A Quadtree-based Spatial Object Search Scheme

Hyeongjun Jeon
Department of Division of Computer Mechatronics Engineering, Sahmyook University,
815 Hwarang-ro, Nowon-gu, Seoul, 01795, Korea
chariot0720@gmail.com
http://www.syu.ac.kr

Eunjin Kang
Department of Division of Computer Mechatronics Engineering, Sahmyook University,
815 Hwarang-ro, Nowon-gu, Seoul, 01795, Korea
ejk0729@naver.com
http://www.syu.ac.kr

Dukshin Oh
Department of Management Information Systems, Sahmyook University,
815 Hwarang-ro, Nowon-gu, Seoul, 01795, Korea
ohds@syu.ac.kr
http://www.syu.ac.kr

Jongwan Kim*
Smith College of Liberal Arts, Sahmyook University,
815 Hwarang-ro, Nowon-gu, Seoul, 01795, Korea
kimj@syu.ac.kr
http://www.syu.ac.kr

Abstract - As artificial intelligence technology advances, studies are actively being conducted on techniques for detecting objects spatially. This paper presents a quadtree-based spatial object search scheme that quickly stores approximations of the distributions of objects that are detected by various methods. The proposed method divides a single space into 4 spaces and finds the distribution of objects in order to obtain quadtree results that are used to understand the approximate distribution of the objects and to calculate their area. The results can be used for robot vision, in which it must be possible to distinguish the locations of obstacles in real time, or for finding areas where fires occur. In this paper, a quadtree-based spatial search algorithm is implemented, and the actual implementation is verified.

***Keywords*:** Spatial Object; Quadtree; Region Quadtree.

1. Introduction

Object detection is a technology that distinguishes objects in video sequences and clusters these objects' pixels [Parekh *et al*., 2014]. It is one of the most fundamental and challenging problems in computer vision, and it has been an active field of research for several decades [Liu *et al*., 2019].

This paper proposes a scheme that quickly stores the approximate distributions of objects that have been detected by various methods i.e. a quadtree-based object search scheme that divides a single space into 4 spaces of the same size, finds the distribution of the objects, and depicts the distribution of the objects as a quadtree data structure. The proposed method divides a single space into 4 spaces and determines whether there are objects in each space. If there are objects in a certain space, that space is again divided into 4 spaces of the same size and searched. This process is performed until reaching the specified depth of the quadtree that the user is searching. The distribution of the objects, i.e. whether or not objects exist in a certain space, is stored in the quadtree. The user can use the stored results, i.e. the quadtree, to understand the objects' approximate distribution.

Objects do not always exist as shapes such as triangles or quadrangles for which it is easy to calculate the object's area. The more complex an object's shape is, the more difficult it is to find the area. If the proposed scheme is used to search the spatial data in quadrant units, the area can be found as the product of the number of quadrants where objects exist, and this makes the object area calculations easy. Because the object distribution

* Corresponding author.

is searched only until reaching the tree level specified by the user, this scheme is suitable for quickly understanding the distribution of objects. It can be used for robot vision, which must distinguish the locations of obstacles in real time and find the next movement path, or for finding areas where fires occur.

This paper is organized as follows. Section 2 introduces related study. Section 3 introduces the proposed quadtree-based spatial search scheme, and it proposes an object search algorithm that uses this scheme. Section 4 verifies the implementation and efficiency of the algorithm through experiments. Finally, Section 5 discusses this study's conclusions and future research.

2. Related Study

2.1. *Region Quadtree*

Hierarchical data structures are a representation technology that is gradually becoming more important in the fields of computer graphics, image processing, computational geometry, geographic information systems, and robot engineering. A quadtree is a type of hierarchical data structure that reduces execution time through a hierarchical nature that depends on the number of image blocks rather than the image size [Samet, 1985]. This paper uses the region quadtree, which is the most-studied type of quadtree [Samet, 1984].

The region quadtree segments a single space into 4 quadrants of the same size. The quadrant data is made into a tree and stored. The internal nodes of the tree are in a state where a single quadrant has been spatially segmented and has 4 child nodes. The leaf nodes store information on whether an object exists in the corresponding quadrant. If an object exists within the entirety of the corresponding quadrant, the leaf node is called BLACK. If an object doesn't exist, the leaf node is called WHITE. All internal nodes are called GRAY. In Fig. 1, the values of an image array 1(a) are converted to *ones* if an object exists and *zeros* if an object does not exist 1(b). These values are spatially segmented into 4 quadrants of the same size 1(c) and saved as a tree 1(d).

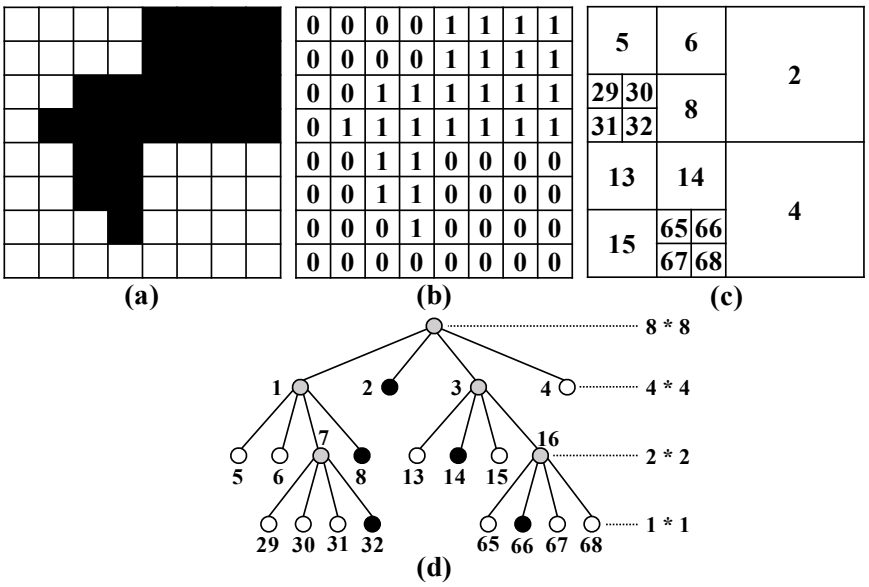

Fig. 1. Region quadtree

3. Quadtree-based Object Search Scheme

The quadtree-based spatial object search scheme divides a single space into 4 spaces of the same size and determines whether there are objects in each of these spaces. If there is an object in a certain space, the space is again divided into 4 spaces of the same size and searched. This process is performed until reaching the specified depth of the quadtree that the user is searching. The distribution of the objects, i.e. whether or not objects exist in a certain space, is stored in the quadtree.

In the proposed method, raster images are inputted as the space that will be searched. Also, the desired parts of the raster images can be saved as objects by setting up the system to convert pixels that have certain values into *true* or *false* according to the actual usage environment and saving them as 2D arrays. In the experiments, the raster images' transparent pixels, which have a transparency level of 0, were converted to *false*, and the non-transparent pixels, which have a transparency level of 1-255, were converted to *true*, and the image was saved as a 2D array of Boolean data. In the 2D array, a *false* value means that there is no object at that index's pixel, and a *true* value means that there is an object at that index's pixel.

Fig. 2 and Table 1 show the algorithm that was implemented by recursively calling the quadtree-based spatial object search scheme. The arguments that the *quadtree_search()* function receives include the range of x coordinates to be searched, the range of y coordinates to be searched, the coordinates where an object was discovered on the higher level, the level, and which index of the quadtree (which is being managed as a one-dimensional array) corresponds to the space that is to be searched. The pixel search is performed from top to bottom and left to right via the *for* statement in lines 10. If an object is discovered in the space that is being searched (line 11), the quadtree's one-dimensional array value at the corresponding index is set to *true* (line 12). If the current tree level has not yet reached the tree level set by the user (line 13), the *quadtree_search()* function is recursively called in lines 15-18. From a logical standpoint, this divides the spaces where objects have been found into 4 spaces of equal size and searches each space as the tree levels increase.

Pseudo code of quadtree_search()

```
     Input  : min_x, max_x, min_y, max_y, current_x, current_y, level, index
     Output: object_grid[the setting level index]
 1   (temp_x, temp_y) ← (temp_x, temp_y)
 2   if max_x < current_x then
 3      object_grid[index] ← false
 4      return
 5   else if (min_x ≤ current_x ≤ max_x) and !(min_y ≤ current_y ≤ max_y) then
 6      temp_y ← min_y
 7      if min_y < current_y then temp_x  ← temp_x++
 8   else if current_x < min_x then
 9      (temp_x, temp_y) ← (min_x, min_y)
10   for (i, j) : (temp_x..max_x, temp_y..max_y)
11      if (i, j) pixel of the image is not transparent then
12         object_grid[index] ← true
13         if level is lower than the setting level then
14            (half_x, half_y) ← ((max_x-min_x+1)/2, (max_y-min_y+1)/2)
15            quadtree_search(min_x, min_x+half_x-1, min_y, min_y+half_y-1, i, j, level+1, index*4+1)
16            quadtree_search(min_x+half_x, max_x, min_y, min_y+half_y-1, i, j, level+1, index*4+2)
17            quadtree_search(min_x, min_x+half_x-1, min_y+half_y, max_y, i, j, level+1, index*4+3)
18            quadtree_search(min_x+half_x, max_x, min_y+half_y, max_y, i, j, level+1, index*4+4)
19            return
20         return
21   end for
```

Fig. 2. Quadtree-based spatial object search algorithm.

Table 1. Variables that are used

Variable	Description
min_x	Minimum x coordinate in the search space
max_x	Maximum x coordinate in the search space
min_y	Minimum y coordinate in the search space
max_y	Maximum y coordinate in the search space
half_x	Median x coordinate in the search space
half_y	Median y coordinate in the search space
current_x	x coordinate that was being searched in the higher level
current_y	y coordinate that was being searched in the higher level
temp_x	x coordinate that is moved to search the current level
temp_y	y coordinate that is moved to search the current level
level	Quadtree level that is to be searched
index	Index of one-dimensional array that is used by the quadtree
object_grid	One-dimensional array that is used by the quadtree

However, if the coordinates that are searched on the higher level are searched again on the lower level, the search is duplicated, which extends the search time and is inefficient. To mitigate the problem of duplicate searches, the coordinates that are being searched on the higher level (*current_x, current_y*) and the coordinates that will be searched on the current level (*temp_x, temp_y*) are distinguished (line 1), and the coordinates that will be searched are reset on lines 2-9.

First, if objects are not found in spaces 1 or 3 but they are found in spaces 2 or 4 during the search of the overall space in Fig. 3(a), the function is called recursively by lines 15-18. Here, the *current_x* coordinate exceeds the *max_x* (line 2) in the *quadtree_search()* function for space 1, which was called by line 15, and the *quadtree_search()* function for space 3, which was called by line 17. Therefore, the searches are not performed, and *false* values are saved in the quadtree array (lines 3), and the functions return (line 4).

Second, if objects are found in spaces 1 or 3 during the search of the overall space in Fig. 3(a), the function is called recursively by lines 15-18. Here, because spaces 2 and 4 have not yet been searched (line 8), the search can start from each space's (*min_x, min_y*) (line 9).

Third, if objects are found in spaces 1 or 2 during the search of the overall space in Fig. 3(a), the function is called recursively by lines 15-18. Here, if an object is found in space 1, the search is complete in space 3 to the left of the *current_x* coordinate, and if an object is found in space 2, the search is complete in space 4 to the left of the *current_x* coordinate. Because of this, the search can begin from the *min_y* coordinate of space 3 or 4 (line 6).

Fourth, if objects are found in spaces 3 or 4 during the search of the overall space in Fig. 3(a), the function is called recursively by lines 15-18. Here, if an object is found in space 3, the search is complete in space 1 up to the *current_x* coordinate, and if an object is found in space 4, the search is complete in space 2 up to the *current_x* coordinate. Because of this, the search can begin from the *min_y* coordinate of space 1 or 2 by moving the *temp_x* coordinate one pixel to the right (line 7).

Figs. 3(b), 3(c), and 3(d) show examples of searching spaces with a quadtree that has one tree level. In Figs. 3(c) and 3(d), space/node 1 corresponds to the fourth case, and spaces/nodes 2 and 4 correspond to the second case.

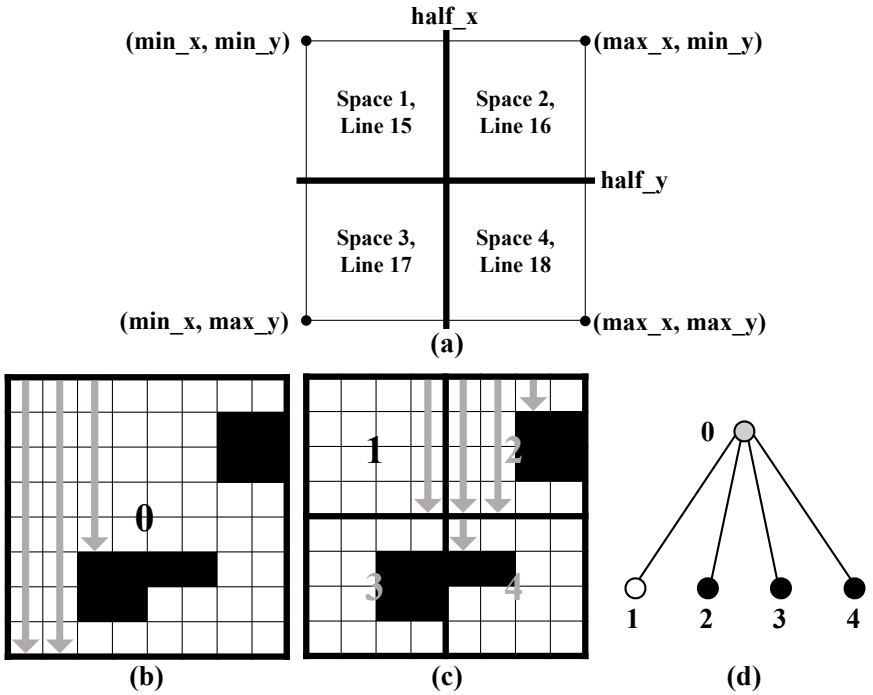

Fig. 3. Quadtree-based spatial object search scheme.

If the *quadtree_search()* function is executed, the quadtree information shown in Fig. 4 is logically converted into a one-dimensional array of Boolean data. The one-dimensional array saves the tree as a full tree, and GRAY nodes and BLACK nodes are not distinguished in the proposed scheme. Because of this, if the *i* index's value is *true* in the one-dimensional array and the *i*4+n* (*n* being a natural number from 1 to 4) index is searched, the next step's space can be searched. Also, if the one-dimensional array's size is less than *i*4+n*, it means that *i* is a leaf node. Therefore, the approximate areas of searched objects can be calculated by adding all of the spatial areas of the values for which the index value that corresponds to a leaf node in the one-dimensional array is *true*. Fig. 5 shows the algorithm that calculates the area of the searched objects.

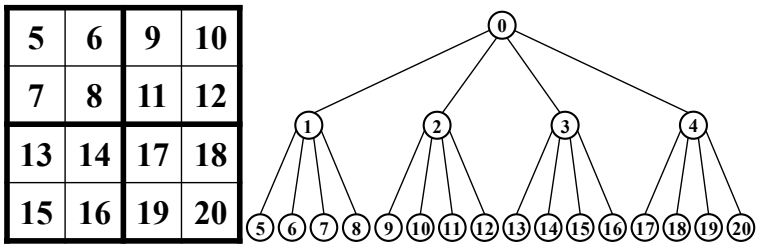

Fig. 4. Full tree.

Pseudo code of area_calc()

```
    Global variable area ← 0
    Input  : level, index
    Output: area
1   if object_grid[index] is false then return
2   else if object_grid[index] is true then
3      if level is equal to the setting level then
4         area += area corresponding to the index
5      else if level is lower than the setting level then
6         area_calc(level+1, index*4+1)
7         area_calc(level+1, index*4+2)
8         area_calc(level+1, index*4+3)
9         area_calc(level+1, index*4+4)
```

Fig. 5. Area calculation algorithm.

4. Implementation and Experiments

In order to import images as objects in a UI environment, Java was used to create an experimental program. The experimental system was equipped with an Intel i7-5960X at 3.00GHz and 16 GB of memory.

Objects are inputted into the white box on the left side of Fig. 6, and the Convert button in the middle is clicked to search for objects. The objects' information is saved in the quadtree, and the objects' distribution is drawn as a grid and outputted in the white box on the right side. The experiments were performed in a space with a size of 512*512. The quadtree levels that are used in the search are determined by the value of the combo box at the top left, and 1 to 7 levels can be set. The area is calculated by the algorithm that was previously introduced in Section 3, and it is outputted at the top right. It was assumed that a 0.001 second delay occurs when searching each pixel, and a sleep function was implemented. This is because it is difficult to make a comparison when the time taken to search each level is similar to within 0.004 seconds if there is no delay in the experiment environment.

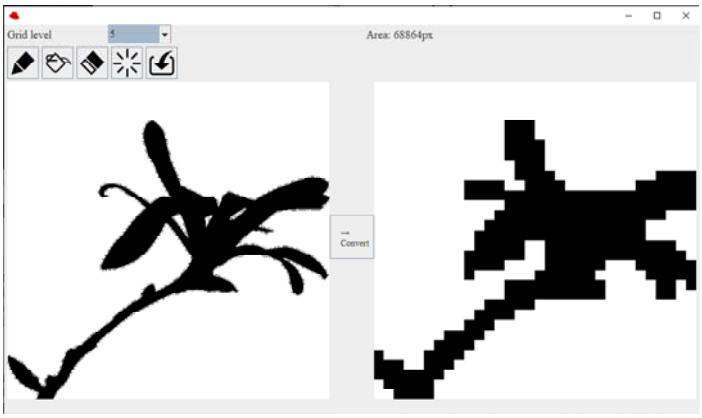

Fig. 6. An App implemented for searching a spatial object.

The fifth icon at the top left can be clicked to import images. In the experimental environment, the transparent pixels, which have an RGBA transparency level of 0, are converted to *false*, and the nontransparent pixels, which have a transparency level of 1-255, are converted to *true*. The *true* pixels, i.e. objects, are shown in black within the white box on the left. Also, by clicking the third icon from the left, objects can be drawn in the white box on the left. The fourth icon can be used to clear the objects.

In the search's best case scenario, objects exist in the first pixel at the top left of all segmented spaces. Fig. 7 shows the average time taken and the number of pixels that were searched when searching all pixels and searching each level 100 times repeatedly.

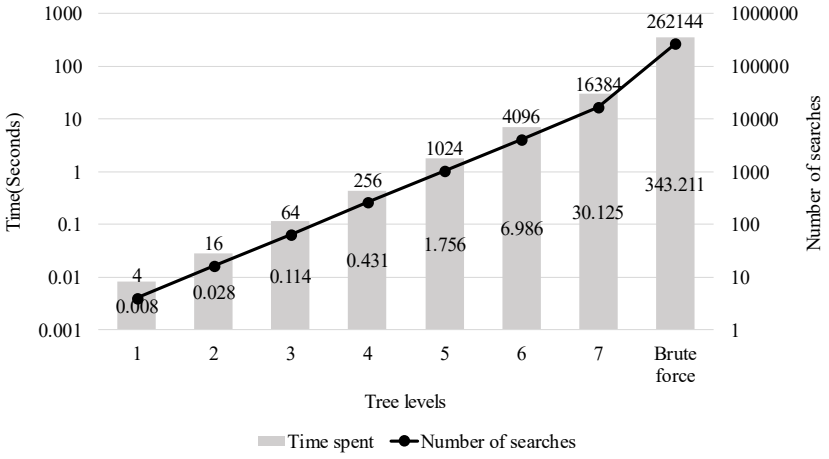

Fig. 7. Average time spent on searches and average number of searches for each tree level.

In the search's worst case scenario, objects do not exist or they are in the last pixel at the bottom right of every segmented space. The average time taken for searching by each level was similar to the brute force search in Fig. 7, and the number of searched pixels was the same at 262,144.

Fig. 8 shows 12 experiment target images that were randomly downloaded from Pixabay [5]. In the experiments, searches were performed on each image 100 times for each tree level, and the time taken by the search and the number of searched pixels were recorded.

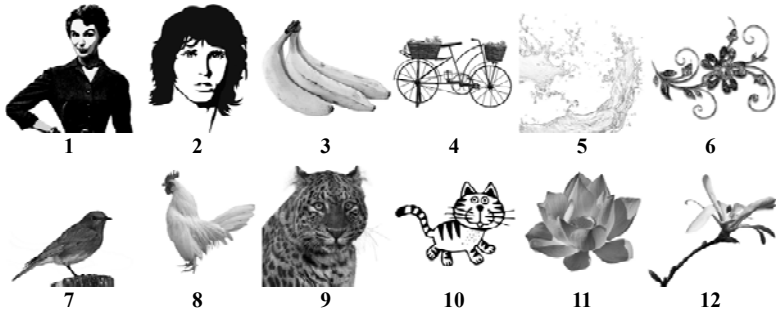

Fig. 8. Experiment images.

Fig. 9 shows the average time spent searching each image and the average number of searches for each tree level. As the tree levels increased, the search became more elaborate, and it can be seen that the search time and the number of searched pixels increased. Despite this, the level 7 tree search was completed about 100 seconds faster than the brute force search in Fig. 7.

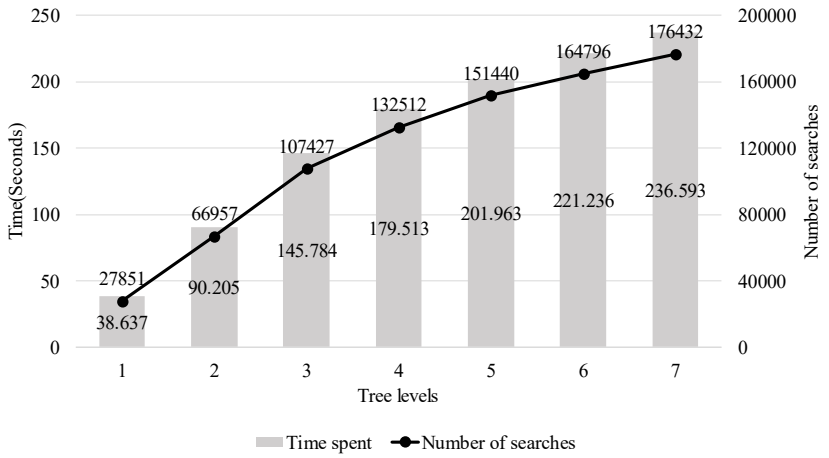

Fig. 9. Average time spent on searches and average number of searches for each tree level.

Fig. 10 shows the average time and average number of searches to search each tree level in each image. When the objects were distributed widely or evenly as in image 3 or image 11, the search time was short. When there were many spaces without objects as in image 1, image 2, or image 6, the search time was long. This occurred because the proposed search algorithm searches sequentially from the top left to the bottom right. This phenomenon can be understood in the context of the proposed algorithm having the same disadvantage that occurs in sequential search algorithms, i.e. that the time complexity is O(N) in the worst case scenario, and the algorithm becomes more inefficient as the size of the array becomes larger [Parmar *et al.*, 2015].

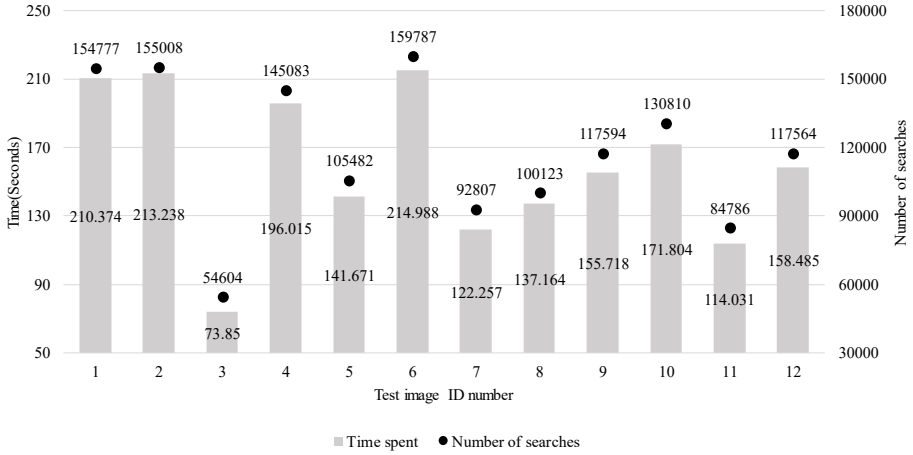

Fig. 10. The average time spent on searches and average number of searches for each test image.

The experiments showed that in the worst case, the search was performed for a similar time and number of pixels as the brute force search. However, in the best case, if the number of searched tree levels was *l*, the objects' distribution could be known by searching 4^*l* pixels, and the difference in computation time became larger as the number of levels became smaller. Furthermore, it was possible to find the approximate distribution of the objects more quickly than the brute force search in the experiments using the images from Fig. 8. Therefore, it can be seen that the proposed method is more efficient in terms of the search time and number of searches than the brute force search when finding the approximate distribution of objects.

5. Conclusions and Future Research

This paper has proposed a scheme that uses a quadtree structure to search for and save objects in order to quickly search for the approximate location of objects in a certain space. Experiments have shown that the proposed method can find the approximate distribution and area of objects without searching an entire *n*m* space that has *n* number of horizontal pixels and *m* number of vertical pixels. Future studies will first extend the size of the space to FHD and test the search time and number of searches. Second, we aim to gain the algorithm's performance improvement by applying other search method.

Acknowledgments

This paper was supported by the National Research Foundation of Korea (NRF) Grant funded by the Korea Government (Ministry of Education) [NRF-2017R1D1A1B03035884, NRF-2018R1D1A1B07045642].

References

[1] Parekh, H.S.; Thakore, D. G.; Jaliya, U. K. (2014): A survey on object detection and tracking methods. International Journal of Innovative Research in Computer and Communication Engineering, 2 (2), pp. 2970-2978.
[2] Liu, L.; Ouyang, W.; Wang, X.; Fieguth, P.; Chen, Liu, X.; Pietikäinen, M. (2019): Deep learning for generic object detection: A survey. International Journal of Computer Vision, 1-60.
[3] Samet, H. (1985): Data structures for quadtree approximation and compression. Communications of the ACM, 28 (9), pp. 973-993.
[4] Samet, H. (1984): The Quadtree and related hierarchical data structures. ACM Computing Surveys (CSUR), 16 (2), 187-260.
[5] Pixabay is available on https://pixabay.com.
[6] Parmar, V. P.; Kumbharana, C. K. (2015): Comparing linear search and binary search algorithms to search an element from a linear list implemented through static array, dynamic array and linked list. International Journal of Computer Applications, 121 (3), pp. 13-17.