MANIPULATOR DESIGN AND CONTROL

http://wiki.ros.org/urdf/Tutorials/Adding%20Physical%20and%20Collision%20Properties%20to%20a%20URDF%20Model

Add extensions in Visual Studio

ros
ros snippet
xml
xml tools
urdf
xml complete
icons

Execute in Terminal #1

```
sudo apt-get install ros-humble-teleop-twist-keyboard
sudo apt-get install ros-humble-joint-state-publisher*
sudo apt-get install ros-humble-joint-trajectory-controller
sudo apt-get install ros-humble-controller-manager
sudo apt install ros-humble-gazebo-*
sudo apt install ros-humble-gazebo-msgs
sudo apt install ros-humble-gazebo-ros
sudo apt install ros-humble-gazebo-ros2-control-demos
sudo apt install ros-humble-ros2-control
sudo apt install ros-humble-ros2-controllers
sudo apt install ros-humble-ros2controlcli
sudo apt install ros-humble-xacro
sudo apt install ros-humble-gazebo-dev
sudo apt install ros-humble-gazebo-plugins
```

```
cd ros2_ws/src/urdf_tutorial/urdf
touch manipulator.urdf
<?xml version="1.0"?>
<robot name="arm">
<!-- https://www.rapidtables.com/web/color/RGB_Color.html -->
    <link name="world"/>
    <link name="base_link">
        <visual>
            <geometry>
                <cylinder length="0.05" radius="0.2"/>
            </geometry>
            <material name="Orange">
                <color rgba="1 0.5 0 1"/>
            </material>
            <origin xyz="0 0 0.025" rpy="0 0 0" />
        </visual>
```

```
    <collision>
      <geometry>
        <cylinder length="0.05" radius="0.2"/>
      </geometry>
      <origin xyz="0 0 0.025" rpy="0 0 0" />
    </collision>

    <inertial>
      <origin rpy="0 0 0" xyz="0 0 0.025"/>
      <mass value="5.0"/>
      <inertia ixx="0.0135" ixy="0.0" ixz="0.0" iyy="0.0135" iyz="0.0" izz="0.05"/>
    </inertial>
  </link>

  <joint name="world_base_joint" type="fixed">
    <parent link="world"/>
    <child link="base_link"/>
    <dynamics damping="10" friction="1.0"/>
  </joint>

  <link name="arm1_link">
    <visual>
      <geometry>
        <cylinder length="0.5" radius="0.08"/>
      </geometry>
      <material name="Blue">
        <color rgba="0 0 1 1"/>
      </material>
      <origin xyz="0 0 0.25" rpy="0 0 0" />
    </visual>

    <collision>
      <geometry>
        <cylinder length="0.5" radius="0.08"/>
      </geometry>
      <origin xyz="0 0 0.25" rpy="0 0 0" />
    </collision>

    <inertial>
      <origin rpy="0 0 0" xyz="0 0 0.25"/>
      <mass value="5.0"/>
      <inertia ixx="0.107" ixy="0.0" ixz="0.0" iyy="0.107" iyz="0.0" izz="0.0125"/>
    </inertial>

  </link>

  <joint name="base_arm1_joint" type="revolute">
    <axis xyz="0 1 0"/>
    <parent link="base_link"/>
    <child link="arm1_link"/>
    <origin xyz="0.0 0.0 0.05" rpy="0 0 0" />
```

```
    <limit lower="-2.14" upper="2.14" effort="100" velocity="100" />
    <dynamics damping="10" friction="1.0"/>
</joint>



<link name="arm2_link">
    <inertial>
        <origin xyz="0 0 0.25" rpy="0 0 0" />
        <mass value="0.01"/>
        <inertia ixx="0.027" ixy="0.0" ixz="0.0" iyy="0.027" iyz="0.0" izz="0.0025"/>
    </inertial>

    <visual>
        <geometry>
        <cylinder length="0.5" radius="0.05"/>
        </geometry>
        <material name="White">
            <color rgba="1 1 1 1"/>
        </material>
        <origin rpy="0 0 0" xyz="0 0 0.25"/>
    </visual>

    <collision>
        <geometry>
            <cylinder length="0.4" radius="0.05"/>
        </geometry>
        <origin xyz="0 0 0.25" rpy="0 0 0" />
    </collision>
</link>

<joint name="arm1_arm2_joint" type="revolute">
    <parent link="arm1_link"/>
    <child link="arm2_link"/>
    <origin xyz="0.0 0.0 0.5" rpy="0 0 0" />
    <axis xyz="0 1 0"/>
    <limit lower="-2.14" upper="2.14" effort="100" velocity="100" />
    <dynamics damping="10" friction="1.0"/>
</joint>

<link name="arm3_link">
    <inertial>
        <origin xyz="0 0 0.15" rpy="0 0 0" />
        <mass value="0.01"/>
        <inertia ixx="0.027" ixy="0.0" ixz="0.0" iyy="0.027" iyz="0.0" izz="0.0025"/>
    </inertial>
    <visual>
        <geometry>
            <cylinder length="0.3" radius="0.03"/>
        </geometry>
        <material name="Red">
            <color rgba="1 0 0 1"/>
```

```
            </material>
            <origin rpy="0 0 0" xyz="0 0 0.15"/>
        </visual>

        <collision>
            <geometry>
                <cylinder length="0.3" radius="0.03"/>
            </geometry>
            <origin xyz="0 0 0.15" rpy="0 0 0" />
        </collision>
    </link>

    <joint name="arm2_arm3_joint" type="revolute">
        <parent link="arm2_link"/>
        <child link="arm3_link"/>
        <origin xyz="0.0 0.0 0.5" rpy="0 0 0" />
        <axis xyz="0 1 0"/>
        <limit lower="-2.14" upper="2.14" effort="100" velocity="100" />
        <dynamics damping="10" friction="1.0"/>
    </joint>

    <gazebo reference="base_link">
        <material>Gazebo/Orange</material>
    </gazebo>

    <gazebo reference="arm1_link">
        <material>Gazebo/Blue</material>
    </gazebo>

    <gazebo reference="arm2_link">
        <material>Gazebo/White</material>
    </gazebo>

    <gazebo reference="arm3_link">
        <material>Gazebo/Red</material>
    </gazebo>

    <gazebo>
        <plugin filename="libgazebo_ros2_control.so" name="gazebo_ros2_control">
            <robot_sim_type>gazebo_ros2_control/GazeboSystem</robot_sim_type>
            <parameters>/home/kashyap/ros_ws/src/manipulator/config/control.yaml</paramet
ers>
        </plugin>
    </gazebo>

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

        <joint name="base_arm1_joint">
            <command_interface name="position">
```

```
            <param name="min">-2.14</param>
            <param name="max">2.14</param>
        </command_interface>
        <state_interface name="position"/>
        <param name="initial_position">0.0</param>
    </joint>
    <joint name="arm1_arm2_joint">
        <command_interface name="position">
        <param name="min">-2.14</param>
        <param name="max">2.14</param>
        </command_interface>
        <state_interface name="position"/>
        <param name="initial_position">0.1</param>
    </joint>
    <joint name="arm2_arm3_joint">
        <command_interface name="position">
            <param name="min">-2.14</param>
            <param name="max">2.14</param>
        </command_interface>
        <state_interface name="position"/>
        <param name="initial_position">0.2</param>
    </joint>

</ros2_control>

</robot>


cd ..
cd launch
touch arm_rviz.launch.py
from launch import LaunchDescription
from launch_ros.actions import Node


def generate_launch_description():

    urdf_file = urdf = '/home/kashyap/ros_ws/src/manipulator/urdf/arm.urdf'

    joint_state_publisher_node = Node(
        package="joint_state_publisher_gui",
        executable="joint_state_publisher_gui",
        name="joint_state_publisher_gui",
        output="screen",
        arguments=[urdf_file]
    )
    robot_state_publisher_node = Node(
        package="robot_state_publisher",
        executable="robot_state_publisher",
        name="robot_state_publisher",
        output="screen",
        arguments=[urdf_file]
```

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

    nodes_to_run = [
        joint_state_publisher_node,
        robot_state_publisher_node,
        rviz_node
    ]

    return LaunchDescription(nodes_to_run)
```

```
touch arm_gazebo.launch.py

import os
from launch import LaunchDescription
from launch.actions import ExecuteProcess
from launch_ros.actions import Node

def generate_launch_description():
    urdf_file = '/home/kashyap/ros_ws/src/manipulator/urdf/arm.urdf'

    return LaunchDescription(
        [
            ExecuteProcess(
                cmd=["gazebo","-s","libgazebo_ros_factory.so",],
                output="screen",
            ),
            Node(
                package="gazebo_ros",
                executable="spawn_entity.py",
                arguments=["-entity","manipulator","-b","-file", urdf_file],
            ),
            Node(
                package="robot_state_publisher",
                executable="robot_state_publisher",
                output="screen",
                arguments=[urdf_file],
            ),

        ]
    )
cd ~/ros2_ws/src/urdf_tutorial
mkdir config
cd config
touch control.yaml
```

```
controller_manager:
  ros__parameters:
    update_rate: 100
    joint_state_broadcaster:
      type: joint_state_broadcaster/JointStateBroadcaster
    joint_trajectory_controller:
      type: joint_trajectory_controller/JointTrajectoryController

joint_trajectory_controller:
  ros__parameters:
    joints:
      - base_arm1_joint
      - arm1_arm2_joint
      - arm2_arm3_joint

    command_interfaces:
      - position

    state_interfaces:
      - position

    state_publish_rate: 50.0
    action_monitor_rate: 20.0

    allow_partial_joints_goal: false
    open_loop_control: true
```

```
    constraints:
      stopped_velocity_tolerance: 0.01
      goal_time: 0.0
      joint1:
        trajectory: 0.05
        goal: 0.03
```

cd src/launch
touch arm_control.launch.py
Edit arm_control.launch.py

```
import os

from launch import LaunchDescription

from launch.actions import ExecuteProcess, IncludeLaunchDescription,
RegisterEventHandler

from launch_ros.actions import Node

from launch.event_handlers import OnProcessExit

from launch.launch_description_sources import PythonLaunchDescriptionSource

from ament_index_python.packages import get_package_share_directory


import xacro


def generate_launch_description():


    urdf_file = '/home/kashyap/ros_ws/src/manipulator/urdf/arm.urdf'

    controller_file = '/home/kashyap/ros_ws/src/manipulator/config/control.yaml'

    robot_description = {"robot_description": urdf_file}


    gazebo = IncludeLaunchDescription(
```

```
        PythonLaunchDescriptionSource("/home/kashyap/ros_ws/src/manipulator/launch/gazebo.launch.py"),
    )

    doc = xacro.parse(open(urdf_file))
    xacro.process_doc(doc)
    params = {'robot_description': doc.toxml()}

    node_robot_state_publisher = Node(
        package='robot_state_publisher',
        executable='robot_state_publisher',
        output='screen',
        parameters=[params]
    )

    spawn_entity = Node(package='gazebo_ros', executable='spawn_entity.py',
        arguments=["-entity","manipulator","-b","-file", urdf_file],
        output='screen'
    )

    load_joint_state_controller = ExecuteProcess(
        cmd=['ros2', 'control', 'load_controller', '--set-state', 'active', 'joint_state_broadcaster'],
        output='screen'
    )

    load_joint_trajectory_controller = ExecuteProcess(
```

```
        cmd=['ros2', 'control', 'load_controller', '--set-state', 'active',
'joint_trajectory_controller'],
        output='screen'
    )


    return LaunchDescription(
        [
            RegisterEventHandler(
                event_handler=OnProcessExit(
                    target_action=spawn_entity,
                    on_exit=[load_joint_state_controller],
                )
            ),

            RegisterEventHandler(
                event_handler=OnProcessExit(
                    target_action=load_joint_state_controller,
                    on_exit=[load_joint_trajectory_controller],
                )
            ),

            gazebo,

            node_robot_state_publisher,

            spawn_entity,
```

```
        Node(
            package="controller_manager",
            executable="ros2_control_node",
            parameters=[robot_description, controller_file],
            output="screen"
        )
    ]
)
```

Execute in Terminal #1

```
colcon build --packages-select urdf_tutorial
```

Execute in Terminal #1

```
ros2 launch urdf_tutorial arm_rviz.launch.py
```

Execute in Terminal #2

```
ros2 launch urdf_tutorial arm_gazebo.launch.py
```

Execute in Terminal #3

```
ros2 launch urdf_tutorial arm_control.launch.py
```

```
cd ros2_ws/src/urdf_tutorial/urdf_tutorial/
touch controller_fk.py
```

chmod +x controller.py

```
import rclpy

from rclpy.node import Node

from sensor_msgs.msg import JointState

from rclpy.clock import Clock

import sys


class TrajectoryPublisher(Node):


    def __init__(self):

        super().__init__('trajectory_node')
```

```
    topic_ = "/joint_states"
    self.joints = ['base_arm1_joint', 'arm1_arm2_joint', 'arm2_arm3_joint']

    # Handle command-line arguments
    if len(sys.argv) < 4:
        self.get_logger().error("Not enough arguments provided. Using default values.")
        self.goal_ = [0.5, 0.5, 0.5]  # Default values
    else:
        try:
            self.goal_ = [float(sys.argv[1]), float(sys.argv[2]), float(sys.argv[3])]
        except ValueError:
            self.get_logger().error("Invalid argument(s) provided. Using default values.")
            self.goal_ = [0.0, 0.0, 0.0]  # Default values

    self.publisher_ = self.create_publisher(JointState, topic_, 10)
    self.timer_ = self.create_timer(0.1, self.timer_callback)

def timer_callback(self):
    msg = JointState()
    current_time = Clock().now().to_msg()
    msg.header.stamp.sec = current_time.sec
    msg.header.stamp.nanosec = current_time.nanosec
    msg.name = self.joints
    msg.position = self.goal_
    self.publisher_.publish(msg)
    self.get_logger().info("Publishing position: {}".format(self.goal_))
```

```
def main(args=None):
    rclpy.init(args=args)
    node = TrajectoryPublisher()
    rclpy.spin(node)
    node.destroy_node()
    rclpy.shutdown()

if __name__ == '__main__':
    main()
```

Edit setup.py as

```
from setuptools import find_packages, setup
import os
from glob import glob

package_name = 'manipulator'

setup(
    name=package_name,
    version='0.0.0',
    packages=find_packages(exclude=['test']),
    data_files=[
        ('share/ament_index/resource_index/packages',
            ['resource/' + package_name]),
        ('share/' + package_name, ['package.xml']),
```

```
        (os.path.join('share',package_name),glob('launch/*')),
        (os.path.join('share',package_name),glob('urdf/*')),
        (os.path.join('share',package_name),glob('config/*')),
    ],
    install_requires=['setuptools'],
    zip_safe=True,
    maintainer='kashyap',
    maintainer_email='kashyapj@todo.todo',
    description='TODO: Package description',
    license='TODO: License declaration',
    tests_require=['pytest'],
    entry_points={
        'console_scripts': [
            "controller_fk = manipulator.controller_fk:main",
                ],
    },
)
```

Execute in Terminal #1

```
colcon build --packages-select urdf_tutorial
```

Execute in Terminal #1

```
ros2 launch urdf_tutorial arm_rviz.launch.py
```

Execute in Terminal #2

```
ros2 launch urdf_tutorial arm_control.launch.py
```

Execute in Terminal #3

```
ros2 run urdf_tutorial controller_fk 0.5 0.2 0.5
```

ROS2 parameters:

Parameters help to provide the values while running the code.

ros2 param list

Edit the controller.py

```
#!/usr/bin/env python3


#colcon build --packages-select urdf_tutorial
#ros2 run urdf_tutorial controller --ros-args -p end_location:=[3.5,1.5,-1.2]

import rclpy
from rclpy.node import Node
from builtin_interfaces.msg import Duration
from trajectory_msgs.msg import JointTrajectory, JointTrajectoryPoint

class TrajectoryPublisher(Node):

    def __init__(self):
        super().__init__('trajectory_node')
        topic_ = "/joint_trajectory_controller/joint_trajectory"
        self.joints = ['joint_1', 'joint_2', 'joint_4']
        #self.goal_ =[1.5, 0.5, 1.2]
        self.declare_parameter("joint_angles", [1.5, 0.5, 1.2])
        self.goal_=self.get_parameter("joint_angles").value
        self.publisher_ = self.create_publisher(JointTrajectory, topic_, 10)
        self.timer_ = self.create_timer(1,self.timer_callback)



    def timer_callback(self):
        msg = JointTrajectory()
        msg.joint_names = self.joints
        point = JointTrajectoryPoint()
        point.positions = self.goal_
        point.time_from_start = Duration(sec=2)
        msg.points.append(point)
        self.publisher_.publish(msg)

def main(args=None):
    rclpy.init(args=args)
    node = TrajectoryPublisher()
    rclpy.spin(node)
    node.destroy_node()
    rclpy.shutdown()

if __name__ == '__main__':
    main()
```

Execute in Terminal #1

#colcon build --packages-select urdf_tutorial

Execute in Terminal #2

ros2 launch urdf_tutorial arm_control.launch.py
ros2 control load_controller --set-state active joint_state_broadcaster
ros2 control load_controller --set-state active joint_trajectory_controller

Execute in Terminal #3

#ros2 run urdf_tutorial controller --ros-args -p joint_angles:=[3.5,1.5,-1.2]

Exercise 1: Replicate the process for UR5e robot given its urdf file.

Exercise 2: Write a python code to move the manipulator to end location using inverse kinematics.

Viva Questions: Compute the inertia parameters for the each block used in the three_wheeled_robot and manipulator.

References

https://docs.ros.org/en/humble/Tutorials/URDF/Using-URDF-with-Robot-State-Publisher.html

https://github.com/benbongalon/ros2-urdf-tutorial/tree/master/urdf_tutorial

https://github.com/cra-ros-pkg/robot_localization/tree/humble-devel

https://github.com/ros/robot_state_publisher/tree/humble

https://github.com/ros/joint_state_publisher/tree/humble

http://gazebosim.org/tutorials?tut=ros_urdf