

# data document

---

## python code

---

### server.py

```
import cv2
import socket
import struct
import pickle
import numpy as np
import utils_PyKinectV2 as utils
from pykinect2.PyKinectV2 import *
from pykinect2 import PyKinectV2
from pykinect2 import PyKinectRuntime

ip = '192.168.0.71' # ip 주소
port = 8080 # port 번호

server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM) # 소켓 객체를 생성
server_socket.bind((ip, port)) # 바인드(bind) : 소켓에 주소, 프로토콜, 포트를 할당
server_socket.listen(10) # 연결 수신 대기 상태(리스닝 수(동시 접속) 설정)
print('클라이언트 연결 대기')

# 연결 수락(클라이언트 소켓 주소를 반환)
client_conn, client_addr = server_socket.accept()
print(client_addr) # 클라이언트 주소 출력

# 카메라 선택
kinect = PyKinectRuntime.PyKinectRuntime(PyKinectV2.FrameSourceTypes_Color |
                                         PyKinectV2.FrameSourceTypes_Depth)

# Default: 512, 424
depth_width, depth_height = kinect.depth_frame_desc.width,
kinect.depth_frame_desc.Height
# Default: 1920, 1080
#color_width, color_height = kinect.color_frame_desc.width,
kinect.color_frame_desc.Height

# 인코드 파라미터
# jpg의 경우 cv2.IMWRITE_JPEG_QUALITY를 이용하여 이미지의 품질을 설정
encode_param = [int(cv2.IMWRITE_JPEG_QUALITY), 10]

while True:
    if kinect.has_new_color_frame() and \
        kinect.has_new_depth_frame():
        # streaming data
        #color_frame = kinect.get_last_color_frame()
        depth_frame = kinect.get_last_depth_frame()
```

```

# text data
text = ""Cam_Info_List
-UID: camera0x11
-Name: camera01
-Type: 3D Depth Camera
-Location: Underground Parking(B2)
-Resolution: 512X424
-FrameRate: 10fps
Event_Info_List
-StartTime: 2020:11:10:13:55:34
-EndTime: 2020:11:10:13:55:39
-EventID: 10""

# scolor_img = color_frame.reshape(((color_height, depth_width,
4))).astype(np.uint8)
# data Resize (1080, 1920, 4) into half (540, 960, 4)
#color_img_resize = cv2.resize(color_img, (0, 0), fx=0.5, fy=0.5)

# data 정제
depth_img = depth_frame.reshape(
    ((depth_height, depth_width))).astype(np.uint16)

depth_colormap = cv2.applyColorMap(cv2.convertScaleAbs(
    depth_img, alpha=255/1500), cv2.COLORMAP_JET)

result, depth_frame = cv2.imencode(
    '.png', depth_colormap, encode_param)

# ***pickle.dumps()*** : data 직렬화
data = pickle.dumps(depth_frame, 0)
# print(data)
size = len(data) # 약 100,000 byte
print("Frame Size : ", size)

# 데이터(프레임) 전송
# struct.pack() :
client_conn.sendall(struct.pack(">L 280s", size, text.encode()) + data)

```

## client.py

```
import socket
import cv2
import pickle
import struct

ip = '192.168.0.71' # ip 주소
port = 8080 # port 번호

# 소켓 객체를 생성 및 연결
client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
client_socket.connect((ip, port))
print('연결 성공')

data = b"" # 수신한 데이터를 넣을 변수
payload_size = struct.calcsize(">L 280s") # = 8

while True:
    # 프레임 수신
    while len(data) < payload_size:
        data += client_socket.recv(4096)
    packed_msg_size = data[:payload_size]
    data = data[payload_size:]

    # frame size 출력 : print(msg_size)
    msg_size = struct.unpack(">L 280s", packed_msg_size)[0]
    # 텍스트 출력 : print(msg_text)
    msg_text = struct.unpack(">L 280s", packed_msg_size)[1].decode()
    while len(data) < msg_size:
        data += client_socket.recv(4096)
    frame_data = data[:msg_size]
    data = data[msg_size:]
    print("(CL)Frame Size : {}".format(msg_size)) # 프레임 크기 출력

    # 역직렬화(de-serialization) : 직렬화된 파일이나 바이트를 원래의 객체로 복원하는 것
    # 직렬화되어 있는 binary file로 부터 객체로 역직렬화
    frame = pickle.loads(frame_data, fix_imports=True, encoding="bytes")
    # print(frame)
    frame = cv2.imdecode(frame, cv2.IMREAD_COLOR) # 프레임 디코딩

    # 영상 출력
    cv2.imshow('TCP_Frame_Socket', frame)

    # 1초 마다 키 입력 상태를 받음
    if cv2.waitKey(1) == ord('q'): # q를 입력하면 종료
        break
```

# 사용 된 python module

## Pickle

- [pickle document](#)
- 사용 된 함수
  - **pickle.dumps**(server.py에서 사용):
    - 객체 obj 의 피클된 표현을 파일에 쓰는 대신 bytes 객체로 반환 (직렬화)
  - **pickle.loads**(client.py에서 사용):
    - 객체의 피클 된 표현 *data*의 재구성된 객체 계층 구조를 반환 (역직렬화)
    - data 는 [바이트열류 객체](#)
- 사용 된 예시
  - server.py :
    - **pickle.dumps**(depth\_frame, 0) : cv2.imencode() 에 의해 인코딩 된 frame data(color\_frame) 를 직렬화
      - 직렬화 전 :

```
[[[137]
 [ 80]
 [ 78]
 ...
 [ 66]
 [ 96]
 [130]]
```
      - 직렬화 후 :

```
.....x18\x8c1\xc8Lz\x04\x11A\x95)
\x879\x1c/\x19c\xa5\xb5\xc6\xcd\x9b79\xdd\xed\x90\x84\x9c4\t\xa9!\x89\x9
3y\xe6\xb8\x1e\xd3\x8e\xde;\x99\xc9\xf9\xf9\xdf}\xfd\rP\x07\x82M\xda\x1
4\x0f\xd9&\x00ID\x04\xb61\xc9f\x9a\x1a\x9f\xf8\xc4'8;;\xe3\xe4\xe4\x84\x
6\x1a\xb6\x98["\xd3\xe4\xba\x92\xa38\x99O\xfb\xed\xb7\xb9w\xef\x0e\x11
A\x01I\xe10c,X""\x90\x04\x04\xe1\x00\x84j\x80\x98Z\x90\x99T\x15P8\xc0N\
x14\xc6\x12\x92\x18\xb9"\x02\x92\x7f\xb0Mf\x81\x03/\xd0@\x12\xb2\xa9u00
0a#\x1bID\x00UD\x04i\x03\xc1\xa6\xaa\xb0\x8d$$!\t;\xa9*\xaa\x8aMD`\x1b
\xdbl"\x82\xaa\x02\x15\xc5C\x92(\x1e\x92\x8d~\xf2\x0b?
\xe5\xef.....
```
  - client.py : **pickle.loads**(frame\_data, fix\_imports=True, encoding="bytes") 는 반대

