day4 : Micro ROS Agent

Micro-ROS란?

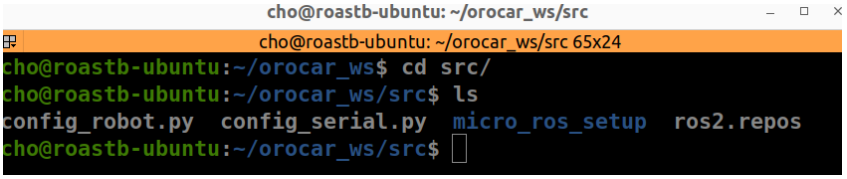
Micro-ROS는 ROS 2를 마이크로컨트롤러(MCU)에서 실행할 수 있도록 만든 프로젝트로, 여기서 **Agent** 는 MCU(예: ESP32, STM32)와 ROS2 환경(Ubuntu 등) 간의 브릿지 역할을 한다.

[1] Micro-Ros Setup (https://github.com/micro-ROS/micro_ros_setup)

1-1. git clone해서 micro_ros_setup부터 가져오기

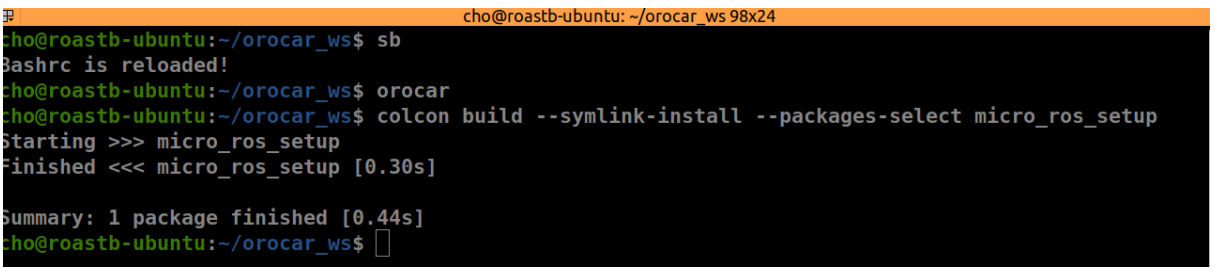
```
git clone -b humble https://github.com/micro-ROS/micro_ros_setup.git
```

1-2. 이후 확인해보면 micro_ros_setup이 있는 것을 확인가능!



1-3. 이제 ROS 환경을 한번 불러오고 colcon build!

```
source /opt/ros/humble/setup.bash
colcon build --symlink-install --packages-select micro_ros_setup
```



cf. micro-ros를 따로 설치하지 않는 이유

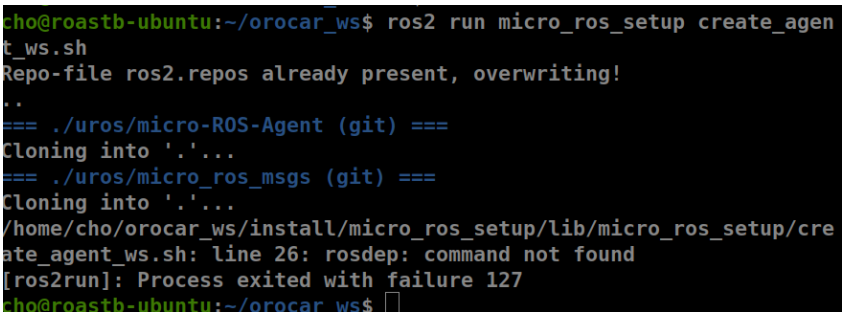
: micro-ros-setup은 사실 **ROS 2의 일반적인 C++ 패키지처럼 빌드해야 하는 게 아니라, Python 기반의 유틸리티 패키지.**

즉, 우리가 git clone으로 가져오면 이미 필요한 스크립트 파일들이 포함되어 있어서, ROS2 패키지처럼 시스템에 따로 micro-ros를 설치할 필요가 없어진다! 이제 바로 agent ws생성하면 된다!

[2] Micro ROS Agent 워크스페이스 생성

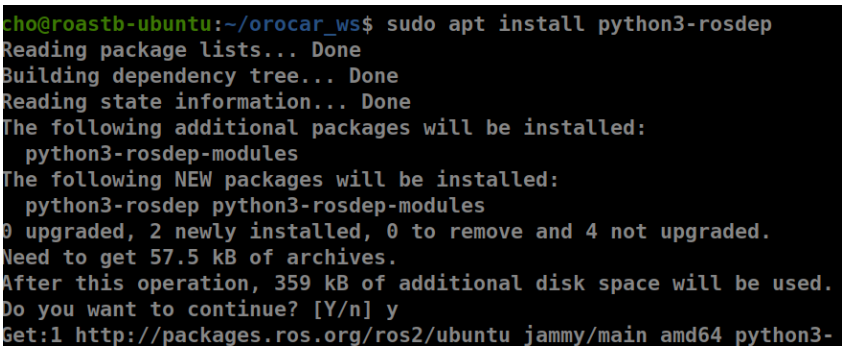
2-1. **Micro ROS Agent**의 워크스페이스를 생성... 하지만 에러!

: 이유는 rosdep가 설치가 안되어있어서!



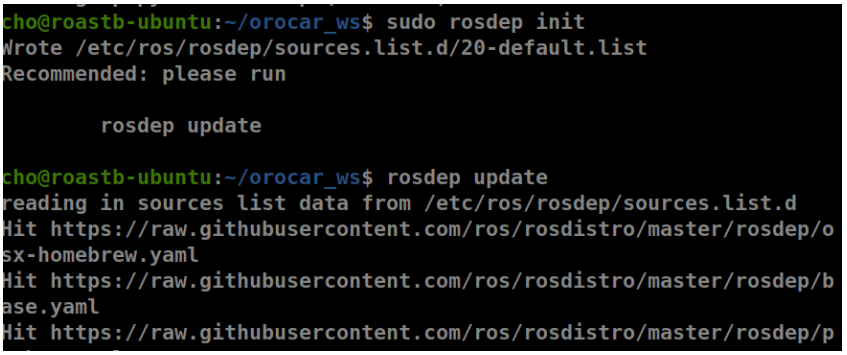
2-2. 먼저 rosdep 설치

```
sudo apt install python3-rosdep
```



2-3. 이후 rosdep 초기화 및 업데이트 한번 진행

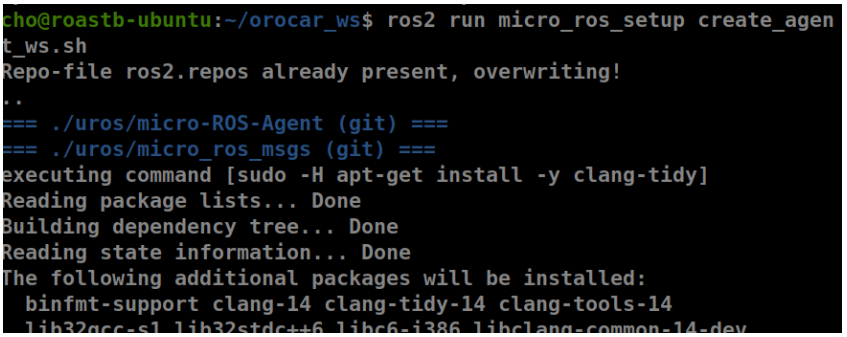
```
sudo rosdep init
rosdep update
```

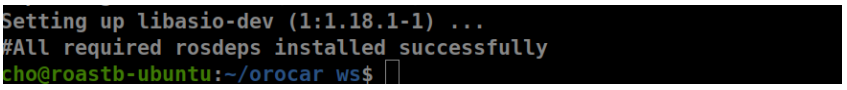


2-4. 이제 micro-ROS-Agent package를 다운로드!

: 잘 설치된 것을 확인할 수 있다! 🙂

```
ros2 run micro_ros_setup create_agent_ws.sh
```

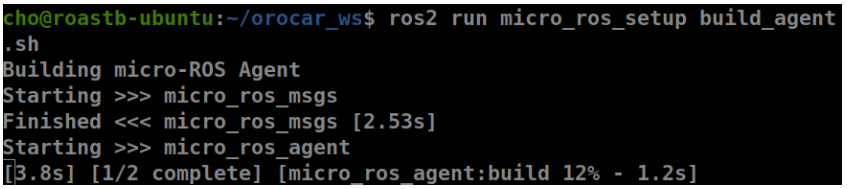




[3] Micro-ROS Agent Build

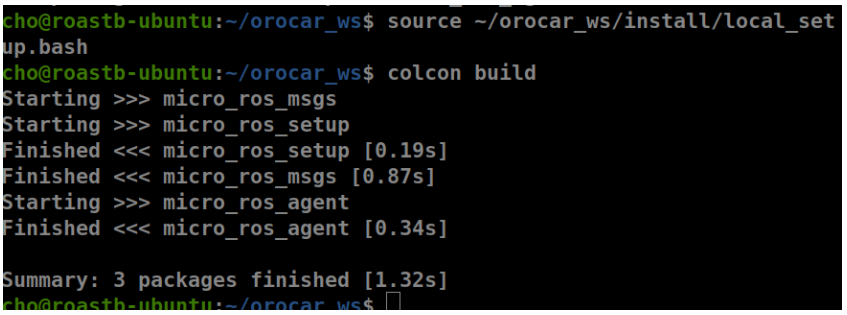
3-1. 이제 실행하기 위해서 Micro-ROS Agent Build → Micro-ROS Agent의 실행 파일(바이너리)을 생성하는 과정

```
ros2 run micro_ros_setup build_agent.sh
```



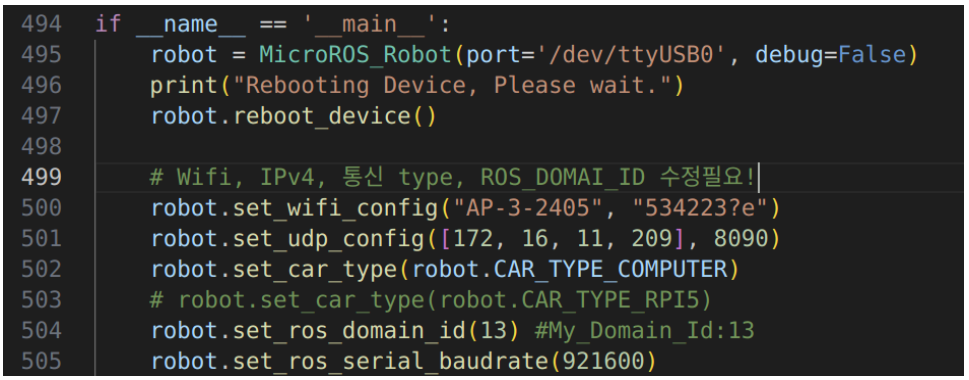
3-2. 다시 ws 환경을 한번 불러오고 colcon build!

```
source ~/orocar_ws/install/local_setup.bash (=orocar)
colcon build
```



[4] Yahboom 자율주행로봇의 ESP32(MCU) 설정 변경

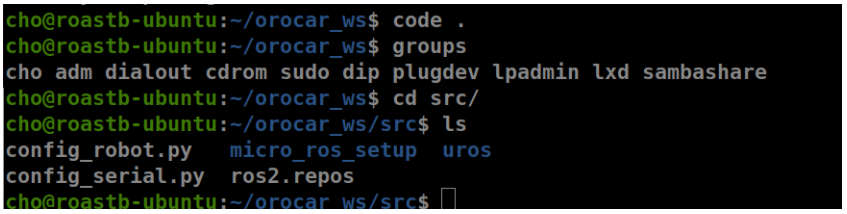
4-1. config_robot.py 수정 : 각자의 맞는 Wifi, IPv4, ROS_DOMAIN_ID로 수정



```
56
57          self.AGENT_TYPE_WIFI_UDP = 0
58          self.AGENT_TYPE_SERIAL = 1
59
60          self.CAR_TYPE_COMPUTER = 0
61          self.CAR_TYPE_RPI5 = 1
62
```

4-2. 이후 3주차에 언급한 것처럼 로봇과 컴퓨터와 연결후 실행하기 전 권한이 있는지 확인 (dialout이 있으면 권한 있는것!)

```
groups
```



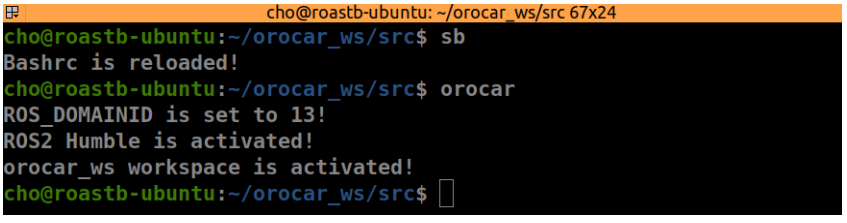
4-3. Yahboom 자율주행로봇 ↔ 컴퓨터 연결

: 이때 삐삐삐 거리는 소리가 나면 흰색 버튼을 눌러서 리셋!

[5] config_robot.py 실행

: 아래와 같이 Please wait후에 정보가 뜨는 걸 확인할 수 있다.

```
sb
orocar //ros_ws 환경을 먼저 불러오기
python3 config_home.py
```



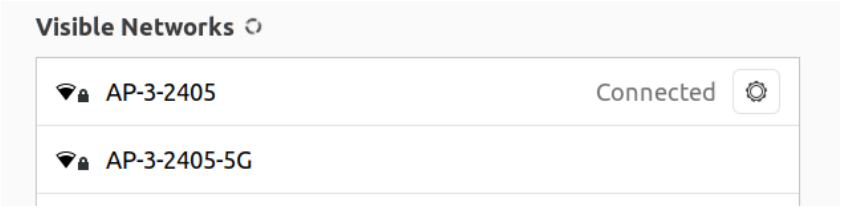
```
cho@roastb-ubuntu:~/orocar_ws/src$ python3 config_home.py
Rebooting Device, Please wait.
version: 2.1.0
ssid: AP-3-2405
passwd: 534223?e
ip_addr: 172.16.11.209
ip_port: 8090
car_type: CAR_TYPE_COMPUTER
domain_id: 13
ros_serial_baudrate: 921600
ros_namespace:
servo_offset: 0, 0
motor pid parm: 1.00, 0.20, 0.20
imu yaw pid parm: 1.00, 0.00, 0.20
Please reboot the device to take effect, if you change some device config.
cho@roastb-ubuntu:~/orocar_ws/src$
```

[6] Micro Ros Agent로 명령

6-1. 만약 아래 사진과 같이 쭉 정보가 뜨지 않는다면 yahboom 자율주행로봇의 Reset 버튼을 누른 후 20초 후 다시 실행!

6-2. 그리고 이때 만약 5G라면 일반 Wifi로 진행하기! (5G로 진행했을 때는 안되더라고요..)

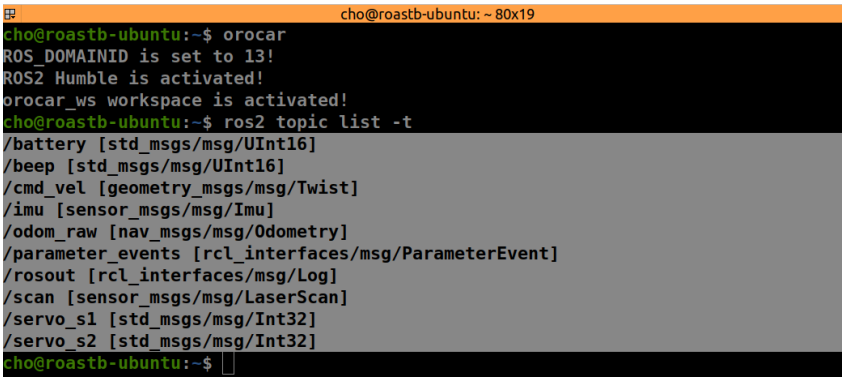
```
ros2 run micro_agent micro_ros_agent udp4 --port 8090 -v4
```




```
cho@roastb-ubuntu:~/orocar_ws/src$ ros2 run micro_ros_agent micro_ros_agent udp4 --port 8090 -v4
[1743212903.521517] info     | UDPv4AgentLinux.cpp | init                     | running...
  | port: 8090
[1743212903.521719] info     | Root.cpp           | set_verbose_level        | logger setup
  | verbose_level: 4
[1743212904.041147] info     | Root.cpp           | create_client            | create
  | client_key: 0x35720E9D, session_id: 0x81
[1743212904.041277] info     | SessionManager.hpp | establish_session        | session established
  | client_key: 0x35720E9D, address: 192.168.56.20:39455
[1743212904.260434] info     | ProxyClient.cpp    | create_participant       | participant created
  | client_key: 0x35720E9D, participant_id: 0x000(1)
[1743212904.286170] info     | ProxyClient.cpp    | create_topic             | topic created
  | client_key: 0x35720E9D, topic_id: 0x000(2), participant_id: 0x000(1)
[1743212904.297601] info     | ProxyClient.cpp    | create_publisher         | publisher created
  | client_key: 0x35720E9D, publisher_id: 0x000(3), participant_id: 0x000(1)
[1743212904.312558] info     | ProxyClient.cpp    | create_datawriter        | datawriter created
  | client_key: 0x35720E9D, datawriter_id: 0x000(5), publisher_id: 0x000(3)
[1743212904.329206] info     | ProxyClient.cpp    | create_topic             | topic created
  | client_key: 0x35720E9D, topic_id: 0x001(2), participant_id: 0x000(1)
[1743212904.340213] info     | ProxyClient.cpp    | create_publisher         | publisher created
  | client_key: 0x35720E9D, publisher_id: 0x001(3), participant_id: 0x000(1)
```

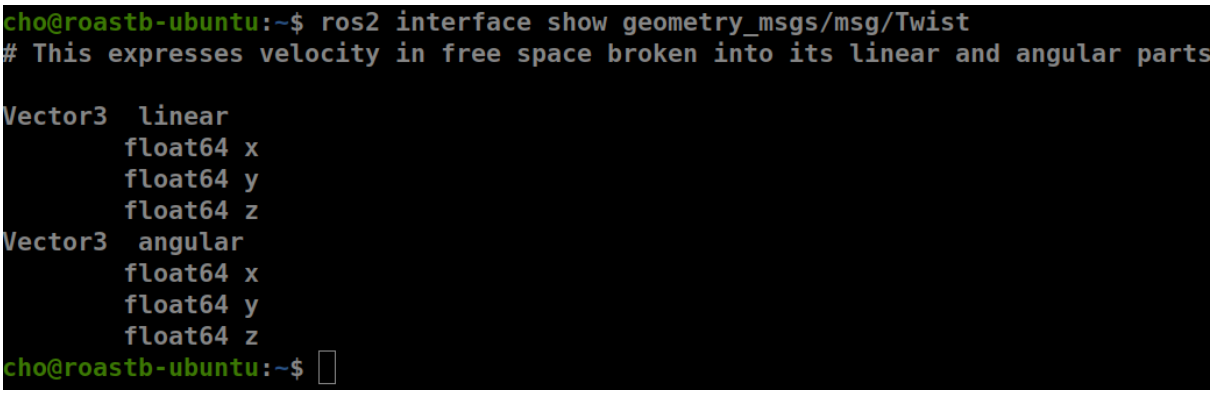
6-3. 또 다른 터미널 열어 ros_ws 환경 불러오고 topic list 확인

```
orocar
ros2 topic list -t
ros2 topic list -v
```



6-4. ros2 insterface show를 통해 cmd_vel의 type확인!

```
ros2 interface show geometry_msgs/msg/Twist
```



6-5. cmd_vel topic으로 yahboom 자율주행로봇 동작 테스트 (앞으로 진행, 정지)

```
ros2 topic pub --once /cmd_vel geometry_msgs/msg/Twist "{linear: {x: 0.07, y: 0.0, z: 0.0}, angular: {x: 0.0, y: 0.0, z: 0.0}}"
```

```
cho@roastb-ubuntu:~$ ros2 topic pub --once /cmd_vel geometry_msgs/msg/Twist "{linear: {x: 0.03
, y: 0.0, z: 0.0}, angular: {x: 0.0, y: 0.0, z: 0.0}}"
publisher: beginning loop
publishing #1: geometry_msgs.msg.Twist(linear=geometry_msgs.msg.Vector3(x=0.03, y=0.0, z=0.0),
 angular=geometry_msgs.msg.Vector3(x=0.0, y=0.0, z=0.0))
```

```
ros2 topic pub --once /cmd_vel geometry_msgs/msg/Twist "{linear: {x: 0.0, y: 0.0, z: 0.0}, angular: {x: 0.0, y: 0.0, z: 0.0}}"
```


```
cho@roastb-ubuntu:~$ ros2 topic pub --once /cmd_vel geometry_msgs/msg/Twist "{linear: {x: 0.0,
 y: 0.0, z: 0.0}, angular: {x: 0.0, y: 0.0, z: 0.0}}"
publisher: beginning loop
publishing #1: geometry_msgs.msg.Twist(linear=geometry_msgs.msg.Vector3(x=0.0, y=0.0, z=0.0),
angular=geometry_msgs.msg.Vector3(x=0.0, y=0.0, z=0.0))
```


[실행결과] 잘 움직인다! 🐢