Worksheet 3 – Introduction to transform libraries and robot model

1. Introduction

The ROS framework has a package to manage the relationships between coordinate frames. This allows the transformation of points, and vectors, between any two coordinate frames at any point in time.

http://wiki.ros.org/tf

In this worksheet we will create the description of the TR5 robotic arm depicted in Figure 1.

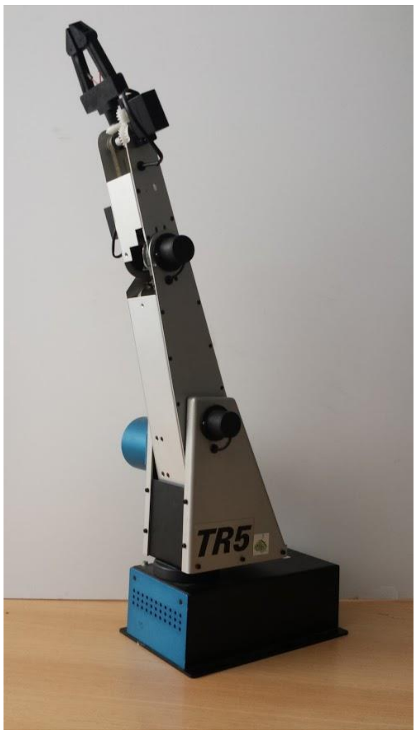


Figure 1 – TR5.

First, you will do a couple of exercises to learn how to define the relationships between the different coordinate frames and share them on the ROS framework. The initial objective is to define them using the ROS TF C++ implementation. Then you will create the robot's definition using a XML based robot description.

Note: To understand the conventions defined by ROS you should read the following page:

http://www.ros.org/reps/rep-0103.html

2. Transform libraries

Part one

Start by creating a ROS package with the needed libraries and packages you need (*roscpp, std_msgs, tf*) (Note: If you don't remember how go back to the previous worksheets).

The first objective is to create a *TF* broadcaster that will publish to the ROS network the transformation between two coordinate frames. Let's imagine that we have an UAV that has a laser sensor that gives us the exact altitude in a topic called */uav/altitude*.

We are going to publish the transform between *base_footprint* that represents the ground level and the *base_link* that geometric centre of the UAV.

So, now you should create a node that receives this information and publishes a TF from the *base_footprint* frame to the *base_link* frame (where all the sensors are attached).

The goal of this first exercise is to subscribe the */uav/altitude* topic of *std_msgs/Float32* type and publish the transform *base_footprint* -> *base_link*. With your new found ROS proficiency you should now be able to perform this task using only the information on the following tutorial. (Note: the tutorial does almost the exact same task and need only a minor adaptation)

http://wiki.ros.org/tf/Tutorials/Writing%20a%20tf%20broadcaster%20%28C%2B%2B%29

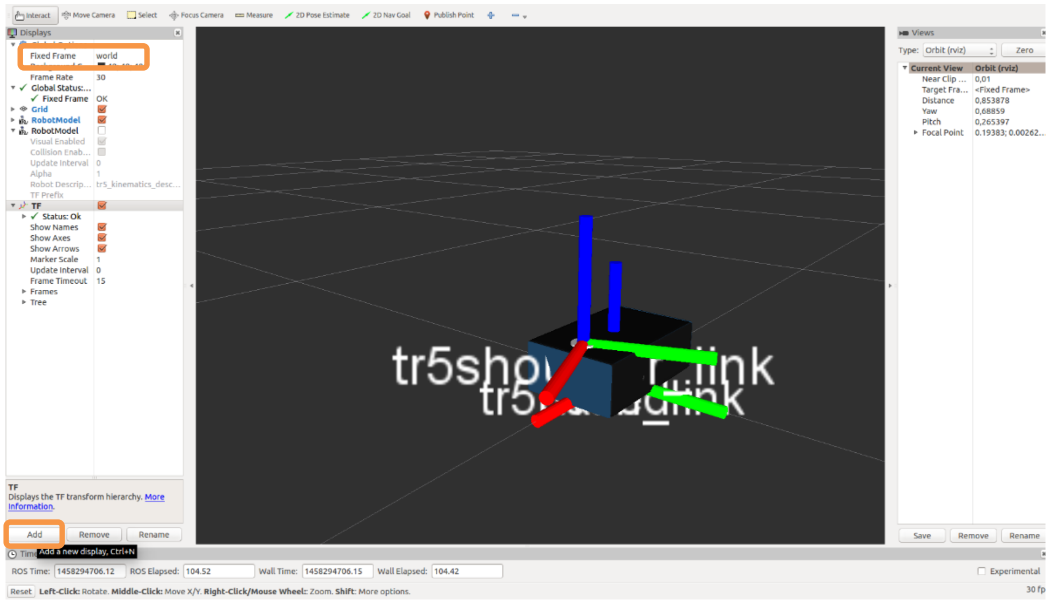
After finishing your code, you need to test it. To simulate the altitude sensor output, open a terminal and publish an altitude value using the aforementioned topic.

```
$ rostopic pub -r 10 /uav/altitude std_msgs/Float32 5
```

Next open the ROS visualization tool rviz (see Figure 2, Figure 3).

```
$ rviz
```

Then you should subscribe to the TF type remembering that the Fixed Frame should be base_footprint. You can also use the different tools that the TF package has to verify if your transformation is being published correctly.



Displays

Fixed Frame	world
Frame Rate	30
✓ Global Status:...	
✓ Fixed Frame	OK
Grid	✓
RobotModel	✓
RobotModel	☐
Visual Enabled	✓
Collision Enab...	☐
Update Interval	0
Alpha	1
Robot Descrip...	tr5_kinematics_desc...
TF Prefix	
TF	✓
✓ Status: Ok	
Show Names	✓
Show Axes	✓
Show Arrows	✓
Marker Scale	1
Update Interval	0
Frame Timeout	15
Frames	
Tree	

TF
Displays the TF transform hierarchy. More Information.

Add | Remove | Rename

Add a new display, Ctrl+N

Views

Type: Orbit (rviz) Zero

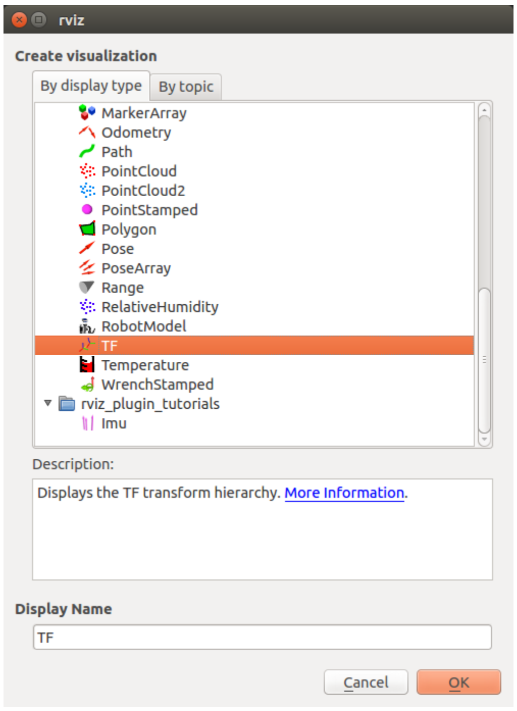
Current View	Orbit (rviz)
Near Clip ...	0,01
Target Fra...	<Fixed Frame>
Distance	0,853878
Yaw	0,68859
Pitch	0,265397
Focal Point	0.19383; 0.00262...

Save | Remove | Rename

ROS Time: 1458294706.12 ROS Elapsed: 104.52 Wall Time: 1458294706.15 Wall Elapsed: 104.42 ☐ Experimental

Reset **Left-Click:** Rotate. **Middle-Click:** Move X/Y. **Right-Click/Mouse Wheel:**: Zoom. **Shift**: More options. 30 fp

Figure 2 - RVIZ visualization tool


rviz

Create visualization

By display type | By topic

- MarkerArray
- Odometry
- Path
- PointCloud
- PointCloud2
- PointStamped
- Polygon
- Pose
- PoseArray
- Range
- RelativeHumidity
- RobotModel
- TF
- Temperature
- WrenchStamped
- rviz_plugin_tutorials
  - Imu

Description:

Displays the TF transform hierarchy. More Information.

Display Name

TF

Cancel | OK

Figure 3 - Add a TF display

Part Two

The next step is to create a subscriber of a TF and print its value to the console. Again there is a very similar tutorial that you should base your implementation on. The transform you are listening now should be *map*->*base_footprint*.

http://wiki.ros.org/tf/Tutorials/Writing%20a%20tf%20listener%20%28C%2B%2B%29

Now to verify that your node is functioning correctly you can use a TF tool.

http://wiki.ros.org/tf#static_transform_publisher

Now open a terminal and write the following on the console:

```
$ rosrun tf static_transform_publisher 10.0 0.0 0.0 0.0 0.0 0.0 map base_footprint 100
```

What you are doing is to publish a TF (transformation) between two frames using the following format:

```
$ rosrun tf static_transform_publisher x y z roll pitch yaw frame1 frame2 period_in_ms
```

Now, the final step is to add another frame. Create a function that receives a frame name in a topic /new_frame (*std_msgs/String*) and publishes a transform between *base_footprint* and this new frame with following values (x: 1, y: 1, z: 1, roll: 0, pitch: 0, yaw: 0). This TF should be published at the same rate as the message received. To test it use the rostopic pub function in the console (Note: If you don't remember how go back to previous worksheets).

```
void TfExample::addFrame(std::string name)
```

3. Introduction to URDF – Universal Robotic Description Format

The TF package simplifies the creation of transformations between coordinate frames. However, as you could ascertain during the previous exercise creating a full description of all the transformations in a Robot would be cumbersome and time consuming.

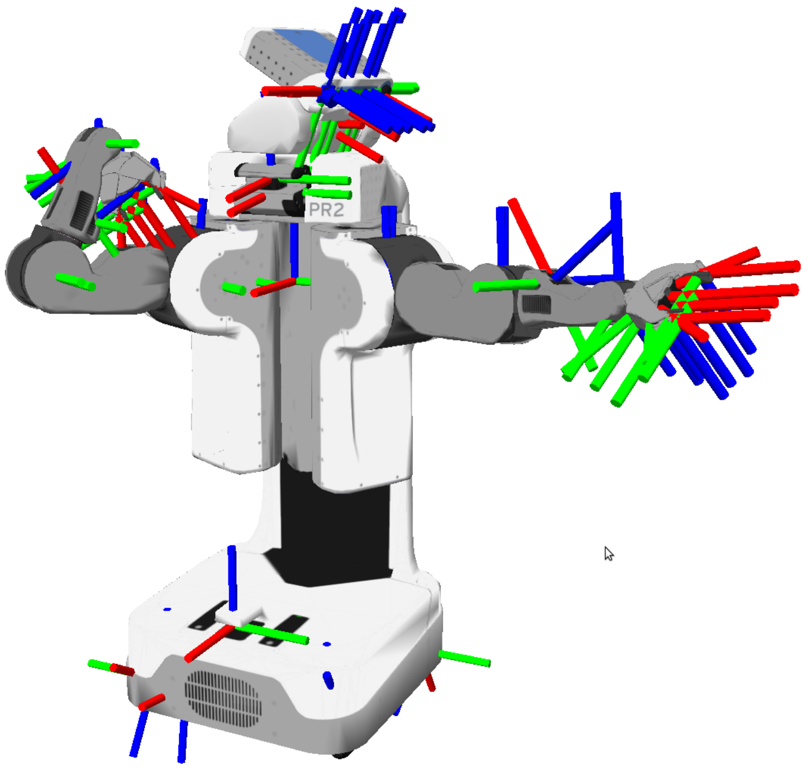


Figure 4 - PR2 TF example

Consequently, ROS also addresses this problem by using a Robot description format (URDF). In the next pages a brief description of the concepts of URDF are depicted. However, you should also see a more complete introduction to URDF on the following link.

wiki.ros.org/urdf

And also the specifications of the XML format of URDF

http://wiki.ros.org/urdf/XML

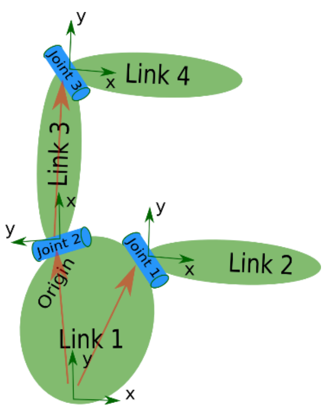


Figure 5- Links and Joints relationship

An URDF description of a robot consists of a set of link elements, and a set of joint elements connecting the links together (see Figure 5). So a typical URDF robot description looks something like this:

```
<robot name="robot">
  <link> ... </link>
  <link> ... </link>
  <link> ... </link>

  <joint>  ....  </joint>
  <joint>  ....  </joint>
  <joint>  ....  </joint>
</robot>
```

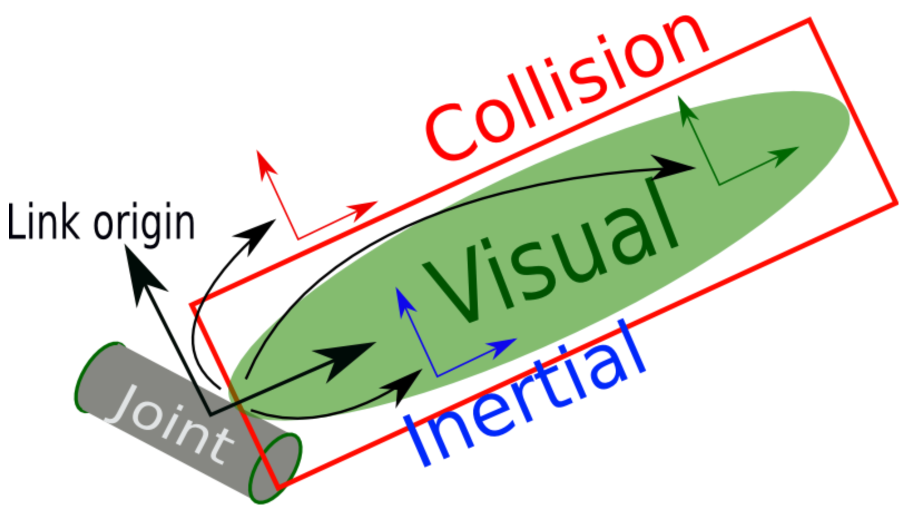


Figure 6- Link definition

A link is what connects two joints it defines the properties of that part of the robot including the visual, collision, inertial characteristics. A typical link definition is something like this:

```
<link name="torso">
    <visual>
      <origin rpy="0 0 0" xyz="0 0 0"/>
      <geometry>
        <mesh filename="package://robot_description/meshes/base_link.DAE"/>
      </geometry>
    </visual>
<link name="torso">
    <visual>
      <origin rpy="0 0 0" xyz="0 0 0"/>
      <geometry>
        <mesh filename="package://robot_description/meshes/base_link.DAE"/>
      </geometry>
    </visual>
    <collision>
      <origin rpy="0 0 0" xyz="-0.065 0 0.0"/>
      <geometry>
        <mesh filename="package://robot_description/meshes/base_link.DAE"/>
      </geometry>
    </collision>
    <collision_checking>
      <origin rpy="0 0 0" xyz="-0.065 0 0.0"/>
      <geometry>
        <cylinder length="0.7" radius="0.27"/>
      </geometry>
    </collision_checking>
    <inertial>
      ...
    </inertial>
  </link>
```

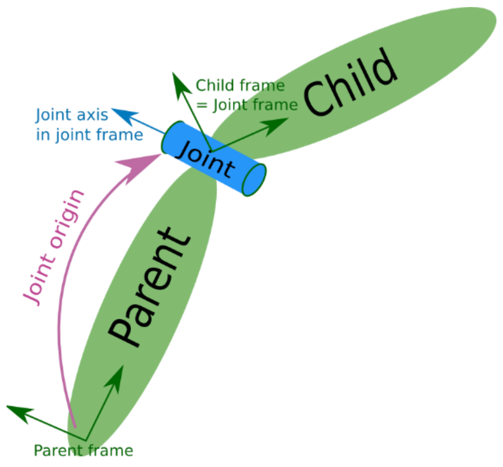


Figure 7- Joint definition

A joint can be of three different types:

- revolute - a hinge joint that rotates along the axis and has a limited range specified by the upper and lower limits.
- continuous - a continuous hinge joint that rotates around the axis and has no upper and lower limits
- prismatic - a sliding joint that slides along the axis, and has a limited range specified by the upper and lower limits.
- fixed - This is not really a joint because it cannot move. All degrees of freedom are locked. This type of joint does not require the axis, calibration, dynamics, limits or safety_controller.
- floating - This joint allows motion for all 6 degrees of freedom.
- planar - This joint allows motion in a plane perpendicular to the axis.

The first step is to understand the convention that ROS normally uses when storing a Robot Model. Usually there is a package with the following name ***robotname_description*** where all the files that contain the description are located. These include the description of all the parts of the robot including actuators, sensors, mechanical components etc. The normal organization of a robot description package is the following:

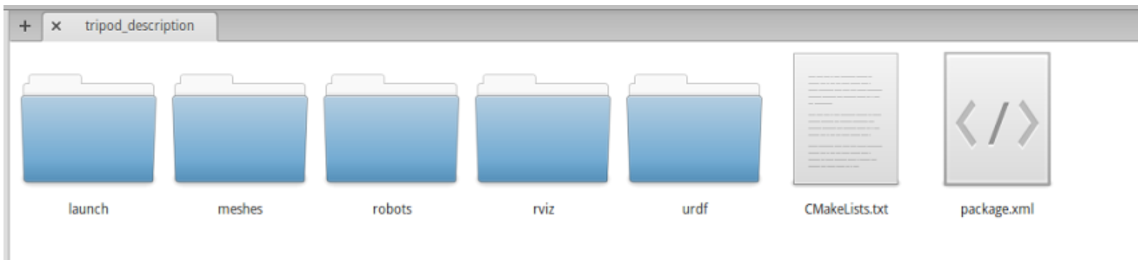


Figure 8- Package example

Each folder exists for a specific reason:

launch: You've probably already created launch folders in other worksheets. This folder is where the .launch files will be placed;

meshes: Where the meshes (3D models) of the robot should be placed;

robots: The master files that include the more detailed description of each file;

rviz: This folder is optional but if you create any RVIZ profile you should place it here;

urdf: Where the detailed description of each part of the robot should be placed;

After this brief overview, let's go into a more detailed explanation of each one and their dependencies.

Figure 9 – Launch folder

In the launch folder there is a normally a master launch file, i.e. robot_name.launch that launches the robot's description so it's available in the ROS framework.

```
<launch>
        <!-- send robot urdf to param server -->
        <arg    name="urdf_file"    default="$(find    xacro)/xacro.py    '$(find    package_name
        )/robots/tripod.urdf.xacro'" />
        <param name="robot_description" command="$(arg urdf_file)" />

        <!-- Use the robot state publisher to send the tf -->
        <node            pkg="robot_state_publisher"            type="robot_state_publisher"
        name="robot_state_publisher" output="screen">
                <param name="publish_frequency" type="double" value="5.0" />
        </node>

        <!-- Only /use_gui = true if there is no other source of joint position information-->
        <node            name="joint_state_publisher"            pkg="joint_state_publisher"
        type="joint_state_publisher">
                <param name="/use_gui" value="true"/>
        </node>

</launch>
```

Note: All the steps where in following code *$(find package_name)* appears, is to find the package if you have a different name you should change it accordingly.

The first part of the code sends to the param server the description defined in the URDF file. The next step is to use the ***robot_state_publisher*** (http://wiki.ros.org/robot_state_publisher) that based on that description publishes the transforms (TF) of joints of the **fixed** type. The final step is to publish the moveable joints in a topic of the (***sensor_msgs/JointState***) type. So we will exploit the ***joint_state_publisher*** (http://wiki.ros.org/joint_state_publisher) node. That will subsequently be used by the ***robot_state_publisher*** to send the transform to the ROS network.

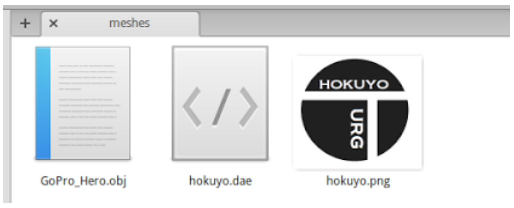


Figure 10- Meshes folder

In the **meshes** folder you should place all the 3D models of the components of the robots. Here you can also create additional folders for the sensors, actuators, etc.

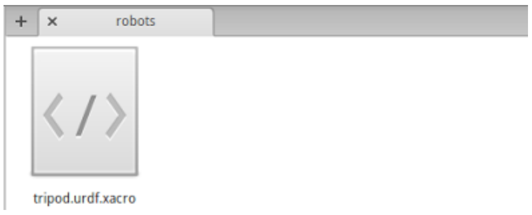


Figure 11 – Robots folder

In the **robots** folder you should have only the "master" file that calls all the other files that compose your robot. In this case we only have the *tripod_class.urdf.xacro*. But we could have another version of the tripod that included a Kinect and so we could have a *tripod_kinect.urdf.xacro* file.

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

<!--
- Base : Tripod
- laser : tilting hokuyo
- cameras          : gopro/flir/usb
-->
<robot name="tripod" xmlns:xacro="http://ros.org/wiki/xacro">
        <xacro:include filename="$(find lesson7)/urdf/tripod_library_class.urdf.xacro" />
</robot>
```

In this case we only call the main URDF file.

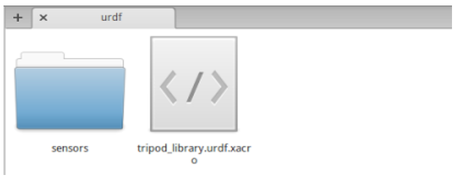


Figure 12 – The URDF folder

Here is where the detailed description of the robot is stored. We have a main xacro.urdf file that subsequently includes all the components of the robot. For example the descriptions of the sensors that are inside the folder named sensors.

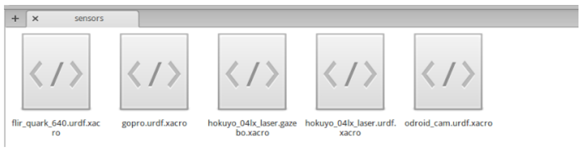

sensors
flir_quark_640.urdf.xacro
gopro.urdf.xacro
hokuyo_04lx_laser.gazebo.xacro
hokuyo_04lx_laser.urdf.xacro
odroid_cam.urdf.xacro

Figure 13 - Sensors folder

So let's remember the basic components of a URDF description a give an example of a robot description. A link is a coordinate frame that has a visual, collision, and physical properties. Where a joint defines the connection between two links (a transformation or TF). Joints can be fixed or movable (continuous, revolute and prismatic) (Note: For more information go to the URDF tutorial on ROS wiki)

Link example

```
<link name="name">
        <!— Visual definition  -->
        <visual>
                <origin xyz="0 0 0" rpy="0 0 0" />
                <geometry>
                        <box size="0 0 0" />
                </geometry>
        </visual>
        <!— Collision definition  -->
        <collision>
                <origin xyz="0 0 0" rpy="0 0 0" />
                <geometry>
                        <box size="0 0 0" />
                </geometry>
        </collision>
        <!— Inertial characteristics  -->
        <inertial>
                <mass value="0.001" />
                <origin xyz="0 0 0" rpy="0 0 0" />
                <inertia ixx="0.0001" ixy="0" ixz="0" iyy="0.000001" iyz="0" izz="0.0001" />
        </inertial>
</link>
```

Joint example

```
<joint name="joint_name" type="fixed">
        <origin xyz="0 0 0" rpy="0 0 0"/>
        <parent link="frame1"/>
        <child link="frame2"/>
</joint>
```

However, there are some shortcuts you can use XACRO that allows you to create macros of XML. The following example is a sensor in this case of a USB camera.

```xml
<?xml version="1.0"?>
<root 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">
<property name="M_PI" value="3.1415926535897931" />
<property name="camera_length" value="0.013" />
<property name="camera_width" value="0.057" />
<property name="camera_height" value="0.029" />
<!-- This is commented below and its assumed that the lens is centered -->
<property name="lens_x" value="0.0" />
<property name="lens_y" value="0.0" />
<property name="lens_z" value="0.0" />
<property name="M_PI" value="3.1415926535897931" />
<xacro:macro  name="odroid_cam"  params="name  parent  cam_length  cam_width  cam_height
*origin">
<joint name="${name}_joint" type="fixed">
        <axis xyz="0 1 0" />
        <insert_block name="origin" />
        <parent link="${parent}"/>
        <child link="${name}_link"/>
</joint>
<link name="${name}_link">
        <inertial>
                <mass value="0.001" />
                <origin xyz="0 0 0" rpy="0 0 0" />
                <inertia ixx="0.0001" ixy="0" ixz="0" iyy="0.000001" iyz="0" izz="0.0001" />
        </inertial>
        <visual>
                <origin xyz="0 0 ${cam_height/2}" rpy="0 0 0" />
                <geometry>
                        <box size="${cam_length} ${cam_width} ${cam_height}" />
                </geometry>
        </visual>
        <collision>
                <origin xyz="0 0 ${cam_height/2}" rpy="0 0 0" />
                <geometry>
                        <box size="${cam_length} ${cam_width} ${cam_height}" />
                </geometry>
        </collision>
</link>
<!-- go from the base of the sensor to the actual sensor location -->
<joint name="${name}_lens_joint" type="fixed">
        <origin xyz="${cam_length/2} 0.0 ${cam_height/2}" rpy="-${M_PI/2} 0 -${M_PI/2}"/>
        <!-- origin xyz="${lens_x} ${lens_y} ${lens_z}" rpy="-${M_PI/2} 0 -${M_PI/2}"/-->
        <parent link="${name}_link"/>
        <child link="${name}_lens_link"/>
</joint>
```

The basic commands you need to know about XACRO during this worksheet are:

XACRO declaration, has a name and parameters here is an example definition:

```
<xacro:macro name="sensor_xacro" params="name param1 param2 param3 *origin">
```

To define a property you need to create the following line:

```
<property name="property_name" value="X:X" />
```

Then to use the value of a parameter or property value you have to call it inside ${}

```
 <box size="${property_name} ${cam_width} ${cam_height}" />
```

It is also possible to do mathematical operations for instance you want half of the height of the camera, you should do something like this ${cam_height/2} where cam_height is previously defined.

For more information you should check the following page:

http://wiki.ros.org/xacro

Also you should start to learn to use the tutorials in ROS wiki. This will allow you in the future to test different packages that you probably need in the final lab work of this module.

4. Implementation:

The main objective is to create a description of the TR5 robotics arm.

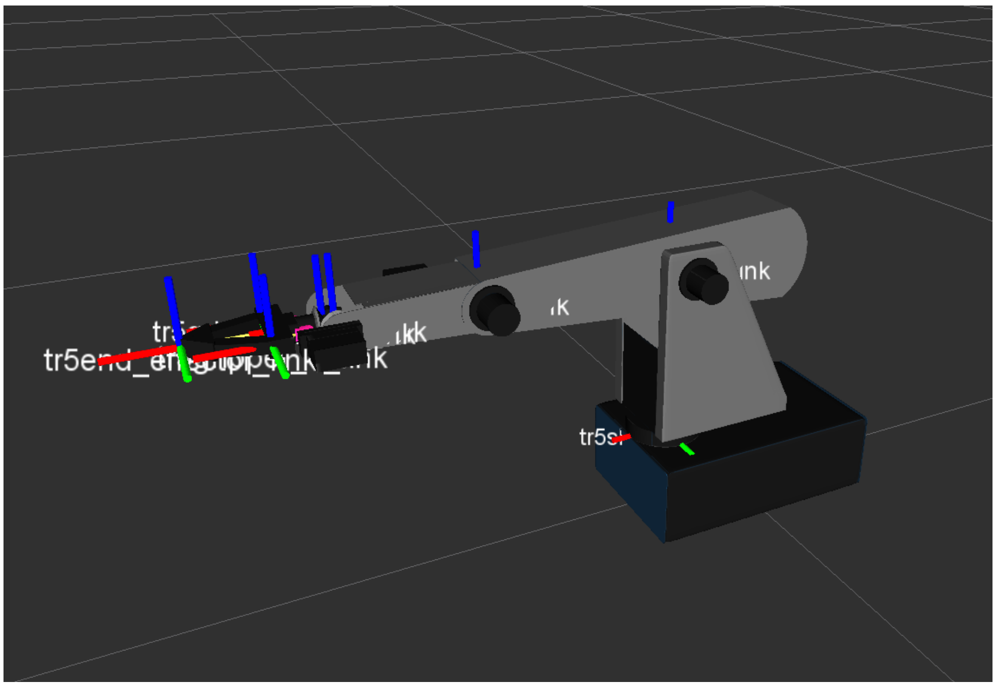
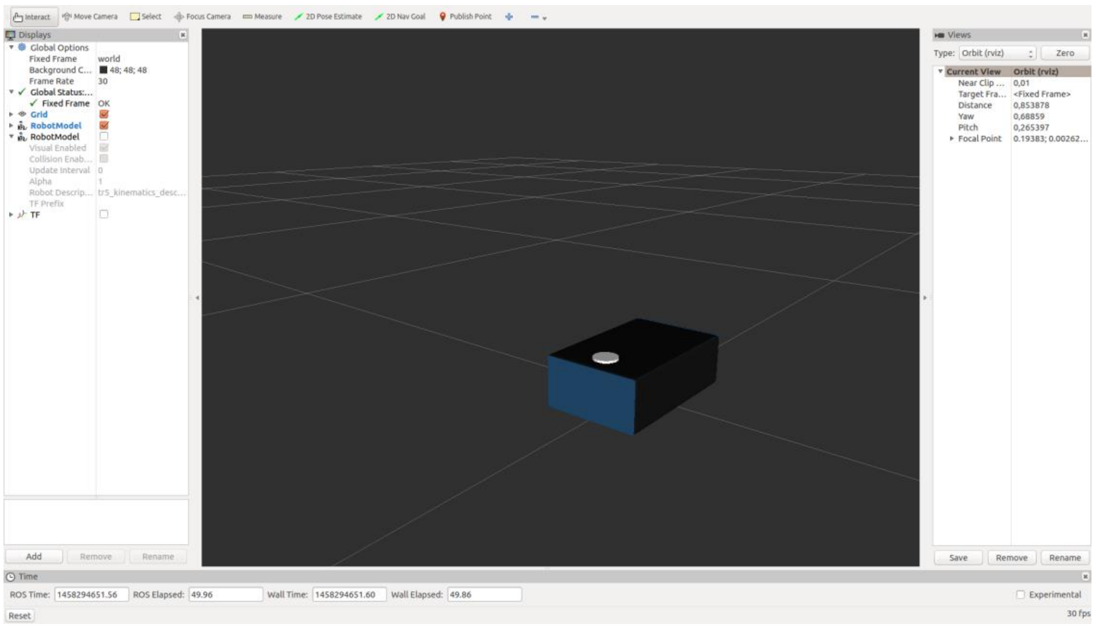


Figure 14 - Robot model depicted in RVIZ. The X, Y and Z axis are represented by red, green and blue respectively

Steps:

- First, you should do the tutorial on how to model the R2D2 robot using URDF http://goo.gl/i6kQDZ
- Then, you will need to edit and fill out the *tr5_basic.urdf.xacro*.
- Download the source code already has the launch files and folder organization https://goo.gl/uKRBE6.
    - You have to add each link and joint of the TR5
    - The joint and link names, ranges and velocities are available in ANNEX – TR5 Datasheet
- In the end of the worksheet you must be able to see the finalized robot_model in RVIZ with all the correct distances between each frame (see Figure 14).

In the source code provided the overall organization and launch files are already available. First, you should launch the *tr5_basic_test.launch* file. Then, run RVIZ and select the File->Open_config option, then select the *TR5.rviz* file inside the *rviz* folder of the provided package. You should then have the same view depicted in Figure 15



Displays

Global Options	
Fixed Frame	world
Background C...	■ 48; 48; 48
Frame Rate	30
✓ Global Status:...	
✓ Fixed Frame	OK
Grid	✓
RobotModel	✓
RobotModel	☐
Visual Enabled	✓
Collision Enab...	☐
Update Interval	0
Alpha	1
Robot Descrip...	tr5_kinematics_desc...
TF Prefix	
TF	☐

Views

Type: Orbit (rviz)

Current View	Orbit (rviz)
Near Clip ...	0,01
Target Fra...	<Fixed Frame>
Distance	0,853878
Yaw	0,68859
Pitch	0,265397
Focal Point	0.19383; 0.00262...

Time

ROS Time: 1458294651.56 ROS Elapsed: 49.96 Wall Time: 1458294651.60 Wall Elapsed: 49.86 ☐ Experimental

30 fps

Figure 15 - RVIZ of the description provided

Inside the ***tr5_robot_basic.urdf.xacro*** file there is already an example of two links the joint between them. Please follow this template for each new link/joint that you create.

```
    <!-- Base %%%%%%%%%%%%%%%%%% -->

    <link name="${prefix}base_link" >
      <visual>
        <geometry>
          <mesh filename="package://tr5_description/meshes/TR5_Base.dae" />
        </geometry>
        <origin xyz="0.0 0.0 0.0" rpy="0.0 0.0 0.0"/>
        <material name="LightGrey">
          <color rgba="0.7 0.7 0.7 1.0"/>
        </material>
      </visual>
      <collision>
        <geometry>
          <box size="0.250 .160 ${shoulder_height}"/>
        </geometry>
        <origin xyz="0.0 0.0 ${shoulder_height/2}" rpy="0.0 0.0 0.0"/>
      </collision>
      <xacro:cylinder_inertial radius="0.06" length="0.05" mass="${base_mass}">
        <origin xyz="0.0 0.0 0.0" rpy="0 0 0" />
      </xacro:cylinder_inertial>
    </link>

    <joint name="${prefix}shoulder_pan_joint" type="revolute">
      <parent link="${prefix}base_link" />
      <child link = "${prefix}shoulder_link" />
      <origin xyz="${shoulder_off_x} 0.0 ${base_height}" rpy="0.0 0.0 0.0" />
      <axis xyz="0 0 1" />
      <xacro:unless value="${joint_limited}">
        <limit lower="-1.39626" upper="1.39626" effort="150.0" velocity="0.802851"/>
      </xacro:unless>
      <xacro:if value="${joint_limited}">
        <limit lower="-1.39626" upper="1.39626" effort="150.0" velocity="0.802851"/>
      </xacro:if>
      <dynamics damping="0.0" friction="0.0"/>
    </joint>~
l
    <!-- Shoulder %%%%%%%%%%%% -->
    <link name="${prefix}shoulder_link">
      <visual>
        <geometry>
          <cylinder length="${shoulder_length/2}" radius="0.02"/>
        </geometry>
        <origin xyz="0.0 0.0 0.0" rpy="0.0 0.0 0.0"/>
        <material name="LightGrey">
          <color rgba="0.7 0.7 0.7 1.0"/>
        </material>
      </visual>
      <collision>
        <geometry>
          <cylinder length="${shoulder_length/2}" radius="0.02"/>
        </geometry>
        <origin xyz="0.0 0.0 ${shoulder_length/2}" rpy="0.0 0.0 0.0"/>
      </collision>
      <xacro:cylinder_inertial radius="0.06" length="0.15" mass="${shoulder_mass}">
        <origin xyz="0.0 0.0 0.0" rpy="0 0 0" />
      </xacro:cylinder_inertial>
```

Figure 16 – The two links and one joint in the provided file

5. ANNEX – TR5 Datasheet

Table 1 - Techincal Specifications of TR5

Technical Specifications	
Kinematics	5-axis link arm robot
Drive systems	DC Servo-motors
Position detection	absolute, potentiometric
Working ranges	
Axis 1 ***(tr5_shoulder_pan_joint)***	160°
Axis 2 ***(tr5_shoulder_lift_joint)***	100º
Axis 3 ***(tr5_elbow_joint)***	100º
Axis 4 ***(tr5_wrist_1_joint)***	200º
Axis 5 ***(tr5_wrist_2_joint)***	200º
Gripper ***(tr5_gripper_1_joint,tr5_gripper_2_joint)***	27º
Resolution	
Axis 1 ***(tr5_shoulder_pan_joint)***	256 steps
Axis 2 ***(tr5_shoulder_lift_joint)***	256 steps
Axis 3 ***(tr5_elbow_joint)***	256 steps
Axis 4 ***(tr5_wrist_1_joint)***	256 steps
Axis 5 ***(tr5_wrist_2_joint)***	256 steps
Gripper ***(tr5_gripper_1_joint,tr5_gripper_2_joint)***	256 steps
Max Joint Speeds	
Axis 1 ***(tr5_shoulder_pan_joint)***	46 degrees/sec.
Axis 2 ***(tr5_shoulder_lift_joint)***	40 degrees/sec.
Axis 3 ***(tr5_elbow_joint)***	100 degrees/sec.
Axis 4 ***(tr5_wrist_1_joint)***	174 degrees/sec.
Axis 5 ***(tr5_wrist_2_joint)***	176 degrees/sec
Max Payload	6Kg

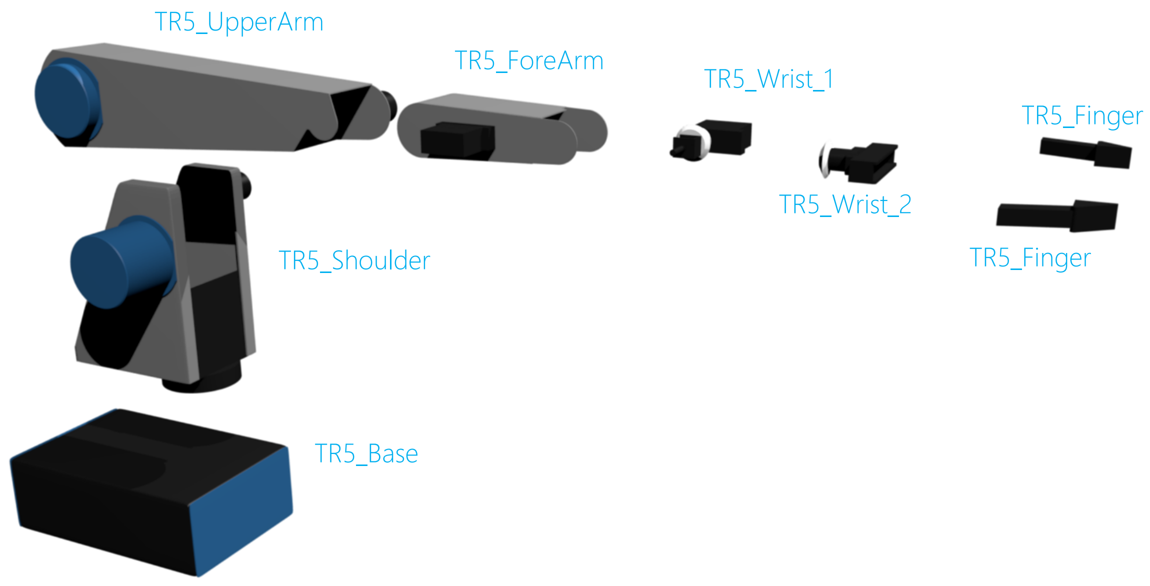


Figure 17- Exploded view of TR5 with the name of each mesh

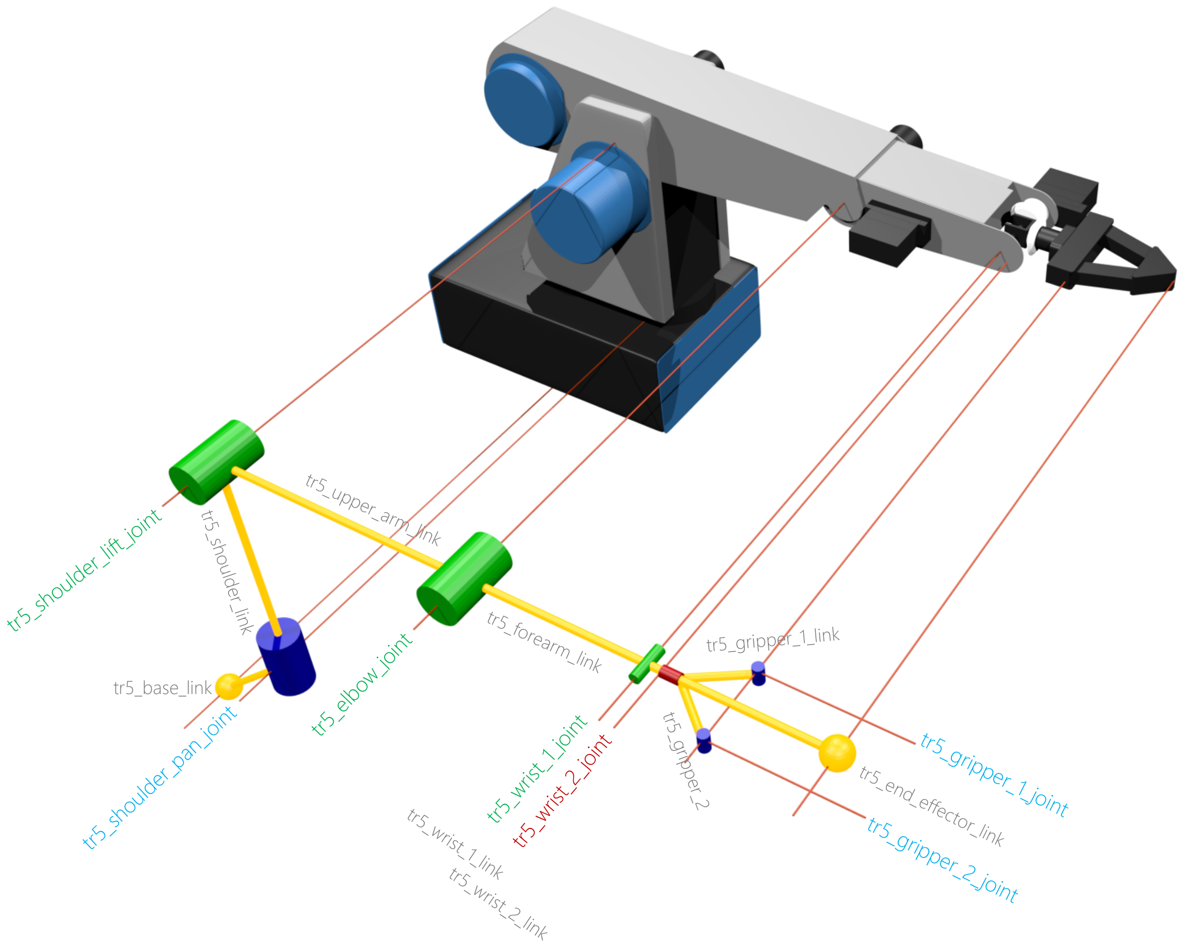


Figure 18- TR5 Skeleton with joints and the links between them

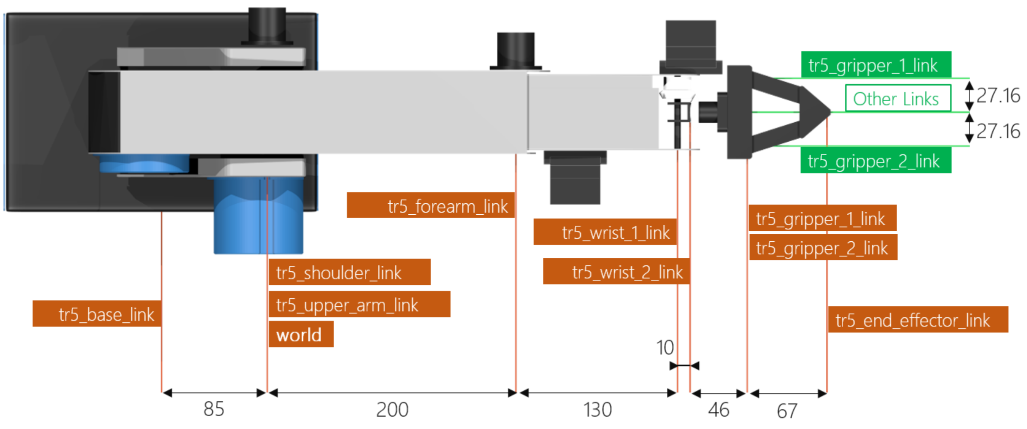


Figure 19 – Top perspective of TR5 with the overall dimensions in mm

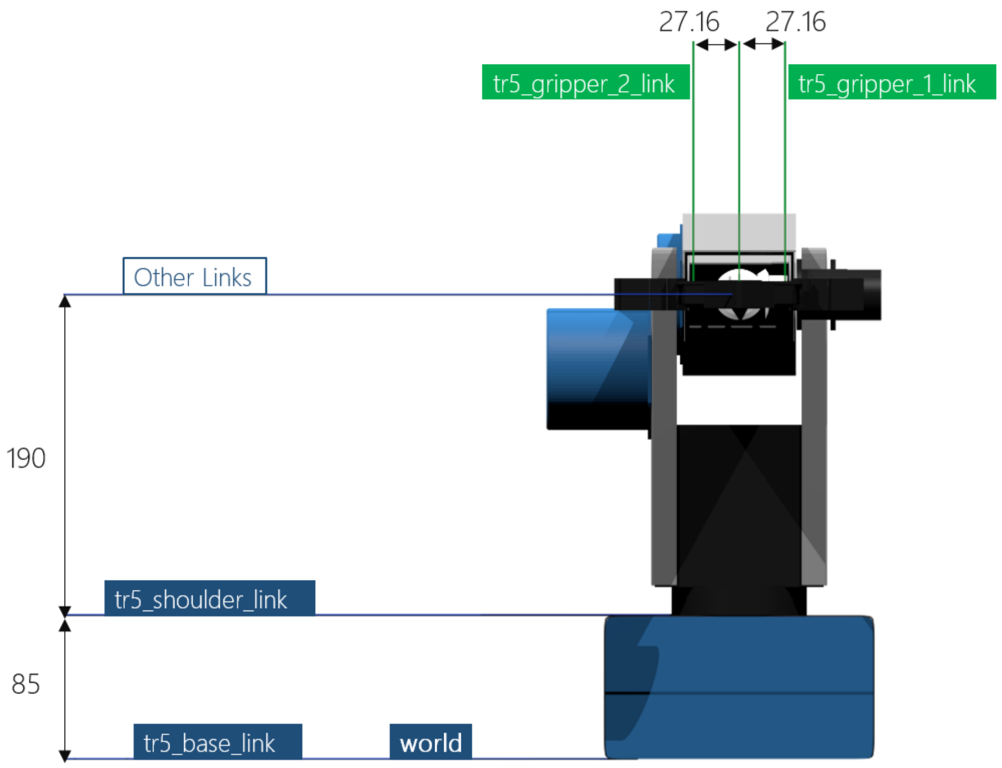


Figure 20- Front perspective of the TR5 with the overall dimensions in mm

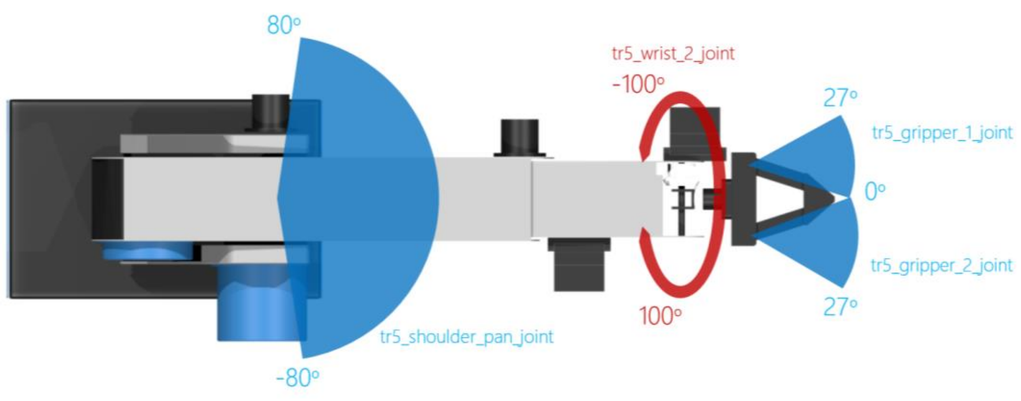


Figure 21 - Top perspective of TR5 with the angles of each joint

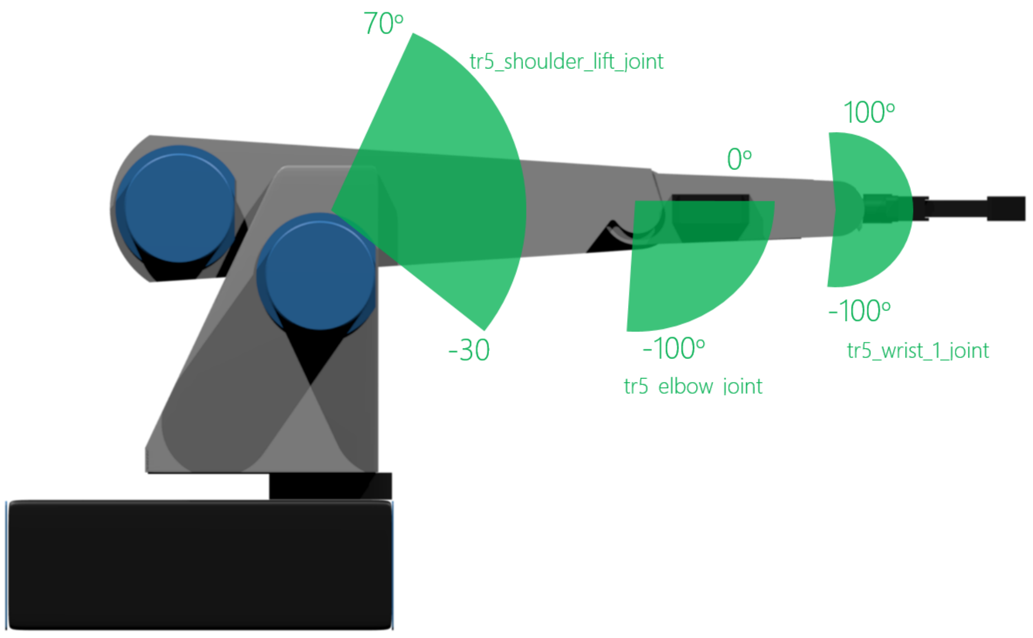


Figure 22 – Lateral perspective with rotation limits of each joint