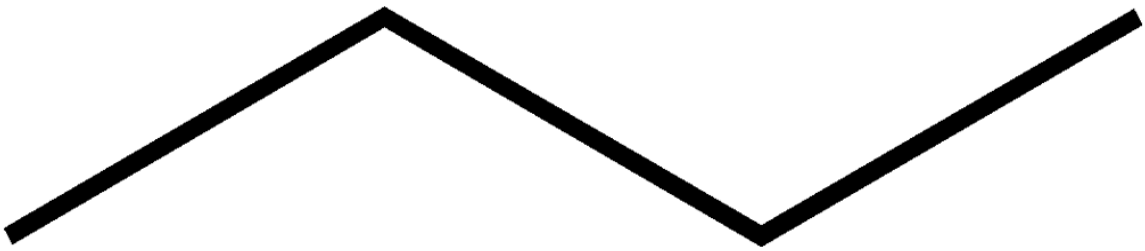
Vectorising black and white image of lines into vectors of no thickness (i.e coordinates)

Asked 11 months ago Modified 11 months ago Viewed 159 times

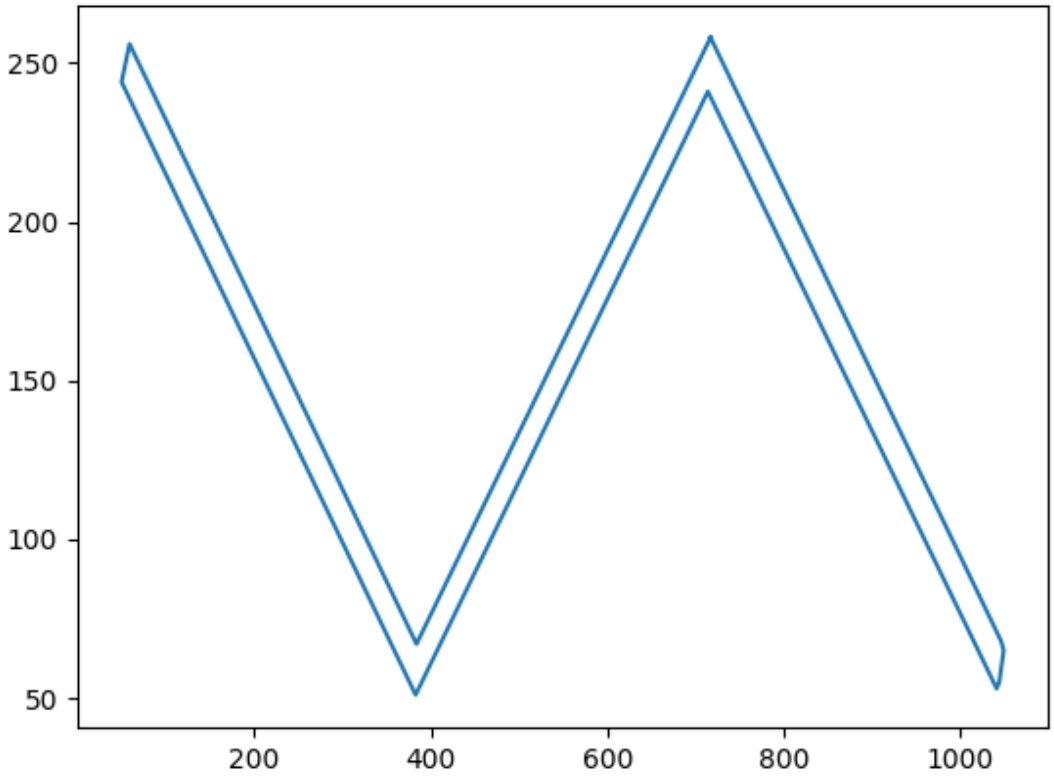
0

1

I want to be able to take in an image for example this:



And then have it return the coordinates of the start and end of each line (e.g. there are 3 lines in the example image). I know there are approaches such as the Hough line transform, and this does give coordinates however since the input image has thickness it returns coordinates of all edges. An example is shown below:



Is there a different way (or tweak to Hough line transform) to get the coordinates where thickness of the line is not taken into account?

python image opencv python-imaging-library vector-graphics

Share Follow

edited Jul 3, 2021 at 23:33
chtz
16k 4 23 52

asked Jul 3, 2021 at 9:53
Jordan D
78 7

2 You could consider getting the medial axis first in order to reduce the line thickness to one pixel... stackoverflow.com/a/59913157/2836621 – Mark Setchell Jul 3, 2021 at 10:38

what do you think about my answer @JordanD? – Prefect Jul 7, 2021 at 18:36

1 @Prefect Beautiful answer thank you for your help! Hope you have a great day :) – Jordan D Jul 7, 2021 at 22:06

1 Answer

Sorted by: Highest score (default)

1

You might like to use skeletonize to get the *skeleton* of your lines.

In order to get the tips/corners of the lines, this is the procedure I am following:

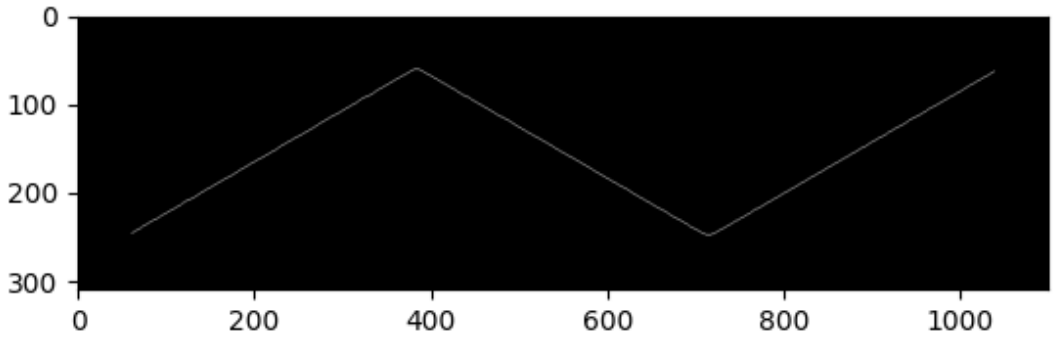
1. Get the skeleton image
2. Find the tips by filtering the skeleton image by the matrix of ones with 3x3 size.
3. Fit lines to the skeleton image by the Hough Transform
4. Find the line corners by the line intersections

Here is my code. Let's start with importing the required libraries.

```
import cv2
import matplotlib.pyplot as plt
import numpy as np
from skimage.morphology import skeletonize
from itertools import combinations
```

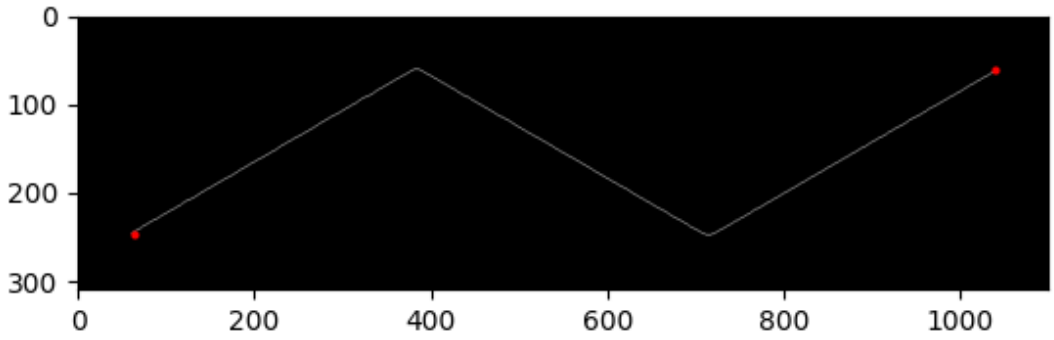
I am starting with getting the skeleton image. I will be using this image to detect all the points. Note that the default method for skeletonize creates forks in the skeleton image. But the method lee works better for your problem. Just watch out for your other images.

```
img_bgr = cv2.imread('lines.png',1)
img = cv2.cvtColor(img_bgr,cv2.COLOR_BGR2GRAY)

_,img = cv2.threshold(img,10,1,cv2.THRESH_BINARY_INV)

skeleton = skeletonize(img,method='lee')

# display purposes
skeleton_bgr = cv2.cvtColor(skeleton.astype(np.uint8),cv2.COLOR_GRAY2BGR)
```



Next thing is to get the tips of the lines.

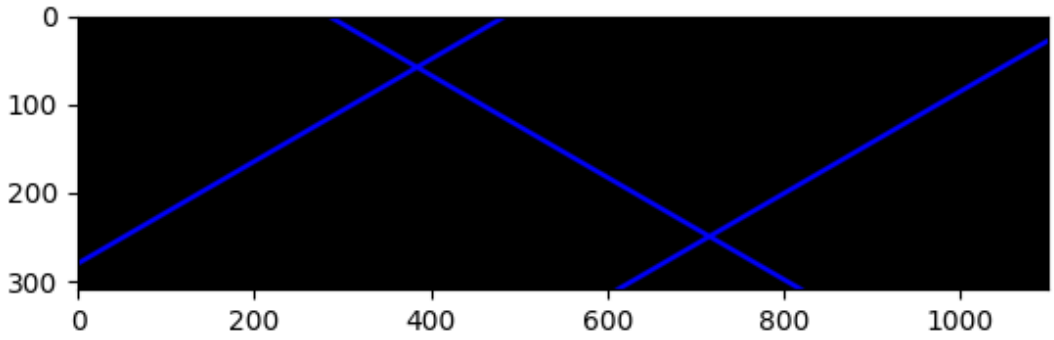
```
img_conv = cv2.filter2D(skeleton.astype(np.uint8),-1,np.ones((3,3))) #
img_conv = img_conv*skeleton
img_tips = img_conv == 2
tips = np.array(np.nonzero(img_tips)).T
```



FYI, the detected tips are a few pixels off from the skeleton, I am missing something probably, I leave this part to you.

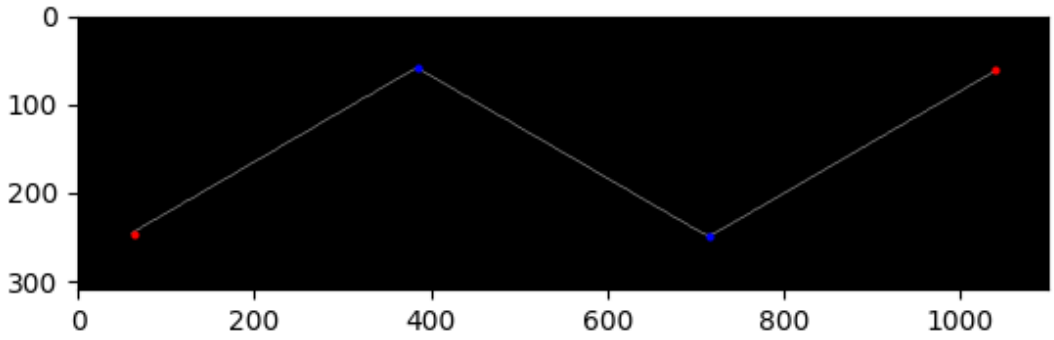
As you suggested in your question, in the next step, I am fitting lines to the skeleton image.

```
lines = cv2.HoughLines(skeleton.astype(np.uint8), 1, np.pi / 180, 150, None, 0, 0)

lines_points = []

# # Draw the lines
for i in range(0, len(lines)):
    rho = lines[i][0][0]
    theta = lines[i][0][1]
    a = np.cos(theta)
    b = np.sin(theta)
    x0 = a * rho
    y0 = b * rho
    pt1 = (int(x0 + 1000*(-b)), int(y0 + 1000*(a)))
    pt2 = (int(x0 - 1000*(-b)), int(y0 - 1000*(a)))
    # cv2.line(skeleton_bgr, pt1, pt2, (0,0,255), 3, cv2.LINE_AA)
    lines_points.append([pt1,pt2])
```

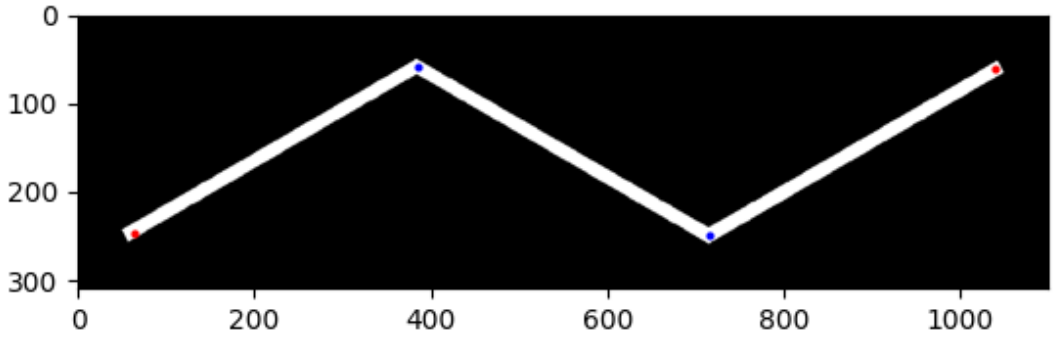


Then, I am getting the corners of the lines by the intersection points.

```
def line_intersection(line1, line2):
    xdiff = (line1[0][0] - line1[1][0], line2[0][0] - line2[1][0])
    ydiff = (line1[0][1] - line1[1][1], line2[0][1] - line2[1][1])

    def det(a, b):
        return a[0] * b[1] - a[1] * b[0]

    div = det(xdiff, ydiff)
    if div == 0:
       raise Exception('lines do not intersect')

    d = (det(*line1), det(*line2))
    x = det(d, xdiff) / div
    y = det(d, ydiff) / div
    return x, y

line_combinations = combinations(lines_points, 2)

fig,ax = plt.subplots(1)
ax.imshow(skeleton, cmap = 'gray')
ax.scatter(tips[0,:],tips[1,:],s=4,color='r')

for line_combination in list(line_combinations):
    line1 = line_combination[0]
    line2 = line_combination[1]

    intersection = line_intersection(line1,line2)

    # clip the intersection points to the image size
    if intersection[0]<img.shape[1] and intersection[1]<img.shape[0]:
        ax.scatter(intersection[0],intersection[1],s=4,color='b')
```



This is the outcome on the original image.



This is the complete code including the plotting functions.

```
import cv2
import matplotlib.pyplot as plt
import numpy as np
from skimage.morphology import skeletonize
from itertools import combinations

def line_intersection(line1, line2):
    xdiff = (line1[0][0] - line1[1][0], line2[0][0] - line2[1][0])
    ydiff = (line1[0][1] - line1[1][1], line2[0][1] - line2[1][1])

    def det(a, b):
        return a[0] * b[1] - a[1] * b[0]

    div = det(xdiff, ydiff)
    if div == 0:
       raise Exception('lines do not intersect')

    d = (det(*line1), det(*line2))
    x = det(d, xdiff) / div
    y = det(d, ydiff) / div
    return x, y


img_bgr = cv2.imread('lines.png',1)
img = cv2.cvtColor(img_bgr,cv2.COLOR_BGR2GRAY)

_,img = cv2.threshold(img,10,1,cv2.THRESH_BINARY_INV)

skeleton = skeletonize(img,method='lee')
skeleton_bgr = cv2.cvtColor(skeleton.astype(np.uint8),cv2.COLOR_GRAY2BGR)


# fig,ax = plt.subplots(1)
# ax.imshow(skeleton, cmap = 'gray')


#get tips
img_conv = cv2.filter2D(skeleton.astype(np.uint8),-1,np.ones((3,3))) #
img_conv = img_conv*skeleton
img_tips = img_conv == 2
tips = np.array(np.nonzero(img_tips)).T



# fig,ax = plt.subplots(1)
# ax.imshow(skeleton, cmap = 'gray')
# ax.scatter(tips[0,:],tips[1,:],s=4,color='r')


lines = cv2.HoughLines(skeleton.astype(np.uint8), 1, np.pi / 180, 150, None, 0, 0)

lines_points = []

# # Draw the lines
for i in range(0, len(lines)):
    rho = lines[i][0][0]
    theta = lines[i][0][1]
    a = np.cos(theta)
    b = np.sin(theta)
    x0 = a * rho
    y0 = b * rho
    pt1 = (int(x0 + 1000*(-b)), int(y0 + 1000*(a)))
    pt2 = (int(x0 - 1000*(-b)), int(y0 - 1000*(a)))
    cv2.line(skeleton_bgr, pt1, pt2, (0,0,255), 3, cv2.LINE_AA)
    lines_points.append([pt1,pt2])
```

```
line_combinations = combinations(lines_points, 2)


fig,ax = plt.subplots(1)
ax.imshow(img, cmap = 'gray')
ax.scatter(tips[0,:],tips[1,:],s=4,color='r')

for line_combination in list(line_combinations):
    line1 = line_combination[0]
    line2 = line_combination[1]

    intersection = line_intersection(line1,line2)

    # clip the intersection points to the image size
    if intersection[0]<img.shape[1] and intersection[1]<img.shape[0]:
        ax.scatter(intersection[0],intersection[1],s=4,color='b')

# fig,ax = plt.subplots(1)
# ax.imshow(skeleton_bgr,'gray')

plt.show()
```

Share Follow

answered Jul 5, 2021 at 6:51

Prefect
1,579 1 6 15