International Research Journal of Engineering and Technology (IRJET) e-ISSN: 2395-0056
Volume: 09 Issue: 04 | Apr 2022 www.irjet.net p-ISSN: 2395-0072

Finding the Relational Description of Different Objects and Their Importance in a Scene

Sabyasachi Moitra[1], Sambhunath Biswas[1,2]

[1]Dept. of Comp. Sc. & Engg., Techno India University, West Bengal, India
[2]Ex-Indian Statistical Institute, Kolkata, India

Abstract - *A scene is composed of many objects. It is needless to say, that these objects have some relational description between them. The relative positions of different objects in a scene define the relational description between its near and far objects with respect to an observer. Such a relational description is not only significant from different perspectives but also is important in many useful applications. In this paper, we propose two different methods to find the relative positions of objects in a scene and based on this information, a hierarchical description or a tree structure is generated. This structure has an immense role in scene estimation and processing of various applications. One of the methods considers simply the Euclidean distance between the image baseline and different objects in the scene. The other method computes the distance considering the depth map of the objects. To study the superiority of the methods, we have made a comparison between them. It is seen that the first method is simpler and faster compared to the second one. We also determine the weights of different objects based on their hierarchical description, which may find an immense role in determining the importance of various objects present in the scene.*

Key Words: Object detection, Object position, Object hierarchy, Object weights.

1. INTRODUCTION

Object detection is a well-known problem in computer vision community to identify and locate objects in a static or dynamic scene, such as a video. The technique draws bounding boxes around detected objects, allowing us to determine the object location and the object class in a given scene. The problem has widespread applications; some of them include self-driving cars, video surveillance, crowd counting, etc. Object detection methods can be divided in two categories that use (i) classical computer vision techniques and the (ii) modern or deep learning-based techniques.

Classical computer vision techniques extract features from an image to identify an object. This finds applications favouring methods described in Viola-Jones [1], HOG [2]. The features for objects are fed into a pre-trained classifier, such as SVM, for prediction of the objects' classes. A sliding window at different positions in images can be used to extract features. These features may or may not correspond to an object at a particular position. In other words, the sliding window locates positions of different objects. On the other hand, in deep learning-based techniques, a deep convolutional neural network [3][4] is used to extract features from an image to classify and localize an object, such as in R-CNN [5], Faster R-CNN [6], and YOLO [7]. For classification and localization of objects in images, these features are fed into a sequence of fully-connected layers or convolutional layers. A convolutional neural network is made up of a series of convolutional and max-pool layers.

To analyze a given scene, the relationships between the detected objects in the scene must be known. This means the relative position of each object with respect to an observer as well as other detected objects are made known (e.g., the current position of each sprinter in a sprint). This paper, presents two methods for determining such relative position of detected objects. This relative positional structure provides a hierarchical description of objects.

The hierarchy has a significant impact in different applications. This is ensured through different weights attached to different detected objects. The weights are computed based on their relative positions in the scene.

2. CAMERA-OBJECT DISTANCE

To find the camera-object distance, we assume that a camera is placed on the z-axis and is horizontal. This is a usual practice. One can have some idea about the camera positioning as referred to in [8]. Objects can, initially be detected using a cutting-edge object detection method. We have used YOLO [7] in our algorithm. The distance between the camera and detected objects is, subsequently computed. This provides the relative position of detected objects in a scene with respect to an observer. We have proposed two different methods to compute the camera-object distance. The first one is simple in nature, while the second one uses the concept based on depth map described in [9].

Method-1:

Method-1 uses an RGB image (scene) as input as shown in Figure 1 and detects objects as indicated in it. It computes the camera-object distance D using the orthogonal distance between the image baseline and the detected objects. The related algorithm is stated below.

Algorithm

STEP-1: Detect objects in a scene, S using the YOLO model,

$$S_{OBJ} = \text{YOLO}(S) = S\left(\left[x_1^i, y_1^i, x_2^i, y_2^i, c^i\right]_{i=1\ to\ n}^T\right), \quad (1)$$

where $\left[x_1^i, y_1^i, x_2^i, y_2^i, c^i\right]^T$ is a vector in which $(x_1^i, y_1^i, x_2^i, y_2^i)$ are the ith bounding box coordinates, i.e., location $((x_1^i, y_1^i)$ and (x_2^i, y_2^i) are the top-left and bottom-right corners respectively) and c^i is the class of the ith object detected.

STEP-2: For each object o_i in S_{OBJ}:

STEP-2.1: Draw a perpendicular p_i from the bottom-right corner of its bounding box (x_2^i, y_2^i) to the baseline of S_{OBJ}.

STEP-2.2: Calculate the length of $p_i(l_{p_i})$ by computing the Euclidean distance between (x_2^i, y_2^i) and the point perpendicularly located on S_{OBJ}'s baseline (x_2^i, Y_2),

$$l_{p_i} = d(u, v) = \sqrt{\sum_{j=1}^{2}(u_j - v_j)^2}, \quad (2)$$

where $d(u, v)$ is the Euclidean distance between two points u and v with $u = (x_2^i, y_2^i)$ and $v = (x_2^i, Y_2)$ ($Y_2 \rightarrow$ bottom-right corner of S_{OBJ}).

STEP-3: Normalize the perpendicular distances computed in STEP-2,

$$D = \{d: d = \frac{l_{p_i}}{\sum_i l_{p_i}}, 1 \le i \le n\}. \quad (3)$$

STEP-4: Compute the position of objects based on the computed D in STEP-3 and get S_{OBJ}^{NEW}.

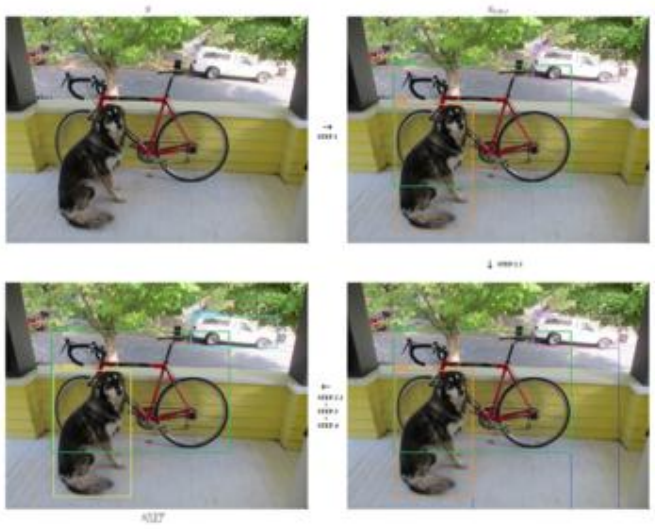

Fig -1: The relative positions of objects in a scene with respect to the observer using Method-1.

Method-2 (Depth Map-Based):

Depth map-based method has its framework in [10]. It also inputs the same RGB image (scene) and detects objects as shown in Figure 2, but it computes the camera-object distance D by computing the average of the depth map image of the detected object.

Algorithm

STEP-1: Detect objects in S using the YOLO model,

$$S_{OBJ} = \text{YOLO}(S) = S\left(\left[x_1^i, y_1^i, x_2^i, y_2^i, c^i\right]_{i=1\ to\ n}^T\right), \quad (4)$$

where $\left[x_1^i, y_1^i, x_2^i, y_2^i, c^i\right]^T$ is a vector in which $(x_1^i, y_1^i, x_2^i, y_2^i)$ are the ith bounding box coordinates, i.e., location $((x_1^i, y_1^i)$ and (x_2^i, y_2^i) are the top-left and bottom-right corners respectively) and c^i is the class of the ith object detected.

STEP-2: Resize S to $h \times w$,

$$S' = S \downarrow_{h \times w}^{H \times W}, \quad (5)$$

where $h = w = 128$.

STEP-3: For each object o_i in S_{OBJ}, map $(x_1^i, y_1^i, x_2^i, y_2^i)$ on S'using [11],

$$x'^i_j = x_j^i * (w/W),\ y'^i_j = y_j^i * (h/H),$$
$$S'_{OBJ} = S'\left(\left[x'^i_1, y'^i_1, x'^i_2, y'^i_2, c^i\right]_{i=1\ to\ n}^T\right), \quad (6)$$

where $j = \{1,2\}$.

STEP-4: Convert S' to grayscale (G) using the weighted/luminosity method [12],

$$G = (0.299 \times R_{S'}) + (0.587 \times G_{S'}) + (0.114 \times B_{S'}), \quad (7)$$

where $R_{S'}$, $G_{S'}$, and $B_{S'}$ are the red, green, and blue values of each pixel in S', respectively.

STEP-5: Compute the albedo (surface reflectivity of an object) (ρ) and illumination direction (I) from G using [13],

$$\rho = \frac{\sqrt{6\pi^2\mu_2 - 48\mu_1^2}}{\pi},$$
$$I = [cos\,\tau\,sin\,\sigma\,, sin\,\tau\,sin\,\sigma\,, cos\,\sigma], \quad (8)$$

where

$$\mu_1 = E\big(G(x,y)\big),$$
$$\mu_2 = E\big(G^2(x,y)\big),$$
$$\tau = tan^{-1}\frac{G_y}{G_x}, \text{ and} \quad (9)$$
$$\sigma = \cos^{-1}\frac{4\mu_1}{\sqrt{6\pi^2\mu_2 - 48\mu_1^2}}$$

($E\big(G(x,y)\big) \rightarrow$ average of the image brightness (pixel intensity), $E\big(G^2(x,y)\big) \rightarrow$ average of the image brightness square, $\tau \rightarrow$ tilt, $\sigma \rightarrow$ slant, $G_x \rightarrow$ image's spatial gradient in x direction, $G_y \rightarrow$ image's spatial gradient in y direction).

STEP-6: Construct the depth map Z from G using [10],

$$Z(x,y) = Z(x,y) - \frac{f(Z(x,y))}{\frac{df(Z(x,y))}{dZ(x,y)} + \varepsilon}, \quad (10)$$

where

$$f(Z(x,y)) = G(x,y) - max\left(0, \frac{1+pi_x+qi_y}{\sqrt{(1+p^2+q^2)}\sqrt{(1+i_x^2+i_y^2)}}\right),$$

$$\frac{df(Z(x,y))}{dZ(x,y)} = \frac{(p+q)(1+pi_x+qi_y)}{\sqrt{(1+p^2+q^2)^3}\sqrt{(1+i_x^2+i_y^2)}} - \frac{(i_x+i_y)}{\sqrt{(1+p^2+q^2)}\sqrt{(1+i_x^2+i_y^2)}}, \quad (11)$$

$$p = Z(x,y) - Z(x-1,y),$$
$$q = Z(x,y) - Z(x,y-1),$$
$$i_x = cos\,\tau\, tan\,\sigma, \text{ and}$$
$$i_y = sin\,\tau\, tan\,\sigma.$$

Initialize $Z(x,y) = 0$ and repeat the step k times. To avoid dividing by 0 a small number ε is added to the denominator of (10).

STEP-7: Compute the average depth of objects $Z_{OBJ} = \{z_1, z_2, \dots, z_n\}$ from Z using the bounding box coordinates obtained in STEP-3,

$$z_i = E\left(Z\left(\left(x'^i_1, x'^i_2\right), \left(y'^i_1, y'^i_2\right)\right)\right), \quad (12)$$

where $E\left(Z\left(\left(x'^i_1, x'^i_2\right), \left(y'^i_1, y'^i_2\right)\right)\right)$ is the average depth of the ith object with box coordinates $(x'^i_1, y'^i_1, x'^i_2, y'^i_2)$.

STEP-8: Normalize Z_{OBJ},

$$D = \{d: d = \frac{z_i}{\sum_i z_i}, 1 \le i \le n\}. \quad (13)$$

STEP-9: Compute the position of objects based on the computed D in STEP-8 and get S_{OBJ}^{NEW}.

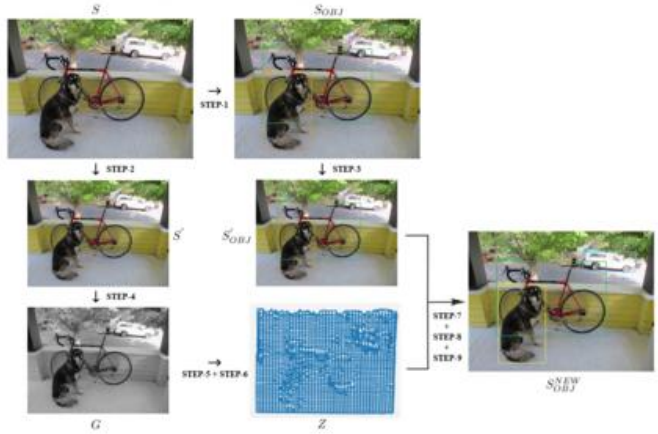

Fig -2: The relative positions of objects in a scene with respect to the observer using Method-2.

3. PROPOSED OBJECT HIERARCHY

In this section, we formulate the hierarchy of detected objects in a scene using objects' relative position computed in Section 2 with respect to an observer, i.e., the camera-object distance D, considering $S(= S_{OBJ}^{NEW})$ as the root and the objects (o_1, o_2, ..., o_n) as the non-root nodes of a tree. The root node resides at the top of the hierarchy (level L_0), whereas the non-root nodes are at different lower levels (L_1, L_2, ..., L_m) based on the camera-object distance information. Objects within a distance d'_{L_1} from the root level L_0 are in level L_1, while objects within a distance d''_{L_2} but greater than d'_{L_1} are in level L_2. Thus, all the objects with an arbitrary distance d satisfying $d'_{L_1} < d \le d''_{L_2}$ are in level L_2, and so on. Objects in a particular level are positioned from left to right in increasing order of camera-object distance. An object o_j at level L_l is a child of a node at its previous level L_{l-1} provided the object has minimum computed distance with respect to this node. All other objects at the level L_l are also the children of the said previous node provided the level L_{l-1} has no other nodes. The same rule holds good for all other nodes.

Algorithm to Create the Hierarchical Tree

STEP-1: **Building levels**

STEP-1.1: Set the root level at $d = 0$.

STEP-1.2: Find the maximum camera-object distance, d_{max}.

STEP-1.3: Set the distance between the levels, d'' and find the number of levels,

$$N_L = d_{max}/d''. \quad (14)$$

STEP-1.4: Select the objects at level L_l with distance d, $d'_{L_{l-1}} < d \le d''_{L_l}$ (e.g., objects of level L_1 satisfies $d'_{L_0} = 0$, $d''_{L_1} = 0.5$ (say)).

STEP-2: **Parent-child allocation**

For all objects at level L_{l-1}, compute their distances with respect to an object at level L_l and find the object for minimum computed distance. The object at level L_l is a child of the object at the level L_{l-1} for which the minimum computed distance is obtained. The process is carried out for all objects at level L_l. If the level L_{l-1} has only one object, then all the objects at L_l are the children of the single object at level L_{l-1}. No child may have two or more than two parents even if they have all equal distances. In this case, the parent will be the first object at level L_{l-1}.

Figure 3 shows the hierarchy of detected objects in the scene as shown in Figure 1.

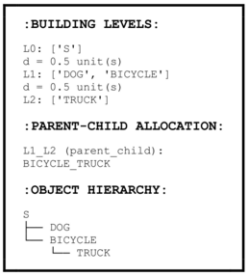

Fig -3: Hierarchy for the scene as shown in Figure 1.

4. COMPUTATION OF WEIGHTS

Depending on the hierarchy of objects in the scene, we now provide a scheme that explicitly computes weights of each object in the hierarchy. The assignment is such that, as the camera-object distance increases, object weight decreases. This means weights of the nearby objects are larger than the far away objects. The algorithm to compute the weights of different objects at different levels in the scene is described below.

Let us suppose we have in the hierarchy, levels $L_1, L_2, ..., L_m$ where L_m is the last level. We further assume that L_1 has $o_1, o_2, ..., o_{k_1}$ objects, i.e., k_1 objects. Similarly, L_2 has k_2 objects, and so on.

We assume the weights of different objects at level L_1 are given by

$$L_1: w_i = \frac{k_1 - (i-1)}{k_1} \cdot \frac{m-0}{m+1} \quad (15)$$
$$i = 1,2, \dots, k_1$$

where k_1 is the number of objects at level L_1 and m is the last level.

Likewise the weights of objects at level L_2 are

$$L_2: w_i = \frac{k_2 - (i-1)}{k_1 k_2} \cdot \frac{m-1}{m+1} \quad (16)$$
$$i = 1,2, \dots, k_2$$

where k_2 is the number of objects at level L_2.

Finally the weights of objects at the last level L_m are given by,

$$L_m: w_i = \frac{k_m - (i-1)}{k_1 k_2 \dots k_m} \cdot \frac{m-(m-1)}{m+1} \quad (17)$$
$$i = 1,2, \dots, k_m$$

where k_m is the number of objects at level L_m.

Note that the generalized formula for computing weights of objects at different levels can be written as,

$$L_i: w_j = \frac{k_i - (j-1)}{k_1 k_2 \dots k_i} \cdot \frac{m-(i-1)}{m+1} \quad (18)$$
$$i = 1,2, \dots, m, j = 1,2, \dots, k_i$$

If the camera-object distance of two consecutive objects is same, their weights are also same.

Algorithm for Computation of Weights

Input: Object hierarchy of a scene, $OH = \{L_0, L_1, L_2, \dots, L_m\}$ $\left(L_1 = \{o_1, \dots, o_{k_1}\}, L_2 = \{o_1, \dots, o_{k_2}\}, \dots, L_m = \{o_1, \dots, o_{k_m}\}\right)$
Output: Object weights, $W = \{w_1, w_2, \dots, w_n\}$

Begin
  $W := \emptyset$
  For $i := 1$ **to** m **Do**
    For $j := 1$ **to** k_i **Do**
      If $j = 1$ **Then**
        $j' := 1$
        $w_j := \frac{k_i - (j'-1)}{k_1 \times k_2 \times \dots \times k_i} \times \frac{m-(i-1)}{m+1}$
        $j' := j' + 1$
      Else
        If camera-object-distance (o_j) = camera-object-distance (o_{j-1}) **Then**
          $w_j := w_{j-1}$
        Else
          $w_j := \frac{k_i - (j'-1)}{k_1 \times k_2 \times \dots \times k_i} \times \frac{m-(i-1)}{m+1}$
          $j' := j' + 1$
        End If
      End If
      Add w_j to W
    End For
  End For
End

Figure 4 shows the weights of detected objects in the scene shown in Figure 1.

:OBJECT WEIGHTS:

OBJECT	W
DOG	0.666667
BICYCLE	0.333333
TRUCK	0.166667

Fig -4: Description of weights for different objects as shown in Figure 1.

5. RESULTS AND DISCUSSION

We have implemented the entire work using Python on a Windows machine with the PROCESSOR of Intel 8th Generation, Core i5 having the RAM capacity of 8GB DDR4. Figure 5 provides the scenes S_1 and S_2 respectively; S_1 contains three different objects, e.g., DOG, BICYCLE and TRUCK, while S_2 contains three different objects, e.g., DOG, PERSON and CHAIR. Table 1 depicts the camera-object distances for different objects in the scene S_1 and S_2. These distances are both for the Method-1 and Method-2. Figure 6 describes the hierarchy for these two scenes using this camera-object distance information.

Fig -5: The scenes.

Table -1: Camera-Object Distance

Scene	Position	Object	*D* units	
			M_1[a]	M_2[b]
S_1				(k[c] = 130)
	1	DOG	0.056027	0.232284
	2	BICYCLE	0.24618	0.23391
	3	TRUCK	0.697793	0.533807
			ET[d]: ≈0.78s	ET: ≈29.07s
S_2				(k = 10)
	1	DOG	0.265152	0.316494
	2	PERSON	0.295455	0.335312
	3	CHAIR	0.439394	0.348194
			ET: ≈0.86s	ET: ≈2.98s

[a]M_1 = Method-1, [b]M_2 = Method-2,[c]k = No. of iterations, [d]ET = Execution Time.

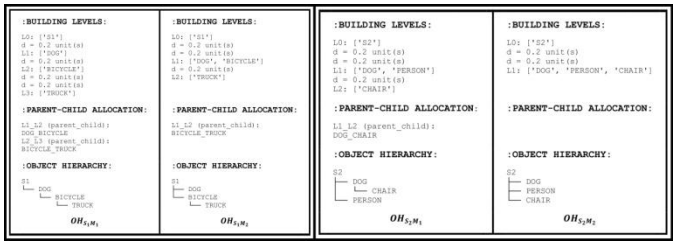

Fig -6: Description of object hierarchies.

Note that $OH_{S_iM_j}(i = 1,2;\ j = 1,2)$ is the object hierarchy for the ith scene of Figure 5, due to the jth method.

The camera-object distance determines the position of objects in the scene. This position of objects in the scene considers the shortest distance as 0 and increases with the increase of camera-object distance. This is clearly depicted in Table 1. However, if two objects have the same camera-object distance then their positions are also the same (e.g., for the first scene in Figure 7). This information determines the nearest and furthest objects in a scene (thus for S_1 in Figure 5, the DOG is the nearest object and the TRUCK is the furthest object). This determines the relative positions of objects in the scene.

The Method-2, on the other hand is not as straightforward as is the Method-1. It computes the depth map (STEP-6 in its algorithm (equation (10) and equation (11))) iteratively. Number of iterations and its execution time are shown in Table 1. Thus, the process of iteration behaves as a tuning parameter. Comparison of these two methods shows that the Method-1 computes the camera-object distance in a much lesser time than the Method-2. The fastness is due to mathematical simplicity over the Method-2.

To construct the object hierarchy we take help of the camera-object distance. It provides a graphical representation of the position of objects in the scene. The tree-like structure with some levels and nodes describes the relative position of objects in the scene. The root node, representing the scene, resides at level L_0, while the non-root nodes, representing objects, resides at non-zero levels of the structure. The level distance is used to create these object levels. Roughly nearby objects belong to the same level. If an object in a level has a minimum distance with respect to an object in the previous level, then it is described as a child of that object. For example, in Figure 6, for the object hierarchy $OH_{S_2M_1}$, two objects, the DOG and the PERSON, are within 0.4 units of level L_0, forming level L_1 of the tree, and only one object, a CHAIR, is within 0.2 units of level L_1, forming level L_2 of the tree. The CHAIR object at level L_2 has a minimum distance with the DOG object of level L_1, so it is described as a child of the DOG object.

Using a level distance of 0.2 units, the object hierarchies $OH_{S_1M_1}$ and $OH_{S_1M_2}$ in Figure 6 for scene S_1 in Figure 5 are fairly reasonable because we can consider and disregard the DOG and BICYCLE as the nearby objects, i.e., both the possibilities can be viewed as logical for the scene. However, the distance between the last and the last but one levels (L_2 and L_1) of $OH_{S_1M_2}$ is 0.2 units, which does not correspond to the scene; it must be more than 0.2 units. For the scene S_2, we can only consider the object hierarchy $OH_{S_2M_1}$ to be reasonably good, not $OH_{S_2M_2}$ because the scene shows that only the DOG and PERSON are almost the close objects, but not the CHAIR. So, based on our previous arguments, we can conclude that the Method-1 for computing camera-object distance is superior to the Method-2.

Figure 7 depicts the objects' relative position in a scene relative to an arbitrary observer, using the Method-1 and its hierarchy based on the computed parameters.

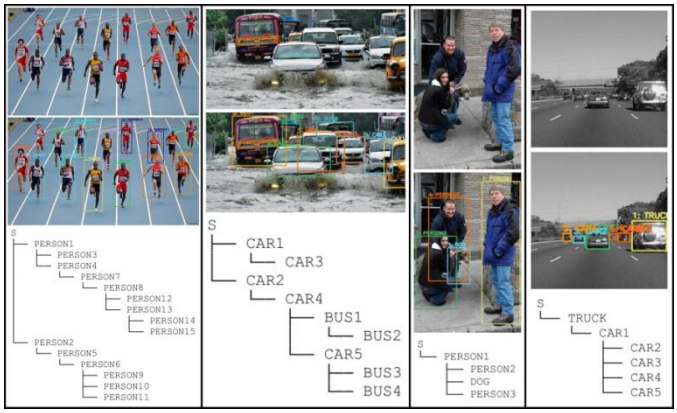

Figure 7: The relative position of objects in a scene with respect to an observer and the corresponding hierarchy of objects.

Object weights play a significant role in analyzing and evaluating a scene. Figure 8 depicts the analysis and evaluation of a scene both without and with using object weights computed from object hierarchy, and Table 2 shows the significance of object weights in a comparative analysis for evaluations.

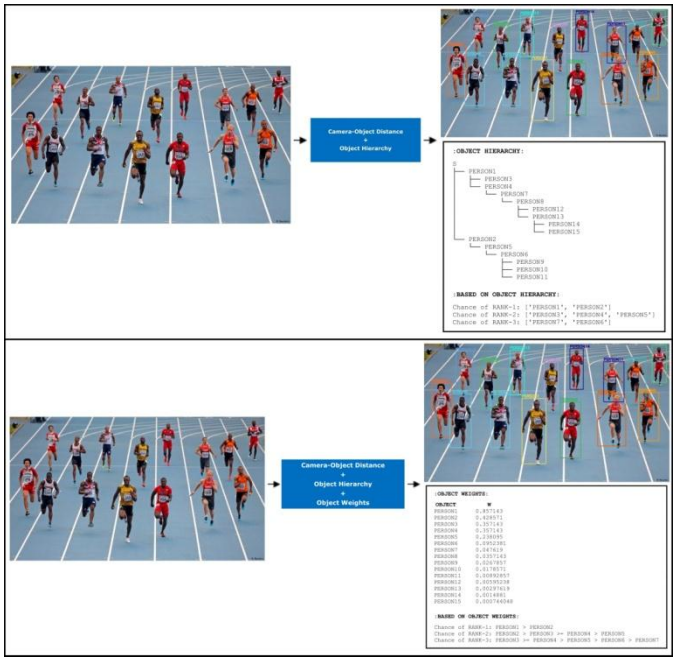

Fig -8: Evaluation of a scene both without (top) and with (bottom) object weights.

Table -2: Comparative Analysis

Problem	Chances of first, second and third place holders in a sprint as shown in Figure 8.	
Solution	**Without Using Object Weights**	**Using Object Weights**
	Interpretation Both PERSON1 and PERSON2 have a chance of winning the first place, PERSON3, PERSON4, PERSON5 the second, and PERSON7 and PERSON6 the third. **Inference** Impossibility for inferencing/decision making for winners.	**Interpretation** PERSON1 has a higher chance of winning the first place than PERSON2. If PERSON1 wins the first place, PERSON2 has a better chance of finishing second than PERSON3, PERSON4, and PERSON5. If PERSON2 takes second place, both PERSON3 and PERSON4 have a higher and equal probability of taking third place than PERSON5, PERSON6, and PERSON7. **Inference** Ability for inferencing/decision making for winners.

6. CONCLUSIONS

In this paper, we find objects' successive positions in a scene, by measuring the distance between an object and an observer (the camera). We have proposed two different methods, in which the first method computes the camera-object distance using the Euclidean distance between the image baseline and the detected objects, and the second method computes the same using the depth map of object. We have also created a hierarchical description of objects in a scene based on this information. This hierarchy is helpful to find the objects relative position in the scene and may find an immense role in analysis as well as in data structure of scenes. The data structure might be helpful in faster processing of scenes. The comparison between the methods shows that Method-1 is superior to Method-2 that uses the depth map of object. We have also computed the object weights from its hierarchical description in the scene. This plays an important role in analysis and evaluation of a scene. Our main objective is to make the whole system more robust and informative, and we shall describe the concerned method in a forthcoming paper.

ACKNOWLEDGEMENT

The authors would like to acknowledge Techno India University, West Bengal for its support to this work.

REFERENCES

[1] Viola P, Jones M. Rapid object detection using a boosted cascade of simple features. In: Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001. vol. 1; 2001. p. I–I.

[2] Dalal N, Triggs B. Histograms of oriented gradients for human detection. In: 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05). vol. 1; 2005. p. 886–893.

[3] Krizhevsky A, Sutskever I, Hinton GE. ImageNet Classification with Deep Convolutional Neural Networks. In: Advances in Neural Information Processing Systems 25. Curran Associates, Inc.; 2012. p. 1097–1105.

[4] Simonyan K, Zisserman A. Very Deep Convolutional Networks for Large-Scale Image Recognition. CoRR. 2015;abs/1409.1556.

[5] Girshick R, Donahue J, Darrell T, Malik J. Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation. In: The IEEE Conference on Computer Vision and Pattern Recognition (CVPR); 2014.

[6] Ren S, He K, Girshick R, Sun J. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. In: Advances in Neural Information Processing Systems 28. Curran Associates, Inc.; 2015. p. 91–99.

[7] Redmon J, Divvala S, Girshick R, Farhadi A. You Only Look Once: Unified, Real-Time Object Detection. In: The IEEE Conference on Computer Vision and Pattern Recognition (CVPR); 2016.

[8] Singh A, Singh S, Tiwari DS. Comparison of face Recognition Algorithms on Dummy Faces. International Journal of Multimedia & Its Applications. 2012 09;4.

[9] Zhang R, Tsai PS, Cryer JE, Shah M. Shape-from-shading: a survey. IEEE Transactions on Pattern Analysis and Machine Intelligence. 1999;21(8):690–706. https://doi.org/10.1109/34.784284.

[10] Ping-Sing T, Shah M. Shape from shading using linear approximation. Image and Vision Computing. 1994;12(8):487–498. https://doi.org/10.1016/0262-8856(94)90002-7.

[11] Cogneethi.: C 7.5 | ROI Projection | Subsampling ratio | SPPNet | Fast RCNN | CNN | Machine learning | EvODN. Available from: https://www.youtube.com/watch?v=wGa6ddEXg7w&list=PL1GQaVhO4f jLxOokW7CS5kY J1t1T17S.

[12] Dynamsoft.: Image Processing 101 Chapter 1.3: Color Space Conversion. Available from: https://www.dynamsoft.com/blog/insights/image-processing/image-processing-101-color-space-conversion/.

[13] S. Y. Elhabian, "Hands on shape from shading," 05 2008.