

# TurtleBot3 ROS2 Humble C++ Development Guide

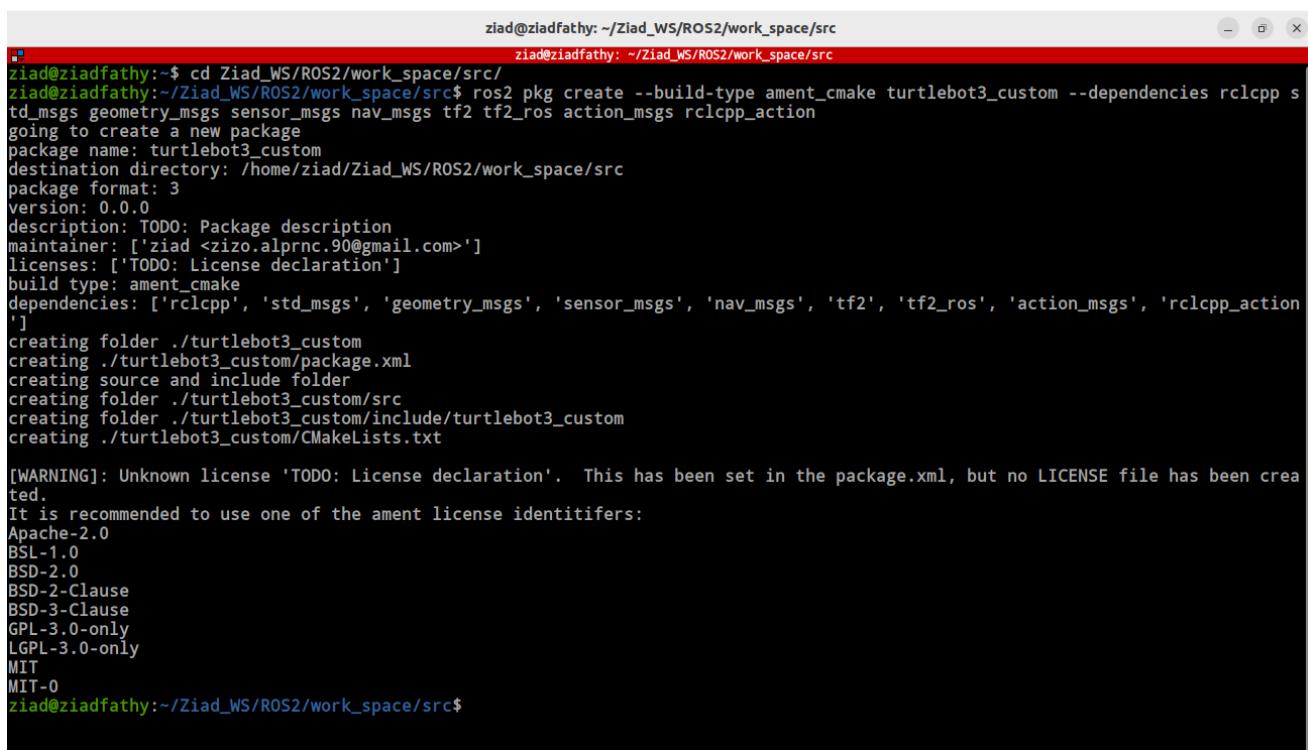
## Prerequisites

Before starting, ensure you have:

- ROS2 Humble installed TurtleBot3
- packages installed
- A workspace set up

## 1. Creating a ROS2 Package

First, create a new package for your TurtleBot3 project:



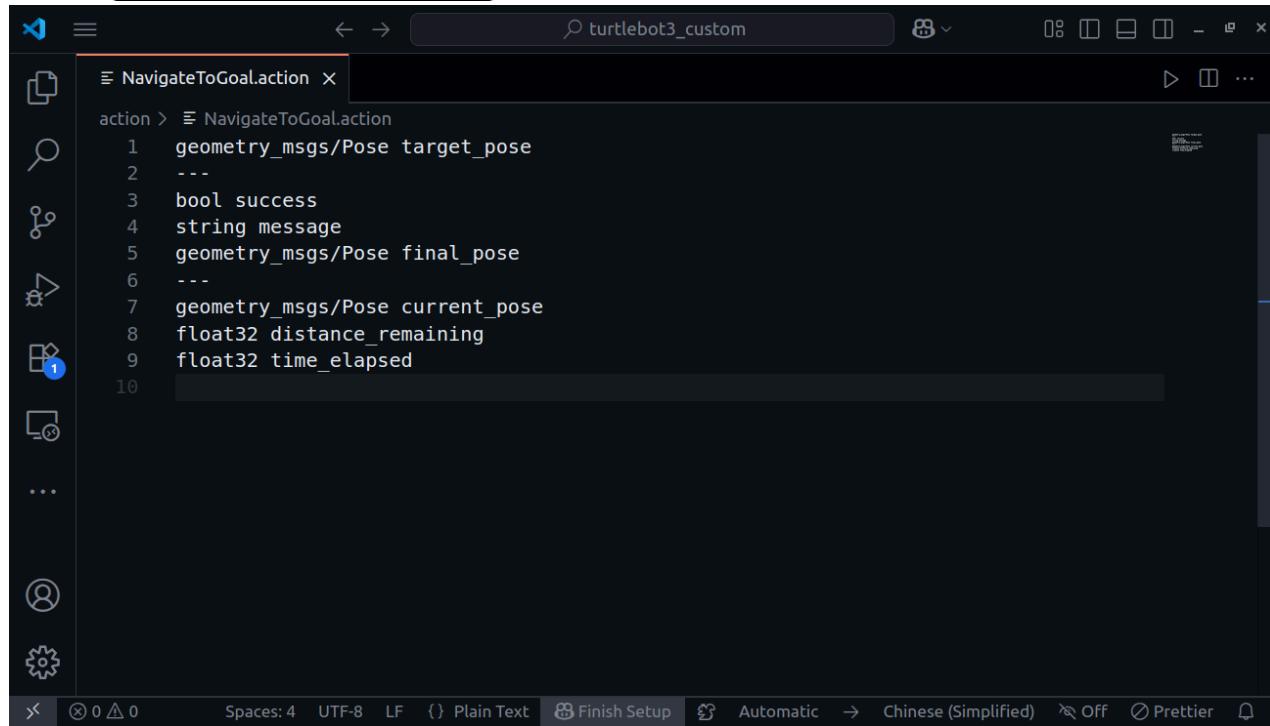
```
ziad@ziadfathy:~/Ziad_WS/ROS2/work_space/src
ziad@ziadfathy:~/Ziad_WS/ROS2/work_space/src$ ros2 pkg create --build-type ament_cmake turtlebot3_custom --dependencies rclcpp std_msgs geometry_msgs sensor_msgs nav_msgs tf2 tf2_ros action_msgs rclcpp_action
going to create a new package
package name: turtlebot3_custom
destination directory: /home/ziad/Ziad_WS/ROS2/work_space/src
package format: 3
version: 0.0.0
description: TODO: Package description
maintainer: ['ziad <zizo.alprnc.90@gmail.com>']
licenses: ['TODO: License declaration']
build type: ament_cmake
dependencies: ['rclcpp', 'std_msgs', 'geometry_msgs', 'sensor_msgs', 'nav_msgs', 'tf2', 'tf2_ros', 'action_msgs', 'rclcpp_action']
creating folder ./turtlebot3_custom
creating ./turtlebot3_custom/package.xml
creating source and include folder
creating folder ./turtlebot3_custom/src
creating folder ./turtlebot3_custom/include/turtlebot3_custom
creating ./turtlebot3_custom/CMakeLists.txt

[WARNING]: Unknown license 'TODO: License declaration'. This has been set in the package.xml, but no LICENSE file has been created.
It is recommended to use one of the ament license identifiers:
Apache-2.0
BSL-1.0
BSD-2.0
BSD-2-Clause
BSD-3-Clause
GPL-3.0-only
LGPL-3.0-only
MIT
MIT-0
ziad@ziadfathy:~/Ziad_WS/ROS2/work_space/src$
```

## 2. Creating Custom Interfaces

### Custom Action Definition

Create `action/NavigateToGoal.action`:



The screenshot shows a code editor window titled "turtlebot3\_custom". The file being edited is "NavigateToGoal.action". The code content is as follows:

```
action > NavigateToGoal.action
1 geometry_msgs/Pose target_pose
2 ---
3 bool success
4 string message
5 geometry_msgs/Pose final_pose
6 ---
7 geometry_msgs/Pose current_pose
8 float32 distance_remaining
9 float32 time_elapsed
```

The editor interface includes a toolbar with icons for file operations, a status bar at the bottom showing file statistics like "Spaces: 4", and a tab bar indicating the file type is "Plain Text".

## Custom Service Definition

Create `srv/GetRobotStatus.srv`:

The screenshot shows a code editor interface with a dark theme. The title bar says "turtlebot3\_custom". The left sidebar has icons for file operations, search, and other settings. The main editor area contains the following code:

```
GetRobotStatus.srv X
srv > GetRobotStatus.srv
1 string query_type
2 ---
3 bool success
4 string status_message
5 geometry_msgs/Pose current_pose
6 sensor_msgs/BatteryState battery_info
7
```

The code defines a service named "GetRobotStatus.srv" with a single request message "GetRobotStatus.srv". This message has fields for "query\_type" (string), "success" (bool), "status\_message" (string), "current\_pose" (geometry\_msgs/Pose), and "battery\_info" (sensor\_msgs/BatteryState).

At the bottom, there are various status indicators and tool buttons: "Plain Text", "Finish Setup", "Automatic", "Chinese (Simplified)", "off", "Prettier", and a bell icon.

# Update CMakeLists.txt for Custom Interfaces

Add to your **CMakeLists.txt**:

```
File Edit Selection View ... ← → ⌂ turtlebot3_custom ⌂ CMakeLists.txt X
1 cmake_minimum_required(VERSION 3.8)
2 project(turtlebot3_custom)
3
4 if(CMAKE_COMPILER_IS_GNUCXX OR CMAKE_CXX_COMPILER_ID MATCHES "Clang")
5 | add_compile_options(-Wall -Wextra -Wpedantic)
6 endif()
7
8 # find dependencies
9 find_package(ament_cmake REQUIRED)
10 find_package(rclcpp REQUIRED)
11 find_package(rclcpp_action REQUIRED)
12 find_package(std_msgs REQUIRED)
13 find_package(geometry_msgs REQUIRED)
14 find_package(sensor_msgs REQUIRED)
15 find_package(nav_msgs REQUIRED)
16 find_package(tf2 REQUIRED)
17 find_package(tf2_ros REQUIRED)
18 find_package(tf2_geometry_msgs REQUIRED)
19 find_package(rosidl_default_generators REQUIRED)
20
21 rosidl_generate_interfaces(${PROJECT_NAME})
22 "action/NavigateToGoal.action"
23 "srv/GetRobotStatus.srv"
24 DEPENDENCIES geometry_msgs sensor_msgs
25 )
26
27 add_executable(turtlebot3_controller src/turtlebot3_controller.cpp)
28 add_executable(robot_status_server src/robot_status_server.cpp)
29 add_executable(status_client src/status_client.cpp)
30 add_executable(navigation_action_server src/navigation_action_server.cpp)
31 add_executable(navigation_client src/navigation_client.cpp)
32
33 ament_target_dependencies turtlebot3_controller
34 rclcpp std_msgs geometry_msgs sensor_msgs nav_msgs tf2 tf2_ros tf2_geometry_msgs
35
36 ament_target_dependencies robot_status_server
37 rclcpp geometry_msgs sensor_msgs nav_msgs
38
39 ament_target_dependencies status_client
40 rclcpp
41
42
43
44
45
46
47 ament_target_dependencies navigation_client
48 rclcpp
49 rclcpp_action
50 geometry_msgs
51 nav_msgs
52
53
54 rosidl_get_typesupport_target(robot_status_server typesupport ${PROJECT_NAME} "rosidl_typesupport_cpp")
55 target_link_libraries(robot_status_server ${robot_status_server_typesupport})
56
57 rosidl_get_typesupport_target(status_client typesupport ${PROJECT_NAME} "rosidl_typesupport_cpp")
58 target_link_libraries(status_client ${status_client_typesupport})
59
60 rosidl_get_typesupport_target(navigation_action_server typesupport ${PROJECT_NAME} "rosidl_typesupport_cpp")
61 target_link_libraries(navigation_action_server ${navigation_action_server_typesupport})
62
63 rosidl_get_typesupport_target(navigation_client typesupport ${PROJECT_NAME} "rosidl_typesupport_cpp")
64 target_link_libraries(navigation_client ${navigation_client_typesupport})
65
66 install(TARGETS
67   turtlebot3_controller
68   robot_status_server
69   status_client
70   navigation_action_server
71   navigation_client
72   DESTINATION lib/${PROJECT_NAME}
73 )
74
75 install(DIRECTORY launch
76   DESTINATION share/${PROJECT_NAME}/
77 )
78
79 if(BUILD_TESTING)
80   find_package(ament_lint_auto REQUIRED)
81   # the following line skips the linter which checks for copyrights
82   # comment the line when a copyright and license is added to all source files
83   set(ament_cmake_copyright_FOUND TRUE)
84   # the following line skips cpplint (only works in a git repo)
85   # comment the line when this package is in a git repo and when
```

```
Ln 53, Col 1 Spaces:2 UTF-8 LF () CMake ⌂ Finish Setup ⌂ Automatic → Chinese (Simplified) ⌂ Off ⌂ Prettier ⌂
File Edit Selection View ... ← → ⌂ turtlebot3_custom ⌂ CMakeLists.txt X
46
47 ament_target_dependencies navigation_client
48 rclcpp
49 rclcpp_action
50 geometry_msgs
51 nav_msgs
52
53
54 rosidl_get_typesupport_target(robot_status_server typesupport ${PROJECT_NAME} "rosidl_typesupport_cpp")
55 target_link_libraries(robot_status_server ${robot_status_server_typesupport})
56
57 rosidl_get_typesupport_target(status_client typesupport ${PROJECT_NAME} "rosidl_typesupport_cpp")
58 target_link_libraries(status_client ${status_client_typesupport})
59
60 rosidl_get_typesupport_target(navigation_action_server typesupport ${PROJECT_NAME} "rosidl_typesupport_cpp")
61 target_link_libraries(navigation_action_server ${navigation_action_server_typesupport})
62
63 rosidl_get_typesupport_target(navigation_client typesupport ${PROJECT_NAME} "rosidl_typesupport_cpp")
64 target_link_libraries(navigation_client ${navigation_client_typesupport})
65
66 install(TARGETS
67   turtlebot3_controller
68   robot_status_server
69   status_client
70   navigation_action_server
71   navigation_client
72   DESTINATION lib/${PROJECT_NAME}
73 )
74
75 install(DIRECTORY launch
76   DESTINATION share/${PROJECT_NAME}/
77 )
78
79 if(BUILD_TESTING)
80   find_package(ament_lint_auto REQUIRED)
81   # the following line skips the linter which checks for copyrights
82   # comment the line when a copyright and license is added to all source files
83   set(ament_cmake_copyright_FOUND TRUE)
84   # the following line skips cpplint (only works in a git repo)
85   # comment the line when this package is in a git repo and when
```

### 3. Creating a Basic Node

## Basic TurtleBot3 Controller Node

Create `[src/turtlebot3_controller.cpp]`:

The image shows a code editor with two tabs open, both titled `turtlebot3_controller.cpp`. The left tab displays the full class definition and main function, while the right tab shows a detailed view of the `determine_next_state` method.

```
File Edit Selection View ... ← → turtlebot3_custom
```

```
src > E turtlebot3_controller.cpp
30  class TurtleBot3Controller : public rclcpp::Node
256 std::string state_to_string(RobotState state)
258     switch (state)
259     {
260         case SEARCHING_PATH:
261             return "SEARCHING_PATH";
262         default:
263             return "UNKNOWN";
264     }
265
266     rclcpp::Publisher<geometry_msgs::msg::Twist>::SharedPtr cmd_vel_pub_;
267     rclcpp::Subscription<sensor_msgs::msg::LaserScan>::SharedPtr laser_sub_;
268     rclcpp::Subscription<nav_msgs::msg::Odometry>::SharedPtr odom_sub_;
269     rclcpp::TimerBase::SharedPtr control_timer_;
270
271     float front_distance_ = std::numeric_limits<float>::max();
272     float left_distance_ = std::numeric_limits<float>::max();
273     float right_distance_ = std::numeric_limits<float>::max();
274     float front_wide_distance_ = std::numeric_limits<float>::max();
275
276     geometry_msgs::msg::Pose current_pose_;
277     double current_yaw_ = 0.0;
278
279     RobotState current_state_;
280     rclcpp::Time state_start_time_;
281
282     int debug_counter_ = 0;
283
284 int main(int argc, char **argv)
285 {
286     rclcpp::init(argc, argv);
287     rclcpp::spin(std::make_shared<TurtleBot3Controller>());
288     rclcpp::shutdown();
289     return 0;
290 }
```

```
File Edit Selection View ... ← → turtlebot3_custom
```

```
src > E turtlebot3_controller.cpp
30  class TurtleBot3Controller : public rclcpp::Node
129 void control_loop()
130 {
131     RobotState determine_next_state(double time_in_state)
132     {
133         const float obstacle_threshold = 0.4f;
134         const float safe_distance = 0.8f;
135         const double max_turn_time = 3.0;
136
137         switch (current_state_)
138         {
139             case RobotState::MOVING_FORWARD:
140                 if (front_distance_ < obstacle_threshold || front_wide_distance_ < 0.6f)
141                 {
142                     if (left_distance_ > right_distance_)
143                     {
144                         return RobotState::TURNING_LEFT;
145                     }
146                     else
147                     {
148                         return RobotState::TURNING_RIGHT;
149                     }
150                 }
151                 return RobotState::MOVING_FORWARD;
152
153             case RobotState::TURNING_LEFT:
154                 if (time_in_state > max_turn_time)
155                 {
156                     return RobotState::TURNING_RIGHT;
157                 }
158                 if (front_distance_ > safe_distance && front_wide_distance_ > safe_distance)
159                 {
160                     return RobotState::MOVING_FORWARD;
161                 }
162                 return RobotState::TURNING_LEFT;
163
164             case RobotState::TURNING_RIGHT:
165                 if (time_in_state > max_turn_time)
166                 {
167                     return RobotState::TURNING_LEFT;
168                 }
169         }
170     }
171 }
```

```
File Edit Selection View < > turtlebot3_custom
src > turtlebot3_controller.cpp
30 class TurtleBot3Controller : public rclcpp::Node
53     void laser_callback(const sensor_msgs::msg::LaserScan::SharedPtr msg)
79     }
80
81     float get_min_distance_in_sector(const sensor_msgs::msg::LaserScan::SharedPtr &msg,
82                                     size_t start_idx, size_t end_idx)
83     {
84         float min_dist = std::numeric_limits<float>::max();
85
86         for (size_t i = start_idx; i <= end_idx && i < msg->ranges.size(); ++i)
87         {
88             if (msg->ranges[i] > msg->range_min && msg->ranges[i] < msg->range_max)
89             {
90                 min_dist = std::min(min_dist, msg->ranges[i]);
91             }
92         }
93
94         return (min_dist == std::numeric_limits<float>::max()) ? 10.0f : min_dist;
95     }
96
97     void odom_callback(const nav_msgs::msg::Odometry::SharedPtr msg)
98     {
99         current_pose_ = msg->pose.pose;
100
101         tf2::Quaternion quat;
102         tf2::fromMsg(current_pose_.orientation, quat);
103         double roll, pitch, yaw;
104         tf2::Matrix3x3(quat).getRPY(roll, pitch, yaw);
105         current_yaw_ = yaw;
106     }
107
108     void control_loop()
109     {
110         auto twist = geometry_msgs::msg::Twist();
111         auto current_time = this->now();
112         auto time_in_state = (current_time - state_start_time_).seconds();
113
114         RobotState new_state = determine_next_state(time_in_state);
115
116         if (new_state != current_state_)
```

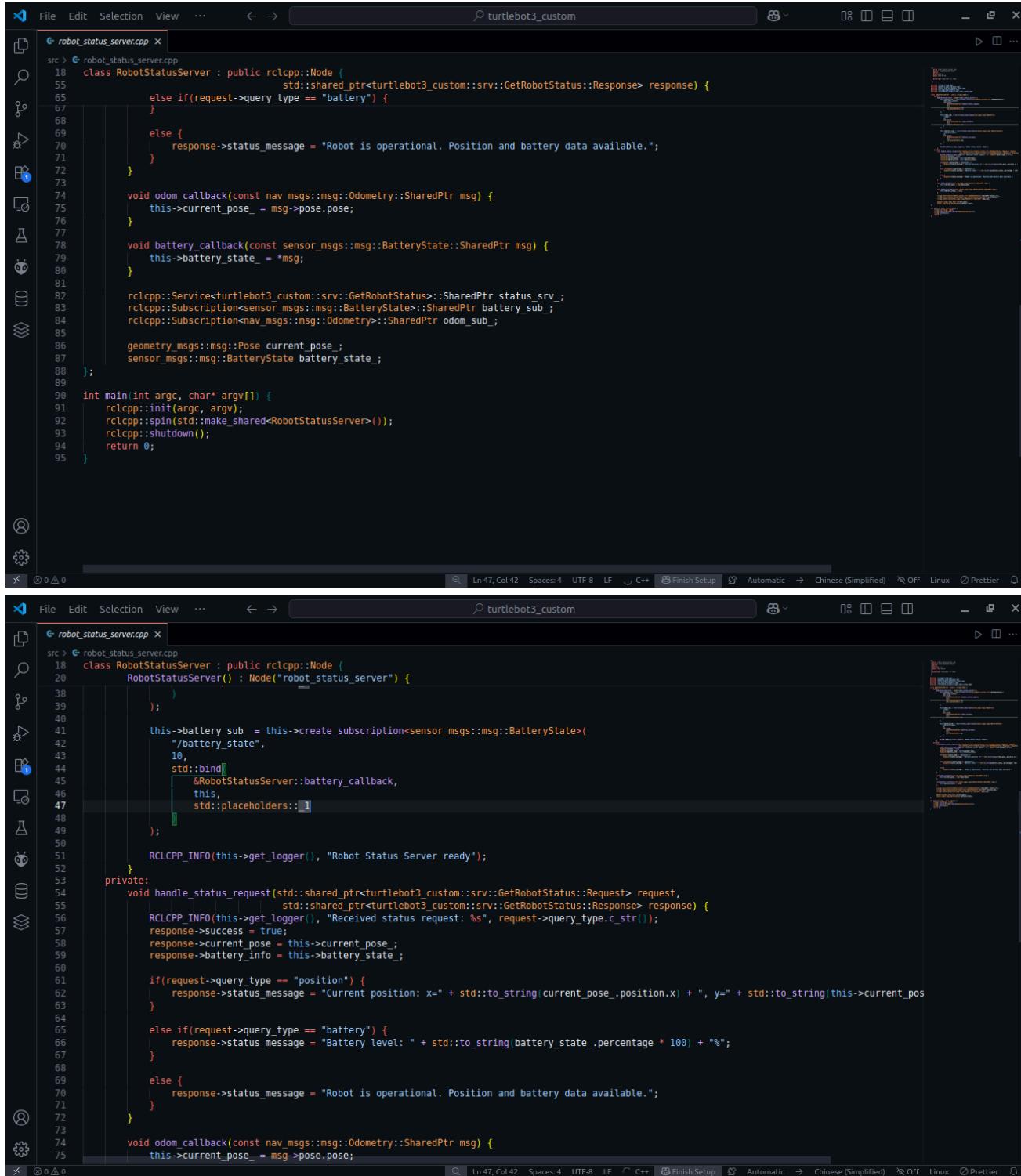


```
File Edit Selection View ... ← → turtlebot3_custom turtlebot3_controller.cpp src > turtlebot3_controller.cpp
1 /**
2  * @file turtlebot3_controller.cpp
3  * @author Ziad Mohammed Fathy
4  * @brief
5  * @version 0.1
6  * @date 2025-08-29
7  *
8  * @copyright Copyright (c) 2025
9  *
10 */
11
12 #include <rclcpp/rclcpp.hpp>
13 #include <geometry_msgs/msg/twist.hpp>
14 #include <sensor_msgs/msg/laser_scan.hpp>
15 #include <nav_msgs/msg/odometry.hpp>
16 #include <tf2/LinearMath/Quaternion.h>
17 #include <tf2_geometry_msgs/tf2_geometry_msgs.hpp>
18 #include <algorithm>
19 #include <vector>
20
21 enum class RobotState
22 {
23     MOVING_FORWARD,
24     TURNING_LEFT,
25     TURNING_RIGHT,
26     OBSTACLE_DETECTED,
27     SEARCHING_PATH
28 };
29
30 class TurtleBot3Controller : public rclcpp::Node
31 {
32 public:
33     TurtleBot3Controller() : Node("turtlebot3_controller")
34     {
35         cmd_vel_pub_ = this->create_publisher<geometry_msgs::msg::Twist>("/cmd_vel", 10);
36
37         laser_sub_ = this->create_subscription<sensor_msgs::msg::LaserScan>(
38             "/scan", 10, std::bind(&TurtleBot3Controller::laser_callback, this, std::placeholders::_1));
39         odom_sub_ = this->create_subscription<nav_msgs::msg::Odometry>(
40             "/odom", 10, std::bind(&TurtleBot3Controller::odom_callback, this, std::placeholders::_1));
41     }
42 }
```

# 4. Creating a Service Server

## Service Server Node

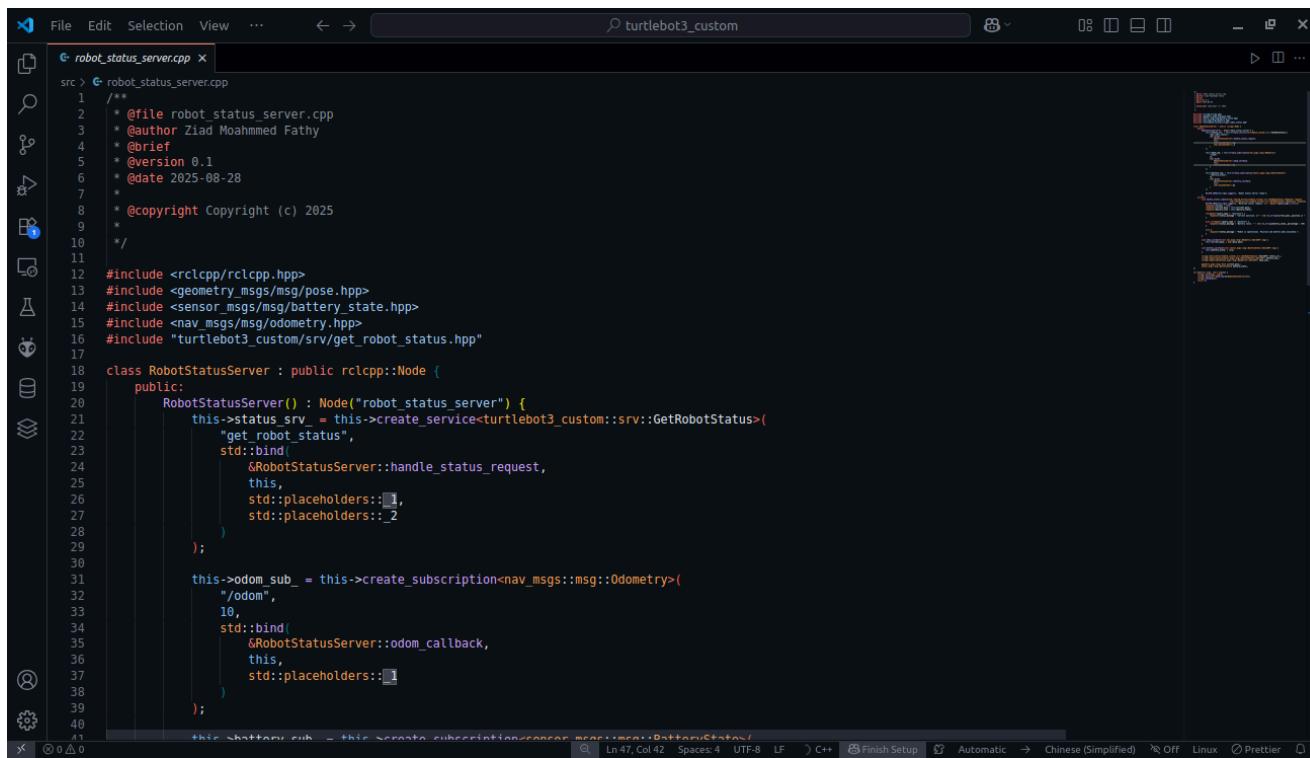
Create [src/robot\_status\_server.cpp]:



```
File Edit Selection View ... ← → ⌘ turtlebot3_custom
src > ⌘ robot_status_server.cpp
18 class RobotStatusServer : public rclcpp::Node {
55     std::shared_ptr<turtlebot3_custom::srv::GetRobotStatus::Response> response_ { nullptr };
65     else if(request->query_type == "battery") {
66     }
67     else {
68         response->status_message = "Robot is operational. Position and battery data available.";
69     }
70 }
71
72 void odom_callback(const nav_msgs::msg::Odometry::SharedPtr msg) {
73     this->current_pose_ = msg->pose.pose;
74 }
75
76 void battery_callback(const sensor_msgs::msg::BatteryState::SharedPtr msg) {
77     this->battery_state_ = *msg;
78 }
79
80 rclcpp::Service<turtlebot3_custom::srv::GetRobotStatus>::SharedPtr status_srv_{};
81 rclcpp::Subscription<sensor_msgs::msg::BatteryState>::SharedPtr battery_sub_{};
82 rclcpp::Subscription<nav_msgs::msg::Odometry>::SharedPtr odom_sub_{};
83
84 geometry_msgs::msg::Pose current_pose_;
85 sensor_msgs::msg::BatteryState battery_state_;
86
87 };
88
89 int main(int argc, char* argv[]) {
90     rclcpp::init(argc, argv);
91     rclcpp::spin(std::make_shared<RobotStatusServer>());
92     rclcpp::shutdown();
93     return 0;
94 }
```

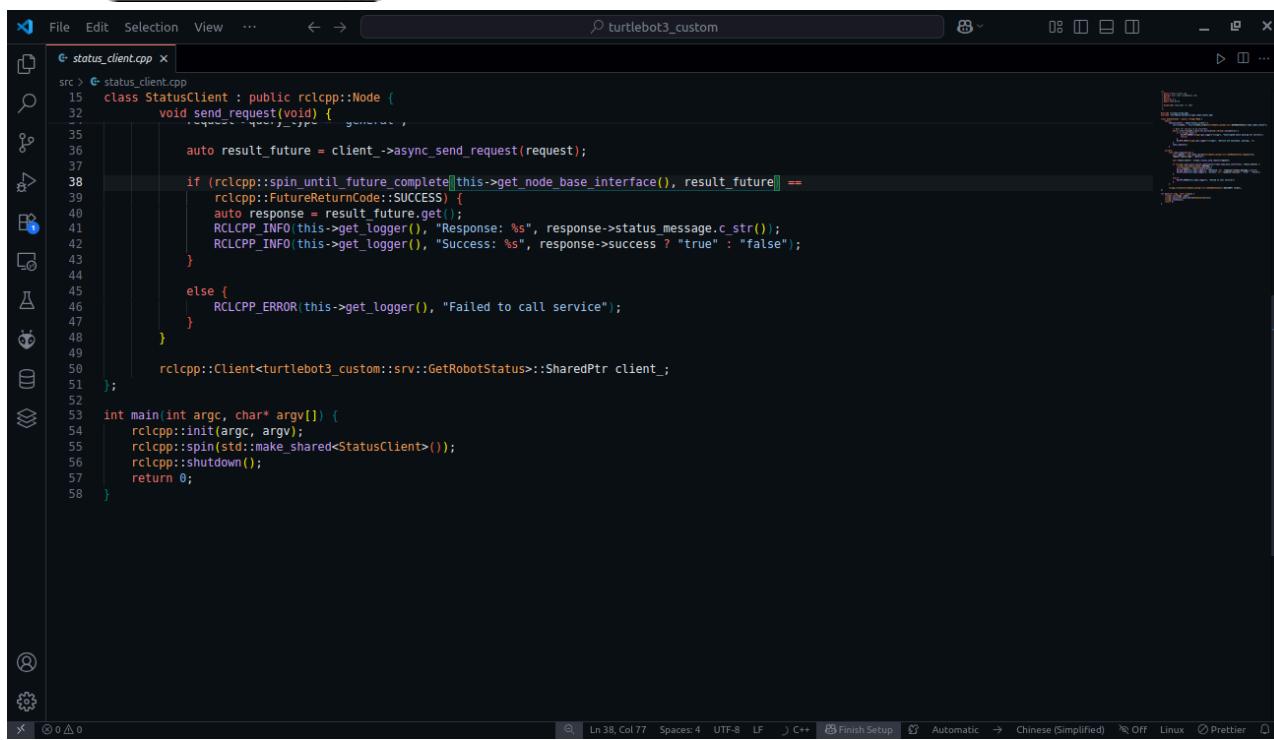
```
File Edit Selection View ... ← → ⌘ turtlebot3_custom
src > ⌘ robot_status_server.cpp
18 class RobotStatusServer : public rclcpp::Node {
20     RobotStatusServer() : Node("robot_status_server") {
21
22     }
23
24     this->battery_sub_ = this->create_subscription<sensor_msgs::msg::BatteryState>(
25         "battery_state",
26         10,
27         std::bind(
28             &RobotStatusServer::battery_callback,
29             this,
30             std::placeholders::_1
31         );
32
33     RCLCPP_INFO(this->get_logger(), "Robot Status Server ready");
34 }
35
36 private:
37     void handle_status_request(std::shared_ptr<turtlebot3_custom::srv::GetRobotStatus::Request> request,
38         std::shared_ptr<turtlebot3_custom::srv::GetRobotStatus::Response> response) {
39         RCLCPP_INFO(this->get_logger(), "Received status request: %s", request->query_type.c_str());
40         response->success = true;
41         response->current_pose = this->current_pose_;
42         response->battery_info = this->battery_state_;
43
44         if(request->query_type == "position") {
45             response->status_message = "Current position: x=" + std::to_string(current_pose_.position.x) + ", y=" + std::to_string(this->current_pos_
46         }
47
48         else if(request->query_type == "battery") {
49             response->status_message = "Battery level: " + std::to_string(battery_state_.percentage * 100) + "%";
50         }
51
52         else {
53             response->status_message = "Robot is operational. Position and battery data available.";
54         }
55
56     void odom_callback(const nav_msgs::msg::Odometry::SharedPtr msg) {
57         this->current_pose_ = msg->pose.pose;
58     }
59 }
```



```
src > robot_status_server.cpp
1 /**
2 * @file robot_status_server.cpp
3 * @author Ziad Moahmed Fathy
4 * @brief
5 * @version 0.1
6 * @date 2025-08-28
7 *
8 * @copyright Copyright (c) 2025
9 *
10 */
11
12 #include <rclcpp/rclcpp.hpp>
13 #include <geometry_msgs/msg/pose.hpp>
14 #include <sensor_msgs/msg/battery_state.hpp>
15 #include <nav_msgs/msg/odometry.hpp>
16 #include "turtlebot3_custom/srv/get_robot_status.hpp"
17
18 class RobotStatusServer : public rclcpp::Node {
19 public:
20     RobotStatusServer() : Node("robot_status_server") {
21         this->status_srv_ = this->create_service<turtlebot3_custom::srv::GetRobotStatus>(
22             "get_robot_status",
23             std::bind(
24                 &RobotStatusServer::handle_status_request,
25                 this,
26                 std::placeholders::_1,
27                 std::placeholders::_2
28             );
29         );
30
31         this->odom_sub_ = this->create_subscription<nav_msgs::msg::Odometry>(
32             "/odom",
33             10,
34             std::bind(
35                 &RobotStatusServer::odom_callback,
36                 this,
37                 std::placeholders::_1
38             );
39         );
40
41         this->shutdown_sub_ = this->create_subscription<sensor_msgs::msg::BatteryState>(
42             "/battery",
43             10,
44             std::bind(
45                 &RobotStatusServer::battery_callback,
46                 this,
47                 std::placeholders::_1
48             );
49     }
50
51     void handle_status_request(
52         const rclcpp::RequestHeader &request,
53         rclcpp::ResponseHeader &response) {
54
55         auto result_future = client_->async_send_request(request);
56
57         if (rclcpp::spin_until_future_complete(this->get_node_base_interface(), result_future) ==
58             rclcpp::FutureReturnCode::SUCCESS) {
59             auto response = result_future.get();
60             RCLCPP_INFO(this->get_logger(), "Response: %s", response->status_message.c_str());
61             RCLCPP_INFO(this->get_logger(), "Success: %s", response->success ? "true" : "false");
62         }
63
64         else {
65             RCLCPP_ERROR(this->get_logger(), "Failed to call service");
66         }
67     }
68
69     void odom_callback(
70         const nav_msgs::msg::Odometry &odom) {
71
72         // Process the Odometry message here
73     }
74
75     void battery_callback(
76         const sensor_msgs::msg::BatteryState &battery) {
77
78         // Process the Battery State message here
79     }
80
81     rclcpp::Client<turtlebot3_custom::srv::GetRobotStatus>::SharedPtr client_;
82 };
83
84 int main(int argc, char* argv[]) {
85     rclcpp::init(argc, argv);
86     rclcpp::spin(std::make_shared<StatusClient>());
87     rclcpp::shutdown();
88     return 0;
89 }
```

## Service Client Example

Create `src/status_client.cpp`:



```
src > status_client.cpp
15 class StatusClient : public rclcpp::Node {
16 public:
17     void send_request(void) {
18
19         auto request = std::make_shared<rclcpp::api::RequestHeader>();
20
21         auto result_future = client_->async_send_request(request);
22
23         if (rclcpp::spin_until_future_complete(this->get_node_base_interface(), result_future) ==
24             rclcpp::FutureReturnCode::SUCCESS) {
25             auto response = result_future.get();
26             RCLCPP_INFO(this->get_logger(), "Response: %s", response->status_message.c_str());
27             RCLCPP_INFO(this->get_logger(), "Success: %s", response->success ? "true" : "false");
28         }
29
30         else {
31             RCLCPP_ERROR(this->get_logger(), "Failed to call service");
32         }
33     }
34
35     rclcpp::Client<turtlebot3_custom::srv::GetRobotStatus>::SharedPtr client_;
36 };
37
38 int main(int argc, char* argv[]) {
39     rclcpp::init(argc, argv);
40     rclcpp::spin(std::make_shared<StatusClient>());
41     rclcpp::shutdown();
42     return 0;
43 }
```

The screenshot shows a code editor window with the following details:

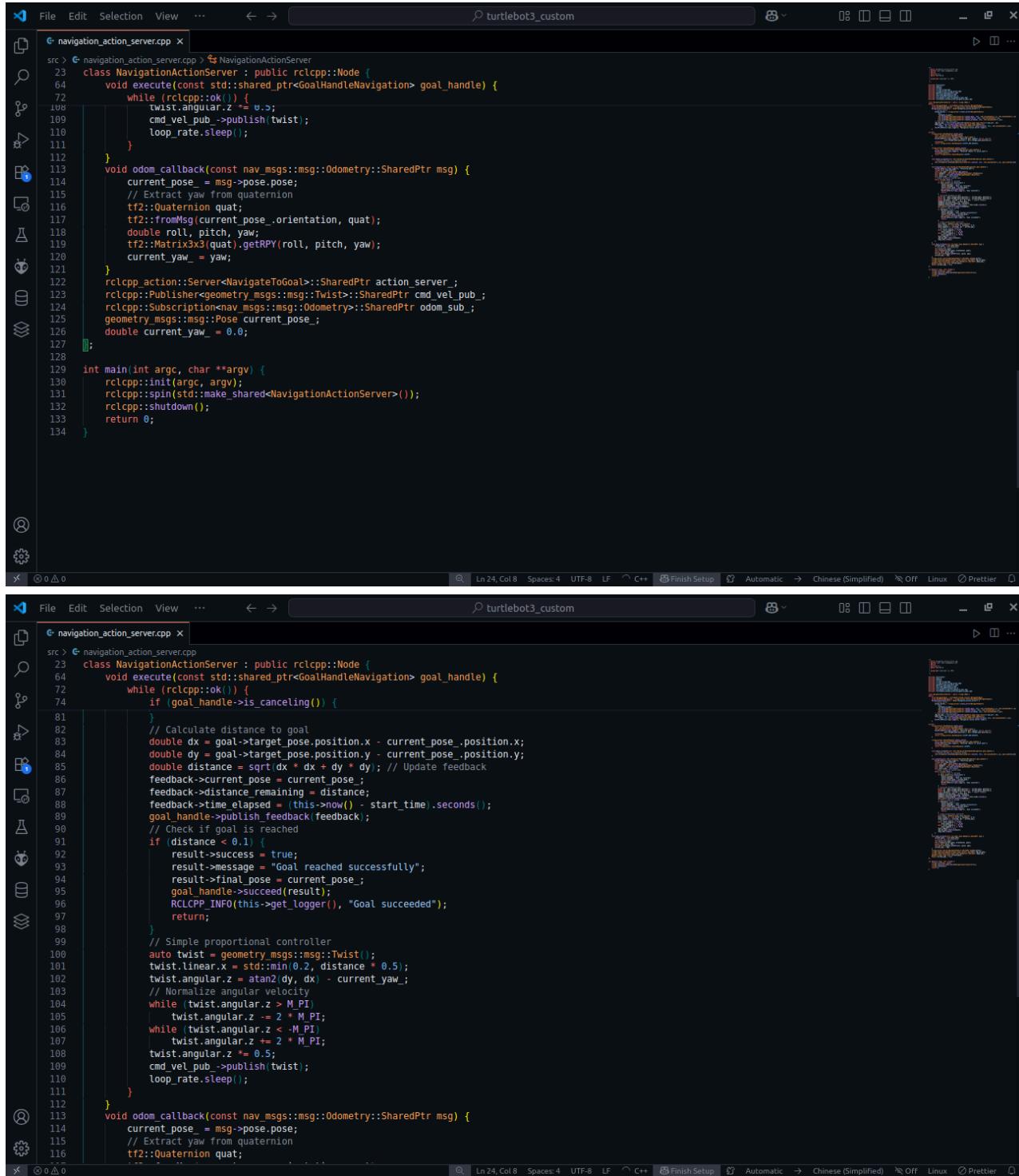
- Title Bar:** File Edit Selection View ... < > turtlebot3\_custom
- File Path:** src > status\_client.cpp
- Code Content:** The code is a C++ file named status\_client.cpp. It includes rclcpp/rclcpp.hpp and turtlebot3\_custom/srv/get\_robot\_status.hpp. It defines a class StatusClient that inherits from rclcpp::Node. The constructor creates a client for the get\_robot\_status service. A while loop waits for the service to be available, checking if the client is OK every second. If not OK, it logs an error and returns. If OK, it sends a request and logs a message. The send\_request function creates a shared request object, sets the query type to "general", and sends it asynchronously. It then spins until the future is complete, checking the return code. If successful, it gets the response.
- Status Bar:** Ln 38, Col 77 Spaces:4 UTF-8 LF C C++ Finish Setup Automatic → Chinese (Simplified) Off Linux Prettier

```
1 /**
2  * @file status_client.cpp
3  * @author your name (you@domain.com)
4  * @brief
5  * @version 0.1
6  * @date 2025-08-28
7  *
8  * @copyright Copyright (c) 2025
9  *
10 */
11
12 #include <rclcpp/rclcpp.hpp>
13 #include "turtlebot3_custom/srv/get_robot_status.hpp"
14
15 class StatusClient : public rclcpp::Node {
16 public:
17     StatusClient() : Node("status_client") {
18         this->client_ = this->create_client<turtlebot3_custom::srv::GetRobotStatus>("get_robot_status");
19
20         // Wait for service to be available
21         while (!this->client_->wait_for_service(std::chrono::seconds(1))) {
22             if (!rclcpp::ok()) {
23                 RCLCPP_ERROR(rclcpp::get_logger("rclcpp"), "Interrupted while waiting for service");
24                 return;
25             }
26             RCLCPP_INFO(rclcpp::get_logger("rclcpp"), "Service not available, waiting...");
27         }
28         send_request();
29     }
30
31 private:
32     void send_request(void) {
33         auto request = std::make_shared<turtlebot3_custom::srv::GetRobotStatus::Request>();
34         request->query_type = "general";
35
36         auto result_future = client_->async_send_request(request);
37
38         if (rclcpp::spin_until_future_complete(this->get_node_base_interface(), result_future) ==
39             rclcpp::FutureReturnCode::SUCCESS) {
40             auto response = result_future.get();
41             RCLCPP_INFO(this->get_logger(), "Received response: %s", response.message.c_str());
42         }
43     }
44 }
```

# 5. Creating an Action Server

## Action Server Node

Create `src/navigation_action_server.cpp`:



```
File Edit Selection View ... ← → turtlebot3_custom
```

```
navigation_action_server.cpp
```

```
src > navigation_action_server.cpp > NavigationActionServer : public rclcpp::Node {
23     void execute(const std::shared_ptr<GoalHandleNavigation> goal_handle) {
24         while (rclcpp::ok()) {
25             twist.angular.z = 0.0;
26             cmd_vel_pub_.publish(twist);
27             loop_rate.sleep();
28         }
29     }
30     void odom_callback(const nav_msgs::msg::Odometry::SharedPtr msg) {
31         current_pose_ = msg->pose.pose;
32         // Extract yaw from quaternion
33         tf2::Quaternion quat;
34         tf2::fromMsg(current_pose_.orientation, quat);
35         double roll, pitch, yaw;
36         tf2::Matrix3x3(quat).getRPY(roll, pitch, yaw);
37         current_yaw_ = yaw;
38     }
39     rclcpp_action::Server<NavigateToGoal>::SharedPtr action_server_;
40     rclcpp::Publisher<geometry_msgs::msg::Twist>::SharedPtr cmd_vel_pub_;
41     rclcpp::Subscription<nav_msgs::msg::Odometry>::SharedPtr odom_sub_;
42     geometry_msgs::msg::Pose current_pose_;
43     double current_yaw_ = 0.0;
44 }
45
46 int main(int argc, char **argv) {
47     rclcpp::init(argc, argv);
48     rclcpp::spin(std::make_shared<NavigationActionServer>());
49     rclcpp::shutdown();
50     return 0;
51 }
```

```
File Edit Selection View ... ← → turtlebot3_custom
```

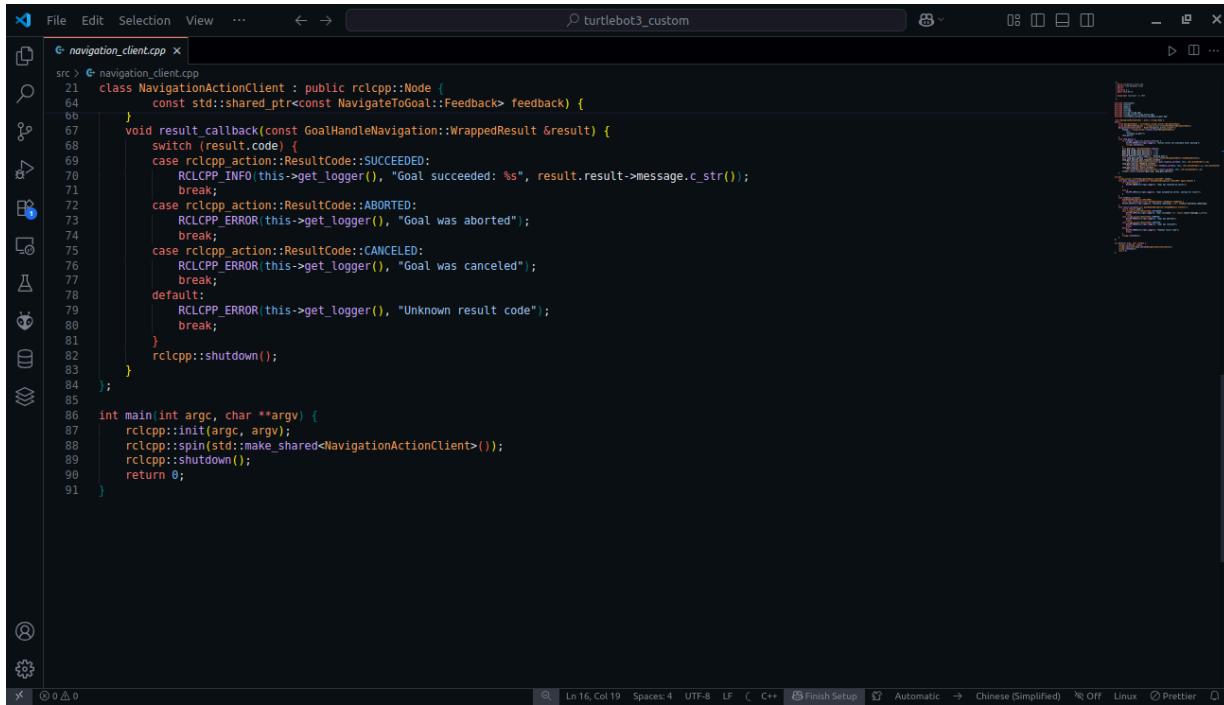
```
navigation_action_server.cpp
```

```
src > navigation_action_server.cpp > NavigationActionServer : public rclcpp::Node {
23     void execute(const std::shared_ptr<GoalHandleNavigation> goal_handle) {
24         while (rclcpp::ok()) {
25             if (goal_handle->is_cancelling()) {
26                 ...
27             }
28             // Calculate distance to goal
29             double dx = goal->target_pose.position.x - current_pose_.position.x;
30             double dy = goal->target_pose.position.y - current_pose_.position.y;
31             double distance = sqrt(dx * dx + dy * dy); // Update feedback
32             feedback->current_pose = current_pose_;
33             feedback->distance_remaining = distance;
34             feedback->time_elapsed = (this->now() - start_time).seconds();
35             goal_handle->publish_feedback(feedback);
36             // Check if goal is reached
37             if (distance < 0.1) {
38                 result->success = true;
39                 result->message = "Goal reached successfully";
40                 result->final_pose = current_pose_;
41                 goal_handle->succeed(result);
42                 RCLCPP_INFO(this->get_logger(), "Goal succeeded");
43                 return;
44             }
45             // Simple proportional controller
46             auto twist = geometry_msgs::msg::Twist();
47             twist.linear.x = std::min(0.2, distance * 0.5);
48             twist.angular.z = atan2(dy, dx) - current_yaw_;
49             // Normalize angular velocity
50             while (twist.angular.z > M_PI)
51                 twist.angular.z -= 2 * M_PI;
52             while (twist.angular.z < -M_PI)
53                 twist.angular.z += 2 * M_PI;
54             twist.angular.z *= 0.5;
55             cmd_vel_pub_.publish(twist);
56             loop_rate.sleep();
57         }
58     }
59     void odom_callback(const nav_msgs::msg::Odometry::SharedPtr msg) {
60         current_pose_ = msg->pose.pose;
61         // Extract yaw from quaternion
62         tf2::Quaternion quat;
63         ...
64     }
65 }
```

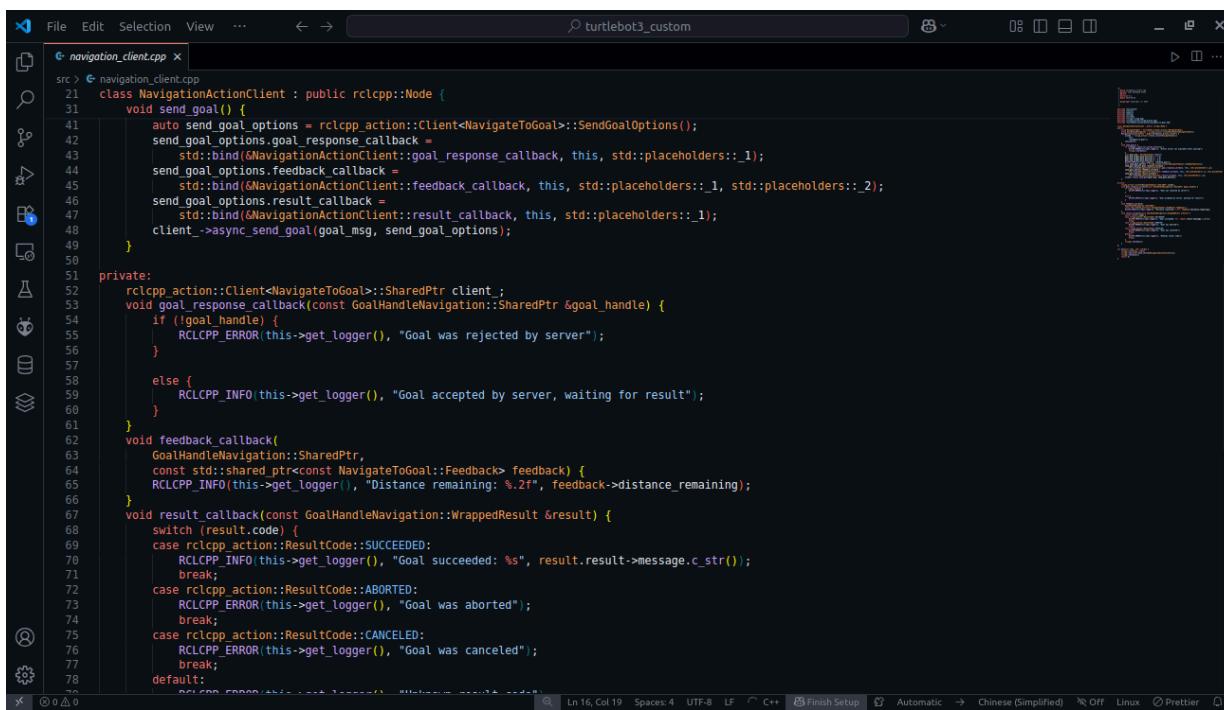
```
File Edit Selection View ... ← → turtlebot3_custom
navigation_action_server.cpp x
src > navigation_action_server.cpp
23 class NavigationActionServer : public rclcpp::Node {
27     NavigationActionServer() : Node("navigation_action_server") {
41
42     private:
43         rclcpp_action::GoalResponse handle_goal(
44             const rclcpp_action::GoalUUID &uuid,
45             std::shared_ptr<const NavigateToGoal> goal) {
46             RCLCPP_INFO(this->get_logger(), "Received goal request: x=%lf, y=%lf",
47                         goal->target_pose.position.x, goal->target_pose.position.y);
48             (void)uuid;
49             return rclcpp_action::GoalResponse::ACCEPT_AND_EXECUTE;
50         }
51
52         rclcpp_action::CancelResponse handle_cancel(
53             const std::shared_ptr<GoalHandleNavigation> goal_handle) {
54             RCLCPP_INFO(this->get_logger(), "Received request to cancel goal");
55             (void)goal_handle;
56             return rclcpp_action::CancelResponse::ACCEPT;
57         }
58
59         void handle_accepted(const std::shared_ptr<GoalHandleNavigation> goal_handle) {
60             // Execute the goal in a separate thread
61             std::thread{std::bind(&NavigationActionServer::execute, this, std::placeholders::_1), goal_handle}.detach();
62         }
63
64         void execute(const std::shared_ptr<GoalHandleNavigation> goal_handle) {
65             RCLCPP_INFO(this->get_logger(), "Executing goal");
66             rclcpp::Rate loop_rate(10);
67             const auto goal = goal_handle->get_goal();
68             auto feedback = std::make_shared<NavigateToGoal::Feedback>();
69             auto result = std::make_shared<NavigateToGoal::Result>();
70             auto start_time = this->now();
71             // Simple navigation towards goal
72             while (rclcpp::ok()) {
73                 // Check if goal is canceled
74                 if (goal_handle->is_canceled()) {
75                     result->success = false;
76                     result->message = "Goal was canceled";
77                     result->final_pose = current_pose_;
78                     goal_handle->canceled(result);
79                     RCLCPP_WARN(this->get_logger(), "Goal was canceled");
80                 }
81             }
82         }
83     }
84 }
```

# Action Client Example

Create [src/navigation\_client.cpp]:



```
File Edit Selection View ... turtlebot3_custom
navigation_client.cpp
src > navigation_client.cpp
21 class NavigationActionClient : public rclcpp::Node {
22     const std::shared_ptr<const NavigateToGoal::Feedback> feedback) {
23 }
24 void result_callback(const GoalHandleNavigation::WrappedResult &result) {
25     switch (result.code) {
26         case rclcpp_action::ResultCode::SUCCEEDED:
27             RCLCPP_INFO(this->get_logger(), "Goal succeeded: %s", result.result->message.c_str());
28             break;
29         case rclcpp_action::ResultCode::ABORTED:
30             RCLCPP_ERROR(this->get_logger(), "Goal was aborted");
31             break;
32         case rclcpp_action::ResultCode::CANCELED:
33             RCLCPP_ERROR(this->get_logger(), "Goal was canceled");
34             break;
35         default:
36             RCLCPP_ERROR(this->get_logger(), "Unknown result code");
37             break;
38     }
39     rclcpp::shutdown();
40 }
41 int main(int argc, char **argv) {
42     rclcpp::init(argc, argv);
43     rclcpp::spin(std::make_shared<NavigationActionClient>());
44     rclcpp::shutdown();
45     return 0;
46 }
```



```
File Edit Selection View ... turtlebot3_custom
navigation_client.cpp
src > navigation_client.cpp
21 class NavigationActionClient : public rclcpp::Node {
22     void send_goal() {
23         auto send_goal_options = rclcpp_action::Client<NavigateToGoal>::SendGoalOptions();
24         send_goal_options.goal_response_callback =
25             std::bind(&NavigationActionClient::goal_response_callback, this, std::placeholders::_1);
26         send_goal_options.feedback_callback =
27             std::bind(&NavigationActionClient::feedback_callback, this, std::placeholders::_1, std::placeholders::_2);
28         send_goal_options.result_callback =
29             std::bind(&NavigationActionClient::result_callback, this, std::placeholders::_1);
30         client_.async_send_goal(goal_msg, send_goal_options);
31     }
32     private:
33         rclcpp_action::Client<NavigateToGoal>::SharedPtr client_;
34         void goal_response_callback(const GoalHandleNavigation::SharedPtr &goal_handle) {
35             if (!goal_handle) {
36                 RCLCPP_ERROR(this->get_logger(), "Goal was rejected by server");
37             }
38             else {
39                 RCLCPP_INFO(this->get_logger(), "Goal accepted by server, waiting for result");
40             }
41         }
42         void feedback_callback(
43             GoalHandleNavigation::SharedPtr,
44             const std::shared_ptr<const NavigateToGoal::Feedback> feedback) {
45             RCLCPP_INFO(this->get_logger(), "Distance remaining: %.2f", feedback->distance_remaining);
46         }
47         void result_callback(const GoalHandleNavigation::WrappedResult &result) {
48             switch (result.code) {
49                 case rclcpp_action::ResultCode::SUCCEEDED:
50                     RCLCPP_INFO(this->get_logger(), "Goal succeeded: %s", result.result->message.c_str());
51                     break;
52                 case rclcpp_action::ResultCode::ABORTED:
53                     RCLCPP_ERROR(this->get_logger(), "Goal was aborted");
54                     break;
55                 case rclcpp_action::ResultCode::CANCELED:
56                     RCLCPP_ERROR(this->get_logger(), "Goal was canceled");
57                     break;
58                 default:
59                     RCLCPP_ERROR(this->get_logger(), "Unknown result code");
60             }
61         }
62     };
63 }
```

```
navigation_client.cpp
1 /**
2  * @file navigation_client.cpp
3  * @author Ziad Mohammed Fathy
4  * @brief
5  * @version 0.1
6  * @date 2025-08-28
7  *
8  * @copyright Copyright (c) 2025
9  *
10 */
11
12 #include <functional>
13 #include <future>
14 #include <memory>
15 #include <string>
16 #include <sstream>
17 #include <rclcpp/rclcpp.hpp>
18 #include <rclcpp_action/rclcpp_action.hpp>
19 #include "turtlebot3_custom/action/navigate_to_goal.hpp"
20
21 class NavigationActionClient : public rclcpp::Node {
22 public:
23     using NavigateToGoal = turtlebot3_custom::action::NavigateToGoal;
24     using GoalHandleNavigation = rclcpp_action::ClientGoalHandle<NavigateToGoal>;
25     NavigationActionClient() : Node("navigation_action_client") {
26         client_ = rclcpp_action::create_client<NavigateToGoal>(
27             this,
28             "navigate_to_goal");
29         send_goal();
30     }
31     void send_goal() {
32         if (!client_->wait_for_action_server()) {
33             RCLCPP_ERROR(this->get_logger(), "Action server not available after waiting");
34             rclcpp::shutdown();
35         }
36         auto goal_msg = NavigateToGoal::Goal();
37         goal_msg.target_pose.position.x = 2.0;
38         goal_msg.target_pose.position.y = 1.0;
39         goal_msg.target_pose.position.z = 0.0;
40         RCLCPP_INFO(this->get_logger(), "Sending goal");
41         auto goal_handle = client_->send_goal(goal_msg);
42         RCLCPP_INFO(this->get_logger(), "Goal sent, id: %d", goal_handle->get_id());
43     }
44     void cancel_goal(GoalHandleNavigation handle) {
45         if (handle) {
46             handle->cancel();
47         }
48     }
49     void cancel_all_goals() {
50         if (client_) {
51             client_->cancel_all_goals();
52         }
53     }
54 };
55
56
57
58
59
60
61
62
63
64
65
66
```

## 6. Launch Files

### Main Launch File

Create `launch/turtlebot3_custom.launch.py`:

```
turtlebot3_custom.launch.py
1 def generate_launch_description():
2     return launch.LaunchDescription([
3         Node(
4             package='turtlebot3_custom',
5             executable='turtlebot3_controller',
6             name='turtlebot3_controller',
7             parameters=[{'use_sim_time': use_sim_time}],
8             output='screen'
9         ),
10        # Robot Status Server
11        Node(
12            package='turtlebot3_custom', executable='robot_status_server',
13            name='robot_status_server',
14            parameters=[{'use_sim_time': use_sim_time}],
15            output='screen'
16        ),
17        # Navigation Action Server
18        Node(
19            package='turtlebot3_custom',
20            executable='navigation_action_server',
21            name='navigation_action_server',
22            parameters=[{'use_sim_time': use_sim_time}],
23            output='screen'
24        ),
25        # Optional: Start RViz
26        Node(
27            package='rviz2',
28            executable='rviz2',
29            name='rviz2',
30            condition=IfCondition(LaunchConfiguration('start_rviz')),
31            output='screen'
32        ),
33    ])
34
```

```
File Edit Selection View ... ← → ⌘ turtlebot3_custom
launch > turtlebot3_custom.launch.py
1 from launch import LaunchDescription
2 from launch_ros.actions import Node
3 from launch.actions import DeclareLaunchArgument, ExecuteProcess
4 from launch.substitutions import LaunchConfiguration
5 from launch.conditions import IfCondition
6
7 def generate_launch_description():
8     # Declare launch arguments
9     use_sim_time = LaunchConfiguration('use_sim_time', default='false')
10    robot_model = LaunchConfiguration('robot_model', default='burger')
11
12    return LaunchDescription([
13        # Launch arguments
14        DeclareLaunchArgument(
15            'use_sim_time',
16            default_value='false',
17            description='Use simulation time if true'
18        ),
19
20        DeclareLaunchArgument(
21            'robot_model',
22            default_value='burger',
23            description='TurtleBot3 model (burger, waffle, waffle_pi)'
24        ),
25
26        DeclareLaunchArgument(
27            'start_rviz',
28            default_value='false',
29            description='Start RViz'
30        ),
31
32        # TurtleBot3 Controller Node
33        Node(
34            package='turtlebot3_custom',
35            executable='turtlebot3_controller',
36            name='turtlebot3_controller',
37            parameters=[{'use_sim_time': use_sim_time}],
38            output='screen'
39        ),
40
41        # Other launch configurations ...
42    ])
43
```

Ln 29, Col 37 Spaces: 4 UTF-8 LF () Python ⚙ Finish Setup 3.10.12 🔍 Automatic → Chinese (Simplified) % Off ⚙ Prettier

## Simulation Launch File

Create `launch/turtlebot3_simulation.launch.py`:

```
File Edit Selection View ... ← → ⌘ turtlebot3_custom
launch > turtlebot3_simulation.launch.py > generate_launch_description
1 import os
2 from launch import LaunchDescription
3 from launch.actions import IncludeLaunchDescription, DeclareLaunchArgument
4 from launch.launch_description_sources import PythonLaunchDescriptionSource
5 from launch.substitutions import LaunchConfiguration
6 from launch_ros.substitutions import FindPackageShare
7
8 def generate_launch_description():
9     # Get the launch directory
10    pkg_gazebo_ros = FindPackageShare(package='gazebo_ros').find('gazebo_ros')
11    pkg_turtlebot3_gazebo = FindPackageShare(package='turtlebot3_gazebo').find('turtlebot3_gazebo')
12
13    # Launch configuration variables
14    use_sim_time = LaunchConfiguration('use_sim_time', default='true')
15    world = LaunchConfiguration('world')
16
17    # Declare the launch arguments
18    declare_world_cmd = DeclareLaunchArgument(
19        'world',
20        default_value=os.path.join(pkg_turtlebot3_gazebo, 'worlds', 'turtlebot3_world.world'),
21        description='Full path to world model file to load')
22
23    # Start Gazebo server
24    start_gazebo_server_cmd = IncludeLaunchDescription(
25        PythonLaunchDescriptionSource(os.path.join(pkg_gazebo_ros, 'launch', 'gzserver.launch.py')),
26        launch_arguments={'world': world.items()})
27
28    # Start Gazebo client
29    start_gazebo_client_cmd = IncludeLaunchDescription(
30        PythonLaunchDescriptionSource(os.path.join(pkg_gazebo_ros, 'launch', 'gzclient.launch.py')))
31
32    # Robot State Publisher
33    robot_state_publisher_cmd = IncludeLaunchDescription(
34        PythonLaunchDescriptionSource(os.path.join(pkg_turtlebot3_gazebo, 'launch', 'robot_state_publisher.launch.py')),
35        launch_arguments={'use_sim_time': use_sim_time}.items())
36
37    # Spawn TurtleBot3
38    spawn_turtlebot_cmd = IncludeLaunchDescription(
39        PythonLaunchDescriptionSource(os.path.join(pkg_turtlebot3_gazebo, 'launch', 'spawn_turtlebot3.launch.py')),
40        launch_arguments={'use_sim_time': use_sim_time}.items())
41
```

Ln 58, Col 14 Spaces: 4 UTF-8 LF () Python ⚙ Finish Setup 3.10.12 🔍 Automatic → Chinese (Simplified) % Off ⚙ Prettier

# Running Options

## Option 1: Simulation (Recommended for Testing)

### Method 1A: Complete Simulation Launch:

colcon build

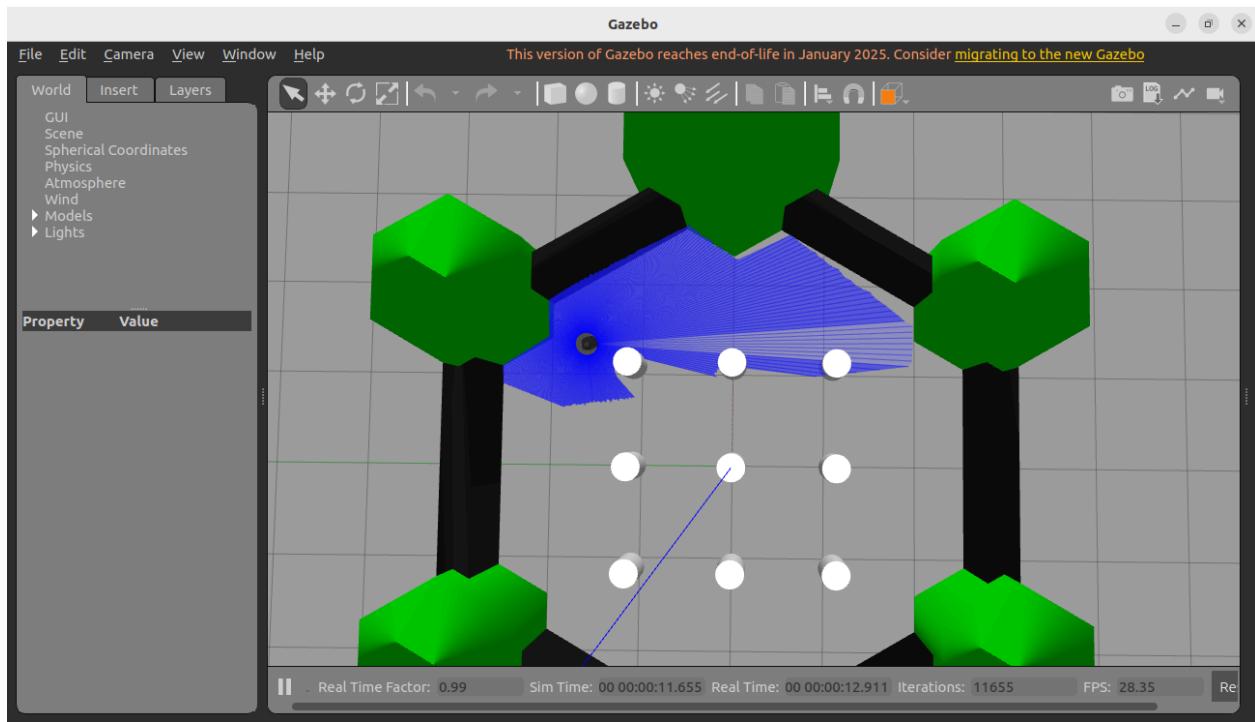
```
ziad@ziadfathy:~/Ziad_WS/ROS2/work_space$ colcon build
Starting >>> publisher_subscriber
Starting >>> robot_model_gazebo
Starting >>> ros2_urdf
Starting >>> services_actions
Finished <<< robot_model_gazebo [1.14s]
Starting >>> turtlebot3_custom
Finished <<< publisher_subscriber [1.84s]
Starting >>> ur_simulation_gazebo
Finished <<< ros2_urdf [1.86s]
Finished <<< ur_simulation_gazebo [0.25s]
Finished <<< services_actions [2.77s]
Finished <<< turtlebot3_custom [26.3s]

Summary: 6 packages finished [28.4s]
ziad@ziadfathy:~/Ziad_WS/ROS2/work_space$
```

source install/setup.bash

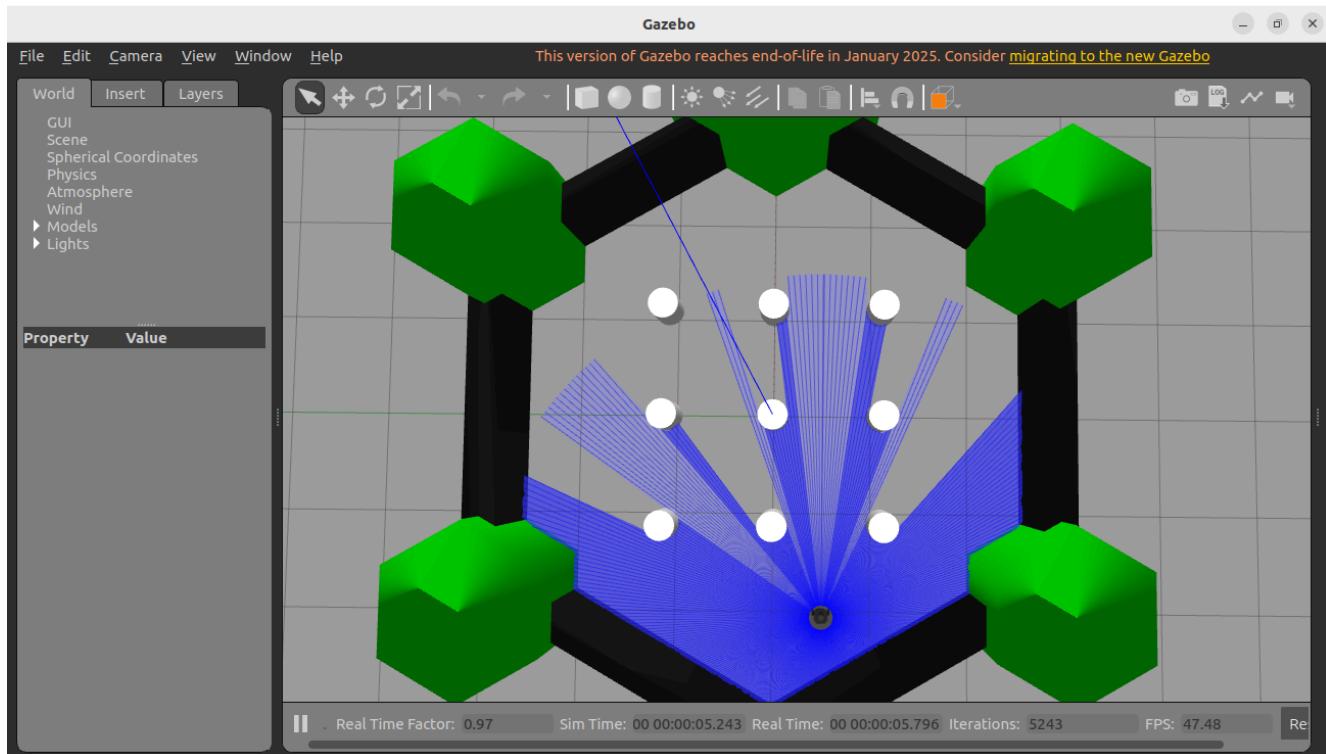
ros2 launch turtlebot3\_custom turtlebot3\_simulation.launch.py

```
ziad@ziadfathy:~/Ziad_WS/ROS2/work_space$ source install/setup.bash
ziad@ziadfathy:~/Ziad_WS/ROS2/work_space$ ros2 launch turtlebot3_custom
  turtlebot3_simulation.launch.py
[INFO] [launch]: All log files can be found below /home/ziad/.ros/log/2
025-08-30-01-48-56-649851-ziadfathy-13938
[INFO] [launch]: Default logging verbosity is set to INFO
urdf_file_name : turtlebot3_burger.urdf
urdf_file_name : turtlebot3_burger.urdf
[INFO] [gzserver-1]: process started with pid [13939]
[INFO] [gzclient-2]: process started with pid [13941]
[INFO] [robot_state_publisher-3]: process started with pid [13943]
[INFO] [spawn_entity.py-4]: process started with pid [13945]
[INFO] [turtlebot3_controller-5]: process started with pid [13947]
[INFO] [robot_status_server-6]: process started with pid [13949]
[INFO] [navigation_action_server-7]: process started with pid [13951]
[turtlebot3_controller-5] [INFO] [1756507738.749358474] [turtlebot3_controller]: Enhanced TurtleBot3 Controller Started
[turtlebot3_controller-5] [INFO] [1756507738.749528557] [turtlebot3_controller]: Robot will automatically avoid obstacles by turning
```

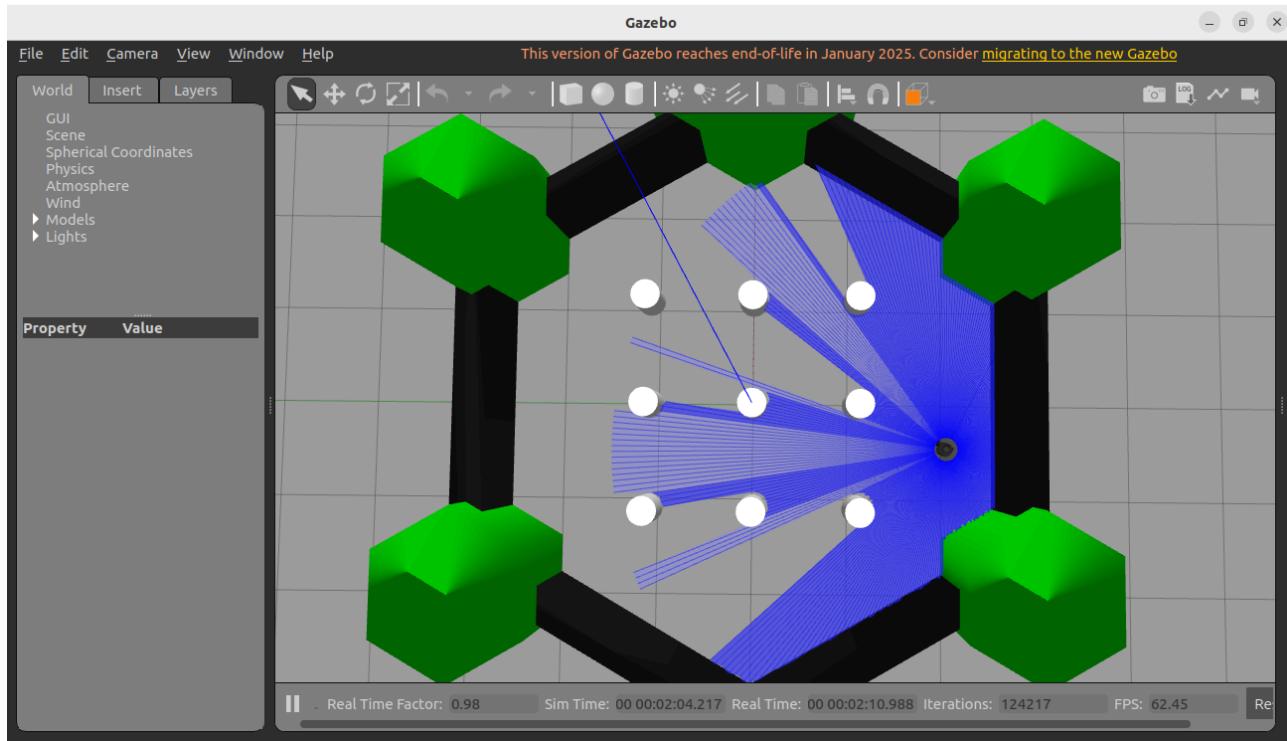


## Method 1B: Step-by-Step Simulation:

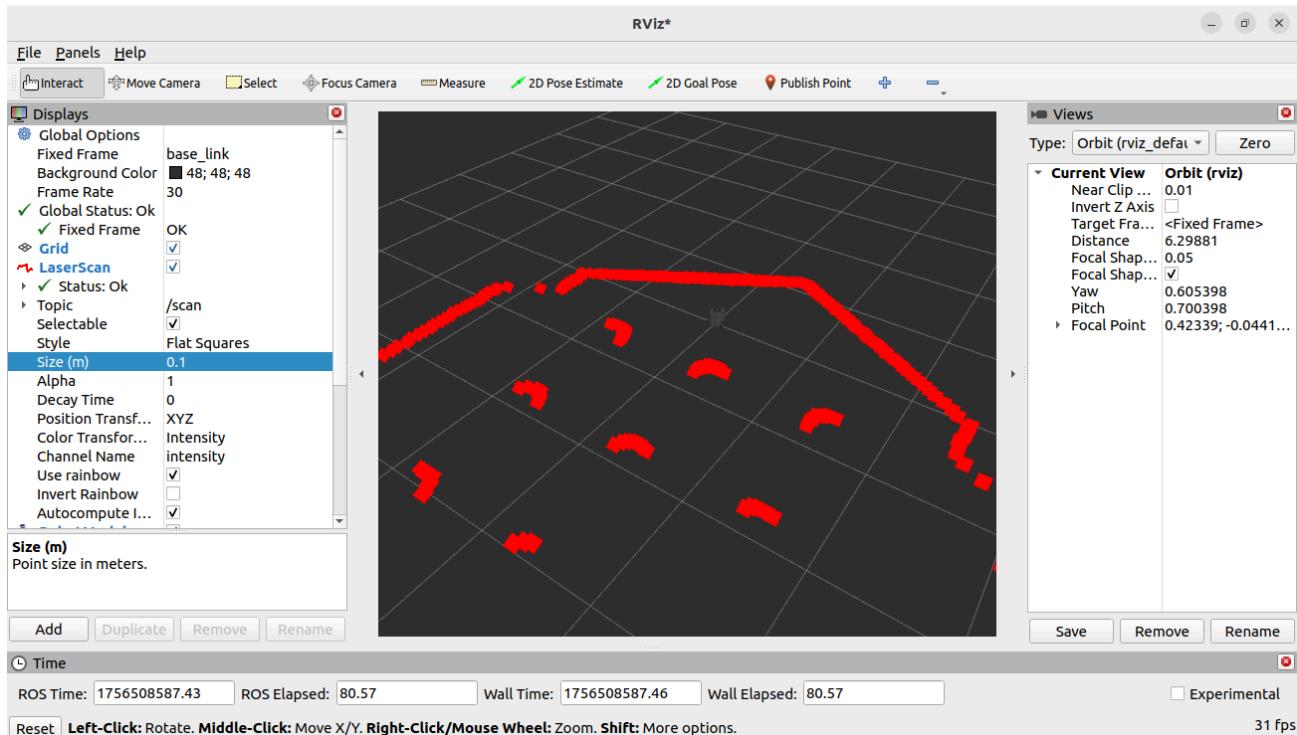
ros2 launch turtlebot3\_gazebo turtlebot3\_world.launch.py



```
ros2 launch turtlebot3_custom turtlebot3_custom.launch.py use_sim_time:=true
```

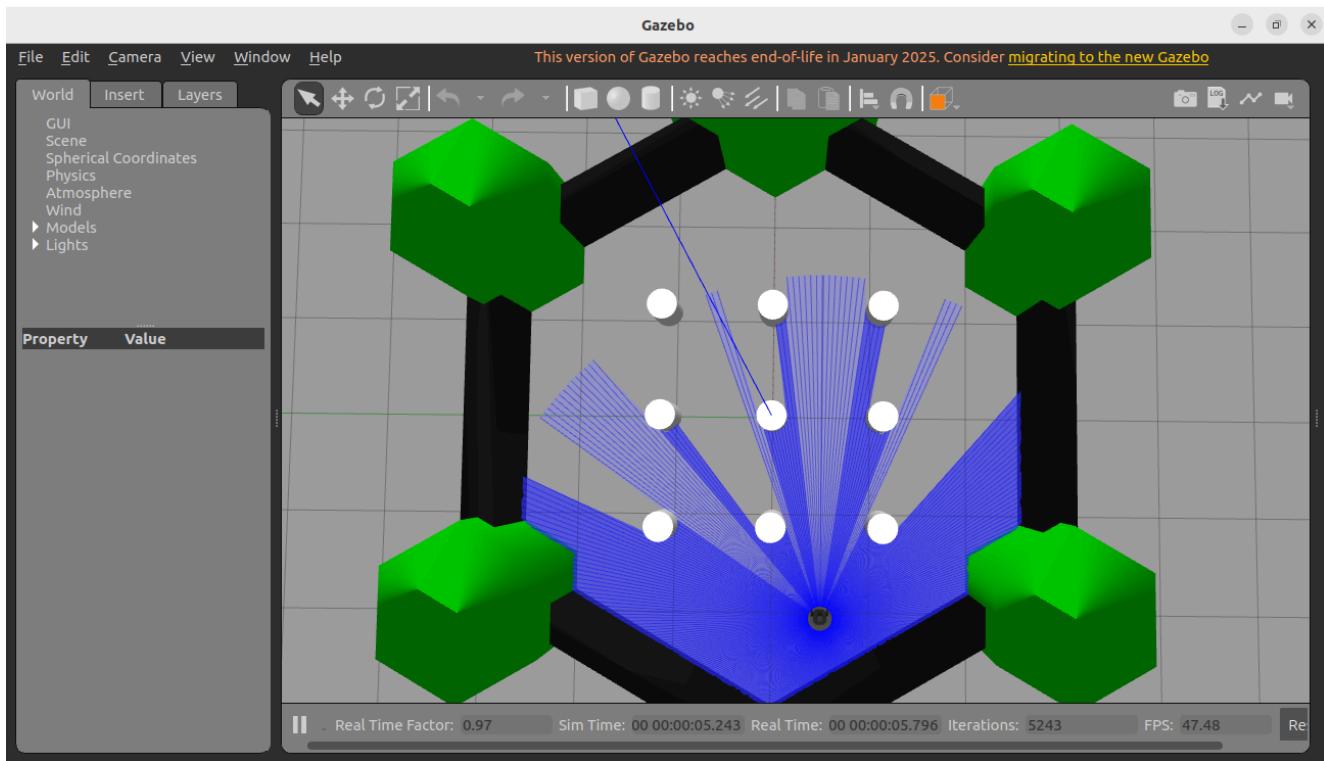


```
ros2 run rviz2 rviz2
```



## Method 1C: Individual Node Testing

```
ros2 launch turtlebot3_gazebo turtlebot3_world.launch.py
```



```
ros2 run turtlebot3_custom robot_status_server
```

```
ros2 run turtlebot3_custom navigation_action_server
```

```
ziad@ziadfathy: ~/Ziad_WS/ROS2/work_space
ziad@ziadfathy:~/Ziad_WS/ROS2/work_space$ ros2 run turtlebot3_custom robot_status_server
[INFO] [1756508954.405405844] [robot_status_server]: Robot Status Server ready

ziad@ziadfathy:~/Ziad_WS/ROS2/work_space$ source install/setup.bash
ziad@ziadfathy:~/Ziad_WS/ROS2/work_space$ ros2 run turtlebot3_custom navigation_action_server
[INFO] [1756508979.348396053] [navigation_action_server]: Navigation Action Server ready

[ gzserver-1 ] [INFO] [1756508950.467238352] [turtlebot3_diff_drive]: Publishing odom transforms between [odom] and [base_footprint]
[ gzserver-1 ] [INFO] [1756508950.481704542] [turtlebot3_joint_state]: Going to publish joint [wheel_left_joint]
[ gzserver-1 ] [INFO] [1756508950.481765712] [turtlebot3_joint_state]: Going to publish joint [wheel_right_joint]
[INFO] [spawn_entity.py-4]: process has finished cleanly [pid 17176]
```

# Testing the Functionality

## 1. Test the Service

```
ros2 run turtlebot3_custom status_client
```

```
ros2 service call /get_robot_status turtlebot3_custom/srv/GetRobotStatus "{query_type: 'battery'}"  
ros2 service call /get_robot_status turtlebot3_custom/srv/GetRobotStatus "{query_type: 'general'}"  
ros2 service call /get_robot_status turtlebot3_custom/srv/GetRobotStatus "{query_type: 'position'}"
```

The screenshot shows two terminal windows side-by-side. Both windows have a title bar 'ziad@ziadfathy: ~/Ziad\_WS/ROS2/work\_space'. The left terminal window displays the output of a 'status\_client' command, which includes detailed battery information and a JSON response. The right terminal window shows the results of several 'service call' commands to the 'GetRobotStatus' service, listing the battery, general, and position types.

```
ziad@ziadfathy: ~/Ziad_WS/ROS2/work_space  
n(x=-0.00266021345631665, y=0.0010440294688355291, z=0.93068779  
47271492, w=0.3658033105546623)), battery_info=sensor_msgs.msg.  
BatteryState(header=std_msgs.msg.Header(stamp= builtin_interface  
s.msg.Time(sec=0, nanosec=0), frame_id='')), voltage=0.0, temper  
ature=0.0, current=0.0, charge=0.0, capacity=0.0, design_capaci  
ty=0.0, percentage=0.0, power_supply_status=0, power_supply_he  
alth=0, power_supply_technology=0, present=False, cell_voltage=[  
], cell_temperature=[], location='', serial_number='')  
  
ziad@ziadfathy:~/Ziad_WS/ROS2/work_space$ ros2 service call /ge  
t_robot_status turtlebot3_custom/srv/GetRobotStatus "{query_ty  
pe: 'battery'}"  
requester: making request: turtlebot3_custom.srv.GetRobotStatus  
_Request(query_type='battery')  
  
response:  
turtlebot3_custom.srv.GetRobotStatus_Response(success=True, sta  
tus_message='Battery level: 0.000000%', current_pose=geometry_m  
sgs.msg.Pose(position=geometry_msgs.msg.Point(x=-0.791156472294  
8307, y=-2.0336047900875154, z=0.03278709583244727), orientatio  
n=geometry_msgs.msg.Quaternion(x=0.3907897746973448, y=-0.12043  
093803878906, z=-0.845433041758332, w=-0.34354433929021494)), b  
attery_info=sensor_msgs.msg.BatteryState(header=std_msgs.msg.He  
ader(stamp= builtin_interfaces.msg.Time(sec=0, nanosec=0), frame  
_id='')), voltage=0.0, temperature=0.0, current=0.0, charge=0.0,  
capacity=0.0, design_capacity=0.0, percentage=0.0, power_suppl  
y_status=0, power_supply_health=0, power_supply_technology=0, p  
resent=False, cell_voltage=[], cell_temperature=[], location=''  
, serial_number='')  
  
ziad@ziadfathy:~/Ziad_WS/ROS2/work_space$
```

```
ziad@ziadfathy: ~/Ziad_WS/ROS2/work_space$ ros2 run t  
ango_icons_vendor tinyxml2_vendor  
tcb_span tinyxml_vendor  
teleop_twist_joy tl_expected  
teleop_twist_keyboard tracetools  
tf2 trajectory_msgs  
tf2_bullet transmission_interface  
ziad@ziadfathy:~/Ziad_WS/ROS2/work_space$ ros2 run turtlebot3_c  
turtlebot3_cartographer turtlebot3_custom  
ziad@ziadfathy:~/Ziad_WS/ROS2/work_space$ ros2 run turtlebot3_c  
turtlebot3_cartographer turtlebot3_custom  
ziad@ziadfathy:~/Ziad_WS/ROS2/work_space$ ros2 run turtlebot3_c  
turtlebot3_cartographer turtlebot3_custom  
ziad@ziadfathy:~/Ziad_WS/ROS2/work_space$ ros2 run turtlebot3_c  
ustom robot_status_server  
[INFO] [1756475685.807091412] [robot_status_server]: Robot Stat  
us Server ready  
[INFO] [1756475693.279604993] [robot_status_server]: Received s  
tatus request: battery
```

## 2. Test the Actions

```
ros2 run turtlebot3_custom navigation_client
```

```
ros2 action send_goal /navigate_to_goal turtlebot3_custom/action/NavigateToGoal "{target_pose: {position: {x: 2.0,  
y: 1.0, z: 0.0}, orientation: {x: 0.0, y: 0.0, z: 0.0, w: 1.0}}}"
```

```
ros2 action send_goal --feedback /navigate_to_goal turtlebot3_custom/action/NavigateToGoal "{target_pose:  
{position: {x: 1.5, y: 0.5, z: 0.0}, orientation: {x: 0.0, y: 0.0, z: 0.0, w: 1.0}}}"
```

```
ziad@ziadfathy: ~/Ziad_WS/ROS2/work_space
ziad@ziadfathy:~/Ziad_WS/ROS2/work_space$ ^C
ziad@ziadfathy:~/Ziad_WS/ROS2/work_space$ ros2 service call /get_robot_status turtlebot3_custom/srv/GetRobotStatus "{query_type: 'general'}"
requester: making request: turtlebot3_custom.srv.GetRobotStatus
Request(query_type='general')

response:
turtlebot3_custom.srv.GetRobotStatus_Response(success=True, status_message='Robot is operational. Position and battery data available.', current_pose=geometry_msgs.msg.Pose(position=geometry_msgs.msg.Point(x=-0.8283911916515801, y=-2.2433957370490205, z=0.04367153904234228), orientation=geometry_msgs.msg.Quaternion(x=-0.01779298535488101, y=-0.263967042697233, z=-0.9044215745815937, w=-0.3347034888870881)), battery_info=sensor_msgs.msg.BatteryState(header=std_msgs.msg.Header(stamp=builtin_interfaces.msg.Time(sec=0, nanosec=0), frame_id=''), voltage=0.0, temperature=0.0, current=0.0, charge=0.0, capacity=0.0, design_capacity=0.0, percentage=0.0, power_supply_status=0, power_supply_health=0, power_supply_technology=0, present=False, cell_voltage=[], cell_temperature=[], location='', serial_number=''))

ziad@ziadfathy:~/Ziad_WS/ROS2/work_space$ 

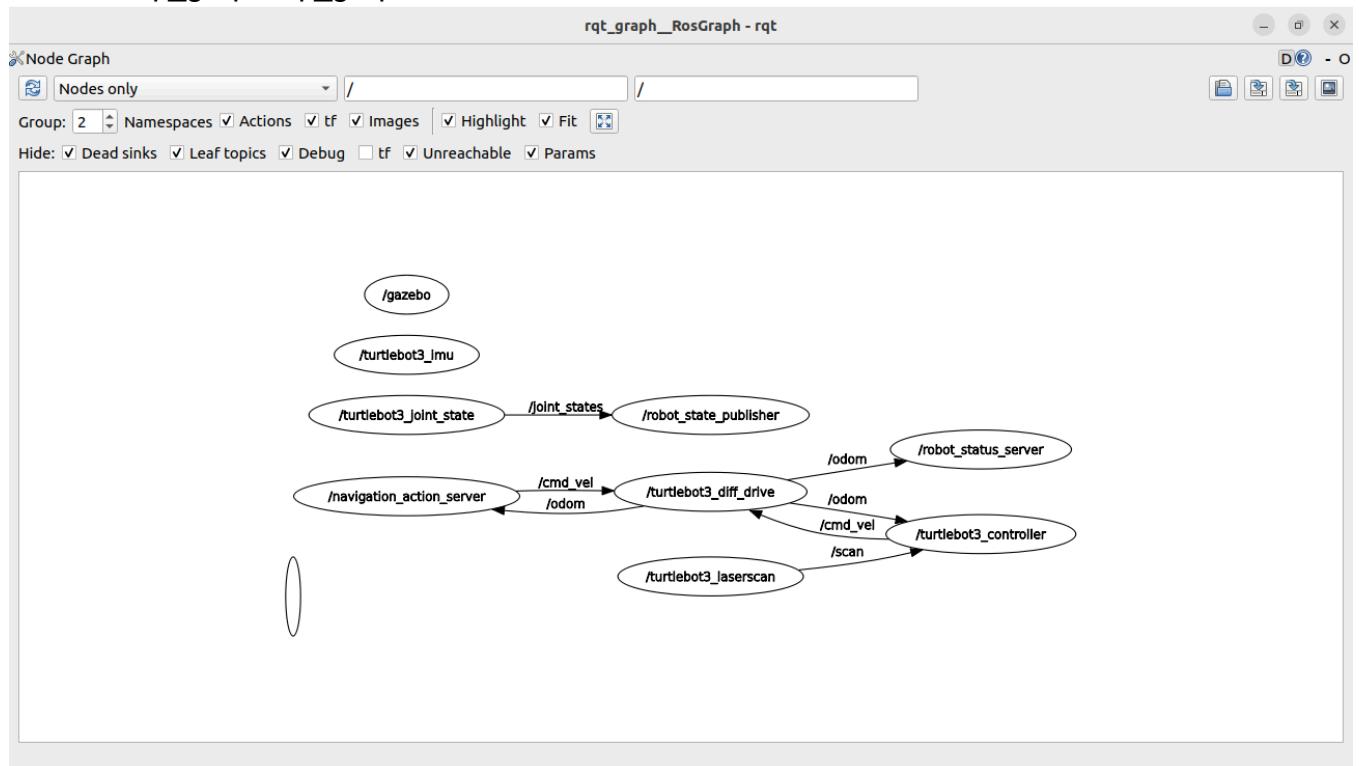
ziad@ziadfathy: ~/Ziad_WS/ROS2/work_space
ziad@ziadfathy:~/Ziad_WS/ROS2/work_space$ ros2 run turtlebot3_custom turtlebot3_cartographer turtlebot3_custom
ziad@ziadfathy:~/Ziad_WS/ROS2/work_space$ ros2 run turtlebot3_custom turtlebot3_cartographer turtlebot3_custom
ziad@ziadfathy:~/Ziad_WS/ROS2/work_space$ ros2 run turtlebot3_custom turtlebot3_cartographer turtlebot3_custom
ziad@ziadfathy:~/Ziad_WS/ROS2/work_space$ ros2 run turtlebot3_custom robot_status_server
[INFO] [1756475685.807091412] [robot_status_server]: Robot Status Server ready
[INFO] [1756475693.279604993] [robot_status_server]: Received status request: battery
[INFO] [1756475747.039700878] [robot_status_server]: Received status request: general
^C[INFO] [1756475751.912466800] [rclcpp]: signal_handler(signum=2)
ziad@ziadfathy:~/Ziad_WS/ROS2/work_space$
```

```
ziad@ziadfathy: ~/Ziad_WS/ROS2/work_space
ziad@ziadfathy:~/Ziad_WS/ROS2/work_space$ ros2 run turtlebot3_custom navigation_action_server
[INFO] [1756475784.728929139] [navigation_action_server]: Navigation Action Server ready
[INFO] [1756475847.298070940] [navigation_action_server]: Received goal request: x=2.00, y=1.00
[INFO] [1756475847.298357618] [navigation_action_server]: Executing goal
[INFO] [1756475918.690997829] [navigation_action_server]: Received goal request: x=2.00, y=1.00
[INFO] [1756475918.691250301] [navigation_action_server]: Executing goal
[INFO] [1756475941.780384385] [navigation_action_server]: Received goal request: x=2.00, y=1.00
[INFO] [1756475941.781254673] [navigation_action_server]: Executing goal
[INFO] [1756475941.781254673] [navigation_action_server]: Goal accepted with ID: e97ce16e2c934003b4adf18562999785
```

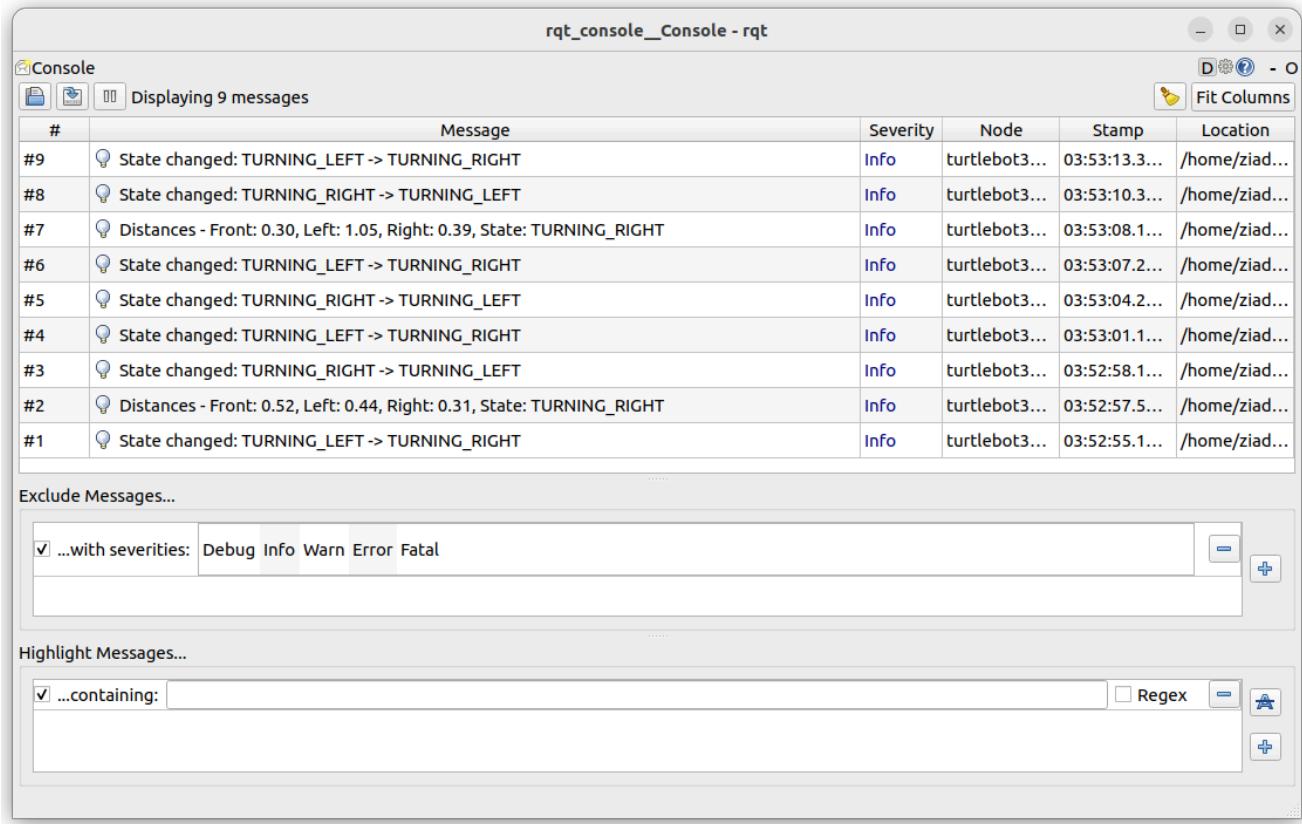
```

ziad@ziadfathy: ~/Ziad_WS/ROS2/work_space
[INFO] [1756475847.298357618] [navigation_action_server]: Executing goal
[INFO] [1756475918.690997829] [navigation_action_server]: Received goal request: x=2.00, y=1.00
[INFO] [1756475918.691250301] [navigation_action_server]: Executing goal
[INFO] [1756475941.780384385] [navigation_action_server]: Received goal request: x=2.00, y=1.00
[INFO] [1756475941.781254673] [navigation_action_server]: Executing goal
[INFO] [1756475948.592619116] [navigation_action_server]: Received request to cancel goal
[INFO] [1756475948.681554634] [navigation_action_server]: Goal canceled
[INFO] [1756475965.203164648] [navigation_action_server]: Received goal request: x=1.50, y=0.50
[INFO] [1756475965.203391622] [navigation_action_server]: Executing goal
[INFO] [1756475980.892146883] [navigation_action_server]: Received request to cancel goal
[INFO] [1756475980.903627445] [navigation_action_server]: Goal canceled
[INFO] [1756475987.454920204] [navigation_action_server]: Received goal request: x=2.00, y=1.00
[INFO] [1756475987.455109590] [navigation_action_server]: Executing goal
[INFO] [1756475998.256634803] [navigation_action_server]: Received request to cancel goal
[INFO] [1756475998.355491295] [navigation_action_server]: Goal canceled
ValueError: generator already executing
ziad@ziadfathy:~/Ziad_WS/ROS2/work_space$
```

ros2 run rqt\_graph rqt\_graph



```
ros2 run rqt_console rqt_console
```



```
ros2 node list
```

```
ros2 node info /turtlebot3_controller
```

The screenshot shows two terminal windows. The left window displays the output of 'ros2 node list', showing nodes like navigation\_action\_server, robot\_status\_server, and turtlebot3\_controller. The right window displays the output of 'ros2 node info /turtlebot3\_controller', providing detailed information about the controller's subscribers, publishers, service servers, and clients. Both windows show log entries from the turtlebot3\_controller node, indicating its status and interactions with other ROS components.

```
ziad@ziadfathy: ~/Ziad_WS/ROS2/work_space
[robot_status_server-2] [INFO] [1756515226.919957630] [rclcpp]: signal_handler(signum=2)
[turtlebot3_controller-1] [INFO] [1756515226.920047469] [rclcpp]: signal_handler(signum=2)
[INFO] [robot_status_server-2]: process has finished cleanly [pid 25895]
[INFO] [turtlebot3_controller-1]: process has finished cleanly [pid 25893]
[INFO] [navigation_action_server-3]: process has finished cleanly [pid 25897]
ziad@ziadfathy: ~/Ziad_WS/ROS2/work_space$ ros2 launch turtlebot3_custom turtlebot3_custom.launch.py
[INFO] [launch]: All log files can be found below /home/ziad/.ros/log/2025-08-30-03-55-47-585811-ziadfathy-26793
[INFO] [launch]: Default logging verbosity is set to INFO
[INFO] [turtlebot3_controller-1]: process started with pid [26794]
[INFO] [robot_status_server-2]: process started with pid [26796]
[INFO] [navigation_action_server-3]: process started with pid [26798]
[robot_status_server-2] [INFO] [1756515347.704560088] [robot_status_server]: Robot Status Server ready
[navigation_action_server-3] [INFO] [1756515347.705191586] [navigation_action_server]: Navigation Action Server ready
[turtlebot3_controller-1] [INFO] [1756515347.705786166] [turtlebot3_controller]: Enhanced TurtleBot3 Controller Started
[turtlebot3_controller-1] [INFO] [1756515347.705915680] [turtlebot3_controller]: Robot will automatically avoid obstacles by turning
```

```
ros2 service list
```

```
ros2 service type /get_robot_status
```

The screenshot shows two terminal windows side-by-side. The left window displays the output of the command `ros2 service list`, which lists various ROS services including `/get_robot_status`. The right window shows the output of `ros2 run turtlebot3_custom status_client`, which prints status messages from the `robot_status_server` about the robot's operational status and battery data.

```
ziad@ziadfathy: ~/Ziad_WS/ROS2/work_space
ziad@ziadfathy:~/Ziad_WS/ROS2/work_space$ ros2 service list
ros2 service type /get_robot_status
/get_robot_status
/robot_status_server/describe_parameters
/robot_status_server/get_parameter_types
/robot_status_server/get_parameters
/robot_status_server/list_parameters
/robot_status_server/set_parameters
/robot_status_server/set_parameters_atomically
/status_client/describe_parameters
/status_client/get_parameter_types
/status_client/get_parameters
/status_client/list_parameters
/status_client/set_parameters
/status_client/set_parameters_atomically
turtlebot3_custom/srv/GetRobotStatus
ziad@ziadfathy:~/Ziad_WS/ROS2/work_space$
```

```
ziad@ziadfathy: ~/Ziad_WS/ROS2/work_space
navigation_client      turtlebot3_controller
ziad@ziadfathy:~/Ziad_WS/ROS2/work_space$ ros2 run turtlebot3_c
ustom
modern_teleop_controller  robot_status_server
navigation_action_server  status_client
navigation_client        turtlebot3_controller
ziad@ziadfathy:~/Ziad_WS/ROS2/work_space$ ros2 run turtlebot3_c
ustom robot_status_server
[INFO] [1756515455.365379856] [robot_status_server]: Robot Stat
us Server ready
[INFO] [1756515470.688749328] [robot_status_server]: Received s
tatus request: general
[INFO] [1756515488.543976468] [robot_status_server]: Received s
tatus request: general
[INFO] [1756515470.689056668] [status_client]: Response: Robot
is operational. Position and battery data available.
[INFO] [1756515470.689146844] [status_client]: Success: true
^C[INFO] [1756515478.531561775] [rclcpp]: signal_handler(signum
=2)
ziad@ziadfathy:~/Ziad_WS/ROS2/work_space$ ros2 run turtlebot3_c
ustom status_client
[INFO] [1756515488.544203226] [status_client]: Response: Robot
is operational. Position and battery data available.
[INFO] [1756515488.544284968] [status_client]: Success: true
```

```
ros2 action list
```

```
ros2 action info /navigate_to_goal
```

The screenshot shows three terminal windows. The first window lists actions with `ros2 action list`, showing one action named `/navigate_to_goal`. The second window shows the `status_client` running, printing general status messages. The third window shows the `navigation_action_server` executing a goal, printing distance remaining updates.

```
ziad@ziadfathy: ~/Ziad_WS/ROS2/work_space
ziad@ziadfathy:~/Ziad_WS/ROS2/work_space$ ros2 action list
ros2 action info /navigate_to_goal
/navigate_to_goal
Action: /navigate_to_goal
Action clients: 1
  /navigation_action_client
Action servers: 1
  /navigation_action_server
ziad@ziadfathy:~/Ziad_WS/ROS2/work_space$
```

```
status request: general
^C[INFO] [1756515530.880440347] [rclcpp]: signal_handler(signum
=2)
ziad@ziadfathy:~/Ziad_WS/ROS2/work_space$ ros2 run turtlebot3_c
ustom navigation_
navigation_action_server  navigation_client
ziad@ziadfathy:~/Ziad_WS/ROS2/work_space$ ros2 run turtlebot3_c
ustom navigation_action_server
[INFO] [1756515547.728591150] [navigation_action_server]: Navig
ation Action Server ready
[INFO] [1756515559.008375712] [navigation_action_server]: Recei
ved goal request: x=2.00, y=1.00
[INFO] [1756515559.008666063] [navigation_action_server]: Execu
ting goal
[INFO] [1756515564.609268121] [navigation_action_client]: Dist
ance remaining: 2.24
[INFO] [1756515564.709239486] [navigation_action_client]: Dist
ance remaining: 2.24
[INFO] [1756515564.809397081] [navigation_action_client]: Dist
ance remaining: 2.24
[INFO] [1756515564.909430314] [navigation_action_client]: Dist
ance remaining: 2.24
[INFO] [1756515565.009398802] [navigation_action_client]: Dist
ance remaining: 2.24
[INFO] [1756515565.109415566] [navigation_action_client]: Dist
ance remaining: 2.24
[INFO] [1756515565.209382033] [navigation_action_client]: Dist
ance remaining: 2.24
```