Robotics and Intelligent Systems Lab

Fall Semester 2023

Professor: Dr. Fazel Naghdy.

Authors: Harshit Mutha, Leonardo Juarez, Nadim Khneisser and Santiago Pinto (Group 8).

Before starting, python was installed with all relevant dependencies, and a package we named “rov” was created within the uuv_simulator_noetic (or uuv_simulator) folder inside the catkin workspace, the complete path to the package is “**~/catkin_ws/src/uuv_simulator_noetic/rov**”.

1) Launch *uuv_gazebo/rexrov_default.launch* and inspect the nodes, topics, services it launches and the message/service types they use to communicate.

a) **Write a short report describing these nodes, topics, services and messages including screenshots of rqt or the terminal output from which you collected this information.**

Before being able to view the nodes, topics and services, going through a few commands is necessary in order to launch the file *rexrov_default.launch*. Firstly a terminal is opened, and after going to the catkin workspace and sourcing it with the command “**source devel/setup.bash**”, “**roscore**” is typed to initialize it, which will be in charge of keeping track of all the nodes that will be running, as well as linking them by providing a way for them to communicate with each other. Secondly, a new terminal is opened and sourced similarly to the one in the previous step, but this time launching *rexrov_default.launch* with the command “**roslaunch uuv_gazebo rexrov_default.launch**”. Thirdly, launching the launch file automatically opens an Rviz interface to visualize the robot’s model in an empty space, as shown in figure 1 below. Now with the program running we can use another terminal to inspect the nodes, topics and services of interest with the commands “**rosnode list**”, “**rostopic list**”, “**rosservice list**” and “**rosmsg list**” to obtain lists 1, 2 3 and 4 (shown in the appendix section) respectively.

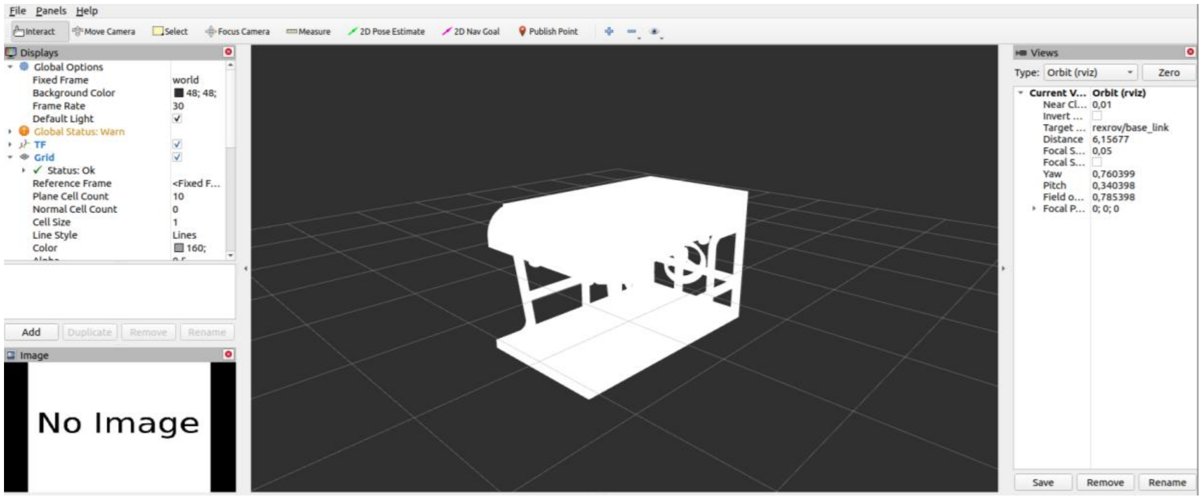

Figure 1: Rviz screenshot after launching file *rexrov_default.launch*

Starting with the nodes, running the command “**rqt_graph**” shows the following graph as figure 2. The leftmost node called “/rexrov/urdf_spawner” is in charge of spawning the robot model when the program is initialized, and node “/rexrov/joy” is in charge of taking joystick input to control the robot, furtherly shown by its direct connection to the “/rexrov/velocity_control” node via the “/rexrov/cmd_vel” topic. This topic can be furtherly inspected to find the message it transmits via running the command “**rostopic type /rexrov/cmd_vel**” and finding the message “geometry_msgs/Twist" which contains two 3-dimensional vectors, each with three float64 values indicating angular and linear velocities in all three axes. The further topic “/rexrov/cmd_accel” behaves similarly but with acceleration.

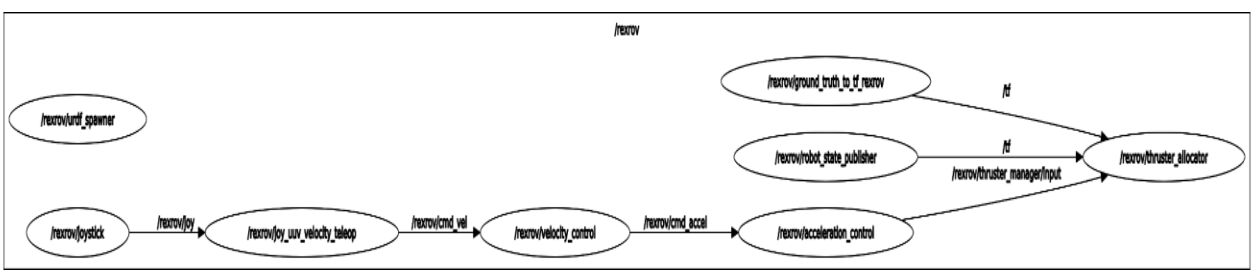

Figure 2 - Output graph after running rqt_graph

The rightmost node “/rexrov/thruster_allocator” is subscribed to the “/rexrov/thruster_manager/input” topic, which comes directly from the user input node chain (joystick node -> velocity node -> acceleration node -> thrusters node), and receives a message named “geometry_msgs/Wrench”, which contains two three dimensional vectors, each with three float64 values, indicating the force and torque that is to be applied in each of the robot’s principal axes, this is to change the robot’s linear and angular momentum respectively, increasing or decreasing its linear or angular velocity over its respective axis.

The used commands are listed in order in which they were used in the whole procedure as table 1 (shown in the appendix section), this includes specific commands and basic linux commands like “**cd**” or “**mkdir**” are omitted.

2) Controlling the robot using “teleop_twist_keyboard”:

a) Write a launch file that launches the rexrov in *uuv_gazebo/rexrov_default.launch* into the world spawned by *uuv_gazebo_worlds/empty_underwater_world.launch* and the “teleop_twist_keyboard” node in such a way that you can use the “teleop” node to control the simulated ROV. Please include comments in this launch file to explain what each line is doing.

The launch file written was named *launch_rexrov.launch* and saved in the directory “**~/catkin_ws/src/uuv_simulator_noetic/rov/launch**”, the file first spawns the world in *emtpy_underwater_world.launch* and then, after including multiple relevant launch files, it launches the “teleop” node inside the “/rexrov” namespace, it remaps the “velocity_control” node from “joy_uuv_velocity_teleop” to “teleop” to control the ROV, and launches Rviz for the model to be visualized. The node graph is shown below as figure 3.

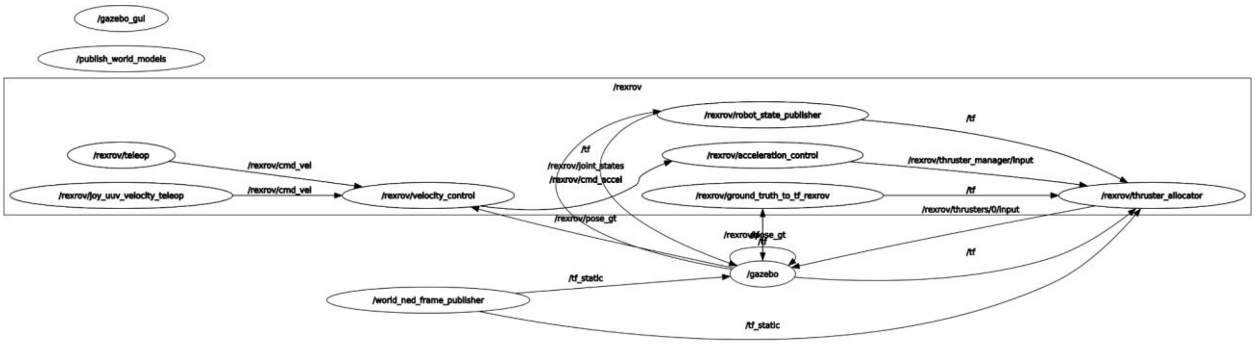

Figure 3 - Node graph while controlling the robot with “teleop” node

b) Log the topic “teleop_twist_keyboard/cmd_vel” using rosbag, move the vehicle around with your keyboard while rosbag is recording. Stop “teleop_twist_keyboard” and control the robot by replaying the generated rosbag. Create a plot of the traced trajectory of the ROV, and attach a screenshot of rqt_graph while the bag is playing.

All messages that went through topic “cmd_vel” were recorded by entering the command **“rosbag record /rexrov/cmd_vel”** in a sourced terminal. The “teleop” node was killed via **“rosnode kill /rexrov/teleop”** and then the bag was replayed by entering **“rosbag play ‘bagFileName’’**, where ‘bagFileName’ was the name of the .bag file created. In figure 3 below we see that a node outside of the “/rexrov” namespace named “/play_17004398133448517 51” was created and its publishing to the “cmd_vel” topic, so we can see that the way that rosbag replays inputs is by directly entering them as if it would be another extra node.

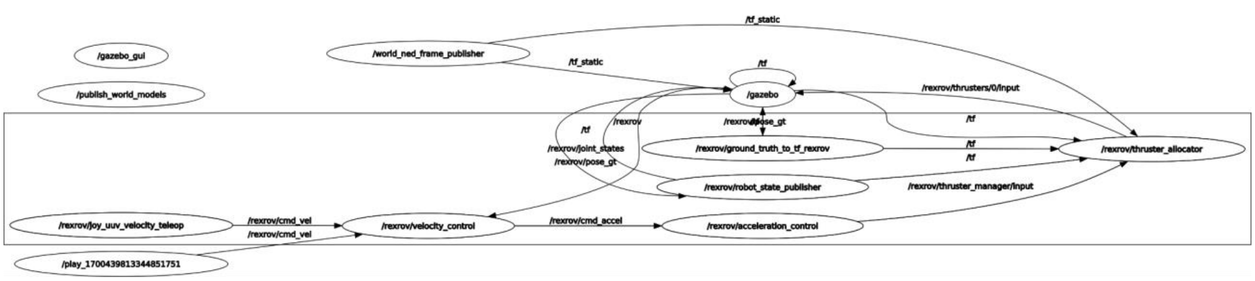

Figure 3 - Output node graph while the bag is playing

Similarly to obtain the plot of the traced trajectory of the ROV, the command **“rqt_bag”** was used, which would give you the timeline of the bag (see in figure 4) where you would need to find the “/rexrov/cmd_vel” topic. Once the topic is found we had to plot it by activating the xyz coordinates which resulted in the following graph (right side of figure 4).

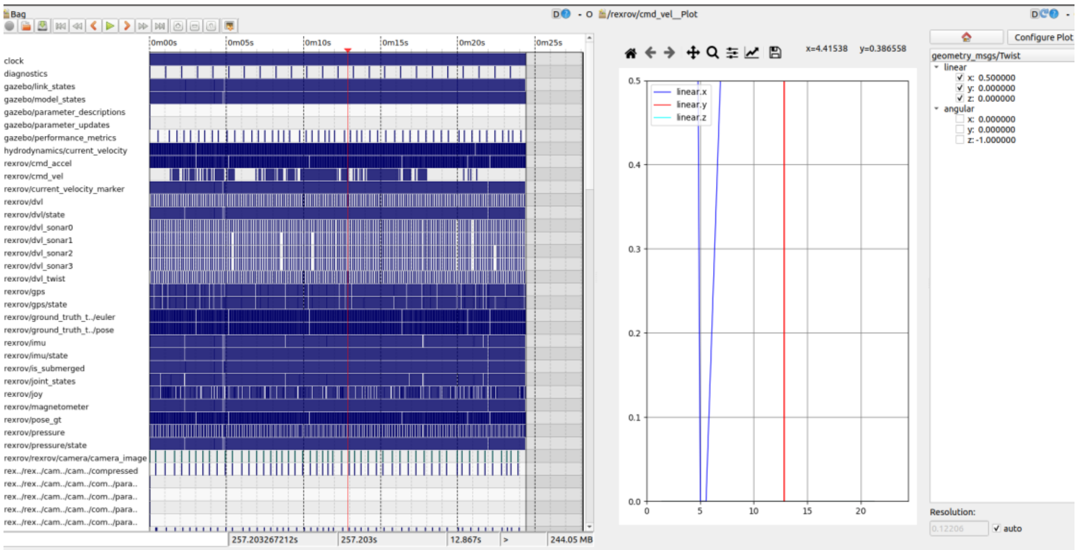

Figure 4 - Trajectory of the bot while the bag is playing (all topics shown)

3) In this exercise you need to create a node that outputs force and torques as input to control the AUV in *uuv_gazebo/rexrov_wrench_control.launch* based on this input. Download this launch file *catkin_ws/src/uuv_simulator/uuv_gazebo/launch/rexrov_demos* and launch in the world spawned by *uuv_gazebo_worlds/empty_underwater_world.launch*

a) Write a node similar to “teleop_twist_keyboard” that publishes forces and torques to control the AUV. Your node must publish to the topic “/rexrov/thruster_manager/input”.

The file *auv.py* is the script which defines the node, and the launch file *auv.launch* initializes the node called “auv_master_control”, they are found in “**~/catkin_ws/src/uuv_simulator_noetic/rov/scripts**” and “**~/catkin_ws/src/uuv_simulator_noetic/rov/launch**” respectively. The essence of the node is the publishing and processing of messages for itself, it works by combining all the control chain into one autonomous node, at the same time as it emulates the function of the “teleop_twist_keyboard” node. It starts by taking input from the keyboard and publishing to the topic “cmd_vel”, acting as “teleop_twist_keyboard” would, to then catch the Twist message (since it is also subscribed to the topic), process it, and publish it as an Accel message to the “cmd_accel” topic as the “velocity_control” node would, and finally it emulates the node “acceleration_control” to process the message and publish a wrench message to the “thruster_manager/input” topic for the model to move.

b) Write a launch file similar to the one in exercise 2 that launches the AUV and your node.

The launch file created in “**~/catkin_ws/src/uuv_simulator_noetic/rov/launch**” was named *launch_auv.launch*, and it does three things, firstly, it launches the world spawned by *empty_underwater_world.launch*, secondly it includes multiple relevant launch files including the *auv.launch* launch file to integrate the “auv_master_control” node, and thirdly it launches Rviz so the model can be visualized. After launching this launch file the graph obtained via rqt_graph was inspected to be the one presented below as figure 4, this graph was taken during runtime.

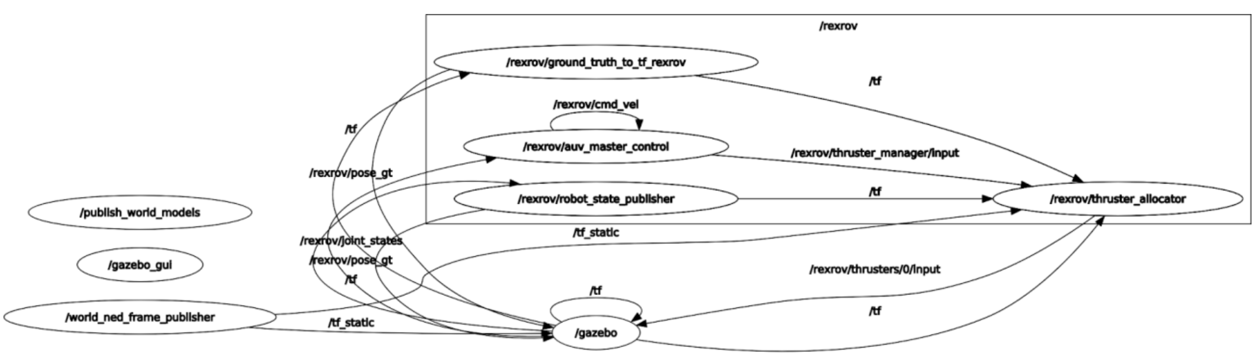

Figure 5 - Output graph during runtime after launch of “launch_auv.launch”

4) Creating and controlling your own robot:

a) **Create an urdf file that describes a simple ROV of your own design using the default geometric shapes (or design your own robot in Blender).**

We have created an urdf file of our ROV model named *ROV.udrf* along with its launch file *ROV_rviz.launch*, they are located in "**~/catkin_ws/src/uuv_simulator_noetic/rov/urdf**" and "**~/catkin_ws/src/uuv_simulator_noetic/rov/launch**" respectively.

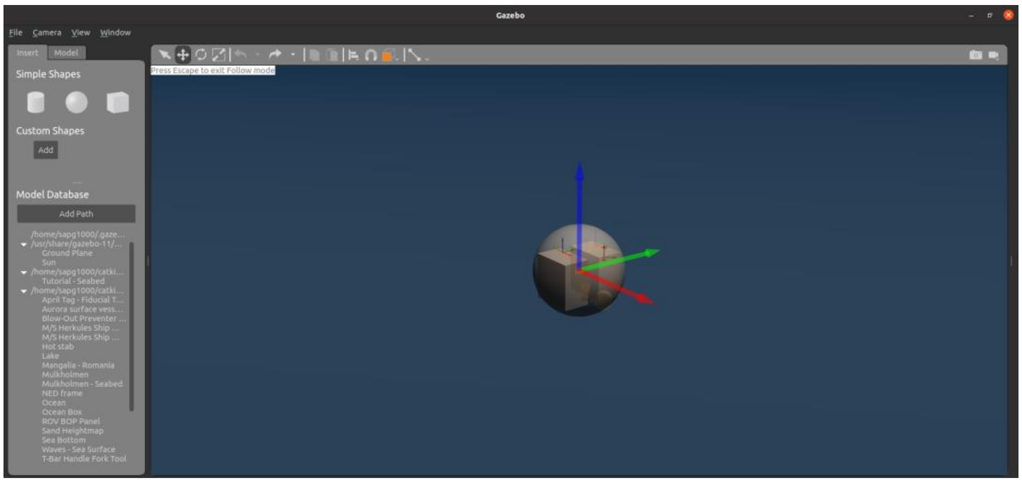

Figure 6 - Structure of the robot model seen in gazebo

b) **Decide on the placement of thrusters for this ROV and give your reasoning.**

Our ROV model incorporates eight thrusters to ensure optimal controllability. Four thrusters are strategically positioned in the base to facilitate lateral movement, set at a 45-degree angle for effective and undisturbed motion. This configuration minimizes interference between thrusters, enhancing the overall movement of the ROV. Additionally, the remaining four thrusters located on the top of the ROV contribute to vertical movement, collectively providing a powerful and versatile means of control.

c) **Write your own node that takes in forces and torques as input and publishes thrust commands for your ROV. The conversion from forces/torques to thrust commands has to be called as a service.**

The script *thrusters.py* defines a node named "thrusters" for managing the thrusters of a remotely operated vehicle (ROV). The script initializes the ROS node, subscribes to the topic “/rov/thruster_manager/input” for receiving force and torque messages, and waits for the Gazebo service “/gazebo/apply_body_wrench” to be available. The main logic involves extracting force and torque values, constructing a wrench message, and applying a perturbation to the specified body (“base_link”) in the simulated environment using the service. The script includes a **ThrustCommander** class with a callback method to update force and torque values. It also synchronizes execution based on a starting time parameter. The applied perturbation details, such as frame, duration, force, and torque, are printed for monitoring. Overall, the script facilitates the control of an ROV's thrusters in a ROS environment, interfacing with Gazebo for simulation purposes. The location for the node script file is “**~/catkin_ws/src/uuv_simulator_noetic/rov/scripts**”.

d) Write a launch file that: (i) Loads this robot into *uuv_gazebo_worlds/empty_underwater_world.launch* in gazebo. (ii) Starts the node you created in exercise 3 to publish force / torque data. (iii) Starts the node you created in part c that publishes thrust commands to your vehicle.

The launch file created was named *ROV_launcher.launch* and saved in “**~/catkin_ws/src/uuv_simulator_noetic/rov/launch**”, this node performs the aforementioned listed tasks, with an exception for the second one (ii), where it launches another node with the same functionality, not exactly the one used in the third exercise, the script defining this node is *teleop_rov.py* and it is found in “**~/catkin_ws/src/uuv_simulator_noetic/rov/scripts**”, an image of the robot in the empty underwater world is shown below as figure 7.

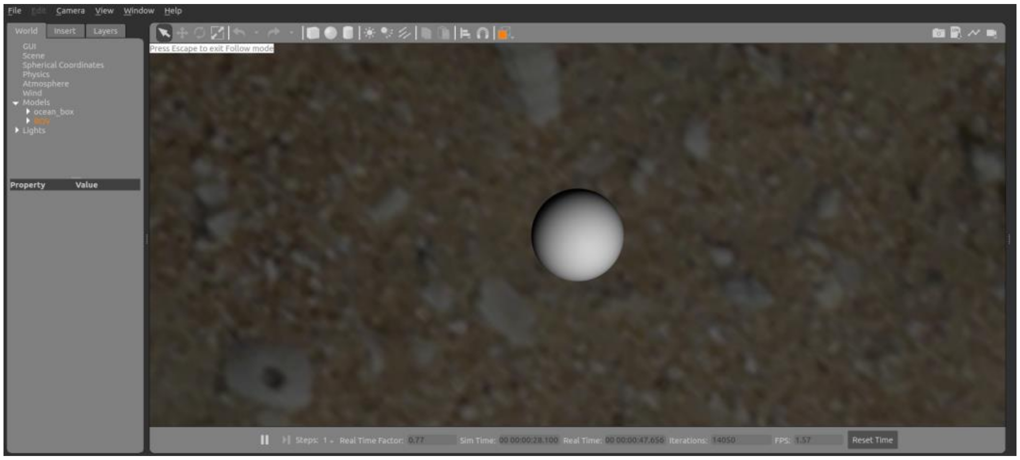

Figure 7 - Robot model spawned in the world seen in gazebo

Due to how our team planned the development of the fourth task, Harshit Mutha was in charge of leading development of the fourth task, and he built the additional node *teleop_rov.py* which serves the purpose of the node we created in task 3, although differently. While the one used in exercise 3 takes input and converts them to velocity, then acceleration, and then forces / torques to not have to worry about figuring out the internal conversion taking place or about forces being applied without accounting for the mass of the robot (which could result in way out of proportion movement, or the lack of movement in the first place), the node used now directly translates keyboard input to forces, this proved useful in a way where it simplifies the internal workings of the node, but in exchange it sacrificed flexibility with other robot models with differing masses.

While the program as a whole works, the results were not as intended. When the “/rov/thruster_manager/input” topic is observed with “**rostopic echo**” this shows that thrust commands are being published and hence received by the “/rov/thrusters” node to be processed and published as thrust commands, so the focus shifted to figure out why the model didnt move in our computers, suddenly stumbling upon the conclusion that it is instead a graphical error, since the robot did seem to move (even though laggy) in a device with higher graphical power. With this node, even though it is not possible to continue experimenting in our own devices due to their graphical limits, the development of a node to publish thrust commands was a success.

Comments and notes on this project:

The realization of this project brought with it multiple troubles, but we quickly found out that due to the intrinsic way ROS works, with lots of individual pieces which work to achieve a final output, we had a lot of control to handle these errors in our attempts to code scripts and build functional nodes, a prime example on this was during the fourth task, the one which presented us with the most difficulties. During the development of it we had trouble connecting the node we used in task three with the one developed for task 4, but we opted out for a new approach where we sacrificed the flexibility of the previous node for a more simplistic one which was very straightforward to implement.

Our learning came in a great part by already analyzing what was provided to us in the first task, by understanding how the launch file for the rexrov worked we could emulate its functions to use it for task 2, and the same happened with task 2, the creation of the node to control the rexrov helped greatly in increasing our understanding of the ROS system to create the culminating nodes which would be the ones used for task 3 and 4, along with their corresponding launch files.

In conclusion, the learning curve for this project was found to be demanding, but this does not mean it was unfair, it means that hours were meant to be dedicated to this project and that the constant troubleshooting in the earlier tasks would improve the toughness of our code steadily enough for us to have the tools to tackle the next task, overall a great learning experience.

APPENDIX

List 1: Terminal output from command “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

List 2: Terminal output from command “rostopic list”

/clicked_point
/diagnostics
/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
/rexrov/joy/set_feedback
/rexrov/pose_gt
/rexrov/rexrov/camera/camera_image

/rexrov/thruster_manager/input
/rexrov/thruster_manager/input_stamped
/rexrov/thrusters/0/input
/rexrov/thrusters/1/input
/rexrov/thrusters/2/input
/rexrov/thrusters/3/input
/rexrov/thrusters/4/input
/rexrov/thrusters/5/input
/rexrov/thrusters/6/input
/rexrov/thrusters/7/input
/rexrov/velocity_control/parameter_descriptions
/rexrov/velocity_control/parameter_updates
/rosout
/rosout_agg
/tf
/tf_static

List 3: Terminal output from command “rosservice list”

control_msgs/QueryCalibrationState
control_msgs/QueryTrajectoryState
control_toolbox/SetPidGains
controller_manager_msgs/ListControllerTypes
controller_manager_msgs/ListControllers
controller_manager_msgs/LoadController
controller_manager_msgs/ReloadControllerLibraries
controller_manager_msgs/SwitchController
controller_manager_msgs/UnloadController
diagnostic_msgs/AddDiagnostics
diagnostic_msgs/SelfTest
dynamic_reconfigure/Reconfigure
gazebo_msgs/ApplyBodyWrench
gazebo_msgs/ApplyJointEffort
gazebo_msgs/BodyRequest
gazebo_msgs/DeleteLight
gazebo_msgs/DeleteModel
gazebo_msgs/GetJointProperties
gazebo_msgs/GetLightProperties
gazebo_msgs/GetLinkProperties
gazebo_msgs/GetLinkState
gazebo_msgs/GetModelProperties
gazebo_msgs/GetModelState
gazebo_msgs/GetPhysicsProperties

gazebo_msgs/GetWorldProperties
gazebo_msgs/JointRequest
gazebo_msgs/SetJointProperties
gazebo_msgs/SetJointTrajectory
gazebo_msgs/SetLightProperties
gazebo_msgs/SetLinkProperties
gazebo_msgs/SetLinkState
gazebo_msgs/SetModelConfiguration
gazebo_msgs/SetModelState
gazebo_msgs/SetPhysicsProperties
gazebo_msgs/SpawnModel
geographic_msgs/GetGeoPath
geographic_msgs/GetGeographicMap
geographic_msgs/GetRoutePlan
geographic_msgs/UpdateGeographicMap
hector_gazebo_plugins/SetBias
hector_gazebo_plugins/SetReferenceGeoPose
laser_assembler/AssembleScans
laser_assembler/AssembleScans2
map_msgs/GetMapROI
map_msgs/GetPointMap
map_msgs/GetPointMapROI
map_msgs/ProjectedMapsInfo
map_msgs/SaveMap
map_msgs/SetMapProjections
nav_msgs/GetMap
nav_msgs/GetPlan
nav_msgs/LoadMap
nav_msgs/SetMap
nodelet/NodeletList
nodelet/NodeletLoad
nodelet/NodeletUnload
pcl_msgs/UpdateFilename
polled_camera/GetPolledImage
robot_localization/FromLL
robot_localization/GetState
robot_localization/SetDatum
robot_localization/SetPose
robot_localization/SetUTMZone
robot_localization/ToLL
robot_localization/ToggleFilterProcessing
roscpp/Empty
roscpp/GetLoggers
roscpp/SetLoggerLevel

roscpp_tutorials/TwoInts
rospy_tutorials/AddTwoInts
rospy_tutorials/BadTwoInts
rviz/SendFilePath
sensor_msgs/SetCameraInfo
std_srvs/Empty
std_srvs/SetBool
std_srvs/Trigger
tf/FrameGraph
tf2_msgs/FrameGraph
topic_tools/DemuxAdd
topic_tools/DemuxDelete
topic_tools/DemuxList
topic_tools/DemuxSelect
topic_tools/MuxAdd
topic_tools/MuxDelete
topic_tools/MuxList
topic_tools/MuxSelect
turtlesim/Kill
turtlesim/SetPen
turtlesim/Spawn
turtlesim/TeleportAbsolute
turtlesim/TeleportRelative
uuv_control_msgs/AddWaypoint
uuv_control_msgs/ClearWaypoints
uuv_control_msgs/GetMBSMControllerParams
uuv_control_msgs/GetPIDParams
uuv_control_msgs/GetSMControllerParams
uuv_control_msgs/GetWaypoints
uuv_control_msgs/GoTo
uuv_control_msgs/GoToIncremental
uuv_control_msgs/Hold
uuv_control_msgs/InitCircularTrajectory
uuv_control_msgs/InitHelicalTrajectory
uuv_control_msgs/InitRectTrajectory
uuv_control_msgs/InitWaypointSet
uuv_control_msgs/InitWaypointsFromFile
uuv_control_msgs/IsRunningTrajectory
uuv_control_msgs/ResetController
uuv_control_msgs/SetMBSMControllerParams
uuv_control_msgs/SetPIDParams
uuv_control_msgs/SetSMControllerParams
uuv_control_msgs/StartTrajectory
uuv_control_msgs/SwitchToAutomatic

uuv_control_msgs/SwitchToManual
uuv_gazebo_ros_plugins_msgs/GetFloat
uuv_gazebo_ros_plugins_msgs/GetListParam
uuv_gazebo_ros_plugins_msgs/GetModelProperties
uuv_gazebo_ros_plugins_msgs/GetThrusterConversionFcn
uuv_gazebo_ros_plugins_msgs/GetThrusterEfficiency
uuv_gazebo_ros_plugins_msgs/GetThrusterState
uuv_gazebo_ros_plugins_msgs/SetFloat
uuv_gazebo_ros_plugins_msgs/SetThrusterEfficiency
uuv_gazebo_ros_plugins_msgs/SetThrusterState
uuv_gazebo_ros_plugins_msgs/SetUseGlobalCurrentVel
uuv_sensor_ros_plugins_msgs/ChangeSensorState
uuv_thruster_manager/GetThrusterCurve
uuv_thruster_manager/GetThrusterManagerConfig
uuv_thruster_manager/SetThrusterManagerConfig
uuv_thruster_manager/ThrusterManagerInfo
uuv_world_ros_plugins_msgs/GetCurrentModel
uuv_world_ros_plugins_msgs/GetOriginSphericalCoord
uuv_world_ros_plugins_msgs/SetCurrentDirection
uuv_world_ros_plugins_msgs/SetCurrentModel
uuv_world_ros_plugins_msgs/SetCurrentVelocity
uuv_world_ros_plugins_msgs/SetOriginSphericalCoord
uuv_world_ros_plugins_msgs/TransformFromSphericalCoord
uuv_world_ros_plugins_msgs/TransformToSphericalCoord

List 4: Terminal output from command “rosmsg list”

actionlib/TestAction
actionlib/TestActionFeedback
actionlib/TestActionGoal
actionlib/TestActionResult
actionlib/TestFeedback
actionlib/TestGoal
actionlib/TestRequestAction
actionlib/TestRequestActionFeedback
actionlib/TestRequestActionGoal
actionlib/TestRequestActionResult
actionlib/TestRequestFeedback
actionlib/TestRequestGoal
actionlib/TestRequestResult
actionlib/TestResult
actionlib/TwoIntsAction
actionlib/TwoIntsActionFeedback
actionlib/TwoIntsActionGoal

actionlib/TwoIntsActionResult
actionlib/TwoIntsFeedback
actionlib/TwoIntsGoal
actionlib/TwoIntsResult
actionlib_msgs/GoalID
actionlib_msgs/GoalStatus
actionlib_msgs/GoalStatusArray
actionlib_tutorials/AveragingAction
actionlib_tutorials/AveragingActionFeedback
actionlib_tutorials/AveragingActionGoal
actionlib_tutorials/AveragingActionResult
actionlib_tutorials/AveragingFeedback
actionlib_tutorials/AveragingGoal
actionlib_tutorials/AveragingResult
actionlib_tutorials/FibonacciAction
actionlib_tutorials/FibonacciActionFeedback
actionlib_tutorials/FibonacciActionGoal
actionlib_tutorials/FibonacciActionResult
actionlib_tutorials/FibonacciFeedback
actionlib_tutorials/FibonacciGoal
actionlib_tutorials/FibonacciResult
bond/Constants
bond/Status
control_msgs/FollowJointTrajectoryAction
control_msgs/FollowJointTrajectoryActionFeedback
control_msgs/FollowJointTrajectoryActionGoal
control_msgs/FollowJointTrajectoryActionResult
control_msgs/FollowJointTrajectoryFeedback
control_msgs/FollowJointTrajectoryGoal
control_msgs/FollowJointTrajectoryResult
control_msgs/GripperCommand
control_msgs/GripperCommandAction
control_msgs/GripperCommandActionFeedback
control_msgs/GripperCommandActionGoal
control_msgs/GripperCommandActionResult
control_msgs/GripperCommandFeedback
control_msgs/GripperCommandGoal
control_msgs/GripperCommandResult
control_msgs/JointControllerState
control_msgs/JointJog
control_msgs/JointTolerance
control_msgs/JointTrajectoryAction
control_msgs/JointTrajectoryActionFeedback
control_msgs/JointTrajectoryActionGoal

control_msgs/JointTrajectoryActionResult
control_msgs/JointTrajectoryControllerState
control_msgs/JointTrajectoryFeedback
control_msgs/JointTrajectoryGoal
control_msgs/JointTrajectoryResult
control_msgs/PidState
control_msgs/PointHeadAction
control_msgs/PointHeadActionFeedback
control_msgs/PointHeadActionGoal
control_msgs/PointHeadActionResult
control_msgs/PointHeadFeedback
control_msgs/PointHeadGoal
control_msgs/PointHeadResult
control_msgs/SingleJointPositionAction
control_msgs/SingleJointPositionActionFeedback
control_msgs/SingleJointPositionActionGoal
control_msgs/SingleJointPositionActionResult
control_msgs/SingleJointPositionFeedback
control_msgs/SingleJointPositionGoal
control_msgs/SingleJointPositionResult
controller_manager_msgs/ControllerState
controller_manager_msgs/ControllerStatistics
controller_manager_msgs/ControllersStatistics
controller_manager_msgs/HardwareInterfaceResources
diagnostic_msgs/DiagnosticArray
diagnostic_msgs/DiagnosticStatus
diagnostic_msgs/KeyValue
dynamic_reconfigure/BoolParameter
dynamic_reconfigure/Config
dynamic_reconfigure/ConfigDescription
dynamic_reconfigure/DoubleParameter
dynamic_reconfigure/Group
dynamic_reconfigure/GroupState
dynamic_reconfigure/IntParameter
dynamic_reconfigure/ParamDescription
dynamic_reconfigure/SensorLevels
dynamic_reconfigure/StrParameter
gazebo_msgs/ContactState
gazebo_msgs/ContactsState
gazebo_msgs/LinkState
gazebo_msgs/LinkStates
gazebo_msgs/ModelState
gazebo_msgs/ModelStates
gazebo_msgs/ODEJointProperties

gazebo_msgs/ODEPhysics
gazebo_msgs/PerformanceMetrics
gazebo_msgs/SensorPerformanceMetric
gazebo_msgs/WorldState
geographic_msgs/BoundingBox
geographic_msgs/GeoPath
geographic_msgs/GeoPoint
geographic_msgs/GeoPointStamped
geographic_msgs/GeoPose
geographic_msgs/GeoPoseStamped
geographic_msgs/GeographicMap
geographic_msgs/GeographicMapChanges
geographic_msgs/KeyValue
geographic_msgs/MapFeature
geographic_msgs/RouteNetwork
geographic_msgs/RoutePath
geographic_msgs/RouteSegment
geographic_msgs/WayPoint
geometry_msgs/Accel
geometry_msgs/AccelStamped
geometry_msgs/AccelWithCovariance
geometry_msgs/AccelWithCovarianceStamped
geometry_msgs/Inertia
geometry_msgs/InertiaStamped
geometry_msgs/Point
geometry_msgs/Point32
geometry_msgs/PointStamped
geometry_msgs/Polygon
geometry_msgs/PolygonStamped
geometry_msgs/Pose
geometry_msgs/Pose2D
geometry_msgs/PoseArray
geometry_msgs/PoseStamped
geometry_msgs/PoseWithCovariance
geometry_msgs/PoseWithCovarianceStamped
geometry_msgs/Quaternion
geometry_msgs/QuaternionStamped
geometry_msgs/Transform
geometry_msgs/TransformStamped
geometry_msgs/Twist
geometry_msgs/TwistStamped
geometry_msgs/TwistWithCovariance
geometry_msgs/TwistWithCovarianceStamped
geometry_msgs/Vector3

geometry_msgs/Vector3Stamped
geometry_msgs/Wrench
geometry_msgs/WrenchStamped
map_msgs/OccupancyGridUpdate
map_msgs/PointCloud2Update
map_msgs/ProjectedMap
map_msgs/ProjectedMapInfo
nav_msgs/GetMapAction
nav_msgs/GetMapActionFeedback
nav_msgs/GetMapActionGoal
nav_msgs/GetMapActionResult
nav_msgs/GetMapFeedback
nav_msgs/GetMapGoal
nav_msgs/GetMapResult
nav_msgs/GridCells
nav_msgs/MapMetaData
nav_msgs/OccupancyGrid
nav_msgs/Odometry
nav_msgs/Path
pcl_msgs/ModelCoefficients
pcl_msgs/PointIndices
pcl_msgs/PolygonMesh
pcl_msgs/Vertices
roscpp/Logger
rosgraph_msgs/Clock
rosgraph_msgs/Log
rosgraph_msgs/TopicStatistics
rospy_tutorials/Floats
rospy_tutorials/HeaderString
sensor_msgs/BatteryState
sensor_msgs/CameraInfo
sensor_msgs/ChannelFloat32
sensor_msgs/CompressedImage
sensor_msgs/FluidPressure
sensor_msgs/Illuminance
sensor_msgs/Image
sensor_msgs/Imu
sensor_msgs/JointState
sensor_msgs/Joy
sensor_msgs/JoyFeedback
sensor_msgs/JoyFeedbackArray
sensor_msgs/LaserEcho
sensor_msgs/LaserScan
sensor_msgs/MagneticField

sensor_msgs/MultiDOFJointState
sensor_msgs/MultiEchoLaserScan
sensor_msgs/NavSatFix
sensor_msgs/NavSatStatus
sensor_msgs/PointCloud
sensor_msgs/PointCloud2
sensor_msgs/PointField
sensor_msgs/Range
sensor_msgs/RegionOfInterest
sensor_msgs/RelativeHumidity
sensor_msgs/Temperature
sensor_msgs/TimeReference
shape_msgs/Mesh
shape_msgs/MeshTriangle
shape_msgs/Plane
shape_msgs/SolidPrimitive
smach_msgs/SmachContainerInitialStatusCmd
smach_msgs/SmachContainerStatus
smach_msgs/SmachContainerStructure
std_msgs/Bool
std_msgs/Byte
std_msgs/ByteMultiArray
std_msgs/Char
std_msgs/ColorRGBA
std_msgs/Duration
std_msgs/Empty
std_msgs/Float32
std_msgs/Float32MultiArray
std_msgs/Float64
std_msgs/Float64MultiArray
std_msgs/Header
std_msgs/Int16
std_msgs/Int16MultiArray
std_msgs/Int32
std_msgs/Int32MultiArray
std_msgs/Int64
std_msgs/Int64MultiArray
std_msgs/Int8
std_msgs/Int8MultiArray
std_msgs/MultiArrayDimension
std_msgs/MultiArrayLayout
std_msgs/String
std_msgs/Time
std_msgs/UInt16

std_msgs/UInt16MultiArray
std_msgs/UInt32
std_msgs/UInt32MultiArray
std_msgs/UInt64
std_msgs/UInt64MultiArray
std_msgs/UInt8
std_msgs/UInt8MultiArray
stereo_msgs/DisparityImage
tf/tfMessage
tf2_msgs/LookupTransformAction
tf2_msgs/LookupTransformActionFeedback
tf2_msgs/LookupTransformActionGoal
tf2_msgs/LookupTransformActionResult
tf2_msgs/LookupTransformFeedback
tf2_msgs/LookupTransformGoal
tf2_msgs/LookupTransformResult
tf2_msgs/TF2Error
tf2_msgs/TFMessage
theora_image_transport/Packet
trajectory_msgs/JointTrajectory
trajectory_msgs/JointTrajectoryPoint
trajectory_msgs/MultiDOFJointTrajectory
trajectory_msgs/MultiDOFJointTrajectoryPoint
turtle_actionlib/ShapeAction
turtle_actionlib/ShapeActionFeedback
turtle_actionlib/ShapeActionGoal
turtle_actionlib/ShapeActionResult
turtle_actionlib/ShapeFeedback
turtle_actionlib/ShapeGoal
turtle_actionlib/ShapeResult
turtle_actionlib/Velocity
turtlesim/Color
turtlesim/Pose
twist_mux_msgs/JoyPriorityAction
twist_mux_msgs/JoyPriorityActionFeedback
twist_mux_msgs/JoyPriorityActionGoal
twist_mux_msgs/JoyPriorityActionResult
twist_mux_msgs/JoyPriorityFeedback
twist_mux_msgs/JoyPriorityGoal
twist_mux_msgs/JoyPriorityResult
twist_mux_msgs/JoyTurboAction
twist_mux_msgs/JoyTurboActionFeedback
twist_mux_msgs/JoyTurboActionGoal
twist_mux_msgs/JoyTurboActionResult

twist_mux_msgs/JoyTurboFeedback
twist_mux_msgs/JoyTurboGoal
twist_mux_msgs/JoyTurboResult
uuid_msgs/UniqueID
uuv_auv_control_allocator/AUVCommand
uuv_control_msgs/Trajectory
uuv_control_msgs/TrajectoryPoint
uuv_control_msgs/Waypoint
uuv_control_msgs/WaypointSet
uuv_gazebo_ros_plugins_msgs/FloatStamped
uuv_gazebo_ros_plugins_msgs/ThrusterConversionFcn
uuv_gazebo_ros_plugins_msgs/UnderwaterObjectModel
uuv_sensor_ros_plugins_msgs/ChemicalParticleConcentration
uuv_sensor_ros_plugins_msgs/DVL
uuv_sensor_ros_plugins_msgs/DVLBeam
uuv_sensor_ros_plugins_msgs/PositionWithCovariance
uuv_sensor_ros_plugins_msgs/PositionWithCovarianceStamped
uuv_sensor_ros_plugins_msgs/Salinity
visualization_msgs/ImageMarker
visualization_msgs/InteractiveMarker
visualization_msgs/InteractiveMarkerControl
visualization_msgs/InteractiveMarkerFeedback
visualization_msgs/InteractiveMarkerInit
visualization_msgs/InteractiveMarkerPose
visualization_msgs/InteractiveMarkerUpdate
visualization_msgs/Marker
visualization_msgs/MarkerArray
visualization_msgs/MenuEntry

Table 1 - Used commands for task (1)

Command	Function
roscd	Changes the current directory to the directory in which ros was sourced. It would normally open the directory "*catkin_ws/devel*" assuming a catkin workspace has been correctly created.
git clone	This command was used to clone the *uuv_simulator_noetic* (or *uuv_simulator*) git branch into the src directory of the defined catkin workspace.
catkin_make	This command was used to build the *uuv_simulator_noetic* (or *uuv_simulator*) package previously cloned from the github branch.
source	Used to source the active terminal to a certain setup.bash file, used to be provided of various environment variables, allowing ROS to properly function. In this case, it was used to source the catkin workspace after it was created with "**source catkin_ws/devel/setup.bash**".
roslaunch	Used to launch the *rexrov_default.launch* file with "**roslaunch uuv_gazebo rexrov_default.launch**".
rqt_graph	Used to visualize node graph shown in figure 2.
rosnode list **rostopic list** **rosservice list** **rosmsg list**	All commands were used to obtain lists 1, 2 and 3 shown in the appendix section.
rostopic type	This command was used to obtain more information about what messages a topic was transmitting.
rosmsg show	This command was used to see what kind of data some messages transmitted.

Fair Contribution Sheet

Group Number: 8

Enter average fair contribution scores here:

Member	Mark 1	Mark 2	Mark 3	Average Mark
Harshit Mutha	10	10	10	10
Leonardo Juarez	10	10	10	10
Nadim Khneisser	10	10	10	10
Santiago Pinto	10	10	10	10

Name and signature of each member + date signed:

Name: Harshit Mutha

Signature: .... Date: 08.12.2023

Name: SANTIAGO PINTO

Signature: .... Date: 08/12/2023

Name: NADIM Khneisser

Signature: .... Date: 08/12/2023

Name: Leonardo Juarez

Signature: .... Date: 08.12.2023