TurtleBot3 ROS2 Humble C++ Development Guide

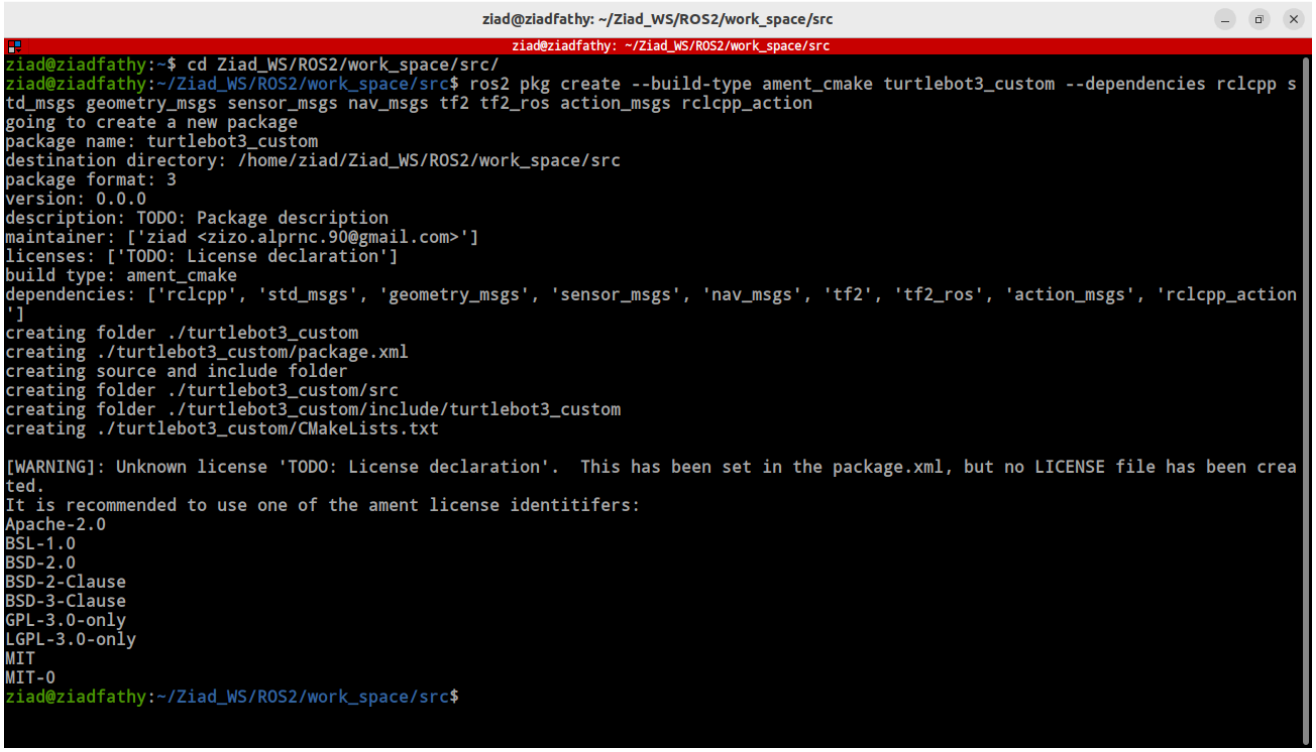
Prerequisites

Before starting, ensure you have:

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

1. Creating a ROS2 Package

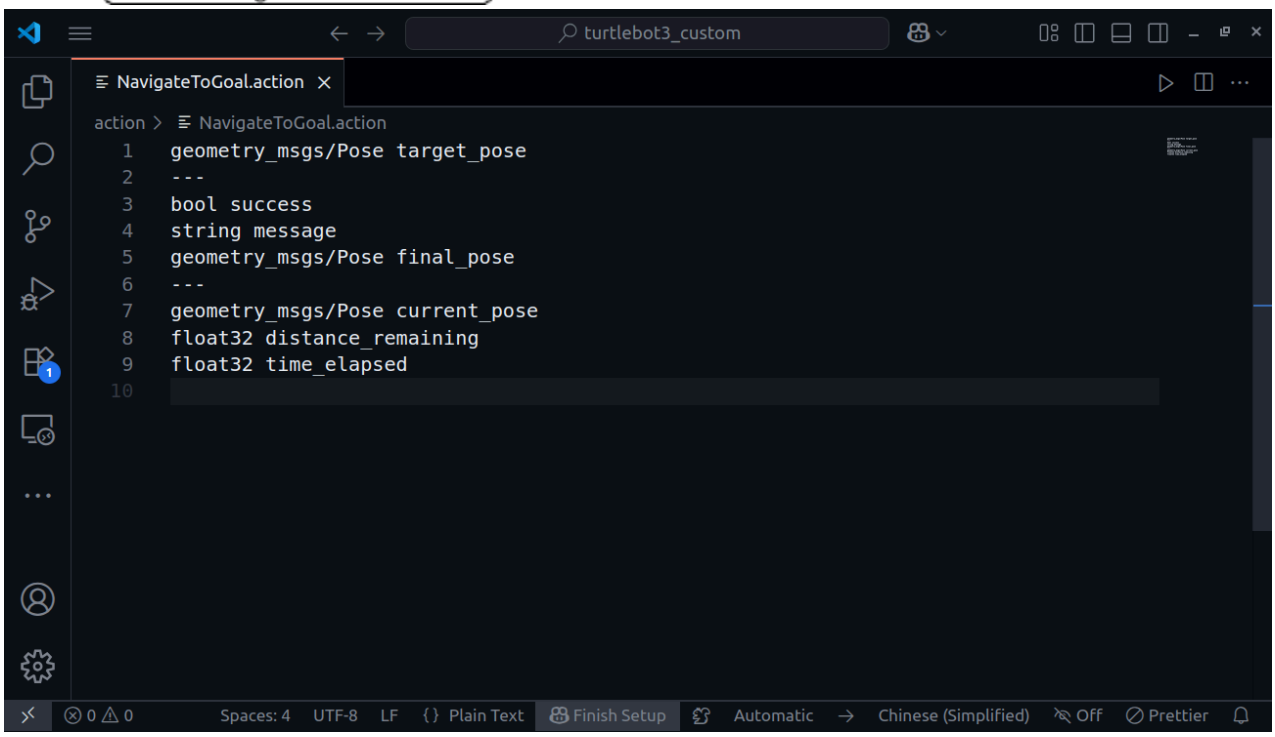
First, create a new package for your TurtleBot3 project:



2. Creating Custom Interfaces

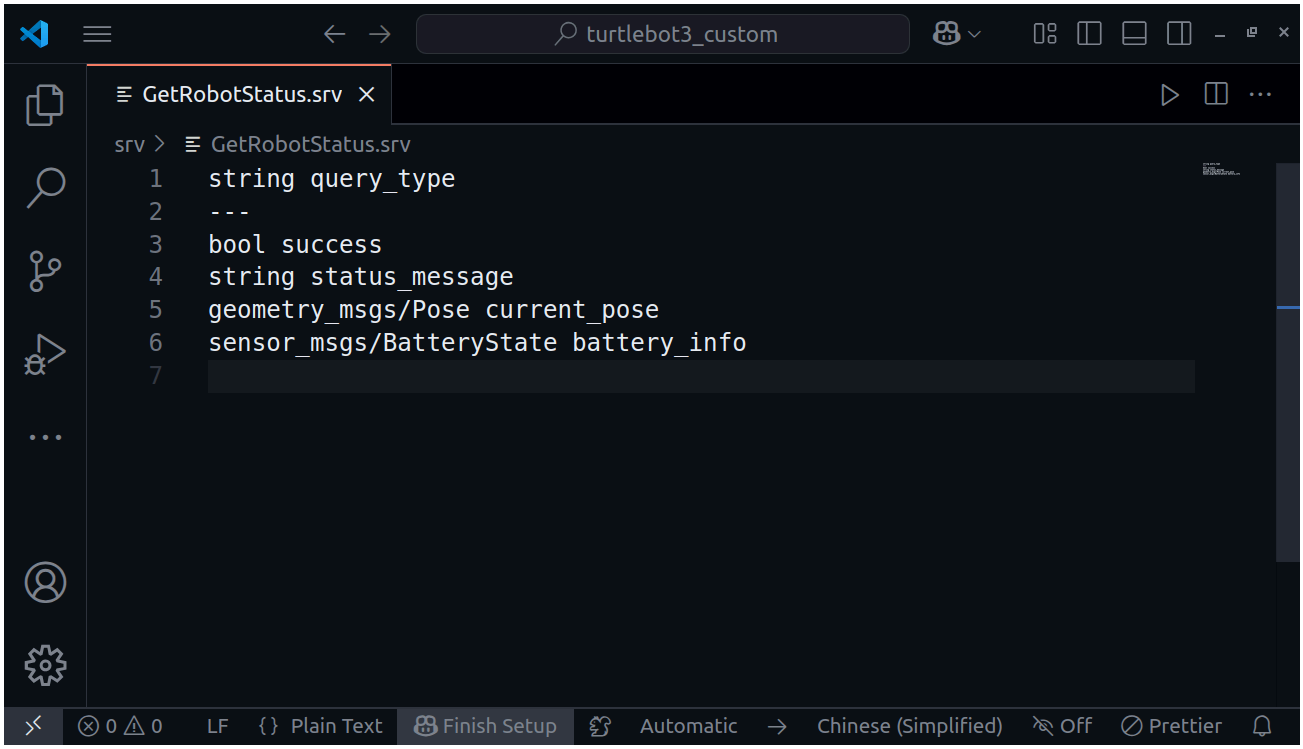
Custom Action Definition

Create `action/NavigateToGoal.action`:



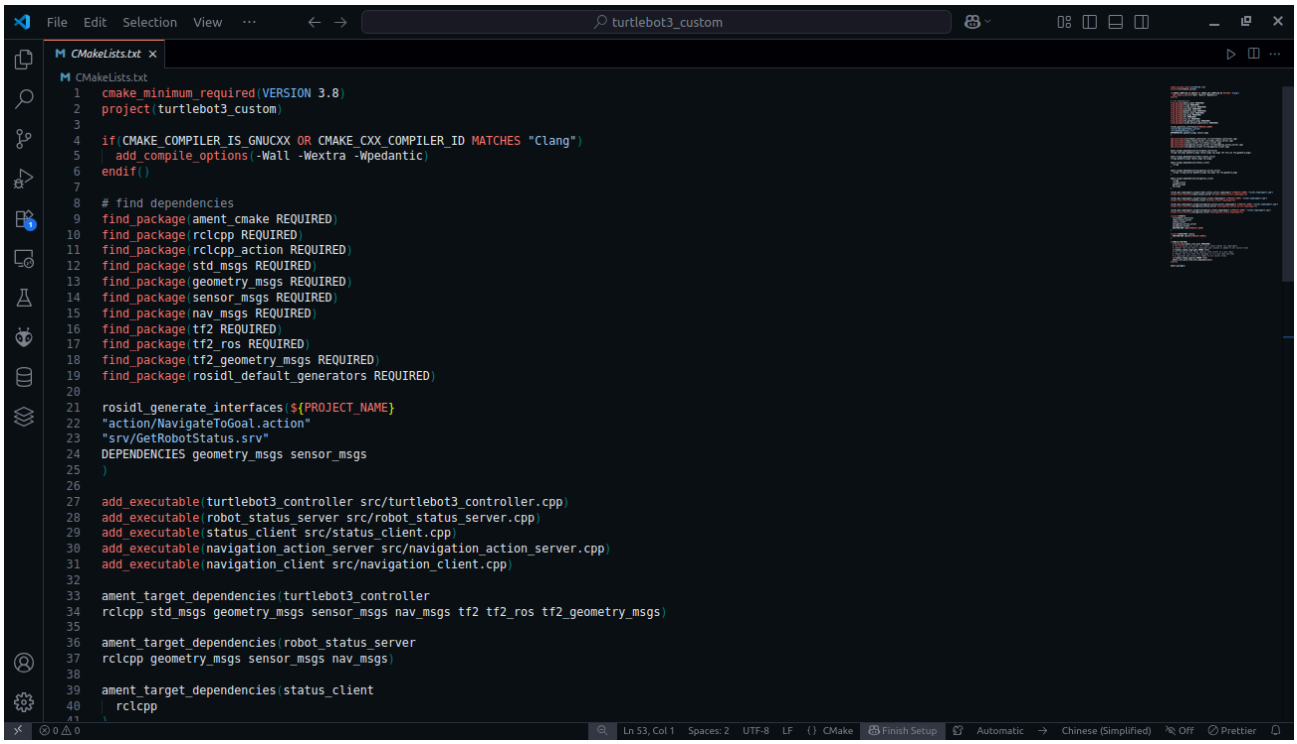
Custom Service Definition

Create `srv/GetRobotStatus.srv`:

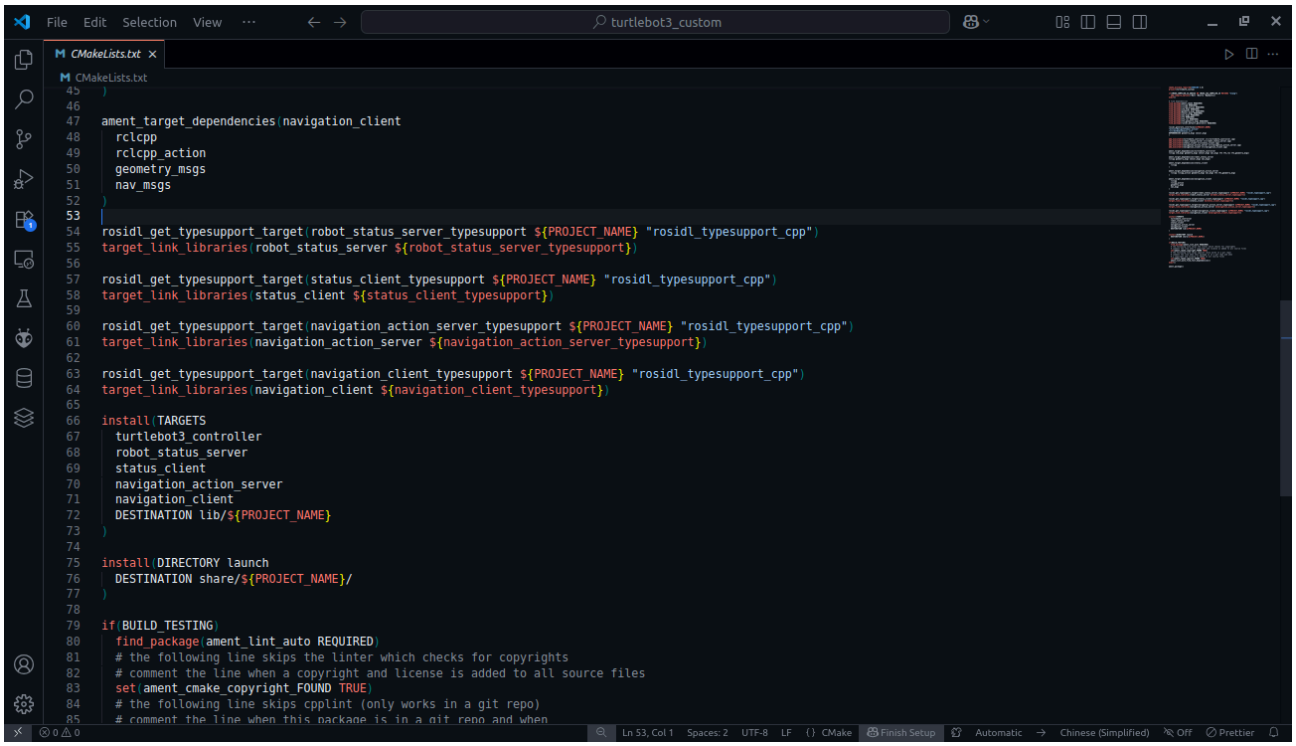


Update CMakeLists.txt for Custom Interfaces

Add to your `CMakeLists.txt`:



```
cmake_minimum_required(VERSION 3.8)
project(turtlebot3_custom)

if(CMAKE_COMPILER_IS_GNUCXX OR CMAKE_CXX_COMPILER_ID MATCHES "Clang")
  add_compile_options(-Wall -Wextra -Wpedantic)
endif()

# find dependencies
find_package(ament_cmake REQUIRED)
find_package(rclcpp REQUIRED)
find_package(rclcpp_action REQUIRED)
find_package(std_msgs REQUIRED)
find_package(geometry_msgs REQUIRED)
find_package(sensor_msgs REQUIRED)
find_package(nav_msgs REQUIRED)
find_package(tf2 REQUIRED)
find_package(tf2_ros REQUIRED)
find_package(tf2_geometry_msgs REQUIRED)
find_package(rosidl_default_generators REQUIRED)

rosidl_generate_interfaces(${PROJECT_NAME}
"action/NavigateToGoal.action"
"srv/GetRobotStatus.srv"
DEPENDENCIES geometry_msgs sensor_msgs
)

add_executable(turtlebot3_controller src/turtlebot3_controller.cpp)
add_executable(robot_status_server src/robot_status_server.cpp)
add_executable(status_client src/status_client.cpp)
add_executable(navigation_action_server src/navigation_action_server.cpp)
add_executable(navigation_client src/navigation_client.cpp)

ament_target_dependencies(turtlebot3_controller
rclcpp std_msgs geometry_msgs sensor_msgs nav_msgs tf2 tf2_ros tf2_geometry_msgs)

ament_target_dependencies(robot_status_server
rclcpp geometry_msgs sensor_msgs nav_msgs)

ament_target_dependencies(status_client
  rclcpp
```

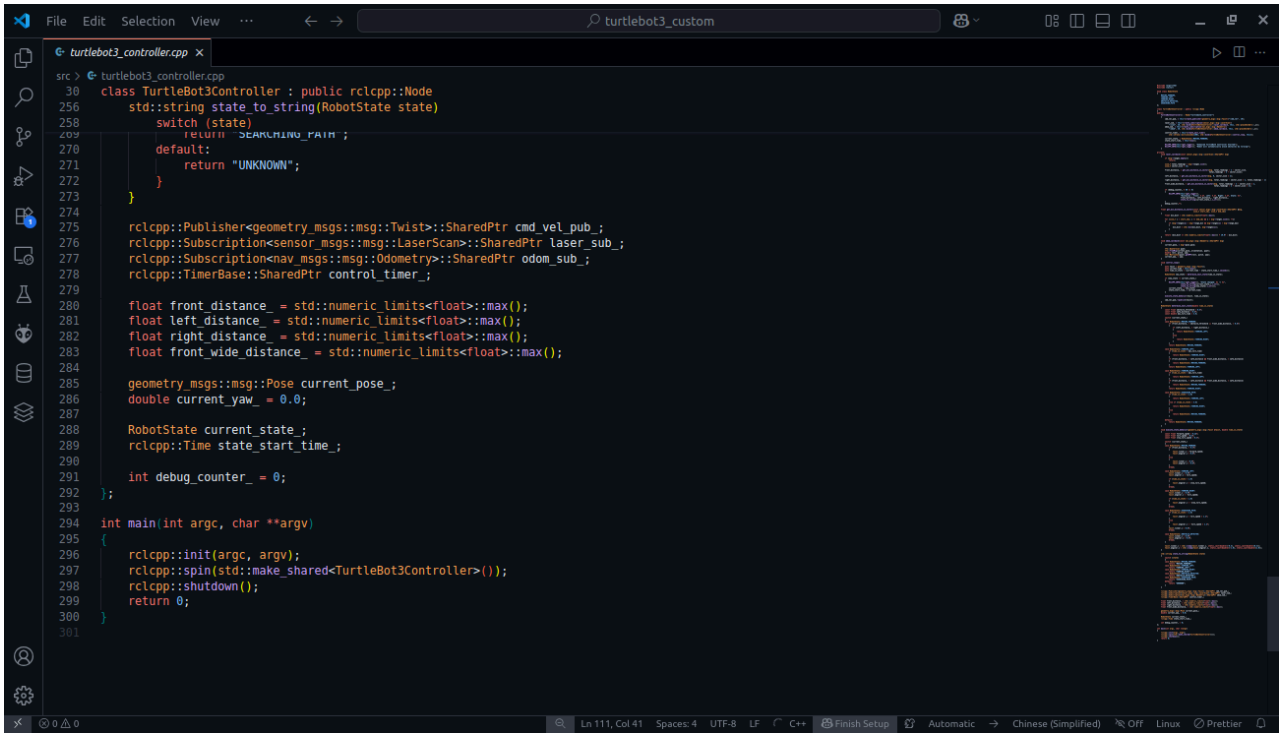


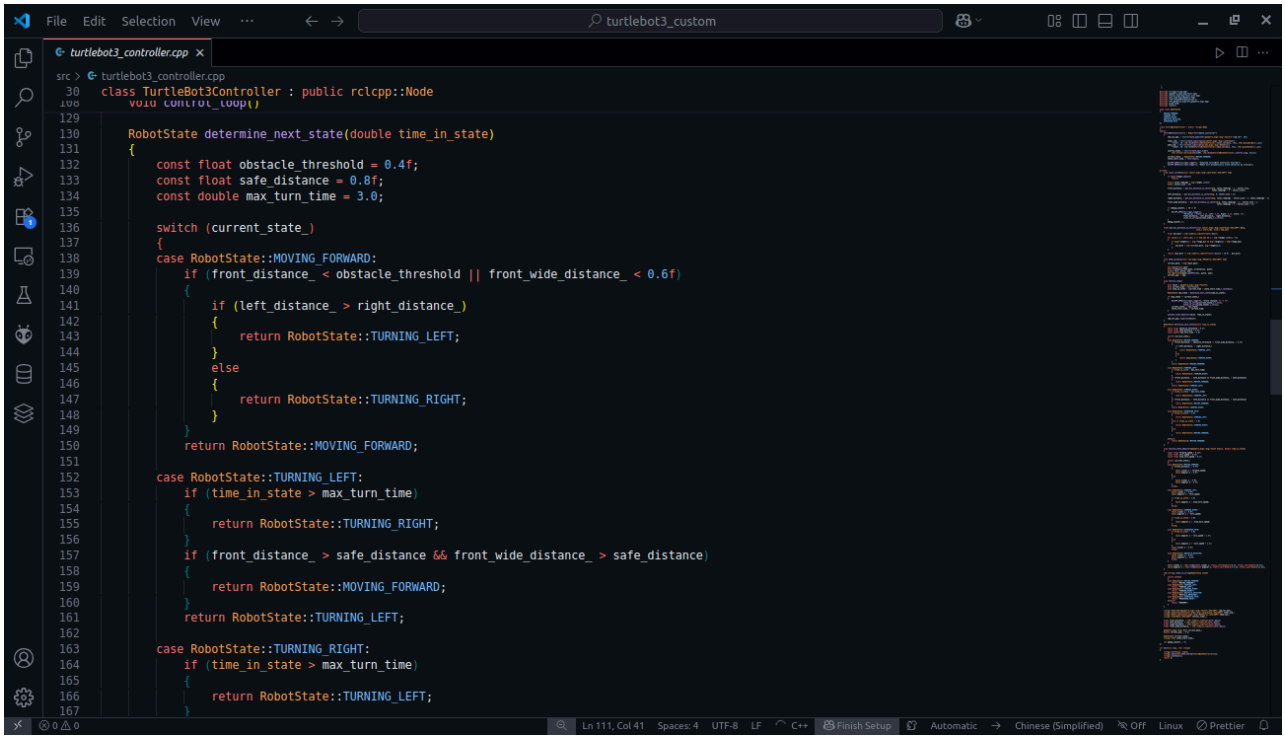
```
)

ament_target_dependencies(navigation_client
  rclcpp
  rclcpp_action
  geometry_msgs
  nav_msgs
)

rosidl_get_typesupport_target(robot_status_server_typesupport ${PROJECT_NAME} "rosidl_typesupport_cpp")
target_link_libraries(robot_status_server ${robot_status_server_typesupport})

rosidl_get_typesupport_target(status_client_typesupport ${PROJECT_NAME} "rosidl_typesupport_cpp")
target_link_libraries(status_client ${status_client_typesupport})

rosidl_get_typesupport_target(navigation_action_server_typesupport ${PROJECT_NAME} "rosidl_typesupport_cpp")
target_link_libraries(navigation_action_server ${navigation_action_server_typesupport})

rosidl_get_typesupport_target(navigation_client_typesupport ${PROJECT_NAME} "rosidl_typesupport_cpp")
target_link_libraries(navigation_client ${navigation_client_typesupport})

install(TARGETS
  turtlebot3_controller
  robot_status_server
  status_client
  navigation_action_server
  navigation_client
  DESTINATION lib/${PROJECT_NAME}
)

install(DIRECTORY launch
  DESTINATION share/${PROJECT_NAME}/
)

if(BUILD_TESTING)
  find_package(ament_lint_auto REQUIRED)
  # the following line skips the linter which checks for copyrights
  # comment the line when a copyright and license is added to all source files
  set(ament_cmake_copyright_FOUND TRUE)
  # the following line skips cpplint (only works in a git repo)
  # comment the line when this package is in a git repo and when
```

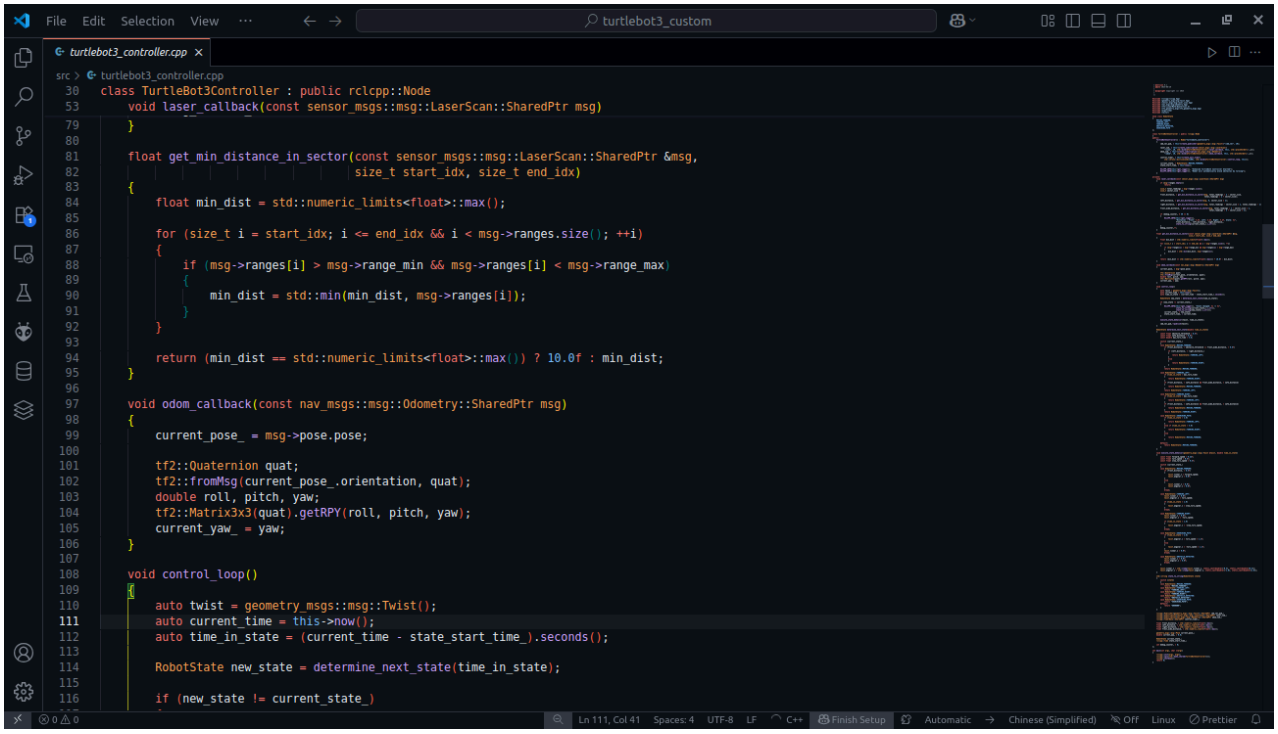
3. Creating a Basic Node

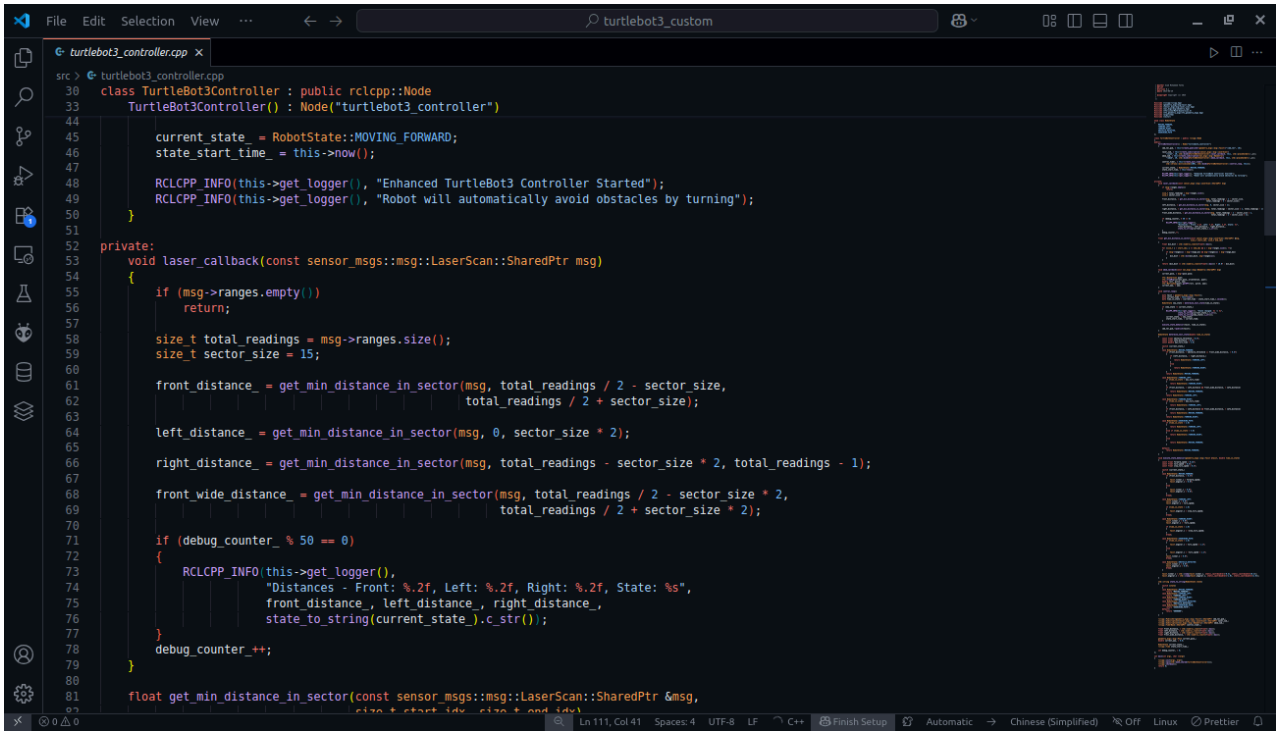
Basic TurtleBot3 Controller Node

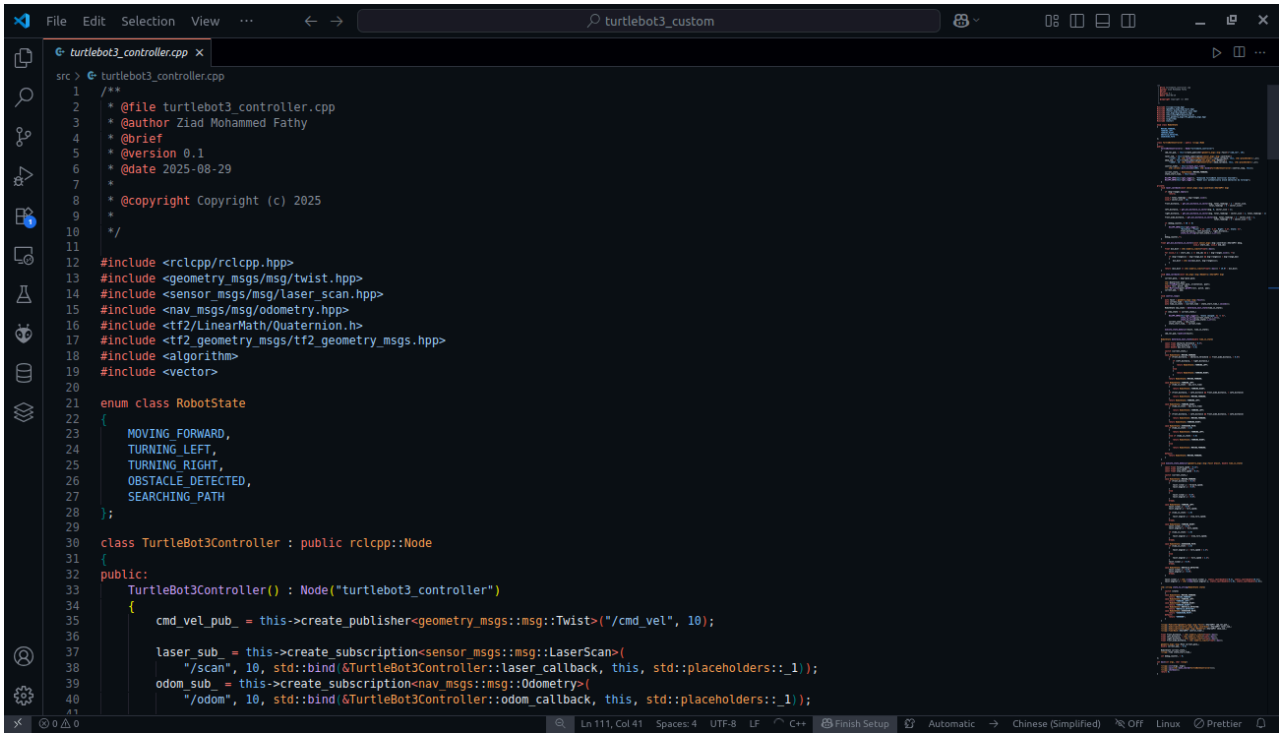
Create `src/turtlebot3_controller.cpp`:






File Edit Selection View
turtlebot3_custom
turtlebot3_controller.cpp
src > turtlebot3_controller.cpp
class TurtleBot3Controller : public rclcpp::Node
void laser_callback(const sensor_msgs::msg::LaserScan::SharedPtr msg)
float get_min_distance_in_sector(const sensor_msgs::msg::LaserScan::SharedPtr &msg,
size_t start_idx, size_t end_idx)
float min_dist = std::numeric_limits<float>::max();
for (size_t i = start_idx; i <= end_idx && i < msg->ranges.size(); ++i)
if (msg->ranges[i] > msg->range_min && msg->ranges[i] < msg->range_max)
min_dist = std::min(min_dist, msg->ranges[i]);
return (min_dist == std::numeric_limits<float>::max()) ? 10.0f : min_dist;
void odom_callback(const nav_msgs::msg::Odometry::SharedPtr msg)
current_pose_ = msg->pose.pose;
tf2::Quaternion quat;
tf2::fromMsg(current_pose_.orientation, quat);
double roll, pitch, yaw;
tf2::Matrix3x3(quat).getRPY(roll, pitch, yaw);
current_yaw_ = yaw;
void control_loop()
auto twist = geometry_msgs::msg::Twist();
auto current_time = this->now();
auto time_in_state = (current_time - state_start_time_).seconds();
RobotState new_state = determine_next_state(time_in_state);
if (new_state != current_state_)
Ln 111, Col 41 Spaces: 4 UTF-8 LF C++ Finish Setup Automatic Chinese (Simplified) Off Linux Prettier

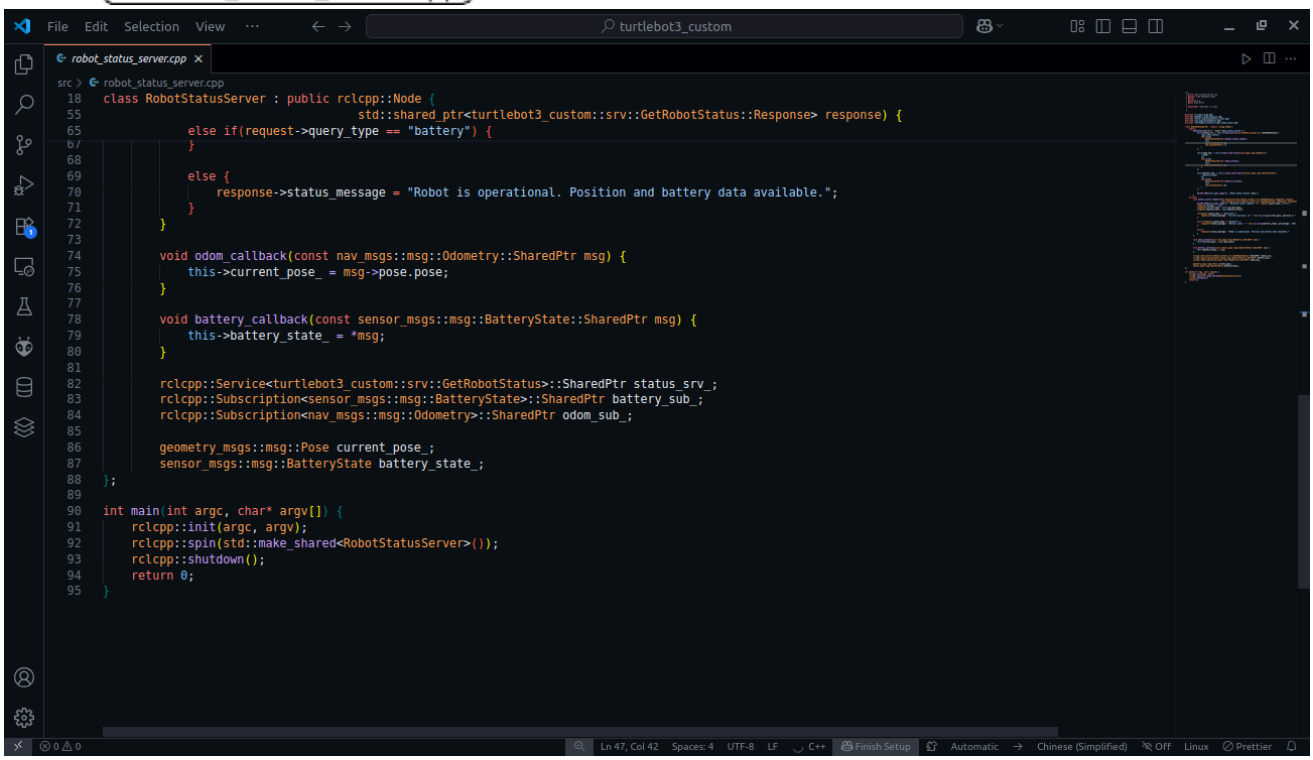

File Edit Selection View
turtlebot3_custom
turtlebot3_controller.cpp
src > turtlebot3_controller.cpp
class TurtleBot3Controller : public rclcpp::Node
TurtleBot3Controller() : Node("turtlebot3_controller")
current_state_ = RobotState::MOVING_FORWARD;
state_start_time_ = this->now();
RCLCPP_INFO(this->get_logger(), "Enhanced TurtleBot3 Controller Started");
RCLCPP_INFO(this->get_logger(), "Robot will automatically avoid obstacles by turning");
private:
void laser_callback(const sensor_msgs::msg::LaserScan::SharedPtr msg)
if (msg->ranges.empty())
return;
size_t total_readings = msg->ranges.size();
size_t sector_size = 15;
front_distance_ = get_min_distance_in_sector(msg, total_readings / 2 - sector_size,
total_readings / 2 + sector_size);
left_distance_ = get_min_distance_in_sector(msg, 0, sector_size * 2);
right_distance_ = get_min_distance_in_sector(msg, total_readings - sector_size * 2, total_readings - 1);
front_wide_distance_ = get_min_distance_in_sector(msg, total_readings / 2 - sector_size * 2,
total_readings / 2 + sector_size * 2);
if (debug_counter_ % 50 == 0)
RCLCPP_INFO(this->get_logger(),
"Distances - Front: %.2f, Left: %.2f, Right: %.2f, State: %s",
front_distance_, left_distance_, right_distance_,
state_to_string(current_state_).c_str());
debug_counter_++;
float get_min_distance_in_sector(const sensor_msgs::msg::LaserScan::SharedPtr &msg,
Ln 111, Col 41 Spaces: 4 UTF-8 LF C++ Finish Setup Automatic Chinese (Simplified) Off Linux Prettier


File Edit Selection View ···
turtlebot3_custom
turtlebot3_controller.cpp
src > turtlebot3_controller.cpp
/**
* @file turtlebot3_controller.cpp
* @author Ziad Mohammed Fathy
* @brief
* @version 0.1
* @date 2025-08-29
*
* @copyright Copyright (c) 2025
*
*/
#include <rclcpp/rclcpp.hpp>
#include <geometry_msgs/msg/twist.hpp>
#include <sensor_msgs/msg/laser_scan.hpp>
#include <nav_msgs/msg/odometry.hpp>
#include <tf2/LinearMath/Quaternion.h>
#include <tf2_geometry_msgs/tf2_geometry_msgs.hpp>
#include <algorithm>
#include <vector>
enum class RobotState
{
MOVING_FORWARD,
TURNING_LEFT,
TURNING_RIGHT,
OBSTACLE_DETECTED,
SEARCHING_PATH
};
class TurtleBot3Controller : public rclcpp::Node
{
public:
TurtleBot3Controller() : Node("turtlebot3_controller")
{
cmd_vel_pub_ = this->create_publisher<geometry_msgs::msg::Twist>("/cmd_vel", 10);
laser_sub_ = this->create_subscription<sensor_msgs::msg::LaserScan>(
"/scan", 10, std::bind(&TurtleBot3Controller::laser_callback, this, std::placeholders::_1));
odom_sub_ = this->create_subscription<nav_msgs::msg::Odometry>(
"/odom", 10, std::bind(&TurtleBot3Controller::odom_callback, this, std::placeholders::_1));
Ln 111, Col 41 Spaces: 4 UTF-8 LF C++ Finish Setup Automatic Chinese (Simplified) Off Linux Prettier

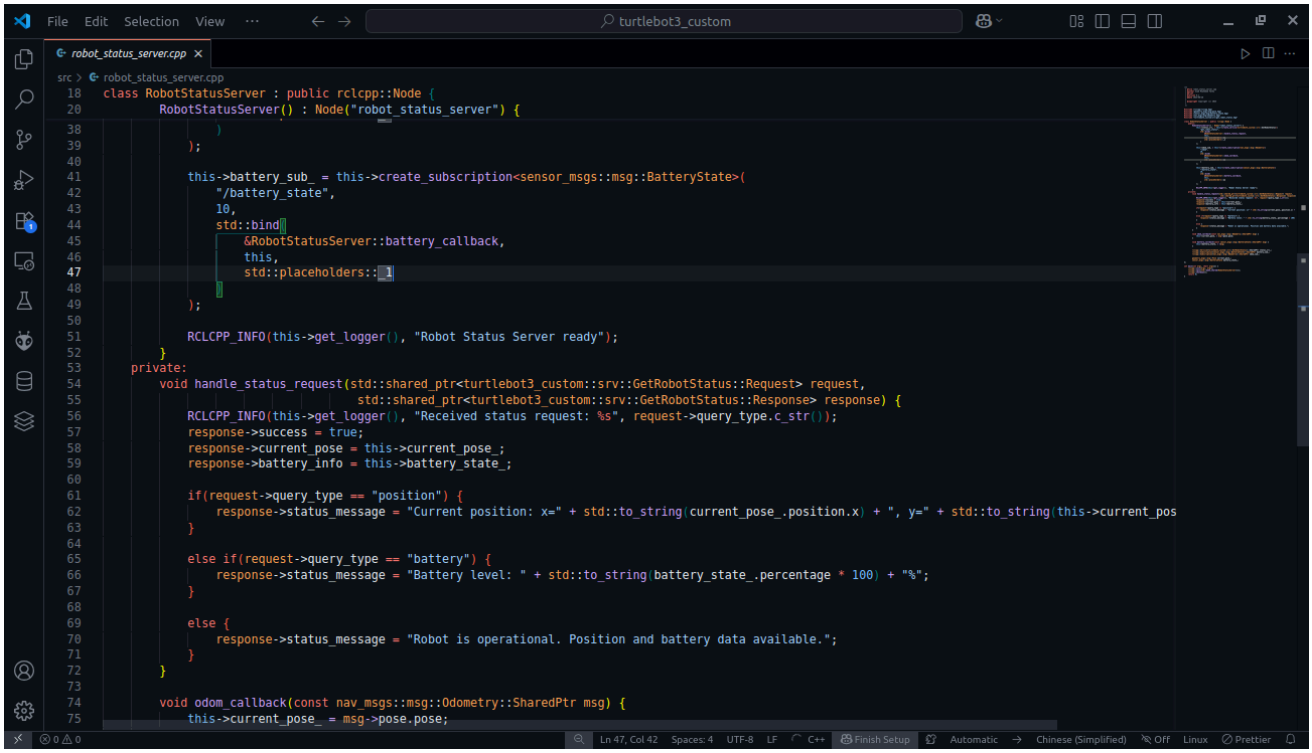
4. Creating a Service Server

Service Server Node

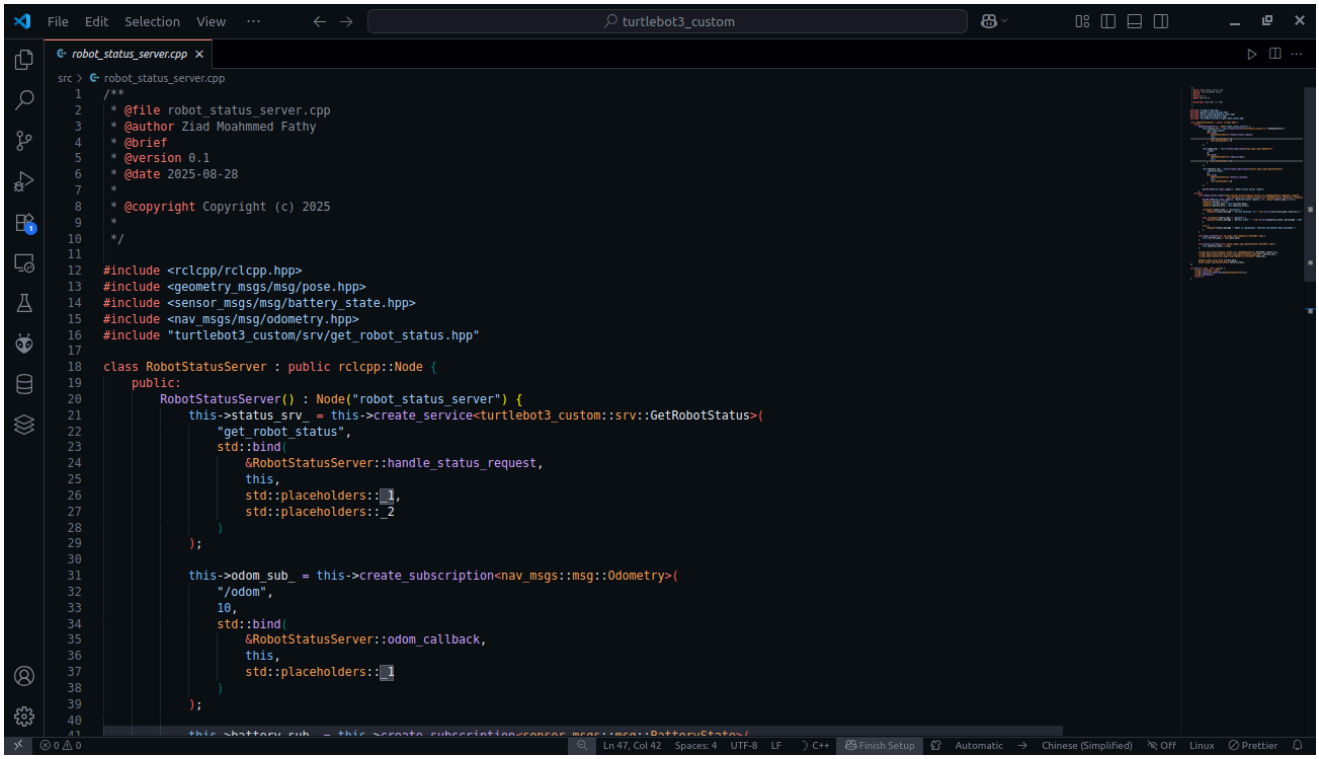
Create `src/robot_status_server.cpp`:



```cpp
class RobotStatusServer : public rclcpp::Node {
                                  std::shared_ptr<turtlebot3_custom::srv::GetRobotStatus::Response> response) {
            else if(request->query_type == "battery") {
            }

            else {
                response->status_message = "Robot is operational. Position and battery data available.";
            }
        }

        void odom_callback(const nav_msgs::msg::Odometry::SharedPtr msg) {
            this->current_pose_ = msg->pose.pose;
        }

        void battery_callback(const sensor_msgs::msg::BatteryState::SharedPtr msg) {
            this->battery_state_ = *msg;
        }

        rclcpp::Service<turtlebot3_custom::srv::GetRobotStatus>::SharedPtr status_srv_;
        rclcpp::Subscription<sensor_msgs::msg::BatteryState>::SharedPtr battery_sub_;
        rclcpp::Subscription<nav_msgs::msg::Odometry>::SharedPtr odom_sub_;

        geometry_msgs::msg::Pose current_pose_;
        sensor_msgs::msg::BatteryState battery_state_;
};

int main(int argc, char* argv[]) {
    rclcpp::init(argc, argv);
    rclcpp::spin(std::make_shared<RobotStatusServer>());
    rclcpp::shutdown();
    return 0;
}
```

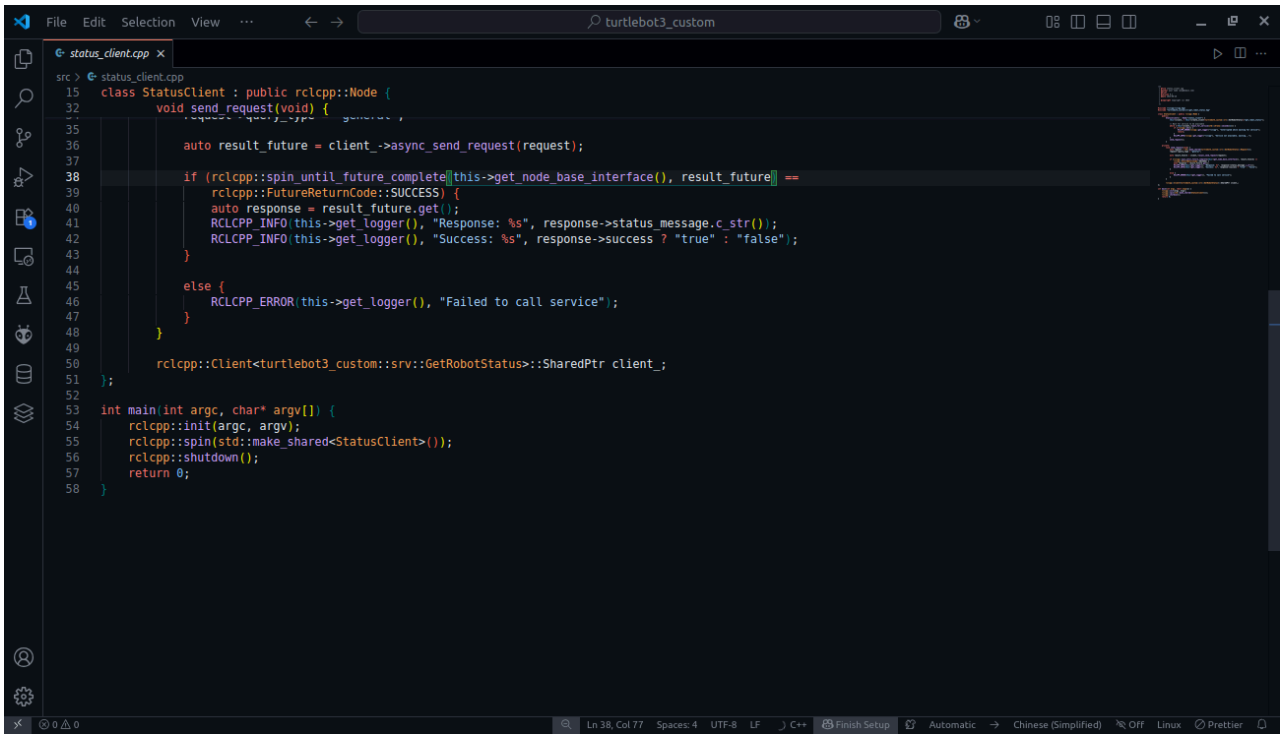


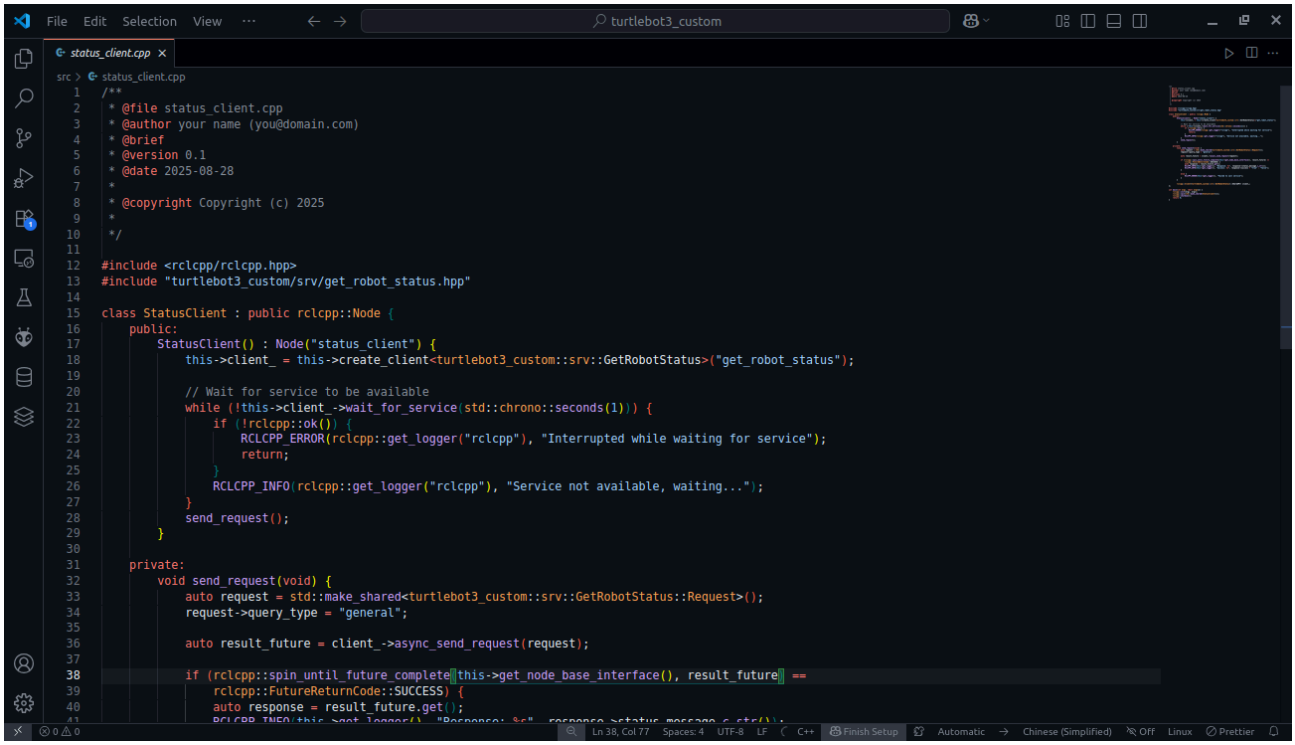
```cpp
class RobotStatusServer : public rclcpp::Node {
        RobotStatusServer() : Node("robot_status_server") {
            )
        );

        this->battery_sub_ = this->create_subscription<sensor_msgs::msg::BatteryState>(
            "/battery_state",
            10,
            std::bind(
                &RobotStatusServer::battery_callback,
                this,
                std::placeholders::_1
            )
        );

        RCLCPP_INFO(this->get_logger(), "Robot Status Server ready");
    }
private:
    void handle_status_request(std::shared_ptr<turtlebot3_custom::srv::GetRobotStatus::Request> request,
                               std::shared_ptr<turtlebot3_custom::srv::GetRobotStatus::Response> response) {
        RCLCPP_INFO(this->get_logger(), "Received status request: %s", request->query_type.c_str());
        response->success = true;
        response->current_pose = this->current_pose_;
        response->battery_info = this->battery_state_;

        if(request->query_type == "position") {
            response->status_message = "Current position: x=" + std::to_string(current_pose_.position.x) + ", y=" + std::to_string(this->current_pos
        }

        else if(request->query_type == "battery") {
            response->status_message = "Battery level: " + std::to_string(battery_state_.percentage * 100) + "%";
        }

        else {
            response->status_message = "Robot is operational. Position and battery data available.";
        }
    }

    void odom_callback(const nav_msgs::msg::Odometry::SharedPtr msg) {
        this->current_pose_ = msg->pose.pose;
```



Service Client Example

Create `src/status_client.cpp`:

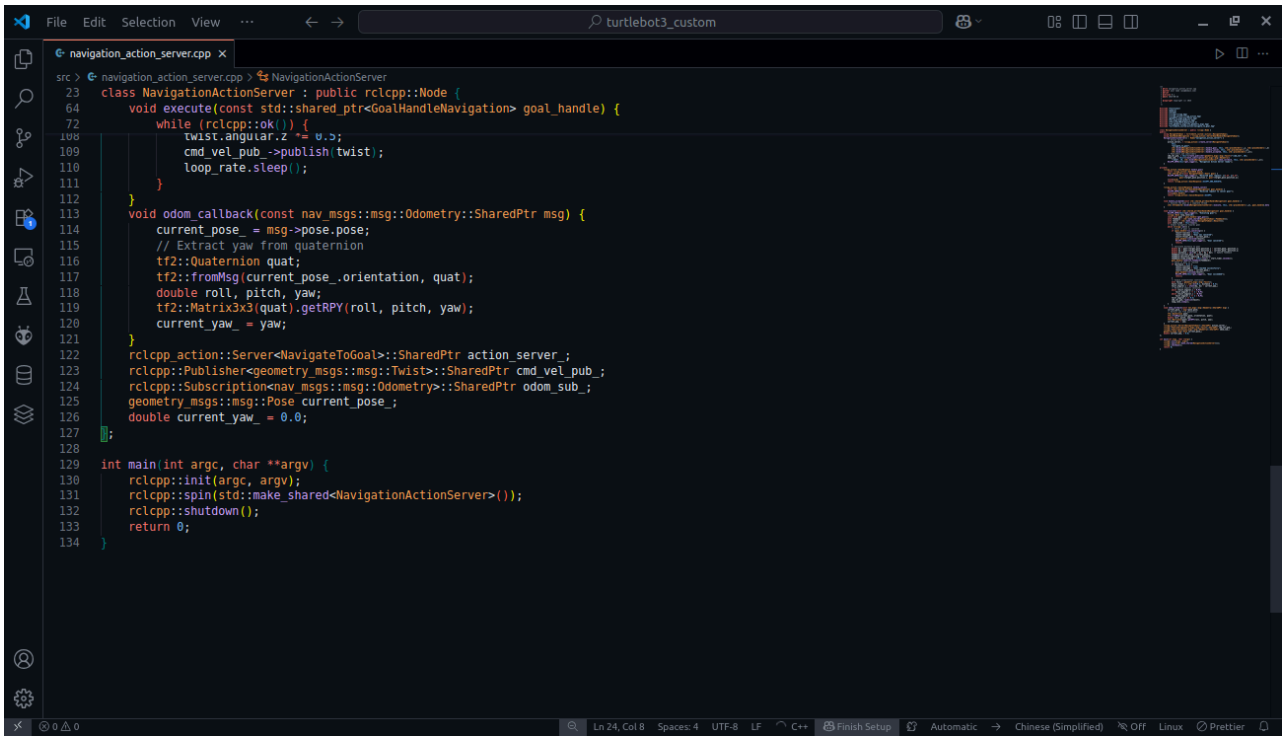


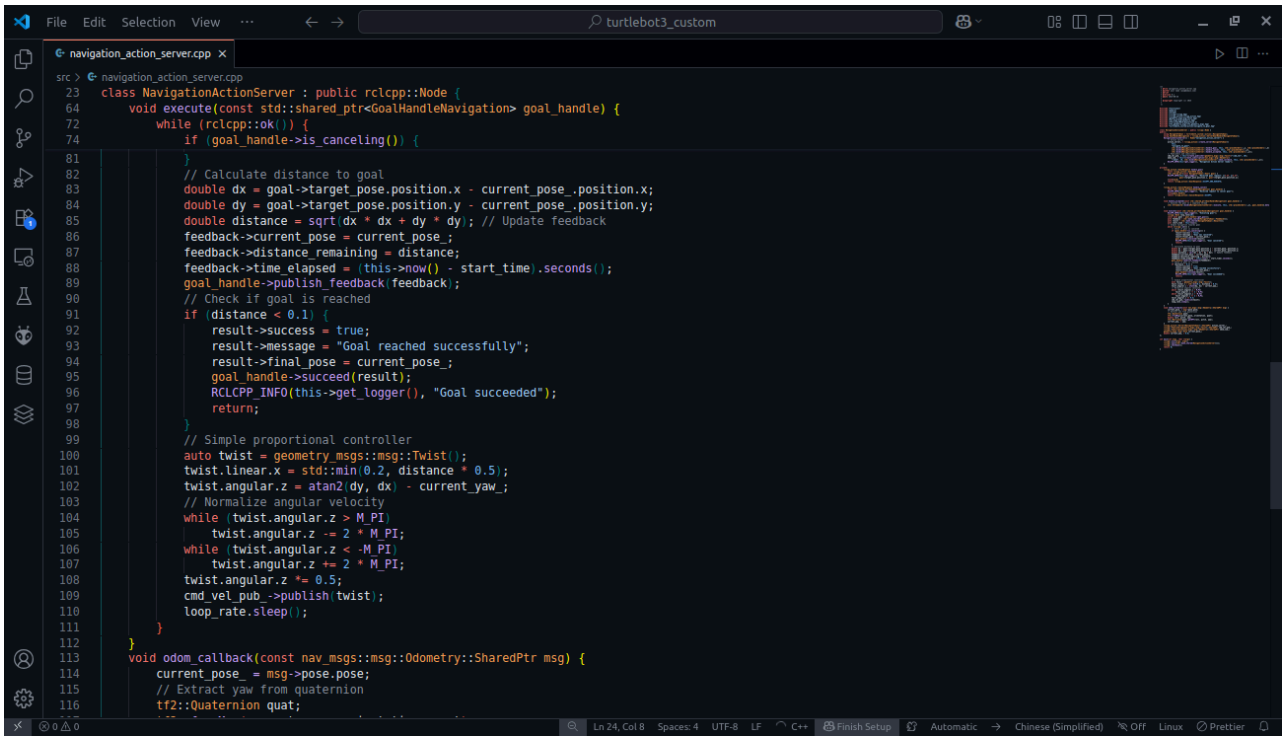

File Edit Selection View ···
turtlebot3_custom
status_client.cpp
src > status_client.cpp
/**
* @file status_client.cpp
* @author your name (you@domain.com)
* @brief
* @version 0.1
* @date 2025-08-28
*
* @copyright Copyright (c) 2025
*
*/
#include <rclcpp/rclcpp.hpp>
#include "turtlebot3_custom/srv/get_robot_status.hpp"
class StatusClient : public rclcpp::Node {
public:
StatusClient() : Node("status_client") {
this->client_ = this->create_client<turtlebot3_custom::srv::GetRobotStatus>("get_robot_status");
// Wait for service to be available
while (!this->client_->wait_for_service(std::chrono::seconds(1))) {
if (!rclcpp::ok()) {
RCLCPP_ERROR(rclcpp::get_logger("rclcpp"), "Interrupted while waiting for service");
return;
}
RCLCPP_INFO(rclcpp::get_logger("rclcpp"), "Service not available, waiting...");
}
send_request();
}
private:
void send_request(void) {
auto request = std::make_shared<turtlebot3_custom::srv::GetRobotStatus::Request>();
request->query_type = "general";
auto result_future = client_->async_send_request(request);
if (rclcpp::spin_until_future_complete(this->get_node_base_interface(), result_future) ==
rclcpp::FutureReturnCode::SUCCESS) {
auto response = result_future.get();
Ln 38, Col 77 Spaces: 4 UTF-8 LF C++ Finish Setup Automatic Chinese (Simplified) Off Linux Prettier

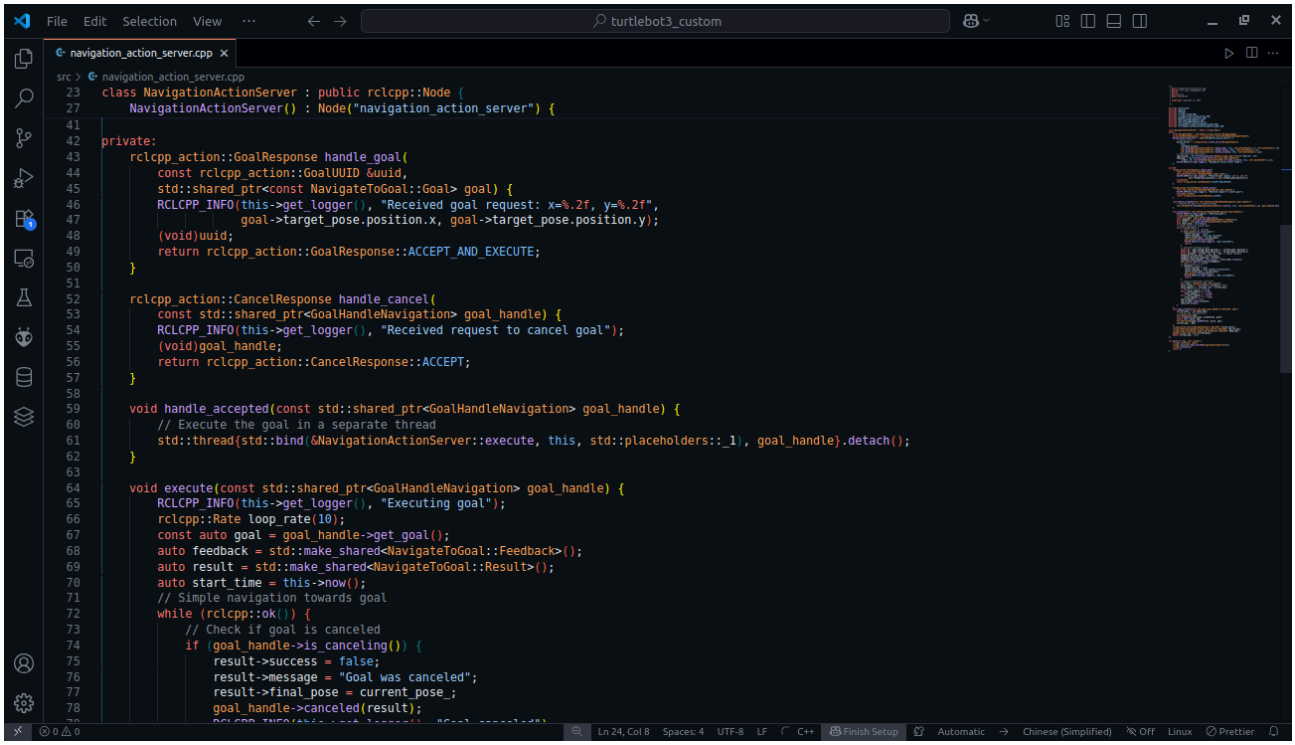
5. Creating an Action Server

Action Server Node

Create `src/navigation_action_server.cpp`:

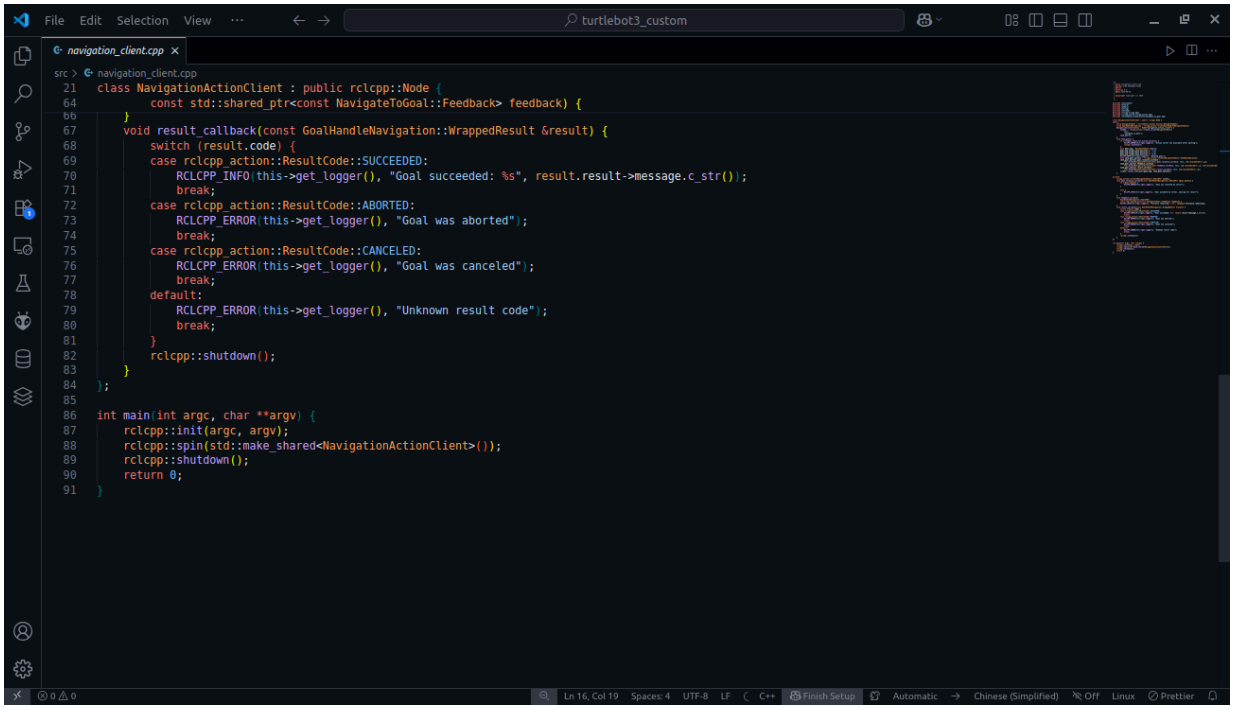




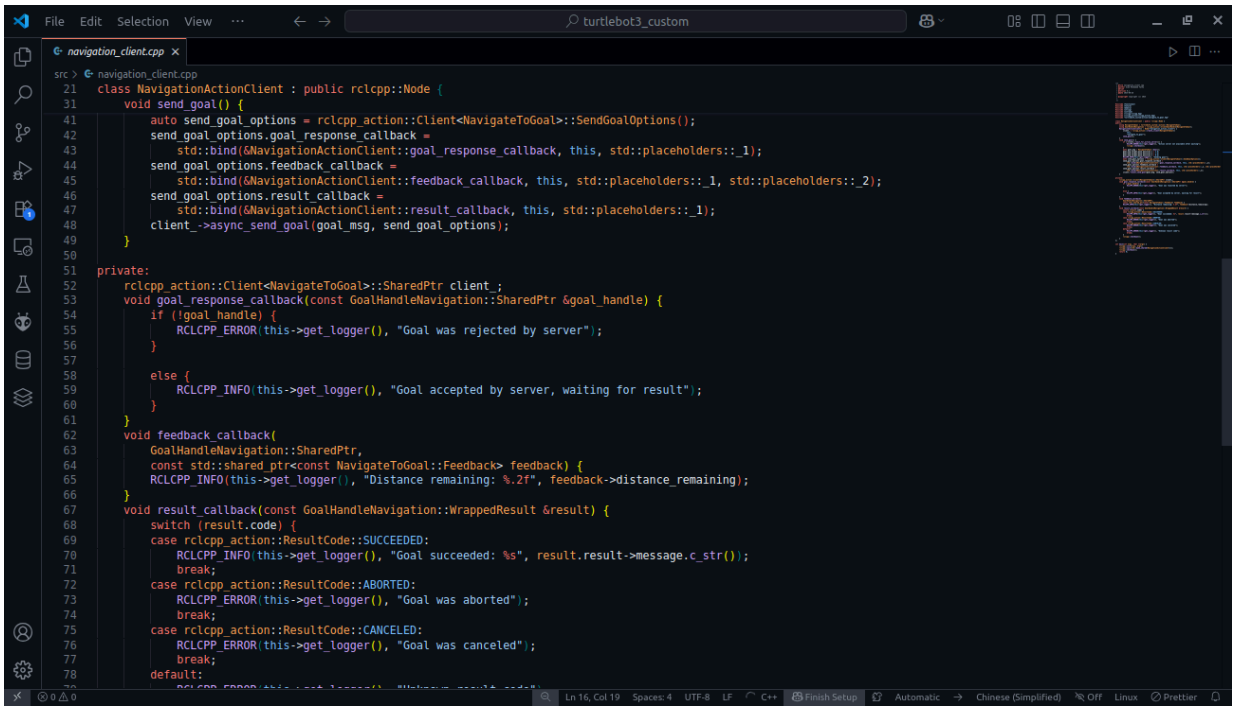

File Edit Selection View ···
turtlebot3_custom
navigation_action_server.cpp
src > navigation_action_server.cpp
class NavigationActionServer : public rclcpp::Node {
NavigationActionServer() : Node("navigation_action_server") {
private:
rclcpp_action::GoalResponse handle_goal(
const rclcpp_action::GoalUUID &uuid,
std::shared_ptr<const NavigateToGoal::Goal> goal) {
RCLCPP_INFO(this->get_logger(), "Received goal request: x=%.2f, y=%.2f",
goal->target_pose.position.x, goal->target_pose.position.y);
(void)uuid;
return rclcpp_action::GoalResponse::ACCEPT_AND_EXECUTE;
}
rclcpp_action::CancelResponse handle_cancel(
const std::shared_ptr<GoalHandleNavigation> goal_handle) {
RCLCPP_INFO(this->get_logger(), "Received request to cancel goal");
(void)goal_handle;
return rclcpp_action::CancelResponse::ACCEPT;
}
void handle_accepted(const std::shared_ptr<GoalHandleNavigation> goal_handle) {
// Execute the goal in a separate thread
std::thread{std::bind(&NavigationActionServer::execute, this, std::placeholders::_1), goal_handle}.detach();
}
void execute(const std::shared_ptr<GoalHandleNavigation> goal_handle) {
RCLCPP_INFO(this->get_logger(), "Executing goal");
rclcpp::Rate loop_rate(10);
const auto goal = goal_handle->get_goal();
auto feedback = std::make_shared<NavigateToGoal::Feedback>();
auto result = std::make_shared<NavigateToGoal::Result>();
auto start_time = this->now();
// Simple navigation towards goal
while (rclcpp::ok()) {
// Check if goal is canceled
if (goal_handle->is_canceling()) {
result->success = false;
result->message = "Goal was canceled";
result->final_pose = current_pose_;
goal_handle->canceled(result);
Ln 24, Col 8 Spaces: 4 UTF-8 LF C++ Finish Setup Automatic Chinese (Simplified) Off Linux Prettier

Action Client Example

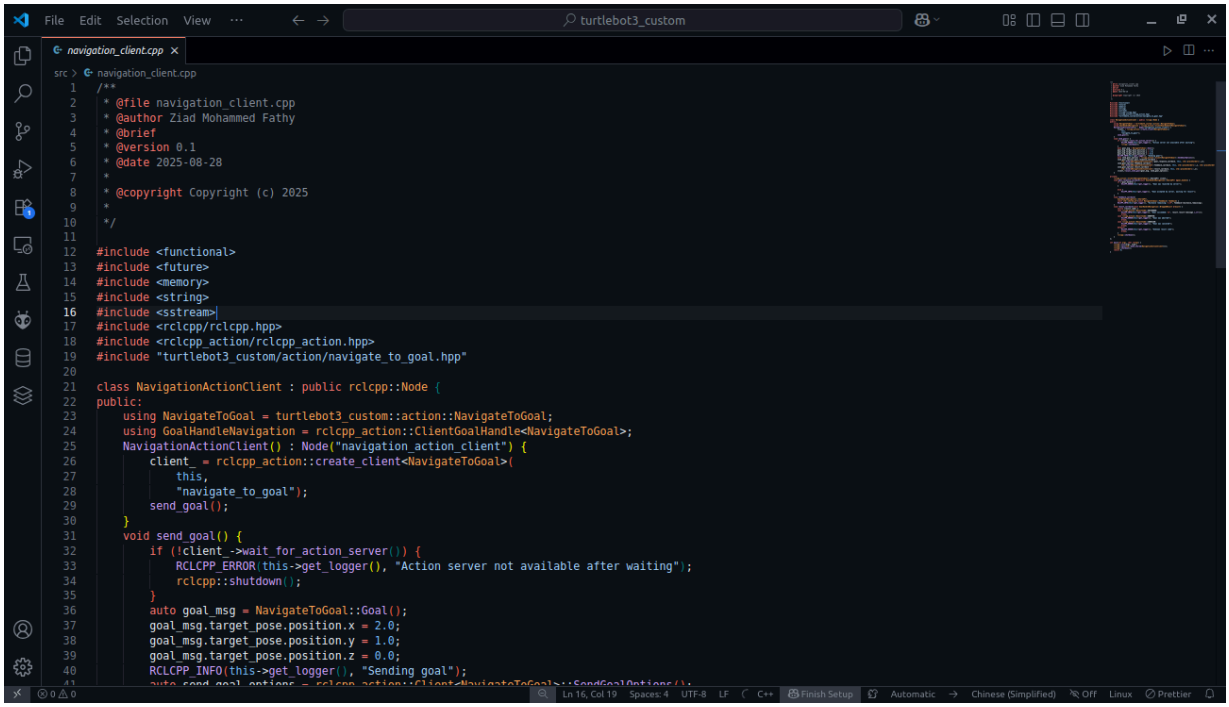
Create `src/navigation_client.cpp`:



```cpp
class NavigationActionClient : public rclcpp::Node {
        const std::shared_ptr<const NavigateToGoal::Feedback> feedback) {
    }
    void result_callback(const GoalHandleNavigation::WrappedResult &result) {
        switch (result.code) {
        case rclcpp_action::ResultCode::SUCCEEDED:
            RCLCPP_INFO(this->get_logger(), "Goal succeeded: %s", result.result->message.c_str());
            break;
        case rclcpp_action::ResultCode::ABORTED:
            RCLCPP_ERROR(this->get_logger(), "Goal was aborted");
            break;
        case rclcpp_action::ResultCode::CANCELED:
            RCLCPP_ERROR(this->get_logger(), "Goal was canceled");
            break;
        default:
            RCLCPP_ERROR(this->get_logger(), "Unknown result code");
            break;
        }
        rclcpp::shutdown();
    }
};

int main(int argc, char **argv) {
    rclcpp::init(argc, argv);
    rclcpp::spin(std::make_shared<NavigationActionClient>());
    rclcpp::shutdown();
    return 0;
}
```



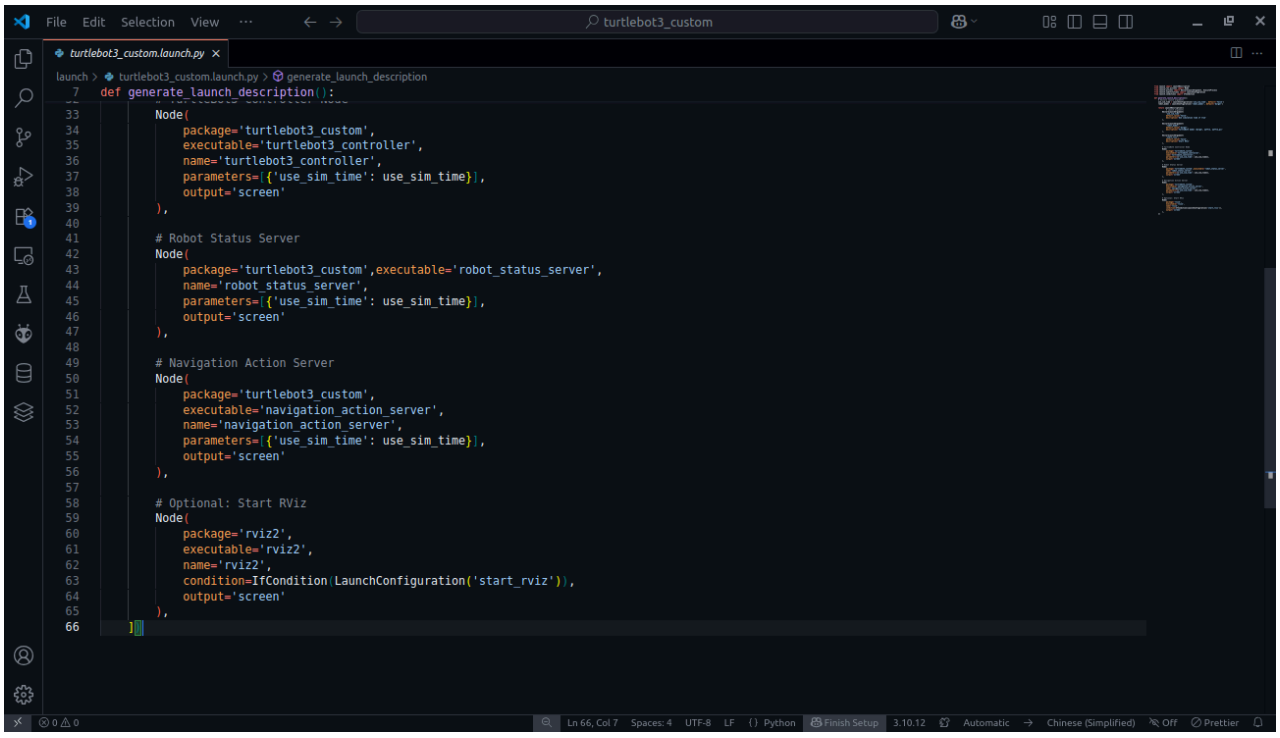
```cpp
class NavigationActionClient : public rclcpp::Node {
    void send_goal() {
        auto send_goal_options = rclcpp_action::Client<NavigateToGoal>::SendGoalOptions();
        send_goal_options.goal_response_callback =
            std::bind(&NavigationActionClient::goal_response_callback, this, std::placeholders::_1);
        send_goal_options.feedback_callback =
            std::bind(&NavigationActionClient::feedback_callback, this, std::placeholders::_1, std::placeholders::_2);
        send_goal_options.result_callback =
            std::bind(&NavigationActionClient::result_callback, this, std::placeholders::_1);
        client_->async_send_goal(goal_msg, send_goal_options);
    }

private:
    rclcpp_action::Client<NavigateToGoal>::SharedPtr client_;
    void goal_response_callback(const GoalHandleNavigation::SharedPtr &goal_handle) {
        if (!goal_handle) {
            RCLCPP_ERROR(this->get_logger(), "Goal was rejected by server");
        }

        else {
            RCLCPP_INFO(this->get_logger(), "Goal accepted by server, waiting for result");
        }
    }
    void feedback_callback(
        GoalHandleNavigation::SharedPtr,
        const std::shared_ptr<const NavigateToGoal::Feedback> feedback) {
        RCLCPP_INFO(this->get_logger(), "Distance remaining: %.2f", feedback->distance_remaining);
    }
    void result_callback(const GoalHandleNavigation::WrappedResult &result) {
        switch (result.code) {
        case rclcpp_action::ResultCode::SUCCEEDED:
            RCLCPP_INFO(this->get_logger(), "Goal succeeded: %s", result.result->message.c_str());
            break;
        case rclcpp_action::ResultCode::ABORTED:
            RCLCPP_ERROR(this->get_logger(), "Goal was aborted");
            break;
        case rclcpp_action::ResultCode::CANCELED:
            RCLCPP_ERROR(this->get_logger(), "Goal was canceled");
            break;
        default:
```

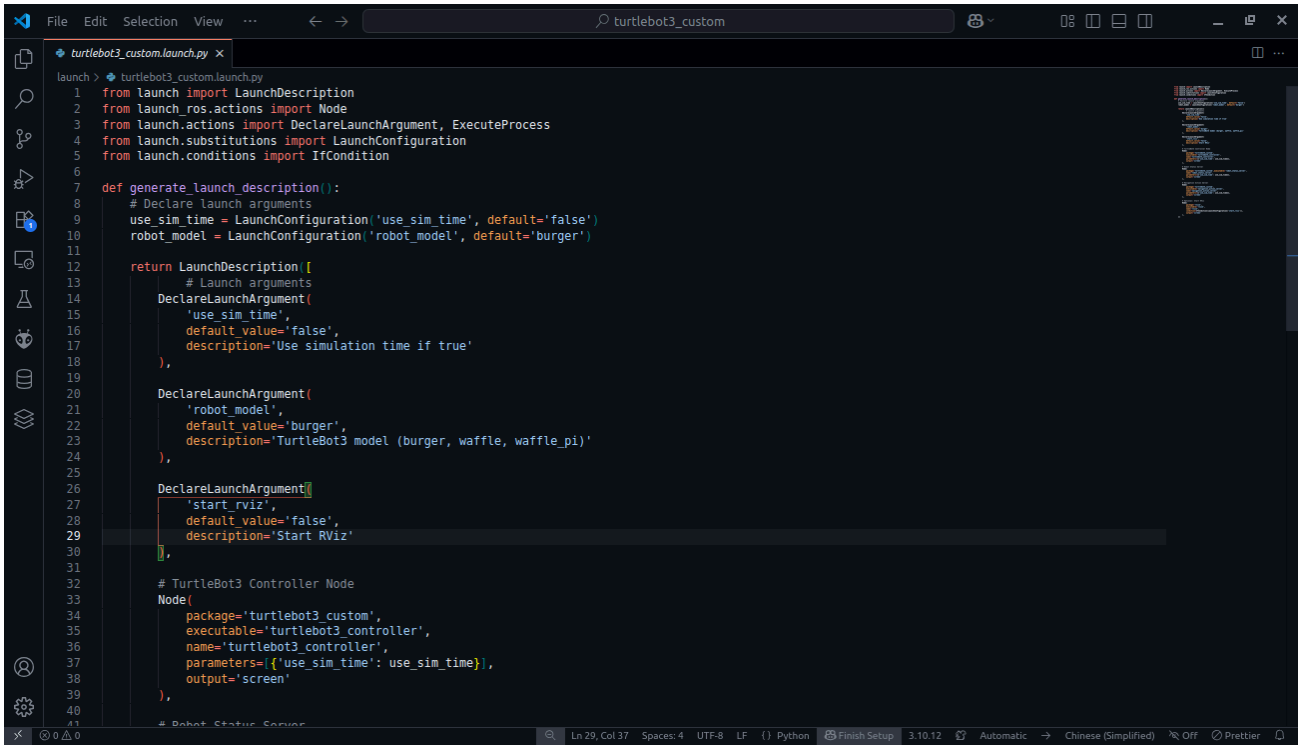


6. Launch Files

Main Launch File

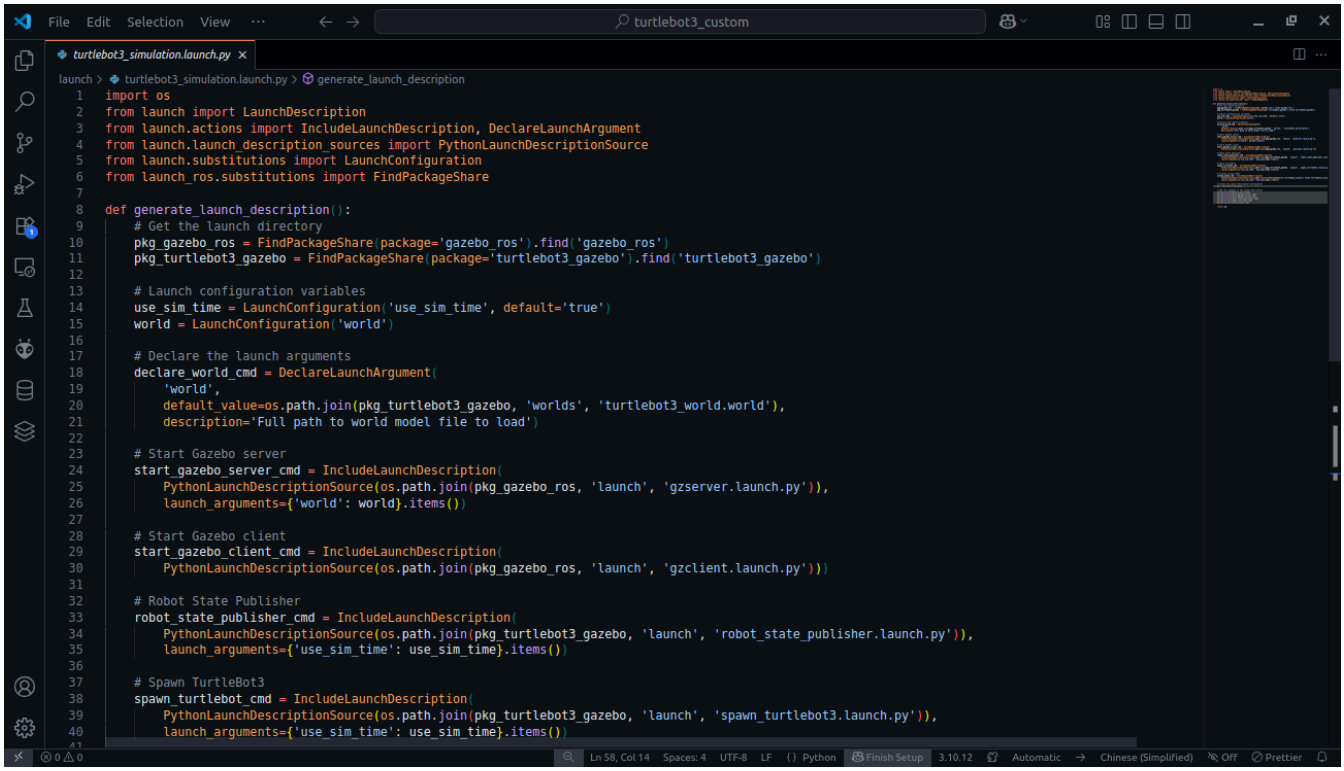
Create `launch/turtlebot3_custom.launch.py`:





Simulation Launch File

Create `launch/turtlebot3_simulation.launch.py`:

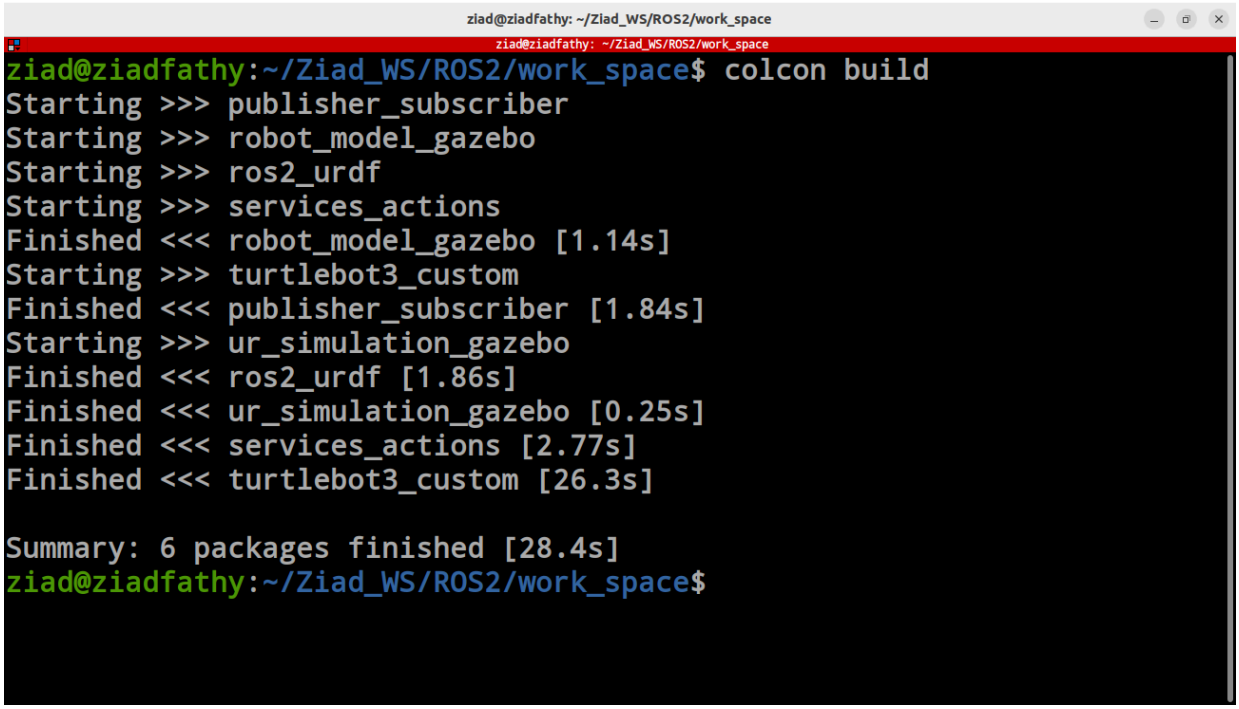


Running Options

Option 1: Simulation (Recommended for Testing)

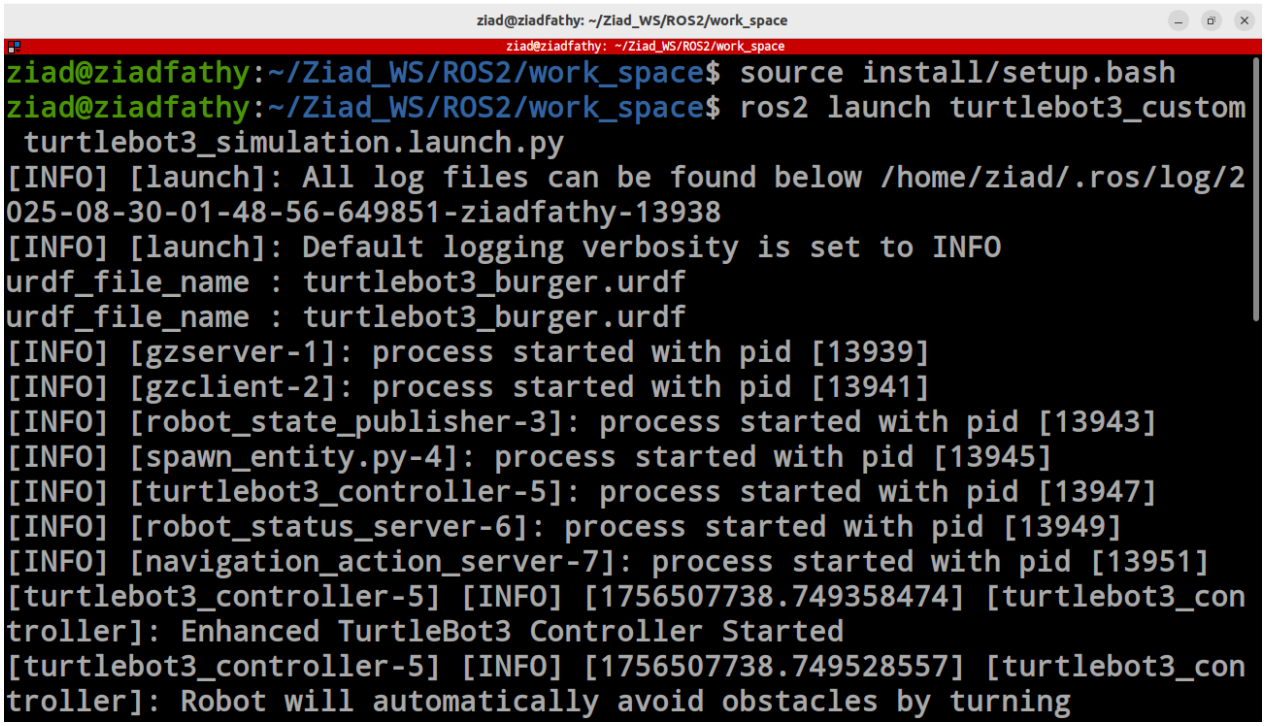
Method 1A: Complete Simulation Launch:

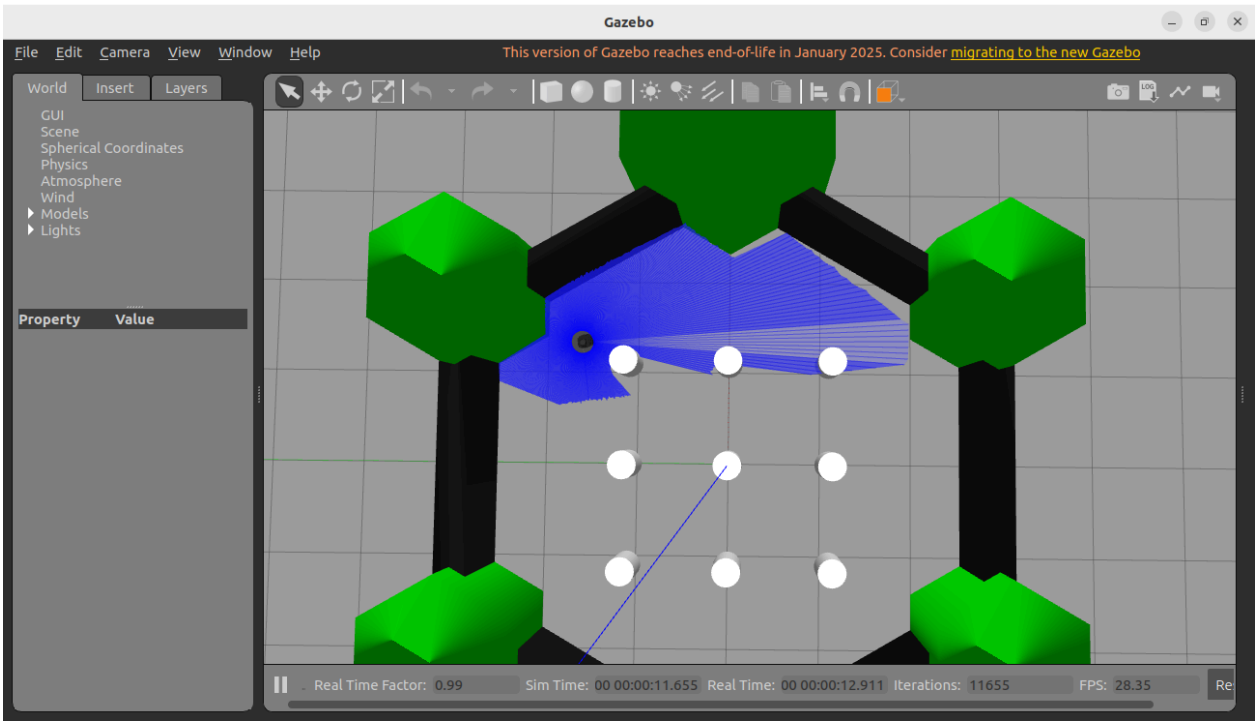
colcon build



source install/setup.bash

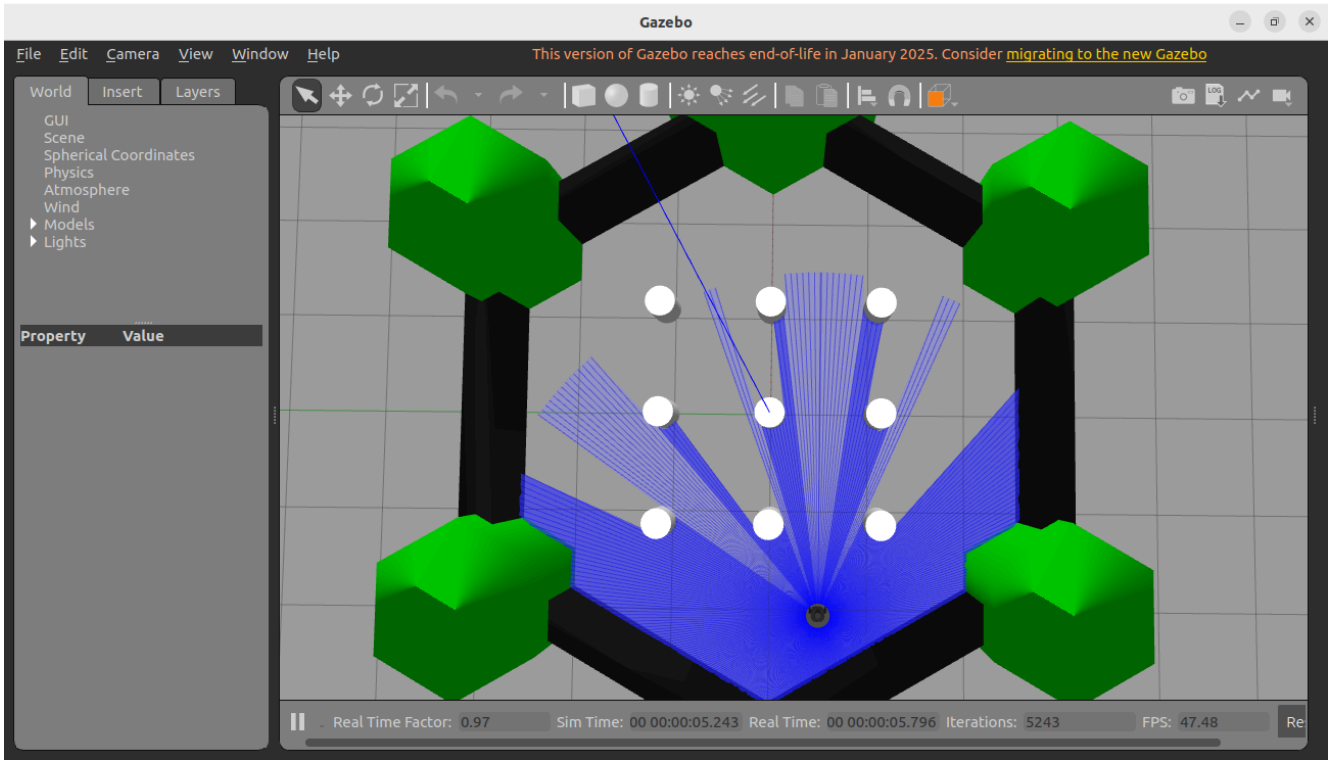
ros2 launch turtlebot3_custom turtlebot3_simulation.launch.py



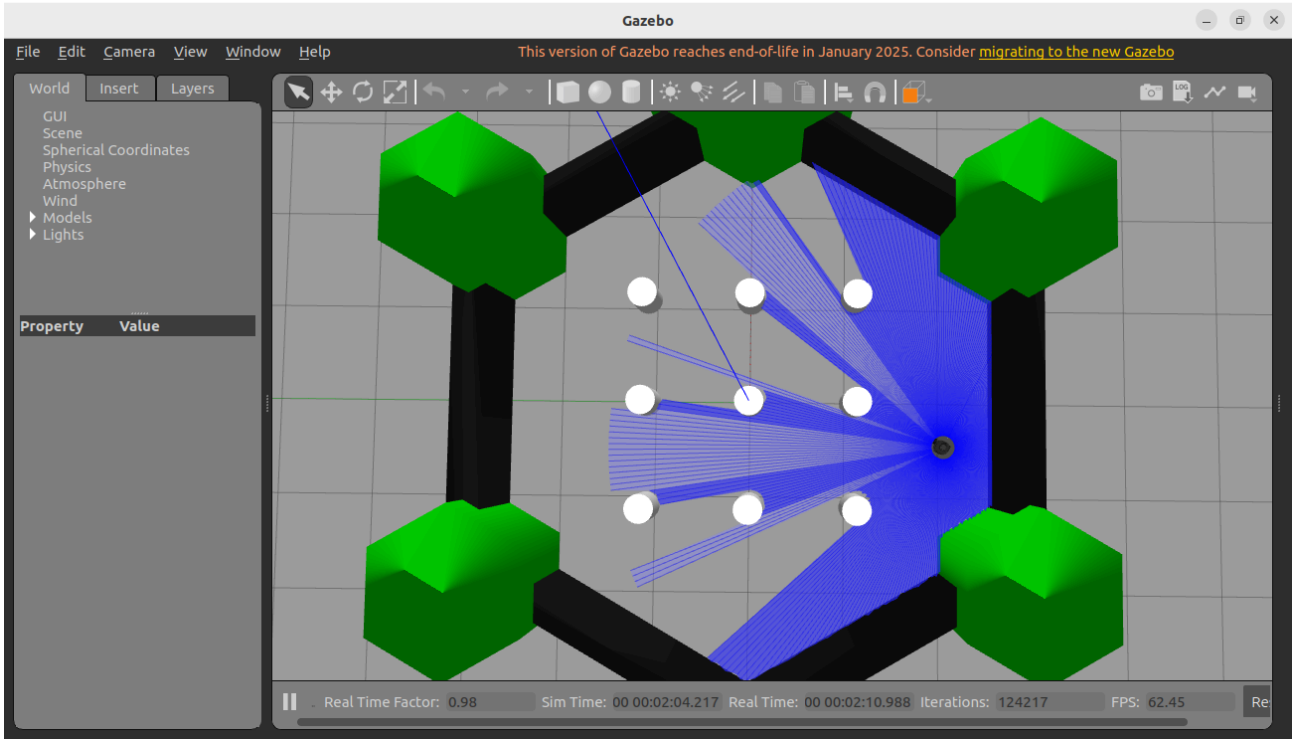


Method 1B: Step-by-Step Simulation:

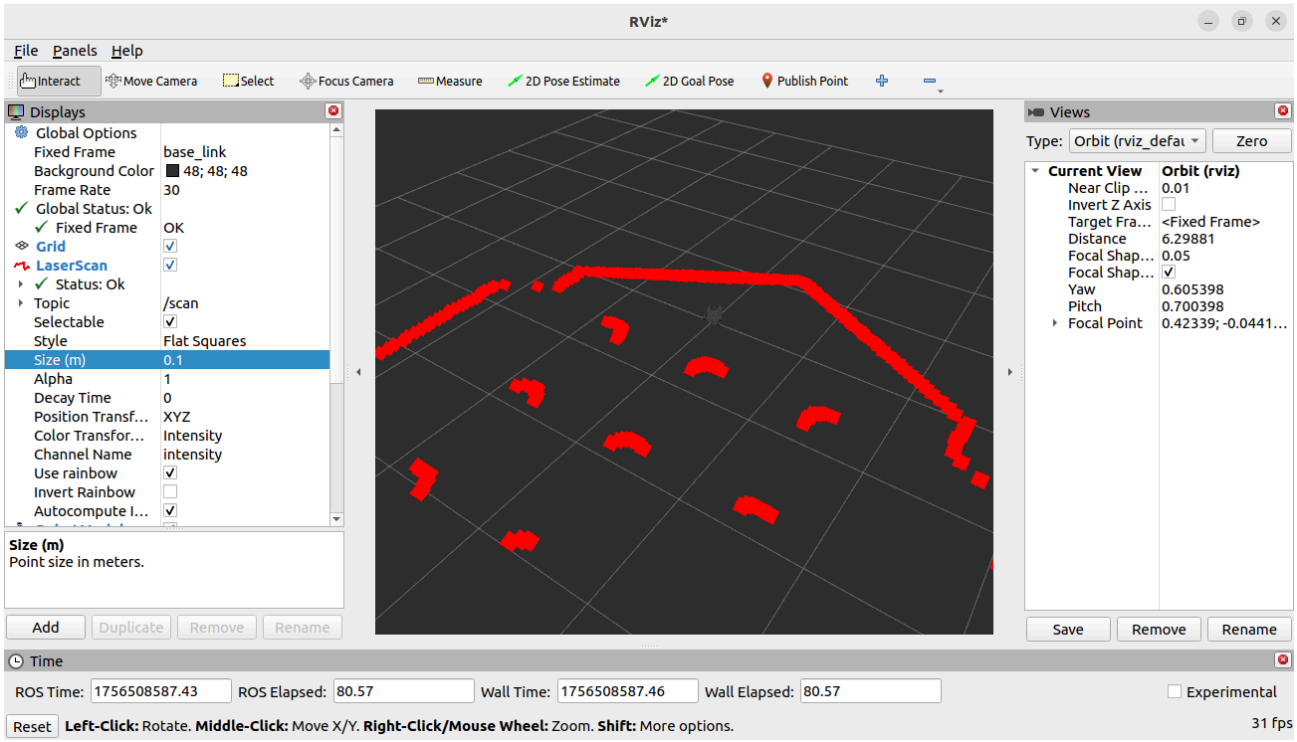
ros2 launch turtlebot3_gazebo turtlebot3_world.launch.py



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

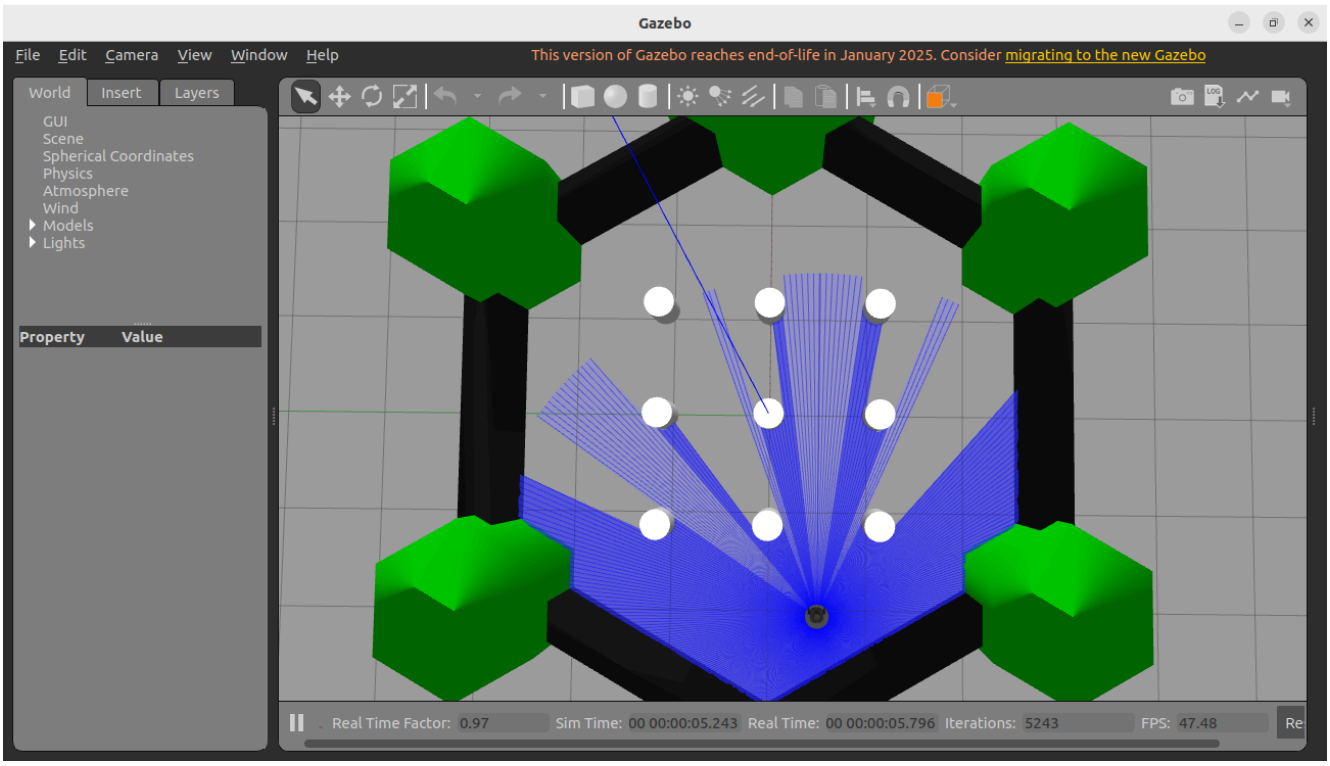


ros2 run rviz2 rviz2



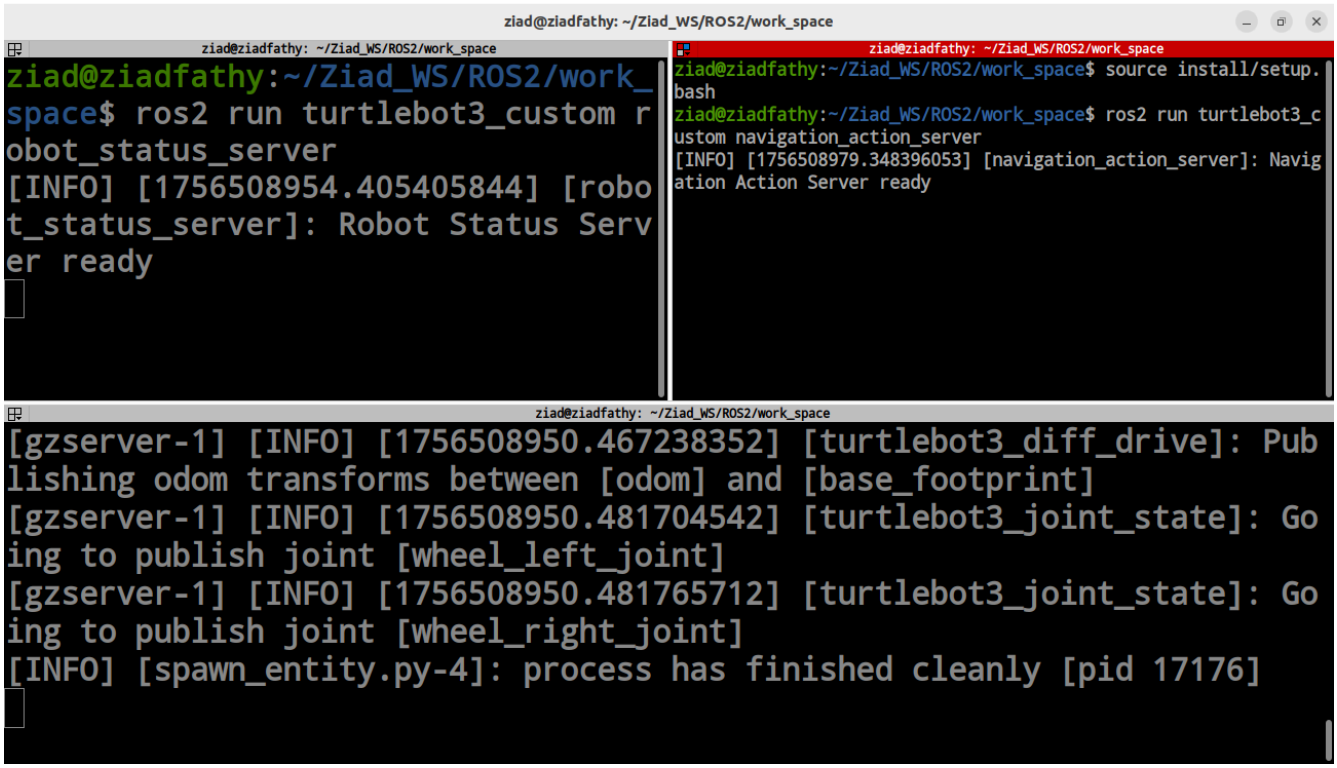
Method 1C: Individual Node Testing

ros2 launch turtlebot3_gazebo turtlebot3_world.launch.py



ros2 run turtlebot3_custom robot_status_server

ros2 run turtlebot3_custom navigation_action_server

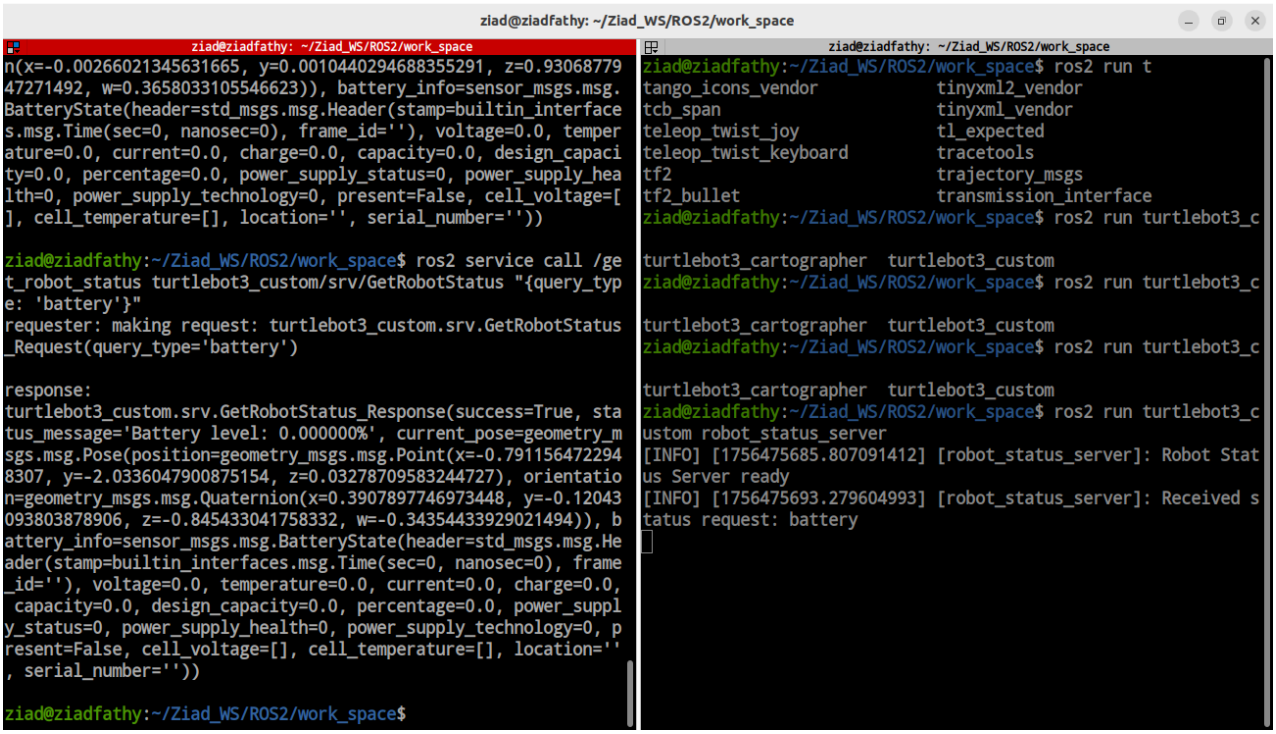


Testing the Functionality

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'}"

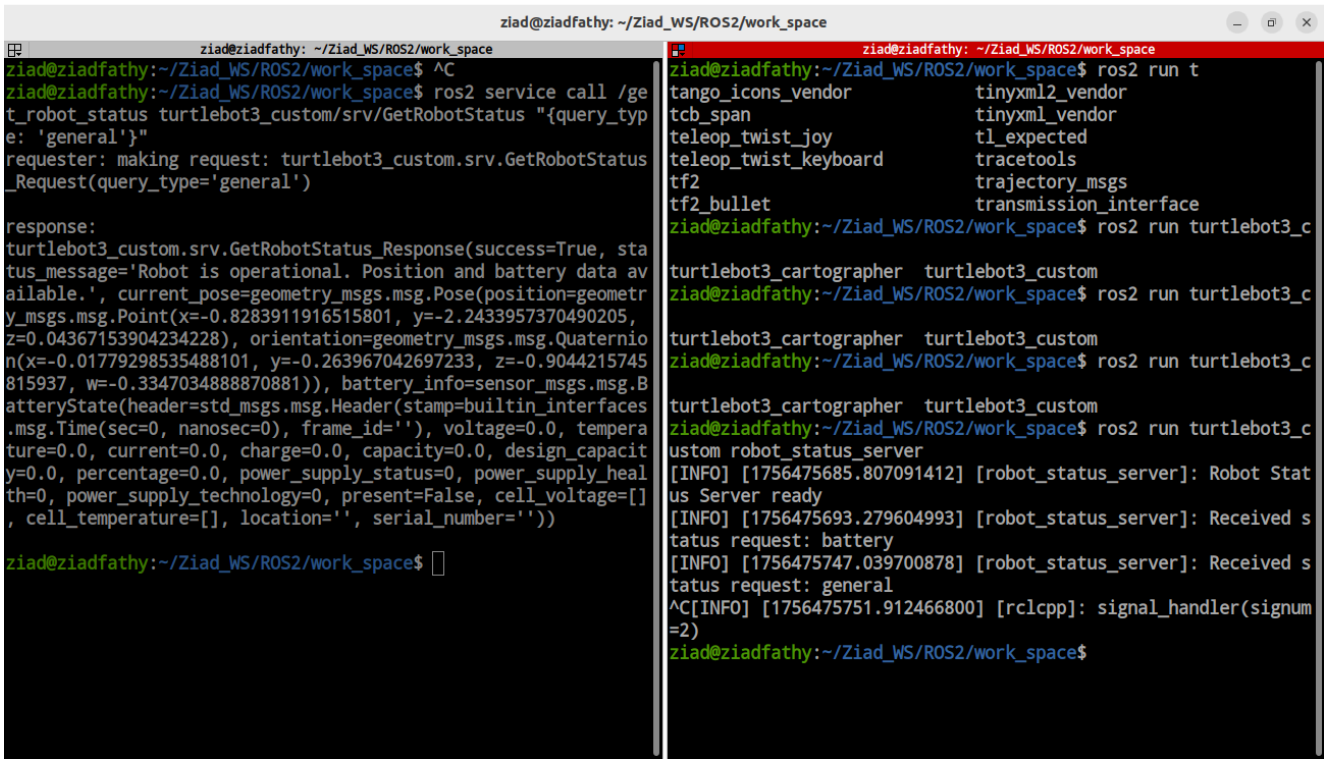


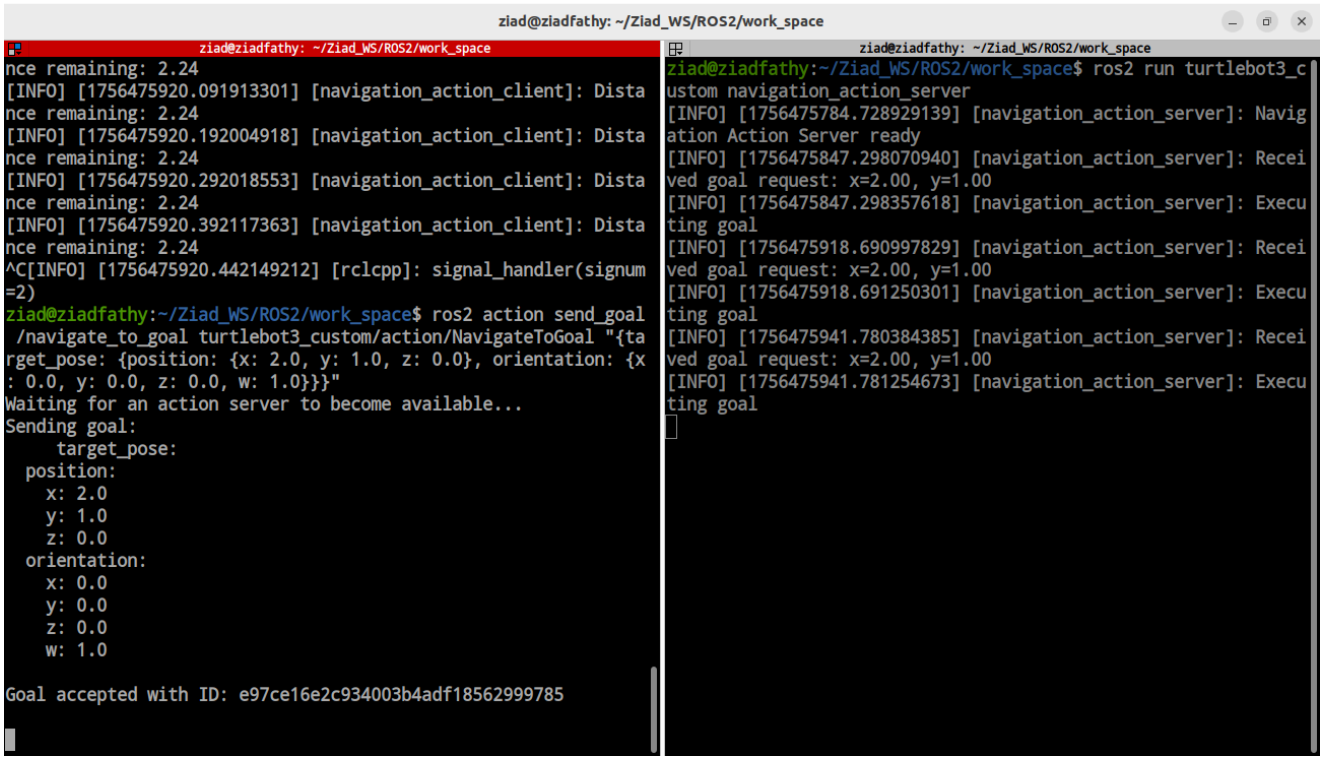
2. Test the Actions

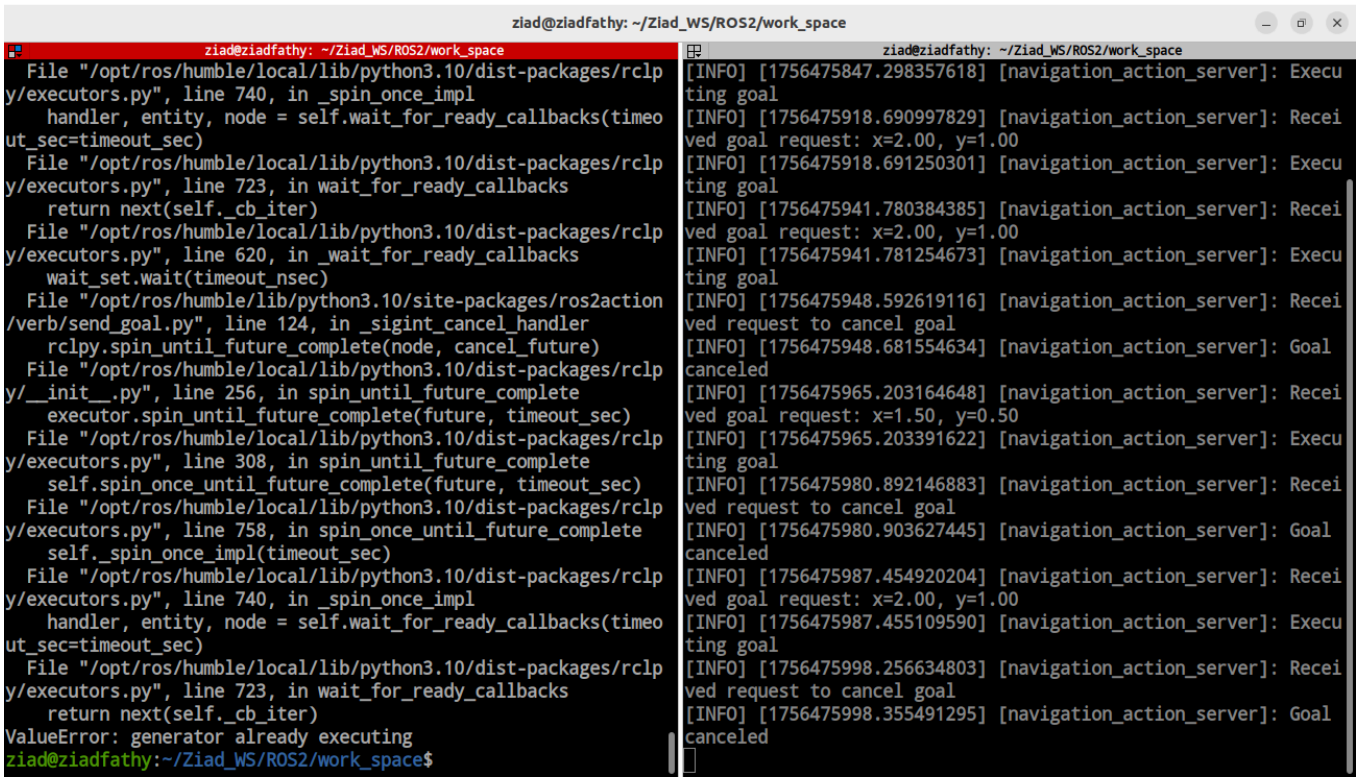
ros2 run turtlebot3_custom navigation_client

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}}}"

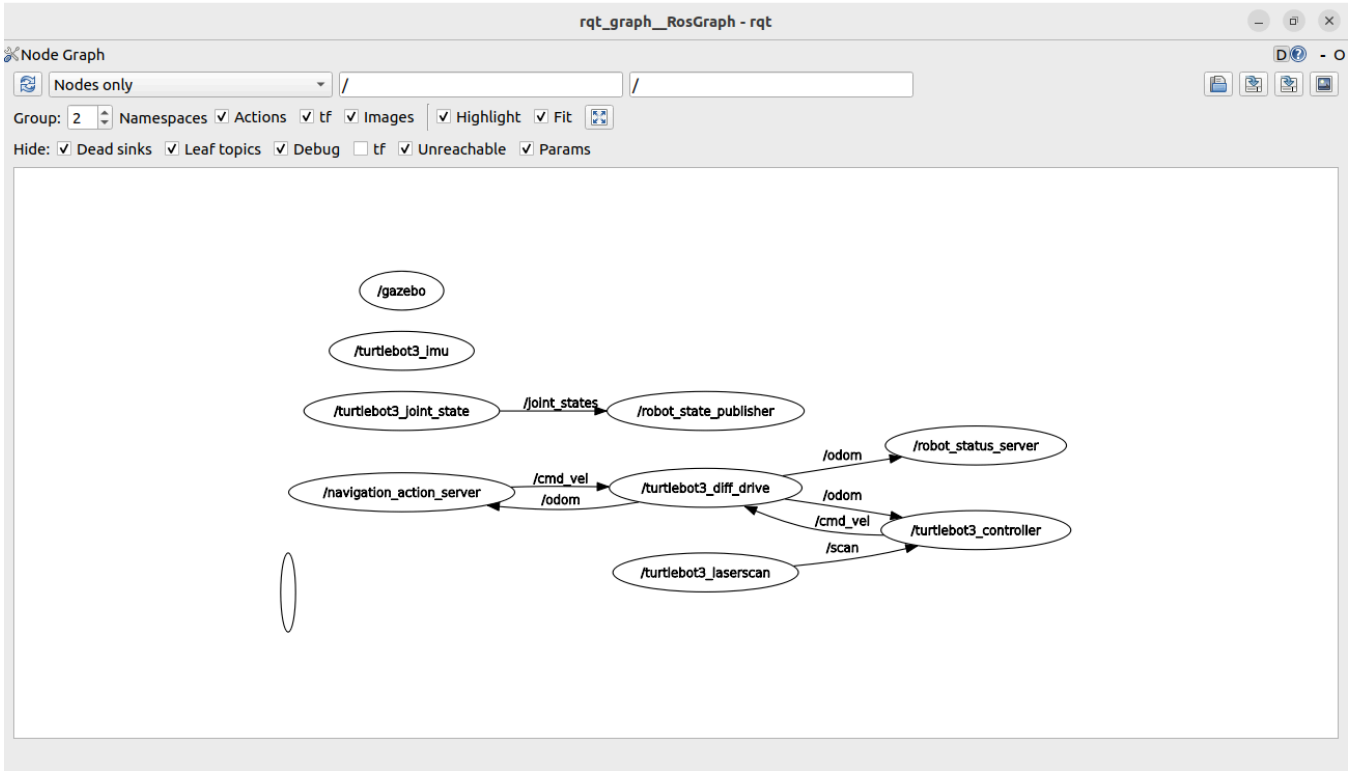
ros2 action send_goal --feedback /navigate_to_goal turtlebot3_custom/action/NavigateToGoal "{target_pose: {position: {x: 1.5, y: 0.5, z: 0.0}, orientation: {x: 0.0, y: 0.0, z: 0.0, w: 1.0}}}"


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

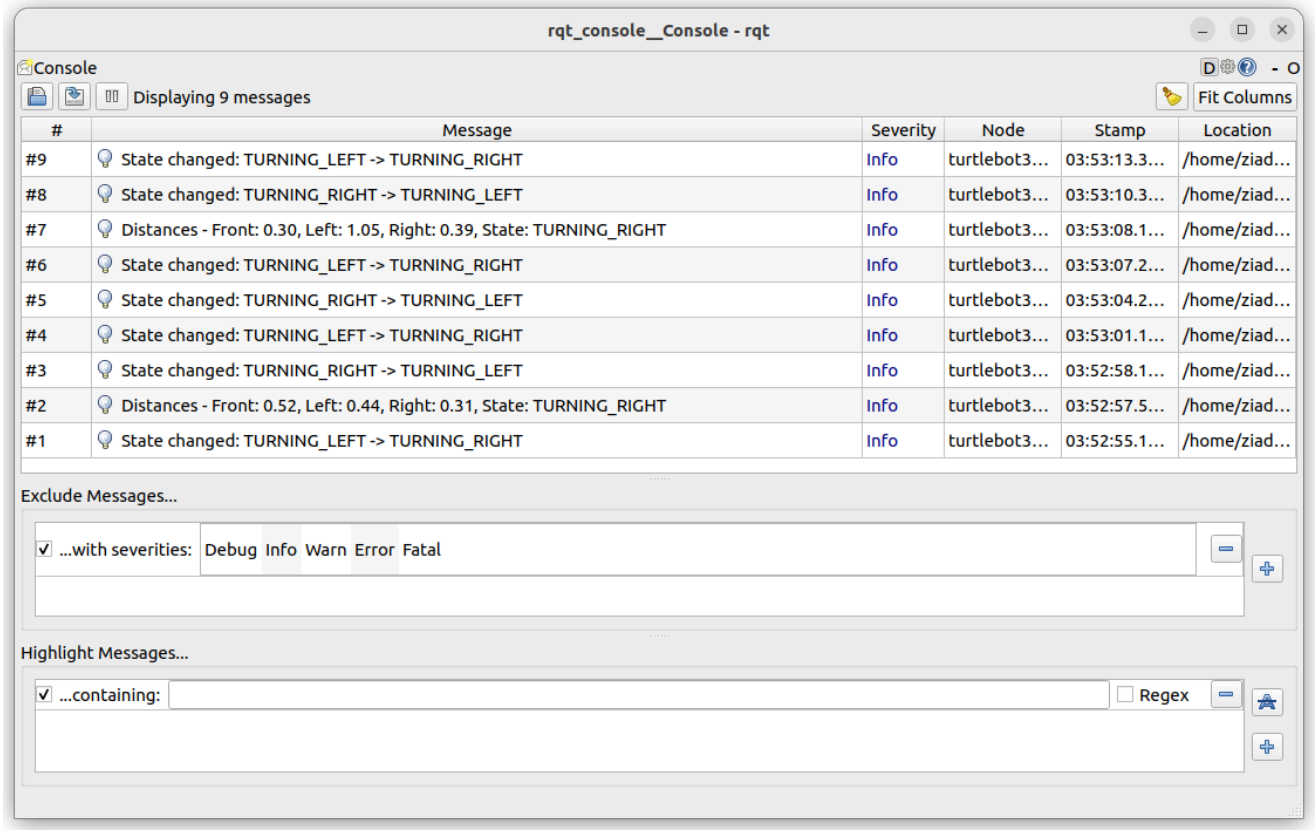

ziad@ziadfathy: ~/Ziad_WS/ROS2/work_space
ziad@ziadfathy: ~/Ziad_WS/ROS2/work_space
nce remaining: 2.24
[INFO] [1756475920.091913301] [navigation_action_client]: Distance remaining: 2.24
[INFO] [1756475920.192004918] [navigation_action_client]: Distance remaining: 2.24
[INFO] [1756475920.292018553] [navigation_action_client]: Distance remaining: 2.24
[INFO] [1756475920.392117363] [navigation_action_client]: Distance remaining: 2.24
^C[INFO] [1756475920.442149212] [rclcpp]: signal_handler(signum=2)
ziad@ziadfathy:~/Ziad_WS/ROS2/work_space$ 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}}}"
Waiting for an action server to become available...
Sending goal:
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
Goal accepted with ID: e97ce16e2c934003b4adf18562999785
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



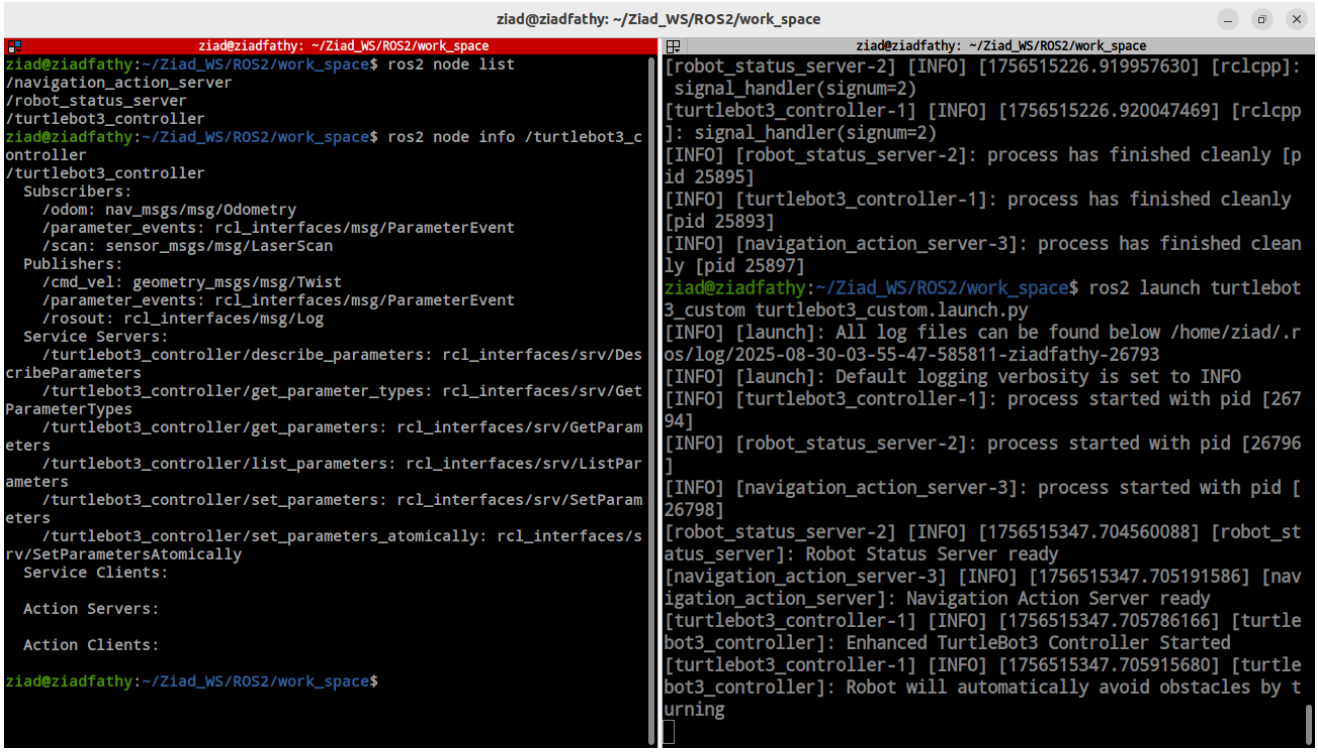
ros2 run rqt_graph rqt_graph



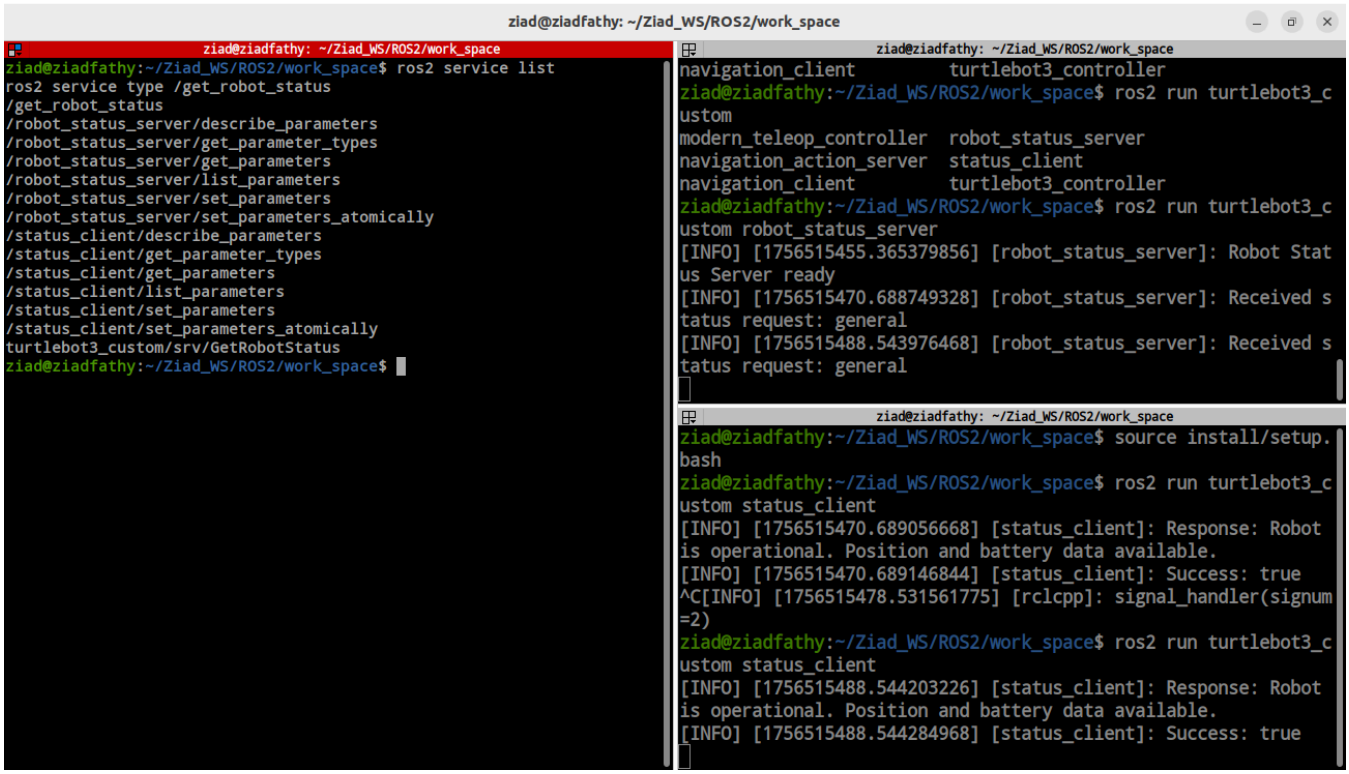
ros2 run rqt_console rqt_console



ros2 node list
ros2 node info /turtlebot3_controller



ros2 service list
ros2 service type /get_robot_status



ros2 action list
ros2 action info /navigate_to_goal