{Learn, Create, Innovate};

ROS

Robot Modelling/Visualisation Tools

Property of MCR2

MCR2 Manchester Robotics

Robot Descriptions

URDF Files

{Learn, Create, Innovate};

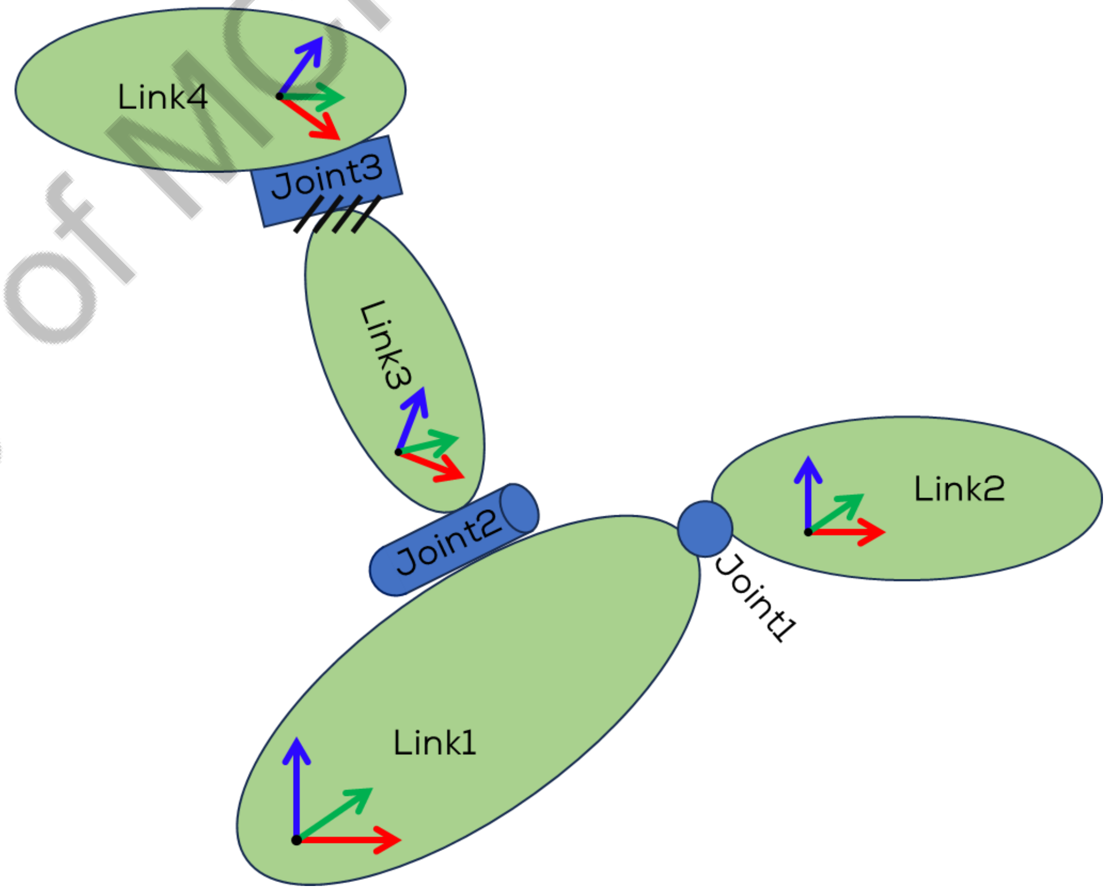
Property of MCR2

MCR2
Manchester Robotics

URDF Files

- URDF (Unified Robot Description Format) files are XML-based files used to describe a robot's structure, kinematics, dynamics, and visual appearance in the context of robotics and simulation.

- URDF is commonly used in robotics frameworks like ROS (Robot Operating System) to represent robots and their components.

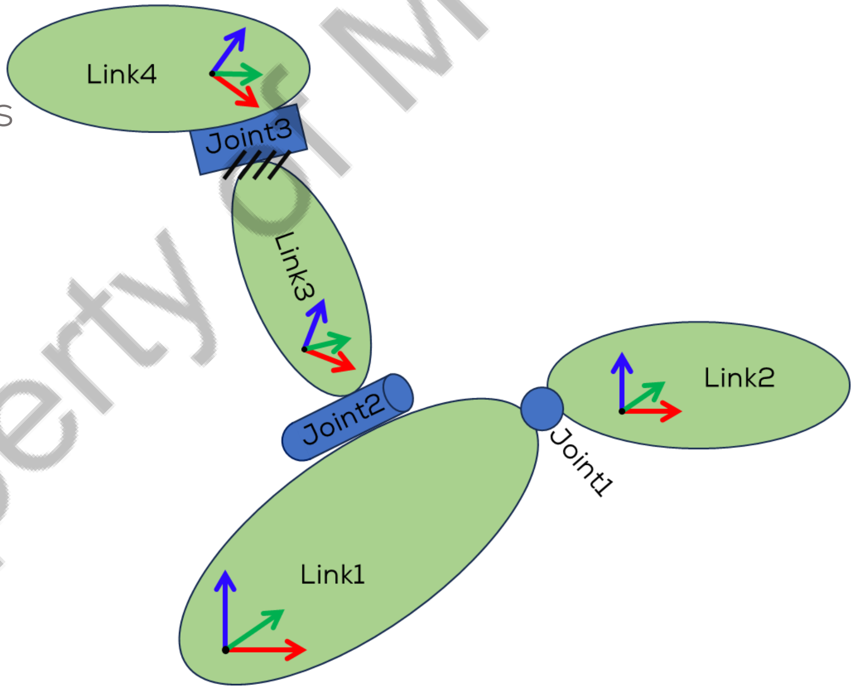


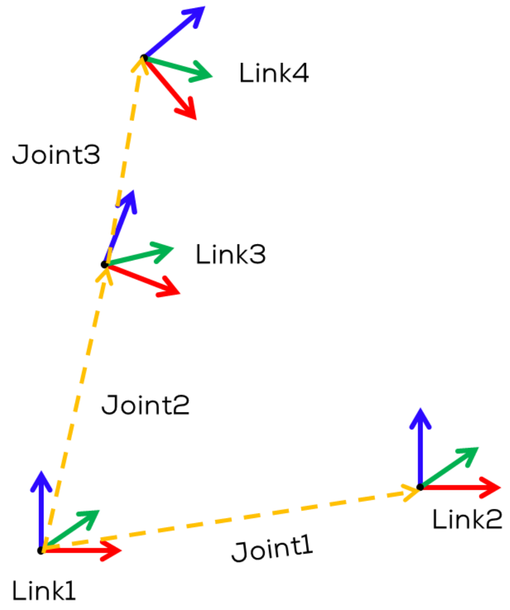
Property of MCR2

URDF Files

- URDF files provide a standardised format to describe robot models, allowing simulation and visualisation tools to load and manipulate robots accurately.

- They are widely used in the robotics community and play a crucial role in developing and integrating robotic systems.

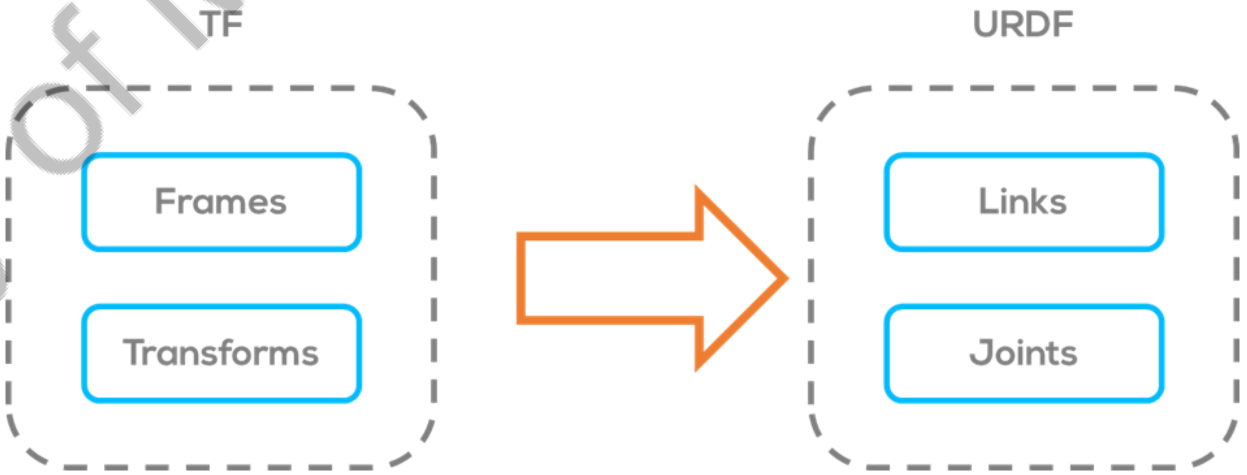




Property of MCR2

URDF Files

- URDF describes a robot as a tree of links, that are connected by joints.
  - The links represent the physical components of the robot, and the joints represent how one link moves relative to another link, defining the location of the links in space.
- This concept can be related to the TF (Transforms) concepts of frames and links as follows
  - Frames -> Links
  - Transforms -> Joints

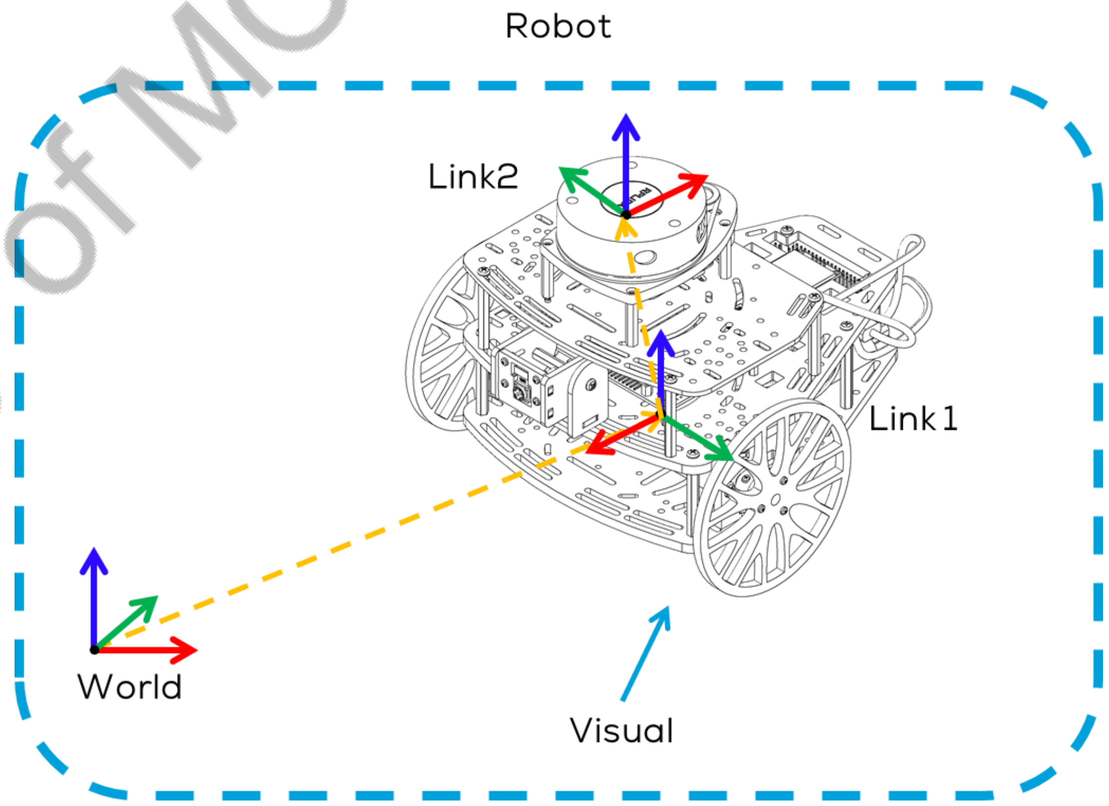


Property of MCR2

URDF Files

Contents

- **Robot:** The root element of the URDF file, representing the entire robot. It encapsulates all the other elements.
- **Links:** A link represents a rigid body or a component of the robot. It describes the visual, inertial, and collision properties of the link. Each link may have one or more visual and collision elements associated with it.
- **Visual:** Defines the visual appearance of a link, including its geometry (shape), material properties (e.g., color, texture), and transformation (position and orientation) relative to the link.



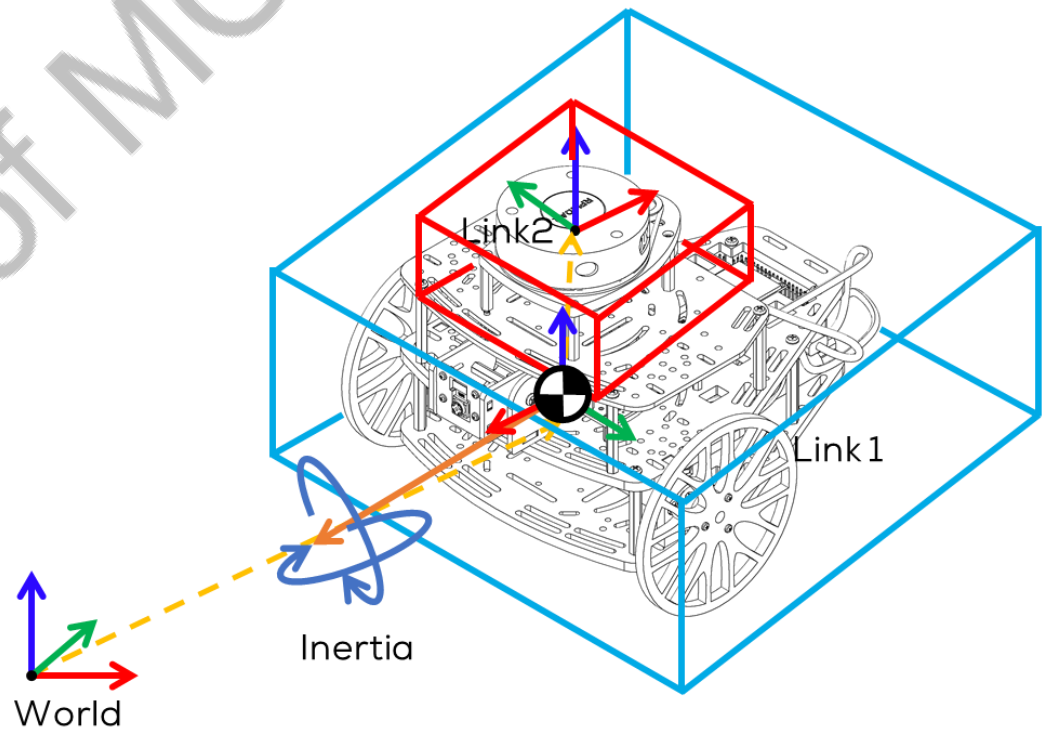
Property of MCR2

URDF Files

Contents

- **Collision:** Specifies the collision properties of a link, such as the collision geometry and its transformation relative to the link.
- **Inertial:** Describes the inertial properties of a link, such as mass, center of mass, and moments of inertia. These properties are used for dynamics calculations.

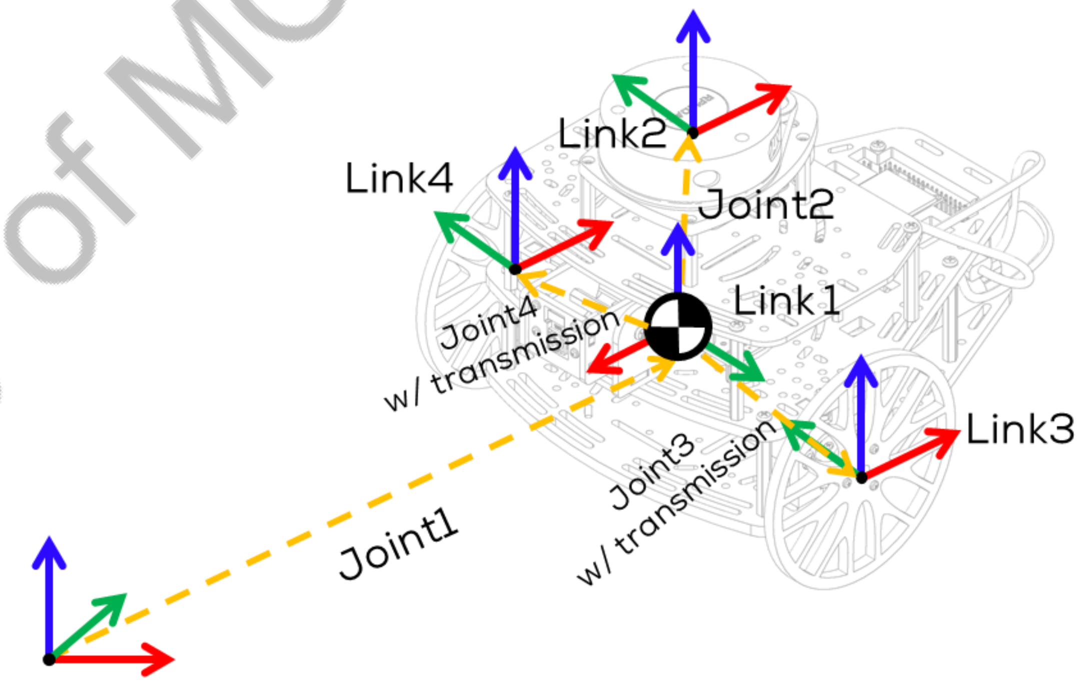
Property of MCR2



URDF Files

Contents

- **Joints:** Joints define the kinematic connections between links. They specify the type of joint (e.g., revolute, prismatic) and its properties (e.g., limits, axis, origin).
- **Transmission:** A transmission element connects a joint to an actuator, specifying how the joint motion is driven.
- **Plugins:** URDF supports plugins, allowing users to extend the capabilities of the robot description. Plugins can provide additional features like custom collision checking or dynamic properties.



Property of MCR2

URDF

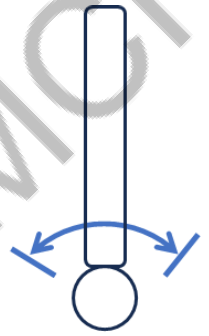
Joints

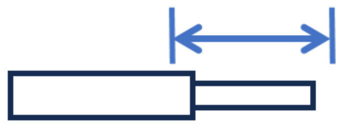
{Learn, Create, Innovate};

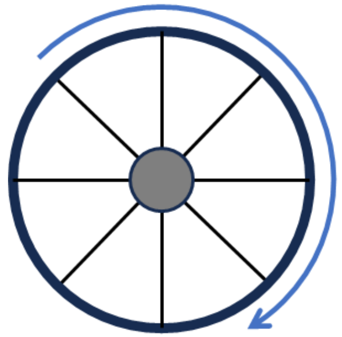
Property of MCR2

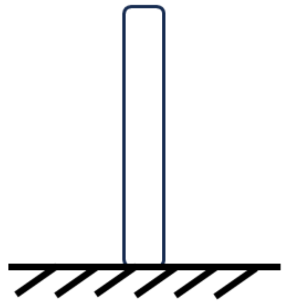
URDF Files: Joints

- Revolute – A rotational motion, with minimum/maximum angle limits.
- Continuous – A rotational motion with no limit (e.g. a wheel).
- Prismatic – A linear sliding motion, with minimum/maximum position limits.
- Fixed – The child link is rigidly connected to the parent link. This is what we use for those "convenience" links.
    - *Some other joints might be available (some deprecated). Some only work on Gazebo.


Revolute


Prismatic


Continuous

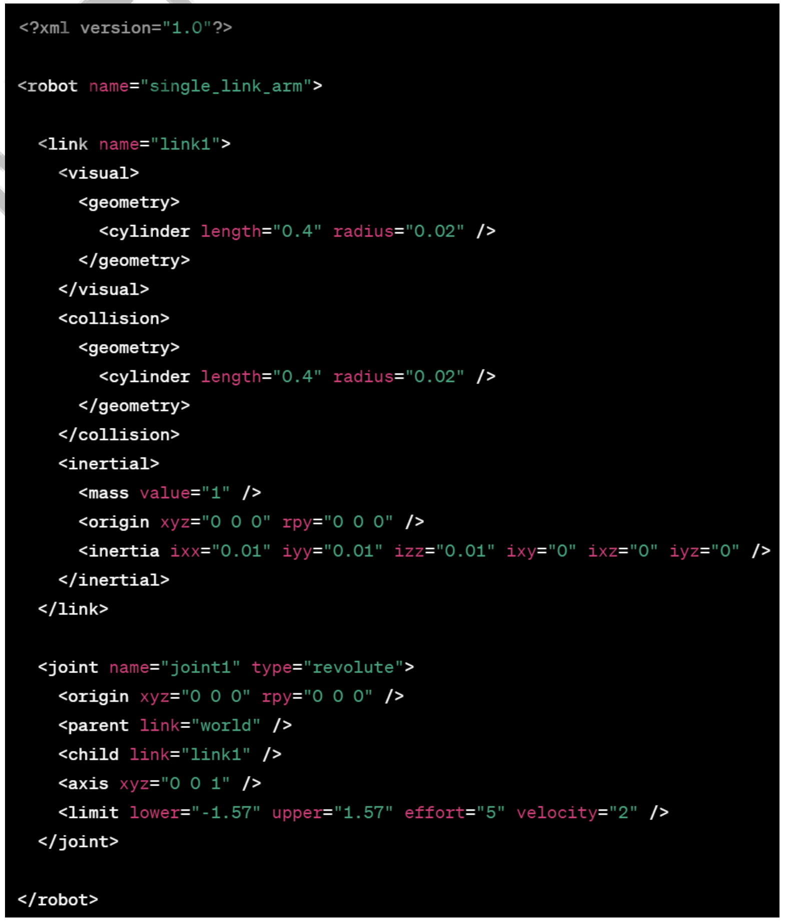

Fixed

Property of MCR2

Syntax

- **XML Declaration:** The file begins with an XML declaration that specifies the version of XML being used. For URDF, it is typically <?xml version="1.0"?>.
- **Root Element:** The root element of the URDF file is <robot>. It encapsulates all the other elements in the file and typically includes attributes like name to specify the robot's name.
- **Links:** Inside the <robot> element, you define the robot's links using the <link> element. Each <link> represents a component or a rigid body of the robot. It may have attributes like name.

Property of MCR2



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

<robot name="single_link_arm">

  <link name="link1">
    <visual>
      <geometry>
        <cylinder length="0.4" radius="0.02" />
      </geometry>
    </visual>
    <collision>
      <geometry>
        <cylinder length="0.4" radius="0.02" />
      </geometry>
    </collision>
    <inertial>
      <mass value="1" />
      <origin xyz="0 0 0" rpy="0 0 0" />
      <inertia ixx="0.01" iyy="0.01" izz="0.01" ixy="0" ixz="0" iyz="0" />
    </inertial>
  </link>

  <joint name="joint1" type="revolute">
    <origin xyz="0 0 0" rpy="0 0 0" />
    <parent link="world" />
    <child link="link1" />
    <axis xyz="0 0 1" />
    <limit lower="-1.57" upper="1.57" effort="5" velocity="2" />
  </joint>

</robot>
```

Syntax

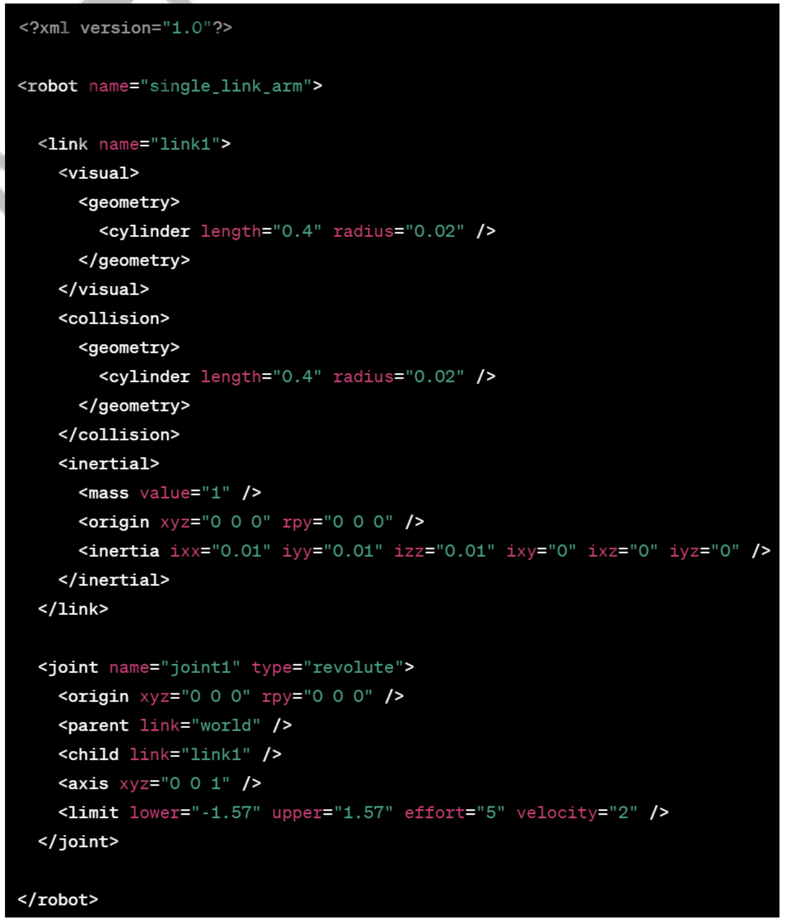
- Visuals: Within each <link>, you can define the visual properties using the <visual> element. It includes attributes like names and contains child elements to describe the visual geometry, material properties, and transformations.
- Collisions: Similar to visuals, collisions are defined using the <collision> element within each <link>. It represents the collision geometry and properties associated with a link.
- Inertial: The <inertial> element is used to specify the inertial properties of a link. It includes child elements such as <mass> to define the mass, <inertia> to specify the moments of inertia, and <origin> to describe the position and orientation of the inertial frame.

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

<robot name="single_link_arm">

  <link name="link1">
    <visual>
      <geometry>
        <cylinder length="0.4" radius="0.02" />
      </geometry>
    </visual>
    <collision>
      <geometry>
        <cylinder length="0.4" radius="0.02" />
      </geometry>
    </collision>
    <inertial>
      <mass value="1" />
      <origin xyz="0 0 0" rpy="0 0 0" />
      <inertia ixx="0.01" iyy="0.01" izz="0.01" ixy="0" ixz="0" iyz="0" />
    </inertial>
  </link>

  <joint name="joint1" type="revolute">
    <origin xyz="0 0 0" rpy="0 0 0" />
    <parent link="world" />
    <child link="link1" />
    <axis xyz="0 0 1" />
    <limit lower="-1.57" upper="1.57" effort="5" velocity="2" />
  </joint>

</robot>
```

Property of MCR2

Syntax

- **Joints:** Joints are defined within <robot> using the <joint> element. Each joint element represents a kinematic connection between two links. Joints have attributes like name, type, and contain child elements to define properties like limits, axis, and origin.
- **Transmission:** The <transmission> element connects a joint to an actuator, specifying how the joint motion is driven. It includes child elements like <type>, <joint> (to specify the joint being controlled), and <actuator> (to define the actuator properties).
- **Plugins:** URDF supports plugins for extending its capabilities. Plugins are added as <plugin> elements within the relevant sections of the URDF file. They can provide additional features or custom functionality.

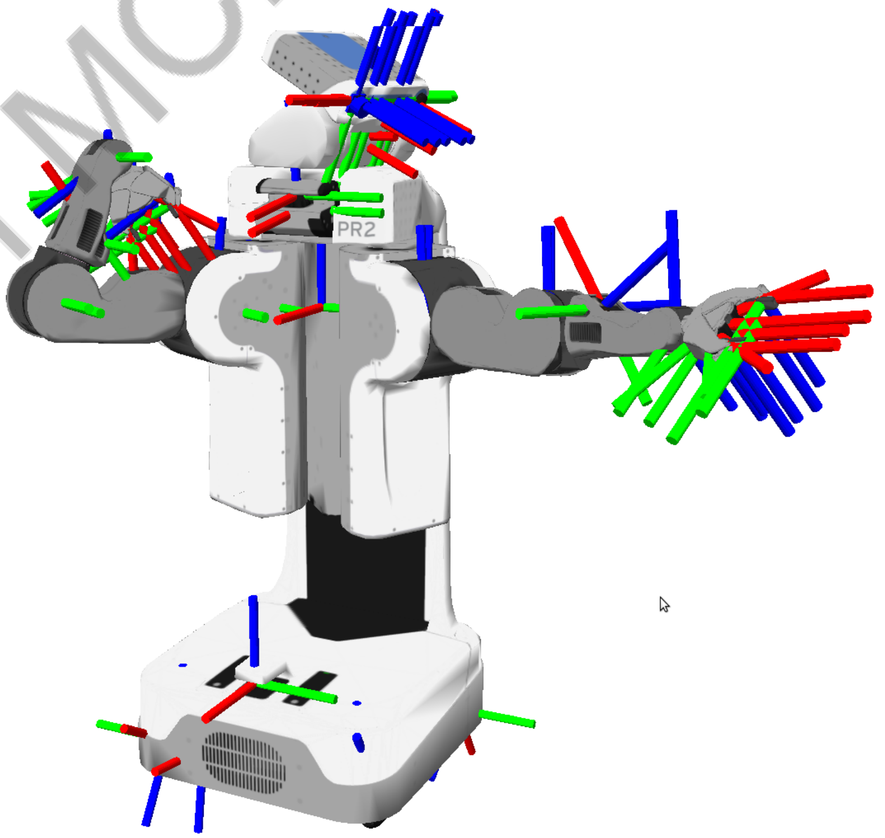


```xml
<?xml version="1.0"?>

<robot name="single_link_arm">

  <link name="link1">
    <visual>
      <geometry>
        <cylinder length="0.4" radius="0.02" />
      </geometry>
    </visual>
    <collision>
      <geometry>
        <cylinder length="0.4" radius="0.02" />
      </geometry>
    </collision>
    <inertial>
      <mass value="1" />
      <origin xyz="0 0 0" rpy="0 0 0" />
      <inertia ixx="0.01" iyy="0.01" izz="0.01" ixy="0" ixz="0" iyz="0" />
    </inertial>
  </link>

  <joint name="joint1" type="revolute">
    <origin xyz="0 0 0" rpy="0 0 0" />
    <parent link="world" />
    <child link="link1" />
    <axis xyz="0 0 1" />
    <limit lower="-1.57" upper="1.57" effort="5" velocity="2" />
  </joint>

</robot>
```

Property of MCR2

Robot State Publisher

- URDF files require a translator, so that ROS can use them.
- Different translators have been developed to “translate” URDF files into TF functions.
- ROS has different packages to manage URDF files.
- To transform the URDF files to robot states, visualise the robot and transform data between different coordinate frames; ROS uses a package called *“robot_state_publisher”*.
- This package allows you to publish the state of a robot to tf2.



Property of MCR2

Robot State Publisher

- Once the state gets published, it is available to all components in the system that also use tf2.
- The package takes the joint angles of the robot as input and publishes the 3D poses of the robot links, using a kinematic tree model of the robot.
- The package can both be used as a library and as a ROS node.

Usage as a ROS Node:

- *"robot_state_publisher"* uses the URDF specified by the parameter *"robot_description"* and the joint positions from the topic *"joint_states"* to calculate the forward kinematics of the robot and publish the results via tf2.

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

    <arg name="fixed_joint_test" default="$(find joints_act)/urdf/fixed_ex.urdf"/>

    <param name="robot_description" command="cat $(arg fixed_joint_test)" />
    <node pkg="robot_state_publisher" type="robot_state_publisher" name="link_ex_pub" >
    </node>

</launch>
```

Property of MCR2

Activity 1

Creating a simple URDF file

`{Learn, Create, Innovate};`

Prope

MCR2 Manchester Robotics

Activity 1

1. Make a new package called "joints_act" with the following library packages

geometry_msgs, nav_msgs, rospy, sensor_msgs, std_msgs, tf2_ros, tf_conversions visualization_msgs

```
catkin_create_pkg joints_act geometry_msgs nav_msgs rospy
sensor_msgs std_msgs tf2_ros tf_conversions
visualization_msgs
```

2. Create a "urdf" folder inside the previously created package and a URDF file "fixed_ex.urdf" inside

```
mkdir urdf && touch urdf/fixed_ex.urdf
```

3. Paste the following code inside the "fixed_ex.urdf" file.
   - Code can be found on GitHub.
4. Create a launch file

```
mkdir launch && touch launch/fixed.launch
```

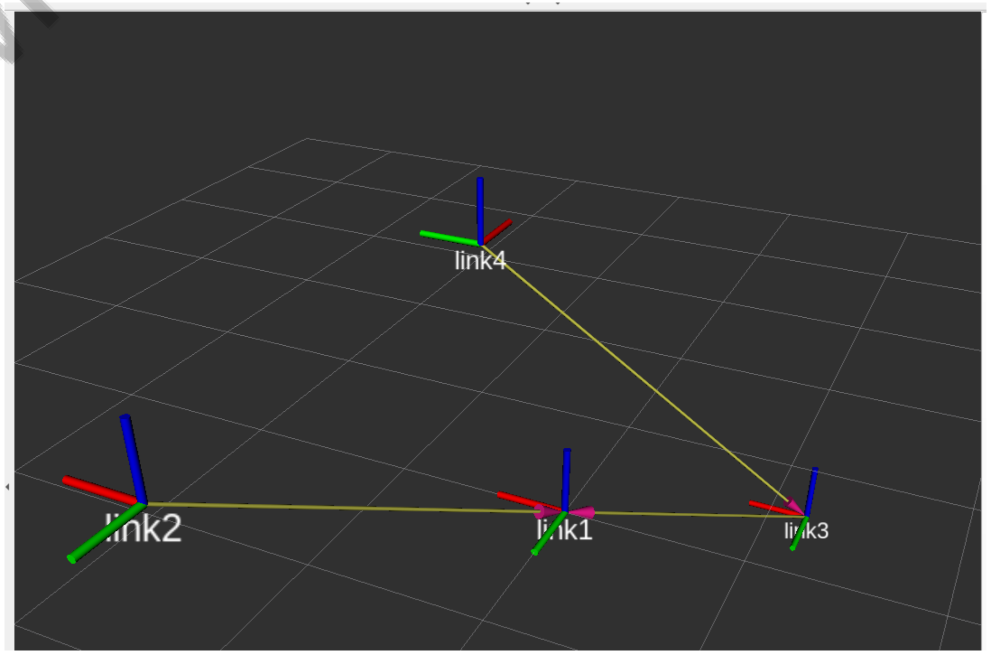
```
<?xml version="1.0"?>

<robot name="link_example">

  <link name="link1" />
  <link name="link2" />
  <link name="link3" />
  <link name="link4" />

    <joint name="joint1" type="fixed">
        <parent link="link1"/>
        <child link="link2"/>
        <origin xyz="1 2 1" rpy="0 0 0" />
    </joint>

    <joint name="joint2" type="fixed">
        <parent link="link1"/>
        <child link="link3"/>
        <origin xyz="-1 -2 -1" rpy="0 0 0" />
    </joint>

    <joint name="joint3" type="fixed">
        <parent link="link3"/>
        <child link="link4"/>
        <origin xyz="2 1 2" rpy="0 0 -1.57" />
    </joint>

</robot>
```

Property of MCR2

Activity 1

5. Paste the following code inside the launch file, compile and launch

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

    <arg name="fixed_joint_test" default="$(find
joints_act)/urdf/fixed_ex.urdf"/>
    <param name="robot_description" command="cat $(arg fixed_joint_test)"
/>
    <node pkg="robot_state_publisher" type="robot_state_publisher"
name="link_ex_pub" >
    </node>

<node name="rviz" pkg="rviz" type="rviz" required="true" />
</launch>
```



6. Make the fixed frame to be “link 1”
7. Add the marker
   - Press Add
   - ››By display type››TF

Property of MCR2

Activity 2

Creating a movable joint using URDF

`{Learn, Create, Innovate};`

Prope

Activity 2

1. Create a new "urdf" named "continuos_ex.urdf" inside the previously created package "joints_act"

```
touch urdf/continuos_ex.urdf
```

2. Paste the following code inside the "continuos_ex.urdf" file.
   - Code can be found on GitHub.
3. Create a launch file for this node (next slide)

```
touch launch/continuos.launch
```

5. Add the TF view
   - Press Add
   - ››By display type››TF
6. Run the tf_tree…

```
rosrun rqt_tf_tree rqt_tf_tree
```

7. **Why nothing appears?!** … Because ROS does not know the state of the movable joints! (that is why ROS does not show the movable joints)

```
<?xml version="1.0"?>                                        <!--Declare XML Version
typical value "1.0"-->

<!--Start a Robot description-->
<robot name="continuos_joint_example">                      <!--Define a new robot
name: Robot Name-->

<!--Declare Links to be used-->
  <link name="link1" />
  <link name="link2" />

<!--Declare Joints to be used-->
<!--Declare Joint element 'name' and 'type' (revolute, continuos,
prismatic, fixed)-->

<joint name="joint1" type="continuous">
 <parent link="link1"/>                     <!--parent: Parent link name -->
 <child link="link2"/>                      <!--child: child link name -->
 <origin xyz="0.5 1 0.5" rpy="0 0 0" />
 <!--origin: This is the transform from the parent link to the child link
xyz: xyz offset rpy:rotaton offset (radians)-->
 <axis xyz="0 0 1 " />
<!--Rotation/Translation axis for the joints(revolute,continuos/prismatic)
noy used with Fixed Joints -->
</joint>

</robot>
```

Property of MCR2

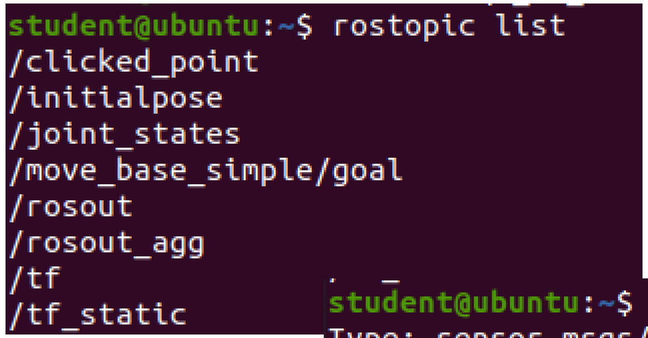
Activity 2

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

    <arg name="continuos_ex" default="$(find joints_act)/urdf/continuos_ex.urdf"/>
    <param name="robot_description" command="cat $(arg continuos_ex)" />
    <node pkg="robot_state_publisher" type="robot_state_publisher" name="continuos_test_pub" >
    </node>

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

</launch>
```

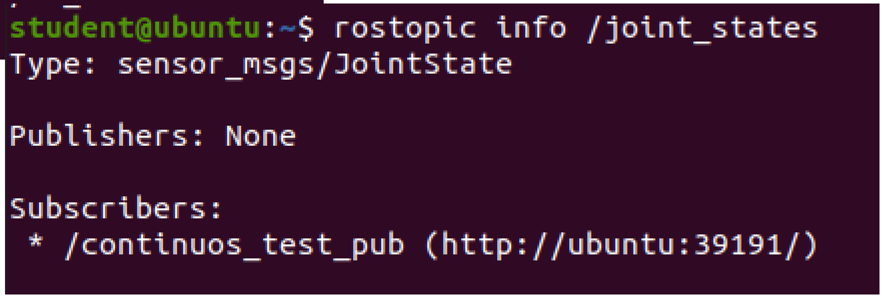
Property of MCR2

Joint State Publisher

- ROS does not know the *"state"* of the joints in a robot i.e., position, velocity and effort.
- The Joint State Publisher is a package in Robot Operating System (ROS), designed to publish joint state information for a robot, so ROS can " know " each joint's state at a point in time.
- It's a critical component in ROS for robotics applications because it allows us to test and publish the states of the robot's joints.
- When using non-fixed joints, ROS will not publish the TF information unless they are "known"; in other words, ROS needs to know the state of the joint. To do this, the state needs to be published in the */joint_states* topic.



```
student@ubuntu:~$ rostopic list
/clicked_point
/initialpose
/joint_states
/move_base_simple/goal
/rosout
/rosout_agg
/tf
/tf_static
```

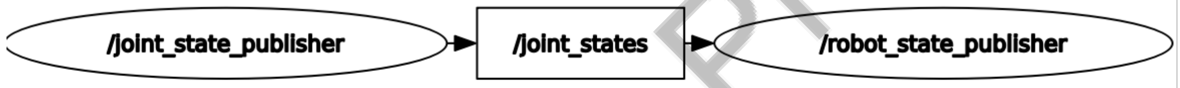


```
student@ubuntu:~$ rostopic info /joint_states
Type: sensor_msgs/JointState

Publishers: None

Subscribers:
 * /continuos_test_pub (http://ubuntu:39191/)
```

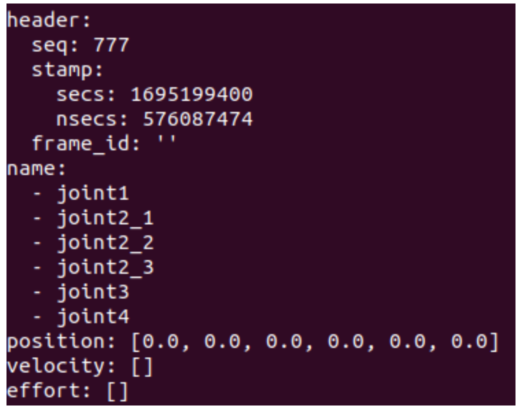
Property of MCR2

Joint State Publisher

- This package publishes sensor_msgs/JointState messages for a robot.
- The package reads the robot_description parameter from the parameter server, finds all the non-fixed joints and publishes a JointState message with all those joints defined.
- The joint state publisher, publishes default positional values to the joints, to the robot state publisher, using the JointState message, in the /joint_state topic.



- Remember!
  - The joint state publisher is not intended to send commands to a joint (except in this activity, where is used to test our URDF file) it is intended to read and visualise the state of joints from the different robot sensors.
  - When testing our robot, in RVIZ, the joint_state_publisher allows us to make a simple test by publishing into the /joint_state topic and moving the Joint; this, however, is only for testing purposes and should not be used when reading real data!



```
header:
  seq: 777
  stamp:
    secs: 1695199400
    nsecs: 576087474
  frame_id: ''
name:
  - joint1
  - joint2_1
  - joint2_2
  - joint2_3
  - joint3
  - joint4
position: [0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
velocity: []
effort: []
```

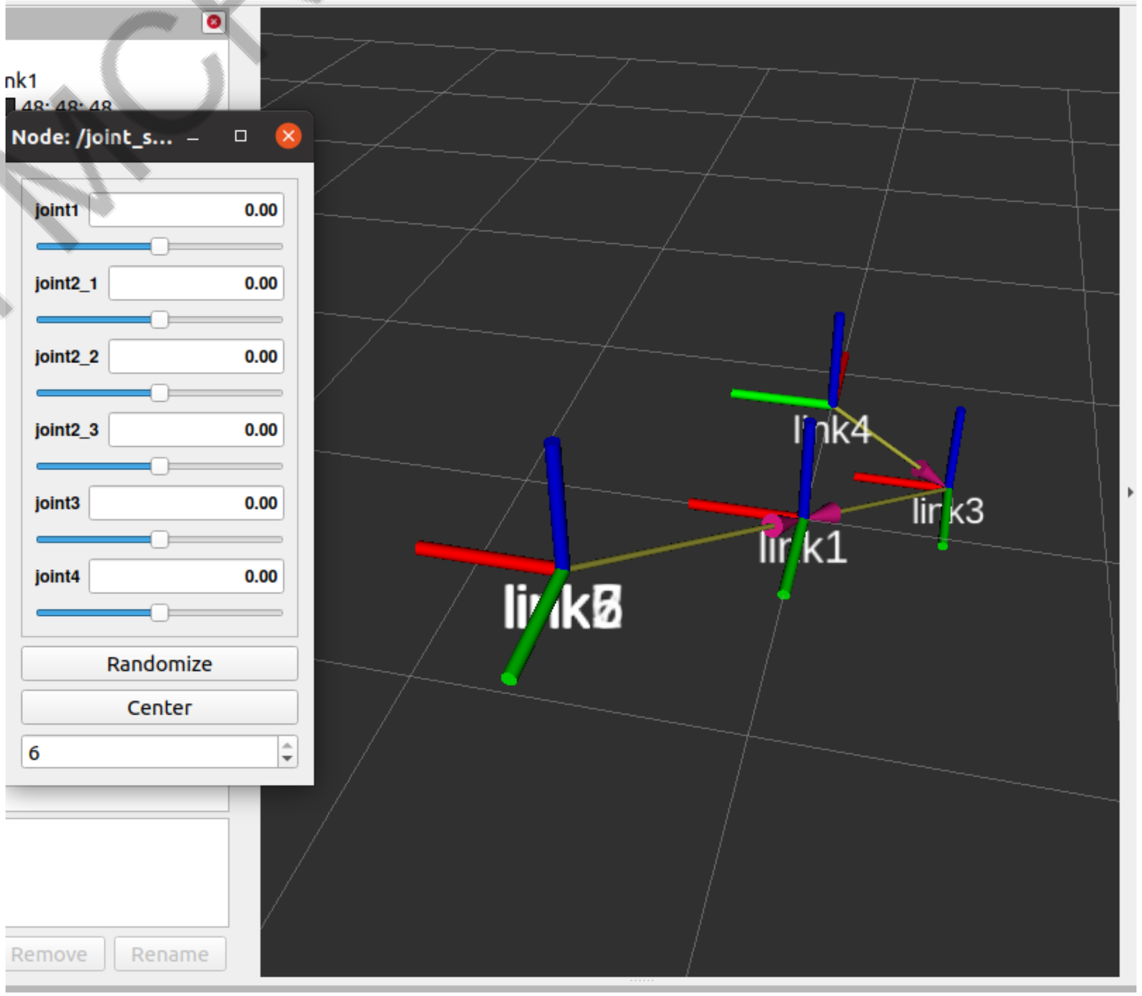
Property of MCR2

Activity 2

8. Change the launch file to add the joint state publisher

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

    <arg name="continuos_ex" default="$(find
joints_act)/urdf/continuos_ex.urdf"/>
    <param name="robot_description" command="cat $(arg continuos_ex)" />
    <node pkg="robot_state_publisher" type="robot_state_publisher"
name="continuos_test_pub" >
    </node>

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

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

</launch>
```

9. Use the GUI to move the joints!
10. Add More continuous joints to this activity!



Property of MCR2

Activity 2: Extension

1. Create a new "urdf" named "prismatic_ex.urdf" inside the previously created package "joints_act"

```
touch urdf/prismatic_ex.urdf
```

2. Define a prismatic joint inside the "prismatic_ex.urdf" file.
   - Code can be found on GitHub.
3. Create a launch file for this node (next slide)

```
touch launch/prismatic.launch
```

4. Add the joint_state publisher to the launch file

```
rosrun rqt_tf_tree rqt_tf_tree
```

5. Add the TF view
   - Press Add
   - ››By display type››TF
6. Run the tf_tree…

```
<joint name="joint2" type="prismatic">  <!--Declare Joint element 'name' and 'type' (revolute, continuos, prismatic, fixed)-->
<parent link="link1"/>              <!--parent: Parent link name -->
<child link="link3"/>               <!--child: child link name -->
<origin xyz="0 0 0" rpy="0 0 0" /> <!--origin: This is the transform from the parent link to the child link xyz: xyz offset rpy:rotaton offset (radians)-->
<axis xyz="1 0 0" />                <!--Rotation/Translation axis for the joints (revolute,continuos/prismatic) noy used with Fixed Joints -->
<limit lower="-2" upper="2" effort="5.0" velocity="0.1"/>   <!--Establish the limits of a joint Lower/Upper: lower/upper limits for revolute/prismatic joints in rad or m, Effort: maximum joint force (torque or force) depend on the joint, Velocity: Max velocity of the joint (rad/s or m/s) -->
</joint>
```

Property of MCR2

Activity 2: Extension 2

1. Create a new “urdf” named “revolute_ex.urdf” inside the previously created package “joints_act”

```
touch urdf/revolute_ex.urdf
```

2. Define a prismatic joint inside the “revolute_ex.urdf” file.
   - Code can be found on GitHub.
3. Create a launch file for this node (next slide)

```
touch launch/revolute.launch
```

4. Add the joint_state publisher to the launch file

```
rosrun rqt_tf_tree rqt_tf_tree
```

5. Add the TF view
   - Press Add
   - ››By display type››TF
6. Run the tf_tree…

```
 <joint name="joint2_2" type="revolute">     <!--Declare Joint element 'name' and 'type' (revolute, continuos, prismatic, fixed)-->
<parent link="link5"/>          <!--parent: Parent link name -->
<child link="link6"/>           <!--child: child link name -->
<origin xyz="0 0 0" rpy="0 0 0" />       <!--origin: This is the transform from the parent link to the child link xyz: xyz offset rpy:rotaton offset (radians)-->
<axis xyz="0 1 0" />                     <!--Rotation/Translation axis for the joints (revolute,continuos/prismatic) noy used with Fixed Joints -->
<limit lower="-1.57" upper="1.57" effort="5.0" velocity="0.1"/>     <!--Establish the limits of a joint Lower/Upper: lower/upper limits for revolute/prismatic joints in rad or m, Effort: maximum joint force (torque or force) depend on the joint, Velocity: Max velocity of the joint (rad/s or m/s) -->
</joint>
```

Property of MCR2

Joint control

Joint State Publisher

{Learn, Create, Innovate};

Property of MCR2

Manchester Robotics

Joint state publisher

- As stated previously, the robot state publisher, will not publish the joints unless something is published into the /joint_states topic.
- The joint state publisher does this by publishing positional values of the joints to the /joint_states topic using a JointState message. This allows the user to test the joints before releasing the model.
- The JointState message holds the data that describes the state of a torque-controlled joint.
- Each joint is uniquely identified by its name
- ROS associate the joint name with the correct states.
- The state of each joint (revolute or prismatic) is defined by:
  - the position of the joint (rad or m),
  - the velocity of the joint (rad/s or m/s)
  - the effort applied in the joint (Nm or N).
- The header specifies the time at which the joint states were recorded.
- This message consists of a multiple arrays, one for each part of the joint state.
- All arrays in this message should have the same size, or be empty.

```
Header header

string[] name
float64[] position
float64[] velocity
float64[] effort
```

Property of MCR2

Activity 3

Making your own joint state publisher

{Learn, Create, Innovate};

Prope

MCR2 Manchester **Robotics**

Activity 3

Description

- In this activity, the user will create a node to control the position of a joint.
- The position will be controlled by publishing a JointState message into the /joint_states topic.

```
from sensor_msgs.msg import JointState

# Declare the output Messages
contJoints = JointState()

# Declare the output Messages
def init_joints():
    contJoints.header.frame_id = "link1"
    contJoints.header.stamp = rospy.Time.now()
    contJoints.name.extend(["joint1", "joint2"])
    contJoints.position.extend([0.0, 0.0])
    contJoints.velocity.extend([0.0, 0.0])
    contJoints.effort.extend([0.0, 0.0])
```

- For this activity, the URDF model “continuos_joint.urdf” will be used. This file can be found in the course repository. (Previous activity)
- The URDF contains 6 joints with the following names

```
"joint1", "joint2_1", "joint2_2", "joint2_3", "joint3",
"joint4"
```

- The joints are of the continuous type.
- Please be aware that the JointMessage contains arrays, and they must be the same size as the number of joints to be controlled.

Property of MCR2

Activity 3

1. Create a node named "continuosJoint.py" inside the previously created package "joints_act"

```
touch scripts/continuosJoint.py
```

2. Give execution permission and add it to the CMakeLists.txt.
3. Copy the code in GitHub for this node.
4. Compile the node.
5. Add the node to the launch file and comment the joint_state_publisher.

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

    <node pkg="joints_act" type="continuousJoint.py" name="continuous_joint_pub" />
```

6. Run the launch file

- This function, declares the JointState message and its initial values.
- The message allows the joints to be treated as a list, with its corresponding state values.
    - contJoints.name[0]
        - contJoints.position[0]
        - contJoints.velocity[0]
        - contJoints.effort[0]

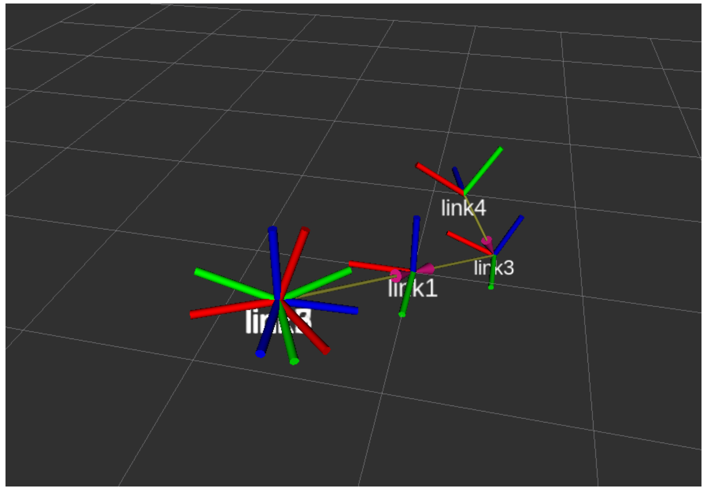
```
from sensor_msgs.msg import JointState

# Declare the output Messages
contJoints = JointState()

# Declare the output Messages
def init_joints():
    contJoints.header.frame_id = "link1"
    contJoints.header.stamp = rospy.Time.now()
    contJoints.name.extend(["joint1", "joint2_1", "joint2_2", "joint2_3", "joint3", "joint4" ])
    contJoints.position.extend([0.0, 0.0, 0.0, 0.0, 0.0, 0.0])
    contJoints.velocity.extend([0.0, 0.0, 0.0, 0.0, 0.0, 0.0])
    contJoints.effort.extend([0.0, 0.0, 0.0, 0.0, 0.0, 0.0])
```

Property of MCR2

Activity 3

- Inside the main loop of the node, the positional values for each joint (6 joints according to the URDF file) are updated using the ROS Time.
- RVIZ only handles positional values, not velocity or effort values, therefore only the position of each joint in the list of joints is being updated.
- Since they are continuous joints (as declared in the previous URDF), the values must be wrapped to PI, before being published.
- The publishing of the message is done as a regular ROS message.

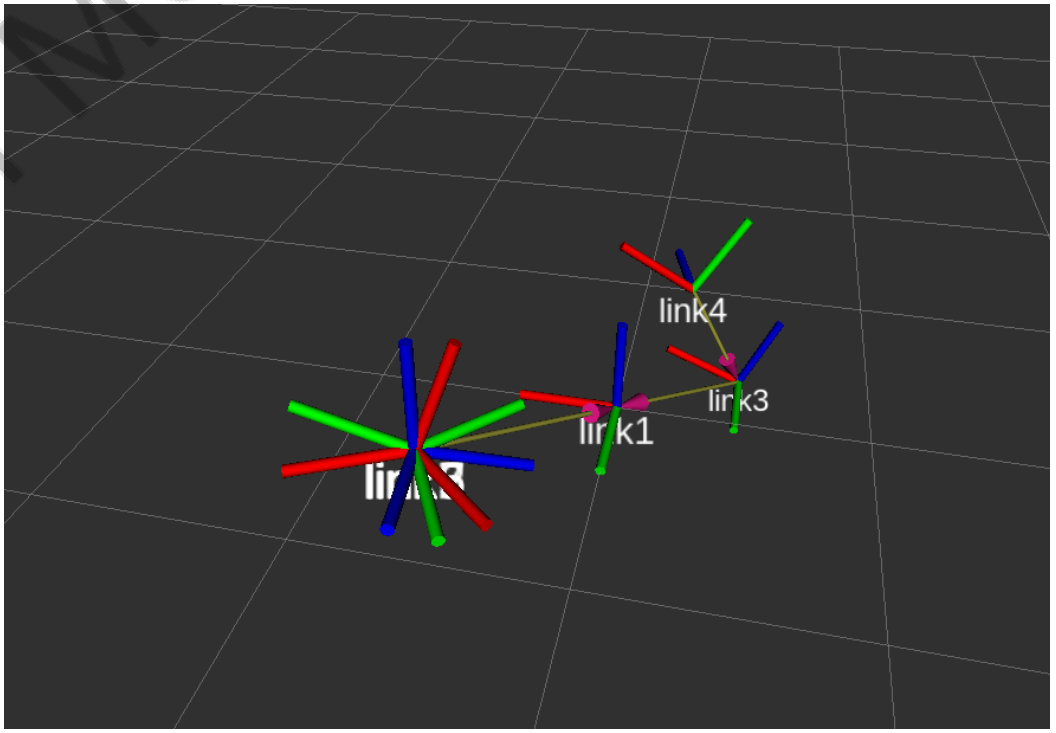
```
while not rospy.is_shutdown():
        t = rospy.Time.now().to_sec()
        contJoints.header.stamp = rospy.Time.now()
        contJoints.position[0] = wrap_to_Pi(t)
        contJoints.position[1] = wrap_to_Pi(0.5*t)
        contJoints.position[2] = wrap_to_Pi(0.5*t)
        contJoints.position[3] = wrap_to_Pi(0.5*t)
        contJoints.position[4] = wrap_to_Pi(0.1*t)
        contJoints.position[5] = wrap_to_Pi(t)

        joint_pub.publish(contJoints)

        loop_rate.sleep()
```



Property of MCR2

Activity 3: Ext

- Repeat the same steps and develop a joint controller for each joint type (each urdf file developed previously).
- Name each node file according to the joint they are controlling.
    - prismatic.py
    - revolute.py



Property of MCR2

URDF

Links

{Learn, Create, Innovate};

Property of MCR2

MCR2 Manchester Robotics

Links

Links description

- As stated previously, links represent a rigid body or a robot component.
- Links describe the visual, inertial, and collision properties of the link.
- Each link may have one or more visual and collision elements associated with it.

```
<link name="wheel_left_link">
       <visual>
           <origin xyz="0 0 0" rpy="0 0 0" />
           <geometry>
               <mesh filename="file://mesh.stl"/>
           </geometry>
           <material name="yellow"/>
       </visual>
       <collision>
           <origin xyz="0 0 0" rpy="0 0 0"/>
           <geometry>
               <cylinder length="1" radius="1" />
           </geometry>
       </collision>
       <inertial>
           <origin xyz="0 0 0" rpy="0 0 0"/>
           <mass value="1"/>
           <inertia ixx="0.4" ixy="0.0" ixz="0.0" iyy="0.4"
           iyz="0.0" izz="0.2"/>
        </inertial>
   </link>
```

Property of MCR2

Links

Visual

- Defines the visual appearance of a link.
- The inbuilt shapes are cylinder, box, and sphere.
- The visual can also set a mesh from a CAD file in format .dae or .stl.
- Origin: Sets the pose of the visual part relative to the link.
- Material: material properties (e.g., color, texture)

```
 <link name="wheel_left_link">
        <visual>
            <origin xyz="0 0 0" rpy="0 0 0" />
            <geometry>
                <mesh filename="file://mesh.stl"/>
            </geometry>
            <material name="yellow"/>
        </visual>
</link>
```

Property of MCR2

Links

Collision

- Defines the collision properties of a link.
- Collision properties can be different from the visual properties of a link (simpler collision models).
- Multiple <collision> tags can exist for the same link.
- The union of the geometry they define forms the collision representation of the link.
- Origin: Sets the pose of the visual part relative to the link frame.
- Geometry: Geometrical description of the collision body The inbuilt shapes are cylinder, box, and sphere.

```
 <link name="wheel_left_link">
       <collision>
            <origin xyz="0 0 0" rpy="0 0 0"/>
            <geometry>
                <cylinder length="1" radius="1" />
            </geometry>
        </collision>
</link>
```

Property of MCR2

Links

Inertial

- Defines the link's mass, the position of its centre of mass, and its central inertia properties.
- Origin: Position and orientation of the link's centre of mass relative to the link frame "L".
- Mass: The mass of the link.
- Inertia: This link's moments of inertia ixx, iyy, izz and products of inertia ixy, ixz, iyz about C (centre of mass) for the unit vectors Ĉx, Ĉy, Ĉz fixed in the centre-of-mass frame C, Relative to L̂x, L̂y, L̂z is specified by the rpy values in the <origin> tag.

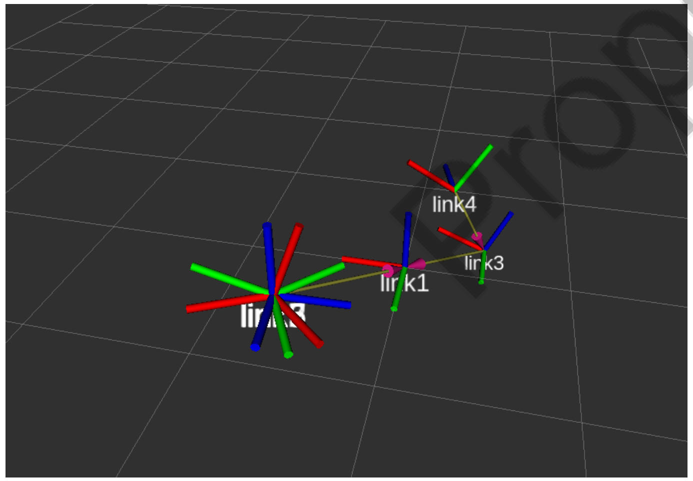
```
 <link name="wheel_left_link">
        <inertial>
           <origin xyz="0 0 0" rpy="0 0 0"/>
           <mass value="1"/>
           <inertia ixx="0.4" ixy="0.0" ixz="0.0" iyy="0.4"
           iyz="0.0" izz="0.2"/>
        </inertial>
</link>
```

Property of MCR2

Links

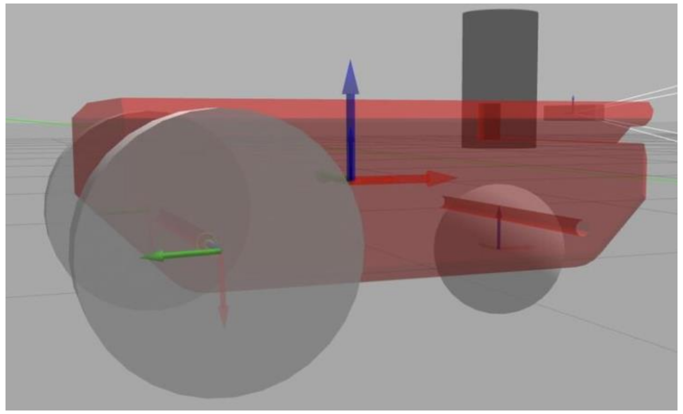
RVIZ

- In RVIZ Links are represented by sets of coordinates.
- RVIZ only requires a name for a link.
- RVIZ only uses the information in the collision and the visual descriptions of the URDF file (both optional).



Gazebo

- The Gazebo simulation environment, requires a plugin to transform the URDF files into SDF files (Gazebo's native file type).
- Gazebo requires information about the visual, inertial, and collision properties, alongside other physics parameters for each link, such as friction, stiffness, and damping.



Property of MCR2

Activity 4

BY^[c Y^ HL ?P

Prope

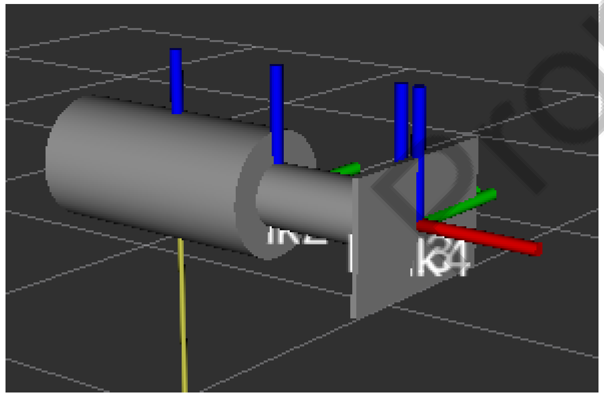
Manchester **Robotics**

{Learn, Create, Innovate};

Activity 4

Description

- In this activity, the user will create a link and a joint state publisher that depicts a pneumatic cylinder moving continuously.
- The position will be controlled by publishing a JointState message into the /joint_states topic.



- Create a package called links_act with the following dependencies

```
nav_msgs    rospy    sensor_msgs   std_msgs   tf2_ros
tf_conversions   visualization_msgs
```

- Create the following folders inside the package
  - Launch, models, scripts, urdf
- Create an urdf file (inside the urdf folder) named "cylinder.urdf"
- Create 5 links, and 4 joints as depicted in the following figure.
- The information about the joints is described on the figure.

Property of MCR2

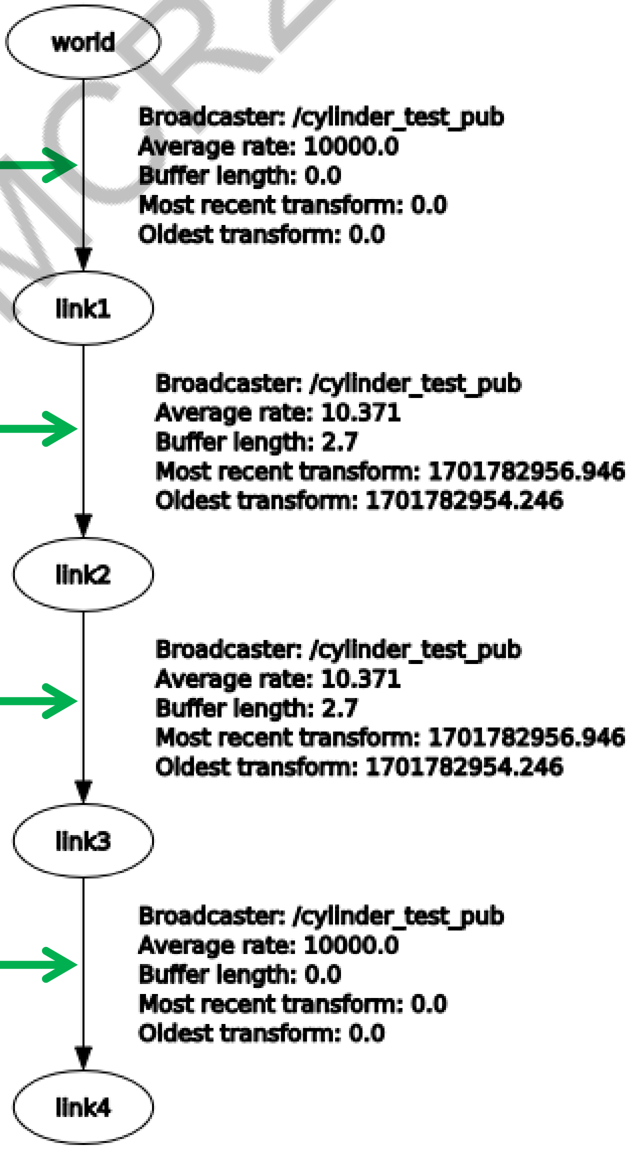
Activity 4

```
<joint name="joint1" type="fixed">
   <origin xyz="0 0 1" rpy="0 0 0" />
   <parent link="world" />
   <child link="link1" />
 </joint>
```

```
<joint name="joint2" type="prismatic">
   <parent link="link1"/>
   <child link="link2"/>
   <origin xyz="0.1 0 0" rpy="0 0 0" />
   <axis xyz="1 0 0" />
   <limit lower="0.0" upper="0.2" effort="5.0" velocity="0.1"/>
 </joint>
```

```
<joint name="joint3" type="revolute">
   <origin xyz="0.225 0 0" rpy="0 0 0" />
   <parent link="link2" />
   <child link="link3" />
   <axis xyz="0 1 0" />
   <limit lower="-0.785" upper="0.785" effort="5" velocity="2" />
 </joint>
```

```
<joint name="joint4" type="fixed">
   <origin xyz="0.03 0 0" rpy="0 0 0" />
   <parent link="link3" />
   <child link="link4" />
 </joint>
```



```
Broadcaster: /cylinder_test_pub
Average rate: 10000.0
Buffer length: 0.0
Most recent transform: 0.0
Oldest transform: 0.0

Broadcaster: /cylinder_test_pub
Average rate: 10.371
Buffer length: 2.7
Most recent transform: 1701782956.946
Oldest transform: 1701782954.246

Broadcaster: /cylinder_test_pub
Average rate: 10.371
Buffer length: 2.7
Most recent transform: 1701782956.946
Oldest transform: 1701782954.246

Broadcaster: /cylinder_test_pub
Average rate: 10000.0
Buffer length: 0.0
Most recent transform: 0.0
Oldest transform: 0.0
```

Property of MCR2

Activity 4

- For each link (Link1 – Link 4) use the following description in the URDF file.
- The link "world" must be empty.

```
<link name="link1">
   <visual>
   <origin xyz="0 0 0" rpy="0 1.57 0"/>
     <geometry>
       <cylinder length="0.4" radius="0.1" />
     </geometry>
     <material name="Grey">
       <color rgba="0.67 0.67 .67 1.0"/>
     </material>
   </visual>
 </link>
```

```
<link name="link2">
   <visual>
   <origin xyz="0 0 0" rpy="0 1.57 0"/>
     <geometry>
       <cylinder length="0.4" radius="0.05" />
     </geometry>
     <material name="Grey">
       <color rgba="0.67 0.67 .67 1.0"/>
     </material>
   </visual>
 </link>
```

```
<link name="link3">
   <visual>
   <origin xyz="0 0 0" rpy="0 1.57 1.57"/>
     <geometry>
       <cylinder length="0.1" radius="0.025" />
     </geometry>
     <material name="Grey">
       <color rgba="0.67 0.67 .67 1.0"/>
     </material>
   </visual>
 </link>
```

```
<link name="link4">
   <visual>
   <origin xyz="0 0 0" rpy="0 0 0"/>
     <geometry>
       <box size="0.01 0.3 0.2" />
     </geometry>
     <material name="Grey">
       <color rgba="0.67 0.67 .67 1.0"/>
     </material>
   </visual>
 </link>
```

Property of MCR2

Activity 4

Testing the model

- Make a lunch file and use the "robot_state_publisher" to publish the information about the state of the cylinder.

```
<arg name="cylinder_test" default="$(find links_act)/urdf/cylinder.urdf"/>
<param name="robot_description" command="cat $(arg cylinder_test)" />
<node pkg="robot_state_publisher" type="robot_state_publisher" name="cylinder_test_pub">
</node>
```

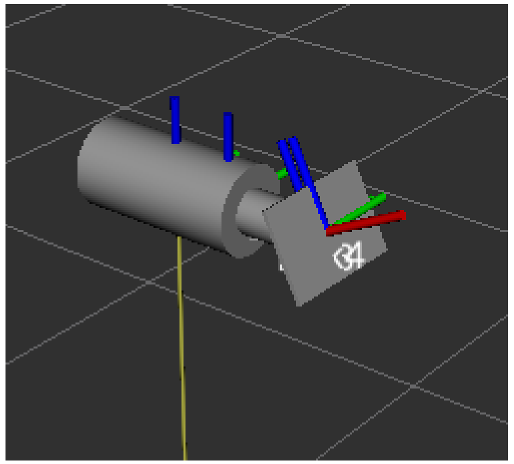
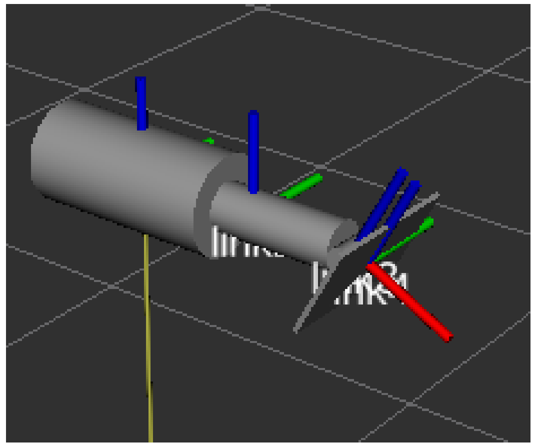
verify if the model is working properly.

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

- To visualise the model in RVIZ press "Add>>By display type>>RobotModel". In the Robot Description add the name of the robot_description parameter (e.g., robot_description).

Finishing the model

- Make a customised joint state controller node called "cylinder_joint.py".
    - Add it to the CMakeLists
- Publish different values to the joints to make the pneumatic piston extend and retract automatically.
    - For this exercise, the joints to be controlled are Joint 2 and Joint 3.

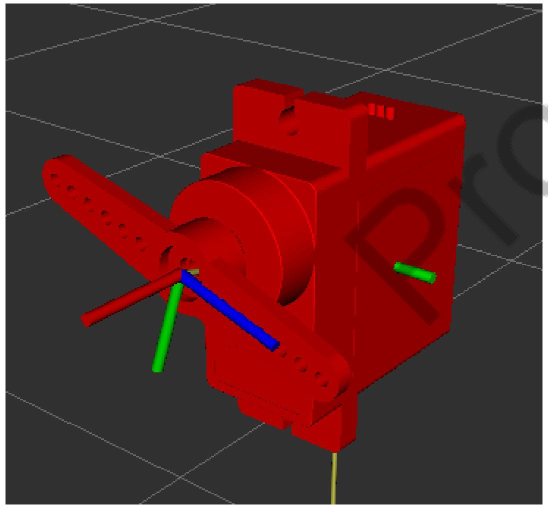



Property of MCR2

Activity 4: Ext

Description

- In this activity, the user will create a link and a joint state publisher that depicts a servo motor moving continuously.
- The position will be controlled by publishing a JointState message into the /joint_states topic.



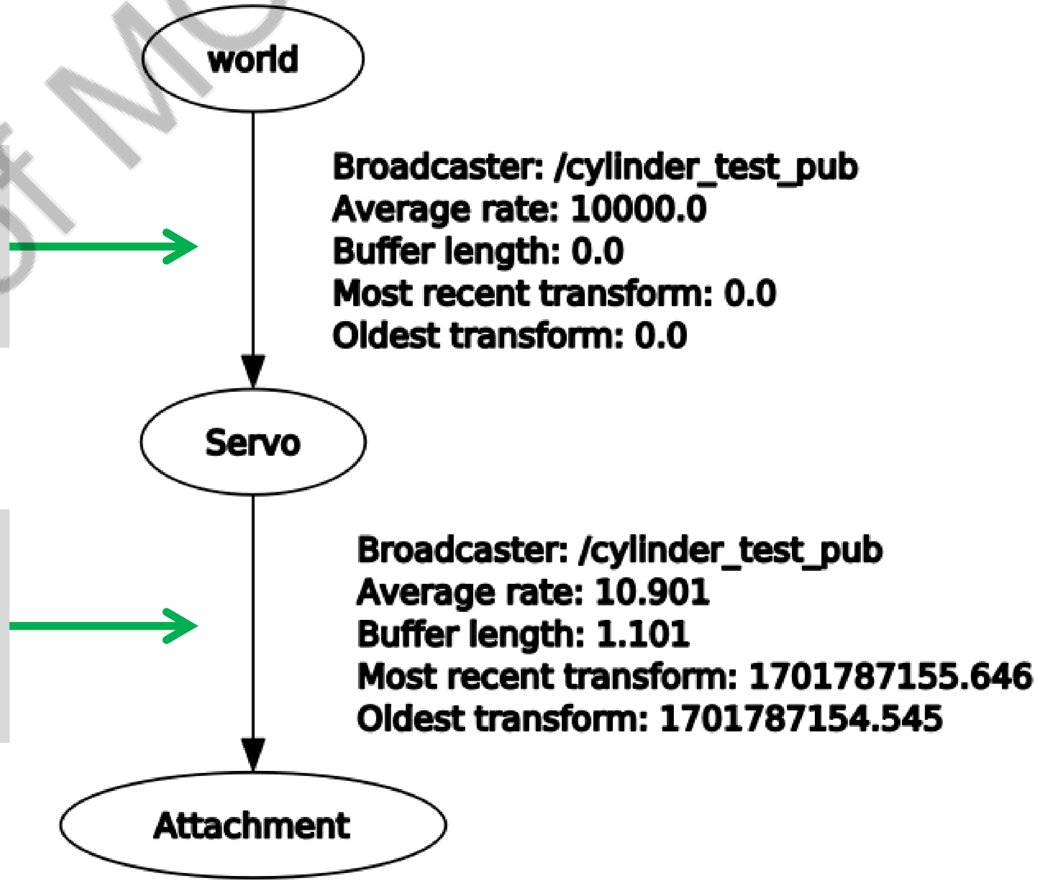
- Use the package previously created in Activity 4.
- Create an urdf file (inside the urdf folder) named "servo.urdf"
- Create 3 links, and 2 joints as depicted in the following figure.
- The information about the joints is described in the figure.

Property of MCR2

Activity 4: Ext

```
<joint name="joint1" type="fixed">
   <origin xyz="0 0 1" rpy="0 0 0" />
   <parent link="world" />
   <child link="Servo" />
 </joint>
```

```
<joint name="joint2" type="revolute">
   <origin xyz="0.0211 0 0.00515" rpy="0 0 0" />
   <parent link="Servo" />
   <child link="Attachment" />
   <axis xyz="1 0 0" />
   <limit lower="-1.57" upper="1.57" effort="5" velocity="2" />
 </joint>
```



```
Broadcaster: /cylinder_test_pub
Average rate: 10000.0
Buffer length: 0.0
Most recent transform: 0.0
Oldest transform: 0.0
```

```
Broadcaster: /cylinder_test_pub
Average rate: 10.901
Buffer length: 1.101
Most recent transform: 1701787155.646
Oldest transform: 1701787154.545
```

Property of MCR2

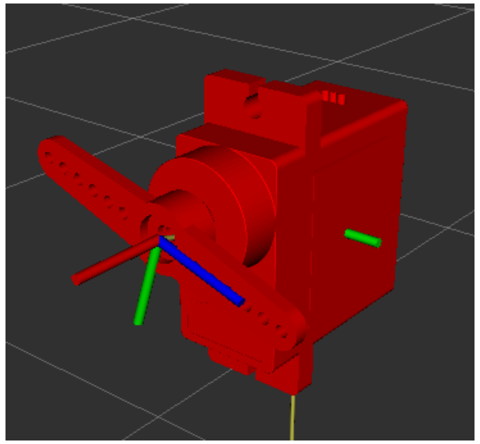
Activity 4: Ext

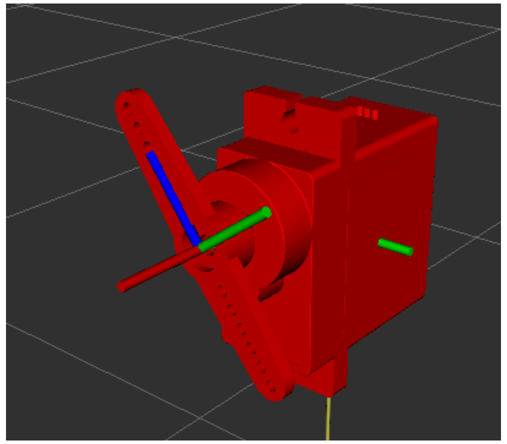
- For each link (Link1 – Link 2) use the following description in the URDF file.
- “world” link must be empty.
- Note that for this exercise the meshes “Micro Servo 9g.stl” and “SG90-Attachment.stl” are used.

```
<link name="Servo">
        <visual>
            <origin rpy="1.57 1.57 0" xyz="0 0 0"/>
            <geometry>
                <mesh filename="package://links_act/models/SG90 -
Micro Servo 9g.stl"/>
            </geometry>
        </visual>
</link>
```

```
<link name="Attachment">
        <visual>
            <origin rpy="1.57 1.57 0" xyz="0 0 0"/>
            <geometry>
                <mesh filename="package://links_act/models/SG90-
Attachment.stl"/>
            </geometry>
        </visual>
</link>
```

- Test your model using the joint_control_gui as performed in the previous activity.
- Make a customised joint state controller node called “servo_joint.py”.
    - Add it to the CMakeLists
- Publish different values to the joints to make the servo motor move automatically.





Property of MCR2

Thank you

{Learn, Create, Innovate};

Property of MCR2

T&C

Terms and conditions

`{Learn, Create, Innovate};`

Manchester Robotics

Property of MCR2

Terms and conditions

- THE PIECES, IMAGES, VIDEOS, DOCUMENTATION, ETC. SHOWN HERE ARE FOR INFORMATIVE PURPOSES ONLY. THE DESIGN IS PROPRIETARY AND CONFIDENTIAL TO MANCHESTER ROBOTICS LTD. (MCR2). THE INFORMATION, CODE, SIMULATORS, DRAWINGS, VIDEOS PRESENTATIONS ETC. CONTAINED IN THIS PRESENTATION IS THE SOLE PROPERTY OF MANCHESTER ROBOTICS LTD. ANY REPRODUCTION, RESELL, REDISTRIBUTION OR USAGE IN PART OR AS A WHOLE WITHOUT THE WRITTEN PERMISSION OF MANCHESTER ROBOTICS LTD. IS STRICTLY PROHIBITED.
- THIS PRESENTATION MAY CONTAIN LINKS TO OTHER WEBSITES OR CONTENT BELONGING TO OR ORIGINATING FROM THIRD PARTIES OR LINKS TO WEBSITES AND FEATURES IN BANNERS OR OTHER ADVERTISING. SUCH EXTERNAL LINKS ARE NOT INVESTIGATED, MONITORED, OR CHECKED FOR ACCURACY, ADEQUACY, VALIDITY, RELIABILITY, AVAILABILITY OR COMPLETENESS BY US.
- WE DO NOT WARRANT, ENDORSE, GUARANTEE, OR ASSUME RESPONSIBILITY FOR THE ACCURACY OR RELIABILITY OF ANY INFORMATION OFFERED BY THIRD-PARTY WEBSITES LINKED THROUGH THE SITE OR ANY WEBSITE OR FEATURE LINKED IN ANY BANNER OR OTHER ADVERTISING.

Property of MCR2