PROJECT STRUCTURE

TEMPLATES:-

WEBSTREAMING:

|  |  |
| --- | --- |
| Team ID | PNT27022TMID22447 |
| Project Name | Real Time Communication System Powered by AI For Specially Abled |

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

# import the necessary packages

from pyimagesearch.motion\_detection import SingleMotionDetector

from imutils.video import VideoStream

from flask import Response

from flask import Flask

from flask import render\_template

import threading

import argparse

import datetime

import imutils

import time

from cv2 import cv2 as cv2

# 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 and allow the camera sensor to

# warmup

#vs = VideoStream(usePiCamera=1).start()

vs = VideoStream(src=0).start()

time.sleep(2.0)

@app.route("/")

def index():

    # return the rendered template

    return render\_template("index.html")

def detect\_motion(frameCount):

    # grab global references to the video stream, output frame, and

    # lock variables

    global vs, outputFrame, lock

    # initialize the motion detector and the total number of frames

    # read thus far

    md = SingleMotionDetector(accumWeight=0.1)

    total = 0

    # loop over frames from the video stream

    while True:

        # read the next frame from the video stream, resize it,

        # convert the frame to grayscale, and blur it

        frame = vs.read()

        frame = imutils.resize(frame, width=400)

        gray = cv2.cvtColor(frame, cv2.COLOR\_BGR2GRAY)

        gray = cv2.GaussianBlur(gray, (7, 7), 0)

        # grab the current timestamp and draw it on the frame

        timestamp = datetime.datetime.now()

        cv2.putText(frame, timestamp.strftime(

            "%A %d %B %Y %I:%M:%S%p"), (10, frame.shape[0] - 10),

            cv2.FONT\_HERSHEY\_SIMPLEX, 0.35, (0, 0, 255), 1)

        # if the total number of frames has reached a sufficient

        # number to construct a reasonable background model, then

        # continue to process the frame

        if total > frameCount:

            # detect motion in the image

            motion = md.detect(gray)

            # cehck to see if motion was found in the frame

            if motion is not None:

                # unpack the tuple and draw the box surrounding the

                # "motion area" on the output frame

                (thresh, (minX, minY, maxX, maxY)) = motion

                cv2.rectangle(frame, (minX, minY), (maxX, maxY),

                    (0, 0, 255), 2)

        # update the background model and increment the total number

        # of frames read thus far

        md.update(gray)

        total += 1

        # acquire the lock, set the output frame, and release the

        # lock

        with lock:

            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)")

    ap.add\_argument("-f", "--frame-count", type=int, default=32,

        help="# of frames used to construct the background model")

    args = vars(ap.parse\_args())

    # start a thread that will perform motion detection

    t = threading.Thread(target=detect\_motion, args=(

        args["frame\_count"],))

    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()