*"""  
Adaptive cell decomposition works by dividing a 2D map into different segments, so that the  
robot knows which space is free and which is not. It can do this by different methods, an example  
being the one I have coded here: Quadtrees.  
The robot uses then these empty segments as nodes to plan a path that will avoid obstacles.  
"""*import matplotlib.pyplot as plt  
import skimage  
import numpy as np  
import pandas as pd  
  
  
# Check if box is empty  
def check\_if\_box\_is\_empty(shape\_check):  
 answer = False  
 for i in range(int(shape\_check[0][0] + 1), int(shape\_check[0][1] - 1)):  
 for j in range(int(shape\_check[1][0] + 1), int(shape\_check[1][1] - 1)):  
 if not binary[i][j]:  
 answer = True  
 return answer  
  
  
# split the input box into new boxes  
def split\_in\_four(shape\_x, shape\_y, shape\_size):  
 # Draw the lines  
 binary[int((shape\_x[1] + shape\_x[0]) / 2), int(shape\_y[0]):int(shape\_y[1])] = False  
 binary[int(shape\_x[0]):int(shape\_x[1]), int((shape\_y[1] + shape\_y[0]) / 2)] = False  
  
 shape1 = (shape\_x[0], (shape\_x[1] + shape\_x[0]) / 2), \  
 (shape\_y[0], (shape\_y[1] + shape\_y[0]) / 2), shape\_size  
 shape2 = (shape\_x[0], (shape\_x[1] + shape\_x[0]) / 2), \  
 ((shape\_y[1] + shape\_y[0]) / 2, shape\_y[1]), shape\_size  
 shape3 = ((shape\_x[1] + shape\_x[0]) / 2, shape\_x[1]), \  
 (shape\_y[0], (shape\_y[1] + shape\_y[0]) / 2), shape\_size  
 shape4 = ((shape\_x[1] + shape\_x[0]) / 2, shape\_x[1]), \  
 ((shape\_y[1] + shape\_y[0]) / 2, shape\_y[1]), shape\_size  
 return shape1, shape2, shape3, shape4  
  
  
# Main function that implement the other functions  
def main(shape\_main, i):  
 if check\_if\_box\_is\_empty(shape\_main) and shape\_main[2] < i:  
 shape\_main = list(shape\_main)  
 shape\_main[2] += 1  
 shape\_main = tuple(shape\_main)  
  
 (shape1, shape2, shape3, shape4) = split\_in\_four(shape\_main[0], shape\_main[1],  
 shape\_main[2])  
 main(shape1, i)  
 main(shape2, i)  
 main(shape3, i)  
 main(shape4, i)  
  
  
image = skimage.io.imread('Tel280.png', as\_gray=True)  
thresh = skimage.filters.threshold\_otsu(image)  
binary = image > thresh  
df = pd.DataFrame(binary)  
shape\_max = np.shape(image)  
  
shape\_x\_direction = (7, shape\_max[0] - 9) # To remove frame  
shape\_y\_direction = (6, shape\_max[1] - 8)  
shape = (shape\_x\_direction, shape\_y\_direction, 0)  
number\_of\_splits = 100  
  
main(shape, number\_of\_splits)  
  
plt.imshow(binary, cmap=plt.cm.gray)  
plt.show()