  
  
**aParkigs-AI System**

SenTerm과 Jetson Nano를 이용한 주차 공간 및   
QR 코드를 통한 주차 위치 확인 시스템

강원준, 제서윤, 황재하

**목차**

[1. 프로젝트 목표 4](#_Toc63690940)

[2. 프로젝트 개발 환경 4](#_Toc63690941)

[2.1. Jetson Nano 환경 설정 4](#_Toc63690942)

[2.1.1. Insert Micro SD Card 4](#_Toc63690943)

[2.1.2 Jetson Nano Start 6](#_Toc63690944)

[2.1.3. Install OpenCV 4.0.1 9](#_Toc63690945)

[2.1.4. Install Yolo 11](#_Toc63690946)

[2.2. Android Studio 환경 설정 (UI를 App으로 만들려고 했으나 완성 못함) 12](#_Toc63690947)

[3. 프로젝트 적용기술 13](#_Toc63690948)

[3.1. aParkings-AI 흐름도 13](#_Toc63690949)

[3.2. 적용기술 13](#_Toc63690950)

[3.2.1. UDP Socket 통신 13](#_Toc63690951)

[3.2.2. 이미지 처리 13](#_Toc63690952)

[3.2.3. UI (User Interface) 14](#_Toc63690953)

[3.2.4. QR code 14](#_Toc63690954)

[4. 프로젝트 시나리오 14](#_Toc63690955)

[4.1. aParkings-AI 14](#_Toc63690956)

[4.2. QR code 15](#_Toc63690957)

[5. 프로젝트 결과 16](#_Toc63690958)

[5.1. 사진 전송 16](#_Toc63690959)

[5.2. 이미지 처리 17](#_Toc63690960)

[5.3. UI 생성 17](#_Toc63690961)

[6. 프로젝트 개선사항 18](#_Toc63690962)

[6.1. SenTerm과 Jetson Nano의 UDP 통신 구축 18](#_Toc63690963)

[6.2. QR 코드 테이블 19](#_Toc63690964)

[6.3. Yolov3의 이미지 처리 속도 향상 19](#_Toc63690965)

[7. 코드 19](#_Toc63690966)

[7.1. SenTerm에서 촬영한 사진을 저장한 PC(client)에서 Jetson Nano(server)로 이미지(사진) 전송 19](#_Toc63690967)

[7.2. Jetson Nano는 Client(PC)로부터 전송을 받은 이미지 처리 20](#_Toc63690968)

[7.3. UI Client(PC)가 Server(Jetson Nano)로 ‘start’를 전송하면 Server에서 Yolo 수행 후 결과를 UI Client에게 전송 22](#_Toc63690969)

[7.4. Server(Jetson Nano)에서 Yolo 수행 결과값을 받은 UI Client(PC)는 pygame과 pandas를 이용해 UI 생성 23](#_Toc63690970)

# 1. 프로젝트 목표

aParkings-AI System은 SenTerm이 촬영한 영상을 Jetson Nano로 전달하여 객체 탐지 알고리즘(Yolo v3 알고리즘)을 통해 주차구역내 자동차의 유무를 파악 후 빈 주차 공간을 알려주는 시스템이다. 이 시스템의 부가적인 서비스로는 QR코드를 통해 주차한 위치를 대략적으로 알 수 있는 애플리케이션도 포함되어 있다.

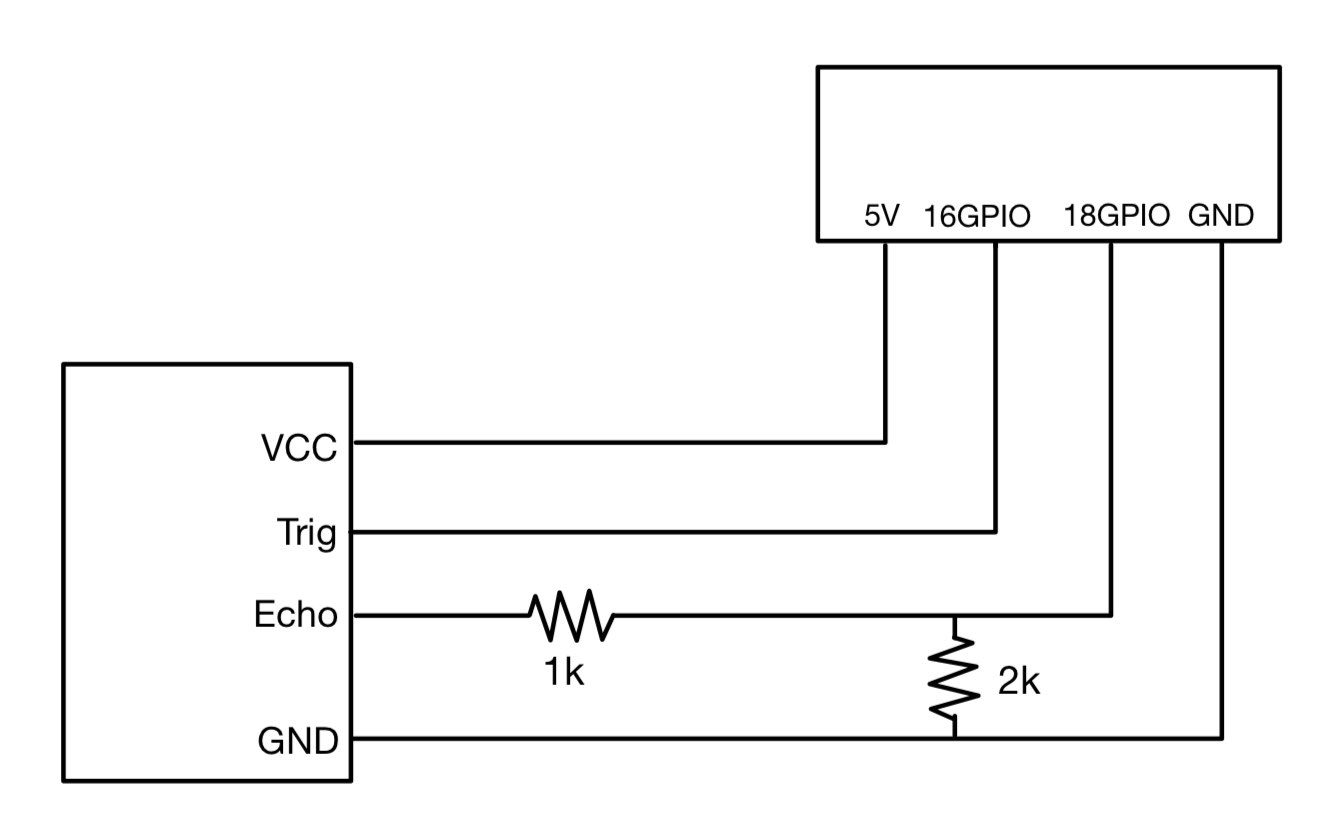
이 프로젝트는 사용자의 편리한 주차 및 주차한 위치를 빠르게 찾을 수 있는 서비스를 제공하는 것이 목표이다.

# 2. 프로젝트 개발 환경

|  |  |  |
| --- | --- | --- |
| 개발환경 | 구분 | 상세 내용 |
| S/W  개발환경 | OS | Windows 7, Ubuntu 18.04 LTS |
| 개발환경(IDE) | pyCharm |
| 개발도구 | OpenCV 4.0.1, GPIO |
| 개발언어 | C, Python3 |
| 기타사항 | Yolo v3 |
| H/W  개발환경 | 디바이스 | Jetson Nano, HC-SR04 |
| 통신 | TCP/IP Socket |
| 언어 | C, Python3 |

2.1 HC-SR04 센서 연결

2.1.1 센서 연결

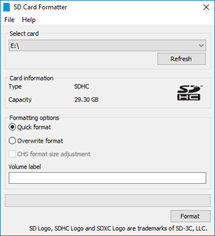


HC-SR04의 Echo 전압은 5V이고, Jetson Nano의 GPIO동작전압은 3.3V이기 때문에 전압 강하를 통해 3.3V를 흘려주어야 한다.

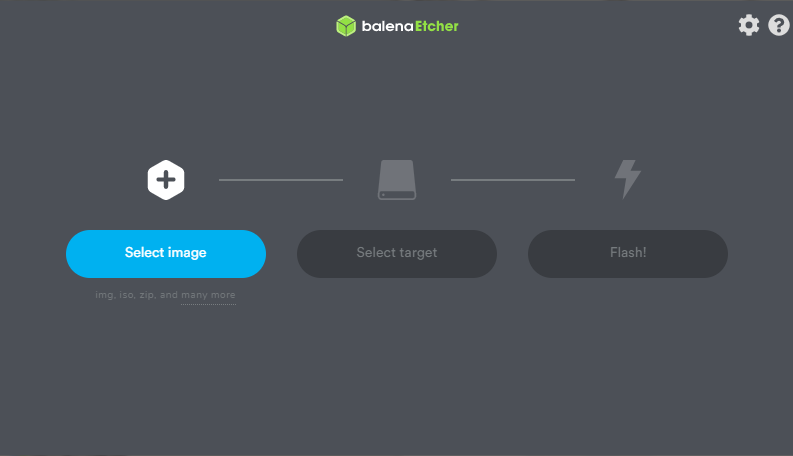
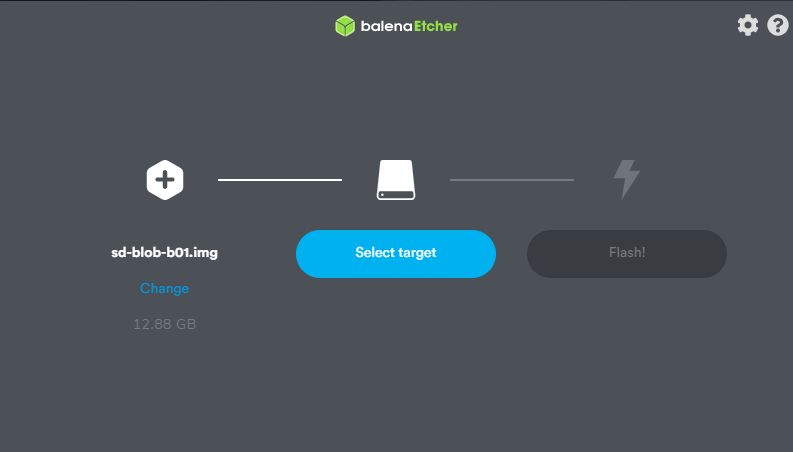
## 2.2 Jetson Nano 환경 설정

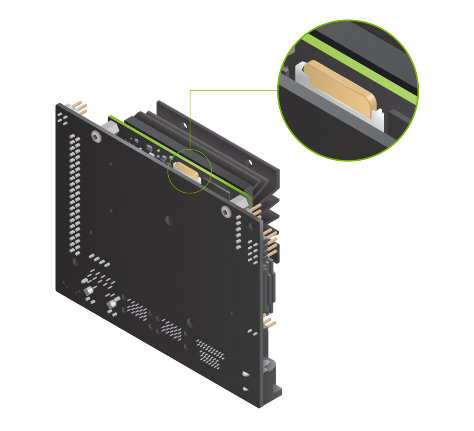
### 2.2.1. Insert Micro SD Card

1. Micro SD Card Format

SD Card Formatter를 다운로드하고 설치한다.  
: <https://www.sdcard.org/downloads/formatter_4/eula_windows/>  
  
SD 카드 드라이브를 선택한 후 Quick format을 선택한다. Volume label은 비워 두고 Format을 클릭한 후 Yes를 눌러 포맷을 시작한다.

2. Write image file to Micro SD Card

이미지(JetPack)를 다운로드 한다.  
: <https://developer.nvidia.com/jetson-nano-sd-card-image-r3231>  
Etcher를 다운로드하여 설치한다  
: <https://www.balena.io/etcher>  
  
Select Image를 클릭해 다운로드한 이미지 압축파일을 선택한 후, PC에 Micro SD Card를 삽입한다.  
  
Select Target을 클릭한 후 Micro SD Card에 해당하는 드라이브를 선택하고 Flash를 클릭한다. (약 10~30분 소요)

3. Insert Micro SD Card into Jetson Nano  


### 2.2.2 Jetson Nano Start

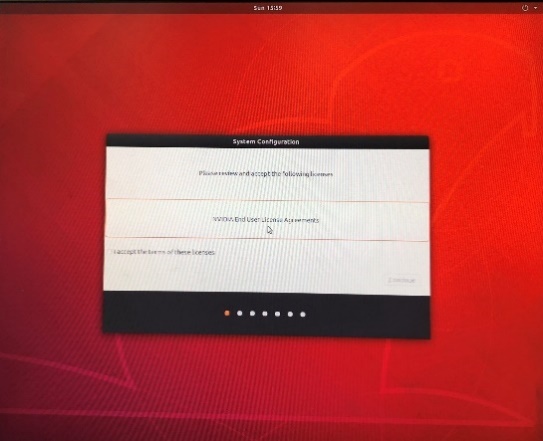
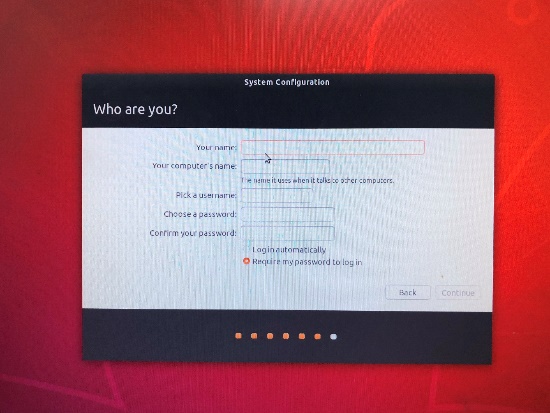
1. Jetson Nano Power Connection

i. Micro USB 단자를 통한 전원 (5V 2A, 저전력 모드)  
ii. Barrel Jack 단자를 통한 전원 (5V 4A, j48핀에 점퍼핀 연결, 일반 모드)  
iii. GPIO Header를 통한 전원 (2개의 5V 3A핀 존재, 총 6A)

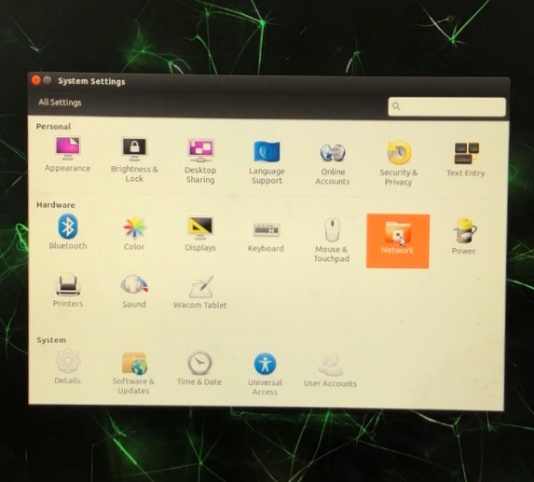
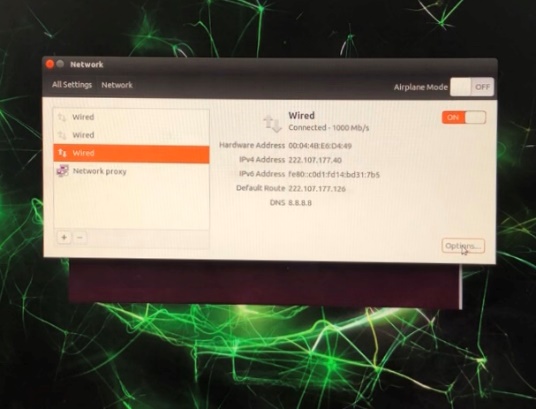
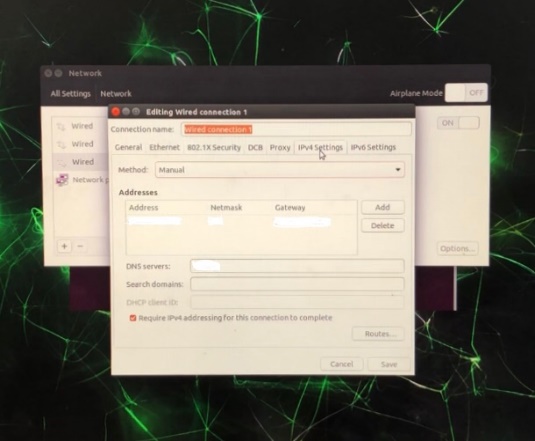
2. Connecti on Port

마우스, 키보드, 모니터, 전원선, LAN 선, J48핀 연결  


3. Start

  
[Continue]를 누른다.  
  
정보 입력 창이 뜨면 정보를 입력한다.

4. Assign IP

  
[Network] 폴더를 누른다.  
  
인터넷이 연결되어 있는 이더넷(wired)을 선택한다.  
  
[IPv4 Settings]를 누른 후 Method를 Manual로 설정하고 Address 정보를 기입한다.

### 2.2.3. Install OpenCV 4.0.1

1. Delete the previously saved OpenCV

|  |
| --- |
| $ pkg-config --modversion opencv  $ sudo apt-get remove libopencv\*  $ sudo apt-get autoremove |

2. Update & Upgrade

|  |
| --- |
| $ sudo apt-get update  $ sudo apt-get upgrade |

3. Install package

|  |
| --- |
| $ sudo apt-get .install build-essential cmake  $ sudo apt-get install pkg-config  $ sudo apt-get install libjpeg-dev libtiff5-dev libjasper-dev libpng12-dev  $ sudo apt-get install libavcodec-dev libavformat-dev libswscale-dev libxvidcore-dev libx264-dev libxine2-dev  $ sudo apt-get install libv4l-dev v4l-utils  $ sudo apt-get install libgstreamer1.0-dev libgstreamer-plugins-base1.0-dev  $ sudo apt-get install libqt4-dev  $ sudo apt-get install mesa-utils libgl1-mesa-dri libqt4-opengl-dev  $ sudo apt-get install libatlas-base-dev gfortran libeigen3-dev  $ sudo apt-get install python2.7-dev python3-dev python-numpy python3-numpy |

4. Create opencv directory

|  |
| --- |
| $ mkdir opencv  $ cd opencv |

5. Download OpenCV 4.0.1

|  |
| --- |
| $ wget -O opencv.zip https://github.com/opencv/opencv/archive/4.0.1.zip  $ unzip opencv.zip  $ wget -O opencv\_contrib.zip https://github.com/opencv/opencv\_contrib/archive/4.0.1.zip  $ unzip opencv\_contrib.zip |

6. Create build directory

|  |
| --- |
| $ cd opencv-4.0.1/  $ mkdir build  $ cd build |

7. Build configuration

|  |
| --- |
| $ cmake -D CMAKE\_BUILD\_TYPE=RELEASE \  -D CMAKE\_INSTALL\_PREFIX=/usr/local \  -D WITH\_TBB=OFF \  -D WITH\_IPP=OFF \  -D WITH\_1394=OFF \  -D BUILD\_WITH\_DEBUG\_INFO=OFF \  -D BUILD\_DOCS=OFF \  -D INSTALL\_C\_EXAMPLES=ON \  -D INSTALL\_PYTHON\_EXAMPLES=ON \  -D BUILD\_EXAMPLES=OFF \  -D BUILD\_TESTS=OFF \  -D BUILD\_PERF\_TESTS=OFF \  -D WITH\_QT=ON \  -D WITH\_GTK=OFF \  -D WITH\_OPENGL=ON \  -D OPENCV\_EXTRA\_MODULES\_PATH=../../opencv\_contrib-4.0.1/modules \  -D WITH\_V4L=ON \  -D WITH\_FFMPEG=ON \  -D WITH\_XINE=ON \  -D BUILD\_NEW\_PYTHON\_SUPPORT=ON \  -D PYTHON2\_INCLUDE\_DIR=/usr/include/python2.7 \  -D PYTHON2\_NUMPY\_INCLUDE\_DIRS=/usr/lib/python2.7/dist-packages/numpy/core/include/ \  -D PYTHON2\_PACKAGES\_PATH=/usr/lib/python2.7/dist-packages \  -D PYTHON2\_LIBRARY=/usr/lib/x86\_64-linux-gnu/libpython2.7.so \  -D PYTHON3\_INCLUDE\_DIR=/usr/include/python3.5m \  -D PYTHON3\_NUMPY\_INCLUDE\_DIRS=/usr/lib/python3/dist-packages/numpy/core/include/ \  -D PYTHON3\_PACKAGES\_PATH=/usr/lib/python3/dist-packages \  -D PYTHON3\_LIBRARY=/usr/lib/x86\_64-linux-gnu/libpython3.5m.so \  ../ |

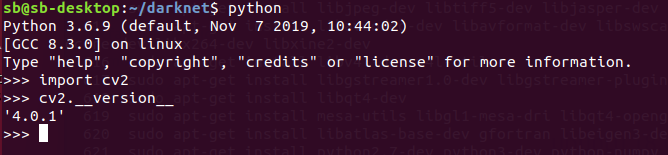
8. Compile

|  |
| --- |
| $ make -j4 |

9. Install OpenCV

|  |
| --- |
| $ sudo make install  $ sudo sh -c echo '/usr/local/lib/' > sudo /etc/ld.so.conf.d/opencv.conf  $ sudo ldconfig |

10. Check OpenCV



### 2.2.4. Install Yolo

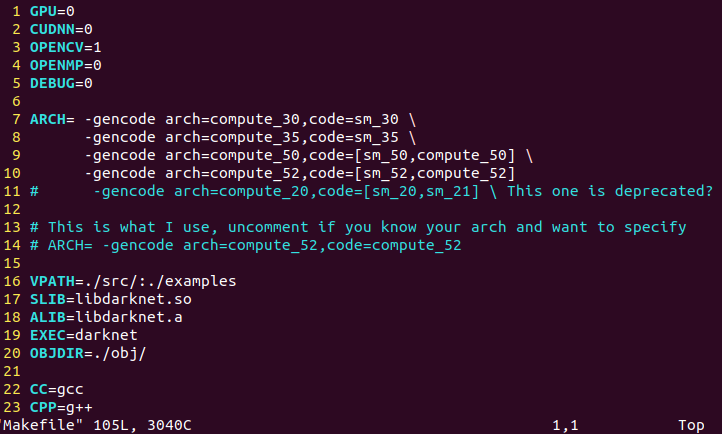
1. Darknet git clone

|  |
| --- |
| $ git clone https://github.com/pjreddie/darknet |

2. Change Makefile

|  |
| --- |
| $ cd darknet  $ sudo vi Makefile |

OPENCV=1, CDNN=1, GPU=1로 변경



3. Compile

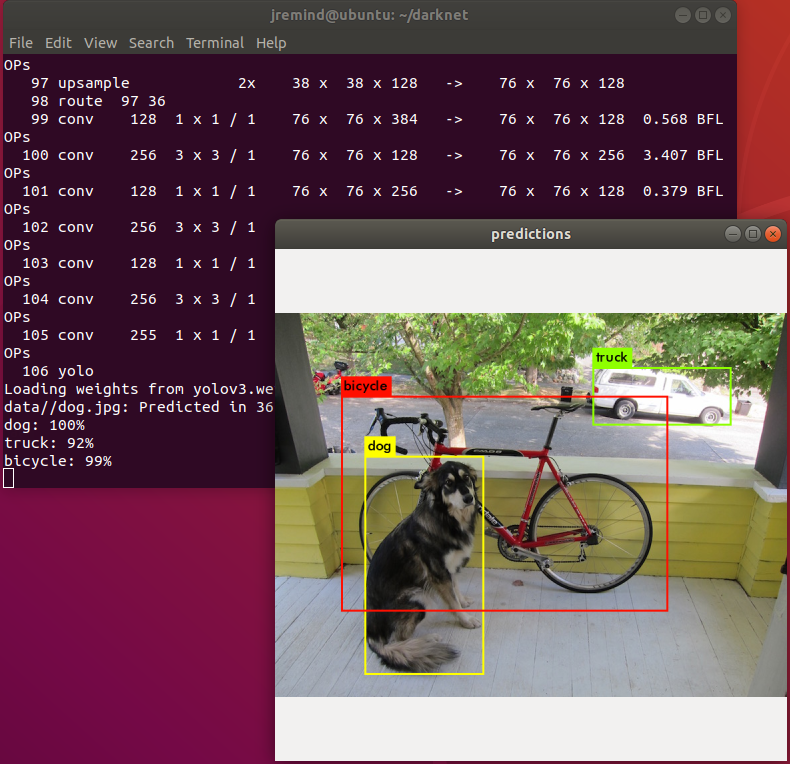
|  |
| --- |
| make |

4. Download trained files from YOLO Darknet.

|  |
| --- |
| $ wget https://pjreddie.com/media/files/yolov3.weights |

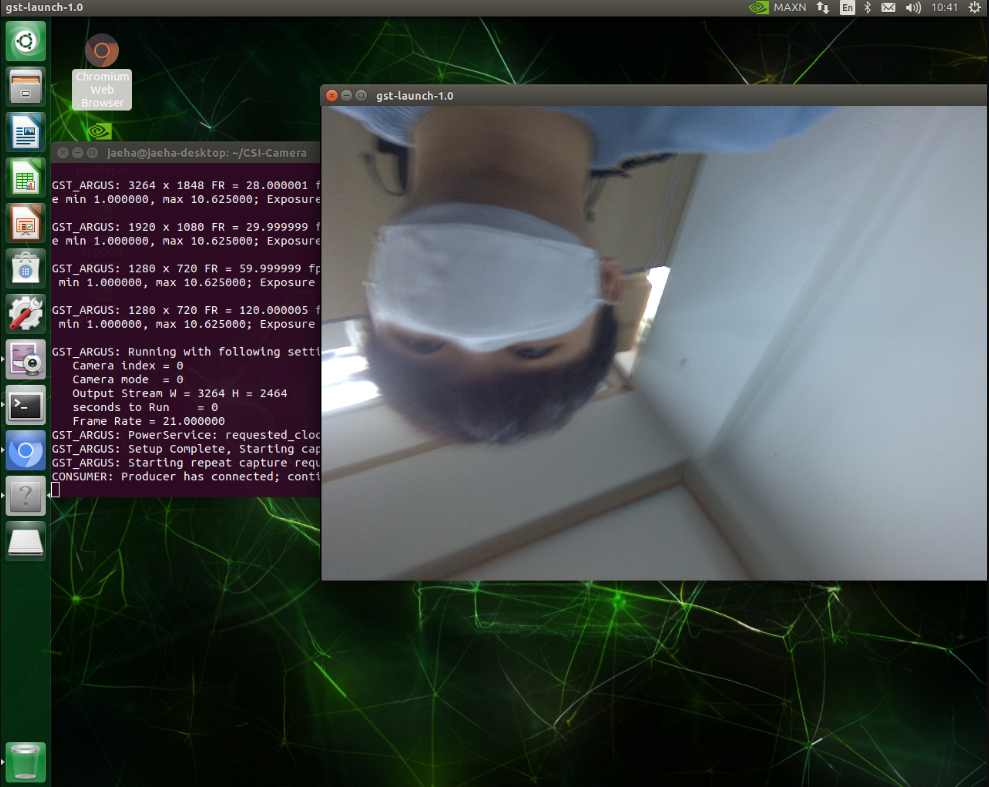
5. Run image example

|  |
| --- |
| $ ./darknet detect cfg/yolov3.cfg yolov3.weights data/dog.jpg |



2.2.5. Install CSI-Camera

|  |
| --- |
| $ git clone <https://github.com/JetsonHacksNano/CSI-Camera.git>  $ cd CSI-Camera  $ gst-launch-1.0 nvarguscamerasrc sensor\_mode=0 ! 'video/x-raw(memory:NVMM),width=3820, height=2464, framerate=21/1, format=NV12' ! nvvidconv flip-method=0 ! 'video/x-raw,width=960, height=616' ! nvvidconv ! nvegltransform ! nveglglessink -e |



## 2.3. Android Studio 환경 설정

# 3. 프로젝트 적용기술

## 3.1. aParkings-AI 흐름도



- 초음파 센서(HC-SR04)를 통해 실시간 물체와의 거리를 탐지하고, 거리의 불연속적 변화 탐지기 Event 발생

- Event 발생시 R-Cam으로 주차장 정면 사진 촬영

-Jetson Nano에서 OpenCV를 이용하여 이미지를 자르고, 잘려진 이미지를 이용해 Yolo를 수행하여 물체가 자동차인지 아닌지 판별

-TCP/IP통신으로 데이터 전송

-Android App으로 사용자에게 주차정보를 제공

## 3.2. 적용기술

## 3.2.1 센서 정보

-브레드 보드 제작

-GPIO 통신

### 3.2.2. UDP Socket 통신

- PC → Jetson Nano  
 ∙ SenTerm에서 촬영한 이미지를 PC를 통해 Jetson Nano로 전송

- Jetson Nano → PC  
 ∙ 이미지 처리에 대한 결과(주차구역 내 주차유무에 대한 정보)를 PC로 전송

### 3.2.3. 이미지 처리

- OpenCV  
 ∙ 이미지의 특정 위치(주차구역)를 자른 이미지 파일 저장

- Yolov3  
 ∙ 자른 이미지 파일이 자동차인지 판단하여 자동차 유무 결정  
 ∙ 자동차 유무에 대한 리스트 저장

### 3.2.4. UI (User Interface)

3.2.5. Android Application

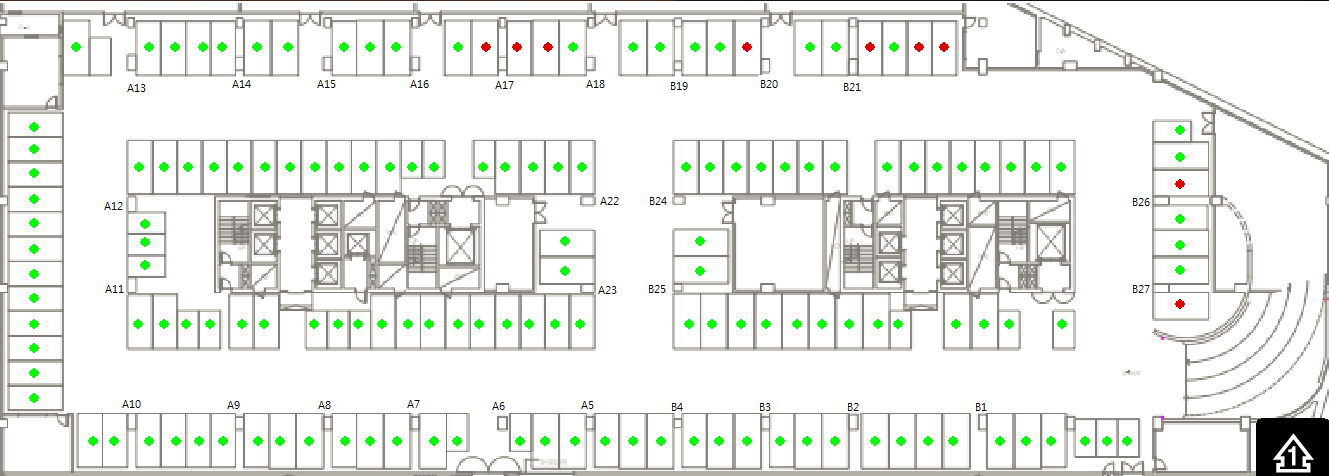
# 4. 프로젝트 시나리오

## 4.1. aParkings-AI

1. 운전자(user)에게 주차장 도면(UI)을 통해 빈 주차 구역을 보여준다. (초록색 : 주차가능, 빨간색 : 주차불가능)



2. 운전자가 빈 자리에 주차하면 일정 시간 후 주차장 도면에 주차 유무에 대한 표시가 업데이트된다.



(운전자가 차량을 빼면 빨간색은 다시 초록색으로 바뀐다.)

3. 주차장 도면에 자동차 유무가 반영되면 주차장 정보 테이블을 출력한다.

|  |  |
| --- | --- |
| 주차 가능한 차량 수 | 주차구역 |
|  |  |
| 입차 | 출차 |
|  |  |

## 4.2. QR code

1. 운전자가 주차 후 근처 기둥에 부착된 QR 코드를 휴대폰으로 촬영한다.



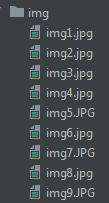
2. 운전자의 휴대폰에 주차 위치 정보를 담은 도면이 출력된다.



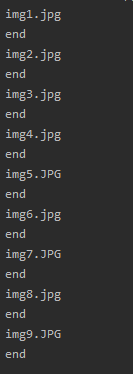
# 5. 프로젝트 결과

## 5.1. 사진 전송

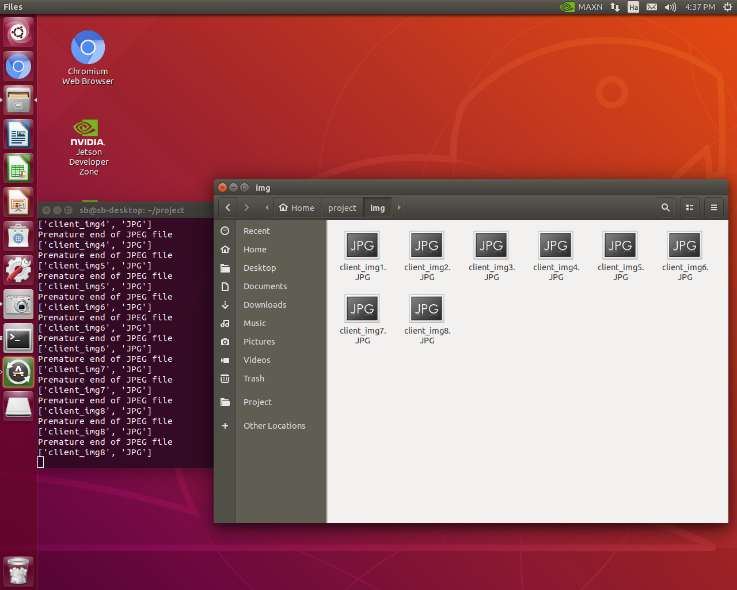
- SenTerm에서 촬영한 사진(이미지)를 SD card 이용해 PC에 저장한다.



- Client(PC)가 Server(Jetson Nano)로 저장한 이미지를 전송한다.

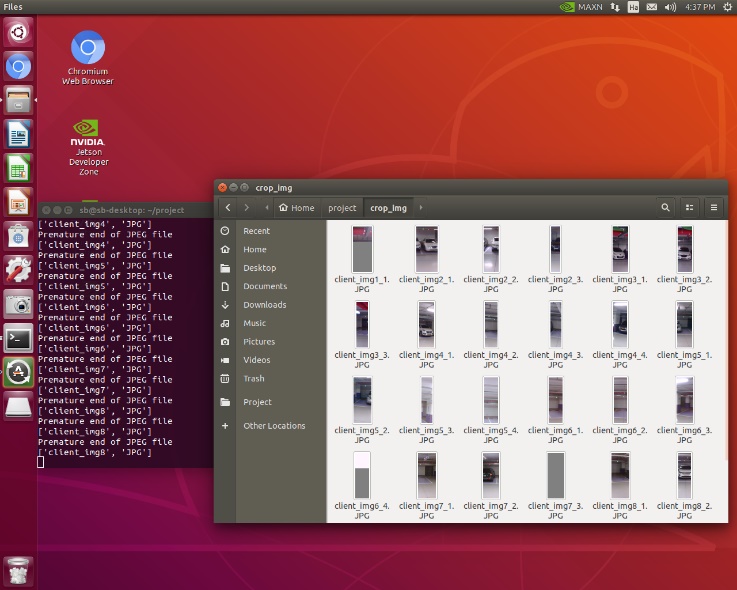


- Server(Jetson Nano)는 전송을 받은 이미지를 저장한다.

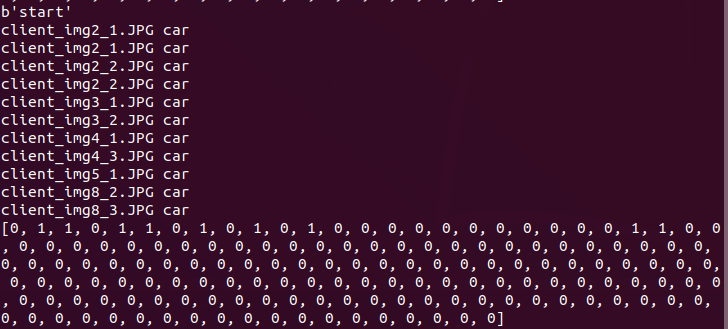


## 5.2. 이미지 처리

- Server(Jetson Nano)에서 이미지를 주차 구역에 맞게 자른다.  
 (주차 구역에 대한 좌표는 텍스트 파일에 저장되어 있다.)



- UI Client(PC)가 ‘start’를 Server(Jetson Nano)로 보내면 Server(Jetson Nano)는 Yolo 수행 후 주차 상황에 대한 정보를 UI Client로 전송한다. (여기서 말하는 정보는 주차 구역 내 자동차 유무에 관한 것이다.)

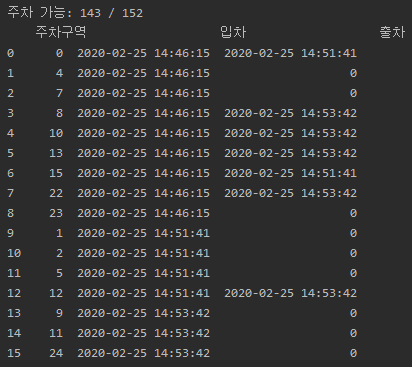


## 5.3. UI 생성

- Server로부터 정보를 받은 UI Client(PC)는 pygame 모듈을 이용해 주차장 도면에 정보를 표시한다.  
 (여기서 말하는 정보는 주차 구역 내 자동차 유무에 관한 것이다.)

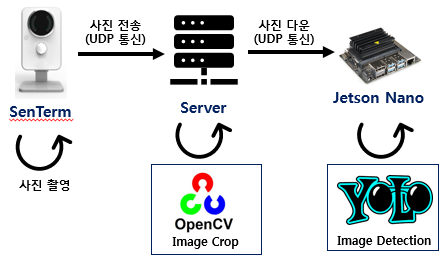


- Server로부터 정보를 받은 UI Client(PC)는 pandas 모듈을 이용해 테이블 형태로 정보를 출력한다.



# 6. 프로젝트 개선사항

## 6.1. SenTerm과 Jetson Nano의 UDP 통신 구축



- 현재 SenTerm에서 촬영한 사진을 SD card를 통해 PC로 옮긴 후 전송

- SenTerm이 server로 사진을 전송하면 Jetson Nano는 server에서 사진을 전송을 받아 Yolo를 수행하는 방식으로   
 수정 필요

- SenTerm과 연결된 server는 C언어로 이루어져 있음

## 6.2. QR 코드 테이블



- QR 코드를 촬영하면 촬영한 순서대로 데이터를 기록할 수 있는 애플리케이션 제작이 필요하다.

- 데이터 기록 방식은 Queue 형식으로, 일정 기간이 지난 오래된 데이터나 저장된 데이터 수가 많은 경우 가장  
 오래된 데이터부터 저장하는 형식이다.

## 6.3. Yolov3의 이미지 처리 속도 향상

- 이미지 처리 속도가 Raspberry pi camera로 촬영하여 실시간으로 처리하는 속도보다 느리다.

- SenTerm 보드에 OpenCV를 설치할 수 없기 때문에 영상에서 Yolo 수행이 불가능하므로 다른 방안을 모색해야   
 할 것 같다.

# 7. 코드

## 7.1. SenTerm에서 촬영한 사진을 저장한 PC(client)에서 Jetson Nano(server)로 이미지(사진) 전송

- client\_Senterm.py (server\_Senterm.py가 실행되어 있어야 함)

import socket  
import os  
import time  
  
UDP\_IP = '222.107.177.42'  
UDP\_PORT = 6667  
  
path = './img'  
img\_list = os.listdir(path)  
img\_list.sort()  
  
clientSock = socket.socket(socket.AF\_INET, socket.SOCK\_DGRAM)  
  
for file\_name in img\_list:  
 clientSock.sendto(file\_name.encode(), (UDP\_IP, UDP\_PORT))  
 # print(file\_name)  
 data\_transferred = 0  
 with open('./img/' + file\_name, 'rb') as f:  
 print(file\_name)  
 try:  
 data = f.read(1024)  
 while data:  
 data\_transferred += clientSock.sendto(data, (UDP\_IP, UDP\_PORT))  
 data = f.read(1024)  
 time.sleep(0.0001)  
 print("sleep")  
 except Exception as e:  
 print(e)  
 clientSock.sendto('end'.encode(), (UDP\_IP, UDP\_PORT))  
 print('end')

## 7.2. Jetson Nano는 Client(PC)로부터 전송을 받은 이미지 처리

- server\_SenTerm.py

import socket  
import imgCrop  
  
UDP\_IP = '222.107.177.42'  
UDP\_PORT = 6667  
  
serverSock = socket.socket(socket.AF\_INET, socket.SOCK\_DGRAM)  
serverSock.bind((UDP\_IP, UDP\_PORT))  
crop = imgCrop.CROP()  
  
def s():  
 while True:   
 data\_transferred = 0  
 title, addr = serverSock.recvfrom(1024)  
 print ("Messsage: ", title)  
 if not title:  
 print("Error")  
 else :  
 print("else")  
 data, addr = serverSock.recvfrom(1024)  
 with open('./img/'+ title.decode(), 'wb') as f:  
 try:  
 while data:  
 f.write(data)  
 data\_transferred += len(data)  
 data, addr = serverSock.recvfrom(1024)  
 if data == b'end':  
 print('end')  
 crop.main()  
 s()  
 print("while end")  
 except Exception as e:  
 print(e)  
  
s()

- imgCrop.py

import os  
  
  
class CROP:  
 def main(self):  
 # Color  
 red = (0, 0, 255)  
  
 # Read image file list in directory  
 path\_img = './img' # Set image directory path  
 img\_file\_list = os.listdir(path\_img) # Read file list in path  
 img\_file\_list.sort()  
  
 # Store light status (0:Green, 1:Red)  
 light\_list = [0] \* 3  
  
 # ------test code-----------------------------------------  
  
 # Open vector\_file\_list  
 cnt = 0  
  
 vector\_list = []  
 f = open("./vector/vec.txt", "r") # read mode  
  
 line = f.readline()  
  
 vector\_list = line.split()  
 vector\_list = [int(i) for i in vector\_list]  
  
 def extractor(img, cnt):  
 cropped\_img = img[vector\_list[cnt + 1]:vector\_list[cnt + 3], vector\_list[cnt]:vector\_list[cnt + 2]]  
 return cropped\_img  
  
 for j in range(0, len(img\_file\_list)):  
 count = 0  
 while 1:  
 count += 1  
 if cnt > len(vector\_list) - 1:  
 break  
 if vector\_list[cnt] == -1:  
 cnt += 1  
 break  
 image = ("./img/" + str(img\_file\_list[j])) # \_1.JPG  
 img = cv.imread(image, cv.IMREAD\_COLOR)  
 # print(vector\_list[cnt], vector\_list[cnt+1],vector\_list[cnt+2], vector\_list[cnt+3])  
 cv.rectangle(img, (vector\_list[cnt], vector\_list[cnt + 1]),  
 (vector\_list[cnt + 2], vector\_list[cnt + 3]), red, 3)  
 img = extractor(img, cnt)  
 file\_name = img\_file\_list[j].split('.')  
 print(file\_name)  
 file\_name\_path = './crop\_img/' + str(file\_name[0]) + '\_' + str(count) + '.' + str(file\_name[-1])  
 cv.imwrite(file\_name\_path, img)  
 cnt = cnt + 4  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 crop = CROP()  
 crop.main()

## 7.3. UI Client(PC)가 Server(Jetson Nano)로 ‘start’를 전송하면 Server에서 Yolo 수행 후 결과를 UI Client에게 전송

- server\_UI.py

import socket  
import yolo\_object\_detection  
  
UDP\_IP = '222.107.177.42'  
UDP\_PORT = 6668  
  
serverSock = socket.socket(socket.AF\_INET, socket.SOCK\_DGRAM)  
serverSock.bind((UDP\_IP, UDP\_PORT))  
detection = yolo\_object\_detection.DETECT()  
  
while True:  
 data, addr = serverSock.recvfrom(1024)  
 print(data)  
 if data == b'start':   
 parkingList = detection.main()  
 message = ''  
 for i in parkingList:  
 message += str(i)  
 serverSock.sendto(message.encode(), addr)

- yolo\_object\_detection.py

import cv2  
import numpy as np  
import os  
  
class DETECT:  
 def main(self):  
 # Load Yolo  
 net = cv2.dnn.readNet("yolov3.weights", "yolov3.cfg")  
 classes = []  
 with open("coco.names", "r") as f:  
 classes = [line.strip() for line in f.readlines()]  
 layer\_names = net.getLayerNames()  
 output\_layers = [layer\_names[i[0] - 1] for i in net.getUnconnectedOutLayers()]  
 colors = np.random.uniform(0, 255, size=(len(classes), 3))  
  
 # Loading image  
 parking\_list = [0]\*152 # list 길이는 주차 공간 수와 같아야 함.  
 path\_crop\_img = './crop\_img' # Set image directory path  
 img\_list = os.listdir(path\_crop\_img) # Read file list in path  
 img\_list.sort()  
 for l in range(len(img\_list)):  
 img = cv2.imread('./crop\_img/'+img\_list[l])  
 #img = cv2.resize(img, None, fx=0.4, fy=0.4)  
 height, width, channels = img.shape  
 # Detecting objects  
 blob = cv2.dnn.blobFromImage(img, 0.00392, (416, 416), (0, 0, 0), True, crop=False)  
 net.setInput(blob)  
 outs = net.forward(output\_layers)  
 # Showing informations on the screen  
 class\_ids = []  
 confidences = []  
 boxes = []  
 for out in outs:  
 for detection in out:  
 scores = detection[5:]  
 class\_id = np.argmax(scores)  
 confidence = scores[class\_id]  
 if confidence > 0.5:  
 # Object detected  
 center\_x = int(detection[0] \* width)  
 center\_y = int(detection[1] \* height)  
 w = int(detection[2] \* width)  
 h = int(detection[3] \* height)  
 # Rectangle coordinates  
 x = int(center\_x - w / 2)  
 y = int(center\_y - h / 2)  
 boxes.append([x, y, w, h])  
 confidences.append(float(confidence))  
 class\_ids.append(class\_id)  
 indexes = cv2.dnn.NMSBoxes(boxes, confidences, 0.5, 0.4)  
 #print(indexes)  
 font = cv2.FONT\_HERSHEY\_PLAIN  
 for i in range(len(boxes)):  
 if i in indexes:  
 x, y, w, h = boxes[i]  
 label = str(classes[class\_ids[i]])  
 print(img\_list[l], label)  
 if label == 'car':  
 parking\_list[l] = 1  
 # color = colors[i]  
 # cv2.rectangle(img, (x, y), (x + w, y + h), color, 2)  
 # cv2.putText(img, label, (x, y + 30), font, 3, color, 3)  
 print(parking\_list)  
 return parking\_list  
  
  
 # cv2.imshow(img\_list[l], img)  
 # cv2.waitKey(0)  
 # cv2.destroyAllWindows()  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 detect = DETECT()  
 detect.main()

## 7.4. Server(Jetson Nano)에서 Yolo 수행 결과값을 받은 UI Client(PC)는 pygame과 pandas를 이용해 UI 생성

- client\_UI.py

import pandas as pd  
import datetime  
import time  
import pygame  
from pygame import \*  
from pygame.locals import \*  
import numpy  
import socket  
  
  
UDP\_IP = '222.107.177.42'  
UDP\_PORT = 6668  
clientSock = socket.socket(socket.AF\_INET, socket.SOCK\_DGRAM)  
  
parkingList = [(1180,303), (1180,269), (1180,244), (1180,218), (1180,183),(1180,156),(1180,129),(944,46),(919,46),(894,46),  
 (870, 46),(836,46) ,(810,46), (747,46), (720,46), (695,46), (660,46), (633,46), (573,46), (548,46),  
 (517,46), (486,46), (458,46), (396,46), (370,46), (344,46), (288,46), (257,46), (222,46), (203,46),  
 (175,46),(149,46), (76,46), (34,126), (34,148), (34,172), (34,198), (34,222), (34,248),(34,273),  
 (34,297), (34,323), (34,347), (34,372), (34,397), (93,440), (114,440), (149,440), (176,440),(199,440),  
 (223,440), (258,440), (280,440), (309,440), (345,440), (370,440), (397,440), (434,440), (457,440),(520,440),  
 (545,440), (575,440), (605,440), (633,440), (661,440), (694,440), (719,440), (747,440), (780,440),(807,440),  
 (835,440), (875,440), (901,440), (928,440), (953,440), (1000,440), (1026,440), (1051,440), (1085,440),(1107,440),  
 (1127,440), (1061,166), (1036,166), (1011,166), (985,166), (962,166), (937,166), (914,166), (887,166),(835,166),  
 (809,166), (785,166), (761,166), (736,166), (710,166), (686,166), (700, 240), (700, 270),(689,323),(712,323),  
 (741,323), (766,323), (797,323), (822,323), (849,323), (878,323), (898,323), (956,323), (984,323),(1009,323),  
 (1062,323), (582,166), (558,166), (533,166), (505,166), (483,166), (434,166), (412,166), (390,166),(364,166),  
 (339,166), (315,166), (290,166), (265,166), (238,166), (214,166), (188,166), (164,166), (139,166),(145,223),  
 (145,241), (145,264), (138,323), (164,323), (186,323), (210,323), (242,323), (264,323), (314,323),(338,323),  
 (358,323), (382,323), (408,323), (429,323), (458,323), (479,323), (507,323), (529,323), (555,323),(580,323),  
 (565,240), (565,270)]  
  
screen = pygame.display.set\_mode((1332,477), pygame.DOUBLEBUF)  
  
class PROCESSING:  
 def main(self):  
 pygame.init()  
 img = pygame.image.load("./parkingLotImg/MoonParkingLot.png")  
 img = pygame.transform.scale(img, (1332, 477))  
 screen.blit(img, (0, 0))  
 for i in range(len(parkingList)):  
 pygame.draw.circle(screen, (0, 255, 0), parkingList[i], 5)  
 pygame.display.flip()  
 carTable = [] # 주차 구역, 입차 시간, 출차 시간  
 carArray = numpy.array(carTable)  
 running = True  
  
 while running:  
 for event in pygame.event.get():  
 if event.type == pygame.QUIT: running = False  
 if event.type == pygame.KEYDOWN:  
 if (event.key == pygame.K\_ESCAPE): running = False  
 pygame.display.flip()  
 clientSock.sendto("start".encode(), (UDP\_IP, UDP\_PORT))  
 data, addr = clientSock.recvfrom(1024)  
 yolo\_detection = data.decode() # yolo result  
 yolo\_detection = [int(i) for i in yolo\_detection]  
 check = [0] \* len(yolo\_detection) # in, not in check  
 print("주차 가능:", len(parkingList) - sum(yolo\_detection), "/", len(parkingList))  
  
 for i in range(len(yolo\_detection)):  
 carArray = numpy.array(carTable)  
 if yolo\_detection[i]:  
 if len(carTable) == 0:  
 check[i] = 1  
 else:  
 for j in range(len(carTable)):  
 carArray = numpy.array(carTable)  
 if str(i) not in carArray[:, 0]:  
 check[i] = 1  
 elif carArray[:, 0][j] == str(i) and carArray[:, 2][j] == str(0):  
 check[i] = 0  
 elif carArray[:, 0][j] == str(i) and carArray[:, 2][j] != str(0):  
 check[i] = 1  
 else:  
 for j in range(len(carTable)):  
 if carTable[j][0] == i and carTable[j][2] == 0:  
 now = datetime.datetime.now()  
 carTable[j][2] = now.strftime('%Y-%m-%d %H:%M:%S')  
 pygame.draw.circle(screen, (0, 255, 0), parkingList[i], 5)  
 # cv2.circle(img, parkingList[i], 5, (0, 255, 0), -1) # Create green circle  
 for i in range(len(check)):  
 if check[i]:  
 now = datetime.datetime.now()  
 carTable.append([i, now.strftime('%Y-%m-%d %H:%M:%S'), 0])  
 pygame.draw.circle(screen, (255, 0, 0), parkingList[i], 5)  
 df = pd.DataFrame(carTable, columns=['주차구역', '입차', '출차'])  
 print(df)  
 pygame.quit()  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 processing = PROCESSING()  
 processing.main()