

[CV] Rotated IoU 计算旋转矩形之间的重叠面积

[CV] Rotated IoU 计算旋转矩形之间的重叠面积

简介

旋转包围盒的编码方式

矢量的旋转公式

包围盒转化为角点

代码表示

相交区域的特点

点在四边形(矩形)内

点积的物理意义

代码

线段交点

判断线段是否相交

相交后转化为直线交点

代码

计算相交区域面积

顶点排序

顶点排序代码

简易版三角剖分

所有代码

简介

在目标检测的领域，基于Anchor的方法需要对Anchor分配正负样本的标签。通常，对于axis-aligned的anchor和ground truth，可以直接通过 `[top left right down]` 四个值计算他们之间的重叠面积。但是针对于旋转的矩形框，这个问题就变得尤为复杂。

我参考了3D目标检测论文[SECOND](#)的源码，来尝试解释一下如何计算旋转包围盒的重叠面积。

代码全部来自[second.pytorch](#)这个项目的早期版本，去掉了numba/cuda加速的代码。

旋转包围盒的编码方式

作者代码使用了两种方式

1. 通过包围盒中心点位置，尺度以及角度来编码 `rbbox`

```
1 rbbox = [x, y, x_d (w), y_d (h), angle]
```

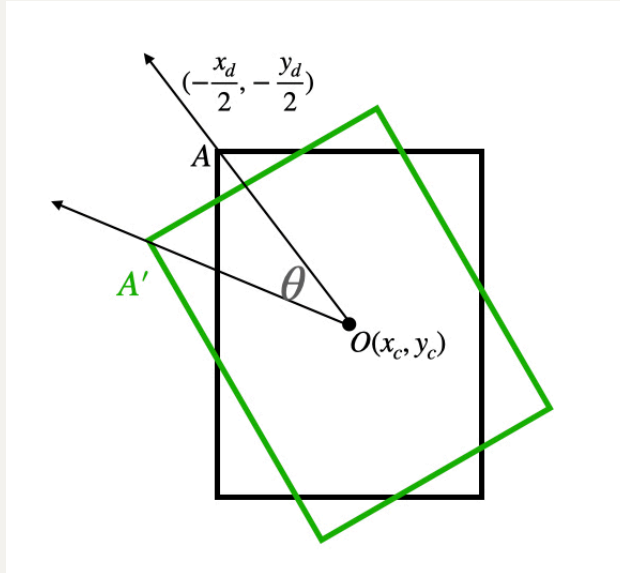
2. 通过包围盒的顶点 `corners` 来编码

矢量的旋转公式

将矢量看作列矢量 $\vec{\alpha} \in \mathbb{R}^{2 \times 1}$ ，则将其逆时针旋转 θ 之后的矢量为：

$$\begin{bmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{bmatrix} \vec{\alpha}$$

包围盒转化为角点



如图，`rbbox`为绿色的包围盒，是原始黑色包围盒通过逆时针旋转 θ 角度得到。分析 A' 的真实坐标：

首先向量 \overrightarrow{OA} 被表示为

$$\overrightarrow{OA} = \left[-\frac{x_d}{2}, -\frac{y_d}{2}\right]^T$$

旋转后的向量可以被表示为

$$\overrightarrow{OA'} = T_\theta \overrightarrow{OA} = \begin{bmatrix} \cos \theta \cdot \frac{-x_d}{2} - \sin \theta \frac{-y_d}{2} \\ \sin \theta \frac{-x_d}{2} + \cos \theta \frac{-y_d}{2} \end{bmatrix}$$

通过 $A' = O + \overrightarrow{OA'}$ 恢复顶点的坐标即可。

代码表示

下段代码将`[x, y, x_d, y_d, angle]`转化为顺时针方向表示的顶点坐标`[x0, y0, x1, y1, x2, y2, x3, y3]`

```
1 import math
2 def rbbox_to_corners(rbbox):
3     # generate clockwise corners and rotate it clockwise
```

```

4      # 顺时针方向返回角点位置
5      cx, cy, x_d, y_d, angle = rbbox
6      a_cos = math.cos(angle)
7      a_sin = math.sin(angle)
8      corners_x = [-x_d / 2, -x_d / 2, x_d / 2, x_d / 2]
9      corners_y = [-y_d / 2, y_d / 2, y_d / 2, -y_d / 2]
10     corners = [0] * 8
11     for i in range(4):
12         corners[2 *
13             i] = a_cos * corners_x[i] + \
14                 a_sin * corners_y[i] + cx
15         corners[2 * i +
16             1] = -a_sin * corners_x[i] + \
17                 a_cos * corners_y[i] + cy
18     return corners

```

测试一下结果：

```

1  rbbox = [0, 0, 2, 4, math.pi / 2]
2  corners = rbbox_to_corners(rbbox)
3  print([round(_) for _ in corners])
4  # [-2, 1, 2, 1, 2, -1, -2, -1]

```

相交区域的特点

两个四边形(矩形)，求交叠面积，可以先求出相交的多边形(Polygon)的顶点，构成多边形的顶点可由两种类型的点构成：

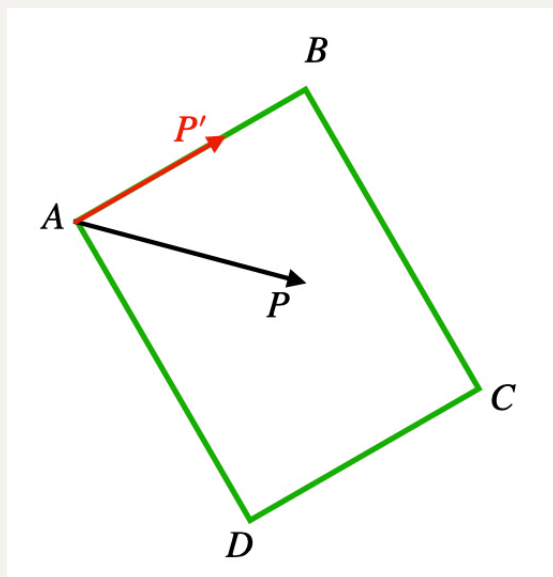
1. 原始四边形的顶点
2. 四边形的边相交产生的交点

对应问题为：

1. 判断点在四边形内
2. 判断线段的交点

点在四边形(矩形)内

如图所示，四边形(矩形)通过 **ABCD** 四个顶点表示，可以使用较强的规则判断 **P** 在矩形内，即 **AP** 在 **AB** 的投影在线段 **AB** 上，在 **AD** 的投影在线段 **AD** 上。



点积的物理意义

两个矢量的点积是标量，点积满足交换律：

1. 矢量的模被定义为 $|a| = \sqrt{a \cdot a}$
2. $a \cdot b = |a||b| \cos \theta$ 实际表示为 b 在 a 上的投影的长度。如果投影与 a 方向相反，则为负值

所以代码的思路就是，通过点积得到投影长度，通过判断投影长度确定点在矩形框内。

代码

SECOND函数为 `point_in_quadrilateral`

```

1  def point_in_quadrilateral(pt_x, pt_y, corners):
2      ab0 = corners[2] - corners[0]
3      ab1 = corners[3] - corners[1]
4
5      ad0 = corners[6] - corners[0]
6      ad1 = corners[7] - corners[1]
7
8      ap0 = pt_x - corners[0]
9      ap1 = pt_y - corners[1]
10
11     abab = ab0 * ab0 + ab1 * ab1
12     abap = ab0 * ap0 + ab1 * ap1
13     adad = ad0 * ad0 + ad1 * ad1
14     adap = ad0 * ap0 + ad1 * ap1
15

```

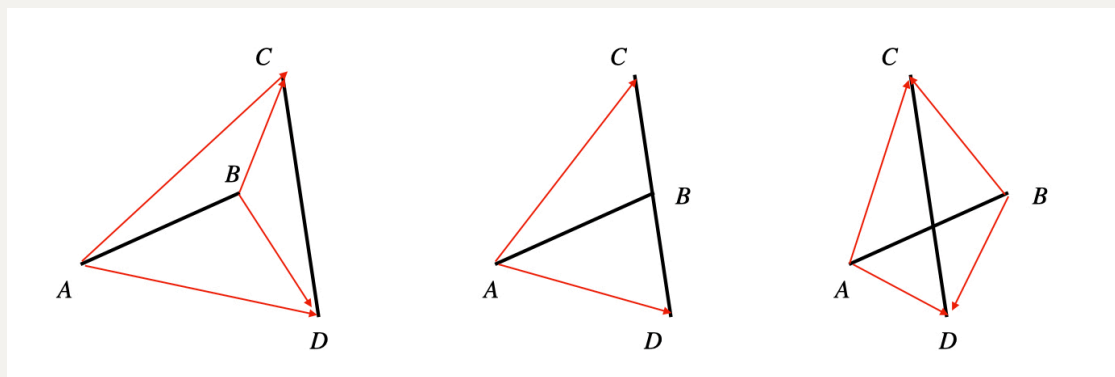
```
16      return abab >= abap and abap >= 0 and adad >= adap and
      adap >= 0
```

线段交点

判断线段是否相交

参考: [判断线段相交的最简方法](#)

对于两个直线是否相交，一种方法是计算线段斜率，首先确定不平行，然后联立方程，计算交点坐标，最后运用定比分点公式，判断交点是否在线段上。但是使用斜率和定比分点，可能会出现精度丢失的现象，同时浮点数运算耗时。



上图可以表示两个线段位置的所有可能情况。定义 $\text{direct}(\mathbf{a}, \mathbf{b})$ 为向量有序对的旋转方向。其旋转方向为使 \mathbf{a} 能够旋转一个小于 180° 的角并与 \mathbf{b} 重合的方向，简记为 $\text{direct}(\mathbf{a}, \mathbf{b})$ 。若 \mathbf{a} 和 \mathbf{b} 反向共线，则旋转方向取任意值。

则上图三种情况可以概括为：

1. $\text{direct}(\mathbf{AC}, \mathbf{AD})$ 和 $\text{direct}(\mathbf{BC}, \mathbf{BD})$ 为顺时针， $\text{direct}(\mathbf{CA}, \mathbf{CB})$ 为逆时针， $\text{direct}(\mathbf{DA}, \mathbf{DB})$ 为顺时针
2. $\text{direct}(\mathbf{AC}, \mathbf{AD})$ 顺时针， $\text{direct}(\mathbf{BC}, \mathbf{BD})$ 为任意方向， $\text{direct}(\mathbf{CA}, \mathbf{CB})$ 为逆时针， $\text{direct}(\mathbf{DA}, \mathbf{DB})$ 为顺时针
3. $\text{direct}(\mathbf{AC}, \mathbf{AD})$ 和 $\text{direct}(\mathbf{DA}, \mathbf{DB})$ 为顺时针， $\text{direct}(\mathbf{BC}, \mathbf{BD})$ 和 $\text{direct}(\mathbf{CA}, \mathbf{CB})$ 为逆时针

可以得知，两条线段相交的充要条件是 $\text{direct}(\mathbf{AC}, \mathbf{AD}) \neq \text{direct}(\mathbf{BC}, \mathbf{BD})$ 和 $\text{direct}(\mathbf{CA}, \mathbf{CB}) \neq \text{direct}(\mathbf{DA}, \mathbf{DB})$

定义 $\angle \vec{a}, \vec{b}$ 为 \vec{a} 逆时针旋转到与 \vec{b} 重合的角度。有：

- $\text{direct}(\mathbf{a}, \mathbf{b})$ 顺时针, $0 \leq \angle \vec{a}, \vec{b} \leq 180, \sin \angle \vec{a}, \vec{b} \geq 0$
- $\text{direct}(\mathbf{a}, \mathbf{b})$ 逆时针, $180 \leq \angle \vec{a}, \vec{b} \leq 360, \sin \angle \vec{a}, \vec{b} \leq 0$

问题可以转化为有向角正弦值的问题。可以使用叉乘来做：

$$\vec{a} \times \vec{b} = a_x \cdot b_y - a_y \cdot b_x$$

叉乘表示 \vec{a}, \vec{b} 构成平行四边形的有向面积

$$|\vec{a} \times \vec{b}| = |\vec{a}| \cdot |\vec{b}| \cdot \sin \theta$$

- 伸出右手，将四指由 \vec{a} 沿小于平角转到 \vec{b} 。若拇指指向纸面上方，则 $\vec{a} \times \vec{b}$ 为正，否则为负。
- 若向量共线，叉积为0

叉积的正负可以判断 $\langle \vec{a}, \vec{b} \rangle$ 的角度范围。所以充要条件等价于叉积符号不同。

相交后转化为直线交点

Wiki [Line-line intersection](#)

Using homogeneous coordinates

直接用wiki的结论，两个点可以确定一条直线，因此定义两条直线 $(x_1, y_1)(x_2, y_2)$ 和 $(x_3, y_3)(x_4, y_4)$ ，可以通过以下公式计算交点的坐标 (P_x, P_y)

$$P_x = \frac{(x_1 y_2 - y_1 x_2)(x_3 - x_4) - (x_1 - x_2)(x_3 y_4 - y_3 x_4)}{(x_1 - x_2)(y_3 - y_4) - (y_1 - y_2)(x_3 - x_4)}$$

$$P_y = \frac{(x_1 y_2 - y_1 x_2)(y_3 - y_4) - (y_1 - y_2)(x_3 y_4 - y_3 x_4)}{(x_1 - x_2)(y_3 - y_4) - (y_1 - y_2)(x_3 - x_4)}$$

代码

代码如下

```
1 def line_segment_intersection(pts1, pts2, i, j):
2     # pts1, pts2 为corners
3     # i j 分别表示第几个交点，取其和其后一个点构成的线段
4     # 返回为 tuple(bool, pts) bool=True pts为交点
5     A, B, C, D, ret = [0, 0], [0, 0], [0, 0], [0, 0], [0, 0]
6     A[0] = pts1[2 * i]
7     A[1] = pts1[2 * i + 1]
8
9     B[0] = pts1[2 * ((i + 1) % 4)]
10    B[1] = pts1[2 * ((i + 1) % 4) + 1]
11
12    C[0] = pts2[2 * j]
13    C[1] = pts2[2 * j + 1]
14
15    D[0] = pts2[2 * ((j + 1) % 4)]
16    D[1] = pts2[2 * ((j + 1) % 4) + 1]
```

```

17     BA0 = B[0] - A[0]
18     BA1 = B[1] - A[1]
19     DA0 = D[0] - A[0]
20     CA0 = C[0] - A[0]
21     DA1 = D[1] - A[1]
22     CA1 = C[1] - A[1]
23     # 叉乘判断方向
24     acd = DA1 * CA0 > CA1 * DA0
25     bcd = (D[1] - B[1]) * (C[0] - B[0]) > (C[1] - B[1]) *
(D[0] - B[0])
26     if acd != bcd:
27         abc = CA1 * BA0 > BA1 * CA0
28         abd = DA1 * BA0 > BA1 * DA0
29         # 判断方向
30         if abc != abd:
31             DC0 = D[0] - C[0]
32             DC1 = D[1] - C[1]
33             ABBA = A[0] * B[1] - B[0] * A[1]
34             CDDC = C[0] * D[1] - D[0] * C[1]
35             DH = BA1 * DC0 - BA0 * DC1
36             Dx = ABBA * DC0 - BA0 * CDDC
37             Dy = ABBA * DC1 - BA1 * CDDC
38             ret[0] = Dx / DH
39             ret[1] = Dy / DH
40             return True, ret
41     return False, ret

```

测试结果:

```

1  if __name__ == '__main__':
2      rbbox1 = [0, 0, 2, 4, 0]
3      rbbox2 = [0, 0, 4, 2, 0]
4      corners1 = rbbox_to_corners(rbbox1)
5      corners2 = rbbox_to_corners(rbbox2)
6      for i in range(4):
7          for j in range(4):
8              ret, pts = line_segment_intersection(corners1,
corners2, i, j)
9              if ret:
10                 print('Px: {}, Py: {}'.format(*pts))
11     """
12     Px: -1.0, Py: 1.0
13     Px: -1.0, Py: -1.0
14     Px: 1.0, Py: 1.0
15     Px: 1.0, Py: -1.0
16     """

```

计算相交区域面积

当有了构成相交区域多边形的顶点后，可以通过以下两部分计算相交区域的面积：

1. 将顶点按照顺时针或者逆时针排序
2. 三角剖分计算面积

顶点排序

在凸多边形内部取一点，与顶点连线，可以通过连线与坐标轴构成的角度排序。操作如下

1. 计算所有顶点的横纵坐标均值，记作中心点
2. 计算中心点到每个单位向量 $[v_x, v_y]$ 。
3. 以x轴正方向为基准，按照顺时针方向扫描360度，对扫描到的点进行排序

关于步骤3，具体操作为：对于已经归一化单位向量，先考虑从180度到360度，有 $v_y \geq 0$ ， $-1 < v_x < 1$ ， v_x 单增。对于从0到180度，有 $v_y < 0$ ， $-1 < v_x < 1$ ，其变化范围是由1到-1(单减)。则排序使用索引k可以为：

1. $v_y > 0, k = v_x$
2. $v_y < 0, k = -2 - v_x$

顶点排序代码

```
1  def sort_vertex_in_convex_polygon(int_pts, num_of_inter):
2      def _cmp(pt, center):
3          vx = pt[0] - center[0]
4          vy = pt[1] - center[1]
5          d = math.sqrt(vx * vx + vy * vy)
6          vx /= d
7          vy /= d
8          if vy < 0:
9              vx = -2 - vx
10         return vx
11
12     if num_of_inter > 0:
13         center = [0, 0]
14         for i in range(num_of_inter):
15             center[0] += int_pts[i][0]
16             center[1] += int_pts[i][1]
17         center[0] /= num_of_inter
18         center[1] /= num_of_inter
19         int_pts.sort(key=lambda x: _cmp(x, center))
```


简易版三角剖分

将多边形转化为多个三角形面积之和，具体操作为固定一个点，按照顺时针顺序依次选择剩下的2个点，计算三角形面积(利用叉积) 最后将三角形面积求和。

代码如下：

```
1 def area(int_pts, num_of_inter):
2     def _triangle_area(a, b, c):
3         return ((a[0] - c[0]) * (b[1] - c[1]) - (a[1] - c[1])
4             *
5             (b[0] - c[0])) / 2.0
6     area_val = 0.0
7     for i in range(num_of_inter - 2):
8         area_val += abs(
9             _triangle_area(int_pts[0], int_pts[i + 1],
10                int_pts[i + 2]))
11     return area_val
```

所有代码

代码汇总如下

```
1 import math
2
3
4 def rbbox_to_corners(rbbox):
5     # generate clockwise corners and rotate it clockwise
6     # 顺时针方向返回角点位置
7     cx, cy, x_d, y_d, angle = rbbox
8     a_cos = math.cos(angle)
9     a_sin = math.sin(angle)
10    corners_x = [-x_d / 2, -x_d / 2, x_d / 2, x_d / 2]
11    corners_y = [-y_d / 2, y_d / 2, y_d / 2, -y_d / 2]
12    corners = [0] * 8
13    for i in range(4):
14        corners[2 *
15            i] = a_cos * corners_x[i] + \
16                a_sin * corners_y[i] + cx
17        corners[2 * i +
18            1] = -a_sin * corners_x[i] + \
```

```

19             a_cos * corners_y[i] + cy
20     return corners
21
22
23 def point_in_quadrilateral(pt_x, pt_y, corners):
24     ab0 = corners[2] - corners[0]
25     ab1 = corners[3] - corners[1]
26
27     ad0 = corners[6] - corners[0]
28     ad1 = corners[7] - corners[1]
29
30     ap0 = pt_x - corners[0]
31     ap1 = pt_y - corners[1]
32
33     abab = ab0 * ab0 + ab1 * ab1
34     abap = ab0 * ap0 + ab1 * ap1
35     adad = ad0 * ad0 + ad1 * ad1
36     adap = ad0 * ap0 + ad1 * ap1
37
38     return abab >= abap and abap >= 0 and adad >= adap and
39     adap >= 0
40
41 def line_segment_intersection(pts1, pts2, i, j):
42     # pts1, pts2 为corners
43     # i j 分别表示第几个交点，取其和其后一个点构成的线段
44     # 返回为 tuple(bool, pts) bool=True pts为交点
45     A, B, C, D, ret = [0, 0], [0, 0], [0, 0], [0, 0], [0, 0]
46     A[0] = pts1[2 * i]
47     A[1] = pts1[2 * i + 1]
48
49     B[0] = pts1[2 * ((i + 1) % 4)]
50     B[1] = pts1[2 * ((i + 1) % 4) + 1]
51
52     C[0] = pts2[2 * j]
53     C[1] = pts2[2 * j + 1]
54
55     D[0] = pts2[2 * ((j + 1) % 4)]
56     D[1] = pts2[2 * ((j + 1) % 4) + 1]
57     BA0 = B[0] - A[0]
58     BA1 = B[1] - A[1]
59     DA0 = D[0] - A[0]
60     CA0 = C[0] - A[0]
61     DA1 = D[1] - A[1]
62     CA1 = C[1] - A[1]
63     # 叉乘判断方向

```

```

64     acd = DA1 * CA0 > CA1 * DA0
65     bcd = (D[1] - B[1]) * (C[0] - B[0]) > (C[1] - B[1]) *
(D[0] - B[0])
66     if acd != bcd:
67         abc = CA1 * BA0 > BA1 * CA0
68         abd = DA1 * BA0 > BA1 * DA0
69         # 判断方向
70         if abc != abd:
71             DC0 = D[0] - C[0]
72             DC1 = D[1] - C[1]
73             ABBA = A[0] * B[1] - B[0] * A[1]
74             CDDC = C[0] * D[1] - D[0] * C[1]
75             DH = BA1 * DC0 - BA0 * DC1
76             Dx = ABBA * DC0 - BA0 * CDDC
77             Dy = ABBA * DC1 - BA1 * CDDC
78             ret[0] = Dx / DH
79             ret[1] = Dy / DH
80             return True, ret
81     return False, ret
82
83
84 def sort_vertex_in_convex_polygon(int_pts, num_of_inter):
85     def _cmp(pt, center):
86         vx = pt[0] - center[0]
87         vy = pt[1] - center[1]
88         d = math.sqrt(vx * vx + vy * vy)
89         vx /= d
90         vy /= d
91         if vy < 0:
92             vx = -2 - vx
93         return vx
94
95     if num_of_inter > 0:
96         center = [0, 0]
97         for i in range(num_of_inter):
98             center[0] += int_pts[i][0]
99             center[1] += int_pts[i][1]
100         center[0] /= num_of_inter
101         center[1] /= num_of_inter
102         int_pts.sort(key=lambda x: _cmp(x, center))
103
104
105 def area(int_pts, num_of_inter):
106     def _triangle_area(a, b, c):
107         return ((a[0] - c[0]) * (b[1] - c[1]) - (a[1] -
c[1]) *

```

```

108         (b[0] - c[0])) / 2.0
109
110     area_val = 0.0
111     for i in range(num_of_inter - 2):
112         area_val += abs(
113             _triangle_area(int_pts[0], int_pts[i + 1],
114                             int_pts[i + 2]))
115     return area_val
116
117
118 if __name__ == '__main__':
119     rbbox1 = [0, 0, 2, 4, 0]
120     rbbox2 = [0, 0, 4, 2, 0]
121     corners1 = rbbox_to_corners(rbbox1)
122     corners2 = rbbox_to_corners(rbbox2)
123     pts, num_pts = [], 0
124     for i in range(4):
125         point = [corners1[2 * i], corners1[2 * i + 1]]
126         if point_in_quadrilateral(point[0], point[1],
127                                   corners2):
128             num_pts += 1
129             pts.append(point)
130     for i in range(4):
131         point = [corners2[2 * i], corners2[2 * i + 1]]
132         if point_in_quadrilateral(point[0], point[1],
133                                   corners1):
134             num_pts += 1
135             pts.append(point)
136     for i in range(4):
137         for j in range(4):
138             ret, point = line_segment_intersection(corners1,
139 corners2, i, j)
140             if ret:
141                 num_pts += 1
142                 pts.append(point)
142     sort_vertex_in_convex_polygon(pts, num_pts)
143     polygon_area = area(pts, num_pts)
144     print('area: {}'.format(polygon_area))

```