AI TRAINING SERIES

# Day 06

# 실습

https://www.raspberrypi.com/software/

# 라즈베리 파이 OS 설치하기

라즈베리 파이에서 지원하는 rpi-imager 프로그램으로 Raspberry Pi OS 를 다운로드 받아 microSD card 에 Write 를 손쉽게 할 수 있다.

Ubuntu 환경에서 아래 명령어로 손쉽게 rpi-imager 를 설치 할 수 있다.

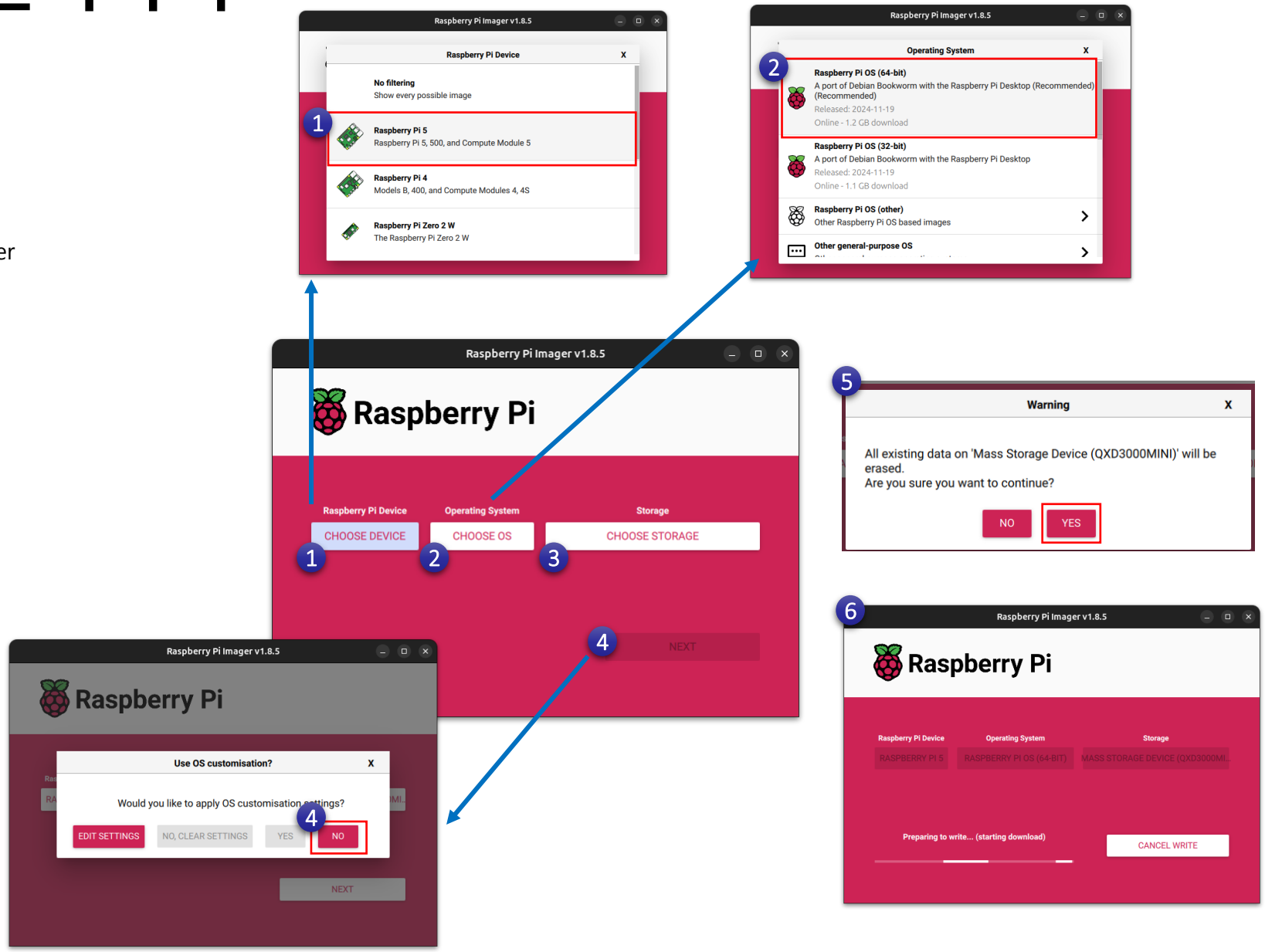
```
$ sudo apt install rpi-imager
```

설치가 완료 된 후 terminal 창에 rpi-imager 를 실행시켜 microSD card 에 Raspberry Pi OS 를 Write 하자.

우리는 Raspberry Pi 5 에 64-bit 시스템 이므로 그에 맞는 항목을 선택하고 Write 를 진행하면 이 툴에서 자동으로 관련 image 들을 다운로드 하고 microSD card 에 write 한다.

이 과정이 모두 완료됐으면 microSD card 를 라즈베라파이 5 보드에 장착 후 부팅해 보자.

User ID: pi, PW: raspberry 로 **무조건 통일**



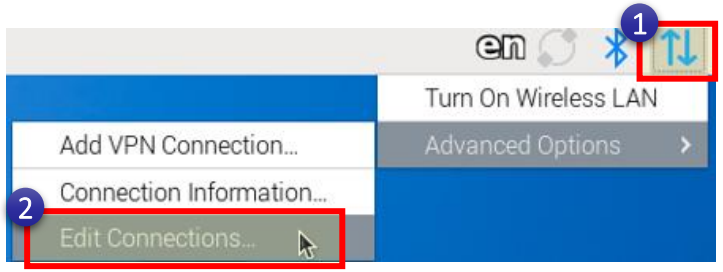
# Wired Connection 용 IP 할당

## Network IP 를 manual 하게 할당하기 (Wireless는 차후 공지 예정)

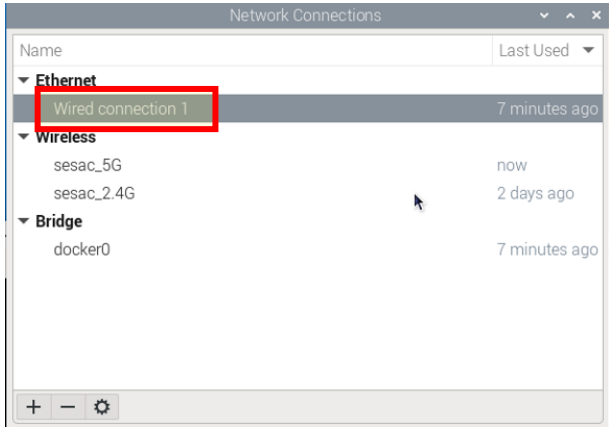
1. 라즈베리파이 무선 네트워크를 Disable


2. 라즈베리파이에 LAN cable 을 연결

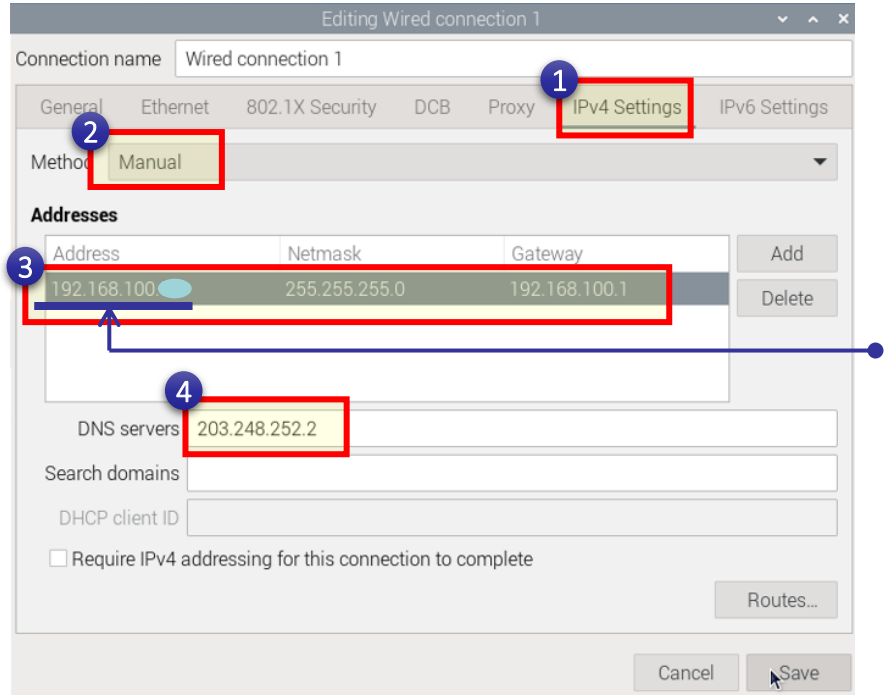
3. Network 설정 변경 메뉴 열기




4. Wired connection 1 선택해 설정메뉴 열기




5. 각자 자신에게 할당된 IP address 를 입력 후 저장




ROS car 용 라즈베리파이의 IP address 를 반드시 자신에게 할당된 것으로 를 입력!!

<Robot Car IP address 범위>
192.168.100.91 ~ 192.168.100.98

# Robot Car 라즈베리파이 IP 할당

## 강사위치

192.168.100.xx ← xx 부분에 각 자리별 할당된 번호를 사용할 것

| | | | |
|---|---|---|---|
| 98 | 97 | 96 | 95 |
| 94 | 93 | 92 | 91 |

| | | | |
|---|---|---|---|
| 85 | 81 | 75 | 71 |
| 86 | 82 | 76 | 72 |
| 87 | 83 | 77 | 73 |
| 88 | 84 | 78 | 74 |
| 89 | | 79 | |

| | | | |
|---|---|---|---|
| 65 | 61 | 55 | 51 |
| 66 | 62 | 56 | 52 |
| 67 | 63 | 57 | 53 |
| 68 | 64 | 58 | 54 |
| 69 | | 59 | |

https://www.raspberrypi.com/documentation/computers/configuration.html#raspi-config

# SSH 접속하기

## 라즈베리파이 Configuration - 원격 접속하기 (SSH: Secure Shell)

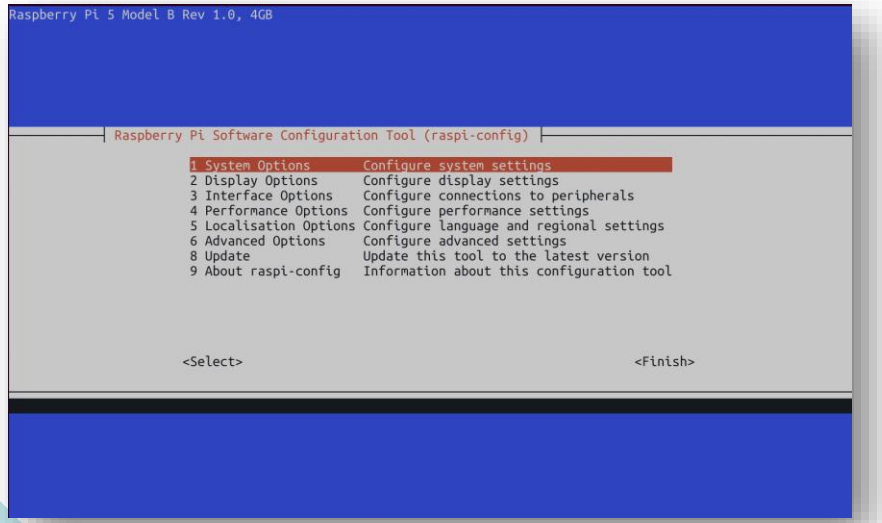
**Note**: SSH는 암호화된 네트워크 프로토콜로, 원격 시스템에 안전하게 로그인하고 명령어를 실행할 수 있고 이를 통해 사용자는 인터넷과 같은 보안되지 않은 네트워크를 통해도 안전하게 서버를 관리

1. 터미널 창을 열어서 아래 명령어를 실행하여 설정창 열기

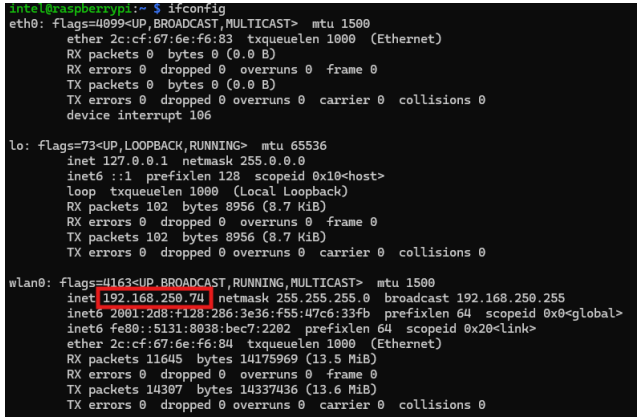
```
$ sudo raspi-config
```

2. 아래 경로로 들어가서 SSH 를 활성화

```
3. Interface Options >> SSH >> Yes
```

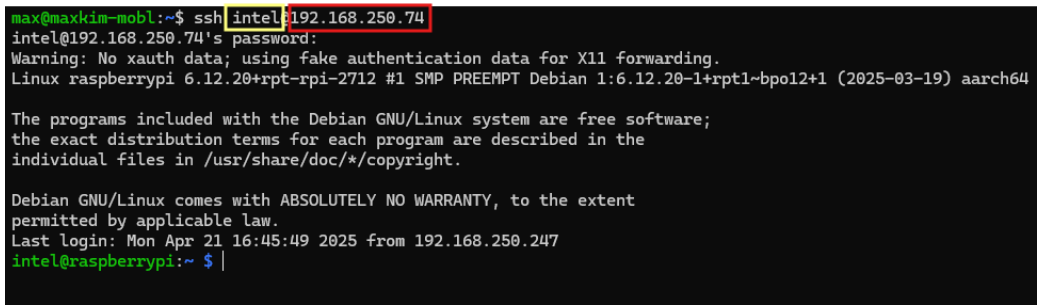



3. 터미널 창에서 ifconfig or ip a 명령어를 실행하여 라즈베리 파이의 IP address 확인




4. 접속할 Ubuntu PC 에서 터미널 창을 열고 아래와 같이 SSH 로 접속

```
$ ssh <라즈베리파이의 user id>@<라즈베리파이의 IP address>
```



# VNC 활성화 (Robot Car 라즈베리파이)

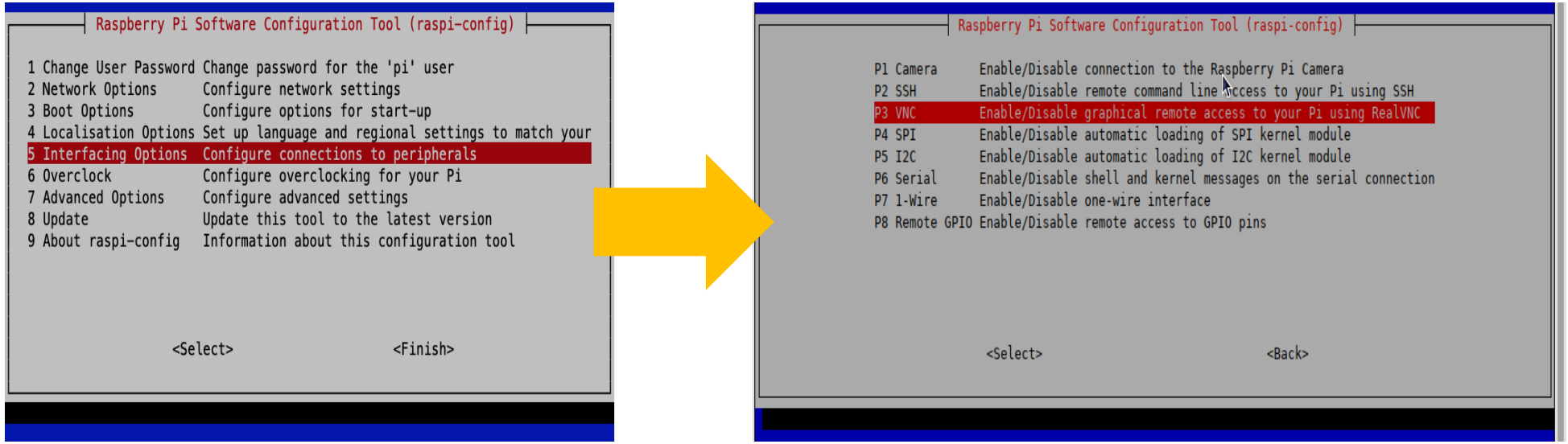
1. Config에서 VNC 활성화

   $ sudo raspi-config

   Interfacing Options -> VNC -> Enable

2. VNC 서버 확인

   $ sudo systemctl status wayvnc



# 라즈베리파이 /etc/udev/rules.d 폴더 밑에 rule 추가

## 99-ttyACM0.rules

```
# Ignore ModemManager for ttyACM0 device
KERNEL=="ttyACM*", SUBSYSTEM=="tty", GROUP="ubuntu", MODE="0777",SYMLINK+="rrc"ATTRS{idVendor}=="1a86",
ATTRS{idProduct}=="55d4", ENV{ID_MM_PORT_IGNORE}="1"
```

angstrong-camera.rules

## ldlidar.rules

```
# set the udev rule , make the device_port be fixed by ldlidar
# CP210x USB Device
KERNEL=="ttyUSB*", ATTRS{idVendor}=="1a86", ATTRS{idProduct}=="7523", MODE:="0777", SYMLINK+="ldlidar"
```

## 99-usb-cam.rules

```
KERNEL=="video*",ATTRS{{idVendor}=="32e6", ATTRS{idProduct}=="9005",MODE:="0777", SYMLINK+="usb_cam"
```

# 라즈베리파이 /etc/udev/rules.d 폴더 밑에 rule 추가 (cont.)

## angstrong-camera.rules

```
# NUWA CAMERA VIDEO
SUBSYSTEMS=="usb", ATTRS{idVendor}=="2dbb", ATTRS{idProduct}=="0300", MODE="0666"
SUBSYSTEMS=="usb", ATTRS{idVendor}=="0c45", ATTRS{idProduct}=="636b", MODE="0666"

# NUWA USB COMMUNICATION PORT
SUBSYSTEMS=="usb", ATTRS{idVendor}=="1a86", ATTRS{idProduct}=="5722", MODE="0666"
SUBSYSTEMS=="usb", ATTRS{idVendor}=="1a86", ATTRS{idProduct}=="5723", MODE="0666"
SUBSYSTEMS=="usb", ATTRS{idVendor}=="1a86", ATTRS{idProduct}=="5724", MODE="0666"
SUBSYSTEMS=="usb", ATTRS{idVendor}=="3482", ATTRS{idProduct}=="5723", MODE="0666"
SUBSYSTEMS=="usb", ATTRS{idVendor}=="3482", ATTRS{idProduct}=="5724", MODE="0666"
SUBSYSTEMS=="usb", ATTRS{idVendor}=="3482", ATTRS{idProduct}=="5725", MODE="0666"
SUBSYSTEMS=="usb", ATTRS{idVendor}=="3482", ATTRS{idProduct}=="5727", MODE="0666"

# VEGA CAMERA VIDEO
SUBSYSTEMS=="usb", ATTRS{idVendor}=="3482", ATTRS{idProduct}=="6789", MODE="0666"
SUBSYSTEMS=="usb", ATTRS{idVendor}=="12d1", ATTRS{idProduct}=="6789", MODE="0666"

# HP60C CAMERA VIDEO
SUBSYSTEMS=="usb", ATTRS{idVendor}=="3482", ATTRS{idProduct}=="6723", MODE="0666"

# KUNLUN CAMERA VIDEO
SUBSYSTEMS=="usb", ATTRS{idVendor}=="0c45", ATTRS{idProduct}=="636d", MODE="0666"
SUBSYSTEMS=="usb", ATTRS{idVendor}=="3482", ATTRS{idProduct}=="7723", MODE="0666"
SUBSYSTEMS=="usb", ATTRS{idVendor}=="3482", ATTRS{idProduct}=="8723", MODE="0666"

# CHANG-A CAMERA VIDEO
SUBSYSTEMS=="usb", ATTRS{idVendor}=="3482", ATTRS{idProduct}=="2789", MODE="0666"
SUBSYSTEMS=="usb", ATTRS{idVendor}=="3482", ATTRS{idProduct}=="278a", MODE="0666"
```

# Docker 설치

[APT 업데이트]

```
sudo apt update && sudo apt upgrade -y
```

[도커 설치]

```
curl -fsSL https://get.docker.com -o get-docker.sh
sudo sh get-docker.sh
rm get-docker.sh
```

[도커 그룹 생성 & 내 계정에 그룹 추가]

```
sudo usermod -aG docker $USER
newgrp docker
sudo reboot
```

# Docker 에 prebuilt ROS2 image load 시키기

## Prebuilt 된 ROS2 docker image 를 Robot Car 용 라즈베리 파이에 다운로드 받자

- ros-humble-export.tar.gz 파일 다운로드 후 gunzip 으로 압축 풀기
  - Option #1: from google drive - link
  - Option #2: from shgusgb PC - scp shgusgb@192.168.100.32:~/ros* ~/Downloads

```
# 아래 gunzip 으로 압축 해제하면 ros-humble-export.tar 파일 생성됨 (size: 11.48GB)
$ gunzip ros-humble-export.tar.gz
```

## 다운로드 후 압축을 해제한 ros-humble-export.tar 파일을 docker image 에 로드하자

- docker image 로 해당 파일을 load

```
$ docker image load -i ros-humble-export.tar
```

- docker image 로드가 완료되면 아래 docker images 명령어로 docker image 확인 (docker image ID 는 각 디바이스별로 다를 수 있음)

```
pi@raspberrypi:~ $ docker images
REPOSITORY   TAG              IMAGE ID       CREATED         SIZE
ros          humble-export    896144ed579c   7 months ago    11.2GB
```

# Docker ROS2 image 자동실행 시키기

## docker run 명령어로 load 된 docker image 자동 실행 시키기

```
docker run -dit \
  --name IntelPi \
  --privileged \
  --restart always \
  --network host \
  -e DISPLAY=:0 \
  -v /dev:/dev \
  -v /tmp/.X11-unix:/tmp/.X11-unix \
  -v /home/max/docker/tmp:/home/ubuntu/shared \
  ros:humble-export \
  tail -f /dev/null
```

- `-d` → 컨테이너를 백그라운드(daemon)로 실행
- `-I` → 표준 입력(터미널 입력)을 계속 열어 둠
- `-t` → TTY(가상 터미널)를 연결해 터미널처럼 쓸 수 있게 함
- `--name IntelPi` → 실행할 컨테이너의 이름을 `IntelPi`로 지정
- `--privileged` → 컨테이너에 모든 커널 기능을 허용합니다. 예: `/dev` 장치 접근, 마운트 등
- `--restart always` → 시스템 재부팅 후에도 자동으로 컨테이너를 재시작
- `--network host` → 컨테이너가 호스트와 동일한 네트워크를 사용
- `-e DISPLAY=:0` → GUI 위한 환경 변수로 `:0`은 호스트의 기본 X 서버를 의미하며, ROS의 `rqt`, `rviz` 같은 GUI 실행에 필요
- `-v /dev` → `/dev` 호스트의 `/dev` 디바이스들을 컨테이너에 그대로 연결해 USB 등의 하드웨어 장치 접근이 가능
- `-v /tmp/.X11-unix:/tmp/.X11-unix` → X11 GUI를 위해 소켓 파일을 공유합니다. GUI 앱을 호스트 화면에 출력할 수 있게 됨
- `-v /home/max/docker/tmp:/home/ubuntu/shared` → 호스트 디렉토리를 컨테이너 내부에 마운트해서 컨테이너 ↔ 호스트 간 파일 공유
- `ros:humble-export` → 사용할 Docker 이미지 이름
- `tail -f /dev/null` → 컨테이너가 꺼지지 않도록 무한 대기 상태 유지

# Docker ROS2 에 shell 연결하기

## docker ps 명령어로 실행중인 docker container 의 ID 확인하기

```
pi@raspberrypi:~ $ docker ps
CONTAINER ID   IMAGE                COMMAND                  CREATED        STATUS          PORTS     NAMES
e71bcb9ab784   ros:humble-export    "tail -f /dev/null"      5 hours ago    Up 24 minutes             IntelPi
```

## docker exec 명령어로 실행중인 docker container 에 zsh 연결하기

```
docker exec \
  -it \
  -u ubuntu \
  -w /home/ubuntu \
  e71bcb9ab784 \
  /bin/zsh
```

- `docker exec`: 실행 중인 컨테이너 안에서 명령어를 실행
- `-i -t (-it)`: 인터랙티브 모드 + TTY. 사용자 입력과 출력을 터미널과 연결
- `-u ubuntu`: 컨테이너 내에서 `ubuntu` 사용자 권한으로 실행
- `-w /home/ubuntu`: 컨테이너 내에서 작업 디렉토리를 `/home/ubuntu` 로 설정
- `e71bcb9ab784`: 대상 컨테이너 ID (또는 이름 사용 가능)
- `/bin/zsh`: 컨테이너 내부에서 실행할 명령어 (기본 `bash`가 아닌 사용자 지정한 `zsh` 쉘)

# zsh vs bash

## zsh는 bash를 더 편리하게 업그레이드한 터미널 쉘

| 항목 | bash | zsh |
|---|---|---|
| 출시 시기 | 1989 | 1990 |
| 자동 완성 | 기본적 | 스마트 자동완성 (옵션, Git 브랜치 등) |
| 스펠 자동 수정 | 없음 | 존재 (sl → ls) |
| 플러그인/테마 | 별도 없음 | oh-my-zsh, powerlevel10k 등 |
| 글자색 강조 | 제한적 | 다양하고 커스터마이징 쉬움 |
| 배열 기능 | 단순 | 더 강력하고 직관적 |
| 경로 단축 | 없음 | cd D/T/B → cd Desktop/Tutorials/Backup |
| Tab 키 기능 | 목록 출력 | 구분된 리스트/설명/네비게이션 가능 |
| 자동로딩 | ~/.bashrc | ~/.zshrc |

# Docker 프로세서들에게 X 서버 권한 주기

## GUI 를 위한 그려주는 일을 해주는 X 서버를 docker 프로세서들이 사용 가능하게 권한 주기

- 라즈베리파이 host 에서 terminal 창을 열어서 아래 명령어 실행
- 만약 ssh 로 아래 명령어를 실행하면 반드시 *export DISPLAY=:0* 로 환경변수 설정 후 실행할 것

```
xhost +local:docker
```

- 로컬 머신 내 사용자 또는 그룹명이 **docker**인 프로세스만 X 서버 접속 권한을 줌

```
xhost -local:docker
```

- 로컬 머신 내 사용자 또는 그룹명이 **docker**인 프로세스만 X 서버 접속 권한을 해제

# Docker ROS2 에 각각 다른 domain ID 부여

## 동일 네트워크 안에서 ROS2 의 ROS_DOMAIN_ID 가 같으면 topic 공유됨

- 다른 팀의 depth camera 를 내가 사용하게 되는 의도치 않은 혼선 가능
- ROS_DOMAIN_ID 를 각 라즈베리파이 장치마다 다르게 가져가서 혼선 방지, 현재 모두 0 으로 설정되어 있음

< 방지법 >

1. docker exec 명령어로 실행중인 docker container 에 zsh 연결하기
2. ~/ros2_ws/.zshrc 파일을 열어서 ROS_DOMAIN_ID 값을 설정

```
# ~/ros2_ws/.zshrc 파일의 맨 아래쪽에 추가, 각 IA address 의 끝자리 사용
# 예: IP address: 192.168.100.91 → export DOS_DOMAIN_ID=1
export DOS_DOMAIN_ID=<사용하는 IP 의 끝자리 숫자>
```

# Depth Camera 동작시켜 보기

## 1. Docker ROS2 의 shell 에 zsh 로 연결

```
docker exec  -it  -u  ubuntu  -w /home/ubuntu  e71bcb9ab784  /bin/zsh
```

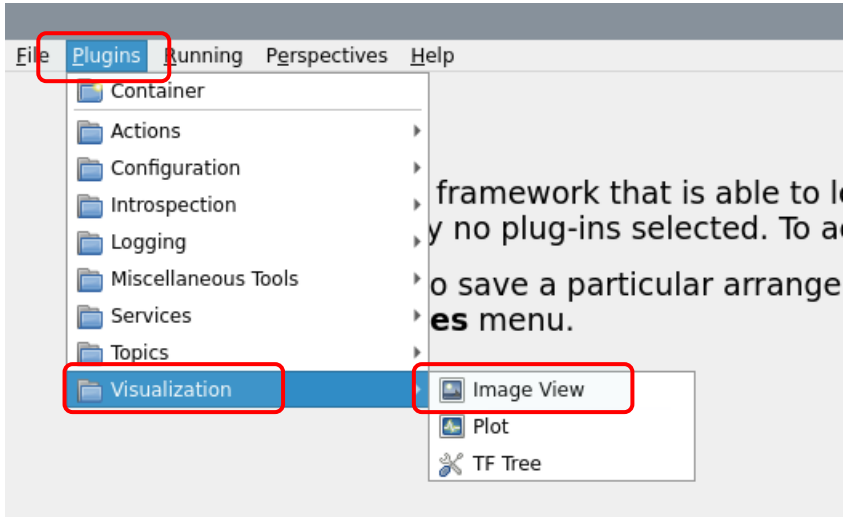
## 2. Depth 카메라 관련 node 들을 launch

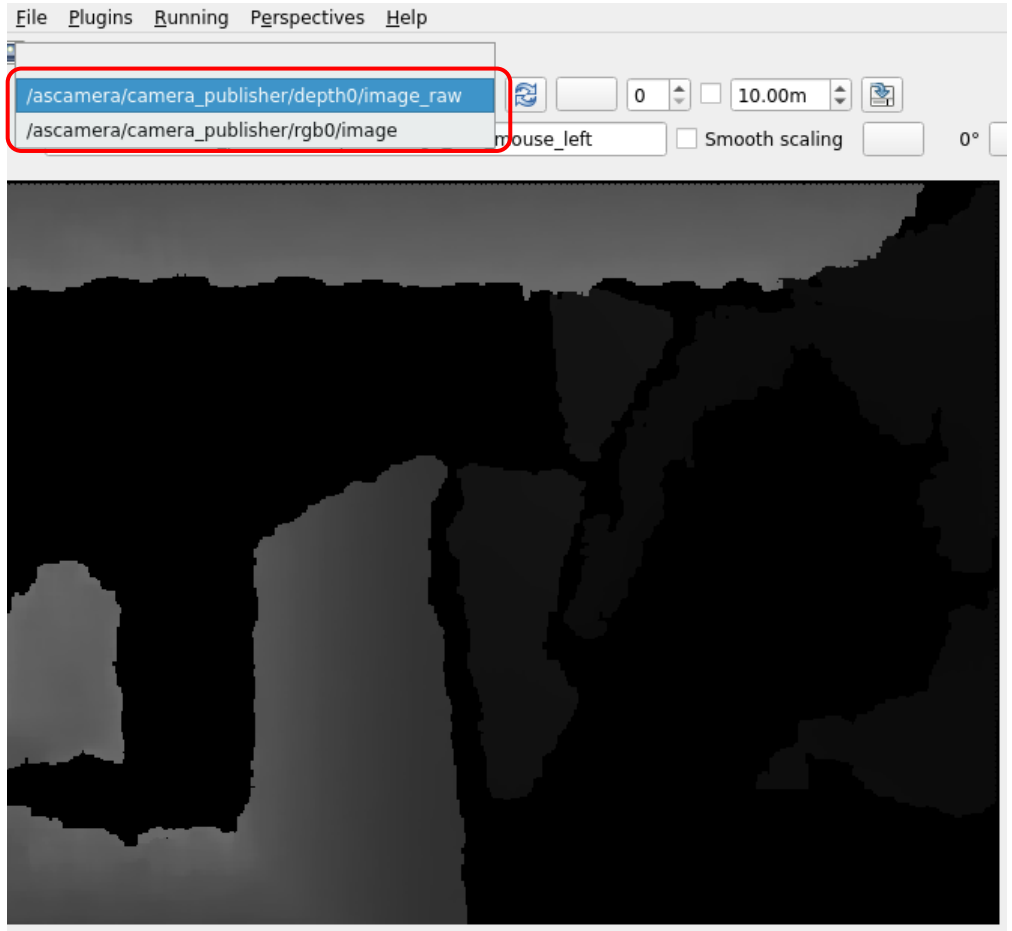
```
ros2 launch peripherals depth_camera.launch.py
```

## 3. Docker ROS2 에 다른 창으로 zsh 연결 후 rqt 실행

```
docker exec  -it  -u  ubuntu  -w /home/ubuntu  e71bcb9ab784  /bin/zsh
# Docker ROS2 console 에서 rqt 실행
rqt
```

# Depth Camera 동작시켜 보기



# OpenCV 로 구현

## 아래 2개의 topic 을 subscribe 하는 Node 를 만들어서 OpenCV 로 preview 구현

- /ascamera/camera_publisher/rgb0/image
- /ascamera/camera_publisher/depth0/image_raw

```python
import rclpy                        # Load ROS2 Python client library
from rclpy.node import Node
from sensor_msgs.msg import Image
from cv_bridge import CvBridge    # ROS Image<->OpenCV Mat conversion utility
import message_filters
import cv2

class DepthRgbFilter(Node): # Define a new ROS node by inheriting from Node
    def __init__(self):
        super().__init__('depth_rgb_filter')
        self.bridge = CvBridge()

        # Subscribe to RGB and depth topics with synchronization
        rgb_sub = message_filters.Subscriber(self, Image, '/ascamera/camera_publisher/rgb0/image')
        depth_sub = message_filters.Subscriber(self, Image, '/ascamera/camera_publisher/depth0/image_raw')

        ts = message_filters.ApproximateTimeSynchronizer([rgb_sub, depth_sub], queue_size=10, slop=0.05)
        ts.registerCallback(self.callback)

    # Callback where the actual image processing happens
    def callback(self, rgb_msg, depth_msg):
        # Convert ROS Image messages to OpenCV format
        rgb = self.bridge.imgmsg_to_cv2(rgb_msg, 'bgr8')
        depth = self.bridge.imgmsg_to_cv2(depth_msg, '16UC1')

        cv2.imshow('RGB Preview', rgb)
        cv2.imshow('Depth Preview', depth)
        cv2.waitKey(1)

def main(args=None):
    rclpy.init(args=args)   # Initialize ROS2 communication
    node = DepthRgbFilter() # Create the node instance
    rclpy.spin(node)        # Enter the callback waiting loop
    node.destroy_node()     # Clean up the node
    rclpy.shutdown()        # Shut down ROS2

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

# OpenCV 로 구현

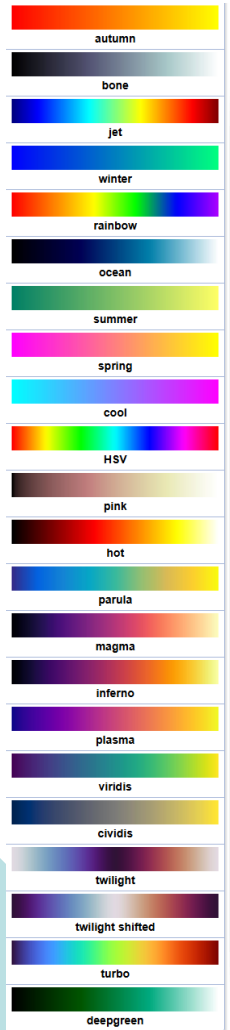
**Preview 화면의 정 가운데 부분에 반지름 5의 원을 그리고 원의 중심점의 depth 정보를 RGB 화면에 cv2.puttext 함수를 사용해 실시간 출력**

```python
import rclpy # Load ROS2 Python client library
from rclpy.node import Node
from sensor_msgs.msg import Image
from cv_bridge import CvBridge # ROS Image<->OpenCV Mat conversion utility
import message_filters
import cv2

class DepthRgbFilter(Node): # Define a new ROS node by inheriting from Node
    def __init__(self):
        super().__init__('depth_rgb_filter')
        self.bridge = CvBridge()

        # Subscribe to RGB and depth topics with synchronization
        rgb_sub = message_filters.Subscriber(self, Image, '/ascamera/camera_publisher/rgb0/image')
        depth_sub = message_filters.Subscriber(self, Image, '/ascamera/camera_publisher/depth0/image_raw')

        ts = message_filters.ApproximateTimeSynchronizer([rgb_sub, depth_sub], queue_size=10, slop=0.05)
        ts.registerCallback(self.callback)

    # Callback where the actual image processing happens
    def callback(self, rgb_msg, depth_msg):
        # Convert ROS Image messages to OpenCV format
        rgb = self.bridge.imgmsg_to_cv2(rgb_msg, 'bgr8')
        depth = self.bridge.imgmsg_to_cv2(depth_msg, '16UC1')

        # Center position of RGB image
        h, w = rgb.shape[:2]
        cx, cy = w // 2, h // 2
        cv2.circle(rgb, center=(cx, cy), radius=6, color=(0, 0, 200), thickness=3)
        depth_val = depth[cy, cx]
        font = cv2.FONT_HERSHEY_SIMPLEX
        cv2.putText(rgb, f"{depth_val} mm", (cx-15, cy-15), font, 1.0, (0, 250, 0), 2, cv2.LINE_AA)

        cv2.imshow('RGB Preview', rgb)
        cv2.imshow('Depth Preview', depth)
        cv2.waitKey(1)

def main(args=None):
    rclpy.init(args=args) # Initialize ROS2 communication
    node = DepthRgbFilter() # Create the node instance
    rclpy.spin(node) # Enter the callback waiting loop
    node.destroy_node() # Clean up the node
    rclpy.shutdown() # Shut down ROS2

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

# OpenCV 로 구현

https://docs.opencv.org/3.4/d3/d50/group__imgproc__colormap.html

## Depth map 의 표현방식을 변경해 보기

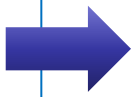



```
enum cv::ColormapTypes {
    cv::COLORMAP_AUTUMN = 0,
    cv::COLORMAP_BONE = 1,
    cv::COLORMAP_JET = 2,
    cv::COLORMAP_WINTER = 3,
    cv::COLORMAP_RAINBOW = 4,
    cv::COLORMAP_OCEAN = 5,
    cv::COLORMAP_SUMMER = 6,
    cv::COLORMAP_SPRING = 7,
    cv::COLORMAP_COOL = 8,
    cv::COLORMAP_HSV = 9,
    cv::COLORMAP_PINK = 10,
    cv::COLORMAP_HOT = 11,
    cv::COLORMAP_PARULA = 12,
    cv::COLORMAP_MAGMA = 13,
    cv::COLORMAP_INFERNO = 14,
    cv::COLORMAP_PLASMA = 15,
    cv::COLORMAP_VIRIDIS = 16,
    cv::COLORMAP_CIVIDIS = 17,
    cv::COLORMAP_TWILIGHT = 18,
    cv::COLORMAP_TWILIGHT_SHIFTED = 19,
    cv::COLORMAP_TURBO = 20,
    cv::COLORMAP_DEEPGREEN = 21
}
GNU Octave/MATLAB equivalent colormaps. More...
```

```python
import rclpy # Load ROS2 Python client library
from rclpy.node import Node
from sensor_msgs.msg import Image
from cv_bridge import CvBridge # ROS Image<->OpenCV Mat conversion utility
import message_filters
import cv2

class DepthRgbFilter(Node): # Define a new ROS node by inheriting from Node
    def __init__(self):
        super().__init__('depth_rgb_filter')
        self.bridge = CvBridge()

        # Subscribe to RGB and depth topics with synchronization
        rgb_sub = message_filters.Subscriber(self, Image, '/ascamera/camera_publisher/rgb0/image')
        depth_sub = message_filters.Subscriber(self, Image, '/ascamera/camera_publisher/depth0/image_raw')

        ts = message_filters.ApproximateTimeSynchronizer([rgb_sub, depth_sub], queue_size=10, slop=0.05)
        ts.registerCallback(self.callback)

    # Callback where the actual image processing happens
    def callback(self, rgb_msg, depth_msg):
        # Convert ROS Image messages to OpenCV format
        rgb = self.bridge.imgmsg_to_cv2(rgb_msg, 'bgr8')
        depth = self.bridge.imgmsg_to_cv2(depth_msg, '16UC1')

        # Apply color map for depth
        color_disp = cv2.applyColorMap(depth.astype('uint8'), cv2.COLORMAP_PLASMA)

        cv2.imshow('RGB Preview', rgb)
        cv2.imshow('Depth Preview', color_disp)
        cv2.waitKey(1)

def main(args=None):
    rclpy.init(args=args) # Initialize ROS2 communication
    node = DepthRgbFilter() # Create the node instance
    rclpy.spin(node) # Enter the callback waiting loop
    node.destroy_node() # Clean up the node
    rclpy.shutdown() # Shut down ROS2
if __name__ == '__main__':
    main()
```

# OpenCV 로 구현

**거리가 1000 mm 보다 멀리 있고 2000 mm 보다 가까이 있는 객체만 RGB 이미지가 preview 로 보이게 하기**



```python
import rclpy # Load ROS2 Python client library
from rclpy.node import Node
from sensor_msgs.msg import Image
from cv_bridge import CvBridge # ROS Image<->OpenCV Mat conversion utility
import message_filters
import cv2
import numpy as np

class DepthRgbFilter(Node): # Define a new ROS node by inheriting from Node
    def __init__(self):
        super().__init__('depth_rgb_filter')
        self.bridge = CvBridge()

        # Subscribe to RGB and depth topics with synchronization
        rgb_sub = message_filters.Subscriber(self, Image, '/ascamera/camera_publisher/rgb0/image')
        depth_sub = message_filters.Subscriber(self, Image, '/ascamera/camera_publisher/depth0/image_raw')

        ts = message_filters.ApproximateTimeSynchronizer([rgb_sub, depth_sub], queue_size=10, slop=0.05)
        ts.registerCallback(self.callback)

    # Callback where the actual image processing happens
    def callback(self, rgb_msg, depth_msg):
        # Convert ROS Image messages to OpenCV format
        rgb = self.bridge.imgmsg_to_cv2(rgb_msg, 'bgr8')
        depth = self.bridge.imgmsg_to_cv2(depth_msg, '16UC1')

        # Create a boolean mask for depth values between 1000 mm and 2000 mm
        mask = (depth >= 1000) & (depth < 2000)

        # Convert boolean mask to uint8 (0 or 255) and apply it to the RGB image
        mask_uint8 = (mask.astype(np.uint8) * 255)
        masked_rgb = cv2.bitwise_and(rgb, rgb, mask=mask_uint8)

        cv2.imshow('RGB Preview', masked_rgb)
        cv2.imshow('Depth Preview', depth)
        cv2.waitKey(1)

def main(args=None):
    rclpy.init(args=args) # Initialize ROS2 communication
    node = DepthRgbFilter() # Create the node instance
    rclpy.spin(node) # Enter the callback waiting loop
    node.destroy_node() # Clean up the node
    rclpy.shutdown() # Shut down ROS2

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

# Python 코드를 ROS2 package 로 관리하기

## 1. Python 용 package 생성하기

```
cd ~/ros2_ws/src
ros2 pkg create --build-type ament_python depth_rgb_filter
```

## 2. 생성된 파일들 확인

```
❯ pwd
/home/ubuntu/ros2_ws/src
❯ tree depth_rgb_filter
depth_rgb_filter
|-- depth_rgb_filter
|   `-- __init__.py
|-- package.xml
|-- resource
|   `-- depth_rgb_filter
|-- setup.cfg
|-- setup.py
`-- test
    |-- test_copyright.py
    |-- test_flake8.py
    `-- test_pep257.py
```

## 3. Python 코드 복사

```
cp 03-preview-depth-range.py \
   ~/ros2_ws/src/depth_rgb_filter/depth_rgb_filter/depth_rgb_filter_node.py
```

# Python 코드를 ROS2 package 로 관리하기 (cont.)

## 4. setup.py

```
from setuptools import setup

package_name = 'depth_rgb_filter'

setup(
    name=package_name,
    version='0.0.0',
    packages=[package_name],
    install_requires=['setuptools'],
    zip_safe=True,
    maintainer='your_name',
    maintainer_email='your@email.com',
    description='Depth + RGB filter node',
    license='Apache License 2.0',
    tests_require=['pytest'],
    entry_points={
        'console_scripts': [
            'depth_rgb_filter = depth_rgb_filter.depth_rgb_filter_node:main'
        ],
    },
)
```

## 5. setup.cfg

```
[develop]
script_dir=$base/lib/depth_rgb_filter
[install]
install_scripts=$base/lib/depth_rgb_filter
```

# Python 코드를 ROS2 package 로 관리하기 (cont.)

## 6. Package 빌드

```
cd ~/ros2_ws
colcon build --packages-select depth_rgb_filter
source install/setup.bash
```

## 7. 실행

```
ros2 run depth_rgb_filter depth_rgb_filter
```

# **THANK** YOU