1. Screenshot 설명
2. 아이콘들 설명
3. 터미널 사용법
4. 자주 사용하는 명령어

sudo apt-get 명령은 주로 Ubuntu나 Debian 같은 리눅스 배포판에서 사용되는 명령입니다. 이 명령은 시스템에 소프트웨어 패키지를 설치, 업데이트, 제거하기 위해 사용됩니다. sudo는 "superuser do"의 약어로, 관리자 권한(또는 루트 사용자 권한)으로 명령을 실행할 때 사용됩니다.

apt-get은 Advanced Packaging Tool (APT)의 일부로, 패키지 관리 도구입니다. 여기서 몇 가지 일반적인 사용 사례를 살펴보겠습니다:

**sudo apt-get install [패키지명]**

* 예를 들어, sudo apt-get install nginx는 Nginx 웹 서버를 설치합니다.
* 패키지 업데이트 및 업그레이드:
  + 패키지 목록 업데이트: 사용 가능한 패키지와 그 버전의 최신 목록을 가져옵니다.

**sudo apt-get update**

* + 설치된 패키지 업그레이드: 시스템에 설치된 모든 패키지를 최신 버전으로 업그레이드합니다.

**sudo apt-get upgrade**

* 패키지 제거: 시스템에서 특정 패키지를 제거합니다.

**sudo apt-get remove [패키지명**]

* 시스템 청소:
  + 더 이상 필요하지 않은 패키지를 자동으로 제거합니다.

**sudo apt-get autoremove**

* + 다운로드된 패키지 파일(.deb)을 삭제하여 디스크 공간을 확보합니다.

**sudo apt-get clean**

sudo apt-get 명령은 시스템 관리 및 유지보수를 위해 자주 사용되며, 시스템을 최신 상태로 유지하고 필요한 소프트웨어를 관리하는 데 중요한 역할을 합니다.

etson@nano:~$ **free -m**

total used free shared buff/cache available

Mem: 1978 1391 138 116 448 370

Swap: **10239**  234 10005

jetson@nano:~$

jetson@nano:~$

jetson@nano:~$ **free -m**

total used free shared buff/cache available

Mem: 1978 1540 146 36 291 302

Swap: **30719**  396 30323

id: jetson

passwd: jetson

jetson@nano:~$  **df -h**

Filesystem Size Used Avail Use% Mounted on

/dev/mmcblk0p1 56G 52G 1,8G 97% /

none 950M 0 950M 0% /dev

tmpfs 990M 92K 990M 1% /dev/shm

tmpfs 198M 12M 186M 7% /run

tmpfs 5,0M 4,0K 5,0M 1% /run/lock

tmpfs 990M 0 990M 0% /sys/fs/cgroup

/dev/loop0 128K 128K 0 100% /snap/bare/5

/dev/loop2 66M 66M 0 100% /snap/gtk-common-themes/1519

/dev/loop1 58M 58M 0 100% /snap/core20/1522

/dev/loop3 49M 49M 0 100% /snap/core18/2406

/dev/loop4 50M 50M 0 100% /snap/core18/2542

/dev/loop5 69M 69M 0 100% /snap/core22/867

/dev/loop6 58M 58M 0 100% /snap/core20/1585

/dev/loop8 198M 198M 0 100% /snap/gnome-3-34-1804/75

/dev/loop7 12M 12M 0 100% /snap/snap-store/963

/dev/loop9 197M 197M 0 100% /snap/gnome-3-34-1804/94

/dev/loop10 41M 41M 0 100% /snap/snapd/16299

/dev/loop11 229M 229M 0 100% /snap/gnome-3-38-2004/88

/dev/loop12 66M 66M 0 100% /snap/gtk-common-themes/1515

/dev/loop13 53M 53M 0 100% /snap/snap-store/559

tmpfs 198M 44K 198M 1% /run/user/1000

# 배터리 절약이 필요한 경우 변경해서 사용하세요.

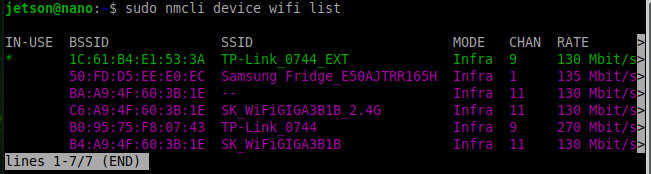
$ sudo nvpmodel -m 1

1.1. Terminal(=SSH)에서 WiFi 연결하기

자율주행하기에 앞서 jetson, notebook을 동일한 AP에 연결해야합니다. 192.168.55.x를 사용하는 USB 연결을 주행에서는 사용할 수 없기 때문입니다. SSH에서 아래 방법으로 AP에 연결할 수 있습니다.

# 실행해야할 명령

$ sudo nmcli device wifi list



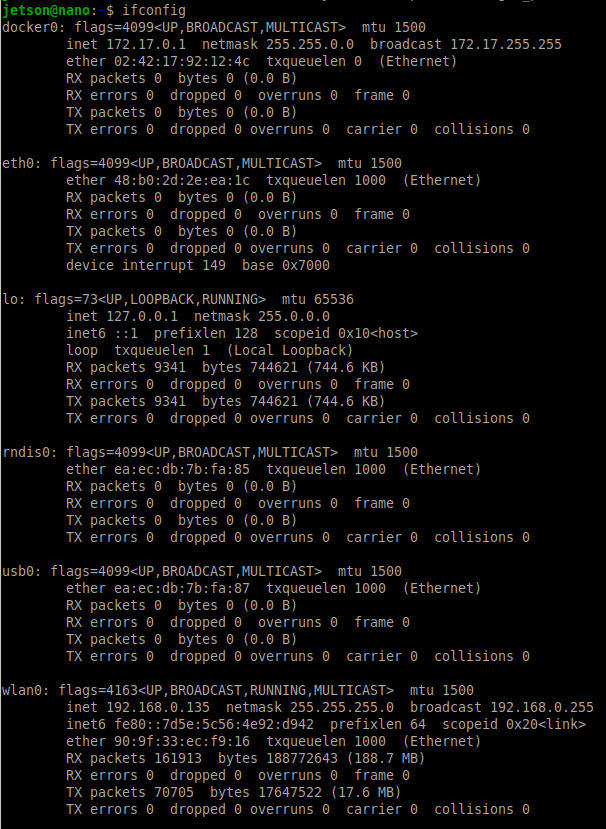
$ sudo nmcli device wifi connect <ssid\_name> password <password>

# password가 있는 경우

jetson@nano:~$ sudo nmcli device wifi connect username password xxxx

jetson@nano:~$ sudo nmcli device wifi connect xxxxxx password yyyyy

$ ifconfig



# password가 없는 경우

jetson@nano:~$ sudo nmcli device wifi connect STARTUP\_LOUNGE\_love

# 연결을 해제할 때

$ sudo nmcli device disconnect wlan0

1.2 Cooling Fan

**sudo sh -c 'echo 128 > /sys/devices/pwm-fan/target\_pwm'**

3선 쿨링팬은 바로 동작합니다만, 4선 PWM 쿨링팬은 아래 작업을 해야 동작합니다. 즉 PWM 신호에 출력을 주어야 동작하는 것입니다. CPU 사용에 따라 쿨링팬 속도를 자동으로 조정하는 utility를 설치하는게 좋습니다.

cd Downloads

git clone https://github.com/jetsonworld/jetson-fan-ctl.git

cd jetson-fan-ctl

sudo sh install.sh

1.3 VNC설치 skip

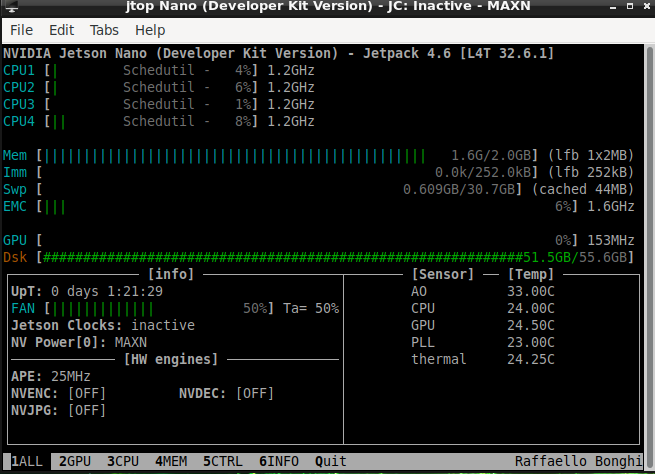
1.4 jtop 사용

설치가 되지 않은 이미지에서는 아래 명령어로 설치할 수 있습니다.

sudo apt install python3-pip

sudo -H pip3 install -U jetson-stats

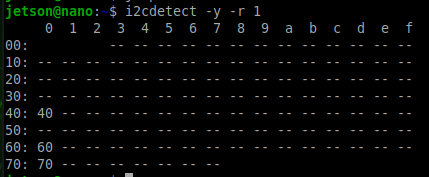
쿨링팬을 돌리기 전에 jtop으로 온도 확인 하고 쿨링팬 돌리고 온도 확인 한다.



1.5 I2C 장치 연결 확인

우리는 pca9685에 납땜을 해 주었기 때문에 60이 찍혀야 정상이다.

jetson@nano:~$ i2cdetect -y -r 1



2. ROS install on PC and Jetson

우분투에서 sftp를 연결하는 방법은 아래 그림을 참고하세요. sftp로 연결하여 Jetson에 있는 파일을 PC에서 쉽게 변경, 편집할 수 있습니다. Jetson Ubuntun에 있는 vi, nano등 툴이 익숙하지 않으면 이 방법을 이용하는 것이 가능합니다. 즉 PC geditor로 Jetson 파일을 수정할 수 있습니다.

2.1 ROS install on Jetson(and PC)

ROS2를 설치할 수 있는 script를 download하여 설치합니다. Jetson에서 아래를 실행해주세요. PC에서 remote 작업을 하려면 PC에도 동일하게 설치해주세요.

jetson@nano:~$ **cd ~/Downloads**

jetson@nano:~/Downloads$ **ls**

**installPiOLED**

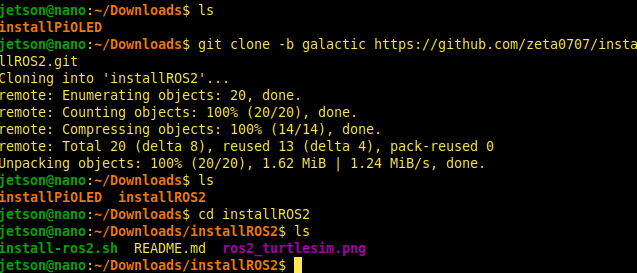
jetson@nano:~/Downloads$ **git clone -b galactic https://github.com/zeta0707/installROS2.git**

jetson@nano:~/Downloads$ ls

**installPiOLED installROS2**

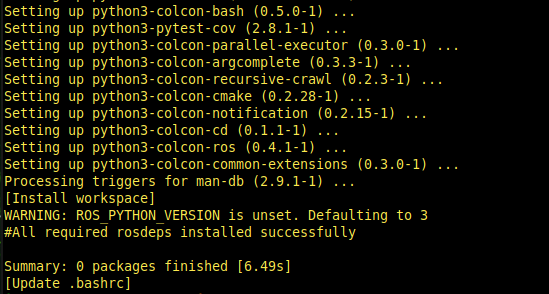
jetson@nano:~/Downloads/installROS2$ ls

install-ros2.sh README.md ros2\_turtlesim.png



jetson@nano:~/Downloads$ cd installROS2

jetson@nano:~/Downloads/installROS2$ ./install-ros2.sh 시간이 많이 걸림 12분



jetson@nano:~$ ls

**Desktop Downloads Pictures ros2\_ws Videos**

**Documents Music Public Templates**

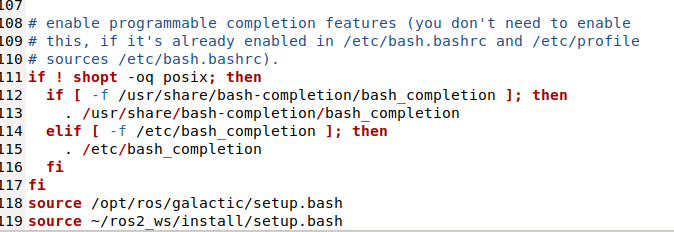
2.2 .bashrc 수정

PC, Jetson에 작업을 편리하게 할 환경을 .bashrc에서 추가해줍니다. alias 명령어추가, PC ↔ Jetson remote 환경을 설정합니다. ROS\_DOMAIN\_ID는 팀원들과 겹치지 않게 선택해주세요.

Jetson

# 또는 편리한 에디터로 .bashrc 열기

jetson@nano:~$ gedit ~/.bashrc



source /opt/ros/galactic/setup.bash

source ~/ros2\_ws/install/setup.bash

# 파일 제일 아래에 다음과 같은 내용 입력

function ccp\_fn(){

rm -rf ~/ros2\_ws/build/"$@" && rm -rf ~/ros2\_ws/install/"$@" ;

}

function ccpa\_fn(){

rm -rf ~/ros2\_ws/build/"$1"\* && rm -rf ~/ros2\_ws/install/"$1"\* ;

}

function cbpa\_fn() {

ls ~/ros2\_ws/src/"$1" | grep "$1" | sed ':a;N;$!ba;s/\n/ /g' | xargs colcon build --packages-select

}

alias ccp=ccp\_fn;

alias ccpa=ccpa\_fn;

alias cbp="colcon build --packages-select"

alias cba="colcon build --symlink-install"

alias cca="rm -rf ~/ros2\_ws/install ~/ros2\_ws/build ~/ros2\_ws/log"

alias cbpa=cbpa\_fn;

alias rt='ros2 topic list'

alias re='ros2 topic echo'

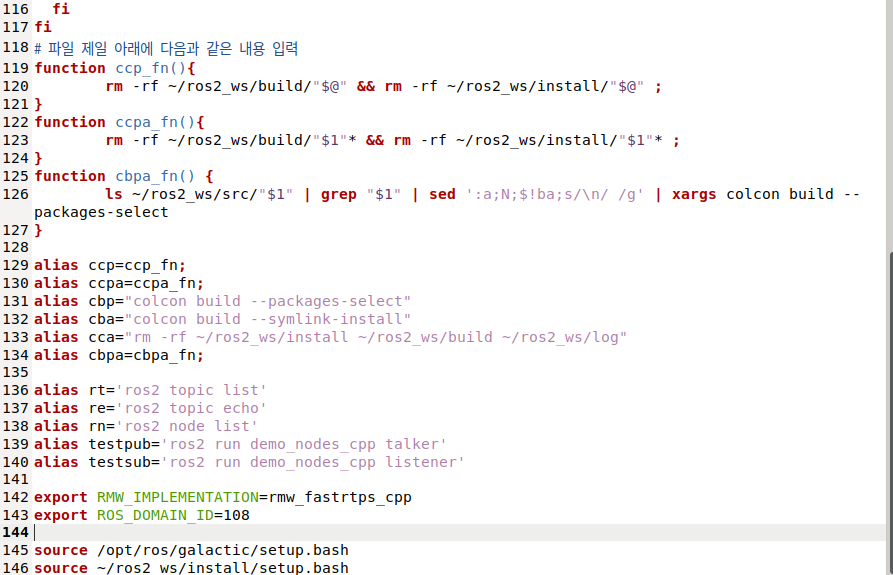
alias rn='ros2 node list'

alias testpub='ros2 run demo\_nodes\_cpp talker'

alias testsub='ros2 run demo\_nodes\_cpp listener'

export RMW\_IMPLEMENTATION=rmw\_fastrtps\_cpp

export ROS\_DOMAIN\_ID=108



에디터 종료 후 terminal에서 .bashrc를 한 번 실행합니다.

jetson@nano:~$ source ~/.bashrc

2.3 ros2 workspace 설치

jetson@nano:~/ros2\_ws$ cca

jetson@nano:~/ros2\_ws$ cba

**Summary: 0 packages finished [9.90s]**

—------------------------------------------------------------- end 0128

2.4 ROS 동작 확인(Option)

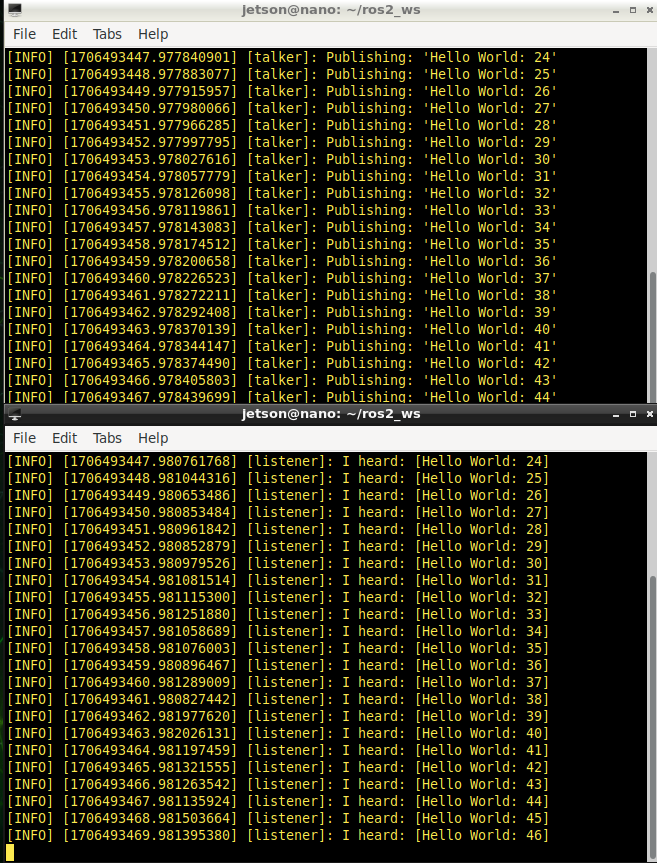
설치 후 제대로 되었는지 demo\_nodes\_cpp로 확인할 수 있습니다. 두개의 창에 하나씩 아래 명령어를 실행해주세요. listener에 ‘Hello World’가 보이면 ROS가 제대로 설치된 것입니다.

# 터미널 #1

jetson@nano:~/ros2\_ws$ ros2 run demo\_nodes\_cpp talker

# 터미널 #2

jetson@nano:~/ros2\_ws$ ros2 run demo\_nodes\_py listener



**3. VS Code SSH extension skip— 나중에 해보시길**

**vscode에서 remote ssh extension를 설치하시면 source 수정 작업이 더 편합니다. 즉 PC vscode를 사용하여 Jetson에 있는 code를 편리하게 수정할 수 있습니다.**

**그 외 여러 extension도 설치해주세요.**

**Highlight Trailing White Spaces**

**EOF Mark**

**CMake**

**C/C++**

**Python**

**4. vim**

**vim을 주 에디터로 사용하는 분은 vim을 설치해주세요.**

**$ sudo apt-get install vim**

**3장, install monicar code**

**1. monicar install on Jetson**

**$ cd ~/ros2\_ws/src**

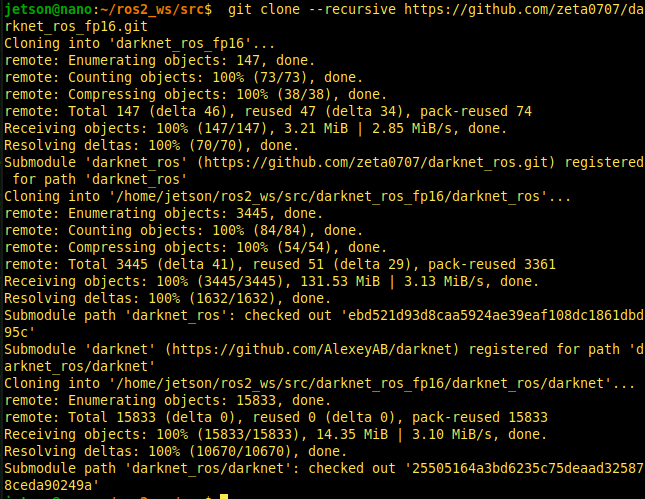
**# camera, teleop, blob\_detect, RCcar setting**

**$ git clone** [**https://github.com/zeta0707/monicar.git**](https://github.com/zeta0707/monicar.git)

****

**# darknet\_ros for yolov4-tiny**

**$ git clone --recursive** [**https://github.com/zeta0707/darknet\_ros\_fp16.git**](https://github.com/zeta0707/darknet_ros_fp16.git)

****

**$ darknet\_ros\_fp16/darknet\_ros/rm\_darknet\_CMakeLists.sh**

**#jetracer로 설정할 경우**

**$ cd ~/ros2\_ws/src/monicar/script**

**$ ./jetRccarParam.sh jetracer**

**# 명령어 사용법**

**./jetRccarParam.sh**

**Usage: ./jetRccarParam.sh target**

**target: select one among these**

**jetracer, jetbot, motorhat2wheel, motorhatSteer, nuriBldc, 298n2Wheel, pca9685Steer**

**우리는 pca9685Steer 입니다.**

**$ cd ~/ros2\_ws/src/monicar/script**

**$ ./jetRccarParam.sh pca9685Steer**

**본인 camera에 맞는 환경파일 만들기 위해 아래 절차를 수행해주세요.**

**우리는 csicam이므로 skip**

**#usbcam로 설정할 경우 실행하세요**

**#default가 csicam이므로 csicam일 경우 실행할 필요가 없습니다.**

**$ cd ~/ros2\_ws/src/monicar/script**

**$ ./camSelect.sh usbcam**

**# 명령어 사용법**

**./camSelect.sh**

**Usage: ./camSelect.sh target**

**target: select one between these**

**csicam, usbcam**

**전체 package빌드 후에 .bashrc를 한 번 더 적용해주세요.**

**# build all packages**

**$ cd ~/ros2\_ws**

**$ cba**

**$ source ~/.bashrc**

**2. monicar install on PC —> skip**

**$ sudo apt update**

**$ sudo apt install ros-galactic-desktop-full**

**3. other ROS package on Jetson**

**$ sudo apt update**

**$ sudo apt install ros-galactic-joy-\* \**

**python3-smbus ros-galactic-ackermann-msgs ros-galactic-image-pipeline python3-pip -y**

**$ pip3 install Adafruit\_PCA9685**

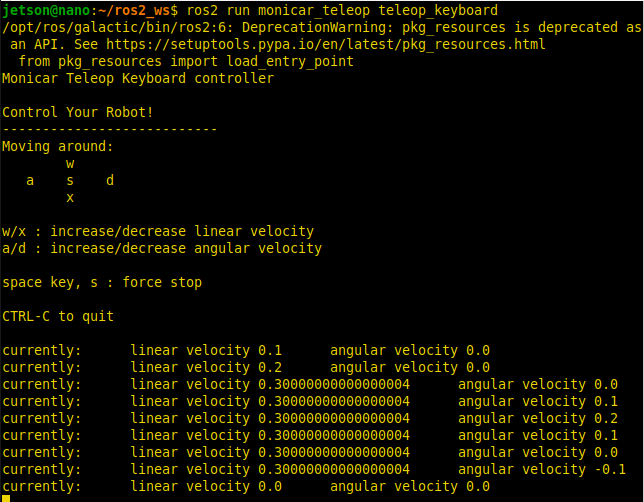
**5장, Teleop by keyboard —-> chapter 4. skip**

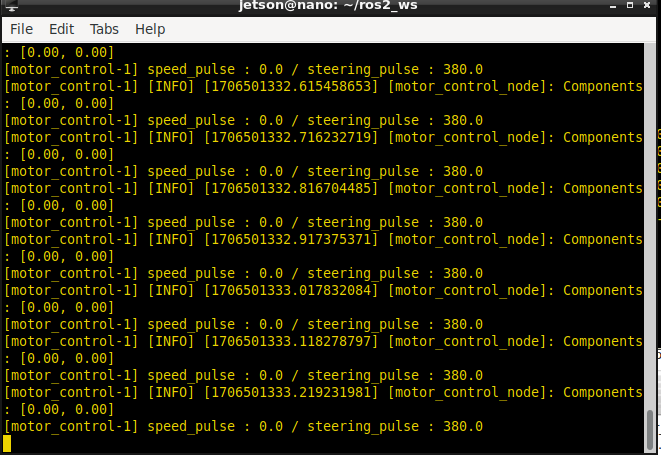
**# terminal #1**

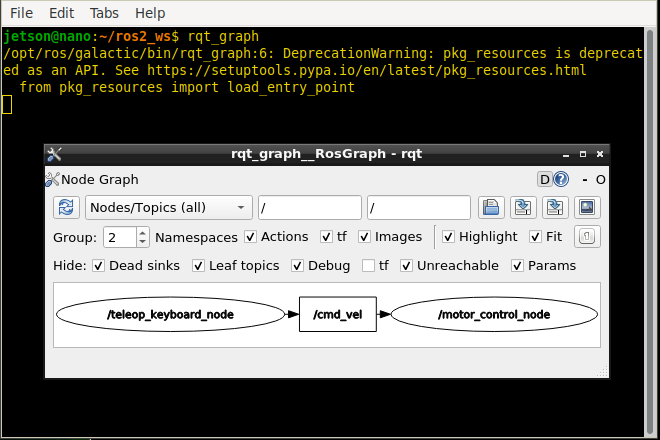
**$ ros2 launch monicar\_control motor.launch.py**

**# terminal #2**

**$ ros2 run monicar\_teleop teleop\_keyboard**

****

****

****

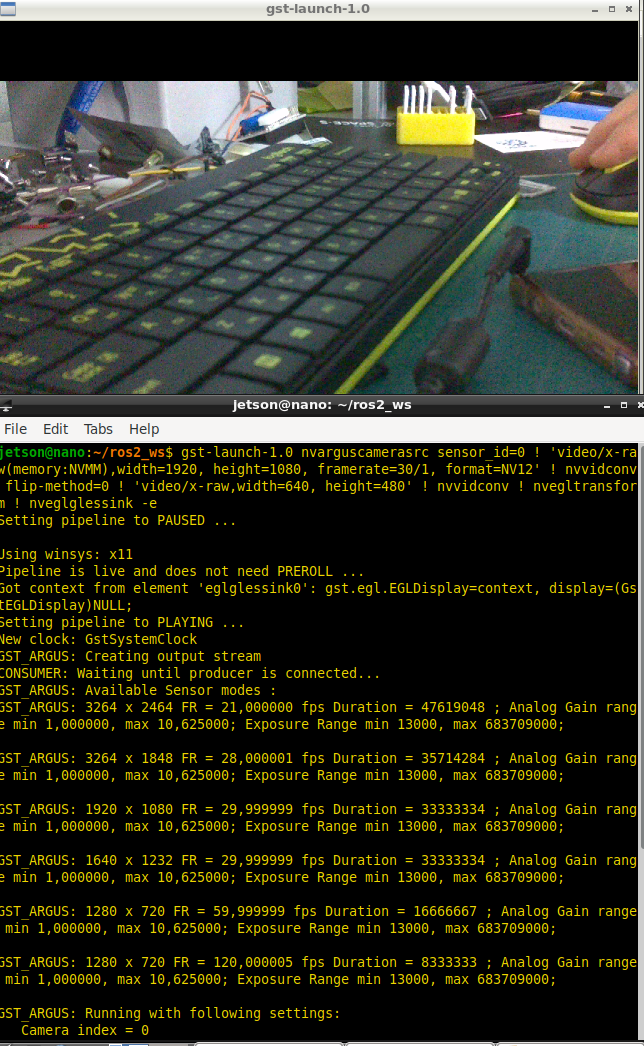
**6장, Verify camera**

1. **Jetson+ TV + Keyboard + mouse**
2. **VNC from PC**
3. **ROS remote 환경(PC + Jetson)**

**1. csicam gst-launch로 확인(Option)**

**camera orientation을 맞게 설정해야합니다. flip-method=0이 jetracer에게 맞는 방향입니다만 camera install에 따라 값을 0 또는 1로 변경해주세요.**

**gst-launch-1.0 nvarguscamerasrc sensor\_id=0 ! 'video/x-raw(memory:NVMM),width=1920, height=1080, framerate=30/1, format=NV12' ! nvvidconv flip-method=0 ! 'video/x-raw,width=640, height=480' ! nvvidconv ! nvegltransform ! nveglglessink -e**

****

**2. launch.py로 확인**

**# terminal #1**

**# CSI camera일 경우**

**$ ros2 launch monicar\_cv csicam.launch.py**

**# USB camera일 경우**

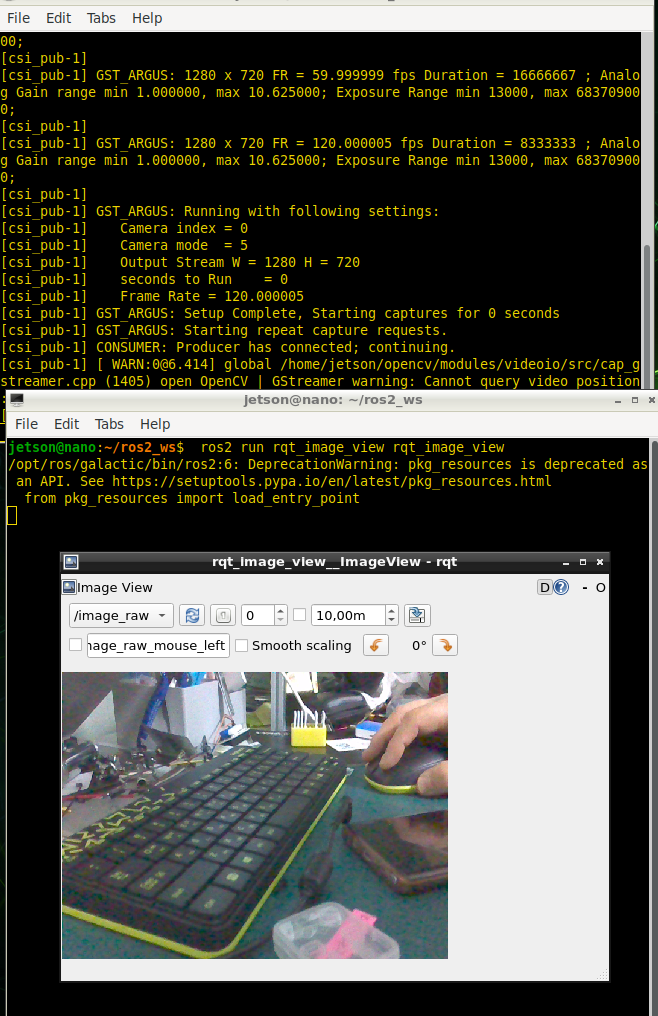
**$ ros2 launch monicar\_cv usbcam.launch.py**

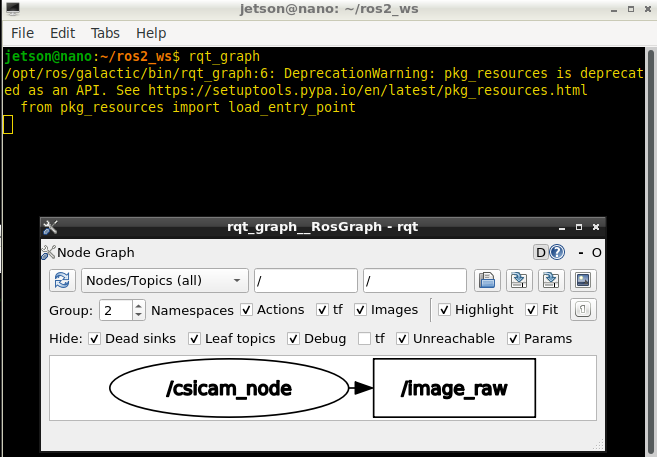
**# terminal #2**

**$ ros2 run rqt\_image\_view rqt\_image\_view**

**# or**

**$ ros2 run image\_view image\_view --ros-args --remap /image:=/image\_raw**

****

****

**2.1 USB camera port 번호 설정**

**jetson@nano:~/ros2\_ws$ ls /dev/video\***

**/dev/video0 /dev/video1 <== usb camera**

**#cvparam.yaml에서 camport 번호 변경 필요**

**$ cd ~/ros2\_ws/src/monicar/monicar\_cv/param**

**# 편한 에디터(vi, nano등)으로 camport 번호를 수정**

**$ vi cvparam.yaml**

**:**

**camport: 1**

**# 다시 빌드해주세요.**

**$ cbpa monicar**

**7장, Blob Tracking**

**Object Following 예제를 시작해봅니다. HSV filter를 통해 단색 물체를 자동차가 따라가는 예제입니다**

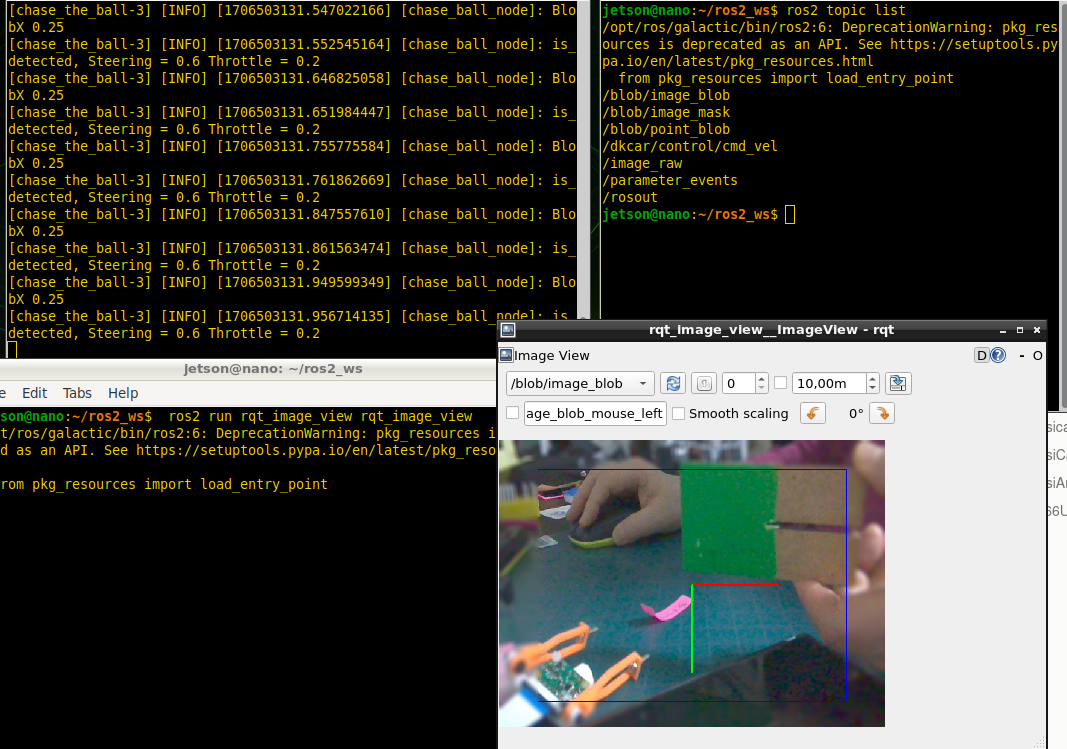
**1. blob\_all launch로 실행**

**# terminal #1**

**$ ros2 launch monicar\_control blob\_all.launch.py**

**# image\_view 실행, 이미지를 확인하기 위함, DISPLAY 필요**

**$ ros2 run rqt\_image\_view rqt\_image\_view**

****

**find\_ball.yaml파일에서 필터 색을 정할 수 있습니다. zeta 이미 몇가지 필터는 만들어 두었습니다.**

**#find\_ball.yaml의 내용**

**define: &blue\_min [55,40,0]**

**define: &blue\_max [150, 255, 255]**

**define: &white\_min [16, 0, 66]**

**define: &white\_max [133, 251, 255]**

**define: &pink\_min [135, 41, 95]**

**define: &pink\_max [255, 196, 255]**

**define: &green\_min [39, 81, 71]**

**define: &green\_max [75, 255, 255]**

**define: &orange\_min [7, 109, 50]**

**define: &orange\_max [76, 218, 234]**

**define: &red\_min [0, 94, 92]**

**define: &red\_max [255, 255, 255]**

**#아래 두 라인으로 필터를 선택합니다.**

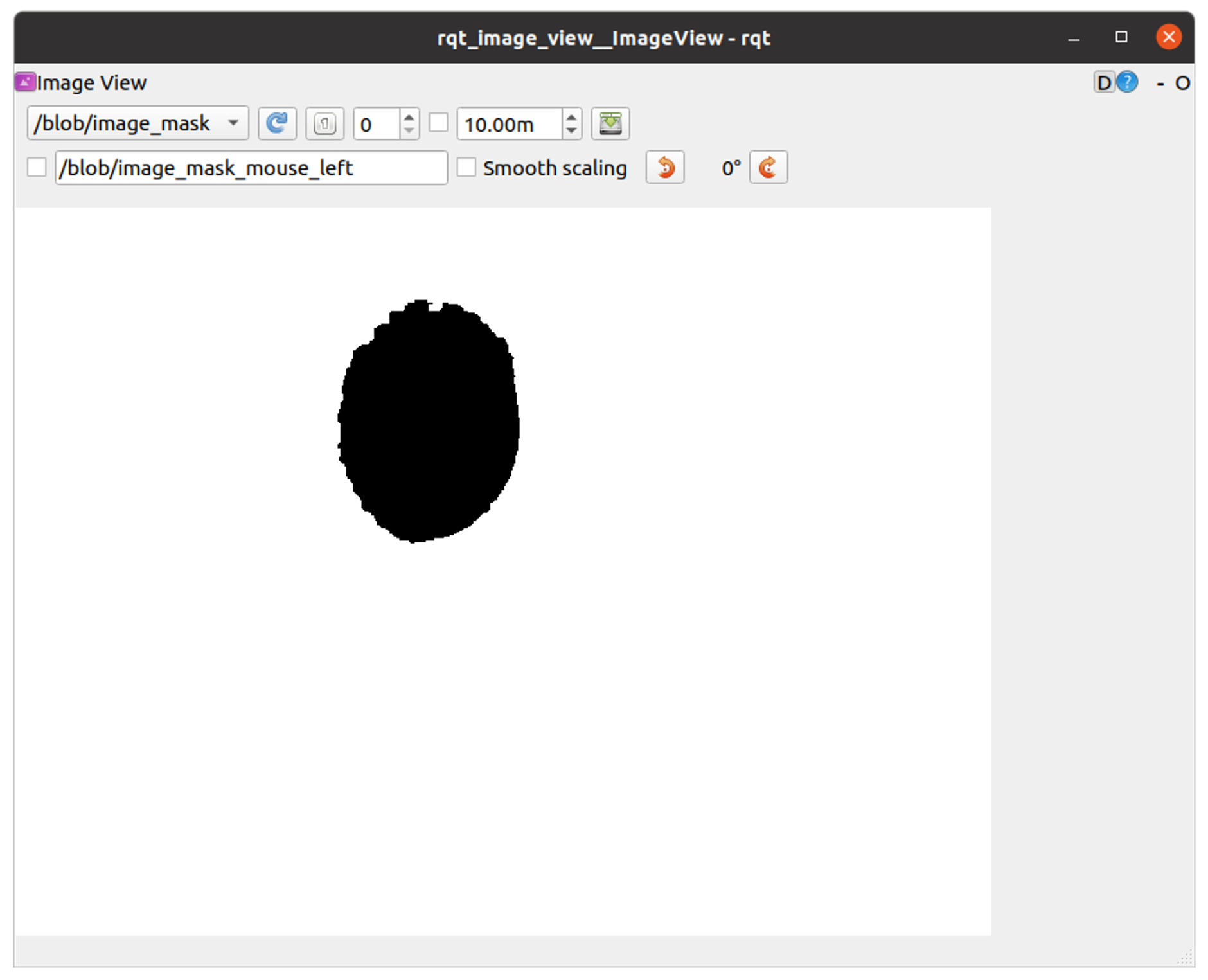
**blob\_detector:**

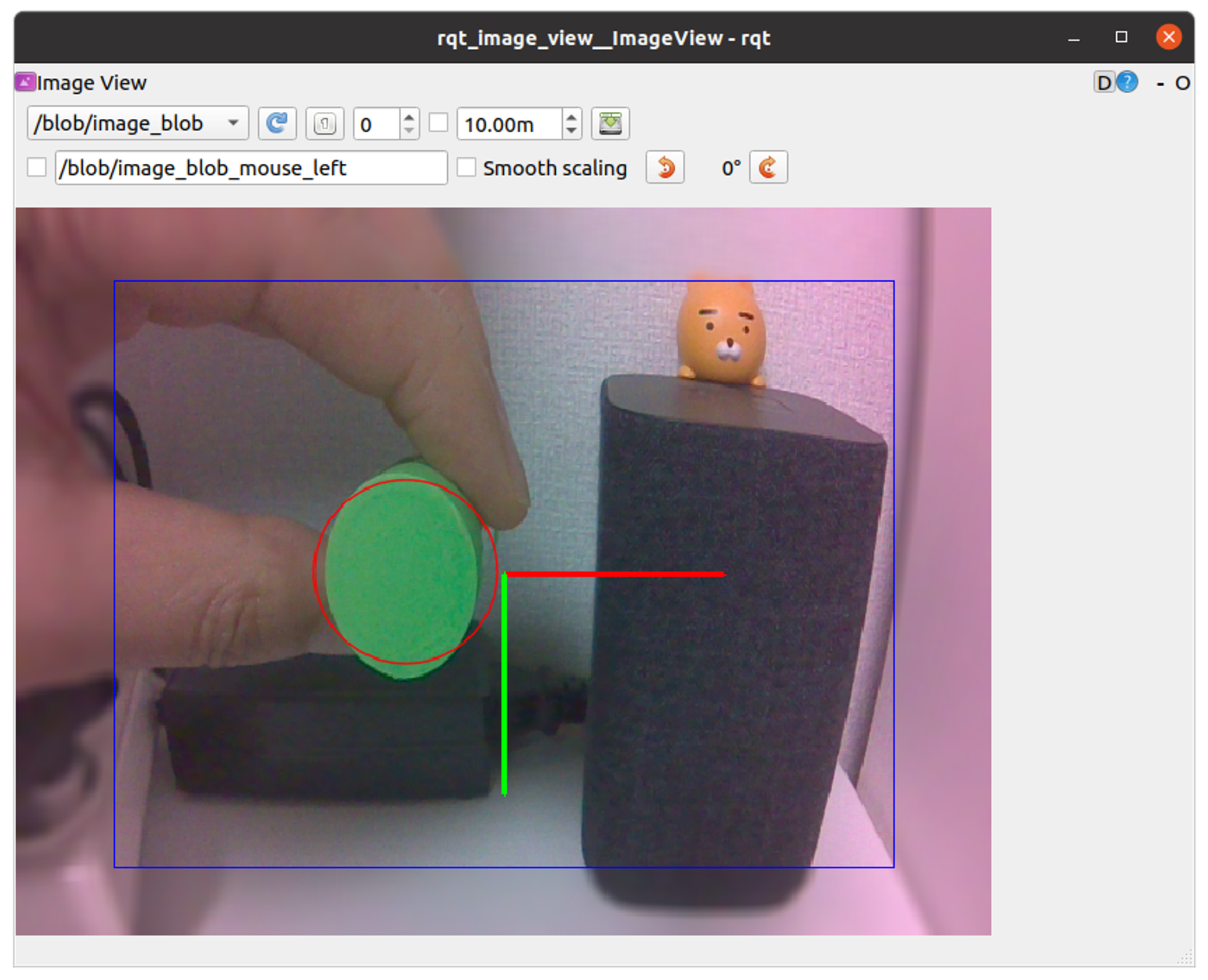
**blob\_min: [39, 81, 71]**

**blob\_max: [75, 255, 255]**

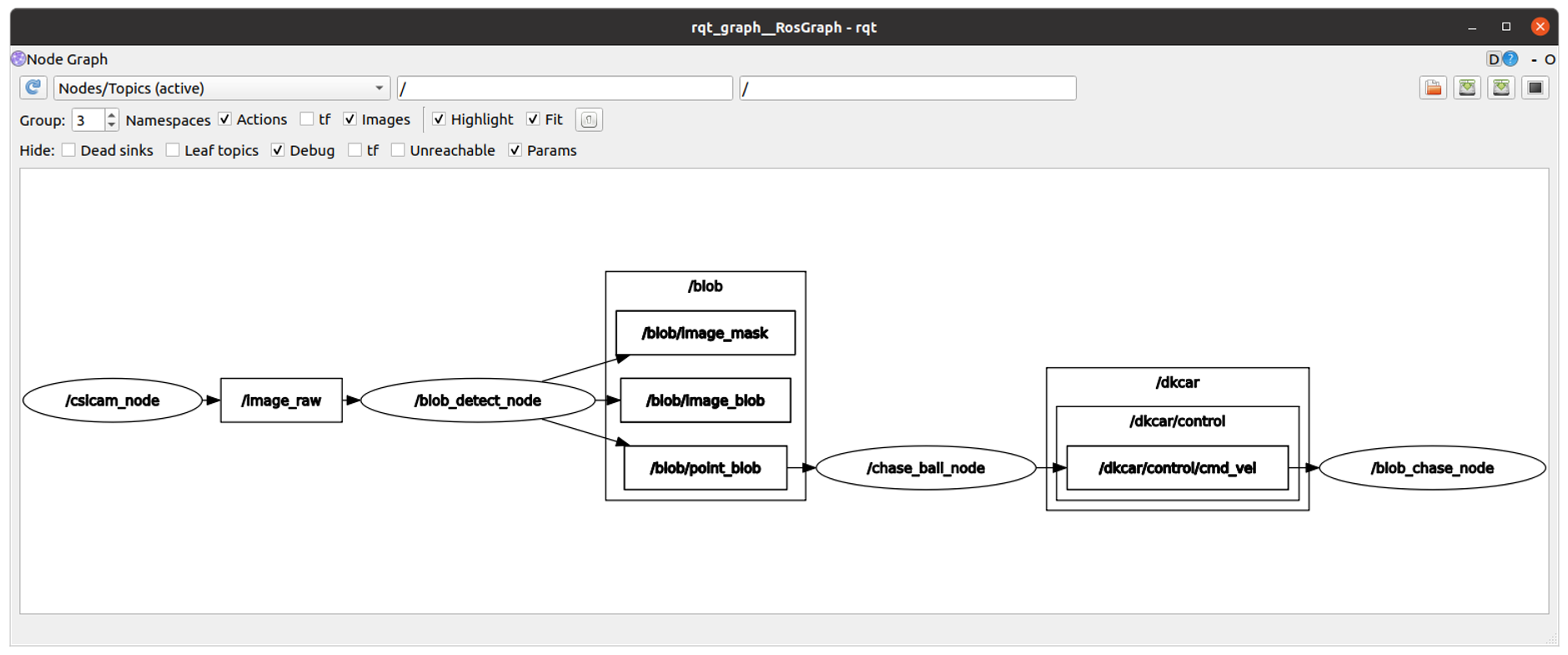
**find\_ball.py에서 "/blob/image\_blob", "/blob/image\_mask", "/blob/point\_blob"를 publish하고 있습니다.**

**아래 그림은 필터를 green으로 설정했을때 image\_mask, image\_blob을 확인해봤습니다.**

****

****

**rqt\_graph로 topic들의 연결을 확인해보면 아래와 같습니다. /blob/pint\_blob이 좌표값에 해당합니다.**

****

**8장, Yolo4-tiny, darknet\_ros**

**이 장에서는 카메라로 물체를 인식하여 어떤 행동을 합니다. 물체 인식후 주행에는 darknet\_ros를 사용합니다. custom dataset 학습에 대해서는 8-1장을 참고하세요**

**1. darknet\_ros**

**AR-Ray Github를 fork했습니다. 주요 수정 내용은 submodule에 있는 darknet\_ros이며, darknet\_ros\_fp16은 거의 그대로 사용합니다. docker관련 내용은 사용하지 않아서 삭제했습니다**

**2. Yolov4-tiny object tracking**

**두 개 launch로 구성되어있으며, 동작은 camera topic publish, yolov4 running, object detect, then publish x,y → car move입니다. yolov4-tiny에 의해 학습된 물체를 tracking 합니다.**

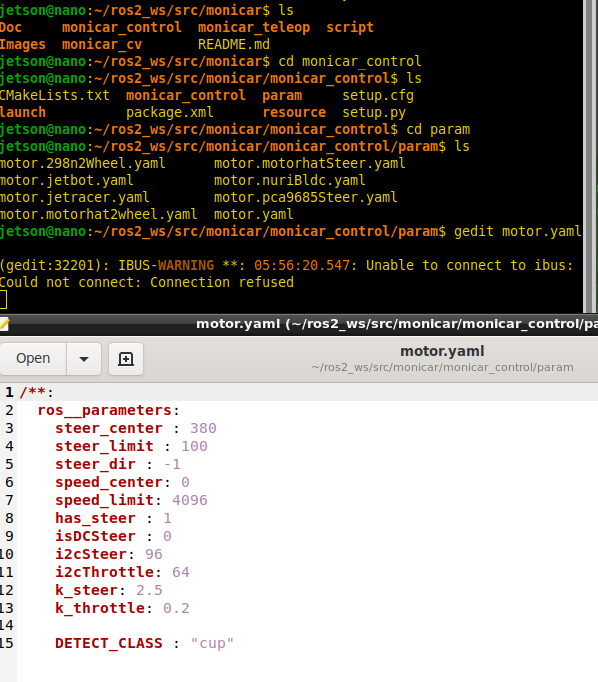
**그리고 coco data중 tracking할 object는 motor.yaml에서 수정할 수 있습니다.**

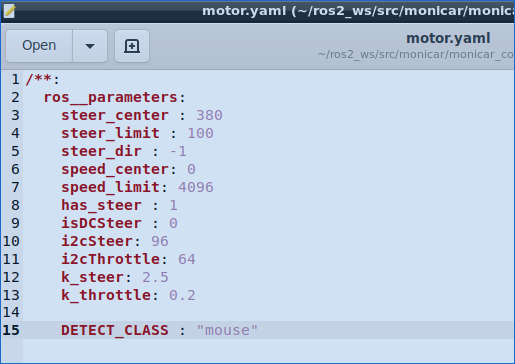
**#위치**

**~/ros2\_ws/src/monicar/monicar\_control/param/motor.yaml**

**#yolo.yaml**

**DETECT\_CLASS : "cup"**

****

****

**#terminal #1**

**#object detect using yolov4-tiny**

**jetson@nano:~/ros2\_ws$ ros2 launch darknet\_ros yolov4-tiny.launch.py**

**#terminal #2, camera publish, object x/y -> car move**

**zeta@zeta-nano:~/catkin\_ws$ ros2 launch monicar\_control yolo\_all.launch.py**

**이 로그들은 ROS 2 (Robot Operating System version 2)를 사용하는 시스템에서 실행된 여러 노드와 프로세스의 상태와 활동을 보여줍니다. 여기에는 YOLO (You Only Look Once)를 사용하여 객체를 탐지하고 관련된 조치를 취하는 시스템의 로그가 포함되어 있습니다. 각 로그 세부 사항을 분석하겠습니다.**

* **로그들을 살펴봅니다.**
* ****
* You
* ```bash
* :~/ros2\_ws$ ros2 launch monicar\_control yolo\_all.launch.py
* [INFO] [launch]: All log files can be found below /home/jetson/.ros/log/2023-07-15-09-13-11-562659-nano-10907
* [INFO] [launch]: Default logging verbosity is set to INFO
* [INFO] [csi\_pub-1]: process started with pid [10909]
* [INFO] [chase\_object\_yolo-2]: process started with pid [10911]
* [INFO] [blob\_chase-3]: process started with pid [10913]
* [blob\_chase-3] [INFO] [1689405194.583385585] [blob\_chase\_node]: Setting Up the Node...
* [blob\_chase-3] hasSteer: 1, i2cSteer: 64, i2cThrottle: 96
* [chase\_object\_yolo-2] [INFO] [1689405194.614365977] [chase\_object\_node]: Setting Up the Node...
* [chase\_object\_yolo-2] k\_steer: 2.5, k\_throttle: 0.2, DETECT\_CLASS: cup
* [blob\_chase-3] [INFO] [1689405194.693279944] [blob\_chase\_node]: Steering Controller Awaked!!
* [blob\_chase-3] Init Throttle ESC
* [chase\_object\_yolo-2] [INFO] [1689405195.186953016] [chase\_object\_node]: Subscriber set
* [chase\_object\_yolo-2] [INFO] [1689405195.202788686] [chase\_object\_node]: Publisher set
* [chase\_object\_yolo-2] [INFO] [1689405195.305869834] [chase\_object\_node]: RUN, Steering = 0.0 Throttle = 0.0
* [chase\_object\_yolo-2] [INFO] [1689405195.405464428] [chase\_object\_node]: RUN, Steering = 0.0 Throttle = 0.0
* [chase\_object\_yolo-2] [INFO] [1689405195.505375485] [chase\_object\_node]: RUN, Steering = 0.0 Throttle = 0.0
* [chase\_object\_yolo-2] [INFO] [1689405195.605568524] [chase\_object\_node]: RUN, Steering = 0.0 Throttle = 0.0
* [chase\_object\_yolo-2] [INFO] [1689405195.705324630] [chase\_object\_node]: RUN, Steering = 0.0 Throttle = 0.0
* [blob\_chase-3] [INFO] [1689405195.717004361] [blob\_chase\_node]: Throttle Controller Awaked!!
* [blob\_chase-3] [INFO] [1689405195.719092721] [blob\_chase\_node]: > Actuators corrrectly initialized
* [blob\_chase-3] [INFO] [1689405195.726205728] [blob\_chase\_node]: > Subscriber corrrectly initialized
* [blob\_chase-3] [INFO] [1689405195.728296015] [blob\_chase\_node]: Initialization complete
* [blob\_chase-3] throttle: 0, steering: 380
* [blob\_chase-3] throttle: 0, steering: 380
* [chase\_object\_yolo-2] [INFO] [1689405195.805593972] [chase\_object\_node]: RUN, Steering = 0.0 Throttle = 0.0
* [blob\_chase-3] 0.0 0.0
* [chase\_object\_yolo-2] [INFO] [1689405195.906115974] [chase\_object\_node]: RUN, Steering = 0.0 Throttle = 0.0
* :
* [chase\_object\_yolo-2] [INFO] [1689405604.979354297] [chase\_object\_node]: RUN, Steering = -0.1 Throttle = 0.2
* [blob\_chase-3] 0.2 -0.1382211538461539
* [chase\_object\_yolo-2] [INFO] [1689405605.013559189] [chase\_object\_node]: object detected: -0.05 0.35
* [blob\_chase-3] throttle: 819, steering: 366
* [chase\_object\_yolo-2] [INFO] [1689405605.072566048] [chase\_object\_node]: BlobX -0.05
* [chase\_object\_yolo-2] [INFO] [1689405605.074582739] [chase\_object\_node]: is \_detected, Steering = -0.1 Throttle = 0.2
* [chase\_object\_yolo-2] [INFO] [1689405605.076904277] [chase\_object\_node]: RUN, Steering = -0.1 Throttle = 0.2
* [blob\_chase-3] 0.2 -0.1322115384615384
* [chase\_object\_yolo-2] [INFO] [1689405605.086201056] [chase\_object\_node]: object detected: -0.06 0.35
* ```
* ```bash
* #terminal #1
* #object detect using Yolo\_v4
* jetson@nano:~/ros2\_ws/src$ ros2 launch darknet\_ros yolov4-tiny.launch.py
* [INFO] [launch]: All log files can be found below /home/jetson/.ros/log/2023-07-15-09-11-21-803955-nano-10478
* [INFO] [launch]: Default logging verbosity is set to INFO
* [INFO] [darknet\_ros-1]: process started with pid [10480]
* [darknet\_ros-1] [INFO] [1689405083.075278542] [darknet\_ros]: [YoloObjectDetector] Node started.
* [darknet\_ros-1] [INFO] [1689405083.078018994] [darknet\_ros]: [YoloObjectDetector] Xserver is not running.
* [darknet\_ros-1] [INFO] [1689405083.078122694] [darknet\_ros]: [YoloObjectDetector] init().
* [darknet\_ros-1] 0 : compute\_capability = 530, cudnn\_half = 0, GPU: NVIDIA Tegra X1
* [darknet\_ros-1] layer filters size/strd(dil) input output
* [darknet\_ros-1] 0 conv 32 3 x 3/ 2 416 x 416 x 3 -> 208 x 208 x 32 0.075 BF
* [darknet\_ros-1] 1 conv 64 3 x 3/ 2 208 x 208 x 32 -> 104 x 104 x 64 0.399 BF
* [darknet\_ros-1] 2 conv 64 3 x 3/ 1 104 x 104 x 64 -> 104 x 104 x 64 0.797 BF
* [darknet\_ros-1] 3 route 2 1/2 -> 104 x 104 x 32
* [darknet\_ros-1] 4 conv 32 3 x 3/ 1 104 x 104 x 32 -> 104 x 104 x 32 0.199 BF
* [darknet\_ros-1] 5 conv 32 3 x 3/ 1 104 x 104 x 32 -> 104 x 104 x 32 0.199 BF
* [darknet\_ros-1] 6 route 5 4 -> 104 x 104 x 64
* [darknet\_ros-1] 7 conv 64 1 x 1/ 1 104 x 104 x 64 -> 104 x 104 x 64 0.089 BF
* [darknet\_ros-1] 8 route 2 7 -> 104 x 104 x 128
* [darknet\_ros-1] 9 max 2x 2/ 2 104 x 104 x 128 -> 52 x 52 x 128 0.001 BF
* [darknet\_ros-1] 10 conv 128 3 x 3/ 1 52 x 52 x 128 -> 52 x 52 x 128 0.797 BF
* [darknet\_ros-1] 11 route 10 1/2 -> 52 x 52 x 64
* [darknet\_ros-1] 12 conv 64 3 x 3/ 1 52 x 52 x 64 -> 52 x 52 x 64 0.199 BF
* [darknet\_ros-1] 13 conv 64 3 x 3/ 1 52 x 52 x 64 -> 52 x 52 x 64 0.199 BF
* [darknet\_ros-1] 14 route 13 12 -> 52 x 52 x 128
* [darknet\_ros-1] 15 conv 128 1 x 1/ 1 52 x 52 x 128 -> 52 x 52 x 128 0.089 BF
* [darknet\_ros-1] 16 route 10 15 -> 52 x 52 x 256
* [darknet\_ros-1] 17 max 2x 2/ 2 52 x 52 x 256 -> 26 x 26 x 256 0.001 BF
* [darknet\_ros-1] 18 conv 256 3 x 3/ 1 26 x 26 x 256 -> 26 x 26 x 256 0.797 BF
* [darknet\_ros-1] 19 route 18 1/2 -> 26 x 26 x 128
* [darknet\_ros-1] 20 conv 128 3 x 3/ 1 26 x 26 x 128 -> 26 x 26 x 128 0.199 BF
* [darknet\_ros-1] 21 conv 128 3 x 3/ 1 26 x 26 x 128 -> 26 x 26 x 128 0.199 BF
* [darknet\_ros-1] 22 route 21 20 -> 26 x 26 x 256
* [darknet\_ros-1] 23 conv 256 1 x 1/ 1 26 x 26 x 256 -> 26 x 26 x 256 0.089 BF
* [darknet\_ros-1] 24 route 18 23 -> 26 x 26 x 512
* [darknet\_ros-1] 25 max 2x 2/ 2 26 x 26 x 512 -> 13 x 13 x 512 0.000 BF
* [darknet\_ros-1] 26 conv 512 3 x 3/ 1 13 x 13 x 512 -> 13 x 13 x 512 0.797 BF
* [darknet\_ros-1] 27 conv 256 1 x 1/ 1 13 x 13 x 512 -> 13 x 13 x 256 0.044 BF
* [darknet\_ros-1] 28 conv 512 3 x 3/ 1 13 x 13 x 256 -> 13 x 13 x 512 0.399 BF
* [darknet\_ros-1] 29 conv 255 1 x 1/ 1 13 x 13 x 512 -> 13 x 13 x 255 0.044 BF
* [darknet\_ros-1] 30 yolo
* [darknet\_ros-1] [yolo] params: iou loss: ciou (4), iou\_norm: 0.07, obj\_norm: 1.00, cls\_norm: 1.00, delta\_norm: 1.00, scale\_x\_y: 1.05
* [darknet\_ros-1] 31 route 27 -> 13 x 13 x 256
* [darknet\_ros-1] 32 conv 128 1 x 1/ 1 13 x 13 x 256 -> 13 x 13 x 128 0.011 BF
* [darknet\_ros-1] 33 upsample 2x 13 x 13 x 128 -> 26 x 26 x 128
* [darknet\_ros-1] 34 route 33 23 -> 26 x 26 x 384
* [darknet\_ros-1] 35 conv 256 3 x 3/ 1 26 x 26 x 384 -> 26 x 26 x 256 1.196 BF
* [darknet\_ros-1] 36 conv 255 1 x 1/ 1 26 x 26 x 256 -> 26 x 26 x 255 0.088 BF
* [darknet\_ros-1] 37 yolo
* [darknet\_ros-1] [yolo] params: iou loss: ciou (4), iou\_norm: 0.07, obj\_norm: 1.00, cls\_norm: 1.00, delta\_norm: 1.00, scale\_x\_y: 1.05
* [darknet\_ros-1] Total BFLOPS 6.910
* [darknet\_ros-1] avg\_outputs = 310203
* [darknet\_ros-1] Allocate additional workspace\_size = 75.50 MB
* [darknet\_ros-1] YOLO V3
* [darknet\_ros-1] Try to load cfg: /home/jetson/ros2\_ws/install/darknet\_ros/share/darknet\_ros/yolo\_network\_config/cfg/yolov4-tiny.cfg, clear = 0
* [darknet\_ros-1] net.optimized\_memory = 0
* [darknet\_ros-1] mini\_batch = 1, batch = 1, time\_steps = 1, train = 1
* [darknet\_ros-1] Create CUDA-stream - 0
* [darknet\_ros-1] Create cudnn-handle 0
* [darknet\_ros-1] nms\_kind: greedynms (1), beta = 0.600000
* [darknet\_ros-1] nms\_kind: greedynms (1), beta = 0.600000
* [darknet\_ros-1] Try to load weights: /home/jetson/ros2\_ws/install/darknet\_ros/share/darknet\_ros/yolo\_network\_config/weights/yolov4-tiny.weights
* [darknet\_ros-1] Loading weights from /home/jetson/ros2\_ws/install/darknet\_ros/share/darknet\_ros/yolo\_network\_config/weights/yolov4-tiny.weights...Done! Loaded 38 layers from weights-file
* :
* :
* # image 인식 후, object 인식 되었을 때
* [darknet\_ros-1] FPS:13.4 : ( MAX:15.1183 MIN:10.114 )
* [darknet\_ros-1] Objects:
* [darknet\_ros-1]
* [darknet\_ros-1] cup: 75% (left\_x: 52 top\_y: 239 width: 266 height: 229)
* **첫 번째 로그 세트 (**ros2 launch monicar\_control yolo\_all.launch.py):
  + 여러 노드(프로세스)가 시작되었습니다: csi\_pub**,** chase\_object\_yolo**,** blob\_chase**.**
  + blob\_chase**와** chase\_object\_yolo **노드는 초기화 과정을 거치며, 주행 및 조향 관련 설정 값들이 로그에 기록되었습니다.**
  + chase\_object\_yolo **노드는** DETECT\_CLASS: cup**으로 설정되어 '컵'을 탐지하도록 설정되었습니다.**
  + **이후 로그는** chase\_object\_yolo**와** blob\_chase **노드가 실행되는 동안의 상태를 보여줍니다. 둘 다 조향과 스로틀 값(예:** Steering = -0.1 Throttle = 0.2**)을 조절하여 로봇이나 자동차의 움직임을 제어합니다.**
* **두 번째 로그 세트 (**ros2 launch darknet\_ros yolov4-tiny.launch.py):
  + YOLOv4-tiny 모델을 사용하여 객체 탐지를 수행하는 darknet\_ros **노드가 시작되었습니다.**
  + **로그는 YOLO 네트워크가 NVIDIA Tegra X1 GPU에서 초기화되는 과정을 보여줍니다.**
  + **다양한 컨볼루션 레이어와 YOLO 레이어의 설정이 나열됩니다.**
  + darknet\_ros **노드는 모델의 가중치를 로드하고, 객체 탐지를 수행합니다.**
* **마지막 로그 세트:**
  + **이 로그는** darknet\_ros **노드가 이미지에서 객체(여기서는 '컵')를 성공적으로 탐지했음을 나타냅니다. '컵' 객체는 75%의 확신도로 탐지되었으며, 위치와 크기가 좌표와 함께 제공됩니다 (**left\_x: 52, top\_y: 239, width: 266, height: 229**).**
  + [darknet\_ros-1] FPS:13.4 : ( MAX:15.1183 MIN:10.114 )**: 이 부분은 YOLO 객체 탐지 알고리즘이 실행되는 프레임 속도(Frames Per Second, FPS)를 나타냅니다. 평균 FPS는 13.4이며, 최대 FPS는 15.1183, 최소 FPS는 10.114입니다. FPS 값은 시스템의 처리 속도와 효율성을 나타내는 중요한 지표로, 높은 FPS는 시스템이 빠르게 이미지를 처리하고 객체를 탐지할 수 있음을 의미합니다.**
  + [darknet\_ros-1] Objects:**: 이 부분은 탐지된 객체에 대한 정보를 시작하는 부분입니다. 로그에서는 구체적으로 어떤 객체가 탐지되었는지 나타내는 내용이 뒤따릅니다. 예를 들어, 이전 로그에서는 '컵'이 75%의 확률로 탐지되었음이 나타났습니다.**

**ros2 launch monicar\_control yolo\_all.launch.py [INFO]**

**[launch]: All log files can be found below /home/jetson/.ros/log/2023-07-15-09-13-11-562659-nano-10907 [INFO]**

**[launch]: Default logging verbosity is set to INFO [INFO]**

**csi\_pub-1]: process started with pid [10909] [INFO]**

**[chase\_object\_yolo-2]: process started with pid [10911] [INFO]**

**[blob\_chase-3]: process started with pid [10913] [blob\_chase-3] [INFO] [1689405194.583385585] [blob\_chase\_node]: Setting Up the Node...**

**[blob\_chase-3] hasSteer: 1, i2cSteer: 64, i2cThrottle: 96**

**[chase\_object\_yolo-2] [INFO] [1689405194.614365977]**

**[chase\_object\_node]: Setting Up the Node...**

**[chase\_object\_yolo-2] k\_steer: 2.5, k\_throttle: 0.2, DETECT\_CLASS: cup**

**[blob\_chase-3] [INFO] [1689405194.693279944]**

**[blob\_chase\_node]: Steering Controller Awaked!!**

**[blob\_chase-3] Init Throttle ESC [chase\_object\_yolo-2] [INFO] [1689405195.186953016] [chase\_object\_node]: Subscriber set**

**[chase\_object\_yolo-2] [INFO] [1689405195.202788686]**

**[chase\_object\_node]: Publisher set [chase\_object\_yolo-2] [INFO] [1689405195.305869834] [chase\_object\_node]: RUN, Steering = 0.0 Throttle = 0.0**

**[chase\_object\_yolo-2] [INFO] [1689405195.405464428]**

**[chase\_object\_node]: RUN, Steering = 0.0 Throttle = 0.0**

**[chase\_object\_yolo-2] [INFO] [1689405195.505375485]**

**[chase\_object\_node]: RUN, Steering = 0.0 Throttle = 0.0**

**[chase\_object\_yolo-2] [INFO] [1689405195.605568524]**

**[chase\_object\_node]: RUN, Steering = 0.0 Throttle = 0.0**

**[chase\_object\_yolo-2] [INFO] [1689405195.705324630]**

**[chase\_object\_node]: RUN, Steering = 0.0 Throttle = 0.0**

**[blob\_chase-3] [INFO] [1689405195.717004361]**

**[blob\_chase\_node]: Throttle Controller Awaked!!**

**[blob\_chase-3] [INFO] [1689405195.719092721]**

**[blob\_chase\_node]: > Actuators corrrectly initialized**

**[blob\_chase-3] [INFO] [1689405195.726205728]**

**[blob\_chase\_node]: > Subscriber corrrectly initialized**

**[blob\_chase-3] [INFO] [1689405195.728296015]**

**[blob\_chase\_node]: Initialization complete**

**[blob\_chase-3] throttle: 0, steering: 380**

**[blob\_chase-3] throttle: 0, steering: 380**

**[chase\_object\_yolo-2] [INFO] [1689405195.805593972]**

**[chase\_object\_node]: RUN, Steering = 0.0 Throttle = 0.0**

**[blob\_chase-3] 0.0 0.0**

**[chase\_object\_yolo-2] [INFO] [1689405195.906115974]**

**[chase\_object\_node]: RUN, Steering = 0.0 Throttle = 0.0 :**

**[chase\_object\_yolo-2] [INFO] [1689405604.979354297]**

**[chase\_object\_node]: RUN, Steering = -0.1 Throttle = 0.2**

**[blob\_chase-3] 0.2 -0.1382211538461539**

**[chase\_object\_yolo-2] [INFO] [1689405605.013559189]**

**[chase\_object\_node]: object detected: -0.05 0.35**

**[blob\_chase-3] throttle: 819, steering: 366**

**[chase\_object\_yolo-2] [INFO] [1689405605.072566048]**

**[chase\_object\_node]: BlobX -0.05**

**[chase\_object\_yolo-2] [INFO] [1689405605.074582739]**

**[chase\_object\_node]: is \_detected, Steering = -0.1 Throttle = 0.2**

**[chase\_object\_yolo-2] [INFO] [1689405605.076904277]**

**[chase\_object\_node]: RUN, Steering = -0.1 Throttle = 0.2**

**[blob\_chase-3] 0.2 -0.1322115384615384**

**[chase\_object\_yolo-2] [INFO] [1689405605.086201056]**

**[chase\_object\_node]: object detected: -0.06**

**이 로그는 ROS 2를 사용하는 모노카(moniCAR) 제어 시스템에서 실행된 프로세스들의 활동을 나타냅니다. 여기에는 객체 추적과 관련된 노드(node)의 상태 및 활동이 포함되어 있습니다. 각 부분을 분석해보겠습니다.**

* **노드 시작과 초기화:**
  + csi\_pub**,** chase\_object\_yolo**,** blob\_chase **세 개의 노드가 시작되었습니다. 각각의 PID(Process ID)가 할당되었습니다.**
  + blob\_chase\_node**는 조향과 스로틀 제어를 위한 노드로 보이며, 초기화 과정을 거칩니다.** hasSteer**,** i2cSteer**,** i2cThrottle **등의 파라미터가 설정됩니다.**
  + chase\_object\_yolo **노드는 YOLO(You Only Look Once) 알고리즘을 사용하여 객체(여기서는 '컵')를 추적하는 노드입니다.** k\_steer**,** k\_throttle**,** DETECT\_CLASS **등의 설정 값이 나타나 있습니다.**
* **노드 실행:**
  + chase\_object\_yolo **노드는 조향(Steering)과 스로틀(Throttle) 값을 0으로 설정하고 실행됩니다. 이는 아마도 대기 상태에서 시작하는 것을 의미할 것입니다.**
  + blob\_chase **노드 역시 조향과 스로틀 제어를 위한 준비를 마치고 초기화를 완료합니다.**
* **객체 탐지와 추적:**
  + chase\_object\_yolo **노드에서 객체가 탐지되었을 때, 조향과 스로틀 값이 조정됩니다 (**Steering = -0.1, Throttle = 0.2**). 이는 탐지된 객체에 따라 차량이나 로봇의 움직임을 조정하기 위한 것입니다.**
  + **탐지된 객체의 위치(**object detected: -0.05 0.35**)가 로그에 기록됩니다. 이 위치 값은 객체의 화면 내 위치를 나타냅니다.**
  + blob\_chase **노드에서는** chase\_object\_yolo **노드로부터 받은 데이터를 기반으로 스로틀과 조향 값을 조정합니다.**

**이 로그는 ROS 2 기반 시스템에서 실시간으로 객체를 탐지하고 추적하는 과정을 보여줍니다. 이 시스템은 탐지된 객체에 따라 동적으로 조향과 스로틀을 조절하여, 자율 주행 차량, 로봇 또는 기타 자동화된 시스템의 움직임을 제어할 수 있습니다.**