# RIS LAB 1 - FALL 2022

---

Lab Report November 2022

Zain Ahmed Samdani
Ramin Udash
Mohamed Mourad Ouazghire

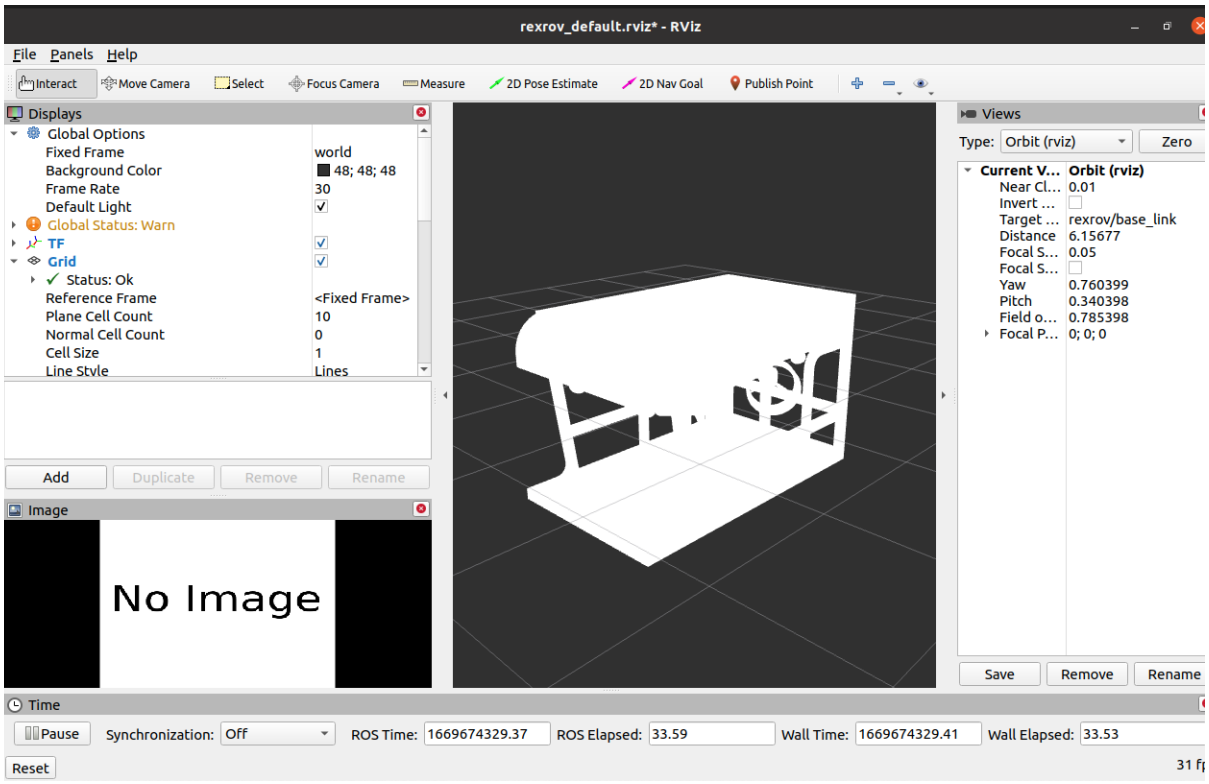
# **Task 1:** Inspecting nodes, topics, and services

Mourad: 40% | Zain: 30% | Ramin: 30%

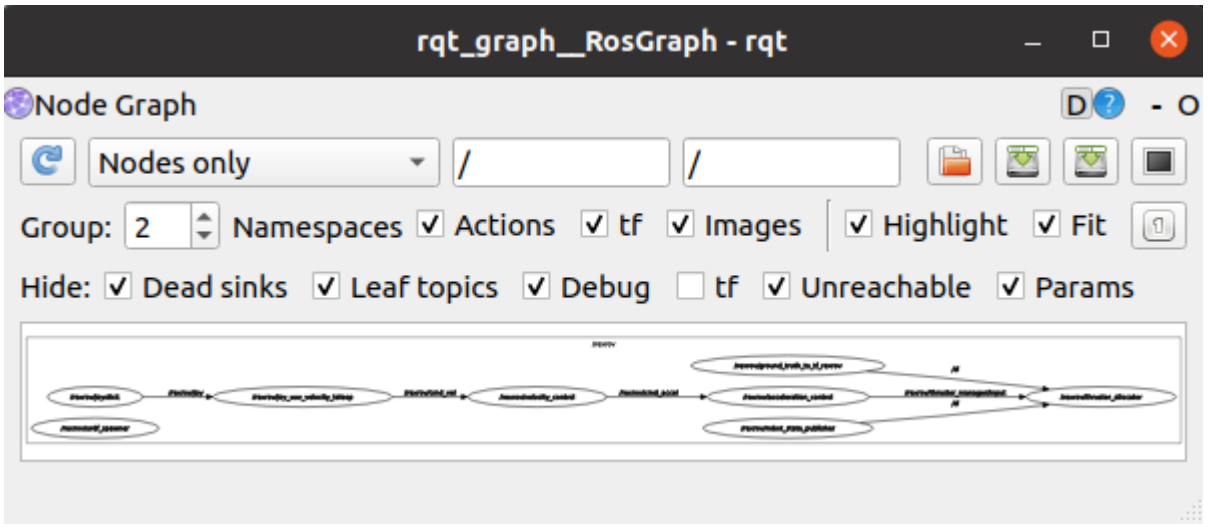
---

We launched uuv_gazebo/rexrov_default.launch and inspected the nodes, topics, services it launches along with the message/service types they use to communicate.

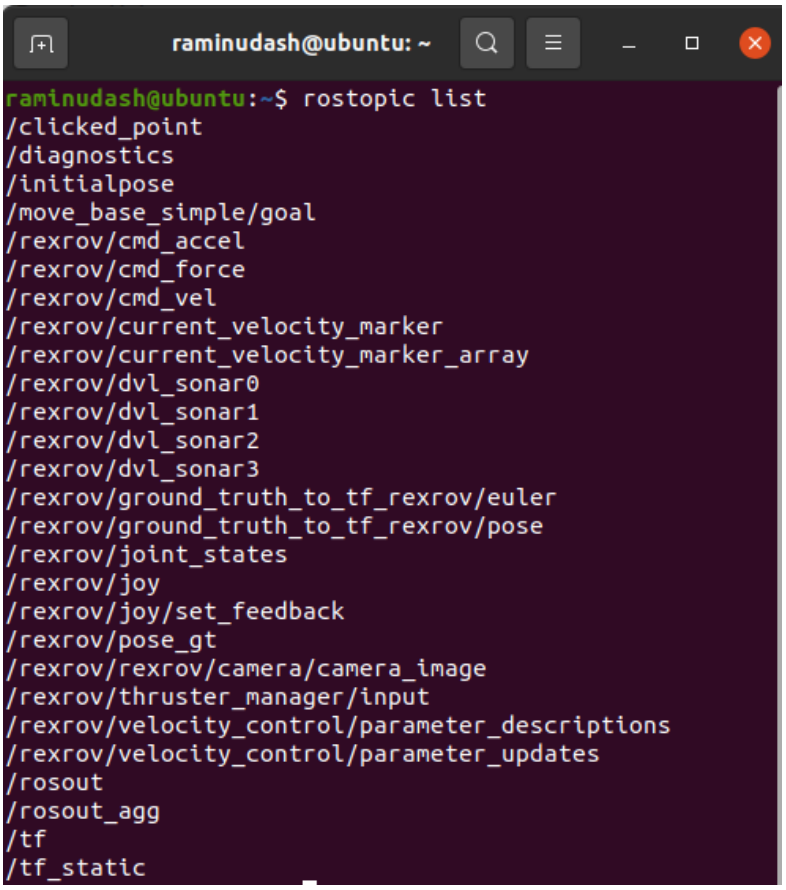
In order to retrieve information about the nodes, topics, services, and messages from “uuv_gazebo/rexrov_default.launch”, we first launched the file by executing the following command on the terminal: `$ roslaunch uuv_gazebo rexrov_default.launch`. This launched a RVIZ simulation of the robot.




By using the command `$ rosrun rqt_graph rqt_graph`, we were able to visualize the current nodes and topics that are active in /rexrov.



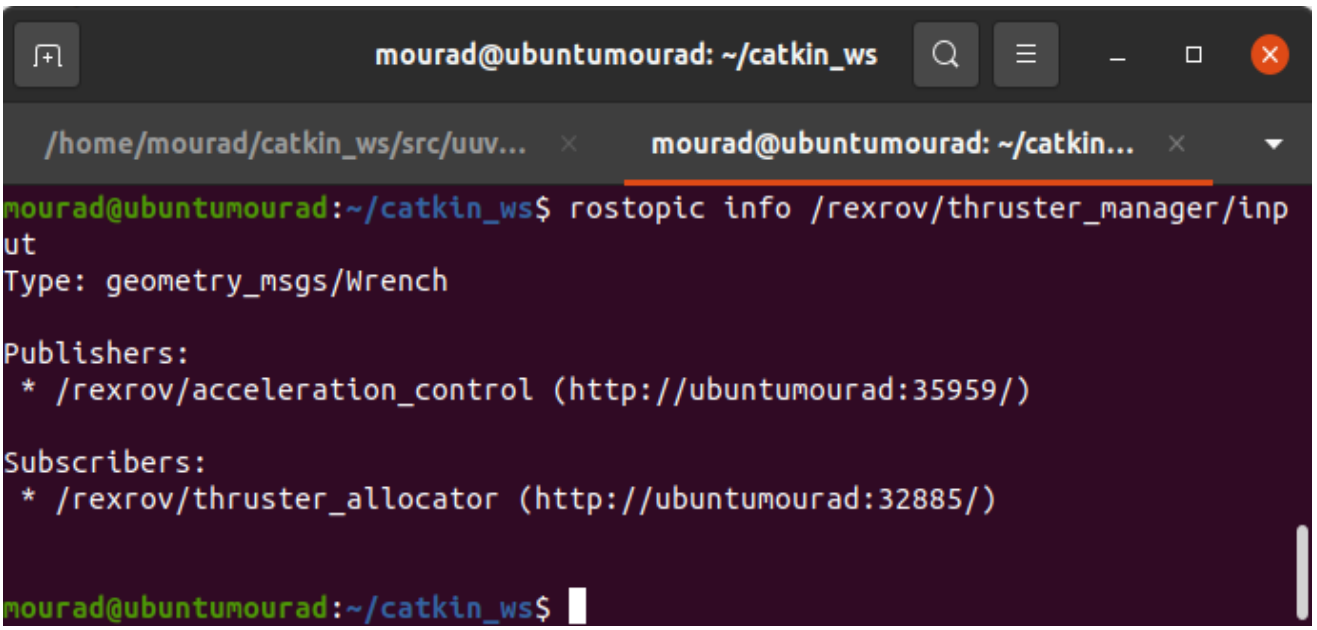
# Topics:




The command $ rostopic list displays a list of active published topics. The command $ rostopic info [topic_name] displays information(subscribers, publishers, message type) on the active specified topic. Multiple nodes can publish messages and subscribe to a single topic at the same time and receive messages.

The topics inspected in the RQT graph are:

rexrov/thruster_manager/input, which receives messages from the /rexrov/acceleration_control node. This is subscribed by /rexrov/thruster_allocator.



# Nodes:

```
raminudash@ubuntu:~$ rosnode list
/rexrov/acceleration_control
/rexrov/ground_truth_to_tf_rexrov
/rexrov/joy_uuv_velocity_teleop
/rexrov/joystick
/rexrov/robot_state_publisher
/rexrov/thruster_allocator
/rexrov/urdf_spawner
/rexrov/velocity_control
/rosout
/rviz
raminudash@ubuntu:~$
```

We use `$ rosnode list` to inspect all currently active nodes and `$ rosnode info [node_name]` to inspect the connection between the nodes, the publication, and the subscription to yield a better understand of the graph. A node is an executable that uses ROS to communicate with other nodes. The ROS_NAMESPACE environment variable lets us change the namespace of a node being launched, thereby remapping all of the names in that node.

**/rexrov/acceleration_control**

```
^Cmourad@ubuntumourad:~/catkin_ws$ rosnode info /rexrov/acceleration_control
--------------------------------------------------------------------------------
Node [/rexrov/acceleration_control]
Publications:
 * /rexrov/thruster_manager/input [geometry_msgs/Wrench]
 * /rosout [rosgraph_msgs/Log]

Subscriptions:
 * /rexrov/cmd_accel [geometry_msgs/Accel]
 * /rexrov/cmd_force [unknown type]

Services:
 * /rexrov/acceleration_control/get_loggers
 * /rexrov/acceleration_control/set_logger_level


contacting node http://ubuntumourad:35959/ ...
Pid: 24773
Connections:
 * topic: /rosout
    * to: /rosout
    * direction: outbound (33101 - 127.0.0.1:35182) [16]
    * transport: TCPROS
 * topic: /rexrov/thruster_manager/input
    * to: /rexrov/thruster_allocator
    * direction: outbound (33101 - 127.0.0.1:35166) [9]
    * transport: TCPROS
 * topic: /rexrov/cmd_accel
    * to: /rexrov/velocity_control (http://ubuntumourad:46347/)
    * direction: inbound
    * transport: TCPROS
```

This node has two publishers of message type geometry_msgs and rosgraph_msgs. A message defines the structure of the data that is passed between nodes. The /rexrov/thruster_manager/input topic is a publisher to this node and it is a subscriber to the active node /rexrov/thruster_allocator. Along with services /rexrov/acceleration_control/get_loggers and /rexrov/acceleration_control/set_logger_level of type roscpp/GetLoggers and roscpp/SetLoggerLevel.

### /rexrov/ground_truth_to_tf_rexrov

```
mourad@ubuntumourad:~/catkin_ws$ rosnode info /rexrov/ground_truth_to_tf_rexrov
--------------------------------------------------------------------------------
Node [/rexrov/ground_truth_to_tf_rexrov]
Publications:
 * /rexrov/ground_truth_to_tf_rexrov/euler [geometry_msgs/Vector3Stamped]
 * /rexrov/ground_truth_to_tf_rexrov/pose [geometry_msgs/PoseStamped]
 * /rosout [rosgraph_msgs/Log]
 * /tf [tf2_msgs/TFMessage]

Subscriptions:
 * /rexrov/pose_gt [unknown type]

Services:
 * /rexrov/ground_truth_to_tf_rexrov/get_loggers
 * /rexrov/ground_truth_to_tf_rexrov/set_logger_level
```

This node has four publishers of message type geometry_msgs, rosgraph_msgs, and tf2_msgs. The /rexrov/ground_truth_to_tf_rexrov/euler topic is only a publisher to this node. The /rexrov/ground_truth_to_tf_rexrov/pose is only a publisher to this node. This node is subscribed to only one topic and has two services of get_loggers and set_loggers_level.

### /rexrov/robot_state_publisher

```
mourad@ubuntumourad:~/catkin_ws$ rosnode info /rexrov/robot_state_publisher
--------------------------------------------------------------------------------
Node [/rexrov/robot_state_publisher]
Publications:
 * /rosout [rosgraph_msgs/Log]
 * /tf [tf2_msgs/TFMessage]
 * /tf_static [tf2_msgs/TFMessage]

Subscriptions:
 * /rexrov/joint_states [unknown type]

Services:
 * /rexrov/robot_state_publisher/get_loggers
 * /rexrov/robot_state_publisher/set_logger_level
```

This node has three publishers of message type rosgraph_msgs and tf2_msgs. The /tf topic has a publisher to this node and to /rexrov/robot_state_publisher. The /tf_static is only a publisher to this node. This node is subscribed to only one topic and has two services of get_loggers and set_loggers_level.

### /rexrov/joy_uuv_velocity_teleop

```
mourad@ubuntumourad:~/catkin_ws$ rosnode info /rexrov/joy_uuv_velocity_teleop
--------------------------------------------------------------------------------
Node [/rexrov/joy_uuv_velocity_teleop]
Publications:
 * /rexrov/cmd_vel [geometry_msgs/Twist]
 * /rexrov/home_pressed [std_msgs/Bool]
 * /rosout [rosgraph_msgs/Log]

Subscriptions:
 * /rexrov/joy [sensor_msgs/Joy]

Services:
 * /rexrov/joy_uuv_velocity_teleop/get_loggers
 * /rexrov/joy_uuv_velocity_teleop/set_logger_level
```

This node has three publishers of message type geometry_msgs, std_msgs, and rosgraph_msgs. The /rexrov/cmd_vel topic is only a publisher to this node. The /rexrov/home_pressed is only a publisher to this node. This node is subscribed to only one topic and has two services of get_loggers and set_loggers_level.

### /rviz

```
mourad@ubuntumourad:~/catkin_ws$ rosnode info /rviz
--------------------------------------------------------------------------------
Node [/rviz]
Publications:
 * /clicked_point [geometry_msgs/PointStamped]
 * /initialpose [geometry_msgs/PoseWithCovarianceStamped]
 * /move_base_simple/goal [geometry_msgs/PoseStamped]
 * /rosout [rosgraph_msgs/Log]

Subscriptions:
 * /rexrov/current_velocity_marker [unknown type]
 * /rexrov/current_velocity_marker_array [unknown type]
 * /rexrov/dvl_sonar0 [unknown type]
 * /rexrov/dvl_sonar1 [unknown type]
 * /rexrov/dvl_sonar2 [unknown type]
 * /rexrov/dvl_sonar3 [unknown type]
 * /rexrov/pose_gt [unknown type]
 * /rexrov/rexrov/camera/camera_image [unknown type]
 * /tf [tf2_msgs/TFMessage]
 * /tf_static [tf2_msgs/TFMessage]

Services:
 * /rviz/get_loggers
 * /rviz/load_config
 * /rviz/load_config_discarding_changes
 * /rviz/reload_shaders
 * /rviz/save_config
 * /rviz/set_logger_level
```

This node has four publishers of message type geometry_msgs and rosgraph_msgs. This node is subscribed to ten topics and has five services.

### /rosout

```
mourad@ubuntumourad:~/catkin_ws$ rosnode info /rosout
--------------------------------------------------------------------------------
Node [/rosout]
Publications:
 * /rosout_agg [rosgraph_msgs/Log]

Subscriptions:
 * /rosout [rosgraph_msgs/Log]

Services:
 * /rosout/get_loggers
 * /rosout/set_logger_level
```

/rosout is the name of the console log reporting mechanism in ROS. It can be thought as comprising several components: The `rosout` node for subscribing, logging, and republishing the messages

### /rexrov/velocity_control

```
mourad@ubuntumourad:~/catkin_ws$ rosnode info /rexrov/velocity_control
--------------------------------------------------------------------------------
Node [/rexrov/velocity_control]
Publications:
 * /rexrov/cmd_accel [geometry_msgs/Accel]
 * /rexrov/velocity_control/parameter_descriptions [dynamic_reconfigure/ConfigDescription]
 * /rexrov/velocity_control/parameter_updates [dynamic_reconfigure/Config]
 * /rosout [rosgraph_msgs/Log]

Subscriptions:
 * /rexrov/cmd_vel [geometry_msgs/Twist]
 * /rexrov/pose_gt [unknown type]

Services:
 * /rexrov/velocity_control/get_loggers
 * /rexrov/velocity_control/set_logger_level
 * /rexrov/velocity_control/set_parameters
```

This node has four publishers of message type geometry_msgs, dynamic_reconfigure, and rosgraph_msgs. The /rexrov/cmd_accel, /rexrov/velocity_control/parameter_updates, and /rexrov/velocity_control/parameter_descriptions topics are only a publisher to this node. This node is subscribed to two topics and has three services of get_loggers and set_loggers_level.

### /rexrov/urdf_spawner

```
mourad@ubuntumourad:~/catkin_ws$ rosnode info /rexrov/urdf_spawner
--------------------------------------------------------------------------------
Node [/rexrov/urdf_spawner]
Publications:
 * /rosout [rosgraph_msgs/Log]

Subscriptions: None

Services:
 * /rexrov/urdf_spawner/get_loggers
 * /rexrov/urdf_spawner/set_logger_level


contacting node http://ubuntumourad:42991/ ...
Pid: 24765
Connections:
 * topic: /rosout
    * to: /rosout
    * direction: outbound (41931 - 127.0.0.1:35000) [9]
    * transport: TCPROS

mourad@ubuntumourad:~/catkin_ws$
```

This node has one publisher of message type rosgraph_msgs. This node is subscribed to no topics and has two services of get_loggers and set_loggers_level.

### /rexrov/joystick

```
mourad@ubuntumourad:~/catkin_ws$ rosnode info /rexrov/joystick
--------------------------------------------------------------------------------
Node [/rexrov/joystick]
Publications:
 * /diagnostics [diagnostic_msgs/DiagnosticArray]
 * /rexrov/joy [sensor_msgs/Joy]
 * /rosout [rosgraph_msgs/Log]

Subscriptions:
 * /rexrov/joy/set_feedback [unknown type]

Services:
 * /rexrov/joystick/get_loggers
 * /rexrov/joystick/set_logger_level
```

This node has three publishers of message type diagnostic_msgs, sensor_msgs, and rosgraph_msgs. The /diagnostics and /rexrov/joy topics are only a publisher to this node. This node is subscribed to only one topic and has two services of get_loggers and set_loggers_level.

### /rexrov/thruster_allocator

```
mourad@ubuntumourad:~/catkin_ws$ rosnode info /rexrov/thruster_allocator
--------------------------------------------------------------------------------
Node [/rexrov/thruster_allocator]
Publications:
 * /rexrov/thrusters/0/input [uuv_gazebo_ros_plugins_msgs/FloatStamped]
 * /rexrov/thrusters/1/input [uuv_gazebo_ros_plugins_msgs/FloatStamped]
 * /rexrov/thrusters/2/input [uuv_gazebo_ros_plugins_msgs/FloatStamped]
 * /rexrov/thrusters/3/input [uuv_gazebo_ros_plugins_msgs/FloatStamped]
 * /rexrov/thrusters/4/input [uuv_gazebo_ros_plugins_msgs/FloatStamped]
 * /rexrov/thrusters/5/input [uuv_gazebo_ros_plugins_msgs/FloatStamped]
 * /rexrov/thrusters/6/input [uuv_gazebo_ros_plugins_msgs/FloatStamped]
 * /rexrov/thrusters/7/input [uuv_gazebo_ros_plugins_msgs/FloatStamped]
 * /rosout [rosgraph_msgs/Log]

Subscriptions:
 * /rexrov/thruster_manager/input [geometry_msgs/Wrench]
 * /rexrov/thruster_manager/input_stamped [unknown type]
 * /tf [tf2_msgs/TFMessage]
 * /tf_static [tf2_msgs/TFMessage]

Services:
 * /rexrov/thruster_allocator/get_loggers
 * /rexrov/thruster_allocator/set_logger_level
 * /rexrov/thruster_allocator/tf2_frames
 * /rexrov/thruster_manager/get_config
 * /rexrov/thruster_manager/get_thruster_curve
 * /rexrov/thruster_manager/get_thrusters_info
 * /rexrov/thruster_manager/set_config
```

This node has nine publishers of message type uuv_gazebo_ros_plugins_msgs It is subscribes to four topics and has seven services.

# ROSPARAM:

```
raminudash@ubuntu:~$ rosparam list
/rexrov/acceleration_control/tf_prefix
/rexrov/ground_truth_to_tf_rexrov/child_frame_id
/rexrov/ground_truth_to_tf_rexrov/footprint_frame_id
/rexrov/ground_truth_to_tf_rexrov/frame_id
/rexrov/ground_truth_to_tf_rexrov/odometry_topic
/rexrov/ground_truth_to_tf_rexrov/stabilized_frame_id
/rexrov/joy_uuv_velocity_teleop/deadman_button
/rexrov/joy_uuv_velocity_teleop/exclusion_buttons
/rexrov/joy_uuv_velocity_teleop/mapping/pitch/axis
/rexrov/joy_uuv_velocity_teleop/mapping/pitch/gain
/rexrov/joy_uuv_velocity_teleop/mapping/roll/axis
/rexrov/joy_uuv_velocity_teleop/mapping/roll/gain
/rexrov/joy_uuv_velocity_teleop/mapping/x/axis
/rexrov/joy_uuv_velocity_teleop/mapping/x/gain
/rexrov/joy_uuv_velocity_teleop/mapping/y/axis
/rexrov/joy_uuv_velocity_teleop/mapping/y/gain
/rexrov/joy_uuv_velocity_teleop/mapping/yaw/axis
/rexrov/joy_uuv_velocity_teleop/mapping/yaw/gain
/rexrov/joy_uuv_velocity_teleop/mapping/z/axis
/rexrov/joy_uuv_velocity_teleop/mapping/z/gain
/rexrov/joy_uuv_velocity_teleop/type
/rexrov/joystick/autorepeat_rate
/rexrov/joystick/dev
/rexrov/pid/inertial/ixx
/rexrov/pid/inertial/ixy
/rexrov/pid/inertial/ixz
/rexrov/pid/inertial/iyy
/rexrov/pid/inertial/iyz
/rexrov/pid/inertial/izz
/rexrov/pid/mass
/rexrov/robot_description
/rexrov/robot_state_publisher/publish_frequency
/rexrov/robot_state_publisher/robot_description
/rexrov/thruster_allocator/tam
/rexrov/thruster_manager/base_link
/rexrov/thruster_manager/conversion_fcn
/rexrov/thruster_manager/conversion_fcn_params/gain
/rexrov/thruster_manager/max_thrust
/rexrov/thruster_manager/tf_prefix
/rexrov/thruster_manager/thruster_frame_base
/rexrov/thruster_manager/thruster_topic_prefix
/rexrov/thruster_manager/thruster_topic_suffix
/rexrov/thruster_manager/timeout
/rexrov/thruster_manager/update_rate
/rexrov/velocity_control/angular_d
/rexrov/velocity_control/angular_i
/rexrov/velocity_control/angular_p
/rexrov/velocity_control/angular_sat
/rexrov/velocity_control/linear_d
/rexrov/velocity_control/linear_i
/rexrov/velocity_control/linear_p
/rexrov/velocity_control/linear_sat
/rexrov/velocity_control/odom_vel_in_world
/rosdistro
/roslaunch/uris/host_ubuntu__37643
/roslaunch/uris/host_ubuntu__46049
/rosversion
/run_id
raminudash@ubuntu:~$
```

We use the command $ rosparam list to inspect and list all the currently used parameters. It works in a similar fashion to $ rostopic list. Parameters are addressed by node name, parameter name, node namespace, and parameter namespace(optional). Parameters are used to configure nodes at startup and during runtime. The lifetime of a parameter is tied to the lifetime of the node.

# Services:


We use the command $ rosservice list to view the list of services. Using the command $ rosservice info (Service_name) gives us information on , the node, the type and the args(types of arguments that are sent). Services are another way to pass data between nodes in ROS. They enable one node to call a function that executes in a different node.

# Task 2: Controlling the robot using teleop_twist_keyboard:

Ramin 50% | Zain: 30% | Mourad 20%

---

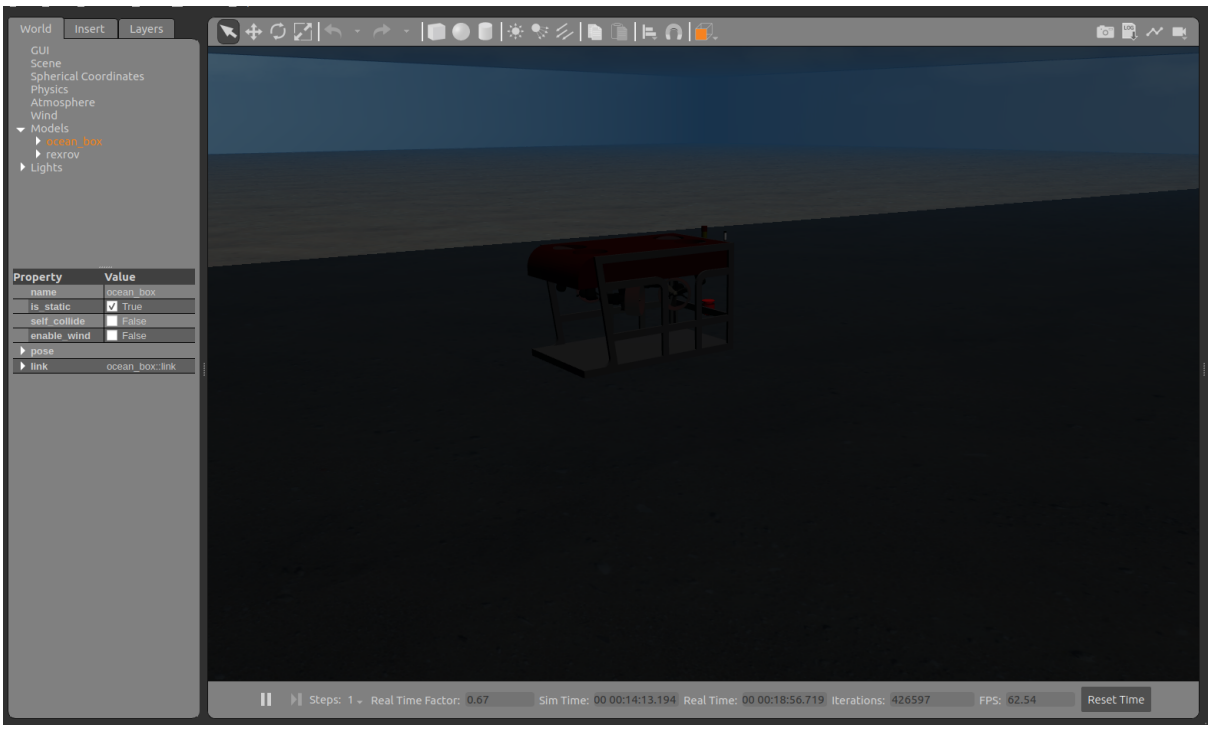
## Part A:

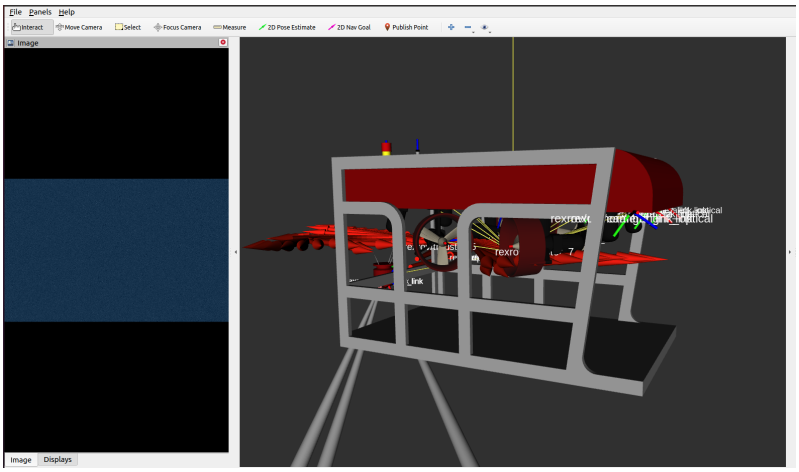
Since we have to spawn `rexrov_default.launch` file within the empty_underwater_world in uuv_gazebo_worlds and use the teleop twist keyboard to control the movements of the rexrov, we use the launch file that looks like follows:

```
<launch>

    <!-- argument cmd_vel to provide commands to the teleop twist keyboard-->
  <arg name = "cmd_vel" default="rexrov/cmd_vel"/>

    <!-- including the rexrov from rexrov_default.launch-->
  <include file="$(find uuv_gazebo)/launch/rexrov_demos/rexrov_default.launch">
  </include>

      <!-- including the empty underwater world from uuv_gazebo_worlds -->
   <include file="$(find uuv_gazebo_worlds)/launch/empty_underwater_world.launch">
  </include>


     <!-- including node teleop twist keyboard-->
  <node pkg="teleop_twist_keyboard" type="teleop_twist_keyboard.py"
   name="teleop" output="screen">

     <!-- maps command from keyboard to the cmd_vel to control the robot-->
    <remap from="cmd_vel" to="$(arg cmd_vel)"/>
  </node>

</launch>]
```

The above file can be found at:
https://github.com/ramin443/RISLAB-project/blob/main/Part%202/launch_rexrov.launch

After launching the launch file via roslaunch launch_rexrov.launch, we were able to control the rexrov via teleop twist keyboard using commands from the keyboard.
The rexrov within the Gazebo looked as in the images below:



In the image above, the rexrov is the one on the right and the camera feed under the water in the image on the left

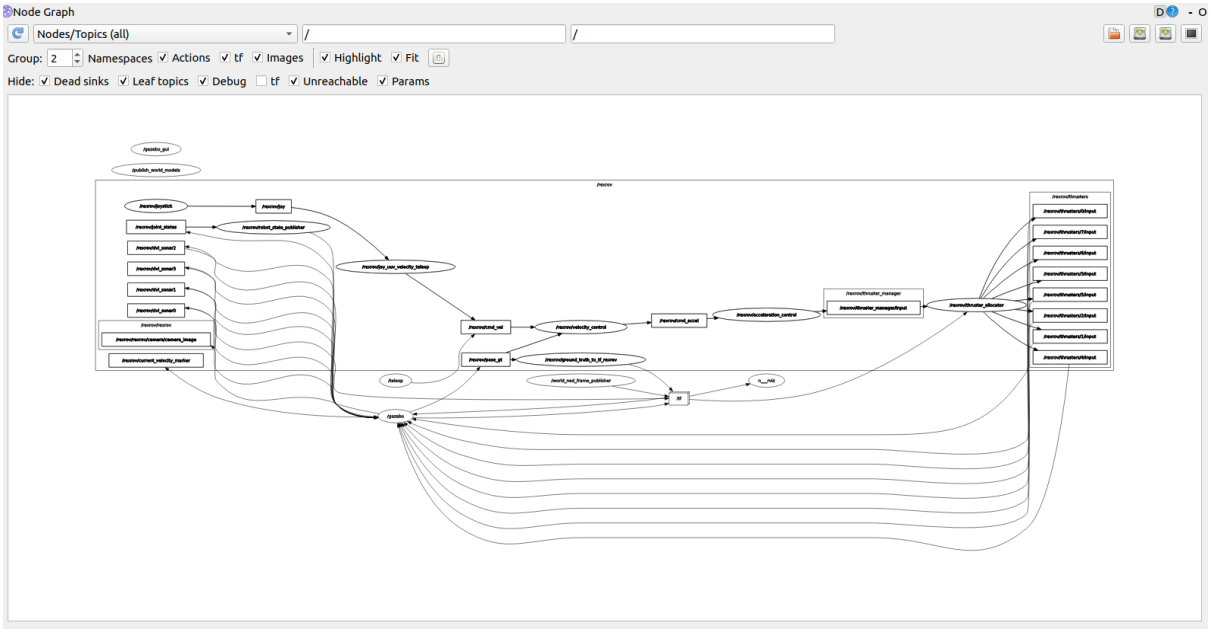
## Part B:

We use **rosbag** to log the movements of the rexrov carried out using teleop_twist_keyboard. The following commands were executed in the following order:

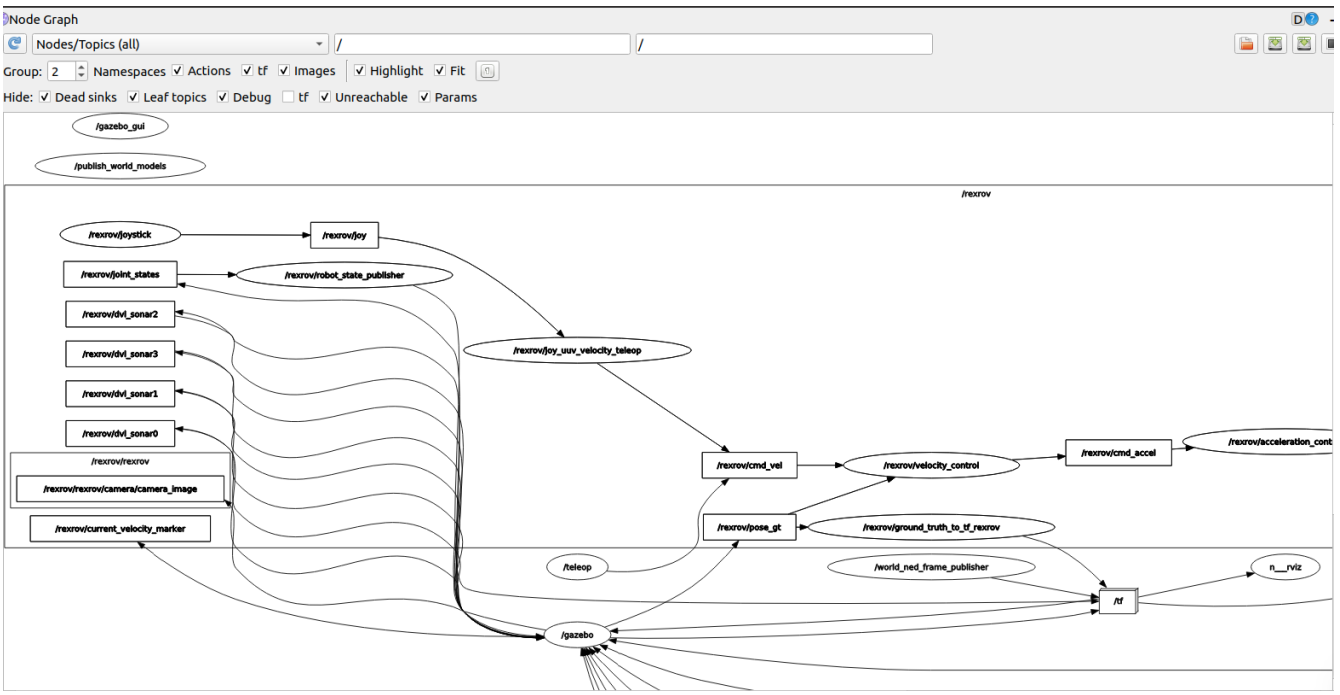
- $ mkdir bagfiles
- $ cd bagfiles
- $ rostopic list (to find out the topics active and filter the one for cmd_vel)
- $ rosrun teleop_twist_keyboard teleop_twist_keyboard.py cmd_vel:=/rexrov/cmd_vel (in alternate terminal)

- $ rosbag record -a
- $ rosbag play <filename>
- $ rosrun rqt_graph rqt_graph

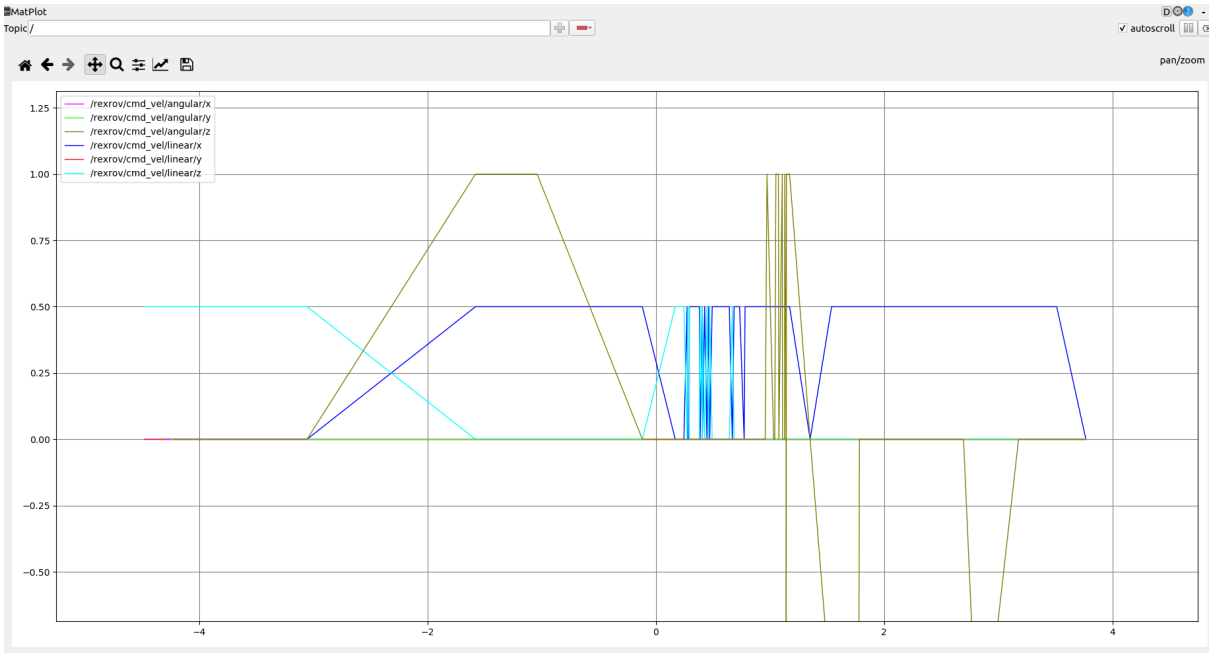
The screenshot of the rqt_plot after executing the command rosbag play <filename> i.e. replaying the file that was recorded is given below:




The graph has been zoomed below:



`To confirm that data is indeed being shared after we try to recreate the movements of the rexrov using rosbag play <logfile.bag>, we plot the cmd_vel messages using rqt_plot using `rosrun rqt_plot rqt_plot /rexrov/cmd_vel/linear:angular` and the screenshot is given below:



# **Task 3:** Output forces and torques as input to control AUV:

Zain: 50% | Ramin: 30% | Mourad: 20%

---

## Part A:

In this section, we created a ROS node similar to teleop_twist_keyboard to publish forces and torques enabling the robot to move at a certain rate. We used a python file from the simulator, found in the the uuv_teleop/scripts/vehicle_keyboard_teleop.py directory as a point of reference for the python code in our node. We created a Rospy Publisher object such that the node must publish to the Wrench message type topic, "/rexrov/thruster_manager/input".

We use a terminal window to display a simple UI that takes input form the keyboard which in turn results in thruster activation causing the robot to move. The program inside the terminal window uses keystrokes to set a speed value. They correspond to torque in x,y and z directions as well as positive and negative directions along the x, y and z axes.

We use the following keyboard keys to execute operation like(similar to the one used for teleop twist keyboard):

```
Control Your Vehicle!
---------------------------
Moving around:
    W/S: X-Axis
    A/D: Y-Axis
    X/Z: Z-Axis
    Q/E: Yaw
    I/K: Pitch
    J/L: Roll
Slow / Fast: 1 / 2
CTRL-C to quit
        """
```

Eg: W is for the movement through the positive X-axis and S is for the negative X-axis.
Upon completing each keystroke, the force and torque data is published as a message.

We create a ROS publisher that publishes to the topic /rexrov/thruster_manager/input with the message type as Wrench.

```
        if self._msg_type == 'wrench':
            self._output_pub = rospy.Publisher('/rexrov/thruster_manager/input', Wrench,
queue_size=10)
```

To store the force and torque values within the python file, we create two 3 dimensional vectors called forces and torques.

```
        else:
            # If no button is pressed reset forces and torque to 0
            self.l = Vector3(0, 0, 0)
            self.a = Vector3(0, 0, 0)

        # Store force and torque message into Wrench format
        cmd.torque = self.a
        cmd.force = self.l
```

Everytime a key is read from the keyboard, we publish the message containing the force and torque data.

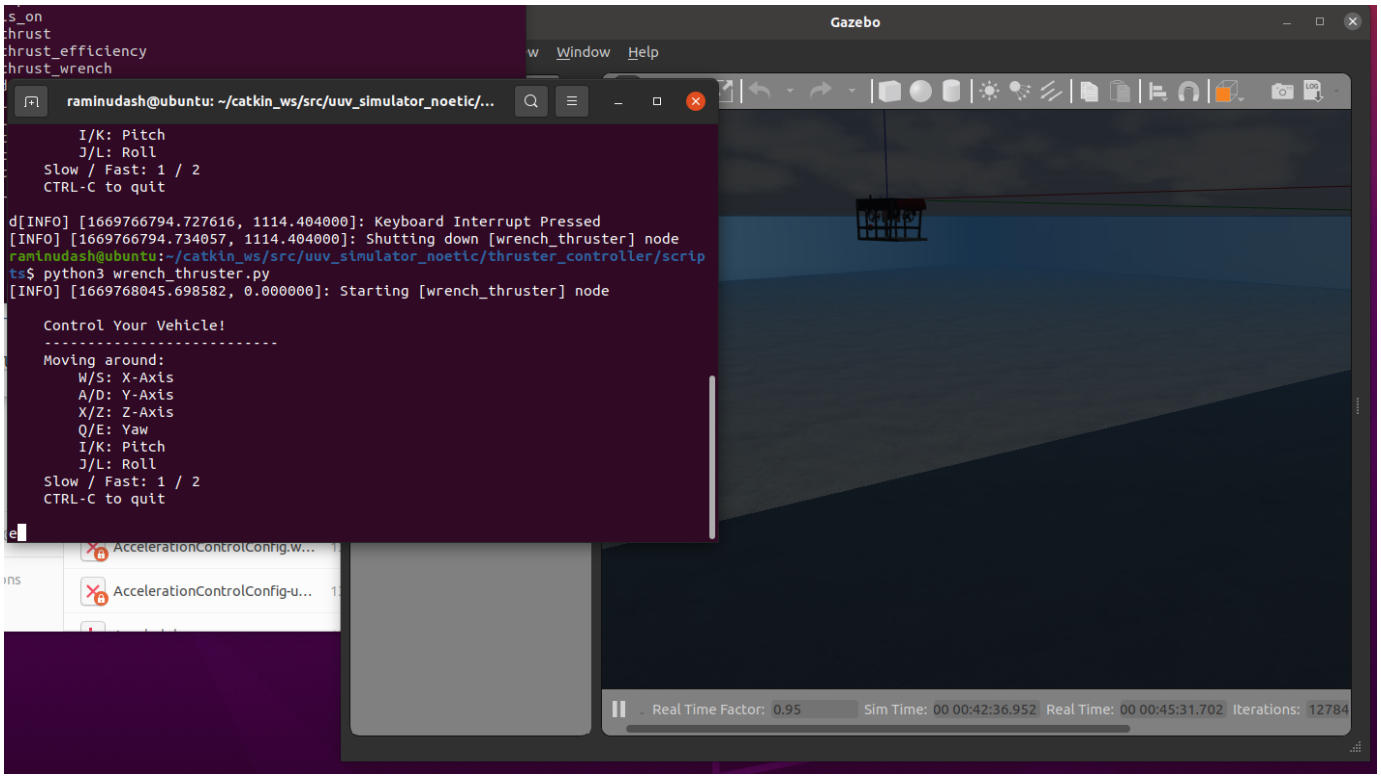
```
        # Publish force and torque data
        self._output_pub.publish(cmd)
```

The python file can be found at:
https://github.com/ramin443/RISLAB-project/blob/main/Part%203/thruster_controller/scripts/wrench_thruster.py

To launch this (without the launch file), the following commands were used:

```
// launch the empty underwater world from gazebo worlds
$ roslaunch uuv_gazebo_worlds empty_underwater_world.launch

// spawn the rexrov from rexrov_default to uuv_descriptions into the underwater world
$ roslaunch uuv_descriptions upload_rexrov_default.launch

// run the wrench controller node
$ rosrun thrust_controller wrench_controller
```

After running the commands above, we get a screen like this and are able to move the rexrov with the aforementioned keyboard commands.



## Part B:

In this section, we write the launch file to execute all the commands above that looks like this:

```
<launch>

  <!-- setting arguments-->
  <arg name = "cmd_vel" default="rexrov/cmd_vel"/>
  <arg name="launch_rviz" default="1"/>

  <node pkg="thruster_controller" type="wrench_thruster.py" name="thruster" output="screen">
      <remap from="cmd_vel" to="$(arg cmd_vel)"/>
 </node>

   <!-- including the empty underwater world from uuv_gazebo_worlds -->
  <include file="$(find uuv_gazebo_worlds)/launch/empty_underwater_world.launch">
  </include>

  <!-- including the rexrov from rexrov_wrench_control.launch-->
  <include file="$(find uuv_gazebo)/launch/rexrov_demos/rexrov_wrench_control.launch">
  </include>


  <group if="$(arg launch_rviz)">
  <node name="rviz" pkg="rviz" type="rviz" output="screen" args="-d $(find uuv_gazebo)/rviz/-
rexrov_default.rviz" />
  </group>
</launch>
```

The above file can be found at:
https://github.com/ramin443/RISLAB-project/blob/main/Part%203/wrench_controller.launch

# **Task 4:** Creating and controlling your own robot:

Mourad: 40% | Zain: 30% | Ramin: 30%

---

For this task we create our own robot by creating a urdf model, we then launched and controled it in the underwater world.

## Part A:

For the robot's design, we opted to use a simple rectangle as our base link. We used a total of 6 links in connecting the other parts of the robot as can be viewed below. Each link contains a collision, visual and inertial tag. During the ideation phase, we brainstormed a plethora of thruster positions. The thrusters were placed to prioritize mobility. We used the collision and inertial tags in order to load the model into gazebo. The collision tag describes the collision space for each objects. This looks identical to the visual element. The inertia tag holds the mass and the inertia tag. Gazebo uses a 3x3 rotational inertia matrix contained in the Inetia tag. This allows us to simulate the underwater physics of robot in the underwater world.

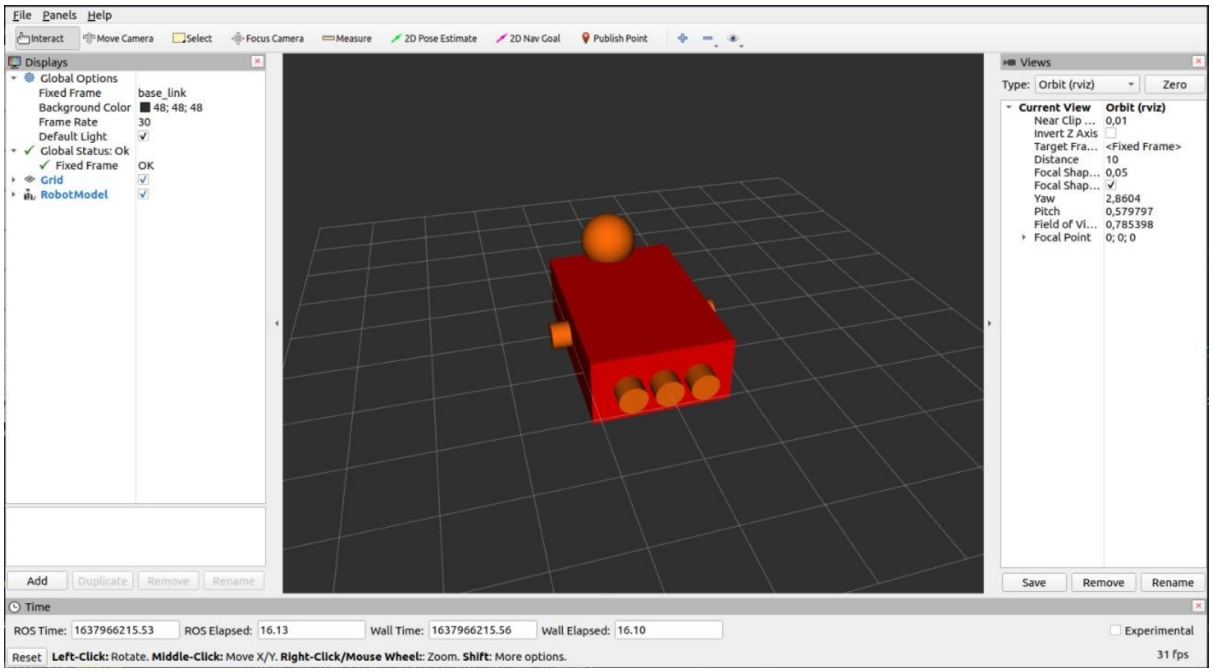
Once the design was finalized, we then worked toward creating our URDF model using RVIZ:

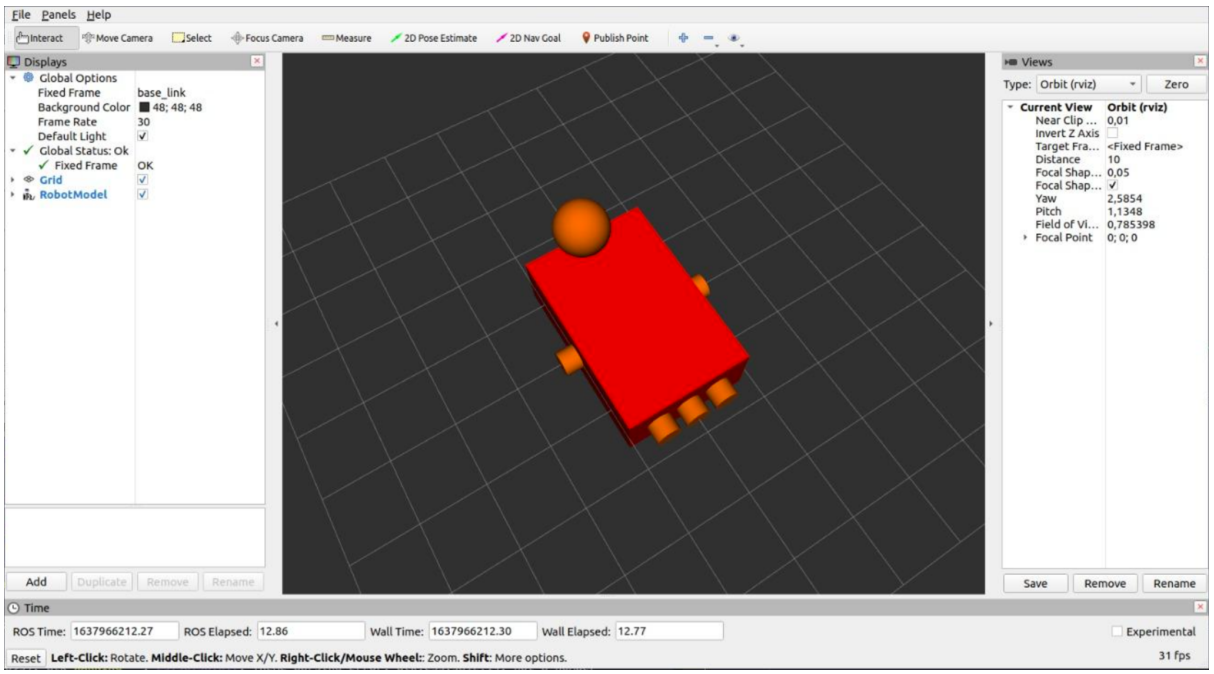
```
<?xml version="1.0" ?>
<robot name="robot_rov" xmlns:xacro="http://www.ros.org/wiki/xacro">

  <material name="orange">
      <color rgba="1.0 0.423529411765 0.0392156862745 1.0"/>
  </material>
   <gazebo reference="base_link">
      <material>Gazebo/Orange</material>
  </gazebo>

  <link name="base_link">
  <inertial>
    <mass value="0.2"/>
    <origin rpy="0 1.5707 1.5707" xyz="0 0 0"/>
    <inertia ixx="0.000526666666667" ixy="0" ixz="0" iyy="0.000526666666667" iyz="0" izz="0.001"/>
  </inertial>
  <collision name="base_link_collision">
    <origin rpy="0 1.5707 1.5707" xyz="0 0 0"/>
    <geometry>
      <cylinder length="0.04" radius="0.1"/>
    </geometry>
  </collision>
      <visual>
          <origin rpy="0 0 0" xyz="0 0 0"/>
          <geometry>
              <box size="3 2 1"/>
          </geometry>
      </visual>

  </link>
  <!-- joint that connects base_link to link_01 -->
  <joint name="base_link_link_01" type="revolute">
      <axis xyz="0 0 1" />
      <limit effort="1000.0" lower="-3.14" upper="3.14" velocity="0.5" />
      <origin rpy="0 0 0" xyz="1.5 0.0 0.7"/>
      <parent link="base_link"/>
      <child link="Control_01"/>
  </joint>
```

```
   <link name="Control_01">
       <visual>
           <geometry>
                   <sphere radius="0.45"/>
           </geometry>
       <material name="orange"/>
       </visual>
       <collision>
           <geometry>
               <sphere radius="0.45"/>
       </geometry>
       </collision>
       <gazebo reference="Control_01">
           <material>Gazebo/Blue</material>
       </gazebo>
       <inertial>
           <mass value="1"/>
           <inertia ixx="0.000526666666667" ixy="0" ixz="0" iyy="0.000526666666667" iyz="0"
izz="0.001"/>
       </inertial>
   </link>
   <joint name= "Control_Module" type="fixed">
       <parent link="base_link"/>
       <child link="Control_01"/>
       <origin xyz="0.5 0.0 0.0"/>
   </joint>
   <xacro:macro name="thruster" params="suffix copy_x copy_y copy_z rot_x rot_y rot_z">
       <link name="thruster_${suffix}">
           <visual>
               <geometry>
               <cylinder radius="0.2" length="0.35"/>
               </geometry>
               <material name="orange"/>
           </visual>
           <collision>
               <geometry>
               <cylinder radius="0.2" length="0.35"/>
               </geometry>
           </collision>
           <inertial>
               <mass value="0.5"/>
                    <inertia ixx="0.000526666666667" ixy="0" ixz="0" iyy="0.000526666666667" iyz="0"
izz="0.001"/>
           </inertial>
       </link>
       <gazebo reference="thruster_${suffix}">
           <material>Gazebo/Red</material>
       </gazebo>
       <joint name="contact_2_thruster_${suffix}" type="prismatic">
            <parent link="base_link"/>
               <child link="thruster_${suffix}"/>
                <limit effort="1000.0" lower="-0.38" upper="0" velocity="0.5"/>
                  <origin rpy="${rot_x} ${rot_y} ${rot_z}" xyz="${copy_x} ${copy_y} ${copy_z}"/>
       </joint>

   </xacro:macro>
   <xacro:thruster suffix="a" copy_x="-1.6" copy_y="0" copy_z="0" rot_x="0" rot_y="1.5757" rot_z="0"/>
   <xacro:thruster suffix="b" copy_x="-1.6" copy_y="-0.5" copy_z="0" rot_x="0" rot_y="1.5757"
rot_z="0"/>
   <xacro:thruster suffix="c" copy_x="-1.6" copy_y="0.5" copy_z="0" rot_x="0" rot_y="1.5757" rot_z="0"/>
   <xacro:thruster suffix="d" copy_x="0" copy_y="1.1" copy_z="0" rot_x="1.5757" rot_y="0" rot_z="0"/>
```

```
    <xacro:thruster suffix="e" copy_x="0" copy_y="-1.1" copy_z="0" rot_x="1.5757" rot_y="0" rot_z="0"/>
    <xacro:thruster suffix="f" copy_x="0" copy_y="0" copy_z="-0.6" rot_x="0" rot_y="0" rot_z="0"/>

<gazebo>
    <plugin name="gazebo_ros_control" filename="libgazebo_ros_control.so">
    <robotNamespace>/</robotNamespace>
    </plugin>
</gazebo>
</robot>
```

We can see the simulation of the ROV in the below images
After much planning and discussion, we came up with the below design



## Part B:

As for our thruster placement, The ROV has 6 thrusters:

- Three thruster for forward and backward motion
- Two thrusters on the side for moving left and right without tilt
- One thruster in the front to allow the robot to dive and emerge (Pitch motion)

The robot is able to achieve the following movements:

- Yaw
- Front and backwards thrust
- Left\right sideways thrust
- Dive
- Emerge
- Pitch

We chose this design and placement for the thrusters as it allows focuses on optimal mobility.
These are the xacros for the thruster rotations:

```
90      <joint name="contact_2_thruster_${suffix}" type="prismatic">
91          <parent link="base_link"/>
92            <child link="thruster_${suffix}"/>
93            <limit effort="1000.0" lower="-0.38" upper="0" velocity="0.5"/>
94              <origin rpy="${rot_x} ${rot_y} ${rot_z}" xyz="${copy_x} ${copy_y} ${copy_z}"/>
95      </joint>
96
97  </xacro:macro>
98  <xacro:thruster suffix="a" copy_x="-1.6" copy_y="0" copy_z="0" rot_x="0" rot_y="1.5757" rot_z="0"/>
99  <xacro:thruster suffix="b" copy_x="-1.6" copy_y="-0.5" copy_z="0" rot_x="0" rot_y="1.5757" rot_z="0"/>
100 <xacro:thruster suffix="c" copy_x="-1.6" copy_y="0.5" copy_z="0" rot_x="0" rot_y="1.5757" rot_z="0"/>
101 <xacro:thruster suffix="d" copy_x="0" copy_y="1.1" copy_z="0" rot_x="1.5757" rot_y="0" rot_z="0"/>
102 <xacro:thruster suffix="e" copy_x="0" copy_y="-1.1" copy_z="0" rot_x="1.5757" rot_y="0" rot_z="0"/>
103 <xacro:thruster suffix="f" copy_x="0" copy_y="0" copy_z="-0.6" rot_x="0" rot_y="0" rot_z="0"/>
104
```

## Part C:

We created our own node using rospy. We used our work from task 3 and subscribed to the topic, /rexrov/thruster manager/input. In order to make our conversions to thrust commands capable of being executed in gazebo, we relied on values obtains for force and torque from the topic: /rexrov/thruster manager/input. and finally utilized the service apply_body_wrench.

The below python file includes comment which further explain the working of the our node:
https://github.com/ramin443/RISLAB-project/blob/main/Part%204/command_thrust/scripts/thrust_commander.py

## Part D(I):

The ROV file was launched into UUV gazebo by adding inertials, masses and gazebo colors.
Use the roslaunch simulation test_gazebo.launch command after sourcing the folder simulation.

PS: The ROV does not spawn perfectly in the world.

The ***test_gazebo.launch*** launch file:

```
<?xml version="1.0" encoding="UTF-8"?>
<launch>
```

```
   <include file="$(find uuv_gazebo_worlds)/launch/empty_underwater_world.launch">
    </include>
   <param name="robot_description" command="cat '$(find simulation)/urdf/test.urdf'" />

   <node name="mybot_spawn" pkg="gazebo_ros" type="spawn_model" output="screen"
         args="-urdf -param robot_description -model robot_rov" />

</launch>
```

## Part D (ii - iii)

Here we created the launch file in a way to generate fake joint values to publish force and torque data on our vehicle after that the joint values were combined, which were included with the thrust commands to our ROV inside the RViz config file “test_config.rviz”.

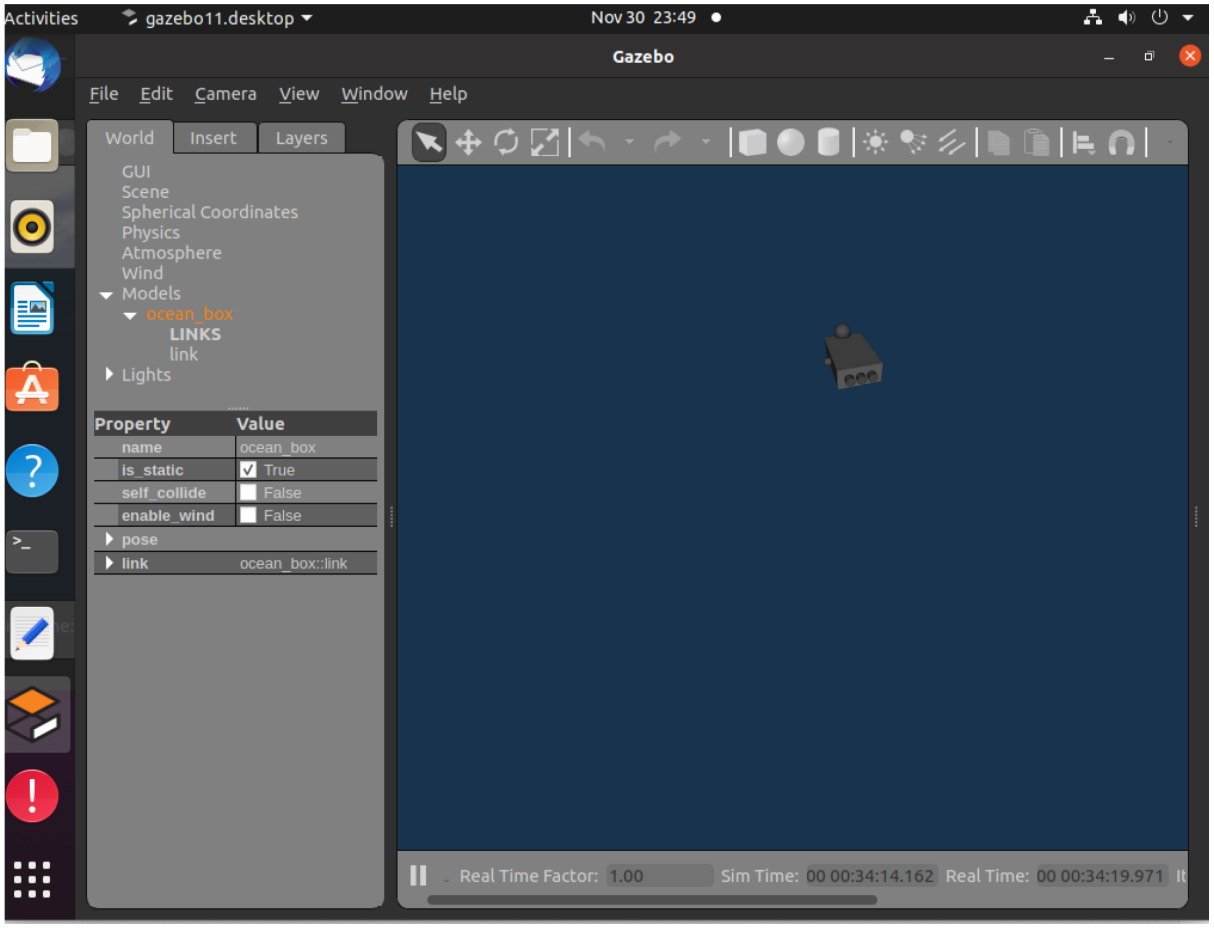
```
<launch>

   <arg name="model" />

   <param name="robot_description" command="$(find xacro)/xacro --inorder '$(find
simulation)/urdf/test.urdf'" />

   <!-- send fake joint values -->

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

       <param name="use_gui" value="True"/>

   </node>



   <!-- Combine joint values -->

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



    <!-- Show in Rviz   -->

   <node name="rviz" pkg="rviz" type="rviz" args="-d $(find
simulation)/launch/test_config.rviz"/>
```

```
</launch>
```

A glance at the content of the RViz config file, after that the launch files were called:

```
Panels:
  - Class: rviz/Displays
    Help Height: 78
    Name: Displays
    Property Tree Widget:
      Expanded:
        - /Global Options1
        - /Status1
      Splitter Ratio: 0.5
    Tree Height: 419
  - Class: rviz/Selection
    Name: Selection
  - Class: rviz/Tool Properties
    Expanded:
      - /2D Pose Estimate1
      - /2D Nav Goal1
      - /Publish Point1
    Name: Tool Properties
    Splitter Ratio: 0.5886790156364441
  - Class: rviz/Views
    Expanded:
      - /Current View1
    Name: Views
    Splitter Ratio: 0.5
  - Class: rviz/Time
    Experimental: false
    Name: Time
    SyncMode: 0
    SyncSource: ""
```

Our robot in the underwater world:




The complete file is uploaded to our github repository under https://github.com/ramin443/RISLAB-project/blob/main/Part%204/simulation/launch/test_config.rviz

Files related to part 4:

https://github.com/ramin443/RISLAB-project/tree/main/Part%204/simulation