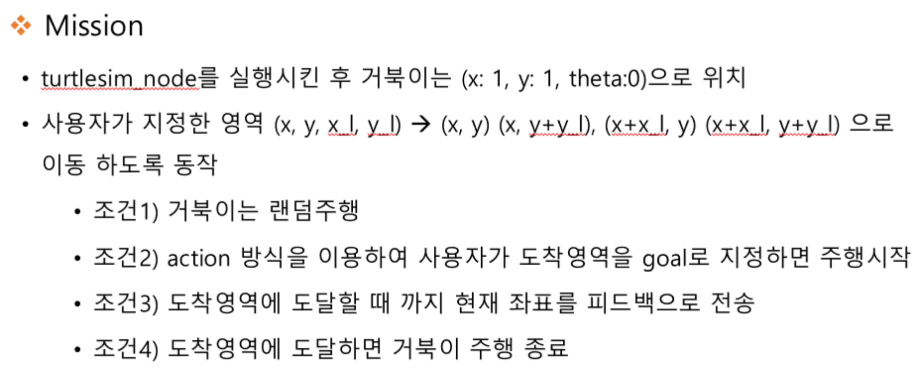
▼ 07_ROS2 PKG Action Server 구현
Mission



❖ Mission

- turtlesim_node를 실행시킨 후 거북이는 (x: 1, y: 1, theta:0)으로 위치
- 사용자가 지정한 영역 (x, y, x_l, y_l) → (x, y) (x, y+y_l), (x+x_l, y) (x+x_l, y+y_l) 으로 이동 하도록 동작
  - 조건1) 거북이는 랜덤주행
  - 조건2) action 방식을 이용하여 사용자가 도착영역을 goal로 지정하면 주행시작
  - 조건3) 도착영역에 도달할 때 까지 현재 좌표를 피드백으로 전송
  - 조건4) 도착영역에 도달하면 거북이 주행 종료

Step-By-Step

Setp-1) Action Control에서 사용할 Massage Type 선언

- Custom Massage type 선언을 위한 Package 선언
  - act_msg

```
ros2 pkg create --build-type ament_cmake act_msg
```

- Action은 Request, Response, Feedback에 대한 데이터 선언 필요.
  - action 폴더 내 MoveInGoal.action 파일 생성

```
######################
# MoveInGoal.action
######################
# Goal: 주행할 사각형 영역 정의
float32 x        # 시작점 X 좌표
float32 y        # 시작점 Y 좌표
float32 x_length  # 사각형의 가로 길이
float32 y_length  # 사각형의 세로 길이
---
# Result: 최종 결과
bool success      # 성공 여부
```

```
---
# Feedback: 실시간 피드백
float32 remaining_distance # 현재 목표 지점까지 남은 거리
```

- Package.xml 수정

```xml
<?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>act_msg</name>
  <version>0.0.0</version>
  <description>TODO: Package description</description>
  <maintainer email="daesung@todo.todo">daesung</maintainer>
  <license>TODO: License declaration</license>

  <buildtool_depend>ament_cmake</buildtool_depend>

  <build_depend>rosidl_default_generators</build_depend>
  <exec_depend>rosidl_default_runtime</exec_depend>
  <member_of_group>rosidl_interface_packages</member_of_group>

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

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

- CMakeLists.txt 수정

```cmake
cmake_minimum_required(VERSION 3.8)
project(act_msg)

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(rosidl_default_generators REQUIRED)

# Action 파일로부터 C++, Python 코드를 생성하기 위한 설정
rosidl_generate_interfaces(${PROJECT_NAME} "action/MoveInGoal.
action")

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 sou
rce 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()
```

- colcon build → source install/local_setup.bash
- 다음과 같은 명령어를 통해 메세지 타입 등록 확인

```
> ros2 interface show act_msg/action/MoveInGoal
######################
# MoveInGoal.action
######################
# Goal: 주행할 사각형 영역 정의
float32 x        # 시작점 X 좌표
```

```
float32 y         # 시작점 Y 좌표
float32 x_length  # 사각형의 가로 길이
float32 y_length  # 사각형의 세로 길이
---
# Result: 최종 결과
bool success      # 성공 여부
---
# Feedback: 실시간 피드백
```

Setp-2) Action Server 구현을 위한 패키지 생성

```
ros2 pkg create --build-type ament_python --node-name action_server act_pkg
```

```
###########################
# action_server.py
###########################

import rclpy
from rclpy.node import Node
from rclpy.action import ActionServer, GoalResponse, CancelResponse
from rclpy.executors import MultiThreadedExecutor
from rclpy.callback_groups import ReentrantCallbackGroup

import math
import time
import random

from turtlesim.msg import Pose
from geometry_msgs.msg import Twist
from std_srvs.srv import Empty
from turtlesim.srv import TeleportAbsolute

from act_msg.action import MoveInGoal
```

```
# -------------------------------------------------------------------
# 동작 로직 계산 클래스
# -------------------------------------------------------------------
class TurtleBehaviors:
    """
    거북이의 개별 동작 로직을 계산하는 헬퍼 클래스.
    """
    def is_near_boundary(self, current_pose: Pose, min_x, max_x, min_y,
max_y, padding=0.5) → bool:
        """지정된 경계에 가까운지 확인합니다."""
        return (current_pose.x < min_x + padding or
                current_pose.x > max_x - padding or
                current_pose.y < min_y + padding or
                current_pose.y > max_y - padding)

    def get_turn_to_target_twist(self, current_pose: Pose, target_x: float, t
arget_y: float) → (Twist, bool):
        """목표 지점을 바라보도록 회전하는 Twist 메시지를 생성합니다."""
        twist_msg = Twist()
        is_done = False
        target_angle = math.atan2(target_y - current_pose.y, target_x - curr
ent_pose.x)
        angle_diff = math.atan2(math.sin(target_angle - current_pose.thet
a),
                                math.cos(target_angle - current_pose.theta))

        if abs(angle_diff) > 0.1:
            twist_msg.angular.z = 4.0 * angle_diff
        else:
            is_done = True
            twist_msg.angular.z = 0.0
        return twist_msg, is_done

    def get_random_drive_twist(self) → Twist:
        """랜덤 주행을 위한 Twist 메시지를 생성합니다."""
        twist_msg = Twist()
        twist_msg.linear.x = random.uniform(1.5, 2.5)
        twist_msg.angular.z = random.uniform(-3.0, 3.0)
        return twist_msg

    def get_reverse_twist(self) → Twist:
        """후진을 위한 Twist 메시지를 생성합니다."""
        twist_msg = Twist()
        twist_msg.linear.x = -1.0
        return twist_msg

# ---------------------------------------------------------------------
# Action Server 노드 클래스
# ---------------------------------------------------------------------
class RectangleActionServer(Node):
    def __init__(self):
        super().__init__('rectangle_action_server')
        self.current_pose_ = None
        self.is_pose_ready_ = False
        self.behaviors = TurtleBehaviors()

        self.callback_group = ReentrantCallbackGroup()

        self.pose_subscriber_ = self.create_subscription(
            Pose, '/turtle1/pose', self.pose_callback, 10, callback_group=self.
callback_group)
        self.cmd_vel_publisher_ = self.create_publisher(
            Twist, '/turtle1/cmd_vel', 10)
        self.clear_client_ = self.create_client(Empty, '/clear', callback_group
=self.callback_group)
        self.teleport_client_ = self.create_client(TeleportAbsolute, '/turtle1/t
eleport_absolute', callback_group=self.callback_group)

        self.action_server_ = ActionServer(
            self, MoveInGoal, 'move_in_rectangle',
            execute_callback=self.execute_callback,
            goal_callback=lambda gr: GoalResponse.ACCEPT,
            handle_accepted_callback=lambda gh: gh.execute(),
            cancel_callback=lambda cr: CancelResponse.ACCEPT,
            callback_group=self.callback_group)
        self.wait_for_services()
        self.get_logger().info("Action Server has been started and is ready f
or goals.")

    def pose_callback(self, msg):
        self.current_pose_ = msg
        self.is_pose_ready_ = True

    def wait_for_services(self):
        """노드 시작 시 필요한 서비스들이 활성화될 때까지 대기합니다."""
        while not self.clear_client_.wait_for_service(timeout_sec=1.0):
            self.get_logger().info('/clear service not available, waiting agai
n...')
        while not self.teleport_client_.wait_for_service(timeout_sec=1.0):
            self.get_logger().info('/turtle1/teleport_absolute service not availa
ble, waiting again...')
        self.get_logger().info("All required services are active.")

    def execute_callback(self, goal_handle):
        self.get_logger().info('New goal received! Resetting simulation stat
e...')

        # 1. 터틀을 초기 위치로 이동하고 화면을 클리어
        teleport_req = TeleportAbsolute.Request(x=2.0, y=2.0, theta=0.78)
        self.teleport_client_.call_async(teleport_req)
        self.clear_client_.call_async(Empty.Request())

        # 서비스 호출이 완료될 때까지 잠시 대기
        time.sleep(1.0)
        self.get_logger().info("Reset complete. Starting random drive from i
nitial position...")

        goal = goal_handle.request

        # 절대 이동 영역 설정
        ABS_MIN_X, ABS_MAX_X = 1.0, 10.0
        ABS_MIN_Y, ABS_MAX_Y = 1.0, 10.0
        ABS_CENTER_X = (ABS_MIN_X + ABS_MAX_X) / 2.0
        ABS_CENTER_Y = (ABS_MIN_Y + ABS_MAX_Y) / 2.0

        # 상태 머신 변수
        drive_state = 'DRIVING'
        state_counter = 0

        # 메인 실행 루프
        while rclpy.ok():
            if not self.is_pose_ready_:
                time.sleep(0.1)
                continue

            if goal_handle.is_cancel_requested:
                return self.handle_cancel(goal_handle)

            if self.is_in_destination_area(self.current_pose_, goal):
                self.get_logger().info("Destination area reached! Goal succeed
ed.")
                self.stop_turtle()
                goal_handle.succeed()
                return MoveInGoal.Result(success=True)

            dist_to_dest_center = math.sqrt((goal.x - self.current_pose_.x)**
2 + (goal.y - self.current_pose_.y)**2)
            goal_handle.publish_feedback(MoveInGoal.Feedback(remaining_
distance=dist_to_dest_center))

            is_near_abs_boundary = self.behaviors.is_near_boundary(
                self.current_pose_, ABS_MIN_X, ABS_MAX_X, ABS_MIN_Y, AB
S_MAX_Y)

            twist_msg = Twist()

            if drive_state == 'DRIVING':
                if is_near_abs_boundary:
                    drive_state, state_counter = 'REVERSING', 5
                    twist_msg = self.behaviors.get_reverse_twist()
                else:
                    twist_msg = self.behaviors.get_random_drive_twist()
            elif drive_state == 'REVERSING':
                twist_msg = self.behaviors.get_reverse_twist()
                state_counter -= 1
                if state_counter <= 0:
                    drive_state = 'TURNING'
            elif drive_state == 'TURNING':
                twist_msg, is_done = self.behaviors.get_turn_to_target_twist(
                    self.current_pose_, ABS_CENTER_X, ABS_CENTER_Y)
                if is_done:
                    drive_state = 'DRIVING'

            self.cmd_vel_publisher_.publish(twist_msg)
            time.sleep(0.1)

        goal_handle.abort()
        return MoveInGoal.Result(success=False)

    def is_in_destination_area(self, pose, goal):
        return (goal.x <= pose.x <= goal.x + goal.x_length and
                goal.y <= pose.y <= goal.y + goal.y_length)

    def handle_cancel(self, goal_handle):
        goal_handle.canceled()
        self.stop_turtle()
        self.get_logger().info('Goal canceled!')
        return MoveInGoal.Result(success=False)

    def stop_turtle(self):
        self.cmd_vel_publisher_.publish(Twist())

def main(args=None):
    rclpy.init(args=args)
    action_server = RectangleActionServer()
    executor = MultiThreadedExecutor()
    executor.add_node(action_server)
    try:
        executor.spin()
    except KeyboardInterrupt:
        pass
    finally:
        executor.shutdown()
        action_server.destroy_node()
        rclpy.shutdown()

if __name__ == '__main__':
    main()
```

- colcon build → source install/local_setup.bash
- turtlesim_node 실행
- action_server 실행

```
> ros2 run act_pkg action_server
[INFO] [1761020477.898016115] [rectangle_action_server]: All required s
ervices are active.
[INFO] [1761020477.898284542] [rectangle_action_server]: Action Serv
er has been started and is ready for goals.
```

- action server로 목표 값 전송

```
ros2 action send_goal --f /move_in_rectangle act_msg/action/MoveInGo
al "{x: 6.0, y: 6.0, x_length: 2.0, y_length: 2.0}"
```