Kyungpook National University

School of Electronics Engineering

자율시스템 설계

#보고서 1

Student ID: 2021115004

Name: 손창우

강의담당교수: 박찬은

1. 강의 내용 요약 및 시뮬레이션 목적

1. ROS란

ROS(Robot Operating System)는 로봇 소프트웨어 개발을 위한 메타운영체제이다. 운영체제(OS)는 아니지만, 로봇 애플리케이션을 구성하는 다양한 기능(하드웨어 추상화, 메시지 전달, 패키지 관리 등)을 지원한다. 복잡한 로봇 시스템을 노드 단위로 나누어 처리할 수 있게 해주며, 노드 간 통신을 통해 데이터를 주고받는 것이 핵심이다.

2. ROS Topic

ROS의 Topic은 노드 간 비동기 통신을 위한 메시지 기반의 채널이다. 데이터가 지속적으로 전송되어야 하는 상황에 적합하다.

- Publisher: 특정 Topic을 통해 데이터를 전송하는 노드
- Subscriber: 특정 Topic을 구독하고, 해당 데이터를 수신하는 노드

특징

다대다 통신 가능 (한 Publisher → 여러 Subscriber / 여러 Publisher → 한 Topic)

비동기적: 데이터를 보낸다고 해서 반드시 누군가가 즉시 받아야 하는 건 아님.

3. ROS Service

ROS Service는 Request-Response 기반의 동기 통신 방식이다. 요청이 있을 때 동작하는 노드간의 통신 방식이다.

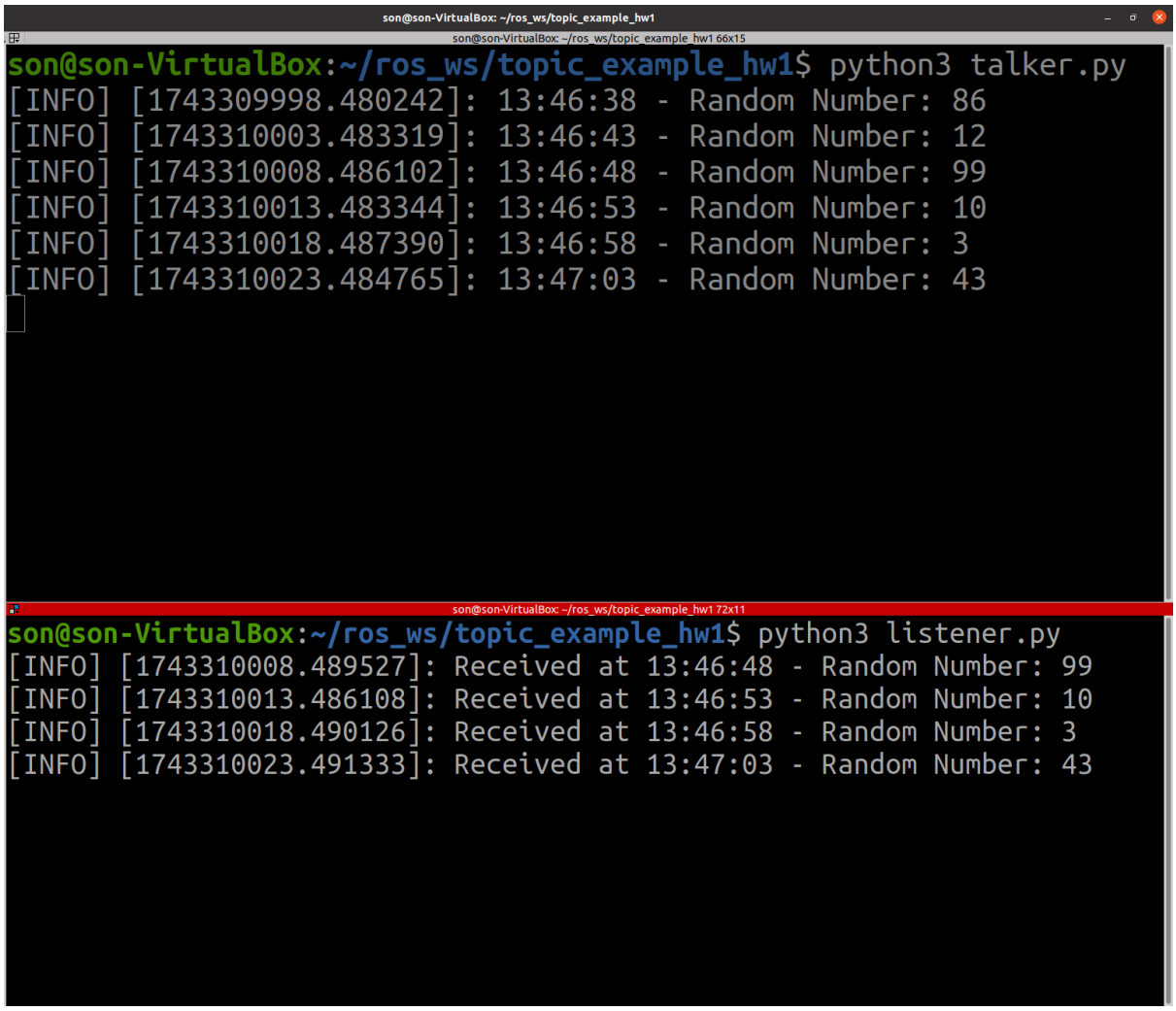
- Request: 클라이언트가 서비스 요청
- Response: 서버가 응답을 반환

특징

일회성 통신: 요청이 있을 때만 동작한다.

서비스 정의 파일 .srv을 통해 request와 response 메시지 구조를 명확히 지정한다.

2. 시뮬레이션 결과



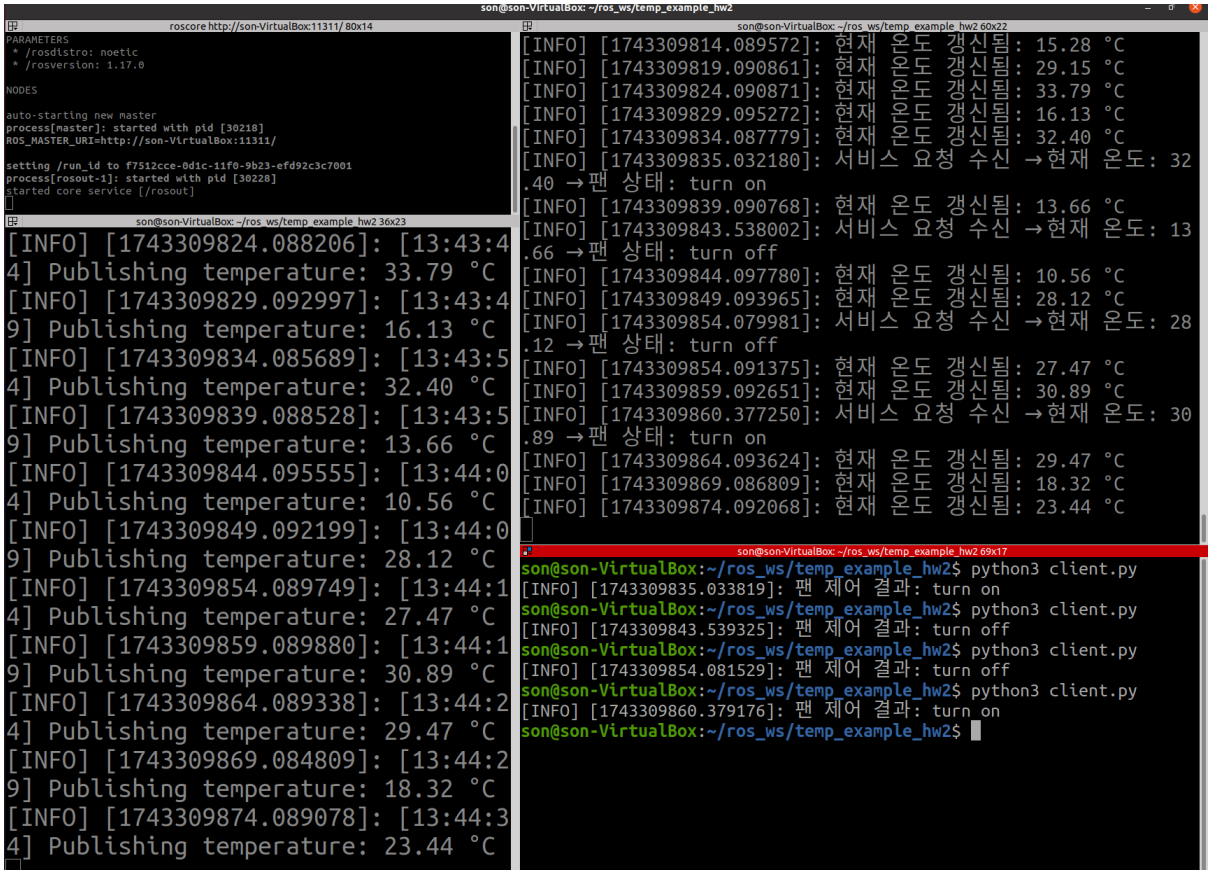
1) 시뮬레이션 1 결과 [Topic 실습]

Talker.py

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

import rospy

import random

from std_msgs.msg import String

from datetime import datetime


def publisher():

    rospy.init_node('changwoo', anonymous=True)  # 본인 이름 사용

    pub = rospy.Publisher('chatter', String, queue_size=10)

    rate = rospy.Rate(0.2)  # 5 초에 한 번 (1 초에 0.2 번)


    while not rospy.is_shutdown():

        rand_num = random.randint(1, 100)  # 1~100 사이의 랜덤 숫자

        now = datetime.now().strftime("%H:%M:%S")

        msg = f"{now} - Random Number: {rand_num}"

        rospy.loginfo(msg)

        pub.publish(msg)

        rate.sleep()


if __name__ == '__main__':

    try:

        publisher()

    except rospy.ROSInterruptException:

        pass
```

listener.py

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

import rospy

from std_msgs.msg import String

from datetime import datetime


def callback(msg):

	now = datetime.now().strftime("%Y-%m-%d %H:%M:%S")

	rospy.loginfo("Received at %s", msg.data)


def subscriber():

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

	rospy.Subscriber('chatter', String, callback)

	rospy.spin()


if __name__ == '__main__':

	try:

		subscriber()

	except rospy.ROSInterruptException:

		pass
```



1) 시뮬레이션 2 결과 [Topic + Service 실습]

Publisher.py

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

import rospy

import random

from std_msgs.msg import Float32

from datetime import datetime



def publisher():

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

    pub = rospy.Publisher('temperature', Float32, queue_size=10)

    rate = rospy.Rate(0.2)   # 5 초 간격
```

```
    while not rospy.is_shutdown():
        temp = random.uniform(10.0, 40.0)
        now = datetime.now().strftime("%H:%M:%S")
        rospy.loginfo(f"[{now}] Publishing temperature: {temp:.2f} °C")
        pub.publish(temp)
        rate.sleep()

if __name__ == '__main__':
    try:
        publisher()
    except rospy.ROSInterruptException:
        pass
```

server.py

```
#!/usr/bin/env python3
import rospy
from std_msgs.msg import Float32
from std_srvs.srv import SetBool, SetBoolResponse

current_temp = 25.0  # 초기 온도

def temp_callback(msg):
    global current_temp
    current_temp = msg.data
```

```
    rospy.loginfo(f"현재 온도 갱신됨: {current_temp:.2f} °C")


def handle_fan_service(req):

    state = "turn on" if current_temp >= 30 else "turn off"

    rospy.loginfo(f"서비스 요청 수신 → 현재 온도: {current_temp:.2f} → 팬 상태: {state}")

    return SetBoolResponse(success=True, message=state)


def server():

    rospy.init_node('fan_service_server')

    rospy.Subscriber('temperature', Float32, temp_callback)

    rospy.Service('set_fan_state', SetBool, handle_fan_service)

    rospy.loginfo("서버 시작: /set_fan_state")

    rospy.spin()


if __name__ == '__main__':

    try:

        server()

    except rospy.ROSInterruptException:

        pass
```

client.py

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

import rospy

from std_srvs.srv import SetBool
```

```
def client():
    rospy.init_node('fan_service_client')
    rospy.wait_for_service('set_fan_state')

    try:
        fan_state = rospy.ServiceProxy('set_fan_state', SetBool)
        resp = fan_state(True)  # 요청 내용은 의미 없음
        rospy.loginfo(f"팬 제어 결과: {resp.message}")
    except rospy.ServiceException as e:
        rospy.logerr(f"서비스 호출 실패: {e}")

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

2. 결론 및 느낀점

메타 운영체제라는 개념이 생소하게 느껴졌는데 직접 실습을 통해 topic과 service 시스템을 직접 실습해보면서 ROS에 대해 명확하게 이해할 수 있었다.

첫 번째 실습에서는 파이썬의 datetime 라이브러리를 사용하여 현재 시간을 나타나게 했다. Talker.py 에서 rospy.Rat(0.2)를 사용하여 5 초마다 메시지를 발행하도록 설정했습니다. Publisher 와 Subscriber 을 이용해 비동기적인 통신을 구현해봤다. Rospy.loginfo()를 통해 터미널에 정보를 출력하고 ROS 로깅 시스템에 기록하는 것을 볼 수 있었다. 나중에 센서의 topic 을 실시간으로 받아올 때 유용하게 쓰일 수 있을 것 같다는 생각이 들었다.

두 번째 실습에서는 토픽기반 통신과 서비스 기반통신을 함께 사용하는 형태여서 좀 까다로웠다. server.py 에서 current_temp 글로벌 변수를 사용하여 최신 온도 데이터를 저장하게 했다. 서버에 30°C 를 기준으로 팬의 상태를 결정하는 로직을 추가했다. 이번 예제에서는 std_msgs/Float32 와 std_srvs/SetBool 같은 ROS 의 표준 메시지와 서비스 타입을 활용하였다. 이는 재사용성을 높이고 코드를 단순화할 수 있었다. 두번 째 실습을 통해서 퍼블리셔, 서버, 클라이언트가 분리된 분산 시스템을 이해해볼 수 있었다.

실습을 해보면서 ROS 는 노드 단위로 통신하기 때문에 C++ , python 모두 지원한다는 말이 이해가 되지 않았는데 실제로 노드들을 구성해보면서 ROS 의 언어 독립적인 통신 구조를 체감할 수 있었다. 이러한 구조가 시스템 개발의 유연성과 확장성을 높인다는 점을 깨달았다.