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 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.

2. System Architecture

The autonomous system is structured into several interconnected components, primarily leveraging the modularity inherent in the Robot Operating System (ROS) architecture. To fully understand the different components of the system, we illustrate it in this figure, which shows a clear representation of the connection and the flow of information between the sections and subsections of this system

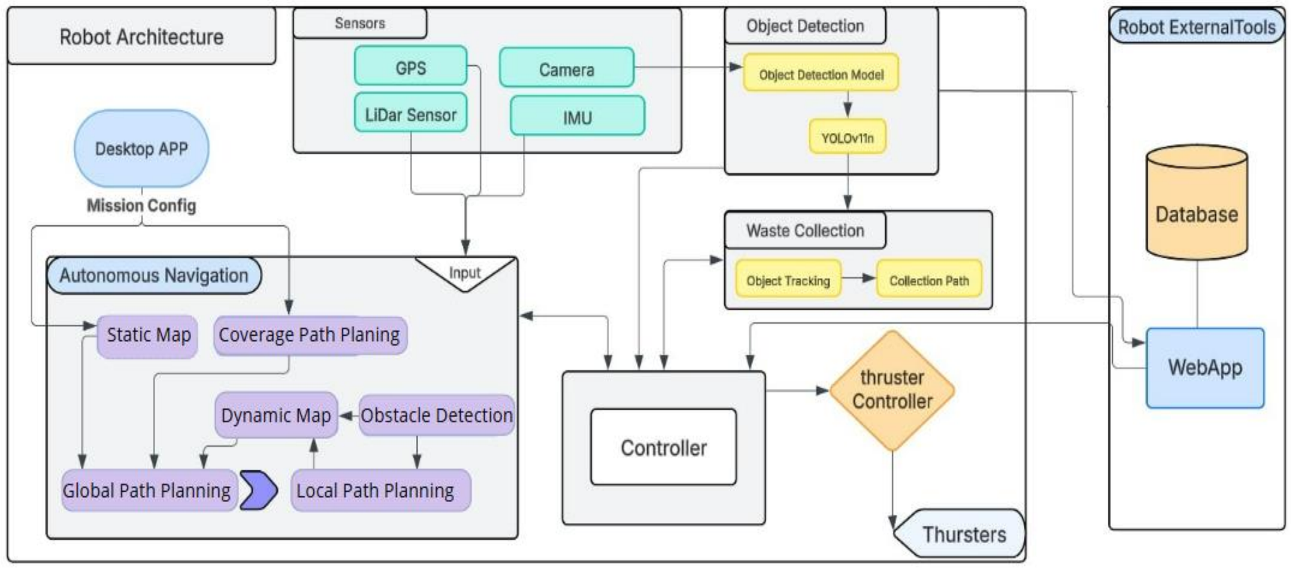


Figure 1 Robot System Architecture

Our autonomous system is divided into multiple sections or components based on ROS architecture. This will allow us to develop each section independently, allowing us to focus on the development of specific functionalities. Further in this chapter, we will talk more about the different sections and components used in this system

3. Robot Operating System (ROS) Implementation

3.1. ROS Framework and Core Packages

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

This project was developed using ROS 1 Noetic Ninjemys, the final ROS 1 distribution, which is officially supported on Ubuntu 20.04 (Focal Fossa). This setup ensures compatibility with key simulation tools such as Gazebo 11 and RViz, which are vital for modeling physical interactions, testing sensor data, and visualizing system behavior.

In this project, several ROS packages were utilized to enable autonomous navigation, localization, and environment mapping for the robotic system. The core packages included "**asv_wave_sim**", "**move_base**", "**amcl**", and "**slam_gmapping**". Together, they provided a simulation environment, path planning, localization, and SLAM (Simultaneous Localization and Mapping) capabilities.

- **asv_wave_sim:**
  A ROS package used to simulate an Autonomous Surface Vehicle (ASV) in a wave-affected water environment. It enables testing and validation of robot behaviour under realistic maritime conditions in Gazebo.

- **move_base:**
  Provides a high-level interface for autonomous navigation by combining global and local planners. It handles path planning, obstacle avoidance, and goal-reaching behaviour.

- **amcl:**
  Implements Adaptive Monte Carlo Localization for estimating the robot's pose in a known map. It uses sensor data (e.g., laser scans) and odometry to perform probabilistic localization.

- **slam_gmapping:**
  Allows the robot to build a 2D occupancy grid map of an unknown environment while tracking its own position. It leverages laser scan data and a particle filter algorithm to perform real-time SLAM (Simultaneous Localization and Mapping).

In addition to standard ROS packages, a custom package named boatcleaning was developed specifically for this project. The boat model was originally designed in SolidWorks, then exported as a URDF (Unified Robot Description Format) file and integrated into the ROS/Gazebo simulation environment. This model served as the primary representation of the autonomous cleaning boat in the project.

3.2. ROS Package Structure

3.2.1. Package Hierarchy Overview

After developing and integrating the core components, all ROS packages for this project were systematically arranged into a modular and well-organized folder structure. This approach improves the system's maintainability and scalability, while facilitating independent development and testing of individual subsystems. The diagram below depicts the comprehensive organization of the ROS packages utilized in this project.

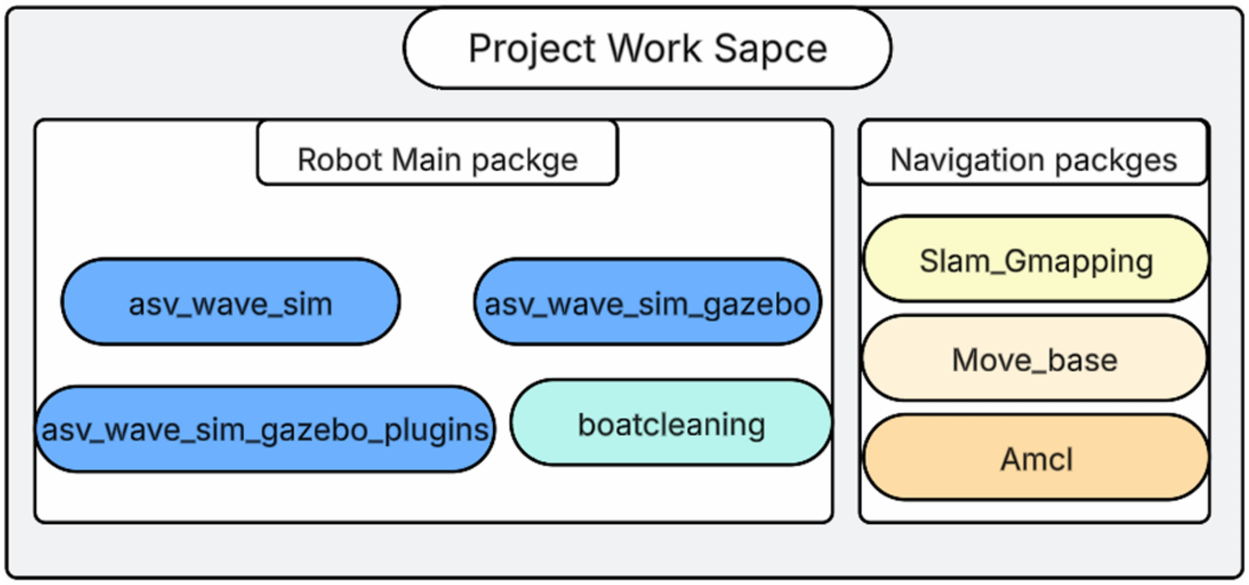
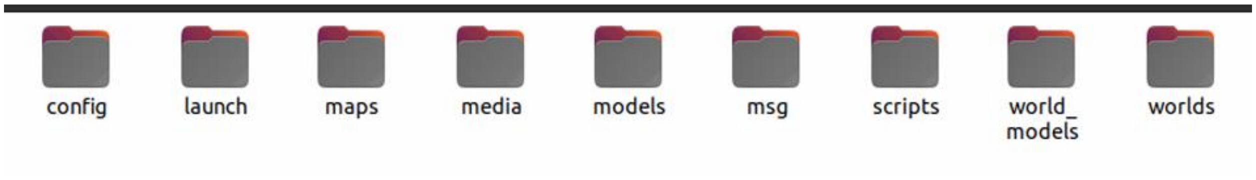


Figure 2 Package Hierarchy Overview

The "**asv_wave_sim**" package serves as the deployment environment for the final system, designed for real-world use on the autonomous boat. Within this framework, the "**asv_wave_sim_gazebo**" package is central, enabling the implementation and testing of various functionalities in a simulated environment using ROS tools such as RVIZ and Gazebo. Additionally, the "**asv_wave_sim_gazebo_plugins**" package incorporates pre-existing code to simulate water physics and ocean behavior. These packages streamline development and minimize risks of material damage or unforeseen errors during early testing phases.

The "**asv_wave_sim_gazebo**" package, which I developed, organizes the system's required functionalities in a clear and structured manner, with its internal folder structure, including directories.



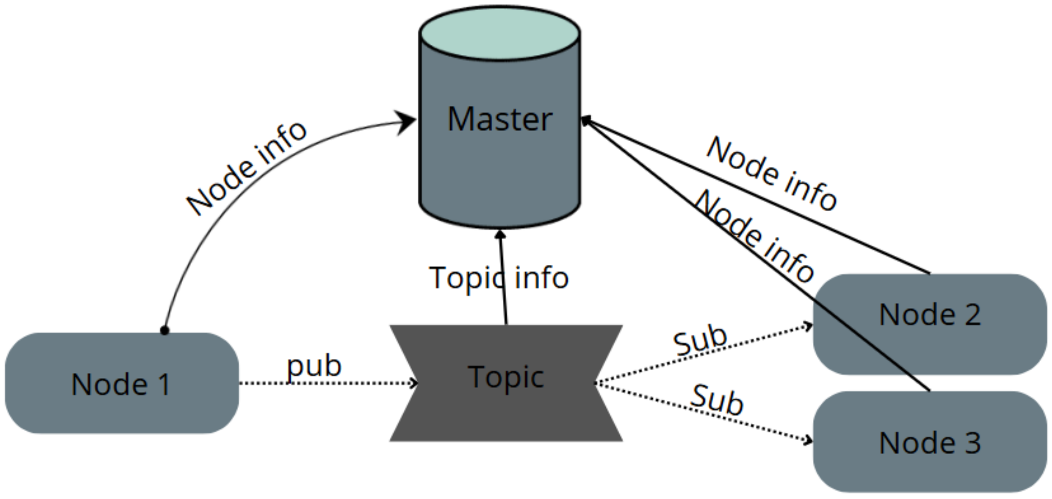
- **config**: This folder holds YAML files for system parameters like navigation and simulation settings. These files enable fine-tuning of the robot's behaviour for various marine environments.
- **launch**: This directory contains launch files that orchestrate the startup of multiple ROS nodes and configurations. It ensures smooth initialization of the system for simulation and deployment.
- **maps**: This directory stores PGM-format virtual maps essential for the navigation stack. These 2D occupancy grids support mapping and localization in static and dynamic marine settings.
- **media**: This folder includes visual assets like water textures to enhance Gazebo simulation realism. These assets improve the visual accuracy of the marine environment for testing.
- **models**: This folder contains 3D models and URDF files of objects like debris for Gazebo simulations. It supports testing of navigation and waste detection functionalities.
- **msg**: This directory defines custom ROS message types for efficient node communication. Messages like Detected Objects enable seamless data exchange for waste detection.
- **scripts**: This folder houses executable nodes for autonomous navigation and waste collection.
- **world_models & worlds**: These folders contain simulation maps and models defining ocean environments for Gazebo. They provide realistic scenarios for testing robot performance.

3.3 ROS1 Noetic Node Interaction and Data Flow

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 or controlling actuators. These nodes are designed to work collaboratively by communicating with one another through topics, services, or actions.

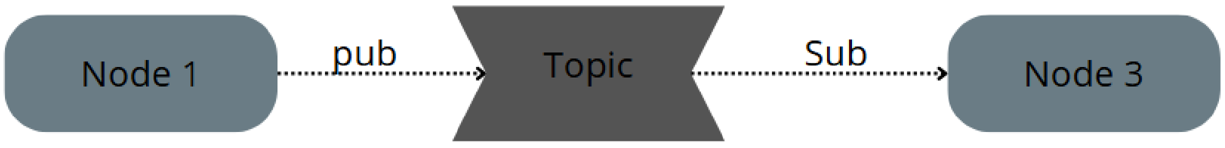
3.3.1 Role of the ROS Master

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.



- **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.



- **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.

3.4 ROS Node Architecture and Messaging

The autonomous marine robot's ROS system, which I developed, leverages a network of interconnected nodes to achieve modular functionality. This design organizes the system into independent yet cooperative components, each handling specific tasks such as sensor processing or actuator control, seamlessly integrated within my custom boatcleaning package.

As shown in Figure {below}, the primary nodes developed for this system are described in the following sections.

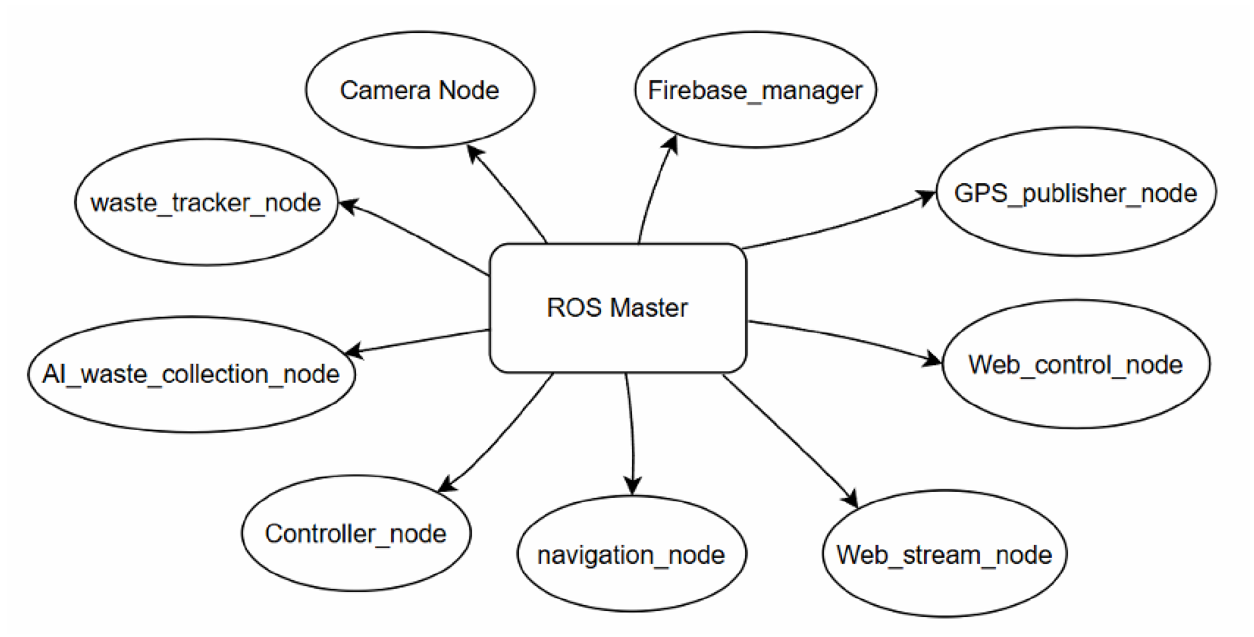


Figure 3ROS Node Architecture

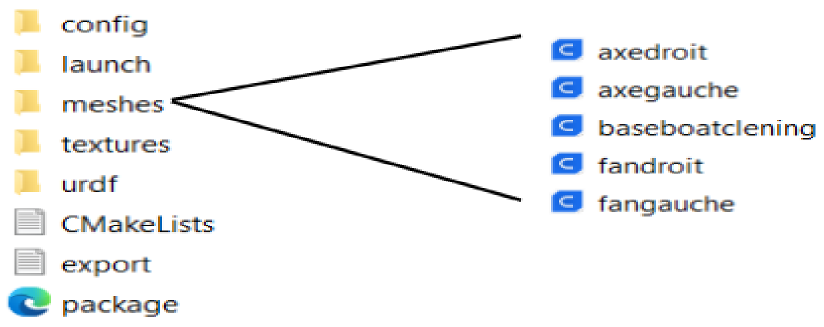
4. Boat Model

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, configuring sensors (camera, LiDAR, and GPS) using URDF plugins and testing the navigation simulation.

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

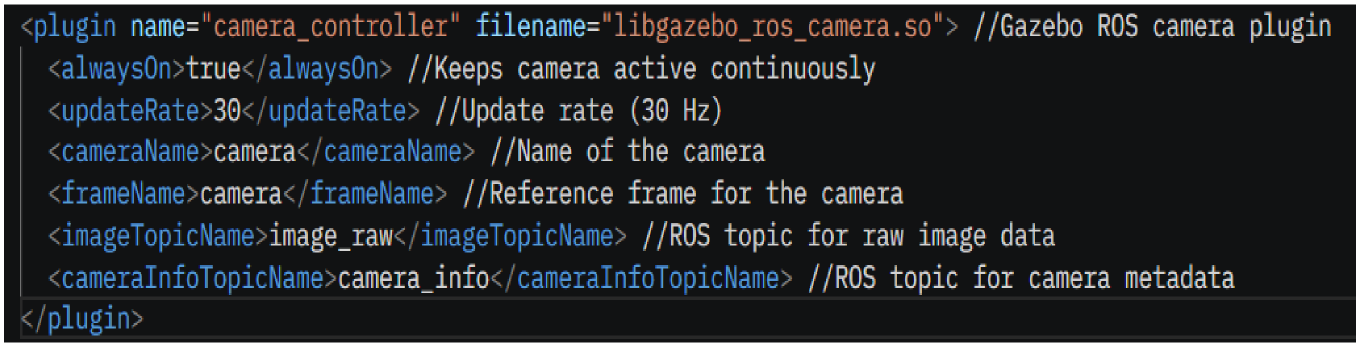
A dedicated ROS workspace was created to organize the exported URDF files and streamline the development process. This workspace setup made it easier to modify the robot description and add custom plugin code as needed. The structure was designed to support further steps such as navigation and simulation,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

- **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.

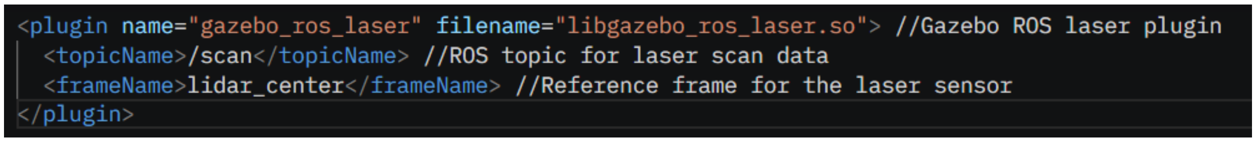


```
<plugin name="camera_controller" filename="libgazebo_ros_camera.so"> //Gazebo ROS camera plugin
  <alwaysOn>true</alwaysOn> //Keeps camera active continuously
  <updateRate>30</updateRate> //Update rate (30 Hz)
  <cameraName>camera</cameraName> //Name of the camera
  <frameName>camera</frameName> //Reference frame for the camera
  <imageTopicName>image_raw</imageTopicName> //ROS topic for raw image data
  <cameraInfoTopicName>camera_info</cameraInfoTopicName> //ROS topic for camera metadata
</plugin>
```

Figure 4 code Camera Plugin

- **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.



```
<plugin name="gazebo_ros_laser" filename="libgazebo_ros_laser.so"> //Gazebo ROS laser plugin
  <topicName>/scan</topicName> //ROS topic for laser scan data
  <frameName>lidar_center</frameName> //Reference frame for the laser sensor
</plugin>
```

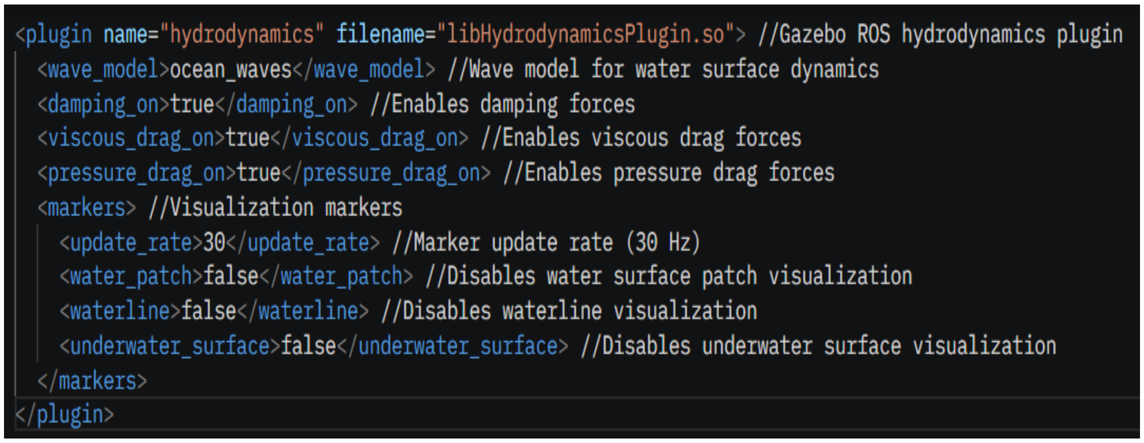
Figure 5 Code Lidar Plugin

4.4 Navigation Simulation

Realistic boat navigation in simulation requires accurate modelling of hydrostatic forces and floatability. The “asv_wave_sim” package, designed for simulating ASVs in Gazebo, includes a hydrostatics plugin for buoyancy and floatation, wave generation, and water surface dynamics, enhancing navigation realism. It also offers prebuilt models like floating buoys and waste objects for scenarios such as obstacle avoidance and waste collection. Fully compatible with ROS Noetic and Ubuntu 20.04, this package ensures realistic hydrodynamic responses, supporting advanced navigation testing.

- **hydrodynamics plugin**

To achieve realistic buoyancy and water interaction, the `libHydrodynamicsPlugin.so` provided by `asv_wave_sim` was utilized. This plugin models the hydrodynamic forces acting on the boat, enabling accurate floating behavior and water resistance simulation. The configuration included damping, viscous drag, and pressure drag, three key components that contribute to the realistic motion of a vessel in water.



```
<plugin name="hydrodynamics" filename="libHydrodynamicsPlugin.so"> //Gazebo ROS hydrodynamics plugin
  <wave_model>ocean_waves</wave_model> //Wave model for water surface dynamics
  <damping_on>true</damping_on> //Enables damping forces
  <viscous_drag_on>true</viscous_drag_on> //Enables viscous drag forces
  <pressure_drag_on>true</pressure_drag_on> //Enables pressure drag forces
  <markers> //Visualization markers
    <update_rate>30</update_rate> //Marker update rate (30 Hz)
    <water_patch>false</water_patch> //Disables water surface patch visualization
    <waterline>false</waterline> //Disables waterline visualization
    <underwater_surface>false</underwater_surface> //Disables underwater surface visualization
  </markers>
</plugin>
```

Figure 6 Code Hydrodynamics plugin

With the integration of the libHydrodynamicsPlugin.so, the boat model successfully navigated in the Gazebo 11 simulation environment. The hydrodynamics plugin provided realistic water interaction, ensuring stable floating behavior and accurate responses to hydrodynamic forces such as damping and drag. The image below illustrates the boat's successful navigation

picture

5. Autonomous Navigation Simulation

Realistic boat navigation in simulation requires accurate modelling of hydrostatic forces and floatability. The “asv_wave_sim” package, designed for simulating ASVs in Gazebo, includes a hydrostatics plugin for buoyancy and floatation, wave generation, and water surface dynamics, enhancing navigation realism. It also offers prebuilt models like floating buoys and waste objects for scenarios such as obstacle avoidance and waste collection. Fully compatible with ROS Noetic and Ubuntu 20.04, this package ensures realistic hydrodynamic responses, supporting advanced navigation testing.

5.3 SLAM and Obstacle Avoidance

Autonomous exploration and mapping are achieved through the integration of SLAM (Simultaneous Localization and Mapping) using LiDAR data. The `gmapping` package was employed to build a 2D occupancy grid map of the environment while simultaneously localizing the boat within it. As the boat moves, LiDAR continuously scans the surroundings, and the collected data is processed in real-time to detect obstacles and incrementally update the map.

For collision-free navigation, an obstacle avoidance layer was implemented using the `move_base` package. The global planner determines an optimal path to the target location, and the local planner dynamically adjusts the trajectory to avoid unexpected obstacles.

This integration allows the boat to navigate autonomously through complex environments, avoid floating hazards, and reach assigned goals reliably.

5.4 Waypoint Planning Interface

I developed a custom desktop application to support waste detection and autonomous navigation planning. The application integrates a real-world map using Google Maps, providing an interactive interface for defining waypoints relevant to each mission. Two waypoint creation modes are available: manual and automatic. In manual mode, I can select specific locations on the map—typically based on visual identification of waste or target zones. In automatic mode, the application generates a systematic grid of waypoints within a selected area to ensure complete coverage.

The generated waypoints are automatically converted into a format compatible with ROS. A dedicated ROS node handles communication between the application and the robot, publishing each waypoint as a navigation goal for the autonomous boat.

This tool streamlines mission setup and enhances both flexibility and efficiency in waste collection scenarios.

6. Waste Detection Simulation