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


def server():

    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]
    msg = client_conn.recv(2).decode()
    print(msg)
    while msg == "0":
        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 950,000 byte
            print("Frame Size : ", size)

            # 데이터(프레임) 전송
            # struct.pack() :
            client_conn.sendall(struct.pack(
                ">L 280s", size, text.encode()) + data)
            msg = client_conn.recv(2).decode()
            print(msg)
            if msg == "1":  # client 만 종료하고 server 재 실행
                server()
            elif msg == "2":  # 모두 종료
                server_socket.shutdown(socket.SHUT_RDWR)
                client_conn.shutdown(socket.SHUT_WR)


server()
```

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:
    client_socket.send('0'.encode())
    # 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 누르면 client 만 종료
        client_socket.send('1'.encode())
        client_socket.shutdown(socket.SHUT_WR)
    elif cv2.waitKey(1) == ord('x'):  # x를 누르면 둘 다 종료
        client_socket.send('2'.encode())
        client_socket.shutdown(socket.SHUT_WR)
```

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 (역질렬화)
    - 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\x93y\xe6\xb8\x1ei\xd3\x8e\xde;\x99\xc9\xf9\xf99\xdf}\xfd\rP\x07\x82M\xda\x14\x0f\xd9&\x00ID\x04\xb61\xc9f\x9a\x1a\x9f\xf8\xc4'8;;\xe3\xe4\xe4\x84\xd6\x1a\xb6\x98['\xd3\xe4\xba\x92\xa38\x99Ox\xfb\xed\xb7\xb9w\xef\x0e\x11A\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*x03I*xd0@*x12*xb2*xa9*\u0000a#\x1bID\x00UD\x04i\x03\xc1\xa6\xaa\xb0\x8d$$!\t;\xa9*\xaa\x8aMD`\x1b\xdbI"\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

Struct

- sturct document
- 사용된 함수
  - **struct.pack(format, v1, v2)**(server.py에서 사용) :
    - 형식 문자열 *format* 에 따라 패킹 된 *v1* , *v2* ,... 값을 포함하는 **bytes 객체**를 반환
    - 인수는 형식에 필요한 값과 정확히 일치해야 함 (예, 2s = 길이가 2 인 문자열)
  - **struct.unpack(format, 버퍼)**(client.py에서 사용) :
    - 형식 문자열 *format* 에 따라 버퍼에서 압축을 해제
    - 결과는 정확히 하나의 항목을 포함하더라도 **튜플 객체**
    - 버퍼의 크기 (바이트)는에 반영된대로 형식에 필요한 크기와 일치해야함
- 사용 된 예시
  - server.py :
    - **struct.pack**(">L 280s", size, text.encode()) :
      - format : ">L 280s", '>L' = 빅 엔디안 unsigned long // '280s' = 280바이트 char[]
      - v1 : size, pickle 즉 직렬화된 data의 사이즈 (보통 100,000 - 단위 byte)
      - v2 : text(메타 데이터), 텍스트를 encode() 를 사용해 데이터형식(b'')으로 변환함
  - client.py : **struct.unpack**(">L 280s", packed_msg_size) 는 반대
  - data 형태
    - **struct.pack**(">L 280s", size, text.encode()) :
      - b'\x00\x0e\x9e\xdfCam_Info_List\n -UID: camera0x11\n -Name: camera01\n -Type: 3D Depth Camera\n -Location: Underground Parking(B2)\n -Resolution: 512X424\n -FrameRate: 10fps\nEvent_Info_List\n -StartTime: 2020:11:10:13:55:34\n -EndTime: 2020:11:10:13:55:39\n -EventID: 10\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00'
    - **struct.unpack**(">L 280s", packed_msg_size):
      - (958175, b'Cam_Info_List\n -UID: camera0x11\n -Name: camera01\n -Type: 3D Depth Camera\n -Location: Underground Parking(B2)\n -Resolution: 512X424\n -FrameRate: 10fps\nEvent_Info_List\n -StartTime: 2020:11:10:13:55:34\n -EndTime: 2020:11:10:13:55:39\n -EventID: 10\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00')

encode() / decode()

- UTF-8 로 인코딩 및 디코딩
- **인코딩** 예시 : str 객체 -> bytes 객체

```
# 파이썬 기본 인코딩 방식 : UTF-8
사용 : "한글".encode()
결과값 : b'\xed\x95\x9c\xea\xb8\x80'

사용 : "한글".encode("UTF-8")
결과값 : b'\xed\x95\x9c\xea\xb8\x80'

# b'' 형식의 의미 = byte 객체
사용 : type(b'\xed\x95\x9c\xea\xb8\x80') # type 함수는 객체의 type 을 반환
결과값 : <class 'bytes'>
```

- **디코딩** 예시 : bytes 객체 -> str 객체

```
# UTF-8 바이트 객체를 str 객체로 디코딩
사용 : b'\xed\x95\x9c\xea\xb8\x80'.decode()
결과값 : '한글'
```