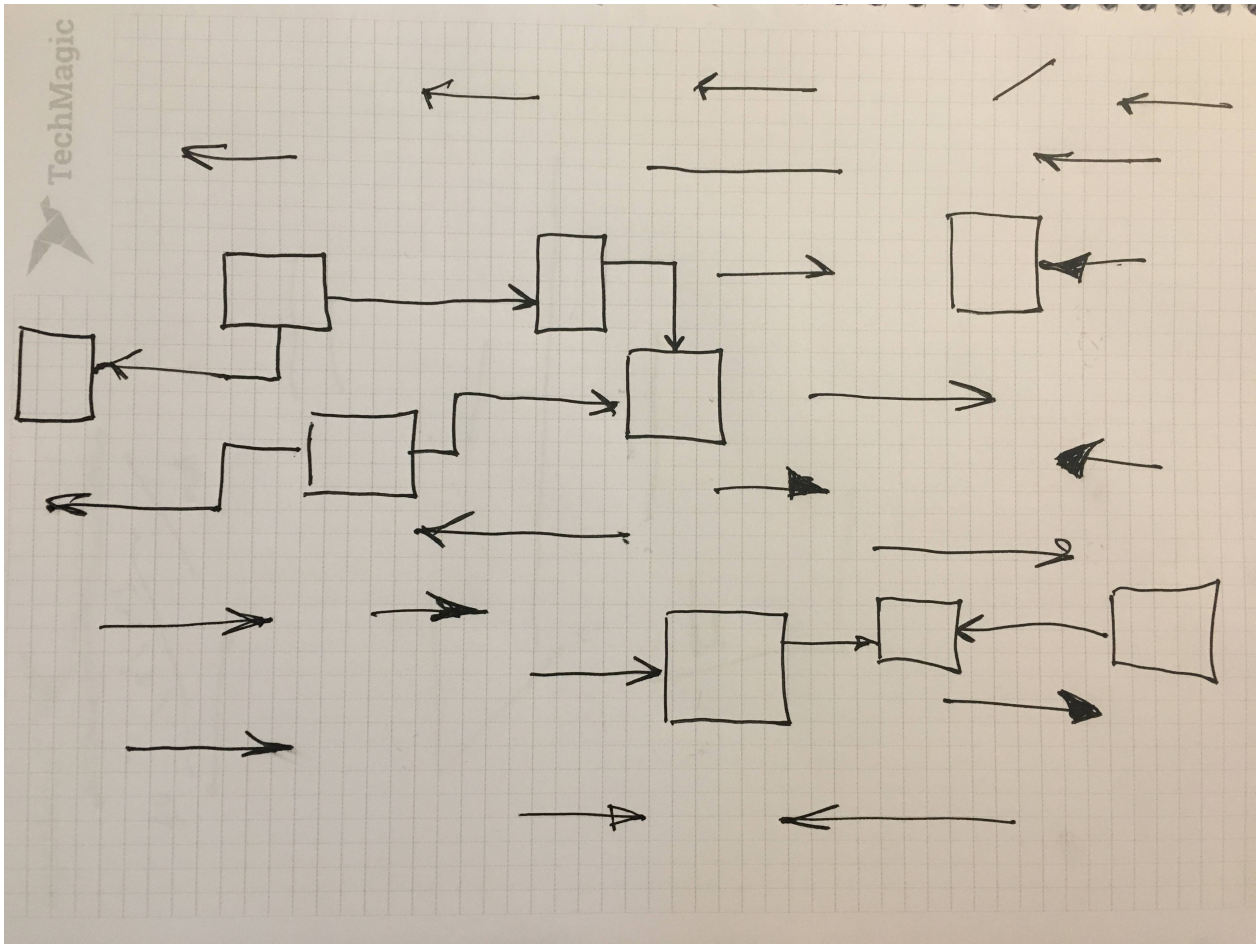
How to merge lines after HoughLinesP?

Asked 3 years, 8 months ago Active 3 months ago Viewed 12k times

13

8

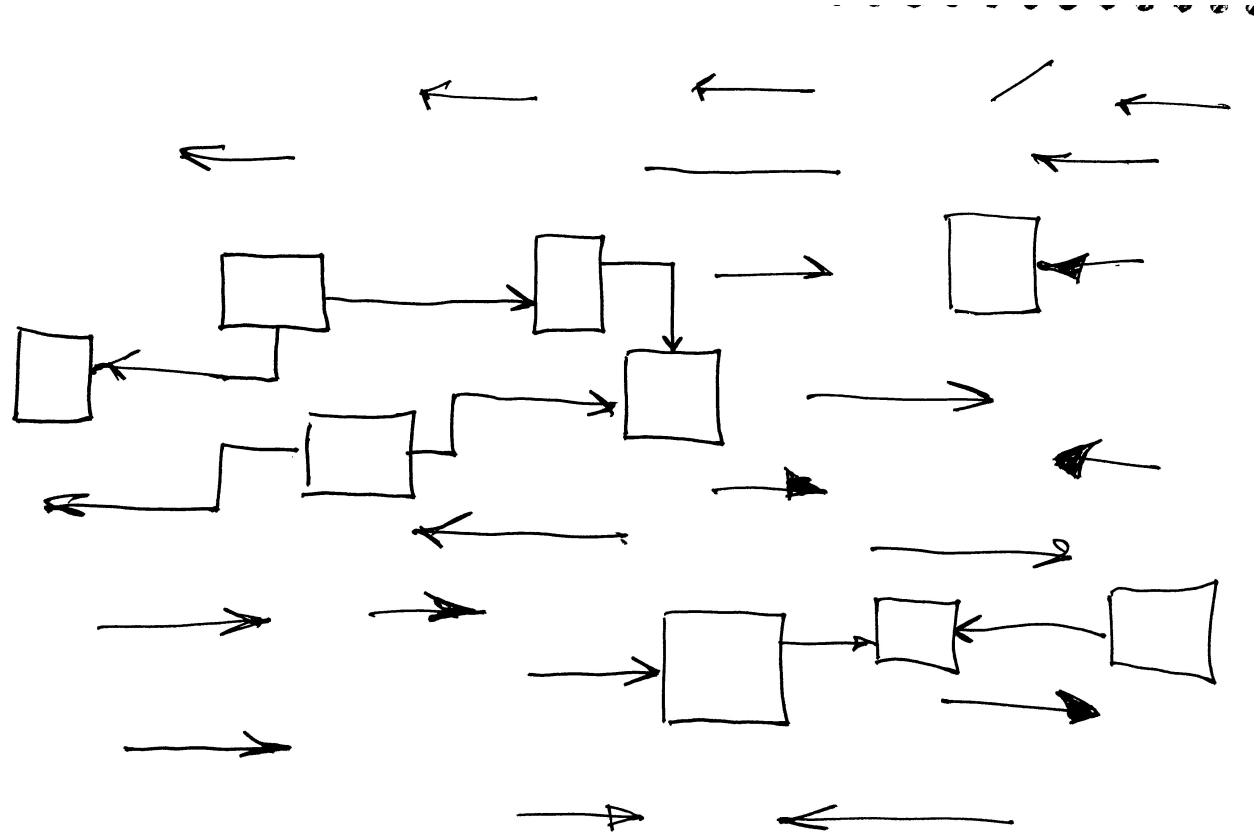
My task is to find coordinates of lines (startX, startY, endX, endY) and rectangles (4 lines). Here is input file:

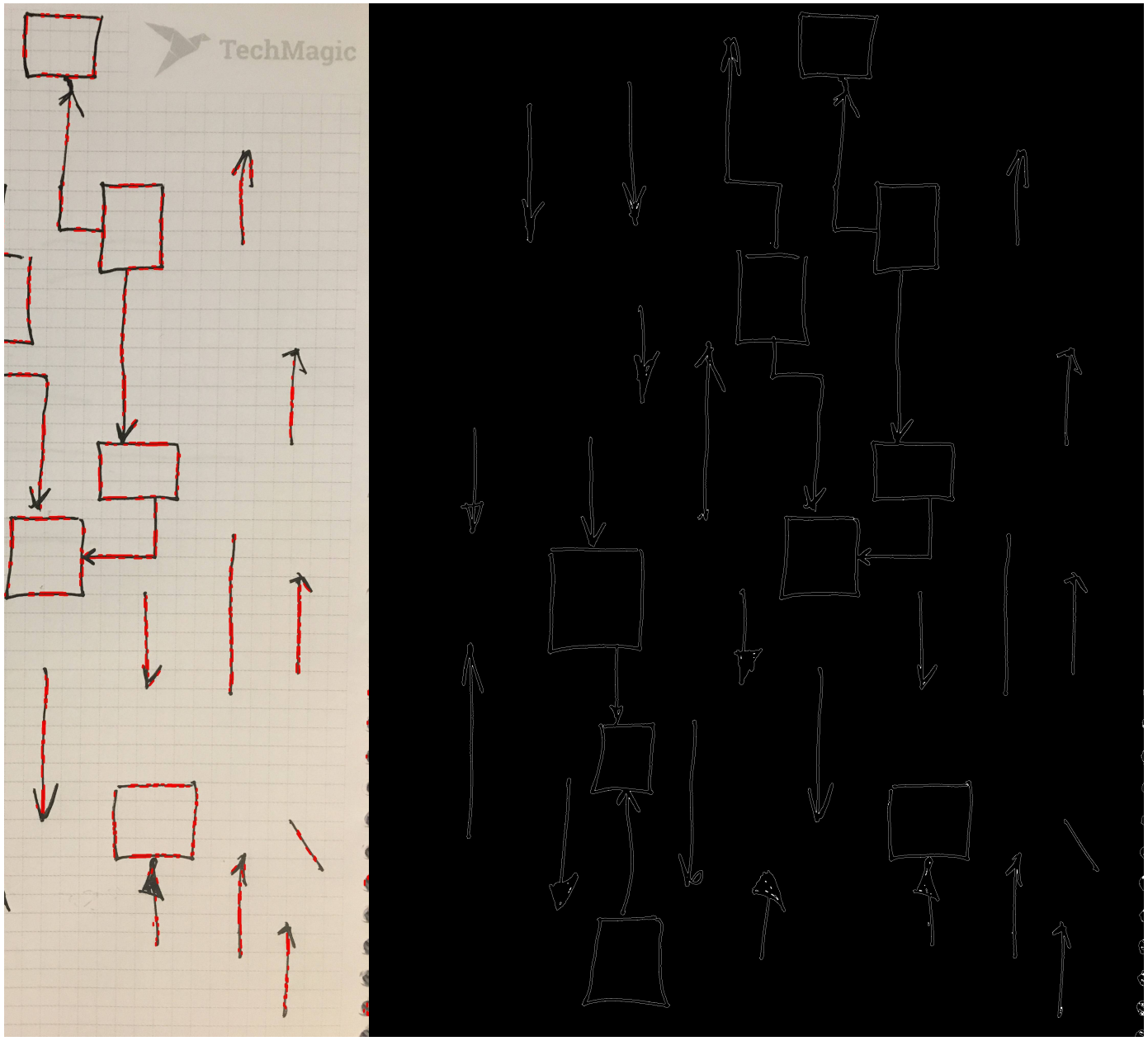


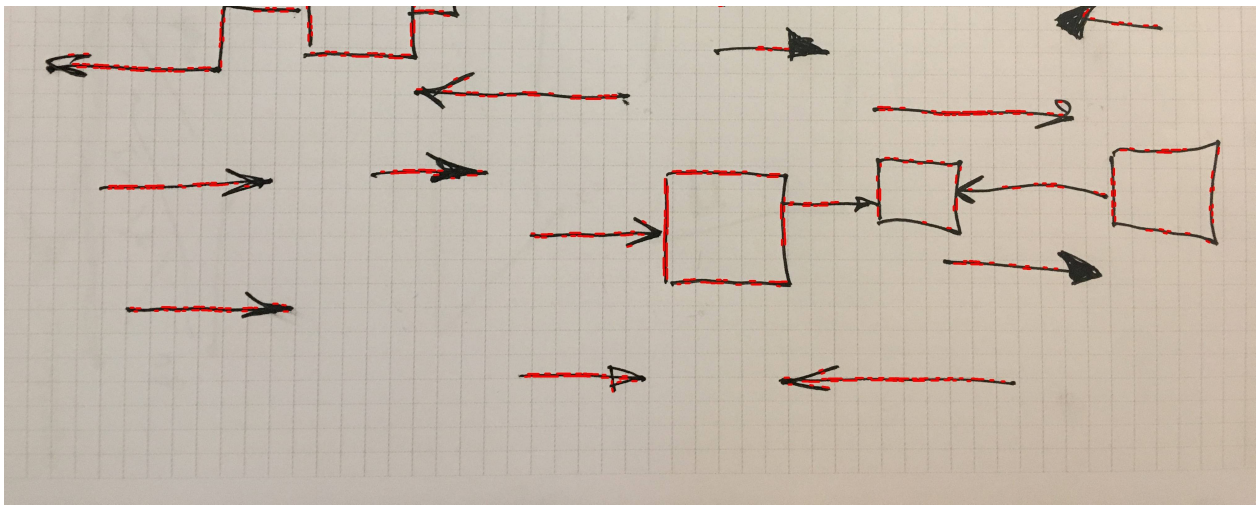
I use the next code:

```
img = cv2.imread(image_src)
gray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
ret, thresh1 = cv2.threshold(gray,127,255,cv2.THRESH_BINARY)

edges = cv2.Canny(thresh1,50,150,apertureSize = 3)

minLineLength = 100
maxLineGap = 10
lines = cv2.HoughLinesP(edges,1,np.pi/180,10,minLineLength,maxLineGap)
print(len(lines))
for line in lines:
    cv2.line(img,(line[0][0],line[0][1]),(line[0][2],line[0][3]),(0,0,255),6)
```

I get the next results:




TechMagic



From the last image you can see big amount of small red lines.

Questions:

1. What is the best way to merge small lines?
2. Why there are a lot of small portions that are not detected by HoughLinesP?

python opencv computer-vision hough-transform cv2

Share Follow

edited Aug 7 '17 at 9:50

asked Aug 6 '17 at 10:31

Oleg Dats
3,307 8 31 60

2 One problem I can see is that you're calling `HoughLinesP` with incorrect parameters. See this answer for explanation. – Dan Mašek Aug 6 '17 at 13:45

1 Google "merging line segments" the first in results list: citeseerx.ist.psu.edu/viewdoc/… – Andrey Smorodov Aug 6 '17 at 14:36

3 You don't need to do Canny edge detection. This will give you the outlines of the arrows instead of the arrows themselves, which you already have. Increase the `rho` parameter a little bit, as it will make the width of an allowable single line larger. When you say you want coordinates of a line, what do you want exactly? The endpoints, or the coordinates of every pixel in a line? – alkasm Aug 6 '17 at 21:02

Thank you guys for your feedbacks. It was really helpful. Dan, you were right. I have fixed parameters. Andrew, it is a great article. I was not able to find execution. Will try to write it manually. Alexander, you were right. I do not need Canny edge detection. I got better results after skipping it. I will try to implement all the improvements and post the answer. – Oleg Dats Aug 7 '17 at 9:46

2 Answers

Active | Oldest | Votes

19

I have finally completed the pipeline:

1. fixed incorrect parameters (as were suggested by Dan)
2. developed my own 'merging line segments' algorithm. I had bad results when I implemented TAVARES and PADILHA algorithm (as were suggested by Andrew).

3. I have skipped Canny and got better results (as were suggested by Alexander)

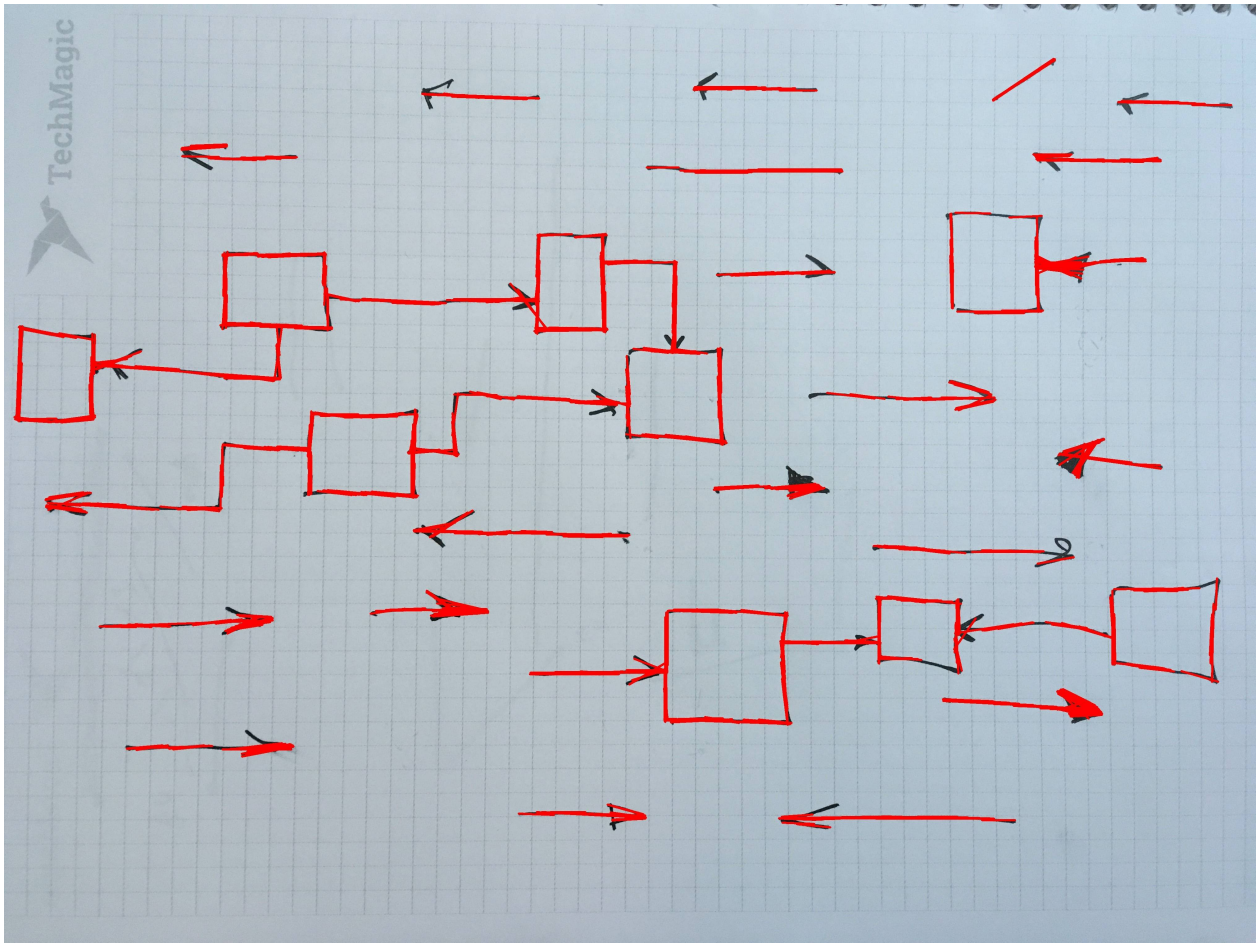
Please find the code and results:

```
def get_lines(lines_in):
    if cv2.__version__ < '3.0':
        return lines_in[0]
    return [l[0] for l in lines_in]


def process_lines(image_src):
    img = mpimg.imread(image_src)
    gray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)

    ret, thresh1 = cv2.threshold(gray,127,255,cv2.THRESH_BINARY)

    thresh1 = cv2.bitwise_not(thresh1)

    edges = cv2.Canny(thresh1, threshold1=50, threshold2=200, apertureSize = 3)

    lines = cv2.HoughLinesP(thresh1, rho=1, theta=np.pi/180, threshold=50,
                            minLineLength=50, maxLineGap=30)

    # l[0] - line; l[1] - angle
    for line in get_lines(lines):
        leftx, boty, rightx, topy = line
        cv2.line(img, (leftx, boty), (rightx,topy), (0,0,255), 6)

    # merge lines

    #------------------
    # prepare
    _lines = []
    for _line in get_lines(lines):
        _lines.append([(_line[0], _line[1]),(_line[2], _line[3])])

    # sort
    _lines_x = []
    _lines_y = []
    for line_i in _lines:
        orientation_i = math.atan2((line_i[0][1]-line_i[1][1]),(line_i[0][0]-line_i[1]
[0]))
        if (abs(math.degrees(orientation_i)) > 45) and abs(math.degrees(orientation_i))
< (90+45):
            _lines_y.append(line_i)
        else:
            _lines_x.append(line_i)

    _lines_x = sorted(_lines_x, key=lambda _line: _line[0][0])
    _lines_y = sorted(_lines_y, key=lambda _line: _line[0][1])

    merged_lines_x = merge_lines_pipeline_2(_lines_x)
    merged_lines_y = merge_lines_pipeline_2(_lines_y)

    merged_lines_all = []
    merged_lines_all.extend(merged_lines_x)
    merged_lines_all.extend(merged_lines_y)
    print("process groups lines", len(_lines), len(merged_lines_all))
    img_merged_lines = mpimg.imread(image_src)
    for line in merged_lines_all:
        cv2.line(img_merged_lines, (line[0][0], line[0][1]), (line[1][0],line[1][1]),
(0,0,255), 6)


    cv2.imwrite('prediction/lines_gray.jpg',gray)
    cv2.imwrite('prediction/lines_thresh.jpg',thresh1)
    cv2.imwrite('prediction/lines_edges.jpg',edges)
    cv2.imwrite('prediction/lines_lines.jpg',img)
```

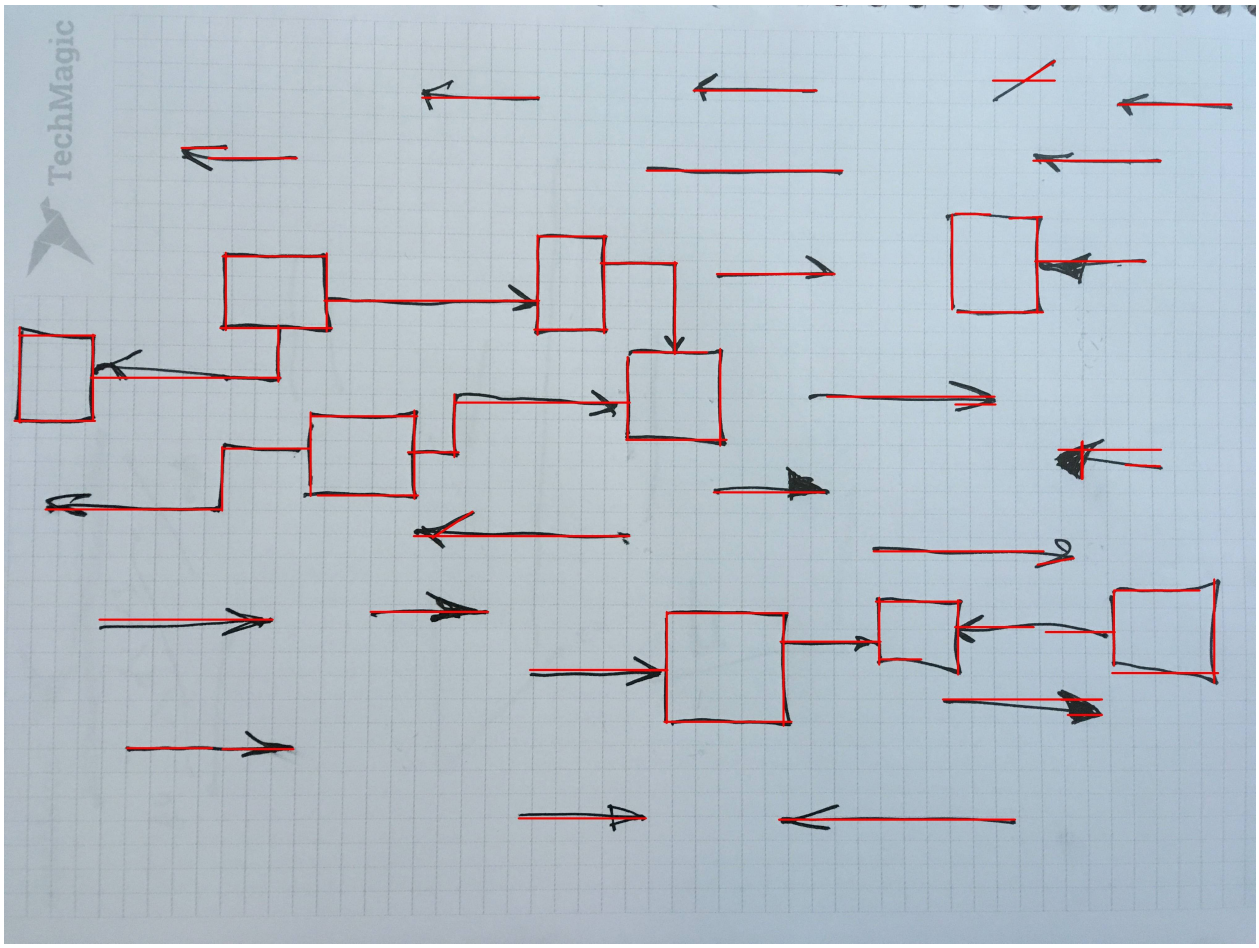
```
    cv2.imwrite('prediction/merged_lines.jpg',img_merged_lines)

    return merged_lines_all

def merge_lines_pipeline_2(lines):
    super_lines_final = []
    super_lines = []
    min_distance_to_merge = 30
    min_angle_to_merge = 30

    for line in lines:
        create_new_group = True
        group_updated = False

        for group in super_lines:
            for line2 in group:
                if get_distance(line2, line) < min_distance_to_merge:
                    # check the angle between lines
                    orientation_i = math.atan2((line[0][1]-line[1][1]),(line[0][0]-
line[1][0]))
                    orientation_j = math.atan2((line2[0][1]-line2[1][1]),(line2[0][0]-
line2[1][0]))


                    if int(abs(abs(math.degrees(orientation_i)) -
abs(math.degrees(orientation_j)))) < min_angle_to_merge:
                        #print("angles", orientation_i, orientation_j)
                        #print(int(abs(orientation_i - orientation_j)))
                        group.append(line)

                        create_new_group = False
                        group_updated = True
                        break

            if group_updated:
                break

        if (create_new_group):
            new_group = []
            new_group.append(line)

            for idx, line2 in enumerate(lines):
                # check the distance between lines
                if get_distance(line2, line) < min_distance_to_merge:
                    # check the angle between lines
                    orientation_i = math.atan2((line[0][1]-line[1][1]),(line[0][0]-
line[1][0]))
                    orientation_j = math.atan2((line2[0][1]-line2[1][1]),(line2[0][0]-
line2[1][0]))

                    if int(abs(abs(math.degrees(orientation_i)) -
abs(math.degrees(orientation_j)))) < min_angle_to_merge:
                        #print("angles", orientation_i, orientation_j)
                        #print(int(abs(orientation_i - orientation_j)))

                        new_group.append(line2)

                        # remove line from lines list
                        #lines[idx] = False
            # append new group
            super_lines.append(new_group)


    for group in super_lines:
        super_lines_final.append(merge_lines_segments1(group))

    return super_lines_final

def merge_lines_segments1(lines, use_log=False):
    if(len(lines) == 1):
```

```
        return lines[0]

    line_i = lines[0]

    # orientation
    orientation_i = math.atan2((line_i[0][1]-line_i[1][1]),(line_i[0][0]-line_i[1][0]))

    points = []
    for line in lines:
        points.append(line[0])
        points.append(line[1])

    if (abs(math.degrees(orientation_i)) > 45) and abs(math.degrees(orientation_i)) <
(90+45):

        #sort by y
        points = sorted(points, key=lambda point: point[1])

        if use_log:
            print("use y")
    else:

        #sort by x

        points = sorted(points, key=lambda point: point[0])

        if use_log:
            print("use x")

    return [points[0], points[len(points)-1]]

#
https://docs.scipy.org/doc/scipy/reference/generated/scipy.spatial.distance.cdist.html
# https://stackoverflow.com/questions/32702075/what-would-be-the-fastest-way-to-find-
the-maximum-of-all-possible-distances-betw
def lines_close(line1, line2):
    dist1 = math.hypot(line1[0][0] - line2[0][0], line1[0][0] - line2[0][1])
    dist2 = math.hypot(line1[0][2] - line2[0][0], line1[0][3] - line2[0][1])
    dist3 = math.hypot(line1[0][0] - line2[0][2], line1[0][0] - line2[0][3])
    dist4 = math.hypot(line1[0][2] - line2[0][2], line1[0][3] - line2[0][3])

    if (min(dist1,dist2,dist3,dist4) < 100):
        return True
    else:
        return False

def lineMagnitude (x1, y1, x2, y2):
    lineMagnitude = math.sqrt(math.pow((x2 - x1), 2)+ math.pow((y2 - y1), 2))
    return lineMagnitude

#Calc minimum distance from a point and a line segment (i.e. consecutive vertices in a
polyline).
# https://nodedangles.wordpress.com/2010/05/16/measuring-distance-from-a-point-to-a-
line-segment/
# http://paulbourke.net/geometry/pointlineplane/
def DistancePointLine(px, py, x1, y1, x2, y2):
    #http://local.wasp.uwa.edu.au/~pbourke/geometry/pointline/source.vba
    LineMag = lineMagnitude(x1, y1, x2, y2)

    if LineMag < 0.00000001:
        DistancePointLine = 9999
        return DistancePointLine

    u1 = (((px - x1) * (x2 - x1)) + ((py - y1) * (y2 - y1)))
    u = u1 / (LineMag * LineMag)

    if (u < 0.00001) or (u > 1):
        #// closest point does not fall within the line segment, take the shorter
distance
        #// to an endpoint
```

```
            ix = lineMagnitude(px, py, x1, y1)
            iy = lineMagnitude(px, py, x2, y2)
            if ix > iy:
                DistancePointLine = iy
            else:
                DistancePointLine = ix
        else:
            # Intersecting point is on the line, use the formula
            ix = x1 + u * (x2 - x1)
            iy = y1 + u * (y2 - y1)
            DistancePointLine = lineMagnitude(px, py, ix, iy)

        return DistancePointLine

    def get_distance(line1, line2):
        dist1 = DistancePointLine(line1[0][0], line1[0][1],
                                  line2[0][0], line2[0][1], line2[1][0], line2[1][1])
        dist2 = DistancePointLine(line1[1][0], line1[1][1],
                                  line2[0][0], line2[0][1], line2[1][0], line2[1][1])
        dist3 = DistancePointLine(line2[0][0], line2[0][1],
                                  line1[0][0], line1[0][1], line1[1][0], line1[1][1])
        dist4 = DistancePointLine(line2[1][0], line2[1][1],
                                  line1[0][0], line1[0][1], line1[1][0], line1[1][1])



        return min(dist1,dist2,dist3,dist4)
```



There are still 572 lines. After my "merging line segments" we have only 89 lines



Share Follow

edited Oct 20 '17 at 7:55

answered Aug 10 '17 at 12:37

Oleg Dats

3,307 8 31 60

11

Rewritten code above, it is 30% faster, shorter and, IMHO, more understandable:

```
class HoughBundler:
    '''Clasterize and merge each cluster of cv2.HoughLinesP() output
```

```
    Clusterize and merge each cluster of cv2.HoughLinesP() output
    a = HoughBundler()
    foo = a.process_lines(houghP_lines, binary_image)
    '''

    def get_orientation(self, line):
        '''get orientation of a line, using its length
        https://en.wikipedia.org/wiki/Atan2
        '''
        orientation = math.atan2(abs((line[0] - line[2])), abs((line[1] - line[3])))
        return math.degrees(orientation)

    def checker(self, line_new, groups, min_distance_to_merge, min_angle_to_merge):
        '''Check if line have enough distance and angle to be count as similar
        '''
        for group in groups:
            # walk through existing line groups
            for line_old in group:
                # check distance
                if self.get_distance(line_old, line_new) < min_distance_to_merge:
                    # check the angle between lines
                    orientation_new = self.get_orientation(line_new)
                    orientation_old = self.get_orientation(line_old)
                    # if all is ok -- line is similar to others in group
                    if abs(orientation_new - orientation_old) < min_angle_to_merge:
                        group.append(line_new)
                        return False
        # if it is totally different line
        return True

    def DistancePointLine(self, point, line):
        """Get distance between point and line
        http://local.wasp.uwa.edu.au/~pbourke/geometry/pointline/source.vba
        """
        px, py = point
        x1, y1, x2, y2 = line

        def lineMagnitude(x1, y1, x2, y2):
            'Get line (aka vector) length'
            lineMagnitude = math.sqrt(math.pow((x2 - x1), 2) + math.pow((y2 - y1), 2))
            return lineMagnitude

        LineMag = lineMagnitude(x1, y1, x2, y2)
        if LineMag < 0.00000001:
            DistancePointLine = 9999
            return DistancePointLine

        u1 = (((px - x1) * (x2 - x1)) + ((py - y1) * (y2 - y1)))
        u = u1 / (LineMag * LineMag)

        if (u < 0.00001) or (u > 1):
            #// closest point does not fall within the line segment, take the shorter
distance
            #// to an endpoint
            ix = lineMagnitude(px, py, x1, y1)
            iy = lineMagnitude(px, py, x2, y2)
            if ix > iy:
                DistancePointLine = iy
            else:
                DistancePointLine = ix
        else:
            # Intersecting point is on the line, use the formula
            ix = x1 + u * (x2 - x1)
            iy = y1 + u * (y2 - y1)
            DistancePointLine = lineMagnitude(px, py, ix, iy)

        return DistancePointLine

    def get_distance(self, a_line, b_line):
        """Get all possible distances between each dot of two lines and second line
```

```
        return the shortest
        """
        dist1 = self.DistancePointLine(a_line[:2], b_line)
        dist2 = self.DistancePointLine(a_line[2:], b_line)
        dist3 = self.DistancePointLine(b_line[:2], a_line)
        dist4 = self.DistancePointLine(b_line[2:], a_line)

        return min(dist1, dist2, dist3, dist4)

    def merge_lines_pipeline_2(self, lines):
        'Clusterize (group) lines'
        groups = []  # all lines groups are here
        # Parameters to play with
        min_distance_to_merge = 30
        min_angle_to_merge = 30
        # first line will create new group every time
        groups.append([lines[0]])
        # if line is different from existing gropus, create a new group
        for line_new in lines[1:]:
            if self.checker(line_new, groups, min_distance_to_merge,
min_angle_to_merge):
                groups.append([line_new])


        return groups

    def merge_lines_segments1(self, lines):
        """Sort lines cluster and return first and last coordinates
        """
        orientation = self.get_orientation(lines[0])

        # special case
        if(len(lines) == 1):
            return [lines[0][:2], lines[0][2:]]

        # [[1,2,3,4],[]] to [[1,2],[3,4],[],[]]
        points = []
        for line in lines:
            points.append(line[:2])
            points.append(line[2:])
        # if vertical
        if 45 < orientation < 135:
            #sort by y
            points = sorted(points, key=lambda point: point[1])
        else:
            #sort by x
            points = sorted(points, key=lambda point: point[0])

        # return first and last point in sorted group
        # [[x,y],[x,y]]
        return [points[0], points[-1]]

    def process_lines(self, lines, img):
        '''Main function for lines from cv.HoughLinesP() output merging
        for OpenCV 3
        lines -- cv.HoughLinesP() output
        img -- binary image
        '''
        lines_x = []
        lines_y = []
        # for every line of cv2.HoughLinesP()
        for line_i in [l[0] for l in lines]:
                orientation = self.get_orientation(line_i)
                # if vertical
                if 45 < orientation < 135:
                    lines_y.append(line_i)
                else:
                    lines_x.append(line_i)

        lines_y = sorted(lines_y, key=lambda line: line[1])
```

```
        lines_x = sorted(lines_x, key=lambda line: line[0])
        merged_lines_all = []

        # for each cluster in vertical and horizantal lines leave only one line
        for i in [lines_x, lines_y]:
                if len(i) > 0:
                    groups = self.merge_lines_pipeline_2(i)
                    merged_lines = []
                    for group in groups:
                        merged_lines.append(self.merge_lines_segments1(group))

                    merged_lines_all.extend(merged_lines)

        return merged_lines_all
```

The part with distance calculation could be changed to

```
def distance_to_line(self, point, line):
    """Get distance between point and line
    https://stackoverflow.com/questions/40970478/python-3-5-2-distance-from-a-point-to-
a-line
    """
    px, py = point
    x1, y1, x2, y2 = line
    x_diff = x2 - x1
    y_diff = y2 - y1
    num = abs(y_diff * px - x_diff * py + x2 * y1 - y2 * x1)
    den = math.sqrt(y_diff**2 + x_diff**2)
    return num / den

def get_distance(self, a_line, b_line):
    """Get all possible distances between each dot of two lines and second line
    return the shortest
    """
    dist1 = self.distance_to_line(a_line[:2], b_line)
    dist2 = self.distance_to_line(a_line[2:], b_line)
    dist3 = self.distance_to_line(b_line[:2], a_line)
    dist4 = self.distance_to_line(b_line[2:], a_line)

    return min(dist1, dist2, dist3, dist4)
```

But you'll get slightly different lines at the end.

Share Follow

answered May 17 '18 at 11:04

banderlog013

895 10 18

Thanks a lot for this question, comments, and answer. It was been very helpful to me finding bordered table grid cells. – user297500 Oct 4 '18 at 14:19