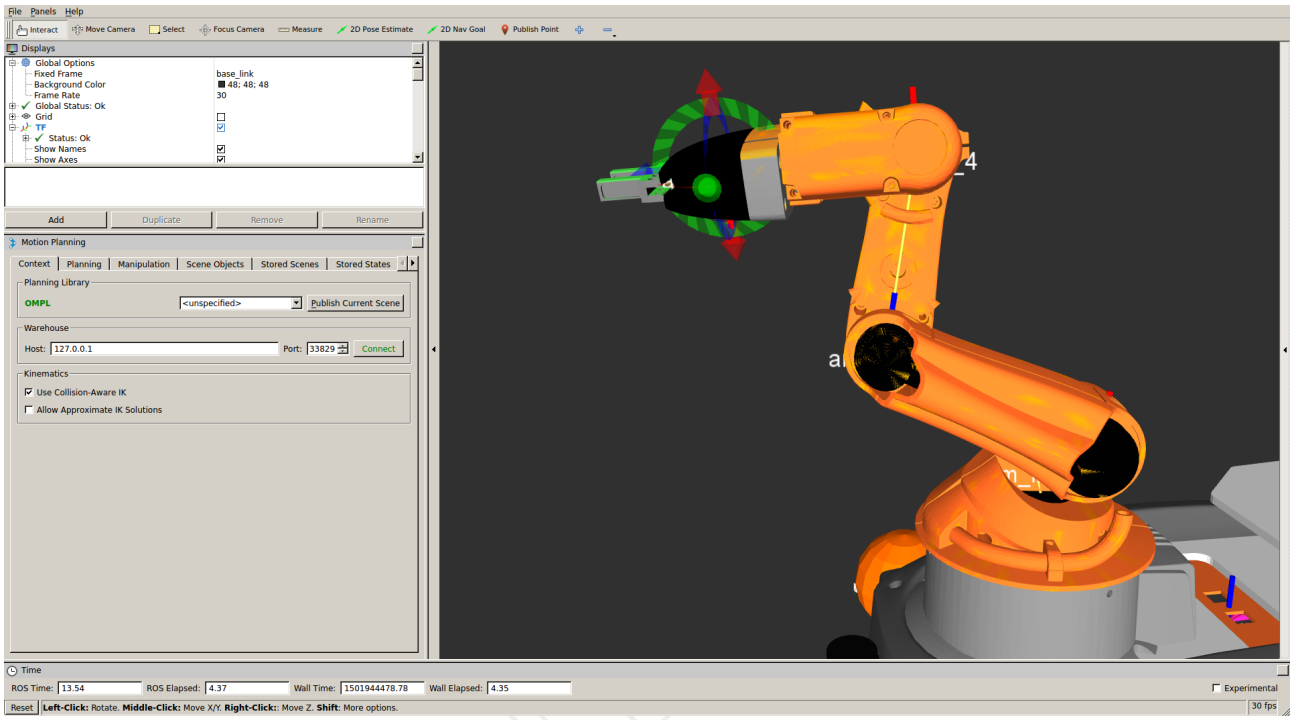

Figure 10: MoveIt! default RViz configuration

Perform the following steps to verify functionality:

1. Select the "Planning" tab on the left
2. Move the marker displayed in figure 11 to a desired position or simply click "Update" in "Select Goal State:" when "<random valid>" is selected.
3. Click "Plan and Execute" on the left.
4. You should observe RViz displaying the motion of the arm. The simulated YouBot should show a moving arm too.

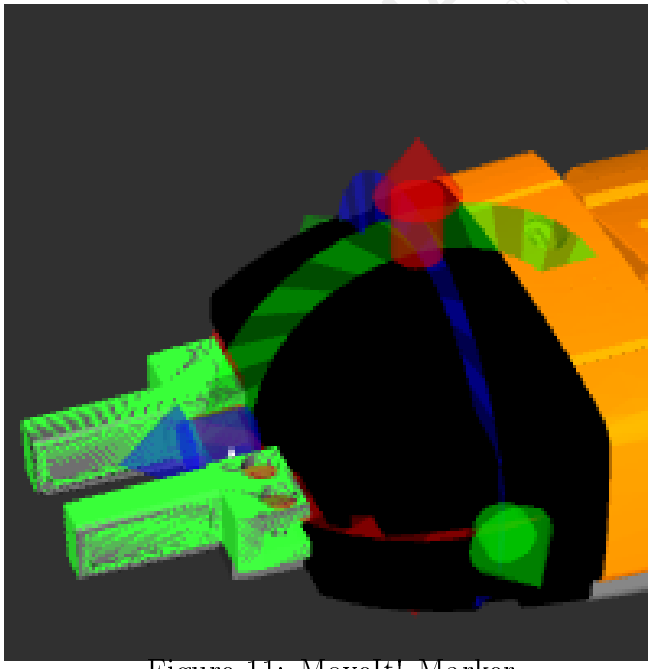

Figure 11: MoveIt! Marker

5 Appendix

Table 1: YouBot arm joints to be inserted

n	origin_xyz	origin_rpy	axis_xyz	effort	velocity
1	0.024 0 0.096	0 0 ${170 * M_PI / 180}	0 0 -1	${9.5}	${M_PI / 2.0}
2	0.033 0 0.019	0 ${-65 * M_PI / 180} 0	0 1 0	${9.5}	${M_PI / 2.0}
3	0.000 0.000 0.155	0 ${146 * M_PI / 180} 0	0 1 0	${6.0}	${M_PI / 2.0}
4	0.000 0.000 0.135	0 ${-102.5 * M_PI / 180} 0	0 1 0	${2.0}	${M_PI / 2.0}
5	-0.002 0 0.130	0 0 ${167.5 * M_PI / 180}	0 0 -1	${1.0}	${M_PI / 2.0}

Table 2: YouBot arm links to be inserted

n	vis_origin_xyz	vis_origin_rpy	col_origin_xyz	col_origin_rpy	material
1	0 0 0	0 0 0	0 0 0	0 0 0	youBot/Orange
2	0 -0.032 0.078	0 0 0	0 -0.032 0.078	0 0 0	
3	0.000 0.028 0.079	0 0 0	0.000 0.028 0.079	0 0 0	youBot/Orange
4	0 -0.010 0.029	0 0 0	0 -0.010 0.029	0 0 0	youBot/Orange
5	0.003 0 -0.034	0 0 0	0.003 0 -0.034	0 0 0	youBot/DarkGrey

Listing 1: URDF joint template

```
<joint name="${name}_joint_n" type="revolute">
        <origin xyz="origin_x origin_y origin_z" rpy="origin_r
          ↪ origin_p origin_y"/>
        <parent link="${name}_link_{n-1}"/>
        <child link="${name}_link_n"/>
        <axis xyz="axis_x axis_y axis_z"/>
        <dynamics damping="1" friction="1" />
        <limit effort="effort" velocity="velocity" lower="0"
          ↪ upper="${joint_n_limit}"/>
</joint>
```

Listing 2: URDF link template

```
<link name="${name}_link_n">
        <inertial>
                <mass value="${link_n_mass}"/>
                <xacro:link_n_inertia_tensor />
        </inertial>

        <visual>
                <origin xyz="vis_origin_x vis_origin_y
                   ↪ vis_origin_z" rpy="vis_origin_r
                   ↪ vis_origin_p vis_origin_y"/>
                <geometry>
                        <mesh filename="package://
                           ↪ youbot_description/meshes/
                           ↪ youbot_arm/armn.dae" />
                </geometry>
                <material name="youBot/Orange" />
        </visual>

        <collision>
                <origin xyz="col_origin_x col_origin_y
                   ↪ col_origin_z" rpy="col_origin_r
                   ↪ col_origin_p col_origin_y"/>
                <geometry>
                        <mesh filename="package://
                           ↪ youbot_description/meshes/
                           ↪ youbot_arm/armn_convex.dae" />
                </geometry>
        </collision>
</link>
```

Listing 3: Example URDF file

```
<?xml version="1.0" encoding="UTF-8"?>
<robot name="my_robot">
  <link name="my_link1">
    <visual>
      <geometry>
        <cylinder length="0.6" radius="0.4"/>
      </geometry>
      <material name="red"/>
        <color rgba="1 0 0 1"/>
    </visual>
    <collision>
      <geometry>
        <cylinder length="0.6" radius="0.4"/>
      </geometry>
      <material name="yellow">
        <color rgba="0 1 1 1"/>
      </material>
    </collision>
  </link>
  <link name="my_link2">
    <visual>
      <geometry>
        <cylinder length="0.5" radius="0.3"/>
      </geometry>
      <material name="blue">
        <color rgba="0 0 0.8 1"/>
    </visual>
    <collision>
      <geometry>
```

```
        <cylinder length="0.4" radius="0.3"/>
      </geometry>
      <material name="yellow">
      </material>
    </collision>
  </link>

  <joint name="my_joint1" type="revolute">
    <parent link="my_link1"/>
    <child link="my_link2"/>
    <origin xyz="0.3 0.55 0" rpy="0 0 -1.57" />
    <axis xyz="0 0 -1"/>
    <limit effort="100" lower="-3.1416" upper="3.1416" velocity="
      ↪ 6.97"/>
  </joint>
</robot>
```