Weighted Unsupervised Learning for 3D Object Detection

Kamran Kowsari
Department of Computer Science,
The George Washington University, Washington DC

Manal H. Alassaf
Department of Computer Science,
The George Washington University, Washington DC
Department of Computer Science,
Taif University, Taif, Saudi Arabia

***Abstract*—This paper introduces a novel weighted unsupervised learning for object detection using an RGB-D camera. This technique is feasible for detecting the moving objects in the noisy environments that are captured by an RGB-D camera. The main contribution of this paper is a real-time algorithm for detecting each object using weighted clustering as a separate cluster. In a preprocessing step, the algorithm calculates the pose 3D position X, Y, Z and RGB color of each data point and then it calculates each data point's normal vector using the point's neighbor. After preprocessing, our algorithm calculates k-weights for each data point; each weight indicates membership. Resulting in clustered objects of the scene.**

***Keywords*—Weighted Unsupervised Learning, Object Detection, RGB-D camera, Kinect**

I. Introduction

Object detection for unlabeled and unsegmented data points [1] is widely studied. In general, visualization and machine learning are the main issues, which are reviewed in existing studies. Machine learning is classified into two categories; supervised and unsupervised learning. In the first case, many researches have addressed the object detection with supervised methods [2], [3], [4], [5], [6], [7], [8]. Kevin Lai's and many other researchers work on the weighted supervised learning for object detection [9], [10], [11], [12] using a hierarchical, multi views, and sparse distance learning. That method can be useful for known objects intended to be detected. Therefore, his algorithm is used for object detection of a specific item that is stored in a database in a preprocessing step such as an apple, egg, etc. Therefore, when using supervised learning for object detection, researchers focus on the accuracy of object detection and detecting known objects. On the other hand, time is very critical in real-time object detection techniques.

The second category addresses the object detection problem with unsupervised learning method [2], [13], [14], [15], [16], [17], [18], [19], [20], [21] using techniques of unsupervised learning such as K-means and spectral clustering. Most of these techniques are not sufficient for real-time applications since they do not address time complexity and memory consumption. With regard to accuracy of object detection using an RGB-D camera, an efficient method for RGB-D camera is weighted clustering since capturing by this kind of camera has more noise introduced by moving objects; thus, labels need to be updated in a few iterations that span less than a second. If we want to compare the clustering between computer graphics and other domains; data points are not changing during running time in most fields such as data mining, but data points in real-time object detection in the field of computer graphics, machine vision, and robotics are continuously changing; frame by frame and second by second. As regards to Liefeng Bo [14] who uses a dictionary for his work, this technique is very efficient for synchronized video, and also this method is fast enough for video processing with around 2 frames per second (FPS). But, in real-time applications, it needs to be faster than 2 FPS. Weighted Unsupervised Learning such as weighted k-means [22] or many other methods [23], [24], [25], [26], [29], [30] are implemented in different domains. Fuzzy object detection and weighted clustering is addressed and used for moving objects [15], [27], [28]. In 2010, Maddalena et al. [15] had worked on fuzzy logics and learning. That work used only pixel-by-pixel as their features, which can be very efficient for image processing. Thus, those methods never use other sensors such as depth information as a specific feature for 3D, whereas Maddalena's method has many limitations for indoor capturing for 3D object detection. Some other researches use Vicon system as object detection and object controlling for real-time applications, but this device cannot capture color and surface of the objects [31], [32], [33], [34], [35] ; thus, Vicon cannot be implemented as a colorful application; however, it can be an efficient method for rigid object detection that could calculate object position as the only feature. Therefore, an unsupervised and RGB-D camera method that addresses the accuracy, time-complexity, and memory consumption and colorful surface capturing simultaneously is unprecedented.

Visualization is an important step in object detection for evaluating the results. Graphical Processing Unit (GPU) based application is a powerful method in the computer graphic domain, but hence not applicable in mobility applications. In general, GPU-based application needs powerful hardware; therefore, for mobility application, researchers focus more on Central Processing Unit (CPU) based applications.

This paper is an extension of the authors' prior work in [13] where we used RGB-D camera and studied the Boolean version of clustering. The color space from RGB was improved to Hunter-Lab color space [36], [37] where the Hunter-Lab color space gave smoother result of clustering in real-time application. In this paper, we use the RGB color space. We improved the clustering part by adding k weights to each

Fig. 1: *a*) Kinect color frame (RGB) with resolution of 1920 X 1080; *b*) Kinect depth frame with resolution of 512 X 424; *c*) Proposed method object detection using k= 15 clusters, and after 15 iterations.

data point. This improvement affects frame rate, accuracy, and memory consumption in the real-time application. We propose a real-time object detection algorithm using k weighted clusters with memory consumption that is useful for mobility application. The maximum memory needed for this algorithm is 650 MB for 50 clusters, less than one GB for 100 clusters, and we use multi-thread processing to improve frame rate.

In short, new contributions and unique features of the proposed method in this paper are as follows. 1. Weighted unsupervised learning is presented, which reduces noise for moving and small objects, has better time complexity, lower memory consumption, and higher accuracy than previous methods, 2. CPU-based implementation is offered to make our method capable of mobility usage, and 3. A segmentation technique is proposed to detect a user defined object.

II. Pipeline and Methods

The pipeline of weighed unsupervised object detection algorithm presented in this paper is illustrated in Figure 2 and composed of 8 steps as follows:

1) Capturing RGB color and depth information;
2) Mapping RGB and depth information;
3) Applying back-projection for generating cloud points in the 3D world coordinate system;
4) Calculating data points normal vectors based on each point neighbors;
5) Segmenting and removing the background to limit the area where we want to detect the objects;
6) Calculating distance between each point and the k-centers of the clusters;
7) Updating the k-weights of each cloud point; and
8) Assigning color to each data point by using the point's k-weights.

This paper is organized as follows: preprocessing is presented in section II-A followed by clustering step in section II-B. Section II-C talks about visualization. Finally, numerical and experimental results are presented in section III. Followed by discussion in section IV and finally the conclusion and future work is in section V.

A. Preprocessing

Preprocessing is used for generating the 3D cloud points from input data that is captured by an RGB-D camera and needed for clustering steps; thus, in preprocessing steps, the algorithm generates cloud points by pose-the 3D as XYZ, color as RGB, and normal as n_x, n_y, n_z. In this research paper, we use Kinect for Windows V2 as an RGB-D camera. In short, preprocessing steps are: get input, perform frames mapping, back projection, and normal calculation.

1) Get Input: The Kinect camera was designed as a hands-free game controller. It has two input sensors, which include an RGB camera with a resolution of 1920 X 1080 pixels, and a depth sensor with the resolution of 512 X 424 pixels. Field of View (FOV) is 84.1 X 53.8 for RGB color space and 70.6 X 60 degree for depth sensor information. The resulting average is about 22 X 20 pixels per degree for RGB and 7 X 7 pixels per degree of depth data [40], [41]. Kinect can capture depth information of objects displaced up to 4.5-5 meters from the camera, but we can limit it manually between a and b meters for indoor object detection [38], [39].

2) Frames Mapping: The two inputs' frames of Kinect have dissimilar resolutions. The mapping is needed for matching color space and depth information in the same space. Our approach is like other studies [42], [43] for generating colored cloud points. After mapping, we have aligned frames with information about the position x, y, z; and color R, G, and B for each data point. These data points will be ready for the next step, back-projection [42], [43].

3) Back Projection: n this step we convert the cloud points into 3D world coordinate system. The Kinect input from mapping step includes x, y, and z parameters. We follow equations 10 through 14 that are demonstrated in the appendix to back-project the points into the world coordinate System. In summary, to generate an accurate 3D position of each data point in the color frame, the 2D position is back-projected using the depth data from the depth frame as z and the Kinect camera intrinsic parameters to obtain the correct 3D position of each cloud point in the real world coordinate system.

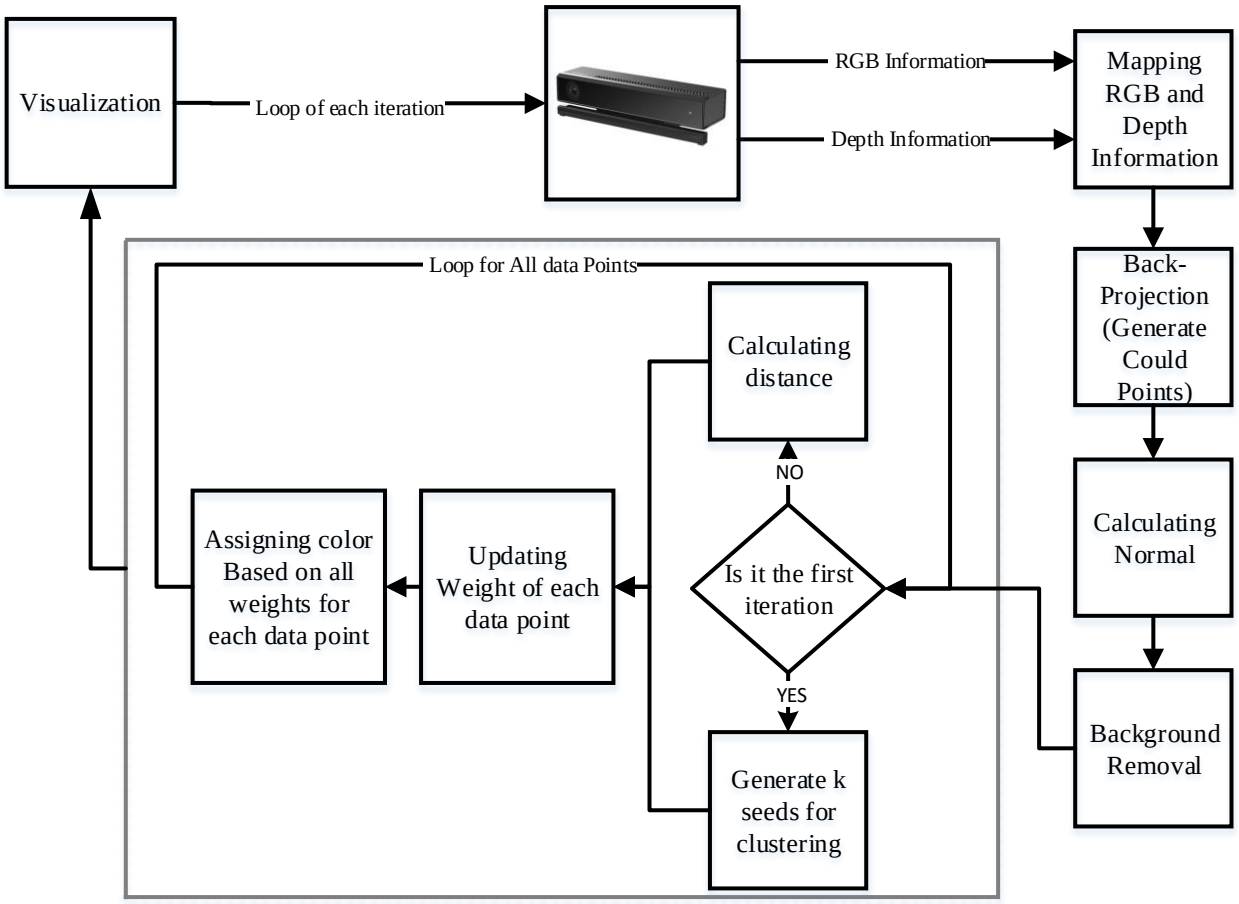

Fig. 2: Pipeline of 3D Object detection using RGB-D camera has two main parts: 1) Preprocessing including Mapping, Back-Projection, Normal Generating, Background removal and 2) Clustering including assigned initial weight, distance calculation, update weight and assign color, and finally visualization to illustrate the results.

4) Normal Calculation: In this step, normal vector of each cloud point is calculated as new feature indicated by n_x, n_y, n_z. The normal of each data point is calculated using its neighbors.

$$p_i = (x\ y\ z\ r\ g\ b\ n_x\ n_y\ n_z)^T. \quad (1)$$

B. Clustering step

The aim of this paper is object detection using weighted unsupervised learning. In our approach, we use k-means clustering algorithm with k-weights for each data point, where k is the number of clusters that is defined by use. The clustering step is divided into two main parts; initial seeding, and updating the weights. First part in the first iteration is initialization of k-seeds for the k-means algorithm, and initialization of k-weights for each data point. We use k-means++ [44], [49] algorithm to obtain the initial seeds of the k-means clustering, while the k-weights for each data point are initialized to zero. The second part is updating the k-weights [22]. Weights are going to be updated at each of the following iterations. At each iteration, each data point will have k values which indicate a membership for one of the k clusters. At the end of each iteration, each point will belong to the cluster that has the highest weight. The algorithm details are given in equation 1, where C_i is the color of each cluster that is assigned at the beginning as unique color for all clusters. The μ_i is k weights of each point, and will continually be changing by each iteration and getting new values. Clustering step includes the labeling, updating the k weights for each data point, and finally visualization is utilized to illustrate the results.

1) Labeling: In the adopted weighted unsupervised learning, each data point has a k-values. At beginning of the running time, all weights (μ_p) values are equal to zero for all data points. After the first iteration, the algorithm starts to update the k weights of each data point using the formula in equation 2. According to equation 2, δ_i is the previous weight of the cluster i resulting from the previous frame, which will be updated using scale of weight Ψ to update all memberships.

$$\delta_i = \delta_i + \Psi \ \ where \ \ 0 < \Psi \leq 1. \quad (2)$$

Fig. 3: a) Kinect color frame (RGB) with resolution of 1920 X 1080; b) Kinect depth frame with resolution of 512 X 424; c) Proposed method object detection using k= 7 clusters, and after 10 iterations. Memory consumption is 320 MB and frame rate is 8.1 $\pm$0.2 FPS.

Fig. 4: a) Kinect color frame (RGB) with resolution of 1920 X 1080; b) Kinect depth frame with resolution of 512 X 424; c) Proposed method object detection using k=10 clusters, and after10 iterations. Memory consumption is 340 MB and frame rate is 6.5$\pm$0.2 FPS.

For each iteration, we update the weights of a data point by equations 2 and 3 where δ_i is weigh before normalization, and τ is the number of iterations. The increasing rate of τ depends on frame rate that is addressed in section III and Figure 7. Therefore, iteration number is increasing around fps_t per second.

$$\tau = \sum_{s=1}^{t} fps_t \tag{3}$$

Equation 4 presents a distance function of the clustering step, Euclidean distance between data points is used as similarity measure in many clustering algorithms including k-means. To consider the color difference of these points while clustering the data point of each frame, the RGB value is incorporated in the Euclidean distance between any two point's v_i and v_c as addressed in the following equation 4, where the scales α and δ insure that geometric distance and the color distance between two points are in the same order of magnitude. Experimentally, the best value of α is between 0.002 and 0.1. The scale of position is denoted by δ that is calculated from α using equation 5. By this equation, we define our new hybrid similarity measure as a function f of two terms; one in the Euclidean distance, $dist$, between two points v_i and v_c which are data point and centroid information, respectively.

$$dist(v_i, v_c) = \sqrt{\begin{pmatrix} \delta^2 \begin{pmatrix} (X_i - X_c)^2 + \\ (Y_i - Y_c)^2 + \\ (Z_i - Z_c)^2 \end{pmatrix} + \\ \alpha^2 \begin{pmatrix} (R_i - R_c)^2 + \\ (G_i - G_c)^2 + \\ (B_i - B_c)^2 \end{pmatrix} \end{pmatrix}}, \tag{4}$$

$$\delta + \alpha \leq 1 \;\; where \;\; 0 < \alpha < 1. \tag{5}$$

After calculating the distance between a point and a cluster centroid using their position and color as two different features, we need to incorporate the effect of normal vectors difference between them; thus, for the Euclidean distance, $dist$, between

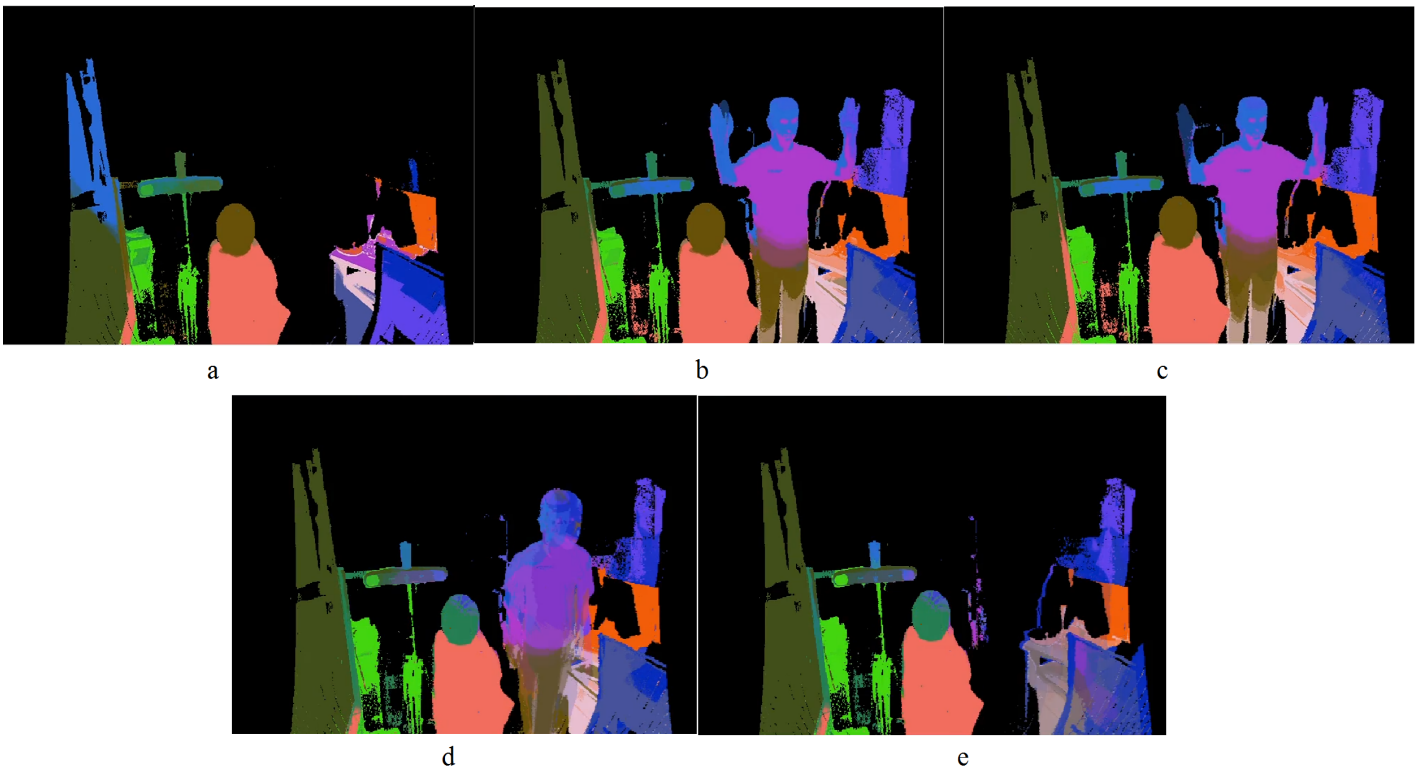

Fig. 5: a) Shows clustered output after 10 iterations and using k=13 clusters; b) Shows a moving object entering the scene. After several iterations the algorithm succeeded in distinguishing the moving object as new object; c) Shows moving hand, where the algorithm successfully detected the moving object; d) Shows a moving object going out of the scene and how the weight of that object was reduced only after one iteration, and finally; e) Shows clustered output after several iterations as detected objects.

two points v_i and v_c which denoted each data point and centroid respectively. The angle between normal of the centroid and normal of each data point is denoted by θ. γ indicates the scale of normal for distance function. The best value of scale of normal, γ, in our experiment was found to be between 0.0001 and 0.01. The final similarity measure between the data points and the centroids that examine the similarity between their positions, colors and normal vectors is given by the following equation 6.

$$f(v_i, v_c) = dist(v_i, v_c) + \gamma(1 - \cos(\theta(n_i - n_c))). \quad (6)$$

2) Update weight: the algorithm updates the weights according to equation 2 and normalizes them for each data point according to equations 7 and 8. Then, the algorithm assigns each data point the label of the cluster with the highest weight. With regard to equations 7 and 8, $(\mu_p i)$ denotes the weight of cluster i of data point p and τ is the number of iterations which are calculated by the equation 3. The summation of k weights of each data point after normalization could be less than or equal to one.

$$\mu_p = \frac{\sum_{i=0}^{\tau}(\delta_i)}{\|\sum_{i=0}^{\tau}(\delta_i)\|}, \quad (7)$$

where

$$\sum_{i=0}^{k} \mu_p^i \leq 1. \quad (8)$$

C. Visualization

After calculating all k weights for all data points at each iteration, we illustrate the results as an object detection or segmentation by assigning each cluster a unique color C_i. Equation 9 indicates assigning one of the k colors to each data point based on its weight μ. According to equation 8, the summation of all weights for data point is less than or equal to one, where μ is the weight of each cluster in a data point. That means, if we have a data point with k different weighted labels, the color of it is assigned by following equation:

$$C_p = \sum_{i=0}^{k} (C_i \star \mu_i) \quad (9)$$

As regards to Algorithm 1, it has one main loop that contains the iteration of our system, the second loop contains clustering step that calculate and update weights. For each data points, the algorithm calculates its distance with respect to all centroids using RGB as color space, XYZ as points point position and n_x, n_y, n_z as normal vector. After that algorithm updates the k weights of each data points after each iteration. Initially in the first iteration, k seeds are obtained by k-means++ and all weights labels are initialized equal to zero. For color assignment, all k weight of each point are used to give the point a label.

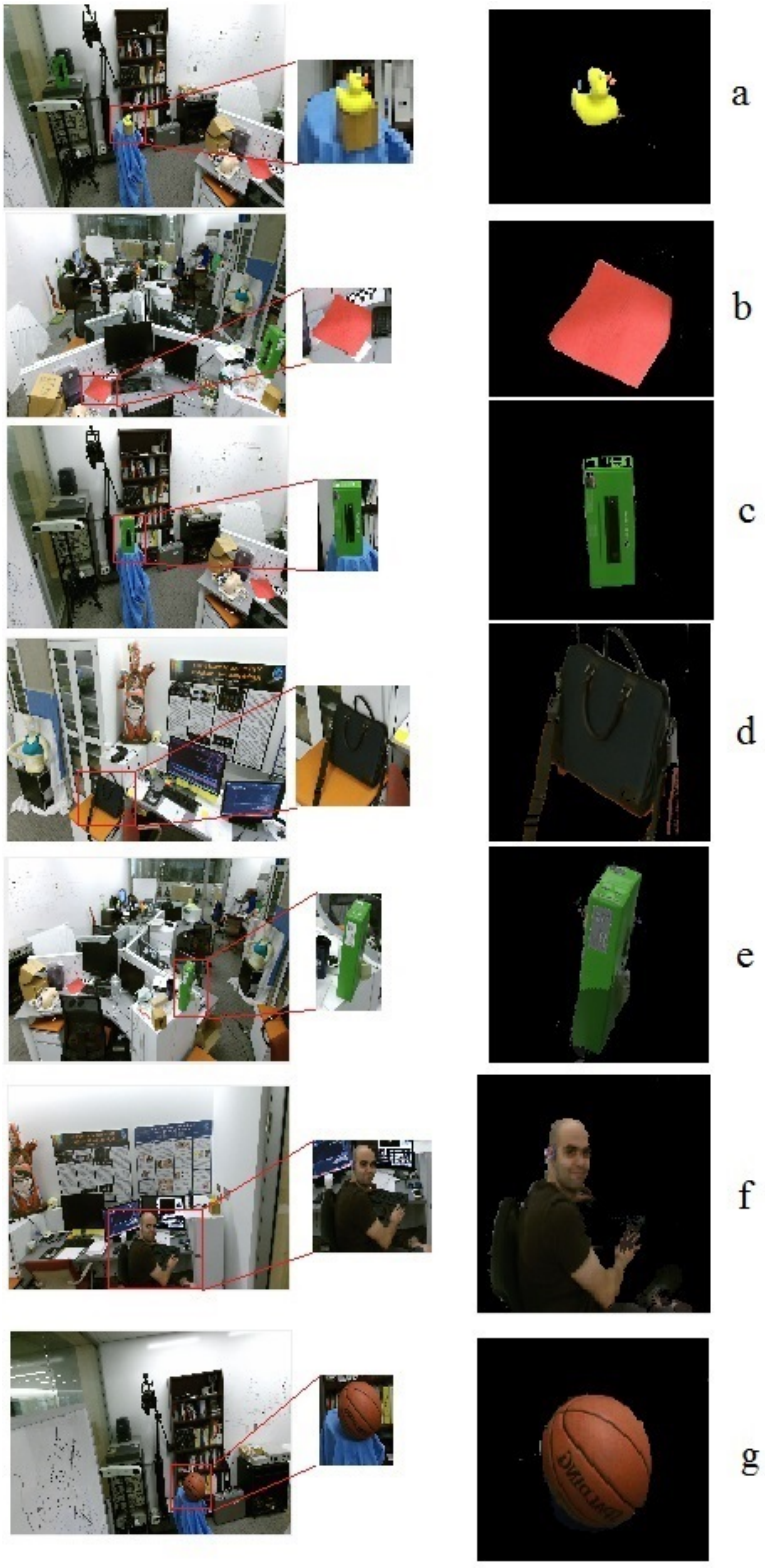

Fig. 6: Results of segmenting scene objects using proposed algorithm; a) Segmentation of small duck; b) Segmentation and detection of piece of red paper; c) Object detection of a box; d) Shows handy bag; e) Segmentation of box, the border of the box has lower weight and it will be completed after several iteration; f) Representation of moving object, segmentation of a person; g) Segmentation of basketball.

Algorithm 1:

while *main*// This is main loop that is starting at the binging and each iteration representing one frame
do
  while *all of data points*// this loop represents the number of data points we have in each frame
  do
    if *Is it the first iteration* // for the first iteration only we need to initialize the seeds
    then
      Assign centroid by using K-means++ // calculating the all centroid using K-means++
      Assign first weighted using regular labels
      // calculating the first label for each data point
    else
      while *all of clusters*// this loop start from 1 to k
      do

$$dist(v_i, v_c) = \sqrt{\begin{pmatrix} \delta^2 \begin{pmatrix} (X_i - X_c)^2 + \\ (Y_i - Y_c)^2 + \\ (Z_i - Z_c)^2 \end{pmatrix} + \\ \alpha^2 \begin{pmatrix} (R_i - R_c)^2 + \\ (G_i - G_c)^2 + \\ (B_i - B_c)^2 \end{pmatrix} \end{pmatrix}}$$

        /* calculating distance between each data point and centroid using position and color of each data point */

$$f(v_i, v_c) = dist(v_i, v_c) + \gamma(1 - \cos(\theta(n_i - n_c)))$$

        /* incorporate the differences between angle of normal vectors of each data point and centroid */
      if $f(v_i) > f(V_{i+1})$ // condition of distance between previous frame and current frame
      then

$$update\ \ labeled\ \ weights\ \ \delta_i = \delta_i + \Psi\ \ where\ \ 0 < \Psi \leq 1$$

        /* updating the weight of data point by scale of weights (one in our experiment) */

$$\mu_p = \frac{\sum_{i=0}^{\tau}(\delta_i)}{\| \sum_{i=0}^{\tau}(\delta_i) \|}$$

$$\sum_{i=0}^{k} \mu_p^i \leq 1$$

        /* normalize all weights for each data point where the summation of all weights should be less or equal than one */
  while *all of data points* // assigning color to each data point for visualization
  do

$$C_p = \sum_{i=0}^{k} (c_i \star \mu_i)$$

[!ht]

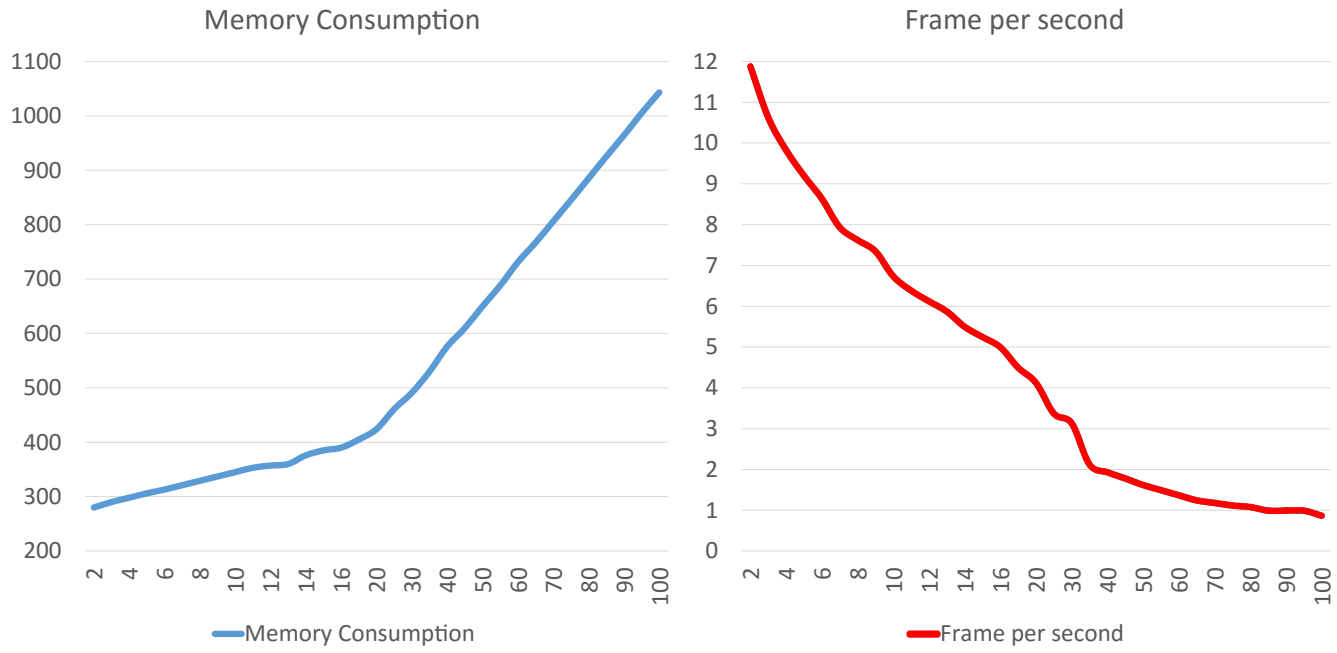

Fig. 7: Left: Chart indicates the memory consumption for different number of clusters where x-axis represents the number of clusters, k, value and y-axis show the memory consumption in MB. The larger the k value the more memory needed to process all scene points. Right: The frame rate is given in the right chart where x-axis is number of clusters, k, and y-axis is frame per second. The larger the k value the less frames per second evaluated.

III. Results

We test our algorithm using different number of clusters, k, and evaluate memory consumption and frame rate. Figure 7 indicates the frame rate experiments with k between 2 and 100 along with the memory consumption of each experiment. The frame rate and memory consumption are tightly affected by the number of clusters. When we have large number of clusters, the algorithm needs more memory, and frame rate will be reduced. Figure 3 shows the result of weighted supervised learning with k = 7 clusters after several iterations; the memory consumption for seven clusters is 322 MB, and frame rate is 7 $\pm$0.2 frame per second. The figure 4 shows the result of k=10 clusters where memory consumption is 344 MB, and frame rate is 6.5 $\pm$0.2 frame per second. The algorithm implementation consumes multi-thread programming in Visual Studio 2015, to implement parallel processing for the prepossessing and clustering parts. All of the loops use the parallel computational model introduced by Microsoft API [45] assign including for each data point back-projection, normal calculation, mapping, and assigning color along with calculating the weights of each data point [46]. The used hardware running our algorithm plays another factor for evaluating the system. The used CPU is capable of multi-thread programming and parallel processing, which is dual Xeon E5 family 2.29 GHz speed. It has 12 core and 24 logical processors. We do not use GPU in this application, but the GPU of the system is Nvidia Quadro K5000 with 4 GB GDDR5 speed [47] with 32 GB memory and having a speed of 1333 MHz. In addition, we use Universal Serial Bus (USB) version 3.0 for Kinect connection. Our input camera is Kinect V2 for Windows [39].

IV. Discussion

With regard to our results, this algorithm provides a unique output that can be useful for researchers in computer graphics, computer vision, robotics, and other related fields. The weighted unsupervised learning for object detection proposed in this paper shows its capabilities to the smooth the output in noisy environments in real-time producing scene objects. The results shown in Figure 7 indicate that this model can be real-time and has the capability for mobility applications with low memory consumption without the need for an expensive GPU. Using three features in clustering step, position, color and normal, gives us the capability to group similar data points to objects with accurate results. Our results show that the use of the weight to indicate the membership for each data point to a cluster and then for object detection is sufficient for detecting moving objects in noise environment, which was captured by an RGB-D camera.

V. Conclusion and Future work

In this paper, we proposed a weighted unsupervised learning for object detection framework using a single RGB-D camera. In the proposed method, we use CPU-based programming and multi-thread processing. The results are provided in real-time. In preprocessing step, we generate 3D data points, and after preprocessing, clustering step is applied by initialization of seeds and updating the weights of each data point. Finally assigning colors is used to illustrate object detection results. Another distinct contribution of this paper is segmentation of a particular object, particularly for moving

objects that can be updated by each frame. The frame rate of this work is between 1 and 12 frame per second, and memory consumption of this algorithm is between 280 MB and 1 GB for different values of clusters. This algorithm can be applied in any object detection and segmentation application in the fields of computer graphics, robotics, vision, or for surveillance and object controlling. Future directions of this work can be summarized in extending this method to GPU based programming, increasing the performance and frame rate simultaneously.

ACKNOWLEDGMENT

The authors want to acknowledge the help, resources and equipment that was provided by Professor James K. Hahn.

REFERENCES

[1] Fergus, Robert, Pietro Perona, and Andrew Zisserman. "Object class recognition by unsupervised scale-invariant learning." Computer Vision and Pattern Recognition, 2003. Proceedings. 2003 IEEE Computer Society Conference on. Vol. 2. IEEE, 2003.

[2] Amit, Yali, and Pedro Felzenszwalb. "Object Detection." Computer Vision: A Reference Guide (2014): 537-542.

[3] Crandall, David J., and Daniel P. Huttenlocher. "Weakly supervised learning of part-based spatial models for visual object recognition." Computer VisionECCV 2006. Springer Berlin Heidelberg, 2006. 16-29.

[4] Cheng, Yanhua, et al. "Semi-supervised Learning for RGB-D Object Recognition." Pattern Recognition (ICPR), 2014 22nd International Conference on. IEEE, 2014.

[5] Pandey, Megha, and Svetlana Lazebnik. "Scene recognition and weakly supervised object localization with deformable part-based models." Computer Vision (ICCV), 2011 IEEE International Conference on. IEEE, 2011.

[6] Felzenszwalb, Pedro F., et al. "Object detection with discriminatively trained part-based models." Pattern Analysis and Machine Intelligence, IEEE Transactions on 32.9 (2010): 1627-1645.

[7] Papageorgiou, Constantine, and Tomaso Poggio. "A trainable system for object detection." International Journal of Computer Vision 38.1 (2000): 15-33.

[8] Girshick, Ross, et al. "Rich feature hierarchies for accurate object detection and semantic segmentation." Computer Vision and Pattern Recognition (CVPR), 2014 IEEE Conference on. IEEE, 2014.

[9] Chacon-Murguia, Mario, and Sergio Gonzalez-Duarte. "An adaptive neural-fuzzy approach for object detection in dynamic backgrounds for surveillance systems." Industrial Electronics, IEEE Transactions on 59.8 (2012): 3286-3298.

[10] Lai, Lien-Fu, et al. "Developing a fuzzy search engine based on fuzzy ontology and semantic search." Fuzzy Systems (FUZZ), 2011 IEEE International Conference on. IEEE, 2011.

[11] Bo, Liefeng, et al. "Object recognition with hierarchical kernel descriptors." Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on. IEEE, 2011.

[12] Lai, Kevin, et al. "A large-scale hierarchical multi-view rgb-d object dataset." Robotics and Automation (ICRA), 2011 IEEE International Conference on. IEEE, 2011.

[13] Alassaf, Manal H., Kamran Kowsari, and James K. Hahn. "Automatic, Real Time, Unsupervised Spatio-temporal 3D Object Detection Using RGB-D Cameras." Information Visualisation (iV), 2015 19th International Conference on. IEEE, 2015.

[14] Bo, Liefeng, Xiaofeng Ren, and Dieter Fox. "Unsupervised feature learning for RGB-D based object recognition." Experimental Robotics. Springer International Publishing, 2013.

[15] Maddalena, Lucia, and Alfredo Petrosino. "A fuzzy spatial coherence-based approach to background/foreground separation for moving object detection." Neural Computing and Applications 19.2 (2010): 179-186.

[16] Nair, Vinod, and James J. Clark. "An unsupervised, online learning framework for moving object detection." Computer Vision and Pattern Recognition, 2004. CVPR 2004. Proceedings of the 2004 IEEE Computer Society Conference on. Vol. 2. IEEE, 2004.

[17] Oneata, Dan, et al. "Spatio-Temporal Object Detection Proposals." Computer VisionECCV 2014. Springer International Publishing, 2014. 737-752.

[18] Ye, Edmund Shanming, and Jitendra Malik. Object detection in rgb-d indoor scenes. Diss. Masters thesis, EECS Department, University of California, Berkeley (Jan 2013), http://www. eecs. berkeley. edu/Pubs/TechRpts/2013/EECS-2013-3. html, 2013.

[19] Winn, John, and Nebojsa Jojic. "Locus: Learning object classes with unsupervised segmentation." Computer Vision, 2005. ICCV 2005. Tenth IEEE International Conference on. Vol. 1. IEEE, 2005.

[20] Liu, David, and Tsuhan Chen. "Semantic-shift for unsupervised object detection." Computer Vision and Pattern Recognition Workshop, 2006. CVPRW'06. Conference on. IEEE, 2006.

[21] Alassaf, Manal H., Yeny Yim, and James K. Hahn. "Non-rigid Surface Registration using Cover Tree based Clustering and Nearest Neighbor Search." Proceedings of the 9th International Conference on Computer Vision Theory and Applications. 2014.

[22] Pairote, Sattayatham, Kerdprasop Kittisak, and Kerdprasop Nittaya. "Weighted K-means for density-biased clustering." (2000).

[23] Kowsari, Kamran, Yammahi, Maryam, Bari, Nima, Vichr, Roman, Alsaby, Faisal and Berkovich, Simon Y. Construction of FuzzyFind Dictionary using Golay Coding Transformation for Searching Applications,International Journal of Advanced Computer Science and Applications (IJACSA) (2015)

[24] Kowsari, Kamran, et al. "Comparison three methods of clustering: k-means, spectral clustering and hierarchical clustering." arXiv preprint arXiv:1312.6117 (2013).

[25] Bari, Nima, Vichr, Roman, Kowsari, Kamran and Berkovich, Simon Y. Novel Metaknowledge-based Processing Technique for Multimedia Big Data clustering challenges, The IEEE International Conference on Multimedia Big Data (BigMM), (2015)

[26] Bari, Nima, Roman Vichr, Kamran Kowsari, and Simon Berkovich. "23-bit metaknowledge template towards big data knowledge discovery and management." In Data Science and Advanced Analytics (DSAA), 2014 International Conference on, pp. 519-526. IEEE, 2014.

[27] Kerre, Etienne E., and Mike Nachtegael, eds. Fuzzy techniques in image processing. Vol. 52. Physica, 2013.

[28] El Baf, Fida, Thierry Bouwmans, and Bertrand Vachon. "Fuzzy integral for moving object detection." Fuzzy Systems, 2008. FUZZ-IEEE 2008.(IEEE World Congress on Computational Intelligence). IEEE International Conference on. IEEE, 2008.

[29] Kowsari, Kamran. Investigation of FuzzyFind Searching with Golay Code Transformations. Diss. M. Sc. Thesis, The George Washington University (2014)

[30] Yammahi, Maryam, Kamran Kowsari, Chen Shen, and Simon Berkovich. "An Efficient Technique for Searching Very Large Files with Fuzzy Criteria Using the Pigeonhole Principle." In Computing for Geospatial Research and Application (COM. Geo), 2014 Fifth International Conference on, pp. 82-86. IEEE, 2014.

[31] Goodarzi, Farhad, Daewon Lee, and Taeyoung Lee. "Geometric nonlinear PID control of a quadrotor UAV on SE (3)." Control Conference (ECC), 2013 European. IEEE, 2013.

[32] Goodarzi, Farhad, Daewon Lee, and Taeyoung Lee. "Geometric stabilization of a quadrotor UAV with a payload connected by flexible cable." American Control Conference (ACC), 2014. IEEE, 2014.

[33] Goodarzi, Farhad A., Daewon Lee, and Taeyoung Lee. "Geometric Adaptive Tracking Control of a Quadrotor UAV on SE (3) for Agile Maneuvers." Journal of Dynamic Systems, Measurement, and Control (2015).

[34] Goodarzi, Farhad A. Geometric Nonlinear Controls for Multiple Cooperative Quadrotor UAVs Transporting a Rigid Body, The George Washington University (2015)

[35] Farhad A. Goodarzi and Taeyoung Lee. Dynamics and Control of Quadrotor UAVs Transporting a Rigid Body Connected via Flexible Cables. Proceedings of American Control Conference (2015)

[36] Hunter, Richard S. "Photoelectric color difference meter." Josa 48.12 (1958): 985-993.

[37] Leon, Katherine, et al. "Color measurement in L a b units from RGB digital images." Food research international 39.10 (2006): 1084-1091.

[38] Frntratt, Hermann, and Helmut Neuschmied. "Evaluating pointing accuracy on Kinect V2 sensor." International Conference on Multimedia and Human-Computer Interaction (MHCI). 2014.

[39] Smisek, Jan, Michal Jancosek, and Tomas Pajdla. "3D with Kinect." Consumer Depth Cameras for Computer Vision. Springer London, 2013. 3-25.

[40] Amon, Clemens, Ferdinand Fuhrmann, and Franz Graf. "Evaluation of the spatial resolution accuracy of the face tracking system for kinect for windows v1 and v2." Proceedings of the 6th Congress of the Alps Adria Acoustics Association. 2014.

[41] Butkiewicz, Thomas. "Low-cost coastal mapping using Kinect v2 time-of-flight cameras." Oceans-St. John's, 2014. IEEE, 2014.

[42] Henry, Peter, et al. "RGB-D mapping: Using Kinect-style depth cameras for dense 3D modeling of indoor environments." The International Journal of Robotics Research 31.5 (2012): 647-663.

[43] Han, Jungong, et al. "Enhanced computer vision with microsoft kinect sensor: A review." Cybernetics, IEEE Transactions on 43.5 (2013): 1318-1334.

[44] Arthur, David, and Sergei Vassilvitskii. "k-means++: The advantages of careful seeding." Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms. Society for Industrial and Applied Mathematics, 2007.

[45] Gregory, Kate, and Ade Miller. "C++ Amp: accelerated massive parallelism with Microsoft Visual C++." (2014).

[46] Microsoft MSDN. Auto-Parallelization and Auto-Vectorization (2015)

[47] Nvidia. NVIDIA Quadro K5000, www.nvidia.com/object/quadro-k5000.html

[48] Mankoff, Kenneth D., and Tess Alethea Russo. "Kinects as sensors in earth science: glaciological." American Geophysical Union, Fall Meeting.

[49] Onoda, Takashi, Miho Sakai, and Seiji Yamada. "Careful seeding method based on independent components analysis for k-means clustering." Journal of Emerging Technologies in Web Intelligence 4.1 (2012): 51-59.

VI. APPENDIX

Converting the projective 3D position to real world with respect to camera is done by following equations 10, 11, 12, 13, and 14, where field of view is denoted by fov, W_x, W_y and W_z is world coordinate system of X, Y, and Z respectively, and P_x, P_y, and P_z shows the projection parameters.

$$\underset{x}{\text{scale}} = 2 \star \tan(\frac{fov_x}{2}) \tag{10}$$

$$Scale_y = 2 \star \tan(\frac{fov_y}{2}) \tag{11}$$

$$W_x = P_z \star Scale_x \star (\frac{P_x}{R_w} - 0.5) \tag{12}$$

$$W_y = P_z \star Scale_y \star (\frac{P_y}{R_w} - 0.5) \tag{13}$$

$$W_z = P_z \tag{14}$$

fov_x for Kinect 1is equal to 1.014468 and fov_y is equal to 0.7898094 [48], and in many researches, people use these numbers that reduced the field of view, but as new resolution of Kinect V2 we use 1.22173047 for fov_x and 1.0471975511 by changing degree to radian $Radian = \frac{Degree \star \pi}{180}$. The weight of each data point can be equal or bigger than zero, $\mu_i > 0$ where $i \in [0,\ k-1]$ and k is the number of clusters. The distance which is calculated in equation 4 and 6 and shown as $f(v_i, v_c)$, can be zero if and only if centroid and data point i are equal to each other, $i = c$. The scale of color and position as shown δ and α, both are positive number and cannot be zero, and as we mention to that, summation of them is less than one. All results are calculated by sampling of one, so if we increase the sampling to more than one our resolution, and memory consumption will decrease and frame rate will be increased.