```python
"""entry_detect.py | Robin Forestier | 09.03.2022

This file is used to detect entry and exit. It uses DoorSelect & PersonneDetect & PersonneTracking.
"""

import math
import pickle
import time
from datetime import datetime
from os import path

import cv2
import numpy as np
import requests

# import personally module
from sample.door_select import DoorSelect
from sample.personne_detect import PersonneDetect
from sample.personne_tracking import PersonneTracking


class DetectEntry:
    """DetectEntry class is used to detect entry and exit."""
    def __init__(self, frame, doors, fleches):
        """The function takes in the frame, the doors and the arrows to calcul of a personne is going in or out.

        :param frame: The image frame that the camera captured
        :type frame: numpy.ndarray
        :param doors: a list of tuples, each tuple is a door, with the first element being the x-coordinate of the
        center of the door, and the second element being the y-coordinate of the center of the door
        :type doors: list
        :param fleches: a list of tuples, each tuple is a pair of points, the first point is the center of the arrow,
        the second is the tip of the arrow
        :type fleches: list
        """

        self.img = frame
        self.doors = doors
        self.fleches = fleches

        self.angle = []
        self.angle2 = []

    def verify_door_pos(self):
        # TODO add door_detect
        pass

    def calculate(self, d_select):
        """Calculate the angle between the center of the door and the arrow

        :param d_select: the door_select object
        :type d_select: DoorSelect
        """

        n_door = 0

        # Used to set the image and the doors and arrows position to the DoorSelect object.
        d_select.img = frame
        d_select.doors = pos
        d_select.fleches = f_pos

        for door in zip(*[iter(self.doors)] * 4):
            rect = np.zeros((4, 2), dtype="float32")

            # Summing the door points.
            s = np.sum(door, axis=1)
            # It's calculating the minimum and maximum value of the door.
            rect[0] = door[np.argmin(s)]
            rect[2] = door[np.argmax(s)]

            # It's calculating the difference between the door points.
            diff = np.diff(door, axis=1)
            # It's calculating the minimum and maximum value of the door.
            rect[1] = door[np.argmin(diff)]
            rect[3] = door[np.argmax(diff)]

            # It's converting the rect from float to int.
            rect = rect.astype(int)

            # It's calculating the center of th bottom of the door.
            center = (int((rect[3][0] + rect[2][0]) / 2), int((rect[3][1] + rect[2][1]) / 2))

            # It's calculating the angle between the center of the door and the bottom of the door.
            y = center[1] - rect[2][1]
            x = center[0] - rect[2][0]
            angle = math.atan2(y, x) * 180 / math.pi

            # It's making sure that the angle is between 0 and 360.
            if angle < 0:
                angle = 360 + angle

            self.angle.append(angle)

            # It's calculating the angle between the center of the door and the tip of the arrow.
            y = center[1] - self.fleches[n_door][1]
            x = center[0] - self.fleches[n_door][0]
            angle2 = math.atan2(y, x) * 180 / math.pi

            # It's making sure that the angle is between 0 and 360.
            if angle2 < 0:
                angle2 = 360 + angle2

            self.angle2.append(angle2)

            n_door = n_door + 1

    def calc_in_out(self, frame, d_select, tracking, info):
        """Calculate if the person is going inside or outside

        :param frame: the frame from your video file or directly from your webcam
        :type frame: numpy.ndarray
        :param d_select: the door selection object
        :type d_select: DoorSelect
        :param tracking: the PersonneTracking object
        :type tracking: PersonneTracking
        :param info: an object of the Info class, which is used to store information about the people detected in the frame
        :type info: Info
        """

        # It's drawing the doors on the frame.
        d_select.img = frame
        d_select.draw_door()

        n_door = 0

        for door in zip(*[iter(d_select.doors)] * 4):
            door = np.array(door)
            x, y, w, h = cv2.boundingRect(door)

            n = 0

            for a in tracking.angle:
                # It's checking if the person is in the door.
                if x < tracking.centroide[n][0] < x + w and y <
                tracking.centroide[n][1] < y + h:
                    if self.angle[n_door] - 180 < 0:
                        angle2 = 360 - (self.angle[n_door] + 180)
                    else:
                        angle2 = self.angle[n_door] - 180

                    if 10 < a - angle2 < 170:
                        if self.angle2[n_door] - angle2 < 170:
                            info.queue.append([time.time(), 0, n_door, n])
                        else:
                            info.queue.append([time.time(), 1, n_door, n])
                    elif 190 < a - angle2 < 350:
                        if self.angle2[n_door] - angle2 < 170:
                            info.queue.append([time.time(), 1, n_door, n])
                        else:
                            info.queue.append([time.time(), 0, n_door, n])
                    else:
                        pass

                n = n + 1
            n_door = n_door + 1


class info:
    """class info is used for sorting the value from calc_in_out (class
    DetectEntry)"""
    def __init__(self):
        """is a list of the last in/out value.
        [time, in (1) / out (0), door_num, personne_num]
        """
        self.queue = []

    def verify(self):
        """Verify check all value in queue list.
        First I delete all value is too old (more than 5sec).
        After if we have 3 or more same detection, a validation is sent (return 0 or
        1)

        :return: 0 or 1
        :rtype: int
        """

        # It's checking if there is more than 3 detection.
        #   If there is, it's checking which door has the most detection.
        #   If there is more than one door with the same number of detection, it's
        checking which person is the
        # closest to the center of the door.
        #   If there is only one door with the most detection, it's checking if the
        person is going inside or outside.
        #    If the person is going inside, it's sending 1 to the server.
        #    If the person is going outside, it's sending 0 to the server.
        if len(self.queue) >= 3:
            # It's converting the queue list to a numpy array.
            out = np.array(self.queue).T
            temp_ex = []

            # It's converting the array to int.
            out = out.astype(int)

            for t in out[0]:
                if t < out[0][len(self.queue) - 1] - 5:
                    temp_ex.append(np.where(out[0] == t))

            # It's deleting the value in the array that are in temp_ex.
            out = np.delete(out, temp_ex, axis=1)

            inout = np.bincount(out[1]).argmax()

            if max(np.bincount(out[1])) <= max(np.bincount(out[2])) and \
                    max(np.bincount(out[1])) <= max(np.bincount(out[3])):
                self.queue = []
```

```python
                # It's returning the value of the verification of the queue list.
                return inout


def send(info):
    """Send the data to the server

    :param info: the information to send
    :type info: int
    """

    # It's getting the current time.
    t = datetime.now()
    current_time = t.strftime("%H:%M")

    # It's opening the file /sys/class/thermal/thermal_zone0/temp and reading the temperature.
    try:
        with open('/sys/class/thermal/thermal_zone0/temp', 'r') as ftemp:
            temp = int(int(ftemp.read()) / 1000)
    except OSError:
        temp = 0

    # It's creating a string that will be sent to the server.
    data = "{}{:03d}{}".format(current_time, temp, info)
    data = {'data': '$,RPWCSD,{:03d},{},0*'.format(len(data), data)}

    try:
        # It's sending the data to the server.
        r = requests.post("http://172.16.32.133/camera", data=data, timeout=0.5)

        # This is checking if the status code is bigger than 299. If it is, it's printing an error message.
        if r.status_code > 299:
            print("[Error] Communication error")
        else:
            # It's getting the data from the server.
            data = r.text
            # if the data is a correct trame ($,...,*)
            if data[0] == "$" and data[::-1][0] == "*":
                data = data.split(',')

                # Communication OK
                if data[1] == "RPWCOK":
                    print("ok")
                # Communication Error
                if data[1] == "RPWCER":
                    print("[ERROR] The cam had send a bad trame.")

    # It's catching the error if the server is not available.
    except requests.exceptions.RequestException as e:
        print(e)


if __name__ == '__main__':
    # It's opening the webcam.
    cap = cv2.VideoCapture(0)

    # It's getting the frame from the webcam.
    _, frame = cap.read()

    # It's resizing the frame to 640x480.
    frame = cv2.resize(frame, (640, 480), interpolation=cv2.INTER_AREA)

    # This is checking if the file "doors.pickle" exists. If it doesn't, it's creating a new DoorSelect object and
    #   saving the doors and the arrows position in the file "doors.pickle".
    #   If the file "doors.pickle" exists, it's loading the data from the file.
    if not path.exists("doors.pickle"):
        d_select = DoorSelect(frame)
        pos = d_select.doors
        f_pos = d_select.fleches
    else:
        d_select = DoorSelect()
        with open('doors.pickle', 'rb') as f:
            pos, f_pos = pickle.load(f)

    # It's creating a PersonneDetect object and storing it in the variable `detect`.
    detect = PersonneDetect()
    # It's creating a PersonneTracking object and storing it in the variable `tracking`.
    tracking = PersonneTracking()
    # It's creating an object of the class DetectEntry and storing it in the variable `entry`.
    entry = DetectEntry(frame, pos, f_pos)
    # It's calculating the angle between the center of the door and the arrow.
    entry.calculate(d_select)
    # It's creating an object of the class `info` and storing it in the variable `inf`.
    inf = info()

    while True:
        # This is checking if the video is over. If it is, it's reseting the frame to the first frame.
        if cap.get(cv2.CAP_PROP_POS_FRAMES) == cap.get(cv2.CAP_PROP_FRAME_COUNT):
            cap.set(cv2.CAP_PROP_POS_FRAMES, 0)

        # It's getting the frame from the webcam.
        _, frame = cap.read()

        # It's resizing the frame to 640x480.
        frame = cv2.resize(frame, (640, 480), interpolation=cv2.INTER_AREA)

        # It's detecting people in the frame.
        frame = detect.personne_detect(frame)
        # It's calculating the centroid of the detected people.
        tracking.calc_centroide(frame, detect.detected)
        # It's calculating if the person is going inside or outside.
        entry.calc_in_out(frame, d_select, tracking, inf)

        # It's checking if the person is going inside or outside.
        #   If the person is going inside, it's sending 1 to the server.
        #   If the person is going outside, it's sending 0 to the server.
        inout = inf.verify()

        if inout == 1:
            print("[INFO] Entrée")
            send(inout)
        elif inout == 0:
            print("[INFO] Sortie")
            send(inout)

        # It's showing the image in a window.
        cv2.imshow("image", frame)

        # It's checking if the user press the key "q". If it is, it's breaking the loop.
        if cv2.waitKey(100) == ord("q"):
            break

    # It's closing the webcam and the window.
    cv2.destroyAllWindows()
    cap.release()
```