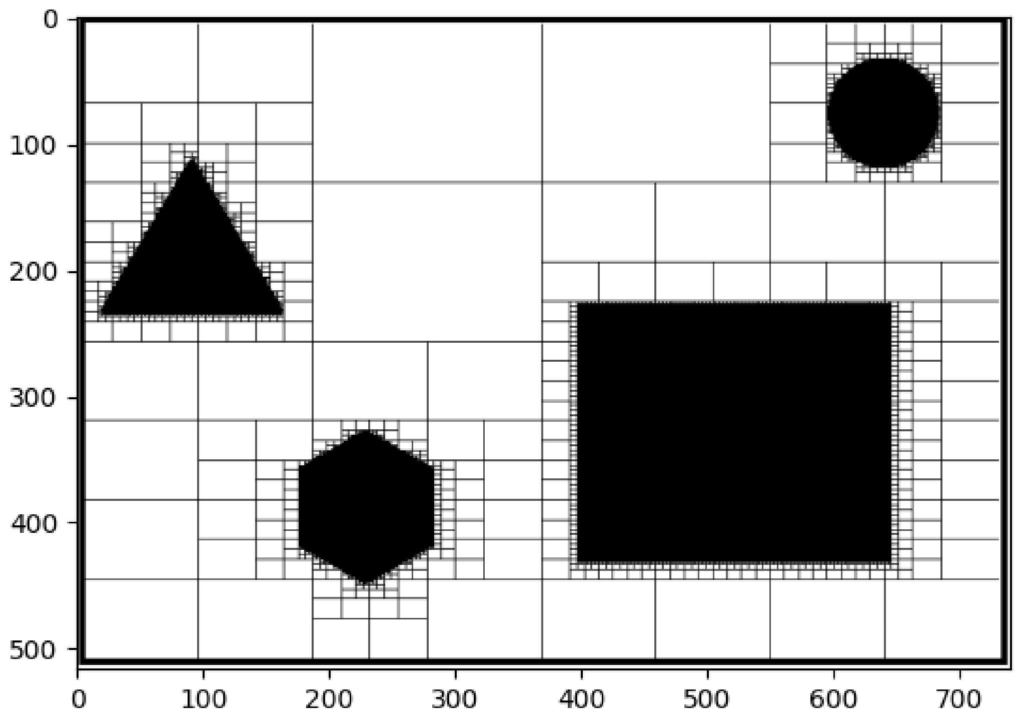
```python
"""
Adaptive cell decomposition works by dividing a 2D map into different
segments, so that the
robot knows which space is free and which is not. It can do this by
different methods, an example
being the one I have coded here: Quadtrees.
The robot uses then these empty segments as nodes to 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)
```

```python
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()
```



0
100
200
300
400
500
0
100
200
300
400
500
600
700