

4) 현실을 가상으로 복제하다 : 디지털 트윈과 언리얼 엔진 연동

지금까지 우리는 파이썬 코드를 작성하고 센서를 연결하여, 현실 세계의 트랙을 스스로 달리는 'AI-Rover'를 완성했습니다. 그런데 만약 여러분 책상 위에 있는 이 로버를, 컴퓨터 속 가상 세계(메타버스)에 쌍둥이처럼 똑같이 복제할 수 있다면 어떨까요?

현실의 기계나 장비를 가상 세계에 똑같이 구현하는 기술을 '디지털 트윈(Digital Twin)'이라고 부릅니다. 테슬라, 현대자동차와 같은 글로벌 모빌리티 기업들은 자율주행 AI를 훈련시킬 때 현실의 도로뿐만 아니라, 완벽하게 통제된 가상 세계의 도로를 달리는 시뮬레이션을 적극적으로 활용하고 있습니다.

① 현실과 가상을 이어주는 마법의 우체국, MQTT

현실의 AI-Rover와 컴퓨터 속 가상 세계는 어떻게 대화를 나눌 수 있을까요? 이때 등장하는 것이 바로 사물인터넷(IoT) 통신의 표준이자 아주 가벼운 메신저인 MQTT(Message Queuing Telemetry Transport) 기술입니다.

□ MQTT의 작동 원리

MQTT의 작동 원리는 '우체국'을 상상하면 아주 쉽습니다.

- 우체국(Broker) 세팅 : 네트워크 한가운데에 메시지를 중계해 줄 우체국(브로커)을 1개 만듦
- 로버의 보고 (Publish) : 현실의 AI-Rover는 초음파 센서로 측정한 장애물 거리나 현재 속도 데이터를 우체국으로 계속 '발행(Publish)'
- 가상의 수신 (Subscribe): 가상 세계는 우체국에서 이 데이터를 '구독(Subscribe)'하여 실시간으로 받음

결과적으로 현실의 로버가 벽에 가까워지면, 모니터 속 가상의 로버도 똑같이 벽을 감지하고 멈춰 서게 됩니다. 반대로 모니터 속에서 조종석 핸들을 꺾으면, 현실의 로버가 그 명령을 '구독'하여 즉시 바퀴의 방향을 튕니다.

□ 우체국(Broker) 세팅 : 진짜 우체국 만들기

파이썬 코드로 편지를 주고받으려면, 네트워크 한가운데서 메시지를 중계해 줄 실제 우체국 프로그램이 24시간 켜져 있어야 합니다. 가장 대표적이고 가벼운 무료 우체국 프로그램으로 '모스퀴토(Eclipse Mosquitto)'가 있습니다. 우체국을 세우는 방법은 크게 두 가지가 있습니다.

- 방법 1. 공개 우체국(Public Broker) 빌려 쓰기 (가장 쉬운 방법)

가장 빠른 테스트 방법은 전 세계 개발자들을 위해 무료로 열려 있는 공용 우체국을 사용하는 것입니다. 별도의 설치 없이, 파이썬 코드의 IP 주소 부분(broker_address)에 "test.mosquitto.org" 또는 "broker.hivemq.com" 이라고 적어주기만 하면 즉시 통신이 가능합니다. 단, 공개된 장소이므로 누구나 내 로버의 데이터를 볼 수 있다는 단점이 있습니다.

- 방법 2. 우리만의 전용 우체국 만들기 (실전 방법)

우리가 만든 라즈베리파이(AI-Rover)나 언리얼 엔진이 돌아가는 PC에 직접 모스퀴토(Mosquitto)를 설치하는 방법입니다. 라즈베리파이의 터미널(명령창)을 열고 아래의 명령어 한 줄만 입력하면, 라즈베리파이 안에 빠르고 안전한 우리만의 전용 우체국이 건설됩니다.

· 설치

```
# sudo apt install mosquitto mosquitto-clients
```

설치가 끝나면 라즈베리파이의 IP 주소(예: 192.168.0.100)가 곧 우체국의 주소가 됩니다. 이제 보안 걱정 없이 언리얼 엔진과 AI-Rover가 실시간으로 비밀 대화를 나눌 수 있습니다!

□ AI-Rover의 보고 (Publish) 이해하기

- AI-Rover가 센서 데이터를 우체국으로 보낼 때 (발행, Publish).

현실의 AI-Rover가 초음파 센서로 거리를 측정하고, 그 데이터를 가상 세계(언리얼 엔진)로 보내기 위해 우체국에 편지를 넣는 코드.

```
file name : mqtt_1_publish.py

import paho.mqtt.client as mqtt
import time
import random

# 1. 우체국(MQTT 브로커) 주소 설정 (라즈베리파이나 PC의 IP 주소)
broker_address = "192.168.0.100"
client = mqtt.Client(client_id="AI_Rover_Sensor")

# 2. 우체국에 연결
client.connect(broker_address)
print("우체국 연결 완료! 데이터 전송을 시작합니다.")

try:
    while True:
        # 3. 초음파 센서 데이터라고 가정 (예시를 위해 랜덤 값 사용)
        # 실제로는 여기서 초음파 센서의 거리 값을 읽어옵니다.
        distance = random.uniform(10.0, 50.0)

        # 4. 'rover/sensor/ultrasonic' 이라는 주제(Topic)로 우체국에 데이터 발행
        client.publish("rover/sensor/ultrasonic", f'{distance:.2f}')
        print(f'[발행] 현재 장애물 거리: {distance:.2f} cm')

        time.sleep(1) # 1초마다 보냄

except KeyboardInterrupt:
    print("통신을 종료합니다.")
    client.disconnect()
```

☐ 가상의 수신 (Subscribe) 이해하기

- 가상 세계(언리얼 엔진)에서 로버에게 명령을 내릴 때 (구독, Subscribe)

이번에는 반대로 언리얼 엔진에서 출발(W) 버튼을 눌렀을 때, 현실의 AI-Rover가 그 명령을 우체국에서 받아와 모터를 움직이는 뼈대 코드

```
file name : mqtt_2_subscribe.py

import paho.mqtt.client as mqtt

# 1. 우체국에서 편지가 도착했을 때 실행될 행동 (콜백 함수)
def on_message(client, userdata, message):
    command = str(message.payload.decode("utf-8"))
    print(f'[수신] 가상 세계로부터 명령이 도착했습니다: {command}')

    # 받은 명령에 따라 모터 제어 (뼈대 로직)
    if command == "FORWARD":
        print('-> 붕붕! AI-Rover 전진합니다!\n')
        # motor_forward()
```

```

elif command == "STOP":
    print('-> 끼익! AI-Rover 정지합니다!\n')
    # motor_stop()

# 2. 통신 준비 및 우체국 연결
broker_address = "192.168.0.100"
client = mqtt.Client(client_id="AI_Rover_Motor")

# 3. 편지가 오면 'on_message' 함수를 실행하라고 예약
client.on_message = on_message
client.connect(broker_address)

# 4. 'rover/command/move' 주제(Topic)를 구독 시작
client.subscribe("rover/command/move")
print('우체국 연결 완료! 가상 세계의 명령을 기다립니다...')

# 5. 프로그램이 종료되지 않고 계속 편지를 기다림
client.loop_forever()

```

□ try 문 활용 코드

```

# file name : mqtt_3_subscribe_try.py

import paho.mqtt.client as mqtt

# 1. 우체국에 연결 성공했을 때 실행될 행동 (콜백 함수)
def on_connect(client, userdata, flags, rc):
    print('우체국 연결 완료! 가상 세계의 명령을 기다립니다...')
    # 통신이 끊겼다가 재연결되어도 다시 구독할 수 있도록 여기에 작성합니다.
    client.subscribe("rover/command/move")

# 2. 우체국에서 편지가 도착했을 때 실행될 행동 (콜백 함수)
def on_message(client, userdata, message):
    command = str(message.payload.decode("utf-8"))
    print(f'[수신] 가상 세계로부터 명령이 도착했습니다: {command}')

    # 받은 명령에 따라 모터 제어 (빠대 로직)
    if command == "FORWARD":
        print('-> 봉봉! AI-Rover 전진합니다!\n')
        # motor_forward()
    elif command == "STOP":
        print('-> 끼익! AI-Rover 정지합니다!\n')
        # motor_stop()

# 3. 통신 준비 (최신 paho-mqtt 2.0 버전에 맞춘 API 버전 명시)
broker_address = "192.168.0.100"
client = mqtt.Client(mqtt.CallbackAPIVersion.VERSION1, client_id="AI_Rover_Motor")

# 4. 편지가 오거나 연결될 때 실행할 함수 예약

```

```

client.on_connect = on_connect
client.on_message = on_message

# 5. 우체국 연결 및 대기
try:
    client.connect(broker_address)
    client.loop_forever() # 프로그램이 종료되지 않고 계속 편지를 기다림
except KeyboardInterrupt:
    print("\n통신을 종료합니다.")
    client.disconnect()

```

☐ on_connect 콜백 함수 활용

현재 코드는 연결 직후 한 번만 subscribe를 실행합니다. 만약 와이파이가 끊겼다가 다시 연결될 경우, loop_forever()가 재연결은 해주지만 구독(Subscribe) 정보는 날아가 버립니다. 우체국에 연결될 때마다 자동으로 다시 구독하도록 on_connect 함수를 만들어 그 안에 subscribe를 넣는 것이 MQTT의 정석입니다.

☐ 깔끔한 종료 처리

프로그램 실행 중 Ctrl + C를 눌러 종료할 때 에러 메시지 없이 깔끔하게 닫히도록 try ~ except KeyboardInterrupt 구문을 추가하면 좋습니다. (앞서 만든 Publish 코드와의 통일성도 챙길 수 있습니다.)

☐ 최신 paho-mqtt 버전 호환성 이슈

최근 paho-mqtt 라이브러리가 2.0 버전으로 업데이트되면서 client_id만 넣으면 경고(Warning)가 발생합니다. 교재 실습 시 혼란을 막기 위해 버전을 명시해 주는 것이 안전합니다.

② 왜 하필 언리얼 엔진(Unreal Engine)일까?

이러한 디지털 트윈을 구현할 때 가장 널리 쓰이는 도구가 바로 언리얼 엔진이나 유니티(Unity) 같은 게임 엔진입니다. 특히 언리얼 엔진은 실사와 구분이 안 갈 정도의 압도적인 그래픽과 실제 현실과 똑같은 중력, 마찰력을 계산하는 강력한 '물리 엔진'을 제공합니다.

현실에서는 로버를 수백 번 벽에 부딪히게 하며 테스트하기 어렵지만, 언리얼 엔진으로 만들어진 가상 세계에서는 비가 오거나 눈이 오는 극한의 상황에서도 안전하게 데이터를 수집하고 AI를 똑똑하게 학습시킬 수 있습니다.

③ 끝이 아닌 새로운 시작

여러분이 이 책을 통해 라즈베리파이와 파이썬으로 모터를 굴리고, 카메라로 객체를 탐지했던 경험은 결코 단순한 장난감 조종으로 끝나지 않습니다. 여러분이 작성한 코드 한 줄, 한 줄은 향후 언리얼 엔진과 같은 거대한 시뮬레이터와 결합하여 미래 산업의 핵심인 '디지털 트윈 기반의 자율주행 연구'로 나아가는 튼튼한 뿌리가 될 것입니다.

우리가 만든 AI-Rover의 한계는 현실의 벽이 아니라, 여러분의 상상력의 크기입니다.

④ source code

```
# file name : mqtt_4_motor.py
```

```

# 모터 제어 프로그램

import paho.mqtt.client as mqtt
from motor_module import * # 작성해둔 AI-Rover 모터 제어 모듈 불러오기

# 1. 우체국에 연결 성공했을 때 실행될 행동
def on_connect(client, userdata, flags, rc):
    print('    우체국 연결 완료! 가상 세계의 명령을 기다립니다...')
    # 통신이 끊겼다가 재연결되어도 다시 구독할 수 있도록 설정
    client.subscribe("rover/command/move")

# 2. 우체국에서 편지가 도착했을 때 실행될 행동 (콜백 함수)
def on_message(client, userdata, message):
    # 만약의 오타나 띄어쓰기를 방지하기 위해 대문자로 변환하고 공백 제거
    command = str(message.payload.decode("utf-8")).strip().upper()
    print(f'[수신] 명령 도착: {command}')

    # 3. 받은 명령에 따라 실제 motor_module.py의 함수 호출
    if command == "FORWARD":
        move_forward()    # 전진 함수 호출 (모듈에 정의된 이름에 맞게 수정)
    elif command == "BACKWARD":
        move_backward()    # 후진 함수 호출
    elif command == "LEFT":
        move_turn_left()    # 좌회전 함수 호출
    elif command == "RIGHT":
        move_turn_right()    # 우회전 함수 호출
    elif command == "STOP":
        move_stop()    # 정지 함수 호출
    else:
        print('Unknown command.')

# 4. 통신 준비 (MQTT 세팅)
broker_address = "192.168.0.100" # 브로커(우체국)가 설치된 PC의 IP 주소
client = mqtt.Client(mqtt.CallbackAPIVersion.VERSION1, client_id="AI_Rover_Motor")

client.on_connect = on_connect
client.on_message = on_message

# 5. 프로그램 실행 및 무한 대기
try:
    print('□ AI-Rover 수신 시스템을 가동합니다...')
    move_stop() # 시작할 때 혹시 모를 급발진 방지를 위해 정지 상태로 초기화

    client.connect(broker_address)
    client.loop_forever() # 편지가 올 때까지 계속 대기

except KeyboardInterrupt:
    print('Stopped. Ctrl+C pressed.')
except Exception as err:

```

```
print(f'Error : {err}')
```

```
finally:
```

```
    # [매주 중요] 프로그램이 꺼질 때 모터가 계속 도는 것(Runaway) 방지
```

```
    move_stop()
```

```
    client.disconnect()
```

```
    print('terminated.')
```