Management Center Innsbruck

Department of Technology & Life Sciences

Master's program Mechatronics & Smart Technologies

Software report

composed as part of the course
Mobile Robotics (MECH-M-2-ROB)

about

TurtlesimAutomata - dipping the toe into ROS

from

Jakob Spindler

Study program	Master's program Mechatronics & Smart Technologies
Year	MA-MECH-23-VZ
Course	Mobile Robotics (MECH-M-2-ROB)
Name of lecturer	Daniel McGuiness
Submission deadline	June 16, 2024

May 31, 2024

Contents

Chapter 1

Introduction

The turtlesimAutomata package is a ROS2 (Robot Operating System) package that uses a finite state machine-like structure to control a turtle in the turtlesim environment. It's purpose is to provide an entry-level example of a ROS2 package to acquaint the user with the basic concepts of ROS2. The turtlesimAutomata package is closely intertwined with the turtlesim package, which is a simple simulator for a mobile robot in the shape of a turtle.

1.1 Assignement

The assignement stated the following requirements for the turtlesimAutomata package:

- The turtlesimAutomata package should work closely together with the turtlesim package.
- The turtle shall start off in a random direction
- The turtle shall move in a straight line until it reaches the edge of the turtlesim window, at which point it shall make a 90 degree turn in clockwise direction and continue moving in the new direction.

Chapter 2

Package overview

The `turtlesimAutomata` package consists of the following nodes:

- `/start_turtle` listens to the */rosout* topic for a message containing "Spawning turtle" which indicates that the `turtlesim_node` node has started successfully. It will then send a randomly computed angle to the `/turn_turtle` node via the */turn_angle* topic and shut itself down afterwards.
- `/drive_turtle_continuously` continuously sends a message to the */turtle1/cmd_vel* topic to drive the turtle in a straight line. The sending of said message may be toggled via the */drive_turtle* topic.
- `/turn_turtle` listens to the */turn_angle* topic for a message containing the angle by which the turtle shall turn. The node includes a client for the */turtle1/teleport_relative* service (not visible in Figure 2.1), which is hosted by the `/turtlesim_node` and is being used to request a turn of the turtle by the received angle. The node functions mostly as a relay station.
- `/edge` listens to the */rosout* topic for a message containing "Oh no! I hit the wall!" which is published by the `turtlesim_node` node when the turtle reaches the edge of the window. It will then tell the `/drive_turtle_continuously` node to stop driving the turtle forwards, send a message via the */turn_angle* topic to request a 90 degree clockwise turn by the `/turn_turtle` node and afterwards tells the `/drive_turtle_continuously` node to continue driving the turtle forwards.

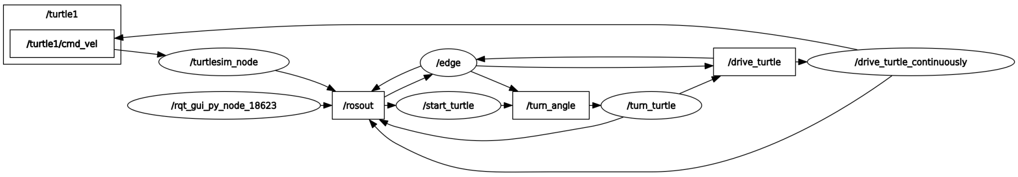
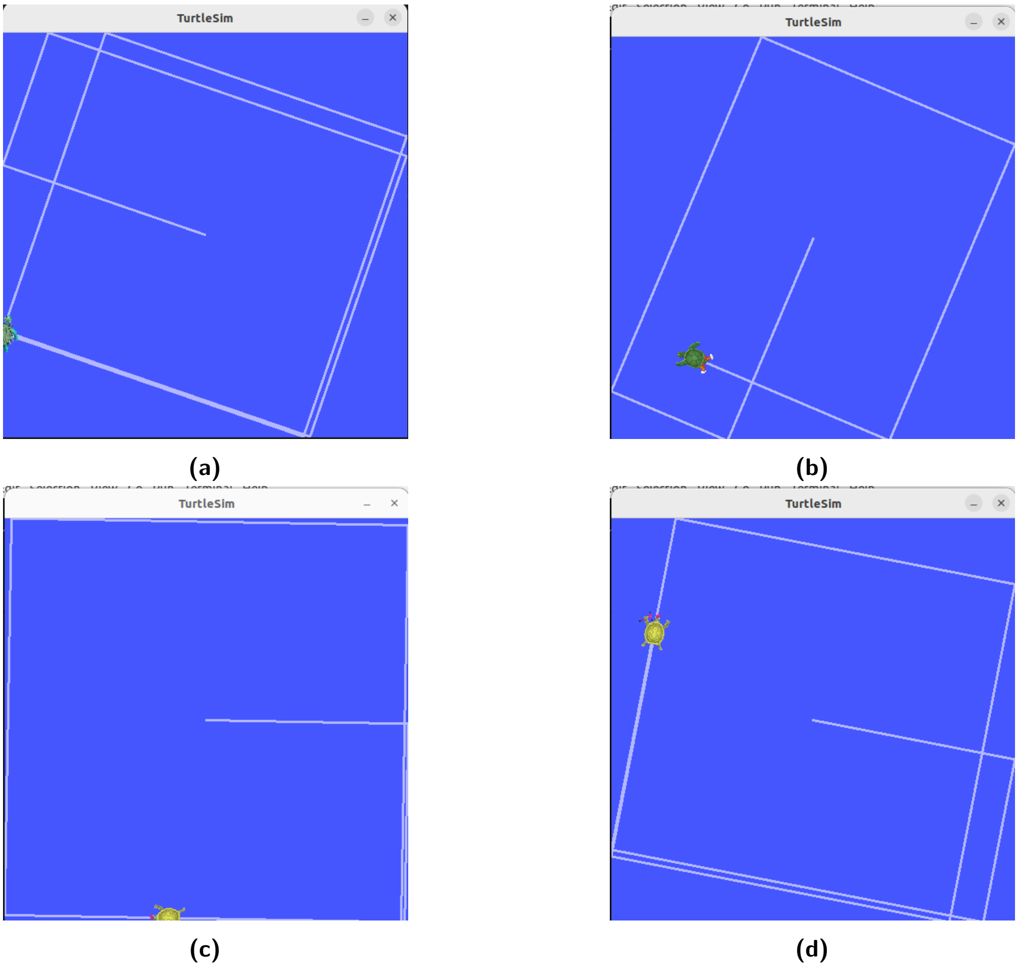


Figure 2.1. Nodes and topics of the `turtlesimAutomata` package shown in `rqt_graph`

Chapter 3

Behaviour of the turtle

The following Figure 3.1 shows examples of the turtle's behaviour, showcasing that the `turtlesimAutomata` package is working as intended.



(a)

(b)

(c)

(d)

Figure 3.1. Examples of the turtle's behaviour

Chapter 4

Additional functionalities

The `turtlesimAutomata` package can be used in combination with the `turtlesim_teleop` package, which allows the user to manually control the turtle in the turtlesim environment, whilst still mainting the automatic behaviour provided by the `turtlesimAutomata` package. This may result in interesting behaviour, as shown in Figure 4.1.

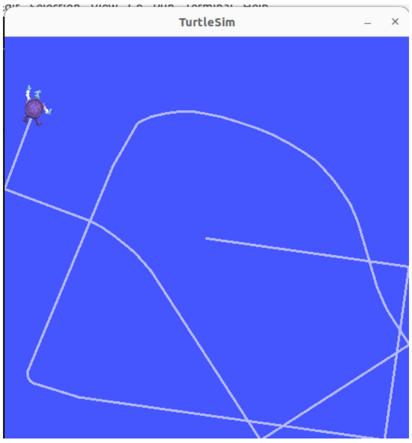


Figure 4.1. `turtlesimAutomata` and `turtlesim_teleop` working together

Figure 4.2 shows the `rqt_graph` of the `turtlesimAutomata` and `turtlesim_teleop` packages working together.

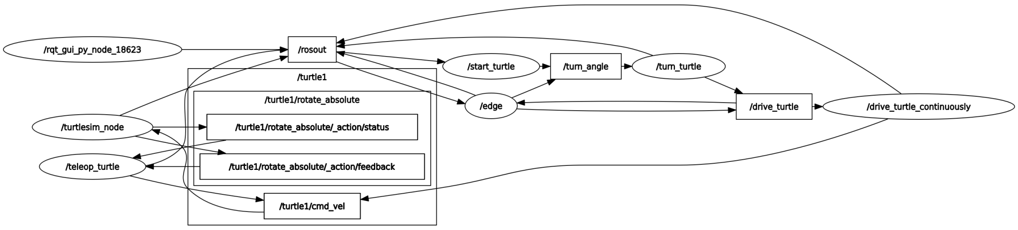


Figure 4.2. turtlesimAutomata and `turtlesim_teleop` working together shown in `rqt_graph`

List of Figures

List of Tables

Appendix A

turtlesimAutomata source code

```
#include "rclcpp/rclcpp.hpp"
#include "rcl_interfaces/msg/log.hpp"
#include "std_msgs/msg/float64.hpp"
#include "std_msgs/msg/bool.hpp"
#include <cstdlib> // Include the C standard library for random number generation

class StartTurtle : public rclcpp::Node
{
public:
    StartTurtle() : Node("start_turtle")
    {
        subscription_ = this->create_subscription<rcl_interfaces::msg::Log>(
            "/rosout", 10, std::bind(&StartTurtle::log_callback, this, std::placeholders::_1));

        angle_publisher_ = this->create_publisher<std_msgs::msg::Float64>("/turn_angle", 1);
        //drive_publisher_ = this->create_publisher<std_msgs::msg::Bool>("/drive_turtle", 1);

        RCLCPP_INFO(this->get_logger(), "Started 'start_turtle' node");
    }

private:
    void log_callback(const rcl_interfaces::msg::Log::SharedPtr msg)
    {
        if (msg->msg.find("Spawning turtle") != std::string::npos)
        {
            RCLCPP_INFO(this->get_logger(), "Detected 'Spawning turtle' message. Proceeding with actions.");
            std::this_thread::sleep_for(std::chrono::milliseconds(1500));
            publish_random_angle();
            //std::this_thread::sleep_for(std::chrono::
```

```
milliseconds(1500));
            //publish_drive_true();
            rclcpp::shutdown();
        }
    }

    void publish_random_angle()
    {
        // Generate a random angle in range of 45 to 90, 135 to
 180, 225 to 270, 315 to 360 degrees
        // this ensures, that the turtle can turn away from the
 wall with one 90 degree clockwise turn,
        // thus keeping the visuals interesting.
        // (all other angles result in the turtle just drawing
a straight line between two walls,
        // as it alsways ahs to perofrm 180 degree rotations)
        /*
        double angle_degrees = rand() % 45;
        int modifier = rand() % 4;
        angle_degrees *=  -1;
        angle_degrees += modifier * 90;
        */

        double angle_degrees = generate_random_angle();

        auto angle_msg = std_msgs::msg::Float64();
        angle_msg.data = angle_degrees;
        angle_publisher_->publish(angle_msg);

        RCLCPP_INFO(this->get_logger(), "Published random angle
: %.2f degrees", angle_msg.data);
    }

    void publish_drive_true()
    {
        auto drive_msg = std_msgs::msg::Bool();
        drive_msg.data = true;
        drive_publisher_->publish(drive_msg);

        RCLCPP_INFO(this->get_logger(), "Published 'true' to /
drive_turtle");
    }

    double generate_random_angle()
    {
        // Seed the random number generator
        srand(time(nullptr));

        // Define the ranges for each quadrant
        int lower_bounds[] = {45, 135, 225, 315};
        int upper_bounds[] = {90, 180, 270, 360};

        // Randomly select a quadrant
        int quadrant = rand() % 4;

        // Generate a random angle within the selected quadrant
        int angle = rand() % (upper_bounds[quadrant] -
lower_bounds[quadrant] + 1) + lower_bounds[quadrant];

            return angle;
        }

        rclcpp::Subscription<rcl_interfaces::msg::Log>::SharedPtr subscription_;
        rclcpp::Publisher<std_msgs::msg::Float64>::SharedPtr angle_publisher_;
        rclcpp::Publisher<std_msgs::msg::Bool>::SharedPtr drive_publisher_;
};

int main(int argc, char *argv[])
{
    rclcpp::init(argc, argv);
    auto node = std::make_shared<StartTurtle>();
    rclcpp::spin(node);
    std::this_thread::sleep_for(std::chrono::seconds(5));
    rclcpp::shutdown();
    return 0;
}
```

Listing A.1. /start_turtle source code

```
#include "rclcpp/rclcpp.hpp"
#include "std_msgs/msg/bool.hpp"
#include "geometry_msgs/msg/twist.hpp"

class DriveTurtleContinuously : public rclcpp::Node
{
public:
    DriveTurtleContinuously() : Node("drive_turtle_continuously"), drive_turtle_(false)
    {
        subscription_ = this->create_subscription<std_msgs::msg::Bool>(
            "/drive_turtle", 10, std::bind(&DriveTurtleContinuously::drive_callback, this, std::placeholders::_1));
        cmd_vel_publisher_ = this->create_publisher<geometry_msgs::msg::Twist>("turtle1/cmd_vel", 10);
        timer_ = this->create_wall_timer(
            std::chrono::milliseconds(100), std::bind(&DriveTurtleContinuously::publish_velocity, this));

        RCLCPP_INFO(this->get_logger(), "Started ’drive_turtle_continuously’ node");
    }

private:
    void drive_callback(const std_msgs::msg::Bool::SharedPtr msg)
    {
        drive_turtle_ = msg->data;
        if (!drive_turtle_) {
            // Immediately stop the turtle
            auto stop_msg = geometry_msgs::msg::Twist();
            cmd_vel_publisher_->publish(stop_msg);
            RCLCPP_INFO(this->get_logger(), "Received stop command. Stopping turtle.");
        } else {
            RCLCPP_INFO(this->get_logger(), "Received drive command. Starting turtle.");
        }
    }

    void publish_velocity()
    {
        if (drive_turtle_)
        {
            auto twist_msg = geometry_msgs::msg::Twist();
            twist_msg.linear.x = 3.0;  // Set linear x velocity to 1.0
            cmd_vel_publisher_->publish(twist_msg);
            //RCLCPP_INFO(this->get_logger(), "Publishing cmd_vel to move turtle.");
        }
    }

    rclcpp::Subscription<std_msgs::msg::Bool>::SharedPtr subscription_;
    rclcpp::Publisher<geometry_msgs::msg::Twist>::SharedPtr cmd_vel_publisher_;
    rclcpp::TimerBase::SharedPtr timer_;
    bool drive_turtle_;
};

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

Listing A.2. /drive_turtle_continuously source code

```
#include "rclcpp/rclcpp.hpp"
#include "std_msgs/msg/float64.hpp"
#include "std_msgs/msg/bool.hpp"
#include "turtlesim/srv/teleport_relative.hpp"
#include "geometry_msgs/msg/twist.hpp"

class TurnTurtle : public rclcpp::Node
{
public:
    TurnTurtle() : Node("turn_turtle")
    {
        subscription_ = this->create_subscription<std_msgs::msg::Float64>(
            "/turn_angle", 1, std::bind(&TurnTurtle::angle_callback, this, std::placeholders::_1));

        client_ = this->create_client<turtlesim::srv::TeleportRelative>("turtle1/teleport_relative");
        //cmd_vel_publisher_ = this->create_publisher<geometry_msgs::msg::Twist>("turtle1/cmd_vel", 1);
        drive_turtle_publisher_ = this->create_publisher<std_msgs::msg::Bool>("/drive_turtle", 1);

        RCLCPP_INFO(this->get_logger(), "Started 'turn_turtle' node");
    }

private:
    void angle_callback(const std_msgs::msg::Float64::SharedPtr msg)
    {
        double angle_degrees = msg->data;
        RCLCPP_INFO(this->get_logger(), "Received angle: %.2f degrees. Stopping and turning turtle.", angle_degrees);
        turn_turtle(angle_degrees);
    }

    void turn_turtle(double angle_degrees)
    {
        // Stop the turtle's current movement
        /*
        auto stop_message = geometry_msgs::msg::Twist();
        stop_message.linear.x = 0.0;
        stop_message.angular.z = 0.0;
        cmd_vel_publisher_->publish(stop_message);
        */
        rclcpp::sleep_for(std::chrono::seconds(1)); // Wait for a second to ensure it stops

        if (!client_->wait_for_service(std::chrono::seconds(1))) {
            RCLCPP_ERROR(this->get_logger(), "Service not available. Make sure the turtlesim node is running.");
            return;
        }

        auto request = std::make_shared<turtlesim::srv::TeleportRelative::Request>();
        request->linear = 0.0;
        request->angular = angle_degrees * (M_PI / 180.0); // Convert degrees to radians

        auto result = client_->async_send_request(request);
        // Don't wait for the service to respond! It does not.

        // Notify that the turning is finished
        auto drive_message = std_msgs::msg::Bool();
        drive_message.data = true;
        rclcpp::sleep_for(std::chrono::seconds(1));
        drive_turtle_publisher_->publish(drive_message);
    }

```

```
        rclcpp::Subscription<std_msgs::msg::Float64>::SharedPtr subscription_;
        rclcpp::Client<turtlesim::srv::TeleportRelative>::SharedPtr client_;
        rclcpp::Publisher<geometry_msgs::msg::Twist>::SharedPtr cmd_vel_publisher_;
        rclcpp::Publisher<std_msgs::msg::Bool>::SharedPtr drive_turtle_publisher_;
    };

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

Listing A.3. /turn_turtle source code

```
#include "rclcpp/rclcpp.hpp"
#include "rcl_interfaces/msg/log.hpp"
#include "std_msgs/msg/float64.hpp"
#include "std_msgs/msg/bool.hpp"

class Edge : public rclcpp::Node
{
public:
    Edge() : Node("edge"), should_look_for_edges_(false)
    {
        drive_turtle_subscription_ = this->create_subscription<std_msgs::msg::Bool>(
            "/drive_turtle", 1, std::bind(&Edge::drive_turtle_callback, this, std::placeholders::_1));
        turn_angle_publisher_ = this->create_publisher<std_msgs::msg::Float64>("/turn_angle", 1);
        drive_turtle_publisher_ = this->create_publisher<std_msgs::msg::Bool>("/drive_turtle", 1);

        RCLCPP_INFO(this->get_logger(), "Started 'edge' node");
    }

private:
    void subscribe_to_rosout()
    {
        rosout_subscription_ = this->create_subscription<rcl_interfaces::msg::Log>(
            "/rosout", 1, std::bind(&Edge::listener_callback, this, std::placeholders::_1));
    }

    void drive_turtle_callback(const std_msgs::msg::Bool::SharedPtr msg)
    {
        should_look_for_edges_ = msg->data;
        if (should_look_for_edges_)
```

```cpp
        {
            RCLCPP_INFO(this->get_logger(), "Received true on /drive_turtle. Subscribing to /rosout for edge detection.");
            subscribe_to_rosout();
        }
        else
        {
            RCLCPP_INFO(this->get_logger(), "Received false on /drive_turtle. Unsubscribing from /rosout.");
            rosout_subscription_.reset();
        }
    }

    void listener_callback(const rcl_interfaces::msg::Log::SharedPtr msg)
    {
        if (should_look_for_edges_ && msg->msg.find("Oh no! I hit the wall!") != std::string::npos)
        {
            RCLCPP_INFO(this->get_logger(), "Received message: 'Oh no! I hit the wall!'. Publishing angle and stopping turtle...");
            publish_drive_turtle(false);
            publish_turn_angle();
            // Unsubscribing from /rosout for 5 milliseconds to avoid multiple wall-hitting messages
            /**/
            rosout_subscription_.reset();
            std::this_thread::sleep_for(std::chrono::milliseconds(50));
            //subscribe_to_rosout();
        }
    }

    void publish_turn_angle()
    {
        auto message = std_msgs::msg::Float64();
        message.data = -90.0; // 90 degrees clockwise
        turn_angle_publisher_->publish(message);
    }

    void publish_drive_turtle(bool data)
    {
        auto message = std_msgs::msg::Bool();
        message.data = data;
        drive_turtle_publisher_->publish(message);
    }

    rclcpp::Subscription<rcl_interfaces::msg::Log>::SharedPtr rosout_subscription_;
    rclcpp::Subscription<std_msgs::msg::Bool>::SharedPtr drive_turtle_subscription_;
    rclcpp::Publisher<std_msgs::msg::Float64>::SharedPtr turn_angle_publisher_;
    rclcpp::Publisher<std_msgs::msg::Bool>::SharedPtr drive_turtle_publisher_;
    bool should_look_for_edges_;
```

```cpp
};

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

Listing A.4. /edge source code

Appendix B

Auxiliary files

```

cmake_minimum_required(VERSION 3.8)
project(turtlesimAutomata)

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)
# uncomment the following section in order to fill in
# further dependencies manually.
# find_package(<dependency> REQUIRED)
find_package(rclcpp REQUIRED)
find_package(std_msgs REQUIRED)
#find_package(rosidl_default_generators REQUIRED)
#find_package(turtlesimAutomata REQUIRED)
find_package(turtlesim REQUIRED)
find_package(geometry_msgs REQUIRED)
find_package(rcl_interfaces)


#rosidl_generate_interfaces(${PROJECT_NAME}
#  "srv/acknowledge.srv"
#  DEPENDENCIES std_msgs
#)

# Include directories
include_directories(include)



add_executable(edge src/edge.cpp)
ament_target_dependencies(edge rclcpp std_msgs)
#  $<BUILD_INTERFACE:${CMAKE_CURRENT_SOURCE_DIR}/include>
#  $<INSTALL_INTERFACE:include>)
#target_compile_features(edge PUBLIC c_std_99 cxx_std_17)  # Require C99 and C++17
```

```

    add_executable(turn_turtle src/turn_turtle.cpp)
    ament_target_dependencies(turn_turtle rclcpp std_msgs turtlesim geometry_msgs)

    add_executable(drive_turtle_continuously src/drive_turtle_continuously.cpp)
    ament_target_dependencies(drive_turtle_continuously std_msgs rclcpp geometry_msgs)

    add_executable(start_turtle src/start_turtle.cpp)
    ament_target_dependencies(start_turtle std_msgs rclcpp rcl_interfaces)
    target_include_directories(turn_turtle PUBLIC
      $<BUILD_INTERFACE:${CMAKE_CURRENT_SOURCE_DIR}/include>
      $<INSTALL_INTERFACE:include>)
    target_compile_features(turn_turtle PUBLIC c_std_99 cxx_std_17)  # Require C99 and C++17


    install(TARGETS
      edge
      turn_turtle
      drive_turtle_continuously
      start_turtle

      DESTINATION lib/${PROJECT_NAME})


    #this is important for the launchfile to be accesible from the ros2_ws
    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
      # a copyright and license is added to all source files
      set(ament_cmake_cpplint_FOUND TRUE)
      ament_lint_auto_find_test_dependencies()
    endif()

    ament_package()
```

Listing B.1. CMakeLists.txt

```
    <?xml version="1.0"?>
<?xml-model href="http://download.ros.org/schema/package_format3.
```

```
    xsd" schematypens="http://www.w3.org/2001/XMLSchema"?>
<package format="3">
  <name>turtlesimAutomata</name>
  <version>0.0.0</version>
  <description>TODO: Package description</description>
  <maintainer email="jakobspindler@gmx.at">jakob</maintainer>
  <license>Apache-2.0</license>

  <!--depend>std_msgs</depend>
  <buildtool_depend>rosidl_default_generators</buildtool_depend>
  <exec_depend>rosidl_default_runtime</exec_depend>
  <member_of_group>rosidl_interface_packages</member_of_group-->

  <buildtool_depend>ament_cmake</buildtool_depend>

  <depend>rclcpp</depend>
  <depend>std_msgs</depend>
  <depend>turtlesim</depend>
  <depend>geometry_msgs</depend>
  <depend>rcl_interfaces</depend>



  <test_depend>ament_lint_auto</test_depend>
  <test_depend>ament_lint_common</test_depend>

  <exec_depend>ros2launch</exec_depend>

  <export>
    <build_type>ament_cmake</build_type>
  </export>
</package>
```

Listing B.2. Package.xml

```
    <launch>
    <!-- Launch the drive_turtle_continuously node -->
    <node pkg="turtlesimAutomata" exec="drive_turtle_continuously"
   name="drive_turtle_continuously" output="screen"/>

    <!-- Launch the edge node -->
    <node pkg="turtlesimAutomata" exec="edge" name="edge" output="
   screen"/>

    <!-- Launch the turn_turtle node -->
    <node pkg="turtlesimAutomata" exec="turn_turtle" name="
   turn_turtle" output="screen"/>

    <!-- Launch the start_turtle node -->
    <node pkg="turtlesimAutomata" exec="start_turtle" name="
   start_turtle" output="screen"/>

    <!-- Launch the turtlesim node -->
    <node pkg="turtlesim" exec="turtlesim_node" name="
   turtlesim_node" output="screen"/>
  </launch>
```

Listing B.3. turtlesimAutomata_launch.xml