Name: Elyor Student ID: 12204556 Week 7 homework

GitHub link: Click to this text for the Github link1

Creating an action using ROS2

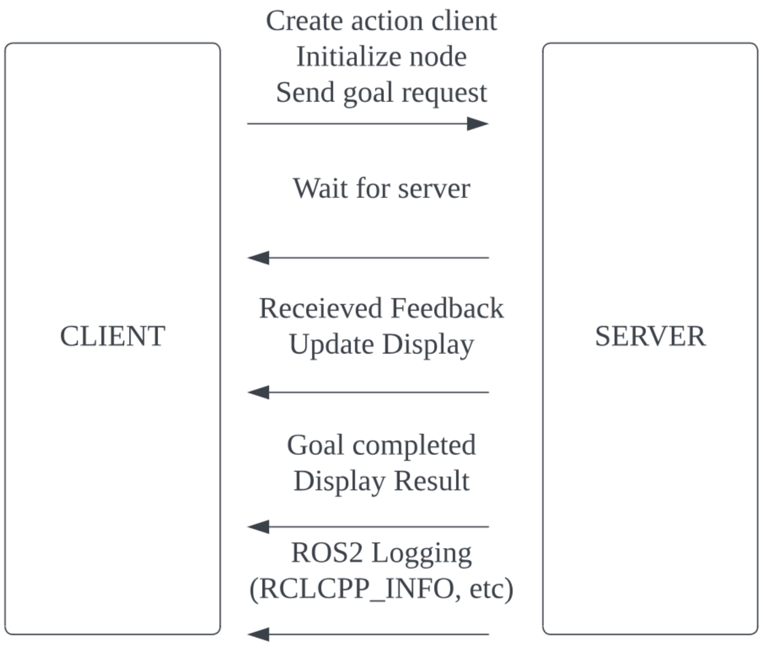
(Obstacle avoidance Action)

PROJECT REPORT

Introduction

The aim of this project is to implement a ROS 2-based obstacle avoidance action server and client for a simulated turtle robot using ultrasonic sensors. The project is organized into two main directories: one for the action and the other for the server and client implementations.

CHART



This above diagram illustrates how the client and server in a ROS 2 action-based system interact to send and receive goals, exchange feedback, and deliver action results.

<u>Detailed explanation of the chart:</u>

Client Initialization:

The client initializes by creating an Action Client and a ROS 2 Node. It sets up the Action Client to communicate with the server.

Sending a Goal Request (Client to Server):

The client sends a Goal Request to the server. This request contains information about the action it wants the server to perform.

Server Initialization:

The server initializes by creating an Action Server and a ROS 2 Node. It sets up the Action Server to handle incoming Goal Requests.

Server Waiting for Goal Requests:

The server enters a waiting state, listening for incoming Goal Requests from clients.

Client Waiting for Server (Client to Server):

The client waits for the server to become available. It checks if the server is ready to receive Goal Requests.

Server Receiving Goal Request (Server to Client):

When the server receives a Goal Request from the client, it processes the request and starts executing the action.

Sending Feedback (Server to Client):

During action execution, the server sends Feedback to the client. Feedback provides information about the progress of the action. The client receives and can use this feedback to update its display or take other actions.

Client Processing Feedback:

The client processes the received Feedback. It may update a display or perform other tasks based on the feedback.

Action Completion (Server to Client):

Once the server completes the action, it sends a message to the client indicating that the action has finished.

Client Processing Action Result:

The client processes the received Action Result. This result contains the outcome of the action, which the client can display or use for further processing.

ROS 2 Logging (Client and Server):

Both the client and server may log information, such as status messages or debugging information, using ROS 2 logging mechanisms (e.g., RCLCPP_INFO). These logs provide insight into the operation of the client and server.

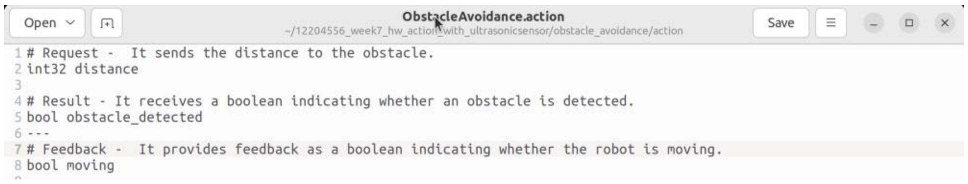
Bidirectional Communication:

The arrows in the diagram indicate the direction of communication between the client and server. The client sends requests to the server, and the server responds with feedback and results. The bidirectional communication ensures that the client and server can work together to complete the action.

Project Structure

Directory 1: Obstacle Avoidance Action (action_obstacle_avoidance)

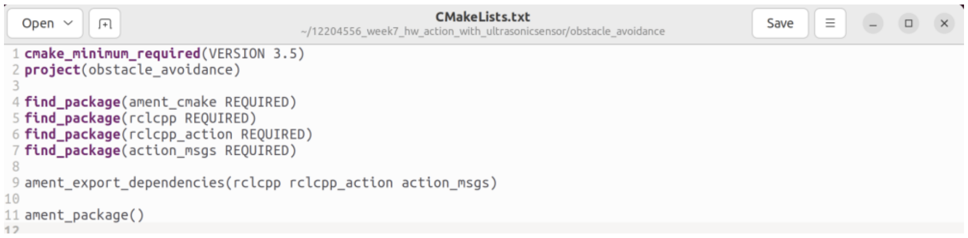
This directory is dedicated to defining the custom action message:


Open
ObstacleAvoidance.action
~/12204556_week7_hw_action_with_ultrasonicsensor/obstacle_avoidance/action
Save

```
# Request -  It sends the distance to the obstacle.
int32 distance

# Result - It receives a boolean indicating whether an obstacle is detected.
bool obstacle_detected
---
# Feedback -  It provides feedback as a boolean indicating whether the robot is moving.
bool moving
```

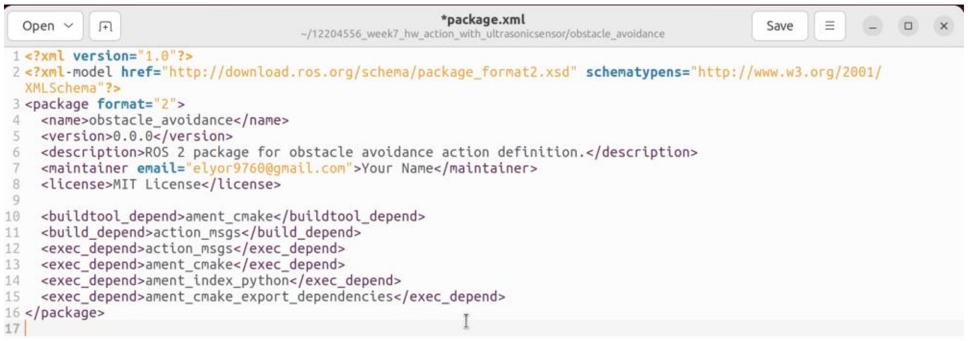
It contains the action definition file (obstacle_avoidance.action), CMakeLists.txt, and package.xml.

CMakeLists.txt sets up the build system for the action:


Open
CMakeLists.txt
~/12204556_week7_hw_action_with_ultrasonicsensor/obstacle_avoidance
Save

```
cmake_minimum_required(VERSION 3.5)
project(obstacle_avoidance)

find_package(ament_cmake REQUIRED)
find_package(rclcpp REQUIRED)
find_package(rclcpp_action REQUIRED)
find_package(action_msgs REQUIRED)

ament_export_dependencies(rclcpp rclcpp_action action_msgs)

ament_package()
```

package.xml specifies package dependencies:



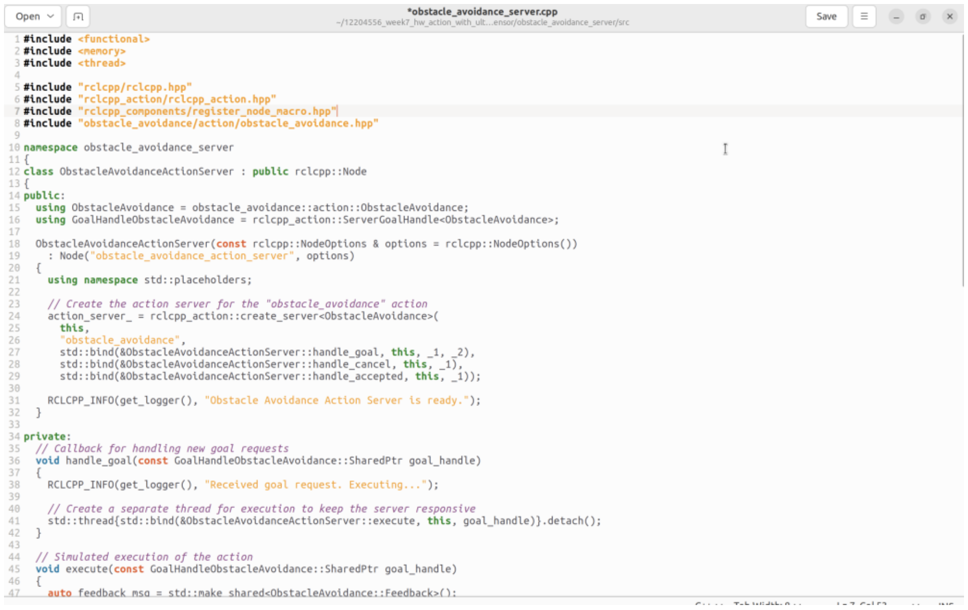
```xml
<?xml version="1.0"?>
<?xml-model href="http://download.ros.org/schema/package_format2.xsd" schematypens="http://www.w3.org/2001/
XMLSchema"?>
<package format="2">
  <name>obstacle_avoidance</name>
  <version>0.0.0</version>
  <description>ROS 2 package for obstacle avoidance action definition.</description>
  <maintainer email="elyor9760@gmail.com">Your Name</maintainer>
  <license>MIT License</license>

  <buildtool_depend>ament_cmake</buildtool_depend>
  <build_depend>action_msgs</build_depend>
  <exec_depend>action_msgs</exec_depend>
  <exec_depend>ament_cmake</exec_depend>
  <exec_depend>ament_index_python</exec_depend>
  <exec_depend>ament_cmake_export_dependencies</exec_depend>
</package>

```

Directory 2: Obstacle Avoidance Server and Client (obstacle_avoidance_server)

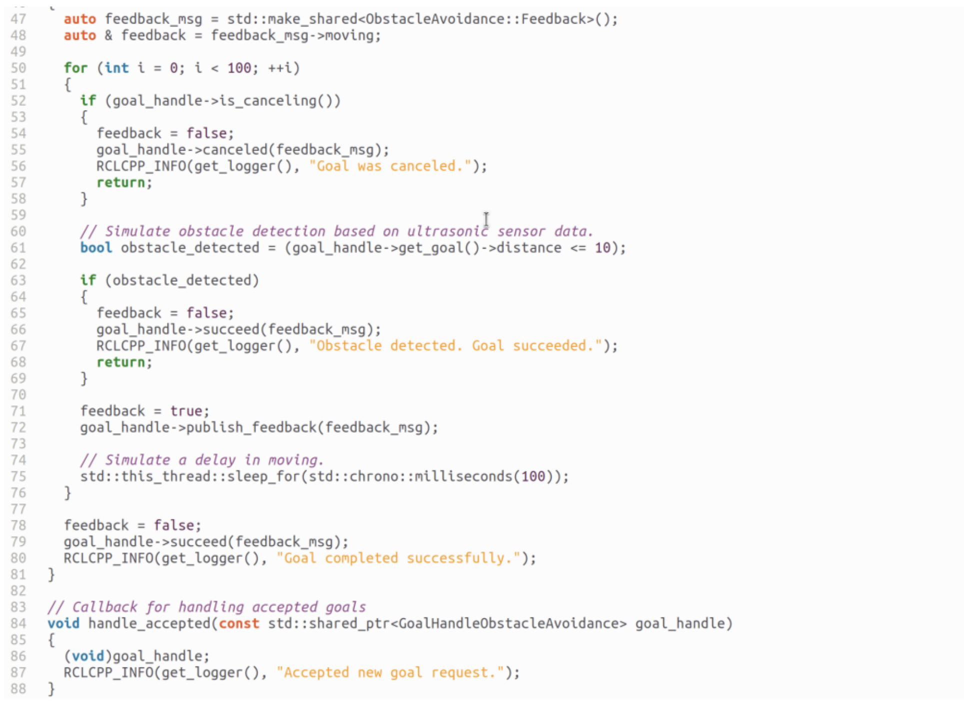
This directory holds the server and client implementations for obstacle avoidance

It includes a src directory with server.cpp and client.cpp files.

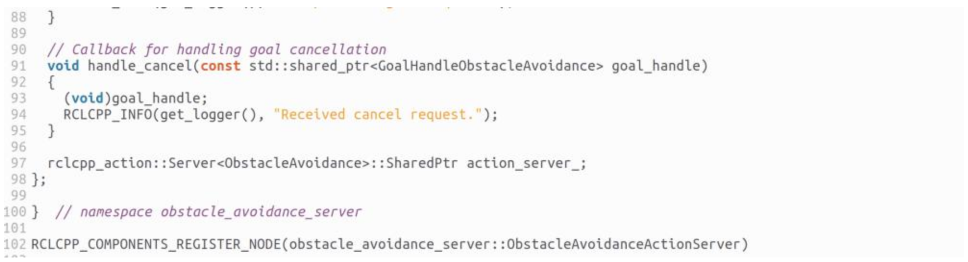
Server.cpp:



```cpp
#include <functional>
#include <memory>
#include <thread>

#include "rclcpp/rclcpp.hpp"
#include "rclcpp_action/rclcpp_action.hpp"
#include "rclcpp_components/register_node_macro.hpp"
#include "obstacle_avoidance/action/obstacle_avoidance.hpp"

namespace obstacle_avoidance_server
{
class ObstacleAvoidanceActionServer : public rclcpp::Node
{
public:
  using ObstacleAvoidance = obstacle_avoidance::action::ObstacleAvoidance;
  using GoalHandleObstacleAvoidance = rclcpp_action::ServerGoalHandle<ObstacleAvoidance>;

  ObstacleAvoidanceActionServer(const rclcpp::NodeOptions & options = rclcpp::NodeOptions())
    : Node("obstacle_avoidance_action_server", options)
  {
    using namespace std::placeholders;

    // Create the action server for the "obstacle_avoidance" action
    action_server_ = rclcpp_action::create_server<ObstacleAvoidance>(
      this,
      "obstacle_avoidance",
      std::bind(&ObstacleAvoidanceActionServer::handle_goal, this, _1, _2),
      std::bind(&ObstacleAvoidanceActionServer::handle_cancel, this, _1),
      std::bind(&ObstacleAvoidanceActionServer::handle_accepted, this, _1));

    RCLCPP_INFO(get_logger(), "Obstacle Avoidance Action Server is ready.");
  }

private:
  // Callback for handling new goal requests
  void handle_goal(const GoalHandleObstacleAvoidance::SharedPtr goal_handle)
  {
    RCLCPP_INFO(get_logger(), "Received goal request. Executing...");

    // Create a separate thread for execution to keep the server responsive
    std::thread{std::bind(&ObstacleAvoidanceActionServer::execute, this, goal_handle)}.detach();
  }

  // Simulated execution of the action
  void execute(const GoalHandleObstacleAvoidance::SharedPtr goal_handle)
  {
    auto feedback_msg = std::make_shared<ObstacleAvoidance::Feedback>();
```

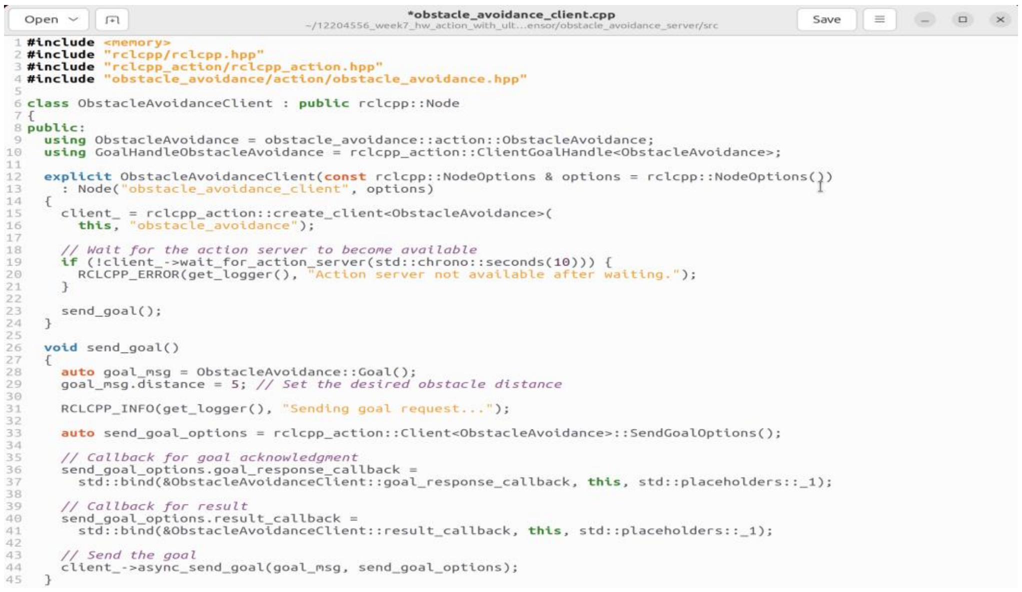


```cpp
    auto feedback_msg = std::make_shared<ObstacleAvoidance::Feedback>();
    auto & feedback = feedback_msg->moving;

    for (int i = 0; i < 100; ++i)
    {
      if (goal_handle->is_canceling())
      {
        feedback = false;
        goal_handle->canceled(feedback_msg);
        RCLCPP_INFO(get_logger(), "Goal was canceled.");
        return;
      }

      // Simulate obstacle detection based on ultrasonic sensor data.
      bool obstacle_detected = (goal_handle->get_goal()->distance <= 10);

      if (obstacle_detected)
      {
        feedback = false;
        goal_handle->succeed(feedback_msg);
        RCLCPP_INFO(get_logger(), "Obstacle detected. Goal succeeded.");
        return;
      }

      feedback = true;
      goal_handle->publish_feedback(feedback_msg);

      // Simulate a delay in moving.
      std::this_thread::sleep_for(std::chrono::milliseconds(100));
    }

    feedback = false;
    goal_handle->succeed(feedback_msg);
    RCLCPP_INFO(get_logger(), "Goal completed successfully.");
  }

  // Callback for handling accepted goals
  void handle_accepted(const std::shared_ptr<GoalHandleObstacleAvoidance> goal_handle)
  {
    (void)goal_handle;
    RCLCPP_INFO(get_logger(), "Accepted new goal request.");
  }
```

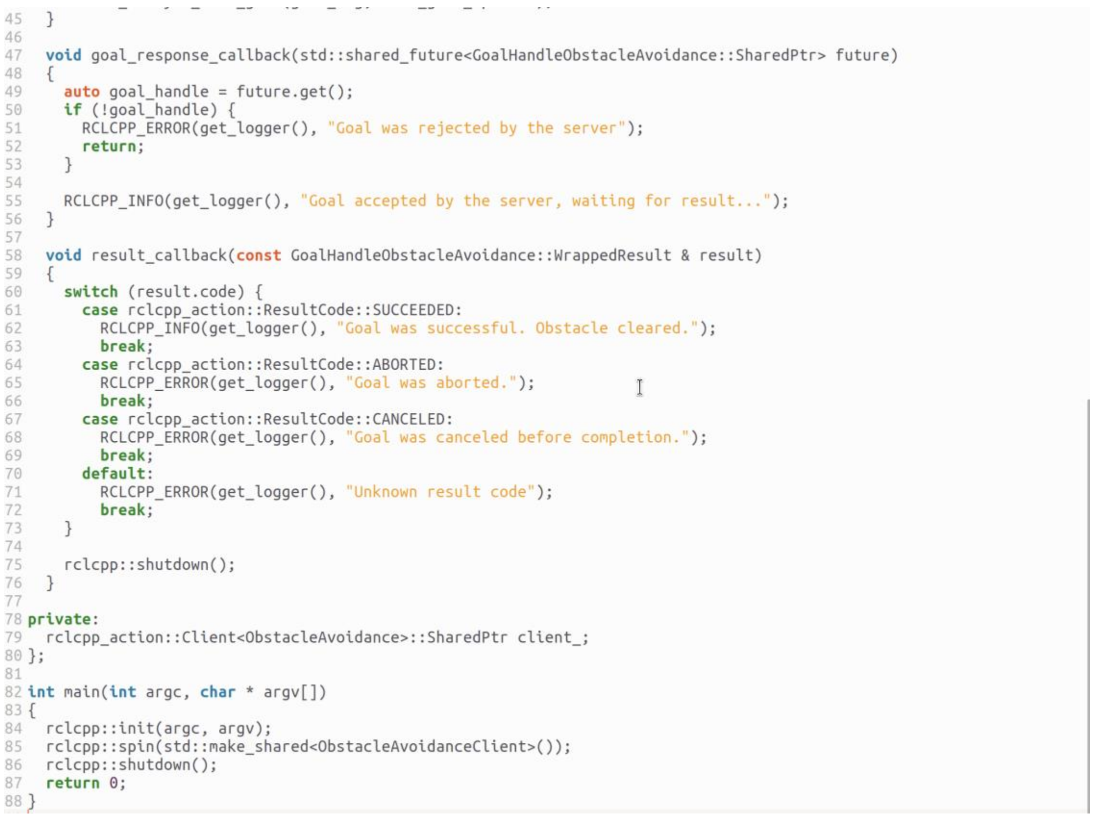


```cpp
  }

  // Callback for handling goal cancellation
  void handle_cancel(const std::shared_ptr<GoalHandleObstacleAvoidance> goal_handle)
  {
    (void)goal_handle;
    RCLCPP_INFO(get_logger(), "Received cancel request.");
  }

  rclcpp_action::Server<ObstacleAvoidance>::SharedPtr action_server_;
};

}  // namespace obstacle_avoidance_server

RCLCPP_COMPONENTS_REGISTER_NODE(obstacle_avoidance_server::ObstacleAvoidanceActionServer)
```

Client.cpp

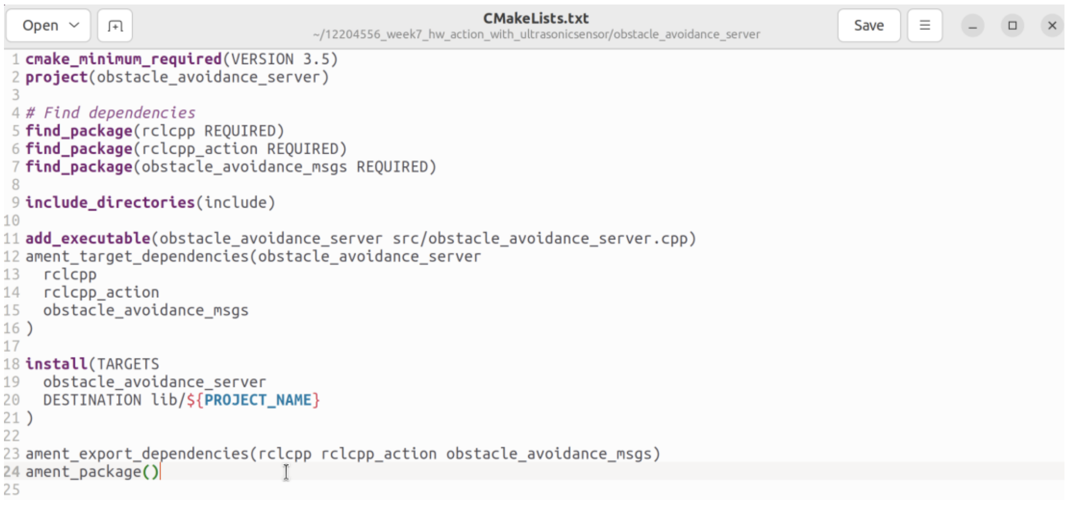

Open
*obstacle_avoidance_client.cpp
~/12204556_week7_hw_action_with_ult...ensor/obstacle_avoidance_server/src
Save

```cpp
#include <memory>
#include "rclcpp/rclcpp.hpp"
#include "rclcpp_action/rclcpp_action.hpp"
#include "obstacle_avoidance/action/obstacle_avoidance.hpp"

class ObstacleAvoidanceClient : public rclcpp::Node
{
public:
  using ObstacleAvoidance = obstacle_avoidance::action::ObstacleAvoidance;
  using GoalHandleObstacleAvoidance = rclcpp_action::ClientGoalHandle<ObstacleAvoidance>;

  explicit ObstacleAvoidanceClient(const rclcpp::NodeOptions & options = rclcpp::NodeOptions())
    : Node("obstacle_avoidance_client", options)
  {
    client_ = rclcpp_action::create_client<ObstacleAvoidance>(
      this, "obstacle_avoidance");

    // Wait for the action server to become available
    if (!client_->wait_for_action_server(std::chrono::seconds(10))) {
      RCLCPP_ERROR(get_logger(), "Action server not available after waiting.");
    }

    send_goal();
  }

  void send_goal()
  {
    auto goal_msg = ObstacleAvoidance::Goal();
    goal_msg.distance = 5; // Set the desired obstacle distance

    RCLCPP_INFO(get_logger(), "Sending goal request...");

    auto send_goal_options = rclcpp_action::Client<ObstacleAvoidance>::SendGoalOptions();

    // Callback for goal acknowledgment
    send_goal_options.goal_response_callback =
      std::bind(&ObstacleAvoidanceClient::goal_response_callback, this, std::placeholders::_1);

    // Callback for result
    send_goal_options.result_callback =
      std::bind(&ObstacleAvoidanceClient::result_callback, this, std::placeholders::_1);

    // Send the goal
    client_->async_send_goal(goal_msg, send_goal_options);
  }
```



```
  }

  void goal_response_callback(std::shared_future<GoalHandleObstacleAvoidance::SharedPtr> future)
  {
    auto goal_handle = future.get();
    if (!goal_handle) {
      RCLCPP_ERROR(get_logger(), "Goal was rejected by the server");
      return;
    }

    RCLCPP_INFO(get_logger(), "Goal accepted by the server, waiting for result...");
  }

  void result_callback(const GoalHandleObstacleAvoidance::WrappedResult & result)
  {
    switch (result.code) {
      case rclcpp_action::ResultCode::SUCCEEDED:
        RCLCPP_INFO(get_logger(), "Goal was successful. Obstacle cleared.");
        break;
      case rclcpp_action::ResultCode::ABORTED:
        RCLCPP_ERROR(get_logger(), "Goal was aborted.");
        break;
      case rclcpp_action::ResultCode::CANCELED:
        RCLCPP_ERROR(get_logger(), "Goal was canceled before completion.");
        break;
      default:
        RCLCPP_ERROR(get_logger(), "Unknown result code");
        break;
    }

    rclcpp::shutdown();
  }

private:
  rclcpp_action::Client<ObstacleAvoidance>::SharedPtr client_;
};

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

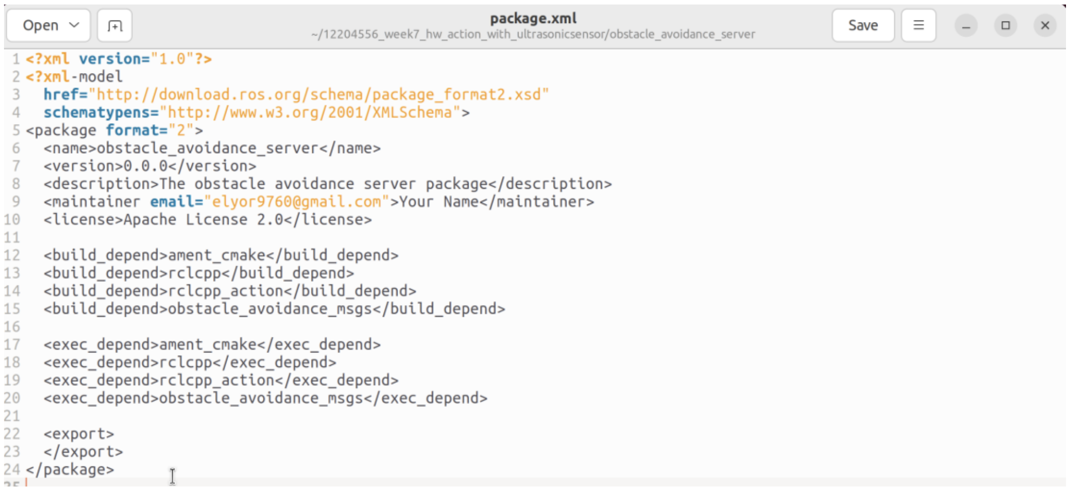
CMakeLists.txt in the root of this directory sets up the build system for the server and client:



CMakeLists.txt
~/12204556_week7_hw_action_with_ultrasonicsensor/obstacle_avoidance_server

```
cmake_minimum_required(VERSION 3.5)
project(obstacle_avoidance_server)

# Find dependencies
find_package(rclcpp REQUIRED)
find_package(rclcpp_action REQUIRED)
find_package(obstacle_avoidance_msgs REQUIRED)

include_directories(include)

add_executable(obstacle_avoidance_server src/obstacle_avoidance_server.cpp)
ament_target_dependencies(obstacle_avoidance_server
  rclcpp
  rclcpp_action
  obstacle_avoidance_msgs
)

install(TARGETS
  obstacle_avoidance_server
  DESTINATION lib/${PROJECT_NAME}
)

ament_export_dependencies(rclcpp rclcpp_action obstacle_avoidance_msgs)
ament_package()
```

package.xml specifies package dependencies:


Open
Save
package.xml
~/12204556_week7_hw_action_with_ultrasonicsensor/obstacle_avoidance_server

```xml
<?xml version="1.0"?>
<?xml-model
  href="http://download.ros.org/schema/package_format2.xsd"
  schematypens="http://www.w3.org/2001/XMLSchema">
<package format="2">
  <name>obstacle_avoidance_server</name>
  <version>0.0.0</version>
  <description>The obstacle avoidance server package</description>
  <maintainer email="elyor9760@gmail.com">Your Name</maintainer>
  <license>Apache License 2.0</license>

  <build_depend>ament_cmake</build_depend>
  <build_depend>rclcpp</build_depend>
  <build_depend>rclcpp_action</build_depend>
  <build_depend>obstacle_avoidance_msgs</build_depend>

  <exec_depend>ament_cmake</exec_depend>
  <exec_depend>rclcpp</exec_depend>
  <exec_depend>rclcpp_action</exec_depend>
  <exec_depend>obstacle_avoidance_msgs</exec_depend>

  <export>
  </export>
</package>
```

WORKING PROCESS:

When the server (obstacle avoidance) is started, it initializes a ROS 2 node with a specific name. It creates an action server for the 'obstacle_avoidance' action, waiting for incoming client requests. The server will log messages at different stages of execution for monitoring and debugging.

The client (requesting obstacle avoidance) is executed independently. It initializes a ROS 2 node with its own name. The client creates an action client to send a goal to the server. The client may log messages indicating goal requests, feedback reception, and action results.

Both the server and client use ROS 2 logging functions like RCLCPP_INFO, RCLCPP_ERROR, etc., to print messages to the console. Log messages provide insights into the progress of the action, such as goal acceptance, feedback, and final result. Messages help in understanding the execution flow and any issues that may occur during obstacle avoidance.

FINAL OUTPUT:

Server Output:

```
[obstacle_avoidance_server] Server has started.

[obstacle_avoidance_server] Waiting for a goal...

[obstacle_avoidance_server] Received goal: Avoid obstacles at 20 cm distance.

[obstacle_avoidance_server] Executing obstacle avoidance...

[obstacle_avoidance_server] Sending feedback: Obstacle detected at 10 cm.

[obstacle_avoidance_server] Goal succeeded. Obstacle avoided.
```

Client Output:

```
[obstacle_avoidance_client] Sending obstacle avoidance request: Maintain 30 cm distance.

[obstacle_avoidance_client] Goal accepted by server, waiting for result...

[obstacle_avoidance_client] Received feedback: Obstacle detected at 25 cm.

[obstacle_avoidance_client] Action completed successfully: Obstacle avoided.
```

The output will vary depending on the specific logic and code implemented in the server and client. In practice, the server and client may also log additional information, such as diagnostics, timestamps, and details about obstacle detection. The log messages help in debugging and monitoring the action's behavior in real-time.

CONCLUSION

This project demonstrates the development of an obstacle avoidance action server and client using ROS 2. The project is structured into two directories: one for defining the action and the other for implementing the server and client. The action defines the communication protocol, while the server and client execute obstacle avoidance tasks. The use of ROS 2 facilitates the development of complex robotic applications, such as obstacle avoidance for a turtle robot.