

Lab 4. Communication and Camera

1. Purpose

Lab 4의 목적은 socket programming을 이해하고 Beaglebone에서 webcam을 통한 remote video streaming system을 구축하며, PC에서 Beaglebone으로의 remote keyboard commander를 설계하는데 있다. 해당 lab에서 구현한 것을 토대로 final project의 robot teleoperation을 설계할 것이다. Robot에 부착된 webcam에서의 data를 PC에서 확인 후 robot arm을 PC에서 teleoperate한다.

- Basic socket programming을 이해한다.
- Remote controller를 위한 코드를 작성한다.
- Webcam을 이용한 streaming system을 테스트해본다.

2. Experiment Sequence

본 lab은 3가지 problem으로 구성되어 있다. 전체적인 lab의 내용은 datagram socket(UDP)를 이용한 server와 client의 개념을 이해하고 구현해보며, PC에서 remote keyboard commander를 구현하여 robot manipulator를 WiFi와 datagram socket을 이용해 teleoperate한다. 마지막으로 FFmpeg를 이용한 Webcam streaming system을 구축한다. 아래는 구현하게 될 3가지 problem이다.

(1) Problem 4A.

Datagram socket(UDP)을 이용한 server와 client간의 통신을 이해하고 구현해본다.

(2) Problem 4B.

PC에서 remote keyboard commander를 구현하여 robot manipulator를 WiFi와 datagram socket을 이용해 teleoperate한다.

(3) Problem 4C.

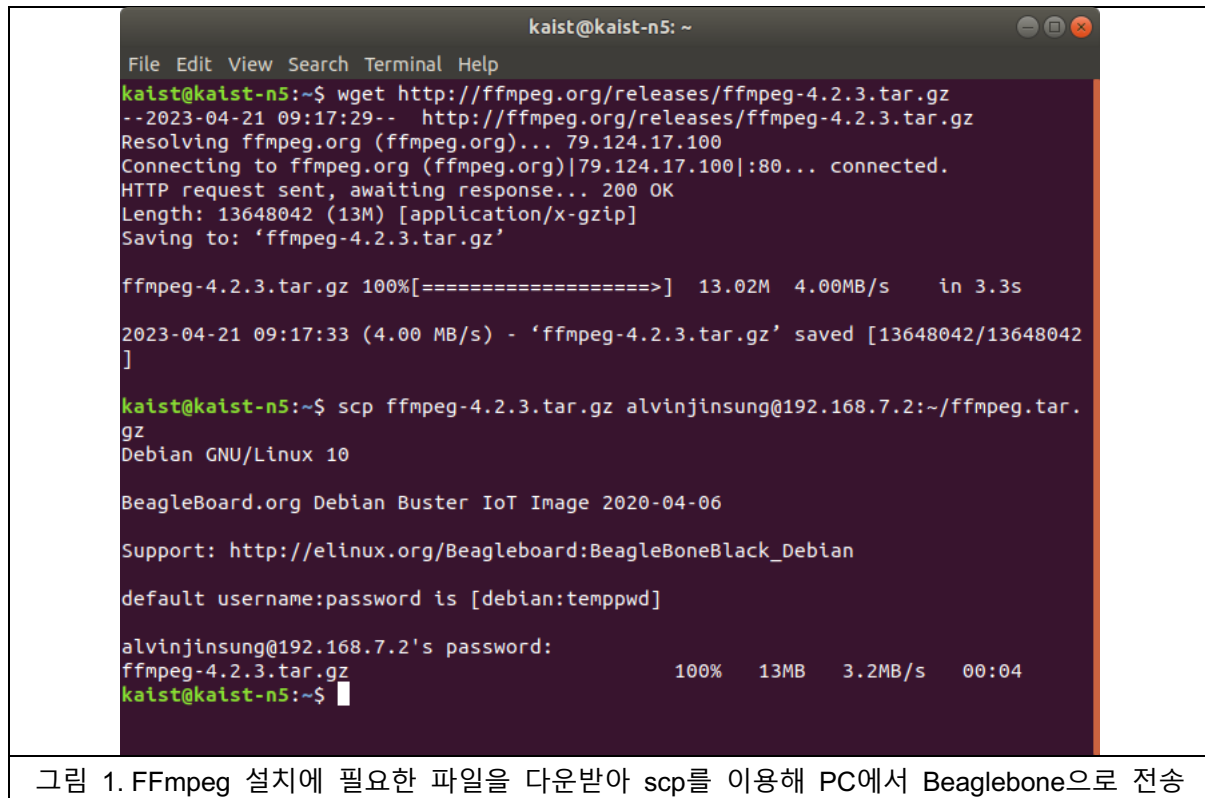
FFmpeg를 이용한 Webcam streaming system을 구축한다. 해당 system은 다음 요소들로 구성되어 있다.

- Beaglebone의 camera (Webcam으로부터 capture된 video를 network로 전송)
- PC의 viewer (Network로부터 video를 수신하여 user에게 display)

3. Experiment Results

Prerequisite

Webcam streaming을 위한 FFmpeg를 사용하기 위해서 higg-version FFmpeg를 설치해야 한다. 이를 위해 설치 파일을 다운받고 PC에서 Beaglebone으로 전송하였다. 아래는 해당 작업을 terminal에서 수행한 모습이다.



이후 Beaglebone에서 아래의 명령어들을 통해 FFmpeg를 설치하였다.

```

$ tar -xvzf ffmpeg.tar.gz
$ cd ffmpeg
$ ./configure && make
$ sudo make install

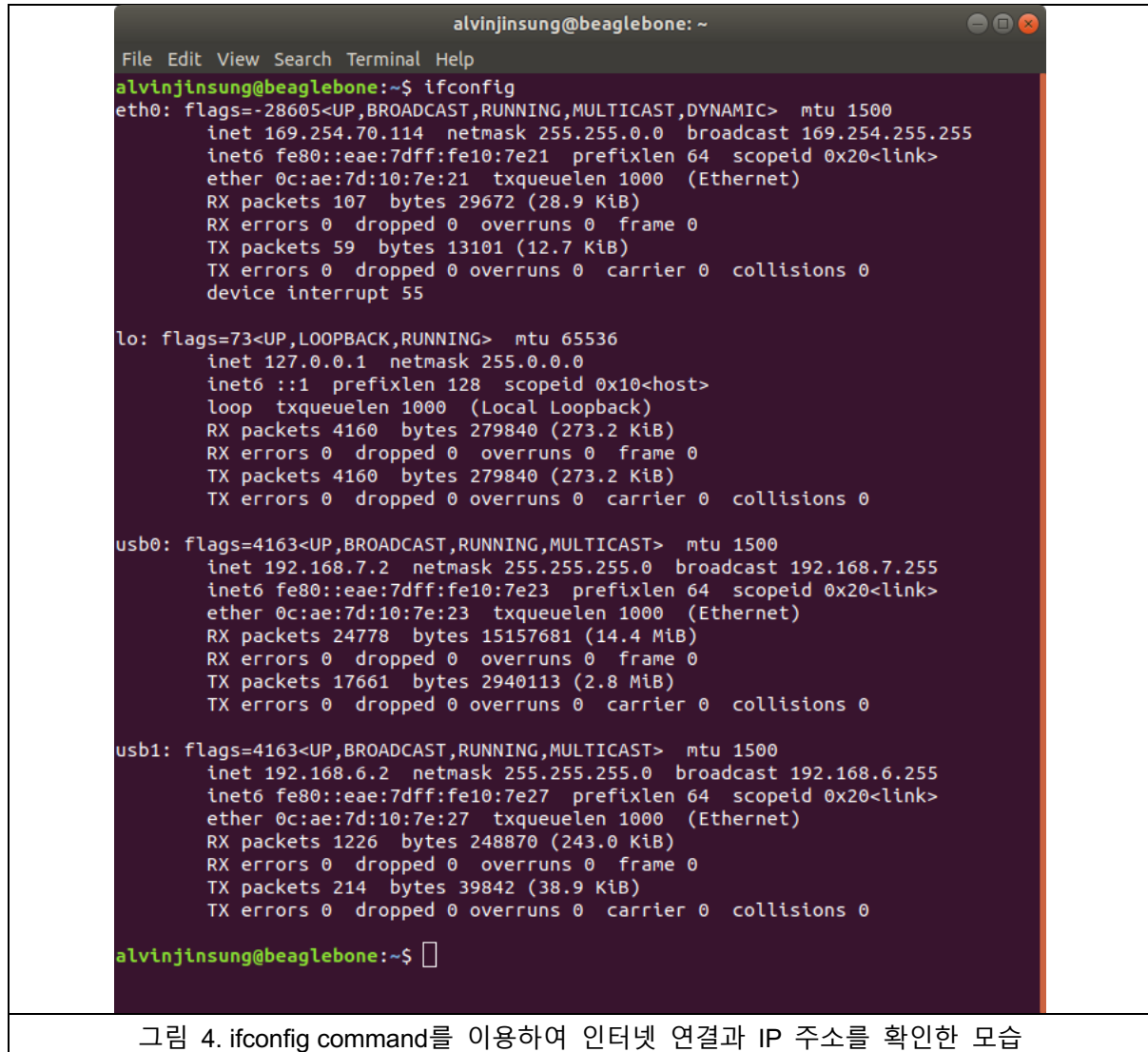
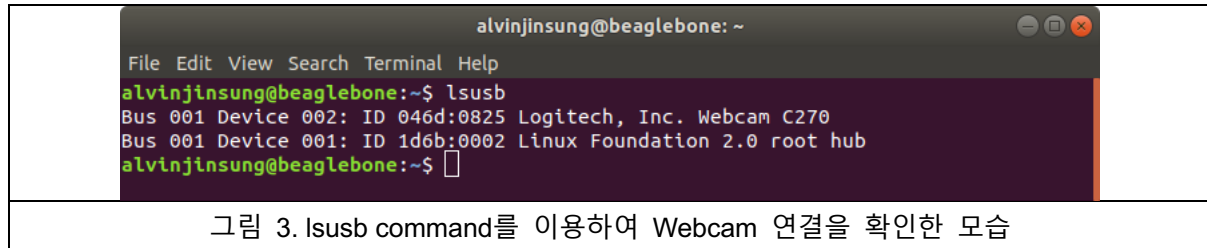
```

다음으로 Beaglebone에 Logitech C270 Webcam과 LAN선을 연결해 인터넷 연결을 설정하였다. 아래는 필요한 부품들을 모두 연결한 모습이다.



그림 2. Beaglebone에 Webcam과 LAN선을 연결한 모습

lsusb command를 이용하여 Webcam이 Beaglebone에 잘 연결되었는지 확인하였으며 ifconfig command를 통해 인터넷 연결 상태 확인 및 IP 주소를 확인하였다.



Problem 4A. Test Stream Socket & Datagram Socket

Stream socket과 Datagram socket을 테스트 해보기 위해 <https://beej.us/guide/bgnet/html/split/> 에서 제공되는 server.c, client.c, listener.c, talker.c 파일을 다운받았다. server.c는 간단한 stream server code로서 "Hello, world!" string을 stream connection에 전송한다. client.c는 간단한 stream client code로서 command line에 지정한 포트에 해당하는 host에 연결한다. listener.c는 Datagram socket에서 지정된 port에서 오는 packet을 수신하는 역할을 수행하며, talker.c는 Datagram socket

에서 지정된 port에 packet을 송신하는 역할을 수행한다. 이를 확인해보기 위해 위의 파일들을 모두 build 한 후 테스트 해보았다. server.c와 listener.c는 Beaglebone에서 실행할 것이기에 cross-compile을 통해 Beaglebone으로 executable을 전송하였으며 client.c와 talker.c는 PC에서 실행할 것이기에 PC에서 native-compile하여 build했다.

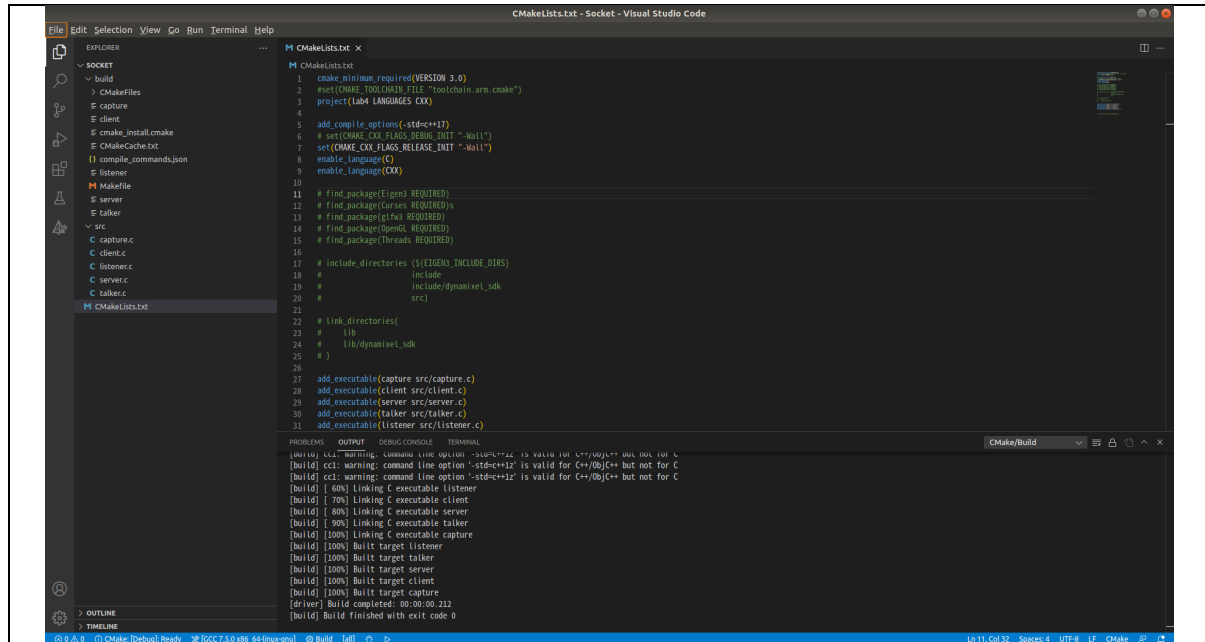


그림 5. client.c, server.c, talker.c, listener.c를 build한 모습

먼저 server-client connection을 테스트 해 보았다. client만을 먼저 실행 시켰을 때 아래와 같은 결과 나타났다.

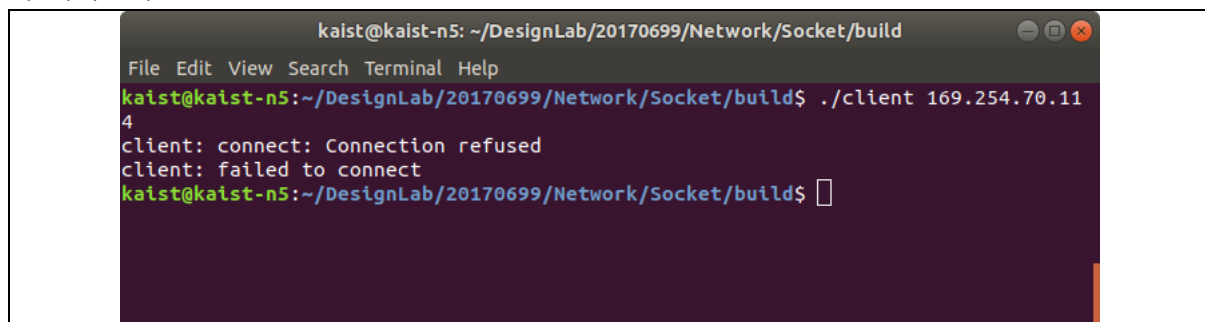


그림 6. client만을 실행한 모습

이는 stream server가 작동하고 있지 않기 때문에 나타난 모습이다. 수신 받을 connection이 존재하지 않기에 정상적으로 작동하지 못하는 것이다. 따라서 server를 Beaglebone에서 키고 client를 PC에서 작동시켰다. 아래와 같이 “Hello, world!” string이 제대로 수신되는 모습을 확인하였다. Server의 경우 이후에도 새로운 client를 기다리는 모습을 볼 수 있다.

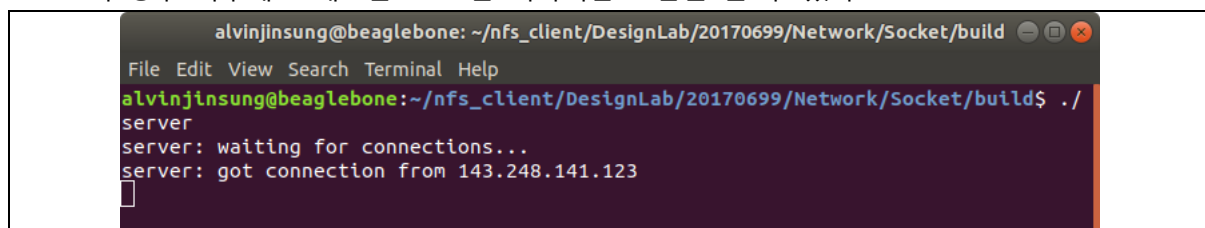
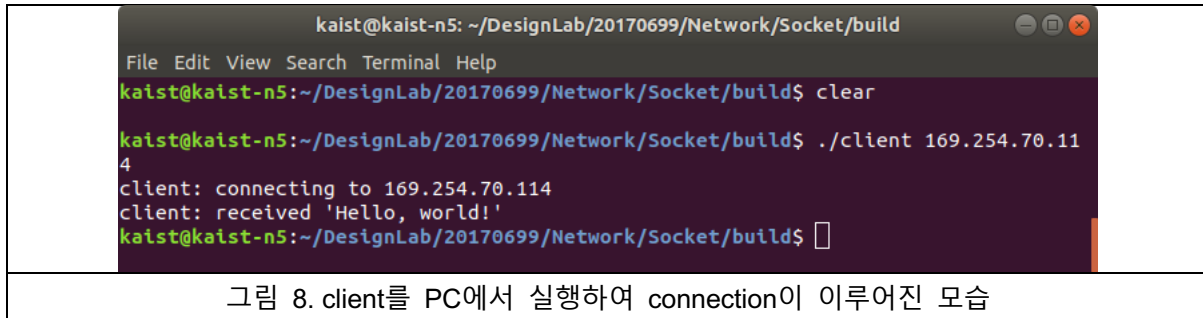
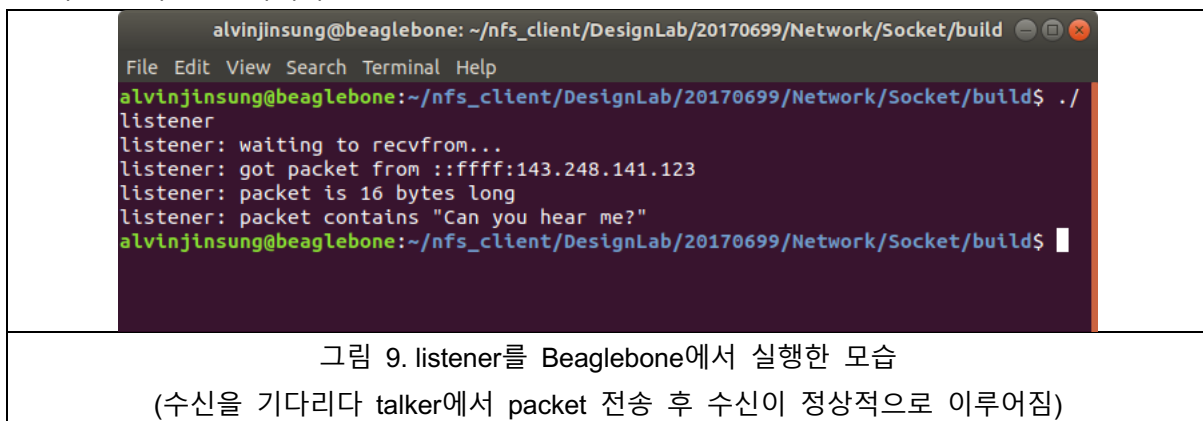


그림 7. server를 Beaglebone에서 실행한 모습



다음으로 listener-talker connection을 테스트 해보았다. Datagram socket의 경우 Stream socket과 다르게 송신 쪽에서는 packet을 지정된 port에 송신만 하며 수신 쪽에서는 지정된 port에서 packet이 도착하면 수신만 하기에 양쪽에서 packet이 제대로 전달되었는지에 대한 것을 신경쓰지 않는다. 그렇기에 이 경우에서 listener만 실행시키더라도 packet이 도착하기를 기다릴 뿐 위의 stream socket의 경우와 같이 connection이 없다고 작동하지 않는 것은 아니다. 이후 talker를 실행시켜 해당 port에 packet을 전달하면 해당 port의 listener가 수신 받게 된다. 아래는 해당 과정을 테스트 해 본 결과이다.



Problem 4B. Remote Commander via UDP

다음 단계로 PC에서 입력된 keyboard input을 Beaglebone에서 UDP를 통해서 종료될때까지 계속해서 수신하는 프로그램을 작성하였다. PC에서는 keyboard input을 프로그램이 종료될때까지 계속해서 받고 입력된 keyboard input을 UDP를 이용하여 Beaglebone에 송신하는 역할을 한다. 위의 문제에서 talker.c를 참고하였다. Beaglebone에서는 PC에서 보낸 keyboard input을 프로그램이 종료될 때까지 수신하여 terminal 창에 프린트하였다. 위의 문제에서 listener.c를 참고하였다.

위의 문제(talker.c, listener.c)와 다른 부분은 프로그램이 종료될 때까지 UDP 통신을 유지해야 하기 때문에 코드가 while문 안에 들어가야 한다는 점이다. 아래는 RemoteController_PC의 코드이다.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdbool.h>
#include <unistd.h>
#include <fcntl.h>
#include <errno.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netdb.h>
#include <sys/time.h>
#include <iostream>
#include <getche.h>
#define SERVERPORT "4950" // the port users will be connecting to
// initialize variables
int main(int argc, char *argv[])
{
    int sockfd;
    struct addrinfo hints, *servinfo, *p;
    int rv;
    int numbytes;

    if (argc != 2) {
        fprintf(stderr, "usage: talker hostname message\n");
        exit(1);
    }

    memset(&hints, 0, sizeof hints);
    hints.ai_family = AF_INET; // set to AF_INET to use IPv4
    hints.ai_socktype = SOCK_DGRAM;

    if ((rv = getaddrinfo(argv[1], SERVERPORT, &hints, &servinfo)) != 0) {
        fprintf(stderr, "getaddrinfo: %s\n", gai_strerror(rv));
        return 1;
    }

    // loop through all the results and make a socket
    for(p = servinfo; p != NULL; p = p->ai_next) {
        if ((sockfd = socket(p->ai_family, p->ai_socktype, p->ai_protocol)) == -1) {
            perror("talker: socket");
            continue;
        }

        break;
    }

    if (p == NULL) {
        fprintf(stderr, "talker: failed to create socket\n");
        return 2;
    }

    for(p = servinfo; p != NULL; p = p->ai_next) {
        if ((sockfd = socket(p->ai_family, p->ai_socktype, p->ai_protocol)) == -1) {
            perror("talker: socket");
            continue;
        }

        break;
    }

    if (p == NULL) {
        fprintf(stderr, "talker: failed to create socket\n");
        return 2;
    }
}
```

```
// initialize variables and error print
// Get argument of destination IP (argv) of Bone
// Init datagram socket. You will use UDP network.
while(1)
// Loop start
{
    // loop through all the results and make a socket

    char a = getch();

    if ((numbytes = sendto(sockfd, &a, sizeof(char), 0, p->ai_addr, p->ai_addrlen)) == -1) {
        perror("talker: sendto");
        exit(1);
    }

    // Add your own script written in Lab 4
    // Please refer to attached code 'talker.c' for writing your own script
    // Print information: key and cmd.

    usleep(1000);
}

freeaddrinfo(servinfo);
close(sockfd);

done:
return 0;
}
```

그림 11. RemoteController_PC 코드

위의 코드에서 주의깊게 봐야할 부분은 while문 안이다. While문 안에서는 Lab1에서 사용한 getch() function을 이용해 character 값을 변수에 저장한다. 그리고 해당 character의 포인터 값과 sizeof()를 이용하여 해당 character를 UDP를 통해 Beaglebone으로 전송한다. While문 안에서 실행되기 때문에 지속적으로 keyboard input을 넣어줄 수 있다. 아래는 해당 프로그램을 실행한 모습이다. 저 상태에서 여러 keyboard input을 입력하여도 프로그램이 종료되지 않고 계속해서 새로운 input을 기다리는 모습을 볼 수 있었다.

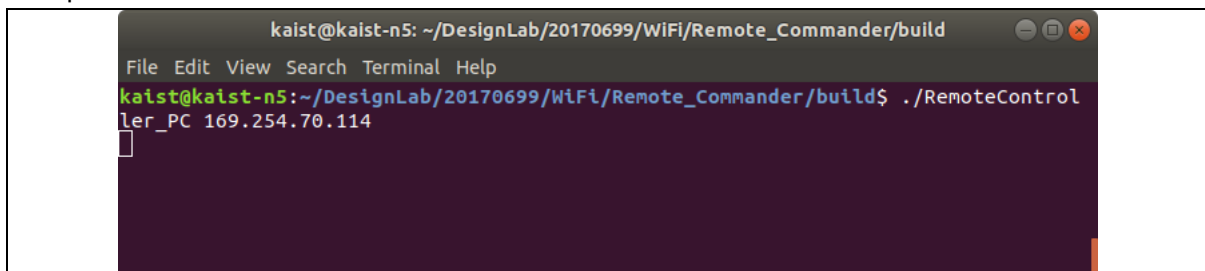


그림 12. RemoteController_PC를 실행한 모습

다음은 RemoteController_Bone의 코드이다.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdbool.h>
#include <unistd.h>
#include <fcntl.h>
#include <errno.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netdb.h>
```



```

#define MYPORT "4950" // the port users will be connecting to

#define MAXBUFLEN 100
// pre-define listener function
// echo macro code
static void echo(char *str, char *file)
{
    int fd = open(file, O_WRONLY);
    if(fd < 0)
    {
        printf("%s:open error.\n", file);
        exit(-1);
    }
    write(fd, str, strlen(str));
    close(fd);
}

// get sockaddr, IPv4 or IPv6:
void *get_in_addr(struct sockaddr *sa)
{
    if (sa->sa_family == AF_INET) {
        return &(((struct sockaddr_in*)sa)->sin_addr);
    }

    return &(((struct sockaddr_in6*)sa)->sin6_addr);
}

int main(int argc, char *argv[])
{
    /*0. Print Title
    1. Set control parameters - gain etc.
    2. Init PWM sysfs.(Optional)
    3. Init GPIO_LED (Optional)
    4. Open datagram socket and bind*/
    /* Print Key guide */

    int sockfd;
    struct addrinfo hints, *servinfo, *p;
    int rv;
    int numbytes;
    struct sockaddr_storage their_addr;
    char buf[MAXBUFLEN];
    socklen_t addr_len;
    char s[INET6_ADDRSTRLEN];

    memset(&hints, 0, sizeof hints);
    hints.ai_family = AF_INET6; // set to AF_INET to use IPv4
    hints.ai_socktype = SOCK_DGRAM;
    hints.ai_flags = AI_PASSIVE; // use my IP

    if ((rv = getaddrinfo(NULL, MYPORT, &hints, &servinfo)) != 0) {
        fprintf(stderr, "getaddrinfo: %s\n", gai_strerror(rv));
        return 1;
    }

    // loop through all the results and bind to the first we can
    for(p = servinfo; p != NULL; p = p->ai_next) {
        if ((sockfd = socket(p->ai_family, p->ai_socktype, p->ai_protocol)) == -1) {
            perror("listener: socket");
            continue;
        }

        if (bind(sockfd, p->ai_addr, p->ai_addrlen) == -1) {
            close(sockfd);
            perror("listener: bind");
            continue;
        }

        break;
    }

    if (p == NULL) {
        fprintf(stderr, "listener: failed to bind socket\n");
        return 2;
    }
}

```



```

freeaddrinfo(servinfo);

printf("listener: waiting to recvfrom...\n");

printf("-----\n");
printf(" R : move 1cm in the world z-axis\n");
printf(" F : move -1cm in the world z-axis\n");
printf(" W : move 1cm in the world z-axis\n");
printf(" S : move -1cm in the world z-axis\n");
printf(" D : move 1cm in the world z-axis\n");
printf(" A : move -1cm in the world z-axis\n");
printf(" E : end the teleoperation\n");
printf(" I : move to the initial position\n");
printf("-----\n");

while(1)
{
    addr_len = sizeof their_addr;
    if ((numbytes = recvfrom(sockfd, buf, MAXBUFLen-1 , 0,
        (struct sockaddr *)&their_addr, &addr_len)) == -1) {
        perror("recvfrom");
        exit(1);
    }

    printf("listener: got packet from %s\n",
        inet_ntop(their_addr.ss_family,
            get_in_addr((struct sockaddr *)&their_addr),
            s, sizeof s));
    printf("listener: packet is %d bytes long\n", numbytes);
    buf[numbytes] = '\0';
    printf("listener: packet contains \"%s\"\n", buf);

    // Please refer to attached code 'listener.c' for writing your own script
    // use strtok() function to parse command to variables
    // use atoi() function to convert a character string to an integer value

    usleep(1000);
}

close(sockfd);
/* Stop PWM */
/* Close socket*/
/* Close GPIO_LED*/
/* Close PWM sysfs files*/
return 0;
}

```

그림 13. RemoteController_Bone 코드

이 코드도 마찬가지로 수신되는 packet을 계속해서 받아야 하기에 while문 안에 수신하는 부분을 넣어 지속적으로 packet을 받을 수 있도록 하였다. 아래는 프로그램을 실행한 모습이다. 여러 개의 keyboard input이 계속해서 수신되고 있는 모습을 확인할 수 있다.

```

alvinjinsung@beaglebone: ~/nfs_client/DesignLab/20170699/WiFi/Remote_Commander/build
File Edit View Search Terminal Help
alvinjinsung@beaglebone:~/nfs_client/DesignLab/20170699/WiFi/Remote_Commander/build$ ./RemoteController_Bone
listener: waiting to recvfrom...
-----
R : move 1cm in the world z-axis
F : move -1cm in the world z-axis
W : move 1cm in the world z-axis
S : move -1cm in the world z-axis
D : move 1cm in the world z-axis
A : move -1cm in the world z-axis
E : end the teleoperation
I : move to the initial position
-----
listener: got packet from ::ffff:143.248.141.123
listener: packet is 1 bytes long
listener: packet contains "h"
listener: got packet from ::ffff:143.248.141.123
listener: packet is 1 bytes long
listener: packet contains "i"
listener: got packet from ::ffff:143.248.141.123
listener: packet is 1 bytes long
listener: packet contains "b"
listener: got packet from ::ffff:143.248.141.123
listener: packet is 1 bytes long
listener: packet contains "o"
listener: got packet from ::ffff:143.248.141.123
listener: packet is 1 bytes long
listener: packet contains "n"
listener: got packet from ::ffff:143.248.141.123
listener: packet is 1 bytes long
listener: packet contains "e"

```

그림 14. RemoteController_Bone을 실행한 모습

Problem 4C. Stream Webcam Screen via V4L2 and FFmpeg

Sebcam streaming system을 구성하기 위해 Video4Linux2와 FFmpeg를 사용한다. V4L2(Video4Linux2)는 Linux의 video capture application programming interface로서 다양한 USB Webcam, TV tuners등의 Device를 support한다. capture.c 코드를 통해 V4L2를 사용하여 Webcam을 capture할 것이다. FFmpeg는 multimedia framework로, decode, encode, transcode, mux, demux, stream, filter, play등 광범위한 부분을 support한다. 이를 이용하여 system을 구성하는데, Beaglebone에서는 FFmpeg가 Webcam screen을 capture.c를 통해 input으로 받는다. 이는 UDP network를 통해 stream되고 PC에서 이를 수신한다. PC에서는 수신된 데이터(Webcam screen)를 FFplay를 통해 보여주게 된다. 먼저 V4L2의 다양한 명령어들을 테스트 해 보았다.

To list the devices:

```
# v4l2-ctl --list-devices
```

To list the formats available:

```
# v4l2-ctl --list-formats
```

```

alvinjinsung@beaglebone: ~/nfs_client/DesignLab/20170699/WiFi/Remote_Commander/build
File Edit View Search Terminal Help
alvinjinsung@beaglebone:~/nfs_client/DesignLab/20170699/WiFi/Remote_Commander/bu
ild$ v4l2-ctl --list-devices
UVC Camera (046d:0825) (usb-musb-hdrc.1-1):
    /dev/video0
    /dev/video1

alvinjinsung@beaglebone:~/nfs_client/DesignLab/20170699/WiFi/Remote_Commander/bu
ild$ v4l2-ctl --list-formats
ioctl: VIDIOC_ENUM_FMT
    Type: Video Capture

    [0]: 'YUYV' (YUYV 4:2:2)
    [1]: 'MJPG' (Motion-JPEG, compressed)

```

그림 15. V4L2 devices와 formats를 list한 모습

Get all of the information available for the camera:

v4l2-ctl --all

```

alvinjinsung@beaglebone: ~/nfs_client/DesignLab/20170699/WiFi/Remote_Commander/build
File Edit View Search Terminal Help
alvinjinsung@beaglebone:~/nfs_client/DesignLab/20170699/WiFi/Remote_Commander/bu
ild$ v4l2-ctl --all
Driver Info:
    Driver name      : uvcvideo
    Card type        : UVC Camera (046d:0825)
    Bus info         : usb-musb-hdrc.1-1
    Driver version    : 4.19.94
    Capabilities     : 0x84a00001
        Video Capture
        Metadata Capture
        Streaming
        Extended Pix Format
        Device Capabilities
    Device Caps      : 0x04200001
        Video Capture
        Streaming
        Extended Pix Format
Media Driver Info:
    Driver name      : uvcvideo
    Model            : UVC Camera (046d:0825)
    Serial           : 34898B50
    Bus info         : usb-musb-hdrc.1-1
    Media version     : 4.19.94
    Hardware revision: 0x00000012 (18)
    Driver version    : 4.19.94
Interface Info:
    ID               : 0x03000002
    Type              : V4L Video
Entity Info:
    ID               : 0x00000001 (1)
    Name              : UVC Camera (046d:0825)
    Function          : V4L2 I/O
    Flags             : default
    Pad 0x01000007    : 0: Sink
        Link 0x02000019: from remote pad 0x100000a of entity 'Extension 4': Da
ta, Enabled, Immutable
Priority: 2
Video input : 0 (Camera 1: ok)
Format Video Capture:
    Width/Height      : 640/480
    Pixel Format       : 'YUYV' (YUYV 4:2:2)
    Field              : None
    Bytes per Line     : 1280
    Size Image         : 614400
    Colospace          : sRGB
    Transfer Function  : Default (maps to sRGB)
    YCbCr/HSV Encoding: Default (maps to ITU-R 601)
    Quantization       : Default (maps to Limited Range)
    Flags              :
Crop Capability Video Capture:
    Bounds            : Left 0, Top 0, Width 640, Height 480
    Default            : Left 0, Top 0, Width 640, Height 480

```

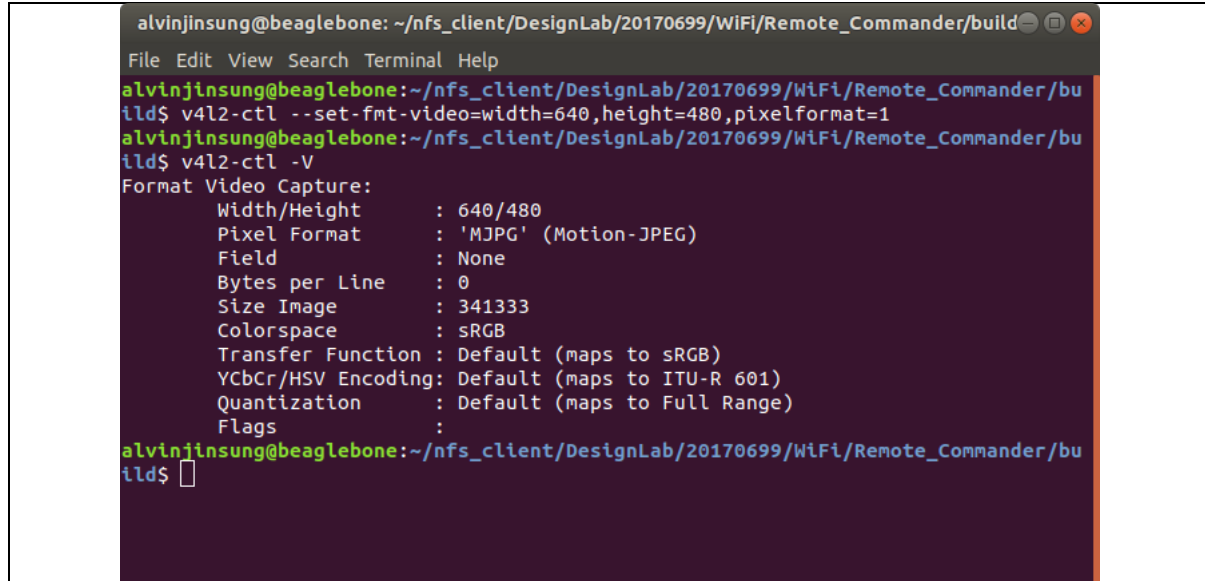
그림 16. V4L2 camera 정보를 표시한 모습

If we wish to modify the resolution:

```
# v4l2-ctl --set-fmt-video=width=640,height=480,pixelformat=1
```

Check current format

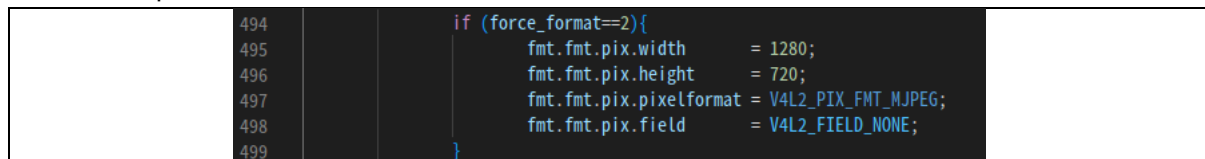
```
# v4l2-ctl -V
```



```
alvinjinsung@beaglebone: ~/nfs_client/DesignLab/20170699/WiFi/Remote_Commander/build
File Edit View Search Terminal Help
alvinjinsung@beaglebone:~/nfs_client/DesignLab/20170699/WiFi/Remote_Commander/bu
ild$ v4l2-ctl --set-fmt-video=width=640,height=480,pixelformat=1
alvinjinsung@beaglebone:~/nfs_client/DesignLab/20170699/WiFi/Remote_Commander/bu
ild$ v4l2-ctl -V
Format Video Capture:
  Width/Height       : 640/480
  Pixel Format       : 'MJPG' (Motion-JPEG)
  Field              : None
  Bytes per Line     : 0
  Size Image        : 341333
  Colospace          : sRGB
  Transfer Function  : Default (maps to sRGB)
  YCbCr/HSV Encoding: Default (maps to ITU-R 601)
  Quantization       : Default (maps to Full Range)
  Flags              :
alvinjinsung@beaglebone:~/nfs_client/DesignLab/20170699/WiFi/Remote_Commander/bu
ild$
```

그림 17. V4L2 resolution을 변경한 모습

다음으로 capture.c를 format에 맞게 코드를 변형하였다. 변형한 코드 부분은 아래와 같다.



```
494 if (force_format==2){
495     fmt.fmt.pix.width      = 1280;
496     fmt.fmt.pix.height    = 720;
497     fmt.fmt.pix.pixelformat = V4L2_PIX_FMT_MJPEG;
498     fmt.fmt.pix.field      = V4L2_FIELD_NONE;
499 }
```

그림 18. capture.c 코드를 변형한 모습



```
191 /* Timeout. */
192 tv.tv_sec = 20;
193 tv.tv_usec = 0;
```

그림 19. capture.c 코드를 변형한 모습

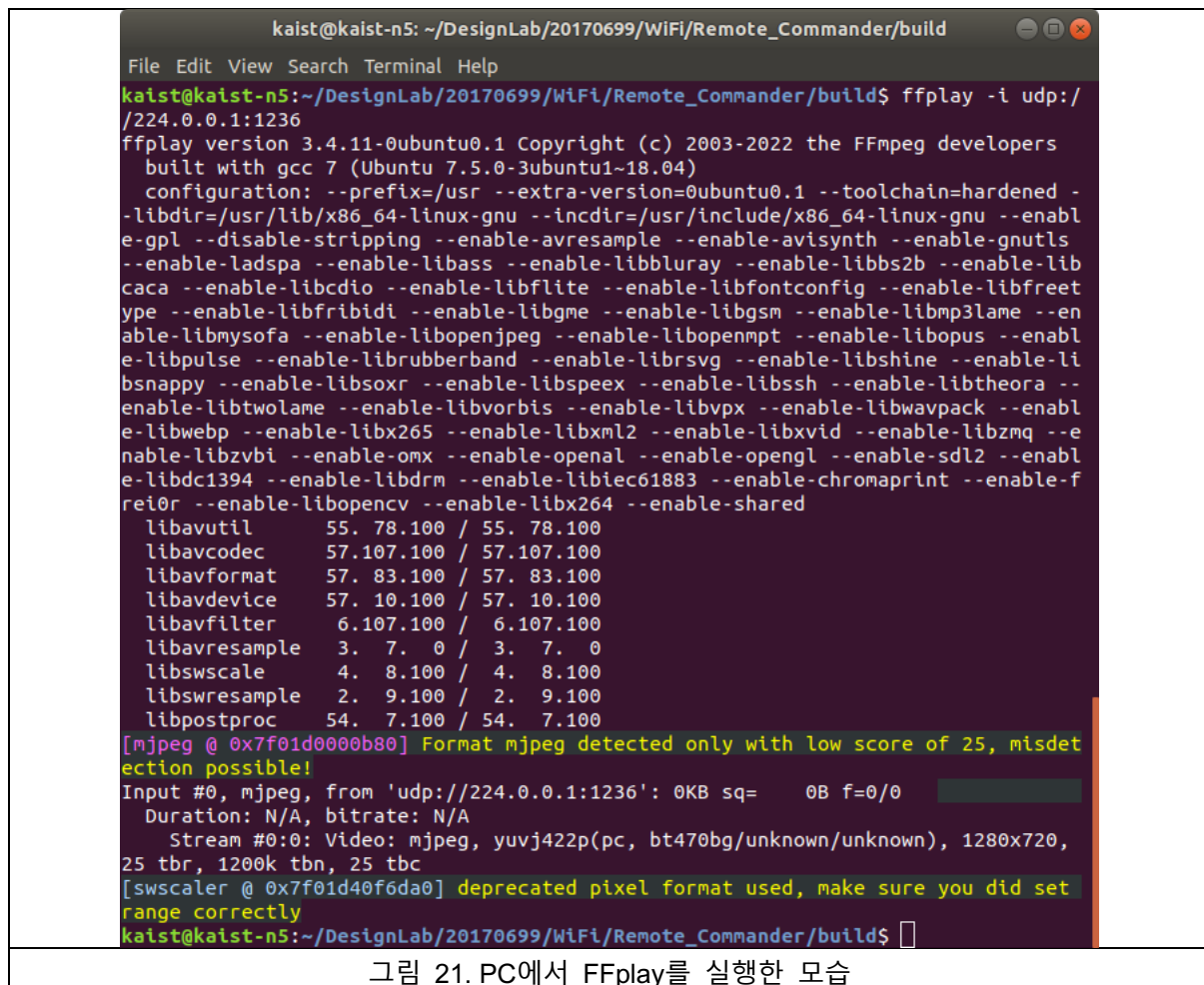
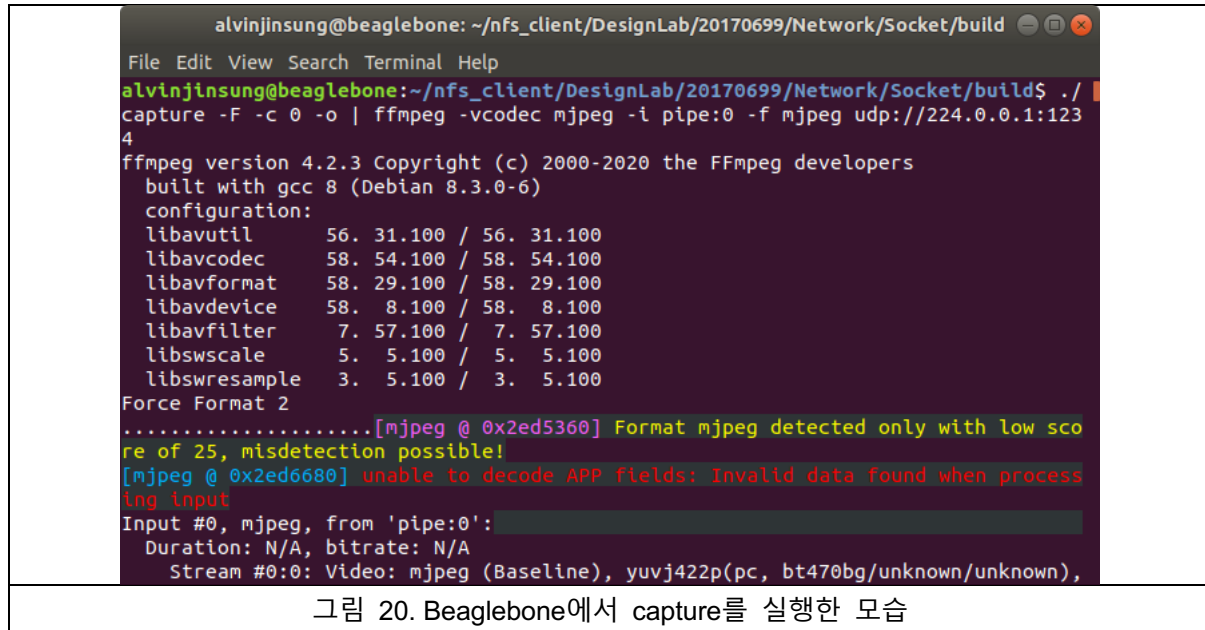
capture.c를 build하였고 최종적으로 Webcam screen을 FFmpeg를 이용하여 streaming 해 보았다. Beaglebone에는

```
$ ./capture -F -c 0 -o | ffmpeg -vcodec mjpeg -i pipe:0 -f mjpeg udp://224.0.0.1:1234
```

명령어를, PC에는

```
$ ffplay -i udp://224.0.0.1:1234
```

명령어를 입력하였고 성공적으로 Webcam screen이 streaming 되는 모습을 확인할 수 있었다. 아래는 해당 결과이다.



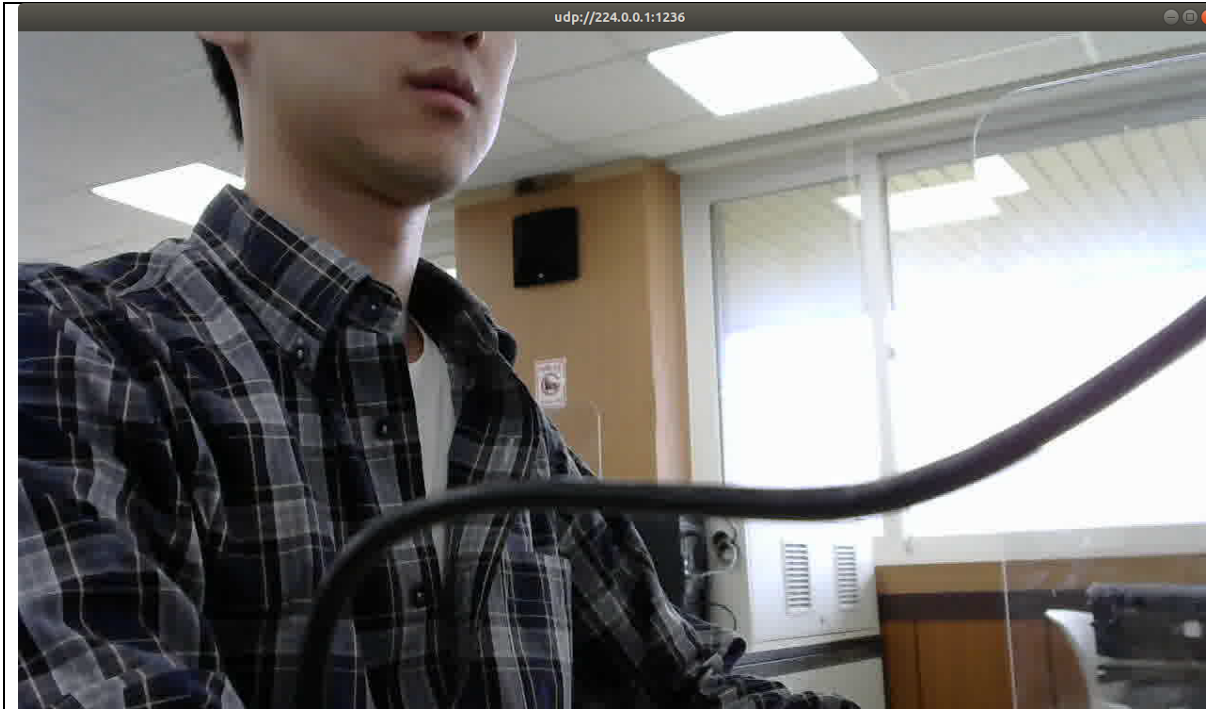


그림 22. Webcam screen이 PC에서 streaming 되는 모습

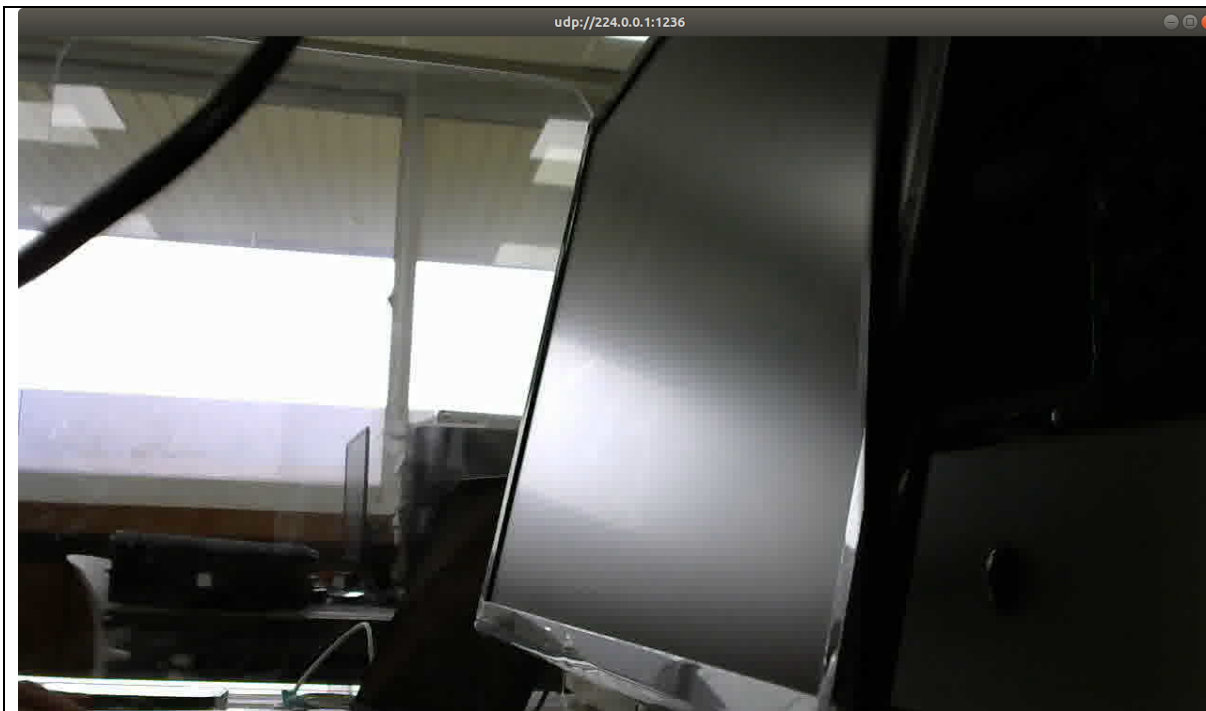


그림 23. Webcam screen이 PC에서 streaming 되는 모습

4. Discussion

(1) Compare TCP and UDP. Explain with examples which method is suitable for which situation.

TCP와 UDP는 computer network에서 사용되는 서로 다른 transport layer protocol로서 TCP는 connection-oriented protocol이고 UDP는 connectionless protocol이다. 둘의 차이점은 크게 4가지로 볼 수 있는데, reliability, connection 유무, 속도, overhead이다.

- Reliability

TCP는 reliable한 data transmission을 보장한다. 이는 data가 전달될 때 올바른 순서로 loss나 duplicate 없이 도달하게 된다는 것을 의미한다. 반대로 UDP의 경우에는 unreliable한 data transmission으로 모든 data가 올바른 순서로 loss없이 도달하는 것을 보장하지 못한다.

- Connection 유무

TCP의 경우에는 data transmission 이전에 connection을 establish하고 유지하는 반면 UDP의 경우에는 data transmission 이전에 connection을 establish하지 않는다.

- 속도

UDP가 TCP에 비해 속도가 빠르다.

- Overhead

TCP는 flow control, error detection, retransmission of lost data 등 고려해야할 요소들이 많아 UDP에 비해 overhead가 크다.

위에서 명시한 특징들로 인해 TCP는 reliable한 data transmission을 요구하는 file transfer, email, web browsing 등에 적합한다. UDP의 경우에는 빠른 속도를 요구하는 real-time video and audio streaming, online gaming, VoIP등에 적합한다.

(2) We will stream the WebCam video via UDP network. Why UDP is prefer method for streaming service?

(1)에서 명시한 특징들로 인해 streaming service에는 UDP를 사용하는 것이 더욱 선호된다. Real-time streaming이 중요한 요소이기에 latency가 낮고 data transmission이 빠른 UDP가 적합하며, video streaming의 경우 일부 data가 loss되더라도 viewing experience에 큰 영향을 주지 않기 때문에 UDP의 단점이 수용될 수 있다.

(3) Choose your own discussion topic related to Lab 4 and provide an answer to it.

Discussion topic:

How do TCP and UDP handle packet loss, and which protocol is better suited for lossy networks?

Answer:

TCP는 reliable protocol로 packet loss가 발생하게 되면 retransmit을 진행하게 되며 UDP의 경우에는 unreliable protocol로 packet loss에 대한 built-in mechanism이 존재하지 않는다. TCP의 경우에는 packet loss가 발생하면 acknowledgement, retransmission, congestion control의 mechanism을 통해 이를 handle 하는데, packet loss가 발생하면 receiver가 negative acknowledgement(NACK)를 보내 packet이

retransmit 되도록 요청한다. 이와 추가로 congestion control을 통해 network congestion이 발생하지 않도록 manage한다. UDP는 위와 같은 과정이 없기 때문에 loss-tolerant한 lossy network의 경우에는 packet loss가 발생해도 괜찮은, packet loss로 인한 overhead가 많이 생기지 않는 UDP가 선호될 수 있다. 그러나 reliability가 중요하다면 congestion이 생기더라도 여전히 TCP를 사용해야 한다.

5. References

- [1] Lab4 Experiment Guide.pdf
- [2] EE405 Electronic Design Lab4_revised_final.pdf
- [3] Lab4_TA_Session.pptx
- [4] Beej's Guide to Network Programming, <https://beej.us/guide/bgnet/html/split/>