Guida alla Configurazione per l'Invio di Frame Video da PC a ESP32

Basato sulle interazioni del 16 Maggio 2025, Ancona

16 maggio 2025

Indice

Questa guida descrive i passaggi per configurare un sistema in cui i frame video da una webcam su un PC (Arch Linux) vengono elaborati e inviati via HTTP POST a un microcontrollore ESP32.

1 Fase 1: Configurazione del Sistema Sorgente su PC (Arch Linux)

1.1 Predisporre la Sorgente Video e il Server RTSP

1. Utilizzare una webcam USB (es. Logitech) come sorgente video, identificando il dispositivo corretto (es. `/dev/video2`).
2. Avviare un server RTSP, ad esempio `mediamtx`, eseguendo il suo comando di avvio in un terminale dedicato:

```
./mediamtx

```

Listing 1: Avvio di mediamtx

3. In un secondo terminale, utilizzare `ffmpeg` per inviare lo stream della webcam a `mediamtx`. Un comando di esempio è:

```
ffmpeg -f v4l2 -video_size 640x480 -i /dev/video2 -c:v h264 -preset
    veryfast -tune zerolatency -an -f rtsp -rtsp_transport tcp rtsp://
    localhost:8554/webcam_stream

```

Listing 2: Comando ffmpeg base per lo streaming RTSP

1.2 Ottimizzare lo Stream `ffmpeg` per Basso Carico

Per ridurre il carico sull'ESP32 e sulla rete, modificare il comando `ffmpeg` per diminuire risoluzione e framerate. Ad esempio:

```
ffmpeg -f v4l2 -video_size 160x120 -i /dev/video2 -c:v h264 -preset veryfast -
    tune zerolatency -r 1/3 -an -f rtsp -rtsp_transport tcp rtsp://localhost
    :8554/webcam_stream
```

Listing 3: Comando ffmpeg ottimizzato

Qui, `-video_size 160x120` imposta la risoluzione e `-r 1/3` imposta il framerate a 1 frame ogni 3 secondi.

1.3 Sviluppare uno Script Python per Acquisire, Convertire e Inviare i Frame

1. Creare un file script Python (es. `frame_sender.py`).
2. Assicurare che le librerie necessarie (`opencv-python`, `requests`) siano installate. Se si utilizza un ambiente virtuale (`venv`):

```
source venv/bin/activate # (se l'ambiente si chiama venv)
pip install opencv-python requests

```

Listing 4: Installazione dipendenze Python in venv

3. Nello script Python:

- Importare le librerie: `import cv2`, `import requests`, `import time`, `import signal`, `import sys`.
- Definire l'URL dello stream RTSP e l'URL di destinazione dell'ESP32. Quest'ultimo deve essere configurato con l'IP corretto dell'ESP32:

```
rtsp_url = "rtsp://localhost:8554/webcam_stream"
# SOSTITUIRE <IP_ESP32_QUI> con l'IP effettivo dell'ESP32
esp32_target_url = "http://<IP_ESP32_QUI>/upload_frame"
jpeg_quality = 85 # Qualita JPEG (0-100)

```

Listing 5: Configurazione URL nello script Python

- Connettersi allo stream RTSP:

```
video_capture = cv2.VideoCapture(rtsp_url)
if not video_capture.isOpened():
    print(f"ERRORE: Impossibile connettersi a {rtsp_url}")
    sys.exit(1)

```

Listing 6: Connessione allo stream RTSP

- Implementare un loop per leggere i frame (es. `while not stop_program_flag:`).
- All'interno del loop, leggere un frame: `success, frame_data = video_capture.read()`.
- Se la lettura ha successo, codificare il frame in formato JPEG:

```
encode_param = [int(cv2.IMWRITE_JPEG_QUALITY), jpeg_quality]
result, encoded_jpeg = cv2.imencode('.jpg', frame_data, encode_param)
if not result:
    print("ERRORE: Durante la codifica JPEG.")
    continue

```

Listing 7: Codifica frame in JPEG

- Inviare il frame JPEG codificato all'ESP32 tramite HTTP POST:

```
try:
    response = requests.post(
        esp32_target_url,
        data=encoded_jpeg.tobytes(), # Invia i byte del JPEG
        headers={'Content-Type': 'image/jpeg'},
        timeout=3 # Timeout in secondi
    )
    # Controllare response.status_code e response.text
    if response.status_code == 200:
        print(f"Frame inviato. Risposta ESP32: {response.text[:100]}")
    else:
        print(f"Errore invio. Status: {response.status_code}")
except requests.exceptions.RequestException as e:
    print(f"ERRORE di connessione/richiesta all'ESP32: {e}")

```

Listing 8: Invio frame JPEG via HTTP POST

- Includere la gestione dell'uscita con `Ctrl+C` e il rilascio delle risorse (es. `video_capture.release()`, `cv2.destroyAllWindows()`).

2 Fase 2: Programmazione del Microcontrollore ESP32 (Ambiente Arduino)

2.1 Sviluppare il Firmware per la Ricezione dei Frame

1. Nel file `.ino` per l'ESP32, includere le librerie necessarie:

```
#include <WiFi.h>
#include <WebServer.h>

```

Listing 9: Inclusione librerie ESP32

2. Definire le credenziali Wi-Fi:

```
// SOSTITUIRE CON LE PROPRIE CREDENZIALI
const char* ssid = "IL_TUO_SSID";
const char* password = "LA_TUA_PASSWORD";

```

Listing 10: Credenziali Wi-Fi ESP32

3. Istanziare il server: `WebServer server(80);`
4. Nella funzione `setup()`:
   - Inizializzare la seriale: `Serial.begin(115200);`
   - Connettersi al Wi-Fi e stampare l'IP:

```
WiFi.begin(ssid, password);
while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
}
Serial.println("\nConnesso al Wi-Fi!");
Serial.print("Indirizzo IP ESP32: ");
Serial.println(WiFi.localIP()); // Annotare questo IP

```

Listing 11: Connessione Wi-Fi e stampa IP ESP32

   - Definire l'handler per la ricezione dei frame:

```
void handleFrameUpload() {
    if (server.method() != HTTP_POST || !server.hasArg("plain")) {
        server.send(400, "text/plain", "Bad Request");
        return;
    }
    String requestBody = server.arg("plain");
    int bodyLength = requestBody.length();
    Serial.printf("Frame JPEG ricevuto! Dimensione: %d bytes\n",
    bodyLength);
    // Qui si potrebbero salvare i dati (requestBody.c_str(),
    bodyLength)
    server.send(200, "text/plain", "Frame ricevuto con successo dall'
    ESP32!");
}

```

Listing 12: Handler ricezione frame ESP32

   - Registrare l'handler e avviare il server:

```
server.on("/upload_frame", HTTP_POST, handleFrameUpload);
// server.on("/", HTTP_GET, handleRoot); // Opzionale per pagina di stato
server.begin();
Serial.println("Server HTTP avviato.");

```

Listing 13: Registrazione handler e avvio server ESP32

5. Nella funzione `loop()`:

```
server.handleClient();

```

Listing 14: Loop principale ESP32

3 Fase 3: Esecuzione e Verifica del Sistema Integrato

1. **Preparare e Avviare l'ESP32:**
   - Caricare il firmware compilato sull'ESP32.
   - Aprire il Monitor Seriale dell'IDE Arduino (baud 115200) per osservare i log di avvio, la connessione Wi-Fi e per annotare l'indirizzo IP assegnato all'ESP32.
2. **Preparare e Avviare i Componenti su PC:**
   - In un terminale, avviare il server RTSP `mediamtx`.
   - In un secondo terminale, avviare lo script `ffmpeg` con i parametri per lo stream a bassa risoluzione e basso framerate, indirizzandolo a `mediamtx`.
   - Nello script Python, aggiornare la variabile dell'URL di destinazione (es. `esp32_target_url`) con l'indirizzo IP effettivo dell'ESP32 (ottenuto dal Monitor Seriale).
   - In un terzo terminale (attivando l'ambiente virtuale Python, se utilizzato), eseguire lo script Python.
3. **Monitorare e Verificare il Flusso Dati:**
   - Osservare l'output del terminale dove è in esecuzione lo script Python: dovrebbero apparire messaggi che indicano l'invio dei frame e la ricezione di risposte positive (HTTP 200) dall'ESP32.
   - Osservare l'output del Monitor Seriale dell'ESP32: dovrebbero apparire messaggi che confermano la ricezione di ogni frame JPEG e ne indicano la dimensione.

Seguendo questi passaggi, si configura un sistema in cui i frame video sono catturati, trasmessi via RTSP, elaborati da uno script Python e infine inviati via HTTP POST a un ESP32 che ne conferma la ricezione.