OBJECT RECOGNITION USING SURF ALGORITHM

Project report submitted in the partial fulfilment of the requirements for the award of degree of

BACHELOR OF ENGINEERING
In
COMPUTER SCIENCE AND ENGINEERING

By

Sunnyhith Kanakanti
(160112733057)

Under the guidance of

Mrs. G. Kavita
Assistant Professor
Dept. of CSE
CBIT, Hyderabad

Department of Computer Science & Engineering
Chaitanya Bharathi Institute of Technology
Gandipet, Hyderabad – 500075
MAY 2016

A Project Report
on

OBJECT RECOGNITION USING SURF ALGORITHM

Submitted for partial fulfilment of the requirements for the award of the degree
of

BACHELOR OF ENGINEERING

IN

COMPUTER SCIENCE AND ENGINEERING

BY

Sunnyhith Kanakanti (160112733057**)**

Under the guidance of

Mrs. G. Kavita
Assistant Professor
Dept. of CSE
CBIT, Hyderabad.

Department of Computer Science and Engineering
Chaitanya Bharathi Institute of Technology
Hyderabad-75
May-2016

Chaitanya Bharathi Institute of Technology
(Regd. No. 855/2009)
(Affiliated to Osmania University, Accredited by NBA (AICTE), Accredited by NAAC (UGC), ISO certified 9001 : 2008)
Chaitanya Bharathi P.O., CBIT Campus, Gandipet, Kokapet (V),
Rajendranagar Mandal, Ranga Reddy District, Hyderabad - 500 075
e-mail : principal@cbit.ac.in; principalcbit1979@gmail.com; Website: www.cbit.ac.in
: 040 - 24193276; 24193277; 24193280; Fax: 040 - 24193278

CERTIFICATE

This is to certify that the project work entitled "**OBJECT RECOGNITION USING SURF ALGORITHM"** is a bonafide work carried out by **Sunnyhith Kanakanti (160112733057)** in partial fulfilment of the requirements for the award of degree of **BACHELOR OF ENGINEERING IN COMPUTER SCIENCE AND ENGINEERING** by the **OSMANIA UNIVERSITY**, Hyderabad, under our guidance and supervision.

The results embodied in this report have not been submitted to any other university or institute for the award of any degree or diploma.

Project Guide
Mrs. G. Kavita
Asst. Professor
Department of CSE
CBIT, Hyderabad

Head of the Department
Dr. Y Rama Devi
Professor and Head
Department of CSE
CBIT, Hyderabad

DECLARATION

This is to certify that the work reported in the present project entitled "**OBJECT RECOGNITION USING SURF ALGORITHM"** is a record of work done by me in the Department of Computer Science and Engineering, Chaitanya Bharathi Institute of Technology, Osmania University. The reports are based on the project work done entirely by me and not copied from any other source.

Sunnyhith Kanakanti

(160112733057)

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude and indebtedness to my project guide Mrs. G. Kavita for her valuable suggestions and interest throughout the course of this project.

I am also thankful to Head of the department Dr. Y Rama Devi for providing excellent infrastructure and a nice atmosphere for completing this project successfully.

I convey my heartfelt thanks to my project coordinators Dr. CRK Reddy and Smt. T Sri Devi, and also to the lab staff for allowing me to use the required equipment whenever needed.

Finally, I would like to take this opportunity to thank my family for their support through the work. I sincerely acknowledge and thank all those who gave their support directly or indirectly in completion of this work.

Sunnyhith Kanakanti (160112733057)

ABSTRACT

Object recognition, as the name implies, is the process of identifying and determining a specified object in a digital image or in a video. This is the central research topic in Computer Vision. Computer Vision is a field that includes methods for acquiring, processing, analyzing, and understanding a single image or a sequence of images from the real world in order to produce numerical or symbolic information.

Computers are unbeatable at processing speeds and at finding the best solutions for many complex problems, multi-tasking various processes, etc. But a computer cannot think. It cannot see the world in the way we see it. A computer cannot understand the environment it is in, unless directed and specified by humans.

This project is an attempt to bring about a change in how a computer sees the world, sees the objects in it. OpenCV has been used for this project. OpenCV is the acronym of Open source Computer Vision. It has a huge collection of programming functions and libraries aimed to improve the vision of the computer. Among these, libraries for keypoint localization, keypoint descriptor, and keypoint matching have been used. SURF (Speeded Up Robust Features) algorithm is used for keypoint localization and keypoint descriptor. Brute Force Matcher is used for matching the keypoints. A query image of an object is taken as an input from the user. The keypoints and descriptors are extracted from the query image and are compared with the already extracted keypoints and descriptors of objects images located in the dataset. If enough number of good matches are found, then the name of the object is displayed.

LIST OF FIGURES

Figure 2.1	The Basic Structure of OpenCV	9
Figure 2.2	Haar Wavelets	12
Figure 2.3	Orientation Assignment	14
Figure 2.4	Waterfall Model	16
Figure 3.1	UML Diagrams	18
Figure 3.2	Level-0 DFD for the system	19
Figure 3.3	Level-1 DFD of the system	20
Figure 3.4	Level-2 DFD of the system	21
Figure 3.5	Activity Diagram of the system	22
Figure 3.6	State Chart Diagram of the system	23
Figure 5.1	Keypoints and Descriptors extracted to a file	27
Figure 5.2	Keypoints and Descriptors of a sample image	28
Figure 5.3	Matches between Keypoints and Descriptors of two images	29
Figure 5.4	Matches are drawn and then name of Object is displayed	30
Figure 5.5	Result when no object is found	31

TABLE OF CONTENTS

Certificate i
Declaration ii
Acknowledgements iii
Abstract.....iv
List of Figures v

CHAPTER 1

INTRODUCTION 01-03

1.1 History of Digital Image Processing 01
1.2 Problem Specification 02
1.3 Methodology 03
1.4 Layout of the thesis 03

CHAPTER 2

LITERATURE SURVEY 04-17

2.1 Object Recognition 04
2.2 OpenCV and Computer Vision 05
2.3 OpenCV Structure and Content 09
2.4 SURF Algorithm 10
2.5 Methods in OpenCV for Object Recognition 14
2.5 Process Models Used 16

CHAPTER 3

SYSTEM DESIGN 18-23

3.1 UML Design 18
3.2 Data Flow Diagrams 19
3.3 UML Diagrams 22

CHAPTER 4

IMPLEMENTATION 24-26

4.1 Preparing the Dataset 24
4.2 Image Acquisition 25
4.3 Comparing and Matching of Keypoints and Descriptors 25

CHAPTER 5
RESULTS AND DISCUSSIONS 27-31

CHAPTER 6
CONCLUSION AND FUTURE WORK 32

REFERENCES 33

APPENDIX 34-43

CHAPTER 1
INTRODUCTION

Digital image processing is the use of computer algorithms to perform image processing on digital images. As a subcategory or field of digital signal processing, digital image processing has many advantages over analog image processing. It allows a much wider range of algorithms to be applied to the input data and can avoid problems such as the build-up of noise and signal distortion during processing. Since images are defined over two or more dimensions digital image processing can be modelled in the form of multidimensional systems. According to Digital Image Processing (3rd Edition) by Rafael C. Gonzalez and Richard E. Woods, the definition of Digital Image Processing as processing of digital images by means of a digital computer [3]. The fundamental steps of Digital Image Processing are - Image acquisition, Image Enhancement, Image Restoration, Color Image Processing, Wavelets and Multiresolution Processing, Compression, Morphological Processing, Segmentation, Representation and Description, Object recognition and Knowledge Base [3].

Image processing is a method to perform some operations on an image, in order to get an enhanced image or to extract some useful information from it. It is a type of signal processing in which input is an image and output may be image or characteristics/features associated with that image. Currently, image processing is one of the rapidly growing technologies. It forms core research area within engineering and computer science disciplines too. Image processing basically includes the following three steps.

- Importing the image via image acquisition tools.
- Analyzing and manipulating the image.
- Output in which result can be altered image or report that is based on image analysis.

1.1 History of Digital Image Processing

Many techniques of digital image processing, or digital picture processing as it often was called, were developed in the 1960s at the Jet Propulsion Laboratory, Massachusetts Institute of Technology, Bell Laboratories, University of Maryland, and a few other research facilities, with

an application to satellite imagery, wire-photo standards conversion, medical imaging, videophone, character recognition, and photograph enhancement. The cost of processing was fairly high, however, with the computing equipment of that era. That changed in the 1970s, when digital image processing proliferated as cheaper computers and dedicated hardware became available. Images then could be processed in real time, for some dedicated problems such as television standards conversion. With the fast computers and signal processors available in the 2000s, digital image processing has become the most common form of image processing and generally it is used, because it is not only the most versatile method, but also the economical.

There are basically two types of methods used for image processing namely, analog and digital image processing. Analog image processing can be used for the hard copies like printouts and photographs. Image analysts use various fundamentals of interpretation while using these visual techniques. Digital image processing techniques help in manipulation of the digital images by using computers. The three general phases that all types of data have to undergo while using digital technique are pre-processing, enhancement and display, information extraction.

1.2 Problem Specification

“To find and detect the objects that are in an image by using SURF algorithm.”

Object recognition, as the name implies, is the process of identifying and determining a specified object in a digital image or in a video. This is the central research topic in Computer Vision. Computer Vision is a field that includes methods for acquiring, processing, analyzing, and understanding images from the real world in order to produce numerical or symbolic information. This project is an attempt to bring about a change in how a computer sees the world, sees the objects in it. OpenCV libraries have been used, in which SURF (Speeded Up Robust Features) algorithm is one of the frameworks in it used for extract the keypoints and descriptors. OpenCV is the acronym of Open source Computer Vision. It has a huge collection of programming functions and libraries aimed to improve the vision of the computer.

1.3 Methodology

The primary intuition behind using OpenCV libraries is that it is open source and has a huge collection of programming functions and libraries aimed to improve the vision of the computer. The approach is to take the input image from the user, find out the attributes from the image and later try to find the best match between these attributes and with the attributes of the images in the dataset, and display the results once the best match is found. The attributes here are keypoints and descriptors. Local features and their descriptors are the building blocks of many computer vision algorithms. Keypoints are the same thing as interest points. They are spatial locations, or points in the image that define what is interesting or what stand out in the image. They are scale invariant i.e. they do not change even when an image is rotated or the size of the image is altered. Descriptors encode interesting information into a series of numbers and act as a sort of numerical "fingerprint" that can be used to differentiate one feature from another.

1.4 Layout of Thesis

The thesis is organized as follows:

1. Chapter 1 deals with the Introduction to the project, Problem Specification, and Methodology.
2. Chapter 2 describes literature survey that has been done in order to do this work.
3. Chapter 3 describes System design and architecture of the model which also includes proposed methodologies, UML diagrams of proposed system design.
4. Chapter 4 is all about how the system is implemented and methods of implementation.
5. Chapter 5 shows the screen shots of the system execution and result analysis.
6. Chapter 6 is about conclusion and future work of this project and is followed by the necessary References and Appendix which useful to understand the concepts.

CHAPTER 2
LITERATURE SURVEY

2.1 Object Recognition

In the field of Computer Vision, object recognition describes the task of finding and identifying objects in an image or video sequence. Humans recognize a multitude of objects in images with little effort, despite the fact that the image of the objects may vary somewhat in different viewpoints, in many different sizes and scales or even when they are translated or rotated. Objects can even be recognized when they are partially obstructed from view. This task is still a challenge for computer vision systems. Many approaches to the task have been implemented over multiple decades. Object recognition is a process for identifying a specific object in a digital image or video. Object recognition algorithms rely on matching, learning, or pattern recognition algorithms using appearance-based or feature-based techniques. Common techniques include edges, gradients, Histogram of Oriented Gradients (HOG), Haar wavelets (Haar functions are the simplest wavelets which are used in many methods of discrete image transforms and processing), and linear binary patterns. Object recognition is useful in applications such as video stabilization, automated vehicle parking systems, and cell counting in bio imaging. Object recognition determines what objects are, where in a digital image and is a central research topic in computer vision. But a person looking at an image will spontaneously make a higher level judgment about the scene as a whole. For example, whether it's a kitchen, or a campsite, or a conference room. Among computer science researchers, the problem known as "scene recognition" has received relatively little attention. Like all machine-learning systems, neural networks try to identify features of training data that correlate with annotations performed by human beings. But unlike the machine-learning systems that produced, say, the voice-recognition software common in today's cellphones, neural nets make no prior assumptions about what those features will look like.

Computers are unbeatable at processing speeds and finding the solution for many complex problems, multi-tasking various processes, storing petabytes of data every hour and doing many wide variety of things. But a computer cannot think. It cannot see the world in the way we see it. A computer cannot understand the environment it is in, unless directed and specified by humans.

On an average, a human can easily distinguish between more than 30,000 visual categories, and can detect objects in the span of a few hundred milliseconds. There are various applications of object recognition. To name a few are Face detection, People Counting, Vehicle detection, Manufacturing Industry, Online images, Security. Detection of our face when we upload a photo in Facebook is one of the simple applications of object detection that we see in our daily life. Object detection can also be used for people counting, it is used for analyzing store performance or crowd statistics during festivals. Similarly when the object is a vehicle, such as a bicycle or car, object detection with tracking can prove effective in estimating the speed of the object. The type of ship entering a port can be determined by object detection (depending on shape, size etc.). This system for detecting ships are currently in development in some European countries. Object detection is also used in industrial processes to identify products. Say you want your machine to only detect circular objects. Hough circle detection transform can be used for detection. Apart from these, object detection can also be used for classifying images found online. Obscene images are usually filtered out using object detection. In future we might be able to use object detection in identifying anomalies in a scene (image frame) such as bombs or explosives. Object recognition also has applications like Android Eyes, Image panoramas, Image watermarking, Global robot localization, Face detection, Optical Character Recognition, Manufacturing Quality Control, Content-Based Image Indexing, Object Counting and Monitoring, Automated vehicle parking systems, Visual Positioning and tracking, Video Stabilization, Pedestrian detection and Face Detection.

2.2 OpenCV and Computer Vision

OpenCV (Open Source Computer Vision) is an open source computer vision and machine learning software library. OpenCV was built to provide a common infrastructure for computer vision applications and to accelerate the use of machine perception in the commercial products. Being a BSD-licensed product, OpenCV makes it easy for businesses to utilize and modify the code.

The library has more than 2500 optimized algorithms, which includes a comprehensive set of both classic and state-of-the-art computer vision and machine learning algorithms. These

algorithms can be used to detect and recognize faces, identify objects, classify human actions in videos, track camera movements, track moving objects, extract 3D models of objects, produce 3D point clouds from stereo cameras, stitch images together to produce a high resolution image of an entire scene, find similar images from an image database, remove red eyes from images taken using flash, follow eye movements, recognize scenery and establish markers to overlay it with augmented reality, etc.

OpenCV has C++, C, Python, Java and MATLAB interfaces and supports Windows, Linux, Android and Mac OS. OpenCV is written natively in C++ and has a template interface that works seamlessly with STL containers. OpenCV was designed to be cross-platform. So, the library was written in C and this makes OpenCV portable to almost any commercial system, from PowerPC Macs to robotic dogs. Since version 2.0, OpenCV includes its traditional C interface as well as the new C++. For the most part, new OpenCV algorithms are now developed in C++. Also wrappers for languages such as Python and Java have been developed to encourage adoption by a wider audience. OpenCV runs on both desktop (Windows, Linux, Android, MacOS, FreeBSD, OpenBSD) and mobile (Android, Maemo, iOS). OpenCV has a modular structure. The main modules of OpenCV are – core, highgui, imgproc, video, objdetect. OpenCV is now extensively used for developing advanced image processing and computer vision applications. It has been a tool for students, engineers and researchers in every nook and corner of the world. Optimized for real time image processing & computer vision applications Primary interface of OpenCV is in C++. There are also C, Python and JAVA full interfaces OpenCV applications run on Windows, Android, Linux, Mac and iOS Optimized for Intel processors.

Most computer scientists and practical programmers are aware of some facet of the role that computer vision plays. But few people are aware of all the ways in which computer vision is used. For example, most people are somewhat aware of its use in surveillance, and many also know that it is increasingly being used for images and video on the Web. A few have seen some use of computer vision in game interfaces. Yet few people realize that most aerial and street-map images (such as in Google's Street View) make heavy use of camera calibration and image stitching techniques. Some are aware of niche applications in safety monitoring, unmanned flying vehicles, or biomedical analysis. But few are aware how pervasive machine vision has

become in manufacturing. Virtually everything that is mass-produced has been automatically inspected at some point using computer vision [5].

The open source license for OpenCV has been structured such that one can build a commercial product using all or part of OpenCV. There is no obligation to open source the product or to return improvements to the public domain. In part because of these liberal licensing terms, there is a large user community that includes people from major companies (IBM, Microsoft, Intel, SONY, Siemens, and Google, to name only a few) and research centers (such as Stanford, MIT, CMU, Cambridge, and INRIA). OpenCV is popular around the world, with large user communities in China, Japan, Russia, Europe, and Israel.

Computer vision is the transformation of data from a still or video camera into either a decision or a new representation. All such transformations are done for achieving some particular goal. The input data may include some contextual information such as “the camera is mounted in a car” or “laser range finder indicates an object is 1 meter away”.

The decision might be “there is a person in this scene” or “there are 14 tumor cells on this slide”. A new representation might mean turning a color image into a grayscale image or removing camera motion from an image sequence. Because humans are such visual creatures, it is easy to be fooled into thinking that computer vision tasks are easy. The human brain divides the vision signal into many channels that stream different kinds of information into human brain. Human brain has an attention system that identifies, in a task-dependent way, important parts of an image to examine while suppressing examination of other areas. There is massive feedback in the visual stream that is, as yet, little understood.

There are widespread associative inputs from muscle control sensors and all of the other senses that allow the brain to draw on cross-associations made from years of living in the world. The feedback loops in the brain go back to all stages of processing including the hardware sensors themselves (the eyes), which mechanically control lighting via the iris and tune the reception on the surface of the retina.

In a machine vision system, however, a computer receives a grid of numbers from the camera or from disk. For the most part, there's no built-in pattern recognition, no automatic control of focus and aperture, no cross-associations with years of experience. For the most part, vision systems are still fairly naïve. What the computer "sees" is just a grid of numbers. Any given number within that grid has a rather large noise component and so by itself gives us little information, but this grid of numbers is all the computer "sees". Our task then becomes to turn this noisy grid of numbers into the perception: "side mirror". Given a two-dimensional (2D) view of a 3D world, there is no unique way to reconstruct the 3D signal. Formally, such an ill-posed problem has no unique or definitive solution. The same 2D image could represent any of an infinite combination of 3D scenes, even if the data were perfect. However, as already mentioned, the data is corrupted by noise and distortions. Such corruption stems from variations in the world (weather, lighting, reflections, movements), imperfections in the lens and mechanical setup, finite integration time on the sensor (motion blur), electrical noise in the sensor or other electronics, and compression artifacts after image capture [5].

OpenCV grew out of an Intel Research initiative to advance CPU-intensive applications. Intel launched many projects including real-time ray tracing and 3D display walls. One of the authors working for Intel at that time was visiting universities and noticed that some top university groups, such as the MIT Media Lab, had well developed and internally open computer vision infrastructures—code that was passed from student to student and that gave each new student a valuable head start in developing his or her own vision application. Instead of reinventing the basic functions from scratch, a new student could begin by building on top of what came before. Thus, OpenCV was conceived as a way to make computer vision infrastructure universally available. With the aid of Intel's Performance Library Team, OpenCV started with a core of implemented code and algorithmic specifications being sent to members of Intel's Russian library team. This is where it started in Intel's research lab with collaboration from the Software Performance Libraries group together with implementation and optimization expertise in Russia.

2.3 OpenCV Structure and Content

OpenCV is broadly structured into five main components, four of which are shown in Figure 2.1.

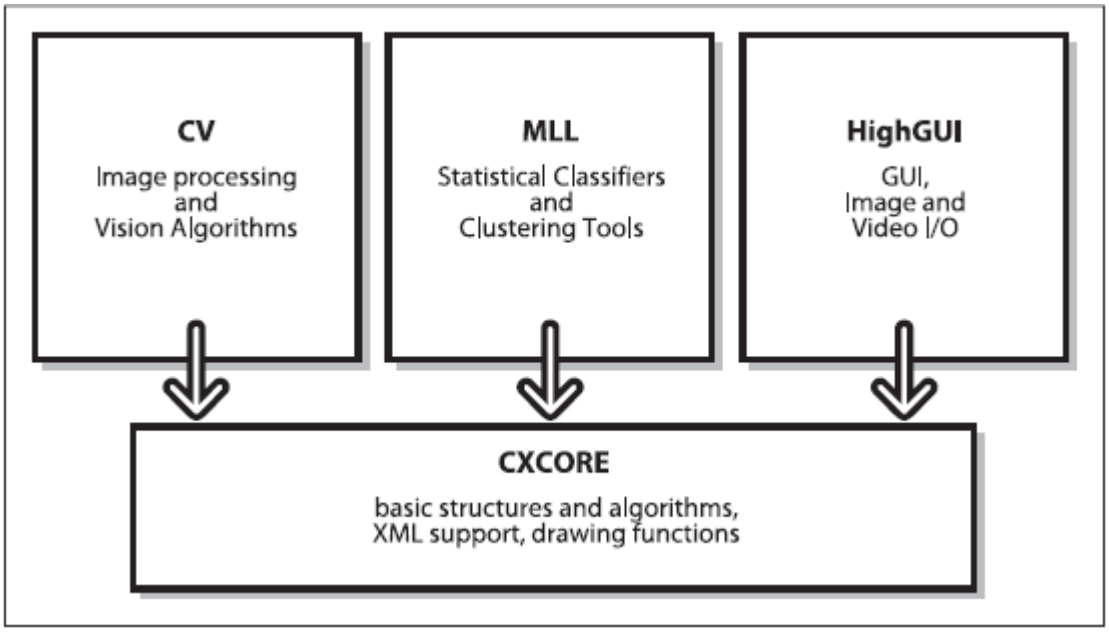

Fig 2.1 The Basic Structure of OpenCV

Figure 2.1 shows the basic structure of OpenCV, consisting of five components. The CV component contains the basic image processing and higher-level computer vision algorithms. MLL is the Machine Learning Library, which includes many statistical classifiers and clustering tools. HighGUI contains I/O routines and functions for storing and loading video and images, and CXCore contains the basic data structures and content. The fifth component CvAux, is not included, which contains both defunct areas (embedded HMM face recognition) and experimental algorithms (background/foreground segmentation) [5].

CvAux covers:

- Eigen objects, a computationally efficient recognition technique that is, in essence, a template matching procedure
- 1D and 2D hidden Markov models, a statistical recognition technique solved by dynamic programming
- Embedded HMMs (the observations of a parent HMM are themselves HMMs)
- Gesture recognition from stereo vision support

- Extensions to Delaunay triangulation, sequences, and so forth
- Stereo vision
- Shape matching with region contours
- Texture descriptors
- Eye and mouth tracking
- 3D tracking
- Finding skeletons (central lines) of objects in a scene
- Warping intermediate views between two camera views
- Background-foreground segmentation
- Video surveillance (see Wiki FAQ for more documentation)
- Camera calibration C++ classes (the C functions and engine are in CV)

2.4 SURF Algorithm

SURF stands for Speeded Up Robust Features. It is an algorithm which extracts some unique keypoints and descriptors from an image. Keypoints are the same thing as interest points. They are spatial locations, or points in the image that define what is interesting or what stand out in the image. The reason why keypoints are special is that, no matter how the image changes whether the image rotates, shrinks or is subject to distortion, the same keypoints are found in the image. What makes keypoints different between frameworks is the way these keypoints are described. These are known as descriptors. Each keypoint that is detected has an associated descriptor that accompanies it. Some frameworks only do a keypoint detection, while other frameworks are simply a description framework and they don't detect the points. And there are a few frameworks that both detect and describe the keypoints. For example, SIFT and SURF [2].

SURF uses an intermediate image representation called Integral Image. The integral image is computed rapidly from an input image and is used to speed up the calculation of any upright rectangular area. Given an input image *I* and a point *(x, y)* the integral image I_{Σ} is calculated by the sum of the values between the point and the origin.

Formally this can be defined by the formula:

$$I_{\sum}(x,y) = \sum_{i=0}^{i \leq x} \sum_{j=0}^{j \leq y} I(x,y)$$

By using the integral image, the task of calculating the area of an upright rectangular region is reduced. This makes computation time invariant to change in size this approach is particularly useful when large areas are required. SURF makes good use of this property to perform fast convolutions of varying size box filters at near constant time.

The SURF detector is based on the determinant of the Hessian matrix. A Hessian matrix is a matrix of second order partial derivatives. Consider a continuous function of two variables $f(x, y)$. Then, the Hessian matrix H, of the function f is given by –

$$H(f(x,y)) = \begin{bmatrix} \frac{\partial^2 f}{\partial x^2} & \frac{\partial^2 f}{\partial x \partial y} \\ \frac{\partial^2 f}{\partial x \partial y} & \frac{\partial^2 f}{\partial y^2} \end{bmatrix}$$

The determinant of this matrix is known as the discriminant and is calculated by:

$$det(H) = \frac{\partial^2 f}{\partial x^2} \frac{\partial^2 f}{\partial y^2} - \left(\frac{\partial^2 f}{\partial x \partial y} \right)^2$$

The value of the discriminant is used to classify the maxima and minima of the function by the second order derivative test. Since the determinant is the product of eigenvalues of the Hessian we can classify the points based on the sign of the result. If the determinant is negative, then the eigenvalues have different signs and hence the point is not a local extremum. If it is positive, then either both eigenvalues are positive or both are negative and in either case the point is classified as an extremum [2].

Translating this theory to work with images rather than a continuous function is a fairly trivial task. First we replace the function values $f(x, y)$ by the image pixel intensities $I(x, y)$. Next we require a method to calculate the second order partial derivatives of the image. We can calculate the derivatives by convolution. The second order scale normalized Gaussian is the chosen filter

as it allows for analysis over scales as well as space. Kernels can be constructed for the Gaussian derivatives in x, y and combined xy direction such that we calculate the four entries of the Hessian matrix. Use of the Gaussian allows us to vary the amount of smoothing during the convolution stage so that the determinant is calculated at different scales. Furthermore, since the Gaussian is an isotropic (i.e. circularly symmetric) function, convolution with the kernel allows for rotation invariance. The Hessian matrix, H, can now be calculated as function of both space x = (x, y) and scale σ.

$$H(\mathbf{x}, \sigma) = \begin{bmatrix} L_{xx}(\mathbf{x}, \sigma) & L_{xy}(\mathbf{x}, \sigma) \\ L_{xy}(\mathbf{x}, \sigma) & L_{yy}(\mathbf{x}, \sigma) \end{bmatrix}$$

Here $L_{xx}(X, \sigma)$ refers to the convolution of the second order Gaussian derivative $\frac{\partial^2 g(\sigma)}{\partial x^2}$ with the image at point X = (x, y) and similarly for L_{yy} and L_{xy}. These derivatives are known as Laplacian of Gaussians. Working from this the determinant of the Hessian can be calculated for each pixel in the image and the value can be used to find interest points.

The SURF descriptor describes how the pixel intensities are distributed within a scale dependent neighborhood of each interest point detected by the Fast-Hessian. This approach is similar to that of SIFT [4] but integral images used in conjunction with filters known as Haar wavelets are used in order to increase robustness and decrease computation time. Haar wavelets are simple filters which can be used to find gradients in the x and y directions.

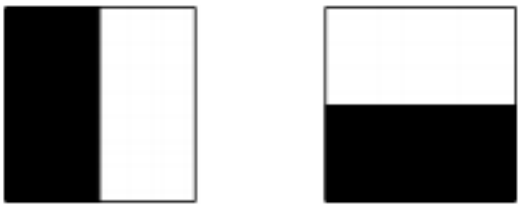

Fig 2.2 Haar Wavelets

In the Figure 2.2, the left filter computes the response in the x-direction and the right the y-direction. Weights are 1 for black regions and -1 for the white. When used with integral images each wavelet requires just six operations to compute.

Extraction of the descriptor can be divided into two distinct tasks. First each interest point is assigned a reproducible orientation before a scale dependent window is constructed in which a 64-dimensional vector is extracted. It is important that all calculations for the descriptor are based on measurements relative to the detected scale in order to achieve scale invariant results. The procedure for extracting the descriptor is explained further in the following.

In order to achieve invariance to image rotation each detected interest point is assigned a reproducible orientation. Extraction of the descriptor components is performed relative to this direction so it is important that this direction is found to be repeatable under varying conditions. To determine the orientation, Haar wavelet responses of size 4σ are calculated for a set pixels within a radius of 6σ of the detected point, where σ refers to the scale at which the point was detected. The specific set of pixels is determined by sampling those from within the circle using a step size of σ.

The responses are weighted with a Gaussian centered at the interest point. In keeping with the rest the Gaussian is dependent on the scale of the point and chosen to have standard deviation 2.5σ. Once weighted the responses are represented as points in vector space, with the x-responses along the abscissa and the y-responses along the ordinate. The dominant orientation is selected by rotating a circle segment covering an angle of $\pi/3$ around the origin. At each position, the x and y-responses within the segment are summed and used to form a new vector. The longest vector lends its orientation the interest point [2].

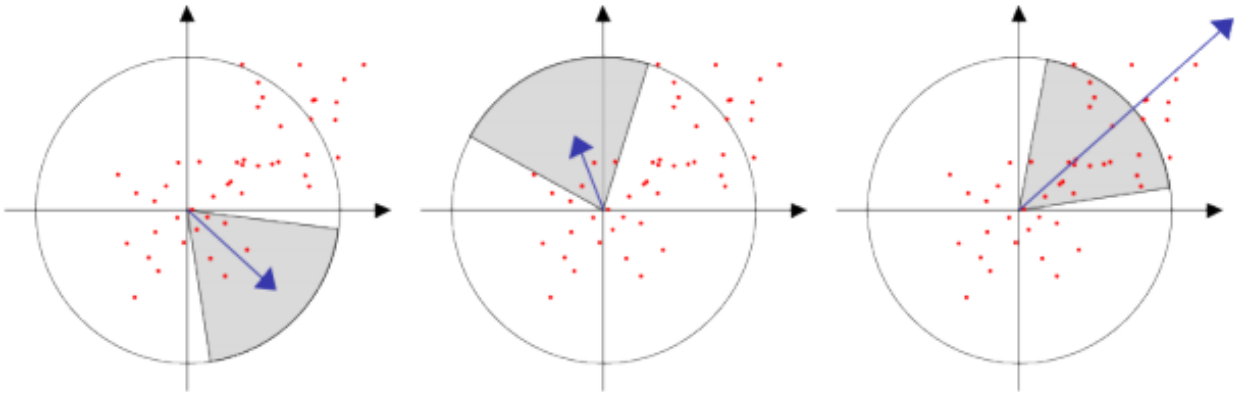

Fig 2.3 Orientation Assignment

Figure 2.3 shows the window sliding around the origin the components of the responses are summed to yield the vectors shown here in blue. The largest such vector determines the dominant orientation.

2.5 Methods in OpenCV for Object Recognition

OpenCV was built to provide a common infrastructure for computer vision applications and to accelerate the use of machine perception in the commercial products. The library has more than 2500 optimized algorithms, which includes a comprehensive set of both classic and state-of- the-art computer vision and machine learning algorithms. These algorithms can be used to detect and recognize faces, identify objects, classify human actions in videos, track camera movements, track moving objects, etc. The basic steps involved in my project of object recognition are – Image acquisition, Extraction of attributes from the acquired image, Finding the matches between the attributes of the query image with that of the images from the dataset, and displaying the result if enough matches are found. OpenCV API Reference has many methods for object recognition. Object detection and tracking using color, face detection using OpenCV, SURF in OpenCV, Features2D + Homography to find a known object, Back Projection, Cascade Classifier Training, etc. It even provides methodologies to detect and track objects using mobile devices, supporting two of the leading mobile operating systems, Android and iOS [6].

The first step is the acquisition of image from the user, which is done by using Mat. Mat is in general a class with two data parts: the matrix header (containing information such as the size of the matrix, the method used for storing, at which address is the matrix stored, and so on) and a pointer to the matrix containing the pixel values (taking any dimensionality depending on the method chosen for storing). The matrix header size is constant, however the size of the matrix itself may vary from image to image and usually is larger by orders of magnitude.

The second step is the extraction of attributes. Attributes here are keypoints and descriptors. Keypoints are the same thing as interest points. They are spatial locations, or points in the image that define what is interesting or what stand out in the image. The reason why keypoints are special is because no matter how the image changes, whether the image rotates, shrinks or is subject to distortion, we will be able to find the same keypoints in the image. Keypoints only obtain information about the position, and sometimes their coverage area of the interest points. While the information about keypoint position might sometimes be useful, it does not say much about the keypoints themselves. So, here come descriptors. They are the way to compare the keypoints. They summarize, in vector format (of constant length) some characteristics about the keypoints. For example, it could be their intensity in the direction of their most pronounced orientation. It's assigning a numerical description to the area of the image the keypoint refers to. SurfFeatureDetector and SurfDescriptorExtractor are used for extracting keypoints and calculating descriptors respectively.

The third step is the matching of keypoints and descriptors of the query image with that of the images from the dataset. For this, the combination of BFMatcher with knnMatch is being used. Brute-Force matcher is simple. It takes the descriptor of one feature in first set and is matched with all other features in second set using some distance calculation. And the closest one is returned. In case of knnMatch, it finds the k best matches for each descriptor from a query set and returns the same.

2.5 Process Models Used

WATERFALL MODEL is used here for justification. The waterfall model is a sequential model. In this model, the software development activity is divided into different phases and each phase consists of series of tasks and has different objectives. It was the first model which was widely used in the software industry.

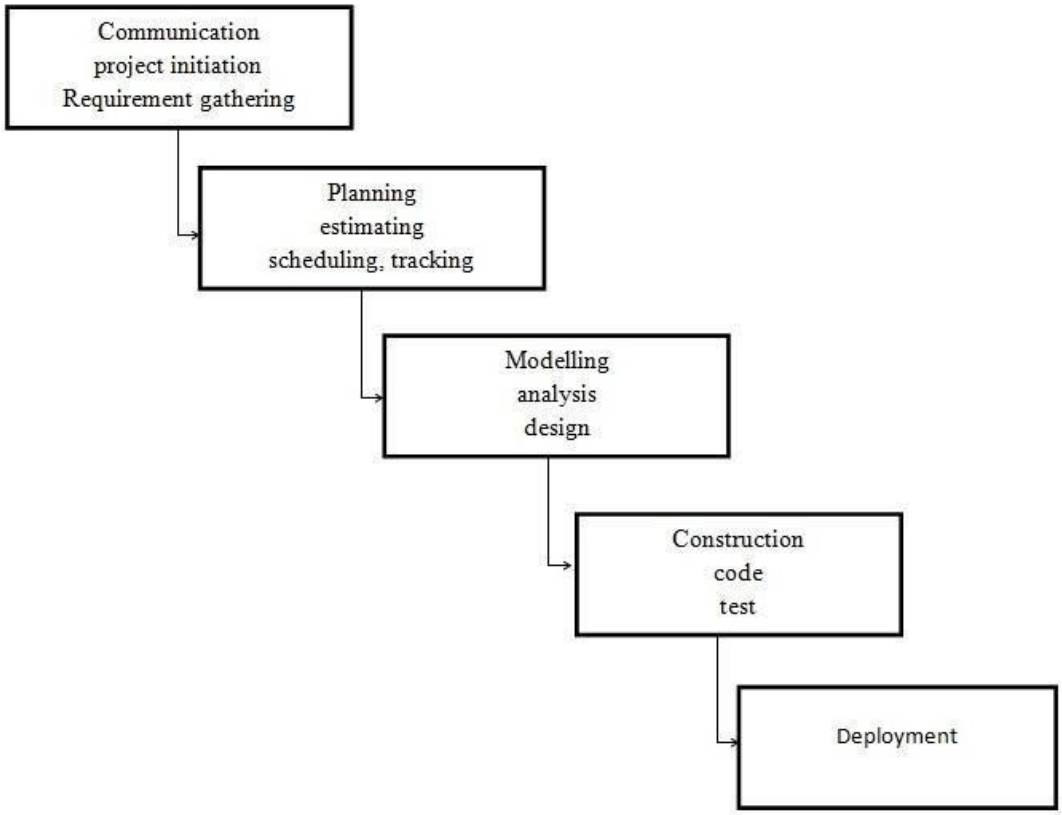

Fig 2.4 Waterfall Model

Figure 2.2 shows the stages of waterfall model incorporated in this project development. It is divided into phases and output of one phase becomes input of the next phase. It is mandatory for a phase to be completed before the next phase starts. There is no overlapping in Waterfall model. In waterfall, development of one phase starts only when the previous phase is complete. Because of this nature, each phase of waterfall model is quite precise well defined. The phases fall from higher level to lower level.

The sequential phases in Waterfall model are:

- Requirement Gathering and analysis: All possible requirements of the system to be developed are captured in this phase and documented in a requirement specification doc.
- System Design: The requirement specifications from first phase are studied in this phase and system design is prepared. System Design helps in specifying hardware and system requirements and also helps in defining overall system architecture.
- Implementation: With inputs from system design, the system is first developed in small programs called units, which are integrated in the next phase. Each unit is developed and tested for its functionality which is referred to as Unit Testing.
- Integration and Testing: All the units developed in the implementation phase are integrated into a system after testing of each unit. Post integration the entire system is tested for any faults and failures.
- Deployment of system: Once the functional and non-functional testing is done, the product is deployed in the customer environment or released into the market.

In addition to these there is a stage called maintenance. There are some issues which come up in the client environment. To fix those issues patches are released. Also to enhance the product some better versions are released. Maintenance is done to deliver these changes in the customer environment. All these phases are cascaded to each other in which progress is seen as flowing steadily downwards (like a waterfall) through the phases.

CHAPTER 3

SYSTEM DESIGN

3.1 UML Design

A good programming practice requires proper designing of processes involved in building a software model. Design is first concern with specification of a software architecture that defines major software components and their relationships. Design also involves reaching a balance between requirements that conflict with each other within implementation environment.

The UML is a graphical language for specifying, visualizing, documenting. All the three major stages of Object Oriented development such as OOA, OOD and OOP can make use of UML. There are 14 UML diagram types to help us to model this behavior. They can be divided into two main categories

- Structural diagrams
- Behavioral diagrams

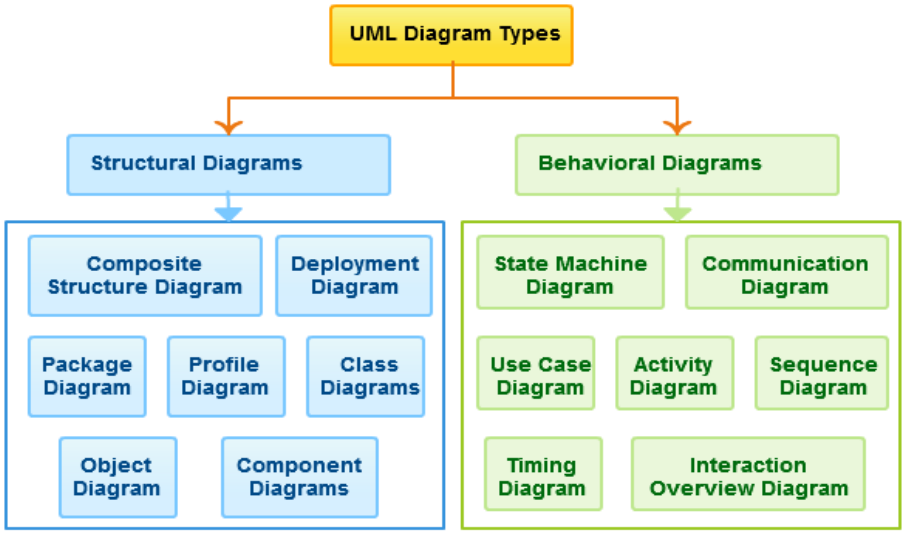

Fig 3.1 UML Diagrams

In the Figure 3.1 the two main types of UML diagrams are shown. The structural diagrams represent the static aspect of the system. These static aspects represent those parts of a diagram which forms the main structure and therefore stable. Behavioral diagrams basically capture the dynamic aspect of a system. Dynamic aspect can be further described as the changing/moving parts of a system.

Advantages of modeling:

- Makes easy to enhance, and manipulation of existing system.
- Helps to carry in visualizing the system to be developed.
- Permits to specify the structure and behavior of the system.
- Used as a template to construct a proposed system.
- For capturing the requirements.

3.2 Data Flow Diagrams

A data flow diagram (DFD) is a graphical representation of the "flow" of data through an information system, modelling its process aspects. A DFD is often used as a preliminary step to create an overview of the system, which can later be elaborated.

Fig 3.2 Level-0 DFD for the system

Figure 3.2 shows the Level-0 DFD of the system which is the basic level DFD at the highest level of abstraction which shows the flow of information through the entire system as a whole with input and output.

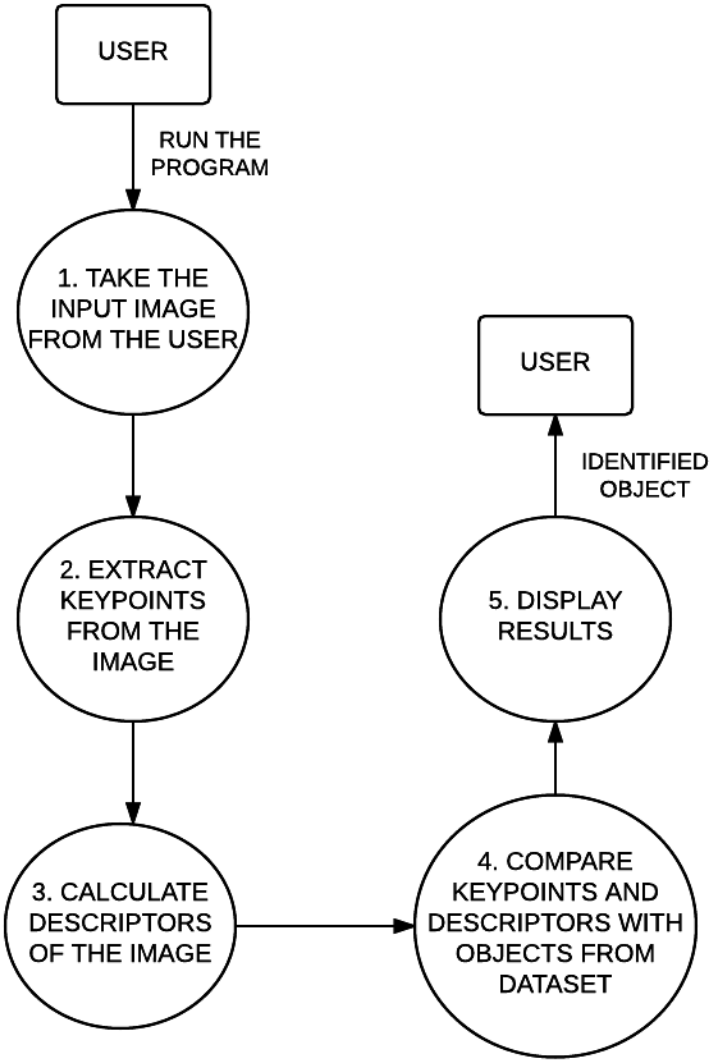

Fig 3.3 Level-1 DFD of the system

Figure 3.3 is the Level-1 DFD which shows the flow of data through different processes. Initially the input image is loaded from the user and later the keypoints are extracted and the descriptors are calculated. Next, the keypoints and descriptors calculated are compared with that of the object images from the dataset. Corresponding results are displayed to the user.

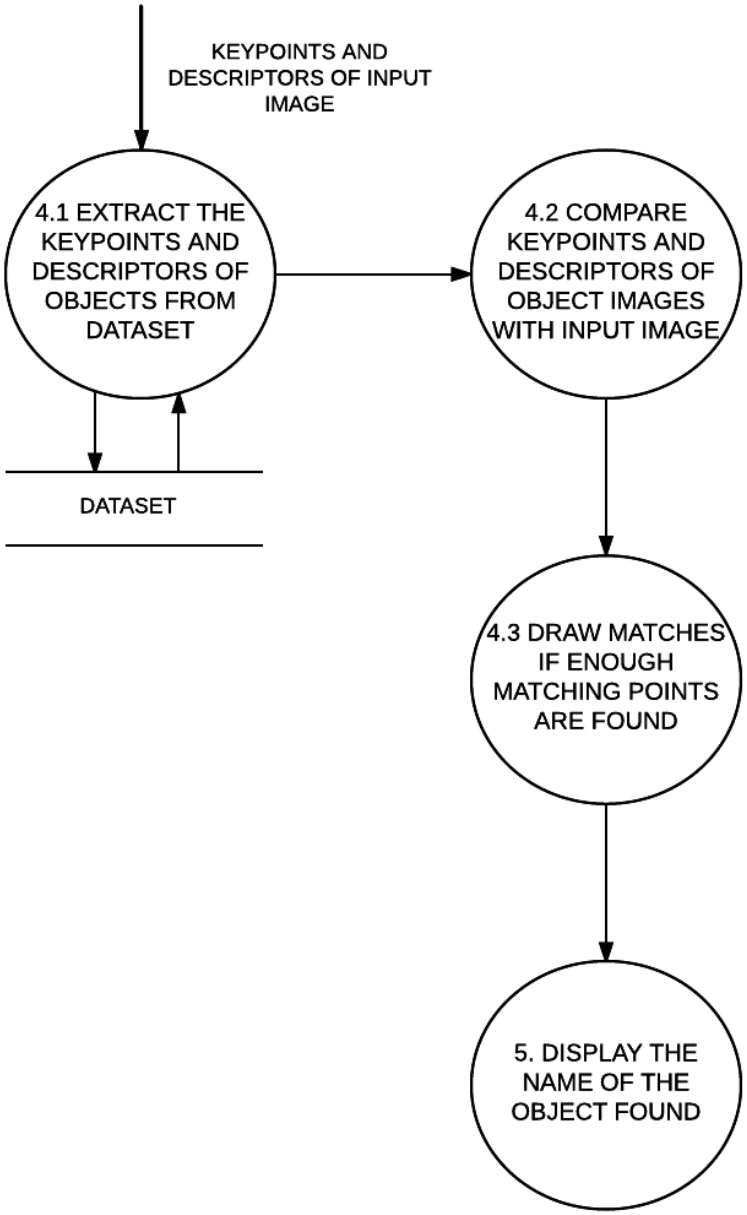

Fig 3.4 Level-2 DFD of the system

Figure 3.4 is a Level-2 DFD which shows the fourth process from Level-1 DFD expanded to a lower level of abstraction. The keypoints and descriptors are now extracted for the objects in the dataset present as .txt files. The keypoints and descriptors are compared and matches are drawn only if enough good matches are found and the results are displayed to the user.

3.3 UML Diagrams

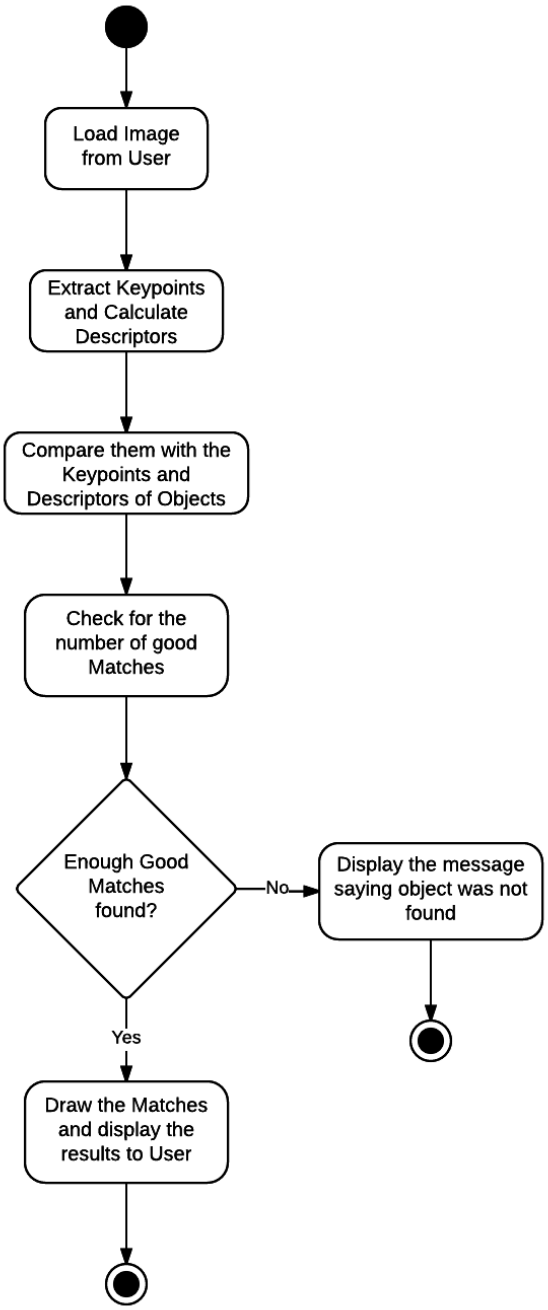

Fig 3.5 Activity Diagram of the system

Figure 3.5 shows the Activity diagram which depicts the flow of control in the system. After the image is loaded correctly, the keypoints and descriptors are calculated. Then they are compared with the keypoints and descriptors of the objects in the dataset. And if good number of matches are found then the name of object is displayed.

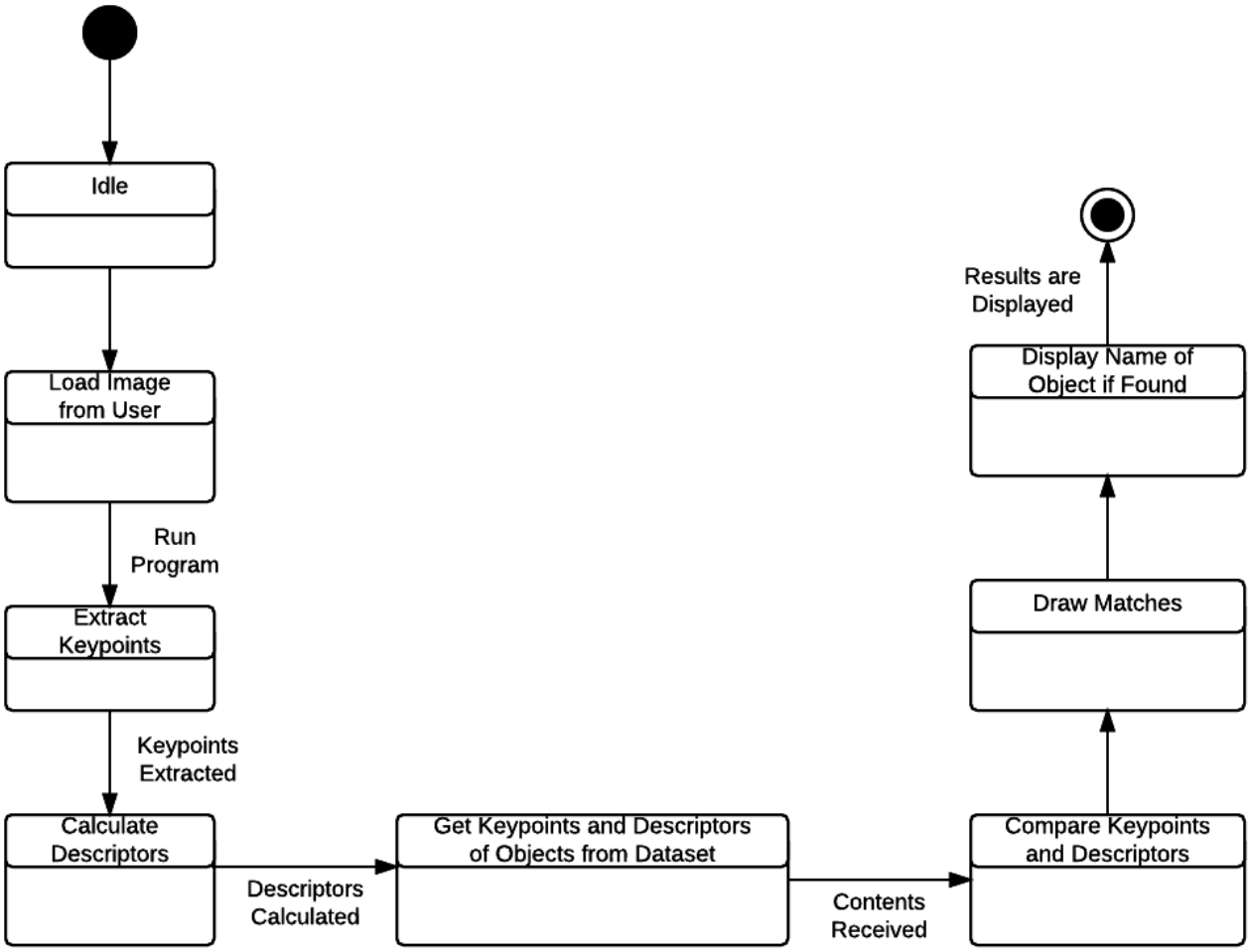

Fig 3.6 State chart Diagram of the system

Figure 3.6 shows the State Chart diagram of the system. State chart diagram is used to represent the event driven state change of a system. It basically describes the state change of a class, interface etc. Initially the system is in idle state. Then each event leads the system to go into different states such as loading image from user, extracting keypoints, calculating descriptors, etc. The system goes into exit state after the final results are displayed.

CHAPTER 4
IMPLEMENTATION

4.1 Preparing the Dataset

Prior to the execution, the dataset containing the objects must be prepared. The attribute of all the images in the dataset must be calculated only once before the execution. All the attributes i.e. the keypoints and descriptors are to be extracted and calculated. The results are stored in a file. The resulted file contains the matrix array values of all the keypoints and descriptors that are found using a feature description algorithm.

The algorithm used for finding the attributes is SURF algorithm. SURF is Speeded-Up Robust Features which was introduced by three people, Bay, H., Tuytelaars, T. and Van Gool, L [2]. Before SURF, there was SIFT for keypoint detection and description. But it was comparatively slow and people needed more speeded-up version. In SIFT [4], Lowe approximated Laplacian of Gaussian with Difference of Gaussian for finding scale-space. SURF goes a little further and approximates LoG() (Laplacian of Gaussian) with Box Filter. One big advantage of this approximation is that, convolution with box filter can be easily calculated with the help of integral images. And it can be done in parallel for different scales. Also the SURF rely on determinant of Hessian matrix for both scale and location. Adequate Gaussian weights are also applied to it. The dominant orientation is estimated by calculating the sum of all responses within a sliding orientation window of angle 60 degrees. For feature description, SURF uses Wavelet responses in horizontal and vertical direction.

After finding the keypoints and descriptors, these values are saved into a text file. For this purpose, FileStorage class is used. XML/YAML file storage class that encapsulates all the information necessary for writing or reading data to/from a file. There are various modes of operations in FileStorage. But I used the function FileStorage::WRITE for writing the contents into a file and the function FileStorage::READ for reading the contents from a file. So, FileStorage::WRITE is applied each time after finding the keypoints and descriptors for each

object, and the file is stored as a text file. Hence, the process of preparing the dataset is complete. This has to be done only once and not every time while executing.

4.2 Image Acquisition

The user has to give an input image while execution. This query image is acquired using Mat class. The class Mat represents an n-dimensional dense numerical single-channel or multi-channel array. It can be used to store real or complex-valued vectors and matrices, grayscale or color images, voxel volumes, vector fields, point clouds, tensors, histograms, etc. Mat is in general a class with two data parts. The matrix and a pointer to the matrix containing the pixel values. The matrix header size is constant, however the size of the matrix itself may vary from image to image and usually is larger by orders of magnitude. Once the array is created, it is automatically managed via a reference-counting mechanism. The array data is de-allocated when no one points to it. To release the data pointed by an array header before the array destructor is called, we can use Mat::release().

The function imread loads an image from the specified file and returns it. If the image cannot be read (because of missing file, improper permissions, unsupported or invalid format), the function returns an empty matrix (Mat::data==NULL). The file formats supported are - *.bmp, *.dib, *.jpeg, *.jpg, *.jpe, *.jp2, *.png, *.pbm, *.pgm, *.ppm .sr, *.ras, *.tiff, *.tif.

4.3 Comparing and Matching of Keypoints and Descriptors

After acquiring the query image from the user, the keypoints and descriptors are calculated using the SURF algorithm as discussed in section 4.1. The methods SurfFeatureDetector and SurfDescriptorExtractor are used to extract the keypoints and to calculate the descriptors respectively. Next, the above found keypoints and descriptors are compared with those of the objects in the dataset. The keypoints and descriptors of objects in dataset are already found and calculated as discussed in section 4.1. They are stored in .txt format. The class FileStorage is used to retrieve the information. Precisely, FileStorage::READ is used to get the information about the objects. Once this is done, the matching should begin. The matching of keypoints and descriptors is done using BFMatcher. BFMatcher is Brute-Force Matcher. It takes the descriptor

of one feature in first set and is matched with all other features in second set using some distance calculation, and the closest one is returned. This descriptor matcher supports masking permissible matches of descriptor sets. There are two important methods - BFMatcher.match() and BFMatcher.knnMatch(). The first one returns the best match. Second method returns k best matches where k is specified by the user. It may be useful when we need to do additional work on that.

Once enough matches are found, then the results are displayed. The original image given by the user and the best matched object from the dataset are displayed along with the matched matches among them. If enough matches are not found, then appropriate message is displayed on the console.

CHAPTER 5

RESULTS AND DISCUSSIONS

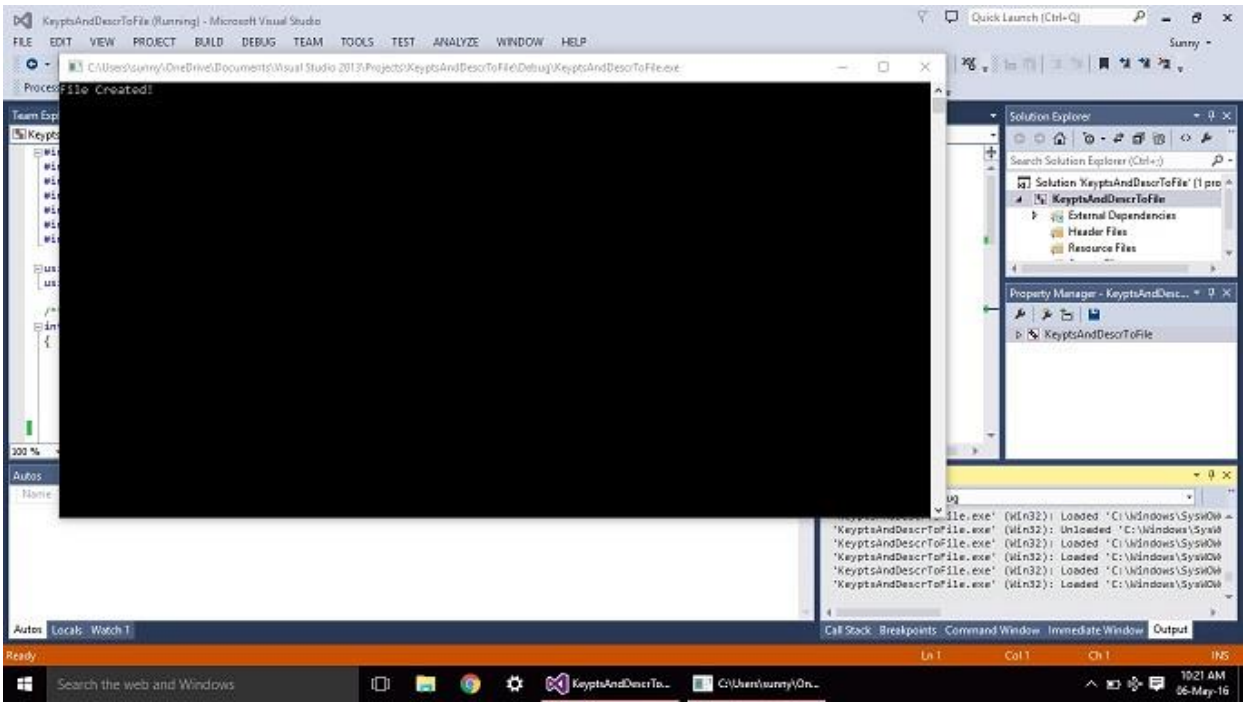

Fig 5.1 Keypoints and Descriptors extracted to a file

Figure 5.1 shows the confirmation message after executing the program successfully. The Keypoints are extracted and the Descriptors are calculated from an object image. The results are stored in a text file.

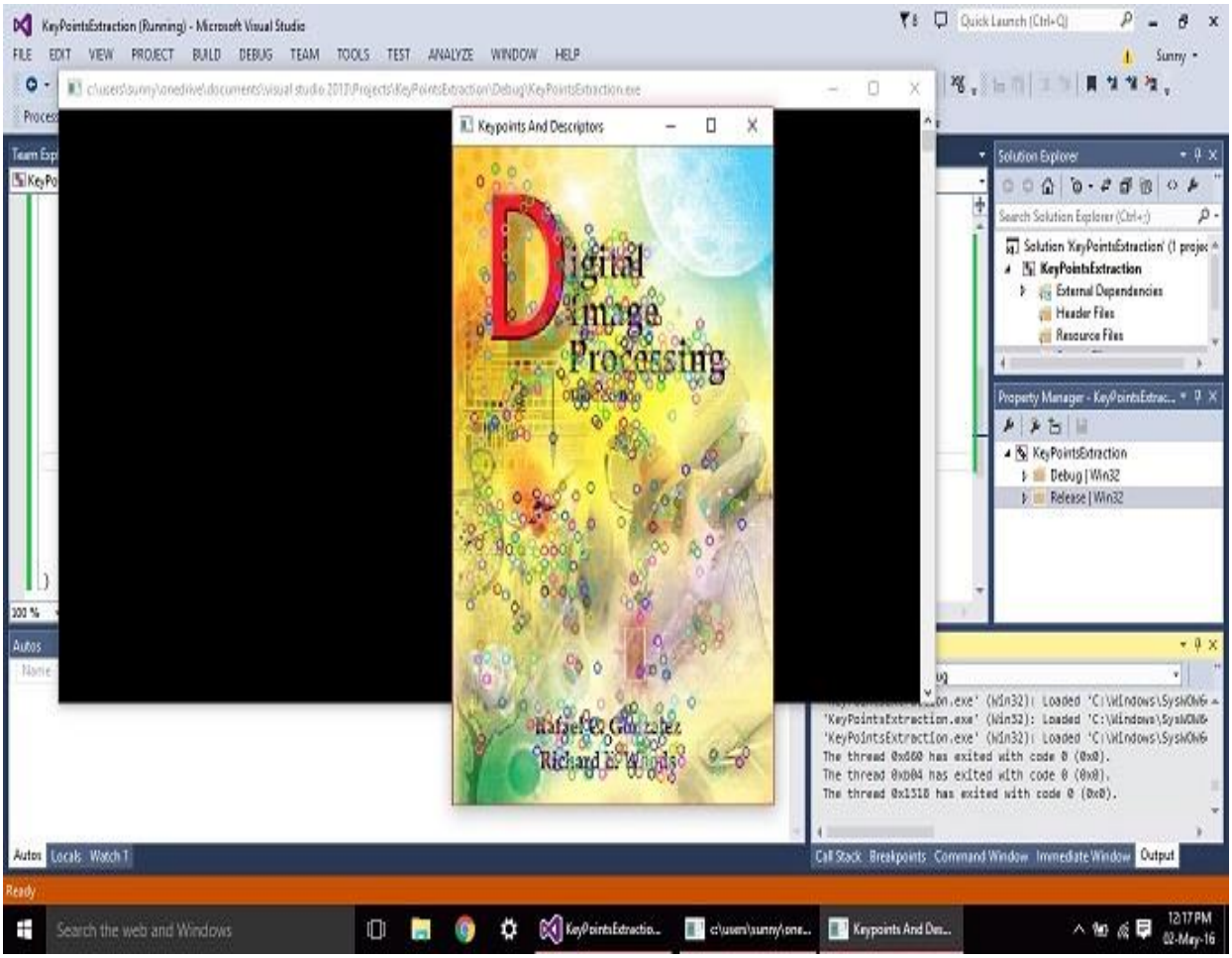

Fig 5.2 Keypoints and Descriptors of a sample image

Figure 5.2 shows all the extracted Keypoints and calculated Descriptors of a sample image. The colored circles seen in the image are actually the Keypoints and Descriptors.

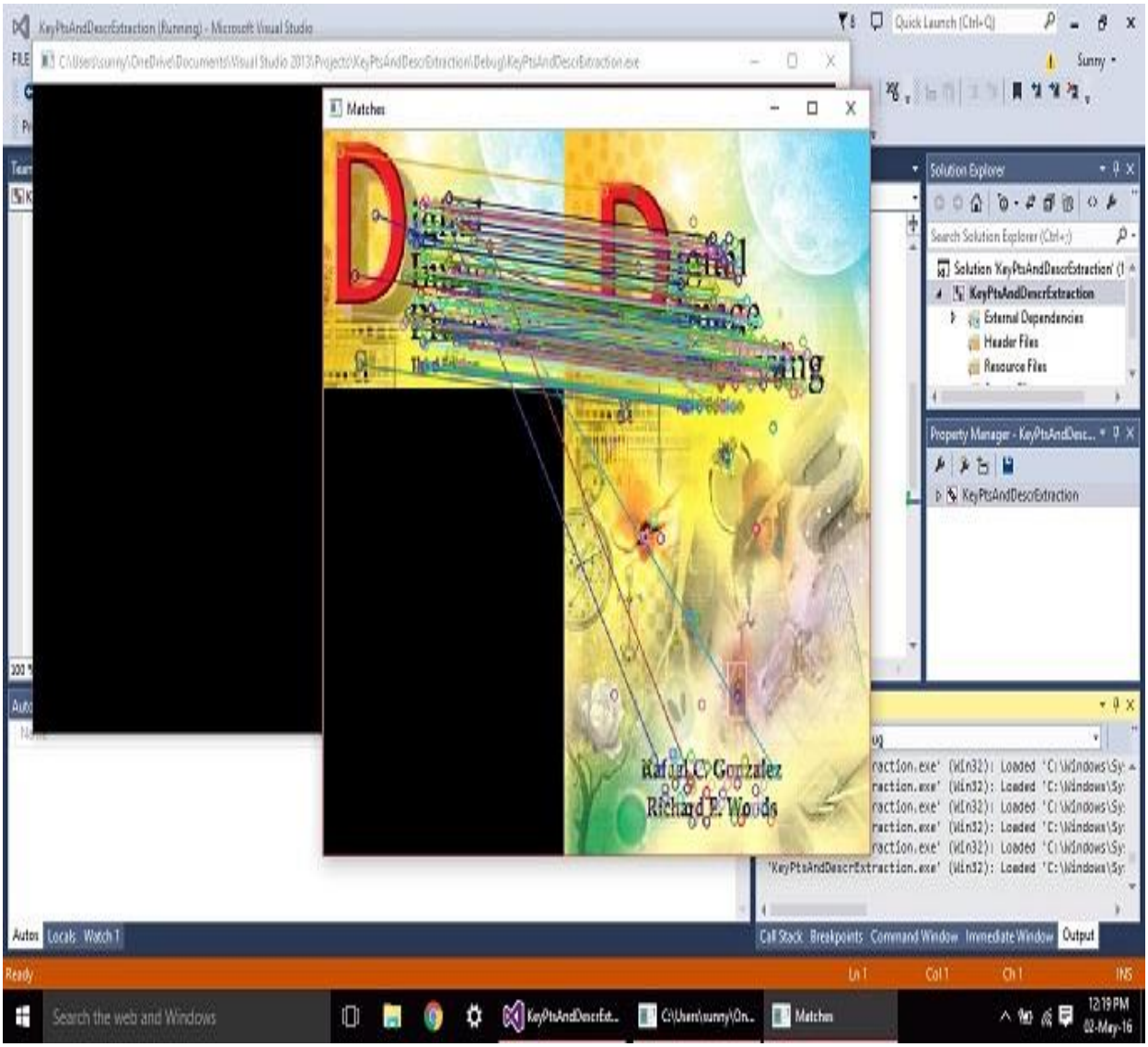

Fig 5.3 Matches between Keypoints and Descriptors of two images

Figure 5.3 shows the result of the matches found between two images. The Keypoints and Descriptors of both the images are first found and then later compared to find the matches. Lines are drawn from the keypoints and descriptors of the first image to the keypoints and descriptors of the second one.

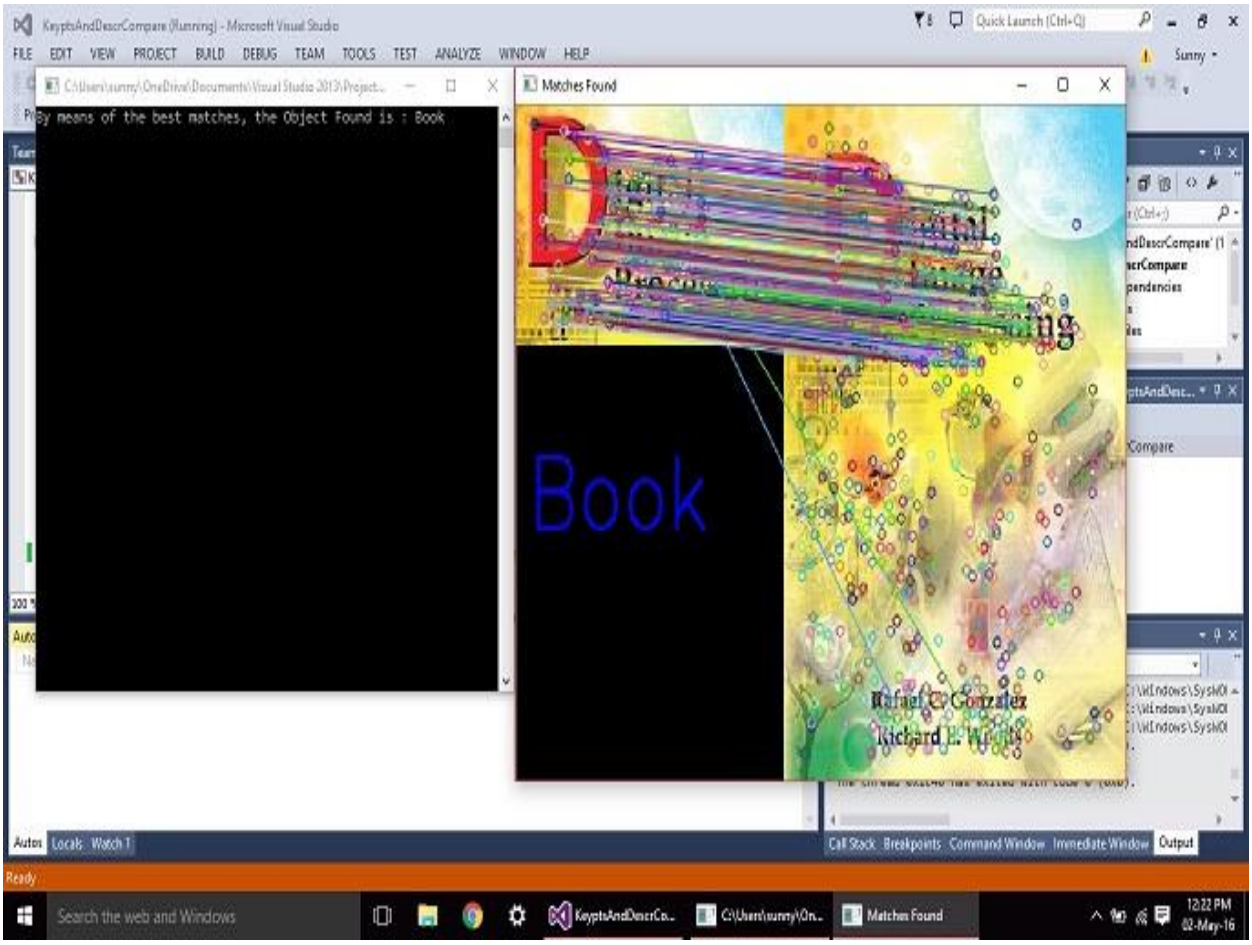

Fig 5.4 Matches are drawn and the name of Object is displayed

Figure 5.4 shows the final output which first finds out the keypoints and descriptors, then finds the number of good matches, and if enough good matches are found, then the lines are drawn and the name of the object is displayed, in the console as well as in the window “Matches Found”.

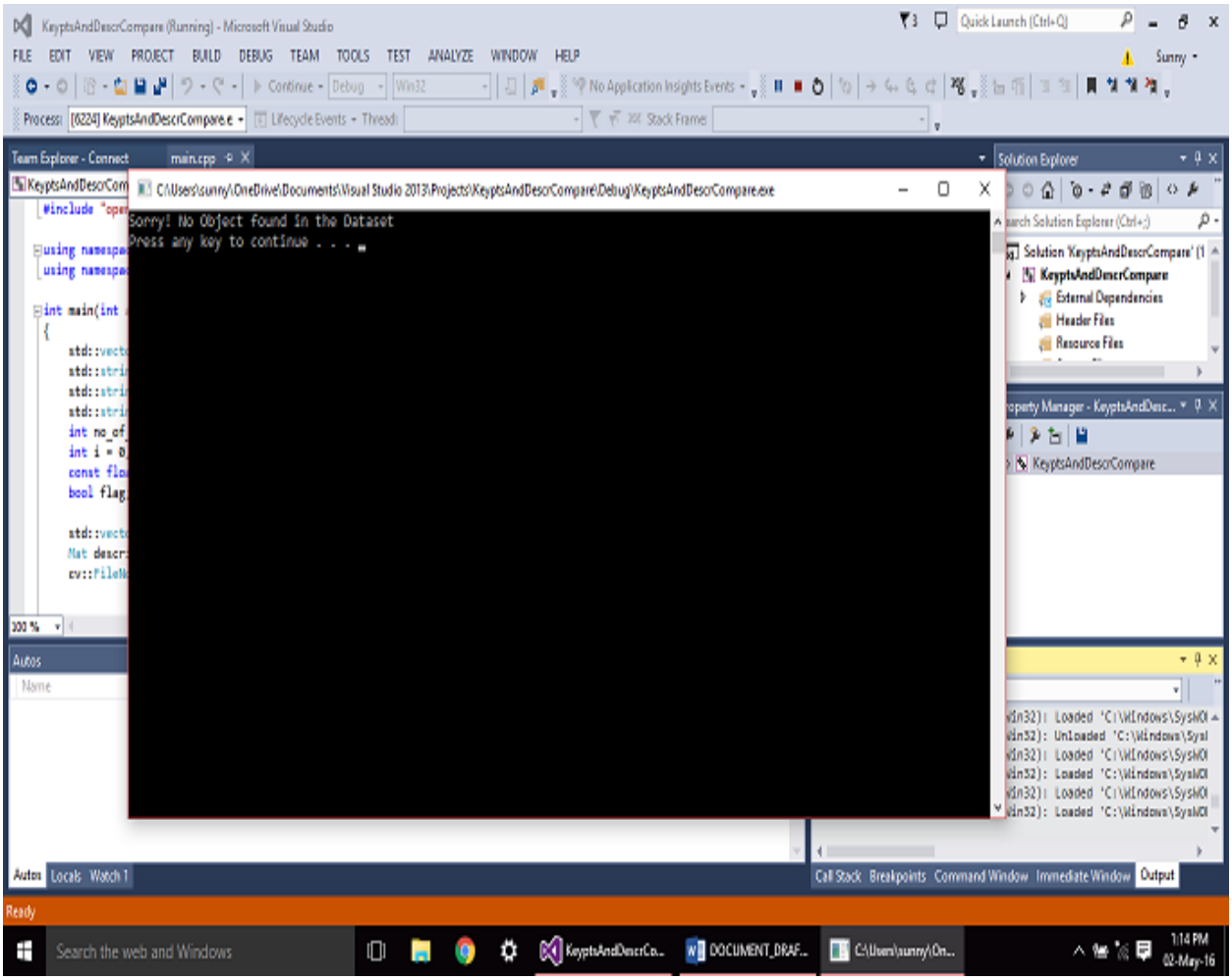

Fig 5.5 Result when no object is found

Figure 5.5 shows the output when there is no object found in the dataset, for the image given by the user. The failure message is shown in the console window.

CHAPTER 6
CONCLUSION AND FUTURE WORK

The project is used for detecting objects in an image. It has been implemented using VC++ (Visual C++) 2013 and OpenCV (Open source Computer Vision) 2.4.9. The technique used for recognizing objects is local and global features. For this, the SURF algorithm has been used. Feature points, i.e. keypoints and descriptors are obtained by using SURF algorithm. This algorithm proves accurate and robust to various scales, orientation, and various illumination conditions.

What objects are recognized depends on the dataset of the objects taken into consideration. Also, the user has to give the query image of the object he wants to detect as an input, which is not always feasible. As object recognition is the concept of identifying the object in an image or a video, it can be used for enhancing the features of specific applications. For example, certain image processing applications in the fields of machine vision, industrial vision, etc. can be enhanced by making use of object recognition concepts.

Accuracy of recognizing objects can be enhanced by combining the current approach with other fields of study like machine learning using neural networks. Other techniques which are based on color, shape, blob and template matching can also be combined with the current technique for improving the efficiency and performance of the project. For training based learning and Machine Learning, OpenCV has HaaR Feature based Cascade Classification for object recognition. Object detection and recognition can be a major milestone in the future, especially in industries to reduce the human work and increase the productivity and profits. It has a huge scope in software for self-driving cars, for Virtual Reality (an artificial environment created with computer hardware and software and presented to the user in such a way that it appears and feels like a real environment) and Augmented Reality (a type of virtual reality that aims to duplicate the world's environment in a computer), in wearable technology like Google Glasses, and many more.

REFERENCES

[1] Xuelong Hu, Yingcheng Tang, Zhenghua Zhang, "Video Object Matching Based on SIFT Algorithm", IEEE Int. Conference Neural Networks & Signal Processing Zhenjiang, China, June 2008.

[2] Herbert Bay, Tinne Tuytelaars, Luc Van Gool, "SURF: Speeded Up Robust Features", Katholieke Universiteit Leuven, September 2007.

[3] Rafael C. Gonzalez, Richard E. Woods, "Digital Image Processing", 3rd Edition, Pearson Publications, pp. 23-50, 2008.

[4] D. Lowe, "Scale-Invariant Feature Transform", University of British Columbia, 2004.

[5] Gary Bradski, Adrian Kaehler, "Learning OpenCV", O'Reilly Media, 1st edition, pp. 1-29, 2008.

[6] http://opencv.org/documentation.html

APPENDIX

SOURCE CODE

main.cpp in **KeyptsAndDescrCompare** Solution

This is the main program. In this program, the query image is taken from the user and the keypoints and descriptors are extracted and compared with that of object images. And if enough good matches are found then the name of the object is displayed.

```
#include <stdio.h>
#include <iostream>
#include <string>
#include "opencv2/core/core.hpp"
#include "opencv2/features2d/features2d.hpp"
#include "opencv2/highgui/highgui.hpp"
#include "opencv2/nonfree/features2d.hpp"

using namespace cv;
using namespace std;

int main(int argc, char** argv)

{

	std::vector<std::string> object = { "Landscape", "Book", "Rack" };
	std::string loc_of_img = { "C:\\Users\\sunny\\Desktop\\Project\\ObjectImages\\" };
	std::string loc_of_txt = { "C:\\Users\\sunny\\Desktop\\Project\\KeyptsAndDescriptors\\"
	};
	std::string location_img = "", cur_object = "", location_txt = "";
	int no_of_objects = object.size();
	int i = 0, j = 0, k = 0, size_of_matches = 0;
	const float ratio = 0.8f;
	bool flag;

	std::vector<KeyPoint> keypoints_object;
	Mat descriptors_object;
	cv::FileNode n1, n2;

	//Loading the query image from the user
	Mat query_image = imread("C:\\Users\\sunny\\Desktop\\Images\\book3.jpg", 1);
```

```
//Displays error message if there is a problem in loading the image
if (!query_image.data)
{
	printf("Error loading file! Please try again.\n");
	system("pause");
	return -1;
}

//Detecting the keypoints using SURF Detector
int minHessian = 400;
SurfFeatureDetector detector(minHessian);
std::vector<KeyPoint> keypoints_query;
detector.detect(query_image, keypoints_query);

//Calculating the descriptors
SurfDescriptorExtractor extractor;
Mat descriptors_query;
extractor.compute(query_image, keypoints_query, descriptors_query);


//Comparing the keypoints and descriptors of query image and each of the object images
to find the best possible match

for (i = 0 ; i < no_of_objects ; i++)
{
	//Considering one object at a time
	cur_object = object[i];
	//Taking the location of the object image
	location_img = loc_of_img + cur_object + ".jpg";

	Mat object_image = imread(location_img, 1);
	if (!object_image.data)
	{
		printf("Error! Please try again.");
		return -1;
	}

	//Getting the txt files containing information about keypoints and descriptors of
	objects
	location_txt = loc_of_txt + cur_object + ".txt";
	cv::FileStorage retrieve(location_txt, cv::FileStorage::READ);

	//Fetching the keypoints values into n1
	n1 = retrieve["keypoints"];
	//Storing the value of n1 in keypoints_object
```

```
cv::read(n1, keypoints_object);

//Fetching the descriptors values into n2
n2 = retrieve["descriptors"];
//Storing the value of n2 in descriptors_object
cv::read(n2, descriptors_object);

//Using Brute Force Algorithm for drawing the match lines
BFMatcher matcher(NORM_L2);
std::vector<vector< DMatch >> matches;

//knnMatch is used to find the k nearest neighbours
matcher.knnMatch(descriptors_query, descriptors_object, matches, 2);

vector<cv::DMatch> good_matches;
size_of_matches = matches.size();

for (j = 0 ; j < size_of_matches ; ++j)
{
        if (matches[j][0].distance < ratio * matches[j][1].distance)
        {
                //Updating the good_matches vector whenever the above condition
                is satisfied
good_matches.push_back(matches[j][0]);

        }
}


if (good_matches.size() > (0.35 * size_of_matches))
{
        flag = true;
        Mat img_matches2;

        //Drawing the matches and storing the result in img_matches2
        drawMatches(query_image, keypoints_query, object_image,
        keypoints_object, good_matches, img_matches2);

        //Used to display text on an image. Here the image is img_matches2 and
        the text is object[i]
        putText(img_matches2, object[i], Point(10, 250),
        FONT_HERSHEY_SIMPLEX, 2, (255, 255, 255), 2);
        cout << "By means of the best matches, the Object Found is : " <<
        cur_object;

        //Creates a new window, named "Matches Found"
```

```
                namedWindow("Matches Found", WINDOW_NORMAL);

                //Used to display the img_matches2 image in the "Matches Found"
                window
                imshow("Matches Found", img_matches2);
                break;
        }

        else flag = false;

        //Used for de-allocating the keypoints and descriptors variables
        descriptors_object.release();
        keypoints_object.clear();

}

//Failure check if incase the object is not found
if ((i == no_of_objects) && (flag == false))
{
        cout << "Sorry! No Object found in the Dataset\n";
        system("pause");
}

waitKey(0);
return 0;

}
```

main.cpp in **KeyptsAndDescrToFile** Solution

This program is used to extract the keypoints, calculate the descriptors and store the values in a text file. This program has to be executed only once for each object in the dataset.

```
#include <stdio.h>
#include <iostream>
#include "opencv2/core/core.hpp"
#include "opencv2/features2d/features2d.hpp"
#include "opencv2/highgui/highgui.hpp"
#include "opencv2/nonfree/features2d.hpp"
#include <fstream>

using namespace cv;
using namespace std;

/** @function main */
int main(int argc, char** argv)
{


	//name_of_object in the below code should be replaced with the name of the object image

	Mat img =
	imread("C:\\Users\\sunny\\Desktop\\Project\\ObjectImages\\name_of_object.jpg", 1);

	if (!img.data)
	{
		printf("Error loading file! Please try again.\n");
		system("pause");
		return -1;
	}

	//Detect the keypoints using SURF Detector
	int minHessian = 400;
	SurfFeatureDetector detector(minHessian);
	std::vector<KeyPoint> keypoints;
	detector.detect(img, keypoints);

	//Calculate descriptors
	SurfDescriptorExtractor extractor;
	Mat descriptors;
	extractor.compute(img, keypoints, descriptors);
```

```
	//Store the keypoints and descriptors into the text file
	cv::FileStorage
	store("C:\\Users\\sunny\\Desktop\\Project\\KeyptsAndDescriptors\\name_of_object.txt",
	cv::FileStorage::WRITE);
	cv::write(store, "keypoints", keypoints);
	cv::write(store, "descriptors", descriptors);
	store.release();

	cout << "File Created!";

	while (1)
	{
		//wait for 'esc' key press for 30 ms. If 'esc' key is pressed, break loop
		if (waitKey(300) == 27)
			break;
	}

	return 0;

}
```

main.cpp in **KeyPointsExtraction** Solution

Here, the keypoints and descriptors are extracted and calculated for an image and they are displayed on the image in a window.

```
#include <stdio.h>
#include <iostream>
#include "opencv2/core/core.hpp"
#include "opencv2/features2d/features2d.hpp"
#include "opencv2/highgui/highgui.hpp"
#include "opencv2/nonfree/features2d.hpp"

using namespace cv;

int main(int argc, char** argv)
{

	Mat object_image = imread("C:\\Users\\sunny\\Desktop\\Images\\book.jpg", 1);

	std::vector<KeyPoint> keypoints_object;

	Mat descriptors_object;

	if (!object_image.data)
	{

		printf("Error loading file! Please try again.\n");
		//cin.get();
		system("pause");
		return -1;

	}

	//Detect the keypoints using SURF Detector
	int minHessian = 500;
	SurfFeatureDetector detector(minHessian);

	detector.detect(object_image, keypoints_object);

	//Calculate the descriptors
	SurfDescriptorExtractor extractor;
	extractor.compute(object_image, keypoints_object, descriptors_object);
```

```
        Mat img_matches;
        drawKeypoints(object_image, keypoints_object, img_matches, Scalar::all(-1));

        namedWindow("Keypoints And Descriptors", WINDOW_NORMAL);
        imshow("Keypoints And Descriptors", img_matches);

        waitKey(0);
        return 0;

}
```

main.cpp in **KeyPtsAndDescrExtraction** Solution

In this program, two images are taken and the keypoints and descriptors are extracted for each. Then the calculated keypoints and descriptors are compared and the matches are drawn.

```
#include <stdio.h>
#include <iostream>
#include "opencv2/core/core.hpp"
#include "opencv2/features2d/features2d.hpp"
#include "opencv2/highgui/highgui.hpp"
#include "opencv2/nonfree/features2d.hpp"

using namespace cv;

int main(int argc, char** argv)
{
	Mat object_image = imread("C:\\Users\\sunny\\Desktop\\Images\\book.jpg", 1);
	Mat query_image = imread("C:\\Users\\sunny\\Desktop\\Images\\book3.jpg", 1);

	std::vector<KeyPoint> keypoints_object;
	std::vector<KeyPoint> keypoints_query;

	Mat descriptors_object;
	Mat descriptors_query;

	if (!query_image.data || !object_image.data)
	{
		printf("Error loading file! Please try again.\n");
		//cin.get();
		system("pause");
		return -1;
	}

	//Detect the keypoints using SURF Detector
	int minHessian = 2500;
	SurfFeatureDetector detector(minHessian);

	detector.detect(object_image, keypoints_object);
	detector.detect(query_image, keypoints_query);

	//Calculate the descriptors
	SurfDescriptorExtractor extractor;
	extractor.compute(object_image, keypoints_object, descriptors_object);
	extractor.compute(query_image, keypoints_query, descriptors_query);
```

```
        BFMatcher matcher(NORM_L2);
        std::vector< DMatch > matches;
        matcher.match(descriptors_query, descriptors_object, matches);

        Mat img_matches;
        drawMatches(query_image, keypoints_query, object_image, keypoints_object, matches,
        img_matches);

        namedWindow("Matches", WINDOW_NORMAL);
        imshow("Matches", img_matches);

        waitKey(0);
        return 0;

}
```