Polyhedral Active Recognition

Sajjad Pakdamansavoji (savoji@yorku.ca)

April 17, 2022

Abstract

This report is written for the final project of Advanced Computer Vision course at York University. The aim of this challenge is to implement an active vision solution for distinguishing amongst sinle polyhedrals.

In short My solution consists of four main modules: observation, feature extraction, comparison, active motion policy. Observation module receives a camera position and object ID from the Active Policy module and presents an image of the object. Feature extraction takes in the observation and creates informative features from it. Comparison module takes in those features and decides whether the two objects are the same. Such decision is then propagated to the Active Motion Policy where the next observation position is created. The result of my implementation is submitted for evaluation; it follows an original idea and presents a reasonable structure. My code is publicly available at `https://github.com/SajjadPSavoji/Active-Polyhedral-Scene-Matching`.

1 Introduction

Existing computer vision challenges(such as image net[1]) heavily emphasize on static object recognition; however when the 3D point-could of objects are projected into the image plane a large proportion of information is thrown away causing ambiguity for recovering the 3D structure. Some works suggest using additional sensory input such as LIDAR to compensate at least a proportion of that missing information. Although such approaches are useful and practical, recognizing objects should better be addressed in an active fashion since the physical constraints of an object imposes the necessity of obtaining multiple view-points to recover the 3D-structure and hence recognize the target fully.

In this study we wish to showcase the imortance of active vision; as such, the task in hand is using simple polyhedral objects. These objects are synthesized using Random Polyhedral Scenes [2] which was developed at York University. This scene generator creates a random object based on a few user parameters, renders the scene from random view points and creates a dataset containing the renderings and corresponding annotation files 1.

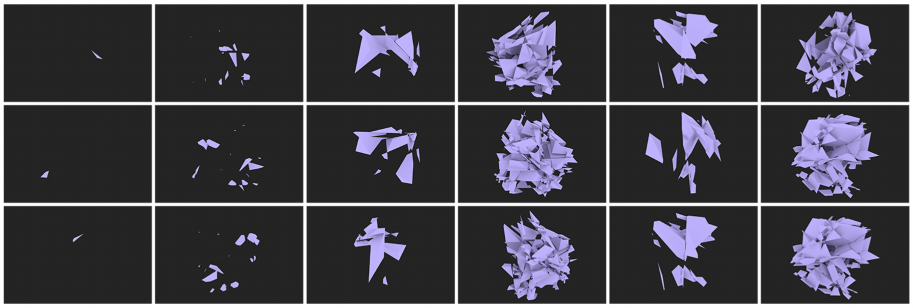

Figure 1: Six different scenes (left to right) rendered from three different views (top to bottom). First row: 1 object. Second row: 18 objects and scene layout set to separate. Third row: 18 objects and scene layout set to touching. Fourth row: 180 objects and scene layout set to separate. Fifth row: 18 objects and scene layout set to intersecting. Second row: 100 objects and intersecting layout.

2 Related Work

This topic, polyhedral recognition, has been studied, both theoretically and empirically, numerous times in different context. As for the theoretical point of view, Kirousis and Papadimitriou proved that having a graph scheme of a polyhedral it is NP-Complete to either decide a legal labeling for them or state whether it is a projection of a given scene[3]. Furthermore, Tsotsos presented two theorems regarding the complexity of Visual Search. These theorems show that Unbounded Visual Search is NP-Complete while Bounded Visual Match has linear time complexity. These theorems point out the fact that knowledge of the scene reduces the complexity of this particular problem[4]. These studies have a number of implications. They first tell us that the data-directed approach to vision is computationally intractable in the general case. Meaning that to solve a vision problem solely based on data, one needs to obtain an uncountable number of images which is not feasible. They also tell us that bounded visual search takes time linearly dependent on the size of the input [5]. Also, even small amounts of task guidance can turn an exponential problem into one with linear time complexity.

All the aforementioned theoretical proofs are for the case of a single image input; hence one might wonder, considering the fact that scene knowledge reduces the time complexity of recognition task, does having multiple view-points change NP-Completeness also? This simple question is the bridge between active perception and object recognition [6]. Active perception is the case in which an active observer has the ability of acquire multiple-view points from a scene. Such a system naturally has to solve a number of sub-tasks including finding the next best position and/or camera configuration.

3 Methodology

3.1 Problem Settings

Given two polyhedral scene IDs $\{I_1, I_2\}$ decide whether the underlying 3D objects are the same or not. To solve this challenge one should follow an active vision scheme in which each observations are denoted as $\{F_i^n, F_j^m\}_{n,m=1}^N$ in which i and j are object indexes and N is the maximum number of view-points taken from an object. that being the case my algorithm consists of four major modules: observation, feature extraction, comparison, and motion policy Fig 2.

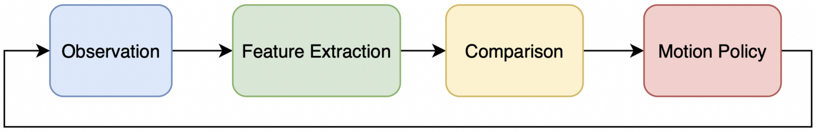

Figure 2: Illustration of four main modules of system

3.2 Observation

For the observation part I am using the on-demand API. This API provides an easy way of getting rendered images from the polyhedral scenes only by providing a few hyper parameters for camera configuration. These hyper parameters include ID, camera coordinates(x, y, z), orientating as quaternion(Q_w, Q_x, Q_y, Q_z), and lightning condition. In my setting I am assuming that the camera is always looking at the origin. With that in mind, the camera location can be represented via a spherial coordinate system (r, ϕ, θ). Furthermore, using the pixel intensity of object, I will adjust the r parameter so that a fixed percentage of vertical and/or horizontal internsity is filled with the object. In that case camera position can be denoted with only two parameters ϕ, θ. Similar to previous works, I will consider a discrete step size of these two parameter; meaning that they can only obtain values which are a multiplier of $\pi/6$.

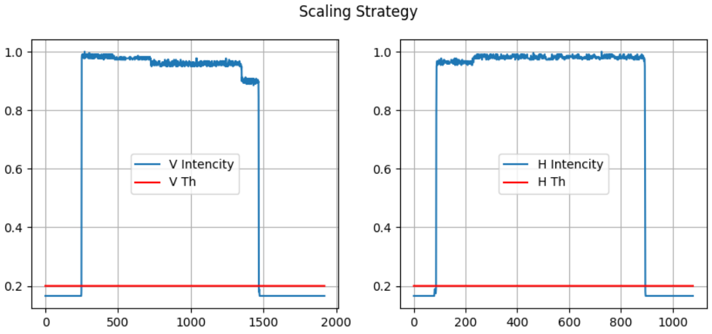

Figure 3: Scaling mechanism to determine r

3.3 Feature Extraction

The feature extraction part itself consists of 6 parts: pre-processing, edge detection, line detection, intersecting lines, clustering joints, and graph representation. In this section each part will be explained.

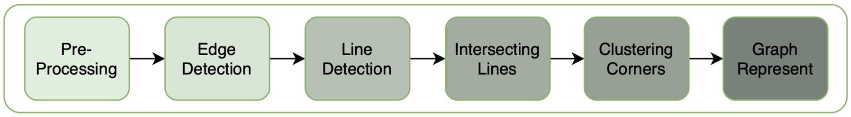

Figure 4: Feature extraction steps

Pre-Processing. As a the generated polyhedrals all have the same color, in the pre-processing a gray-scale version is generated. After that, all the functions will be aplied to this image. It is worth mentioning that most edge detection bydefault only work with gray-scale images; in that sense this step is inevitable.

Edge Detection. Edge detection is an image-processing technique, which is used to identify the boundaries (edges) of objects, or regions within an image. Edges are among the most important features associated with images. We come to know of the underlying structure of an image through its edges. Computer vision processing pipelines therefore extensively use edge detection in applications. Edges are characterized by sudden changes in pixel intensity. To detect edges, we need to go looking for such changes in the neighboring pixels. Sobel Edge Detection is one of the most widely used algorithms for edge detection. The Sobel Operator detects edges that are marked by sudden changes in pixel intensity. This approach states that edges can be detected in areas where the gradient is higher than a particular threshold value. In addition, a sudden change in the derivative will reveal a change in the pixel intensity as well. With this in mind, we can approximate the derivative, using a 3×3 kernel. We use one kernel to detect sudden changes in pixel intensity in the X direction, and another in the Y direction. These are the kernels used for Sobel Edge Detection:

$$K_x = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix} \qquad K_y = \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

When these kernels are convolved with the original image, you get a 'Sobel edge image'. If we use only the Vertical Kernel, the convolution yields a Sobel image, with edges enhanced in the X-direction Using the Horizontal Kernel yields a Sobel image, with edges enhanced in the Y-direction. Let G_x and G_y represent the intensity gradient in the x and y directions respectively. If K_x and K_y denote the X and Y kernels defined above, then the joint XY detection follows the equations below.

$$G_x = K_x * I \qquad G_y = K_y * I \qquad G_{xy}\sqrt{G_x^2 G_y^2} \qquad \Theta = arcTan(G_y/G_x)$$

Line Detection. For the line detection part the Hough Line Transform is used. It is a transform used to detect straight lines only. To apply the Transform, first an edge detection pre-processing is desirable which was covered in the previous section. As you know, a line in the image space can be expressed with two variables $(m, b)or(r, \theta)$ For Hough Transforms, we will express lines in the Polar system. Hence, a line equation can be written as

$$y = \left(-\frac{cos(\theta)}{sin(\theta)}\right) x + \left(\frac{xcos(\theta) + ysin(\theta)}{sin(\theta)}\right)$$

In general for each point (x_0, y_0), we can define the family of lines that goes through that point as $r_\theta = x_0 cos\theta + y_0 sin\theta$. Meaning that each pair (r_θ, θ) represents each line that passes by (x_0, y_0). If for a given (x_0, y_0) we plot the family of lines that goes through it, we get a sinusoid. We can do the same operation above for all the points in an image. If the curves of two different points intersect in the plane $\theta - r$, that means that both points belong to a same line. t means that in general, a line can be detected by finding the number of intersections between curves.The more curves intersecting means that the line represented by that intersection have more points. In general, we can define a threshold of the minimum number of intersections needed to detect a line. This is what the Hough Line Transform does. It keeps track of the intersection between curves of every point in the image. If the number of intersections is above some threshold, then it declares it as a line with the parameters (θ, r_θ) of the intersection point.

Line Intersection. While the edges of the polydedral are important features, having them alone will not be of use as it will not be representative enough. As such, it is possible to extract the corners of the polyhedral too. To do so I am using a hybrid method wich combines the Harris corner detector with my own line intersection algorithm. As for the latter case, once a line equation is given as $y = mx + b$ one can use it to detect the intersection point of two lines. using the following formula:

$$y_1 = ax + b, \quad y_2 = cx + d \quad \rightarrow \quad P_0 = \left(\frac{d-c}{a-b}, a\frac{d-c}{a-b} + c\right)$$

This simple idea is combined with Harris corner detection. Corners are regions in the image with large variation in intensity in all the directions. One early attempt to find these corners was done by Chris Harris Mike Stephens in their paper A Combined Corner and Edge Detector in 1988, so now it is called the Harris Corner Detector. He took this simple idea to a mathematical form. It basically finds the difference in intensity for a displacement of (u, v) in all directions.

$$E(u,v) = \sum_{x,y} \underbrace{w(x,y)}_{\text{window function}} \, [\underbrace{I(x+u, y+v)}_{\text{shifted intensity}} - \underbrace{I(x,y)}_{\text{intensity}}]^2$$

The window function is either a rectangular window or a Gaussian window which gives weights to pixels underneath. We have to maximize this function

$E(u,v)$ for corner detection. That means we have to maximize the second term. Applying Taylor Expansion to the above equation and using some mathematical steps (please refer to any standard text books you like for full derivation), we get the final equation as shown below. Then comes the main part. After this, they created a score, basically an equation, which determines if a window can contain a corner or not.

$$E(u,v) \approx \begin{bmatrix} u & v \end{bmatrix} M \begin{bmatrix} u \\ v \end{bmatrix} \qquad M = \sum_{x,y} w(x,y) \begin{bmatrix} I_x I_x & I_x I_y \\ I_x I_y & I_y I_y \end{bmatrix}$$

$$R = \det(M) - k(\operatorname{trace}(M))^2$$

Clustering Corners. Since the corners detected in the previous step are far more than the actual corners, a clustering algorithm will be used to cluster them to unified corners. Once they are clustered, the center of each cluster will be used as the new object corner. For this specific task, I chose DBSCAN as it does not require the number of clusters in its hyper parameters. The DBSCAN algorithm views clusters as areas of high density separated by areas of low density. Due to this rather generic view, clusters found by DBSCAN can be any shape, as opposed to k-means which assumes that clusters are convex shaped. The central component to the DBSCAN is the concept of core samples, which are samples that are in areas of high density. A cluster is therefore a set of core samples, each close to each other (measured by some distance measure) and a set of non-core samples that are close to a core sample. There are two parameters to the algorithm, min-samples and eps, which define formally what we mean when we say dense. Higher min-samples or lower eps indicate higher density necessary to form a cluster.

Graph Representation. Last but not least, once the corners and edges of the polyhedral are extracted, these information will be treated as a underacted graph with scaled edges. This means that an adjacency matrix M is computed for each graph in which nodes are the polyhedral corners and the edges are its corresponding edges. As for the cost of each eadge, it is computed via the pixed distance of corners devided by the camera scale parameter r. It is worth mentioning that the index of each corner(node) in the graph is determined by its pixel distance ranking from the mean corner. Using this scheme the produce graph representation M is invariant to scale and orientation.

$$M = [M_{ij}] \qquad M_{ij} = PixelDist(corner_i, corner_j)/r \quad or \;\; = 0$$

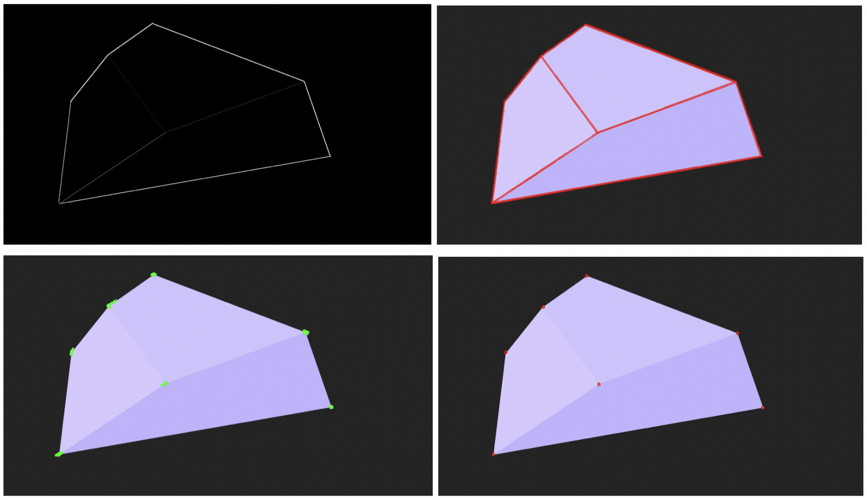

Figure 5: top-left: edges of sobel algorithm, top-right: lines of Hough detection, bottom-left: all the joints detected , bottom-right: center of corner clusters

3.4 Comparison

To compare two object the extracted features, adjacency matrices are used in the following way. For a set of image observations $\{F_i^n, F_j^m\}_{n,m=1}^N$, the corresponding features are denoted as $\{X_i^n, X_j^m\}_{n,m=1}^N$. Then the distance of two objects is defined as the minimum distance between all feature pairs $\{X_i^n, X_j^m\}$ as illustrated below.

$$D(obj_i, obj_j) = min_{m,n}\{d(\{X_i^n, X_j^m\})\} \qquad d(\{X_i^n, X_j^m\}) = Mean(abs(M_i^n - M_j^m))$$

3.5 Active Motion Policy

Once the distance of features are computed, there should be a policy deciding what should the next best frame be. Towards that end, with the intuition and corners that are far apart has the potential to present more information, out active policy finds out that specific corner and adjusts the camera location parameters (ϕ, θ) in a way that the chosen corner will be closer to the center of image. Note that these changes in camera coordinates are discrete with step size of $\pi/6$.

4 Experiment

For experiments, a nmber of test cases where provided for each students. The answers for the test cases are submitted in a csv file.

5 Conclusion

To evaluate the model one should have had grand-truth which was not available for my case. In sum I believe that the approach used in my work follows a common pipeline and hence has room for improvements. A comprehensive evaluation of my solution was not possible as the on demand API has significant latency and therefore capturing a models performance via test cases takes a lot of time. With that said, the proposed solution still has the necessary components of an active vision system and I hope it has reasonable performance.

References

[1] Deng, J., Dong, W., Socher, R., Li, L.J., Li, K., Fei-Fei, L.: Imagenet: A large-scale hierarchical image database. In: 2009 IEEE conference on computer vision and pattern recognition, Ieee (2009) 248–255

[2] Solbach, M.D., Voland, S., Edmonds, J., Tsotsos, J.K.: Random Polyhedral Scenes: An Image Generator for Active Vision System Experiments. ArXiv e-prints (March 2018)

[3] Kirousis, L.M.: A polynomial algorithm for recognizing images of polyhedra. In Makedon, F., Mehlhorn, K., Papatheodorou, T., Spirakis, P., eds.: VLSI Algorithms and Architectures, Berlin, Heidelberg, Springer Berlin Heidelberg (1986) 194–204

[4] Tsotsos, J.K.: The complexity of perceptual search tasks. In: IJCAI. Volume 89. (1989) 1571–1577

[5] Wolfe, J.M.: Visual search. (2015)

[6] Bajcsy, R., Aloimonos, Y., Tsotsos, J.K.: Revisiting active perception. Autonomous Robots **42**(2) (2018) 177–196