Design & Implemenation: Internet Video, Voice & Text Chatting Program

Tan Wei Xuan (49003140) tanweixuan@postech.ac.kr
Zhang Xin Yue (49003143) xyzhang@postech.ac.kr

April 20, 2019

1 Project Overview

The **Internet Video, Voice and Text Messaging Program** that I have implemented is a **Multi-threaded Chat Application** that utilises the Transmission Control Protocol *(TCP)* and allows for mutiple clients to send/receive text, voice messages as well as a video stream over a server, to one another, at the same time. The source code files are as follow:

1. **Server Files**
   - server.py
2. **Client Files**
   - client.py

2 Environment and Dependencies

My program is written in **Python 3.6.7** and is developed on **Windows running the Ubuntu 18.04.2 bit subsystem**, using **PyCharm** as the **Integrated Development Environment**. *The program may not work as inteneded if it is ran on other environments.* ***Ensure that these dependencies are being installed on your system.***

1. **PyAudio**

```
#On Ubuntu Terminal
$ sudo apt-get install python-pyaudio python3-pyaudio
```

2. **Threading**

```
#On Ubuntu Terminal
$ sudo apt install python3-pip #Only if pip is not installed
$ pip install threaded
```

3. **libasound**

```
#On Ubuntu Terminal
$ sudo apt-get install libasound2-devf
```

4. **Pillow**

```
#On Ubuntu Terminal
$ pip install Pillow==2.2.1
```

5. **OpenCV**

```
#On Ubuntu Terminal
$ pip install opencv-python
```

3 Implementation - Video Chat

3.1 Server - Handling Video

Below is the handler for how a server distribute the video data that is being received from one client to another. The Video port on the server is *1026*.

```
#Server.py
def handle_client_video(client):
  """
  Handles Video from the given socket.
  """
  # Receive Video
  clients_video[client] = 1
  while client in clients_video:
          ttlrec = 0
          metarec = 0
          msgArray = []
          metaArray = []

          while metarec < 8:
              chunk = client.recv(8 - metarec)
              metaArray.append(chunk)
              metarec += len(chunk)

          length = int(chunk.decode("utf8"))

          while ttlrec < length:
              chunk = client.recv(length-ttlrec)
              if chunk == '' :
                  raise RuntimeError("Socket␣Connection␣has␣broken")
```

```
                msgArray.append(chunk)
                ttlrec += len(chunk)

            broadcast(b''.join(metaArray + msgArray),dtype="video", sd=client)
```

3.2 Client - GUI

We utilised the *Tkinter module*, an inbuilt GUI implemenation of Python's library to serve as the Graphic User Interface for our program. The video stream as well as the chat window for exchanging text messages will all be displayed on the GUI.

```
#Client.py
#TKinter Settings
top = tkinter.Tk()
top.title("Internet␣Voice,Video␣and␣Text␣Chatting␣Programme")

video_frame = tkinter.Frame(top)
video_frame.pack(side=tkinter.TOP)

messages_frame = tkinter.Frame(top)
messages_frame.pack(side=tkinter.TOP)

button_frame = tkinter.Frame(top)
button_frame.pack(side=tkinter.TOP)

my_msg = tkinter.StringVar()
my_msg.set("Enter␣your␣Name")

scrllBar = tkinter.Scrollbar(messages_frame)

entry_field = tkinter.Entry(button_frame, textvariable=my_msg)
entry_field.bind("<Return>", send_text)
entry_field.pack(side=tkinter.LEFT, padx = 10)

send_button = tkinter.Button(button_frame, text="Send", command = send_text)
send_button.pack(side=tkinter.LEFT, padx = 10)

msg_list = tkinter.Listbox(messages_frame, height = 15, width =50, yscrollcommand
    = scrllBar.set)
scrllBar.pack(side=tkinter.RIGHT, fill=tkinter.Y)
msg_list.pack(side=tkinter.TOP, fill=tkinter.BOTH)
```

3.3 Client - Video Capture

We utilised the OpenCV library for the capturing of the video stream through the use of the client's webcam. The frames being captured through the webcam are treated as **Images**

```
#Client.py
#Capture for Webcam
cap = cv2.VideoCapture(0)

#Work around in case webcam is not opened
if cap.read() == False:
  cap.open()
```

3.4 Client- Sending Video

The frame (image) being captured from the webcam is subsequently (in real-time) being sent to the server through the handler below.

```
#Server.py
#Send Video to the server
def send_video():
  global cap

  while not stop_video.is_set():
    ret_val, img = cap.read()
    img = cv2.resize(img, (360,360))

    cv2_im = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    pil_im = Image.fromarray(cv2_im)
    b_io = io.BytesIO()
    pil_im.save(b_io,'jpeg')
    framestr = b_io.getvalue()

    ttlsent = 0
    metasent = 0
    length = len(framestr)
    lengthstr = str(length).zfill(8)

    while metasent < 8:
      sent = client_socket_video.send(lengthstr[metasent::].encode("utf8"))
      if sent == 0:
        raise RuntimeError("Socket Connection has broken")
      metasent += sent

    while ttlsent < length:
      sent = client_socket_video.send(framestr[ttlsent::])
      if sent == 0:
        raise RuntimeError("Socket Connection has broken")
      ttlsent += sent
```

3.5 Client - Displaying Video Stream

The GUI of our program displays two video streams on it. One being the client's own video capture and the other being the video stream being received from the other clients on the server. This handler reads the frame (image) from the client's own webcam and displays it on the GUI.

```
#Client.py
#Cam Capture to Screen
def show_my_video():
  global cap
  panel = None

  while not stop_video.is_set():
    ret_val, img = cap.read()

    img = cv2.resize(img, (120, 120))

    image = cv2.cvtColor(img,cv2.COLOR_BGR2RGB)
    image = Image.fromarray(image)
    image = ImageTk.PhotoImage(image)

    if panel is None:
      panel = tkinter.Label(image=image)
      panel.image = image
      panel.place(height=110,width=110,x=256,y=5)
    else:
      panel.configure(image=image)
      panel.image = image
clientSocket.close();
```

3.6 Client - Receving Video

The video data that is being recevied by the client throgh the server is handled through the function below. The video data received will be displayed on the GUI.

```
#Client.py
# For Receiving Video
def receive_video():
  global panel
  while not stop_video.is_set():
    try:
      ttlrec = 0
      metarec = 0

      msgArray = []
```

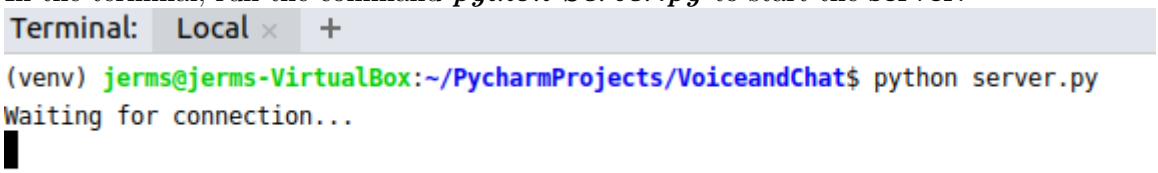
```
    metaArray = []

    while metarec < 8:
      chunk = client_socket_video.recv(8 - metarec).decode("utf8")
      if chunk == '':
        raise RuntimeError("Socket␣Connection␣has␣been␣broken")
      metaArray.append(chunk)
      metarec += len(chunk)

    lengthstr = ''.join(metaArray)
    length = int(lengthstr)

    while ttlrec < length:
      chunk = client_socket_video.recv(length - ttlrec)
      if chunk == '':
        raise RuntimeError("Socket␣Connection␣has␣been␣broken")
      msgArray.append(chunk)
      ttlrec += len(chunk)

    frame = b''.join(msgArray)
    pil_bytes = io.BytesIO(frame)
    pil_img = Image.open(pil_bytes)
    img = ImageTk.PhotoImage(pil_img)

    if panel is None:
      panel = tkinter.Label(video_frame, image =img)
      panel.image = img
      panel.pack(side=tkinter.TOP, expand = True)
    else:
      panel.configure(image = img)
      panel.image = img

  except OSError:
    break
```

4 Running the Program

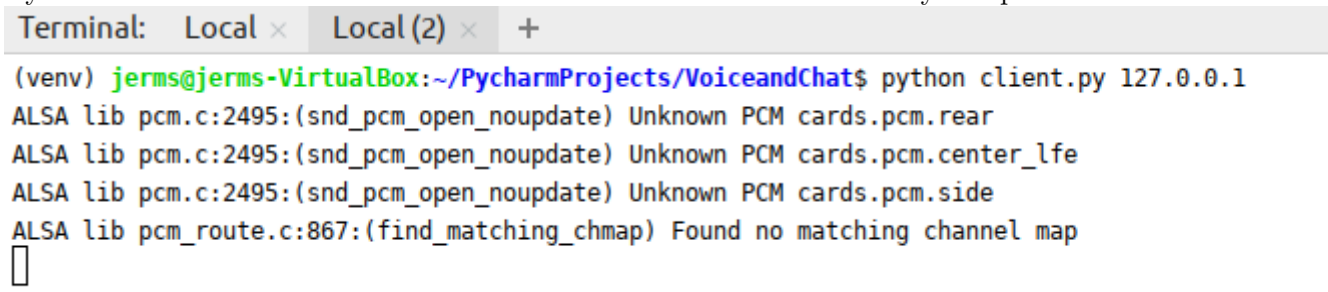
*On the client, the input for host should be **127.0.0.1** unless a specific host has been defined.*

4.1 Terminal

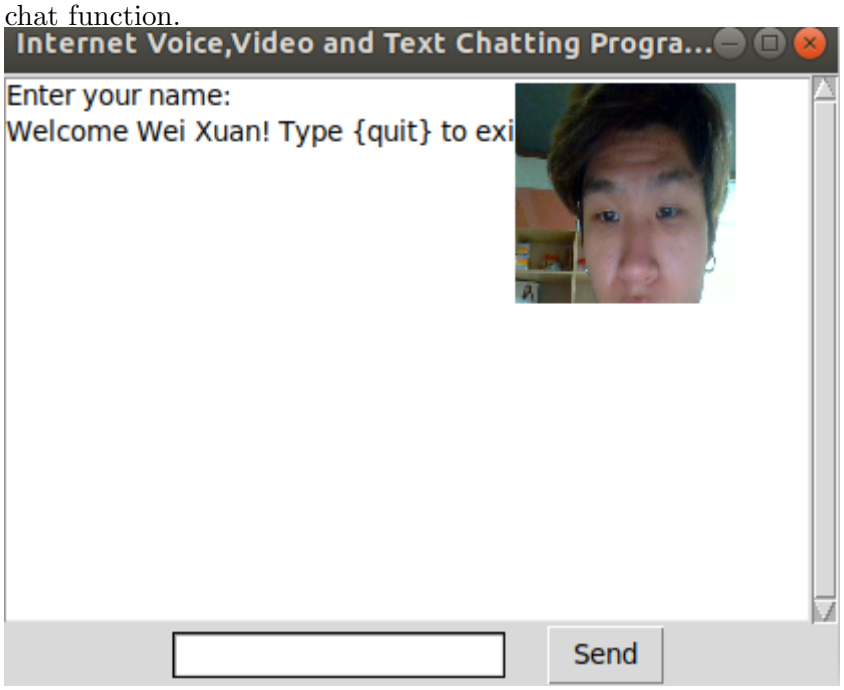
1. In the terminal, run the command ***python Server.py*** to start the **server**.



2. In the terminal, run the command ***python Client.py #hostname*** to start the **client**. By default the host name will be **127.0.0.1**. The GUI will automatically be opened.



3. Enter your **name** in the textbox and press the *"Send"* button to begin using the text chat function.



4. When more than 1 user has connected to the server, both **Video** and **Voice** will be automatically enabled. You do not have to input any additional commands to enable them. **Video, Voice and Text** can be sent/received simultaneously.

5. In the **GUI Text Box**, type **{quit}** to exit from the server.

5 Limitations

Testing is done one a single computer and as such, we **unable to open multiple instances of the computer's webcam**. We are thus limited to only being able to demonstrate the sending of video from one client to the receiving of this video on another client in our *Screen Record of Program Execution*. Also, the voice echo signifies that the voice has been received on the other client. If two separate computers are being connected to the server thorugh a network, our program will be able to send/receive videos between the clients running on the two separate computers.

Our Program currently only allow for peer-to-peer video communications (between two parties). Further improvement can be made to allow more than 2 clients to connect to the server and chat through video communications at any point of time. The Video Stream sent are compressed through the conversion into JPEG images and thus the resolution for then recevied Video Stream will not be of the best quality. We could implement better compression algorithms for the Video Stream to improvde the Video quality in the future.