UNIVERSIDADE FEDERAL DE PERNAMBUCO

Grupo de Pesquisa em Engenharia Biomédica

Computação Visual

Vídeo Chamada

Carlos Gabriel

Jairo Magno

Recife, 2023

Sumário

1 Objetivos

Este relatório é referente ao trabalho de computação visual da Sprint 03 : 15/07/2023 - 28/07/2023 do grupo de IC's do GPEB. O principal objetivo desse trabalho é criação de um video chat usando Python *OpenCV* e programação de soquete. Nele será documentado a construção e funcionamento dos projetos propostos.

2 Descrição

Esta seção é responsável por descrever a implementação de uma aplicação de videochamada de tempo real em rede local, utilizando dois scripts em Python, um para o servidor e outro para o cliente. A aplicação permite que dois computadores na mesma rede local estabeleçam uma videochamada, na qual o cliente envia os frames de vídeo capturados por sua webcam para o servidor, que exibe os frames recebidos em sua tela. Além disse a chamada é gravada e armazenada no computador do servidor.

2.1 Lado Servidor

Na próxima seção será detalhado o funcionamento do código para sua utilização como receptor do vídeo do cliente.

2.1.1 Servidor Recebendo Vídeo

Começando pelo servidor com papel de receber o vídeo, pode-se verificar o código criado na Figura 2.1.

```python
import socket, pickle, struct, imutils
import cv2 as cv

from wireless_adapter import get_wifi_ip

# Criar socket
server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
host_ip = get_wifi_ip()
print('HOST IP:', host_ip)
port = 8080
socket_address = (host_ip, port)
#vinculando o socket
server_socket.bind(socket_address)
#escutando o socket para alguma conexão
server_socket.listen(5)
print("ESCUTANDO EM:", socket_address)
# Aceitar a conexão do cliente
client_socket, addr = server_socket.accept()
print('CONEXÃO RECEBIDA DE:', addr)

data = b""
payload_size = struct.calcsize("Q")
fourcc = cv.VideoWriter_fourcc(*'XVID')
out = cv.VideoWriter('video.avi', fourcc, 20.0, (320, 240))
host_end = True

try:
    while True:
        # Recebe os frames do video do cliente
        while len(data) < payload_size:
            packet = client_socket.recv(4 * 1024)  # 4K
            if not packet:
                raise ValueError("Cliente encerrou a Conexão.")
            data += packet

        packed_msg_size = data[:payload_size]
        data = data[payload_size:]
        msg_size = struct.unpack("Q", packed_msg_size)[0]

        while len(data) < msg_size:
            packet = client_socket.recv(4 * 1024)
            if not packet:
                raise ValueError("Cliente encerrou a Conexão.")
            data += packet

        frame_data = data[:msg_size]
        data = data[msg_size:]
        frame = pickle.loads(frame_data)

        # Mostra o frame do video recebido localmente
        cv.imshow("RECEBENDO VIDEO (CLIENTE)", frame)
        out.write(frame)
        if cv.waitKey(1) == ord('q'):  # Aperte 'q' para sair
            break

except (ConnectionResetError, ValueError):
    host_end = False
    print('Video Chamada encerrada pelo CLIENTE')

cv.destroyAllWindows()
client_socket.close()
server_socket.close()
out.release()
if host_end: print('Vídeochamada encerrada pelo HOST.') #encerrando a chamada pelo lado do host

```

Figura 2.1: Código do lado do servidor

O trecho das linhas 1 até 4 é responsável pelas importação das bibliotecas que serão utilizadas e do módulo 'get_wifi_ip'. Esse módulo tem como objetivo adquirir diretamente o ip da rede em que o servidor esteja conectado. Seu código será explicado em uma outra seção. As bibliotecas incluem 'socket' para a comunicação via rede, 'pickle' para serialização de objetos Python, 'struct' para empacotar e desempacotar dados binários, 'imutils' para redimensionar os frames de vídeo e 'cv2' (OpenCV) para o processamento de vídeo.

Na linha 7 está sendo criado um novo objeto de socket utilizando a família de endereços IPv4 (socket.AF_INET) e o tipo de socket TCP (socket.SOCK_STREAM). Essa combinação cria um socket TCP/IP.

Das linhas 8 até 13 são responsáveis por obter o endereço IP do servidor usando o método 'get_wifi_ip()', definir o número da porta em que o servidor estará escutando e criar uma tupla 'socket_address' para armazenar esses dados. Após isso será vinculado o 'server_socket' com a tupla criada. O servidor estará escutando conexões nesse endereço.

O trecho da linha 15 até 19 são responsáveis por iniciar o modo de escuta do servidor para aguardar conexões de clientes através do 'server_socket.listen(5)'. O valor 5 no método 'listen()' indica o número máximo de conexões pendentes na fila de espera. Quando o servidor estiver ocupado, as conexões adicionais serão enfileiradas. Quando um cliente se conecta ao servidor, o método 'accept()' é chamado. Ele aguarda até que um cliente se conecte e retorna um novo socket ('client_socket') para comunicação com o cliente e o endereço (addr) do cliente.

As linhas 21 até 25 inicializam algumas variáveis importantes, como: 'data' que é usada para armazenar os dados recebidos do cliente, 'payload_size' que representa o tamanho do cabeçalho dos dados que contém o tamanho do frame de vídeo, 'fourcc' que é usado para definir o codec de compressão para gravar o vídeo recebido pelo servidor e 'out' que é o objeto 'VideoWriter' que será usado para gravar o vídeo em um arquivo local.

Dentro do bloco 'try' a partir da linha 27 é onde começa o loop principal do servidor para receber os frames de vídeo do cliente. O loop continuará de forma indefinida até que a videochamada seja encerrada ou algum erro venha a acontecer. O servidor usa o método 'recv()' para receber os dados do cliente e armazena-los dentro da variável 'data'. O servidor irá continuar recebendo os dados até que o cabeçalho ('payload_size') esteja completo, o que indica o tamanho do frame a ser recebido. Uma vez que o cabeçalho foi recebido e processado, o servidor recebe o restante dos dados do frame de vídeo e o reconstitui usando 'pickle.loads()'. Após reconstruir o frame, o servidor o exibe localmente usando 'cv.imshow()'. O servidor também grava o frame recebido em um arquivo de

vídeo utilizando o objeto 'VideoWriter'. Se o servidor detectar que o usuário pressionou a tecla 'q', ele encerra a transmissão do vídeo.

O bloco 'except' é responsável por lidar com exceções que podem ser lançadas caso o cliente encerre a conexão abruptamente. Se isso acontecer, o servidor exibirá uma mensagem informando que a videochamada foi encerrada pelo cliente.

Após a saída do loop principal, o servidor libera os recursos, fecha os sockets e libera o objeto 'VideoWriter'. Se a chamada for encerrada pelo servidor (usuário pressiona 'q'), ele exibirá uma mensagem informando que a videochamada foi encerrada pelo host.

2.2 Lado Cliente

O cliente será quem fará a comunicação com servidor. Será explicado nessa seção o código no qual o cliente será o responsável por enviar o vídeo.

2.2.1 Cliente Enviando Vídeo

Começando pelo cliente com papel de enviar o vídeo (comunicação com script do servidor responsável por receber o vídeo), pode-se verificar o código criado na Figura 2.1.

```python
import socket, pickle, struct
import cv2 as cv

from wireless_adapter import get_wifi_ip

client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
host_ip = get_wifi_ip()  # IP do servidor
port = 8080
client_socket.connect((host_ip, port))  # Conectar com o servidor

video = cv.VideoCapture(0)
while True:
    ret, frame = video.read()
    frame = cv.resize(frame, (320, 240))
    frame = cv.cvtColor(cv.flip(frame, 1), cv.COLOR_BGR2RGB) #invertendo a imagem para perspectiva de selfie
    frame = cv.cvtColor(frame, cv.COLOR_RGB2BGR)
    data = pickle.dumps(frame)
    message = struct.pack('Q', len(data)) + data
    client_socket.sendall(message)

    cv.imshow('TRANSMITINDO VIDEO', frame)
    if cv.waitKey(1) == ord('q'):  # Press 'q' para parar a transmissão
        break

# libera os recurso e fecha o socket
video.release()
cv.destroyAllWindows()
client_socket.close()
```

Figura 2.2: Código do lado do cliente

As linhas 6 até 11 são responsáveis por criar um novo objeto de socket para o cliente, utilizando a família de endereços IPv4 (socket.AF_INET) e o tipo de socket TCP (socket.SOCK_STREAM). Essa combinação cria um socket TCP/IP para o cliente.

Em seguida, é obtido o endereço IP do servidor utilizando a função 'get_wifi_ip()' da biblioteca 'wireless_adapter'. Isso é necessário para que o cliente saiba para onde se conectar. É possível que ocorra erro ao utilizar esse módulo, para isso basta digitar manualmente o ip em que o servidor está conectado dentro da variável 'host_ip'.

Finalmente, o cliente se conecta ao servidor através do método 'connect()', passando o endereço IP e a porta do servidor. Essa chamada estabelece a conexão com o servidor e permite que o cliente envie dados.

Após isso será inicializado a captura de vídeo da webcam do cliente usando o método 'cv.VideoCapture(0)'. O valor 0 indica que será usado a câmera padrão do sistema.

Dentro do loop, no trecho das linhas 13 a 16 o cliente captura um frame de vídeo da sua webcam usando 'video.read()'. Em seguida, redimensiona o frame para um tamanho de 320x240

pixels usando 'cv.resize()'. Além disso, inverte horizontalmente o frame usando 'cv.flip()' para que o cliente veja o vídeo como uma perspectiva de selfie (espelhado horizontalmente). Isso é útil para uma visualização mais natural da própria imagem durante a videochamada.

As linhas 17 até 19 são responsáveis por pegar o frame e serializar usando 'pickle.dumps()', transformando-o em uma sequência de bytes que pode ser enviada pela rede. Em seguida, o cliente empacota o frame serializado junto com o tamanho da mensagem (payload) em um cabeçalho usando 'struct.pack()'. O formato 'Q' é usado novamente para representar um número inteiro não assinado de 64 bits (8 bytes), que representa o tamanho do payload.

O cliente envia a mensagem para o servidor usando 'client_socket.sendall()', que envia todos os bytes da mensagem em uma única chamada de sistema.

Após isso o frame é exibido localmente usando 'cv.imshow()', permitindo que o cliente veja o vídeo capturado em tempo real. O cliente também aguarda uma tecla ser pressionada. Se o usuário pressionar a tecla 'q', o cliente encerra a transmissão do vídeo e sai do loop.

Finalmente, após sair do loop, o cliente libera os recursos da captura de vídeo, fecha a janela de visualização do vídeo usando 'cv.destroyAllWindows()' e fecha o socket de comunicação com o servidor usando '_socket.close()'. Isso encerra a conexão com o servidor e finaliza a videochamada.

2.3 Módulo 'get_wifi_ip'

Como visto nas seções anteriores, o módulo 'get_wifi_ip' foi utilizado com intuito de deixar o código mais dinâmico na procura do endereço IP da rede sem fio em que o código está sendo executado, assim o usuário não iria precisar procurar manualmente e colocar diretamente no código. O código criado pode ser visto na Figura 2.3.

```python
import socket

def get_wifi_ip():

    try:
        s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
        s.connect(("8.8.8.8", 80))
        ip = s.getsockname()[0]
        s.close()
        return ip

    except socket.error as e:
        print("Erro ao obter o IP do adaptador de rede sem fio (Wi-Fi):", e)
        return None

```

Figura 2.3: Script de captura de IP Wi-Fi

Primeiro foi importado a biblioteca 'socket' e depois criado uma função chamada 'get_wifi_ip'.

Dentro da função, é feito a utilização de um bloco 'try-except' para lidar com possíveis erros que podem ocorrer durante a execução do código.

1. s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM): Cria-se um socket UDP (datagrama) usando 'socket.socket()'. O argumento socket.AF_INET especifica que deve-se utilizar a família de endereços IPv4, e socket.SOCK_DGRAM indica que será usado o protocolo UDP.

2. s.connect(("8.8.8.8", 80)): Aqui foi criado uma conexão com o servidor DNS do Google (8.8.8.8) na porta 80 (porta padrão para HTTP). Essa conexão é feita apenas para fins de teste e não tem impacto na rede.

3. ip = s.getsockname()[0]: Após a conexão ser estabelecida, pode-se obter o endereço IP local do adaptador de rede Wi-Fi usando o método 'getsockname()'. O resultado desse método é uma tupla que contém o endereço IP e o número da porta utilizada na conexão. Como interesse é apenas no endereço IP, utiliza-se o índice [0] para acessar o primeiro elemento da tupla, que é o endereço IP.

4. s.close(): Finalmente, o socket é fechado.

Se tudo ocorrer bem, a função retorna o endereço IP obtido do adaptador Wi-Fi. Caso ocorra algum erro, como a impossibilidade de se conectar ao servidor DNS do Google, a exceção 'socket.error' será capturada e uma mensagem de erro será exibida, e a função retornará 'None'. Isso permite que o código que chama essa função saiba que houve um problema na obtenção do endereço IP e possa tomar as medidas necessárias.

3 Referências Bibliográficas

[1] Python OpenCV Socket Programming. Disponível em:
<https://adyraj.medium.com/video-streaming-using-python-ed73dc5bcb30>.

[2] Python FFMPEG. Disponível em :
<https://www.hadronepoch.org/op/python/ffmpeg-streaming/start>.