Documentation du Système de flux vidéo Embedded Web Player(EWP)

Louis Hermier

10/01/2021

Contenu

Introduction **3**

Préparation du Raspberry pi **3**

Lancer le code **3**

Pour lancer le programme manuellement 3

Annexe **5**

Introduction

Cette partie du code se sépare en deux parties: La partie gérant l'affichage du flux vidéo sur une page web (ici) et une autre partie qui crée un réseau wifi et redirige le traffic / une URL spécifique vers la page créée précédemment.

Préparation du Raspberry pi

Nous utilisons un Raspberry pi 4B avec 4G de RAM. Cependant, le programme devrait pouvoir fonctionner sur n'importe quel modèle de raspberry pi (avec des limitations évidemment). Nous utilisons Rasperry pi OS lite (anciennement Raspbian) comme système d'exploitation (Basé sur Debian → Gestionnaire de paquet APT, init SystemD, base stable, pas de serveur X / DE / VM). L'administration se fait par SSH en étant sur le même réseau. La caméra du raspberry pi doit être allumée, et certains logiciels doivent être installés avant de pouvoir lancer le code:

1. Python3 (installé par défaut normalement)
2. pip (`sudo apt install python3-pip`)
3. flask (`sudo apt install python3-flask`)
4. opencv (`sudo apt install python3-opencv`)
5. imutils (`pip3 install imutils && sudo pip3 install imutils`)

Lancer le code

Le code peut normalement être lancé automatiquement en placant le dossier `stream-video-browser` dans le dossier home de l'utilisateur pi (`/home/pi/`). On peut ensuite placer le fichier `.service` fourni pour SystemD dans le dossier `/etc/systemd/system/`. Le programme peut alors être lancé sur le port 80 avec la commande suivante:

```
sudo systemctl start droneEWP.service
```

stoppé avec

```
sudo systemctl stop droneEWP.service
```

Pour que le programme se lance automatiquement au démarrage:

```
sudo systemctl enable droneEWP.service
```

Pour retirer cela:

```
sudo systemctl disable droneEWP.service
```

La commande d'invocation, qui correspond à la ligne:

```
ExecStart=/usr/bin/python3 /home/pi/stream-video-browser/webstreaming.py -i 0.0.0.0 -o 80
```

dans le fichier `.service`, est celle qui doit être modifiée pour changer le port (défaut: 80, port standard HTTP, pas besoin de le préciser dans la barre d'adresse du navigateur) ou l'adresse (défaut: 0.0.0.0, écoute sur toutes les adresses possibles) sur laquelle le programme attend des connexions.

Pour lancer le programme manuellement

Pour lancer le programme manuellement, il suffit de se placer dans le dossier `stream-video-browser` et de lancer la commande suivante:

```
python3 webstreaming.py -i 0.0.0.0 -o 5050
```

`-i` doit être suivi de l'adresse sur laquelle le programme écoute, et `-o` définit le port.

Attention! Pour lancer le programme sur certains ports, comme le port 80, le programme a besoin des droits d'administrateur. Pour cela, il suffit de préfixer la commande par `sudo` (super user do) pour que cela fonctionne (par exemple `sudo python3 webstreaming.py -i 0.0.0.0 -o 80`)

Le fichier HTML servant de base à la page web se trouve dans le dossier `templates` situé dans le dossier `stream-video-browser`. La balise qui contient la vidéo est la suivante:

```
<img src="{{ url_for('video_feed') }}">
```

Ce fichier HTML peut bien évidemment être relié à des feuilles de style, des fichiers javascript, … Cependant rien n'est prévu pour d'éventuels fichiers PHP ou autres technologies coté serveur pour l'instant, car python remplit ce rôle.

Annexe

Fichier de configuration `droneEWP.service` complet:

```
[Unit]
Description=Système de streaming Caméra / Raspberry pi
After=network-online.target

[Install]
WantedBy=default.target

[Service]
# needs to run as root to start on port 80!
User=root
# 0.0.0.0: listens on everything
ExecStart=/usr/bin/python3 /home/pi/stream-video-browser/webstreaming.py -i 0.0.0.0 -o 80
```

Code complet du fichier `webstreaming.py`

```
# USAGE
# python webstreaming.py --ip 0.0.0.0 --port 8000

# import the necessary packages
# grabbing frames from the camera
from imutils.video import VideoStream
# web server (builds the web page)
from flask import Response
from flask import Flask
from flask import render_template
# allows multiple clients to connect at the same time
import threading
# parse arguments
import argparse
# time / date
import datetime
import time as t
# rework / treat images
import imutils
# openCV
import cv2

# distance from the border for the text
OFFSET = 10

# temp width, adjust with smartphone / ... width
WIDTH = 800

# timestamp FORMATTING: dd/mm/yyyy, hh:mm:ss
FORMAT = "%d/%m/%Y, %H:%M:%S"

# general text stuff
# set font
font = cv2.FONT_HERSHEY_SIMPLEX
# set color
fontColor = (0, 0, 255) # red
# set line thickness
lineThickness = 1
```

```
# text stuff for the watermark
watermark = "Drone V2 ISFATES - Camera Embarquee"
# size of the text
fontScaleWatermark = 0.5
# get the width and height to position the text better
(tmpWidth, tmpHeight) = cv2.getTextSize(watermark, font, fontScaleWatermark, lineThickness)
# position for the watermark. Bottom left corner.
positionWatermark = (OFFSET, OFFSET + tmpHeight * 2)

# positioning stuff for the timestamp
# position for the timestamp. Bottom left corner
positionTimestamp = (OFFSET, (OFFSET + tmpHeight*2)*2)
# get the time
time = datetime.datetime.now()
# font size
fontScaleTimestamp = 0.4


# initialize the output frame and a lock used to ensure thread-safe
# exchanges of the output frames (useful for multiple browsers/tabs
# are viewing tthe stream)
outputFrame = None
lock = threading.Lock()

# initialize a flask object
app = Flask(__name__)

# initialize the video stream
vs = VideoStream(usePiCamera=1).start()
#vs = VideoStream(src=0).start()
# camera warmup time
t.sleep(2.0)


@app.route("/")
def index():
    # return the rendered template
    return render_template("index.html")

def process_frame():
    # needs to run in a loop, otherwise the app will only display one frame
    # updates the image on every run
    while True:
        # grab global references to the video stream, output frame, and
        # lock variables
        global vs, outputFrame, lock
        # get a frame from the cam
        frame = vs.read()
        #resize it using the width we want
        frame = imutils.resize(frame, width=WIDTH)
        # add the watermark to the image
        cv2.putText(frame, watermark, positionWatermark, font, fontScaleWatermark, fontColor, lineTh
        # get time
        time = datetime.datetime.now()
        # write time to the image
        cv2.putText(frame, time.strftime(FORMAT), positionTimestamp, font, fontScaleTimestamp, fontC
        # acquire the lock, set the output frame, and release the
        # lock
```

```
        with lock:
            # write the "finished" frame to the variable to be updated on the webpage
            outputFrame = frame.copy()

def generate():
    # grab global references to the output frame and lock variables
    global outputFrame, lock

    # loop over frames from the output stream
    while True:
        # wait until the lock is acquired
        with lock:
            # check if the output frame is available, otherwise skip
            # the iteration of the loop
            if outputFrame is None:
                continue

            # encode the frame in JPEG format
            (flag, encodedImage) = cv2.imencode(".jpg", outputFrame)

            # ensure the frame was successfully encoded
            if not flag:
                continue

        # yield the output frame in the byte format
        yield(b'--frame\r\n' b'Content-Type: image/jpeg\r\n\r\n' +
            bytearray(encodedImage) + b'\r\n')

@app.route("/video_feed")
def video_feed():
    # return the response generated along with the specific media
    # type (mime type)
    return Response(generate(),
        mimetype = "multipart/x-mixed-replace; boundary=frame")

# check to see if this is the main thread of execution
if __name__ == '__main__':
    # construct the argument parser and parse command line arguments
    ap = argparse.ArgumentParser()
    ap.add_argument("-i", "--ip", type=str, required=True,
        help="ip address of the device")
    ap.add_argument("-o", "--port", type=int, required=True,
        help="ephemeral port number of the server (1024 to 65535)")
    args = vars(ap.parse_args())

    # start a thread that will process the frames
    t = threading.Thread(target=process_frame)
    t.daemon = True
    t.start()

    # start the flask app
    app.run(host=args["ip"], port=args["port"], debug=True,
        threaded=True, use_reloader=False)

# release the video stream pointer
vs.stop()
```