A

Project Report on

"IMPLEMENTATION OF VISION FOR ROBOTS"

Submitted in partial fulfillment of the requirements for the award of the degree of

BACHELOR OF TECHNOLOGY

IN

ELECTRONICS AND COMMUNICATION ENGINEERING

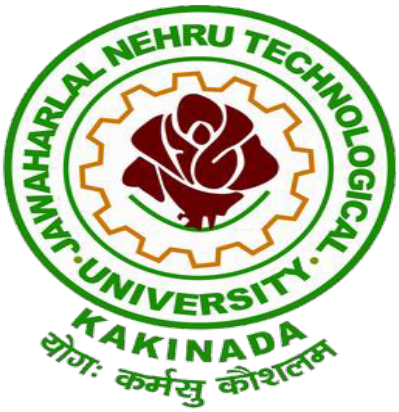
BY

B. GURURAJA	**(17VV1A0407)**
G. KRISHNA RAO	**(18VV5A0465)**
K. ESHWAR AJAY	**(17VV1A0423)**
K. SAICHARAN	**(17VV1A0426)**
MD. SHOAIB SHARIFF	**(17VV1A0433)**

Under the esteemed guidance of

DR.CH.SRINIVASA RAO.

Professor of ECE Department



DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

UNIVERSITY COLLEGE OF ENGINEERING VIZIANAGARAM
JNTUK VIZIANAGARAM

2017-2021

UNIVERSITY COLLEGE OF ENGINEERING VIZIANAGARAM
JNTUK VIZIANAGARAM
DEPARTMENT OF
ELECTRONICS AND COMMUNICATION ENGINEERING

CERTIFICATE

This is to certify that the project report titled **"IMPLEMENTATION OF VISION FOR ROBOTS",** being submitted by B.Gururaja (17VV1A0407), G.Krishna Rao (18VV5A0465), K.Eshwar Ajay (17VV1A0423), K.Sai Charan (17VV1A0426), Md.Shoaib Shariff (17VV1A0433) in partial fulfillment of the requirements for the award of the Degree of Bachelor of Technology in ELECTRONICS AND COMMUNICATION ENGINEERING. It is a record of bonafide work carried out by them under the esteemed guidance andsupervision of **Dr.Ch.Srinivasa Rao**.

Project Guide	**Head of Department**
Dr.Ch.Srinivasa Rao,	**Dr. K.C.B. RAO,**
Professor	**Professor & H.O.D,**
Department of E.C.E.	**Department of E.C.E.**

External Examiner

DECLARATION

We, B.Gururaja (17VV1A0407), G.Krishna Rao (18VV5A0465), K.Eshwar Ajay (17VV1A0423), K.Sai Charan (17VV1A0426), Md.Shoaib Shariff (17VV1A0433) declare that the project report titled "IMPLEMENTATION OF VISION FOR ROBOTS", submitted to University College of Engineering Vizianagaram, JNTUK in partial fulfillment of the requirements for the award of the degree of **B.Tech** in **Electronics and Communication Engineering** is a record of original and independent research work done by us during 2020-2021 under the supervision of **Dr.Ch.Srinivasa Rao**, Professor, **Department of E.C.E** and it has not formed the basis for the award of any Degree/Diploma/Associate Ship/Fellowship or other similar title to any candidate in any university.

DATE:

ROLL.NO	**NAME**	**SIGNATURE**
17VV1A0407	B. GURURAJA	
18VV5A0465	G. KRISHNA RAO	
17VV1A0423	K. ESHWAR AJAY	
17VV1A0426	K. SAICHARAN	
17VV1A0433	MD. SHOAIB SHARIFF	

ACKNOWLEDGEMENTS

We deeply express our hearty gratitude and thanks to our project guide **Dr.Ch.Srinivasa Rao**, Professor, Department of Electronics and Communication Engineering, for his guidance and cooperation for completing this project. We convey our heartfelt thanks to him for his inspiring assistance till the end of our project.

We have great pleasure in expressing our deep sense of gratitude to **Prof.K.C.B.RAO, HOD,** Department of Electronics and Communication Engineering, for his inspiring guidance, constant encouragement and support throughout our project work.

We express our deep sense of gratitude to the project review committee members, of Electronics and Communication Engineering Department**, Sri.R.Gurunadha, Dr.T.S.N.Murthy,Sri.A.Gangadhar, Sri.G.Appalanaidu, Smt.B.Nalini, Smt.M.Hema.** We are very much obliged to them for their help, affection and strain taken in reviewing the project at every stage of its progress.

We would like to thank **Prof.G.Swami Naidu,** Principal of **UNIVERSITY COLLEGE OF ENGINEERING VIZIANAGARAM JNTUK**, for the facilities provided in the college in carrying out the project successfully.

We thank the **Teaching and Non-teaching staff members** of our **E.C.E DEPARTMENT,** the college and administration who helped us directly or indirectly in carrying out this project successfully.

Yours Sincerely,

B. GURURAJA	17VV1A0407
G. KRISHNA RAO	18VV5A0465
K. ESHWAR AJAY	17VV1A0423
K. SAICHARAN	17VV1A0426
MD. SHOAIB SHARIFF	17VV1A0433

TABLE OF CONTENTS

ABSTRACT

Computer Vision is an important feature for Robots in many real time applications. Object detection, recognition and tracking can be done using this vision. The field of computer vision provides number of algorithms for object detection and recognition. Scale Invariant Feature Transform (SIFT) is an efficient feature detection algorithm. The Proposed work concentrates on implementation of SIFT algorithm using sophisticated hardware on a Field Programmable Gate Array (FPGA) using Xilinx Software.

The field of digital image processing refers to Processing digital images by means of a digital computer. Before 1960's, the size of a computer was as big as a room. Later due to the invention of transistors and introduction of Very Large Scale Integration (VLSI) technology has drastically changed the size and speed of a computer. The individual components of a computer like Control unit, Arithmetic and Logical Unit and memory can now be integrated on a single chip. Now the size of a computer is as small as our palm. This gives us a good scope in utilization of these small computers in robots for performing digital image processing in real time environment.

Feature based image matching is a key task in many computer vision applications like object recognition. SIFT Proposed by David Lowe is one of the best feature recognition algorithm. The interesting points of any object in an image are extracted using feature description. This feature description extracted from a training image is then used to identify the object in a test image. The features extracted from the training image must be detectable even under changes in image scale, noise and illumination. Such points usually lie at object edges i.e. on high contrast regions. SIFT feature descriptor is invariant to uniform scaling, orientation and partially invariant to illumination changes.

The first stage in SIFT is Scale Space peak selection which consists of selection of potential interest points which are identified by scanning the image over location and scale. Second stage includes localization of candidate keypoints and are eliminated if found to be unstable. The third stage identifies the dominant orientations for each keypoint. The final stage builds a local image descriptor for each keypoint based upon the image gradients in its local neighborhood.

Robots in real time need to capture picture using a digital camera. The captured image is to be processed using an onboard computer. This process identifies the features of the image by comparing with the images existing in the database.

This robot then takes a necessary action based on the output from the image processing section. For detection of features in an image, SIFT algorithm is used. The day to day improvement in VLSI technology helps in building more efficient hardware processing units for our daily computational purposes. In our research, we will design and implement an efficient hardware on FPGA to run SIFT algorithm. The same concept can be further implemented for modified SIFT algorithm also for video processing. The coding for hardware implementation will be done in Very High Speed Integrated Circuit Hardware Description Language (VHDL) on Xilinx Integrated Software Environment (ISE).

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

1.1 INTRODUCTION TO SIFT

Scale Invariant Features Transform (SIFT) is an algorithm used to locate the keypoints features of an image and create a descriptor for each keypoint. These descriptors can be used to detect an object or a pattern in an image. Object recognition is used in many applications such as: Manufacturing Industry, face recognition for security purposes, vehicle tracking or any real time embedded system that requires object identification from an image or a stream of images. Object recognition is one of the main problems that Computer Vision tries to solve. Object recognition techniques try to identify an object in an image or a stream of images. The main purpose of these techniques is to find a way of describing the desired object, and use this description to identify the same object in another image with reliable accuracy and satisfactory performance. This represents a huge challenge, because the accuracy depends generally on the number of keypoints used to describe an object. And this will result in decreasing the performance due to the need of more computations. Keeping in mind that the same process above will be used to find the keypoints in the target image to identify its keypoints and match their descriptors.

The classical approach for object recognition was to explore all locations in the target image in order to locate the desired object. Many techniques were developed to solve object recognition problem by extracting features rather than the computationally-expensive classical approach. Some techniques used grouping of edges or line segments, these techniques worked well for some classes of object, but their results were not always reliable. Other techniques used Harris corner detector to find interest points and create a descriptor, these techniques gave good results in terms of speed and dealing with cluttered images, but if the scale has been changed they fail.

SIFT was introduced by David G. Lowe in 1999 as an object recognition algorithm to identify locations in an image that are invariant with respect to scaling, rotation, image translation and minimally affected by noise and distortions.

SIFT keypoints are the maxima or minima of scale space after applying Difference of Gaussian (DoG) and these keypoints are assigned to canonical orientation so that the descriptor is invariant to image orientation. The descriptor is created by including any pixel in the neighbour of the key location by a certain distance and measure their orientation relative to the key location orientation.

SIFT can be implemented as follows:

- Scale-space extrema detection
- Keypoint Localization

- Orientation Assignment
- Keypoint descriptor

➢ **Scale Space Extrema Detection**

SIFT uses DoG as an approximation to Laplacian of Gaussian, because the Laplacian of Gaussian is computationally expensive compared to DoG. This step is used to make it possible to detect the scaled keypoint using the same window. Then the extrema is obtained from the resulting image. Each pixel is compared to its 8 direct neighbouring pixels (in the same scale) and with the 18 pixels of the previous and the next scales. Thus, in total each pixel is compared with its 26 neighbouring pixels.

➢ **Keypoint Localization**

DoG has high response for edges, so edges should be removed. For this, SIFT used Hessian matrix to compute the principal curvature and if the ratio is greater than a threshold the edge is removed.

$$H = \begin{bmatrix} D_{xx} & D_{xy} \\ D_{xy} & D_{yy} \end{bmatrix}$$

$$Tr(H) = D_{xx} + D_{yy}$$

$$Det(H) = D_{xx}D_{yy} - (D_{xy})^2$$

$$R = \frac{Tr(H)^2}{Det(H)}$$

Where H is the Hessian matrix of the image, Tr(H) is the trace of the matrix H, DET(H) is the determinant of H, R is the ratio.

➢ **Orientation Assignment**

Depending on the image scale a neighbourhood of pixels is chosen and the orientation of each pixel relative to the keypoint pixel is calculated and used to create an orientation histogram with a pre-specified number of bins.

$$m(x,y) = \sqrt{(A(x+1,y) - A(x-1,y))^2 + (A(x,y+1) - A(x,y-1))^2}$$

$$\theta(x,y) = tan^{-1}\left(\frac{A(x,y+1) - A(x,y-1)}{A(x+1,y) - A(x-1,y)}\right)$$

The highest peak in the histogram is taken along with any value greater than 0.8 of this peak value. The result is used to create keypoint of the same location and scale but with different directions to increase the matching stability.

➢ **Keypoint Descriptor**

A descriptor is created using the orientation bin histogram represented as a vector, forming keypoint descriptor along with some measurements to achieve robustness. Once the descriptor

is created it can be used to identify an object in another image by comparing this descriptor with target image's descriptor. Usually the matching process take into account the nearest neighbour, but sometimes the second-closest match is closer than the nearest match due to noise. Hence, a ratio between the closest and the second-closest distance is taken and if it's greater than 0.8 they are rejected. Overall steps for SIFT is shown in Figure 1.1.

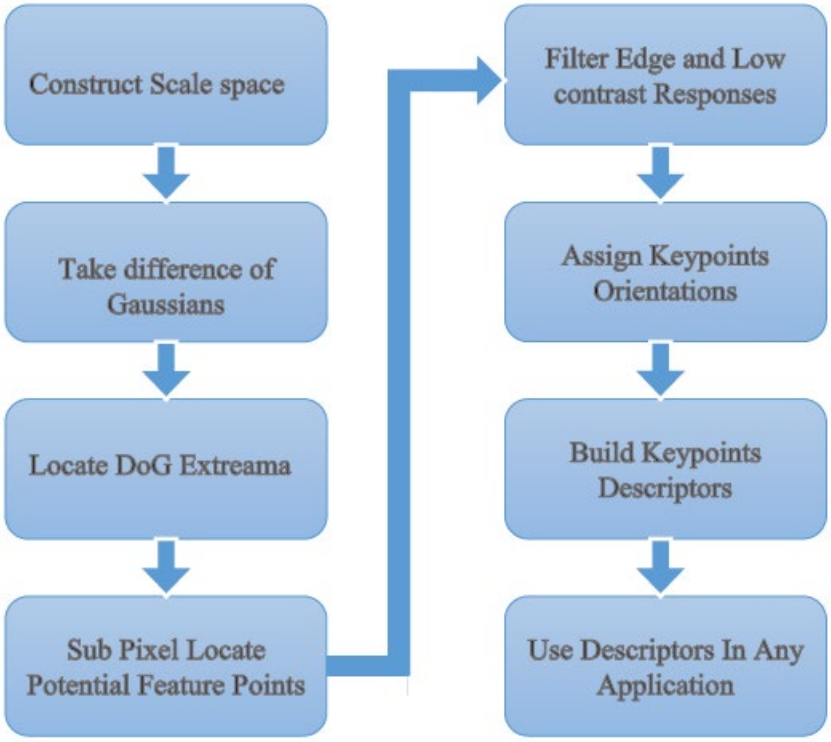


Fig 1.1 SIFT FLOWCHART

1.2. MATLAB

MATLAB is a high-performance language for technical computing. The name MATLAB stands for matrix laboratory. MATLAB was originally written to provide easy access to matrix software developed by the LINPACK and EISPACK projects, which together represent the state of art in software for matrix computation. It integrates computation, visualization, and programming in an easy-to-use environment where problems and solutions are expressed in familiar mathematical notation. Typical uses include Math and Computation algorithm development, Data acquisition modeling, Simulation, and Prototyping data analysis, exploration, and visualization, Scientific and engineering graphics application development, including graphical user interface building. MATLAB is an interactive system whose basic data element is an array that does not require dimensioning. MATLAB feature a family of application specific solutions called toolboxes. Toolboxes are comprehensive collection of MATLAB functions (M-file) that extends the MATLAB environment to solve particular classes of problems. Areas in which toolboxes are available include signal processing, image processing, control system, neural networks, fuzzy logic and

many others .This high performance numerical computation package with built in function for graphics and animation. It can perform symbolic algebra like mathematical, Maple and Matrix X. This allows solving many technical computing problems, especially those with matrix and vector formulations, in a fraction of the time it would take to write a program in a scalar non-interactive language such as C or FORTRAN environment to solve particular classes of problems. Areas in which toolboxes are available include signal processing, control systems, neural networks, fuzzy logic, wavelets, simulation and many others.

- **Features of MATLAB**

  - High-level language for technical computing. Development environment for managing code, files, and data.
  - Interactive tools for iterative exploration, design, and problem solving.
  - Mathematical functions for linear algebra, statistics, Fourier analysis, filtering, optimization, and numerical integration.
  - 2-D and 3-D graphics functions for visualizing data.
  - Tools for building custom graphical user interfaces.
  - Functions for integrating MATLAB based algorithms with external applications and languages, such as C, C++, FORTRAN, Java TM, COM, and Microsoft Excel.

- **MATLAB System**

The MATLAB system consists of these main parts: Desktop Tools and Development Environment This part of MATLAB is the set of tools and facilities that help you use and become more productive with MATLAB functions and files. Many of these tools are graphical user interfaces. It includes: the MATLAB desktop and Command Window, an editor and debugger, a code analyzer, and browsers for viewing help, the workspace, and folders. Mathematical Function Library This library is a vast collection of computational algorithms ranging from elementary functions like sum, sine, cosine, and complex arithmetic, to more sophisticated functions like matrix inverse, matrix Eigen values, Bessel functions, and fast Fourier transforms. The MATLAB language is a high-level matrix/array language with control flow statements, functions, data structures, input/output, and object-oriented programming features. It allows both “programming in the small” to rapidly create quick programs you do not intend to reuse. You can also do “programming in the large” to create complex application programs intended for reuse. Graphics MATLAB has extensive facilities for displaying vectors and matrices as graphs, as well as annotating and printing these graphs. It includes high-level functions for two dimensional and three-dimensional data visualization, image processing, animation, and presentation graphics. It also includes low-level functions that allow you to fully customize the appearance of graphics as wells as to build complete graphical user interfaces on your MATLAB applications.

- **External Interfaces**

The external interfaces library allows you to write C/C++ and FORTRAN programs that interact with MATLAB. It includes facilities for calling routines from MATLAB (dynamic linking), for calling MATLAB as a computational engine, and for reading and

writing MAT files.

- **Development Environment**

This section explains about the developing environment of MATLAB like how to start &quit MATLAB and about MATLAB tools etc.

- **Introduction**

This chapter provides a brief introduction to starting and quitting MATLAB, and the tools and functions that help you to work with MATLAB variables and files. For more information about the topics covered here, see the corresponding topics under Development Environment in the MATLAB documentation, which is available online as well as in print.

- **Start and Quit**

Starting MATLAB on a Microsoft Windows platform, to start MATLAB, double click the MATLAB shortcut icon on your Windows desktop. On a UNIX platform, to start MATLAB, type MATLAB at the operating system prompt. After starting MATLAB, the MATLAB desktop opens - see MATLAB Desktop. You can change the directory in which MATLAB starts, define startup options including running a script upon startup, and reduce startup time in some situations Quitting MATLAB To end your MATLAB session, select Exit MATLAB from the File menu in the desktop, or type quit in the Command Window. To execute specified functions each time MATLAB quits, such as saving the workspace, you can create and run a 'finish.m' script.

- **MATLAB Desktop**

When you start MATLAB, the MATLAB desktop appears, containing tools (graphical user interfaces) for managing files, variables, and applications associated with MATLAB. The first time MATLAB starts, the desktop appears as shown in the following illustration, although your Launch Pad may contain different entries. You can change the way your desktop looks by opening, closing, moving, and resizing the tools in it. You can also move tools outside of the desktop or return them back inside the desktop (docking). All the desktop tools provide common features such as context menus and keyboard shortcuts. You can specify certain characteristics for the desktop tools by selecting Preferences from the File menu. For example, you can specify the font characteristics for Command Window text. For more information, click the Help button in the Preferences dialog box.

- **Desktop Tools**

This section provides an introduction to MATLAB's desktop tools. You can also use MATLAB functions to perform most of the features found in the desktop tools. The tools are

- Current Directory Browser
- Workspace Browser
- Array Editor
- Editor/Debugger
- Command Window
- Command History

- Launch Pad
- Help Browser Current Directory Browser

MATLAB file operations use the current directory and the search path as reference points. Any file you want to run must either be in the current directory or on the search path.

- **Workspace Browser**

The MATLAB workspace consists of the set of variables (named arrays) built up during a MATLAB session and stored in memory. Variables can be added to the workspace by using functions, running M-files, and loading saved workspaces. To view the workspace and information about each variable, use the Workspace browser, or use the functions who and who's. To delete variables from the workspace, select the variable and select Delete from the Edit menu. Alternatively, use the clear function. The workspace is not maintained after you end the MATLAB session. To save the workspace to a file that can be read during a later MATLAB session, select Save Workspace As from the File menu, or use the save function. This saves the workspace to a binary file called a MAT-file, which has a .mat extension. There are options for saving to different formats. To read in a MAT-file, select Import Data from the File menu, or use the load function.

- **Help Browser**

Use the Help browser to search and view documentation for all your Math Works products. The Help browser is a Web browser integrated into the MATLAB desktop that displays HTML documents. To open the Help browser, click the help button in the toolbar, or type help browser in the Command Window. The Help browser consists of two panes, the Help Navigator, which you use to find information, and the display pane, where you view the information. Help Navigator Use to Help Navigator to find information. It includes: Product filter - Set the filter to show documentation only for the products you specify. Contents tab- View the titles and tables of contents of documentation for your products. Index tab - Find specific index entries (selected keywords) in the Math Works documentation for your products.

Search tab - Look for a specific phrase in the documentation. To get help for a specific function, set the Search type to Function Name. Favorites tab - View a list of documents you previously designated as favorites.

- **Display Pane**

After finding documentation using the Help Navigator, view it in the display pane. While viewing the documentation, you can: Browse to other pages - Use the arrows at the tops and bottoms of the pages, or use the back and forward buttons in the toolbar. Bookmark pages - Click the Add to Favorites button in the toolbar. Print pages - Click the print button in the toolbar.

Find a term in the page - Type a term in the Find in page field in the toolbar and click Go. Other features available in the display pane are: copying information, evaluating a selection and viewing Web pages. Search Path to determine how to execute functions you call, MATLAB uses a search path to find M-files and other MATLAB-related files, which are organized in directories on your file system. Any file you want to run in MATLAB must reside in the current directory or in a directory that is on the search path. By default, the files supplied with MATLAB and Math Works toolboxes are included in the search path.

- **Expressions**

Like most other programming languages, MATLAB provides mathematical expressions, but unlike most programming languages, these expressions involve entire matrices. The building blocks of expressions are:

- Variables.
- Numbers.
- Operators.
- Functions.
- Variables.

MATLAB does not require any type declarations or dimension statements. When MATLAB encounters a new variable name, it automatically creates the variable and allocates the appropriate amount of storage. If the variable already exists, MATLAB changes its contents and, if necessary, allocates new storage. For example, if no.of students = 25 Creates a 1-by-1 matrix named num-students and stores the value 25 in its single element. Variable names consist of a letter, followed by any number of letters, digits, or underscores. MATLAB uses only the first 31 characters of a variable name. MATLAB is case sensitive; it distinguishes between uppercase and lowercase letters. 'A' and 'a' is not the same variable. To view the matrix assigned to any variable, simply enter the variable name.

1.3. XILINX-SYSTEM-GENERATOR:

The System Generator is a DSP design tool from Xilinx that enables the use of the MathWorks model-based Simulink design environment for FPGA design. The design tools facilitate the design process by obscuring the technical knowledge necessary for FPGA a Register Transfer Level (RTL) design. Instead, a design is modeled using the intuitive visual environment within Simulink that uses several specific block sets accelerate the development. Additionally, System Generator can perform the FPGA implementation steps: synthesis, mapping, and place and route to generate the FPGA executable file. System Generator is part of the ISE® Design Suite and provides Xilinx DSP Block set such as adders, multipliers, registers, filters and memories for application specific design. These blocks leverage the Xilinx IP core generators to deliver optimized results for the selected device. Designs are captured in the DSP friendly Simulink modeling environment using a Xilinx specific Block set. All of the downstream FPGA implementation steps including

synthesis and place and route are automatically performed to generate an FPGA programming file. Advantage of using Xilinx system generator for hardware implementation is that Xilinx Block set provides close integration with MATLAB Simulink that helps in co-simulating the FPGA module with pixel vector provided by MATLAB Simulink Block. For accomplishing Image processing task using Xilinx System Generator needs two Software tools to be installed. One is MATLAB Version R2013a or higher and Xilinx ISE 14.x. The System Generator token available along with Xilinx has to be configured to MATLAB.

1.4 GOOGLE COLAB-PYTHON:

- **Introduction:**

Google is quite aggressive in AI research. Over many years, Google developed AI framework called Tensor Flow and a development tool called Collaboratory. Today Tensor Flow is open-sourced and since 2017, Google made Collaboratory free for public use. Colaboratory is now known as Google Colab or simply Colab. Another attractive feature that Google offers to the developers is the use of GPU. Colab supports GPU and it is totally free. The reasons for making it free for public could be to make its software a standard in the academics for teaching machine learning and data science. It may also have a long term perspective of building a customer base for Google Cloud APIs which are sold per-use basis. Irrespective of the reasons, the introduction of Colab has eased the learning and development of machine learning applications.

Most importantly, it does not require a setup and the notebooks that you create can be simultaneously edited by team members - just the way you edit documents in Google Docs. Colab supports many popular machine learning libraries which can be easily loaded.

- **System:**

As a programmer the following functions has been performed using Google Colab.

- Write and execute code in Python.
- Document the code that supports mathematical equations.
- Create/Upload/Share notebooks.
- Import/Save notebooks from/to Google Drive.
- Import/Publish notebooks from GitHub.
- Import external datasets e.g. from Kaggle.
- Integrate PyTorch, TensorFlow, Keras, OpenCV.
- Free Cloud service with free GPU.

As Colab implicitly uses Google Drive for storing your notebooks, ensure that you are logged in to your Google Drive account before proceeding further.We need to follow following steps in order to execute program.

Step 1: Open the following URL in your browser: https://colab.research.google.com
Step 2: Click on the NEW PYTHON 3 NOTEBOOK link at the bottom of the screen.

There is a code window in which you would enter your Python code.

- **Setting Notebook Name**

  To rename the notebook, click on this name and type in the desired name. We will call this notebook as My First Colab Notebook. So type in this name in the edit box and hit ENTER. The notebook will acquire the name that you have given now.

- **Entering Code:**

  You will now enter a trivial Python code in the code window and execute it.

- **Execution code:**

To execute the code, click on the arrow on the left side of the code window. After a while, you will see the output underneath the code window.

- **Adding code cell:**

  To add more code to your notebook, select the following menu options

- **Insert / Code Cell**

Alternatively, just hover the mouse at the bottom center of the Code cell. When the CODEand TEXT buttons appear, click on the CODE to add a new cell. A new code cell will be added underneath the current cell.To run the entire code in your notebook without an interruption, execute the following menu options:
Runtime / Reset and run all.

- **Charging cell order:**

  When your notebook contains a large number of code cells, you may come across situations where you would like to change the order of execution of these cells. You can do so by selecting the cell that you want to move and clicking the UP CELL or DOWN You may click the buttons multiple times to move the cell for more than a single position.

- **Deleting Cell:**

  During the development of your project, you may have introduced a few now-unwanted cells in your notebook. You can remove such cells from your project easily with a single click. Click on the vertical-dotted icon at the top right corner of your code cell. Click on the Delete cell option and the current cell will be deleted.

As the code cell supports full Python syntax, we may use Python comments in the code window to describe code. However, many a time we need more than a simple text based comments to illustrate the ML algorithms. ML heavily uses mathematics and to explain those terms and equations to your readers you need an editor that supports LaTex - a language for mathematical representations. Colab provides Text Cells for this purpose. Text Cells are

formatted using markdown.

- **Saving to Google Drive :**

Colab allows you to save your work to your Google Drive. The action will create a copy of your notebook and save it to your drive.

- **Saving to GitHub:**

Save your work to your GitHub repository. File / Save a copy in GitHub.

- **Sharing notebook:**

To share the notebook that you have created with other co-developers, you may share copy that you have made in your Google Drive. To publish the notebook to general audience, you may share it from your GitHub repository. There is one more way to share your work and that is by clicking on the SHARE link at the top right hand corner of your Colab notebook. This will open the share box, we may enter the email IDs of people with whom you would like to share the currentdocument. We can set the kind of access by selecting from the three options.

- Specified group of people.
- Colleagues in your organization.
- Anyone with the link.
- All public on the web.

- **System Aliases**

You will see the list in the output window as shown below:

bzcmp@ less* systemd-ask-password*
bzdiff* lessecho* systemd-escape*
bzegrep@ lessfile@ systemd-hwdb*
bzexe* lesskey* systemd-inhibit*
bzfgrep@ lesspipe* systemd-machine-id-setup*
bzgrep* ln* systemd-notify*
bzip2* login* systemd-sysusers*
bzip2recover* loginctl* systemd-tmpfiles*
bzless@ ls* systemd-tty-ask-password-agent*
bzmore* lsblk* tar*
cat* lsmod@ tempfile*
chgrp* mkdir* touch*
chmod* mknod* true*
chown* mktemp* udevadm*
cp* more* ulockmgr_server*
dash* mount* umount*
date* mountpoint* uname*
dd* mv* uncompress*
df* networkctl* vdir*
dir* nisdomainname@ wdctl*

dmesg* pidof@ which*
dnsdomainname@ ps* ypdomainname@
domainname@ pwd* zcat*
echo* rbash@ zcmp*
egrep* readlink* zdiff*
false* rm* zegrep*
fgrep* rmdir* zfgrep*
findmnt* run-parts* zforce*
fusermount* sed* zgrep*
grep* sh@ zless*
gunzip* sh.distrib@ zmore*
gzexe* sleep* znew*
gzip* stty*
hostname* su*

Execute any of these commands.

- **Mounting drive:**

  First, you need to mount your Google Drive in Colab. Select the following menu options:Tools / Command palette

- **Listing Drive Contents**
  You can list the contents of the drive using the ls command as follows:
  !ls "/content/drive/My Drive/Colab Notebooks"

- **Running Python Code**

  Now, let us say that you want to run a Python file called hello.py stored in your GoogleDrive. Type the following command in the Code cell:
  !python3 "/content/drive/My Drive/Colab Notebooks/hello.py"

- **Graphical output:**

Note that the graphical output is shown in the output section of the Code cell. Likewise, you will be able to create and display several types of charts throughout your program code.

- **ML-libraries:**

  - **Keras**
    Keras, written in Python, runs on top of TensorFlow, CNTK, or Theano. It enables easy and fast prototyping of neural network applications. It supports both convolutional networks (CNN) and recurrent networks, and also their combinations. It seamlessly supports GPU.
    To install Keras, use the following command:
    !pip install -q keras

- **Open CV**

  OpenCV is an open source computer vision library for developing machine learning applications. It has more than 2500 optimized algorithms which support several applications such as recognizing faces, identifying objects, tracking moving objects, stitching images, and so on. Giants like Google, Yahoo, Microsoft, Intel, IBM, Sony, Honda, Toyota use this library. This is highly suited for developing real-time vision applications.
  To install OpenCV use the following command:
  !apt-get -qq install -y libsm6 libxext6 && pip install -q -U opencv-python

- **XGBoost**

  XGBoost is a distributed gradient boosting library that runs on major distributed environments such as Hadoop. It is highly efficient, flexible and portable. It implements ML algorithms under the Gradient Boosting framework. To install XGBoost, use the following command:

  !pip install -q xgboost==0.4a30

- **GraphViz**

  Graphviz is an open source software for graph visualizations. It is used for visualization in networking, bioinformatics, database design, and for that matter in many domains where a visual interface of the data is desired.
  To install GraphViz, use the following command:
  !apt-get -qq install -y graphviz && pip install -q pydot
  By this time, you have learned to create Jupyter notebooks containing popular machine learning libraries. You are now ready to develop your machine learning models. This requires high processing power. Colab provides free GPU for your notebooks.

1.5. MOTIVATION

Object recognition is one of the main problems that Computer Vision tries to solve. Object recognition techniques try to identify an object in an image or a stream of images. The main purpose of these techniques is to find a way of describing the desired object, and use this description to identify the same object in another image with reliable accuracy and satisfactory performance. This represents a huge challenge, because the accuracy depends generally on the number of keypoints used to describe an object. And this will result in decreasing the performance due to the need of more computations.

In this we will implement SIFT algorithm for Image Processing. The same concept can be further implemented for modified SIFT algorithm also for video processing. The coding for hardware implementation will be done in Very High Speed Integrated Circuit Hardware Description Language (VHDL) on Xilinx Integrated Software Environment(ISE).

1.6. OBJECTIVES

The objectives of this work are mainly to match features of images using SIFT. They are as follows:

- To implement SIFT by using Matlab.
- To implement SIFT by using Xilinx-System-Generator.
- To implement SIFT by using Python.

1.7. PROJECT OVERVIEW

SIFT Algorithm, implementation of SIFT are reviewed in this chapter. Various important aspects that motivated us and objectives of the proposed work are presented in this chapter.

Literature survey i.e. a brief introduction about SIFT is discussed in chapter 2. Implementation of SIFT using Matlab, Xilinx-System-Generator, Python are presented in chapter 3.

Results obtained and their detailed discussion is presented in chapter 4.Conclusions and future scope is given in chapter 5

CHAPTER 2

LITERATURE SURVEY

2.1. INTRODUCTION

Image matching is a fundamental aspect of many problems in computer vision, including object or scene recognition, solving for 3D structure from multiple images, stereo correspondence, and motion tracking. This describes image features that have many properties that make them suitable for matching differing images of an object or scene. The features are invariant to image scaling and rotation, and partially invariant to change in illumination and 3D camera viewpoint. They are well localized in both the spatial and frequency domains, reducing the probability of disruption by occlusion, clutter, or noise. Large numbers of features can be extracted from typical images with efficient algorithms. In addition, the features are highly distinctive, which allows a single feature to be correctly matched with high probability against a large database of features, providing a basis for object and scene recognition.

The cost of extracting these features is minimized by taking a cascade filtering approach, in which the more expensive operations are applied only at locations that pass an initial test.

2.2. STAGES OF SIFT ALGORITHM

Following are the major stages of computation used to generate the set of image features:

- Scale-space extrema detection.
- Keypoint localization.
- Orientation assignment
- Keypoint descriptor

This approach has been named the Scale Invariant Feature Transform (SIFT), as it transforms image data into scale-invariant coordinates relative to local features.

An important aspect of this approach is that it generates large numbers of features that densely cover the image over the full range of scales and locations. A typical image of size 500x500 pixels will give rise to about 2000 stable features (although this number depends on both image content and choices for various parameters). The quantity of features is particularly important for object recognition, where the ability to detect small objects in cluttered backgrounds requires that at least 3 features be correctly matched from each object for reliable identification.

For image matching and recognition, SIFT features are first extracted from a set of reference images and stored in a database. A new image is matched by individually comparing each feature from the new image to this previous database and finding candidate matching features based on Euclidean distance of their feature vectors. This will discuss fast nearest-neighbor algorithms that can perform this computation rapidly against large

databases.

The keypoint descriptors are highly distinctive, which allows a single feature to find its correct match with good probability in a large database of features. However, in a cluttered image, many features from the background will not have any correct match in the database, giving rise to many false matches in addition to the correct ones. The correct matches can be filtered from the full set of matches by identifying subsets of keypoints that agree on the object and its location, scale, and orientation in the new image. The probability that several features will agree on these parameters by chance is much lower than the probability that any individual feature match will be in error. The determination of these consistent clusters can be performed rapidly by using an efficient hash table implementation of the generalized Hough transform.

Each cluster of 3 or more features that agree on an object and its pose is then subject to further detailed verification. First, a least-squared estimate is made for an affine approximation to the object pose. Any other image features consistent with this pose are identified, and outliers are discarded. Finally, a detailed computation is made of the probability that a particular set of features indicates the presence of an object, given the accuracy of fit and number of probable false matches. Object matches that pass all these tests can be identified as correct with high confidence.

The development of image matching by using a set of local interest points can be traced back to the work of Moravec (1981) on stereo matching using a corner detector. The Moravec detector was improved by Harris and Stephens (1988) to make it more repeatable under small image variations and near edges. Harris also showed its value for efficient motion tracking and 3D structure from motion recovery (Harris, 1992), and the Harris corner detector has since been widely used for many other image matching tasks. While these feature detectors are usually called corner detectors, they are not selecting just corners, but rather any image location that has large gradients in all directions at a predetermined scale.

The initial applications were to stereo and short-range motion tracking, but the approach was later extended to more difficult problems. Zhang et al. (1995) showed that it was possible to match Harris corners over a large image range by using a correlation window around each corner to select likely matches. Outliers were then removed by solving for a fundamental matrix describing the geometric constraints between the two views of rigid scene and removing matches that did not agree with the majority solution. At the same time, a similar approach was developed by Torr (1995) for long-range motion matching, in which geometric constraints were used to remove outliers for rigid objects moving within an image.

The ground-breaking work of Schmid and Mohr (1997) showed that invariant local feature matching could be extended to general image recognition problems in which a feature was matched against a large database of images. They also used Harris corners to select interest points, but rather than matching with a correlation window, they used a rotationally invariant descriptor of the local image region. This allowed features to be matched under arbitrary orientation change between the two images. Furthermore, they demonstrated that multiple feature matches could accomplish general recognition under occlusion and clutter by identifying consistent clusters of matched features.

The Harris corner detector is very sensitive to changes in image scale, so it does not provide a good basis for matching images of different sizes. Earlier work by the author (Lowe, 1999) extended the local feature approach to achieve scale invariance. This work also described a new local descriptor that provided more distinctive features while being less sensitive to local image distortions such as 3D viewpoint change. This current survey provides a more in-depth development and analysis of this earlier work, while also presenting a number of improvements in stability and feature invariance.

There is a considerable body of previous research on identifying representations that are stable under scale change. Some of the first work in this area was by Crowley and Parker (1984), who developed a representation that identified peaks and ridges in scale space and linked these into a tree structure. The tree structure could then be matched between images with arbitrary scale change. More recent work on graph-based matching by Shokoufandeh, Marsic and Dickinson (1999) provides more distinctive feature descriptors using wavelet coefficients. The problem of identifying an appropriate and consistent scale for feature detection has been studied in depth by Lindeberg (1993, 1994). He describes this as a problem of scale selection, and we make use of his results below.

Recently, there has been an impressive body of work on extending local features to be invariant to full affine transformations (Baumberg, 2000; Tuytelaars and Van Gool, 2000; Mikolajczyk and Schmid, 2002; Schaffalitzky and Zisserman, 2002; Brown and Lowe, 2002). This allows for invariant matching to features on a planar surface under changes in orthographic 3D projection, in most cases by resampling the image in a local affine frame. However, none of these approaches are yet fully affine invariant, as they start with initial feature scales and locations selected in a non-affine-invariant manner due to the prohibitive cost of exploring the full affine space. The affine frames are are also more sensitive to noise than those of the scale-invariant features, so in practice the affine features have lower repeatability than the scale-invariant features unless the affine distortion is greater than about a 40 degree tilt of a planar surface (Mikolajczyk, 2002). Wider affine invariance may not be important for many applications, as training views are best taken at least every 30 degrees rotation in viewpoint (meaning that recognition is within 15 degrees of the closest training view) in order to capture non-planar changes and occlusion effects for 3D objects.

While the method to be presented in this SIFT is not fully affine invariant, a different approach is used in which the local descriptor allows relative feature positions to shift significantly with only small changes in the descriptor. This approach not only allows the descriptors to be reliably matched across a considerable range of affine distortion, but it also makes the features more robust against changes in 3D viewpoint for non-planar surfaces. Other advantages include much more efficient feature extraction and the ability to identify larger numbers of features. On the other hand, affine invariance is a valuable property for matching planar surfaces under very large view changes, and further research should be performed on the best ways to combine this with non-planar 3D viewpoint invariance in an efficient and stable manner.

Many other feature types have been proposed for use in recognition, some of which could be used in addition to the features described in this Implementation to provide further

matches under differing circumstances. One class of features are those that make use of image contours or region boundaries, which should make them less likely to be disrupted by cluttered backgrounds near object boundaries. Matas et al., (2002) have shown that their maximally-stable extremal regions can produce large numbers of matching features with good stability. Mikolajczyket al., (2003) have developed a new descriptor that uses local edges while ignoring unrelated nearby edges, providing the ability to find stable features even near the boundaries of narrow shapes super imposed on background clutter. Nelson and Selinger (1998) have shown good results with local features based on groupings of image contours. Similarly, Pope and Lowe (2000) used features based on the hierarchical grouping of image contours, which are particularly useful for objects lacking detailed texture.

The history of research on visual recognition contains work on a diverse set of other image properties that can be used as feature measurements. Carneiro and Jepson (2002) describe phase-based local features that represent the phase rather than the magnitude of local spatial frequencies, which is likely to provide improved invariance to illumination. Schiele and Crowley (2000) have proposed the use of multidimensional histograms summarizing the distribution of measurements within image regions. This type of feature may be particularly useful for recognition of textured objects with deformable shapes. Basri and Jacobs (1997) have demonstrated the value of extracting local region boundaries for recognition. Other useful properties to incorporate include color, motion, figure-ground discrimination, region shape descriptors, and stereo depth cues. The local feature approach can easily incorporate novel feature types because extra features contribute to robustness when they provide correct matches, but otherwise do little harm other than their cost of computation. Therefore, future systems are likely to combine many feature types.

CHAPTER 3

IMPLEMENTATION OF SIFT

3.1 STEPS IN SIFT ALGORITHM

- Constructing a Scale Space.
- DoG Approximation.
- Finding Key points.
- Get rid of bad key points.
- Assigning an Orientation to the Key points.
- Generate Sift Features.

3.2 CONSTRUCTING A SCALE SPACE

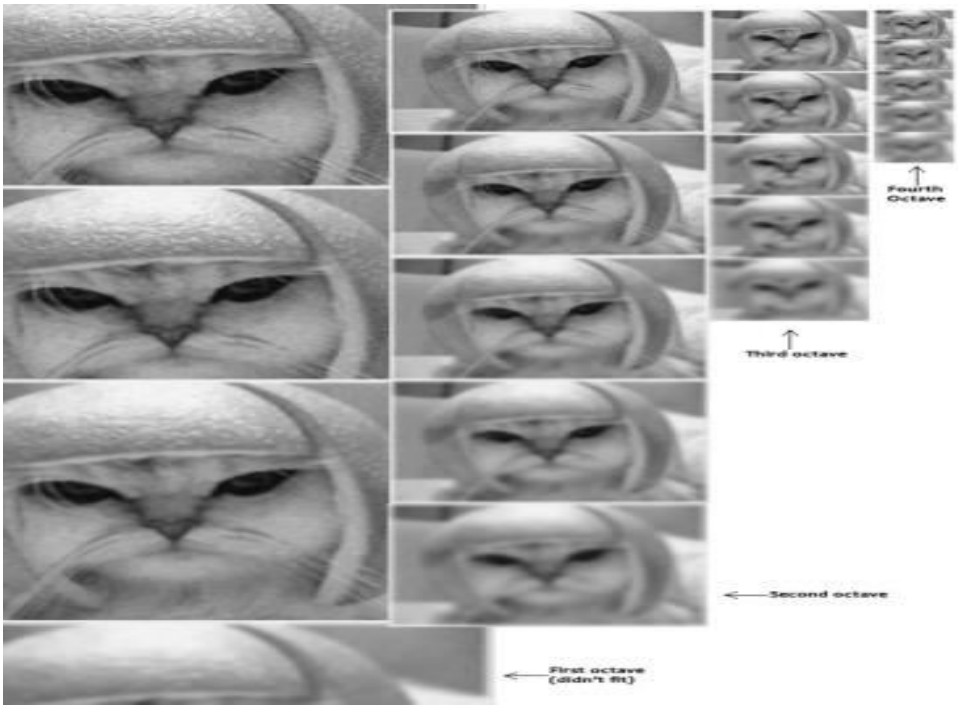


Fig 3.1 Constructing a Scale Space

SIFT takes scale spaces to the next level. Take the original image, and generate progressively blurred out images. Then, you resize the original image to half size and generate blurred out images again and keep repeating. The creator of sift suggests that 4 octaves and 5 blur levels are ideal for the algorithm. If the original image is doubled in size and antialiased a bit (by blurring it) then the algorithm produces more four times more keypoints. The more the keypoints, the better.

Blurring

$$L(x,y) = G(x,y) * I(x,y) \qquad (1)$$

Where * is the convolution operation in x and y, and

$$G(x, y, \sigma) = \frac{1}{2\pi\sigma^2} e^{-(x^2+y^2)/2\sigma^2} \qquad (2)$$

3.2.1. DOG APPROXIMATION:

This is the stage where the interest points, which are called keypoints in the SIFT framework, are detected. For this, the image is convolved with Gaussian filters at different scales, and then the difference of successive Gaussian-blurred images is taken. Keypoints are then taken as maxima/minima of the Difference of Gaussians (DoG) that occur at multiple scales. Specifically, a DoG image $D(x, y, \sigma)$ is given by

$$D(x, y, \sigma) = L(x, y, k_i\sigma) - L(x, y, k_j\sigma) \qquad (3)$$

Where $L(x, y, k_i\sigma)$ is the convolution of the original image I(x,y) with the Gaussian blur $G(x, y, k_i\sigma)$ at scale $k\sigma$,

i.e.,

$$L(x, y, k\sigma) = G(x, y, k\sigma) * I(x, y) \qquad (4)$$

Hence a DoG image between scales $k_i\sigma$ and $k_j\sigma$ is just the difference of the Gaussian-blurred images at scales $k_i\sigma$ and $k_j\sigma$. For scale space extrema detection in the SIFT algorithm, the image is first convolved with Gaussian-blurs at different scales. The convolved images are grouped by octave and the value of k_i is selected so that we obtain a fixed number of convolved images per octave. Then the Difference-of-Gaussian images are taken from adjacent Gaussian-blurred images per octave.

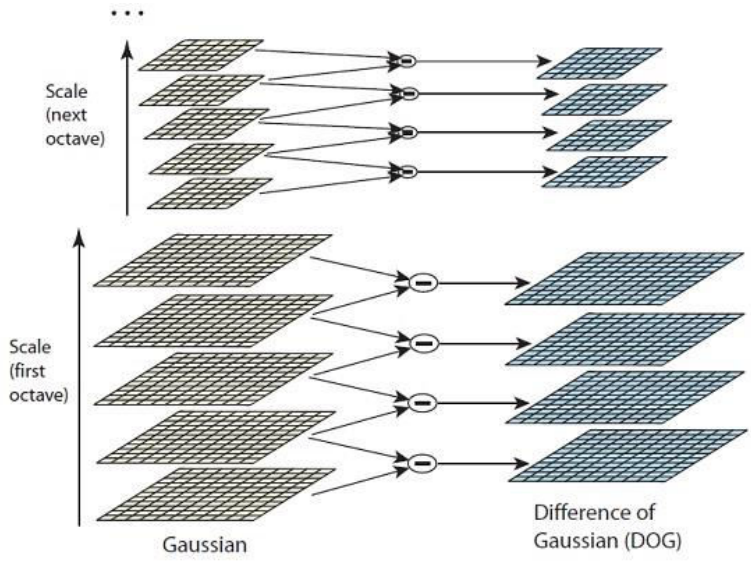


Fig 3.2.1 DOG Approximation

3.2.2 FINDING KEY POINTS:

Once DoG images have been obtained, keypoints are identified as local minima/maxima of the DoG images across scales. This is done by comparing each pixel in the DoG images to its eight neighbours at the same scale and nine corresponding neighbouring pixels in each of the neighbouring scales. If the pixel value is the maximum or minimum among all compared pixels, it is selected as a candidate keypoint.

(a) (b)

Fig 3.2.2 (a) Locate maxima/minima in DoG images (b) Subpixel maxima/minima

The new approach calculates the interpolated location of the extremum, which substantially improves matching and stability. The interpolation is done using the quadratic Taylor expansion of the Difference-of-Gaussian scale-space function, $D(x, y, \sigma)$ with the candidate keypoint as the origin. This Taylor expansion is given by

$$D(x) = D + \frac{\partial D^T}{\partial x} x + \frac{1}{2} x^T \frac{\partial^2 D}{\partial x^2} x \qquad (5)$$

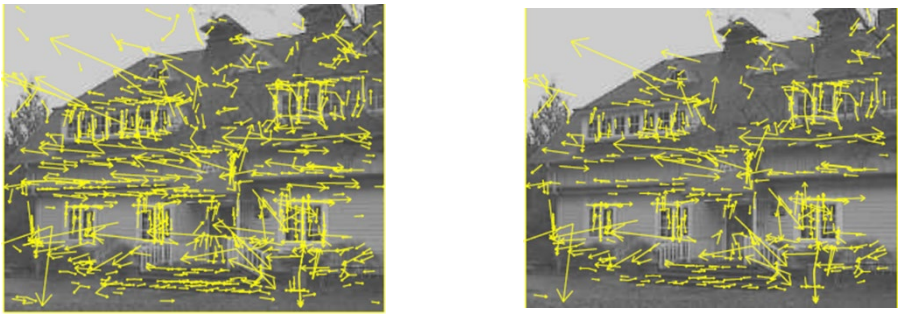
where D and its derivatives are evaluated at the candidate keypoint and $x = L(x, y, \sigma)$ is the offset from this point.

The location of the extremum, $\hat{x}$, is determined by taking the derivative of this function with respect to x and setting it to zero. If the offset $\hat{x}$ is larger than 0.5 in any dimension, then that's an indication that the extremum lies closer to another candidate keypoint. In this case, the candidate keypoint is changed and the interpolation performed instead about that point.

Otherwise the offset is added to its candidate keypoint to get the interpolated estimate for the location of the extremum. A similar subpixel determination of the locations of scale-space extrema is performed in the realtime implementation based on hybrid pyramids developed by Lindeberg and his co-workers.

3.2.3 GET RID OF BAD KEY POINTS:

After scale space extrema are detected (their location being shown in the uppermost image) the SIFT algorithm discards low contrast keypoints (remaining points are shown in the middle image) and then filters out those located on edges. Resulting set of keypoints is shown on last image.



(a) (b)

Fig 3.2.3 Get rid of bad key points (a) The initial 832 keypoints locations at maxima and minima of the DoG, (b) The final 536 keypoints that remain following an additional threshold on ratio of principal curvatures

Scale-Space extrema detection produces too many keypoint candidates, some of which are unstable. The next step in the algorithm is to perform a detailed fit to the nearby data for accurate location, scale, and ratio of principal curvatures. The value of the second-order Taylor expansion D(x) is computed at the offset $\hat{x}$.

If this value is less than 0.03, the candidate keypoint is discarded. Otherwise it is kept, with final scale-space location $y + \hat{x}$, where is the original location of the keypoint. we need to eliminate the keypoints that have poorly determined locations but have high edge responses. For poorly defined peaks in the DoG function, the principal curvature across the edge would be much larger than the principal curvature along it. Finding these principal curvatures amounts to solving for the eigenvalues of the second-order Hessian matrix, H

$$H = \begin{bmatrix} D_{xx} & D_{xy} \\ D_{xy} & D_{yy} \end{bmatrix}$$

The eigenvalues of H are proportional to the principal curvatures of D. It turns out that the ratio of the two eigenvalues, say α is the larger one, and β the smaller one, with ratio γ = α/β, is sufficient for SIFT's purposes. The trace of H, i.e., $D_{xx} + D_{yy}$, gives us the sum of the two eigenvalues, while its determinant, i.e., $D_{xx}D_{yy} - {D_{xy}}^2$, yields the product.

The ratio R = $Tr(H)^2/Det(H)$ can be shown to be equal to $(r+1)^2/r$, which depends only on the ratio of the eigenvalues rather than their individual values. R is minimum, when the eigenvalues are equal to each other. Therefore the higher the absolute difference between the two eigenvalues, which is equivalent to a higher absolute difference between the two

principal curvatures of D.

The higher the value of R.It follows that, for some threshold eigenvalue ratio r_{th}, if R for a candidate keypoint is larger than $(r_{th}+1)^2/r_{th}$, that keypoint is poorly localized and hence rejected. The new approach uses $r_{th} = 10$.

3.2.4. ASSIGNING AN ORIENTATION TO THE KEY POINTS:

Each keypoint is assigned one or more orientations based on local image gradient directions. This is the key step in achieving invariance to rotation as the keypoint descriptor can be represented relative to this orientation and therefore achieve invariance to image rotation.

First, the Gaussian-smoothed image $L(x, y, \sigma)$ at the keypoint's scale σ is taken so that all computations are performed in a scale-invariant manner. For an image sample $L(x, y)$ at scale σ, the gradient magnitude, $m(x, y)$, and orientation, $\theta(x, y)$, are precomputed using pixel differences.

$$m(x,y) = \sqrt{(L(x+1,y) - L(x-1,y))^2 - (L(x,y+1) - L(x,y-1))^2}$$

$$\theta(x,y) = tan^{-1}\left(\frac{L(x,y+1) - L(x,y-1)}{L(x+1,y) - L(x-1,y)}\right)$$

The magnitude and direction calculations for the gradient are done for every pixel in a neighboring region around the keypoint in the Gaussian-blurred image L. An orientation histogram with 36 bins is formed, with each bin covering 10 degrees.

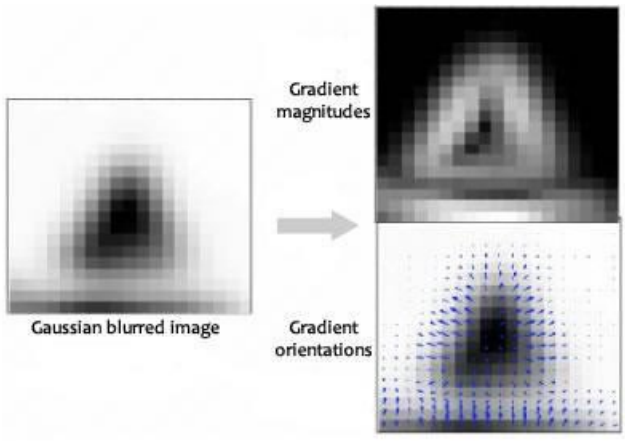


Fig 3.2.4 Assigning Orientation to Keypoints

Each sample in the neighboring window added to a histogram bin is weighted by its gradient magnitude and by a Gaussian-weighted circular window with a σ that is 1.5 times that of the scale of the keypoint. The peaks in this histogram correspond to dominant orientations.

3.2.5. GENERATE SIFT FEATURES:

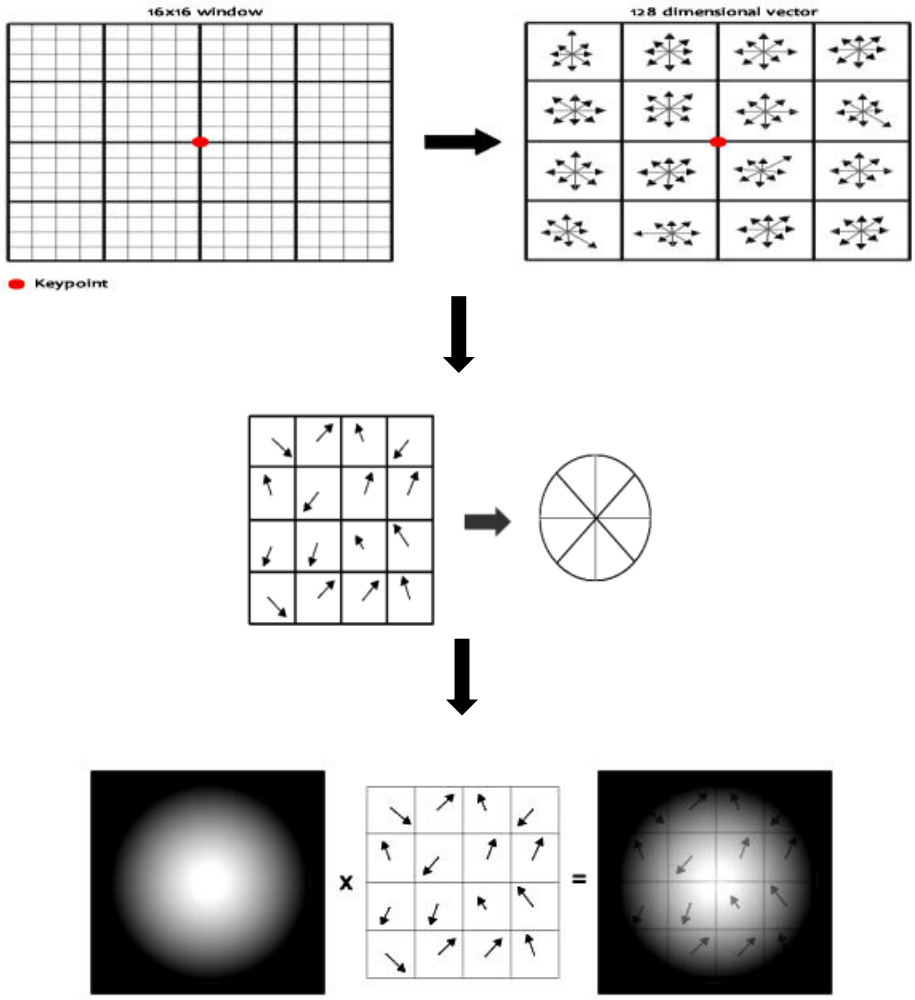


Fig 3.2.5 Generating SIFT Features

Take a 16×16 window of "in-between" pixels around the keypoint. Split that window into sixteen 4×4 windows. From each 4×4 window you generate a histogram of 8 bins. Each bin corresponding to 0-44 degrees, 45-89 degrees, etc. gradient orientations from the 4×4 are put into these bins. This is done for all 4×4 blocks. Finally, normalize the 128 values you get. To solve a few problems, you subtract the keypoint's orientation and also threshold the value of each element of the feature vector to 0.2 (and normalize again).

3.3 SIFT USING MATLAB:

Image processing toolbox:

The basic data structure in MATLAB is the array, an ordered set of real or complex elements. This object is naturally suited to the representation of images, real-valued ordered sets of color or intensity data. MATLAB stores most images as two-dimensional arrays (i.e., matrices), in which each element of the matrix corresponds to a single pixel in the displayed image. (Pixel is derived from picture element and usually denotes a single dot on a computer display.) For example, an image composed of 200 rows and 300 columns of different colored dots would be stored in MATLAB as a 200-by-300 matrix. Some images, such as RGB, require a three-dimensional array, where the first plane in the third dimension represents the red pixel intensities, the second plane represents the green pixel intensities, and the third plane represents the blue pixel intensities. This convention makes working with images in MATLAB similar to working with any other type of matrix data, and makes the full power of MATLAB available for image processing applications. For example, you can select a single pixel from an image matrix using normal matrix subscripting. I(2,15) This command returns the value of the pixel at row 2, column 15 of the image I. Storage Classes in the Toolbox. By default, MATLAB stores most data in arrays of class double. The data in these arrays is stored as double precision (64-bit) floating-point numbers. All of MATLAB's functions and capabilities work with these arrays. For image processing, however, this data representation is not always ideal. The number of pixels in an image may be very large; for example, a 1000-by-1000 image has a million pixels. Since each pixel is represented by at least one array element, this image would require about 8 megabytes of memory. To reduce memory requirements, MATLAB supports storing image data in arrays of as 8-bit or 16-bit unsigned integers, class uint8 and uint16. These arrays require one eighth or one fourth as much memory as double arrays. The toolbox supports a wide range of image processing operations, including:

- Geometric operations
- Neighborhood and block operations
- Linear filtering and filter design
- Transforms
- Image analysis and enhancement
- Binary image operations
- Region of interest operations

MATLAB can import/export several image formats:

- BMP (Microsoft Windows Bitmap)
- GIF (Graphics Interchange Files)
- HDF (Hierarchical Data Format)
- JPEG (Joint Photographic Experts Group)
- PCX (Paintbrush)
- PNG (Portable Network Graphics)
- TIFF (Tagged Image File Format)

- XWD (X Window Dump)
- raw-data and other types of image data

Data types in MATLAB:

- Double (64-bit double-precision floating point)
- Single (32-bit single-precision floating point)
- Int32 (32-bit signed integer)
- Int16 (16-bit signed integer)
- Int8 (8-bit signed integer)
- Uint32 (32-bit unsigned integer)
- Uint16 (16-bit unsigned integer)
- Uint8 (8-bit unsigned integer)

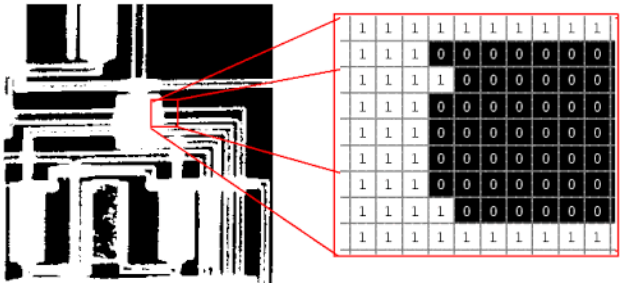


Fig 3.3.1 Example of binary image

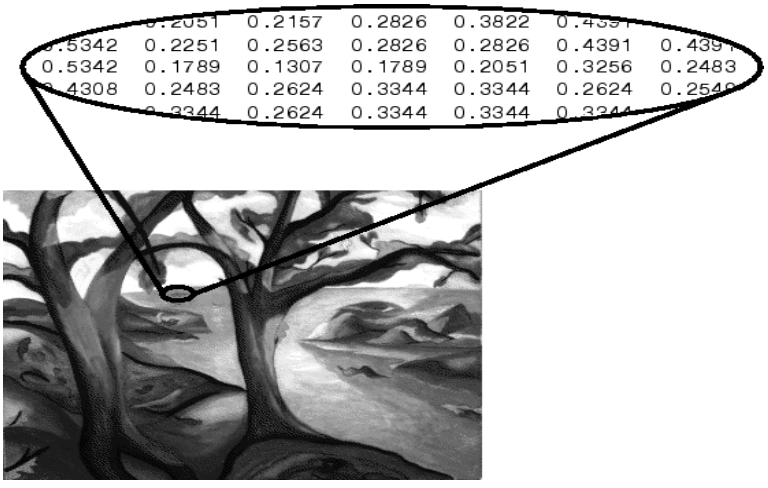


Fig 3.3.2 example of intensity image

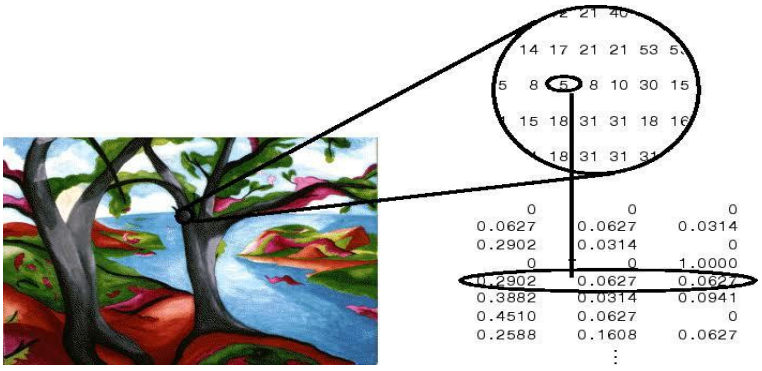


Fig 3.3.3 example of RGB image

Image Import and Export:

In MATLAB, the primary way to display images is by using the image function. This function creates a Handle Graphics image object, and it includes syntax for setting the various properties of the object. MATLAB also includes the images function, which is similar to image but which automatically scales the input data. The Image Processing Toolbox includes an additional display routine called imshow. Like image and imagesc, this function creates a Handle Graphics image object. However, imshow also automatically sets various Handle Graphics properties and attributes of the image to optimize the display. This section discusses displaying images using imshow. In general, using imshow for image processing applications is preferable to using image and imagesc. It is easier to use and in most cases, displays an image using one image pixel per screen pixel.

Read and write images in matlab

- img = imread('apple.jpg');
- dim = size(img);
- figure;
- imshow(img);
- imwrite(img, 'output.bmp', 'bmp');

Alternatives to imshow

- imagesc(I)
- imtool(I)
- image(I)

Image Display in MATLAB:

- image - create and display image object
- imagesc - scale and display as image
- imshow - display image
- colorbar - display colorbar
- getimage - get image data from axes
- truesize - adjust display size of image

- zoom - zoom in and zoom out of 2D plot

Image Conversion in MATLAB:

- gray2ind - intensity image to index image
- im2bw - image to binary
- im2double - image to double precision
- im2uint8 - image to 8-bit unsigned integers
- im2uint16 - image to 16-bit unsigned integers
- ind2gray - indexed image to intensity image
- mat2gray - matrix to intensity image
- rgb2gray - RGB image to grayscale
- rgb2ind - RGB image to indexed image
- Image Operations in MATLAB:
- RGB image to gray image
- Image resize
- Image crop
- Image rotate
- Image histogram
- Image histogram equalization
- Image DCT/IDCT
- Convolution

3.4 SIFT USING XILINX-SYSTEM-GENERATOR:

In VHDL we cannot upload the image directly so we give binary values as input. The binary values are obtained by converting the input image pixel values into binary values using matlab. After simulating in the Xilinx System Generator, we will get the output in the form of binary code. The binary code obtained from Xilinx System Generator should match the output of the matlab binary code.

We can create or open a System Generator project, and add blocks to define your design. The Quick Start section of the Getting Started Page provides links for easy access to the following steps:

- Create a System Generator project using the New Project wizard.
- Open existing System Generator projects.
- Open example Simulink projects provided by Xilinx System Generator.

After creating a new System Generator project we will get a window in which we have to add blocks along with Xilinx System Generator block and have to select the language as VHDL. Now a window is displayed with Generate Code Button as shown in below fig.

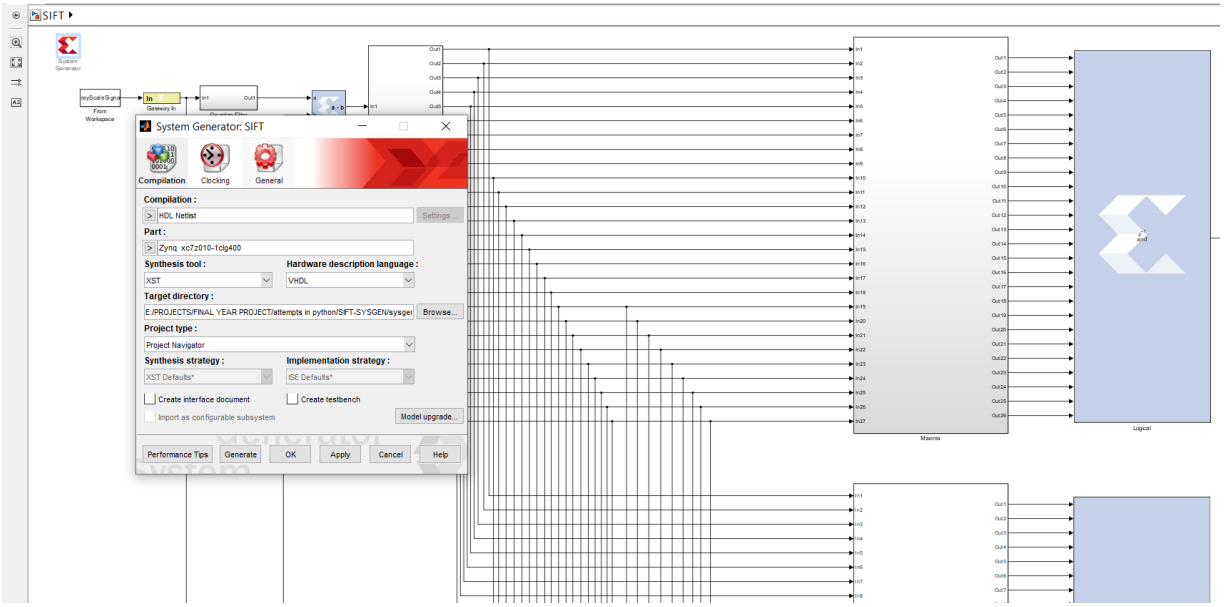


Fig 3.4 Xilinx-System Generator Window

In project manager our project will be present. If we open that we will get a window in which we can add required blocks. After that in the file navigator we have run the simulation in order check the errors, if we have any errors it will show after the completion of simulation. If the simulation is done perfectly then we will get the output to our workspace.

3.5 SIFT USING GOOGLE-COLAB PYTHON:

Google is quite aggressive in AI research. Over many years, Google developed AI framework called Tensor Flow and a development tool called Collaboratory. Today Tensor Flow is open-sourced and since 2017, Google made Collaboratory free for public use.Colaboratory is now known as Google Colab or simply Colab.Another attractive feature that Google offers to the developers is the use of GPU. Colab supports GPU and it is totally free. The reasons for making it free for public could be to make its software a standard in the academics for teaching machine learning and data science. It may also have a long term perspective of building a customer base for Google Cloud APIs which are sold per-use basis. Irrespective of the reasons, the introduction of Colab has eased the learning and development of machine learning applications.

Most importantly, it does not require a setup and the notebooks that you create can be simultaneously edited by team members - just the way you edit documents in Google Docs. Colab supports many popular machine learning libraries which can be easily loaded.

Steps involved:

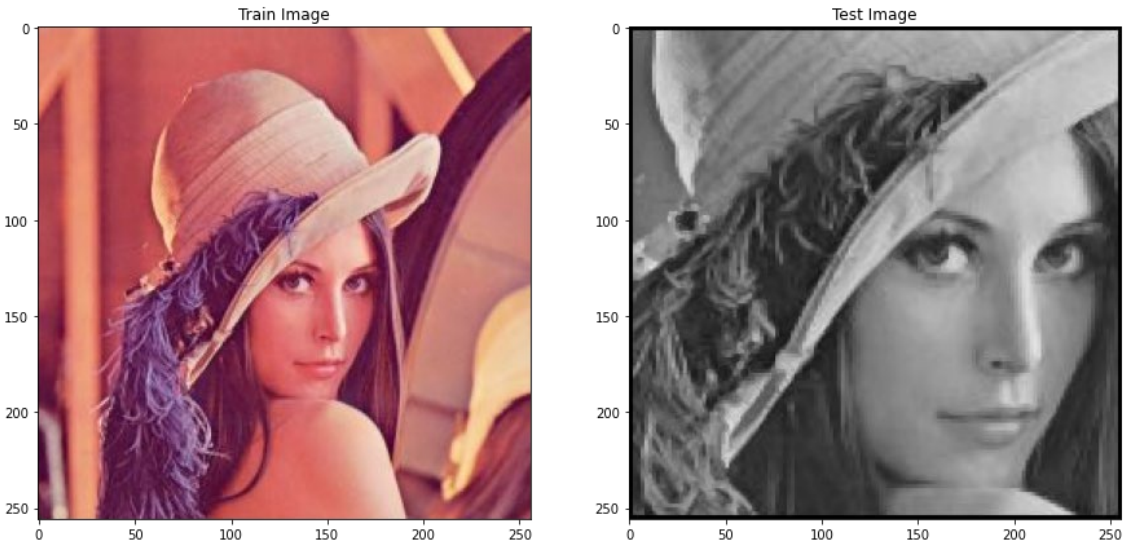
- Upload images to Google drive
- Open Google colab https://colab.research.google.com/
- Connect Google Drive to Google colab
- Create a New Note-book in Google colab
- Select GPU or CPU for processing
- Write code in cell
- Execute cell on clicking run all option, then we will get the output if there are no errors

CHAPTER 4

RESULTS AND DISCUSSIONS

After implementing SIFT using different softwares the obtained results are as follows

RESULTS OBTAINED BY USING PYTHON:



(a) **(b)**

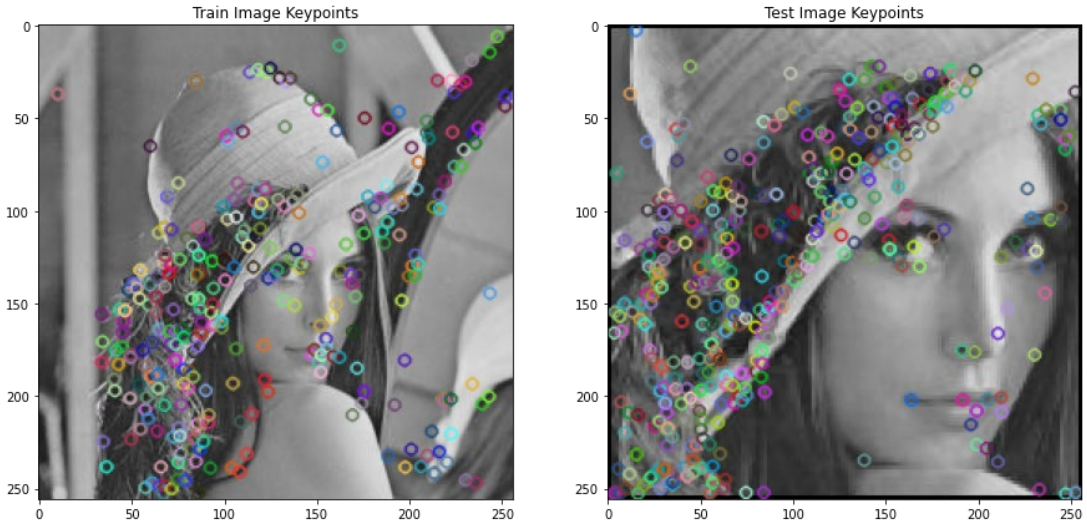
Fig 4.1. (a) Original image, (b) Original Cropped Image

From Figure 4.1, we can observe the images that are used for training and testing of SIFT algorithm i.e extraction of keypoints from the image (a) and (b) and mapping the similar keypoints present in both images (a) and (b).

(a) **(b)**

Fig 4.2 (a) GrayScale Image, (b) GrayScale Cropped Image

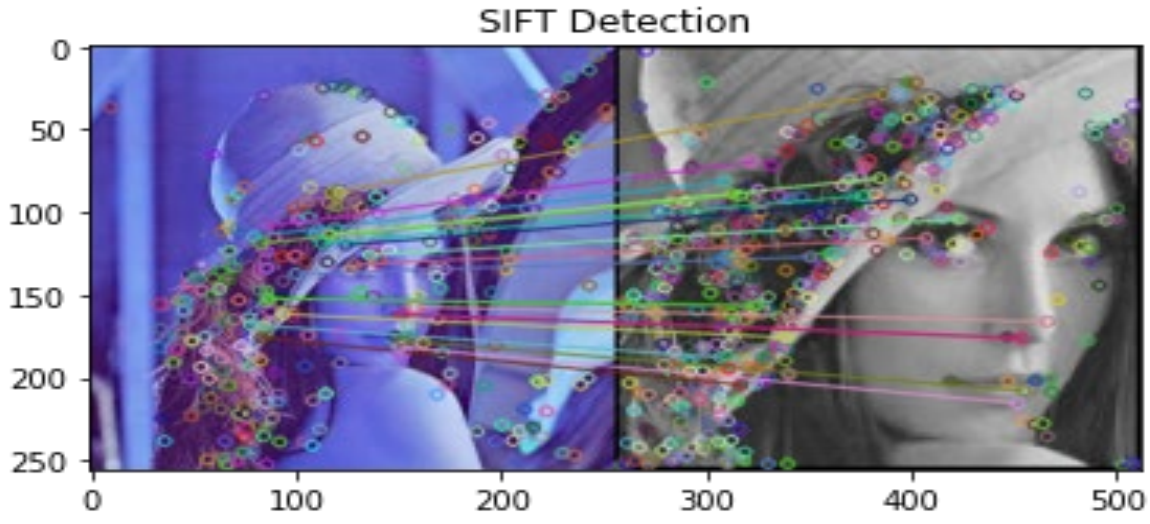
From Figure 4.2, we can observe the grayscale images obtained by converting the RGB images to grayscale.



(a) **(b)**

Fig 4.3 Results obtained by plotting identified keypoints, (a) Original image, (b) Cropped Image

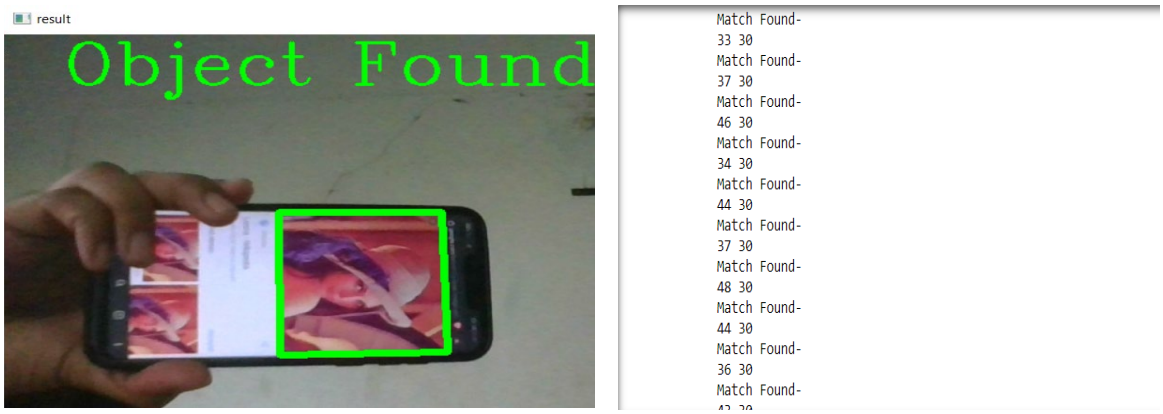
From Figure 4.3, we can observe keypoints plotted on images (a) and (b) that are obtained using SIFT algorithm.



(a) **(b)**

Fig 4.4 Results obtained after matching keypoints, (a) Original image, (b) Cropped Image

From Figure 4.4, we can observe the mapping of similar keypoints identified on images (a) and (b).



(a) **(b)**

Fig 4.5 Results obtained in python using Open CV, (a) Detection of object, (b) No. of Matched keypoints and Minimum no.of keypoints

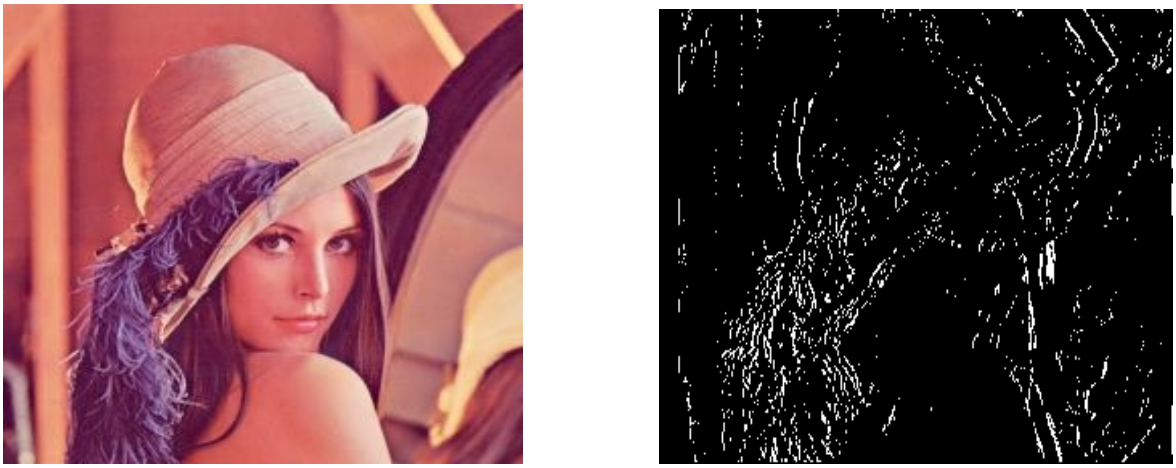
From Figure 4.5, we can observe in image (a) that the SIFT algorithm detected the rotated image of Lena and in image (b) the actual number of keypoints matched and minimum number of keypoints to be matched are displayed for each frame.

From the above results we can observe that SIFT algorithm can detect and match features of images in realtime with great accuracy. The peak signal to noise ratio (PSNR) and structural similarity index measure (SSIM) values of images are tabulated below.

S.NO	Different Images (Lena)	PSNR	SSIM
1	Cropped Image	6.1369	0.2904
2	Cropped Image with Keypoints	11.8850	0.2506

Table 4.6 PSNR and SSIM Values of Different Images (Lena) in python

RESULTS OBTAINED BY XILINX-SYSTEM-GENERATOR:



(a) **(b)**

Fig 4.7 Result of Image Keypoint Detection in Simulink

From Figure 4.7, we can observe the keypoints extracted from the image (a) using Xilinx System Generator.

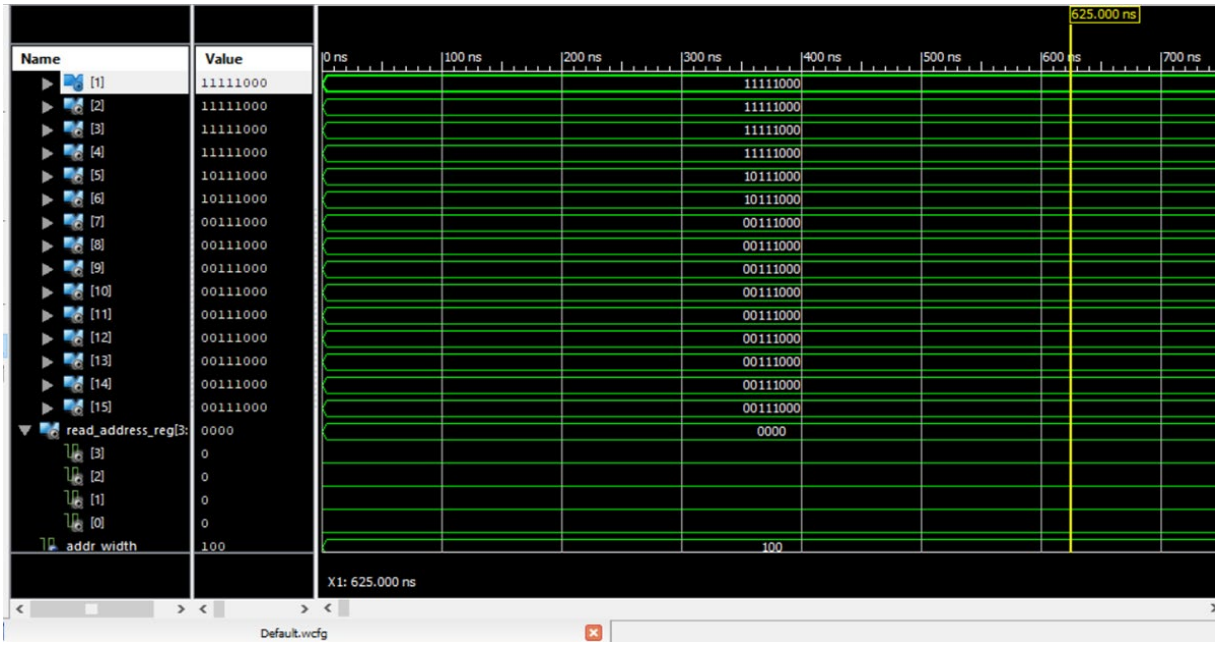


Fig 4.8 Result Obtained Using Xilinx ISE 14.7 Isim

From Figure 4.8, we can observe the simulation results obtained by executing SIFT algorithm implemented in Xilinx System Generator using Xilinx ISE 14.7. The device utilization summary obtained from the synthesis report generated by Xilinx ISE 14.7 is tabulated below.

Device Utilization Summary			
Slice Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slice Registers	2,419	32,640	7%
Number used as Flip Flops	2,419		
Number of Slice LUTs	4,119	32,640	12%
Number of occupied Slices	1,381	8,160	16%
Number of LUT Flip Flop pairs used	4,419		
Number of fully used LUT-FF pairs	2,119	4,419	47%
Number of bonded IOBs	17	480	3%
Number of BlockRAM/FIFO	88	132	66%
Number using BlockRAM only	88		
Number of 18k BlockRAM used	154		
Total Memory used (KB)	2,772	4,752	58%

Table 4.9 Utilized Hardware for the Feature detection

RESULTS OBTAINED BY USING MATLAB:

(a)

(b)

(c)

(d)

(e)

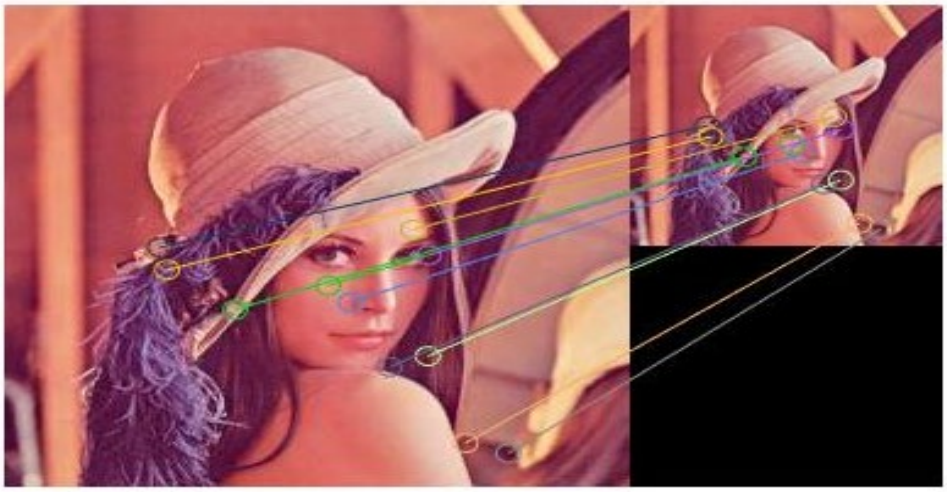
Fig 4.10 Result of Keypoints Detection, (a) original image, (b) Cropped Image, (c) Noise Image, (d) Rotated Image, (e) Scaled Image

From the above results, we can observe the keypoints extracted from the original image (a), cropped image (b), image with gaussian noise (c), rotated image (d) and scaled image (e).

(a)

(b)

(c)



(d)

Fig 4.11 Result of Feature Matching, (a) Cropped Image, (b) Noise Image, (c) Rotated Image, (d) Scaled Image

From the above results we can observe that using SIFT Algorithm we can match similar features of images even they are varied in scale (d), rotation (c), noise (b) and cropped (a). the peak signal to noise ratio (PSNR) and structural similarity index measure (SSIM) values are tabulated below.

S.NO	Different Images (Lena)	PSNR	SSIM
1	Cropped Image	15.1963	0.7925
2	Image with Gaussian Noise	27.6841	0.9702
3	Rotated Image	15.5792	0.8288
4	Scaled Image	11.9194	0.7002

Table 4.12 PSNR and SSIM Values of Different Images (Lena) in Matlab

CHAPTER 5

CONCLUSION AND FUTURE SCOPE

5.1 CONCLUSION

Feature extraction (FE) techniques have become an apparent need in many processes which have much to do with computer vision, object detection and location, image processing, image retrieval.

In this we discussed about the SIFT algorithm, it's implementation. We examined steps to be carried in SIFT algorithm for feature extraction on different software's such as MATLAB, XILINX-SYSTEM-GENERATOR and PYTHON. Feature extraction is used in object recognition. Feature extraction is a popular and useful approach in many applications and fields of study. It is observed that the applications of FE determine the types of features to be extracted and in addition, the accuracy and performance of extraction techniques are major factors of concern when performing Feature extraction.

Here we observed the criteria to extract the interest points using Laplacian of Gaussian for efficient image matching with the help of Gaussian filters. We also observed the typical values of scaling factor and σ required for extracting essential keypoints i.e., k=1.414 and σ = 1.6 and also studied about outlier rejection using difference of Gaussian.

5.2 FUTURE SCOPE

The proposed techniques for feature extraction has been done i.e, implementing SIFT for images. Depending on the interval length in the scale space keypoints are preferred in SIFT algorithm. In future a complete framework can be developed which can check the image and automatically matches the keypoints with the large database of images with less computational complexity. The same can be further implemented also for video processing.

REFERENCES

[1] Lifan Yao, Hao Feng, Yiqun Zhu, Zhiguo Jiang, Danpei Zhao, and Wenquan Feng- "An architecture of optimised SIFT Feature Detection for an FPGA Implementation of an Image Matcher", IEEE FPT, pp. 30-37, 2009.

[2] Vanderlei Bonato, Eduardo Marques and Gearge A. Constantinides "A Parallel Hardware Architecture for Scale and Rotation Invariant Feature Detection", IEEE Transactions on Circuits and Systems for Video Technology. (Vol.1 8-No12), pp.1-11,2008.

[3] Ana Brandusa Pavel and Catalin Buiu - "Development of an embedded Artificial Vision System for an Autonomous robot"-International Journal of Innovative Computing Information and Control, Volume-7 Number-2 Issue-11 February 2011 pp.745-762, 2011.

[4] David G. Lowe - "Distinctive Image Features from Scale-Invariant Keypoints"-International Journal of Computer Vision,2004.

[5] Sudipta N, Sinha, Jan-Michael Frahm, Mare Pollefeys and Yakup Genc - "Feature tracking and matching in Video using programmable graphics hardware"- Springer Veralag London Limited, Machine Vision and Apllications,2007.

[6] Kosuke Mizuno, Hiroki Noguchi, Guangji He, Yosuke Terachi, Tetsuya Kamino, Hiroshi Kawaguchi and Masahiko Yoshimoto - "Fast and Low memory bandwidth architecture of SIFT descriptor generation with scalability on speed and accuracy for VGA video", IEEE International Conference on Field Programmable Logic and applications, pp.608-611, 2010.

[7] Lakshmana Kumar.A, Dr.R.Ganeshan "Improved navigation for visually challenged with high authentication using a modified algorithm", International Journal of Advanced Research in Computer Science and Technology Vol.2 Issue Special 1, pp. 434-438, 2014.

[8] Valeriu Codreanu, Feng Dong, Baoquan Liu, Jos B.T.M.Roerdink, David Williams, Po Yang and Burhan Yasar - "GPU-ASIFT: A Fast Fully Affine- Invariant Feature Extraction Algorithm", IEEE-2013.

APPENDIX

MATLAB SOURCE CODE:

Main File :

```
clear;
clc;
close all;

% The threshold of the Euclidean distance of two feature vectors
dist_thr = 0.5;
[path1, ~] = imgetfile();
img1 = imread(path1);
img1_rotate = imrotate(img1,90);
img1_small = imresize(img1,0.5,'bicubic');
img1_crop = imcrop(img1,[20,40,360,420]);
img1_noise = imnoise(img1,'gaussian');
% [path2, ~] = imgetfile();
% img2 = imread(path2);
[descr1, loc1, ori1, scl1] = sift_features(img1,3,1.6,0.07,2);
[descr_rotate1, loc_rotate1, ori_rotate1, scl_rotate1] =
sift_features(img1_rotate,3,1.6,0.07,2);
[descr_small1, loc_small1, ori_small1, scl_small1] =
sift_features(img1_small,3,1.6,0.07,2);
[descr_crop1, loc_crop1, ori_crop1, scl_crop1] =
sift_features(img1_crop,3,1.6,0.07,2);
[descr_noise1, loc_noise1, ori_noise1, scl_noise1] =
sift_features(img1_noise,3,1.6,0.07,2);
% [descr2, loc2, ori2, scl2] = sift_features(img2,3,1.6,0.04,10);

matched_rotate = match(descr1, descr_rotate1, dist_thr);
matched_small = match(descr1, descr_small1, dist_thr);
matched_crop = match(descr1, descr_crop1, dist_thr);
matched_noise = match(descr1, descr_noise1, dist_thr);

draw_features(img1, loc1, ori1, scl1, 1);
draw_features(img1_rotate, loc_rotate1, ori_rotate1, scl_rotate1, 2);
draw_features(img1_small, loc_small1, ori_small1, scl_small1, 3);
draw_features(img1_crop, loc_crop1, ori_crop1, scl_crop1, 4);
draw_features(img1_noise, loc_noise1, ori_noise1, scl_noise1, 5);
% draw_features(img2, loc2, ori2, scl2, 4);
draw_matched(matched_rotate, img1, img1_rotate, loc1, loc_rotate1, 6);
draw_matched(matched_small, img1, img1_small, loc1, loc_small1, 7);
draw_matched(matched_crop, img1, img1_crop, loc1, loc_crop1, 8);
draw_matched(matched_noise, img1, img1_noise, loc1, loc_noise1, 9);
```

Sift Features Function:

```
function [descrs,locs,oris,scls] = sift_features(img, intvls, init_sigma,
contr_thr, curv_thr)
%SIFT_FEATURES
%%%%%%%%%%%%%%%%%%%%%%%% initialize the parameters %%%%%%%%%%%%%%%%%%%%%%%%
if ~exist('intvls', 'var') || isempty(intvls)
    intvls = 3;
end
if ~exist('init_sigma', 'var') || isempty(init_sigma)
    init_sigma = 1.6;
end
```

```
if ~exist('contr_thr', 'var') || isempty(contr_thr)
    contr_thr = 0.04;
end
if ~exist('curv_thr', 'var') || isempty(curv_thr)
    curv_thr =10;
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% transform the rgb to gray, and normalize it
if(size(img,3)==3)
    img = rgb2gray(img);
end
img = im2double(img);

% default cubic method
img = imresize(img,2);
% assume the original image has a blur of sigma = 0.5
img = gaussian(img,sqrt(init_sigma^2-0.5^2*4));
% smallest dimension of top level is about 8 pixels
octvs = floor(log(min(size(img)))/log(2) - 2);

% build Pyramid
[gauss_pyr, dog_pyr] = build_pyr(img, octvs, intvls, init_sigma);

% find the currect keypoints and remove the unstable keypoints
keypoints = get_keypoints(dog_pyr, octvs, intvls, contr_thr, curv_thr,
init_sigma);

% assign the directions for each keypoints, called features
features = get_directed_keypoints(keypoints, gauss_pyr, intvls, init_sigma);

% generate the discriptors
[descrs, locs, oris, scls] = compute_discriptors(features, keypoints, gauss_pyr);

end
```

Build Pyramid Function:

```
function [gauss_pyr, dog_pyr] = build_pyr(img, octvs, intvls, init_sigma)
%build the gauss and DOG pyramid
%   img: input image, octvs: , intvls: , init_sigma:

k = 2^(1/intvls);
sigma = ones(1,intvls+3);
sigma(1) = init_sigma;
sigma(2) = init_sigma*sqrt(k*k-1);
for i = 3:intvls+3
    sigma(i) = sigma(i-1)*k;
end

[img_height,img_width] =  size(img);
gauss_pyr = cell(octvs,1);
dog_pyr = cell(octvs,1);

octvs_size = [img_height,img_width];

% build gaussian pyramid
for i = 1:octvs
    if (i~=1)
        octvs_size = [round(octvs_size(1)/2),round(octvs_size(2)/2)];
    end
    gauss_pyr{i} = zeros(octvs_size(1),octvs_size(2),intvls+3);
```

```
    dog_pyr{i} = zeros(octvs_size(1),octvs_size(2),intvls+2);
end
for i = 1:octvs
    for j = 1:intvls+3
        if (i==1 && j==1)
            gauss_pyr{i}(:,:,j) = img;
        % downsample for the first image in an octave, from the s+1 image in
previous octave.
        elseif (j==1)
            gauss_pyr{i}(:,:,j) = imresize(gauss_pyr{i-1}(:,:,intvls+1),0.5);
        else
            gauss_pyr{i}(:,:,j) = gaussian(gauss_pyr{i}(:,:,j-1),sigma(j));
        end
    end
end

% build DOG pyramid
for i = 1:octvs
    for j = 1:intvls+2
        dog_pyr{i}(:,:,j) = gauss_pyr{i}(:,:,j+1) - gauss_pyr{i}(:,:,j);
    end
end

for i = 1:size(dog_pyr,1)
    for j = 1:size(dog_pyr{i},3)

imwrite(imbinarize(im2uint8(dog_pyr{i}(:,:,j)),0),['./dog_pyr/dog_pyr_',num2str(i)
,num2str(j),'.png']);
    end
end

end
```

Get Keypoints Function:

```
function [keypoints] = get_keypoints(dog_pyr, octvs, intvls, contr_thr, curv_thr,
init_sigma)
% find the currect keypoints and remove the unstable keypoints

sift_img_border = 5;
max_interp_steps = 5; % the maximum interpolation step
prelim_contr_thr = 0.5*contr_thr/intvls;

keypoints = struct('x',0,'y',0,'octv',0,'intvl',0,'x_hat',[0,0,0],'scl_octv',0);
keypoints_index = 1;

for i = 1:octvs
    [height, width] = size(dog_pyr{i}(:,:,1));
    dog_img_list = dog_pyr{i};
    % find extrema in middle intvls
    for j = 2:intvls+1
        dog_img = dog_img_list(:,:,j);
        for x = sift_img_border+1:height-sift_img_border
            for y = sift_img_border+1:width-sift_img_border
                % preliminary check on contrast
                if(abs(dog_img(x,y)) > prelim_contr_thr)
                    % check 26 neighboring pixels
                    if(isExtremum(dog_img_list,j,x,y))
                        ddata =
interp_extremum(dog_img_list,height,width,i,j,x,y,sift_img_border,contr_thr,max_in
terp_steps, intvls, init_sigma);
                        if(~isempty(ddata))
```

```
if(~is_too_edge_like(dog_img,ddata.x,ddata.y,curv_thr))
                                    keypoints(keypoints_index) = ddata;
                                    keypoints_index = keypoints_index + 1;
                                end
                            end
                        end
                    end
                end
            end
        end

end
```

Get Directed Keypoints Function:

```
function [features] = get_directed_keypoints(keypoints, gauss_pyr, intvls,
init_sigma)
%GET_DIRECTED_KEYPOINTS

% determines gaussian sigma for orientation assignment
sift_ori_sig_factr = 1.5;
% number of bins in histogram
sift_ori_hist_bins = 36;
% orientation magnitude relative to max that results in new feature
sift_ori_peak_ratio = 0.8;
% number of passes of orientation histogram smoothing
sift_ori_smooth_passes = 2;
% array of feature
features = struct('ddata_index',0,'x',0,'y',0,'scl',0,'ori',0,'descr',[]);
feat_index = 1;

n = size(keypoints,2);

for i = 1:n
    ddata = keypoints(i);
    ori_sigma = sift_ori_sig_factr * ddata.scl_octv;
    % generate a histogram for the gradient distribution around a keypoint
    hist =
ori_hist(gauss_pyr{ddata.octv}(:,:,ddata.intvl),ddata.x,ddata.y,sift_ori_hist_bins
,round(3*ori_sigma),ori_sigma);
    for j = 1:sift_ori_smooth_passes
        hist = smooth_ori_hist(hist,sift_ori_hist_bins);
    end
    hist_max = max(hist); % this is the dominant orientation
    % generate feature from ddata and orientation hist peak
    % add orientations greater than or equal to 80% of the largest orientation
magnitude
    [features, feat_index] =
add_good_ori_features(i,features,feat_index,ddata,hist,sift_ori_hist_bins,hist_max
*sift_ori_peak_ratio,intvls,init_sigma);
end
end
```

Compute Descriptors Function :

```
function [descrs, locs, oris, scls] = compute_discriptors(features, keypoints,
gauss_pyr)
%COMPUTE_DISCRIPTORS

% number of features
n = size(features,2);
```

```
% width of 2d array of orientation histograms
sift_descr_width = 4;
% bins per orientation histogram
sift_descr_hist_bins = 8;
% threshold on magnitude of elements of descriptor vector
sift_descr_mag_thr = 0.2;

descr_length = sift_descr_width * sift_descr_width * sift_descr_hist_bins;

for feat_index = 1:n
    feat = features(feat_index);
    ddata = keypoints(feat.ddata_index);
    gauss_img = gauss_pyr{ddata.octv}(:,:,ddata.intvl);
    hist = descr_hist(gauss_img, ddata.x, ddata.y, feat.ori, ddata.scl_octv,
sift_descr_width, sift_descr_hist_bins, descr_length);
    features(feat_index) = hist_to_descr(feat,hist,sift_descr_mag_thr);
end

% sort the descriptors by descending scale order
features_scl = [features.scl];
[~,features_order] = sort(features_scl,'descend');
% return descriptors and locations
descrs = zeros(n,descr_length);
locs = zeros(n,2);
oris = zeros(n,1);
scls = zeros(n,1);
for i = 1:n
    descrs(i,:) = features(features_order(i)).descr;
    locs(i,1) = features(features_order(i)).x;
    locs(i,2) = features(features_order(i)).y;
    oris(i) = features(features_order(i)).ori;
    scls(i) = features(features_order(i)).scl;
end

end
```

Good Orientation Features Function :

```
function [features, feat_index] =
add_good_ori_features(ddata_index,features,feat_index,ddata,hist,n,mag_thr,intvls,
init_sigma)

for i = 1:n
    if (i==1)
        l = n;
        r = 2;
    elseif (i==n)
        l = n-1;
        r = 1;
    else
        l = i-1;
        r = i+1;
    end
    if ( hist(i) > hist(l) && hist(i) > hist(r) && hist(i) >= mag_thr )
        interp_hist_peak = 0.5*(hist(l)-hist(r))/(hist(l)-2*hist(i)+hist(r));
        bin = i + interp_hist_peak;
        if ( bin < 1 )
            bin = bin + n;
        elseif ( bin > n)
            bin = bin - n;
        end
        accu_intvl = ddata.intvl + ddata.x_hat(3);
        features(feat_index).ddata_index = ddata_index;
```

```
        % first octave is double size
        features(feat_index).x = (ddata.x + ddata.x_hat(1))*2^(ddata.octv-2);
        features(feat_index).y = (ddata.y + ddata.x_hat(2))*2^(ddata.octv-2);
        features(feat_index).scl = init_sigma * power(2,ddata.octv-2 +
(accu_intvl-1)/intvls);
        features(feat_index).ori = 2*pi*(bin-1)/n - pi;
        feat_index = feat_index + 1;
    end
end
end
```

Calculate Magnitude, Orientation Function:

```
function [mag, ori, flag] = calc_grad_mag_ori(img,x,y)

[height,width] = size(img);
if (x > 1 && x < height && y > 1 && y < width)
    dx = img(x,y+1) - img(x,y-1);
    dy = img(x+1,y) - img(x-1,y);
    mag = sqrt(dx^2+dy^2);
    ori = atan2(dx,dy);
    flag = 1;
else
    mag = -1;
    ori = -1;
    flag = 0;
end
```

Calculate dx,dy,ds Function:

```
function [result] = deriv(dog_img_list, intvl, x, y)
%DERIV
dx = (dog_img_list(x+1,y,intvl) - dog_img_list(x-1,y,intvl))/2;
dy = (dog_img_list(x,y+1,intvl) - dog_img_list(x,y-1,intvl))/2;
ds = (dog_img_list(x,y,intvl+1) - dog_img_list(x,y,intvl-1))/2;
result = [dx,dy,ds]';
end
```

Descriptor Histogram Function:

```
function [hist] = descr_hist(gauss_img, ddata_x, ddata_y, feat_ori, scl_octv, d,
n, descr_length)
%DESCR_HIST

% determines the size of a single descriptor orientation histogram
sift_descr_scl_fctr = 3.0;

hist_width = sift_descr_scl_fctr * scl_octv;
radius = round( hist_width * (d + 1) * sqrt(2) / 2 );
hist = zeros(1,descr_length);
for i = -radius:radius
    for j = -radius:radius
        j_rot = j*cos(feat_ori) - i*sin(feat_ori);
        i_rot = j*sin(feat_ori) + i*cos(feat_ori);
        r_bin = i_rot/hist_width + d/2 - 0.5;
        c_bin = j_rot/hist_width + d/2 - 0.5;
        if (r_bin > -1 && r_bin < d && c_bin > -1 && c_bin < d)
            [mag, ori, flag] = calc_grad_mag_ori(gauss_img,ddata_x+i,ddata_y+j);
            if (flag == 1)
                ori = ori - feat_ori;
                while (ori < 0)
                    ori = ori + 2*pi;
                end
```

```
                while (ori >= 2*pi)
                    ori = ori - 2*pi;
                end
                o_bin = ori * n / (2*pi);
                w = exp( -(j_rot*j_rot+i_rot*i_rot) / (2*(0.5*d*hist_width)^2) );
                hist = interp_hist_entry(hist,r_bin,c_bin,o_bin,mag*w,d,n);
            end
        end
    end
end

end
```

Draw Features Function:

```
function [] = draw_features(img,loc,ori,scl,figure_num)
%DRAW_FEATURES
figure(figure_num);
imshow(img);
hold on;
[point_num,~] = size(loc);
for i = 1:point_num
    color = rand(3,1);
    viscircles([loc(i,2) loc(i,1)],scl(i),'Color',color,'LineWidth',0.5);
    x1 = loc(i,2) + scl(i)*sin(ori(i));
    y1 = loc(i,1) + scl(i)*cos(ori(i));
    plot([loc(i,2) x1], [loc(i,1) y1], 'Color',color,'LineWidth',0.5);
end

end
```

Draw Matches Function:

```
function [] = draw_matched(matched, img1, img2, loc1, loc2, figure_num)
%DRAW_MATCHED
figure(figure_num);

size1 = size(img1);
size2 = size(img2);
if (size1(1) < size2(1))
    img1(size2(1),:) = 0;
elseif (size1(1) > size2(1))
    img2(size1(1),:) = 0;
end
merge_img = [img1 img2];
imshow(merge_img);
hold on;

[~,n] = size(matched);
for i = 1:n
    color = rand(1,3);
    plot(loc1(matched(1,i),2),loc1(matched(1,i),1),'o','Color',color);
    plot(loc2(matched(2,i),2)+size1(1),loc2(matched(2,i),1),'o','Color',color);
    plot([loc1(matched(1,i),2) loc2(matched(2,i),2)+size1(1)],
[loc1(matched(1,i),1) loc2(matched(2,i),1)], 'Color',color,'LineWidth',0.5);
end
end
```

Gaussian Function:

```
function [ out_img ] = gaussian( img, sigma )
% Function: Gaussian smooth for an image
k = 3;
hsize = round(2*k*sigma+1);
if mod(hsize,2) == 0
    hsize = hsize+1;
end
g = fspecial('gaussian',hsize,sigma);
out_img = conv2(img,g,'same');
end
```

Hessian Function:

```
function [result] = hessian(dog_img_list, intvl, x, y)
%HESSIAN
center = dog_img_list(x,y,intvl);
dxx = dog_img_list(x+1,y,intvl) + dog_img_list(x-1,y,intvl) - 2*center;
dyy = dog_img_list(x,y+1,intvl) + dog_img_list(x,y-1,intvl) - 2*center;
dss = dog_img_list(x,y,intvl+1) + dog_img_list(x,y,intvl-1) - 2*center;

dxy = (dog_img_list(x+1,y+1,intvl)+dog_img_list(x-1,y-1,intvl)-dog_img_list(x+1,y-
1,intvl)-dog_img_list(x-1,y+1,intvl))/4;
dxs = (dog_img_list(x+1,y,intvl+1)+dog_img_list(x-1,y,intvl-1)-
dog_img_list(x+1,y,intvl-1)-dog_img_list(x-1,y,intvl+1))/4;
dys = (dog_img_list(x,y+1,intvl+1)+dog_img_list(x,y-1,intvl-1)-dog_img_list(x,y-
1,intvl+1)-dog_img_list(x,y+1,intvl-1))/4;

result = [dxx,dxy,dxs;dxy,dyy,dys;dxs,dys,dss];
end
```

Histogram to Descriptor Function:

```
function [feat] = hist_to_descr(feat,descr,descr_mag_thr)
%HIST_TO_DESCR

descr = descr/norm(descr);
descr = min(descr_mag_thr,descr);
descr = descr/norm(descr);
feat.descr = descr;
end
```

Extremum Function:

```
function [ddata] = interp_extremum(dog_img_list, height, width, octv, intvl, x, y,
img_border, contr_thr, max_interp_steps, intvls, init_sigma)
%INTERP_EXTREMUM

i = 1;
while (i <= max_interp_steps)
    df = deriv(dog_img_list,intvl,x,y);
    H = hessian(dog_img_list,intvl,x,y);
    [U,S,V] = svd(H);
    T=S;
    T(S~=0) = 1./S(S~=0);
    svd_inv_H = V * T' * U';
    x_hat = - svd_inv_H*df;
    if( abs(x_hat(1)) < 0.5 && abs(x_hat(2)) < 0.5 && abs(x_hat(3)) < 0.5)
        break;
    end
    x = x + round(x_hat(1));
```

```
    y = y + round(x_hat(2));
    intvl = intvl + round(x_hat(3));
    if (intvl < 2 || intvl > intvls+1 || x <= img_border || y <= img_border || x >
height-img_border || y > width-img_border)
        ddata = [];
        return;
    end
    i = i+1;
end
if (i > max_interp_steps)
    ddata = [];
    return;
end
contr = dog_img_list(x,y,intvl) + 0.5*df'*x_hat;
if ( abs(contr) < contr_thr/intvls )
    ddata = [];
    return;
end
ddata.x = x;
ddata.y = y;
ddata.octv = octv;
ddata.intvl = intvl;
ddata.x_hat = x_hat;
ddata.scl_octv = init_sigma * power(2,(intvl+x_hat(3)-1)/intvls);

end
```

Histogram Entry Function:

```
function [hist] = interp_hist_entry(hist,r,c,o,m,d,obins)
%INTERP_HIST_ENTRY

r0 = floor(r);
c0 = floor(c);
o0 = floor(o);
d_r = r - r0;
d_c = c - c0;
d_o = o - o0;

for i = 0:1
    r_index = r0 + i;
    if (r_index >= 0 && r_index < d)
        for j = 0:1
            c_index = c0 + j;
            if (c_index >=0 && c_index < d)
                for k = 0:1
                    o_index = mod(o0+k,obins);
                    value = m * ( 0.5 + (d_r - 0.5)*(2*i-1) ) * ( 0.5 + (d_c -
0.5)*(2*j-1) ) * ( 0.5 + (d_o - 0.5)*(2*k-1) );
                    hist_index = r_index*d*obins + c_index*obins + o_index +1;
                    hist(hist_index) = hist(hist_index) + value;
                end
            end
        end
    end
end

end
```

Valid Keypoint Function:

```
function [result] = is_too_edge_like(dog_img, x, y, curv_thr)
%IS_TOO_EDGE_LIKE

d = dog_img(x,y);
dxx = dog_img(x,y+1) + dog_img(x,y-1) - 2*d;
dyy = dog_img(x+1,y) + dog_img(x-1,y) - 2*d;
dxy = (dog_img(x+1,y+1) - dog_img(x+1,y-1) - dog_img(x-1,y+1) + dog_img(x-1,y-1))
/ 4;
tr = dxx + dyy;
det = dxx * dyy - dxy * dxy;
if ( det <= 0 )
    result = 1;
    return;
end
if ( tr*tr / det < (curv_thr + 1)^2 / curv_thr )
    result = 0;
else
    result = 1;
end
end
```

Is Extremum Function:

```
function [result] = isExtremum(dog_img_list, intvl, x, y)
%ISEXTREMUM
value = dog_img_list(x,y,intvl);
block = dog_img_list(x-1:x+1,y-1:y+1,intvl-1:intvl+1);
if ( value > 0 && value == max(block(:)) )
    result = 1;
elseif ( value == min(block(:)) )
    result = 1;
else
    result = 0;
end

end
```

Match Function:

```
function [matched] = match(descr1,descr2,dist_thr)
%MATCH

[n1, dim1] = size(descr1);
[n2, dim2] = size(descr2);

if (dim1~=dim2)
    disp('Two dimensions should be equal!');
    matched = -1;
    return
else
    matched = zeros(2,min(n1,n2));
    index = 1;
    for i = 1:n1
        min_distance = sqrt(sum((descr1(i,:)-descr2(1,:)).^2));
        min_loc2 = 1;
        for j = 2:n2
            s = sqrt(sum((descr1(i,:)-descr2(j,:)).^2));
            if (s<min_distance)
                min_distance = s;
                min_loc2 = j;
```

```
            end
        end
        if (min_distance<dist_thr)
            matched(1,index) = i;
            matched(2,index) = min_loc2;
            index = index + 1;
        end
    end
end
matched(:,find(sum(abs(matched),1)==0))=[];
end
```

Orientation Histogram Function:

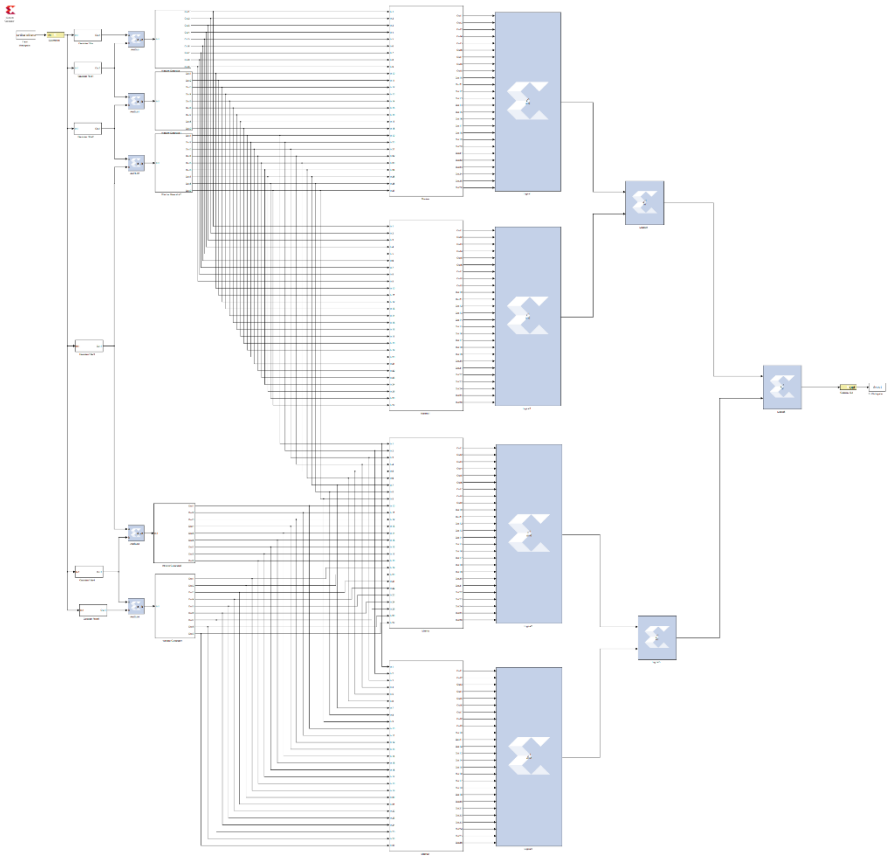
```
function [hist] = ori_hist(img, x, y, n, rad, sigma)
%ORI_HIST

hist = zeros(n, 1);
exp_denom = 2.0 * sigma * sigma;
for i = -rad:rad
    for j = -rad:rad
        [mag, ori, flag] = calc_grad_mag_ori(img,x+i,y+j);
        if (flag==1)
            w = exp(-(i^2+j^2)/exp_denom);
            bin = 1 + round(n * (ori+pi) / (2*pi));
            if (bin>=n+1)
                bin = 1;
            end
            hist(bin) = hist(bin) + w * mag;
        end
    end
end
end
```

Smooth Orientation Histogram Function:

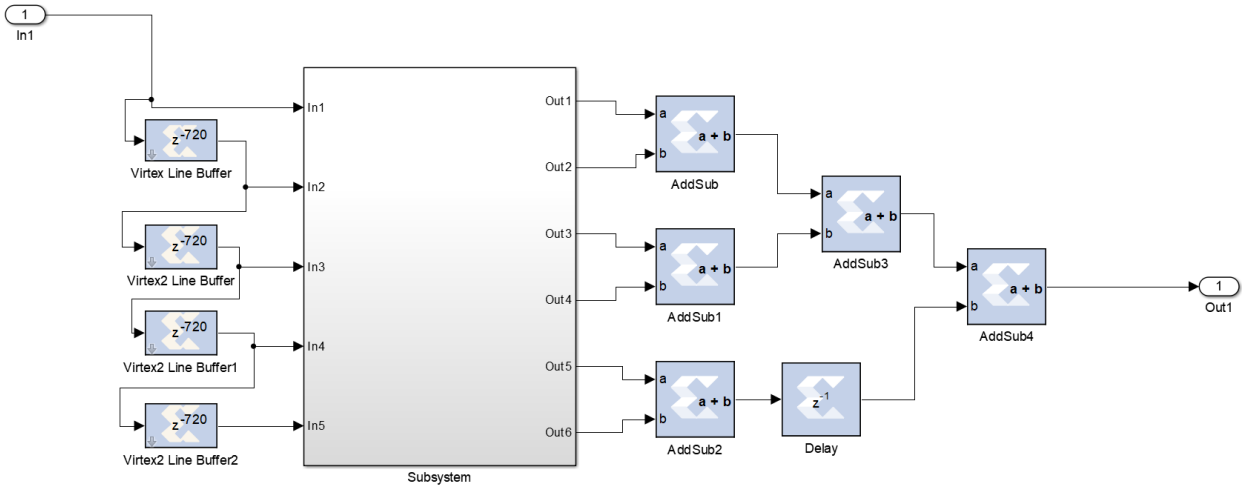
```
function [hist] = smooth_ori_hist(hist,n)
%SMOOTH_ORI_HIST
h0 = hist(1);
prev = hist(n);
for i = 1:n
    tmp = hist(i);
    if (i==n)
        hist(i) = 0.25*prev + 0.5*hist(i) + 0.25*h0;
    else
        hist(i) = 0.25*prev + 0.5*hist(i) + 0.25*hist(i+1);
    end
    prev = tmp;
end
end
```

SYSGEN IMPLEMENTATION:

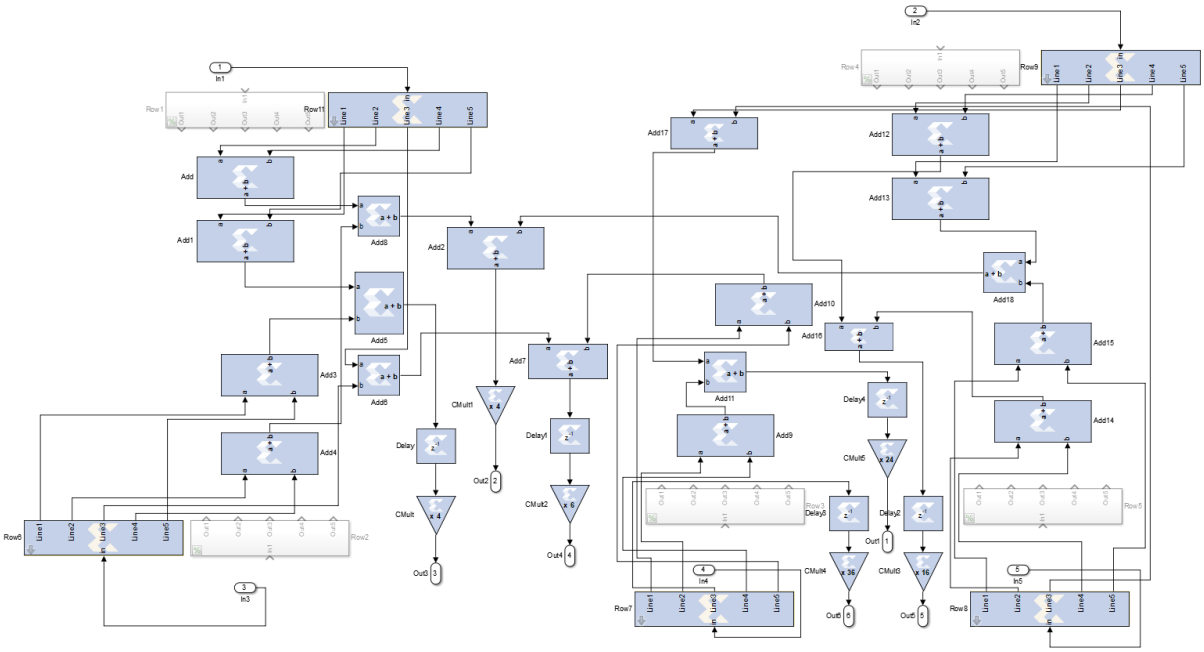
SIFT:



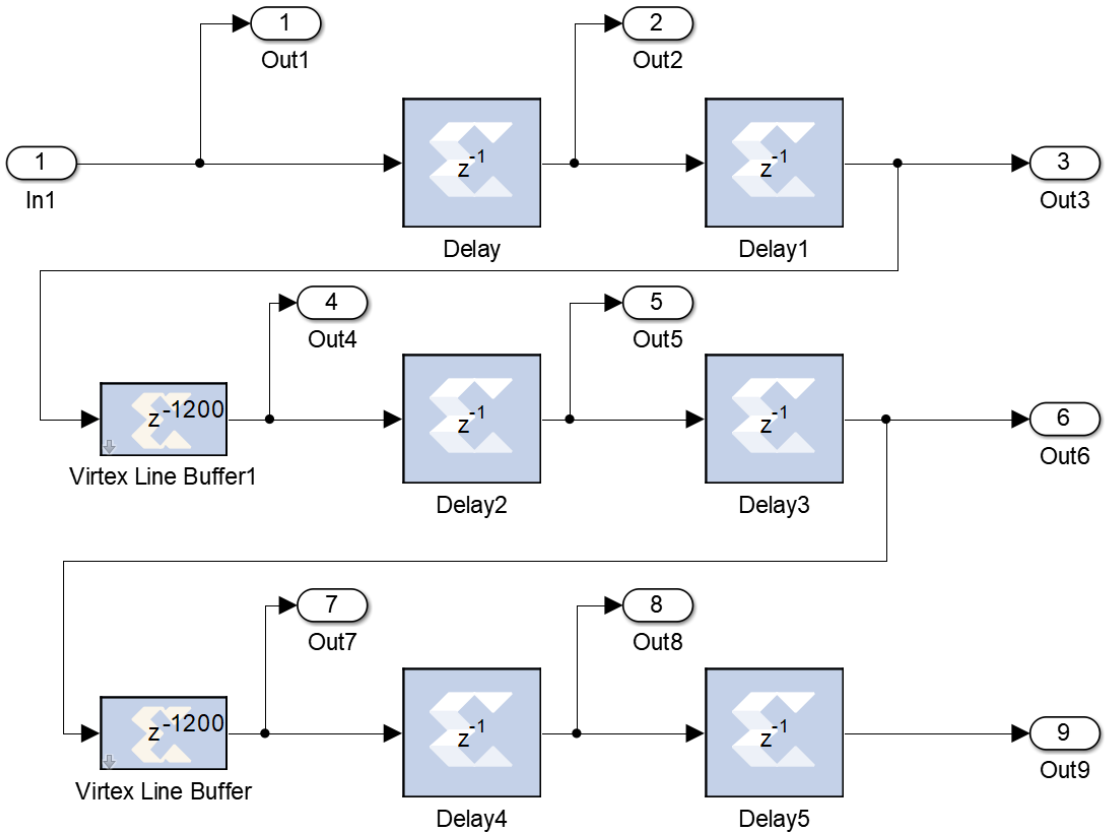
Gaussian Filter:



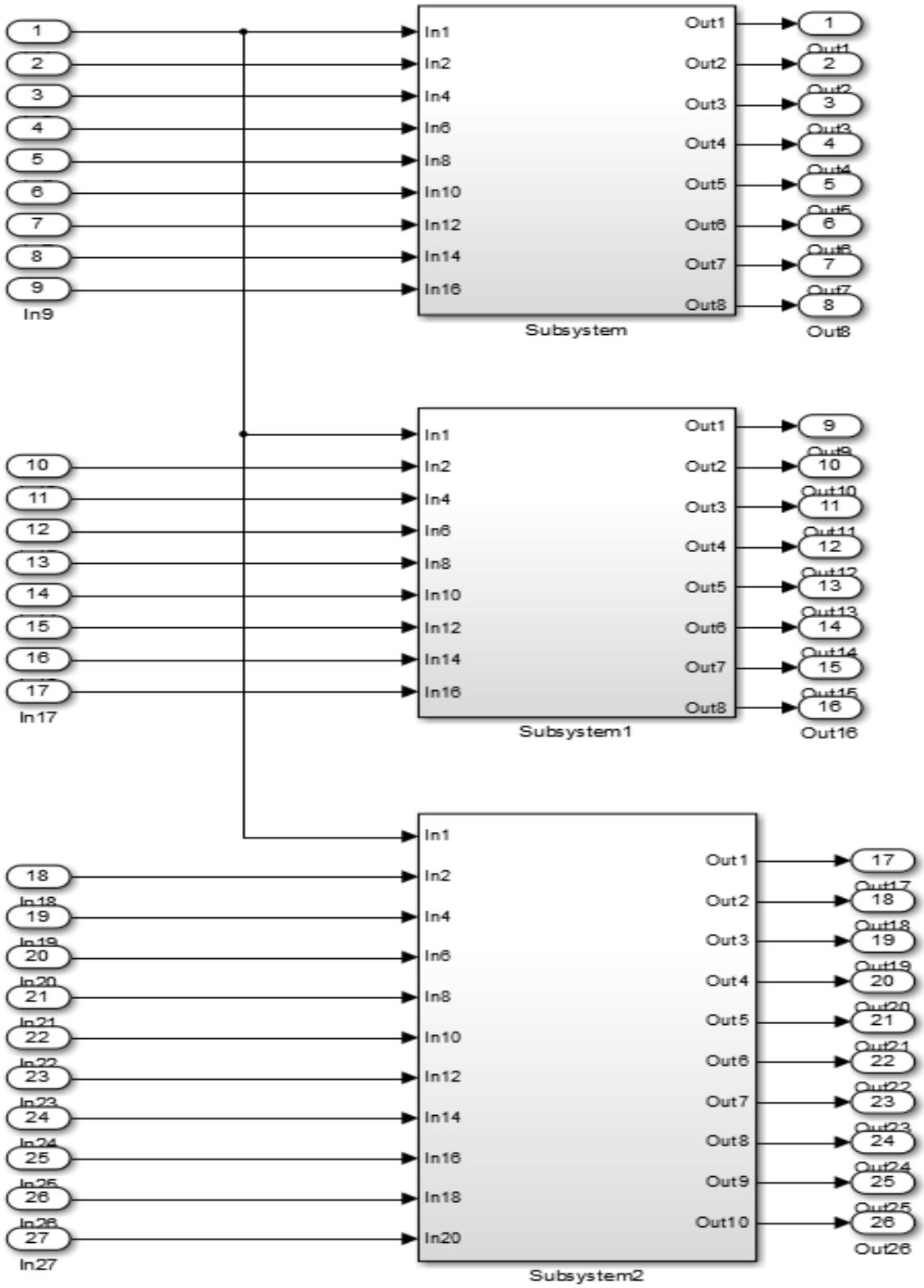
Gaussian Filter Subsystem:



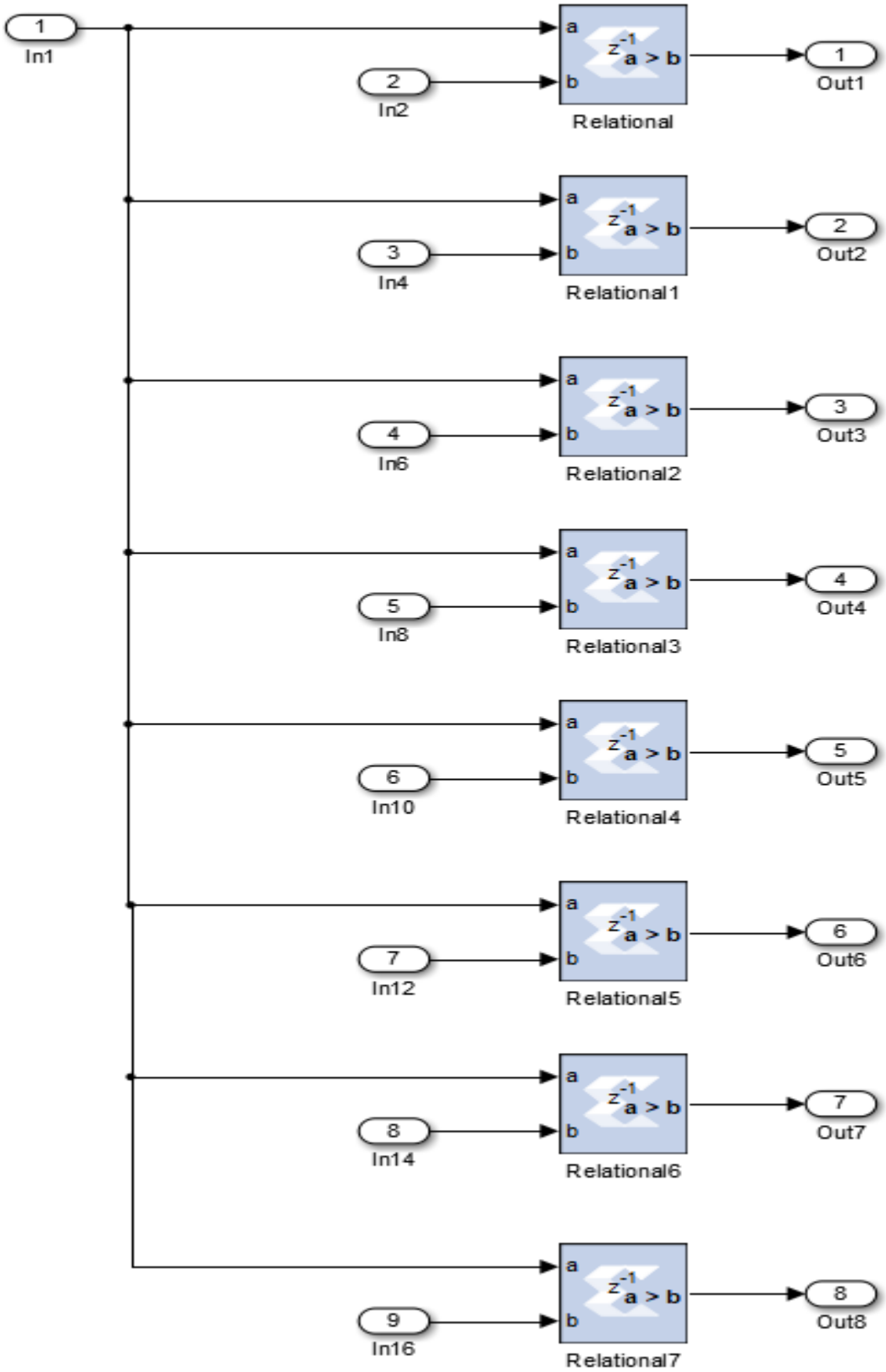
Window Generator:



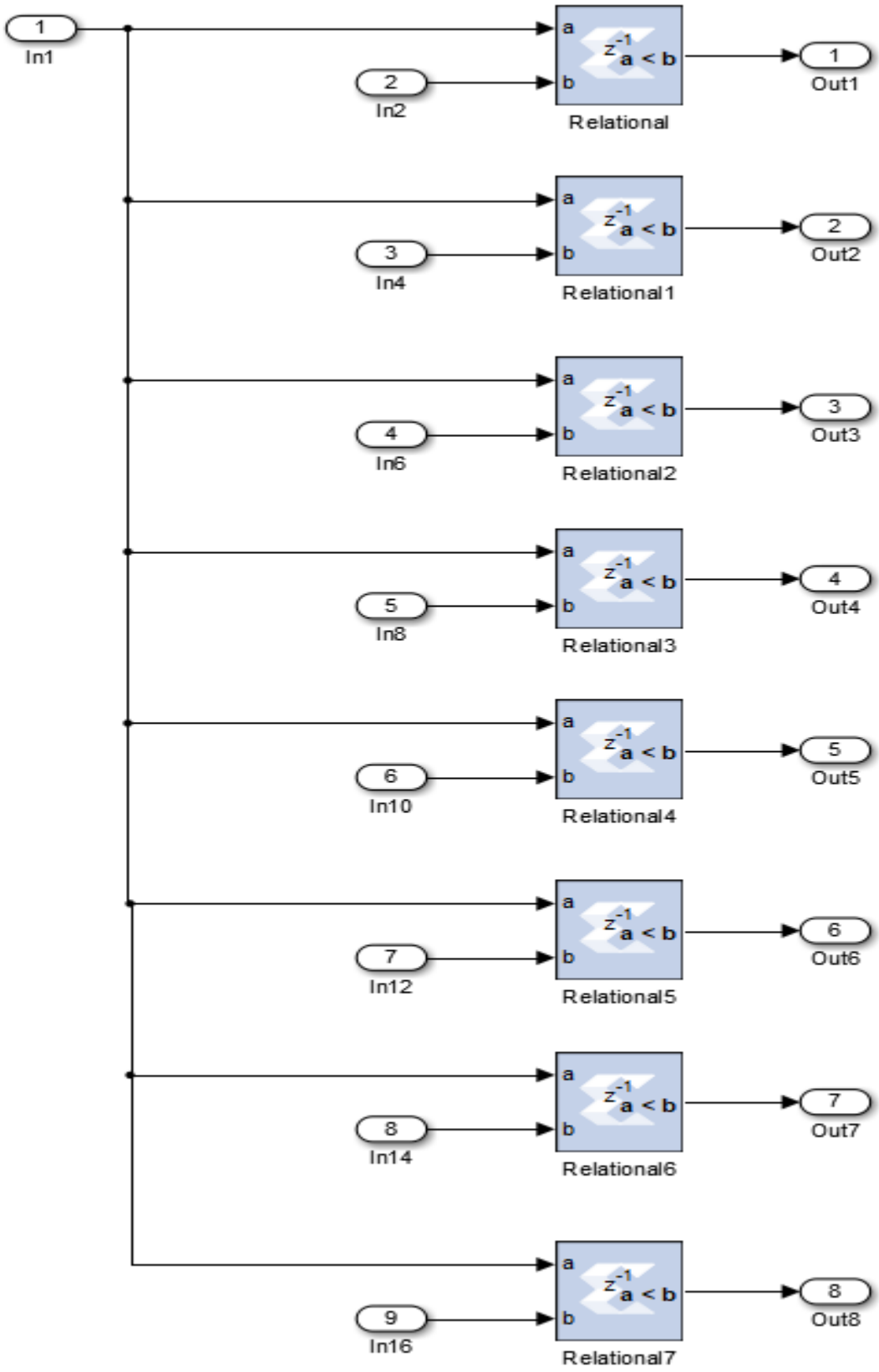
Maxima / Minima:



Maxima Subsystem:



Minima Subsystem:



PYTHON SOURCE CODE:

Keypoint Matching :

```
#importing required libraries
import cv2
import numpy as np
import matplotlib.pyplot as plt

#reading test image
train_img = cv2.imread('./lenatrain.jpg')

#reading train image
test_img = cv2.imread('./lenatest.jpg')

#function to plot signed 32bit signed matrix images side by side
def plot_32bs_images(img1, img2, title1="", title2=""):
    fig = plt.figure(figsize=[15, 15])
    axis1 = fig.add_subplot(121)
    axis1.imshow(cv2.cvtColor(img1, cv2.CV_32S))
    axis1.set(title=title1)
    axis2 = fig.add_subplot(122)
    axis2.imshow(cv2.cvtColor(img2, cv2.CV_32S))
    axis2.set(title=title2)


#function to plot gray images side by side
def plot_images(img1, img2, title1="", title2=""):
    fig = plt.figure(figsize=[15, 15])
    axis1 = fig.add_subplot(121)
    axis1.imshow(img1, cmap="gray")
    axis1.set(title=title1)
    axis2 = fig.add_subplot(122)
    axis2.imshow(img2, cmap="gray")
    axis2.set(title=title2)

# Show Original Images
plot_32bs_images(train_img, test_img, "Train Image", "Test Image")

# changing Images to grayscale
train_gray_img = cv2.cvtColor(train_img, cv2.COLOR_BGR2GRAY)
test_gray_img = cv2.cvtColor(test_img, cv2.COLOR_BGR2GRAY)

#plot grayscale images
plot_images(train_gray_img, test_gray_img, 'Train Image - Gray',
'Test Image - Gray')

# Initialise Open CV SIFT detector
sift = cv2.SIFT_create()

#function to get keypoints and descriptors
def get_KP_DESC(img):
    return sift.detectAndCompute(img, None)

# Generate SIFT keypoints and descriptors
train_kp, train_desc = get_KP_DESC(train_gray_img)
test_kp, test_desc = get_KP_DESC(test_gray_img)

#function to draw keypoints on image
def draw_KP(gray_img_1, kp1, orig_img_1, title1, gray_img_2, kp2,
orig_img_2, title2):
    img1 = cv2.drawKeypoints(gray_img_1, kp1, orig_img_1.copy())
```

```
    img2 = cv2.drawKeypoints(gray_img_2, kp2, orig_img_2.copy())
    fig = plt.figure(figsize=[15, 15])
    axis1 = fig.add_subplot(121)
    axis1.imshow(img1)
    axis1.set(title=title1)
    axis2 = fig.add_subplot(122)
    axis2.imshow(img2)
    axis2.set(title=title2)

#draw detected keypoints on images
draw_KP(train_gray_img, train_kp, train_img, 'Train Image Keypoints',
test_gray_img, test_kp, test_img, 'Test Image Keypoints')

# create a Brute Force Matcher object which will match the SIFT features
brute_force = cv2.BFMatcher(cv2.NORM_L2, crossCheck=True)

matches = brute_force.match(train_desc, test_desc)

# Sort the matches in the order of their distance in ascending order.
matches = sorted(matches, key = lambda x:x.distance)

# draw the top N matches
N_MATCHES = 20

matched_img = cv2.drawMatches(train_img, train_kp, test_img, test_kp,
matches[:N_MATCHES], test_img.copy(), flags=0)

#Plotting matched image
plt.figure()
plt.imshow(matched_img)
plt.title('SIFT Detection')
plt.show()
```

Live Detection :

```
import time
import cv2
import numpy as np
import matplotlib.pyplot as plt

# Threshold
MIN_MATCH_COUNT=30

# Initiate SIFT detector
sift=cv2.SIFT_create()

# Create the Flann Matcher object
FLANN_INDEX_KDITREE=0
flannParam=dict(algorithm=FLANN_INDEX_KDITREE,tree=5)
flann=cv2.FlannBasedMatcher(flannParam,{})

# train image
train_img= cv2.imread("./res/lenatrain.jpg",0)

# find the keypoints and descriptors with SIFT
kp1,desc1= sift.detectAndCompute(train_img,None)

# draw keypoints of the train image
train_img_kp= cv2.drawKeypoints(train_img,kp1,None,(255,0,0),4)
plt.imshow(train_img_kp)    # show the train image keypoints
plt.title('Train Image Keypoints')
```

```
plt.show()

# start capturing video
cap = cv2.VideoCapture(0)

# turn the frame captured into grayscale
while True:
    ret, frame = cap.read()
    gray = cv2.cvtColor(frame,cv2.COLOR_BGR2GRAY)
    kp2, desc2 = sift.detectAndCompute(gray,None)

   # find the keypoints and descriptors with SIFT  of the frame captured
   # Obtain matches using K-Nearest Neighbor Method
   # the result 'matches' is the number of similar matches found in both images
    matches=flann.knnMatch(desc2,desc1,k=2)


    # store all the good matches as per Lowe's ratio test.
    goodMatch=[]
    for m,n in matches:
        if(m.distance<0.75*n.distance):
            goodMatch.append(m)


    # If enough matches are found, we extract the locations of matched keypoints
    # in both the images.
    # They are passed to find the perpective transformation.
    # Once we get this 3x3 transformation matrix, we use it to transform the
corners
    # of query image to corresponding points in train image. Then we draw it.

    if(len(goodMatch)>MIN_MATCH_COUNT):
        tp=[]  # src_pts
        qp=[]  # dst_pts
        for m in goodMatch:
            tp.append(kp1[m.trainIdx].pt)
            qp.append(kp2[m.queryIdx].pt)
        tp,qp=np.float32((tp,qp))

        H,status=cv2.findHomography(tp,qp,cv2.RANSAC,3.0)


        h,w = train_img.shape
        train_outline= np.float32([[[0,0],[0,h-1],[w-1,h-1],[w-1,0]]])
        query_outline = cv2.perspectiveTransform(train_outline,H)

        cv2.polylines(frame,[np.int32(query_outline)],True,(0,255,0),5)
        cv2.putText(frame,'Object Found',(50,50), cv2.FONT_HERSHEY_COMPLEX, 2
,(0,255,0), 2)
        print("Match Found-")
        print(len(goodMatch),MIN_MATCH_COUNT)

    else:
        print("Not Enough match found-")
        print(len(goodMatch),MIN_MATCH_COUNT)
    cv2.imshow('result',frame)

    if cv2.waitKey(1) == 13:
        break
cap.release()
cv2.destroyAllWindows()
```