Chapter 21
Machine Vision

Abstract The goal of computer vision is to extract information from images—a method that can produce a structure from motion, can recover a three-dimensional model of an object from a sequence of views, e.g., grasping by robot, medical imaging, and graphical modeling. This chapter presents the machine vision applications, the basic principle of vision, cognition and classification, and cognitive architecture. The cognition is—going from image-to-scene, which can be achieved through inversion by fixing scene parameters, inversion by restricting the problem domain, and inversion by acquiring additional images. The machine vision techniques are presented here for low-level, middle-level, and high-level vision. The last one requires indexing of images which can be searched through geometric hashing. One of advanced areas of vision—the object tracking, is presented in-depth, which requires the sequences—subtraction of image from background, segmentation of the image, and learning, followed with tracking. Finally, the axioms of vision, tools for computer vision, chapter summary, and practice exercises are presented.

Keywords Computer vision goals · 3D models · Vision principles · Vision application · Cognition · Object classification · Cognitive architecture · Machine vision techniques · Low-level vision · Middle-level vision · High-level vision · Image segmentation · Computer vision tools · Vision axioms · Computer vision tools

21.1 Introduction

The field of Machine Vision (also called Computer Vision) is growing rapidly. It is concerned with analysis, modification, and understanding of images. The objective of this field is to understand what is happening in front of a camera, and use that understanding to control a computer or robotic system, or making use of these images provide people new images that are more informative or aesthetically more pleasing than the original images. The machine vision has vast applications, some of the important application areas of machine vision are the following: automotive systems, photography, video surveillance, biometrics, movie productions, Web search,

K. R. Chowdhary, *Fundamentals of Artificial Intelligence*,
https://doi.org/10.1007/978-81-322-3972-7_21

medicines, medical and health sciences, augmented reality, gaming, new user interfaces with computers, and in fact, there cannot be an exhaustive list of all possible areas, as they are continuously increasing in numbers.

As part of advances in machine vision, there are cameras that can focus automatically on our face, and can trigger the shutter when we smile. On the other hand, using an optical text-recognition system one can transform scanned documents into text, which can be analyzed or read aloud by a voice synthesizer.

It is common to have driver-assistance systems in the newly built cars that help the driver to park the car, or give warning signal to the driver in potentially dangerous situations. Intelligent video surveillance systems exist, that can monitor the security of public areas. As new smartphones come progressively equipped with more and more processing power and better resolution of cameras, these phones are becoming fertile field for potential computer vision applications. These devices have built-in capability to merge number of photographs into a high-resolution panorama, they can also read a quick response code, can recognize it and retrieve information about the product from Internet. The mobile computer vision technology is moving fast toward touch interface. However, still the computer vision is computationally expensive to achieve all these in ideal sense.

The computer vision applications have real use only when they can be executed in real time, for which, it is necessary that processing of each frame needs to be completed within $30 - 40$ msec duration. This is a challenging task, particularly when the computation takes place, e.g., in a smartphone or some other embedded computing environment. Generally, it is possible to trade quality for speed or vice versa. For example, in the *panorama stitching* algorithm, if time is provided more liberally, it is possible to find out more matches in source images, and one can synthesize an image of higher quality. To meet the constraints of time and computations to be carried out, one can compromise either on quality or spend longer time optimizing the code for a given hardware architecture.

Further, some of the goals of *computer vision* or *machine vision* can be to extract information from images. For example, an algorithm that produces a *structure* from a *motion*, can recover a 3D model of an object from a sequence of views. As other examples, from a robot grasping an object it is possible to construct a 3D view; construction of a 3D object through medical imaging; and graphical modeling to produce a 3D view. Model-based recognition methods can determine the best matches of stored objects for image data, for use in visual inspection and image database searches. Through visual motion analysis, it is possible to recover image motion patterns for use in vehicle guidance and processing digital video.

The field of computer vision has a close relation to the field of image processing, as follows: In image processing, the focus is on transformation of images, but in computer vision, the focus is on extracting information from image data. To extract a 3D from 2D images is the case of computer vision, and not of an image processing. However, image filtering and color enhancement are the tasks of image processing.

Though many other computer science areas (computation), and board game playing (chess, and tic-tac-toe), have progressed enough and have speed far higher than human, the area of computer vision heavily lags behind compared to human capa-

bilities. The successful example of visual information processing is recognizing of printed text. However, the Optical Character Recognition (OCR) systems make mistakes that even school level students do not make. The problem with *computer vision*[1] systems is their properties of brittleness—a small change in input causes a big change in the output.

Learning Outcomes of this Chapter:

1. Summarize the importance of image and object recognition in AI and indicate several significant applications of this technology. [Familiarity]
2. List at least three image-segmentation approaches, such as thresholding, edge-based, and region-based algorithms, along with their defining characteristics, strengths, and weaknesses. [Familiarity]
3. Implement 2D object recognition based on contour-and/or region-based shape representations. [Usage]
4. Provide at least two examples of a transformation of a data source from one sensory domain to another, e.g., data interpreted as single-band 2D image to 3D object. [Familiarity]
5. Describe an algorithm combining features into higher level percepts, e.g., a contour or polygon from visual primitives. [Usage]
6. Describe a classification algorithm that segments input percepts into output categories and quantitatively evaluates the resulting classification. [Usage]
7. Evaluate the performance of the underlying feature extraction, relative to at least one alternative possible approach in its contribution to the classification task. [Assessment]
8. Describe at least three classification approaches, their prerequisites for applicability, their strengths, and their shortcomings. [Familiarity]
9. Tools for Computer Vision. [Usage]

21.2 Machine Vision Applications

Originally the computer vision systems were designed with the aim to be used in industrial and military applications. Some of the common military applications of computer vision are: recognition of far-off targets, visual-based guidance systems for autonomous vehicles, and interpretation of images. In the recent times, many computer vision applications have emerged in medical imaging and multimedia systems, like preoperative scans of patients in the operating room, complex surgery using robots, etc. Computer vision techniques are also used for virtual reality applications, and image database retrieval systems. The successful applications of computer vision exists as a consequence to two important effects: 1. A technical force push toward limiting the domain that have sufficient structure or constraint that the many-to-one

[1] In AI literature, the words: "computer vision" and "machine vision" are interchangingly used, hence in this chapter also, both the terms mean the same.

inversion problem can be made tractable, and 2. Economic force push toward having a problem of sufficient base to be financially viable. The possible solution is an intersection of these two conditions. The following is a brief account of machine vision applications [4].

Visual Inspection

One of the most successful applications of computer vision is in industrial quality control, for example, in automation of visual inspection tasks. This is in replacement for human visual inspection, which is crucial to quality control across a broad spectrum of manufacturing, that ranges from low-volume custom products to high-volume and bulk products.

Assembly and Material Handling

In automatic assembly of products, the central driver is the uncertainty in the factory. To solve this problem, robot vision is typically used. The robot vision is based on computer vision algorithms, which have applications in navigation and path control of robot vehicles and robot manipulators. This application is governed by economic drive as well as quality control.

Automatic Target Recognition (ATR)

Due to the progress of sophistication of weapon systems, it has become necessary to provide rapid and accurate identification and tracking of targets. The main drive force toward the progress in this direction is accuracy of recognition, with tracking of multiple targets at the same time.

Photo Interpretation

This work requires the digitization of image data, and interpret them using vision algorithms.

Extraction of 3D Structure

There are many engineering applications, that require construction of complex 3D geometric models and terrains, such applications are: engineering simulation, numerically controlled machines, virtual reality, and advanced cartography. The input to such models are: drawings, intensity images, and 3D sensors like laser triangulation. The algorithms used in these applications derive automatically a 3D slid model of an object that drives a finite element or graphic simulation program.

21.3 Basic Principles of Vision

One of the most basic and essential characteristics of living beings is the ability to recognize and identify objects. All the higher animals depend on this ability for their survival. Without this they are unable to function even in a static and unchanging environment. Sight is our most impressive sense. It gives us, without conscious

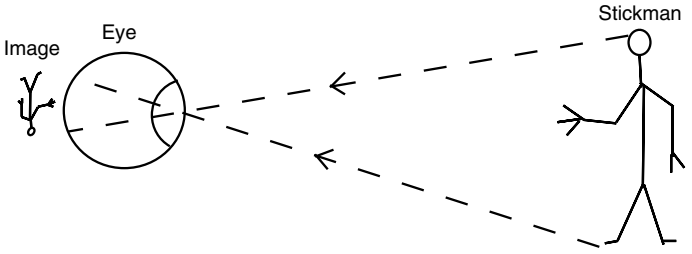

Fig. 21.1 Image of a "Stickman"

efforts, the detailed information about the shape of the world around us. It is also the challenging area in AI.

The most natural place to begin our study is *transduction* of physical phenomena into internal representation. Vision starts with an eye—a device for capturing and focusing the light that bounces off the objects. Every point on an object (other than a mirror) scatters incident light in all directions. An eye has a lens that directs the light from one point on an object to just one point on retina (see Fig. 21.1). The image on retina is upside down, but human mind corrects it. We note that image is 2D, while the object is 3D [1].

Definition 21.1 *(Vision Problem)* A vision problem may be stated as, given a 2D image, infer the objects that produced it, as well as infer their shapes, position, sizes, and colors.

However, one imprecision exists in the above definition, it is—how all these characteristics counts for object. For monochrome images, an image may be thought as a function, giving a gray-level at every point on the image plane. The gray-level varies from 0 (maximum black) to 1 (maximum bright). The latter is corresponding to the maximum response the eye can make. We assume that surface on which image is constructed is flat in x and y coordinates, which is appropriate for computer vision also. We approximate the gray-level function by a 2D array *image*, which breaks the image into squares, and records average gray-level over each square, with a pixel at a point, say (i, j), is cell $image(i, j)$. The value of gray-level function at each pixel is in the range [0, 1].

It is challenge to reason back from image to the scene that created that. Figure 21.1 suggests that each spot in image corresponds to one spot in the real world, namely, the piece of surface the light bounced off in order to make the image spot. The recognition of images is a process of pattern recognition. The recognition requires learning, which means mapping the images patterns to some internal representation—this is scene transformation into images. However, recognition of objects is inversion of this process, i.e., reconstruction of scene features from the images, which is a complex task.

Though it is not yet fully established, but it is hypothesized that human follow the following process for identification or classification of objects:

New objects are introduced to a human through activation of sensor stimuli. The sensors, depending on their physical properties, are sensitive in varying degrees to certain attributes, serve to characterize the objects, and the sensor output tends to be proportional to the more prominent attributes. Having perceived a new object, a cognitive model is formed from the stimuli patterns and stored in the memory. Recurrent experiences in perceiving the same object or similar objects strengthen and refine the similarity patterns. Repeated perceptions result in the generalized models of objects classes which become later useful in matching, and hence recognition of similar objects.

The *recognition* is, in fact, a process of establishing a close match between some new stimulus and the previously stored stimulus pattern. Object recognition is the task of finding and labeling parts of a 2D image of scene that corresponds to objects in the scene. For example, from an aerial photo, to identify the various objects, like building, roads, forests, etc. It is a task as it might be carried out by a human observer with marking pen, to mark and label the objects.

Object *classification* is closely related to recognition. The ability to classify various objects or group of objects accordingly to some commonly shared features is a form of class recognition. Classification is essential for decision-making, learning, and many other cognitive acts. Like recognition, classification depends on the ability to discover common patterns among objects. This ability must be acquired through some learning process. For the learning to take place, the prominent feature patterns that characterize classes of objects must be discovered, generalized, and stored for subsequent recall and comparison.

To recognize an object, it is required to first establish a general description of each object to be recognized. Most often, a model includes texture, shape, and the context knowledge about the occurrence of such objects in a scene. For instance, a mathematical description of a set of shaded rectangles may be used to generate buildings as objects. A 3D building object could be modeled as a set of rectangular solids. Texture information might include colors or knowledge about the layout of a building's window.

Corresponding to each occurrence or instance of a model in the image, a label is attached, called *model label*, that can be thought of as a tag pinned to an area in the image that we consider showing an instance of the corresponding object model. The "roads" and "buildings/houses" in Fig. 21.2, shown by arrows, are examples of model labels. Note that a model may be $2D$ or $3D$, but the labels always show $2D$ model instances.

There are many important distinctions about the kind of information we are interested within a digital image, corresponding the scene of that image. The most elementary type of information is called *syntactic* information, which is concerned only with pixel values, and their meanings. The other information, called *semantic information*, deals with knowledge and meaning of the information. Hence, a syntactic image operator's role is to blindly apply an algorithm to the pixel values, without

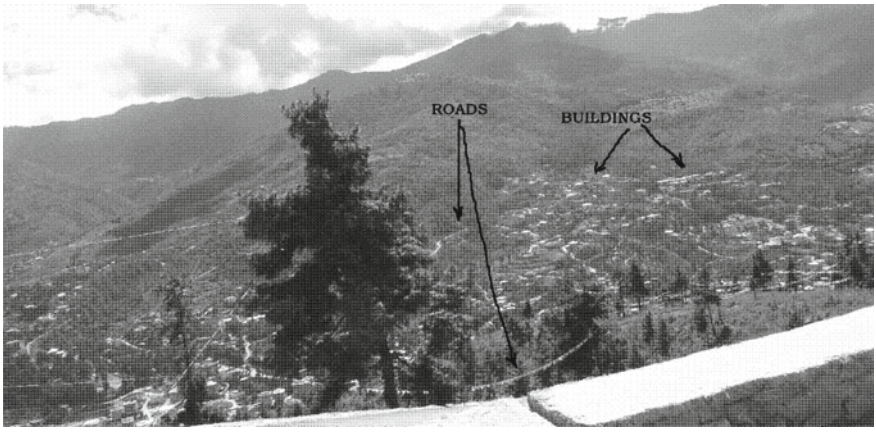

Fig. 21.2 An aerial image of a Hilly Town with labels attached to buildings and roads

concerned to the meaning associated with those pixels. An example is, a procedure that assembles group of adjacent pixels which have, say, a high contrast with respect to their neighbors. On the contrary, a semantic operator uses models of the scene and the image production process. They incorporate symbolic knowledge about how the image is organized, such as "a part may be lying on top of one another."

21.4 Cognition and Classification

For a process of mechanized recognition requires learning new objects carried out through the following steps:

1. The stimuli produced by objects is perceived by the sensory devices. The more prominent attributes, like size, shape, color, and texture produce the strong stimuli. The values of these attributes and their relations are used to characterize an object in the form of a *pattern vector* as a string. This string is represented as a classification tree, a description graph, or some other form. The range of characteristic attribute value is known as the *measurement space*.
2. A subset of attributes whose values provide cohesive object grouping or clustering, consistent with some goals associated with the object classifications, are selected. The range of the subset of attribute values is *feature space*.
3. Using the selected attribute values, the object or class characterization models are learned by forming generalized prototype description, classification rule, or decision functions. These models are stored for subsequent recognition. The range of decision function values or classification rules is *decision space*.

Recognition of familiar objects is achieved through application of the rules learned by step 3 above, by comparison and matching of object features with the stored

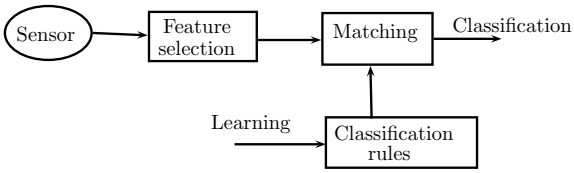

Fig. 21.3 Pattern recognition process

models. Refinements and adjustments can be performed continually thereafter to improve the quality and speed of recognition. These process steps are shown in Fig. 21.3.

Conceptual View of Cognitive Architecture

For an artificial system it is a primary requirement that it should build a rich internal representation of the external environment. This internal representation should be such that it is supportive to draw the inferences, make the decisions, and allows to perform the reasoning process in general related to its own tasks. One approach for representation is *classical logic*, where symbols are given the meaning by relating them to abstract entities as per semantics of model-theoretic approach. However, this approach turns out to be incomplete for the requirement of machine vision, as it requires to find out two things, 1. The meanings of its symbols within the internal representation, and 2. Its interaction with the external world.

A machine vision system requires a cognitive architecture with effective internal representation of the environment. This environment is built through the processes that are defined over suitable intermediate level. The processes act as intermediary between sensory data input and the internal symbolic level representation. One approach to model the visual perception through a process, such that the representation of *information* and *knowledge*, as well as the processing of both, take place at different levels of abstractions. The two levels of abstraction are: 1. The lowest level, which is directly related to features of stimuli and 2. The highest level, where knowledge is in symbolic form, that is concerned with the perceived object. A general assumption of computer vision is that a vision process concludes with 3D reconstruction of shapes using some *geometric primitives*.

For the designs related to cognitive architectures, there are three cognitive representation levels, are as follows [2]:

- *Sub-symbolic level*. The information at this level is related only to sensory data and to nothing else.
- *Linguistic level*. This level uses the information represented using symbolic forms.
- *Intermediate level*. It is also called prelinguistic conceptual level. The characteristic of information at this level (in terms of metric spaces) is defined by a number of cognitive dimensions, that is language independent.

The *intermediate* level representation is useful for generating the essential representation of external environment of a cognitive agent. In addition, the intermediate level provides a precise interpretation of the linguistic level, where interpretation of

conceptual categories is concerned with some standard problems. One such problem is that, perceptual commonsense concepts never correspond to any clear classic category which can be described in terms of necessary and sufficient conditions. For example, the membership in perceptive concepts categories is never in 0 and 1 classes, but usually there is need to consider a prototype of the category.

The information available in a cognitive system strictly depend on data acquired through measurement process. Hence, the knowledge derived through this information at conceptual level will also be affected due to measurement errors. To solve this problem, a model mapping is carried out between conceptual and linguistic levels using a *connectionist*[2] device. The use of Neural Networks (NN) eliminate the need for exhaustive description of conceptual categories at the symbolic level. The latter becomes possible because, a prototype is built based on associative mapping taking place during the training phase of NNs. A measure of similarity between the prototype and the given object is implicit in the behavior of the NN, which is determined during the learning phase of this network.

21.5 From Image-to-Scene

For a creature, of type either *biological* or *mechanical* (in case of latter, e.g., robot), for effective interaction with its environment, it needs to know *what* are the objects, and *where* they exists? The computer vision provides the basic methods to understand as how to make intelligent decisions about the environment of interactions, on the basis of sensory inputs it receives. To support the intelligent interaction with the environment, the vision system must know the information about objects in the world (with which it interacts). This knowledge about the objects and their positions become known to the vision system only on the basis of the measurements of inputs received in the form of reflected brightness. To convert this brightness into the world of measurements, first it is necessary to know how the objects are mapped into the image brightness. The magnitude of light recorded by an individual sensor is the result complex interaction of the following parameters:

- Orientation and position of the object, as well as the position and orientation of the light sources with respect to the sensor,
- $3D$ shape of the object,
- The reflective properties of the object's surface (i.e., the rules of physics that govern the reflection of light rays from surface of the object),
- Spectral sensitivity and quantum efficiency of the sensor, and
- Spectral properties of the light sources.

For generating an image from a given set of scene parameters (i.e., mapping from scene to image) is a well-defined process. However, to invert an image to compute the scene parameters that gave rise to the image is an ill-posed and challenging

[2]Labeling of unsegmented sequence of data with recurrent neural networks.

problem. This is because, it requires an inversion of a single number to deduce many parameters. If an image is treated as a set of independent brightness values, there can be an infinite number of scenes that could have produced this image ! Hence, deciding the precise scene that might have given rise to so and so image is a challenging task. However, as human beings (and other living creatures), we generally do not have any trouble in concluding the true original scene by interpreting any given image. Thus, the image brightness is generally not independent, and goal of computer vision is to find out the sufficient additional constraints to invert the brightness into scene parameters. In the following discussions, we present major approaches to object recognition through the inversions.

21.5.1 Inversion by Fixing Scene Parameters

One approach to perform inversion from image-to-scene that created the image is through fixing some scene parameters. For example, if we have information about the *surface reflectivity*[3] of an isolated object, and about the position of light source, there is a nonlinear equation available that directly relates image brightness at a point to the object's local surface orientation at that point. Since there are two unknown variables for each measured brightness, we need more constraints, to determine the scene.

One method, called *shape-from-shading*, assumes that the surface that is locally smooth is sufficient to provide solution for the object's shape. The other methods are called *photometric stereo methods*. Their working is based on taking several images with a single position of camera, but with different light sources, and a set of brightnesses are used corresponding to a single point, to determine the local surface shape.

21.5.2 Inversion by Restricting the Problem Domain

There is a class of methods for going from image-to-scene, i.e., image inversion, that is based on restricting the problem domain. Consider the case of printed character recognition, like Optical Character Recognition (OCR), where domain consists of $2D$ objects, with known shapes and restricted range of orientation and positions. The inversion problem, in this case, is basically separating individual objects (letters, digits, and punctuation marks) in the presence of noisy brightnesses, and then matching the shapes of those objects against the canonical models. There are in fact many successful commercial tools available for recognizing the printed characters, but the problem becomes more complex if we allow in the range of characters, all the cursive scripts with all their possible variants!

[3]The fraction of radiant energy that is reflected from a surface, also called coefficient of reflection.

Consider the problem of identification and locating planar objects in cluttered scenes. The image inversion in this case typically requires finding the invariants in the image forming process. That is, to infer a sudden change in a scene parameter, like edge of an object, based on the observation of a sudden change in a recorded brightness in the image. The features like this have the advantage of being insensitive to variations in light sources and camera positions, hence they are more reliable indicators of scene features.

After having extracted the hypothesized instances of the object features, the next step to recover the scene is to match the geometric shapes possessing such features against the known models. However, in the following, we present an inversion technique that is based on acquiring additional images.

21.5.3 Inversion by Acquiring Additional Images

Yet another method for image inversion states that, it is possible to achieve image inversion by acquiring additional images. For example, using *stereo vision*, which obtains two views of a scene, it is possible to extract $3D$ shapes of objects. The principle of stereo vision is as follows: if one can determine a point in two images that are projections of a single point in the original scene, and if relative orientation of two cameras is known, then the method of triangulation can produce a $3D$ reconstruction of the shapes of the objects that are existing in the real world. A similar principle holds for motion sequence of images, where either the camera or the objects in the real world move between the images. This process again involves a matching problem, this time there is matching between features from two images, instead of an image and a model. Most of the machine vision algorithms depend on matching schemes that measure the similarity between features in the two images, and make use of multi-resolution methods to control the complexity of such matchings.

Once the $3D$ shape of the scene is extracted, recognition of the scene is made possible by matching $3D$ shape descriptions, while the navigation in the $3D$ world is supported by identifying the collision-free paths through the reconstruction world.

A large variety of matching schemes exists, that also includes the methods that directly search the space, of all possible pairings of models and image features, called, *correspondence space*. There are other methods, that directly search the space of all possible poses of the objects, and there are hybrid methods that can combine both the type of spaces. Most of the methods rely on geometric constraints to reduce the complexity of search, the constraints that typically encode information about objects' shape. The constraints are also used to measure the feasibility of matching the data features to model features, such that they are consistent with the legal transformations of the object. The following are typically the difficult parts of this problem: 1. efficiently deciding about the image features that belong to a single object in the presence of scene clutter and occlusion, and 2. effectively match the partial description of the object in the presence of uncertainty in sensor input.

From the above discussions, we find it common that, vision methods attempt to extract scene parameters, like surface material type and object shape given the observed brightnesses, and then to use these extracted parameters to match against known-object models to support the recognition.

21.6 Machine Vision Techniques

The machine vision systems operates on *digital images*, i.e., quantized collection of discrete values in 2D space with varying intensity. The methods used in machine vision are also classified as low-level vision, middle-level vision, and high-level visions. Though, this is not the only classification for vision systems, but it provides a useful way of classifying the computer vision problems [9].

The *Low-level vision* techniques operate directly on images and produce the output images in the same coordinate system as that of the input. For example, an *edge detection algorithm* may take an image with different intensity values as input, and produce a binary output that indicates the edges in the picture.

The *Middle-level vision* techniques take an image or the output produced by low-level vision algorithm as input, and may produce the output something other than pixels of the image coordinate system. For example, the stereo-vision system may produce an output shape in $3D$ using input as two $2D$ images. As another example, a *structure-from-motion* algorithm takes as input a set of image features and produce at output $3D$ coordinate of these features.

The *high-level vision* techniques take as input the results of low- or middle-level vision algorithms and produce in the output some abstract data structures. For example, a *model-based recognition* system may take a set of image features as input, and provide at output the geometric transformations that map the models in its database to the respective locations in the image.

21.6.1 Low-Level Vision

The low-level vision computations operate directly on images and produce pixel-based outputs, that remains within the original image coordinates only. The examples of these computations are: finding the intensity edges in an image, smoothing images using filters, computing visual motion fields, and analyzing color information in images [9].

In the following, we discuss edge detection and smoothing of image, in more detail.

21.6.2 Local Edge Detection

The edge detection of a real-world object is carried out to find out the geometry of the image and to be used in higher level image processing. There are some physical events that cause intensity changes, or appearance of an edge in the images. For example, the object boundaries produce intensity changes which are caused due to discontinuity in the depth or difference in surface color, or difference in texture. The surface boundaries produce intensity changes due to a difference in the surface orientation. However, there are some phenomena of intensity changes that do not reflect directly on the geometry of the object, the examples are shadows and interreflections [9].

As an example, we shall represent gray-level image by $A(x, y)$, as an intensity of a function of the image coordinate system (x, y). The intensity of edges in the image correspond to sudden changes in the value of function $A(x, y)$. For this image, we compute the *gradient magnitude* using the expression,

$$\|\nabla A\|^2 = \left(\frac{\partial A}{\partial x}\right)^2 + \left(\frac{\partial A}{\partial y}\right)^2. \tag{21.1}$$

In simple words, where the squared gradient magnitude (i.e., $\|\nabla A\|^2$) is large, it is an indication of an edge. The Laplacian operator (∇^2) is another local differential operator that is used for edge detection, expressed by

$$\nabla^2 A = \frac{\partial^2 A}{\partial x^2} + \frac{\partial^2 A}{\partial y^2} \tag{21.2}$$

This second-derivative operator retains the information as which side of the edge is brighter. The zero crossing of $\nabla^2 A$ indicates that the intensity edges of the image, and the sign on each side of a zero crossing indicates which side is brighter.

The Machine Vision Systems

The images input to machine vision systems are digitized with respect to both the space and intensity, and are available as an array $A[x, y]$ of discrete intensity values. The approximations based on finite difference are used to estimate the derivatives for computing the local differential operators. A discrete 1D sampled function is represented as a vector $\mathbf{f}[i]$. The first-order derivative of this vector is $d\mathbf{f}/dx$, which can be approximated as $\mathbf{f}[i+1] - \mathbf{f}[i]$. And, the second-order derivative is computed as an approximation $\mathbf{f}[i-1] - 2\mathbf{f}[i] + \mathbf{f}[i+1]$. Hence, the gradient magnitude can be approximated as

$$\left(\frac{\partial A}{\partial x}\right)^2 + \left(\frac{\partial A}{\partial y}\right)^2 \approx (A[j+1, k+1] - A[j, k])^2 + (A[j, k+1] - A[j+1, k])^2. \tag{21.3}$$

Image Smoothing and Filtering

Filtering of images in computer vision is carried out using *convolution* operation. Consider a function $h(x, y)$ defined in terms of $f(x, y)$, and $g(x, y)$, as

$$h(x, y) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} f(x - \xi, y - \eta)\ g(\xi, \eta) d_{\xi} d_{\eta} \tag{21.4}$$

In the above, h is called the convolution of f and g, and expressed as $h = f \otimes g$, and the value of h at any given point (x, y) depend on the values of f and g at all points. However, this complex looking computation can be simplified, as convolution is *commutative* and *associative*, hence the computations can be rearranged in any order it is convenient, to compute them efficiently.

To carry out the discrete approximation for a convolution, the sum of products can be represented as four nested loops over two arrays representing the sampled functions f and g. Let the array $g[i, j]$ be $m \times m$, and the array $f[x, y]$ be $n \times n$, for $n > m$. The initial values of indexes are 0. Algorithm 21.1 computes the discrete convolution of sampled functions f and g.

The role of convolution is to smooth an image (i.e., to act as low-pass filter), to handle the problem of high-frequency variations. The latter is indicated by sudden change of intensity from a pixel to next pixel. Gaussian method is used for representation here. The Gaussian function G_{σ} in 1D is expressed by

Algorithm 21.1 Compute discrete convolution of f and g

```
1: for x ← xmin to xmax do
2:   for y ← ymin to ymax do
3:     Let s = 0
4:     for i ← 0 to m − 1 do
5:       for j ← 0 to m − 1 do
6:         s ← s + g[i, j] f[x − ⌊m/2⌋ + i, y − ⌊m/2⌋ + j]
7:       end for
8:     end for
9:     h[x, y] = s
10:  end for
11: end for
```

$$G_{\sigma}(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{\frac{-x^2}{2\sigma^2}}, \tag{21.5}$$

which is a canonical bell-shaped distribution, also called *normal distribution*. The maximum value of this function is obtained at $x = 0$, and the area of the function under the curve is 1. The parameter σ controls the width of function G_{σ}, the larger the σ, the wider the bell. The Gaussian function in 2D (x, y) is given by the following function.

$$G_{\sigma}(x, y) = \frac{1}{2\pi\sigma} e^{\frac{-(x^2+y^2)}{2\sigma^2}}. \tag{21.6}$$

In discrete-based system, the values of integral steps over some range are used as approximations, which are generally at $\pm 4\sigma$ steps. These values are normalized so that they sum to 1, just like in the case of continuous values, with integral as 1.

21.6.3 Middle-Level Vision

The middle-level vision techniques receives the images or results from the output of low-level vision algorithms, and produce some output other than pixels, within the coordinate systems of the original image. One goal of the middle-level vision is to extract the 3D geometric information from the images, which are in fact of 2D—usually referred to as *shape-from x*, i.e., recovering $3D$ structure from $2D$ images. In fact, presence of a shading in an image reveals information about $3D$ objects. An example is, perceiving the shape of a sphere from its image, which we conclude as a solid rather than a circular disk. This perception is due to the uniform change taking place in brightness away from the light source [9].

Information about 3D shapes of objects is perceived due to the mirror-like reflections from the objects. Yet, one more source of 3D shape information is due to change in location of an object in an image, from one image to another image of the same object's views. The commonly used techniques for extracting image shapes from multiple images are: 1. stereopsis and 2. structure-from-motion. Apart from extraction of shape-from images, another goal of middle-level vision is to extract structural description of images. What relations exist between these structures can be found out using grouping methods, called perceptual grouping. The latter is responsible for recovering non-accidental alignments of image primitives, like colinear line segments or co-circular arcs.

Stereopsis

Computer stereo vision is the extraction of 3D information from digital images. By comparing the information about same object about a scene from two panels. The human beings compare two images, based on their being superimposed in a stereoscopic device, such that the image from the right camera being shown to the observer's right eye and that from the left camera is shown to the left eye. In a basic stereo-vision system, two cameras are maintained for observing a scene. The idea behind this system is that objects closer to camera will appear more displaced between two views than those which are farther away. By comparing the two images, it is possible to find out the relative depth information in the form of a map. This map encodes the difference in horizontal coordinates of corresponding image points.

For any given point in a scene, the amount of this point's motion between two views is called *disparity*. If the camera system is calibrated in world coordinates, then the actual distance to a point can be found out using the magnitude of its disparity. Given any two images of a scene, the disparity between two is often the intensity image having bright points corresponding to larger intensities, i.e., the things that are closer to the cameras.

Example 21.1 Finding out the dept using stereo vision.

Let a simple pinhole camera model has a focal point (optical center) o, such that all the rays of light pass through this, and are projected onto an image plane $A(x, y)$ (see Fig. 21.4).

Note that in case of human eyes the image plane is retina of eye. In case of camera, it is medium (i.e., the $2D$ surface) inside the camera where image is formed. The *optical axis* (parallel to axis z in this figure) of cameras is perpendicular to the image plane $A(x, y)$ and passes through the focal point o. The *focal length* f (shown as (o_l, o'_l) and (o_r, o'_r)) is the distance from the optical center o to the image plane $A(x, y)$.[4] We will consider a simple stereo camera geometry where the optical axes of the two cameras are parallel to one another, and perpendicular to the *baseline* b, that connects the two cameras' centers o_l and o_r. We assume that the both the cameras have equal focal lengths, and the origin of the world coordinate frame is placed along the baseline, at the point which is in equal distance between the two camera centers (at distance $b/2$ from each o_l and o_r).

Let the origin of the coordinate system for the left image plane L be the projection of its optic axis, i.e., o'_l. Similarly, the origin of the right image plane R is at o'_r. Each of the L and R planes are (x, y) planes.

We consider a point $p = (x, y, z)$ (see Fig. 21.4) in the world coordinate, that is projected onto plane L at location $p'_l = (x'_l, y'_l)$ and onto plane R at location $p'_r = (x'_r, y'_r)$. From the geometry of two cameras, we have

$$\frac{x'_l}{f} = \frac{x + b/2}{z}, \tag{21.7}$$

$$\frac{x'_r}{f} = \frac{x - b/2}{z}, \tag{21.8}$$

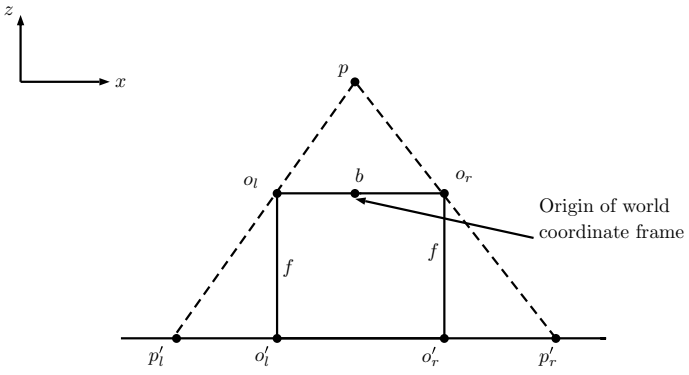

Fig. 21.4 A simple stereo camera geometry in image plane $A(x, y)$

[4]The plane (x, y) is formed by lines (o'_l, o'_r), and axis y, the latter is perpendicular to plane of paper.

$$\frac{y'_l}{f} = \frac{y'_r}{f} = \frac{y}{z}. \tag{21.9}$$

Note that in this simple camera geometry, only the x location (x'_l and x'_r) of a projected point differs between the left and right images, and the y location (y'_l and y'_l) of a given point in space is the same for both images. The disparity is distance between p'_l and p'_r, is just $x'_l - x'_r$. We have from Eqs. 21.7 and 21.8,

$$\frac{x'_l - x'_r}{f} = \frac{b}{z}. \tag{21.10}$$

Hence, if b and f are known, depth z of the point p can be computed from the disparity, giving the third dimension, hence creating a 3D view using 2D image. If b and f are unknown, the relative depths of points can be determined, and we can grasp the relative $3D$ views of two images. But, in this case, their absolute distance from the camera cannot be determined. □

21.6.4 High-Level Vision

The High-level vision systems are used to make abstract decisions, or based on the visual data they perform classification of objects. These methods usually make use of the outputs produced by low-level or middle-level vision algorithms discussed above. The high-level vision systems have main applications as object recognition systems, or as object tracking systems.

- *Object recognition systems.* These systems are used to find out whether or not a particular objects is present in the scene, and also determine the locations of other objects with respect to camera;
- *Object tracking systems.* As the name implies, the object tracking systems follow a moving object in a video sequence, and hence can guide a mobile robot or an autonomous vehicle.

In the following, we will discuss the principles used in the first application.

Object Recognition Systems

The Object recognition systems compare new image to stored object models to determine whether any of the models is present in the image. Many object recognition systems perform both the recognition and localization tasks, i.e., identifying what is the object present in the image, and also recovering its location in the image or world coordinate systems. The location of an object is called its *pose*; it is generally specified using a transformation that maps the model coordinate system to the image or world coordinate system.

The object recognition systems can be classified based on the type of problems they solve. A simple recognition task requires identification of 2D objects that are

fully un-occluded (all are fully visible), they appear in a uniform background, and the lighting conditions are controlled, i.e., there are no reflections or shadows in the view. The large majority of industrial inspections fall in this category of object recognition, and can be handled accurately using the commercially available systems.

The object recognition problems are difficult, in the following conditions:

Background of the objects is highly textured,
Lighting conditions are unknown and uncontrolled,
Scene comprises too many objects,
Number of objects models are very large in the storage, and
Some objects may be touching and occluding other objects.

In our discussions, we will consider methods to handle images with multiple objects, the objects are partially occluded, and there is some amount of cutter in the background. The methods that can solve the object recognition with the presence of these challenges, first extract geometric information, like intensity edges from an image. Then, the next step is to compare 2D geometry with 3D geometry.

An alternative method is computing invariant representations of the image geometry, i.e., it remains unchanged in spite of changes in the viewpoints. Such representations can be used to create an index of object models in a library of collections of object models. One of the challenges in geometric recognition is to find out which portion of the image corresponds to a given object from the library collections. The *recognition problem* is often defined as recovering a correspondence between local features of an image and an object model. There are three classes of approaches as how to search possible match between a model and image feature.

1. *Transformation space methods.* These methods use the space of possible transformations that map the model to the image.
2. *Correspondence methods.* These methods use the space of possible corresponding features to match with the image.
3. *Hypothesize and test.* The hypothesize and test methods consider k-tuples of model and data features for matching with the image.

We will be discussing the correspondence-based methods in more detail.

Correspondence Search

It is a tree search to find out which model's features match with the image features. Considering a set of *image features* as $A = \{a_1, \ldots, a_n\}$, and a set of *model features* as $M = \{m_1, ..., m_r\}$, an interpretation of the image can be expressed as a set of pairs, $N = \{(m_i, a_j), ...\}$. The interpretations states, which model features correspond to the image features.

If the model features are unclear (hidden or occluded), and some features are outside the image features, in that case the set N is any subset of all "pairs of model and image features." A search method used, called *interpretation tree approach* makes use of pruned search in otherwise exponential space of possible interpretations. In these interpretations, the method makes use of all possible ways of pairings of

models and image features. In simple terms, the search method makes use of pairwise relations between features to prune a tree of possible model and image feature pairs. A traversal path in this tree corresponds to a sequence of interpretations.

To limit the number of pairs (m_i, a_j) and (m_p, a_q) in the interpretations, the distances $\|m_p - m_i\|$ and $\|a_q - a_j\|$ must be equal, i.e., are within the allowable error limits.

The search for interpretations of an image, a pruned-tree search algorithm is used, which works as follows:

- Each level of the tree, other than root node, corresponds to an image.
- Each branch at a node corresponds to a *model feature* or there is special branch called *null face*.
- Accordingly, each node of the tree specifies a pair: ⟨image feature at that level, model feature taken from the previous level⟩. This model feature may be null also.
- The three search carried out is Depth-First search (DFS). Any given node is expanded only when it is pairwise consistent with all the nodes along the path from the current node back to the root of the tree. That is, a given node is paired with each node along the path back to the root, and for each such pair, if the distance constraint is satisfied, the node is expanded otherwise not.

In the above, the null face branch is considered as always consistent.

A path from root to leaf node of the search tree indicates that zero or more model features exists. However, a null branch path does not account for a model feature. A path that accounts for k model features is called *k-interpretation*. Consider that a reference level to filter out any hypothesis that does not account for enough model features, is set at a threshold, say k_0. In this case, a matcher algorithm reports only those k interpretations whose threshold is $k > k_0$. All these k-interpretations are guaranteed to be pairwise consistent. Hence, an additional step, to verify the model is carried out. This step estimates the best transformation for each k-interpretation and checks that this transformation brings each mode feature within some error range of each corresponding image feature.

21.7 Indexing and Geometric Hashing

The significance of indexing in vision is to match the images efficiently with the models stored in the repository. Having this facility, it is possible, in principle, to search an object in a library of stored models, with time complexity, that is independent of number of models stored. To carry out the indexing, it requires finding the *geometric invariants* in model recognition—those attributes of images that do not change due to changes of viewpoints of objects. One key difference of this model indexing with hashing is that, various instances of the same object in different images will not generate exactly the same key due to sensing uncertainty.

There are number of methods to use invariants in object recognition. These are categorized based on: geometric differential, photometric, and thermal properties

of images. The indexing methods for structures make use of combinations of simple features such as points and segments. Other geometric techniques use invariant properties of curves and co-planar conics. The advantage of using curve features is lower combinatorial complexity, but it has disadvantage of requiring to extract features from noisy and cluttered images. However, the intensity information from images provides their richer description than the geometric features, like points and line segments.

The *geometric hashing* is used to explain the invariants in recognition. For this2D *affine transformation* is used, with geometric hashing using three points to define a coordinate system, and with respect to these points, the other points are encoded in invariant manner. The geometric hashing comprises the following two basic steps:

a. Construction of a model hash table, and
b. Matching of model to image.

The hash table stores redundant and the representation of each object that are transformation-invariant. For any given model, each ordered triple, m_1, m_2, m_3 forms an affine basis having origin as $o = m_1$, and axes $x = m_2 - m_1$, $y = m_3 - m_1$. For each such basis every additional model point m_i is expressed as (α_i, β_i), such that $m_i - o = \alpha_i x + \beta_i y$. The basis triple (o, x, y) and the point m_i are then stored in a hash table using affine invariant indices (α_i, β_i). For r number of model points, a table results with $O(r^4)$ entries, i.e., space complexity is combinatorial. The table is formed using buckets instead of purely a hashing scheme. This is because, due to the uncertainty in sensing in real-life data makes it difficult to use exact values for retrieval purpose.

At the time of recognition, the hash table is searched to find out what models are present in the image. The algorithm steps for recognition of an object are as follows.

i First, the image points are rewritten in terms of an *image basis*.
ii As the next step, corresponding to an instance of a model that model basis will be retrieved from the table.
iii An image basis is formed using each ordered triple of image points s_1, s_2, s_3, such that origin is $O = s_1$, and axes are $X = s_2 - s_1$, $Y = s_3 - s_1$.
iv For this image basis, each additional image point s_i is written as (α_i', β_i'), such that $s_i - O = \alpha_i' X + \beta_i' Y$.
v The indices (α_i', β_i') are used for retrieving matching model basis from the hash table. Along with retrieving each model basis, corresponding counter is incremented in a histogram.
vi When all the retrieved points for the model basis have been considered for an image basis, the histogram contains the votes for those model bases which could match to the current image basis (O, X, Y).
vii If peak in the histogram for a given model basis (o, x, y) is above a threshold, the corresponding basis is selected as a potential match. When a next image basis is chosen, the histogram counts reset.

21.8 Object Representation and Tracking

The availability of inexpensive high-quality video cameras and the increasing demand for automated video analysis, has resulted to lot of interest for *object tracking*. The video analysis comprises three key steps: 1. detection of moving objects, 2. tracking of these objects from one image frame to another frame, and 3. analysis of the object tracks to recognize the object behavior, and sometimes its geometry also.

The goal of object tracking is to generate *trajectory* of an object by locating its positions in every frame of the video, over a time. Object tracking also provides information about the region occupied by the object at every time instant in the image. The two tasks, i.e., detecting the object, and establishing correspondence between the object instances from frame to frame, can either be performed separately or jointly. When only one job is required to be performed, i.e., to detect an object, the possible object regions in every frame are obtained using an algorithm for object detection. Then, as next step, the tracker can find the correspondence in frame sequences.

In the second approach, when detection and tracking are done together, the object regions and the correspondence is jointly estimated through iteratively updating the object location and region information. The information about location and region are obtained from the previous frames. What type of model is selected to represent an object's shape can decide the type of motion or deformation it can undergo. For instance, if an object is represented as a point, then, only a translational model can be used for this object [10].

In situations where a geometric shape representation, e.g., an ellipse is used for the object, parametric motion models like *affine*[5] or transformations like *projective* are preferred. The motion of any rigid object in a scene can be approximated by these representations. For nonrigid objects, *silhouette* or *contour*-based approach is preferred, being the most descriptive form of representation. Both the parametric as well as nonparametric models can be used to specify their motion using these approaches.

It is possible to simplify the object tracking job by restricting the motion of the object or its appearance, or both of these; for example, assuming that object motion is uniform, i.e., constant velocity, or constant acceleration. The objects can be represented in many ways, but some representations are considered as more suitable than others. For instance, fish in an aquarium, boats in the river, vehicles on a road, and birds in the air, are the sets of objects that may be important to track in a specific domain. Depending on the type of objects, they may be represented by their appearances and shapes.

- *Points.* An object is represented by a point as its centroid (see Fig. 21.5a), or can be represented by a set of points, as shown in Fig. 21.5b.

[5] Affine transformation is a linear mapping method in geometry, such that it preserves points, straight lines, and planes. For example, after an affine transformation, a set of parallel lines remain parallel, say, a transformation from parallelogram to rectangle and vice versa. Affine transformation is typically used to correct for geometric distortions or deformations that occur with nonideal camera angles.

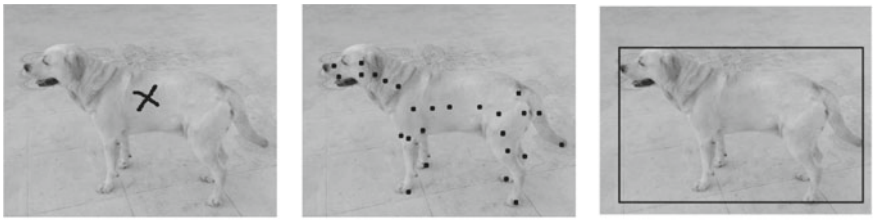

Fig. 21.5 Object representations-I: **a** Centroid **b** Multiple points **c** Rectangular patch

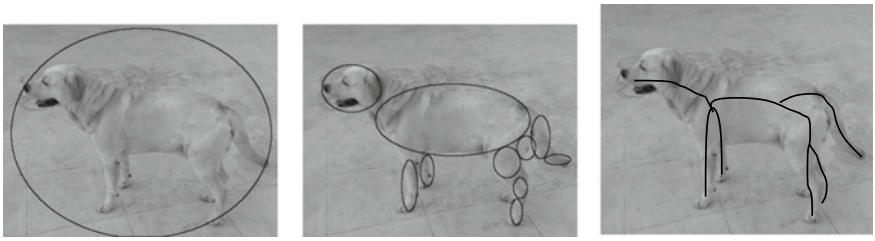

Fig. 21.6 Object representations-II: **a** Elliptical patch **b** Part-based multiple patches **c** Object skeleton

- *Primitive geometric shapes.* An Object can be represented using a shape, e.g., a rectangle, or an ellipse (see Fig. 21.5c, 21.6a).
- *Articulated shape models.* The articulated objects comprise body parts, which are held together with the help of joints. For example, a pet's body is an articulated object with hands, legs, head, feet, and the torso connected together by joints (see Fig. 21.6b).
- *Skeletal models.* In these models, the object skeleton can be extracted by applying medial axis transform to the object silhouette (see Fig. 21.6c).
- *Object contour.* The contour representation defines the boundary of an object (see Fig. 21.7a, b).
- *Object silhouette.* The region inside the contour is called the *silhouette* of the object (see Fig. 21.7c).

The object tracking is carried out to estimate the trajectory of an object in the plane of the image, as the object moves in the scene. During the tracking, the tracker assigns labels to the tracked object to mark its positions in different frames of a video. Apart from this, depending on what is the domain used, a tracker can also provide object-centric information, such as area, shape of the object, and about orientation of the object. However, the tracking process sometimes turn out to be complex, due to reasons mentioned below.

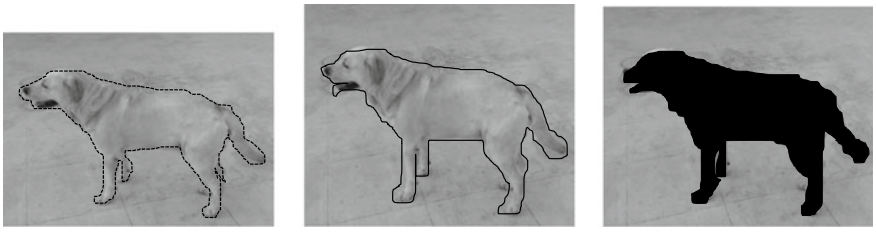

Fig. 21.7 Object representations-III: **a** complete object contour (dotted lines) **b** control points on object contour **c** object silhouette

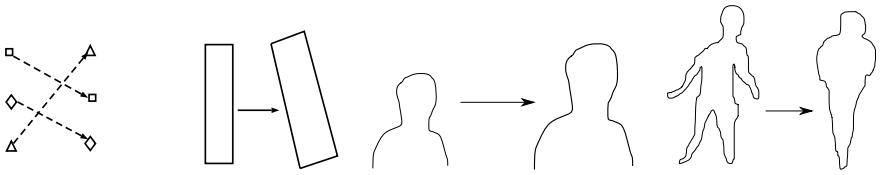

Fig. 21.8 Object tracking. **a** (i) Multipoint correspondence (ii) Kernel Tracking **b** Evolution of contours **c** Silhouette Tracking and Evolution of contours

Complex object motions,
Articulated or nonrigid nature of objects,
Occlusion of objects in full or partially,
Complex shapes of objects,
Illumination changes in the scene,
Need of real-time processing of the scenes,
Loss of information due to projection of $3D$ world on a $2D$ images, and
Image noise.

The following is brief introduction to the commonly used tracking categories:

Point Tracking

In point tracking, objects detected in consecutive frames are represented by points. The association between consecutive points is established based on the previous state of the object, which may comprise object position and its motion. This approach needs a separate external mechanism to detect the objects in every frame. An example of correspondence between the objects is shown in Fig. 21.8a, with mapping through broken lines.

Kernel Tracking

A kernel is concerned with the object shape and its appearance. For example, a kernel can be a rectangular template or an elliptical shape with an associated histogram (see Fig. 21.8a, with rectangles). The objects are tracked by computing the motion of

the kernel in consecutive frames. The motion is usually in the form of a parametric transformation, like rotation, translation, or affine.

Silhouette Tracking

Object tracking can also be performed by estimating the object's region in each frame. Information encoded inside an object's region is used by the Silhouette tracking, which is either in the form of appearance density or shape models (i.e., in the form of edge maps). Further, given an object model, the silhouettes are tracked by matching the shapes (Fig. 21.8b), or they can be matched by contour evolution (Fig. 21.8c). Both of these methods can be used for object segmentation, which is applied in the temporal domains.

The following are major applications of object tracking.

- *Video indexing.* It comprises automatic annotation and retrieval of the videos in multimedia databases.
- *Motion-based recognition.* Motion-based recognition has applications in identification of humans based on gait and in automatic object detection.
- *Automated surveillance.* It is concerned with monitoring a scene to detect suspicious activities or unlikely events.
- *Human–computer Interaction.* The HCI is concerned with applications, like gesture recognition and eye-gaze tracking, for data input to computers.
- *Vehicle navigation.* It is used in video-based path planning and obstacle avoidance capabilities.
- *Traffic monitoring.* It is concerned with real-time gathering of traffic statistics to direct the traffic flow.

21.9 Feature Selection and Object Detection

The feature selection is a process of deciding those attributes of a picture that help in identifying the object in the picture frame, while object detection or object identification is a process to identify the object in the frame, based on these features. Therefore, it is necessary that the features decided should give a strong evidence of the corresponding object.

Feature Selection

To efficiently distinguish the objects in the feature space, it is desired that the visual features be unique. The selection of feature is related closely with the objects representation. For example, for histogram-based representations, the color is a required feature, and for contour-based representations, object edges are important features. Many times, the features with various combinations are useful. Some examples of features as given below [10].

Color

The color, in appearance of an object, is primarily influenced by two physical features of the object: (1) light source's spectral power distribution, and (2) object's surface reflection properties. The color space used for an object is RGB (red, green, blue). Since, the RGB does not have uniform perceptions, HSV (hue, saturation, value)—more closer to uniform color space—is preferred as a representation to obtain uniform perception.

Edges

Since the object boundaries are the cause of strong changes in image intensities, the edge detection is used to identify these changes. The edges intensities are also less sensitive to the level of illumination. Hence, the algorithms used for tracking boundaries of the objects, make use of edges as the important representative features of the image.

Texture

Texture of an object surface causes an intensity variation in the reflected light from its surface, hence quantifies properties of the object surface such as smoothness and regularity. The texture requires an additional processing step to generate the descriptors.

An object's representative features usually depend on the application domain; however, the automatic feature selections is in common practice. The features are either *filter* based or they are *wrapper* based. The methods that are filter based, make use of general criteria, e.g., the features need to be correlated. On the other hand, the wrapper methods select the features based on their usefulness in the concerned problem domain, for example, classifying the performance using the subset of features. An example of filter method, called Principle Component Analysis (PCA), is useful in reduction of number of features. The PCA is based on transformation of possibly a number of correlated variables into a smaller number of variables that are not correlated, called principal components. The first principal component chosen should account for maximum possible variability of the data, and each succeeding component should account for most of the remaining variability of the components possible. *Adaboost* algorithm is an example of wrapper-based method for selection of discriminatory features for tracking a class of objects. This method identifies a strong classifier using a combination of moderately less accurate and weak classifiers. Having given a large set of features, it is possible to train one such classifier for each feature [3].

Among all the features used for object tracking, color is the one most widely used feature. The colors' histogram can be used to represent the object's appearance. In spite of the popularity of color as an important feature, large majority of color bands are sensitive to variation in illumination. Thus, in situations where this effect cannot be stopped, other features are considered to model the object's appearance.

21.9.1 Object Detection

A tracking method always requires a mechanism for object detection, either in every frame or when the object appears in the video for the first time. The commonly used approach for object detection is to use the information from a single frame. However, some method makes use of temporal information computed from a sequence to reduce the false detection of objects. The temporal information is in the form of *frame differencing*, i.e., it highlights the changes in regions from frame to frame. Based on the object regions available in an image, the trackers establish correspondence from one frame to next frame of same object, and generate the tracks [10].

Some commonly used methods for object detection as discussed below.

Point Detectors

The point detectors have application in finding the *interest points* in an image that have expressive texture in their localities. An important characteristic of interest point is its invariance with respect to changes in the illumination and camera viewpoint. A procedure, called *Moravec's* operator-based method, is used for finding the interest points, which computes the variation in image intensity in a 4×4 patch in horizontal, vertical, diagonal, and anti-diagonal directions, and selects the minimum of the four variations as representative values for the window. A point in this case is declared as interest point if the intensity variation in a local maxima is 12×12 patch.

Image Background Subtraction

It is possible to build a representation of a scene, called *background model*, and then to find the deviation from the model for the incoming frame, to detect the object. If there is a significant change in the image region from the background model, then it indicates that the object is moving. This approach can also be used to detect if a new object has been introduced into the scene, or some object has escaped from the scene. In this method, the pixels representing the regions that undergone the change are marked for further processing. In such cases, it is common to use connected component algorithm to obtain the required objects in the image. The process used, as it appears, is called *background subtraction*.

Figure 21.9 shows background subtraction results: The part (a) of figure shows an input image of highway with moving vehicles, pedestrians, and nearby houses as objects, the part (b) is reconstructed image after projecting input image onto the eigenspace, and part (c) is an image obtained as difference of two images.

Image Segmentation

Aim of an image-segmentation algorithm is to partition an image into perceptually similar regions. A segmentation comprises solution of two problems: 1. What is criteria for good partition, and 2. What is procedure for achieving efficient partitioning? In the following we discuss some of commonly used segmentation techniques that are useful for object tracking.

Fig. 21.9 Background subtraction. **a** input image with objects **b** reconstructed image **c** difference image

Image segmentation using mean-shift clustering

This approach for image segmentation finds the clusters in the joint color+spatial space, $[u, v, w, x, y]$, where $[u, v, w]$ represents color, and $[x, y]$ represents the spatial location in $2D$ system. The color may be either RGB or HSV. The algorithm steps are as given below.

i. For any given image, large number of random hypothetical cluster centers are initialized, which are chosen from a given data set;
ii. Each cluster center is moved to the mean of the data lying inside a multidimensional ellipsoid whose center is the cluster center;
iii. Using the old and new cluster centers a vector, called as *mean-shift vector*, is defined;
iv. This mean-shift vector is iteratively computed until the cluster centers are stabilized (i.e., they do not change their positions).

Image segmentation using graph cuts

The image-segmentation problem can also be solved like partitioning of a graph. Consider that $\mathbf{V} = \{v_1, v_2, \ ..., v_n\}$ be vertices (pixels) in a graph (image), and the graph is represented by $G = (\mathbf{V}, \mathbf{E})$. Let us assume that this graph is partitioned into N disjoint subgraphs, with regions $A_1, ..., A_N$, by its weighted edges $\mathbf{E}$. The partitioning is subject to the condition, that

$$\bigcup_{i=1}^{N} A_i = \mathbf{V}, \tag{21.11}$$

and

$$A_i \cap A_j = \phi, \quad \text{for } i \neq j. \tag{21.12}$$

In any graph, the total weight of the pruned edges between its two subgraphs is called a *cut*. The weights are typically computed based on color of the image, its brightness, and texture similarity between the vertices. In one approach, minimum cut (*mincut*) criteria is used, with the aim to find out the partition that minimizes a cut. The weights in this approach are defined based on color similarity. However, this approach has disadvantage of bias toward *over-segmenting* the image. This over-segmenting is caused due to increase in cost of a cut with number of edges going across the partitioned segments.

The problem of over-segmentation can be solved by using a *normalized cut*, where the cut depends on the sum of edge weights in the cut, like before. In addition, the cut also depends on the ratio of, total connection weight of the vertices in each partition to weight of all vertices of the graph taken together. For the segmentation based on images, weights between the vertices are defined by the product of the *spatial proximity* and *color similarity*. Once the weights are computed between each pair of vertices, a *weight matrix* $\mathbf{W}$ and a *diagonal matrix* $\mathbf{D}$ are computed, such that

$$\mathbf{D}_{i,i} = \sum_{i=1}^{N} \mathbf{W}_{i,j}. \tag{21.13}$$

The segmentation is performed first by computing the eigenvectors and the eigenvalues of the generalized eigensystem, as per the expression,

$$(\mathbf{D} - \mathbf{W})\mathbf{y} = \lambda \mathbf{D}\mathbf{y}. \tag{21.14}$$

Next, the second-smallest eigenvector is used for dividing the image into two segments. For each of the new segment, this process is performed recursively until a threshold is reached.

In a segmentation that is based on normalized cut, solution to the generalized eigensystem for large images can be expensive in terms of memory requirements as well as time requirements. However, this method requires far less number of manually selected parameters, compared to that of *mean-shift* clustering-based method.

21.10 Supervised Learning for Object Detection

The *supervised learning* can be used to automatically learn different object views from a set of examples to perform the object detection. Learning of different object views exempts the need of storing the complete set of templates. The supervised learning makes use of a set of examples to generate a function which maps any input to desired output. A standard formulation of supervised learning is in the form

of a classification problem. A learner in the supervised learning approximates the behavior of a function by generating an output either as continuous value or class labels. When output is continuous value, it is called *regression* model, and when the output is class labels, it is called *classification model*. For carrying out the object detection, the learning examples comprise pairs: ⟨object feature, object class⟩, with both quantities manually defined.

Selection of proper features is important for effectiveness of the classification, only those features need to be used that are able to discriminate one class from the other class. Apart from the features, like, color and texture, other features are also possible to use, such as area of the object, its orientation, and appearance. These features can be used in the form of density function, for example, in histogram. Once the features are finalized, all required appearances of an object can be learned through supervised learning only. Some of the learning approaches to supervised learning are decision trees, neural networks, and support vector machines. The learning methods compute a *hypersurface*, which separates one object class from the other in a high dimensional space.

For supervised learning to be useful, it requires a large collection of samples from each object class, where each collection has been manually labeled. Along with the supervised learning, *co-training* is also carried out to reduce the size of manually labeled data. The co-training approach trains two classifiers using a small set of labeled data, such that features used for each classifier are independent. Once training is complete, each classifier assigns unlabeled data to the training set of other classifiers. Using this method, we start with a small set of labeled data with two sets of statistically independent features, and the co-training can provide accurate classification rule. In the following, we discuss two approaches, i.e., *adaptive boosting* and *support vector machines* for object detection.

Adaptive Boosting

This method combines many base and moderately accurate classifiers to produce a more accurate classifier. The steps (of algorithm) of this classifier are as follows.

1. The training phase of the algorithm constructs an initial distribution of weights over a training set;
2. A base classifier having least error is selected by a boosting mechanism. This error is proportional to the weights of the misclassified data;
3. The weights associated with data, that are misclassified by the selected base classifier, are increased;
4. The iteration step is repeated after selecting a new classifier that performs better on the misclassified data, and the process is repeated, until there is no misclassified data.

The simple operators can act as weak classifiers for object detection, e.g., a set of thresholds. These classifiers are applied to the object features extracted from the image. In an approach where adaptive-boosting framework is used, say, for detecting the pedestrians in the presence of other traffic, perceptrons can be chosen to act as the weak classifiers. These perceptrons can be trained on the image features extracted

through a combination of temporal and spatial operators. Operators in the temporal domains will be in the form of frame differencing, and they encode some form of motion information (a required feature of pedestrians). When temporal domain makes use of the frame differencing operator, there is less likelihood of false detection, because it enforces the object detection in the region where the motion occurs.

Support Vector Machines

When used as a classifier, a Support Vector Machine (SVM)[6] is used for classifying the data into two classes by finding a maximum marginal hyperplane which separates one class from the other. The margin of this hyperplane is maximized, as defined by the distance between the hyperplane and its closest data points. The data points that lie on the boundary of the margin of this hyperplane are called *support vectors*, hence the name of this technique.

For detecting an object, these classes, separated by the hyperplane, correspond to the object class (positive samples) and the non-object class (negative samples). Using manually generated training examples, labeled as object and non-object, one hyperplane is computed from among an infinite number of possible hyperplanes using a method called *quadratic programming*.[7]

In spite of being a linear classifier, the SVM can also be used as a nonlinear classifier by application of a technique, called *kernel-trick*,[8] to the feature vector extracted from the input. The kernel-trick is applied to a set of data that is not linearly separable, to transform them to a higher dimensional space so that it becomes separable.

21.11 Axioms of Vision

The term axiom (also called *primitives*) is considered from the mathematical usage, using these as the basic building blocks it is possible to derive new constructs. This approach has the advantage of abstracting the representation, the type, and the algorithm details, within those components so that it is possible to combine them together. In the vision axioms, each axiom performs a subset of the tasks such that these tasks can be grouped together to accomplish a more complex task. The axioms are low-level enough so that the majority of the subsets of vision problems can be represented by them, and after combining them, it results in high-level solutions [5].

The axioms share common elements defined globally, such as images and elementary data types. The time, images, and colors are treated as continuous signals in the axioms of vision. For example, color is RGB, with each channel represented

[6] For more about SVM, refer p. 515 in Chap. 17.

[7] *Quadratic Programming* is process of solving a linearly constrained quadratic optimization problem (minimizing or maximizing) a quadratic function of several variables subject to linear constraints on these variables.

[8] The technique of *kernel-trick* does not require explicit mapping used for linear learning algorithms to learn a nonlinear function.

in the interval of [0, 1]. The width and height of an image are represented in similar way, with additional value for the aspect ratio. Since time is treated as a continuous signal, it allows different inputs to the vision system.

The classes of common axioms for vision are: mathematical axioms, source axioms, model axioms, and construct axioms. These are based in a problem-centric taxonomy. Each axiom is further divided into three parts: a. description of its tasks, b. its representations, and c. the processing it would perform when implemented.

21.11.1 Mathematical Axioms

The functionality is encapsulated in these axioms, such as to perform transformations, optimizations, and other necessary formulations in machine vision.

Optimize

This axiom comprises optimization functions of dynamic programming, linear programming, graph cuts, and greedy methods.

Transform

The necessary mathematical descriptions are contained in this axiom to carry out the transformation of vision objects. It includes operations of linear algebra, geometry, and other formulations necessary for computer vision. The transformation may comprise the operations, like matrix multiplications and projections.

Similarity

The similarity axiom is used for recognition, correspondence, and tracking. It comprises various matrices for evaluating similarity.

21.11.2 Source Axioms

The source axioms are used for encapsulating the source information. Thus, they also describe the acquired data or the source of the acquired data, such as cameras, images, and properties of images like noise and illuminations in the images.

Noise

This axiom provides the description of noise in the data.

Light

The lighting models are required to approximate the lighting in a scene. It may have simple systems like in computer graphics, or variation of intensity across images, to add the effect of flash, or combining of the images, etc.

Camera

The camera axioms provide the general description of the camera, and also include other models and specific parameters for calibration, like focal length, principal point, etc.

Image

The image comprises descriptions for low-level image processing techniques, and representation of image, like resize and filter operations.

Blur

The blur is due to motion of camera or the object. The focus is deliberately used for creative effect or measurement of depth from focus.

21.11.3 Model Axioms

The model axioms are used for representing abstract concepts used within computer vision, such that they are accessible.

Model

This axiom contains descriptions of models used in the vision, like geometric model, probabilistic model, or example-based model. A model unit can provide conversion from one format to another format.

Object

This axiom provides types and the descriptions of objects.

Shape

It finds the similarly shaped regions within an image, which is carried out in conjunction with matching and similarity axioms.

Texture

It performs synthesis and analysis with the help of texture descriptor. It also finds regions of similar texture, with the help of matching and similarity axioms.

21.11.4 Construct Axioms

The construct axioms represent constructs used within vision for the purpose of modeling and used as output to other modeling packages.

Mesh

The mesh axiom is for centralized description of a mesh, using triangles, quads, etc. A mesh unit provides an input–output system, for construction of meshes from an image base or from an arbitrary basis in $3D$, for reconstruction of output.

Grid

The grid axiom is used for providing N-dimensional grids' description, for use in vision. For example, a $2D$ grid would be a discretized image, and a $3D$ grid would be a set of voxels,[9] etc. A grid can be indexed at intersections of grid lines. A grid unit also provides methods of grid construction, conversion, and for IO methods.

Algorithm Composition

The vision axioms can be combined together through loose coupling to form more sophisticated algorithms. The basic axioms already encapsulate the sophisticated concepts and algorithms; and combining these we can create higher level systems such as correspondence systems, tracking, $3D$ reconstructions, etc. As an example, for creating a simple $3D$ reconstruction such as *visual hull*, a camera can provide calibrations, grid can provide volumetric constructs, transformation will give projected grid points, and mesh can create a model for output from volumetric grid.

It is possible to create a simple tracking system with direct use of axioms: initialization of this system is done using *object*, *matching*, and *similarity*. And then, an object can be tracked through a sequence of images through use of *move* and *optimization* axioms.

However, the algorithm composition is not suited to be used by developers, because it requires expert knowledge of vision. Instead of that use, we look at top most level, which is problem itself. This is because the problem is decomposed into smaller level parts, and we use this as a basis for solving the problem. This simplifies the implementation of computer vision using the axioms.

21.12 Computer Vision Tools

The open source computer vision library, the OpenCV is available under a Berkley Software Distribution (BSD) license and it is free for use both by academic and commercial users. The OpenCV has interfaces with C++, C, Python, and Java, and it is supported by all major Operating Systems. OpenCV was designesd keeping the computational efficiency as one of the goals, and with a strong focus on real-time applications. The OpenCV is written in optimized C/C++ code, the library can take advantage of the availability *multicore processing* in the computer being used for running these library programs. The OpenCV is enabled with OpenCL, and it can take advantage of the hardware acceleration of the underlying heterogeneous

[9] *Voxels:* Each array of elements of 3D volumetric space.

computer systems. Usage of OpenCV range from interactive art, to mines inspection, to stitching maps on the web through advanced robotics [6].

The OpenCV is designed with other goals of building tools for solving computer-vision problems. It comprises a mixture of low-level image processing functions and high-level algorithms for the applications of, face recognition, face detection, pedestrian detection, feature matching, and object tracking.

The following is an example of a simple program in OpenCV to load and display a given image file, whose name is provided as input at Linux command line [7].

Example 21.2 Load and display an image file.

```
/* loaddisp.cpp, OpenCV ver 3.2.0 */
#include <opencv2/highgui/highgui.hpp>
#include <opencv2/core/core.hpp>
#include <iostream>
using namespace std;
using namespace cv;
int main(int argc, char** argv)
{
    if(argc != 2)
    {
     cout << "error" << std::endl;
     return -1;
    }

    Mat imagemat;
    // Read image file
    imagemat = imread(argv[1], CV_LOAD_IMAGE_COLOR);
    // Check for invalid input
    if(! imagemat.data)
     {
        cout <<  "error" << std::endl;
        return -1;
    }
    //Create a display window
    namedWindow("Display Window", WINDOW_AUTOSIZE);
    //Show the image inside window
    imshow("Display Window", imagemat);
    //press any key to terminate the program
    waitKey(0);
    return 0;
}
```

The above program is compiled at command level in Ubuntu 18.04.1 by

```
g++ loaddisp.cpp -ggdbJpkg-config --cflags --libs opnecvJ
```

and run by command

```
./a.out krc.jpg
```

On run, the screen will display the image in file krc.jpg. □

All modern smartphones and most tablets also contain one or more cameras, and OpenCV is also available on both Android and iOS operating systems. With all these components, it is possible to create vision applications for mobile platform also.

21.13 Summary

The vision system has significance due to the fact that, it is one of the essential characteristics of living beings, and specifically, all the higher animals depend on this ability for their survival. The vision gives us detailed information about and around us, without much conscious efforts.

The goal of *computer vision* or *machine vision* is to extract the information from images, so that one can produce a structure from image, e.g., a 3D view from source images that are 2D. Having this available, it can recover the 3D model of an object, which in turn can be useful in medical imaging to recover 3D objects, say tumor. The methods can find best matches for stored objects of image data, for example, we can search some individual person in a collection of library of photographs, or in collection of video.

Initially, the vision systems were designed for military applications, like visual guidance for autonomous vehicles, target recognition, etc. However, in recent many applications have emerged in medical, for example, for preoperative scan for patients, virtual reality, image-based data retrieval, assembly and material handling, visual inspection, and photo interpretation.

Every object, other than mirror, scatters the light incident on it in all directions, this reflected light is projected on the retina of our eye through a lens, and we are able to recognize the object, through the interpretation by brain. The vision problem, which is related to this statement is, given the 2D image, how to infer the shape of the object that produced this image. However, it is a challenge to reason back from image to the scene that created it. The recognition requires learning, i.e., mapping image patterns to some internal representation—transformation of scene to image. This requires construction of models, i.e., general description of each object that is to be recognized. A model includes shape, texture, and context knowledge.

Going from image-to-scene is a challenging task. A vision system receives only the reflected brightness as input in different intensities. To convert these brightnesses into world measurements, we first need to understand how objects are mapped into image

brightnesses. The light received by individual sensor call is a complex interaction of: position and object's orientation, its 3D shape, reflection properties of its surface, properties of light source as well as its spectral sensitivity.

For vision, what is required is inversion of brightness into scene parameters, using any of the following methods: inversion by fixing scene parameters, by restricting problem domain, or by acquiring additional images.

A general approach to computer vision is that a vision process ends with 3D reconstruction of shapes by means of *geometric primitives* at symbolic level, linguistic level, and intermediate. However, the interpretation of the conceptual categories at linguistic levels involves problems as the concepts hardly correspond to classic categories that can be described in terms of necessary and sufficient conditions. In fact, the membership of the categories is not 0 and 1. Hence, knowledge in conceptual categories is affected by measurement errors. The neural networks make it possible to avoid the exhaustive description of conceptual categories, accordingly, they are better fit as models.

Depending on the complexities involved, the vision is classified as low-level vision (image and output are in same coordinate system), medium level vision (input is low-level vision, and output is something other than pixels of the image, say 3D image), and high-level vision (input is low and middle-level vision algorithms, and output is abstract data structures).

One of the applications of vision system is *indexing* of images, or scenes in a video. Having this facility, it is possible to lookup an object in a library of stored models, and that too in time that is independent of number of models in storage. To carry out the indexing, it requires finding the *geometric invariants* in model recognition, i.e., attributes of images that do not change due to change of viewpoints of the object. For indexing, we make use of geometric hashing approach, which has two basic steps: 1. construction of a hash table for models, and 2. matching of model to the current image.

One important area of vision systems is *object tracking*. Due to lot of demand for automated video analysis, interest in video analysis has increased. The video analysis has three steps: a. detecting the moving objects, b. object tracking from frame to frame, and c. objects' analysis to recognize their behavior. The major applications of object tracking are: automated surveillance, video indexing, motion-based recognition, traffic monitoring, Human–Computer Interaction (HCI), and vehicle navigation.

The tracking process is a complex task due to loss of information caused by projections, noise in images, partial and full object occlusions, complex shapes of the objects, and the requirements of real-time processing.

To detect an object, point detectors are used to: find points of interest, subtraction of image background, and segmentation of the image. Object detection is performed by learning different views from a set of examples using supervised learning. A standard formulation of supervised learning is: classification problem, such that the learner approximates the behavior of a function by generating an output in the form of either continuous value, called *regression*, or labeling of classes, called, *classification*.

The *support vector machines* and *adaptive boosting* are the examples of supervised learning.

The complexity of vision systems, where basic primitives are identical, for example, in 3D objects, and in dynamics of objects, can be simplified by defining and constructing *axioms* of vision. These are the basic building blocks from which we derive new constructs. The axioms are ways for representations, as well as correspond to certain algorithms within these, and are useful for common subtasks in vision systems.

The common axioms in vision are: mathematical axioms (transformation, optimization, linearity), source axioms (camera, image, light), model axioms (object, shape, texture), and construct axioms (mesh, grid, algorithm).

One of the important tool for Vision system research is the open source computer vision library, OpenCV released under a BSD license, is free for academics as well as commercial use. It provides C++, C, Python, and Java interfaces, and has been designed for efficiency of computation with a focus mainly on real-time applications.

Exercises

1. Suggest some experimental observations about human vision that support the idea that *vision is graphics* — what we see is explicable only partly by the optical image itself, but is more strongly determined by top-down knowledge, model-building, and inference processes.
2. Explain in your own words, with supporting geometry, as how the human vision system inverts the image formed on the retina of eye(s) to visualize the object.
3. How the vision is a learning and cognition process? Justify.
4. Why it is deterministically not possible to recognize an object given an image?
5. We can always map the coordinates from an object to its image, but mapping the image coordinates to object is challenging. Why?
6. Suggest any five new applications of machine vision, which are not discussed in this chapter.
7. Most smartphones available now have feature of face detection. Why this feature is important? Explain the process of face detection in your own words.
8. What are the *syntax* and *semantics* information in a vision? Explain in brief.
9. How the axioms of vision are helpful in machine vision?
10. What is image inversion? Explain some of the techniques for this, and the significance of each of them.
11. Give proper examples of low-level, middle-level, and high-level visions.
12. Like, two stereo cameras, our eyes are able to find out the depth of vision, e.g., how far roughly it is from the eyes. Write an algorithm, to compute this depth of vision.
13. Does the width between the centers of human eyes, anyway make difference in computing accuracy of depth of vision? Give a proof for this, based on some computations.

14. Given any image having number of human faces, write an algorithm, to count the number of faces in the given image.
15. Write an algorithm for panorama-stitching for given n number of images.
16. Suggest various approaches to speedup the processing of image transformations.

References

1. Charniak E, McDermott D (1998) Introduction to artificial intelligence, Addison-Wesley publication
2. Chella A et al (1997) A cognitive architecture for artificial vision. Artif Intell 89:73–111
3. Felzenszwalb P et al (2013) Visual object detection with deformable part models. Commun ACM 56(9):97–105. https://doi.org/10.1145/2494532
4. Grimson WEL, Mundy JL (1994) Computer vision applications. Commun ACM 37(3):44–51
5. Miller G et al (2011) A conceptual structure for computer vision. In: Canadian conference on computer and robot vision. https://doi.org/10.1109/CRV.2011.29
6. https://opencv.org/. Cited on Dec 2017
7. Pulli K et al (2012) Real-time computer vision with OpenCV. ACMQUEUE 10(4):1–17
8. Suetens P et al (1992) Computational strategies for object recognition. ACM Comput Surv 24(1):5–61
9. Tucker AB Jr (1997) The computer science and engineering handbook, CRC Press
10. Yilmaz A et al (2006) Object tracking: a survey. ACM Comput Surv 38(4):1–45