```python
"""door_select.py | Robin Forestier | 07.03.2022

This file is used to select doors in images.
"""

import cv2
import numpy as np
import pickle


class DoorSelect:
    """The DoorSelect class is a class that contains a method that allows the user to select a door."""
    def __init__(self, img = None):
        """ __init__ is the constructor of the class DoorSelect.
        :param img: the image that will be used to select the doors.
        :type img: numpy.ndarray
        """

        # the image
        self.img = img


        self.doors = []
        """Doors
        List containing the 4 corners coordinations of each door.
        """

        self.fleches = []
        """Fleches
        List containing the coordination of the arrowhead.
        """

        # number of point already placed (return to 0 after each door)
        self.npoints = 0

        if img is not None:
            # a copy of the clean image (used when you delete a selection)
            self.img_copy = img.copy()
            cv2.namedWindow("door select")
            cv2.setMouseCallback('door select', self.mouse_event)
            self.run()

    def run(self):
        """
        [INFO] Doors selection phase.
        [INFO] To select a door:
        [INFO] 1. Left click the four corner of the door.
        [INFO] 2. left click on the side where you enter the room through the door.
        [INFO] Repeat these 2 steps as many times as you have doors.
        [INFO] If you want to delete a selection, right click on it.
        [INFO] Press < SPACE > when you are done.
        """

        print(self.run.__doc__)

        # The loop is infinite until the user press the <SPACE> key.
        while cv2.waitKey(1) != 32:
            cv2.imshow("door select", self.img)

        # check if doors are selected (with an arrow)
        if len(self.doors) % 4 == 0 and len(self.fleches)  == len(self.doors) / 4:
            with open('doors.pickle', 'wb') as f:
                # Save the list self.doors and self.fleches with pickle
                save = (self.doors, self.fleches)
                pickle.dump(save, f)

            print("[INFO] Selected doors successfully saved.\n")

            i = 0
            # This is a way to iterate over the list self.doors 4 by 4.
            for door in zip(*[iter(self.doors)] * 4):
```

```
72                  arr = np.array(door)
73                  x, y, w, h = cv2.boundingRect(arr)
74                  #cv2.rectangle(self.img, (x, y), (x + w, y + h), (0, 255, 0), 2)
75                  img_croped = self.img_copy[y:y+h, x:x+w]
76                  cv2.imwrite("door_{0}.png".format(i), img_croped)
77                  i = i + 1
78
79          else:
80              print("[ERROR] No door select ! \n")
81
82          print("[INFO] Press < ENTER > to close the app.")
83          print("[INFO] Press < SPACE > to select a new door. \n")
84
85          # retry or close the app
86          while True:
87              key = cv2.waitKey(1)
88              if key == 32: # space
89                  self.run()
90              elif key == 13: # enter
91                  print("[INFO] Stop the app.")
92                  cv2.destroyAllWindows()
93                  break
94
95      def sort_points(self, door):
96          """sort_points is used for sorting the 4 corner of the door like that:
97          0      1
98           +----+
99           |    |
100          +----+
101         2      3
102
103         :param door: a list of four points that represent the four corners of the door
104         :type door: list
105         :return: a list of 4 points that represent the corners of the door.
106         :rtype: list
107         """
108
109         # It creates a numpy array of 4 rows and 2 columns.
110         rect = np.zeros((4, 2), dtype="float32")
111
112         # It's summing the y coordinates of the points.
113         s = np.sum(door, axis=1)
114
115         # min -> corner 0
116         # max -> corner 2
117         rect[0] = door[np.argmin(s)]
118         rect[2] = door[np.argmax(s)]
119         # diff of the poits y
120
121         # It contains the difference between the y coordinates of the points.
122         diff = np.diff(door, axis=1)
123         # min -> corner 1
124         # max -> corner 3
125         rect[1] = door[np.argmin(diff)]
126         rect[3] = door[np.argmax(diff)]
127
128         return rect
129
130     def draw_door(self):
131         """Draw the door outline and the arrow"""
132
133         n_door = 0
134         # drawing of the 4 points arround the door
135         for point in self.doors:
136             cv2.circle(self.img, point, 4, (0, 0, 255), -1)
137         # drawing of the arrowhead
138         for point in self.fleches:
139             cv2.circle(self.img, point, 4, (0, 255, 0), -1)
140
141         # drawing of all the lines
142         for door in zip(*[iter(self.doors)] * 4):
143             # It's sorting the 4 points of the door in order to draw the door.
```

```
144                 rect = self.sort_points(door)
145
146                 # It's converting the points from float to integer.
147                 rect = rect.astype(int)
148
149                 # It's drawing the door.
150                 cv2.line(self.img, tuple(rect[0]), tuple(rect[1]), (0, 0, 255), 1)
151                 cv2.line(self.img, tuple(rect[1]), tuple(rect[2]), (0, 0, 255), 1)
152                 cv2.line(self.img, tuple(rect[2]), tuple(rect[3]), (0, 0, 255), 1)
153                 cv2.line(self.img, tuple(rect[3]), tuple(rect[0]), (0, 0, 255), 1)
154
155                 # It's computing the center of the bottom of the door.
156                 center = (int((rect[3][0] + rect[2][0]) / 2), int((rect[3][1] +
                    rect[2][1]) / 2))
157
158                 cv2.circle(self.img, center, 4, (0, 0, 255), -1)
159
160                 # This is a way to draw the arrowhead.
161                 if len(self.fleches) >= n_door + 1:
162                     cv2.line(self.img, center, self.fleches[n_door], (255, 0, 0), 2)
163
164                 n_door = n_door + 1
165
166         def delete_door(self, x, y):
167             """Delete a door already selected by right click on it
168
169             :param x: The x-coordinate of the mouse-click
170             :type x: int
171             :param y: The y-coordinate of the point
172             :type y: int
173             """
174
175             n_door = 0
176             for door in zip(*[iter(self.doors)] * 4):
177                 # sort the points
178                 rect = self.sort_points(door)
179                 rect = rect.astype(int)
180                 # if you click between the point 0 and 3
181                 if rect[0][0] < x < rect[2][0] and rect[0][1] < y < rect[2][1]:
182                     print("[INFO] Door deleted \n")
183
184                     # delete the 4 points of the door
185                     del(self.doors[n_door:n_door + 4])
186
187                     # if it's create delete the coresponding arrow
188                     if len(self.fleches)  >= (len(self.doors) + 4) / 4:
189                         del(self.fleches[int(n_door / 4)])
190                     else:
191                         # if the arrow has not been created, the next point should not
                            be the arrow.
192                         self.npoints = 0
193
194                     self.img = self.img_copy.copy()
195                     self.draw_door()
196
197                 n_door = n_door + 4
198
199         def mouse_event(self, event, x, y, flags, params):
200             """Execute when mouse is used on image.
201
202             :param event: The event that took place (left mouse button pressed, left
                mouse button released, mouse movement, etc)
203             :type event: int
204             :param x: The x-coordinate of the event
205             :type x: int
206             :param y: The y-coordinate of the click
207             :type y: int
208             :param flags: The flags are the optional parameters to the mouse callback
                function
209             :type flags: int
210             :param params: extra parameters passed to the callback function
211             :type params: int
```

```
212             """
213
214             # Checking if the left button of the mouse is pressed.
215             if event == cv2.EVENT_LBUTTONDOWN:
216                 # outline selection
217                 if self.npoints <= 3:
218                     self.doors.append((x, y))
219                     self.npoints = self.npoints + 1
220                 # fleche selection
221                 elif self.npoints < 5:
222                     self.fleches.append((x, y))
223                     self.npoints = self.npoints + 1
224
225                 # new selection
226                 if self.npoints >= 5:
227                     self.npoints = 0
228
229             # This is a way to draw the door when the mouse is released.
230             elif event == cv2.EVENT_LBUTTONUP:
231                 self.draw_door()
232
233             # Checking if the right button of the mouse is pressed.
234             if event == cv2.EVENT_RBUTTONDOWN:
235                 self.delete_door(x, y)
236
237
238     if __name__ == '__main__':
239         # read the file video_d.avi
240         cap = cv2.VideoCapture(0)
241         # take the first img of the video
242         _, img = cap.read()
243         img = cv2.resize(img, (640, 480), interpolation=cv2.INTER_AREA)
244         # create object DoorSelect
245         d = DoorSelect(img)
246
```

```python
"""door_detect.py | Robin forestier | 08.03.2022

This code is used for finding the door with the existing template create by <door_select.py >.
"""

import glob
import cv2

# It's used to capture the video from the camera.
cap = cv2.VideoCapture(0)


def load_images_from_folder():
    """Load all the template images from the current directory

    :return: A list of images.
    :rtype: list
    """

    # It's a function that return a list of all the files with the extension .png in the current directory.
    filenames = glob.glob("*.png")
    # Sort it by name
    filenames.sort()
    images = []

    # It's a loop for loading all the images in the current directory.
    for img in filenames:
        n = cv2.imread(img)
        if n is not None:
            print("[INFO] Door template loaded.")
            images.append(n)
        else:
            print("[Error]  " + img + " Not load.")

    return images


def detectDors(img, template):
    """We use template matching to detect the doors

    :param img: The image we want to detect the template on
    :type img: numpy.ndarray
    :param template: the template image
    :type template: numpy.ndarray
    :return: the image with the rectangles around the detected doors, the max location of the template and the width and
    height of the template.
    :rtype: numpy.ndarray, tuple, tuple
    """

    # It's converting the image from BGR to gray.
    gray_img = cv2.cvtColor(img, cv2.COLOR_BGRA2GRAY)
    # It's making a copy of the image to draw the rectangles on it.
    copy = img.copy()

    # for each template
    for tmp in template:
        # It's converting the template from BGR to gray.
        tmp = cv2.cvtColor(tmp, cv2.COLOR_BGRA2GRAY)
        # It's getting the width and the height of the template.
        w, h = tmp.shape[::-1]
        # It's matching the template to the image.
        res = cv2.matchTemplate(gray_img, tmp, cv2.TM_CCOEFF_NORMED)
        # Normalize result
        cv2.normalize(res, res, 0, 1, cv2.NORM_MINMAX, -1)
        # Detect the max location.
        (_, max_val, _, max_loc) = cv2.minMaxLoc(res)

        # Draw the rect around the detected template
        cv2.rectangle(copy, max_loc, (max_loc[0] + w, max_loc[1] + h), (255, 0, 0), 2)
```

```
70          cv2.rectangle(copy, (max_loc[0] + 1, max_loc[1] + 1), (max_loc[0] + w,
            max_loc[1] + int(h / 2)), (255,255,0), -1)
71          cv2.rectangle(copy, (max_loc[0] + 1, max_loc[1] + h - 1), (max_loc[0] + w,
            max_loc[1] + int(h / 2) + 1), (0, 255, 255), -1)
72
73      return copy, max_loc, w, h
74
75
76  if __name__ == '__main__':
77      # It's loading all the template images from the current directory.
78      template = load_images_from_folder()
79
80      while True:
81          # It's getting the image from the camera and storing it in the variable `img`.
82          _, img = cap.read()
83
84          # resize image form (2592, 1944) -> (640, 480)
85          img = cv2.resize(img, (640, 480), interpolation=cv2.INTER_AREA)
86          result = detectDors(img, template)
87          # It's showing the image with the rectangles around the detected template.
88          cv2.imshow("Result", result)
89
90          # It's waiting for the user to press the key `q` to quit the program.
91          if cv2.waitKey(1) == ord("q"):
92              break
93
94      # It's closing the camera and the windows.
95      cv2.destroyAllWindows()
96      cap.release()
97
```

```python
"""personne_detect.py | Robin Forestier | 8.03.2022

Detecting moving personne on video.
"""

# import OpenCV
import cv2


class PersonneDetect:
    """This class is used to detect people in a video stream."""
    def __init__(self):
        self.img = []
        self.copy = []
        self.detected = []
        self.backSub = cv2.createBackgroundSubtractorKNN(history=100,
        dist2Threshold=500.0, detectShadows=True)

    def img_to_gray(self):
        """If the image is in color, convert it to grayscale """

        if len(self.img.shape) == 3:
            self.img = cv2.cvtColor(self.img, cv2.COLOR_BGR2GRAY)
        else:
            pass

    def contour_detect(self, threshold):
        """Detect the biggest contours in the image and store them in a list

        :param threshold: The threshold image that was used
        :type threshold: numpy.ndarray
        """

        self.detected = []

        # Finding contours in the image.
        cnts, hierarchy = cv2.findContours(threshold, cv2.RETR_EXTERNAL,
        cv2.CHAIN_APPROX_SIMPLE)

        # for eache contour
        for cnt in cnts:
            # if the perimeter is bigger than 100
            if 100 < cv2.arcLength(cnt, True) < 2000:
                # creting a bounding rect around it.
                # Creating a bounding rectangle around the contour.
                x, y, w, h = cv2.boundingRect(cnt)
                # store it
                self.detected.append([x, y, w, h])
                # draw a green rectangle.
                cv2.rectangle(self.copy, (x, y), (x + w, y + h), (0, 255, 0), 3)

    def personne_detect(self, img):
        """Detecting personne on image with background subtraction (KNN)

        :param img: The input image
        :type img: numpy.ndarray
        :return: the copy of the image with the green rectangle around the detected
        personne.
        :rtype: numpy.ndarray
        """

        self.img = img
        self.copy = img.copy()

        # Converting the image to grayscale if it is in color.
        self.img_to_gray()
        # Applying the background substractor to the image.
        fgmask = self.backSub.apply(self.img)
        # Blurring the image to remove the noise.
        blurImage = cv2.GaussianBlur(fgmask, (5, 5), 0)
        # Thresholding the image to make it binary.
        _, th = cv2.threshold(blurImage, 1, 255, cv2.THRESH_BINARY)
```

```
70
71          # Realising 3 morphology transformation to clear the image of impure pixel.
72          # To dilate the shape and close it.
73          # kernel = np.ones((5, 5), np.uint8)
74          kernel = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (7, 7))
75          th = cv2.morphologyEx(th, cv2.MORPH_OPEN, kernel)
76          th = cv2.morphologyEx(th, cv2.MORPH_CLOSE, kernel)
77          th = cv2.dilate(th, kernel, iterations=1)
78
79          # call contour_detect for detect them.
80          self.contour_detect(th)
81
82          # return th copy of the img (with the green rectangle)
83          return self.copy
84
85
86  if __name__ == '__main__':
87      # Opening the video file.
88      cap = cv2.VideoCapture("video_d.avi")
89      # Creating an object of the class PersonneDetect.
90      p = PersonneDetect()
91
92      while True:
93          # Reading the next frame from the video file.
94          _, img = cap.read()
95          # Resizing the image to a smaller size to make the algorithm faster.
96          img = cv2.resize(img, (640, 480), interpolation=cv2.INTER_AREA)
97
98          # Calling the function `personne_detect` of the class `PersonneDetect` and
            passing the image `img` as argument.
99          img = p.personne_detect(img)
100
101         # Showing the image in a window named "img".
102         cv2.imshow("img", img)
103
104         # Stop the program when the user press the key `q`.
105         if cv2.waitKey(50) == ord("q"):
106             break
107
108     # Closing the video file and destroying all the windows.
109     cv2.destroyAllWindows()
110     cap.release()
111
```

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

After Personne detection we want to track it.
For tracking the displacement of a mooving object, I start by calculate his centroide.
After I save his last centroide to ave 2 points by mooving object.
With this points a calculate the euclidean distance to find the nearest.
And I finishe by calculate the angle of displacement.
"""

import cv2
import math
import numpy as np

# It's importing the PersonneDetect class from the personne_detect.py file.
from personne_detect import PersonneDetect

class PersonneTracking:
    """Is used for track the trajectory of any people detected by PersonneDetect."""
    def __init__(self):
        """The function initializes the class"""
        self.img = []
        self.prev_img = []
        self.centroide = []
        self.centroide_lp = []

        self.angle = []

    def calc_centroide(self, img, rects):
        """Calculate the centroid of the bounding rect

        :param img: The image on which the contour was found
        :type img: numpy.ndarray
        :param rects: a list of tuples, where each tuple is (x, y, w, h)
        :type rects: list
        """

        self.img = img
        # save the last centroid
        self.centroide_lp = self.centroide
        self.centroide = []

        for rect in rects:
            # It's the coordinates of the point where the line is drawn.
            x = int(rect[0] + (rect[2] / 2))
            y = int(rect[1] + (rect[3] / 2))
            self.centroide.append((x,y))
            # It's drawing a circle on the image.
            cv2.circle(self.img, (x, y), 2, (0,0, 255), -1)

        # It's calculating the euclidean distance of each centroid and last centroid
        for predict the move of a person.
        self.centroide_last_pose()

    def centroide_last_pose(self):
        """Calculate the angle of displacement of each centroid and last centroid"""

        self.angle = []

        centroide_np = np.array(self.centroide_lp)

        if self.centroide_lp and self.centroide:
            for last_point in self.centroide:
                # This is the code that is used to find the index of the minimum
                value in the array.
                idx = np.array([np.linalg.norm(x + y) for (x, y) in centroide_np -
                last_point]).argmin()

                cv2.circle(self.img, last_point, 2, (255, 0, 0), -1)
                cv2.line(self.img, last_point, self.centroide_lp[idx], (255, 255,
                0), 5, cv2.LINE_AA)

                # It's calculating the angle of the line between the two points.
```

```
                y = last_point[1] - self.centroide_lp[idx][1]
                x = last_point[0] - self.centroide_lp[idx][0]
                angle = math.atan2(y, x) * 180 / math.pi

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

                # add it to the list angle
                self.angle.append(angle)

if __name__ == '__main__':
    # It's using the video file to capture the frames.
    cap = cv2.VideoCapture('video_d.mp4')

    # It's creating an object of the class PersonneDetect.
    p = PersonneDetect()
    # It's creating an instance of the class PersonneTracking.
    pt = PersonneTracking()

    while True:
        # This is a way to reset the video to the first frame if the video is
        finished.
        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 image from the video.
        _, img = cap.read()

        # It's using the PersonneDetect class to detect people in the image.
        result = p.personne_detect(img)
        # It's calculating the centroid of the bounding rect of the detected people.
        pt.calc_centroide(result, p.detected)

        # It's showing the image on the screen.
        cv2.imshow("result detect", result)

        prev = img

        # It's breaking the loop when the user press the `q` key.
        if cv2.waitKey(700) == ord('q'):
            break

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

```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)
```

```
67              # It's calculating the minimum and maximum value of the door.
68              rect[0] = door[np.argmin(s)]
69              rect[2] = door[np.argmax(s)]
70
71              # It's calculating the difference between the door points.
72              diff = np.diff(door, axis=1)
73              # It's calculating the minimum and maximum value of the door.
74              rect[1] = door[np.argmin(diff)]
75              rect[3] = door[np.argmax(diff)]
76
77              # It's converting the rect from float to int.
78              rect = rect.astype(int)
79
80              # It's calculating the center of th bottom of the door.
81              center = (int((rect[3][0] + rect[2][0]) / 2), int((rect[3][1] +
                rect[2][1]) / 2))
82
83              # It's calculating the angle between the center of the door and the
                bottom of the door.
84              y = center[1] - rect[2][1]
85              x = center[0] - rect[2][0]
86              angle = math.atan2(y, x) * 180 / math.pi
87
88              # It's making sure that the angle is between 0 and 360.
89              if angle < 0:
90                  angle = 360 + angle
91
92              self.angle.append(angle)
93
94              # It's calculating the angle between the center of the door and the tip
                of the arrow.
95              y = center[1] - self.fleches[n_door][1]
96              x = center[0] - self.fleches[n_door][0]
97              angle2 = math.atan2(y, x) * 180 / math.pi
98
99              # It's making sure that the angle is between 0 and 360.
100             if angle2 < 0:
101                 angle2 = 360 + angle2
102
103             self.angle2.append(angle2)
104
105             n_door = n_door + 1
106
107     def calc_in_out(self, frame, d_select, tracking, info):
108         """Calculate if the person is going inside or outside
109
110         :param frame: the frame from your video file or directly from your webcam
111         :type frame: numpy.ndarray
112         :param d_select: the door selection object
113         :type d_select: DoorSelect
114         :param tracking: the PersonneTracking object
115         :type tracking: PersonneTracking
116         :param info: an object of the Info class, which is used to store information
                about the people detected in the frame
117         :type info: Info
118         """
119
120         # It's drawing the doors on the frame.
121         d_select.img = frame
122         d_select.draw_door()
123
124         n_door = 0
125
126         for door in zip(*[iter(d_select.doors)] * 4):
127             door = np.array(door)
128             x, y, w, h = cv2.boundingRect(door)
129
130             n = 0
131
132             for a in tracking.angle:
133                 # It's checking if the person is in the door.
134                 if x < tracking.centroide[n][0] < x + w and y <
```

```
                    tracking.centroide[n][1] < y + h:
135                     if self.angle[n_door] - 180 < 0:
136                         angle2 = 360 - (self.angle[n_door] + 180)
137                     else:
138                         angle2 = self.angle[n_door] - 180
139
140                     if 10 < a - angle2 < 170:
141                         if self.angle2[n_door] - angle2 < 170:
142                             info.queue.append([time.time(), 0, n_door, n])
143                         else:
144                             info.queue.append([time.time(), 1, n_door, n])
145                     elif 190 < a - angle2 < 350:
146                         if self.angle2[n_door] - angle2 < 170:
147                             info.queue.append([time.time(), 1, n_door, n])
148                         else:
149                             info.queue.append([time.time(), 0, n_door, n])
150                     else:
151                         pass
152
153                 n = n + 1
154             n_door = n_door + 1
155
156
157 class info:
158     """class info is used for sorting the value from calc_in_out (class
        DetectEntry)"""
159     def __init__(self):
160         """is a list of the last in/out value.
161         [time, in (1) / out (0), door_num, personne_num]
162         """
163         self.queue = []
164
165     def verify(self):
166         """Verify check all value in queue list.
167         First I delete all value is too old (more than 5sec).
168         After if we have 3 or more same detection, a validation is sent (return 0 or
            1)
169
170         :return: 0 or 1
171         :rtype: int
172         """
173
174         # It's checking if there is more than 3 detection.
175         #   If there is, it's checking which door has the most detection.
176         #   If there is more than one door with the same number of detection, it's
            checking which person is the
177         # closest to the center of the door.
178         #   If there is only one door with the most detection, it's checking if the
            person is going inside or outside.
179         #    If the person is going inside, it's sending 1 to the server.
180         #    If the person is going outside, it's sending 0 to the server.
181         if len(self.queue) >= 3:
182             # It's converting the queue list to a numpy array.
183             out = np.array(self.queue).T
184             temp_ex = []
185
186             # It's converting the array to int.
187             out = out.astype(int)
188
189             for t in out[0]:
190                 if t < out[0][len(self.queue) - 1] - 5:
191                     temp_ex.append(np.where(out[0] == t))
192
193             # It's deleting the value in the array that are in temp_ex.
194             out = np.delete(out, temp_ex, axis=1)
195
196             inout = np.bincount(out[1]).argmax()
197
198             if max(np.bincount(out[1])) <= max(np.bincount(out[2])) and \
199                     max(np.bincount(out[1])) <= max(np.bincount(out[3])):
200                 self.queue = []
201
```

```
202                 # It's returning the value of the verification of the queue list.
203                 return inout
204
205
206 def send(info):
207     """Send the data to the server
208
209     :param info: the information to send
210     :type info: int
211     """
212
213     # It's getting the current time.
214     t = datetime.now()
215     current_time = t.strftime("%H:%M")
216
217     # It's opening the file /sys/class/thermal/thermal_zone0/temp and reading the
    temperature.
218     try:
219         with open('/sys/class/thermal/thermal_zone0/temp', 'r') as ftemp:
220             temp = int(int(ftemp.read()) / 1000)
221     except OSError:
222         temp = 0
223
224     # It's creating a string that will be sent to the server.
225     data = "{}{:03d}{}".format(current_time, temp, info)
226     data = {'data': '$,RPWCSD,{:03d},{},0*'.format(len(data), data)}
227
228     try:
229         # It's sending the data to the server.
230         r = requests.post("http://172.16.32.133/camera", data=data, timeout=0.5)
231
232         # This is checking if the status code is bigger than 299. If it is, it's
        printing an error message.
233         if r.status_code > 299:
234             print("[Error] Communication error")
235         else:
236             # It's getting the data from the server.
237             data = r.text
238             # if the data is a correct trame ($,...,*)
239             if data[0] == "$" and data[::-1][0] == "*":
240                 data = data.split(',')
241
242                 # Communication OK
243                 if data[1] == "RPWCOK":
244                     print("ok")
245                 # Communication Error
246                 if data[1] == "RPWCER":
247                     print("[ERROR] The cam had send a bad trame.")
248
249     # It's catching the error if the server is not available.
250     except requests.exceptions.RequestException as e:
251         print(e)
252
253
254 if __name__ == '__main__':
255     # It's opening the webcam.
256     cap = cv2.VideoCapture(0)
257
258     # It's getting the frame from the webcam.
259     _, frame = cap.read()
260
261     # It's resizing the frame to 640x480.
262     frame = cv2.resize(frame, (640, 480), interpolation=cv2.INTER_AREA)
263
264     # This is checking if the file "doors.pickle" exists. If it doesn't, it's
    creating a new DoorSelect object and
265     #   saving the doors and the arrows position in the file "doors.pickle".
266     #   If the file "doors.pickle" exists, it's loading the data from the file.
267     if not path.exists("doors.pickle"):
268         d_select = DoorSelect(frame)
269         pos = d_select.doors
270         f_pos = d_select.fleches
```

```
271         else:
272             d_select = DoorSelect()
273             with open('doors.pickle', 'rb') as f:
274                 pos, f_pos = pickle.load(f)
275 
276         # It's creating a PersonneDetect object and storing it in the variable `detect`.
277         detect = PersonneDetect()
278         # It's creating a PersonneTracking object and storing it in the variable
            `tracking`.
279         tracking = PersonneTracking()
280         # It's creating an object of the class DetectEntry and storing it in the
            variable `entry`.
281         entry = DetectEntry(frame, pos, f_pos)
282         # It's calculating the angle between the center of the door and the arrow.
283         entry.calculate(d_select)
284         # It's creating an object of the class `info` and storing it in the variable
            `inf`.
285         inf = info()
286 
287         while True:
288             # This is checking if the video is over. If it is, it's reseting the frame
                to the first frame.
289             if cap.get(cv2.CAP_PROP_POS_FRAMES) == cap.get(cv2.CAP_PROP_FRAME_COUNT):
290                 cap.set(cv2.CAP_PROP_POS_FRAMES, 0)
291 
292             # It's getting the frame from the webcam.
293             _, frame = cap.read()
294 
295             # It's resizing the frame to 640x480.
296             frame = cv2.resize(frame, (640, 480), interpolation=cv2.INTER_AREA)
297 
298             # It's detecting people in the frame.
299             frame = detect.personne_detect(frame)
300             # It's calculating the centroid of the detected people.
301             tracking.calc_centroide(frame, detect.detected)
302             # It's calculating if the person is going inside or outside.
303             entry.calc_in_out(frame, d_select, tracking, inf)
304 
305             # It's checking if the person is going inside or outside.
306             #   If the person is going inside, it's sending 1 to the server.
307             #   If the person is going outside, it's sending 0 to the server.
308             inout = inf.verify()
309 
310             if inout == 1:
311                 print("[INFO] Entrée")
312                 send(inout)
313             elif inout == 0:
314                 print("[INFO] Sortie")
315                 send(inout)
316 
317             # It's showing the image in a window.
318             cv2.imshow("image", frame)
319 
320             # It's checking if the user press the key "q". If it is, it's breaking the
                loop.
321             if cv2.waitKey(100) == ord("q"):
322                 break
323 
324         # It's closing the webcam and the window.
325         cv2.destroyAllWindows()
326         cap.release()
327 
```

```python
"""personne_detect.py | Robin Forestier | 28.03.2022

[WARN] The camera is placed on top of the door.

Detecting moving personne on video.
"""

# import OpenCV
import cv2
import numpy as np

class PersonneDetect:
    """This class is used to detect people in a video stream."""
    def __init__(self):
        self.img = []
        self.copy = []
        self.detected = []
        self.backSub = cv2.createBackgroundSubtractorKNN(history=100,
        dist2Threshold=500.0, detectShadows=False)

    def img_to_gray(self):
        """If the image is in color, convert it to grayscale """

        if len(self.img.shape) == 3:
            self.img = cv2.cvtColor(self.img, cv2.COLOR_BGR2GRAY)
        else:
            pass

    def contour_detect(self, threshold):
        """Detect the biggest contours in the image and store them in a list

        :param threshold: The threshold image that was used
        :type threshold: numpy.ndarray
        """

        self.detected = []

        # Finding contours in the image.
        cnts, hierarchy = cv2.findContours(threshold, cv2.RETR_EXTERNAL,
        cv2.CHAIN_APPROX_SIMPLE)

        # for eache contour
        for cnt in cnts:
            # if the perimeter is bigger than 100
            if 200 < cv2.arcLength(cnt, True) < 2000:
                # creting a bounding rect around it.
                # Creating a bounding rectangle around the contour.
                x, y, w, h = cv2.boundingRect(cnt)
                # store it
                self.detected.append([x, y, w, h])
                # draw a green rectangle.
                cv2.rectangle(self.copy, (x, y), (x + w, y + h), (0, 255, 0), 3)

    def personne_detect(self, img):
        """Detecting personne on image with background subtraction (KNN)

        :param img: The input image
        :type img: numpy.ndarray
        :return: the copy of the image with the green rectangle around the detected
        personne.
        :rtype: numpy.ndarray
        """

        self.img = img
        self.copy = img.copy()

        # Converting the image to grayscale if it is in color.
        self.img_to_gray()
        # Applying the background substractor to the image.
        fgmask = self.backSub.apply(self.img)
        # Blurring the image to remove the noise.
        blurImage = cv2.GaussianBlur(fgmask, (5, 5), 0)
```

```
70          # Thresholding the image to make it binary.
71          _, th = cv2.threshold(blurImage, 1, 255, cv2.THRESH_BINARY)
72
73          # Realising 4 morphology transformation to clear the image of impure pixel.
74          # To dilate the shape and close it.
75          kernel = np.ones((9, 9), np.uint8)
76          #kernel = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (9, 9))
77          cv2.imshow("th", th)
78          # th = cv2.erode(th, kernel, iterations=1)
79          # th = cv2.morphologyEx(th, cv2.MORPH_OPEN, kernel)
80          th = cv2.morphologyEx(th, cv2.MORPH_CLOSE, kernel)
81          th = cv2.dilate(th, kernel, iterations=2)
82          th = cv2.morphologyEx(th, cv2.MORPH_CLOSE, kernel)
83          cv2.imshow("t", th)
84
85          # call contour_detect for detect them.
86          self.contour_detect(th)
87
88          # return th copy of the img (with the green rectangle)
89          return self.copy
90
91
92  if __name__ == '__main__':
93      # Opening the video file.
94      cap = cv2.VideoCapture("vue_top.mp4")
95      # Creating an object of the class PersonneDetect.
96      p = PersonneDetect()
97
98      while True:
99          # Reading the next frame from the video file.
100         _, img = cap.read()
101         # Resizing the image to a smaller size to make the algorithm faster.
102         # img = cv2.resize(img, (640, 480), interpolation=cv2.INTER_AREA)
103
104         # Calling the function `personne_detect` of the class `PersonneDetect` and
            passing the image `img` as argument.
105         img = p.personne_detect(img)
106
107         # Showing the image in a window named "img".
108         cv2.imshow("img", img)
109
110         # Stop the program when the user press the key `q`.
111         if cv2.waitKey(50) == ord("q"):
112             break
113
114     # Closing the video file and destroying all the windows.
115     cv2.destroyAllWindows()
116     cap.release()
117
```

```python
""" personne_tracking.py | Robin Forestier | 28.03.2022

[WARN] The camera is placed on top of the door.

After Personne detection we want to track it.
For tracking the displacement of a moving object, I start by calculate his centroid.
After I save his last centroid to ave 2 points by moving object.
With these points I calculate the euclidean distance to find the nearest.
With these 2 points, I know the travel of the personne.
"""

# Imports
import cv2
import numpy as np

# It's importing the PersonneDetect class from the personne_detect.py file.
from personne_detect import PersonneDetect

class PersonneTracking:
    """Is used for track the trajectory of any people detected by PersonneDetect."""

    def __init__(self):
        """The function initializes the class"""
        self.img = []
        self.prev_img = []
        self.centroide = []
        self.centroide_lp = []

        self.inout = []

    def calc_centroide(self, img, rects):
        """Calculate the centroid of the bounding rect

        :param img: The image on which the contour was found
        :type img: numpy.ndarray
        :param rects: a list of tuples, where each tuple is (x, y, w, h)
        :type rects: list
        """

        self.img = img
        # save the last centroid
        self.centroide_lp = self.centroide
        self.centroide = []

        for rect in rects:
            # It's the coordinates of the point where the line is drawn.
            x = int(rect[0] + (rect[2] / 2))
            y = int(rect[1] + (rect[3] / 2))
            self.centroide.append((x,y))
            # It's drawing a circle on the image.
            cv2.circle(self.img, (x, y), 2, (0,0, 255), -1)

        # It's calculating the euclidean distance of each centroid and last centroid for predict the move of a person.
        self.centroide_last_pose()

    def centroide_last_pose(self):
        """Calculate if the persone is pacing the center line."""

        # to calculate the distance between the points, I start by knowing which list is the smallest.
        if len(self.centroide) <= len(self.centroide_lp):
            pos1 = np.array(self.centroide)
            pos2 = np.array(self.centroide_lp)
        else:
            pos1 = np.array(self.centroide_lp)
            pos2 = np.array(self.centroide)

        # pos1 have less points than pos2
        # It happens when you start to detect or stop detecting someone.
        if len(pos1) and len(pos2):
            # Size off the image
```

```
71              height = self.img.shape[0]
72              width = self.img.shape[1]
73
74              # line in the middle of the image.
75              cv2.line(self.img, (int(width / 2), 0), (int(width / 2), height),
                (0,255,255), 2, cv2.LINE_AA)
76
77              for i in range(len(pos1)):
78                  # This is the code that is used to find the index of the minimum
                    value in the array.
79                  idx = np.array([np.linalg.norm(x + y) for (x, y) in pos2 -
                    pos1[i]]).argmin()
80
81                  # Draw points and line
82                  cv2.circle(self.img, tuple(pos1[i]), 2, (255, 0, 0), -1)
83                  cv2.line(self.img, tuple(pos1[i]), tuple(pos2[idx]), (255, 255, 0),
                    5, cv2.LINE_AA)
84
85                  # It's checking if the centroid list is smaller than the last
                    centroid list.
86                  # If it's the case, just invert the two variable.
87                  if len(self.centroide) > len(self.centroide_lp):
88                      i, idx = idx, i
89
90                  # It's calculating if the person is moving to the left or to the
                    right.
91                  if self.centroide[i][0] < width / 2 < self.centroide_lp[idx][0]:
92                      self.inout.append(1)
93                  elif self.centroide[i][0] > width / 2 > self.centroide_lp[idx][0]:
94                      self.inout.append(0)
95
96
97      @property
98      def inout(self):
99          """The inout function returns the value of the private variable _inout
100
101         :return: The value of the instance variable _inout
102         :rtype: list
103         """
104         return self._inout
105
106     @inout.setter
107     def inout(self, value):
108         """Set the value of the private variable _inout
109
110         :param value: The value to be set
111         :type value: list
112         """
113         self._inout = value
114
115
116 if __name__ == '__main__':
117     # It's using the video file to capture the frames.
118     cap = cv2.VideoCapture('vue_top.mp4')
119
120     # It's creating an object of the class PersonneDetect.
121     p = PersonneDetect()
122     # It's creating an instance of the class PersonneTracking.
123     pt = PersonneTracking()
124
125     while True:
126         # This is a way to reset the video to the first frame if the video is
            finished.
127         if cap.get(cv2.CAP_PROP_POS_FRAMES) == cap.get(cv2.CAP_PROP_FRAME_COUNT):
128             cap.set(cv2.CAP_PROP_POS_FRAMES, 0)
129
130         # It's getting the image from the video.
131         _, img = cap.read()
132
133         # It's using the PersonneDetect class to detect people in the image.
134         result = p.personne_detect(img)
135         # It's calculating the centroid of the bounding rect of the detected people.
```

```
136             pt.calc_centroide(result, p.detected)
137 
138             # It's showing the image on the screen.
139             cv2.imshow("result detect", result)
140 
141             prev = img
142 
143             # It's breaking the loop when the user press the `q` key.
144             if cv2.waitKey(70) == ord('q'):
145                 break
146 
147         # It's closing the window and release the capture.
148         cv2.destroyAllWindows()
149         cap.release()
150 
```

```python
"""entry_detect.py | Robin Forestier | 28.03.2022

[WARN] The camera is placed on top of the door.

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

# Imports
from datetime import datetime
import cv2
import requests

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


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.
        # Change the url to your own 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, code : ", r.status_code)
        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)

```

```
71      # It's creating a PersonneDetect object and storing it in the variable `detect`.
72      detect = PersonneDetect()
73      # It's creating a PersonneTracking object and storing it in the variable
        `tracking`.
74      tracking = PersonneTracking()
75
76      while True:
77          # It's getting the frame from the webcam.
78          ret, frame = cap.read()
79
80          # It's resizing the frame to 640x480.
81          frame = cv2.resize(frame, (640, 480), interpolation=cv2.INTER_AREA)
82
83          # It's detecting people in the frame.
84          frame = detect.personne_detect(frame)
85          # It's calculating the centroid of the detected people.
86          tracking.calc_centroide(frame, detect.detected)
87
88          inouts = tracking.inout
89
90          for inout in inouts:
91              if inout == 1:
92                  print("[INFO] Entrée")
93                  send(inout)
94              else:
95                  print("[INFO] Sortie")
96                  send(inout)
97
98          tracking.inout.clear()
99
100         # It's showing the image in a window.
101         cv2.imshow("image", frame)
102
103         t = datetime.now()
104         print(t.strftime("%H:%M"))
105
106         # It's checking if the user press the key "q". If it is, it's breaking the
            loop.
107         if cv2.waitKey(1) == ord("q"):
108             break
109
110     # It's closing the webcam and the window.
111     cv2.destroyAllWindows()
112     cap.release()
113
```

```python
"""main.py | Robin Forestier

Fichier de test pour Opencv
"""

# `import cv2` imports the OpenCV library, while `import numpy as np` imports the numpy library.
import cv2
import numpy as np

# Show information about OpenCV
print(__doc__)
print("[INFO] Version d'OpenCV : ", cv2.__version__)
print("[INFO] Version de numpy : ", np.__version__)
print("\n[INFO] Press any buton to quit.")

# Reading the image and storing it in the variable `img`.
img = cv2.imread("test.png")

# Displaying the image `img` in a window called `test`.
cv2.imshow("test", img)

# `cv2.waitKey(0)` waits for a key to be pressed. `cv2.destroyAllWindows()` destroys all windows.
cv2.waitKey(0)
cv2.destroyAllWindows()

```

```python
"""test_communication.py | Robin Forestier | 16.02.2022

This file is used for communication test with the server flask.
The programme send the current time (13:46), the temperature of the camera (054) for 54°C and a bool value 1 or 0
(1 is for entry).
"""

# `time` is a module that contains a lot of functions to work with time.
# `datetime` is a module that contains a lot of functions to work with date and time.
import time
from datetime import datetime

# `requests` is a module that allows to send HTTP requests.
import requests

while True:

    # `t` is a variable that contains the current time.
    t = datetime.now()
    # `strftime` is a function of the `datetime` module. It allows to convert a date into a string.
    current_time = t.strftime("%H:%M")

    # Take the temperature of the RPi
    # Error also temp is 0
    try:
        with open('/sys/class/thermal/thermal_zone0/temp', 'r') as ftemp:
            temp = int(int(ftemp.read()) / 1000)
    except OSError:
        temp = 0

    # Creating a string with the current time and the temperature of the camera.
    data = "{}{:03d}1".format(current_time, temp)
    data_length = len(data)
    data = {'data': '$,RPWCSD,{:03d},{},0*'.format(data_length, data)}

    print(data)

    try:
        # Sending the data to the server (change the ip to your server IP).
        r = requests.post("http://172.16.32.27/camera", data=data, timeout=0.5)
        # Checking if the status code is bigger than 299.
        if r.status_code > 299:
            print("[Error] Communication error")
        else:
            # Reading the data send by 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.")

    # This is a way to catch all the exceptions that can occur when you try to send a request to the server.
    except requests.exceptions.RequestException as e:
        print(e)

    time.sleep(10)

```