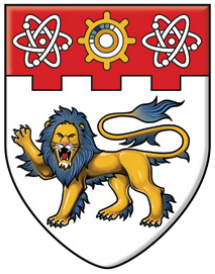
Hierarchical Techniques for Curve Matching to Aid Class-Level Object Recognition



NANYANG TECHNOLOGICAL UNIVERSITY

CONFIRMATION REPORT

Author:

Vittal Premachandran

G0902291K

School of Computer Engineering

Nanyang Technological University

Supervisor:

Assoc. Prof. Ramakrishna Kakarala

April 2011

Abstract

Object recognition has become an important area of research in the field of Computer Vision. The benefits of having a computer that can "see" like humans is there for everyone to see. The ability of humans to identify objects in natural scenes from different perspectives is what helps us to understand the nature of the scene. Such understanding of the scene is what is required if we are to build a full-fledged automatic robot.

Traditionally, object recognition has been perceived at the level of specific objects. Not enough research has been done in the development of class-level object recognition algorithms. In order to achieve the same, it is important to be aware of the properties of the objects. Different object classes have different characteristics that can be used to categorise them uniquely. These class-specific characteristics enable us to develop better feature descriptors that will eventually help in generic object recognition. Hence, analyzing and making sense of these object characteristics are an important part of Computer Vision.

Current day image processing techniques have developed greatly and are aiding high-level computer vision. The low-level image processing techniques develop primitives for the use of higher-level computer vision techniques. However, many of the high-level computer vision techniques do not make use of the relations between the low-level primitives. One area that will benefit hugely, by incorporating the relation between these object primitives, is the field of object recognition from shapes. Object shapes are unique features of a particular object class. The angles at which the edges intersect with each other, the distance ratios between various corners, etc., is what makes an

object uniquely different from objects belonging to other classes. We aim to make use of these class-specific relations for enhanced object recognition.

Along with finding the image primitives, analysis of the relationship between them is essential to understand the object at a holistic level. Hence, we propose to approach the problem in a hierarchical fashion. Features from image shapes are extracted in the first step to identify candidate matches. Next, we aim to use the relations between the different features, which are unique to an object class, to further reduce the search space. We believe that such an approach would take into consideration both the local and the global properties of the objects. The initial step would filter out objects whose local properties do not match the local properties of the object being searched. The subsequent steps would look at global properties to narrow down on the object more accurately.

Analysis of global properties involves the analysis of local primitives in relation to each other. We propose a unique way to analyse the relationships of primitives by taking into account the primitive adjacency obtained using contour-following methods. We show how such methods can be used for narrowing down on the object and eventually pinpointing the same.

Contents

List of Figures

Chapter 1

Introduction

In the final quarter of the 20th century, when there was a rapid development in the electronics industry, the world saw the birth of personal computers. These machines could perform various tasks at very high speeds. More intelligence was being added into the computers at an exponential rate, which enabled computers to perform more and more complex tasks automatically. Computers started performing tasks so well and so meticulously that engineers and scientists believed that they would be able to build full-fledged robots in 20-30 years' time that would be as intelligent as humans if not more. There were dreams of keeping such robots as house cleaners that would do our daily chores while the humans took care of the "more important" jobs. The number of applications that sprang to the mind was infinite. The then computer scientists initially thought that these were achievable targets and were confident of getting the task done within the predicted period. The exciting opportunities that would arise out of building robots drove them into researching in the field of Artificial intelligence. Little did they know that building such a robot would not only turn out to be a difficult feat, but would also turn out to be a task that was next to impossible to accomplish.

Today, we are in the second decade of the 21st century and still nowhere close to achieving the target that was set approximately 25 years ago. Let alone the sizeable task of building full-fledged robots, we are not even able to automate major tasks that are natural to us human beings, e.g. Vision, Natural Language

Processing, etc. There is still a long way to go in order to achieve the final all-important goal of building human-like robots.

Hence, before aiming for far-fetched dreams such as 'building of a fully automated robot', we have to shift our focus and try to achieve perfection in its component tasks. Once its components are perfected, we will be in a better state to achieve the final goal. One of the major component areas, which require automation, is vision. Computer vision is a branch where in techniques are developed to enable computers to "see" just like human beings. Computer Vision is an important area of artificial intelligence that needs to be mastered in order to perform many of the vision related automation jobs. A robot that can "see" as human beings see, can navigate paths better, interact with objects better, and learn new environments, thus building up its cognitive database. To achieve human-like vision capabilities, the computer should be able to recognize objects in its field of view, analyze the scene's environmental conditions, and understand the relations between the objects in the scene. A major area that intersects with all the above-mentioned goals is the area of object recognition. It is only when a computer can recognize the objects in its field of view that it can analyze the scene's environmental conditions and understand the relationship between the various objects in the scene. The term 'object' refers to any entity in the scene that can be seen. Ex: Roads, sky, trees, tables, books, etc. Object recognition involves the recognition of such objects in the image. Figure 1.1 shows different types of objects that can be recognised in a scene. This PhD work is dedicated for research in the area of object recognition.

1.1 Object Recognition

The foundation of many image analysis functionalities usually boils down to object recognition. The importance of object recognition in automated computer vision techniques cannot be underestimated. Be it in a face recognition feature of a camera, or a smile detection algorithm, or an algorithm that analyses the aesthetic appeal of a photograph; object recognition plays an important role. Object recognition can be thought of as an integral analysis tool whose roots lie

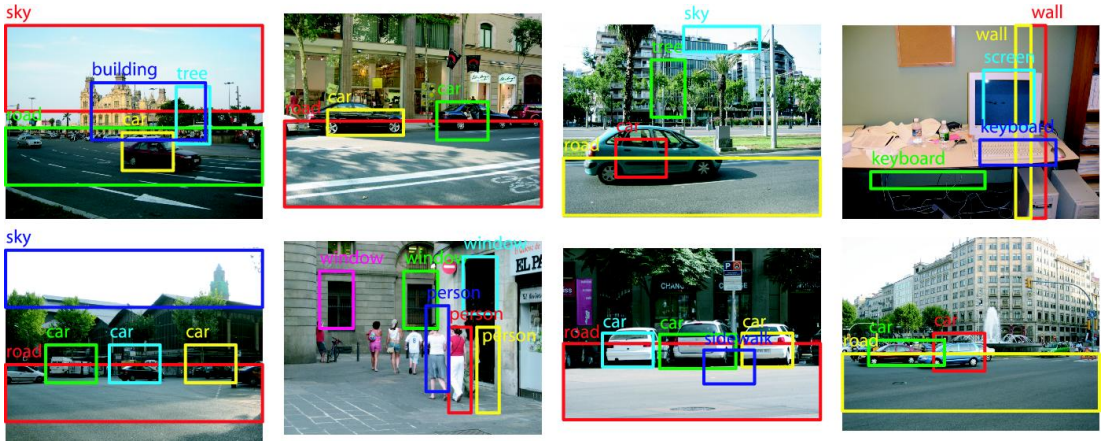


Figure 1.1: Objects in a scene - Entities in an image that can be called an object. There are far more objects in the image than what is labeled. Recognition of such objects is an unsolved problem in the field of Computer Vision. (Image from LabelMe Database [45])

deep within the field of pattern recognition i.e. analysis of various patterns in an image.

The human brain has an uncanny knack of recognizing objects and deciphering patterns. The Human Visual System (HVS) comprises of both the human brain and the eyes. The eyes perform the activity of seeing in conjunction with the brain. The eyes, cannot on their own, recognize objects. They send signals to the brain that are later interpreted into meaningful data. The eyes capture the image on the retina, and send the signals via the optic nerve to the brain for interpretation. The first phase of this process has been well replicated in the digital world. Digital cameras are used to capture the image and store the data in digital format. The interpretation of this data is the most important part of object recognition that needs to be analyzed. How is it that the brain can convert electric signals into meaningful data? How is it that different inferences are combined into one single inference? Further research has to be done to answer these questions.

With a brain as intelligent as ours, we are still not able to come up with algorithms that can be made to run automatically, and thus help the computers to "see" just like the HVS. The benefits of building a fully automated Computer

Vision system is there for everyone to see and hence the field of Automated Computer Vision recognition system has become a major field of research in the Computer Science society. The researchers in this field have tried to mimic the human brain in order to identify objects in the visual scene that our brains recognize without much of a hassle. The fact that the human brain can do these tasks so well acts as a motivation to mimic its functionalities onto the digital world. We can take cues from the methods that the human brain adopts and convert them into digital solutions enabling computers to solve the problems in a similar way.

However, mimicking the functionalities of the human brain is not enough. Learning different features is equally important. While we can recognize a car, or an airplane, or any other object that we have already seen, human beings cannot recognize all types of objects at all times. We first have to learn the properties of the objects before we are able to recognise them. Learning is a vital process in object recognition. Hence, if a computer is being programmed to recognize objects by mimicking the human brain's methods, we have to allow it to learn unique features over a period to achieve robust identification.

Objects can be recognized either stand-alone or in conjunction with its environment. Recognition of objects based on its environment is called Context-Based object classification. This is described in section 1.1.1. On the other hand, stand-alone object recognition does not take into consideration the object's surrounding environments. Methods for stand-alone object recognition are described in section 1.1.2.

1.1.1 Context-Based Recognition

The environment in which an object is placed plays a very important role in recognizing the objects. Many objects that we interact with daily are easily recognized by us immaterial of the environment that it is placed in. However, some objects need the help of their surroundings in order for us to classify them. For example, let us say that we are shown the picture of a flat screen display device and nothing else in its vicinity. Just looking at the picture, we may be able to classify it into a broader class of "display screens". Now suppose we

were shown the same picture with a computer CPU alongside it. We will then be able to classify it as a "Computer Monitor". If on the other hand, the same screen was shown to be alongside a TV Remote, our brain would then be more inclined to classify it as a "TV Screen". This clearly shows that our brains make classifications based on the context in which the object is present.

Context-based object recognition is mainly used in scene analysis. Given a scene, the objects in the scene are identified and the relation between the objects are sought out to understand the scene's characteristics. This technique is used for indoor/outdoor scene classification, map navigation, etc [28, 50, 52]. This kind of learning is useful for very high-level computer vision. In many cases, context-based object learning techniques require the knowledge of other objects in the scene, which have to be recognized using the stand-alone techniques.

1.1.2 Stand-Alone Object Recognition

Stand-alone object recognition uses information obtained from an object and tries to classify the object based on the information that it has received. This is usually a very difficult task to do, as there are far too many classes of objects in real world scenes to keep track of. Since we cannot take into consideration all the classes of objects, we aim to narrow down to the true class in a hierarchical fashion. The human brain also adopts a hierarchical object recognition technique.

Most of the objects in real world are classified into different classes in a hierarchical fashion with each class having its own unique set of features. The difference between the features belonging to different classes is usually high, while the difference between features belonging to objects of the same class is usually very small. We can make use of this property to narrow down at each step, the list of classes of which the object is a part. Hence, if we adopt a hierarchical approach for object classification, we can reduce the search space at each level of classification and slowly narrow down towards the true class of the object. For example, before we recognize a car in front of us to be, say, a Ferrari, we first classify the object into a higher-level category of "Car". It is only after classifying it as a car that we try to find out its make and model.

The human visual system is very well adapted for this kind of hierarchical classifications. As mentioned in the example in the previous paragraph, we first try to classify the object into a higher level before going down to the specifics. The HVS can perform this higher-level classification with minimal amount of information. We are sometimes able to classify objects even without knowing its color, texture, smell, etc.

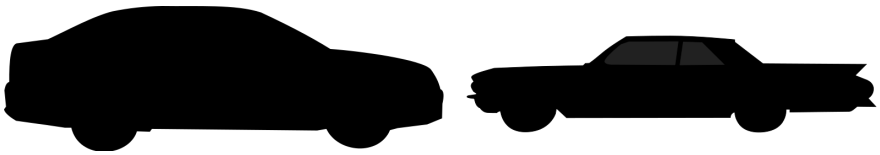


Figure 1.2: Car Silhouettes - The objects shown above can be recognised as cars even though it is just their silhouette that is visible. (Source: Google Images)

One of the steps in hierarchical object recognition is recognition of objects based on shapes. Shapes are important features of objects that can be used to uniquely describe a particular object, or at least, the object class. Many objects are purely dependent on their shape. The texture and color vary widely across different instances of the objects. In such cases, it becomes almost impossible to detect the category of the object. Shape, in such cases, is the only feature that remains consistent across different instances. Hence, in such cases, the only useful information that we can make use of in object classification is the shape information. Look at Figure 1.2 for example. The two objects in the image can easily be recognised as cars even though there is no information about their colour and texture. In certain cases of object recognitnion, it is sufficient if we have just the shape information. Extra information is usually not required for generic object classification. The aim of this PhD is to obtain better techniques for the recognitnion of such objects. Our research is focused on obtaining unique shape specific features that will help in object recognition.

1.2 Motivation

Many of the current day techniques concentrate on texture-based segmentation and color-based segmentation of objects from images. Such techniques work well only when the object has a distinctive texture or color characteristic. In addition, most of current day techniques work well for the recognition of specific objects. For example, if the training images contain a red snooker ball, the algorithm can be used to identify red snooker balls in other test images very well. However, if the test images contain a white snooker ball, the algorithms will fail to recognize it as belonging to the same class of "snooker balls". We would like it if the red snooker ball and the white snooker ball were both categorized under a broader class, namely, snooker balls. This requires shape analysis, as both the snooker balls have the same shape but different body color. There are many such examples that can be given, but are reserved for later chapters. It suffices to say that, not much research has been done in category-level object recognition and shape-based object recognition. Hence, there is a need to develop good techniques that will aim to achieve the same.

1.3 Research Objectives

The aim is to approach the problem of object recognition using shapes, in a hierarchical fashion. The first stage will involve the feature extraction stage, which will aim to obtain descriptive features from object shapes. These descriptive features will be used for localizing upon candidate objects that have similar features to the shape being searched. The features that will be produced are aimed to represent the shapes even after basic similarity transformations. In the second step, we plan to use the inter-feature relations for reducing false positives and hence aim to nail down the object that is being searched. The relations between the features are what make an object unique at the category level. Hence, we aim to utilize this unique property for class-level object recognition.

1.4 Report Organisation

The rest of the report is organised as follows. Chapter 2 surveys the literature in the field of shape recognition. The benefits and drawbacks of various current day methodologies are discussed. Also, the different primitives that can be used for shape analysis are examined and the importance of using corners as shape primitives is explained in detail. Chapter 3 discuses how corners can be detected on the shape contours using two different techniques. The advantages and disadvantages of both these techniques are also analysed in this chapter. Chapter 4 details the proposal for future work, which focuses on the development of feature descriptors for shapes. The corners are considered as feature locations and their adjacency to other corners is used as an important property for feature description. Finally, Chapter 5 summarises the report followed by some important references.

Chapter 2

Literature Survey

The importance of object recognition based on their shapes was brought to light in the previous chapter. An object has its own standard shape characteristics because of the various corners and edges that constitute its boundary. Hence, finding the relations between these primitives becomes an important part of shape matching. While corners contain the most eye-catching information in shapes, it is not enough to describe an object based on just its corners. For example, a square and a rectangle both have same number of corners and the same angles subtending at their corners. If we just use the number of corners to describe a shape, it would not be possible to distinguish between a square and a rectangle. This calls for methods to incorporate the relation between the corners as an extra metric for shape analysis.

Similarity of shapes can be quantified by using shape-matching algorithms. Shape matching deals with the matching of shapes under different scales, orientations and affine transformations. Given two shapes, the shape matching algorithms find out how similar they are to each other. Before the matching can actually take place, they should first be represented in some standard format that can be matched. [59] provides an in-depth review of different shape representations and description techniques. A shape can also be represented as set of features. These features should be able to uniquely describe the shape without any kind of ambiguity. The extraction of such features is an important step in analysing the shape characteristics. Features can either be region-based or

boundary-based. Section 2.1 describes the general principle of region-based feature descriptors, stating their benefits and situations where they fail. Section 2.2 describes shape matching techniques that are based on the shape of the object.

2.1 Region-Based Feature Descriptors

Region-based descriptors are those that take into account a small region of the image around an identified key point. Such descriptors make use of the gray level, the gradient map, etc, for describing the features. There are some very good feature descriptors for gray scale images. Ex: SIFT [33, 34]. This group of feature descriptors will be explained further by taking SIFT as an example. However, most of the claims made regarding SIFT will hold good even for other forms of region-based shape descriptors.

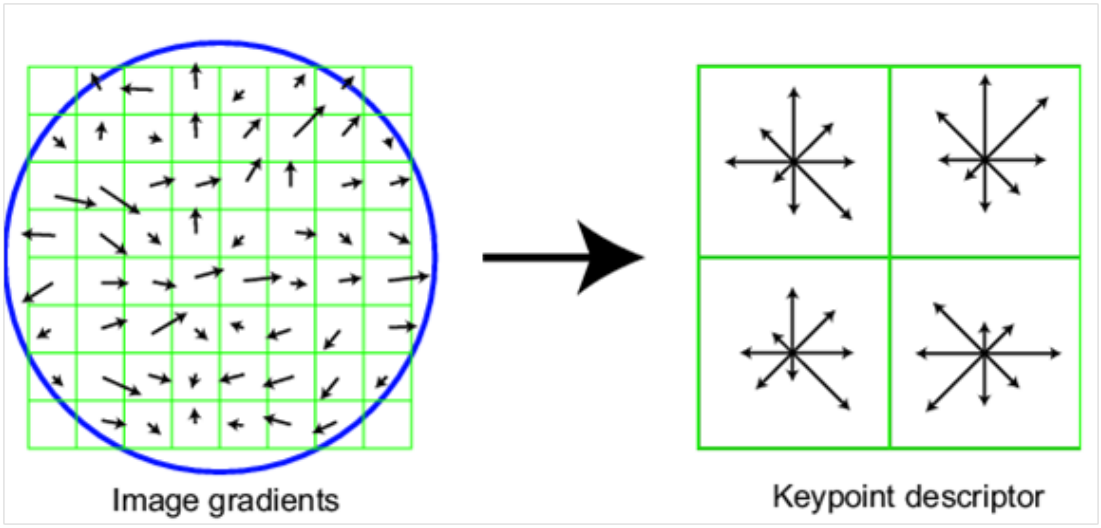


Figure 2.1: SIFT Descriptor - The figure shows a 2x2 descriptor. However, the actual implementation by Lowe, uses 4x4 blocks around the keypoint, with each block including 8 pixels. A histogram of gradients is generated for each of these 16 blocks, with each histogram having 8 bins. This results in a total of 4x4x8 = 128 dimensional SIFT vector (Source: [34])

The development of SIFT features has helped to solve a number of problems in the field of computer vision. The features have shown tremendous stability

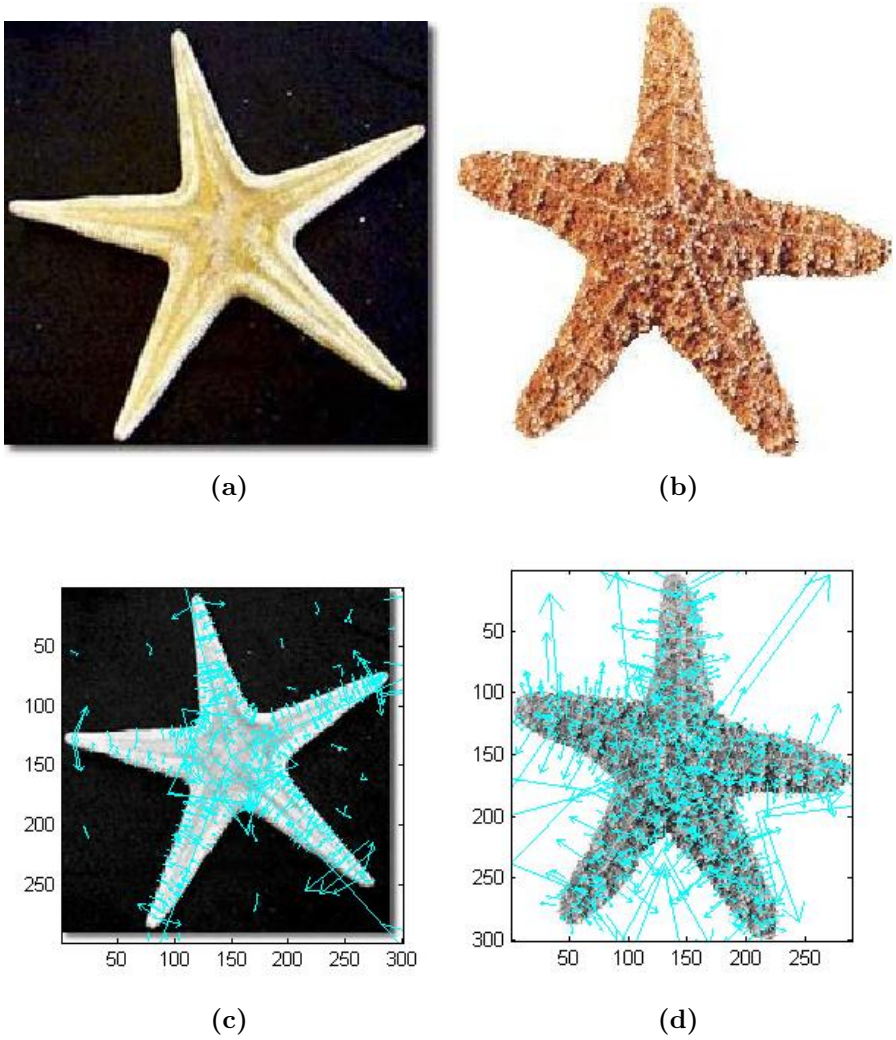


Figure 2.2: Star Fish - (a) and (b) The two starfishes are very similar in shape but, have different texture and colour; (c) 466 SIFT keypoints were found for the starfish in (a); (d) 747 SIFT keypoints were found for the starfish in (b). Even with such large numbers of keypoints, none of the keypoints found for the starfish in (a) matched any of the keypoints found for the starfish in (b).

in recognising objects at various scales, and is also invariant to translation and 2D rotation. The SIFT technique looks for interest points at various scales and obtains a gradient histogram of the region surrounding the key point (Figure 2.1). It then tries to match this high-dimensional vector with those obtained from the

test object. Though it performs well in matching certain types of objects, its approach is too localised. The high-dimensional vector of histograms does not have any semantic meaning when we are trying to analyse shapes. In addition, the feature vector does not account for spatial relations between neighbouring features. Hence, such an approach will not be sufficient for generic object detection. Moreover, it should be noted that not all objects have key points. Ex: A billiards ball is smooth throughout its body. SIFT will fail to obtain any kind of key point from such situations.

Describing features using the gray scale image might not be that useful when we are trying to match objects that have similar shape characteristics but different textures and colour. As stated in section 1.1.2, the only representation that remains consistent across such objects are their shapes. Take for example the two different star fishes shown in Figure 2.2. Both the fishes have a very strong and similar shape characteristics but, different body colour and texture. When the two images were given as inputs to SIFT, the algorithm failed to produce a single match even though there were a number of keypoints generated for both. Hence we can conclude that shape matching cannot be done by using region based descriptors. In such cases, the shape information that lies in the object's shape boundary needs to be extracted and represented in a descriptive manner. Therefore, there is a need for specialised shape descriptors that can do the job of shape matching.

2.2 Background on Shape Matching

Matching of shapes involves finding out how similar the given shapes are to each other. Such matching can be done either by using distance-based techniques or by using feature-similarity matching. Distance-based shape matching primarily involves matching two shapes using some sort of a distance metric. If the "distance" between the two given shapes is less, then the shapes are said to be similar to each other. Section 2.2.1 explains this technique in detail. Feature-similarity or descriptor-based shape matching involves the matching of the extracted shape features. If the features are similar to each other, then the shapes are said to be similar. More details in Section 2.2.2.

2.2.1 Distance-Based Shape Matching

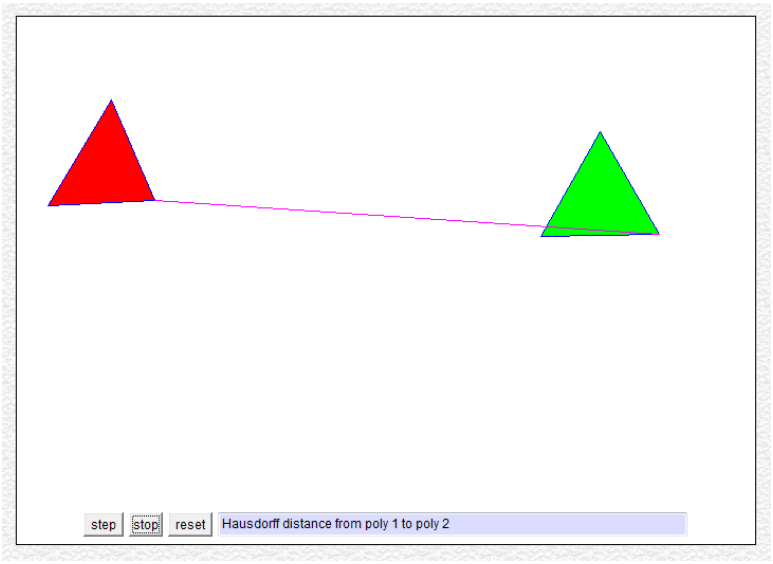
Given two shapes, their similarity could be computed using certain distance-based techniques such as the Hausdorff distance [26, 48] and the Chamfer distance [13, 47]. These distance-based techniques have been used for template matching. These techniques primarily compute the "distance" between the two given shapes and see how similar they are to each other. The smaller the distance value, the more similar the shapes are to each other. Equation 2.1 gives the general equation of Hausdorff distance.

$$h(A, B) = \max_{a \in A}\{\min_{b \in B}\{d(a, b)\}\} \tag{2.1}$$

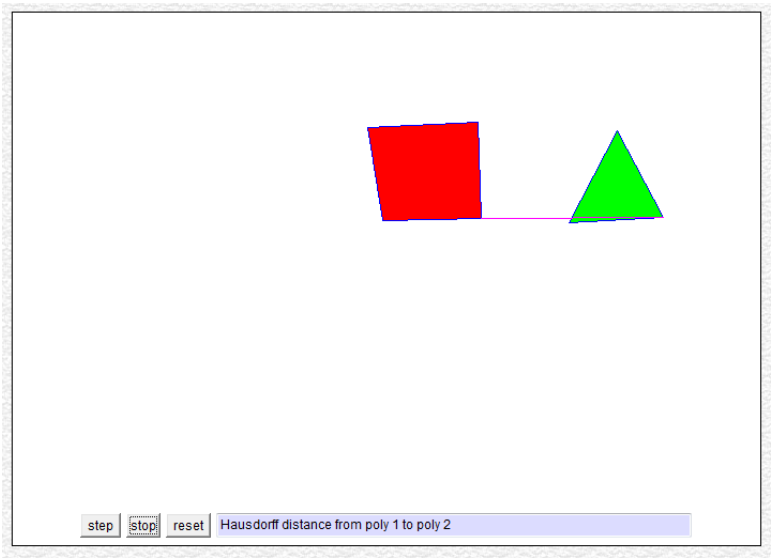
Where $h(A, B)$ is the Hausdorff distance between two sets of points A and B, a is a point in the set A, b is a point in the set B, and $d(a, b)$ is the distance between the point a and point b. The distance metric can be Euclidean distance or any other distance metric, based on the application. For shape matching, the most common distance metric that is used is the Euclidean distance. The distance obtained using Equation 2.1 is usually not used as the final metric for shape matching. This is because, $h(A, B)$ is not always symmetric ($h(A, B) \neq h(B, A)$). Also, $h(A, B) = 0$ does not imply that $A = B$. It only implies that $A \subseteq B$. If the Hausdorff distance is modified as given is Equation 2.2, it can then be used as a metric for shape matching.

$$H(A, B) = \max\{h(A, B), h(B, A)\} \tag{2.2}$$

Early implementations of these techniques for shape matching, were not scale and rotation invariant [5]. Consider this example. Let X, Y and Z be three sets of points with shapes of X and Y being similar to each other, while the shape Z being not-so-similar to either X or Y. Now, if the physical location of the shape Z is nearer to the physical location of the shape Y, when compared to the physical location of X with the physical location of Y, then $H(X, Y)$ will be greater than $H(Z, Y)$ even though X is "more similar" to Y than Z. Figure 2.3 illustrates the above example. The green triangle in the figure is the set labelled as Y, the red triangle is the set labelled X, and the quadrilateral is the set labelled Z.



(a)



(b)

Figure 2.3: Hausdorff Distance - The magnitude of the pink line joining the two shapes shows the Hausdorff distance between the two shapes. (a) The two triangles are similar in shape, but are physically far from each other, resulting in a large Hausdorff distance. (b) The physical locations of the two shapes are close to each other resulting in a small Hausdorff distance even though the two shapes are not similar to each other [22].

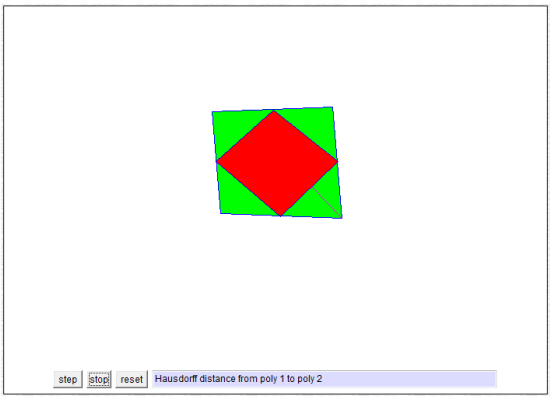
The magnitude of the pink line joining the two shapes in each subfigure, is the Hausdorff distance between the two shapes.

Clearly, Equation 2.2 does not account for translation. Recent techniques have been developed to make them invariant to the common transformations [27]. However, [27] does not account for rotation. The main idea that the authors of [27] have used for achieving translation and scale invariance is to make use of a sliding window and a multi-scale window, respectively. Such methods are considered as brute force solutions and it usually takes a lot of time to scan the complete image. Moreover, as can be seen from the examples in Figure 2.4, even after taking translation into account, it is not possible for us to obtain a similarity between two or more shapes. The Hausdorff distance metric is not invariant to rotation and scaling. Apart from the lack of invariance to similarity transformations, the Hausdorff distance metric is not a good measure for similarity between shapes. For example, even though the shapes of the two "red" polygons in Figure 2.4(a) and Figure 2.4(b) are quite different from each other, the Hausdorff distance between them and the "green" polygon is almost the same. This shows that high level similarity cannot be measured from such measures.

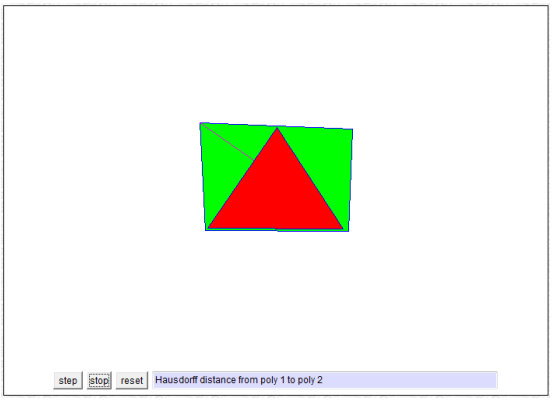
Chamfer distance is another distance metric, which was was first introduced by [7]. It is defined as the average of the distances from a nearest point in a set B to all the points in a template set A. Equation 2.3 defines the chamfer distance mathematically.

$$d_{cham}^{A,B} = \frac{1}{|A|} \sum_{\mathbf{x}_a \in A} \arg\min_{\mathbf{x}_b \in B} ||\mathbf{x_a} - \mathbf{x}_b||_2 \tag{2.3}$$

Where A and B are two sets that are being matched, $|A|$ is the number of points in the set A, $\mathbf{x}_a$ is a point in set A and $\mathbf{x}_b$ is a point in set B. The above equation is one of the most basic versions of the Chamfer distance. As in the case for Hausdorff distance, it can be seen from Equation 2.3 that the distance metric does not account for similarity tranformations. Researchers have been some modifications made to the above equation to incorporate invariance to translation. However, scale still remains a problem. Some authors have tried to introduce invariance to transformations using other techniques that are not related to distance matching. One example is the use of stereo to calculate the



(a)



(b)

Figure 2.4: Hausdorff Problems - (a) The figure shows the inability of Hausdorff distance to account for rotation. (b) The figure shows how a similar distance metric is given to the triangle, as was given to the rotated square shown in Figure 2.4(a), even though both are significantly different shapes [22].

scale and then use the Chamfer distance to compute the shape similarity [25]. Many of these invariance techniques are either application specific or require the use of external equpiment such as stereo vison cameras for scale analysis, which prevent them from being ported onto other systems. Hence, such distance-based techniques cannot be relied upon for robust shape matching.

2.2.2 Shape Descriptors

Obtaining feature descriptors for shape analysis is quite a difficult task to perform as the algorithm usually has just an edge map to work with. The features should be distinctive and easy to compute. Early work used shape silhouettes as the input data and tried to extract descriptors out of it. Fourier descriptors [39, 58] are an example of such shape descriptors. Fourier descriptors try to model the shape boundary as a periodic function and represent it in the frequency domain. Such a representation takes into account the spatial relations between the corners and hence produces a shape descriptor that represents the complete object. However, these methods are not widely used because there is a restriction on the input data to be a closed-contour silhouette, and the extraction of the same is not a simple task.

The logical direction to go into shape matching would be to use shape-specific features. The most important and descriptive features that could be used to represent an object shape are those obtained using the object's cue. An object's cue is that distinctive feature, which is apparent from the object's shape and uniquely distinguishes the same from the rest of the objects. Ex: A ball is spherical, a coffee cup has an opening at the top and a handle at the side, a bottle is mostly cylindrical with a cap at the top, etc. These are all high-level cues that help us to find the object being searched. The problem though, is in representing these high-level cues in a mathematical fashion. How do you represent the handle of a cup? How do you represent the cap of a bottle? They all come in so many different shapes and sizes.

Representing these objects as a combination of low-level primitives will be a good starting point. Researchers have tried to extract sets of primitives from a given image. Some even without the intention of expressing them as cues;

these primitives could be used as object cues if represented properly. Features such as lines, ellipses, etc, and their combination [40], are used to describe the objects. There are a number of line detection algorithms [6, 14, 17, 54], ellipse detection algorithms [17, 20], rectangle detection algorithms [29], etc. While these techniques are a step up from using distance-based techniques on an edge map, they still have problems in places where the spatial relation between the extracted primitives is to be acquired. All of these primitive generating algorithms "just" generate the primitives. Moreover, may of them output the primitives in a random order (or in an order resulting from a raster scan). There is just no way to obtain the adjacency of these primitives.

Bridging of the gap between target semantic classes and low-level visual descriptors is an unsolved problem. Classification of objects at an entity level requires the analysis of the characteristics of the object as a whole. Development of shape descriptors that describe the object globally have been looked into in the past [2, 10, 46]. The authors in [2] use a tree structure to maintain the keypoints and use the relations (distances) between the keypoints to uniquely identify shapes.

Belongie, Malik *et al.* have published a number of papers [8, 9, 10, 11, 36] leading to their final paper [10], wherein, they develop a shape descriptor, which tries to completely represent the object's shape characteristics. Their feature descriptor, named 'Shape Context', extracts a global feature for each point on the shape. Initially, the shape context at a particular point was defined as a set of vectors from that point to all other points on the object's shape. Figure 2.5 shows how a shape context at a particular point was thought of to be.

Later, it was found that vectors from every point to every other point made the descripter "too" rich, and hence the shape context was reduced to a histogram distribution of points relative to a given point. Figure 2.6 shows the circular histogram superimposed on a given point in the shape. The shape contexts are obtained for various points on a given image and are compared with the shape contexts obtained from various points in the test image. If the shape contexts produce a match for a threshold number of points or more, the shapes are pronounced to be similar. This is a very good way of representing the object shape as the distances are all relative to a point on the shape and also because,

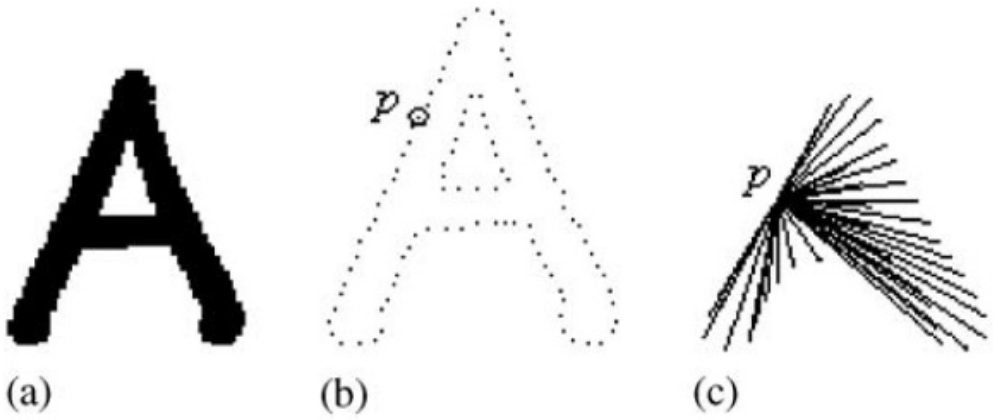
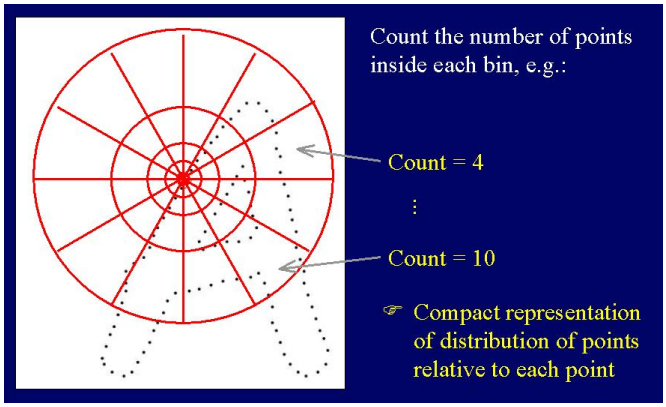


Figure 2.5: Shape Context - (a) The shape of a character; (b) Points sampled from the edge map of (a); (c) Vectors emanating from a point 'p' on the edge. (Figure taken from [9])

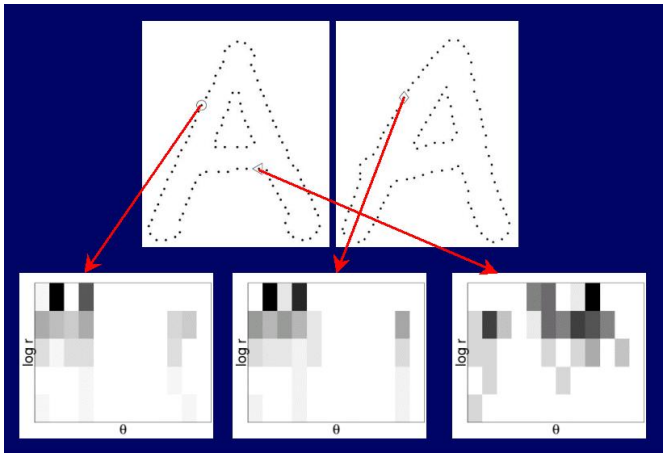
the shape context accounts for the global positioning of the different points on the shape, relative to a particular point. However, due to the sheer number of points that can be extracted from a shape, the matching might be cumbersome. A relatively efficient means of matching these points has been proposed in [37].

The problem with using the shape context is that, the list of points lying on the boundary of the object being considered should be known before vectors can be constructed from a relative point. In cluttered scenes and textured objects, we do not have this vital information and hence obtaining the shape context from a particular point is not possible. In addition, when the object is occluded, we might not get a complete context of the shape. Hence, this approach can be used only when the silhouettes of the objects have been extracted from an image and are available for matching.

Other shape matching methods such as the one proposed in [46] also require the input shape contour to be closed. Obtaining a continuous contour is not such an easy task. Apart from the traditional derivative-based edge detection approach, researchers have tried to obtain closed contours using probabilistic models [4, 44] and also by segmenting out regions [3]. Such techniques do produce better results than other edge detection algorithms like Canny [15] in certain



(a)



(b)

Figure 2.6: Shape Context Histogram - (a) The red circular disc is a log-polar histogram with twelve angular orientations and five distance bins. This histogram is placed over a point under consideration to calculate its shape context (b) The shape context at a given point is calculated by counting the number of points falling in each bin of the log-polar histogram. The shape contexts for three different points is shown in this sub figure [12].

cases. However, they are extremely time consuming and hence not suitable for real time applications.

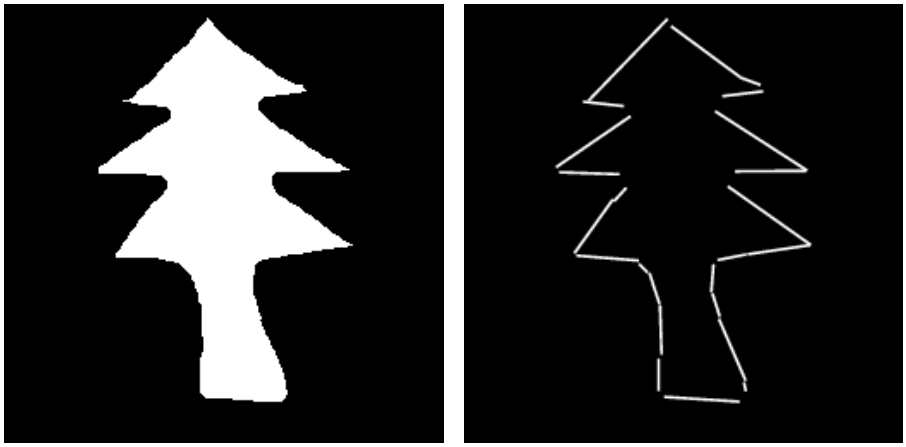
The problems that prevent complete contour generation are so fundamental that, no amount of higher-level techniques can solve them completely. Such persistent problems at the lower levels of image processing are hindering the de-

velopment of the field. These problems are because of the digitisation of real world analog signals, noise in primitive generation, etc. For example, the edge generating algorithms like Canny ([15]), produce broken edges and prevents algorithms from using the property of an object boundary being a closed contour. In addition, outliers, which are not part of the object boundary, are difficult to identify. If such outliers are considered for the description of shapes, we will not be able to get a perfect descriptor for the object. In order to alleviate these problems, we need to add in more intelligence at intermediate levels to facilitate robust high-level computer vision. More intelligence has to be added in the feature extraction stage. What primitives to choose, on what basis are they chosen, and, how to obtain distinguishable features using those primitives, are explained in the following section.

2.3 Primitive Selection for Shape Feature Description

The best way to obtain robust classification techniques is by using probabilistic approaches on strong descriptive feature sets. The feature descriptors that we would like to develop should help us overcome the problems mentioned above. Analogous to SIFT [34] (which works on interest points in gray scale images), we should develop features for shapes from the edge map. Those features should take into consideration, the spatial relations between the neighbouring primitives and uniquely describe a set of primitives for a given object. The problem now lies in primitive selection, for use in such cases. The primitives that could be used are lines, ellipses, parabolas, rectangles, etc. The list of primitives can go on. One way to choose the primitives could be based on the application that the algorithm will be working on. While this is logical, it would become difficult to generalise such an algorithm. In order to make the algorithms application independent, we have to choose a primitive that is universal.

We propose to choose corner points as the primitives. This is because, they are the lowest level of primitives that can be used to uniquely describe the shape of an object, if described adequately. There is no doubt that higher-order primitives



(a) (b)

Figure 2.7: Line Segment Detector Output - (a) The silhouette of a tree taken from the MPEG-7 database [1]. (b) The output of a line detector algorithm [54]. Notice the breaks in continuity near the corners. Also notice that, some of the edges are made up of multiple smaller line segments though the complete edge is in the same direction.

will be able to describe an object in better and more meaningful ways. However, finding the relations between the higher-order primitives is not such an easy task. Let us say we are searching for rectangles in a given image. If we were to describe a rectangle as a shape with four lines, then the feature extraction step would involve the extraction of lines in an image. The line extraction algorithms will be able to produce decent enough lines for the human eye to recognise the shape to be a rectangle. However, as mentioned in Section 2.2.2, the algorithms output the different lines extracted from the image in a random (or raster) manner. Consider the example shown in Figure 2.7. Figure 2.7a shows the silhouette of a tree taken from the MPEG-7 dataset [1]. Figure 2.7b shows the output obtained from a line segment detector [54]. We can see from Figure 2.7b that the output of such primitive detectors do not produce a continuous shape boundary. The Human Visual System is capable of joining such broken lines into a complete contour (by using illusory contours) because, the operating principle of the human brain is supposed to be holistic. This principle arises from Gestalt Psychology, which

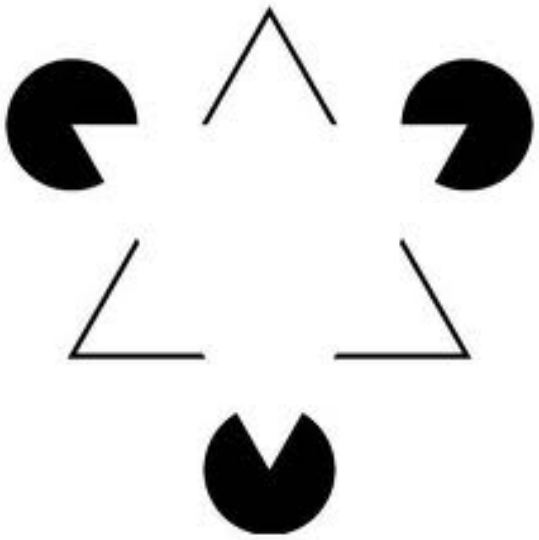


Figure 2.8: Kanizsa Triangle - The HVS is able to join the broken contours to see a "black" equilateral triangle and is also able to perceive a "white" equilateral triangle even though there are no edges forming the latter [56].

states that, "The whole is greater than the sum of parts" [55]. Look at Figure 2.8 for another example. Not only are we able to complete the broken contours of the upright "black" equilateral triangle, we are also able to perceive an inverted "white" equilateral triangle (above the "black" triangle) even though there are no contours creating the latter. There are some methods that can "complete" the object given parts of the contour, such as [57]. However, they require the corners of the broken contours to be present in the given image. Corner points are considered as seed points for directional contour following. This again, emphasises the importance of corner points in any given image for object recognition.

Hence, the methods that do not allow us to utilise the spatial relations between the primitives will not be helpful for holistic understanding of the objects. The same can be said about other higher-order primitive generation techniques like the ellipse, circles, etc. For that matter, even random order corner-detectors have the same problem. Such difficulties can be overcome to a certain extent if we use contours for the extraction of corners(Chapter 3), which we propose to

implement. The interconnections between the primitives can be worked out to understand the properties of the edges that connect them, which can then be used to understand the overall properties of the object. For starters, we extract corners out of the contours and describe features at corner locations. The contour following approach can also be used for higher-order primitive generation. A review of corner detectors is now provided in section 2.4. Further details about features description at corners and how the relation between neighbouring corners can be used for object recognition will be given in Chapter 4.

2.4 Corner Detectors

The importance of detecting corners was emphasised in Section 2.3. However, extraction of corners is not a done deal. The literature in computer vision has many papers related to corner detection, line detection, etc. Extraction of such primitives from the image is an important pre-processing step to object recognition. Though much research has taken place in the extraction of these primitives, there exists no method to find a relation between them.

An object boundary is an entity that consists of corners, which connect to other corners by lines or curves. Obtaining the relationship between the corners in the form of a corner adjacency map or any other means will enable us to represent the boundary logically and mathematically. A logical relation between the extracted primitives will benefit the computer vision society much more than a relatively random set of primitives. An object boundary is made up of a mixture of lines and curves that intersect with each other to form corners. A number of corner detection techniques have been proposed in the literature. Harris, [23] SUSAN [49] and FAST Corner detector [53] are some of the corner detectors that are frequently used in many applications. Harris corner detector is probably the most famous corner detectors of all and is hence described in Section 2.4.1. For an in-depth comparison between each of these corner detectors please refer to [51].

2.4.1 The Harris Corner Detector

The Harris corner detector [23] assumes regions of maxima to be corners. It calculates a cornerness value C for each point in the image. All those pixels that have a cornerness value C greater than some predefined threshold are declared to be corners. The algorithm initially calculates the intensity gradient in the X direction (I_x) and the intensity gradient in the Y direction (I_y). The intensity gradients I_x and I_y are the image derivatives in the X and Y direction, respectively. The first order differentials can be obtained by convolving the image with a differential kernel. Harris uses a 3x3 differential mask.

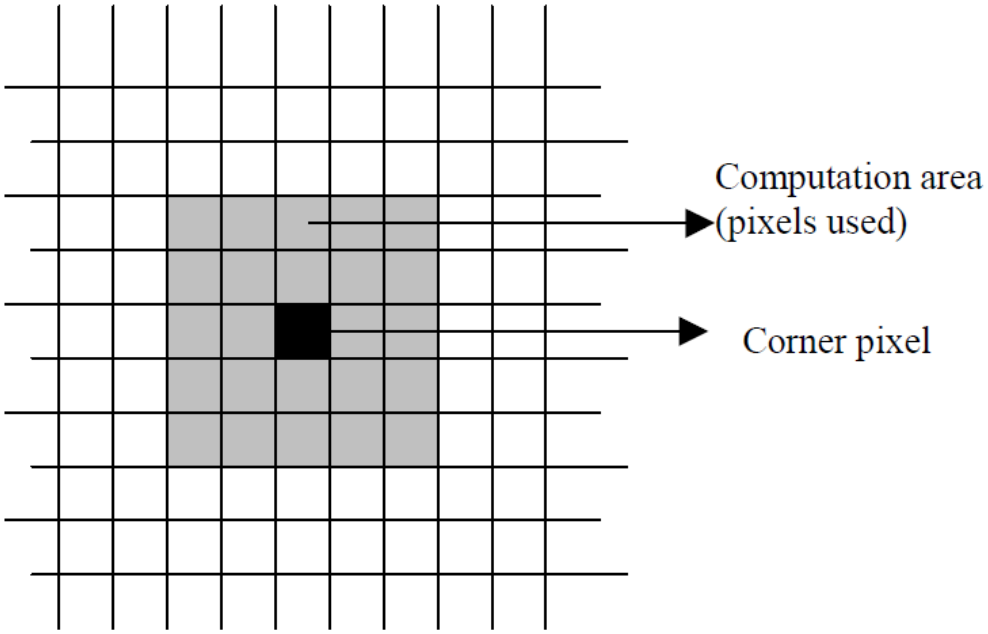


Figure 2.9: Haris Pixel Area - Pixel area that is considered while calculating the Harris cornerness value C when a 3x3 Gaussian kernel is used for smoothing [51].

These derivatives are now smoothed using a gaussian filter G, of some standard deviation σ, as given in Equation 2.4.

$$\begin{aligned} I_x^g &= I_x^2 \otimes G \\ I_y^g &= I_y^2 \otimes G \\ I_{xy}^g &= I_x I_y \otimes G \end{aligned} \tag{2.4}$$

Where $\otimes$ is the convolution operator. Finally, the cornerness value is calculated as given in Equation 2.5.

$$C = \frac{(I_x^g * I_y^g - I_{xy}^2)}{I_x^g + I_y^g} \tag{2.5}$$

If the cornerness value $C(x, y)$, at a given pixel (x, y), is greater than some threshold, say λ, then, the pixel at location (x, y) is considered as a corner. Figure 2.9 shows the area that is used when calculating the cornerness value C.

2.4.2 Problems With Area-Based Corner Detectors

Most of area-based corner detectors work in a similar way to the Harris Corner detector. The only difference lies in selecting the window size and shape, and, in the way they calculate the cornerness value. Ex: SUSAN [49] uses a circular window compared to the square window used by Harris. All these corner detectors detect "corner-like" features in real world images. Hence, it results in labeling many pixels that are not actually corners, but, are in fact surrounded by areas that produce a large cornerness value.

Below are four problems that are usually encountered with this category of corner detectors.

1. The corners are output in a random fashion, or in a raster order. This does not allow us to utilise the spatial relations between the detected corners.

2. While the window-based approach for identifying the corner points produces a high true positive rate, it also has a tendency to generate a large number of false positives.

3. If the window size is selected to be too small, there is a high possibility that the corner detectors miss rounded corners where the edge seems to be relatively straight within the constraints of the window.

4. Finally, they tend to localise on points that are close to the contour of the object, but not exactly on the object boundary. If the corner does not lie on the object boundary, finding the adjacent corner points becomes difficult.

Figure 2.10 shows pictorially, the different problems that we face while using area-based corner detectors, which were enumerated above. Harris corner detector was used on the image shown in Figure 2.7a. Figure 2.10a shows the order in which the corners were detected and output by the algorithm. We can see that the corners that are adjacent in the image need not be adjacent in the output. Figure 2.10b shows yellow boxes surrounding all those pixels that produced a cornerness value greater than a selected threshold. We can see that many points, though not corners, being labeled as corners by the algorithm. Figure 2.10c shows a zoomed in version of Figure 2.10b. The corners produced by the algorithm were close to the object boundary, though not on the object boundary.

Another approach to corner detection is by using the contours from the edge map. The advantage that the contour-based approach has over other window-based approaches is that, by using the former we can get more information about the object shape. Corner points are those points on the edge where a drastic change in direction is observed while travelling along the edge. Corner detectors, like the Harris corner detector, detect corners that lie close to the actual corners but not on them. In an experiment that was performed on a logical edge map of silhouettes, none of the corners detected by the Harris detector fell on the edge lines. All of them were close to corner points but none fell on them. If at all we are to work with corner points that are output from such algorithms, then it becomes difficult to follow the contour, which is required for finding the corner adjacency map.

In Chapter 3, we describe two different ways for obtaining the relation between the corners. The methods described are aimed to produce corner points on the object boundary with as few false positives as possible. These techniques would not only give us the corners on the object boundary, but would also readily give out the adjacency information for further shape description.

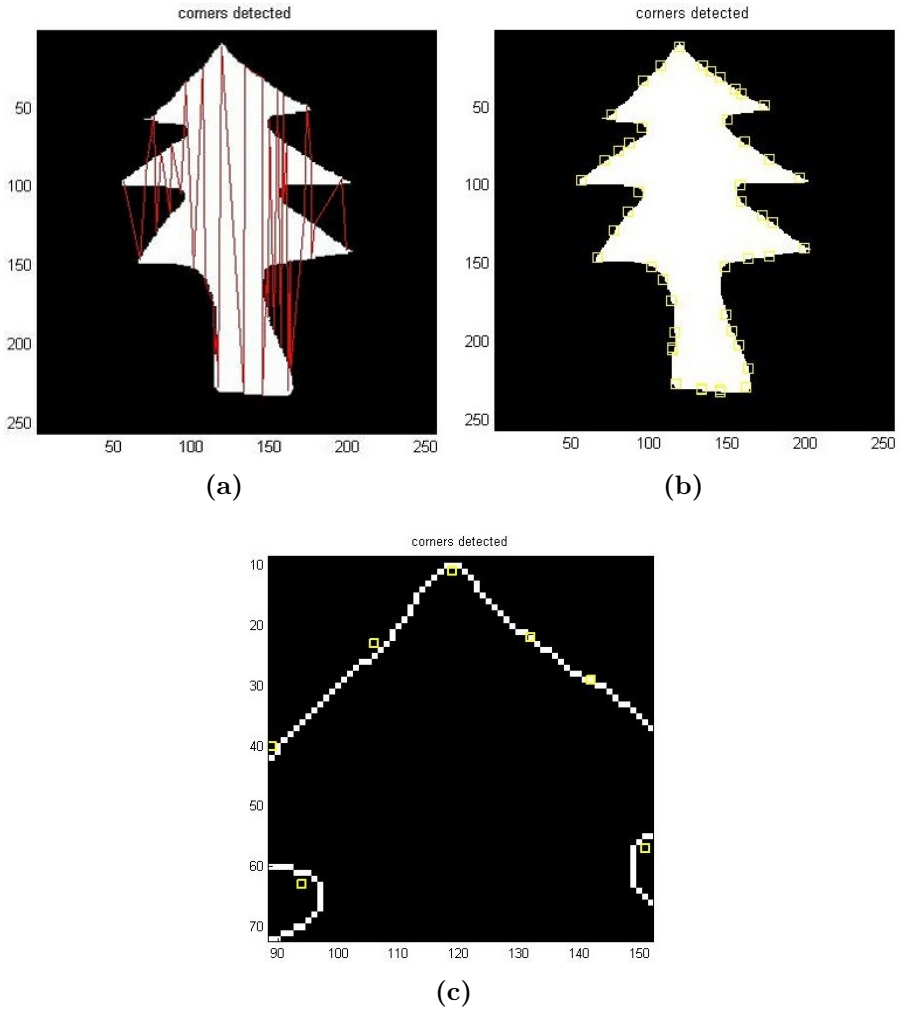


Figure 2.10: Harris Corner Detector Output - (a) The red lines show the order in which the corners were detected and output by the algorithm. It can be seen that the corners that are adjacent in the image need not be adjacent in the output. (b) The yellow boxes surround all those pixels, which produced a cornerness value greater than a selected threshold. It can be seen that many points, though not corners were labeled as corners by the Harris detector. (c) This is a zoomed in image of 2.10b. Here, we can see that, the corners were not localised on the contour of the object. They were "just" close to the object boundary.

Chapter 3

Contour Extraction And Corner Detection

The silhouette of the object contains much of the object's shape information. Hence, it is logical to say that starting with the object silhouette is a good way to start extracting the features. However, extraction of silhouettes is not an easy task. Objects of interest are usually found cluttered with other objects in the scene. Sometimes, other objects may even occlude them. Segmentation may help in the extraction of the object boundary, but it is a time consuming process. In addition, most segmentation techniques are not 100% accurate. In such cases, it becomes next to impossible to extract a well-defined object silhouette. As a result, we propose to use the edge map, obtained from any edge detector, and use the edge information to extract distinct features that can help identify the shape. The features are aimed to be distinctive so that interesting points on the object's contour can be uniquely identified from a cluttered scene. The features, once made distinctive can be used for classification using a bag-of-words approach. The detailed description of the features is given in Chapter 4.

The best locations for obtaining features are at the corners. As discussed in section 2.4 of Chapter 2, traditional corner detectors have their limitations and may not be of much use for corner extraction from contours. Hence, we aim to obtain corner points that lie on the contour using different methods. The points where there is a considerable deviation in the contour direction are the points that we are interested in analysing. These points can be labelled as corner points. The

definition of corner points and techniques for their extraction from contours will be given in Section 3.2. However, before meaningful features can be extracted from the contours, we first have to extract the contours themselves from cluttered images. Section 3.1 explains the contour extraction process.

3.1 Contour Extraction

The edge maps obtained from gray scale or RGB images are made of edge fragments. These edge fragments can consist of either the closed object contour or fragments of contours. The extraction of object shape information is relatively easier in the case of closed contour than in the case of open contours. The reason it is difficult in the latter case is because, the adjacency of the broken contours is usually not known. This prevents us from identifying the exact set of contours that make up the object's boundary. Hence, different techniques have to be developed for helping us identify the set of contours that form an object boundary. Before seeing how contour adjacencies can be obtained, let us first try to extract the different contour fragments from the image.

The edge map consists of many different contours scattered across a two dimensional area. The job of extracting the contours can be thought of as a segmentation process where the segmentation involves separation of all those points that are connected to each other. Any two points, $p1$ and $p2$, are said to be a part of the contour if there exists a path to reach $p1$ from $p2$ and vice-versa. Similarly, two points, say, $p3$ and $p4$, are said to belong to different contours, if there exists no path connecting the two.

Contour points can be classified into three types of points.

1. Endpoints
2. Junction Points
3. Intermediate Points.

Contour ends are those points, which are neighbouring to just one other contour pixel in a 3x3 neighbourhood. Figure 3.1 shows two examples of a contour

endpoint. There are 8 different orientations of contour endpoints that can occur in an image.

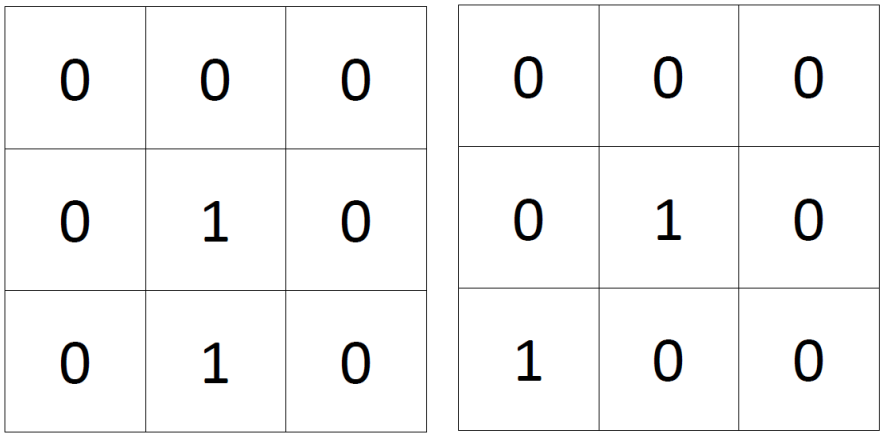


Figure 3.1: End Points - Two examples of end points that can occur in an image.

Apart from contour endpoints, real world images also have to take care of junction points. Junction points are those points where two or more edges intersect with each other. Junction points are present in the image mainly because of occlusion. The two major types of junctions are, the T-Junction and the '+' or 'X' junction. They are named so because of their visual appearance. Figure 3.2 show examples of junctions.

Intermediate Points are those points that lie in the middle of the contour. They are neighboured by exactly two pixels, each with a value '1'. The rest have values '0'. Figure 3.3 has examples of intermediate points.

In order to obtain all adjoining points, to a given point on the contour, we would need to traverse the contour in all possible directions radiating from the point under consideration. Traversing the contour in a multiple directions is not an easy job, as it would require the maintenance of additional data structures to keep track of the directions. As the number of contours increase, the cost of managing the data structures would also increase. Hence, we would like to perform the contour traversal from the ends of the contour, which would result in

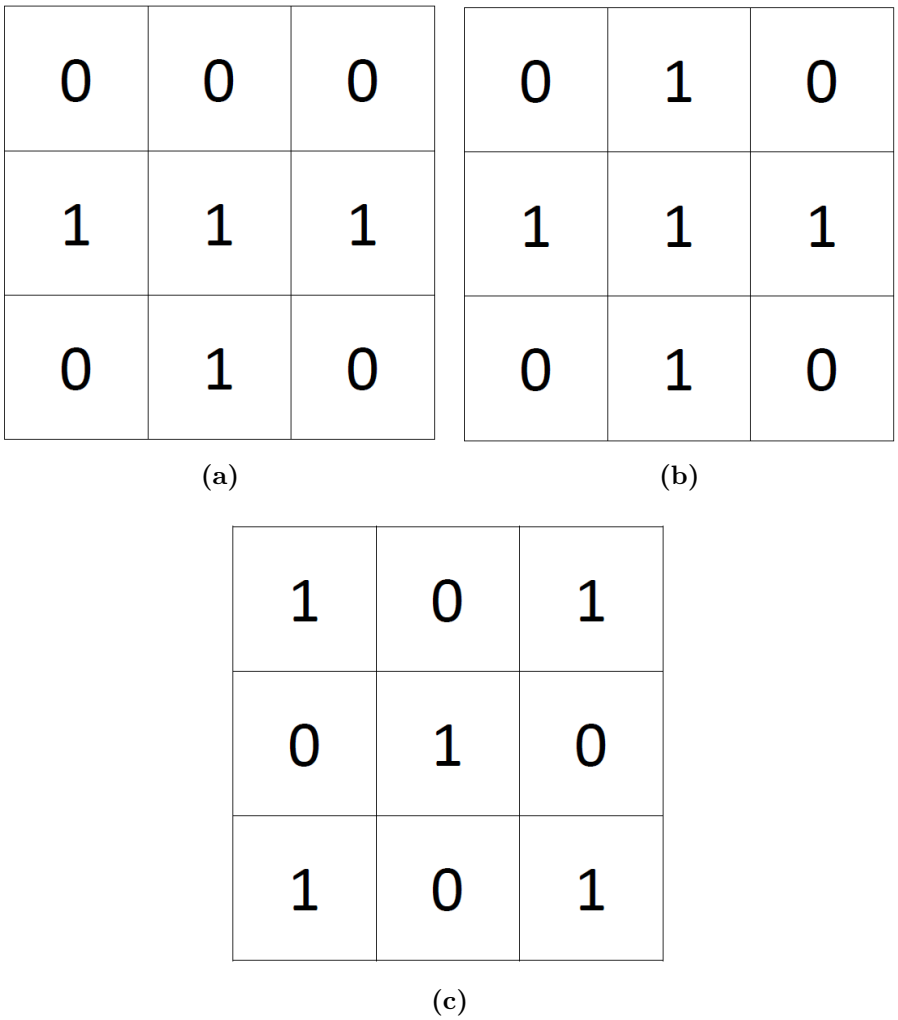


Figure 3.2: Junction Points - Examples of junction points that can occur in an image.

following the contour in just one particular direction. This requires us to identify all the contour ends in the image.

The endpoints and junctions have certain characteristics at pixel level that can be used for their identification. When we place a 3x3 mask on a binary image centred at an endpoint, we see a '1' at the centre of the mask and a '1' in one of the surrounding 8 pixels. The total sum of pixels in that mask excluding the centre pixel is 1. For a junction point, apart from the centre pixel, there exists at least

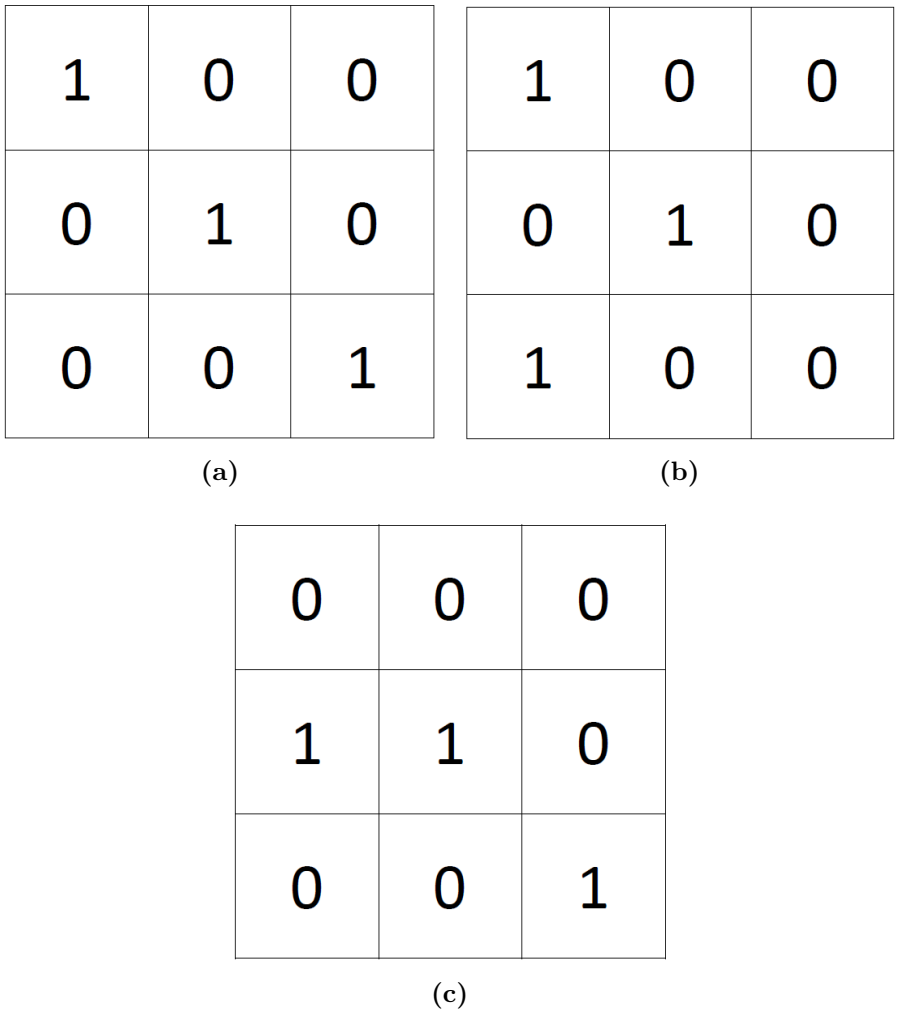


Figure 3.3: Intermediate Points - Examples of intermediate points.

three other pixels in the 3x3 neighbourhood that have a value '1'. Kovesi [38] uses an efficient method for the detection of such pixels. He calculates the number of transitions in the pixel values of the surrounding pixels, to the pixel under consideration, to decide whether the pixel is an endpoint, a junction point, or, an intermediate point. To calculate the transitions, he generates two vectors vect(a) and vect(b), where each vector is a list of the pixel values of the surrounding pixels. If we label the pixels in the mask as shown in Figure 3.4, we can generate the two vectors as given in Equation 3.1.

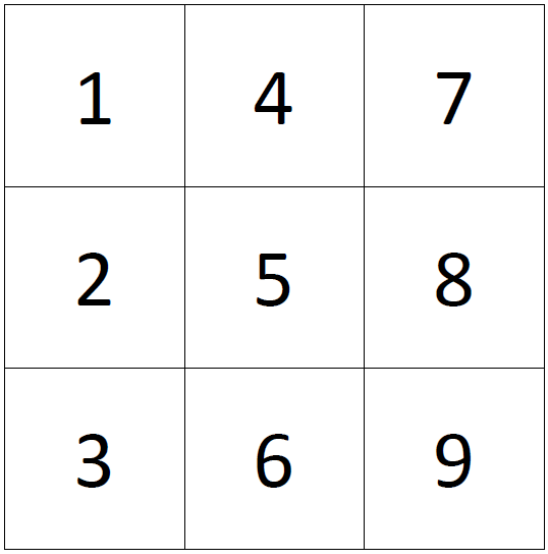


Figure 3.4: Pixel Numbering - Numbering of pixels in a 3x3 window.

$$\begin{aligned} \mathbf{a} &= [x(1)\ x(2)\ x(3)\ x(6)\ x(9)\ x(8)\ x(7)\ x(4)]^T \\ \mathbf{b} &= [x(2)\ x(3)\ x(6)\ x(9)\ x(8)\ x(7)\ x(4)\ x(1)]^T \end{aligned} \tag{3.1}$$

a is a list of pixel values taken in counter clockwise manner starting at the pixel labelled '1'. **b** is a list of pixel values taken in counter clockwise manner starting at pixel labelled '2'. Both of these pixels do not consider the center pixel as it is known to have a value 1.

To find the number of transitions, it is enough if we obtain the absolute difference between these two vectors. The number of transitions ($sum(abs(\mathbf{a}-\mathbf{b}))$ is used to classify the pixel under consideration. If there are just two transitions, then the pixel under consideration is classified as endpoint of the contour. If on the other hand, the number of transitions is greater than or equal to 6, then the pixel is labelled a junction point. A 3x3 window is placed on every pixel in the image and the number of transitions in the neighbourhood surrounding the pixel under consideration, is calculated. Based on the values generated, the pixels are classified according the to rules given above. All the endpoints and junction points identified, by the sliding window, are stored in separate arrays and are later used for contour extraction.

We should also remember that most real world images are peppered with noise. Noisy pixels are those pixels that occur in isolation. A pixel is considered as a noisy pixel if it has a value '1', with all its surrounding pixels having values '0'. We find all such pixels and replace the pixel's value with '0' to eliminate noisy pixels.

Now that we have the list of endpoints and junction points, we can obtain the list of contours in an image. A contour is defined as a set of pixels beginning at an endpoint or a junction point and terminating at another endpoint or junction point. Therefore, to extract the contours, we start by iterating through the list of endpoints. For each endpoint in the list, we follow the contour until we reach another endpoint or a junction point. If we reach an endpoint, then the endpoint that we started off with, and the endpoint that we just reached, are flagged off from the list. If we reach a junction point, the contour is still extracted, but the junction point is not flagged off, as there might be other contour fragments radiating out of the junction point. All intermediate points that are encountered in the contour following process are listed in a separate array. This process is repeated until there are no more endpoints to consider from the endpoints list. Once all endpoints are exhausted, we start iterating through the list of junction points and extract all contours radiating from each of the junction points, in a similar fashion.

Finally, if there are still some contour fragments that have not been traversed, it is only because those contour fragments are closed contours. In such cases, a point is selected randomly from the closed contour and contour following is performed in one of the two directions until the starting point is reached again. The whole process is succinctly listed in Algorithm 1. We now have all the contour fragments in the image stored as separate arrays for further processing. This "further processing" involves the extraction of inflexion points that can be used for object classification. Methods for corner detection are given in Section 3.2.

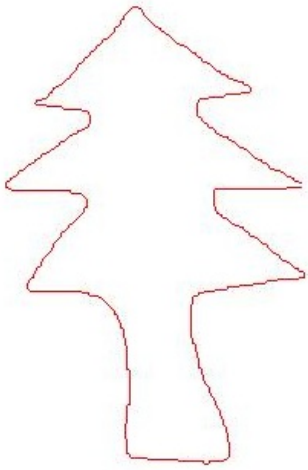
Below, in Figure 3.5, you can see the contours extracted from different kinds of images using the method just described above. Separate contours extracted from the respective images are randomly coloured to show the separation.

Algorithm 1 Algorithm for following contours and extracting them into separate lists

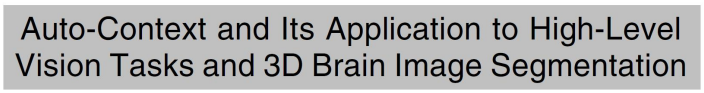
Obtain list of endpoints and store in list E;
Obtain list of junctionpoints and store in list J;

for each point p in list E **do**
  flag point p from the list E;
  follow contour until an endpoint of a junction point is encountered;
  store extracted contour as a list;
  if $k \in E$ **then**
    flag k from E;
    continue;
  else
    continue;
  end if
end for
for each point j in list J **do**
  for each untraced contour radiating from j **do**
    follow contour and store as a separate list;
  end for
end for
for each remaining closed contour **do**
  follow contour until the starting point is reached;
  store the extracted contour as a list;
end for

(a)



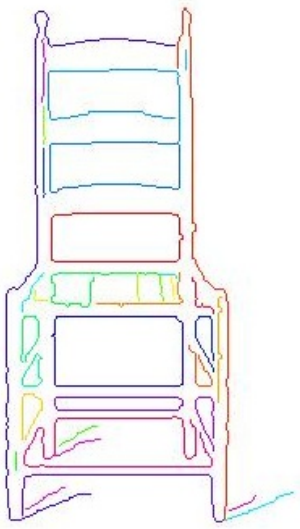
(b)



(c)

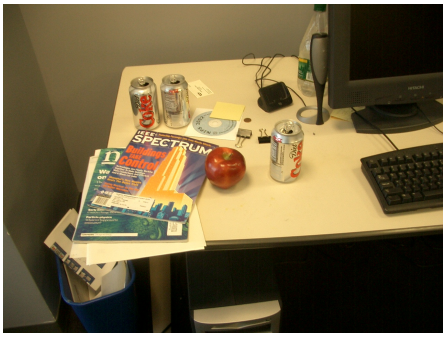
(d)

(e)

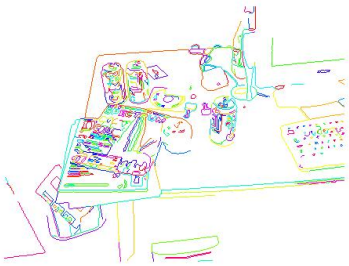


(f)

Figure 3.5



(g)



(h)

Figure 3.5: Contours - Contours extracted from the method described in Section 3.1 on different kinds of images. Separate contours extracted from the respective images are randomly coloured to show the separation. (a) Silhouette of a tree. (b) The contour of (a). (c) A text image (d) Contour image of the image in (c). (e) Image of a chair, taken from [18]. (f) Contour image of the image in (e). (g) A "real world" image taken from [45]. (h) The contour image of the image in (g).

3.2 Corner Extraction

Now that we have extracted contours from the image, we would like to uniquely describe those contour fragments. The contour fragments are analogous to strings, which can be of different lengths and shapes. We would like to extract this shape information from that contours that give it the unique shape property. It has been mentioned regularly throughout this report that corners are the points that contain the shape information in an object boundary. Hence, it would be beneficial to look at different methods of corner extraction from the contour fragments.

3.2.1 Polygon Fit Algorithm

There is a notion to define corners as those points where there is a considerable deviation in the contour direction. Let us visualise an open-ended contour fragment with at least one corner. This contour can be approximated to be consisting of two or more line segments intersecting at the points of "considerable"

deviation. This is similar to the idea of a curve being approximated to a set of line segments, and is known as the Ramer-Douglas-Peucker algorithm [16, 43]. Ramer [43] first introduced this idea in 1972 and the idea was also independently suggested by David Douglas and Thomas Peuker [16] in 1973. This idea has been modified to cater the needs in image processing. In the field of image processing, this algorithm is called as the polygon-fit algorithm, which is explained in detail in [21]. Below, is just a brief summary of how the algorithm works.

Let us visualise an open-ended contour with an imaginary line joining the two endpoints. The point that is farthest away from this imaginary line is bound to be a corner point, unless the contour fragment is in itself a straight line joining the two ends. This is the intuition behind the development of the algorithm. The method is to first align all pixel locations along the contour in an array, with the first and last pixels being the two endpoints of the contour (if it is an open-ended contour), or, with the first and last pixels being the left most pixel and the right most pixel respectively (if it is a closed contour). We then try to approximate the curve, between the first and last point, to be made up of a set of line segments.

To do so, a straight line is initially drawn between the first and last point in the array. Then, the point that is furthest away from this straight line is identified by using a pre-defined distance threshold. The distance between this point and the line must be greater than a particular set threshold value. The selection of this threshold value depends on the application. If the threshold value is set too low, then the algorithm becomes sensitive to outliers. If the threshold value is too high, then the algorithm will not produce the best polygonal fit to the given contour fragment.

Once the point that is furthest away from the line joining the starting two points is identified, a new line is drawn from the first point to this just-identified new point and from this new point to the last point. Now, the parts of the contour between the first point and the farthest lying point, and the farthest lying point and the first point, are treated as separate contour fragments and the whole process is repeated for these new contour fragments. This procedure is repeated until no new points are found that are at a distance greater than the selected threshold distance. Figure 3.6 provides a step-by-step illustration of the complete procedure.

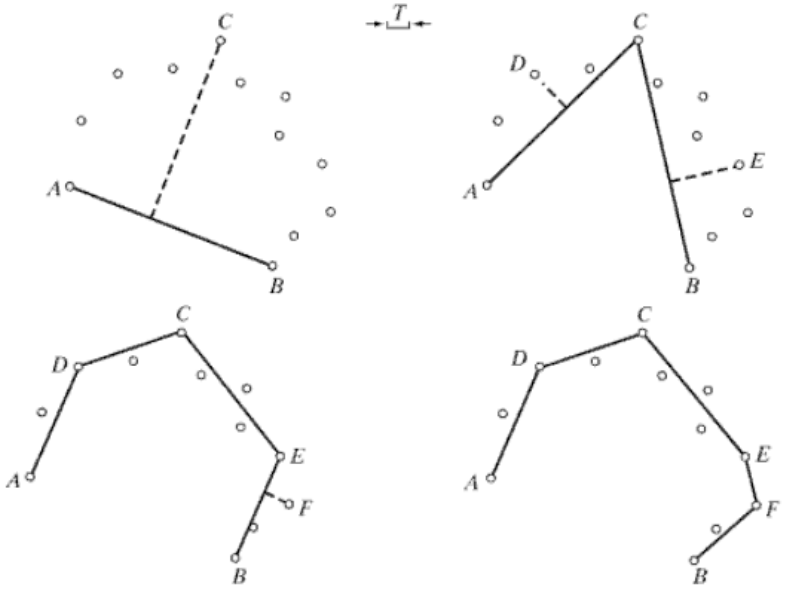


Figure 3.6: Polygon Fit - Open Contours - A step-by-step illustration of the polygon fit algorithm for open contours [21]. The algorithm starts with the state shown in the top-left figure. The figure in top-right shows that a vertex has been identified that is farthest from the initial line joining AB. Vertices D and E are next identified as shown in the bottom-left figure. Once the final vertex, F, is identified the algorithm stops and the result is shown bottom-right sub-figure.

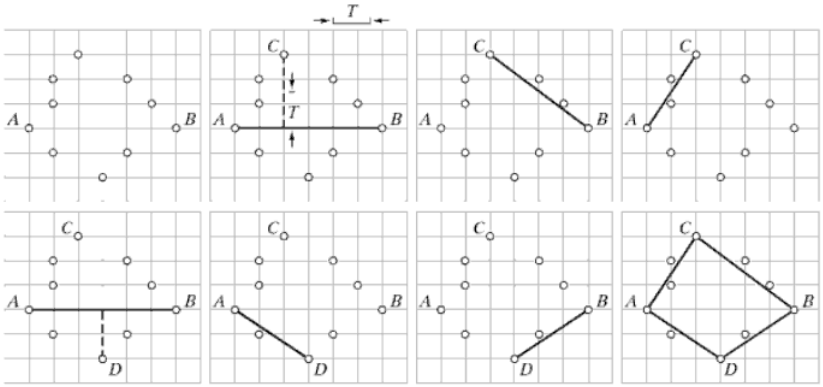


Figure 3.7: Polygon Fit - Closed Contours - A step-by-step illustration of the polygon fit algorithm for closed contours [21]. The sub-figures shown in the top row fit a polygon to the points lying above the line joining the points A and B. The sub-figures in the bottom row fit a polygon to the points lying below the line joining the points A and B. The final figure (bottom-right) shows the complete polygon that has been fit for the set of points shown in the first sub-figure (top-left).

It must be noted that, for contours that are closed, the polygon is completed in two phases. In the first phase, the polygon is fit for the contour that lies above (or below) the line joining the left most and the right most points. In the second phase the polygon is fit for the contour that lies below (or above) the line joining the first and the last points. The side of the contour that is handled first is chosen arbitrarily. Figure 3.7 illustrates this procedure.

3.2.1.1 Advantages and Disadvantages of Polygon-Fit Algorithm

The advantage of this algorithm is that, it is a very simple to implement and can be easily drafted for corner generation. The algorithm is able to identify all those points that lie farthest away from a line and thus producing the list of points, which can be considered as corner points. This may be a good solution for approximations of curves into line segments. However, it may not be sufficient if used stand-alone for corner generation because of the problems given below.

One of the disadvantages of this algorithm is that, it is highly sensitive to outliers. Noisy points that lie outside the contour will be detected as corner point even though they are just deviations due to jaggedness in the edge. Secondly, it is

localised in its view i.e., it does not consider the neighbouring contour curvature while identifying corners. This means that, the algorithm will end up labelling the rounded corners as corners. There is no clear definition of a corner in the literature and hence it becomes difficult to develop any algorithm for corner identification. However, the decision for rounded corners has to be made based on the surrounding contour curvature and not just the curvature at one particular point. Consider the example of a circle. The algorithm will identify a set of "corners" even though a circle is a figure with a smooth contour with uniform curvature. In addition, certain points might appear to be corners when viewed on a local scale. However, they may just be deviations on the contour when viewed in a global scale. The polygon-fit algorithm does not consider the global properties of the contour. Other algorithms like [30] and [35], also end up considering only the local contour properties. These problems make this method of corner generation unreliable.

3.3 Corner Detector Based on Local and Global Properties

The problems mentioned in the previous section call for the development of multi-scale corner detection techniques. The authors of [24] have developed an algorithm that is good enough for identifying corners on the contour with very high precision levels. They boast their algorithm to have very low error rate, with the errors occurring ony at rounded corners, or when the "corners" do not appear to be corners at a global scale while they are at lower scales. Such ambiguity exists even when humans are asked to list out the corners. Some might term rounded corners as true corners, while others might label them to be curves in the object silhouette and not as corner points. The algorithm slowly converges onto the real set of corners in multiple stages. Below, we summarise their algorithm pointing out the advantages that it has over its competitors.

The algorithm starts as any other contour following algorithm by extracting the contours from any given image (section 3.1). The contours are then checked whether they are closed or open. Since the canny edge detector tends to produce

broken edges, the algorithm tries to join them using simple methods. A threshold is set to check if gaps in contours are because of breaks in the contour or because the contours are totally unrelated to each other. If the gap between two or more contour ends is greater than the selected threshold, then the contours are assumed to be separate contour fragments. If, on the other hand, the gap is less than or equal to the selected threshold, then the contours fragments are joined into a bigger contour by filling in the gaps. This is an essential step as it helps in increasing the stability of later stages.

Upon contour joining, the algorithm proceeds towards corner detection by calculating the curvature at each point on the contour using the equation given in Equation 3.2.

$$K_i^j = \frac{\Delta x_i^j \Delta^2 y_i^j - \Delta^2 x_i^j \Delta y_i^j}{[(\Delta x_i^j)^2 + (\Delta y_i^j)^2]^{1.5}} \tag{3.2}$$

Where j is for the jth contour, i is for the ith pixel in the jth contour, $\Delta x_i^j = (x_{i+1}^j - x_{i-1}^j)/2$, $\Delta y_i^j = (y_{i+1}^j - y_{i-1}^j)/2$, $\Delta^2 x_i^j = (\Delta x_{i+1}^j - \Delta x_{i-1}^j)/2$, $\Delta^2 y_i^j = (\Delta y_{i+1}^j - \Delta y_{i-1}^j)/2$, x_i^j is the x-coordinate of the ith pixel in the jth contour, and y_i^j is the y-coordinate of the ith pixel in the jth contour. This standard equation is used in various Curvature Scale Space (CSS) algorithms. The deviation from other standard procedures in corner detection comes in the following step. While other CSS algorithms hastily label the points with maximum curvature as the corner points, the algorithm in [24] postpones the labelling for until after the analysis of the curvature values. The algorithm makes use of the curvature values for eliminating round corners and for eliminating false corners. Figure 3.8 shows the comparison between round corners and obtuse corners. The curvature values of the obtuse corners are significantly larger than the curvature values of its surrounding pixels. However, in the case of round corners, the rate of change of curvature values is not so drastic. The algorithm uses this property to eliminate the rounded corners. The authors define a surrounding area, namely, *Region of Support* (ROS) to decide the fate of candidate corners. The ROS of a corner is defined as "the segment of contour bounded by the corner's two nearest curvature minima". If the absolute curvature of the candidate point is greater than the mean curvature of the points in the ROS, then the candidate is retained for

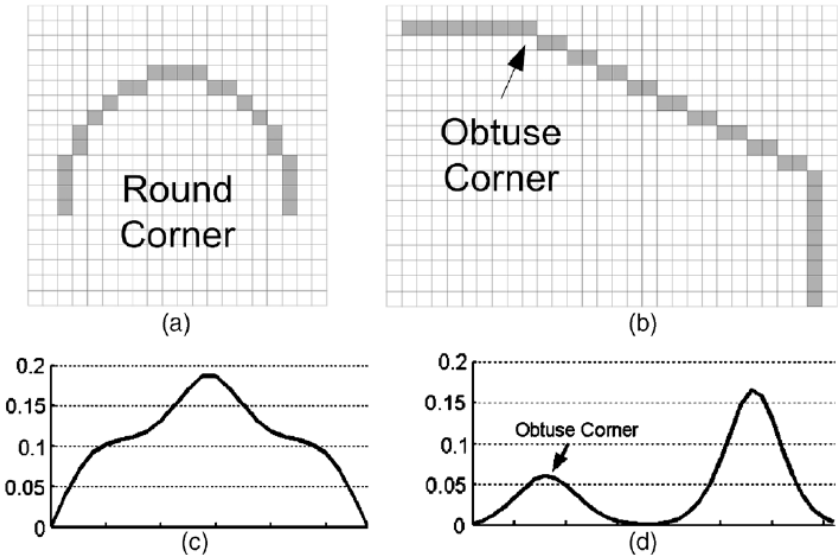


Figure 3.8: Round and Obtuse Corners - (a) Contour of a round corner (b) Contour of an obtuse corner (c) Map of the curvature of the pixels in the round corner's contour, and (d) Map of the curvature of the pixels in the obtuse corner's contour [24].

further processing. Else, it is treated as a rounded corner and is eliminated from further processing.

The second advantage of this algorithm is its power in false corner removal. An ambiguous case of a false corner is shown in Figure 3.9. If the point numbered '3' is looked at a single scale, locally, then there is a high chance that the point may be labelled as a corner point. However, when we look at the complete contour, there is a feeling that the point numbered '3' may have developed due to jaggedness in digitisation and edge detection stages. In order to eliminate such false corners, the algorithm uses multiple scales for corner point analysis. The scale is varied by varying the ROS around a point under consideration. The varying of the ROS takes place automatically as the algorithm iterates over multiple steps. In the initial step, the points numbered '2' and '4' may be eliminated after round-corner removal. This, in effect, would change the ROS of point numbered '3' in successive steps. In the step following the round corner removal, the ROS of point '3' now spans up to points '1' and '5'. It is here that the point '3' appears to be

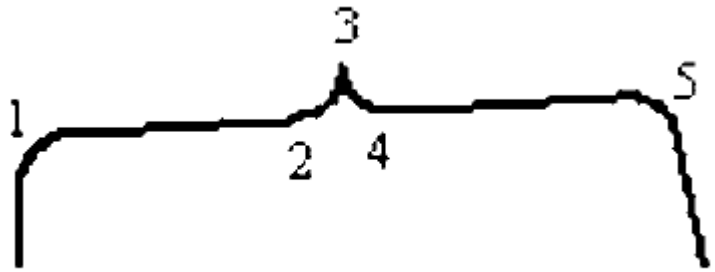


Figure 3.9: False Corners - Contour containing a debatable corner [24].

an aberration and removed from further processing. This complete procedure is repeated successively until there is no more change in the number of corner points. The final set of corner points, which are obtained after successive iterations, are considered as the true set of corner points. This true set can now be used for further processing and understanding of the image and objects in it.

3.4 Summary

In this chapter, we have seen how the contour extraction takes place from gray scale images. We have also seen different ways of corner point extraction from the contours. The problems in traditional approaches and the improvements brought about by recent developments have helped the extraction of those points that appear to be as corners for the human visual system. Automatic implementation of these stages has been completed. We are now looking towards different techniques for uniquely identifying contour fragments using characteristics that are specific to a contour from a particular object. In the next chapter, we propose the future work that we would like to pursue and the direction that we would be keen to take, which would lead us towards our final goal.

Chapter 4

Feature Selection and Proposed Work

Now that we have extracted contours from images, and the corner points from the contours, it is time to represent this corner information in a meaningful manner, for use in learning and classification. In this chapter, we will look at what information from the corners can be used for feature description, their stability to similarity transformations, and how a learning algorithm can use these features for the classification of new data.

4.1 Corner Feature Description

Corners extracted from the techniques in the previous chapter consist of all those points on the contour where a considerable deviation in the contour direction can be seen. Moreover, the algorithm also eliminates most of the false corners and does not include round corners in its output. With this set of corners at our disposal, we now look forward towards extracting features that are unique to the corners in a given contour.

One of the most important and noticeable features that can be extracted from the corners are the angles subtended by the contour fragments meeting at these corner points. Corner angles are stable and robust features that can are invariant to translation, 2-D rotation and uniform scaling. Moreover, a set of corner angles can be used to uniquely identify an object to a high degree of accuracy. Ex: If

we were to stumble upon a closed contour fragment with three corner angles, say, 70 degrees, 50 degrees, and 60 degrees, it is evident from the information that the contour fragment must result in a triangle. This is true only when the polygons are convex polygons. To account for concave objects, we will introduce other supporting information to make the distinction. Information such as the direction of the vector bisecting the corner angle is one way to account for concave shapes.

Robustness to some of the above-mentioned transformations makes corner angles the most important feature that can be used to denote the properties of a corner. However, unless we are dealing with synthetic data, it will be hard to obtain consistent values of corner angles. Even in the cases of synthetic data, there is bound to be some small amount of variation in the values of the corner angles. Hence, we plan to model the distribution of corner angles as a gaussian.

Let us say we sample the value of the corner angle from a particular corner, from multiple training examples. This set of corner angles is given in Equation 4.1.

$$\Theta_i = \{\theta_i^1, \theta_i^2, ..., \theta_i^j, ..., \theta_i^m\} \tag{4.1}$$

Where Θ_i is the ith corner angle set and θ_i^j is the jth sample of corner i, and m denotes the total number of training samples. The object that we are training can now be represented in a compact form as given in Equation 4.2

$$Ob_k = \{\Theta_1, \Theta_2, ..., \Theta_i, ..., \Theta_{n_k}\} \tag{4.2}$$

Where Ob_k is the kth object, Θ_i's are the corner angles, and n_k is the number of corners in the kth object. We can now model each of these corner angles into a gaussian by calculating their means and standard deviations. The gaussian model of a particular corner can be given as in Equation 4.3.

$$\mu_i = \frac{\sum_j \theta_i^j}{m}$$
$$\sigma_i = \frac{\sum_j (\theta_i^j - \mu_i)^2}{m} \tag{4.3}$$
$$N_i(\theta; \mu_i, \sigma_i) = \frac{1}{\sqrt{2\pi}\sigma_i} e^{-\frac{(\theta-\mu_i)^2}{2\sigma_i^2}}$$

Where μ_i is the mean and σ_i is the standard deviation of the angles at the ith corner, and $N_i(\theta; \mu_i, \sigma_i)$ is the gaussian model that is fit to the ith corner angle.

When we have modeled all the corner points in an object using the training examples, we would then have n_k gaussian models, at the max, representing the class that the kth object belongs to, in a 1D visualisation, or, a single gaussian in the n_kth dimension. If the polygon has two or more corners with similar corner values, then the number of gaussians for the object's class, in 1D space, will reduce by the same amount. Ex: Each of the corner angles in a square is equal to 90 degrees and hence, we will have only one gaussian for the object class, square. The impact of the corner can be accounted by assigning weights to each of the n_k gaussians in the 1D visualisation.

4.2 Training and Testing

With the modelling of corner angles done, we are left with training a probabilistic algorithm for classification purposes. We propose to use the Bayes classifier for this purpose. Two different versions of the Bayes classifier are given in Equation 4.4. One is the binary classifier and the other is a multivariable classifier.

$$\arg \max_{i \in \{0,1\}} \; P(Y = i|\mathbf{X})$$
$$\arg \max_{i \in \{1,2,3,\ldots,k\}} \; P(Y = i|\mathbf{X}) \tag{4.4}$$

Where $\mathbf{X}$ is the test feature vector, Y denotes the output class, and k is the total number of classes. Given a test contour, we can extract the corners in the contour and represent it as a feature vector as shown in Equation 4.5.

$$\mathbf{X} = [\theta_1, \theta_2, ..., \theta_{n_t}]^T \tag{4.5}$$

We refrain from using the subscript n_i and instead use the subscript n_t in the above equation because, the extracted number of corners may or may not be the same as the number of corners identified during training. There are bound to be noisy data, which may result in the identification of greater than, equal to, or less than the identified number of "true" corners n_i for a given object Ob_i.

The orientation of the extracted corner angles need not be the same as the orientation of the corner angles used during training. A set of corner angles for a given object are bound to be cyclic when extracted using the contour following approach. In order to find the mapping between the corner value and the original corner, we find the best matching gaussian model for the input corner angle. We can map a gaussian model corresponding to θ_i by finding the maximum likelihood from the set of training gaussians as given below in Equation 4.6.

$$\arg \max_{i \in \{1,2,...,n_k\}} L(N = N_i|\theta_i) \tag{4.6}$$

Here, L is the likelihood function given by the normal equation shown in Equation 4.3. If θ_i maps to the gaussian model N_i, then the feature vector $\mathbf{X}$ given in Equation 4.5 can be transformed into the feature vector as given in Equation 4.7.

$$\mathbf{X}_N = [N_1, N_2, ..., N_{n_t}] \tag{4.7}$$

For simplicity, $\mathbf{X}_N$ will henceforth be denoted as just $\mathbf{X}$. The classification can now be done by finding the maximum likelihood at the object belonging to a given class as in Equation 4.4 above. Individual probabilities for calculating the maximum likelihood can be obtained using Bayes rule (Equation 4.8).

$$P(Y = i|\mathbf{X}) = \frac{P(\mathbf{X}|Y = i)P(Y = i)}{P(\mathbf{X})} \tag{4.8}$$

The individual elements in the RHS of Equation 4.8 are obtained using the training examples and can be calculated as given below in Equation 4.9.

$$P(Y = i) = \frac{\#\ of\ occurrences\ of\ object\ \text{“}i\text{”}\ in\ training\ set}{total\ \#\ of\ objects\ in\ the\ training\ set}$$

$$P(\mathbf{X}|Y = i) = P(N_1, N_2, N_3, ..., N_{n_t}|Y = i) \tag{4.9}$$

$$P(\mathbf{X}) = \sum_{i \in 1,...,k} P(\mathbf{X}|Y = i)P(Y = i)$$

Traditional bag-of-word approaches assume conditional independence while calculating the value of the term $P(\mathbf{X}|Y = i) = P(N_1, N_2, N_3, ..., N_{n_t}|Y = i)$. However, we would like to make use of the additional information that we have for the same calculation. The fact that we have followed a contour-based approach for corner extraction means that we have the information of corner adjacency. What the corner adjacency does is that, it gives us more information about the occurrence of a particular corner in relation to other corner angles. This additional information will help us in achieving higher accuracy than the traditional bag-of-words approach where adjacency of features is not taken into consideration.

This also makes sense intuitively. A set of corners is hardly enough to ascertain their source. However, if the set of corners were provided as inputs along with their adjacency information, it would be easy to reconstruct the object under consideration and get closer matches during classification.

The complete and most generic solution to the term $P(\mathbf{X}|Y = i)$ would be as given in Equation 4.10.

$$P(\mathbf{X}|Y = i) = P(N_1, N_2, N_3, ..., N_{n_t}|Y = i) =$$
$$P(N_1|Y = i)P(N_2|Y = i, N_1)...P(N_{n_t}|Y = i, N_1, N_2, ..., N_{n_t-1}) \tag{4.10}$$

In fact, even a cyclic permutation of the above equation (as given in Equation 4.11) should hold good for calculating the probability value.

$$P_{Ob_i}(N_1, N_2, N_3, ..., N_{n_t}|Y = i) = P_{\{\pi_{k\in\{1,...,n_t\}Ob_i}\}}(\pi_{k\in\{1,...,n_t\}}(N_1, N_2, N_3, ..., N_{n_i})|Y = i) \tag{4.11}$$

Let us consider the above equation for the two variable case. What Equation 4.11 states is that, $P_{N_1,N_2}(N_1, N_2|Y = i) = P_{N_2,N_1}(N_2, N_1|Y = i)$. Such a classifier is robust only for rigid objects, which have strong corners that are consistently appearing at the same locations. This method cannot be used for deformable objects and when the projection of the object results in an oblique shape boundary. In order to account for deformable objects, we would need more information about the contours. Information from just the corner points would not be enough for such comparisions. We aim to tackle the cases of deformable objects a part of our future work. One way of solving the problem caused by oblique projections is to build up a database with all different projections and use that in our learning algorithm.

4.3 Proposal for Thesis Work

In this report, we have introduced the intuition behind the contour based approach for object recognition. The first part of our future work would include the testing of the above-mentioned method. We would like to see how the probability values obtained could be used to reduce the number of contours that will be remaining for matching in the subsequent steps. We will proceed to measure the robustness of the corner points in the identification of contour fragments that are likely to belong to the object being searched and improve on the set of corner points by identifying them at different scales. Searching over different scale spaces would be similar to the work done by [34], as was originally proposed by [32]. We would like to take this forward by adding in more robustness to the parameters by the introduction of supporting information to the corner angles.

One way to add in supporting information could be by incorporating the direction of the vector bisecting the corners on the non-reflex side of an angle. This direction would help us in distinguishing between concave closed-contours and convex closed-contours. All the vectors on the non-reflex side would point towards the centroid of a convex-closed contour. On the other hand, for a concave closed contour, there is bound to be at least one vector that is pointing away from the centroid of the object.

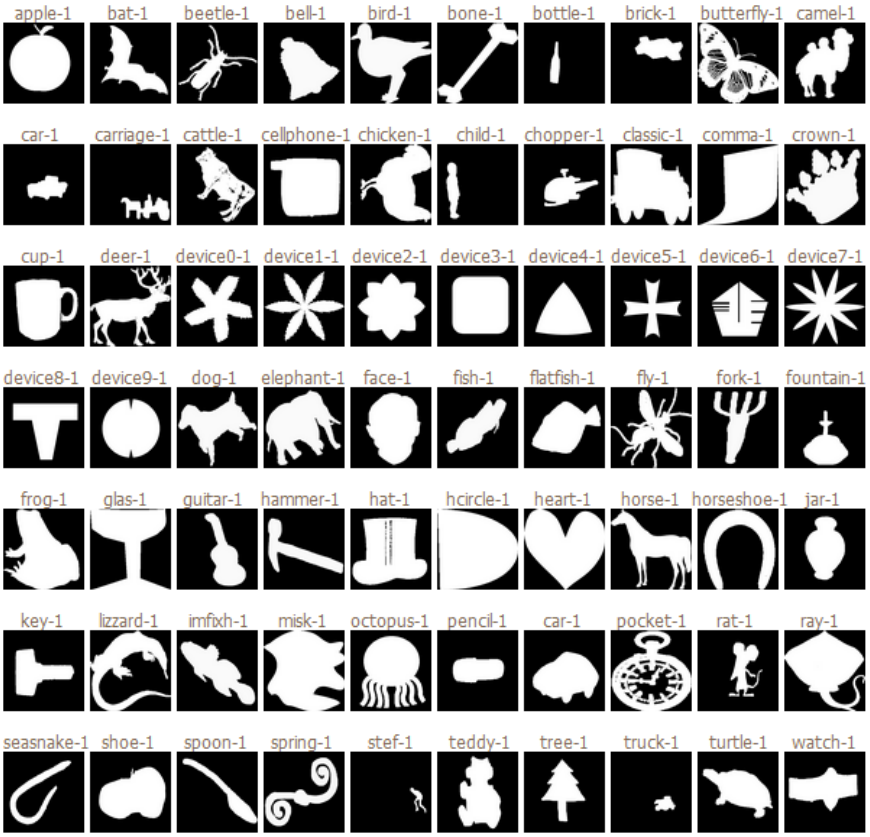


Figure 4.1: MPEG-7 Database - A collection of images from the MPEG-7 Database. There are a total of 70 subimages. Each subimage shows one example from a particular class [42].

Corners cannot give out more information than the curvature (or angle) at the point and the direction of the corner opening. So, this information along with the adjacency information will be used in our process of curve matching (eventually leading to object recognition) in narrowing down the list of contours that have to be scanned or matched to identify an object. These tests would be performed on the MPEG-7 database [1], which is considered as a standard for 2-D shape analysis. The MPEG-7 dataset is a strong dataset to test out the claims that we have made as it hosts a varied number of test images. It has a total of 1400 images made of 70 different classes with each class housing 20 deformed shapes. Figure 4.1 shows some of the example images from the database.

There is also a lack of good "real world" database with images whose projective planes are known. The MPEG-7 database is a 2-D database with silhouettes of all images neatly segmented out. However, with real world images, there tends to be 3-D objects projected onto a 2-D sensor. If the projection parameters are not known, it is difficult to obtain the inverse projection of the image in order to make it appear flat. Many of the contour-based matching algorithms face this problem and hence end up using the MPEG-7 database as the major database for testing purposes. Some who use the real world images use them selectively in such a way that the 3D object's planar projection is a well-defined silhouette that can be used for matching. Hence, we see the need for a real world database to be set up with the projection parameters of each image made available and/or with images all captured with one set of projective parameters that is also used in training. Since there exists no such database for use, we aim to create one that can be used for testing class specific object recognition.

There are of course, hundreds of contours in real world images. Performing expensive contour matching on each such contour will not be a viable option. Unlike [19], where each contour was matched to every other contour, we would like to save some computation time by reducing the list of contours to only those that have the properties of being the candidate contour points. The number of contours for further processing could be reduced from these initial steps by retaining only those contours that have the landmarks, which the object that we are looking for is known to have. Corner points are our landmarks. This will help us to proceed in a hierarchical fashion by eliminating all those contours that do not satisfy the curvature criterion. We would thus have reduced the list of contours to just those set of contours, which have the corner specifications.

Once the list of contours for matching has been reduced from its initial vast number, we plan to extend the contour following approach onto other higher-order primitives (e.g. lines) for further elimination of false contours. All portions of contours between the corner points could be fit into lines or higher-order polynomials using regression-fitting methods. These higher order primitives will define the contour under consideration in a better way than just the corners on their own. Similar to corner adjacency, the higher-order primitive adjacency information can also be obtained if the primitives are extracted from the contours.

The higher order primitives can be parameterised to define them succinctly. For example, a line could be parameterised as in the Hough transform, which would give us the orientation and the distance the line is from the global origin. Strictly speaking, the distance of the line from the global origin and its slope will not be of much use on their own, if we want to maintain the invariance to transformation and 2-D rotation. However, if these parameters were used for comparison along with the parameters of neighbouring primitives, we would then end up with a set of parameters that are relative to the object's shape. This would help in maintaining the algorithm's invariance to similarity transformations.

Further, information such as the length of the line can now be obtained and used for scale analysis of the object. It was explained in Section 2.2.1 how curve matching techniques suffer from their inability to account for similarity transformation, causing them unusable for curve matching. With the curve dimensions now available, such as the length of the line, length of the arc between two landmark corner points, etc., scale of the object can now be calculated and the curve matching techniques can now be used for calculating the similarity between the curves, thus helping in object recognition.

One of the main problems in curve matching lies in the inability of the algorithms to identify the endpoints of the curve over which the curve has to be matched. A curve cannot be matched over any two endpoints. There is a need to locate certain landmarks along the curve that will help to align two or more curves to see how good the curves match each other. The corner points that we have obtained from the previously described methods can act as landmarks. Let us assume we have two curves labelled as C_1 and C_2. Let $C_1 = \{p_1, p_2, ..., p_n\}$ be denoted by the set of landmark points p_i's in curve C_1, and let $C_2 = \{q_1, q_2, ..., q_m\}$ be denoted by the set of landmark points q_j's in curve C_2. The set of intermediate landmark points may or may not be the same. In order to match the two curves, we cannot match it over the complete set of points as the endpoints in themselves may not match with each other. If we were to define a function $matchCurve(C_1, C_2)$ as $matchBetweenPoints(p_1, p_n, q_1, q_m)$ the result would be a very poor match if the end points themselves are not matched. Hence we have to first identify those points in the set C_1 that are most similar to the points in the set C_2 and match the curve fragments between those two landmark points.

Moreover, this matching of curves between similar landmark points has to be over similar scale, translation and rotation values. There is no point in matching two curves that are similar in shape, but described at different scales, if the matching algorithm does not account for the difference in the scales. Once two similar points are identified from the curves C_1 and C_2, we proceed to eliminate translation by shifting the origin to one of the two landmark points. In the next step, we can account for rotation by making sure that the other landmark endpoint is also at the same horizontal level as first landmark endpoint. In other words, these two steps move the curve fragment between the two endpoints onto the x-axis of the co-ordinate system, with one of the endpoints residing on the global origin and the other somewhere along the x-axis.

The final transformation that is left to be accounted for is the scale factor. Let us consider two landmark points in curve C_1, say, p_i and p_j $(i \neq j)$, which match to two landmark points in curve C_2, say, q_k and q_l, where $(k \neq l)$. The distance between the two points is proportional to the scale of the curve fragment between the same. Let d_1 be the distance between the two points p_i and p_j, and d_2 be the distance between the two points q_k and q_l. The distances d_1 and d_2 are simple Euclidean distances calculated as given in Equation 4.12.

$$\begin{aligned} d_1 &= \sqrt{((x_i - x_j)^2 + (y_i - y_j)^2)} \\ d_2 &= \sqrt{((x_l - x_k)^2 + (y_l - y_k)^2)} \end{aligned} \tag{4.12}$$

The individual points are given by $p_i = [x_i, y_i]^T$, $p_j = [x_j, y_j]T]$, $q_l = [x_l, y_l]^T$ and $q_k = [x_k, y_k]^T$. Let us denote the list of points between the two landmark points p_i and p_j as a matrix $\mathbf{X}^{C_1}_{p_i,p_j}$ and the list of points between the two landmark points q_k and q_l as $\mathbf{X}^{C_2}_{q_l,q_k}$. These two matrices are defined as in Equation 4.13.

$$\mathbf{X}^{C_1}_{p_i,p_j} = \begin{bmatrix} x_i & y_i \\ x_{i+1} & y_{i+1} \\ . & . \\ . & . \\ x_j & y_j \end{bmatrix} \quad \mathbf{X}^{C_2}_{q_k,q_l} = \begin{bmatrix} x_k & y_k \\ x_{k+1} & y_{k+1} \\ . & . \\ . & . \\ x_l & y_l \end{bmatrix} \tag{4.13}$$

We can now normalise the scale of the two set of points as shown in Equation 4.14.

$$\begin{aligned} \mathbf{X}^{C_1}_{(p_i,p_j),norm_{\mathbf{X}^{C_2}_{q_l,q_k}}} &= \frac{d_2}{d_1}\mathbf{X}^{C_1}_{p_i,p_j} \\ Or & \\ \mathbf{X}^{C_2}_{(q_k,q_l),norm_{\mathbf{X}^{C_1}_{p_i,p_j}}} &= \frac{d_1}{d_2}\mathbf{X}^{C_2}_{q_k,q_l} \end{aligned} \tag{4.14}$$

$\mathbf{X}^{C_2}_{(q_k,q_l),norm_{\mathbf{X}^{C_1}_{p_i,p_j}}}$ represents the curve fragment in the curve C_2, ranging between the points q_k and q_l, which is normalised to the scale of the curve fragment in the curve C_1, ranging between the landmark points p_i and p_j. $\mathbf{X}^{C_1}_{(p_i,p_j),norm_{\mathbf{X}^{C_2}_{q_l,q_k}}}$ is defined similarly. With the two sets of points now normalised in similarity transformations, we can now proceed towards curve matching. We need to produce a match between the curve $\mathbf{X}^{C_1}_{p_i,p_j}$ and $\mathbf{X}^{C_2}_{(q_k,q_l),norm}$ or between $\mathbf{X}^{C_1}_{(p_i,p_j),norm_{\mathbf{X}^{C_2}_{q_l,q_k}}}$ and $\mathbf{X}^{C_2}_{(q_k,q_l),norm_{\mathbf{X}^{C_1}_{p_i,p_j}}}$.

In a paper of ours [41], we had introduced a novel way of comparison between two or more ROC curves, and have also explained different existing means for ROC curve comparisons. We propose to use the Area Under the Curve (AUC) metric for curve fragment comparison. Given two normalised curves that are similar to each other, intuititely, we can say that the area between them is close to zero. If the two curves are exactly similar to each other, then once they are normalised, then they will lie on top of each other causing the area between the two curves to be zero. The closer the area between the two curves is to zero, the more similar they are to each other.

This seems to be a good metric for comparison between the two curves. Further experimentation and analysis is required to see its effectiveness while comparing between jagged curves. This process has to be repeated over all curve fragments in the two curves to obtain a match. Performing matching hierarchically at the primitive level and at the contour level will take into account both the local and the global shape properties. Upon successfully comparing two or more curves, we will be able to tell how good the two curves match each other and hence, how similar the complete shape is, to the one being matched with.

4.4 Summary

Here, we summarise the main areas that we will be working on. Firstly, we plan to investigate the robustness of corner points on contour fragments over different scales, i.e., the ability of the algorithm to consistently detect the presence of these corners over different scale spaces. Similar idea was used by Lowe in [34] to detect the stability of the regions of maxima or minima over different scales. The presence of corner points on contour fragments give a better knowledge of the contour fragment than just a set of points that join together to form the contour. In addition, based on the robustness of these corner points, we propose to assign them as landmarks for contour fragment matching. Checking for the presence of these landmark points can help us in narrowing down the list of contour fragments that are left to be matched. For example, if a particular contour contains a landmark point that is known to be absent in the matching contour, we can easily reject the contour before performing expensive matching and later rejecting the same.

Secondly, we would like to devise a technique for curve-matching under normalised circumstances. The problem that contour matching techniques face is the lack of a definition of the starting and ending points. Matching two contours between two random points is not a fair way of matching. With the knowledge of landmark points, we can better define the start and endpoints (which would be the landmark points), and then compare the two curve fragments between these landmark points under normalised circumstances. Finding similar contours would help us to localise upon areas in the image that are most likely to have the object being searched. A multi-scale sliding window, such as the one described in [31], can also be used to localise on the areas in the image, that have a high response rate.

Finally, we aim to create a "real world" database - as was explained in Section 4.3 - in addition to the above-mentioned tasks, to help provide a strong set of test images, which would include different objects of similar classes, placed in similar and different backgrounds.

Chapter 5

Summary and Conclusions

In this report, we have emphasised the role of object recognition in the development of computer vision. Traditional methods of object recognition have looked at specific class recognition problems and not much research has been done on generic object recognition. In order to achieve generic class recognition, we have proposed to use the shape of the object as the main distinguishing feature.

Available methods for shape recognition were discussed and the advantages and disadvantages of each were pointed out. However, throughout the discussion of the previously published algorithms, what was noticeable was the absence of the use of inter-feature relations between the extracted features. There is a bridge between the target semantic classes, which aid in high-level object recognition, and the low-level visual descriptors.

Intuitively, it makes sense that inter-feature relations would help in building robust classifiers. Yet their use seems to be few and far between. The few that use inter-feature relations see the efficiency of their algorithms shoot up compared to the algorithms that do not. However, they make a number of assumptions in their algorithms, most of which are not practical. This prompted for the development of other techniques to help incorporate inter-feature relations between primitives and hence led to the proposal of using contour-based approaches for object recognition.

Contour-based approaches readily give out the inter-feature relations, at least to a certain degree, if not completely. Theoretically, it was shown how a contour-based approach could be used for finding adjacency between corners and how

this adjacency information could help in the development of more robust classifiers. Further experiments need to be performed to confirm the practicality of the theory.

In this age, researchers are trying out every method that they can think of for increasing the efficiency of current day algorithms and still find it hard to overcome certain barriers. These barriers are mainly because of the ability (or lack of) of lower level algorithms whose efficiency has a maximum threshold. As more and more complex algorithms use these low-level algorithms, which have their own limitations, the error rates accumulate and tend to increase exponentially. Hence, there needs to be a fundamental change in the primitive extraction phase. It is only then that we can try to achieve higher-levels of accuracy at higher-levels of the algorithms.

Bibliography

[1] Premachandran, V.; Kakarala, R.; , "Measuring the effectiveness of bad pixel detection algorithms using the ROC curve," *Consumer Electronics, IEEE Transactions on* , vol.56, no.4, pp.2511-2519, November 2010

References

[1] "MPEG-7 Core Experiment CE-Shape-1 Test Set." [Online]. Available: http://knight.cis.temple.edu/~shape/MPEG7/MPEG7dataset.zip 22, 52

[2] Y. Amit, D. Geman, and K. Wilder, "Joint induction of shape features and tree classifiers," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 19, no. 11, pp. 1300 –1305, Nov. 1997. 18

[3] P. Arbelaez, M. Maire, C. Fowlkes, and J. Malik, "From contours to regions: An empirical evaluation," in *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, 2009, pp. 2294 –2301. 19

[4] ——, "Contour detection and hierarchical image segmentation," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. PP, no. 99, p. 1, 2011. 19

[5] M. Atallah, "A linear time algorithm for the Hausdorff distance between convex polygons," *Information Processing Letters*, vol. 17, no. 4, pp. 207–209, 1983. 13

[6] D. Ballard, "Generalizing the Hough transform to detect arbitrary shapes," *Pattern recognition*, vol. 13, no. 2, pp. 111–122, 1981. 18

[7] H. Barrow, J. Tenenbaum, R. Bolles, and H. Wolf, "Parametric Correspondence and Chamfer Matching: Two New Techniques For Image Matching," in *Proc. DARPA IU Workshop*. Citeseer, 1978, pp. 21–27. 15

[8] S. Belongie, J. Malik, and J. Puzicha, "Shape context: A new descriptor for shape matching and object recognition," in *In NIPS*. Citeseer, 2000. 18

[9] ——, "Matching shapes," in *Computer Vision, 2001. ICCV 2001. Proceedings. Eighth IEEE International Conference on*, 2001. 18, 19

[10] ——, "Shape matching and object recognition using shape contexts," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pp. 509–522, 2002. 18

[11] S. Belongie, G. Mori, and J. Malik, "Matching with shape contexts," *Statistics and Analysis of Shapes*, pp. 81–105, 2006. 18

[12] S. Belongie, J. Malik, and J. Puzicha, "Matching with Shape Contexts," 2001. [Online]. Available: http://www.eecs.berkeley.edu/Research/Projects/CS/vision/shape/sc_digits.html 20

[13] G. Borgefors, "Hierarchical chamfer matching: A parametric edge matching algorithm," *IEEE Transactions on pattern analysis and machine intelligence*, vol. 10, no. 6, pp. 849–865, 1988. 13

[14] J. B. Burns, A. R. Hanson, and E. M. Riseman, "Extracting straight lines," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. PAMI-8, no. 4, pp. 425 –455, 1986. 18

[15] J. Canny, "A computational approach to edge detection," *Readings in computer vision: issues, problems, principles, and paradigms*, vol. 184, 1987. 19, 21

[16] D. Douglas and T. Peucker, "Algorithms for the reduction of the number of points required to represent a digitized line or its caricature," *Cartographica: The International Journal for Geographic Information and Geovisualization*, vol. 10, no. 2, pp. 112–122, 1973. 39

[17] R. Duda and P. Hart, "Use of the Hough transformation to detect lines and curves in pictures," *Communications of the ACM*, vol. 15, no. 1, pp. 11–15, 1972. 18

[18] L. Fei-Fei, R. Fergus, and P. Perona, "Learning generative visual models from few training examples: An incremental bayesian approach tested on 101 object categories," *Computer Vision and Image Understanding*, vol. 106, no. 1, pp. 59–70, 2007. 38

[19] P. Felzenszwalb and J. Schwartz, "Hierarchical matching of deformable shapes," in *2007 IEEE Conference on Computer Vision and Pattern Recognition*. IEEE, 2007, pp. 1–8. 53

[20] A. Fitzgibbon, M. Pilu, and R. Fisher, "Direct least square fitting of ellipses," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 21, no. 5, pp. 476–480, 2002. 18

[21] R. Gonzalez and R. Woods, *Digital image processing. 2002*. Pearson Education, 2009. 39, 40, 41

[22] N. Gregorie and M. Bouillot, "Hausdorff Distance Between Convex Polygons." 14, 16

[23] C. Harris and M. Stephens, "A combined corner and edge detector," in *Alvey vision conference*, vol. 15. Manchester, UK, 1988, p. 50. 24, 25

[24] X. He and N. Yung, "Corner detector based on global and local curvature properties," *Optical Engineering*, vol. 47, p. 057008, 2008. 42, 43, 44, 45

[25] S. Helmer and D. Lowe, "Using stereo for object recognition," in *Robotics and Automation (ICRA), 2010 IEEE International Conference on*. IEEE, 2010, pp. 3121–3127. 17

[26] D. Huttenlocher, G. Klanderman, and W. Rucklidge, "Comparing images using the Hausdorff distance," *IEEE Transactions on pattern analysis and machine intelligence*, pp. 850–863, 1993. 13

[27] D. Huttenlocher and W. Rucklidge, "A multi-resolution technique for comparing images using the hausdorff distance," in *Computer Vision and Pattern Recognition, 1993. Proceedings CVPR '93., 1993 IEEE Computer Society Conference on*, Jun. 1993, pp. 705 –706. 15

[28] S.-B. Im and S.-B. Cho, "Context-based scene recognition using bayesian networks with scale-invariant feature transform," in *Advanced Concepts for Intelligent Vision Systems*, ser. Lecture Notes in Computer Science, J. Blanc-Talon, W. Philips, D. Popescu, and P. Scheunders, Eds. Springer Berlin / Heidelberg, 2006, vol. 4179, pp. 1080–1087, 10.1007/11864349_98. [Online]. Available: http://dx.doi.org/10.1007/11864349_98 5

[29] C. Jung and R. Schramm, "Rectangle detection based on a windowed Hough transform," in *Computer Graphics and Image Processing, 2004. Proceedings. 17th Brazilian Symposium on.* IEEE, 2004, pp. 113–120. 18

[30] L. Kitchen and A. Rosenfeld, "Gray-level corner detection," *Pattern Recognition Letters*, vol. 1, no. 2, pp. 95–102, 1982. 42

[31] C. Lampert, M. Blaschko, and T. Hofmann, "Efficient subwindow search: A branch and bound framework for object localization," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 31, no. 12, pp. 2129 –2142, 2009. 57

[32] T. Lindeberg, "Feature detection with automatic scale selection," *International Journal of Computer Vision*, vol. 30, no. 2, pp. 79–116, 1998. 51

[33] D. G. Lowe, "Object recognition from local scale-invariant features," *Computer Vision, IEEE International Conference on*, vol. 2, p. 1150, 1999. 10

[34] D. Lowe, "Distinctive image features from scale-invariant keypoints," *International journal of computer vision*, vol. 60, no. 2, pp. 91–110, 2004. 10, 21, 51, 57

[35] F. Mokhtarian and R. Suomela, "Robust image corner detection through curvature scale space," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 20, no. 12, pp. 1376 –1381, Dec. 1998. 42

[36] G. Mori, S. Belongie, and J. Malik, "Shape contexts enable efficient retrieval of similar shapes," *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on*, 2001. 18

[37] ——, "Efficient shape matching using shape contexts," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 27, no. 11, pp. 1832–1837, 2005. 19

[38] P. D. Kovesi, "Matlab and octave functions for computer vision and image processing." [Online]. Available: http://www.csse.uwa.edu.au/~pk/Research/MatlabFns/ 33

[39] E. Persoon and K.-S. Fu, "Shape discrimination using fourier descriptors," *Systems, Man and Cybernetics, IEEE Transactions on*, vol. 7, no. 3, pp. 170 –179, 1977. 17

[40] D. Prasad, "Object detection in real images," School of Computer Engineering, Nanyang Technological University, Tech. Rep., April 2010. 18

[41] V. Premachandran and R. Kakarala, "Measuring the effectiveness of bad pixel detection algorithms using the ROC curve," *Consumer Electronics, IEEE Transactions on*, vol. 56, no. 4, pp. 2511–2519, 2010. 56

[42] R. Ralph, "MPEG 7 Shape Matching." [Online]. Available: http://knight.cis.temple.edu/~{}shape/MPEG7/dataset.html 52

[43] U. Ramer, "An iterative procedure for the polygonal approximation of plane curves," *Computer Graphics and Image Processing*, vol. 1, no. 3, pp. 244 – 256, 1972. [Online]. Available: http://www.sciencedirect.com/science/article/B7GXF-4RDXR1S-2/2/23a76082c3f28735a8be9dcc36f54239 39

[44] X. Ren, C. Fowlkes, and J. Malik, "Learning probabilistic models for contour completion in natural images," *International Journal of Computer Vision*, vol. 77, no. 1, pp. 47–63, 2008. 19

[45] B. Russell, A. Torralba, K. Murphy, and W. Freeman, "LabelMe: a database and web-based tool for image annotation," *International Journal of Computer Vision*, vol. 77, no. 1, pp. 157–173, 2008. 3, 38

[46] K. Schindler and D. Suter, "Object detection by global contour shape," *Pattern Recognition*, vol. 41, no. 12, pp. 3736–3748, Dec. 2008. [Online]. Available: http://dx.doi.org/10.1016/j.patcog.2008.05.025 18, 19

[47] J. Shotton, A. Blake, and R. Cipolla, "Multiscale categorical object recognition using contour fragments," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pp. 1270–1281, 2007. 13

[48] D. Sim and R. Park, "Two-dimensional object alignment based on the robust oriented Hausdorff similarity measure," *Image Processing, IEEE Transactions on*, vol. 10, no. 3, pp. 475–483, 2002. 13

[49] S. Smith and J. Brady, "SUSAN A new approach to low level image processing," *International journal of computer vision*, vol. 23, no. 1, pp. 45–78, 1997. 24, 26

[50] T. Strat and M. Fischler, "Context-based vision: recognizing objects using information from both 2d and 3d imagery," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 13, no. 10, pp. 1050 –1065, Oct. 1991. 5

[51] P. Tissainayagam and D. Suter, "Assessing the performance of corner detectors for point feature tracking applications," *Image and Vision Computing*, vol. 22, no. 8, pp. 663–679, 2004. 24, 25

[52] A. Torralba, K. Murphy, W. Freeman, and M. Rubin, "Context-based vision system for place and object recognition," in *Computer Vision, 2003. Proceedings. Ninth IEEE International Conference on*, 2003, pp. 273 –280 vol.1. 5

[53] M. Trajkovi and M. Hedley, "Fast corner detection," *Image and Vision Computing*, vol. 16, no. 2, pp. 75–87, 1998. 24

[54] R. von Gioi, J. Jakubowicz, J. Morel, and G. Randall, "LSD: A Fast Line Segment Detector with a False Detection Control," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 32, no. 4, pp. 722–732, 2010. 18, 22

[55] M. Wertheimer, *Gestalt theory.* Hayes Barton Press, 1944. 23

[56] Wikipedia, "Kanisza Triangle." [Online]. Available: http://en.wikipedia.org/wiki/Kanizsa_triangle 23

[57] L. Williams and D. Jacobs, "Stochastic completion fields: A neural model of illusory contour shape and salience," *Neural Computation*, vol. 9, no. 4, pp. 837–858, 1997. 23

[58] C. T. Zahn and R. Z. Roskies, "Fourier descriptors for plane closed curves," *Computers, IEEE Transactions on*, vol. C-21, no. 3, pp. 269 –281, 1972. 17

[59] D. Zhang and G. Lu, "Review of shape representation and description techniques," *Pattern recognition*, vol. 37, no. 1, pp. 1–19, 2004. 9