```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 
```