Второй отборочный этап

Описание пакета задач

1. Представленные задачи являются декомпозицией задачи заключительного этапа. Они направлены на формирование знаний и компетенций, требуемых на заключительном этапе. Задачи связаны с каждым из трёх видов, программируемых на заключительном этапе устройств, составляющих автономную транспортную систему.
2. Решение представленных задач требует командного подхода: распределение ролей и определение участков работы и разграничения ответственности. Освоение компетенций, необходимых для решения задач одним человеком, невозможно в течении второго этапа. Для решения всех четырёх задач требуются компетенции всех участников команды и их слаженное взаимодействие.
3. Для решения задач второго этапа требуется следующие компетенции: анализ и обработка наборов датасетов, обучение и применение нейросетевых детекторов, знание ROS, создание полётных программ для квадрокоптера «Пионер», чтение технической документации, поиск объектов по цветам, построение алгоритмов и их реализация в программном коде, теория графов.

Все дополнительные материалы к задачам доступны по ссылке: https://disk.yandex.ru/d/gYIk77bOq-0xpA.

Задача IV.1. Подсчёт цветных полос на маркировке грузов (15 баллов)

Темы: компьютерное зрение, детектирование по цвету, бинаризация, контурный анализ.

Условие

На изображении представлены несколько цветовых маркировок грузов. На каждой маркировке может быть от 1-й до 3-х полос. Ваша задача написать функцию, подсчитывающую число цветных полос на каждой маркировке и выводящую полученные значения в порядке возрастания. На изображении не может быть больше 4-х маркировок грузов.

Для вас подготовлены несколько файлов «`.py`», набор изображений и аннотации к нему. Среди подготовленных файлов есть:

- `eval.py` — файл с функцией поиска цветных полос. Именно эту функцию вам необходимо дописать!
- `main.py` — файл проверяющий точность работы вашего алгоритма. Не редактируйте его. `main.py` использует, написанные вами, функции из eval.py и сверяет истинные метки с предсказанием вашего алгоритма.
- `annotations.csv` — файл, устанавливающий соотношение между изображениями и числом цветных полос на нём. В каждой строке файла содержится путь к файлу и количество полос на маркировках по возрастанию.

В качестве решения, необходимо сдать отредактированный файл `eval.py`. Либо архив «`*.zip`» с файлом `eval.py` и остальными файлами, требующимися для его работы.

Технические ограничения

Размер решения ограничен: не более 2 Мбайт. Если ваш алгоритм успешно проверен платформой, то следующее решение можно прислать только через 10 мин. Если ваш алгоритм в ходе проверки выдал сообщение об ошибке, то следующее решения можно прислать сразу.

Пакеты, ориентированные на работу с изображениями и данными, использующиеся на платформе проверки: `Python 3.8.10`.

`dlib 19.24.0; keras 2.8.0; Keras-Preprocessing 1.1.2; imutils 0.5.4; numpy 1.22.4; opencv-python 4.6.0.66; pandas 1.4.3; scikit-image 0.19.3; scikit-learn 1.1.1; scipy 1.8.1; tensorflow-cpu 2.8.2`. Используйте совместимые пакеты.

Решение

Из анализа представленных данных следует, что все маркировки имеют форму чёрных квадратов с несколькими цветными полосами. Фон на всех изображениях однородный зелёный. Задачу можно свести к двум этапам: первый — локализация маркировок — чёрных квадратов, подсчёт цветных полос в каждом квадрате.

Применение нейросетевых методов в данной задаче затруднительно из-за ограничения на размер решения. Локализовать маркировки можно применив детектирование по цвету (пороговая бинаризация + поиск контуров), этот вариант представлен в предлагаемом решении. Возможны альтернативные методы локализации маркировок. Например, сегментация изображения одним из методов: `WaterShed`, `MeanShift`, `FloodFill`. И поиск подходящих по цвету или по площади областей.

Подсчёт цветных полос на каждой маркировке можно выполнить так же, применив детектирование по цвету. Однако для каждой цветной полосы необходимо проводить отдельную операцию бинаризации по уникальным порогам, как это сделано в предлагаемом решении. Возможно определение количества цветных полос по площади занимаемой ими поверхности.

Пример программы-решения

Ниже представлено решение на языке Python 3.

```
1  # -*- coding: utf-8 -*-
2  import cv2
3
4  def find_markers(image) -> list:
5      """
6      Функция для поиска маркировок грузов на изображении и подсчета количества
   ↪  цветных полос на них.
7
8      Входные данные: изображение (bgr), прочитано cv2.imread
9      Выходные данные: список из количества цветных полос на маркировках в порядке
   ↪  возрастания
```

```
10                          на одной маркировке от 0 до 3 полос
11
12      Примеры вывода:
13          [1, 3, 3] - 3 маркировки, на одном из них 1 цветная полоса, на двух других
    ↪   по 3 полосы
14
15          [2] - 1 маркировки, на нем 2 цветные полоски
16
17          [] - маркеры не найдены или отсутствуют цветные полосы
18      """
19      # Алгоритм проверки будет вызывать функцию find_markers,
20      # остальные функции должны вызываться из неё.
21
22      hsv = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)
23
24      blue = cv2.inRange(hsv, (70, 98, 98), (155, 255, 255))
25      green = cv2.inRange(hsv, (35, 100, 100), (85, 255, 255))
26      yellow = cv2.inRange(hsv, (20, 100, 100), (30, 255, 255))
27      red = cv2.inRange(hsv, (174, 110, 110), (178, 255, 255))
28
29      gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
30      ret, thresh = cv2.threshold(gray, 100, 255, cv2.THRESH_BINARY_INV)
31
32      contours, hierarchy = cv2.findContours(thresh, cv2.RETR_TREE,
        ↪   cv2.CHAIN_APPROX_SIMPLE)
33
34      contours = [cnt for cnt in contours if cv2.contourArea(cnt) > 10000]
35
36      bboxes = [cv2.boundingRect(cnt) for cnt in contours]
37      answer = []
38      for i, (x, y, w, h) in enumerate(bboxes):
39          box = hsv[y:y + h, x:x + w]
40          cv2.rectangle(image, (x, y), (x + w, y + h), (0, 0, 255), 2)
41          cv2.putText(image, str(i), (x, y - 10), cv2.FONT_HERSHEY_SIMPLEX, 1, (0,
            ↪   0, 255), 2)
42
43          cnts = cv2.findContours(blue[y:y + h, x:x + w], cv2.RETR_EXTERNAL,
            ↪   cv2.CHAIN_APPROX_SIMPLE)
44          cnts = cnts[0] if len(cnts) == 2 else cnts[1]
45          cnts = [cnt for cnt in cnts if 1000 < cv2.contourArea(cnt) < 5000]
46          blue_slices = len(cnts)
47          print('blue:', blue_slices)
48
49          cnts = cv2.findContours(green[y:y + h, x:x + w], cv2.RETR_EXTERNAL,
            ↪   cv2.CHAIN_APPROX_SIMPLE)
50          cnts = cnts[0] if len(cnts) == 2 else cnts[1]
51          cnts = [cnt for cnt in cnts if 1000 < cv2.contourArea(cnt) < 5000]
52          green_slices = len(cnts)
53          print('green:', green_slices)
54
55          cnts = cv2.findContours(yellow[y:y + h, x:x + w], cv2.RETR_EXTERNAL,
            ↪   cv2.CHAIN_APPROX_SIMPLE)
56          cnts = cnts[0] if len(cnts) == 2 else cnts[1]
57          cnts = [cnt for cnt in cnts if 1000 < cv2.contourArea(cnt) < 5000]
58          yellow_slices = len(cnts)
59          print('yellow:', yellow_slices)
60
61          cnts = cv2.findContours(red[y:y + h, x:x + w], cv2.RETR_EXTERNAL,
            ↪   cv2.CHAIN_APPROX_SIMPLE)
62          cnts = cnts[0] if len(cnts) == 2 else cnts[1]
```

```
63          cnts = [cnt for cnt in cnts if 1000 < cv2.contourArea(cnt) < 5000]
64          red_slices = len(cnts)
65          print('red:', red_slices)
66
67          print(f'answer for box {i}:', blue_slices + green_slices + yellow_slices +
            ↪  red_slices)
68          answer.append(blue_slices + green_slices + yellow_slices + red_slices)
69
70      return sorted(answer)
```

Задача IV.2. Поиск кратчайшего маршрута (40 баллов)

Темы: компьютерное зрение, детектирование по цвету, бинаризация, контурный анализ.

Условие

На изображении представлена схема дорожной разметки города. Синей точкой обозначено стартовое положение, красной точкой обозначен пункт назначения. Ваша задача — написать программу, выводящую номера перекрёстков в порядке их появления на кратчайшем пути от синей точки до красной.

Схема дорожной разметки не изменяется от изображения к изображению, изменяются стартовое положение и пункт назначения. Дорога делится на две полосы прерывистой линией разметки. Движение правостороннее. Пересечение прерывистой полосы запрещено. Преодолевать перекрёстки можно тремя способами: прямо, направо, налево. Перекрёстки нумеруются с 1, сверху вниз и слева направо. Точки начала и конца пути не могут быть на перекрёстке.

Для вас подготовлены несколько файлов «`.py`», набор изображений и аннотации к нему. Среди подготовленных файлов есть:

- `eval.py` — файл с функцией поиска кратчайшего пути. Именно эту функцию вам необходимо дописать!
- `main.py` — файл проверяющий точность работы вашего алгоритма. Не редактируйте его. `main.py` использует, написанные вами, функции из `eval.py` и сверяет истинные метки с предсказанием вашего алгоритма.
- `annotations.csv` — файл, устанавливающий соотношение между изображениями и кратчайшим путем для него. В каждой строке файла содержится путь к файлу и список перекрёстков, составляющих кратчайший путь.

В качестве решения необходимо сдать отредактированный файл `eval.py`. Либо архив «`*.zip`» с файлом `eval.py` и остальными файлами, требующимися для его работы.

Технические ограничения

Размер решения ограничен: не более 5 Мбайт. Если ваш алгоритм успешно проверен платформой, то следующее решение можно прислать только через 10 мин. Если ваш алгоритм в ходе проверки выдал сообщение об ошибке, то следующее решения можно прислать сразу.

Пакеты, ориентированные на работу с изображениями и данными, использующиеся на онлайн-платформе: `Python 3.8.10`.

`dlib 19.24.0; keras 2.8.0; Keras-Preprocessing 1.1.2; imutils 0.5.4; numpy 1.22.4; opencv-python 4.6.0.66; pandas 1.4.3; scikit-image 0.19.3; scikit-learn 1.1.1; scipy 1.8.1; tensorflow-cpu 2.8.2.`

Используйте совместимые пакеты.

Решение

Для решения данной задачи необходимо познакомиться с теорией графов, алгоритмами их обхода и поиска кратчайшего пути. Схему разметки города удобно представить в виде ориентированного взвешенного графа. Рёбра графа моделируют полосы дорожной разметки, вес ребра соответствует длине полосы движения. Крестообразный перекрёсток моделируется 8-ми вершинами графа и рёбрами, соответствующими маршрутам пересечения перекрёстка. 4 вершины — возможные въезды на перекрёсток, 4 вершины — возможные выезды. 12 рёбер — возможные маршруты пересечения перекрёстка, по три из каждого въезда: прямо, налево, направо. Двухполосная двусторонняя дорога моделируется двумя направленными рёбрами, начинающимися и заканчивающимися между вершинами разных перекрёстков. Можно использовать любое доступное на ЭВМ представление графа, в предлагаемом решении используется отдельный класс, хранящий матрицу смежности: https://ru.wikipedia.org/wiki/Граф_(математика)#Способы_представления_графа_в_информатике.

После того как граф построен, необходимо определить, на каких рёбрах расположены начальная и конечная точки маршрута. Проще всего это сделать, задав соответствие между координатами пикселей на изображении и ребрами графа, но возможны и другие способы. Когда определены начальная вершина маршрута и конечная, остаётся найти кратчайший путь между ними и зафиксировать номера всех вершин, на этом пути. Для поиска кратчайшего пути можно использовать один из множества алгоритмов: https://ru.wikipedia.org/wiki/Задача_о_кратчайшем_пути. Номера вершин пройденного маршрута преобразуются в номера перекрёстков.

Пример программы-решения

Ниже представлено решение на языке Python 3.

```
# -*- coding: utf-8 -*-
import cv2
import numpy as np

class Node:
    def __init__(self, name=None):
        self.name = name
        self.connections = []

    def add_connection(self, node: 'Node', con_len: int):
        self.connections.append((node, con_len))

    def __str__(self):
        return str(self.name)

class Way:
    def __init__(self, start_node: Node):
        self.start_node = start_node
```

```
19          self.nodes = [start_node]
20          self.len = 0
21
22      @property
23      def last_node(self):
24          return self.nodes[-1]
25
26      def add_node(self, node: Node, length: int):
27          self.nodes.append(node)
28          self.len += length
29
30      def copy(self) -> 'Way':
31          new_way = Way(self.start_node)
32          new_way.nodes = self.nodes.copy()
33          new_way.len = self.len
34          return new_way
35
36      def __str__(self):
37          return ' -> '.join(map(str, self.nodes))
38
39  # CROSS #1
40  cross1_r_in = Node(1)
41  cross1_b_in = Node(1)
42  cross1_l_in = Node(1)
43
44  cross1_r_out = Node('cross1_r_out')
45  cross1_b_out = Node('cross1_b_out')
46  cross1_l_out = Node('cross1_l_out')
47
48  cross1_r_in.add_connection(cross1_b_out, 14)
49  cross1_r_in.add_connection(cross1_l_out, 13)
50
51  cross1_l_in.add_connection(cross1_b_out, 9)
52  cross1_l_in.add_connection(cross1_r_out, 13)
53
54  cross1_b_in.add_connection(cross1_r_out, 9)
55  cross1_b_in.add_connection(cross1_l_out, 14)
56
57  # CROSS #2
58  cross2_r_in = Node(2)
59  cross2_b_in = Node(2)
60  cross2_t_in = Node(2)
61
62  cross2_r_out = Node('cross2_r_out')
63  cross2_b_out = Node('cross2_b_out')
64  cross2_t_out = Node('cross2_t_out')
65
66  cross2_r_in.add_connection(cross2_b_out, 14)
67  cross2_r_in.add_connection(cross2_t_out, 9)
68
69  cross2_b_in.add_connection(cross2_t_out, 13)
70  cross2_b_in.add_connection(cross2_r_out, 9)
71
72  cross2_t_in.add_connection(cross2_b_out, 13)
73  cross2_t_in.add_connection(cross2_r_out, 14)
74
75  # CROSS #3
76  cross3_l_in = Node(3)
77  cross3_r_in = Node(3)
78  cross3_b_in = Node(3)
```

```
79  cross3_t_in = Node(3)
80
81  cross3_l_out = Node('cross3_l_out')
82  cross3_r_out = Node('cross3_r_out')
83  cross3_b_out = Node('cross3_b_out')
84  cross3_t_out = Node('cross3_t_out')
85
86  cross3_l_in.add_connection(cross3_r_out, 13)
87  cross3_l_in.add_connection(cross3_t_out, 14)
88  cross3_l_in.add_connection(cross3_b_out, 9)
89
90  cross3_r_in.add_connection(cross3_l_out, 13)
91  cross3_r_in.add_connection(cross3_t_out, 9)
92  cross3_r_in.add_connection(cross3_b_out, 14)
93
94  cross3_b_in.add_connection(cross3_r_out, 9)
95  cross3_b_in.add_connection(cross3_t_out, 13)
96  cross3_b_in.add_connection(cross3_l_out, 14)
97
98  cross3_t_in.add_connection(cross3_r_out, 14)
99  cross3_t_in.add_connection(cross3_l_out, 9)
100 cross3_t_in.add_connection(cross3_b_out, 13)
101
102 # CROSS #4
103 cross4_l_in = Node(4)
104 cross4_b_in = Node(4)
105 cross4_t_in = Node(4)
106
107 cross4_l_out = Node('cross4_l_out')
108 cross4_b_out = Node('cross4_b_out')
109 cross4_t_out = Node('cross4_t_out')
110
111 cross4_l_in.add_connection(cross4_b_out, 9)
112 cross4_l_in.add_connection(cross4_t_out, 14)
113
114 cross4_b_in.add_connection(cross4_t_out, 13)
115 cross4_b_in.add_connection(cross4_l_out, 14)
116
117 cross4_t_in.add_connection(cross4_b_out, 13)
118 cross4_t_in.add_connection(cross4_l_out, 9)
119
120 # CROSS #5
121 cross5_l_in = Node(5)
122 cross5_r_in = Node(5)
123 cross5_t_in = Node(5)
124
125 cross5_l_out = Node('cross5_l_out')
126 cross5_r_out = Node('cross5_r_out')
127 cross5_t_out = Node('cross5_t_out')
128
129 cross5_l_in.add_connection(cross5_r_out, 13)
130 cross5_l_in.add_connection(cross5_t_out, 14)
131
132 cross5_r_in.add_connection(cross5_l_out, 13)
133 cross5_r_in.add_connection(cross5_t_out, 9)
134
135 cross5_t_in.add_connection(cross5_r_out, 14)
136 cross5_t_in.add_connection(cross5_l_out, 9)
137
138 # CONNECTIONS BETWEEN CROSSES
```

```
139 cross1_l_out.add_connection(cross2_t_in, 33)
140 cross1_r_out.add_connection(cross4_t_in, 28)
141 cross1_b_out.add_connection(cross3_t_in, 10)
142
143 cross2_t_out.add_connection(cross1_l_in, 28)
144 cross2_r_out.add_connection(cross3_l_in, 12)
145 cross2_b_out.add_connection(cross5_l_in, 33)
146
147 cross3_l_out.add_connection(cross2_r_in, 12)
148 cross3_r_out.add_connection(cross4_l_in, 12)
149 cross3_b_out.add_connection(cross5_t_in, 10)
150 cross3_t_out.add_connection(cross1_b_in, 10)
151
152 cross4_l_out.add_connection(cross3_r_in, 12)
153 cross4_t_out.add_connection(cross1_r_in, 33)
154 cross4_b_out.add_connection(cross5_r_in, 28)
155
156 cross5_l_out.add_connection(cross2_b_in, 28)
157 cross5_r_out.add_connection(cross4_b_in, 33)
158 cross5_t_out.add_connection(cross3_b_in, 10)
159
160 road_radius = 27
161
162 vertical_roads = {
163     'top_left': [44, 106, 215],
164     'top_middle': [307, 106, 215],
165     'top_right': [570, 106, 215],
166     'bottom_left': [44, 350, 459],
167     'bottom_middle': [307, 350, 459],
168     'bottom_right': [570, 350, 459]
169 }
170
171 horizontal_roads = {
172     'left_top': [39, 111, 240],
173     'left_middle': [283, 111, 240],
174     'left_bottom': [526, 111, 240],
175     'right_top': [39, 375, 503],
176     'right_middle': [283, 375, 503],
177     'right_bottom': [526, 375, 503]
178 }
179
180
181 def get_start_point_coordinates(img):
182     blue = cv2.inRange(img, (50, 0, 0), (255, 0, 0))
183
184     x = np.nonzero(np.argmax(blue, axis=0))[0]
185     y = np.nonzero(np.argmax(blue, axis=1))[0]
186     start_x = int(sum(x) / len(x))
187     start_y = int(sum(y) / len(y))
188
189     return start_x, start_y
190
191
192 def get_end_point_coordinates(img):
193     red = cv2.inRange(img, (0, 0, 50), (0, 0, 255))
194
195     x = np.nonzero(np.argmax(red, axis=0))[0]
196     y = np.nonzero(np.argmax(red, axis=1))[0]
197     end_x = int(sum(x) / len(x))
198     end_y = int(sum(y) / len(y))
```

```
199
200     return end_x, end_y
201
202
203 def check_on_the_same_road(start_road_name, end_road_name, start_direction,
    ↪  end_direction, start_x, start_y,
204                             end_x, end_y, start_road_type, end_road_type):
205     # check if on the same road pointing the same direction
206     if start_road_name == end_road_name and start_direction == end_direction:
207         if start_road_type == 'vertical':
208             if start_direction == 'up':
209                 if start_y > end_y:  # start point should be downer
210                     print('No need to move to cross')
211                     return True
212             else:
213                 if start_y < end_y:  # start point should be upper
214                     print('No need to move to cross')
215                     return True
216         else:
217             if start_direction == 'right':
218                 if start_x < end_x:
219                     print('No need to move to cross')
220                     return True
221             else:
222                 if start_x > end_x:
223                     print('No need to move to cross')
224                     return True
225
226     # check if points on the turn
227     conditions = []
228     # TOP LEFT TURN
229     conditions.append(all([start_road_name == 'left_top',
230                            start_direction == 'left',
231                            end_road_name == 'top_left',
232                            end_direction == 'down']))
233     conditions.append(all([start_road_name == 'top_left',
234                            start_direction == 'up',
235                            end_road_name == 'left_top',
236                            end_direction == 'right']))
237
238     # TOP RIGHT TURN
239     conditions.append(all([start_road_name == 'right_top',
240                            start_direction == 'right',
241                            end_road_name == 'top_right',
242                            end_direction == 'down']))
243     conditions.append(all([start_road_name == 'top_right',
244                            start_direction == 'up',
245                            end_road_name == 'right_top',
246                            end_direction == 'left']))
247
248     # BOTTOM LEFT TURN
249     conditions.append(all([start_road_name == 'left_bottom',
250                            start_direction == 'left',
251                            end_road_name == 'bottom_left',
252                            end_direction == 'up']))
253     conditions.append(all([start_road_name == 'bottom_left',
254                            start_direction == 'down',
255                            end_road_name == 'left_bottom',
256                            end_direction == 'right']))
257
```

```
258         # BOTTOM RIGHT TURN
259         conditions.append(all([start_road_name == 'right_bottom',
260                                start_direction == 'right',
261                                end_road_name == 'bottom_right',
262                                end_direction == 'up']))
263         conditions.append(all([start_road_name == 'bottom_right',
264                                start_direction == 'down',
265                                end_road_name == 'right_bottom',
266                                end_direction == 'left']))
267 
268         if any(conditions):
269             print('No need for CROSS!')
270             return True
271 
272         return False
273 
274 
275     def find_the_shortest_way(image) -> list:
276         """
277         Функция для нахождения кратчайшего маршрута из точки A (синяя точка) в точку B
    ↪    (красная точка).
278 
279         Входные данные: изображение (bgr), прочитано cv2.imread
280         Выходные данные: список из пройденных перекрестков:
281             [1, 2, 3, 4, 5], где 1, 2, 3, 4, 5 - перекрёстки, которые необходимо
    ↪    преодолеть
282             Всего есть 5 перекрёстков, которые можно проехать.
283 
284         Примеры вывода:
285             [4, 5, 3] - проехать через перекрестки 4, 5, 3
286 
287             [4] - преодолеть перекресток 4
288 
289             [] - перекрёстки пересекать не требуется
290         """
291         # Алгоритм проверки будет вызывать функцию find_the_shortest_way,
292         # остальные функции должны вызываться из неё.
293 
294         start_x, start_y = get_start_point_coordinates(image)
295         end_x, end_y = get_end_point_coordinates(image)
296 
297         cv2.circle(image, (start_x, start_y), 20, (0, 255, 0), 1)
298         cv2.circle(image, (end_x, end_y), 20, (0, 255, 0), 1)
299 
300         print(start_x, start_y)
301         print(end_x, end_y)
302 
303         for road_name, (road_x, road_start, road_end) in vertical_roads.items():
304             # cv2.rectangle(img, (road_x-road_radius, road_start),
            ↪  (road_x+road_radius, road_end), (255, 0, 0), 2)
305 
306             if road_x - road_radius <= start_x <= road_x + road_radius and road_start
            ↪  <= start_y <= road_end:
307                 start_road_type = 'vertical'
308                 start_road_name = road_name
309                 if start_x > road_x:
310                     start_direction = 'up'
311                 else:
312                     start_direction = 'down'
313 
```

```
314             if road_x - road_radius <= end_x <= road_x + road_radius and road_start <=
                ↪  end_y <= road_end:
315                 end_road_type = 'vertical'
316                 end_road_name = road_name
317                 if end_x > road_x:
318                     end_direction = 'up'
319                 else:
320                     end_direction = 'down'
321 
322             # cv2.imshow('img', img)
323             # cv2.waitKey(0)
324 
325         for road_name, (road_y, road_start, road_end) in horizontal_roads.items():
326             # cv2.rectangle(img, (road_start, road_y-road_radius), (road_end,
                ↪  road_y+road_radius), (255, 0, 0), 2)
327 
328             if road_y - road_radius <= start_y <= road_y + road_radius and road_start
                ↪  <= start_x <= road_end:
329                 start_road_type = 'horizontal'
330                 start_road_name = road_name
331                 if start_y > road_y:
332                     start_direction = 'right'
333                 else:
334                     start_direction = 'left'
335 
336             if road_y - road_radius <= end_y <= road_y + road_radius and road_start <=
                ↪  end_x <= road_end:
337                 end_road_type = 'horizontal'
338                 end_road_name = road_name
339                 if end_y > road_y:
340                     end_direction = 'right'
341                 else:
342                     end_direction = 'left'
343 
344         print(
345             f"[START] Start point {start_x, start_y} is on {start_road_type}
                ↪  {start_road_name} road pointing {start_direction}")
346         print(f"[END] End point {end_x, end_y} is on {end_road_type} {end_road_name}
            ↪  road pointing {end_direction}")
347 
348         if check_on_the_same_road(start_road_name, end_road_name, start_direction,
            ↪  end_direction, start_x, start_y, end_x,
349                                   end_y, start_road_type, end_road_type):
350             return []
351 
352         # CREATE START NODE
353         start_node = Node("START")
354         # vertical roads
355         if start_road_name == 'top_left':
356             if start_direction == 'up':
357                 start_node.add_connection(cross1_l_in, 23)
358             else:
359                 start_node.add_connection(cross2_t_in, 5)
360 
361         elif start_road_name == 'top_middle':
362             if start_direction == 'up':
363                 start_node.add_connection(cross1_b_in, 5)
364             else:
365                 start_node.add_connection(cross3_t_in, 5)
366 
```

```
367         elif start_road_name == 'top_right':
368             if start_direction == 'up':
369                 start_node.add_connection(cross1_r_in, 28)
370             else:
371                 start_node.add_connection(cross4_t_in, 5)
372 
373         elif start_road_name == 'bottom_left':
374             if start_direction == 'up':
375                 start_node.add_connection(cross2_b_in, 5)
376             else:
377                 start_node.add_connection(cross5_l_in, 28)
378 
379         elif start_road_name == 'bottom_middle':
380             if start_direction == 'up':
381                 start_node.add_connection(cross3_b_in, 5)
382             else:
383                 start_node.add_connection(cross5_t_in, 5)
384 
385         elif start_road_name == 'bottom_right':
386             if start_direction == 'up':
387                 start_node.add_connection(cross4_b_in, 5)
388             else:
389                 start_node.add_connection(cross5_r_in, 23)
390 
391         # horizontal roads
392         elif start_road_name == 'left_top':
393             if start_direction == 'right':
394                 start_node.add_connection(cross1_l_in, 5)
395             else:
396                 start_node.add_connection(cross2_t_in, 28)
397 
398         elif start_road_name == 'left_middle':
399             if start_direction == 'right':
400                 start_node.add_connection(cross3_l_in, 5)
401             else:
402                 start_node.add_connection(cross2_r_in, 5)
403 
404         elif start_road_name == 'left_bottom':
405             if start_direction == 'right':
406                 start_node.add_connection(cross5_l_in, 5)
407             else:
408                 start_node.add_connection(cross2_b_in, 23)
409 
410         elif start_road_name == 'right_top':
411             if start_direction == 'right':
412                 start_node.add_connection(cross4_t_in, 23)
413             else:
414                 start_node.add_connection(cross1_r_in, 5)
415 
416         elif start_road_name == 'right_middle':
417             if start_direction == 'right':
418                 start_node.add_connection(cross4_l_in, 5)
419             else:
420                 start_node.add_connection(cross3_r_in, 5)
421 
422         elif start_road_name == 'right_bottom':
423             if start_direction == 'right':
424                 start_node.add_connection(cross4_b_in, 28)
425             else:
426                 start_node.add_connection(cross5_r_in, 5)
```

```
427
428         # CREATE END NODE
429         end_node = Node("END")
430         # VERTICAL
431         if end_road_name == 'top_left':
432             if end_direction == 'up':
433                 cross2_t_out.add_connection(end_node, 5)
434             else:
435                 cross1_l_out.add_connection(end_node, 28)
436
437         elif end_road_name == 'top_middle':
438             if end_direction == 'up':
439                 cross3_t_out.add_connection(end_node, 5)
440             else:
441                 cross1_b_out.add_connection(end_node, 5)
442
443         elif end_road_name == 'top_right':
444             if end_direction == 'up':
445                 cross4_t_out.add_connection(end_node, 5)
446             else:
447                 cross1_r_out.add_connection(end_node, 23)
448
449         elif end_road_name == 'bottom_left':
450             if end_direction == 'down':
451                 cross2_b_out.add_connection(end_node, 5)
452             else:
453                 cross5_l_out.add_connection(end_node, 23)
454
455         elif end_road_name == 'bottom_middle':
456             if end_direction == 'down':
457                 cross3_b_out.add_connection(end_node, 5)
458             else:
459                 cross5_t_out.add_connection(end_node, 5)
460
461         elif end_road_name == 'bottom_right':
462             if end_direction == 'down':
463                 cross4_b_out.add_connection(end_node, 5)
464             else:
465                 cross5_r_out.add_connection(end_node, 28)
466
467         # HORIZONTAL
468         elif end_road_name == 'left_top':
469             if end_direction == 'right':
470                 cross2_t_out.add_connection(end_node, 23)
471             else:
472                 cross1_l_out.add_connection(end_node, 5)
473
474         elif end_road_name == 'left_middle':
475             if end_direction == 'right':
476                 cross2_r_out.add_connection(end_node, 5)
477             else:
478                 cross3_l_out.add_connection(end_node, 5)
479
480         elif end_road_name == 'left_bottom':
481             if end_direction == 'right':
482                 cross2_b_out.add_connection(end_node, 28)
483             else:
484                 cross5_l_out.add_connection(end_node, 5)
485
486         elif end_road_name == 'right_top':
```

```
487             if end_direction == 'right':
488                 cross1_r_out.add_connection(end_node, 5)
489             else:
490                 cross4_t_out.add_connection(end_node, 28)
491 
492         elif end_road_name == 'right_middle':
493             if end_direction == 'right':
494                 cross3_r_out.add_connection(end_node, 5)
495             else:
496                 cross4_l_out.add_connection(end_node, 5)
497 
498         elif end_road_name == 'right_bottom':
499             if end_direction == 'right':
500                 cross5_r_out.add_connection(end_node, 5)
501             else:
502                 cross4_b_out.add_connection(end_node, 23)
503 
504         ways = [Way(start_node)]
505         best_way = None
506         best_way_len = 0
507 
508         run = True
509         while run:
510             new_ways = []
511             shorter_ways = False
512             shorter_ways_lst = []
513             for way in ways:
514                 for next_node, length in way.last_node.connections:
515                     if next_node == end_node:
516                         if best_way:
517                             way.add_node(next_node, length)
518                             if way.len == best_way.len:
519                                 print('EQUAL WAYS!!!!')
520                                 print("[best_way]", best_way.len, best_way)
521                                 print("[cur_way]", way.len, way)
522                                 open('equal_ways.txt', 'w').close()
523                             if way.len < best_way.len:
524                                 best_way = way
525                                 best_way_len = way.len
526                                 print("[New BEST!]", way.len, way)
527                         else:
528                             way.add_node(next_node, length)
529                             best_way = way
530                             best_way_len = way.len
531                             shorter_ways = True
532                         print("[Reach the End!]", way.len, way)
533                     else:
534                         way_copy = way.copy()
535                         way_copy.add_node(next_node, length)
536                         new_ways.append(way_copy)
537                         if best_way:
538                             if way_copy.len < best_way_len:
539                                 shorter_ways = True
540                                 shorter_ways_lst.append(way)
541 
542             if best_way:
543                 print("[BEST NODE]", best_way.len, best_way)
544                 print('shorter ways =', shorter_ways)
545                 for way in shorter_ways_lst:
546                     print(way.len, way)
```

```
547             print("-" * 60)
548
549         if best_way and not shorter_ways:
550             break
551         ways = new_ways.copy()
552
553     print(best_way, best_way.len)
554     ans = []
555     for node in best_way.nodes:
556         if node.name in (1, 2, 3, 4, 5):
557             ans.append(node.name)
558
559     return ans
```

Задача IV.3. Детектирование знаков дорожного движения (30 баллов)

Темы: компьютерное зрение, детектирование по цвету, бинаризация, контурный анализ.

Условие

На изображении представлены знаки дорожного движения. Ваша задача написать функцию, определяющую, где на изображении расположен знак дорожного движения и как он называется. Возможных вариантов знаков пять: Road works, Parking, No entry, Pedestrian crossing, Movement prohibition, Artificial roughness, Give way, Stop.

Для вас подготовлены несколько файлов «`.py`», набор изображений знаков и аннотации к нему. Среди подготовленных файлов есть:

- `eval.py` — файл с ключевыми функциями. Именно эти функции вам необходимо дописать!
- `main.py` — файл проверяющий точность работы вашего алгоритма. Не редактируйте его. `main.py` использует, написанные вами функции из `eval.py` и сверяет истинные метки с предсказанием вашего алгоритма.
- `annotations.csv` — файл, устанавливающий соотношение между изображениями, координатами знаков и их названиями. В каждой строке файла содержится путь к файлу с изображением, название знаков и координаты ограничивающих знак рамок.

В качестве решения, необходимо сдать отредактированный файл `eval.py`. Либо архив «`*.zip`» с файлом `eval.py` и остальными файлами, требующимися для его работы.

Технические ограничения

Размер решения ограничен: не более 30 Мбайт. Если ваш алгоритм успешно проверен платформой, то следующее решение можно прислать только через 10 мин. Если ваш алгоритм в ходе проверки выдал сообщение об ошибке, то следующее решения можно прислать сразу.

Пакеты, ориентированные на работу с изображениями и данными, использующиеся на платформе проверки: `Python 3.8.10`.

`dlib 19.24.0; keras 2.8.0; Keras-Preprocessing 1.1.2; imutils 0.5.4; numpy 1.22.4; opencv-python 4.6.0.66; pandas 1.4.3; scikit-image 0.19.3; scikit-learn 1.1.1; scipy 1.8.1; tensorflow-cpu 2.8.2`. Используйте совместимые пакеты.

Решение

Учитывая многообразие объектов, которые необходимо детектировать, неоднородность и разнообразие фонов, для решения предпочтительно использовать нейросетевой детектор. Один из детекторов, подходящих для решения задачи нейросеть `YOLO4-tiny`, её размер меньше 30 Мбайт. В представленном решении используется `YOLO4-tiny`.

Предоставленный датасет следует использовать как тренировочный для обучения детектора. Процесс обучения необходимо проводить в соответствии с документацией в авторском репозитории: https://github.com/AlexeyAB/darknet. Гиперпараметры и настройки процесса обучения необходимо подбирать экспериментально, ориентируясь на точность получаемой модели.

Код инференса нейросетевого детектора и интерпретация выходных данных приведены ниже. Возможно использование других детекторов, средств обучения и инференса.

Пример программы-решения

Ниже представлено решение на языке Python 3.

```
# -*- coding: utf-8 -*-
import cv2

all_classes = ['Road works', 'Parking', 'No entry', 'Pedestrian crossing',
    'Movement prohibition',
                'Artificial roughness', 'Give way', 'Stop']
CONFIDENCE_THRESHOLD = 0.4
NMS_THRESHOLD = 0.2

def load_model():
    """
    Функция осуществляет загрузку модели(ей) нейросети(ей) из файла(ов).
    Выходные параметры: загруженный(е) модели(и)

    Если вы не собираетесь использовать эту функцию, пусть возвращает пустой
    список []
    Если вы используете несколько моделей, возвращайте их список [model1, model2]

    То, что вы вернёте из этой функции, будет передано вторым аргументом в функцию
    predict_box()
    """

    # Модель нейронной сети, загрузите вместе с решением.
    # Если вы не собираетесь использовать эту функцию, пусть возвращает пустой
    список []

    net = cv2.dnn.readNetFromDarknet('yolo.cfg', 'yolo.weights')
    model = cv2.dnn_DetectionModel(net)
    model.setInputParams(scale=1/255, size=(416, 416), swapRB=True)
```

```python
    return model

def detect_road_signs(image, model) -> list:
    """
    Функция для детектирования знаков дорожного движения.

    Входные данные: изображение (bgr), список моделей
    Выходные данные: список из обнаруженных знаков дорожного движения в формате:
        [[label, (x1, y1, x2, y2)]],
            где label - название дорожного знака
            (x1, y1) - координаты верхнего левого угла рамки, ограничивающей знак
            (x2, y2) - координаты нижнего правого угла рамки, ограничивающей знак

        если знаков несколько, то в следующем виде:
            [[label1, (x1, y1, x2, y2)], [label2, (x1, y1, x2, y2)]]

    Примеры вывода:
        [["Road works", (34, 54, 130, 127)], ["Parking", (204, 323, 240, 350)]] - для 2-х знаков

        [["Give way", (30, 45, 105, 93)]] - для одного знака

        [] - если знаки не обнаружены
    """
    # Алгоритм проверки будет вызывать только функции load_model() и detect_road_signs().
    # Остальные функции должны вызываться из вышеперечисленых.

    result = []
    classes, scores, boxes = model.detect(image, CONFIDENCE_THRESHOLD, NMS_THRESHOLD)
    for cls, score, box in zip(classes, scores, boxes):
        if cls == 0:
            continue
        x, y, w, h = box
        result.append([all_classes[cls-1], (x, y, x+w, y+h)])

    return result
```

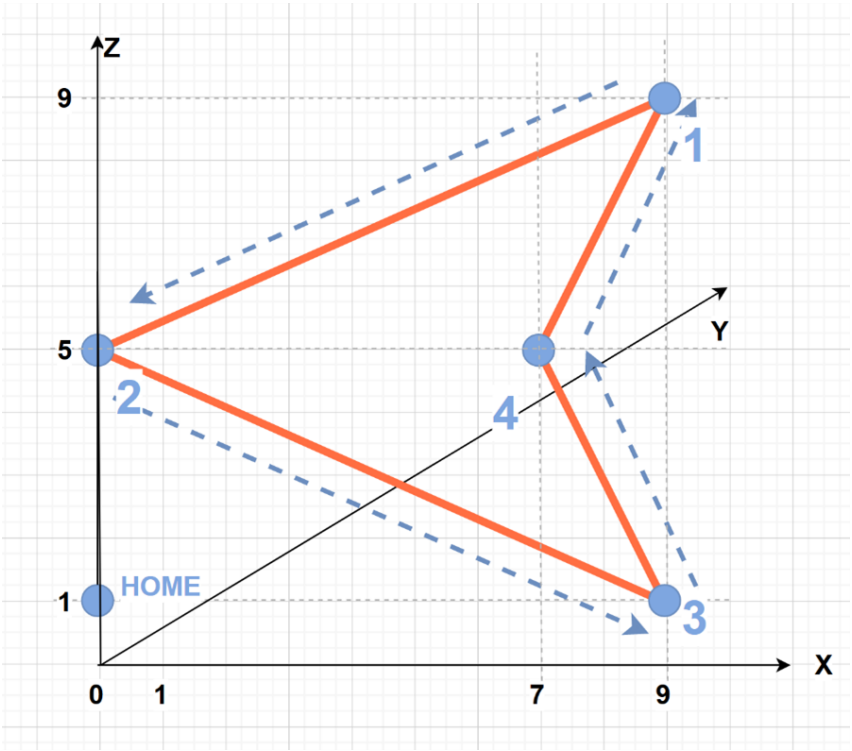
Задача IV.4. Автономный полёт квадрокоптера (15 баллов)

Темы: ROS, отладка программ на удалённом устройстве, прямоугольная система координат, полётная программа квадрокоптера.

Условие

Необходимо создать полетное задание, в котором коптер взлетает и перемещается в заданную точку HOME, далее перемещается на точку №1 траектории, пролетает заданную траекторию в указанном направлении, возвращается на точку HOME и совершает посадку.

Полетное задание (траектория полета) представляет собой массив точек в переменной `coordinates`, по которым последовательно летит коптер. Точка взлета и посадки (точка HOME) находится в переменной `home_point`. Вам необходимо отредактировать значения в обеих переменных в соответствии с верными координатами, представленными на изображении ниже.



Полет совершается в вертикальной плоскости. Изменяются координаты по осям X и Z. Координата Y неизменна, $Y = 0$ всегда.

Для вашего удобства на рисунке нанесены номера точек и соответствующие им координаты на координатной сетке.

Взлет осуществляется коптером самостоятельно, функцией `ap.takeoff()`, посадка – функцией `ap.landing()`.

Будьте внимательны! В алгоритм полета вкрались ошибки, которые вы должны обнаружить и исправить. Также вам нужно вписать в правильное место функцию посадки `ap.landing()`, иначе при проверке результат не зачтется.

Упрощенный вариант корректно работающего полетного скрипта вы можете увидеть здесь: https://github.com/geoscan/gs_example/blob/noetic/src/flight_test.py.

Для вас подготовлен файл `eval.py`. Он содержит шаблон полетного задания с некоторыми ошибками и недоработками. Вам необходимо внимательно ознакомиться с файлом, решить, какие части кода требуют изменений, и отредактировать их.

В качестве решения необходимо сдать на проверку отредактированный файл `eval.py`.

Внимание! Для решения задачи вы вносите изменения только в файл `eval.py`. Никаких дополнительных библиотек, кроме подключенных в коде `eval.py`, использовать нельзя.

Ознакомьтесь с документацией (https://github.com/geoscan/geoscan_pioneer_max) и примерами программирования «Пионер Макс».

Технические ограничения

Если ваш алгоритм успешно проверен платформой, то следующее решение можно прислать только через 10 мин. Если ваш алгоритм в ходе проверки выдал сообщение

об ошибке, то следующее решение можно прислать сразу.

Решение

Для решения задания необходимо ознакомиться с документацией на квадрокоптер и примерами полётных заданий. Особое внимание следует обратить на порядок перечисления координат точки в трёхмерном пространстве и последовательность выполняемых операций.

Пример программы-решения

Ниже представлено решение на языке Python 3.

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

import rospy
from gs_flight import FlightController, CallbackEvent
from gs_board import BoardManager

rospy.init_node("flight_test_node")

home_point = [0, 0, 1]

coordinates = [
    [9,0,9],
    [0,0,5],
    [9,0,1],
    [7,0,5],
    [9,0,9]
]

run = True
position_number = 0

def callback(event):
    global ap
    global run
    global coordinates
    global position_number

    event = event.data
    if event == CallbackEvent.ENGINES_STARTED:
        print("engine started")
        ap.takeoff()
    elif event == CallbackEvent.TAKEOFF_COMPLETE:
        print("takeoff complete")
        ap.goToLocalPoint(home_point[0], home_point[1], home_point[2])
    elif event == CallbackEvent.POINT_REACHED:
        print("point {} reached".format(position_number))
        if position_number < len(coordinates):
            ap.goToLocalPoint(coordinates[position_number][0],
            ↪  coordinates[position_number][1], coordinates[position_number][2])
            position_number += 1
        elif position_number == len(coordinates):
            ap.goToLocalPoint(home_point[0], home_point[1], home_point[2])
        else:
```

```
44              ap.landing()
45      elif event == CallbackEvent.COPTER_LANDED:
46          print("program finished")
47          run = False
48
49  board = BoardManager()
50  ap = FlightController(callback)
51
52  once = False
53
54  while not rospy.is_shutdown() and run:
55      if board.runStatus() and not once:
56          print("start program")
57          ap.preflight()
58          once = True
59      pass
```