ROS Simulation Implementation Report

Oscar Echeverria, Bogdan Belenis and Stephan Daaboul

Jacobs University of Bremen

CO-542 RIS Lab

Bilal Wehbe

November 21 2021

https://github.com/Daaboulex/RIS-Lab-1-uuv_simulator

USE THIS TO SEE ALL FILES OF CATKIN

Author Note

First paragraph: Task 1

Second paragraph: Task 2

Third paragraph: Task 3

Fourth paragraph: Task 4

Task 1

Initial Setup

To do this project,git clone uuv_simulator(https://github.com/myvname-is-D/uuv_simulator) packages and build them. You also have to set up the catkin directory. Important note, have ROS documentation on hand and take note of the used directories. If any packages are missing when running launch files, just look for the missing file and git clone it into the correct directory.

In this task we launched *uuv_gazebo/rexrov_default.launch* and later inspected the nodes, topics, services it launches and the message/service types they use to communicate.

Rosnode List

```
Command
$ rosnode list
Console Output Example
/rexrov/acceleration_control
/rexrov/ground_truth_to_tf_rexrov
/rexrov/joy_uuv_velocity_teleop
/rexrov/robot_state_publisher
/rexrov/thruster_allocator
/rexrov/urdf_spawner
/rexrov/velocity_control
/rosout
/rviz
```

Utilizing the *rosnode list* command lists all active nodes.

Rosnode Info

```
Command
$ rosnode info /rexrov/acceleration_control

Console Output Example
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://ubuntu:42143/ ...
Pid: 4091
Connections:
 * topic: /rosout
    * to: /rosout
    * direction: outbound (41859 - 127.0.0.1:34808) [14]
    * transport: TCPROS
 * topic: /rexrov/thruster_manager/input
    * to: /rexrov/thruster_allocator
    * direction: outbound (41859 - 127.0.0.1:34802) [9]
    * transport: TCPROS
 * topic: /rexrov/cmd_accel
    * to: /rexrov/velocity_control (http://ubuntu:38191/)
    * direction: inbound
    * transport: TCPROS
```

Utilizing *rosnode info <directory>* users extracts information about a node, publications and subscriptions.

Implementing this command on *acceleration_control* demonstrates its publications which are */rexrov/thruster_manager/input* and */rosout*. The subscriptions to the topics are */rexrov/thruster_manager/input* and */rosout*. Also, it is subscribed to the topics */rexrov/acceleration_control/get_loggers* and */rexrov/acceleration_control/set_logger_level*.

Rosmsg List

```
Command
$ rosnode info /rexrov/acceleration_control

Console Output Example
actionlib/TestAction
actionlib/TestActionFeedback
actionlib/TestActionGoal
actionlib/TestActionResult
actionlib/TestFeedback
actionlib/TestGoal
actionlib/TestRequestAction
actionlib/TestRequestActionFeedback
actionlib/TestRequestActionGoal
```

```
actionlib/TestRequestActionResult
...... 270 more lines of code
```

rosmsg and *rossrv* are handy command-line tools that provide reference information for developers and also serve as a powerful introspection tool for learning more about data being transmitted in ROS.

Rostopic List

```
Command
$ rostopic list

Console Output Example
/clicked_point
/initialpose
/move_base_simple/goal
/rexrov/cmd_accel
/rexrov/cmd_force
/rexrov/cmd_vel
/rexrov/current_velocity_marker
/rexrov/current_velocity_marker_array
/rexrov/dvl_sonar0
/rexrov/dvl_sonar1
/rexrov/dvl_sonar2
/rexrov/dvl_sonar3
/rexrov/ground_truth_to_tf_rexrov/euler
/rexrov/ground_truth_to_tf_rexrov/pose
/rexrov/home_pressed
/rexrov/joint_states
/rexrov/joy
...... 18 more lines of code
```

The rostopic command-line tool displays information about ROS topics. It can display a list of active topics, the publishers and subscribers of a specific topic, the publishing rate of a topic, the bandwidth of a topic, and messages published to a topic. The display of messages is configurable to output in a plotting-friendly format.

Rosservice List

```
Command
$ rosservice list

Console Output Example
/rexrov/acceleration_control/get_loggers
```

```
/rexrov/acceleration_control/set_logger_level
/rexrov/ground_truth_to_tf_rexrov/get_loggers
/rexrov/ground_truth_to_tf_rexrov/set_logger_level
/rexrov/joy_uuv_velocity_teleop/get_loggers
/rexrov/joy_uuv_velocity_teleop/set_logger_level
/rexrov/robot_state_publisher/get_loggers
/rexrov/robot_state_publisher/set_logger_level
/rexrov/thruster_allocator/get_loggers
/rexrov/thruster_allocator/set_logger_level
/rexrov/thruster_allocator/tf2_frames
...... 16 more lines of code
```

The rosservice command implements a variety of commands that let you discover which services are currently online from which nodes and further drill down to get specific information about a service, such as its type, URI, and arguments. You can also call a service directly from the command line.

Task 2

Initial Setup

In this task we created *our own launch file* and later inspected the log topic teleop_twist_keyboard/cmd_vel’ after running rosbag record. We stopped teleop_twist_keyboard to control the robot by replaying the generated rosbag. Later we created a plot of the traced trajectory of the ROV, and then we attached a screenshot of rqt_graph while the bag was playing.

Make Own Launch File

File Task2.launch

```
<launch>
  <include file="$(find
uuv_gazebo_worlds)/launch/empty_underwater_world.launch">
  <arg name="paused" value="false"/>
<!--This are the arguments we modified-->
  </include>
<!--We are calling the underwater_world launch file to get the “map”-->

  <include file="$(find
uuv_gazebo)/launch/rexrov_demos/rexrov_default.launch">
    <arg name="namespace" value="rexrov"/>
<!--This are the arguments we modified-->
  </include>
<!--We are spawning the rexrov rover inside of the underwater_world-->

  <include file="$(find uuv_teleop)/launch/uuv_keyboard_teleop.launch">
    <arg name="uuv_name" value="rexrov"/>
<!--This are the arguments we modified-->
  </include>
<!--We are launching the keyboard_teleop launch file to use wasd as
movement-->
</launch>
```

Rosbag Record/Play

```
Command
$ rosbag record /rexrov/cmd_vel
Console Output
[ INFO] [1637369661.738114572]: Subscribing to /rexrov/cmd_vel
[ INFO] [1637369661.982153610, 3.378000000]: Recording to
'2021-11-20-01-54-21.bag'.
```

Rosbag record recorded all the *cmd_vel* movements of the Rexrov rover. This can be later used after killing the *teleop_twist_keyboard* that lets you use *wasd* to move on 8 angles and *z* & *x* to move up and down.

```
Command
$ rosnode kill /rexrov/keyboard_uuv_velocity_teleop
Console Output
killing /rexrov/keyboard_uuv_velocity_teleop
killed
```

Afterwards we use the rosbag play command to replay the recorded movements, also using the rosrun *rqt_graph* and *rqt_plot to gather data*.

Rqt_plot

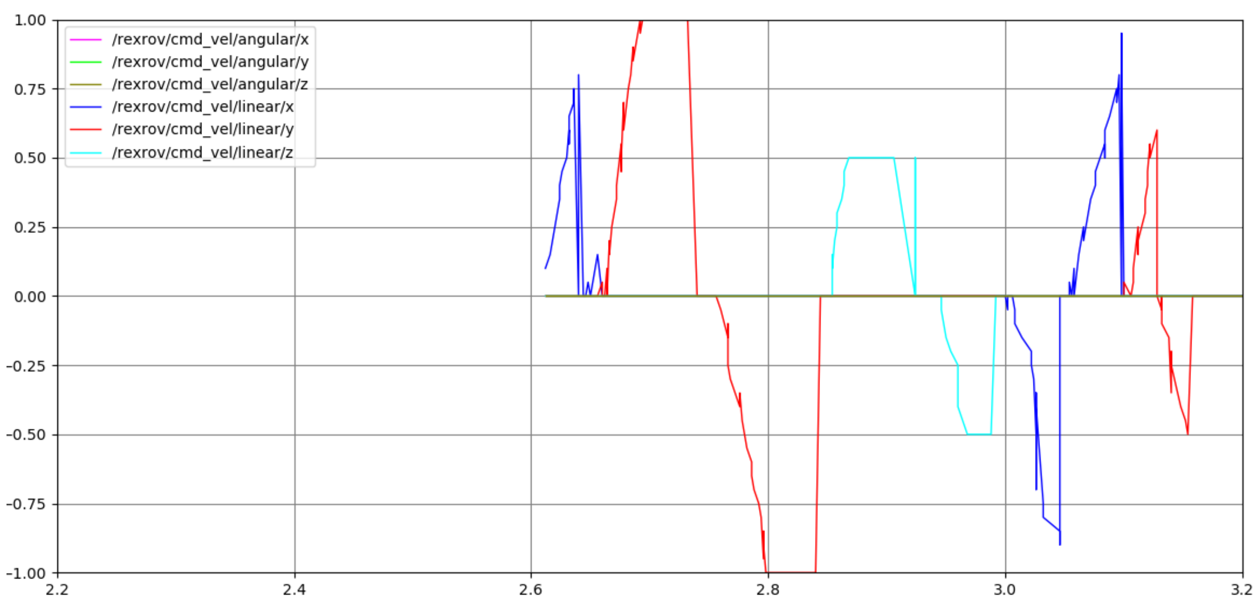


Figure 1

Rqt_plot plots all the directions of the rexrov and how much it moved. This are all angles and linear movements.

Rqt_graph

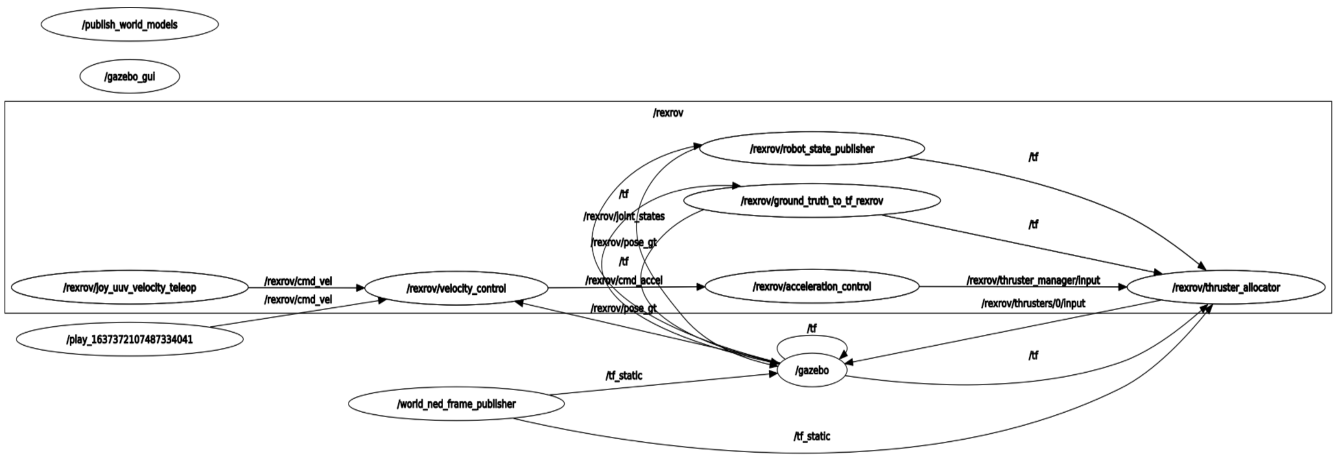


Figure 2

Rqt_graph graphs all the active nodes running at the moment as shown above. This is useful because it shows how all of them depend on each other.

Task 3

Initial Setup

In this task we created a node that outputs forces and torques as input to control the AUV in uuv_gazebo/rexrov_wrench_control.launch based on this input. You will download this file to *catkin_ws/src/uuv_simulator/uuv_gazebo/launch/rexrov_demos*. A node was written in which forces and torques are used to control the AUV. Our node must publish to the topic */rexrov/thruster_manager/input*.

Task3.launch

File Task3.launch

```
<launch>
  <arg name="uuv_name" default = "rexrov" />
  <include file = "$(find
uuv_gazebo_worlds)/launch/empty_underwater_world.launch">
      <arg name = "paused" value ="false"/>
  </include>

  <include file = "$(find
uuv_gazebo)/launch/rexrov_demos/rexrov_wrench_control.launch">
      <arg name = "namespace" value = "rexrov" />
  </include>

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

</launch>
```

This launch file initiates the empty underwater world.Then it initiates the wrench default rexrov UUV. Finally it initiates the node that we made which controls movement.

kbd_force.launch

File kbd_force.launch

```
<launch>
  <arg name="uuv_name" />
  <arg name="output_topic" default="cmd_force"/>
  <!-- The type of message can also be geometry_msgs/Accel -->
  <arg name="message_type" default="wrench"/>

  <group ns="$(arg uuv_name)">
      <node pkg="uuv_teleop" type="kbd_force.py" name="kbd_force"
output="screen">
      <remap from="output" to="/$(arg uuv_name)/$(arg
output_topic)"/>
      <rosparam subst_value="true">
      type: $(arg message_type)
      </rosparam>
      </node>
  </group>
</launch>
```

This is the launch file that initiates the kbd_force force/torque node. We put this file in the teleop/launch directory.

Snippet Kbd_force.py

```
File kbd_force.py
THIS IS A SNIPPET. APOLOGIES FOR IT HAVING TO BE LONG.

class KeyBoardVehicleTeleop:
      def __init__(self):
      #these are the variables used for the setup of the class
      self.settings = termios.tcgetattr(sys.stdin)
      # Speed setting
      self.speed = 2
      self.f = Vector3(0, 0, 0)
      self.t = Vector3(0, 0, 0)
      self.force_increment = 0.0
      self.force_limit = 1
      self.torque_increment = 0.05
      self.torque_limit = 0.5
      self._msg_type = 'wrench'
      #Publishing to /rexrov/thruster_manager/input
      if self._msg_type == 'wrench':
            self._output_pub =
rospy.Publisher('/rexrov/thruster_manager/input', Wrench,
queue_size=1)
      print(self.msg)
      # Choose ros message accordingly
      if self._msg_type == 'wrench':
            cmd = Wrench()
            # If no button is pressed reset velocities to 0
            self.f = Vector3(0, 0, 0)
            self.t = Vector3(0, 0, 0)
      # Store velocity message into wrench format
      cmd.torque = self.t
      cmd.force = self.f
      # Publish message
      self._output_pub.publish(cmd)
```

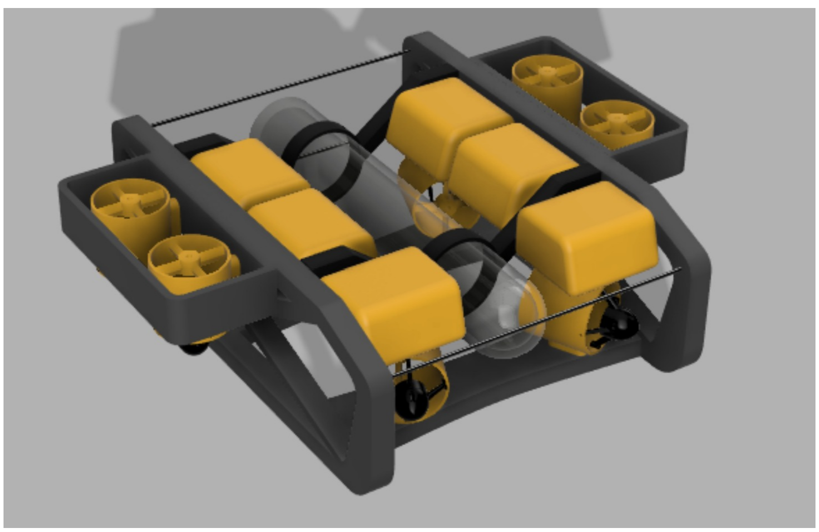
Also found in the uuv_teleop directory. This node is very similar to the teleop one that we worked on in the previous tasks, however instead of angular and linear speeds this node deals with force and torque. Force is used instead of linear movement and torque is used instead of angular speeds. We took the already existing code of the teleop and modified it to the best of our abilities to instead take force/torque, therefore some similarities between the two can be found in the code which is to be expected. Finally, another key difference is that the node publishes its data in the matrix in the thruster_manager topic found in the rexrov. In the code snippets the setup of the class and the publisher is shown. File is located in teleop scripts.

Task 4

Initial Setup

In this task we had to create and control our own robot. There were many steps to this task such as creating the URDF file for our initial design, deciding the most optimal placement for the thrusters in our Robot. Additionally, we had to write our own node for the movement that takes in torques and forces, while publishing thrust commands in a conversion tat is taking place as a service and finally writing a launch file that loads the robot in the underwater world using the node created in part 3 and the node created in earlier in the task.

Modeling of Rover

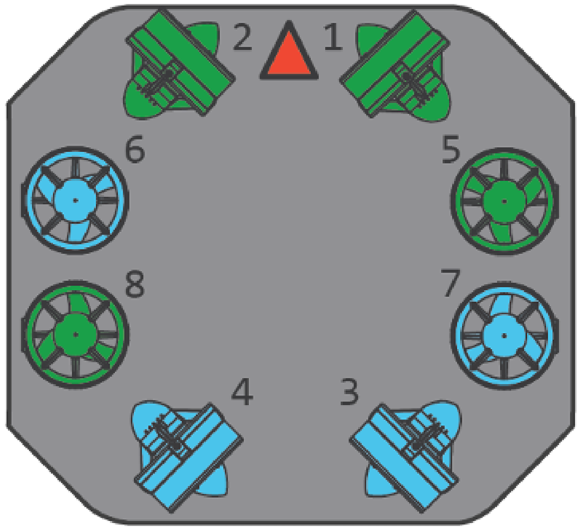


For the modeling of the robot we used fusion 360 and used fusion2urdf to make the 3D model into a URDF.

We modeled the ROV to in Fusion 360 and then we exported it to URDF using a plugin called fusion2urdf. This exports an urdf file of the 3d model.

Thruster Placement

We would place the thrusters in the following configuration:



In this way, by placing thrusters 1, 2, 3, 4 at a 45% angle, we can get high adaptability and maneuvering in the 2D plane. Thus, we can very easily yaw, move laterally, front, back, and on diagonal.

By adding thrusters 5, 6, 7, 8 we add the possibility to move vertically, but also to pitch and roll very easily. For example, for rolling the robot to the right, we can only actuate thrusters 6 and 8. Also, for pitching forward, we could actuate thrusters 7 and 8.

References

Wiki.ros.org. 2021. *ros_comm - ROS Wiki*. [online] Available at: <http://wiki.ros.org/ros_comm?distro=noetic> [Accessed Fall 2021].

GitHub. 2021. *GitHub - uuvsimulator/uuv_simulator: Gazebo/ROS packages for underwater robotics simulation*. [online] Available at: <https://github.com/uuvsimulator/uuv_simulator> [Accessed Fall 2021].