Kyungpook National University
School of Electronics Engineering

자율시스템 설계

#보고서 2	Student ID: 2021115004 Name: 손창우 강의담당교수: 박찬은

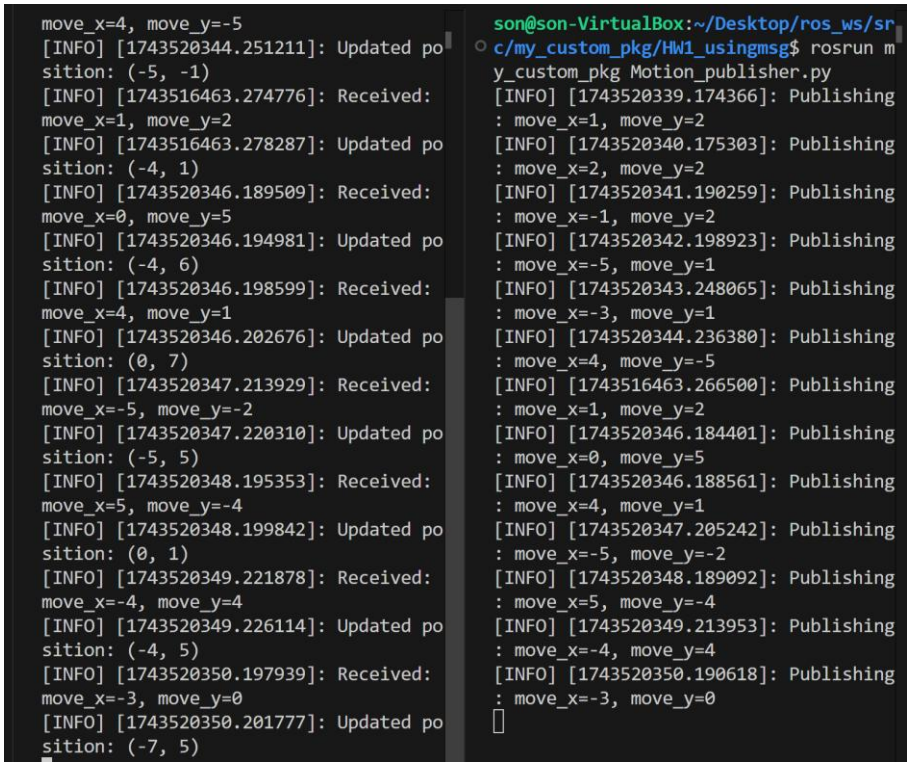
1. 강의 내용 요약 및 시뮬레이션 목적

ROS custom message
ROS에서 기본 제공하는 메시지 타입 외에, 사용자가 직접 정의한 메시지 타입이다.
.msg 파일을 만들어 필드와 타입을 정의하고, 빌드 시스템에 등록하여 사용한다.

ROS parameter
노드가 실행될 때 외부에서 값을 주입할 수 있도록 해주는 설정 해주는 전역변수.
Rosparam 명령어나 코드로 설정가능하다.
전역적 혹은 노드 로컬 범위로 사용가능하다.

ROS action
서비스/메시지 혼합 형태의 통신 방법
구성요소
Goal - 클라이언트가 서버에 요청하는 작업의 목표
Feedback - 작업이 진행되는 동안 클라이언트에 보내는 중간 결과
Result - 작업 중단/완료 후 서버가 클라이언트에 반환하는 최종 결과

2. 시뮬레이션 결과



1) 시뮬레이션 1 결과

Motion_publisher.py

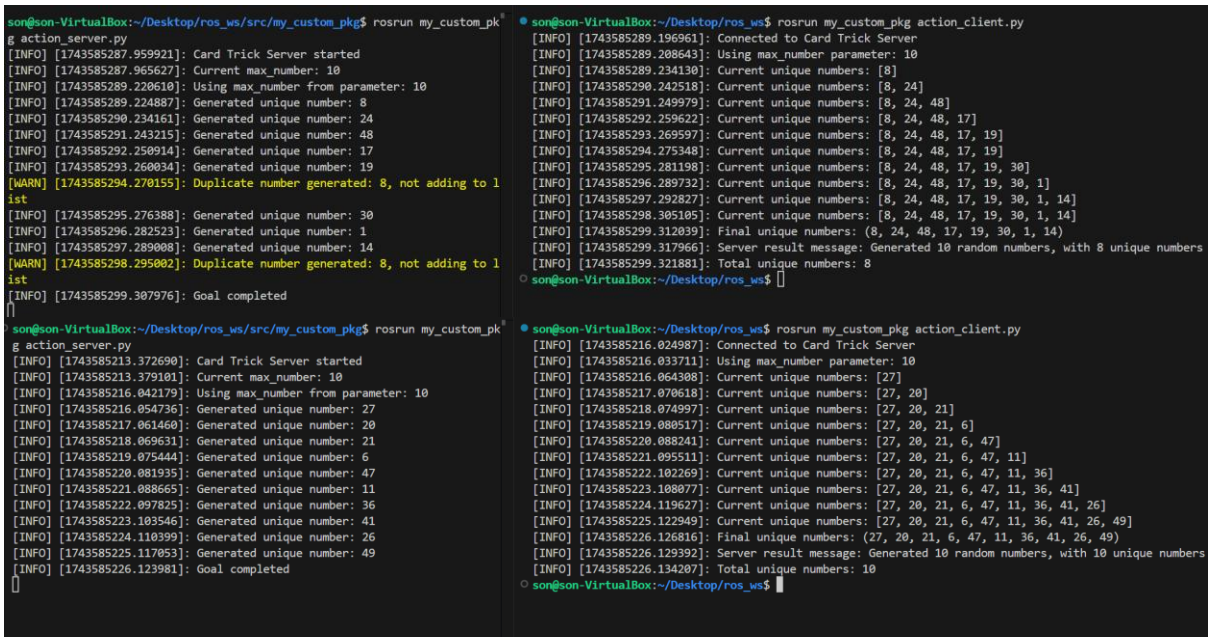
```
#!/usr/bin/env python3

import rospy

import random

from my_custom_pkg.msg import Motion   # 커스텀 메시지 임포트


def motion_publisher():

    rospy.init_node('motion_publisher', anonymous=True)

    pub = rospy.Publisher('motion_info', Motion, queue_size=10)

    rate = rospy.Rate(1)   # 1 Hz (1 초에 한 번씩 메시지 전송)
```

```
    while not rospy.is_shutdown():

        # 메시지 객체 생성

        motion_msg = Motion()

        # 랜덤하게 -5~5 사이의 값 생성

        motion_msg.move_x = random.randint(-5, 5)

        motion_msg.move_y = random.randint(-5, 5)



        # 메시지 퍼블리싱

        rospy.loginfo(f"Publishing: move_x={motion_msg.move_x},
move_y={motion_msg.move_y}")

        pub.publish(motion_msg)



        rate.sleep()



if __name__ == '__main__':

    try:

        motion_publisher()

    except rospy.ROSInterruptException:

        pass
```

Motion_subscriber.py

```
#!/usr/bin/env python3

import rospy

from my_custom_pkg.msg import Motion   # 커스텀 메시지 임포트
```

```
# 현재 좌표값 (초기 좌표는 (0,0))

current_position = [0, 0]



def motion_callback(data):

    # 현재 좌표 업데이트

    current_position[0] += data.move_x

    current_position[1] += data.move_y



    # 결과 출력

    rospy.loginfo(f"Received: move_x={data.move_x}, move_y={data.move_y}")

    rospy.loginfo(f"Updated position: ({current_position[0]},
{current_position[1]})")



def motion_subscriber():

    rospy.init_node('motion_subscriber', anonymous=True)

    rospy.Subscriber('motion_info', Motion, motion_callback)

    rospy.loginfo(f"Starting at position: ({current_position[0]},
{current_position[1]})")



    rospy.spin()  # 콜백 함수가 계속 실행될 수 있도록 유지

if __name__ == '__main__':

    motion_subscriber()
```



1) 시뮬레이션 2 결과

action_server.py

```
#!/usr/bin/env python3

import rospy

import actionlib

from my_custom_pkg.msg import CountingAction, CountingGoal


class CardTrickClient:

    def __init__(self):

        self.client = actionlib.SimpleActionClient('card_trick', CountingAction)

        self.client.wait_for_server()

        rospy.loginfo("Connected to Card Trick Server")


    def send_goal(self):
```

```
        # 전역 파라미터에서 max_number 값을 가져오기
        max_number = rospy.get_param('/card_trick/max_number', 5)
        rospy.loginfo(f"Using max_number parameter: {max_number}")

        # 빈 목표 전송
        goal = CountingGoal()

        # 피드백 콜백 함수 등록
        self.client.send_goal(goal, feedback_cb=self.feedback_callback)

        # 결과 대기
        self.client.wait_for_result()
        return self.client.get_result()

    def feedback_callback(self, feedback):
        # 서버로부터 받은 현재까지의 고유 숫자 목록 표시
        if feedback.current_numbers:
            numbers_list = list(feedback.current_numbers)
            rospy.loginfo(f"Current unique numbers: {numbers_list}")

if __name__ == '__main__':
    rospy.init_node('card_trick_client')

    client = CardTrickClient()
```

```
    result = client.send_goal()

    rospy.loginfo(f"Final unique numbers: {result.final_numbers}")
    rospy.loginfo(f"Server result message: {result.result_message}")
    rospy.loginfo(f"Total unique numbers: {len(result.final_numbers)}")
```

action_client.py

```
#!/usr/bin/env python3
import rospy
import actionlib
from my_custom_pkg.msg import CountingAction, CountingGoal

class CardTrickClient:
    def __init__(self):
        self.client = actionlib.SimpleActionClient('card_trick', CountingAction)
        self.client.wait_for_server()
        rospy.loginfo("Connected to Card Trick Server")

    def send_goal(self):
        # 전역 파라미터에서 max_number 값을 가져오기
        max_number = rospy.get_param('/card_trick/max_number', 5)
        rospy.loginfo(f"Using max_number parameter: {max_number}")

        # 빈 목표 전송
```

```
        goal = CountingGoal()

        # 피드백 콜백 함수 등록
        self.client.send_goal(goal, feedback_cb=self.feedback_callback)

        # 결과 대기
        self.client.wait_for_result()
        return self.client.get_result()

    def feedback_callback(self, feedback):
        # 서버로부터 받은 현재까지의 고유 숫자 목록 표시
        if feedback.current_numbers:
            numbers_list = list(feedback.current_numbers)
            rospy.loginfo(f"Current unique numbers: {numbers_list}")

if __name__ == '__main__':
    rospy.init_node('card_trick_client')

    client = CardTrickClient()
    result = client.send_goal()

    rospy.loginfo(f"Final unique numbers: {result.final_numbers}")
    rospy.loginfo(f"Server result message: {result.result_message}")
    rospy.loginfo(f"Total unique numbers: {len(result.final_numbers)}")
```

3. 코드 리뷰 & 핵심 요약

과제 1: 모션 퍼블리셔와 서브스크라이버

Motion_publisher.py 분석

Motion_publisher.py 는 ROS 노드를 생성하여 모션 데이터를 발행한다. 커스텀 메시지 타입인 Motion 을 사용하여 로봇의 이동 명령을 전송한다. 'motion_publisher' 노드는 'motion_info' 토픽에 Motion 타입 메시지를 1 초마다(1Hz) 발행한다. 메시지에는 -5 부터 5 사이의 랜덤한 x, y 이동값이 포함되며, 이 값들은 로봇의 상대적 이동량을 나타낸다. 발행 시 로그를 남겨 현재 발행 중인 데이터를 모니터링할 수 있다.

Motion_subscriber.py 분석

Motion_subscriber.py 는 'motion_info' 토픽을 구독하여 퍼블리셔가 발행한 모션 데이터를 처리한다. 노드는 current_position 전역 변수를 통해 (0,0)에서 시작하는 가상 로봇의 위치 상태를 관리한다. 메시지 수신 시 호출되는 motion_callback 함수는 받은 x, y 이동값을 현재 위치에 더하여 좌표를 갱신하고, 이 정보를 로그로 출력한다. rospy.spin() 함수로 노드가 종료될 때까지 콜백 함수가 호출될 수 있도록 대기 상태를 유지한다.

과제 2: 액션 서버와 클라이언트

action_server.py 분석

action_server.py 는 'card_trick'이라는 이름의 ROS 액션 서버를 구현한다. 서버는 CountingAction, CountingFeedback, CountingResult 커스텀 액션 메시지를 사용한다. 서버의 핵심 로직은 execute 메서드에 구현되어 있으며, ROS 파라미터에서 가져온 max_number(기본값 20, 최대 50)만큼 1~50 사이의 랜덤 숫자를 생성한다. 서버는 중복 없는 숫자만 unique_numbers 리스트에 추가하고, 매 숫자 생성 후 현재까지의 고유 숫자 목록을 피드백으로 클라이언트에게 전송한다. 모든 숫자 생성이 완료되면 최종 고유 숫자 목록과 결과 메시지를 클라이언트에게 반환한다.

action_client.py 분석

action_client.py 는 'card_trick' 액션에 대한 클라이언트를 구현한다. CardTrickClient 클래스는 SimpleActionClient 를 초기화하고 서버에 연결한다. send_goal 메서드는 ROS 파라미터에서 max_number 값(기본값 5)을 가져와 빈 CountingGoal 객체를 서버에 전송한다. 또한 피드백을 처리할 callback 함수를 등록하고, 서버의 결과가 도착할 때까지 대기한다. feedback_callback 메서드는 서버에서 피드백이 도착할 때마다, 현재까지 생성된 고유 숫자 목록을 로그로 출력한다. 최종적으로 서버로부터 받은 결과에서 고유 숫자 목록, 결과 메시지, 총 고유 숫자 개수를 출력한다.

느낀점

이번과제중에서 과제 2: 액션 서버와 클라이언트가 좀 많이 어려웠다. 파라미터 서브에 load 시키는 것 까지 꽤걸렸다. 다하고 나니 확실히 ROS 토픽,서비스,액션에 관해 이해가 넓어진 듯 하다.