# Chapter 3: Virtual Simulation

1. **Introduction**

This chapter presents the virtual simulation environment developed to test and evaluate the autonomous functionalities of the cleaning boat. As part of the preliminary research phase, two well-known robotic software platforms were studied and evaluated: **Webots** and **ROS1 Noetic**.

The decision to explore both platforms was based on their unique strengths. Webots is a modern robotic simulator with an intuitive graphical interface and built-in physics, making it suitable for rapid prototyping and visualization. ROS1 Noetic, on the other hand, is a mature and modular robotic framework that offers strong community support, extensive libraries, and integration capabilities.

After a comparative analysis of both tools, considering factors such as flexibility, ease of sensor and actuator integration, documentation availability, and long-term maintainability, **ROS1 Noetic Ninjemys** **was selected as the primary platform** for the simulation work. This choice was primarily due to its robust ecosystem, comprehensive documentation, and seamless support for developing custom nodes, which are essential for simulating real-world autonomous behavior.

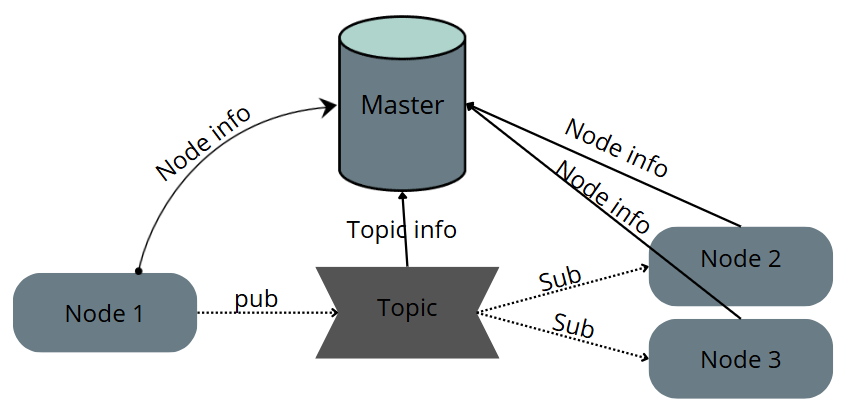
The remainder of this chapter details the simulation environment setup using ROS1 Noetic, including the boat model, navigation strategy, and waste detection process. The simulation replicates the core functionalities of the autonomous cleaning boat, which include autonomous navigation, detection and localization of floating waste, and the ability to plan and follow a trajectory to collect debris in a defined area. These functionalities were implemented and tested in the virtual environment to ensure the system performs reliably before physical deployment.

1. **Principle of ROS1 Noetic Communication**

In ROS1 Noetic, the fundamental building blocks of any robotic system are called **nodes**. A node is an independent executable that performs a specific task, such as reading sensor data, performing localization, or controlling actuators. These nodes are designed to work collaboratively by communicating with one another through topics, services, or actions. This modular node-based design allows developers to divide complex robotic behavior into manageable and reusable components, improving system scalability and development efficiency.

**2.1 Role of the ROS Master**

The **ROS Master** acts as the central coordination service in any ROS1 Noetic system. Its main role is to manage the **registration of nodes**, **topics**, and **services**. When a node wants to publish or subscribe to a topic, it first communicates with the ROS Master to register its intent. The master then facilitates the **connection** between publisher and subscriber nodes by sharing their network addresses, enabling them to establish a direct peer-to-peer communication link. Although the master is essential for establishing these connections, it does not handle or forward any actual message data once the nodes are connected.



**2.2 Communication Between Nodes**

Communication in ROS1 Noetic is primarily handled using a **publish/subscribe model via topics**. A node that generates data (like a camera or IMU) publishes messages to a named topic, while other nodes that require this data subscribe to the same topic. This connection is dynamic and made possible through the ROS Master, which ensures both sides are compatible in message type



**2.3 Importance of Handling Multiple Publishers on One Topic**

When multiple nodes publish to the same topic, such as a velocity command topic (/cmd\_vel), the **subscriber will always receive only the latest message** sent. ROS does not provide built-in arbitration or prioritization between publishers.

1. **Simulation Environment**

ROS (Robot Operating System) is an open-source framework designed to simplify the development of complex and modular robotic systems. It provides essential tools, libraries, and communication mechanisms that allow developers to build robot software in a structured and scalable way.

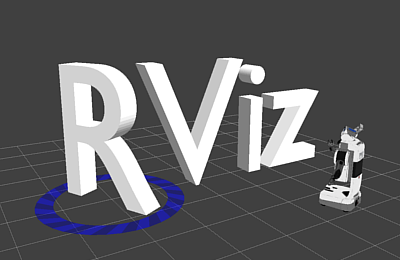


The development environment was set up on **Ubuntu 20.04 (Focal Fossa)**, which is the only officially supported version for ROS1 Noetic. This ensures compatibility with key simulation tools such as **Gazebo 11** and **RViz**, which are essential for modeling physical interactions, testing sensor data, and visualizing system behavior.

**Gazebo 11** is a powerful open-source 3D robotics simulator that provides a realistic environment for testing robot models with accurate physics, sensor simulation, and dynamic interactions. It is the final long-term support (LTS) release of the Gazebo classic series and is fully compatible with **ROS1 Noetic**. Gazebo 11 allows developers to simulate complex scenarios, making it ideal for evaluating robot behavior before physical implementation.



**RViz** (ROS Visualization) is a 3D visualization tool for ROS that allows developers to view sensor data, robot models, and coordinate frames in real time. It is especially useful for debugging and understanding how a robot perceives its environment and executes navigation tasks. In this project, RViz was used to visualize the boat's position, orientation, sensor outputs, and planned paths during simulation.



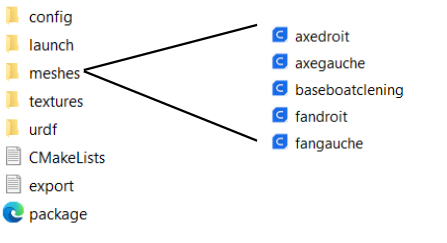
1. **Boat Model**

This section details the step-by-step process of creating and integrating the boat model into the ROS 1 Noetic simulation environment, including exporting the model from SolidWorks, importing it into the ROS workspace, and configuring sensors (camera, LiDAR, and GPS) using URDF plugins.

**4.1 Exporting from SolidWorks**

To ensure compatibility with ROS and Gazebo, the SolidWorks model was exported using the **SolidWorks to URDF Exporter**, a plugin developed by the ROS community. This plugin generates a URDF (Unified Robot Description Format) file along with the associated mesh files (in .STL or .DAE format). Before exporting, all parts were properly named, assembled, and assigned coordinate frames in SolidWorks to ensure correct transformations and joint placement in ROS.

In our case, the exportation method provided mesh files specifically in the .STL format.



**4.2 Integration into ROS Workspace**

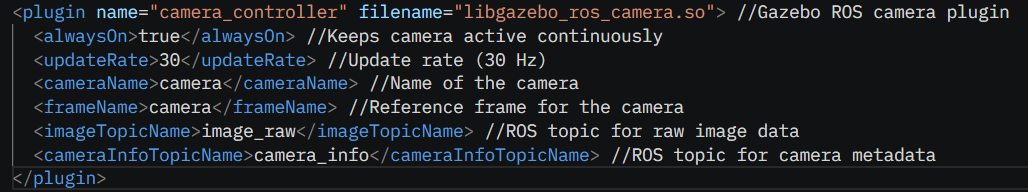
A dedicated ROS workspace was created to organize the exported URDF files and streamline the development process. This workspace setup made it easier to modify the robot description and add custom plugin code as needed. The structure was designed to support further steps such as navigation and simulation, ensuring a smooth transition into the next phases of the project.

**4.3 Sensor Integration and Plugins**

To simulate autonomous capabilities, key sensors were integrated into the boat model using Gazebo plugins within the URDF description. These sensors provide essential data for perception, localization, and mapping in simulation. Each sensor was configured using <gazebo> tags inside the URDF file, specifying the plugin type, sensor parameters (such as update rate, resolution, and noise), and the sensor’s pose relative to the boat’s base link.

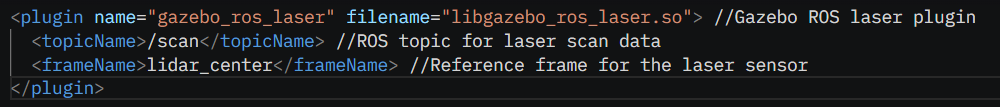
1. **Camera Plugin**

The camera was integrated using the libgazebo\_ros\_camera.so plugin, which allows Gazebo to simulate an RGB camera and publish its data to ROS topics for use in perception tasks.



1. **Lidar Plugin**

The LIDAR sensor was integrated using the libgazebo\_ros\_laser.so plugin, which enables the simulation of a 2D laser scanner and publishes scan data to a ROS topic for tasks such as obstacle detection and mapping.



1. **Navigation Simulation**

Realistic boat navigation in simulation requires accurate modeling of hydrostatic forces and floatability to reflect how a vessel interacts with water. After exploring available solutions, I found the “asv\_wave\_sim” package, which is specifically designed for simulating autonomous surface vehicles (ASVs) in Gazebo.

This package includes a hydrostatics plugin that models buoyancy and floatation, providing a physically accurate water interaction model. It also supports wave generation and water surface dynamics, which significantly enhances the realism of the navigation environment.

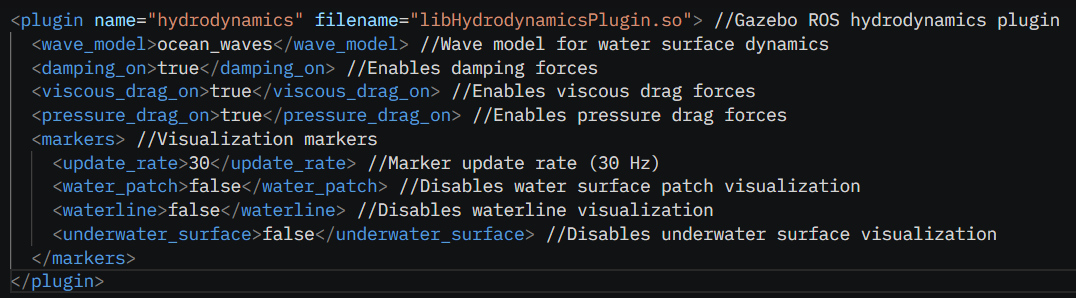
In addition to its physics capabilities, the package also provides useful prebuilt models such as floating buoys and waste objects, which are valuable for simulating navigation scenarios involving obstacle avoidance and waste collection.

Importantly, the package is fully compatible with **ROS Noetic** and **Ubuntu 20.04**, making it a suitable choice for integration into the existing simulation framework. By leveraging asv\_wave\_sim, the virtual boat can float and respond to hydrodynamic forces in a way that mirrors real-world behavior, forming a solid foundation for advanced navigation testing.

**5.1 hydrodynamics pluggin**

I used the libHydrodynamicsPlugin.so provided by asv\_wave\_sim to simulate realistic buoyancy and water interaction within the ocean environment. This plugin models the hydrodynamic forces acting on the boat, enabling accurate floating behavior and water resistance simulation. The configuration enabled damping, viscous drag, and pressure drag—three key components contributing to the realistic motion of a vessel in water.

This hydrodynamics plugin successfully enabled the boat to float stably and respond realistically to the simulated ocean environment, confirming the effectiveness of the setup for virtual navigation development.



**5.2 Odometry node**

Odometry is a fundamental component of autonomous navigation, as it provides continuous estimates of a robot’s position and velocity based on sensor data. For a mobile platform like an autonomous cleaning boat, accurate odometry is essential for tasks such as path planning, localization, and motion control. It enables the robot to understand its movement over time within a given environment, especially in the absence of external positioning systems or during temporary GPS loss. In simulated environments, odometry also plays a crucial role in synchronizing sensor data and visualizing the robot’s trajectory in tools like RViz or during mapping processes.

As part of this project, I developed a dedicated **C++ ROS node** to publish the boat’s odometry during simulation. This node subscribes to the “/gazebo/link\_states” topic, extracts the pose and velocity of the boat’s base link (boatcleaningc::baseboatclening), and publishes the corresponding “nav\_msgs/Odometry” message to the odom topic. It also broadcasts the transform between the odom frame and the boat’s base frame using the tf package, enabling accurate localization and sensor alignment. To ensure smooth operation during simulation, the node was added to the main launch file so that it starts automatically alongside Gazebo and other ROS components.