

방송용 웹캠을 이용한 실시간 yolo 객체 검출 ros2 패키지

김민규

한국철도기술연구원

프로젝트 주소

Profrog/ros2yolo\_python (github.com) : 본 모듈

Profrog/yolo\_scan\_algorithm (github.com) : 서비스 코드

## 초록

Deeplearning 을 이용한 object detection 에는 여러 종류가 있으나 그 중 yolo 가 속도 및 정확성 양면에서 상호 절충 가능한 성능을 가지고 있어서 보편적인 사용을 하기에 좋다. 해당 프로젝트의 목적은 일반적인 상황에서 object 검출 + 임베디드 연계 과제 및 실험을 위한 접근의 진입점을 낮추는 통합 모듈을 만드는 것이라 해당 model 을 선택하게 되었다.

Ros2 를 이용하면 복잡한 프로그램을 package 화 하여서 node 간 message 를 주고받는 식으로 시스템 구성을 할 수가 있으며 본 프로젝트에서 ros2 node 를 이용하여 yolo 검출 데이터를 publish 해 줌으로써 ros2 안에서 해당 제작 모듈이 쉽게 포함될 수 있게 구성하였다

해당 보고서에서는 webcam 에서 받은 image 데이터가 yolo 를 거쳐서 detect 되는 과정 detect 된 데이터를 ros2 에 publish 하는 과정, 추가 응용 예시로써 algorithm 을 적용하였을 때의 자료를 실제 사용법과 덧붙여 설명하도록 하겠다.

*keyword:* yolo, ros2

## 목차

### 제 1 장 서론(p.4)

#### 1.1 전체 흐름도 및 설명(p.4)

#### 1.2 환경 세팅 및 패키지 사용(p.5)

### 제 2 장 Webcam 제어(p.13)

#### 2.1 Webcam 제어 python 프로그래밍(p.13)

### 제 3 장 Yolo(p.15)

#### 3.1 Yolo 응용 python 프로그래밍(p.15)

### 제 4 장 Ros2(p.23)

#### 4.1 Ros2 응용 python 프로그래밍(p.23)

### 제 5 장 통합응용(p.30)

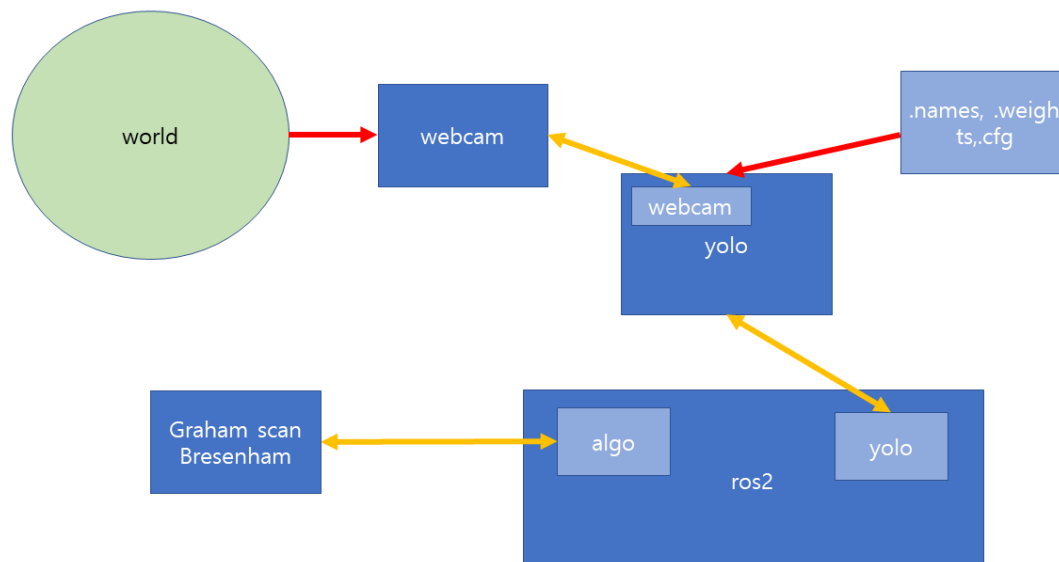
#### 5.1 Algorithm 을 적용하여 yolo 검출 객체를 2D 맵핑 프로그래밍(p.30)

### 제 6 장 결론(p.36)

#### 6.1 ros2yolo 에 대한 결론(p.36)

## 제 1 장 서론

## 1.1 전체 흐름도 및 설명



전체적인 흐름은 webcam 제어 코드를 python 으로 작성하여 webcam 으로 들어오게 되는 실시간 영상을 호출시기 마다 capture 하여서 image 를 yolo 를 사용하여 검출하고 검출데이터 값(x, y, w, h: 물체의 시작 좌표 및 가로 세로 크기)와 원본데이터를 publish 하는 패키지이다. 추가적으로는 grahamscan + bresenham algorithm 을 적용시킨 node 를 추가함으로써 bash 를 이용한 알고리즘 적용 패키지 버전도 해당 github 주소에 새 branch 를 파서 올려두었다.

가상환경 모듈과의 연동은 아직 구현하지 않았으나 만약 unity 를 이용한다면 unity 와 연동할 수 있는 node 를 ros2 패키지에 추가하여 검출되는 객체들을 가상환경으로 옮기는 시스템도 부가적으로 시도해 볼 수 있다고 생각한다.

실제환경에서의 사용은 현재 모듈은 개선 및 수정 작업이 필요한 단계의 패키지라 사용이 제한적이지만 webcam 에 인식되는 객체의 좌표를 얻어와 객체간의 거리관계 객체간의 조합 분석 혹은 생물체의 tracking 및 mapping 에도 적용될 수 있다.

만약, 물체 간의 거리가 세밀한 단계로 정확하게 출력되어야 한다면 본 패키지의 구성은 거리 정확성이 낮을 수 있으므로 레이더 등의 추가 센서를 이용하여 보정이 필요하다.

## 1.2 환경 세팅 및 패키지 사용

해당 모듈을 사용하기 위해서는 ubuntu 기준으로 ros2 와 yolo 가 모두 setting 되어 있어야 하며, 필자의 시스템 환경 기준은 다음과 같다.

```
### about yolo
```

```
yolo version : yolov4
```

```
notebook : HP-Pavilion-Gaming-Laptop-16-a0xxx  
process : Intel® Core™ i7-10750H CPU @ 2.60GHz × 12  
gpu1 : NVIDIA Corporation TU116M [GeForce GTX 1660 Ti Mobile]  
gpu2 : Intel Corporation UHD Graphics  
cuda: 11.2  
cuDNN : v8.1.1  
NVIDIA-SMI 460.39  
Driver Version: 460.39  
opencv :4.4.0  
python :3.8.5  
cmake : 3.20.0  
os : Ubuntu 20.04.2 LTS x86_64
```

```
### about ros2
```

```
ros2 version : foxy
```

ros2 foxy 설치는 해당 사이트를 참조하면 된다

<https://docs.ros.org/en/foxy/Installation.html>

yolov4 설치는 해당 사이트를 참조하면 된다.

[AlexeyAB/darknet: YOLOv4 / Scaled-YOLOv4 / YOLO - Neural Networks for Object Detection](#)

[\(Windows and Linux version of Darknet \) \(github.com\)](#)

그 이후부터 ros2yolo package 를 준비하기 까지의 setting 은

[ros2yolo\\_python/README\(KOREAN\).md at main · Profrog/ros2yolo\\_python \(github.com\)](#)

해당 내용을 참조바란다.

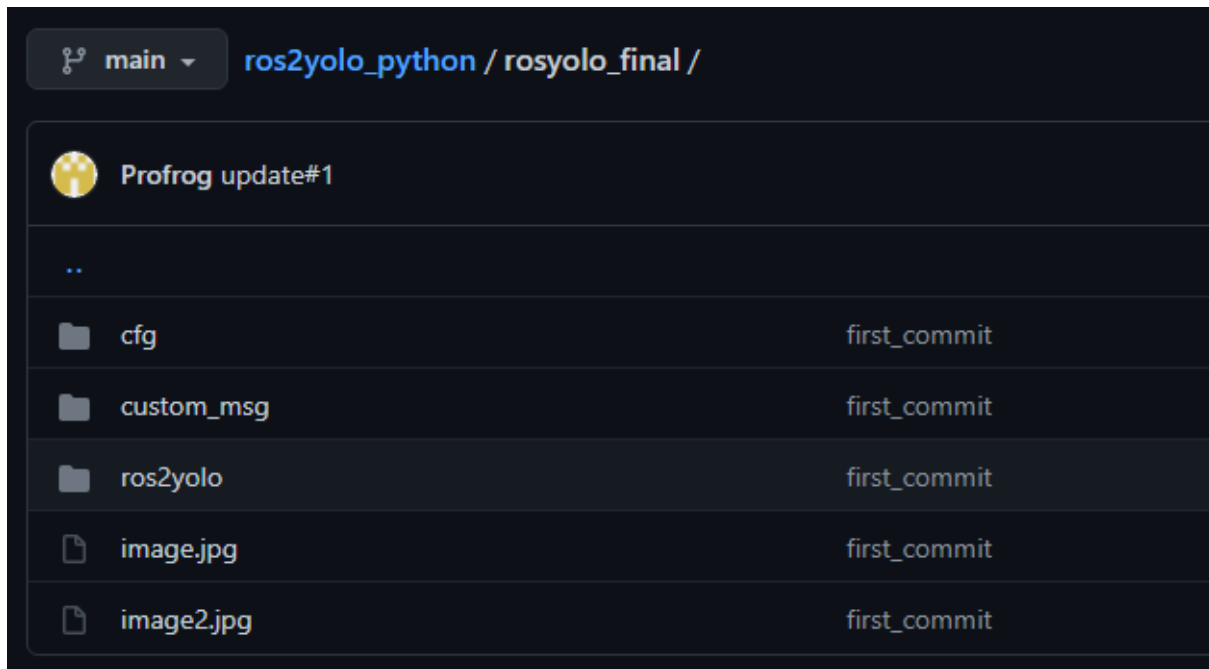
또한 전체 모듈을 부분적으로 나눈 중간 과정 프로그램 및 자료가공 도구 프로그램들도 github 에 모아서 정리해 두었다.

- camera.py : 파이썬 웹캠 제어(바코드 인식 가능)
- camera1.py : camera.py의 가벼운 버전(웹캠 제어만 가능)
- convert.py : gnss 장비 파싱 프로그램
- gnss\_convert.py : convert.py의 가벼운 버전
- graham\_scan(matplot + bresenham).py : grahamscan으로 얻어낸 점의 좌표를 채우는 것 까지 가능한 파일
- graham\_scan(matplot).py : 랜덤한 크기와 개수를 이용하여 grahamscan 알고리즘 검증용 파일
- graham\_scan.py : graham scan으로 점의 좌표만 얻어내는 파일(가벼운 버전)
- yolo\_webcam.py : in darknet it can realtime-yolo-detecting
- rosyolo : yolo 데이터 값을 ros2에 올린다.
- yolo\_webcam.py : 웹캠으로 얻어낸 실시간 이미지를 yolo로 검출하여 그 결과 값을 보여준다
- crawling.py : .txt의 내용들을 크롤링 하여 이미지로 저장
- crawling(auto).py : crawling. update 버전
- sir2020\_serial.py : sir2020값을 .txt로 로깅
- sir2020\_map.py : .txt로 로깅된 sir2020값을 matplot형태로 보여줌
- coco\_divide.py : image + annotation(yolo) 기준에서 특정 라벨을 가진 이미지만 추출
- label\_match.py : annotation이 있는 이미지만 save 해주는 코드
- pluslabel.py : 데이터셋에서 다른 weights에 있는 label 중 해당 names와 일치하는 것들은 추가 라벨링을 해주는 프로그램
- train\_txt.py : 해당 파일의 이미지 경로(jpg)를 모아 train.txt를 만들어준다
- precision.py : 해당 묶음의 dataset에 있는 사진들을 yolo를 돌려 class 별 검출률을 조사한다.

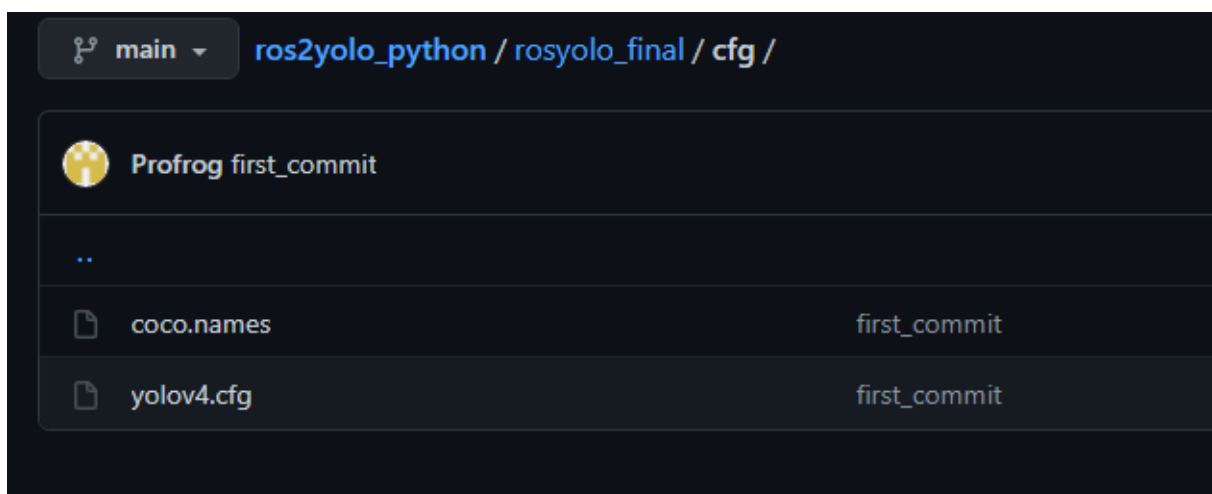
각각의 코드에 대한 설명 및 사용법은 해당 코드와 관련 있는 장에서 언급하며 설명하겠다.

[Profrog/yolo\\_scan\\_algorithm \(github.com\)](#)

이어서 ros2yolo package 의 구성에 대하여 좀 더 자세히 설명해 보도록 하겠다.

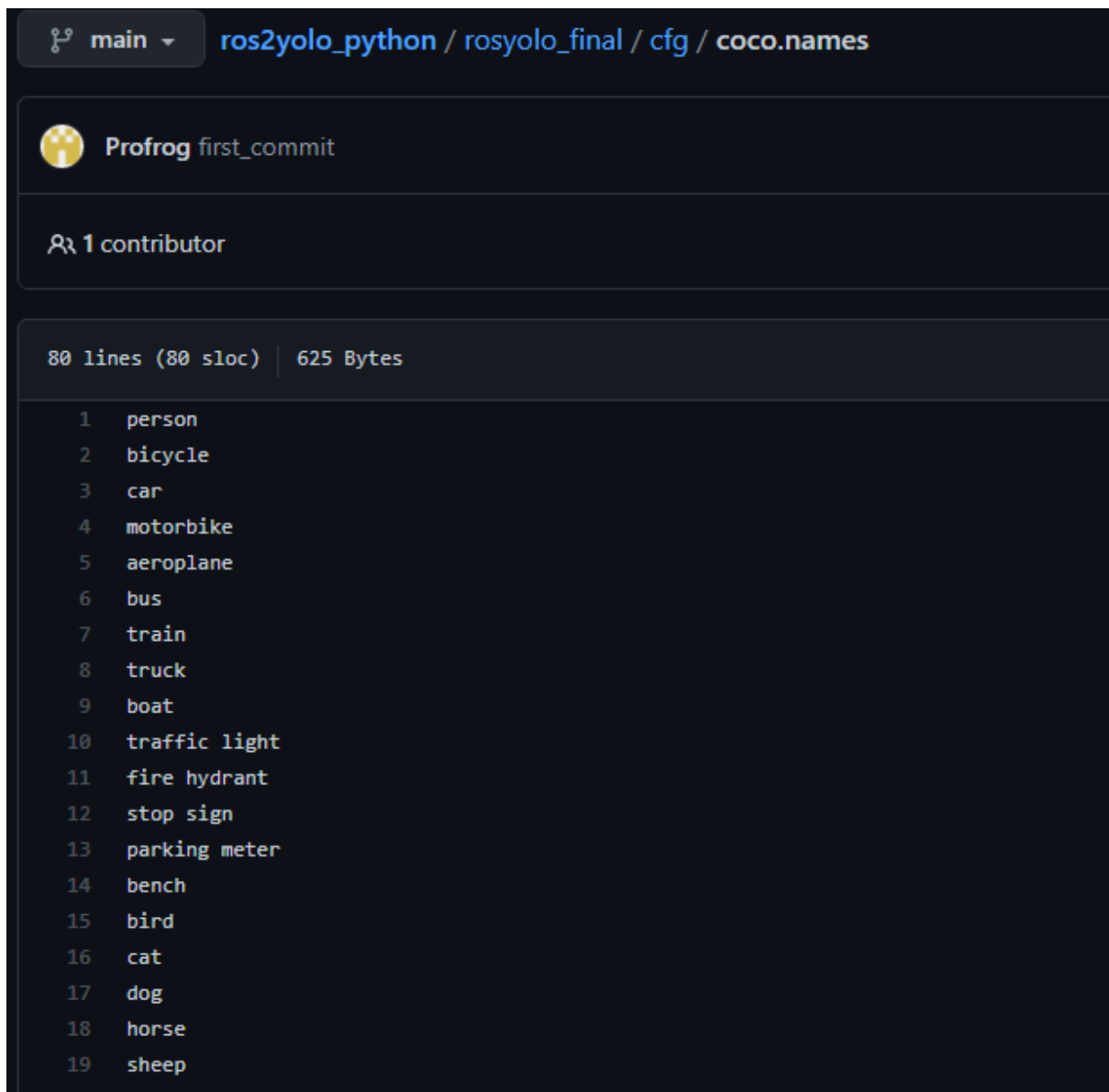


### 1.2.1 cfg 폴더



Coco.names 는 샘플 names 로 80 개의 class 로 이루어져 있다.

(만약에 사용자가 직접 training 시켜서 갖고 있는 .weights, .names , .cfg 파일이 있다면 그것을 사용하여도 된다.)



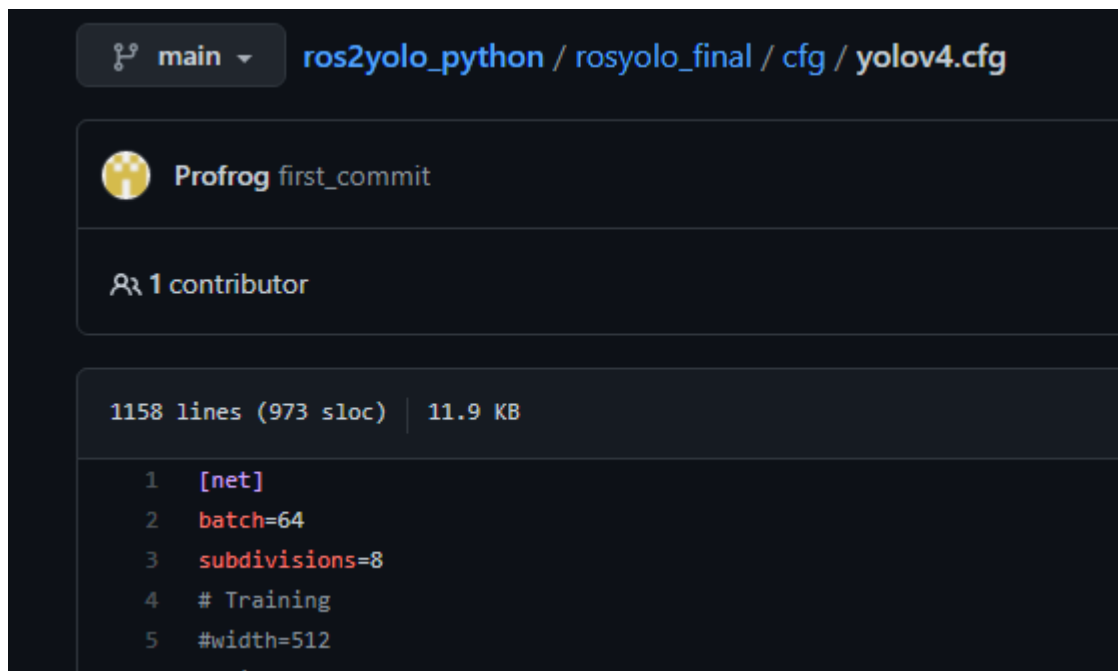
The screenshot shows a GitHub repository page for 'ros2yolo\_python' / 'rosyolo\_final' / 'cfg' / 'coco.names'. The repository is owned by 'Profrog' and has 'first\_commit'. It has 1 contributor. The file 'coco.names' is 80 lines (80 sloc) and 625 Bytes. The content of the file is a list of 19 object classes, each on a new line, numbered 1 to 19.

```
1 person
2 bicycle
3 car
4 motorbike
5 aeroplane
6 bus
7 train
8 truck
9 boat
10 traffic light
11 fire hydrant
12 stop sign
13 parking meter
14 bench
15 bird
16 cat
17 dog
18 horse
19 sheep
```

(names 는 검출할 class 의 목록들을 나타낸다. 샘플 .weights, .names, .cfg 파일은 darknet 디렉토리

안에도 존재하므로 해당 파일들을 복사하여서 사용해도 된다.)



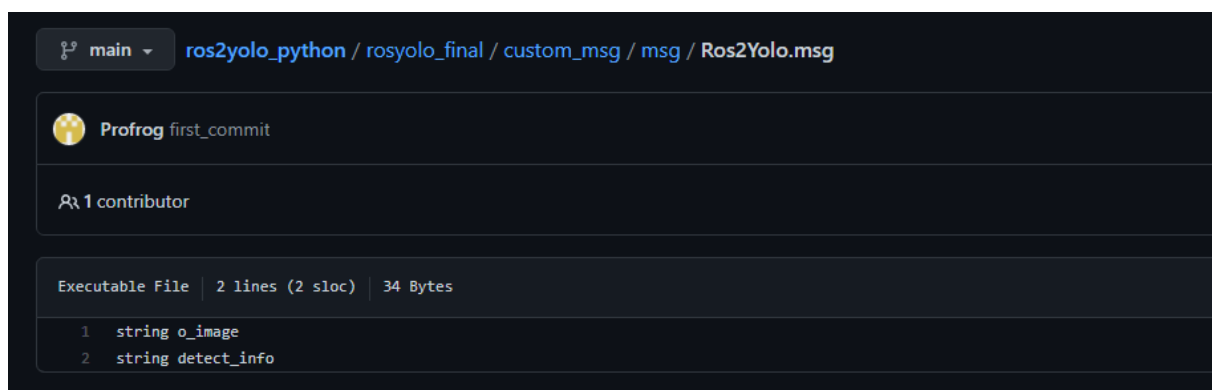


모델 구조 및 train 에 관련된 설명이 들어있는 파일로 마찬가지로 샘플 .cfg 파일이 darknet 안에 존재한다.

.weights 는 신경망 연산에서 분류할 때 가중치를 주기 위해 주어지는 file 로 용량이 github 에 올리기에선 상당히 큰 관계로 필자 github 안에는 존재하지 않으나 다운받을 수 있는 방법에 대해서는 서술해 놓았으므로 참조바란다.

### 1.2.2 custom\_msg 폴더

ros2 에서 subscribe, publish 할 message 타입의 목록이다.



기본값은 이미지 파일을 (byte -> string)으로 저장한 o\_image 변수와 검출 정보를 나타내는 detect\_info 변수로 이루어져 있으나, 만약에 추가로 subscribe 혹은 publish 하고 싶은 변수 목록이 있다면 추가 후 ros2python 의 디렉토리에 colcon build 후 생겨 있는 build, install, log 폴더를 삭제 후 다시 colcon build 를 명령어 창에 넣어주면 새로운 message 타입으로서 반영된다.

(만약 위의 과정을 통해 build, install, log 폴더를 갱신해 주지 않으면 .msg 파일을 수정하여도 반영되지 않는다)

Message type 변형 응용방식의 예시는 5 장에서 다시 한 번 서술하도록 하겠다

### 1.2.3 image.jpg, image2.jpg

image.jpg 는 webcam 에서 capture 한 사진을 담고 있으며

image2.jpg 는 image.jpg 를 yolo 로 detect 한 객체를 box 로 표시해둔 사진을 담고 있다.

Image.jpg -> image2.jpg 가 되는 과정은 밑에서 서술하겠다.

### 1.2.4 ros2yolo 폴더

해당 모듈 제어 코드가 담겨 진 폴더이자 ros2 node code 가 들어있는 폴더이다.

```

global yolo_image #after yolo detecting image
yolo_image = 'image2.jpg'

global original_image #before yolo detecting image
original_image = 'image.jpg'

global src_ros2

global dir_weights
dir_weights = 'yolov4.weights'

global dir_cfg
dir_cfg = 'cfg/yolov4.cfg'

global dir_coco
dir_coco = 'cfg/coco.names'

```

필자는 사용자가 해당 모듈을 커스터마이징 할 수 있도록 외부 파일의 directory 를 전역변수로 따로 선언해 두었으며 사용목적에 맞게 경로를 수정하여 쓸 수 있다.

Original\_image, yolo\_image 는 사진의 경로로 의미하는 바는 위의 1.2.3 을 참조바란다

dir\_weights, dir\_cfg, dir\_coco 는 각각 weights, cfg, names 파일의 경로를 나타내며 의미하는 바는 1.2.1 을 참조바란다.

```

global path1
path1 = os.path.abspath('/home/mingyu/git_yolo/darknet/rosyolo2/ros2-vn300/dew_ws/image.jpg')

```

현재 모듈은 ros2 package 내부의 상대좌표로 경로가 구성되어 있는데 혹시 절대좌표를 써서 경로를 나타낼 때는 위의 코드를 응용하여서 쓰면 된다.

```

colors = np.random.uniform(0, 255, size=(len(classes), 3))

vc = cv2.VideoCapture(0) # 0 = notebook_webcam, 2 = usb_webcam (in my computer)
print("waiting...")

```

해당 코드는 webcam 를 잡는 코드로서 VideoCapture(n)은 n 번째 카메라 모듈을 usb 를 통해 인식하여 객체로서 넣는 것을 의미한다. webcam 이 내장된 노트북 같은 경우는 0 이 내장 webcam, 2 가 추가로 연결한 webcam 으로 인식하는 경우도 많으니 원하는 카메라가 잡히지 않는 경우에는 해당 코드를 확인하는 것이 좋다.

```

def main(args=None):

    rclpy.init(args=args)
    ros2yolo_publisher = Ros2yoloPublisher(1) # sensor frequency is here
    rclpy.spin(ros2yolo_publisher)

    ros2yolo_publisher.destroy_node()
    rclpy.shutdown()

```

해당 코드에서 Ros2yoloPublisher(n)에서 n 은 호출 주기를 의미하는 것으로서 n 초에 한번 호출됨을 의미한다. 즉, Ros2yoloPublisher(1)일 경우 1 초에 한 번씩 웹캠의 사진을 capture 하여 yolo 를 돌리고 detect 결과를 publish 해주는 동작을 한다.

## 제 2 장 webcam 제어

### 2.1 Webcam 제어 python 프로그래밍

현재 모듈에서는 python 의 opencv 를 통한 카메라 제어를 하고 있는데 필자의 카메라 datasheet 는 다음과 같다



[Microsoft LifeCam VX-2000 - web camera Series Specs - CNET](#)

필자의 카메라는 ros2yolo 에서 잘 작동하나 몇 가지 종류의 카메라에 대해서 issue 가 있었으며, 기존 프로그래밍 코드의 호환성이 부족하다 생각될 경우 개선의 여지가 있다. 특히, 고 해상도 혹은 시스템과의 전원 및 port 연결 길이가 길어 연결이 불안정한 경우 잘 작동을 하지 않을 수 있으며, 성능이 좋은 컴퓨터 혹은 c++로 짜인 camera node 를 연결시켜 테스트 해보는 방향으로 개선 및 연구 방향성이 보이고 있다. Delay 에 관해서는 구체적인 추가 테스트가 필요하며 본 프로젝트 진행시에는 필자의 통합코드를 다시 block 화 시켜 최적화 과정에 진입할 때 비교 가능할 것으로 생각되어 진다.

서비스 코드 1: 해당 웹캠이 python opencv 로 잘 돌아가는 지를 확인할 때는 다음의 코드를 사용하면 된다.

[yolo\\_scan\\_algorithm/camera.py at main · Profrog/yolo\\_scan\\_algorithm \(github.com\)](#)

서비스 코드 2: 해당 웹캠 + 실시간 yolo 검출도 확인하고 싶을 때는 다음의 코드를 사용하면 된다.

[yolo\\_scan\\_algorithm/yolo\\_webcam.py at main · Profrog/yolo\\_scan\\_algorithm \(github.com\)](#)

```
16
17     start2 = 0
18
19     net = cv2.dnn.readNet("yolov4.weights", "cfg/yolov4.cfg")
20     classes = []
21     with open("cfg/coco.names", "r") as f:
22         classes = [line.strip() for line in f.readlines()]
23     layer_names = net.getLayerNames()
24     output_layers = [layer_names[i[0] - 1] for i in net.getUnconnectedOutLayers()]
25     colors = np.random.uniform(0, 255, size=(len(classes), 3))
26
27
```

본 코드는 webcam 제어 외에도 yolo 를 사용하기 때문에 .weights , .cfg , .names 파일이 올바른 경로에 있어야 한다.

기본은 cfg 폴더 내에 .cfg, .names 가 위치하고 본 디렉토리 바로 안에 .weights 가 위치하도록 설정해 두었다.

## 제 3 장 Yolo

### 3.1 Yolo 응용 python 프로그래밍



본 모듈에서 yolo 는 webcam 에서 capture 된 이미지에서 object 를 검출하는데 사용되며 때문에 접근 방향은 크게 2 개로 나누어 진다.

#### 3.1.1 해당 object 의 precision 올리기

#### 3.1.2 실시간 영상에서 capture 한 n 개의 사진에서 일부는 object 를 검출하고, 일부는

#### 검출하지 못한 상황에 대한 보정 처리

3.1.1 의 경우 영향을 미칠 수 있는 것을 크게 3 종류로 정리하였다.

##### 3.1.1.a dataset

##### 3.1.1.b Labeling

##### 3.1.1.c 다른 dataset(ex :coco)의 응용

yolo 에서 training 시키기 위한 data set 을 만들기 위해서는 여러 장의 image 가 필요하다.(3.1.1.a)

따라서 일정 해상도 이상의 사진을 web crawling 하는 코드가 필요하여 서비스 코드로 준비해 두었다.

서비스 코드 1: [yolo\\_scan\\_algorithm/crawling\(auto\).py at main · Profrog/yolo\\_scan\\_algorithm \(github.com\)](https://github.com/Profrog/yolo_scan_algorithm/blob/main/yolo_scan_algorithm/crawling(auto).py)

본 프로그램은 사용자로부터 검색어와 저장 받을 이미지의 수를 입력받아 해당 개수만큼 chrome 에서 crawling 하여 폴더에 저장하는 프로그램이다.

```
global url0
url0 = "https://www.google.co.kr/imghp?hl=ko&ogbl"

global file_name
file_name = 't_image'

global image_name
image_name = "road_w_aa"
```

url0 는 구글 이미지 창에서 이미지를 검색함을 의미한다. 필요에 따라 주소를 바꿀 수 있다.

file\_name 은 이미지가 저장될 폴더의 이름이다. 마찬가지로 필요에 따라 주소를 바꿀 수 있다.

image\_name 은 이미지가 저장될 이름이다. crawling 된 이미지는 이 "이름 + n 번째.jpg"의 형태로 저장된다.

```
data2 = open('crol_box.txt' , 'w+') #web code
```

crol\_box.txt 는 웹에서 복사한 웹 데이터가 그대로 저장되는 파일이다.

```
data3 = open('crol.txt', 'w+') # link configz
data4 = open('error_link.txt', 'w+')
```



crol.txt 는 이미지를 crawling 한 링크가 저장되는 파일이다.

error\_link.txt 는 이미지가 웹에 존재하나 다운로드가 실패한 링크가 저장되는 파일이다.

그 후에는 labeling 작업을 하게 된다.(3.1.1.b) 해당 내용은 중요하지만 어렵지는 않으므로

[tzutalin/labellmg](#):  Labellmg is a graphical image annotation tool and label object bounding

[boxes in images \(github.com\)](#)

본 링크를 활용바란다.

labeling 작업 중 dataset 으로 쓰기에 적합하지 않은 이미지는 작업을 하지 않아 annotation 파일이 생성되지 않을 텐데, 해당 이미지들을 제외하고 image, annotation 세트가 존재하는 dataset 만 따로 정리하는 서비스 코드를 제공할것다.

서비스 코드 2 : [yolo\\_scan\\_algorithm/label\\_match.py at main · Profrog/yolo\\_scan\\_algorithm](#)

[\(github.com\)](#)

```
global path
path = "t_image/"

global search1
search1 = "working1"
```

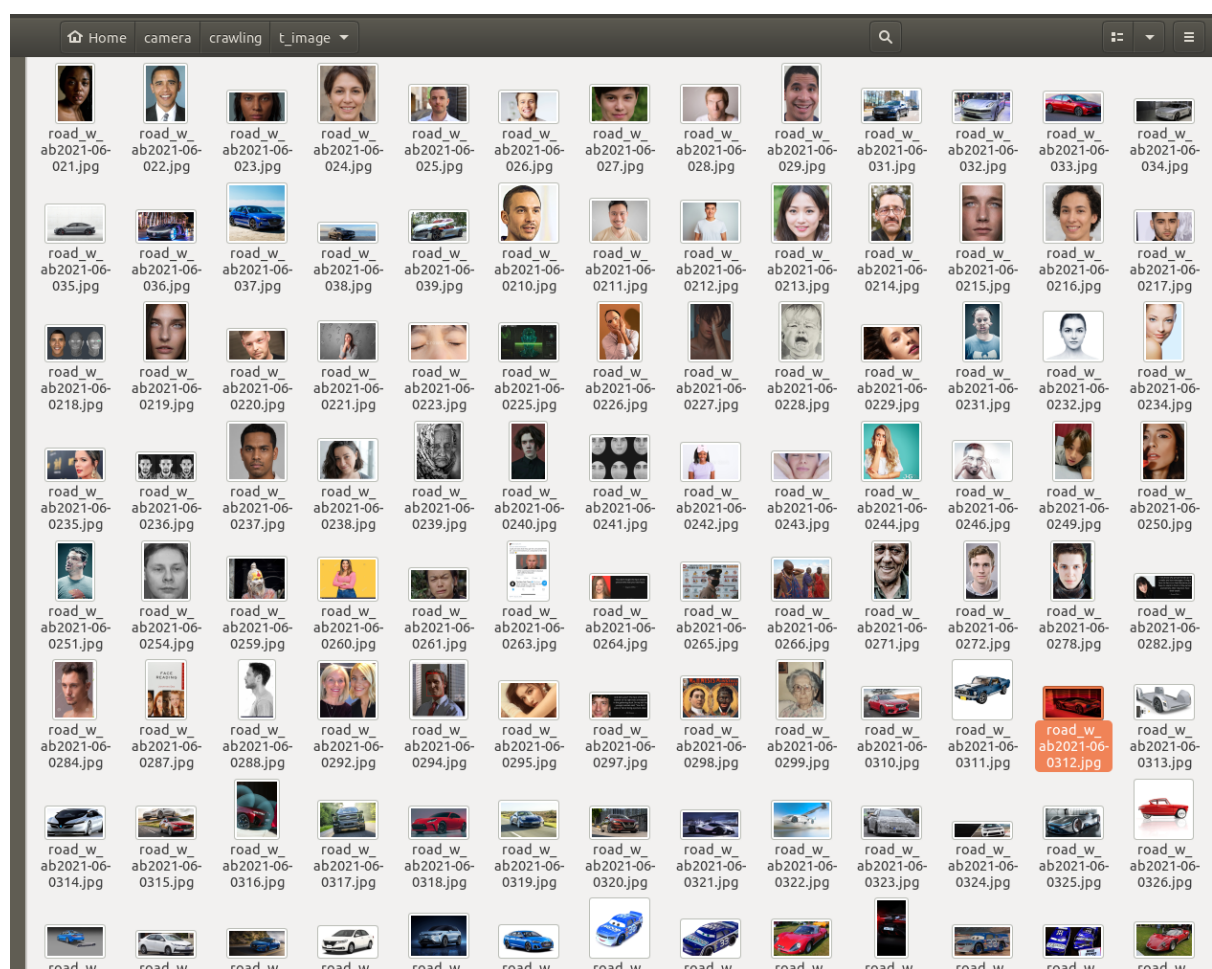
path 는 원본 dataset 이 있는 폴더 명,

search1 은 annotation 이 포함된 dataset 들만 옮겨갈 폴더 명이다.

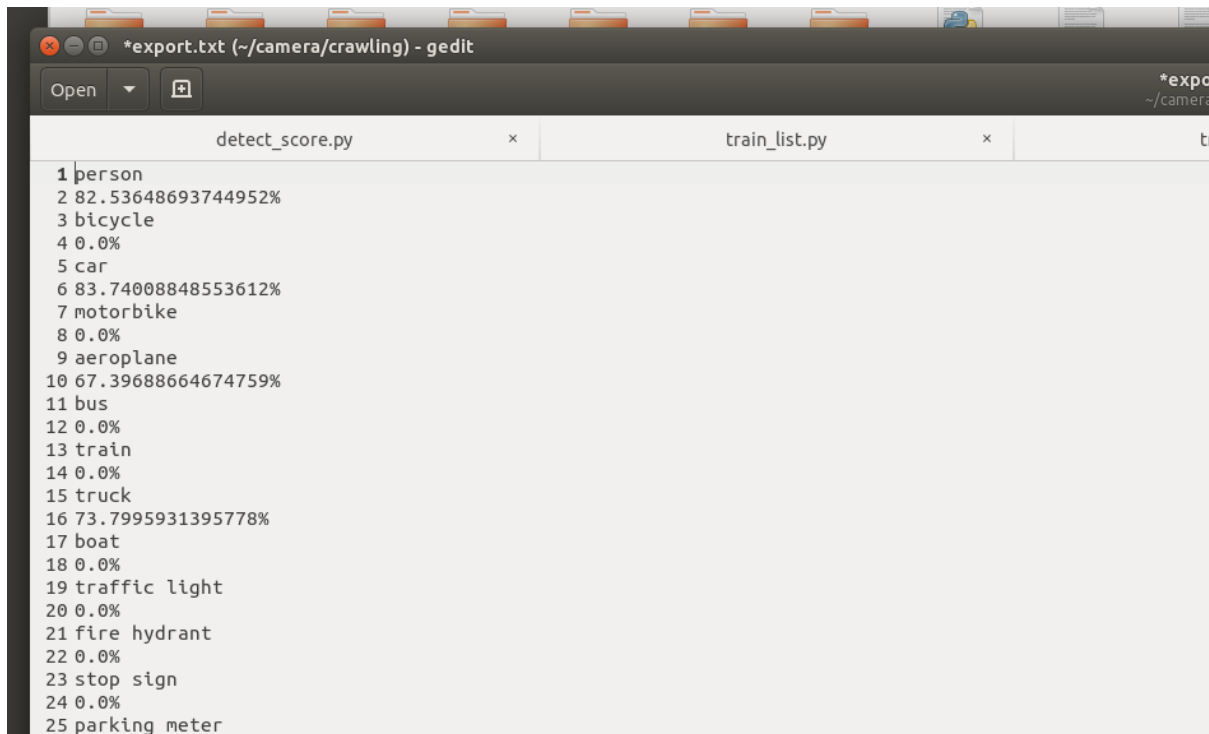
여기까지가 dataset 을 만드는 과정이며, 더 좋은 dataset 을 만들기 위해서는 기존 dataset 을 분석하거나 다른 dataset 과의 비교 및 통합 과정이 필요한데 해당 과정을 도와주는 서비스 코드를 제공한다.(3.1.1.c)

서비스 코드 3 : [yolo\\_scan\\_algorithm/precision.py at main · Profrog/yolo\\_scan\\_algorithm](https://github.com/Profrog/yolo_scan_algorithm/blob/main/yolo_scan_algorithm/precision.py)

([github.com](https://github.com))



(해당 dataset 은 3 장 서비스 코드 1 로 car, person 에 관련된 사진만 모은 것이다)



해당 서비스 코드는 사용자가 가진 dataset 에서 class 별 precision 을 나타내 주는 코드이다.

해당 코드를 이용하여 class 별 precision 을 올리는 방향으로 연구하면 될 것 같다.

```

global path
path = "t_image/"

global search1
search1 = "working2"

global dir_weights
dir_weights = 'yolov4.weights'

global dir_cfg
dir_cfg = 'cfg/yolov4.cfg'

global dir_coco
dir_coco = 'cfg/coco.names'

```

(변수들의 뜻은 1 장 서비스 코드 2, 2 장 서비스 코드 2 의 설명을 참조하면 된다.)

precision 을 올리기 위해선 기존에 제공되는 dataset 에서 필요한 class 의 dataset 만 가져와서 하나의 dataset 으로 학습시켜 쓰는 방법이 있는 데, 그 중 하나인 coco data set 추출에 대해서 설명 및 서비스 코드를 제공할 것이다.

서비스 코드 4: [yolo\\_scan\\_algorithm/coco\\_divide.py at main · Profrog/yolo\\_scan\\_algorithm](#)

([github.com](#))

```
global name_file
name_file = "coco.names"
#data1 = open(name_file, 'a')

global path
path = "image/"
```

.names 파일과 원본 데이터(coco image)저장 경로를 변수로 받아  
사용자로부터 class label 명을 입력 받은 후 그 label 이 .names 파일의 class 목록에 있으면,  
데이터 셋 각각의 Annotation 파일에서 해당 class 가 있는지 찾고, 있으면 특정 폴더("image/")로  
모아주는 서비스 코드이다.(class label 을 가진 image,annotation 이 폴더로 모이게 된다.)

서비스 코드 5 : [yolo\\_scan\\_algorithm/pluslabel.py at main · Profrog/yolo\\_scan\\_algorithm](#)

([github.com](#))

```
global path
path = "t_image/"

global search1
search1 = "working2"

global dir_weights
dir_weights = 'yolov4.weights'

global dir_cfg
dir_cfg = 'cfg/yolov4.cfg'

global dir_coco
dir_coco = 'cfg/coco.names'

global name_file
name_file = 'cfg/classes.txt'
```

(변수들의 뜻은 1 장 서비스 코드 2, 2 장 서비스 코드 2 의 설명을 참조하면 된다.)

서비스 코드 5 는 잘 학습된 weights 가 있을 경우(ex: coco dataset's weights)

사용자가 작업한(ex: crawling 한) dataset 에서 해당 weights 의 class 중 사용자가 원하는 class 를

자동으로 사용자의 dataset 에 추가로 labeling 해 주는 프로그램이다.

weights 의 .names (coco) -> dir\_coco

사용자가 쓸 .names (갖고 있는 데이터 셋)-> name\_file

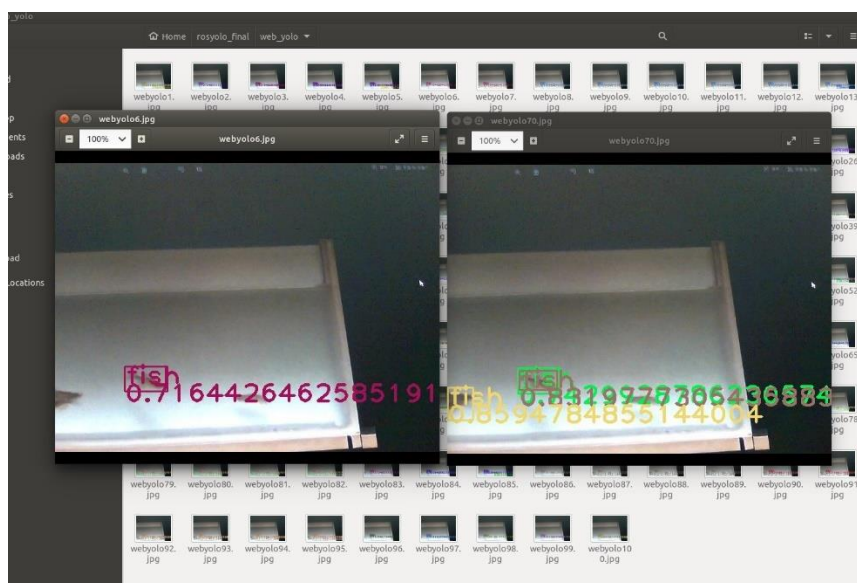
로 받아서 겹치는 class 만 labeling 해준다.

여기까지 data set 을 모으기 위한 사전과정과 서비스 코드 제공(1,2)

precision 을 높이기 위한 해결방안과 서비스 코드 제공(3,4,5)에 대해 서술하였다.

(3.1.1 에 대한 설명)

3.1.2 같은 경우 실시간 capture 에서 frame 이 어긋날 경우 image 가 흐려질 수 있는데, 이 때 yolo 는 이를 객체가 없는 경우로 인식할 수 있어 이와 관련된 연구 및 보정코드가 추가로 필요하다는 판단이다.



예시로 두 개의 검출 사진에서 왼쪽의 물고기는 검출이 되지 않는 경우가 있었으며, 시스템에서는 이를 물체가 사라졌다 나타나는 것을 반복하는 것으로 인식할 수 있다.

현재 필자의 의견으로는 연속되는 몇 장의 capture 된 사진에서 detect 된 class 를 100%로 두고 n 장의 사진에서 평균을 내어 확률적으로 접근하는 방법을 시험해 볼 수 있을 것 같다.

Yolo 를 제어하는 필자의 코드도 위와 마찬가지로 실상황에서는 사용하기 전에 실제 object precision 과 인지 속도등을 여러 조건에서 테스트하여 개선이 필요하다는 점을 미리 언급해 둔다.

## 제 4 장 Ros2

### 4.1 Ros2 응용 python 프로그래밍

본 모듈에서 ros2 를 사용하는 이유는 해당 모듈을 node 단위로 나누거나 혹은 다른 모듈이랑 통합하여 사용할 때 ros2 가 통합 코드 플랫폼으로서 작동할 수 있기 때문이다.

만약에 최적화를 위해 ros2yolo 를 기능에 따라 node 단위로 분할시킬 때는 코드를 부분적으로 python-class 화 시켜서 node(.py)에 나누어 붙여 message 를 publish, subscribe 하는 방식으로 쓸 수 있다.

필자의 모듈을 다른 프로젝트에 붙여서 쓸 때는 주고받는 message-type 을 맞춰서 publish, subscribe 하여 통합시킬 수 가 있다. 자세한 내용은 제 5 장에서 서술하도록 하겠다.

3.1.2 에서 yolo 의 precision 을 올리기 위한 여러 경우의 수를 생각하며 보정 코드에 대해 언급하였었는데, 그 중 카메라 이외의 sensor 모듈을 package 에 붙여서 data 의 신뢰도를 높이는 방법이있다.

그 중 하나인 gnss(=gps)장비를 예시로 ros2 package 다루는 법을 서술하겠다.

Uart serial 통신을 하는 여러 종류의 gps 장비를 제공하는 서비스 코드로 돌려볼 수 있으며, 필자는 Vn300, tdr3000-sir2020 센서 사용기준으로 작성하였다.

Vn300 datasheet : [vn-300-datasheet.pdf \(vectornav.com\)](https://www.vectornav.com/vn-300-datasheet.pdf)

Tdr-3000 datasheet : [TDR-3000 | SYNEREX](https://www.synerex.com/tdr-3000-datasheet.pdf)

서비스 코드 1 : [https://github.com/Profrog/yolo\\_scan\\_algorithm/blob/main/serial.py](https://github.com/Profrog/yolo_scan_algorithm/blob/main/serial.py)

```
ser = serial.Serial('/dev/ttyUSB0', 115200, timeout = 1)
data = open('data2.txt', 'a')
```

115200 은 baudrate 로 모듈마다 다를 수 있으니 갖고 있는 모듈의 datasheet 를 확인바란다.

Serial port 로 모듈이 전송하는 값은 data2.txt 에 logging 된다.

서비스 코드 2 : [https://github.com/Profrog/yolo\\_scan\\_algorithm/blob/main/sir2020\\_serial.py](https://github.com/Profrog/yolo_scan_algorithm/blob/main/sir2020_serial.py)

nmea 데이터로 들어오는 값들 중 GNGGA, GPHDT 를 parsing 하여 쓰는 것이다.

nmea format 에 대한 참조는 다음 링크를 참조바란다.

[http://navspark.mybigcommerce.com/content/NMEA\\_Format\\_v0.1.pdf](http://navspark.mybigcommerce.com/content/NMEA_Format_v0.1.pdf)

```
ser = serial.Serial('/dev/ttyUSB0', 115200, timeout = 1)
data = open('data_map2.txt', 'w+')
```

해당 코드는 sir-2020 을 기준으로 parsing-data 를 뽑아내는 코드이며, nmea format 에 따라 일부 데이터만 logging 하여 사용할 때 쓰면 된다.

서비스 코드 3 : [https://github.com/Profrog/yolo\\_scan\\_algorithm/blob/main/sir2020\\_map.py](https://github.com/Profrog/yolo_scan_algorithm/blob/main/sir2020_map.py)

logging 된 gps 데이터를 matplotlib 을 이용하여 visualization 해 주는 코드로 서비스 코드 2 와 세트로 작동한다.

Logging 된 gps global 좌표를 일정 크기의 2D list 에 저장해서 누적된 좌표를 container 안에 저장한다.



```
global size_a
size_a = 10000

global zoom
zoom = 30
```

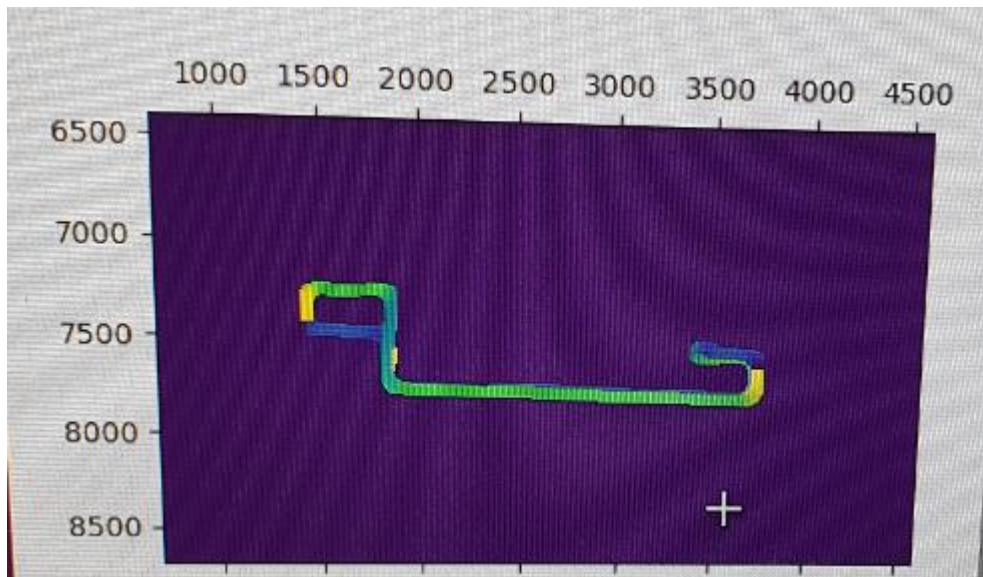
size\_a 는 설정할 global 좌표계의 크기이다. cm 단위 이므로 default 는 100m\*100m 이다.

Zoom 은 경로를 지나는 물체의 크기이다. (배율) default 는 30cm 으로 적용된다.

```
data2 = open('data_map2.txt', 'r')
data3 = open('map_result.txt', 'a')
```

data2 는 읽어 올 logging 데이터 파일의 경로를 받는다.

data3 는 전체 좌표계를 2 차원 list 로 저장할 파일의 경로이다.



(연구소를 한 바퀴 돌았을 때의 자동차의 이동 경로를 보여주는 실행 결과이다.

색이 다른 것은 heading 값을 색에 따라 표현하여서 그렇다.)

정리하면 서비스 코드(1,2,3)을 사용하여 센서의 값을 serial 통신으로 받아서 logging 할 수 있고 2D list 에서의 경로를 표현할 수 있다.

사용자의 필요성에 따라서는 서비스 코드를 조금 수정하여 레이더 같은 다른 sensor 모듈에 대해서도 작동하도록 할 수 있다

또한 여기에 ros2 까지 쓸 줄 안다면 레이더와 gps 모듈을 상호 보완하여 안정성을 높인 package 를 구성할 수도 있을 것이다. 이것이 ros2 를 쓰는 이유이다.

다음 차례에는 서비스 코드(1,2)의 내용을(serial 로 gps 데이터를 받아와서 logging) txt 를 사용하지 않고, ros2-message-type 을 이용하여 .msg format 으로 사용하여 보겠다.  
(logging 까지 원한다면 message publish 후 rosbag 을 사용하여 logging 하면된다.)

서비스 코드 4 : [https://github.com/Profrog/yolo\\_scan\\_algorithm/tree/ros2\\_exe](https://github.com/Profrog/yolo_scan_algorithm/tree/ros2_exe)

해당 branch 의 readme 를 읽어보고 command 를 그대로 치면 package 가 작동할 것이다.

그 후 rqt 명령어를 통해 topic 을 구독하여 message 값들을 확인하고 필요에 따라 rosbag 을 통하여 logging 할 수 있다.

Ros2 rqt 관련 : [Introducing turtlesim and rqt — ROS 2 Documentation: Foxy documentation](#)

Ros2 rosbag 관련 : [GitHub - ros2/rosbag2](#)

그 다음은 ros2 node 파일을 수정하는 단계이다.

```
def __init__(self, portCls, Hz):
    super().__init__('tdr3000_publisher')
    self.frame_id = 'tdr3000/num'+portCls.portNum
    self.publisher_ = self.create_publisher(Tdr3000, self.frame_id , 1)
    timer_period = 1/(Hz*2) # seconds
    self.timer = self.create_timer(timer_period, self.timer_callback)
    self.ser = portCls.ser
```

[yolo\\_scan\\_algorithm/tdr\\_3000.py at ros2\\_exe · Profrog/yolo\\_scan\\_algorithm · GitHub](#)

Self.timer 를 보면 timer\_period(주기)에 따라 한 번씩 timer\_callback 을 호출하고 있음을 알 수 있다.

```
def timer_callback(self):
    line_gng = self.ser.readline() # $GNGGA
    line_gph = self.ser.readline() # $GPHDT
    gph_list = line_gph.decode().replace('\x00','').split(',')

    if gph_list[0] == '$GNGGA': #except process
        a = line_gph
        line_gph = line_gng
        line_gng = a
        gph_list = line_gph.decode().replace('\x00','').split(',')

    #heading_data = float(gph_list[1])
    gng_list = line_gng.decode().replace('\x00','').split(',')
    #latitude0 = gng_list[2]
    #longitude0 = gng_list[4]

    #msgList0 = self.tdr3000(gphdt)
```

Time\_callback 을 수정하여 송출할 message 의 목적에 따라 쓰면 된다.

```
msg.serial_data = serial_a

# pup msg
self.publisher_.publish(msg)
```

. msg serial\_data 변수에 serial\_a(string 값)을 넣고 publish 해서 message 를 송출하는 부분이다.

참고로 서비스 코드 4 의 msg 구성은 다음과 같다.

```

1  std_msgs/Header header
2  float64 utc
3  float64 latitude
4  string hemisphere_lati
5  float64 longitude
6  string hemisphere_long
7  float64 indicator
8  float64 num_satellites
9  float64 hdop
10 float64 altitude
11 string unit_altitude
12 float64 geoidal_sep
13 string unit_geoidal
14 float64 heading_degree
15 string serial_data

```

[yolo\\_scan\\_algorithm/Tdr3000.msg at ros2\\_exe · Profrog/yolo\\_scan\\_algorithm · GitHub](https://github.com/Profrog/yolo_scan_algorithm/blob/master/ros2_exe/tdr3000/src/py_tdr3000/py_tdr3000.msg)

만약 custom 된 node 를 생성하고 싶다면

[https://github.com/Profrog/yolo\\_scan\\_algorithm/tree/ros2\\_exe/tdr3000/src/py\\_tdr3000/py\\_tdr3000](https://github.com/Profrog/yolo_scan_algorithm/tree/ros2_exe/tdr3000/src/py_tdr3000/py_tdr3000)

부분에 새 python 파일을 작성하고

```

entry_points={
    'console_scripts': [
        'tdr_3000 = py_tdr3000.tdr_3000:main',
        'sir_2020 = py_sir2020.sir_2020:main',
    ],
}

```

[https://github.com/Profrog/yolo\\_scan\\_algorithm/blob/master/ros2\\_exe/tdr3000/src/py\\_tdr3000/setup.py](https://github.com/Profrog/yolo_scan_algorithm/blob/master/ros2_exe/tdr3000/src/py_tdr3000/setup.py)

ros2 package 에서 setup.py 를 찾고,

여기에 python 파일 명과 class 명을 잘 맞추어 같은 format 으로 한 줄 추가해 주면 된다.

요약하면,

serial 통신으로 sensor 에서 data 를 얻어와서 visualization 하는 과정을 다루는 서비스 코드(1,2,3)

해당 코드를 ros2 packge 로 만들어 message 를 publish 하는 서비스 코드(4)로 정리가능하다.

Yolo 로 object 를 검출하고 기입된 물체의 실제 크기와 이미지 파일 안의 물체의 크기를 비례

관계를 이용하여 거리를 추정하는 방법은 오차가 기준치 이상 있을 수가 있어서,

적외선 센서 등으로 물체와의 거리를 보정하는 node 를 추가하게 된다면 ros2yolo 를 좀 더

발전시켜서 사용해 볼 수 있을 것 같다.

다른 방향으로 message 의 data 값을 여러 algorithm 을 통해 2 차 가공하여 결과값을

publish 해주는 node 를 만들어 볼 수도 있다. 구체적인 적용방법은 이어서 제 5 장에서 다뤄보도록

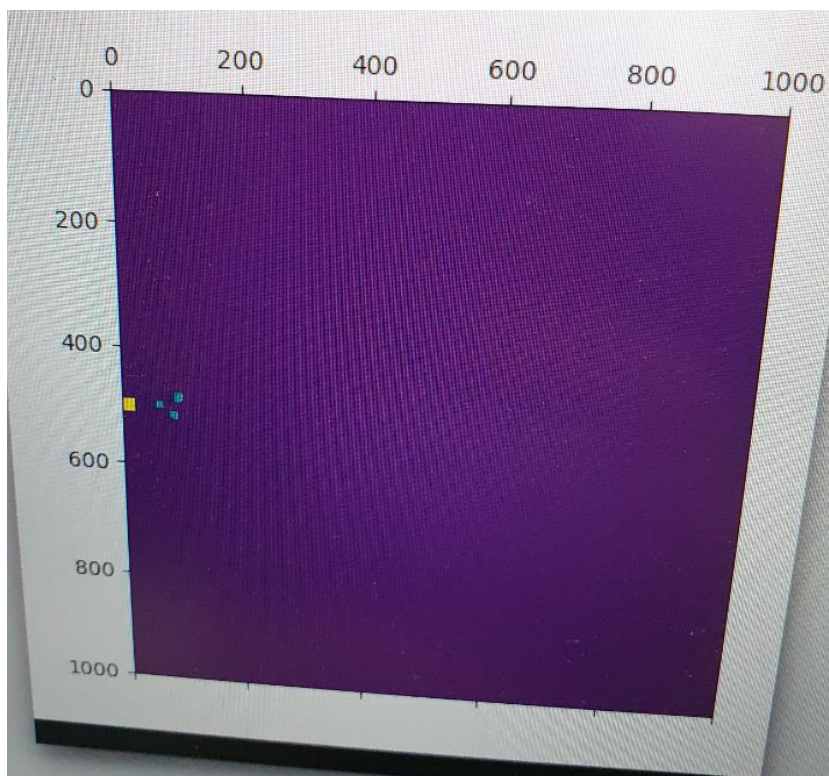
하겠다.

## 제5 장 통합응용

### 5.1 Algorithm 을 적용하여 yolo 검출 객체를 2D 맵핑 프로그래밍

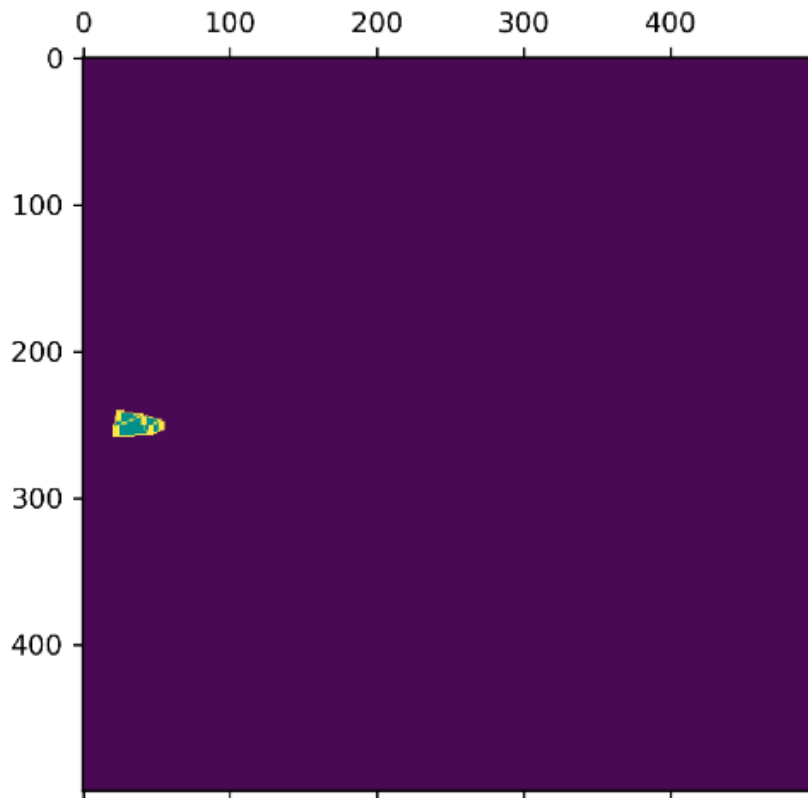
해당 장에서는 ros2yolo 에 카메라의 화각을 이용하여 detect 되는 object 와의 거리를 산출하고 그 object 의 좌표를 topview 관점에서 2D list 에 저장하여 matplotlib 으로 visualization 하여 표현하고 더 추가적으로는 bresenham algortihm 과 graham scan alogorithm 을 사용하여 object 들의 좌표 중 가장 바깥 쪽을 line 으로 싸는 도형을 만들고 그 영역을 채우는 응용 프로그래밍을 진행하겠다.

#### 5.1.1 카메라의 화각을 이용하여 yolo 로 탐지된 object 의 좌표를 2d list in matplotlib 으로 표현



[GitHub - Profrog/ros2yolo\\_python at yolomap](#)

5.1.2 영역 탐지 algorithm 을 사용하여 object 의 좌표 중 가장 바깥쪽 영역을 포함하는 도형을 만들고 그 영역을 채워넣는 과정



[GitHub - Profrog/ros2yolo\\_python at areadetect](#)

2~4 장에서 서술하였던 내용 및 서비스 코드가 들어가므로 앞 장의 내용들을 확인할 필요가 있다.

5.1.1 에 대하여 좀 더 자세히 서술하여 보도록 하겠다. 시점은

World -> webcam -> yolo -> mapping(ros2) 순으로 이동한다.

물체의 거리를 재는 데 사용되는 카메라의 화각 개념에 대해서는 아래 내용을 참조바란다.

[화각 \(사진술\) - 위키백과, 우리 모두의 백과사전 \(wikipedia.org\)](#)



예를 들어, yolo 로 검출된 다음 image 가 있다고 했을 때, 검출된 class 는 person, cell phone 두 종류이다. 그 중 예시로 들기 쉬운 cell phone 을 기준으로 한다면

제 3 장에서 다룬 yolo 프로그래밍 코드 중 일부를 사용하여 image 상에서 검출된 cell phone 의 pixel 크기를 구할 수 있다. 또한 일반적인 크기의 cell phone 의 크기를 이미 알 수 있으므로 카메라의 화각을 이용해 image 가 떨어진 거리에 따라 어느정도 크기가 변형되는 지를 계산하는 공식을 사용하면 물체와 카메라가 떨어진 거리를 일정 오차 범위 안에서 구할 수 있다.

제 4 장의 서비스 코드 2,3 을 응용한 것처럼 좌표 체계를 2D list 로 나타내고 거기에 물체와의 거리와 각도를 이용하면 위치를 특정할 수 있으며 물체의 크기를 알고있기 때문에 범위로 표현이 가능하여 matplotlib 으로 visualization 할 수 있다.

물체의 좌표뿐만 아니라 class label 명도 알 수 있으므로 검출된 객체에 따라 node 를 분리하여 다른 동작을 수행하게끔 모듈을 수정해 볼 수도 있다. (예시로는 반경 10m 안에 검출된 객체가 사람일 경우는 이동체의 속도를 50%, 다른 객체일 때는 30% 줄이도록 할 수 있다.)



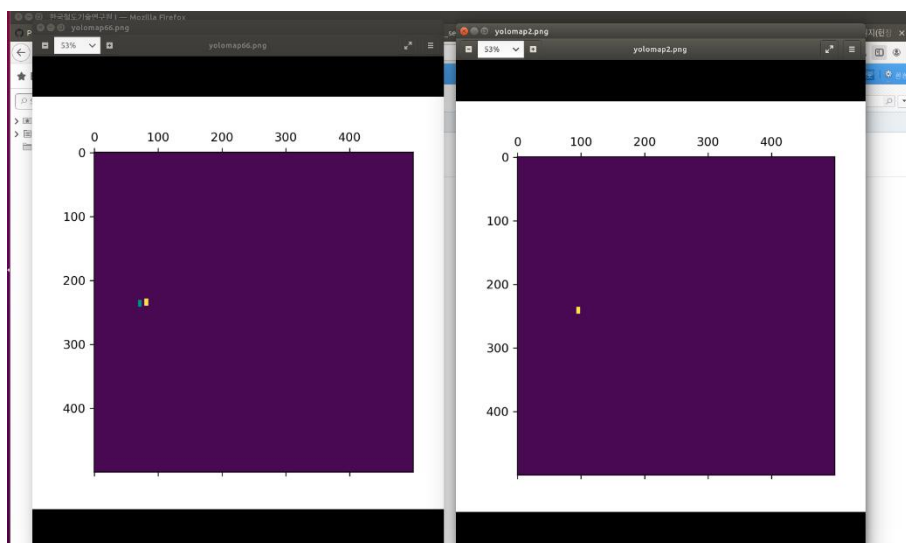
그 다음에는 5.1.1 의 과정에 bresenham, graham scan algorithm 을 붙여서 검출된 object 간 영역 탐지가 가능한 node 를 만들어 보겠다.

[Bresenham's line algorithm - Wikipedia](#)

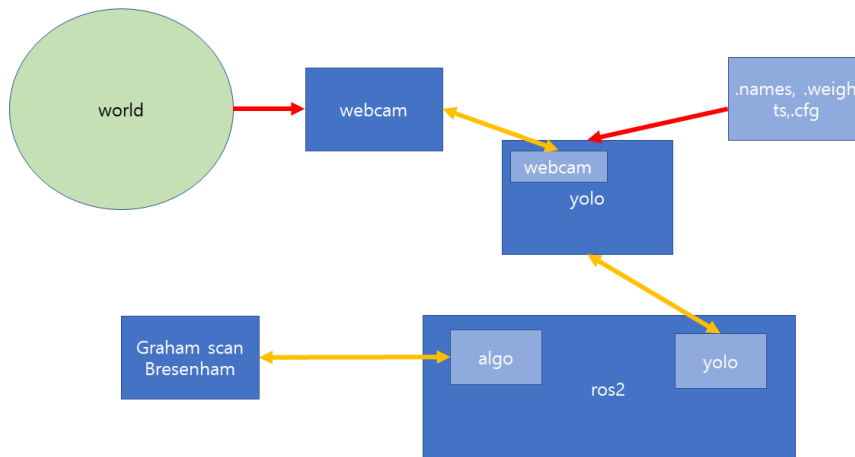
[Graham scan - Wikipedia](#)

Algorithm 의 대한 자세한 내용은 위의 링크를 참조바란다. 요약하면 bresenham 은 2d list 좌표체계에서 두 점 사이의 line 을 만들어주는 algorithm 이고 graham scan 은 여러 개의 점 중 가장 outline 들로 만들어지는 도형의 선분을 만들 수 있는 점을 선별하는 algorithm 이다.

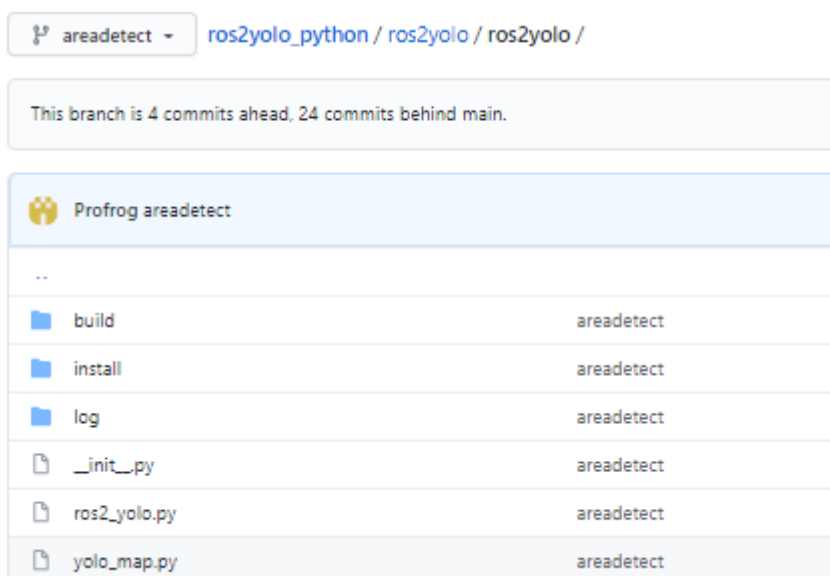
따라서, 5.1.1 의 package 에 붙여서 쓸 때는 object 의 detect 탐지 좌표와 크기를 알 수 있으므로, 사각형의 모양으로 가정할 때, 4 개의 꼭짓점의 좌표를 얻어 낼 수 있고, 이 좌표들의 list 를 graham scan 의 input 값으로 사용하여 outline 을 이루는 도형의 점의 좌표를 획득 가능하고, 여기에 bresenham 을 적용하여 도형을 만든 뒤, 영역을 채우는 algorithm 을 적용하여 node 를 완성시킬 수 있다.



다만 3.1.2 에서도 제시했던 issue 와 같이 frame 별로 object 를 검출하지 못하는 경우도 있어서 개선이 필요하다.










흐름도를 보면서 전반적으로 정리를 하자면, webcam 은 world 를 실시간으로 image format 으로 저장하고, 해당 webcam 의 화각 정보와 yolo 를 거쳐서 detect 된 object 정보는 1 차로 ros2 에서 publish 된다. (ros2\_yolo.py)



그러면 mapping 시켜주는 node 에서는 webcam, yolo 의 data 를 이용하여 2d list 형태의 map 을 구성하고 detect 된 object 의 좌표, 크기를 반영하여 map 에 표현하며, 여기에 더 나아가서

bresenham, graham scan algorithm 을 적용하여 detect 를 원하는 object 의 집단의 판정 영역을 구하여 2d list 에 반영된 값들을 publish 한다. ( yolo\_map.py)

 rosyolo	areadetect	24 days ago
 rosyolo_camera_sh	areadetect	24 days ago
 rosyolo_yolomap_sh	areadetect	24 days ago
 size.txt	areadetect	24 days ago
 test2_sh	areadetect	24 days ago
 yolomap0.png	areadetect	24 days ago
 yolomap1.png	areadetect	24 days ago

또한 사용자 편리성을 위하여 shell script 형태로 시스템을 지원하고 있으며,

Rosyolo : 시스템 전체를 돌리는 shell script

Rosyolo\_camera\_sh : 카메라만 작동시키는 shell script

Rosyolo\_yolomap\_sh : mapping data 를 생성하는 node 를 제어하는 shell script

제 5 장을 마지막으로 연구내용에 대한 기술적인 정리는 마무리하였으며, 해당 보고서의 필자이자 모듈 제작자의 의견으로서는 필요에 따라 일부 혹은 전체를 가져다가 여러 응용 형태로 변형하여 최적화 모듈을 제작하기 좋은 프로토 타입 형태의 모듈이라고 정의하고 싶다. 전반적으로 다뤄야 할 분야와 양이 많은 관계로 기술적인 명칭을 생략하거나, 깊이 있는 이해가 부족한 상태에서 직관적으로 서술한 점도 분명 있을것이고, 오히려 서술이 부족하여 사용하기에 어려운 부분도 있을 것이라고 판단된다. 그런 경우에는 제작자의 이메일로 feedback 을 보내준다면 관련 사항 수정을 해보도록 하겠다. 다음 장은 전체적인 내용을 마무리지며 해당 보고서를 마치도록 하겠다.

[ache159@naver.com](mailto:ache159@naver.com)

## 제 6 장 결론

### 6.1 Ros2yolo 에 대한 결론

해당 장은 보고서의 마지막 장이자 결론에 대해서 서술하나 기술적인 언급을 추가로 하지는

않으므로 필자의 모듈에 대한 사용법이 필요한 사용자는 제 5 장 까지만 읽어도 무방하다.

일단 서술하기에 앞서서 여기까지 읽은 사용자들을 향해 경의를 보낸다. 20 장 가까이 되는

보고서를 정독하는 것은 쉬운 일이 아니다. 개인적인 서술로는 사실 좀 더 package 를 최적화시키고,

좀 더 쓰기 쉽게 만들고 싶은 마음도 있지만(gui interface 를 붙인다던지..) 역시나 쉬운 일이 아니다.

최적화도 아쉬운 부분이 있다. 특히 5.1.2 장 와서는 delay 문제가 심심찮게 발생하니 참조하도록 하자

해당 모듈은 deep learning 을 사용한 객체 검출 실질적인 프로젝트에(yolo 가 사람의 눈에 해당하니

사람의 손발을 대신할 수 있는 임베디드 모듈 관련이 필자의 의도에 가장 가깝다) 조금 더 쉽게

사용해 볼 수 있게 하는 용도면 충분하다고 생각한다. 아니면 진짜 고수들은 커스터마이징해서 더

유용하게 사용할 수도 있을 것이다.

해당 모듈 제작 기간 및 보고서 작성 까지는 근 3 개월 가량이 걸렸다. 이전까지는 ros2, yolo 등등

생전 처음듣는 얘기라서 쉽지 않았지만 그래도 경험을 바탕으로 어떻게든 방법을 찾아냈다.

지속 가능성 있는 방향인지는 아직까지는 모르겠다.

중간중간에 영어가 섞인 부분은(사실 오타자도 많을 것이다.) 필자의 짧은 영어에 대해 아는 척하기

위한 장치라 보다는 이렇게 쓰는 것이 좀 더 가독성을 높이기 위한 방법이라고 생각해서 해당

스타일로 작성하였다.

해당 보고서를 작성하는 글쓴이를 필자라고 표현하였는데 딱히 이유는 없다. 마음에 들지 않는다면

글을 직접 쓰기 바란다. (농담이다)

여러 서비스 코드들 또한 제공하였는데 해당 git 에 보면 그 외에도 여러가지 코드가 있을 것이다.

아니면 서비스 코드의 일부 중간 코드도 업로드 되어 있으니 필요에 따라 쓰면 된다.

본 모듈에는 license 가 걸려있는데 상업성이 아닌 단순한 학생용이다. 제작자 이름만 기억하고

마음껏 쓰길 바란다!!

이정도로 정리하였으면 해당 보고서에 대한 정리를 마칠 수 있을 것 같고, 해당 모듈을 만드는 동안

배려해 준 병용이 형, 소강이, 한솔이 누나 그리고 김대현 박사님이랑 박찬호 박사님께 감사드립니다!

## 참고 자료

참조 링크 모음 : [기술 연구 일지\(현장 실습팀\) - Google Docs](#)