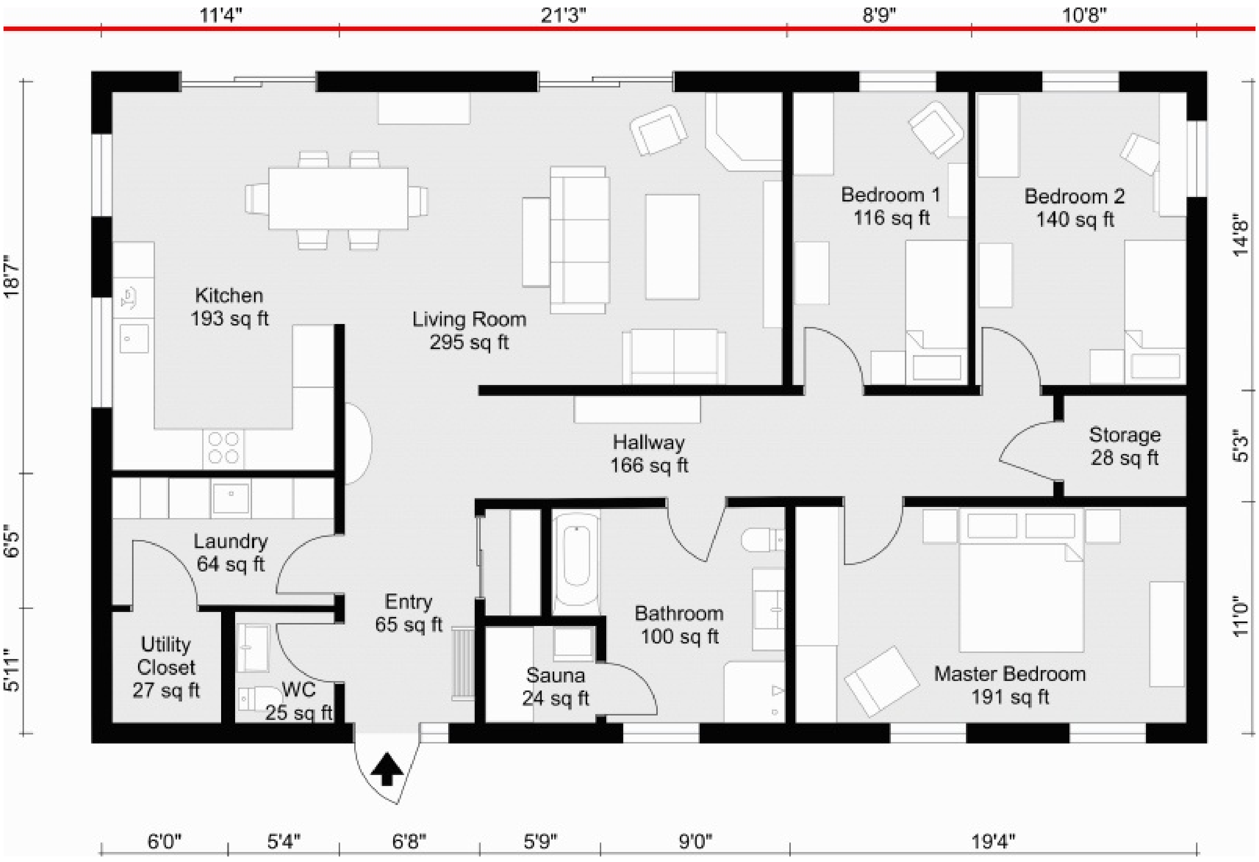
Separate rooms in a floor plan using OpenCV

Asked 2 years, 2 months ago   Active 2 years, 1 month ago   Viewed 5k times

4

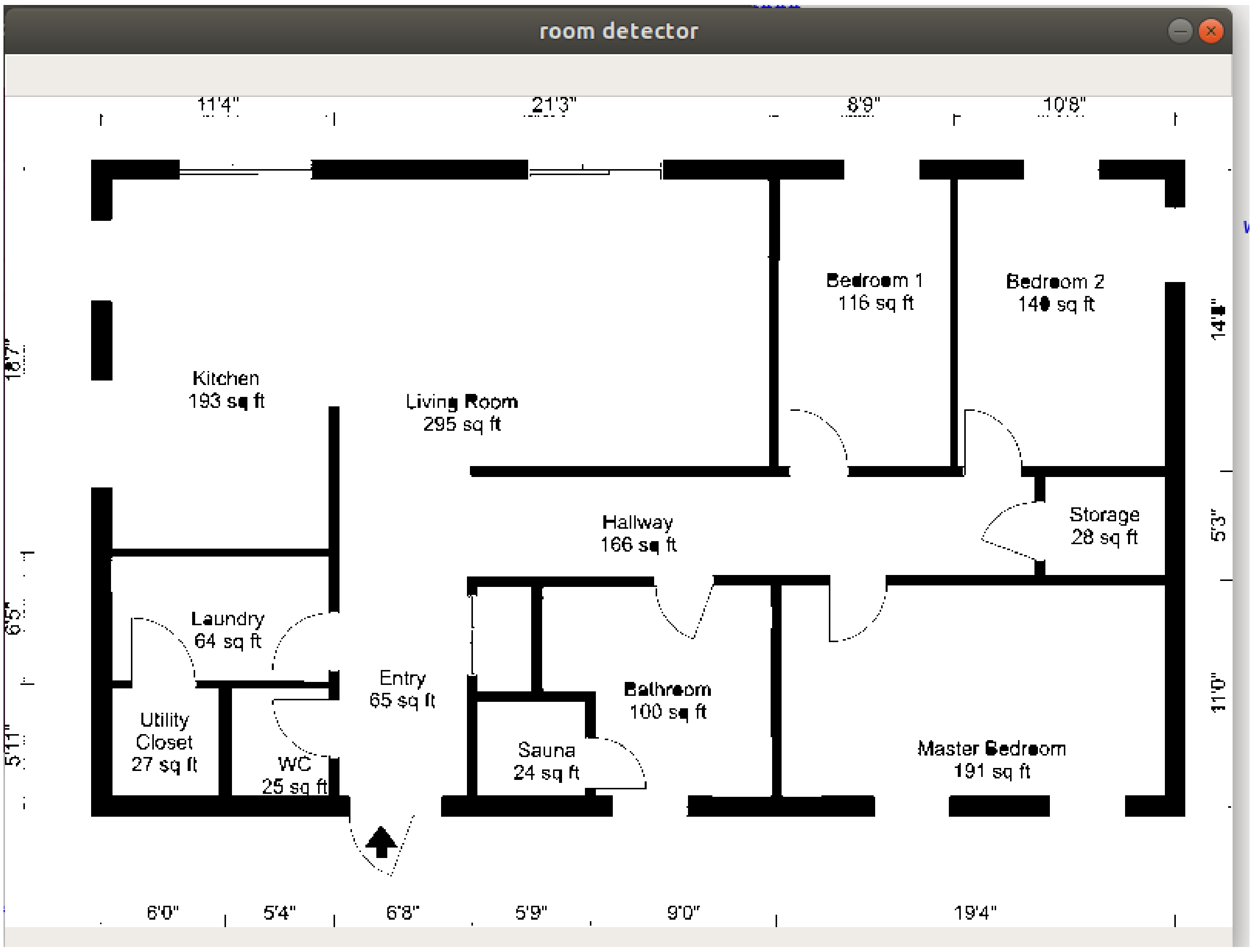
5

Input floor plan image



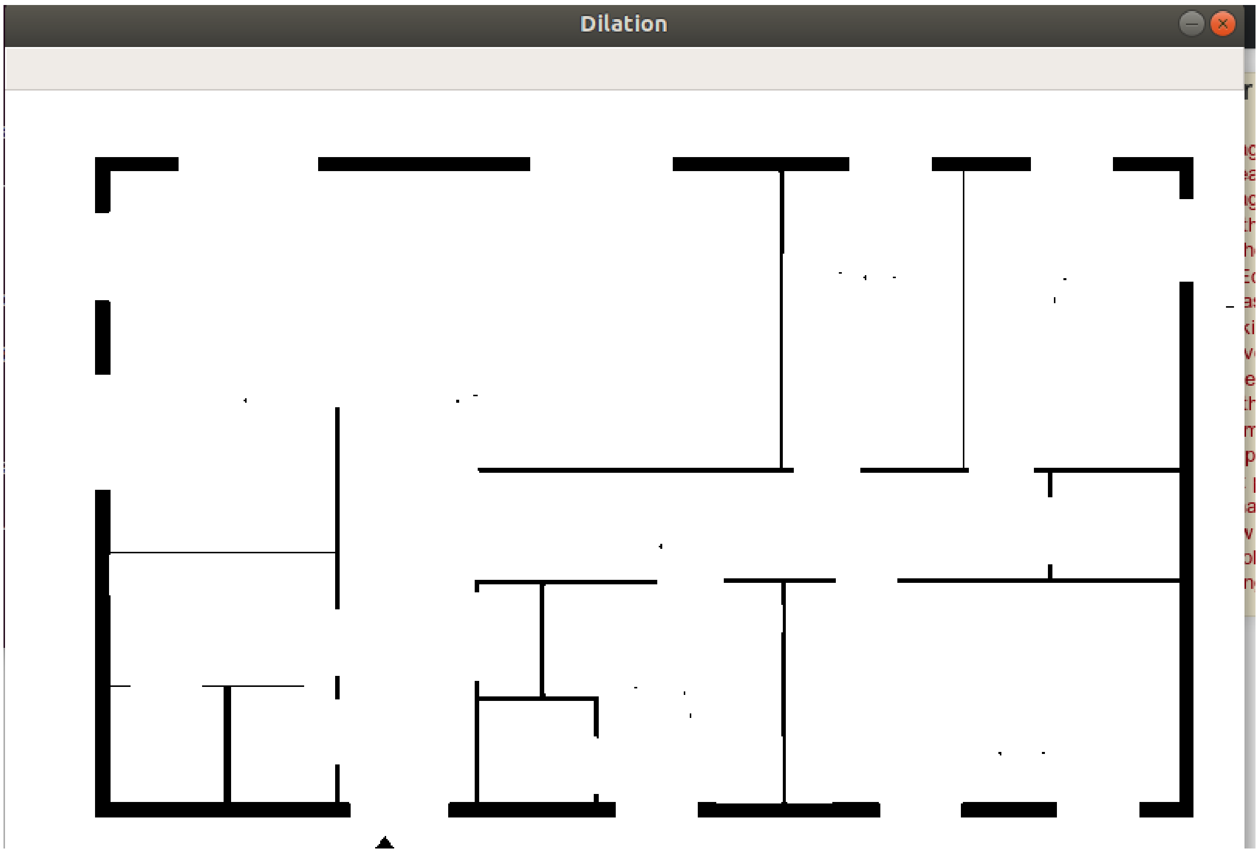
Above images are my input floor plan and I need to identify each room separately and then crop those rooms. after that, I can use those images for the next steps. So far I was able to Remove Small Items from input floor plans by using cv2.connectedComponentsWithStats. So that I think it will help to identify wall easily. after that my input images look like this.

output image after removing small objects



Then I did MorphologicalTransform to remove text and other symbols from image to leave only the walls. after that my input image look like this.

after MorphologicalTransform



So I was able to identify walls. then how I use those wall to crop rooms from the original input floor plan. Can someone help me? You can find my python code in this link. Download My Code or

```
#Import packages
import os
import cv2
import numpy as np
import tensorflow as tf
import sys

# This is needed since the notebook is stored in the object_detection folder.
sys.path.append("..")

# Import utilites
from utils import label_map_util
from utils import visualization_utils as vis_util

# Name of the directory containing the object detection module we're using
MODEL_NAME = 'inference_graph'
IMAGE_NAME = 'floorplan2.jpg'
#Remove Small Items
im_gray = cv2.imread(IMAGE_NAME, cv2.IMREAD_GRAYSCALE)
(thresh, im_bw) = cv2.threshold(im_gray, 128, 255, cv2.THRESH_BINARY | cv2.THRESH_OTSU)
thresh = 127
im_bw = cv2.threshold(im_gray, thresh, 255, cv2.THRESH_BINARY)[1]
#find all your connected components
nb_components, output, stats, centroids = cv2.connectedComponentsWithStats(im_bw,
connectivity=8)
#connectedComponentswithStats yields every seperated component with information on each
of them, such as size
#the following part is just taking out the background which is also considered a
component, but most of the time we don't want that.
sizes = stats[1:, -1]; nb_components = nb_components - 1

# minimum size of particles we want to keep (number of pixels)
#here, it's a fixed value, but you can set it as you want, eg the mean of the sizes or
whatever
min_size = 150

#your answer image
img2 = np.zeros((output.shape))
#for every component in the image, you keep it only if it's above min_size
for i in range(0, nb_components):
    if sizes[i] >= min_size:
        img2[output == i + 1] = 255

cv2.imshow('room detector', img2)
#MorphologicalTransform
kernel = np.ones((5, 5), np.uint8)
dilation = cv2.dilate(img2, kernel)
erosion = cv2.erode(img2, kernel, iterations=6)

#cv2.imshow("img2", img2)
cv2.imshow("Dilation", dilation)
cv2.imwrite("Dilation.jpg", dilation)
#cv2.imshow("Erosion", erosion)


# Press any key to close the image
cv2.waitKey(0)

# Clean up
cv2.destroyAllWindows()
```

python opencv image-processing

Share Follow

edited Mar 14 '19 at 10:02
RunOrVeith
2,143 1 20 31

asked Jan 20 '19 at 8:05
kavin_Ab
159 1 10

2 Answers

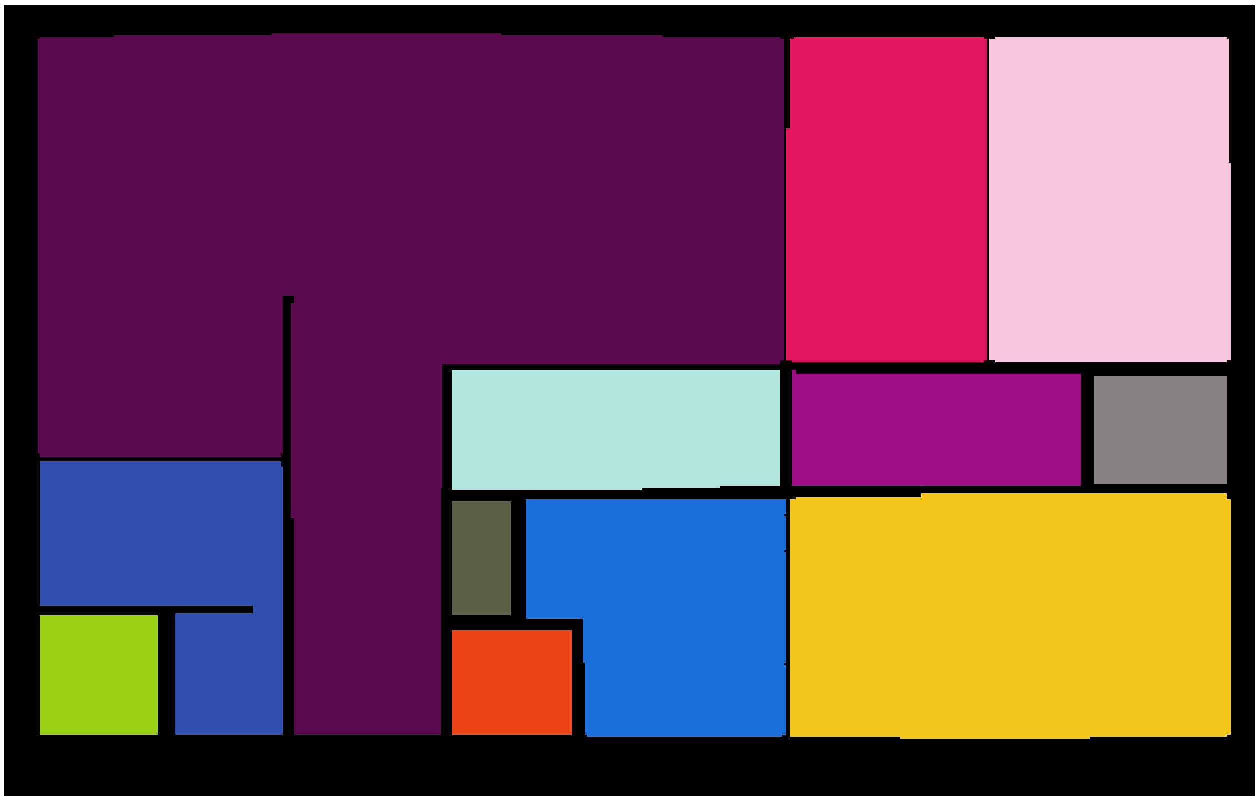
Active Oldest Votes

2

Here is something that I've come up with. It is not perfect (I made some comments what you might want to try), and it will be better if you improve the input image quality.

```
import cv2
import numpy as np



def find_rooms(img, noise_removal_threshold=25, corners_threshold=0.1,
               room_closing_max_length=100, gap_in_wall_threshold=500):
    """

    :param img: grey scale image of rooms, already eroded and doors removed etc.
    :param noise_removal_threshold: Minimal area of blobs to be kept.
    :param corners_threshold: Threshold to allow corners. Higher removes more of the
house.
    :param room_closing_max_length: Maximum line length to add to close off open doors.
    :param gap_in_wall_threshold: Minimum number of pixels to identify component as
room instead of hole in the wall.
    :return: rooms: list of numpy arrays containing boolean masks for each detected
room
             colored_house: A colored version of the input image, where each room has a
random color.
    """
    assert 0 <= corners_threshold <= 1
    # Remove noise left from door removal

    img[img < 128] = 0
    img[img > 128] = 255
    _, contours, _ = cv2.findContours(~img, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
    mask = np.zeros_like(img)
    for contour in contours:
        area = cv2.contourArea(contour)
        if area > noise_removal_threshold:
            cv2.fillPoly(mask, [contour], 255)

    img = ~mask

    # Detect corners (you can play with the parameters here)
    dst = cv2.cornerHarris(img ,2,3,0.04)
    dst = cv2.dilate(dst,None)
    corners = dst > corners_threshold * dst.max()

    # Draw lines to close the rooms off by adding a line between corners on the same x
or y coordinate
    # This gets some false positives.
    # You could try to disallow drawing through other existing lines for example.
    for y,row in enumerate(corners):
        x_same_y = np.argwhere(row)
        for x1, x2 in zip(x_same_y[:-1], x_same_y[1:]):

            if x2[0] - x1[0] < room_closing_max_length:
                color = 0
                cv2.line(img, (x1, y), (x2, y), color, 1)

    for x,col in enumerate(corners.T):
        y_same_x = np.argwhere(col)
        for y1, y2 in zip(y_same_x[:-1], y_same_x[1:]):
            if y2[0] - y1[0] < room_closing_max_length:
                color = 0
                cv2.line(img, (x, y1), (x, y2), color, 1)


    # Mark the outside of the house as black
    _, contours, _ = cv2.findContours(~img, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
    contour_sizes = [(cv2.contourArea(contour), contour) for contour in contours]
    biggest_contour = max(contour_sizes, key=lambda x: x[0])[1]
    mask = np.zeros_like(mask)
    cv2.fillPoly(mask, [biggest_contour], 255)
    img[mask == 0] = 0

    # Find the connected components in the house
    ret, labels = cv2.connectedComponents(img)
    img = cv2.cvtColor(img,cv2.COLOR_GRAY2RGB)
    unique = np.unique(labels)
    rooms = []
    for label in unique:
        component = labels == label
        if img[component].sum() == 0 or np.count_nonzero(component) <
gap_in_wall_threshold:
            color = 0
        else:
            rooms.append(component)
            color = np.random.randint(0, 255, size=3)
        img[component] = color

    return rooms, img



#Read gray image
img = cv2.imread("/home/veith/Pictures/room.png", 0)
rooms, colored_house = find_rooms(img.copy())
cv2.imshow('result', colored_house)
cv2.waitKey()
cv2.destroyAllWindows()
```

This will show an image like this, where each room has a random color: You can see that it sometimes finds a room where there is none, but I think this is a decent starting point for you.



I've used a screenshot of the image in your question for this. You can use the returned masks of each room to index the original image and crop that. To crop just use something like (untested, but should work for the most part):

```
for room in rooms:
    crop = np.zeros_like(room).astype(np.uint8)
    crop[room] = original_img[room]  # Get the original image from somewhere
    # if you need to crop the image into smaller parts as big as each room
    r, c = np.nonzero(room)
    min_r, max_r = r.argmin(), r.argmax()
    min_c, max_c = c.argmin(), c.argmax()
    crop = crop[min_r:max_r, min_c:max_c]
    cv2.imshow("cropped room", crop)
    cv2.waitKey()
    cv2.destroyAllWindows()
```

Share Follow

edited Jan 21 '19 at 9:08

answered Jan 20 '19 at 15:09

RunOrVeith

2,143 1 20 31

Thank you so much @RunOrVeith. Code works better than i expected. – kavin_Ab Jan 20 '19 at 17:06

My final goal is to crop my original image rooms from these colored image rooms. I want to take this code a step further to crop the final image. I tried to debug this code. But I could not find a way to do this. @RunOrVeith Could you help me for that? – kavin_Ab Jan 21 '19 at 6:05

sorry, I don't have time to test it today – RunOrVeith Jan 21 '19 at 9:52

I think you can also try it yourself, it should not be that hard. `rooms` is a list of boolean masks with the same shape as `img`, and elements are True for each room – RunOrVeith Jan 21 '19 at 10:23

I used three for loops to crop each room.

1

```
height, width = img.shape[:2]
rooms, colored_house = find_rooms(img.copy())
roomId = 0
images = []
for room in rooms:
    x = 0
    image = np.zeros ((height, width, 3), np.uint8)
    image[np.where ((image == [0, 0, 0]).all (axis=2))] = [0, 33, 166]
    roomId = roomId + 1
    for raw in room:
        y = 0
        for value in raw:
            if value == True:
                image[x,y] = img[x,y]

            y = y +1
            #print (value)
            #print (img[x,y])
        x = x + 1
    cv2.imwrite ('result' + str(roomId)+ '.jpg', image)
```

Share Follow

answered Jan 22 '19 at 7:20

kavin_Ab

159 1 10