Robotics and Intelligent Systems Lab 1

CO-542-A

Fall Semester 2022

Prof. Dr. Francesco Maurelli

Jacobs University Bremen

Report conducted by: Jose Tejeda Guzman.

Date of execution: Nov. 3rd, 2022 – Nov. 30th, 2022.

INTRODUCTION:

To describe how ROS middleware and tools work, **uvv_simulator** packages were used during the elaboration of this project.

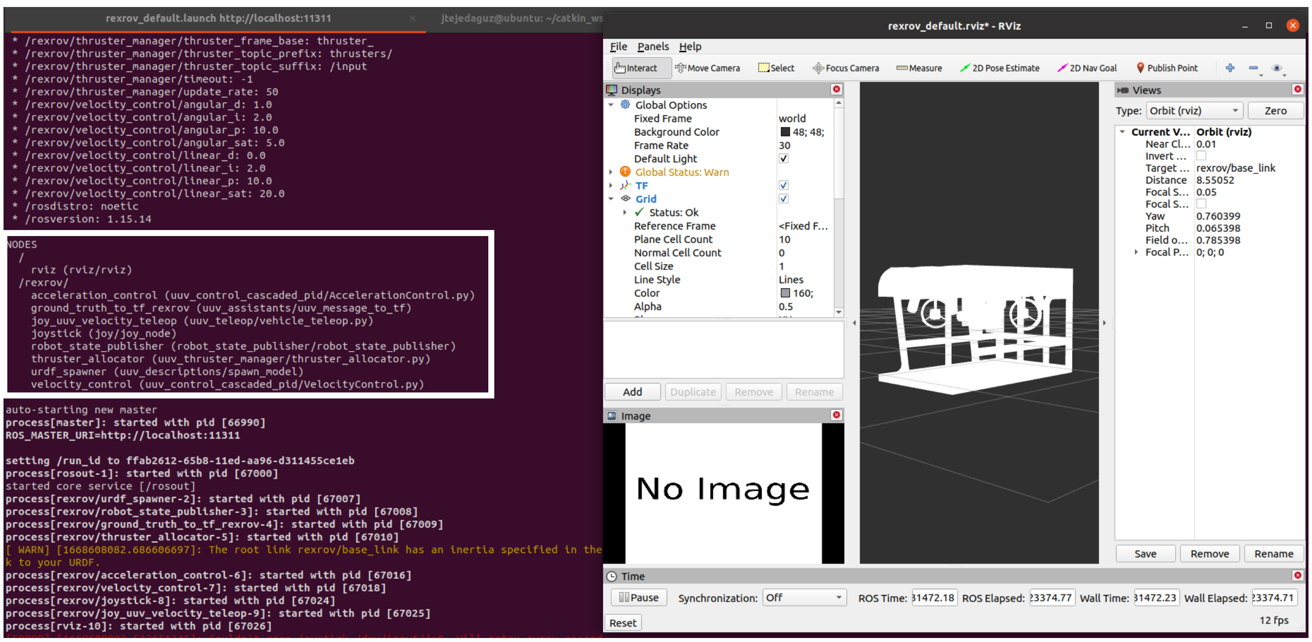
Along the written report graphical data such as images will be presented for a better understanding of ROS.

1. (a)

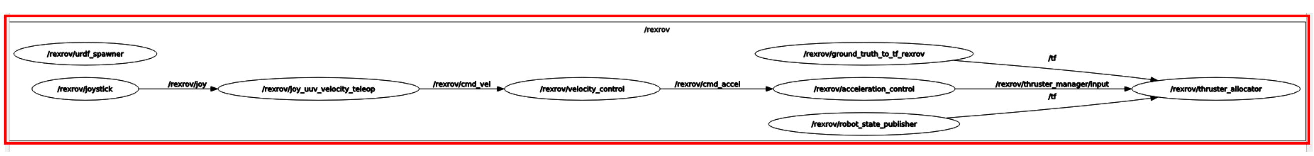
Estimated elaboration time: 4:00 hrs.
Contribution: Jose Tejeda (100%).

In order to describe and have a visual representation of node and topic connections inside the launch file **uuv_gazebo/rexrov,** rqt_grapg debugging ROS tool is used. Other graphical sources like terminal screenshots are also used along the written report.

Information about the nodes is easy to find when the file is launched. An example of the previous statement is shown below.

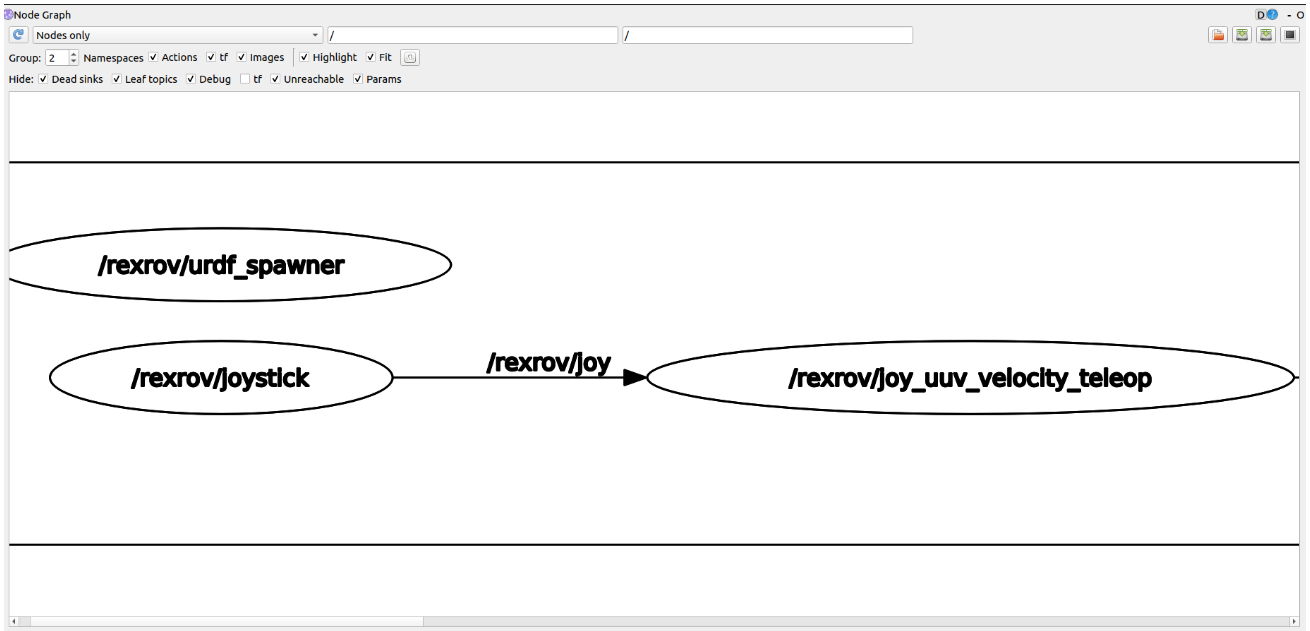


Rexrov launch file is represented by rqt_graoh as the whole red rectangle in the image, while the nodes as circles connected by topics.

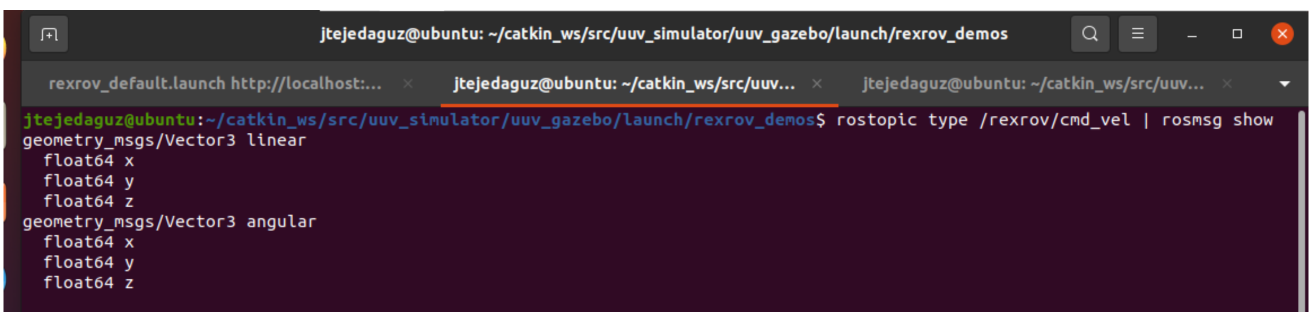


The initial node **/rexrov/joystick** is a single purposed executable program which receives linear and angular numerical data for x-axis, y-axis and z-axis from an external joystick (usually an Xbox controller). Once the executable program inside the node computes the data, information about position (defined by the axis) and movement (defined by pitch, yaw and roll) will be published by using **/rexrov/joy** topic as channel of communication.

Since **/rexrov/joy_uuv_velocity_teleop** is subscribed to **/rexrov/joy**, it will receive the information stored in the message. By a ROSPY program called **/vehicle_teleop_py** the node computes angular and linear velocities.

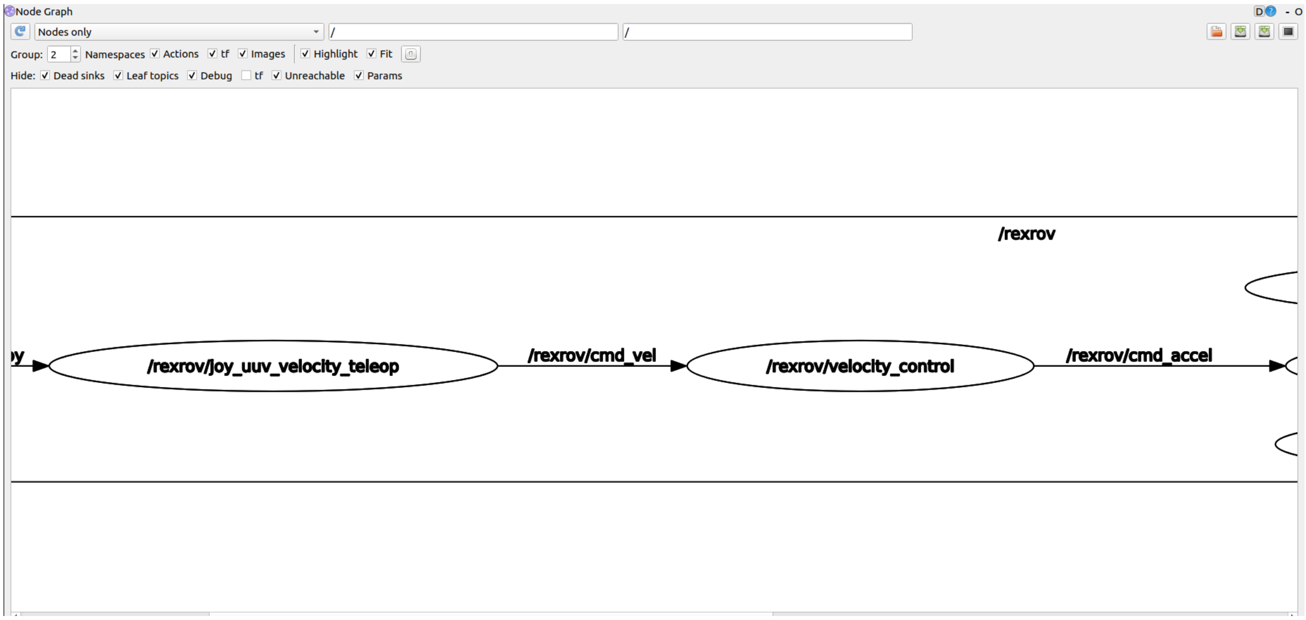


After angular and linear velocities are computed, **/rexrov/joy_uuv_velocity_teleop** becomes into a publisher node taking **/rexrov/cmd_vel** as topic. Is important to mention this topic stores the desired velocity and will be modified later by the ROSPY program into a callback function where it handles updated set velocity callbacks. In this occasion, the subscriber node will be **/rexrov/velocity_control** which main ROSPY infrastructure is based in self-called functions. These functions use very specific ROS messages; such as geometry_msgs/Twist.msg, where standard datatype is float64 as it can be seen in the image below.



As the name of the node suggest, the principal purpose of the calculations inside the ROSPY program is control the velocity until the desire velocity is reached.

Now that calculations are made, the actual node publishes a new message for the subscriber by the **/rexrov/cmd_accel** topic. As the name of the topic suggests, the ROS message used to move the data is a geometry_msgs/Accel.msg, this expresses acceleration broken into its angular and linear parts. As geometry_msgs/Twist.msg, the datatype is float64.

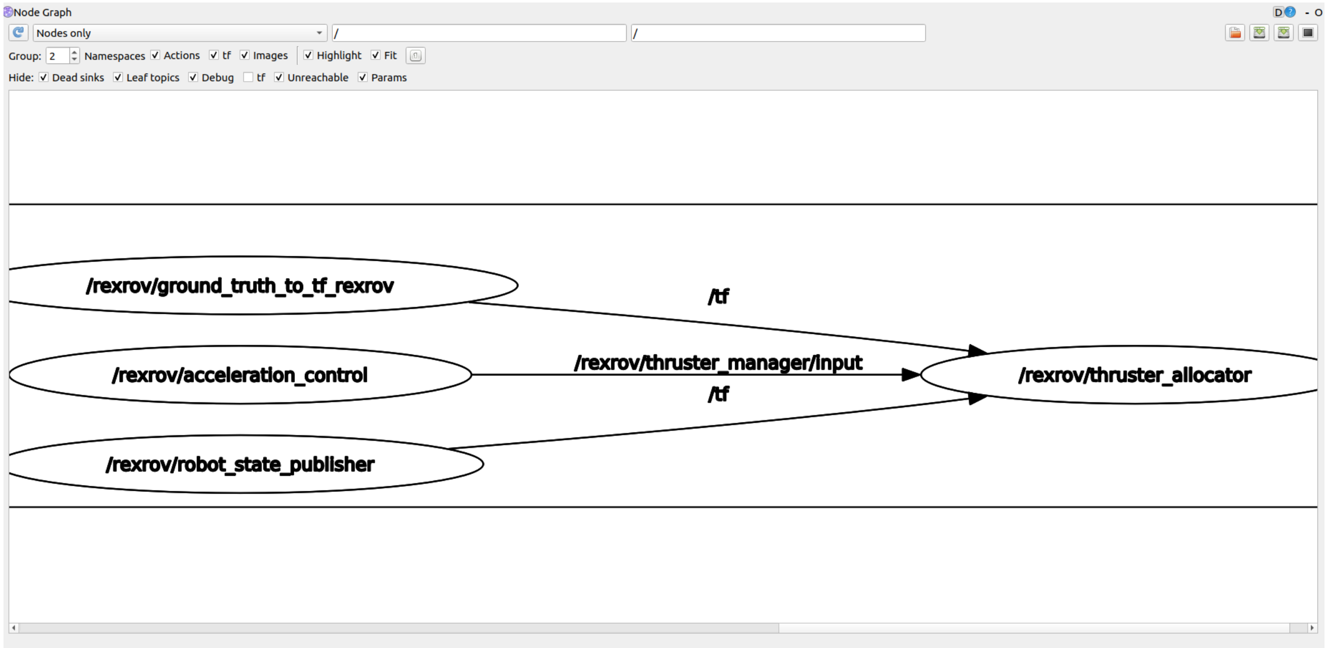


The subscriber uses the communication channel created by **/rexrov/cmd_accel** to get information about the vehicle acceleration. Since acceleration is involved in Classical Physics concepts such as; torque and moment of inertia, **/rexrov/acceleration_control** uses the mass of the vehicle and the data mentioned before to obtain 6D forces and torque vector. Once the final values are generated by the ROSPY program, geometry_msgs/Wrench.msg will be used to represents the force in free space, separated into its linear and angular parts.

The appearance of two mainly publisher nodes suggest other sensors are capturing information from the environment. For example, /**rex/rov/ground_truth_to_tf_revrov** node, uses data about frame vehicle positioning. By the ROSCPP program **uuv_message_to_tf** data captured by the motion sensors of the vehicle is processed and odometry functions inside the program estimates the change of position over time. Using geometry_msgs/PoseStamped.msg pose of the vehicle with reference coordinate frame and timestamp is shared to the publisher by the topic **/tf**. In the other hand, the publisher node **/rexrov/robot_state_publisher** uses data regarding the actual state of the robot, for example; positioning, velocity, effort and name of every joint of the vehicle. It also shares the data to the subscriber by the topic **/tf**.

The last node is working as subscriber, which main ROSPY program calculates an optimal combination of individual thrust forces based on the messages received by the three topics (**/tf's** and **/rexrov/thruster_manager/input**).

Along the whole process, services such as; InitCircularTrajectory.srv, GetThrusterState.srv SetSMControllerParams.srv are used to enable request/response communication between the nodes, where request parts usually include datatype float64, int32 and bool statements. Response parts usually returns a bool statement.



Nodes, topics, messages and services communicate between each other to develop a thruster control system that produce desired net force and torque for the marine vehicle.

1. (b)

Estimated elaboration time: 2:00 hrs.
Contribution: Jose Tejeda (100%).

By launching uuv_gazebo/rexrov_default.launch using **roslaunch** command, ROS master starts and communication middleware between nodes is allowed. Gazebo library and its capabilities for the marine vehicle are available now and graphical tools such as RVIZ start debugging the data. A breve summary about the parameters and their outputs are shown in the terminal, nodes and their ROSPY programs are also visible.

Access to the information about every node is possible due to **rosnode info <node_name>**, where subscriptions, publications and services are displayed on the terminal. Connections carried out by the node show the data transported, usually TCPROS (Messages and services), direction and topic used as communication channel are also visible in this part. To know more information about the topics, **rostopic list -v** shows the whole list of topics and **rostopic info <topic_name>** shows type of message communicated between nodes. Subscriber and publisher are also projected in the screen. It is also possible see the message type and their description more in detail if **rostopic type <node_name> | rosmsg show** command is typed in the terminal.

Inspect services and the type itself is viable if **rosservice type <service_node> | rossrv show** is typed on the terminal. Information of the “string” type is very common in the uuv_simulator during the inter-node communication, this communication between nodes is possible due to the request and response parts contained in every service. To retrieve data about request/response parts **rossrv show <service_node>** can be also used in the terminal. Similar to services, command used to know datatype and name contained in messages is **rosmsg info <message_type>**.

As mentioned before, **rqt_graph** plots a graphical representation where nodes and their connection by topics are shown on the screen. One direction arrow suggests nodes are working as servers; this means they only provide the service.

2. (a)

Estimated elaboration time: 2:00 hrs.
Contribution: Jose Tejeda (100%).

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

<!--Adding the arguments needed to change the way how Gazebo works-->

<!--Launching user interface window of Gazebo to have visual representation-->
    <arg name="gui" default="true"/>
<!--Gazebo starts running since the first moment is called-->
    <arg name="paused" default="false"/>
<!--Timeout to set teleop_twist_keyboard parameters is not added in Gazebo-->
    <arg name="set_timeout" default="false"/>
<!--As mentioned before, timeout is not added, this means default values are equalize to zero-->
    <arg name="timeout" default="0.0"/>

<!--Launching rexrov in uuv_gazebo/rexrov_default.launch-->
    <include file="$(find uuv_gazebo)/launch/rexrov_default.launch">
<!--Empty_uderwater.world is given as a value for the argument 'name = world_value'-->
        <arg name="world_name" value="worlds/empty_underwater.world"/>
<!--Calling the value previously defined as argument-->
        <arg name="paused" value="$(arg paused)"/>
<!--Telling the nodes to get Gazebo-publised time -->
        <arg name="use_sim_time" value="true"/>
<!--Calling the value previously defined as argument-->
        <arg name="gui" value="$(arg gui)"/>
<!--Unable recording for Gazebo state-->
        <arg name="headless" value="false"/>
        <arg name="debug" value="false"/>
<!--Allowing gzserver and gzclient printing errors and warnings to the terminal-->
        <arg name="verbose" value="true"/>
    </include>

<!--Including Teleop_twist_keyboard node in such a way it can be used to control the ROV.-->

<!--Package where the node is allocated, its type and method used as output must be mentioned-->
    <node pkg="uuv_teleop"
       type="teleop_twist_keyboard.py"
       name="teleop"
       output="screen">

    </node>
</launch>
```

3. (a)

Estimated elaboration time: 3:00 hrs.
Contribution: Jose Tejeda (100%).

```python
#!/usr/bin/env python

#Importing geometry_msg libraries to use messages

import rospy
import numpy

from geometry_msgs.msg import Twist
from geometry_msgs.msg import TwistStamped
from geometry_msgs.msg import Wrench


""" Reading from the keyboard and publishing to
uuv_gazebo/rexrov_wrench_control.launch through
---------------------------
Moving around:

   u    i    o
   j    k    l
   m    ,    .

---------------------------
   U    I    O
   J    K    L
   M    <    >

t : up (+z)
b : down (-z)

anything else : stop

CTRL-C to kill rosrun """

moveBindings = {
        'i':(1,0,0,0),
        'o':(1,0,0,-1),
        'j':(0,0,0,1),
        'l':(0,0,0,-1),
        'u':(1,0,0,1),
        ',':(-1,0,0,0),
        '.':(-1,0,0,1),
        'm':(-1,0,0,-1),
        'O':(1,-1,0,0),
        'I':(1,0,0,0),
        'J':(0,1,0,0),
        'L':(0,-1,0,0),
        'U':(1,1,0,0),
        '<':(-1,0,0,0),
        '>':(-1,-1,0,0),
        'M':(-1,1,0,0),
        't':(0,0,1,0),
        'b':(0,0,-1,0),
    }


#Creating node

class SimilarTeleopTwistNode:

        #Defining functions and their parameters

        def __init__(self):

                print('SimilarTeleopNode: initializing node')

                #Creating the publisher and subscriber parts of the node, where topic and type-message are specified

                self.subscribe_accel= rospy.Subscriber('cmd_accel', numpy_msg(Accel), self.accel_callback)
                self.publish_force = rospy.Publisher('thruster_manager/input', Wrench, queue_size=1)
                self.x = 0.0
                self.y = 0.0
                self.z = 0.0
                self.speed = 0.0

```

```python
    def accel_callback (self, msg):

        #Calculating force and torque based on the information received by the node 'cmd_accel'
        #Angular and linear velocity are necesary to calculate the final force

        force = numpy.array((msg.accel.linear.x, msg.accel.linear.y, msg.accel.linear.z))

        torque = numpy.array((msg.accel.angular.x, msg.accel.angular.y, msg.accel.angular.z))

        #Stack force and torque vector
        force_torque = numpy.hstack((force, torque)).transpose()

        #Allocating the values into the force_msg
        force_msg = Wrench()
        force_msg.force.x = force[0]
        force_msg.force.y = force[1]
        force_msg.force.z = force[2]

        force_msg.torque.x = torque[0]
        force_msg.torque.y = torque[1]
        force_msg.torque.z = torque[2]

        #Publishing the values
        self.publish_force.publish(force_msg)

    def twist_run (self):

        wrench_msg = WrenchMsg()

        if stamped:
            wrench = wrench_msg.wrench
            wrench_msg.header.stamp = rospy.Time.now()
            wrench_msg.header.frame_id = wrench_frame
        else:
            wrench = wrench_msg
        while not self.done:
            if stamped:
                wrench_msg.header.stamp = rospy.Time.now()
                self.condition.acquire()

                # Waiting for a new message
                self.condition.wait(self.timeout)


                # Allocating the actual state into wrench message.
                wrench.force.x = self.x * self.speed
                wrench.force.y = self.y * self.speed
                wrench.force.z = self.z * self.speed
                wrench.torque.x = 0
                wrench.torque.y = 0
                wrench.torque.z = 0

                self.condition.release()

                self.publish_force.publish(wrench_msg)

if __name__ == '__main__':

    settings = saveTerminalSettings()

    print('starting similar_teleop_twist.py')
    rospy.init_node('similar_twist')

    #Creating values based on the parameter call
    key_timeout = rospy.get_param("~key_timeout", 0.5)
    twist_frame = rospy.get_param("~frame_id", '')

    try:
        node = SimilarTeleopTwistNode()
        rospy.spin()

        while(1):

            #Receiving the input from the keyboard
            key = getKey(settings, key_timeout)
            #If a valid key is given
            if key in moveBindings.keys():

                x = moveBindings[key][0]
                y = moveBindings[key][1]
                z = moveBindings[key][2]


            else:

            #If the user stops the input, neutral state
            if key == '' and x == 0 and y == 0 and z == 0:
                continue
                    x = 0
                    y = 0
                    z = 0
            #Escaping character
            if (key == '\x03'):
                break
    except rospy.ROSInterruptException:
        print('caught exception')
    print('exiting')
```

3.(b)

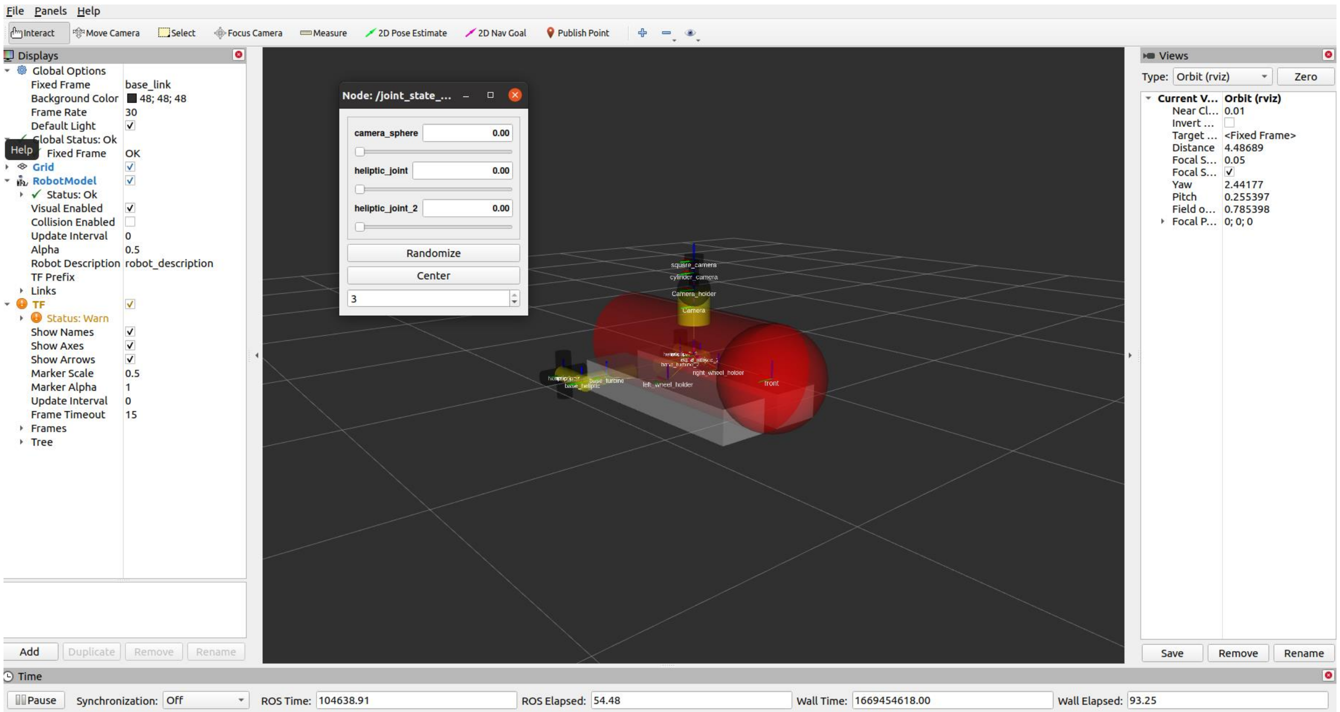
Estimated elaboration time: 3:00 hrs.
Contribution: Jose Tejeda (100%).

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

<!--Creating Gazebo world based on the arguments previoulsy made
x, y, z arguments for the free movement in the space-->

    <arg name="namespace" default="rexrov"/>
    <arg name="x" default="0"/>
    <arg name="y" default="0"/>
    <arg name="z" default="-70"/>

<!--Launching user interface window of Gazebo to have visual representation-->
    <arg name="gui" default="true"/>
<!--Gazebo starts running since the first moment is called-->
    <arg name="paused" default="false"/>
<!--Timeout to set teleop_twist_keyboard parameters is not added in Gazebo-->
    <arg name="set_timeout" default="false"/>
<!--As mentioned before, timeout is not added, this means default values are equalize to zero-->
    <arg name="timeout" default="0.0"/>

<!--Launching rexrov in uuv_gazebo/rexrov_default.launch-->
    <include file="$(find uuv_gazebo)/launch/rexrov_demos/rexrov_wrench_control.launch">
<!--Empty_uderwater.world is given as a value for the argument 'name = world_value'-->
        <arg name="world_name" value="worlds/empty_underwater.world"/>
<!--Calling the value previously defined as argument-->
        <arg name="paused" value="$(arg paused)"/>
<!--Telling the nodes to get Gazebo-publised time -->
        <arg name="use_sim_time" value="true"/>
<!--Calling the value previously defined as argument-->
        <arg name="gui" value="$(arg gui)"/>
<!--Unable recording for Gazebo state-->
        <arg name="headless" value="false"/>
        <arg name="debug" value="false"/>
<!--Allowing gzserver and gzclient printing errors and warnings to the terminal-->
        <arg name="verbose" value="true"/>
        <arg name="namespace" value="$(arg namespace)"/>
<!--Calling arguments-->
        <arg name="x" value="$(arg x)"/>
        <arg name="y" value="$(arg y)"/>
        <arg name="z" value="$(arg z)"/>
    </include>

<!--Including Teleop_twist_keyboard node in such a way it can be used to control the ROV.-->


<!--Package where the node is allocated, its type and method used as output must be mentioned-->
    <node pkg="uuv_teleop"
        type="similar_teleop_twist.py"
        name="teleop"
        output="screen">
    </node>

</launch>
```

4. (a)

Estimated elaboration time: 3:30 hrs.
Contribution: Jose Tejeda (100%).

The implementation of ROS tools such as; Unified Robot Description Format and RVIZ, allowed the creation of the next marine robot model.



Some XML specifications used to describe robot properties are: joints (revolute and fixed), links and sensors (camera). Images below shown in detail the creation of the specifications mentioned before.

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

      <material name="red">
        <color rgba="1 0 0 1"/>
      </material>

      <material name="yellow">
        <color rgba="1 1 0 1"/>
      </material>

     <material name="white">
       <color rgba="1 1 1 1"/>
     </material>

     <material name="black">
       <color rgba="0 0 0 1"/>
     </material>

     <link name="base_link">
       <inertial>
         <mass value="5"/>
         <origin rpy="0 1.57 0" xyz="0 0 0"/>
         <inertia ixx="0.2" ixy="0" ixz="0" iyy="0.2" iyz="0" izz="0.1" />
       </inertial>
       <visual>
         <geometry>
           <cylinder length="1.3" radius="0.3"/>
         </geometry>
         <origin rpy="0 1.57 0" xyz="0 0 0"/>
         <material name="red"/>
       </visual>
       <collision>
         <geometry>
           <cylinder length="1.3" radius="0.3"/>
         </geometry>
       </collision>
     </link>

     <link name="front">
       <visual>
         <geometry>
           <sphere radius="0.3"/>
         </geometry>
         <material name="red"/>
       </visual>
       <collision>
         <geometry>
           <sphere radius="0.3"/>
         </geometry>
       </collision>
     </link>

     <joint name="base_front" type="fixed">
       <parent link="base_link"/>
       <child link="front"/>
        <origin xyz="-0.65 0 0"/>
     </joint>

     <link name="right_wheel_holder">
        <visual>
          <geometry>
            <box size="1.3 0.3 0.2"/>
          </geometry>
          <origin rpy="0 0 0" xyz="0 0 -0.08"/>
          <material name="white"/>
        </visual>
        <collision>
          <geometry>
            <box size="1.3 0.3 0.2"/>
          </geometry>
          <origin rpy="0 0 0" xyz="0 0 -0.08"/>
        </collision>
    </link>

        <joint name="base_to_wheel" type="fixed">
          <parent link="base_link"/>
          <child link="right_wheel_holder"/>
          <origin xyz="0 -0.2 -0.12"/>
        </joint>

        <link name="left_wheel_holder">
        <visual>
          <geometry>
            <box size="1.3 0.3 0.2"/>
          </geometry>
          <origin rpy="0 0 0" xyz="0 0 -0.08"/>
          <material name="white"/>
        </visual>

        <collision>
          <geometry>
            <box size="1.3 0.3 0.2"/>
          </geometry>
          <origin rpy="0 0 0" xyz="0 0 -0.08"/>
          </collision>
        </link>
```

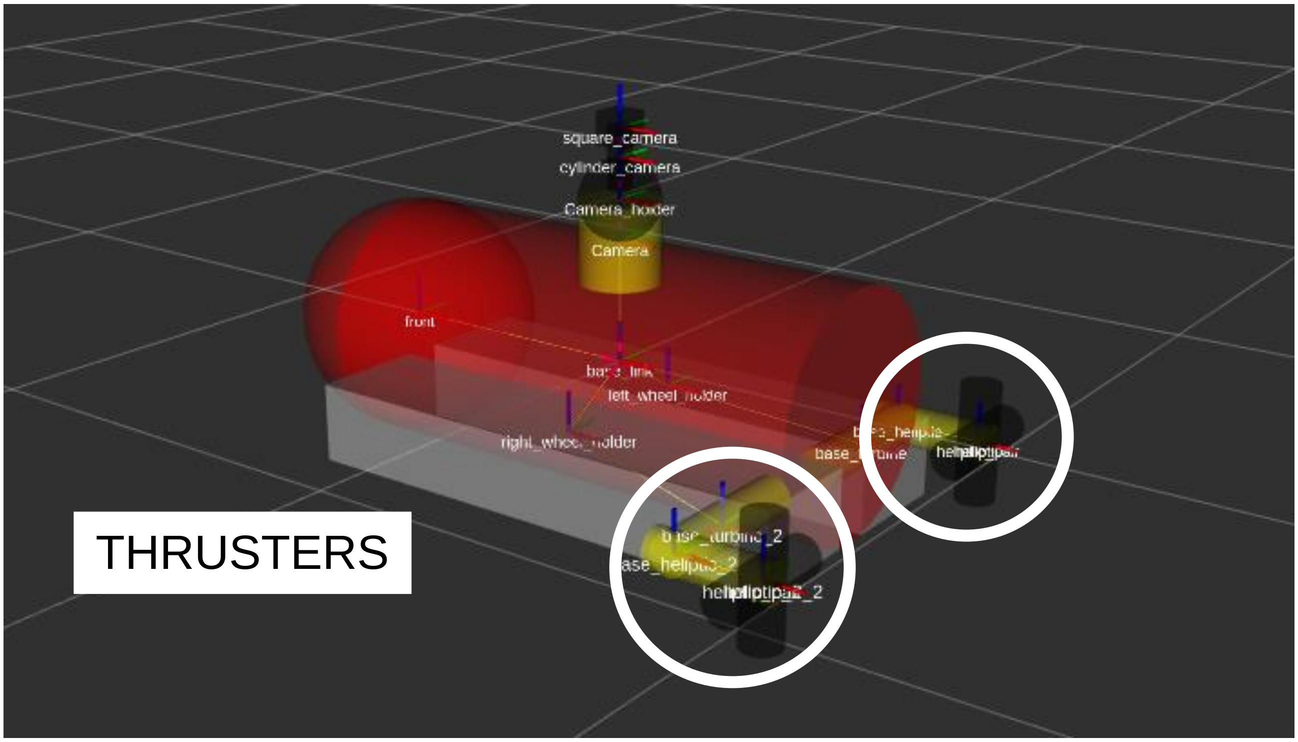
```xml

    <joint name="base_to_wheel_2" type="fixed">
      <parent link="base_link"/>
      <child link="left_wheel_holder"/>
      <origin xyz="0 0.2 -0.12"/>
    </joint>

    <link name="Camera">
      <visual>
        <geometry>
          <cylinder length="0.2" radius="0.1"/>
        </geometry>
        <origin rpy="0 0 0" xyz="0 0 0"/>
        <material name="yellow"/>
      </visual>
    </link>

    <joint name="camera_s" type="fixed">
      <parent link="base_link"/>
      <child link="Camera"/>
      <origin xyz="0 0 0.3"/>
    </joint>

    <link name="Camera_holder">
      <visual>
        <geometry>
          <sphere radius="0.1"/>
        </geometry>
        <origin rpy="0 0 0" xyz="0 0 0"/>
        <material name="black"/>
      </visual>
    </link>

    <joint name="camera_sphere" type="revolute">
      <limit lower="0" upper="6.28" effort="20" velocity="0.5"/>
      <axis xyz= "0 0 1"/>
      <parent link="Camera"/>
      <child link="Camera_holder"/>
      <origin xyz="0 0 0.1"/>
    </joint>

    <link name="cylinder_camera">
      <visual>
        <geometry>
          <cylinder length="0.15" radius="0.03"/>
        </geometry>
        <origin rpy="0 0 0" xyz="0 0 0"/>
        <material name="black"/>
      </visual>
    </link>


    <joint name="cylinder_joint" type="fixed">
      <parent link="Camera_holder"/>
      <child link="cylinder_camera"/>
      <origin xyz="0 0 0.1"/>
    </joint>

    <link name="square_camera">
      <visual>
        <geometry>
          <box size="0.08 0.08 0.08"/>
        </geometry>
        <material name="black"/>
      </visual>
    </link>

    <joint name="square_joint" type="fixed">
      <parent link="cylinder_camera"/>
      <child link="square_camera"/>
      <origin xyz="0 0 0.07"/>
    </joint>

    <sensor name="cam_sensor" update_rate="15">
      <parent link="square_camera"/>
      <origin xyz="0 0 0" rpy="0 0 0"/>
      <camera>
        <image width="640" height="400" hfov="1.57" format="RGB8" near="0.01" far="40.0"/>
      </camera>
    </sensor>

    <link name="base_turbine">
      <visual>
        <geometry>
          <cylinder length="0.4" radius="0.05"/>
        </geometry>
        <origin rpy="1.57 0 0" xyz="0 0 0"/>
        <material name="yellow"/>
      </visual>
    </link>
```

```xml

        <joint name="base_to_turbine" type="fixed">
          <parent link="left_wheel_holder"/>
          <child link="base_turbine"/>
          <origin xyz="0.5 0.1 -0.07"/>
        </joint>

        <link name="base_helipctic">
          <visual>
            <geometry>
              <cylinder length="0.3" radius="0.05"/>
            </geometry>
            <origin rpy="1.57 0 1.57" xyz="0.1 0 0"/>
            <material name="yellow"/>
          </visual>
          <collision>
            <geometry>
              <cylinder length="0.3" radius="0.05"/>
            </geometry>
          <origin rpy="0 1.57 0" xyz="0.1 0 0"/>
          </collision>
        </link>

        <joint name="base_to_helipctic" type="fixed">
          <parent link="base_turbine"/>
          <child link="base_helipctic"/>
          <origin xyz="0 0.18 0"/>
        </joint>

        <link name="base_turbine_2">
          <visual>
            <geometry>
              <cylinder length="0.4" radius="0.05"/>
            </geometry>
            <origin rpy="1.57 0 0" xyz="0 0 0"/>
            <material name="yellow"/>
          </visual>

        </link>

        <joint name="base_to_turbine_2" type="fixed">
          <parent link="right_wheel_holder"/>
          <child link="base_turbine_2"/>
          <origin xyz="0.5 -0.1 -0.07"/>
        </joint>


        <link name="base_helipctic_2">
          <visual>
            <geometry>
              <cylinder length="0.3" radius="0.05"/>
            </geometry>
            <origin rpy="1.57 0 1.57" xyz="0.1 0 0"/>
            <material name="yellow"/>
          </visual>
          <collision>
            <geometry>
              <cylinder length="0.3" radius="0.05"/>
            </geometry>
          <origin rpy="0 1.57 0" xyz="0.1 0 0"/>
          </collision>
        </link>

        <joint name="base_to_helipctic_2" type="fixed">
          <parent link="base_turbine_2"/>
          <child link="base_helipctic_2"/>
          <origin xyz="0 -0.18 0"/>
        </joint>

        <link name="helipctic">
          <visual>
            <geometry>
              <cylinder length="0.3" radius="0.05"/>
            </geometry>
            <origin rpy="1.57 0 0" xyz="0 0 0"/>
            <material name="black"/>
          </visual>
          <collision>
            <geometry>
              <cylinder length="0.3" radius="0.05"/>
            </geometry>
          <origin rpy="1.57 0 0 " xyz="0 0 0"/>
          </collision>
        </link>

        <joint name="helipctic_joint" type="revolute">
          <limit lower="0" upper="6.28" effort="20" velocity="0.5"/>
          <axis xyz= "1 0 0"/>
          <parent link="base_helipctic"/>
          <child link="helipctic"/>
          <origin xyz="0.22 0 0"/>
        </joint>
```

```
        <link name="heliptic_pair">
          <visual>
            <geometry>
              <cylinder length="0.3" radius="0.05"/>
            </geometry>
            <origin rpy="0 0 1.57" xyz="0 0 0"/>
            <material name="black"/>
          </visual>

          <collision>
            <geometry>
              <cylinder length="0.3" radius="0.05"/>
            </geometry>
          <origin rpy="0 0 1.57" xyz="0 0 0"/>
          </collision>
        </link>

        <joint name="heliptic_pair_point" type="fixed">
          <parent link="heliptic"/>
          <child link="heliptic_pair"/>
          <origin xyz="0 0 0"/>
        </joint>

        <link name="heliptic_2">
          <visual>
            <geometry>
              <cylinder length="0.3" radius="0.05"/>
            </geometry>
            <origin rpy="1.57 0 0" xyz="0 0 0"/>
            <material name="black"/>
          </visual>
          <collision>
            <geometry>
              <cylinder length="0.3" radius="0.05"/>
            </geometry>
          <origin rpy="1.57 0 0 " xyz="0 0 0"/>
          </collision>
        </link>

        <joint name="heliptic_joint_2" type="revolute">
          <limit lower="0" upper="6.28" effort="20" velocity="0.5"/>
          <axis xyz= "1 0 0"/>
          <parent link="base_heliptic_2"/>
          <child link="heliptic_2"/>
          <origin xyz="0.22 0 0"/>
        </joint>


        <link name="heliptic_pair_2">
          <visual>
            <geometry>
              <cylinder length="0.3" radius="0.05"/>
            </geometry>
            <origin rpy="0 0 1.57" xyz="0 0 0"/>
            <material name="black"/>
          </visual>
          <collision>
            <geometry>
              <cylinder length="0.3" radius="0.05"/>
            </geometry>
          <origin rpy="0 0 1.57" xyz="0 0 0"/>
          </collision>
        </link>

        <joint name="heliptic_pair_point_2" type="fixed">
          <parent link="heliptic_2"/>
          <child link="heliptic_pair_2"/>
          <origin xyz="0 0 0"/>
        </joint>

</robot>

```

4.(b)

Estimated elaboration time: 0:40 hrs.
Contribution: Jose Tejeda (100%).



The main objective about the placement of thrusters in this ROV is based in the free movement in three-dimensional space, where translational and rotational motions are necessary in order to obtain six degrees of freedom (6DOF).

Since both thrusters have 2π rotation in the radian measure (360 degrees) around x-axis and a rotation from $\frac{\pi}{4}$ to $\frac{7\pi}{4}$ in the y-axis. Lineal motion, horizontal straightness, yawing, pitching and vertical straightness is possible.

4.(c)

Estimated elaboration time: 3:00 hrs.
Contribution: Jose Tejeda (100%).

Taking the representation of force in free space is possible by calling the service file VectorInts in the server node. The request part of the service is called by the node and the response part is published once the thrust force is calculated by the client node.

Service file, server node and client node are shown in the order previously mention.

```
geometry_msgs/Vector3 force
float64 x
float64 y
float64 z

geometry_msgs/Vector3 torque
float64 x
float64 y
float64 z
---
float64 x
float64 y
float64 z
```

SERVICE FILE

```python
#!/usr/bin/env python

import sys
import rospy
import numpy
import geometry_msgs.msg as geometry_msgs

from beginner_tutorials.srv import vectorInts

class ThrusterController:

        def __init__ (self):

                print('ThrusterController: initializing node')

                # geometry_msgs/Vector3.msg to represent Trust as a vector in free space
                self.pub_thruster = rospy.Publisher('thruster_trust', geometry_msgs.Vector3, queue_size=10)


        def handle_thruster(req):

                #Taking the request part of the message
                force = req.force
                torque = req.torque

                #Creating a vector with the information received
                vector_force = numpy.array([req.force.x, req.force.y, req.force.z])
                vector_torque = numpy.array([req.torque.x, req.torque.y, req.torque.z])

                #Calculating the output
                thruster_output = (vector_force * vector_torque)

                #Publishing the in
                thruster_thrust = geometry_msgs.Vector3()
                self.pub_thruster.publish(thruster_thrust)

                return thruster_output

        def thruster_server():

                #Initializing the server
                rospy.init_node('thruster_server')
                s = rospy.Service('thruster', vectorInts, handle_thruster)
                print ("Calculating Thrust")
                rospy.spin()


if __name__ == '__main__':
        thruster_server()

        try:
                node = ThrusterController()

        except rospy.ROSInterruptException:
                print('caught exception')
           print('exiting')

```

SERVER NODE

```python
#!/usr/bin/env python

import sys
import rospy
from uuv_assistants.srv import *

def thruster_client(force, torque):
        #Calling the server
        rospy.wait_for_service('thruster_thrust')
        try:
                thruster_thrust = rospy.ServiceProxy('thruster_thrust', vectorInts)
                req = vectorIntsRequest()
                req.force = force
                req.torque = torque
                resp1 = thruster_thrust(req)
                return resp1.thruster_output
        except rospy.ServiceException as e:
                print("Service call failed: %s" %e)

if __name__ == "__main__":
        if len(sys.argv) == 3
                force = float(sys.argv[1.0])
                torque = float(sys.argv[2.0])

        else:
                #Publish the thrust force
                print("Thruster_client.py")
                sys.exit(1)
                print("Thrust = %s"%(thruster_client(force, torque))
```

CLIENT NODE

4.(d)

Estimated elaboration time: 2:00 hrs.

Contribution: Jose Tejeda (100%).

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

                <arg name="gui" default="true"/>
                <arg name="paused" default="false"/>
                <arg name="set_timeout" default="false"/>
                <arg name="timeout" default="0.0"/>

                <arg name="x" default="0"/>
                <arg name="y" default="0"/>
                <arg name="z" default="-70"/>

                <include file="$(find uuv_gazebo)/launch/rexrov_demos/rexrov_wrench_control.launch">
                <arg name="world_name" value="$(find robot)/worlds/empty_underwater.world"/>

                <arg name="paused" value="$(arg paused)"/>
                <arg name="use_sim_time" value="true"/>
                <arg name="gui" value="$(arg gui)"/>
                <arg name="headless" value="false"/>
                <arg name="debug" value="false"/>
                <arg name="verbose" value="true"/>
                <arg name="recording" value="false"/>
                <arg name="x" value="$(arg x)"/>
                <arg name="y" value="$(arg y)"/>
                <arg name="z" value="$(arg z)"/>

                </include>

                <node pkg="uuv_teleop"
                type="similar_teleop_twist.py"
                name="teleop"
                output="screen">
                </node>

                <include file="$(find uuv_assistants)/launch/thruster_node.launch">

                <node pkg="uuv_assistants"
                        type="thruster_node.py"
                        name="thruster_node"
                        output="screen">


                 </node>

                 <node pkg="uuv_assistants"
                         type="Thruster_client.py"
                         name="Thruster_client"
                         output="screen">

                 </node>

                 <node name="spawn_urdf"
                 pkg = "uuv_gazebo_worlds"
                 type = "spawn_model"
                 args = "-param -x 13 -y -13 -z 1.2 -urdf -model robot"/>

        </launch>
```