A STUDY OF OBJECT RECOGNITION IN TRANSFORMATIONAL SCENES

Himanshu Aggarwal

Bachelors of Computer Science
Professor PD Sharma

May 30, 2017

ABSTRACT

Object recognition is one of the most initial problems in computer vision which is still being solved. It becomes more difficult to recognise objects if the object is undergoing transformations. In the past couple of decades, several algorithms have been developed to recognise objects, which are invariant to transformations. This study is intended to give valuable knowledge to those who wants to advance to this particular domain, discuss various possible approaches to this problem, and also to shed some light on new areas in this domain.

CERTIFICATE

This is to certify that the thesis entitled '***A Study of Object Recognition in Transformational Scenes***' is the bonafide record of the project work carried out by ***Himanshu Aggarwal*** at ***SGTB Khalsa College, University of Delhi*** in Semester VII of ***Bachelors of Technology (Computer Science)*** degree. This project has not been submitted to any other University/Institute for award of any other degree/diploma.

Himanshu Aggarwal

Prof. P.D. Sharma
(Project Guide)
Associate Professor
SGTB Khalsa College
University of Delhi

Dr. Jaswinder Singh
Principal
SGTB Khalsa College
University of Delhi

ACKNOWLEDGEMENT

I take this opportunity with much pleasure to thank all the people who have helped me through the course of my journey towards producing this thesis.

I sincerely thank Mr PD Sharma, my project guide, for his supervision, cooperation and motivation throughout the project work. He cheered me at every step and took the pain of resolving my doubts, big or small, with the same sincerity. He has been a source of inspiration for me. His insight into the academic subjects and perspective towards things is what has encouraged me to pursue this research project under him.

I would like to extend my regards to SGTB Khalsa College for providing me all the infrastructural facilities and tools for the entire duration of my project.

Himanshu Aggarwal

TABLE OF CONTENTS

1. INTRODUCTION

Object recognition has gained a lot of progress in past few decades. Most of the algorithms developed so far captures invariant local features of the object detected. Transformational scenes of the objects to be recognised, make the task more difficult.

This report deals with the implementation of several algorithms to solve this problem and also discusses new findings out of one such algorithm.

1.1 Computer Vision

A human brain makes reasonable assumptions based on experiences and makes a plausible description of the world for us. The brain takes in as input the images through eyes and gives the output of all the available information and all the interpretations that can be made according to the learnings it has done in last so many years. Similar is the way we want computer machines to act. We want to feed the machines with images, videos, data, and want machines to process them and give as output the information that is visible and also that is not visible or can be interpreted. We want the computers to see. This is what computer vision is.

In literature, it is defined [4] as a subfield of Artificial intelligence, which deals with the transformation of data from a still or video camera into either a decision or a new representation. All such transformations are done for achieving some particular goal. These goals can be detecting or recognising objects or persons, tracking some activity, or reconstructing a three-dimensional structure of a scene out of multiple images.

Computer vision is being used today in a wide variety of real-world applications, which include:

- Optical Character recognition
- Machine inspection
- 3D model building
- Face detection and Visual authentication, and many more.

1.2 Problems with Computer Vision

At present, computers do not fully understand what they see. This level of comprehension is still faraway goals for computers, as the ability to understand an image is not just to collect some pixels, but to interpret something incredible out of it.
Many computer vision problems are difficult to specify, especially because the information its lost between transformation from 3D world to 2D image. The complexity of computer vision is aggravated by the fact that we are dealing with a large amount of data. A typical greyscale image has 320x240 pixels, each with 8 intensity values, so the size of the whole image is 320x240 = 614400 bits.

In computer vision, we are trying to describe the world that we see in one or more images and to reconstruct its properties, such as shape, illumination, and color distributions. It is amazing that humans and animals do this so effortlessly, while computer vision algorithms are so error prone. People who have not worked in the field often under- estimate the difficulty of the problem.

1.3 OpenCV : Open Source Library

OpenCV[1] is an open source computer vision and machine learning software library. OpenCV was built to provide a common infrastructure for computer vision applications and to accelerate the use of machine perception in the commercial products. Being a BSD-licensed product, OpenCV makes it easy for businesses to utilise and modify the code.

It has C++, C, Python, Java and MATLAB interfaces and supports Windows, Linux, Android and Mac OS. OpenCV leans mostly towards real-time vision applications and takes advantage of MMX and SSE instructions when available.There are over 500 algorithms and about 10 times as many functions that compose or support those algorithms. OpenCV is written natively in C++ and has a templated interface that works seamlessly with STL containers.

[1] Open Source Computer Vision Library: Documentation, Downloads and Code is available at www.opencv.org

1.3.1 Why OpenCV?

An alternative to OpenCV is Matlab[2]. But following are the reasons for using OpenCV:

- Specific

  OpenCV was made for image processing. Each function and data structure was designed with the Image Processing coder in mind. Matlab, on the other hand, is quite generic. You get almost anything in the world in the form of toolboxes.

- Speedy

  Matlab is just way too slow. Matlab itself is built upon Java. And Java is built upon C. So when you run a Matlab program, your computer is busy trying to interpret all that Matlab code. Then it turns it into Java, and then finally executes the code.
  If you use C/C++ you don't waste all that time. You directly provide machine language code to the computer, and it gets executed. So ultimately you get more image processing, and not more interpreting.

- Efficient

  Matlab uses just way too much system resources. With OpenCV, you can get away with as little as 10mb RAM for a realtime application.

[2] **MATLAB** (**mat**rix **lab**oratory) is a multi-paradigm numerical computing environment and fourth-generation programming language. For more information visit: https://www.mathworks.com/products/matlab.html

1.4 Object Detection and Recognition

1.4.1 Object detection

Object detection is the process of finding instances of real-world objects such as faces, bicycles, and buildings in images or videos. Object detection algorithms typically use extracted features and learning algorithms to recognise instances of an object category. It is commonly used in applications such as image retrieval, security, surveillance, and automated vehicle parking systems.

Object detection can be done using various models :

- Feature based object detection
- Viola-Jones object Detection
- SVM classification with histograms of oriented gradients (HOG) features
- Image segmentation and blob analysis

1.4.2 Object Recognition

Process for identifying specific object in digital image or video is called object recognition. Object recognition algorithms rely on matching, learning, or pattern recognition algorithms using appearance-based or feature-based techniques. Common techniques include edges, gradients, Histogram of Oriented Gradients (HOG), Haar wavelets, and linear binary patterns. Object recognition is useful in applications such as video stabilisation, automated vehicle parking systems, and cell counting in bioimaging. Many approaches to the task have been implemented over multiple decades.

A. Appearance-based methods

This method uses template images of objects to perform recognition.

- Edge matching
- Divide and conquer search
- greyscale matching, etc.

B. Feature Based Methods

- Interpretation trees
- SIFT (Scale Invariant Feature Transform)
- SURF (Speed Up Robust Features)
- Geometric hashing, etc.

2. OBJECTIVES

The objective is to construct a system which can recognise objects in transformational scenes.

General Objective. The purpose of this thesis is to explore the field of Computer Vision and implement a software project in which the learned methods and programming languages would be used.

Specific Objective.

- Getting acquainted with the field of Computer Vision
- Study and implement object recognition algorithms
- Identify new areas of research within the scope of the problem
- Construct a new system which can address the given problem

3. SCOPE OF THE PROJECT

This study facilitate readers to understand the literature behind recognising objects and how transformations make it complex. It enables them to understand various approaches and algorithms for object detection and feature extraction.

This study also opens new areas of research for gathering new information out of an existing object recognition algorithm and developing new approaches for the computation of this algorithm. Due to time constraints, the scope of this project is somewhat confined. It is limited to the discussion of the possible approach and no complete algorithm is developed. The argument has been discussed in detail.

Lastly, this study also provides an opportunity to think on how the given problem is applicable to medical imagery.

4. PROJECT PLAN

<u>First Semester Plan</u>

- Understanding Human Vision
- Studying basics about the field of Computer Vision
- Understanding the problem statement
- Studying and implementing the existing algorithms related to the problem

<u>Second Semester Plan</u>

- Extracting more information out of SURF algorithm
- Converting SURF descriptor into an image and finding new approach to recognition
- Trying to expand the domain on which the problem can be applied
- Constructing a software system to address the problem

5. PROBLEM DESCRIPTION

Problem Statement:

" Recognise objects in transformational scenes"

The above problem statement means that in a series or a set of digital images which contain some object in different transformations, that object should be recognised in all the images containing it.

The problem can be better understood by breaking it into parts.

I. "Recognise Objects"

As already stated in Section 1, object recognition is a process for identifying specific object in digital image or video. Here this means that we want to recognise a specific object not in a single image, but in multiple images of a given set.

II. "transformational scenes"

Transformations here mean geometric transformations. It can be scaling, rotation, translation, shearing, or a combination of these.

Basically, the problem here is to recognise an object in a set of images, where the specified object has undergone some above mentioned transformations. In other words, the project at hand is to develop an algorithm to recognise objects, which is invariant to scaling, rotation, translation and shearing.

6. LITERATURE REVIEW

We present here all the literature that have been studied. Code for all the algorithms has been added in Appendix.

6.1 Interest Point Detection

Interest point is a point in the image at which the direction of the object changes abruptly. It has a clear, preferably mathematically well founded definition, and has a well defined position in the image space. It is stable under local and global perturbations in the image domain as illumination/ brightness variations, such that the interest points can be reliably computed with high degree of repeatability.

6.2 Edge Detection

Technically, edge detection is the process of locating the edge pixels. Edge detection is one of the most commonly used operations in image analysis, and there are probably more algorithms in the literature for enhancing and detecting edges than any other single subject. The reason for this is that edges form the outline of an object, in the generic sense. Objects are subjects of interest in image analysis and vision systems. An edge is the boundary between an object and the background, and indicates the boundary between overlap- ping objects. This means that if the edges in an image can be identified accurately, all the objects can be located, and basic properties such as area, perimeter, and shape can be measured. Since computer vision involves the identification and classification of objects in an image, edge detection is an essential tool. General edge detection algorithm involves defining what an edge is, and then using this definition to suggest methods of enhancement and identification.

6.2.1 Marr-Hildreth Edge Detector

Marr studied the literature on mammalian visual systems and developed the following algorithm for detection:

1. Convolve the image *I* with a two-dimensional Gaussian[1] function.
2. Compute the Laplacian[2] of the convolved image; call this *L*.
3. Find the edge pixels — those for which there is a zero crossing in *L*.

Code for this algorithm first creates a two-dimensional, sampled version of the LoG (Laplacian of Gaussian) and convolves this in the obvious way with the input image. Then the zero crossings are identified and pixels at those positions are marked.

6.2.2 Canny Edge Detector

The Canny Edge detector, also known as optimal detector, aims to satisfy three main criteria:

- **Low error rate** - it means a good detection of only excitant edges

---

[1] Smoothening Filter

[2] The orientation independent differential operator of lowest order is the *Laplacian*.

- **Good localisation** - The distance between edge pixels detected and real edge pixels have to be minimised.
- **Minimal response** - Only one detector response per edge.

It is a multi-stage algorithms:

1. Filter out any noise. The Gaussian filter is used for this purpose.
2. Find the intensity gradient of the image. Apply a pair of convolution masks. Find the gradient strength and direction.
3. Non-maximum suppression is applied. This removes pixels that are not considered to be part of an edge. Hence, only thin lines will remain.
4. Hysteresis: The final step.

a. If a pixel gradient is higher than the *upper* threshold, the pixel is accepted as an edge

b. If a pixel gradient value is below the *lower* threshold, then it is rejected.

c. If the pixel gradient is between the two thresholds, then it will be accepted only if it is connected to a pixel that is above the *upper* threshold.

6.3 Corner Detection

Corner detection is an approach used within computer vision systems to extract certain kinds of features and infer the contents of an image. A corner can be defined as the intersection of two edges. A corner can also be defined as a point for which there are two dominant and different edge directions in a local neighbourhood of the point. Corner detection is frequently used in motion detection, image registration, video tracking, image mosaicing, panorama stitching, 3D modelling and object recognition.

6.3.1. Harris Corner Detection

The Harris Corner Detector is a mathematical operator that finds features in an image. It is simple to compute, and is fast enough to work on computers. Also, it is popular because it is rotation, scale and illumination variation independent. It has a corner selection criteria. A score is calculated for each pixel, and if the score is above a certain value, the pixel is marked as a corner. The score is calculated using two eigenvalues. That is, you gave the two eigenvalues to a function. The function manipulates them, and gave back a score.

Algorithm for this is:

- Compute horizontal and vertical derivatives of the image, as I_x and I_y.
- Compute three images corresponding to three terms in matrix M (I_x^2, I_y^2, I_xI_y).
- Convolve these three images with a larger gaussian(window).
- Compute scalar cornerness value using one of the R measures.
- Find local maxima above some threshold as detected interest point.

The score for **Harris** corner detector is calculated like this (R is the score):

$$R = \det M - k(trace\, M)^2$$

$$\det M = \lambda_1 \lambda_2$$

$$trace\, M = \lambda_1 + \lambda_2$$

$R<0$ indicates an edge
$R>0$ indicates a corner

6.3.2 Shi-Tomasi Corner Detector

The Shi-Tomasi corner detector is based entirely on the Harris corner detector. However, one slight variation in a "selection criteria" made this detector much better than the original. It works quite well where even the Harris corner detector fails. It's calculated like this:

$$R = min(\lambda_1, \lambda_2)$$

6.4 SIFT (Scale Invariant Feature Transform)

SIFT is feature extraction and matching algorithm. When all images are similar in nature (same scale, orientation, etc) simple corner detectors can work. But when images are of different scales and rotations, SIFT needs to be used. It is invariant to scale, rotation, illumination and viewpoint.

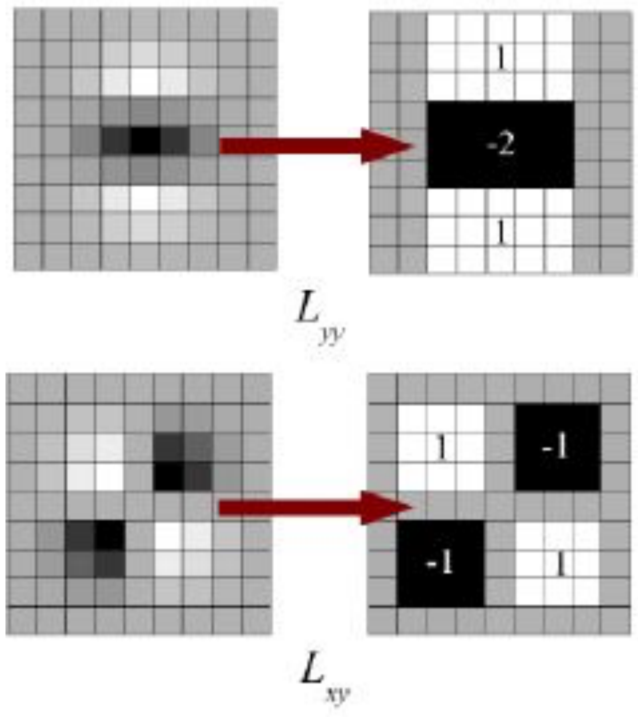
SIFT is quite an involved algorithm. It follows as below:

1. **Constructing a scale space** This is the initial preparation. You create internal representations of the original image to ensure scale invariance. This is done by generating a "scale space".

2. **LoG Approximation** The Laplacian of Gaussian is great for finding interesting points (or key points) in an image. But it's computationally expensive. So we cheat and approximate it using the representation created earlier.
3. **Finding keypoints** With the super fast approximation, we now try to find key points. These are maxima and minima in the Difference of Gaussian image we calculate in step 2.
4. **Get rid of bad key points** Edges and low contrast regions are bad keypoints. Eliminating these makes the algorithm efficient and robust. A technique similar to the Harris Corner Detector is used here.
5. **Assigning an orientation to the keypoints** An orientation is calculated for each key point. Any further calculations are done relative to this orientation. This effectively cancels out the effect of orientation, making it rotation invariant.
6. **Generate SIFT features** Finally, with scale and rotation invariance in place, one more representation is generated. This helps uniquely identify features.

6.5 SURF (Speed Up Robust Features)

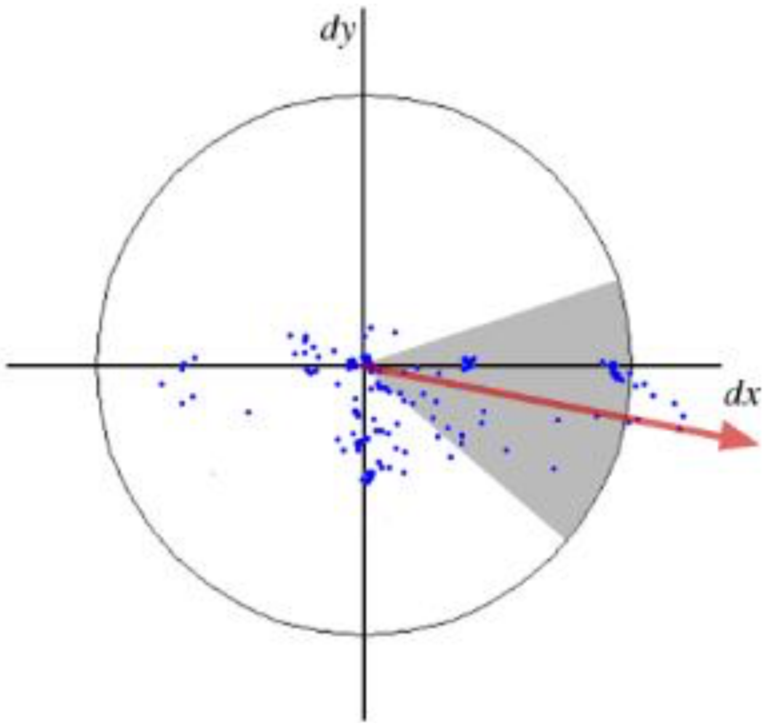
In SIFT, Lowe approximated Laplacian of Gaussian with Difference of Gaussian for finding scale-space. SURF goes a little further and approximates LoG with Box Filter. One big advantage of this approximation is that, convolution with box filter can be easily calculated with the help of integral



images. And it can be done in parallel for different scales. Also the SURF rely on determinant of Hessian matrix for both scale and location.

For orientation assignment, SURF uses wavelet responses in horizontal and vertical direction for a neighbourhood of size 6s. Adequate guassian weights are also applied to it. Then they are plotted in a space as given in below image. The dominant orientation is estimated by calculating the sum of all responses within a sliding orientation window of angle 60 degrees. Interesting thing is that, wavelet response can be found out using integral images very easily at any scale. For many applications, rotation invariance is not required, so no need of finding this orientation, which speeds up the process. SURF provides such a functionality called Upright-SURF or U-SURF. OpenCV supports both, depending upon the flag, upright. If it is 0, orientation is calculated. If it is 1, orientation is not calculated and it is more faster.

For feature description, SURF uses Wavelet responses in horizontal and vertical direction. A neighbourhood of size 20sX20s is taken around the keypoint where s is the size. It is divided into 4x4 subregions. For each



subregion, horizontal and vertical wavelet responses are taken and a vector is formed like this, $v = \left(\sum d_x, \sum d_y, \sum |d_x|, \sum |d_y|\right)$. This when represented as a vector gives SURF feature descriptor with total 64 dimensions. Lower the dimension, higher the speed of computation and matching, but provide better distinctiveness of features.

In short, SURF adds a lot of features to improve the speed in every step. Analysis shows it is 3 times faster than SIFT while performance is comparable to SIFT. SURF is good at handling images with blurring and rotation, but not good at handling viewpoint change and illumination change.

6.6 Medical Imaging

Medical imaging is the technique and process of creating visual representations of the interior of a body for clinical analysis and medical intervention, as well as visual representation of the function of some organs or tissues. It seeks to reveal internal structures hidden by the skin and bones, as well as to diagnose and treat disease. In its widest sense, it is part of biological imaging and incorporates radiology which uses the imaging technologies of X-ray radiography, magnetic resonance imaging(MRI), ultrasound, thermography, PET and many more. Medical imaging is often perceived to designate the set of techniques that non-invasively produce images of the internal aspect of the body. In this restricted sense, medical imaging can be seen as the solution of mathematical inverse problems. This means that cause (the properties of living tissue) is inferred from effect (the observed signal).

Medical images are often seen as images with no transformation of any part in it. They are treated as static images and applied upon with algorithms to contour targets of treatment and organs at risk.

7. PROPOSED SOLUTION

After studying and implementing all the algorithms mentioned in Section 6, we stopped at SURF algorithm to further study it. The feature descriptor formed in the SURF algorithm is of the most interest. This descriptor is formed for both test and sample images. As already discussed, it represents keypoints in vector form, and is used to find matches between the images. Our solution revolves around the usage of this ‘descriptor’.

7.1 Why use descriptor and not keypoints themselves

An important thing to understand about object recognition algorithms is that after extracting the keypoints, only information about their position, and sometimes their coverage area in the image is obtained. While the information about keypoint position might sometimes be useful, it does not say much about the keypoints themselves. Depending on the algorithm used to extract keypoint, some general characteristics of the extracted keypoints are known but how different or similar one keypoint is to the other, can not be known by this.

Therefore, descriptors come onto play. They summarize, in vector format (of constant length) some characteristics about the keypoints. For example, it could be their intensity in the direction of their most pronounced orientation. It's assigning a numerical description to the area of the image the keypoint refers to.

Below mentioned points highlights the importance of descriptors:

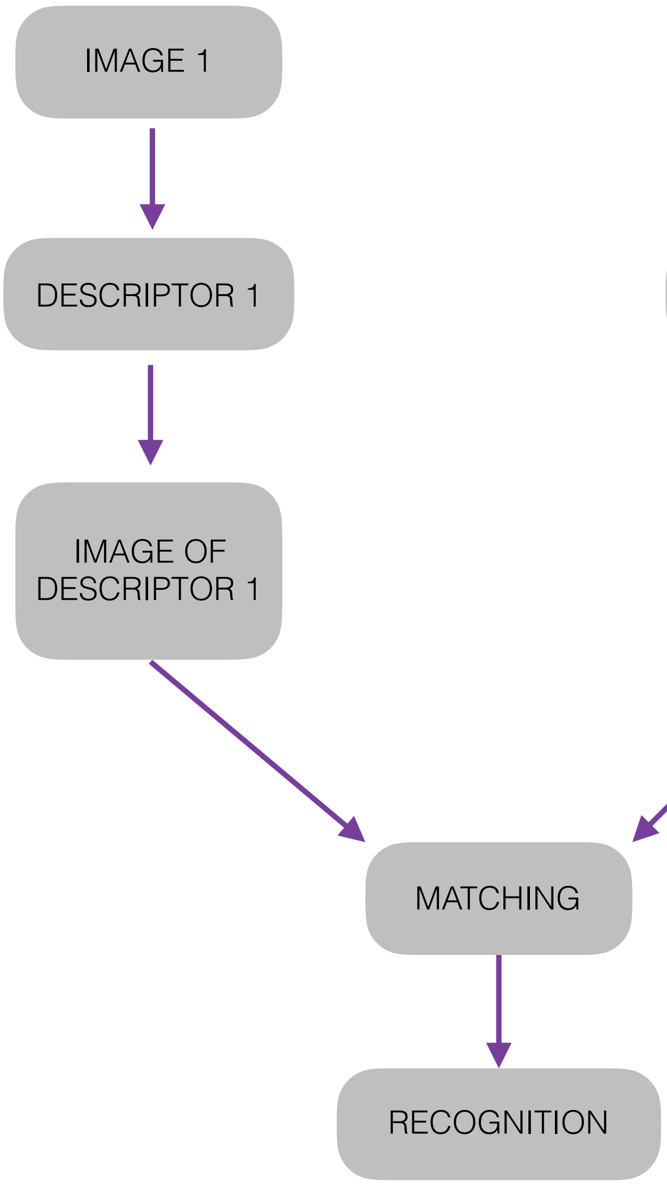
- they are independent of keypoint position
- they are robust against image transformations
- they are scale independent

Of course, no descriptor is completely robust against all transformations (nor against any single one if it is strong, e.g. big change in perspective).
Different descriptors are designed to be robust against different transformations which is sometimes opposed to the speed it takes to calculate them. After the descriptors for keypoints have been calculated, now there is a way to compare those keypoints.
These reasons justify our study to revolve around descriptors and not keypoints.

7.2 Solution

Since we now have a representation of the keypoints, descriptor, which is robust to scaling and rotation, we can use this descriptor for the recognition purpose in a different way, and not just calculate euclidian distance between keypoints in it and find matches.

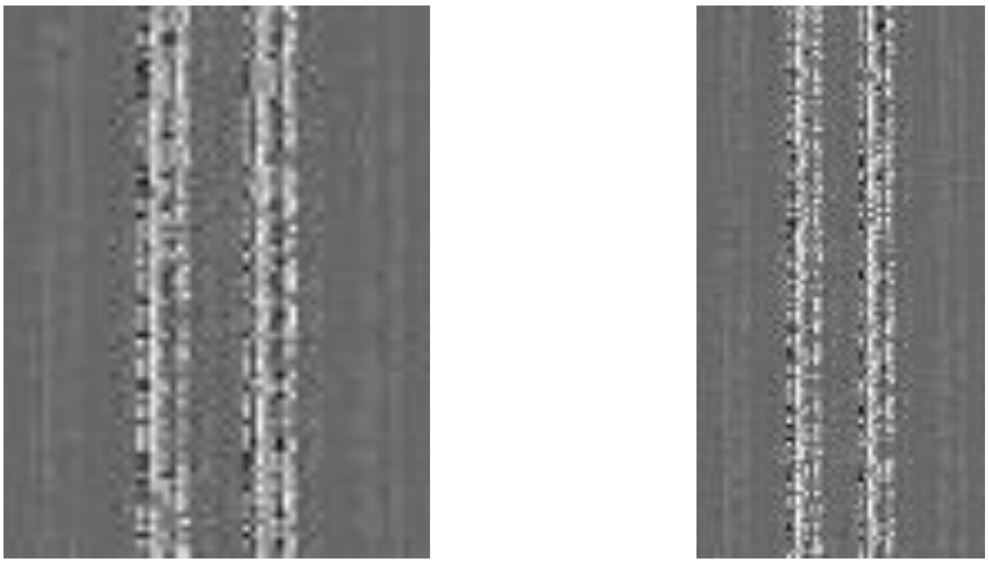
The proposal here is to convert this descriptor, which contains vectors of numbers, into an image and then apply some other sort of matching algorithm between such images of descriptors, from different images of objects. This can be better understood using the following diagram.



FLOW CHART REPRESENTING THE PROPOSED SOLUTION.

8. RESULTS AND DISCUSSION

The result proposed in the last section opens up a new approach towards solving the given problem. But the outcome of this approach is yet to be found since the complete algorithm has not been developed yet. Although, the part of the solution that converts the descriptors of the keypoints into images has been completed. The results for the same are displayed below[1]. (These are just fragments of the images. Entire image can be viewed in Appendix)



IMAGES OF DESCRIPTORS FROM SAMPLE AND TEST IMAGES

Discussion. The progress till now gives a positive feedback about the direction taken for solving the problem. Existing algorithms, although, solves the problem to some extent, they are not the ultimate solution for it. An algorithm

[1] Code for the same can be found in Appendix

which is invariant to all the transformations, is what is desired out of this project. The approach taken seems to be promising for this.

Extended discussion. Our study about medical imaging has revealed an existing but unattended challenge of transformations prevailing in medical images. Object recognition in transformational scenes is believed to be associated with just discrete objects. But in reality transformations do exist in medical images. We want to put this argument forward through this thesis.

Many radiology modalities have faced this as a drawback.

- In PET [11] of lungs for detecting lung tumour, due to highly inhomogeneous moving target, lung tumour, there is lack in accuracy of delineation. This is an example of transformational object/part in medical images.
- During MRI of chest, lungs expand and collapse due to breathing, affecting the images and shapes of other tissues and organs. Here again recognition is required beforehand so as to make the process less tedious.
- An other example can be 3-D reconstruction [12] in multi detector CT of the abdominal. During CT of urinary track, transformations can be seen due to pressure of urine in the track or release of it.

Like this, there exist several examples where transformations can be seen within the medical image. Therefore, we argue to consider medical imagery under the domain of the given problem. This can be a new area of research in the field of medical imagery.

9. LIMITATIONS AND FUTURE SCOPE

Object recognition has always been an area of interest from the beginning of Computer Vision. Its popularity is growing and is expanding to different fields. This project has increased my knowledge to develop such systems and its useful applications in different fields.

9.1 Limitations of the Project

My system is capable of performing object recognition using several existing approaches. It can detect features like edges and corners in the digital images, and can can generate and store feature descriptors for further use. However, this study is limited to just proposing a new approach to recognition of object in transformational scenes, and does not provide a complete working code for the same.

Also, the report does not provide extensive details on the proposed idea of the new domain for the given problem statement, i.e., transformations in medical imagery.

9.2 Future Scope

In regard to future development, I believe that the project can can be developed further if the proposed solution can be converted into a complete algorithm and then into an executable code. In order to improve the project further, research and developed on the segmentation of the object from the image, as well as research on using the SURF descriptor more efficiently, needs to be critically examined. Once these issues are resolved and code is fully implemented, this can serve as a basis for further development.

The study proposes a new idea that clinical images too have transformations, but this idea is not extensively examined and discussed right now. Further research in this domain can help develop better systems to help doctors and researchers in medical field.

Right now the system just looks into physical objects and works on it. But with further development, it can work on clinical images as well. This system can then aid doctors and researchers in recognising tumours, dead tissues, etc. in clinical images like MRI, CT Scan, etc.

The project, therefore, has immense scope in terms of functionality that can be changed to gain better results. Object recognition in transformational scenes can be used in industries to develop better autonomous robots for the use in factories; in medical field as discussed above; in robotics field improving self driving cars, household robots, and in several other areas.

Summarising the above,

- The proposed solution can be converted into a complete algorithm and then into a code to execute it.
- The new idea proposed, to detect transformations in clinical imagery, can be further developed in to a full fledged algorithm.
- Both the proposed new approaches can be combined and developed into a commercial product.

10. CONCLUSION

In this thesis we analysed and reviewed object recognition methods, focusing on finding some new approach to the same in transformational scenes. We represented a literature survey and also provided successful application of those algorithms, demonstrating their strengths. New areas of research have also been stated in this thesis, which fulfils the aim of the project.

An idea of using SURF descriptors for object recognition has been proposed. However, a complete algorithm is out of the current scope of the project.

It is very well understood from the previous section (Section 7.1) that descriptor is the best choice for approaching a problem like object recognition in transformational scenes.

Also, the thesis presents an argument to consider clinal imagery in the domain of transformational scenes, and further develop an algorithm for the recognition purpose in the medical field.

11. PERSONAL REFLECTION

This research project was a great learning experience as it made me realise my inclination towards research field. It highlighted my strengths as well as the areas where I could improve my skills for future projects. This project was as much challenging as it was interesting. The eagerness to explore the field of artificial intelligence led me to choose computer vision as the field of study for my thesis.

I believe that much of my work is centred around understanding and building code for the existing approaches for the problem. All the code is completely developed from the scratch and is open source[1]. So the coding part was time consuming, but a good learning experience. I spent a lot of time in familiarising myself with the new library of OpenCV and understanding how things work in computer vision.

Although, the project does not completely meet the stated objective, I have been pleased with how far I have come from having no idea of what computer vision is to gradually studying algorithms and finding new areas of research.

My weakness, however, was software and time management. Time required for learning, implementing and further research, was underestimated. I feel that I could have handled this project much better in terms of sticking to

[1] Please find repository at: https://github.com/himanshu0113/cvod/

schedule or sticking to the approaches I had taken during the development cycle. In future, a more realistic plan for the completion and execution could be used to avoid nay disappointment or delays in the task completion.

12. REFERENCES

1. Bigun, Josef, 2006. *Vision with Direction*. Springer.
2. Parker, J.R., 2011. *Algorithms for Image Processing and Computer Vision*, Second Edition. Wiley Publishing, Inc.
3. Szeliski, Richard, 2010. *Computer Vision: Algorithms and Applications*. Springer.
4. Bradski, G., Kaehler, A., 2008. *Learning OpenCV*. O'Reilly Media, Inc.
5. Bay, H., Tuytelaars, T., Gool, L. V., 2008. *Speed-Up Robust Features.* Journal: Computer Vision and Image Understanding, Elsevier Science Inc.
6. Yang, M., Kpalma, K., Ronsin, J., 2008. *A Survey of Shape Feature Extraction Techniques*. Peng-Yeng Yin. Pattern Recognition, IN-TECH.
7. Shaikh, S.H., Saeed, K., Chaki, N., 2014. *Moving Object Detection Using Background Subtraction*. Springer.
8. Lowe, D.G., 2001. *Local Feature View Clustering for 3D Object Recognition.* IEEE.
9. Anitha, J., Deepa, S.M., 2014. *Tracking and Recognition of Objects using SURF Descriptor and Harris Corner Detection*. International Journal of Current Engineering and Technology.
10. Azhari, E.M., Muhd. Mudzakkir Mohd. Hatta, Htike, Z.Z., Win, S.L., 2014. *Tumour Detection in Medical Imaging: A Survey*. International Journal of Advanced Information Technology.

11. Cheung, K.Y., 2006. *Intensity Modulated Radiotherapy: Advantages, Limitations and Future Developments*. Biomedical Imaging and Intervention Journal.
12. Maher, M.M., Kalra, M.K., Sahani, D.V., Perumpillichira, J.J., Rizzo, S., Saini, S., and Mueller, P.R., 2004. *Techniques, Clinical Applications and Limitations of 3D Reconstruction in CT of the Abdomen*. Korean Journal of Radiology.
13. *A Short Guide to Writing Your Final Year Project Report Or MSc Dissertation,* 2011. Cardiff University School of Computer Science and Informatics.
14. *Purdue Online Writing Lab.* Purdue University.
15. *The Chicago Manual of Style*, Sixteenth Edition, September 2010.
16. Website: *www.OpenCV.org*
17. Website: *www.AIShack.in*

APPENDIX

APPENDIX A: CODE

A. Canny Edge Detector

```
Mat s, src_gray;

Mat dst, detected_edges;


int edgeThresh = 1;

int lowThreshold;

int const max_lowThreshold = 100;

int RATIO = 3;

int kernel_size = 3;

char* window_name = "Edge Map";


/**

* @function CannyThreshold

* @brief Trackbar callback - Canny thresholds input with
a ratio 1:3

*/

void CannyThreshold(int, void*)

{

     /// Reduce noise with a kernel 3x3

     blur(src_gray, detected_edges, Size(3, 3));


     /// Canny detector
     Canny(detected_edges, detected_edges, lowThreshold,
lowThreshold*RATIO, kernel_size);


     /// Using Canny's output as a mask, we display our
result

     dst = Scalar::all(0);


     s.copyTo(dst, detected_edges);

     imshow(window_name, dst);

}




/** @function main */

int callcanny(Mat src)

{

     /// Load an image

     if (!src.data)

     {

          return -1;

     }


     /// Create a matrix of the same type and size as src
(for dst)

     dst.create(src.size(), src.type());
```

```
    /// Convert the image to grayscale

    cvtColor(src, src_gray, CV_BGR2GRAY);


    /// Create a window

    namedWindow(window_name, CV_WINDOW_AUTOSIZE);


    /// Create a Trackbar for user to enter threshold

    createTrackbar("Min Threshold:", window_name,
&lowThreshold, max_lowThreshold, CannyThreshold);


    src.copyTo(s);

    /// Show the image

    CannyThreshold(0, 0);


    /// Wait until user exit program by pressing a key

    waitKey(0);

    return 0;

}
```

B. Harris Corner Detection

```
int callharris(Mat src)

{
     Mat harris_dst, harris_src_gray;

     Mat harris_dst_scaled, harris_dst_norm;


     int blocksize, ksize;

     double k;

     int thresh = 100;


     if (!src.data)

     {

          return -1;

     }


     blocksize = 4;

     k = 0.04;

     ksize = 3;


     /// Convert the image to grayscale

     cvtColor(src, harris_src_gray, CV_BGR2GRAY);


     /// Image to store Harris detector response

     harris_dst.create(harris_src_gray.size(),
CV_32FC(6));


     ///calling harris detector
```

```
	cornerHarris(harris_src_gray, harris_dst, blocksize,
ksize, k, BORDER_DEFAULT);


	//changes

	normalize(harris_dst, harris_dst_norm, 0, 255,
NORM_MINMAX, CV_32FC1, Mat());

	convertScaleAbs(harris_dst_norm, harris_dst_scaled);


	//harris_dst_norm.copyTo(harris_dst_scaled);

	//drawing circle

	for (int i = 0; i < harris_src_gray.rows; i++)

	{

		for (int j = 0; j < harris_src_gray.cols; j++)

		{

			if (harris_dst_scaled.at<uchar>(i, j) >
thresh)			//error due to size mismatch-
convertscaleabs changes size to 8bit - resolved

			{

				circle(harris_dst_scaled, Point(j,
i), 5, Scalar(0), 2, 8, 0);

			}

		}

	}



	/// Create a windows

	namedWindow("Response", CV_WINDOW_AUTOSIZE);
```

```
    imshow("Response", harris_dst_scaled);


    waitKey(0);

    return 0;

}
```

C. Marr Hildreth Edge Detector

```
float norm(float x, float y) {
     return (float)sqrt((double)(x*x + y*y));
}


float distance(float a, float b, float c, float d) {
     return norm((a − c), (b − d));
}


float** f2d(int n, int m) {
     float **temp = new float*[n];
     for (int i = 0; i < n; i++)
          temp[i] = new float[m];
     return temp;
}


     // Gaussian
float gauss(float x, float sigma) {
     return (float)exp((double)((−x*x) / (2 *
sigma*sigma)));
}


float LoG(float x, float sigma) {
     float x1;

     x1 = gauss(x, sigma);

     return (x*x - 2 * sigma*sigma) /
(sigma*sigma*sigma*sigma) * x1;

}



void convolution(Mat im, float **mask, int nr, int nc,
float **res, int NR, int NC) {

     int i, j, ii, jj, n, m, k, kk;

     float x, y;


     k = nr / 2;

     kk = nc / 2;


     for (i = 0; i < NR; i++)

          for (j = 0; j < NC; j++) {

               x = 0;

               for (ii = 0; ii<nr; ii++)

               {

                    n = i - k + ii;

                    if (n<0 || n >= NR) continue;

                    for (jj = 0; jj<nc; jj++)

                    {

                         m = j - kk + jj;

                         if (m<0 || m >= NC) continue;
```

```
                              x += mask[ii][jj] *
(im.at<uchar>(n, m));
                         }
                    }
                    res[i][j] = x;
               }
}

void zero_cross(float **lapim, Mat im) {
     int i, j, k, n, m, dx, dy;
     float x, y, z;
     int xi, xj, yi, yj, count = 0;
     Mat deriv;

     for (i = 1; i < im.rows - 1; i++)
          for (j = 0; j < im.cols - 1; j++) {
               im.at<uchar>(i, j) = 0;
               if (lapim[i - 1][j] * lapim[i + 1][j]<0)
{ im.at<uchar>(i, j) = 255; continue; }
               if (lapim[i][j - 1] * lapim[i][j + 1]<0)
{ im.at<uchar>(i, j) = 255; continue; }
               if (lapim[i + 1][j - 1] * lapim[i - 1][j +
1]<0){ im.at<uchar>(i, j) = 255; continue; }
               if (lapim[i - 1][j - 1] * lapim[i + 1][j +
1]<0){ im.at<uchar>(i, j) = 255; continue; }
```

```
          }
}


void marr(double s, Mat im) {
     int width;
     float **smx;
     int i, j, k, n;
     float **lgau, z;


     // Create a Gaussian and a derivative of Gaussian
filter mask
     width = 3.35*s + 0.33;
     n = width + width + 1;
     printf("Smoothing with a Gaussian of size %dx%d\n",
n, n);
     lgau = f2d(n, n);
     for (i = 0; i < n; i++)
          for (j = 0; j < n; j++) {
               lgau[i][j] = LoG(distance((float)i,
(float)j, (float)width, (float)width), s);
          }


     // Convolution of source image with a Gaussian in X
and Y directions
     smx = f2d(im.rows, im.cols);
```

```
    printf("Convolution with LoG : \n");
    convolution(im, lgau, n, n, smx, im.rows, im.cols);

    // Locate the zero crossings
    printf("Zero crossings:\n");
    zero_cross(smx, im);

    printf("Zero crossings returned:\n");
     // Clear the boundary
    for (i = 0; i<im.rows; i++)
    {
         for (j = 0; j <= width; j++) im.at<uchar>(i, j)
= 0;
         for (j = im.cols - width - 1; j<im.cols; j++)
              im.at<uchar>(i, j) = 0;
    }

    for (j = 0; j<im.cols; j++) {
         for (i = 0; i <= width; i++) im.at<uchar>(i, j)
= 0;
         for (i = im.rows - width - 1; i < im.rows; i++)
              im.at<uchar>(i, j) = 0;
    }
```

```
	free(smx[0]); free(smx);
	free(lgau[0]); free(lgau);
}

std::string getImageType(int number)
{
	// find type
	int imgTypeInt = number % 8;
	std::string imgTypeString;

	switch (imgTypeInt)
	{
	case 0:
		imgTypeString = "8U";
		break;
	case 1:
		imgTypeString = "8S";
		break;
	case 2:
		imgTypeString = "16U";
		break;
	case 3:
		imgTypeString = "16S";
```

```
            break;
        case 4:
            imgTypeString = "32S";
            break;
        case 5:
            imgTypeString = "32F";
            break;
        case 6:
            imgTypeString = "64F";
            break;
        default:
            break;
        }

        // find channel
        int channel = (number / 8) + 1;

        std::stringstream type;
        type << "CV_" << imgTypeString << "C" << channel;

        return type.str();
    }

    int callMH(Mat src)
```

```
{
     int i, j;
     double s = 1.0;
     Mat scr_gray, img2;

     //img1 = imread("img3.jpg");
     //imshow("Original", img1);
     cvtColor(src, scr_gray, CV_BGR2GRAY);

     s = 1.2;

     string str = getImageType(src.type());
     printf("%s", str.c_str());

     scr_gray.copyTo(img2);

     marr(s - 0.8, scr_gray);
     marr(s + 0.8, img2);

     for (i = 0; i < src.rows; i++)
          for (j = 0; j < src.cols; j++) {
               if (scr_gray.at<uchar>(i, j)>0 &&
img2.at<uchar>(i, j)>0)
                    scr_gray.at<uchar>(i, j) = 0;
```

```
                else
                    scr_gray.at<uchar>(i, j) = 255;
            }


    imshow("Filtered", scr_gray);
    waitKey(0);
    return 0;
}
```

D. Shi-Tomasi Corner Detector

```
// Global variables

Mat gsrc, gsrc_gray;


int maxCorners = 10;

int maxTrackbar = 25;


RNG rng(12345);


/** @function main */

void calltomasi(Mat src)

{

     /// Load source image and convert it to gray

     gsrc = src;

     cvtColor(src, gsrc_gray, CV_BGR2GRAY);


     /// Create Window

     namedWindow("Image", CV_WINDOW_AUTOSIZE);


     /// Create Trackbar to set the number of corners

     createTrackbar("Max  corners:", "Image",
&maxCorners, maxTrackbar, goodFeaturesToTrack_Demo);


     imshow("Image", src);
```

```
    goodFeaturesToTrack_Demo(0, 0);

    waitKey(0);
}

/**
* @function goodFeaturesToTrack_Demo.cpp
* @brief Apply Shi-Tomasi corner detector
*/
void goodFeaturesToTrack_Demo(int, void*)
{
    if (maxCorners < 1) { maxCorners = 1; }

    /// Parameters for Shi-Tomasi algorithm
    vector<Point2f> corners;
    double qualityLevel = 0.01;
    double minDistance = 10;
    int blockSize = 3;
    bool useHarrisDetector = false;
    double k = 0.04;

    /// Copy the source image
```

```
     Mat copy;

     copy = gsrc.clone();


     /// Apply corner detection

     goodFeaturesToTrack(gsrc_gray, corners, maxCorners,
qualityLevel, minDistance, Mat(), blockSize,
useHarrisDetector, k);




     /// Draw corners detected

     cout << "** Number of corners detected: " <<
corners.size() << endl;

     int r = 4;

     for (int i = 0; i < corners.size(); i++)

     {

          circle(copy, corners[i], r,
Scalar(rng.uniform(0, 255), rng.uniform(0, 255),
rng.uniform(0, 255)), -1, 8, 0);

     }



     /// Show what you got

     namedWindow("Image", CV_WINDOW_AUTOSIZE);

     imshow("Image", copy);



     /// Set the neeed parameters to find the refined
corners

     Size winSize = Size(5, 5);
     Size zeroZone = Size(-1, -1);

     TermCriteria criteria = TermCriteria(CV_TERMCRIT_EPS
+ CV_TERMCRIT_ITER, 40, 0.001);


     /// Calculate the refined corner locations

     cornerSubPix(gsrc_gray, corners, winSize, zeroZone,
criteria);


     /// Write them down

     for (int i = 0; i < corners.size(); i++)

     {

          cout << " -- Refined Corner [" << i << "]  ("
<< corners[i].x << "," << corners[i].y << ")" << endl;

     }

}
```

E. SIFT Algorithm

```
int callSift(Mat src)

{

     Mat src_gray, dst;


     cvtColor(src, src_gray, CV_BGR2GRAY);


     Feature2D detector;

     std::vector<KeyPoint> keypoints;


     detector.detectAndCompute(src_gray, noArray(),
keypoints, dst, false);


     drawKeypoints(src_gray, keypoints, dst);

     namedWindow("SIFT", CV_WINDOW_AUTOSIZE);

     imshow("SIFT", dst);




     return 0;

}
```

F. SURF Algorithm

```
using namespace cv::xfeatures2d;

const int LOOP_NUM = 10;
const int GOOD_PTS_MAX = 50;
const float GOOD_PORTION = 0.15f;

int64 work_begin = 0;
int64 work_end = 0;

static void workBegin()
{
    work_begin = getTickCount();
}

static void workEnd()
{
    work_end = getTickCount() - work_begin;
}

static double getTime()
{
     return work_end / ((double)getTickFrequency())*
1000.;

}


struct SURFDetector

{

     Ptr<Feature2D> surf;

     SURFDetector(double hessian = 800.0)

     {

          surf = SURF::create(hessian);

     }

     template<class T>

     void operator()(const T& in, const T& mask,
std::vector<cv::KeyPoint>& pts, T& descriptors, bool
useProvided = false)

     {

          surf->detectAndCompute(in, mask, pts,
descriptors, useProvided);

     }

};


template<class KPMatcher>

struct SURFMatcher

{

     KPMatcher matcher;

     template<class T>
```

```
    void match(const T& in1, const T& in2,
std::vector<cv::DMatch>& matches)

    {

        matcher.match(in1, in2, matches);

    }

};


static Mat drawGoodMatches(

    const Mat& img1,

    const Mat& img2,

    const std::vector<KeyPoint>& keypoints1,

    const std::vector<KeyPoint>& keypoints2,

    std::vector<DMatch>& matches,

    std::vector<Point2f>& scene_corners_

    )

{

    //-- Sort matches and preserve top 10% matches

    std::sort(matches.begin(), matches.end());

    std::vector< DMatch > good_matches;

    double minDist = matches.front().distance;

    double maxDist = matches.back().distance;


    const int ptsPairs = std::min(GOOD_PTS_MAX, (int)
(matches.size() * GOOD_PORTION));
```

```
     for (int i = 0; i < ptsPairs; i++)

     {

          good_matches.push_back(matches[i]);

     }

     std::cout << "\nMax distance: " << maxDist << 
std::endl;

     std::cout << "Min distance: " << minDist << 
std::endl;


     std::cout << "Calculating homography using " << 
ptsPairs << " point pairs." << std::endl;


     // drawing the results

     Mat img_matches;


     drawMatches(img1, keypoints1, img2, keypoints2,

          good_matches, img_matches, Scalar::all(-1), 
Scalar::all(-1),

          std::vector<char>(), 
DrawMatchesFlags::NOT_DRAW_SINGLE_POINTS);


     //-- Localize the object

     std::vector<Point2f> obj;

     std::vector<Point2f> scene;


     for (size_t i = 0; i < good_matches.size(); i++)
     {
          //-- Get the keypoints from the good matches

obj.push_back(keypoints1[good_matches[i].queryIdx].pt);

scene.push_back(keypoints2[good_matches[i].trainIdx].pt);
     }
     //-- Get the corners from the image_1 ( the object
to be "detected" )
     std::vector<Point2f> obj_corners(4);
     obj_corners[0] = Point(0, 0);
     obj_corners[1] = Point(img1.cols, 0);
     obj_corners[2] = Point(img1.cols, img1.rows);
     obj_corners[3] = Point(0, img1.rows);
     std::vector<Point2f> scene_corners(4);


     Mat H = findHomography(obj, scene, RANSAC);
     perspectiveTransform(obj_corners, scene_corners, H);


     scene_corners_ = scene_corners;


     //-- Draw lines between the corners (the mapped
object in the scene - image_2 )
     line(img_matches,
          scene_corners[0] + Point2f((float)img1.cols,
0), scene_corners[1] + Point2f((float)img1.cols, 0),
          Scalar(0, 255, 0), 2, LINE_AA);

     line(img_matches,

          scene_corners[1] + Point2f((float)img1.cols,
0), scene_corners[2] + Point2f((float)img1.cols, 0),

          Scalar(0, 255, 0), 2, LINE_AA);

     line(img_matches,

          scene_corners[2] + Point2f((float)img1.cols,
0), scene_corners[3] + Point2f((float)img1.cols, 0),

          Scalar(0, 255, 0), 2, LINE_AA);

     line(img_matches,

          scene_corners[3] + Point2f((float)img1.cols,
0), scene_corners[0] + Point2f((float)img1.cols, 0),

          Scalar(0, 255, 0), 2, LINE_AA);

     return img_matches;

}


////////////////////////////////////////////////////

// This program demonstrates the usage of SURF_OCL.

// use cpu findHomography interface to calculate the
transformation matrix

void callsurf(Mat src1, Mat src2)

{

     UMat img1, img2;

     src1.copyTo(img1);

     src2.copyTo(img2);

    double surf_time = 0.;
    //declare input/output

    std::vector<KeyPoint> keypoints1, keypoints2;

    std::vector<DMatch> matches;


    UMat _descriptors1, _descriptors2;

    Mat descriptors1 = _descriptors1.getMat(ACCESS_RW),

        descriptors2 = _descriptors2.getMat(ACCESS_RW);


    //instantiate detectors/matchers

    SURFDetector surf;


    SURFMatcher<BFMatcher> matcher;


    //-- start of timing section


    for (int i = 0; i <= LOOP_NUM; i++)

    {

        if(i == 1) workBegin();

        surf(img1.getMat(ACCESS_READ), Mat(), keypoints1,
descriptors1);

        surf(img2.getMat(ACCESS_READ), Mat(), keypoints2,
descriptors2);

        matcher.match(descriptors1, descriptors2,
matches);
```

```
    }

    workEnd();

    std::cout << "FOUND " << keypoints1.size() << " 
keypoints on first image" << std::endl;

    std::cout << "FOUND " << keypoints2.size() << " 
keypoints on second image" << std::endl;


    surf_time = getTime();

    std::cout << "SURF run time: " << surf_time / 
LOOP_NUM << " ms" << std::endl<<"\n";




    std::vector<Point2f> corner;

    Mat img_matches = 
drawGoodMatches(img1.getMat(ACCESS_READ), 
img2.getMat(ACCESS_READ), keypoints1, keypoints2, 
matches, corner);


    //-- Show detected matches

    namedWindow("surf matches", 0);

    imshow("surf matches", img_matches);

    imwrite("test.jpg", img_matches);


    waitKey(0);

    //return EXIT_SUCCESS;

}
```

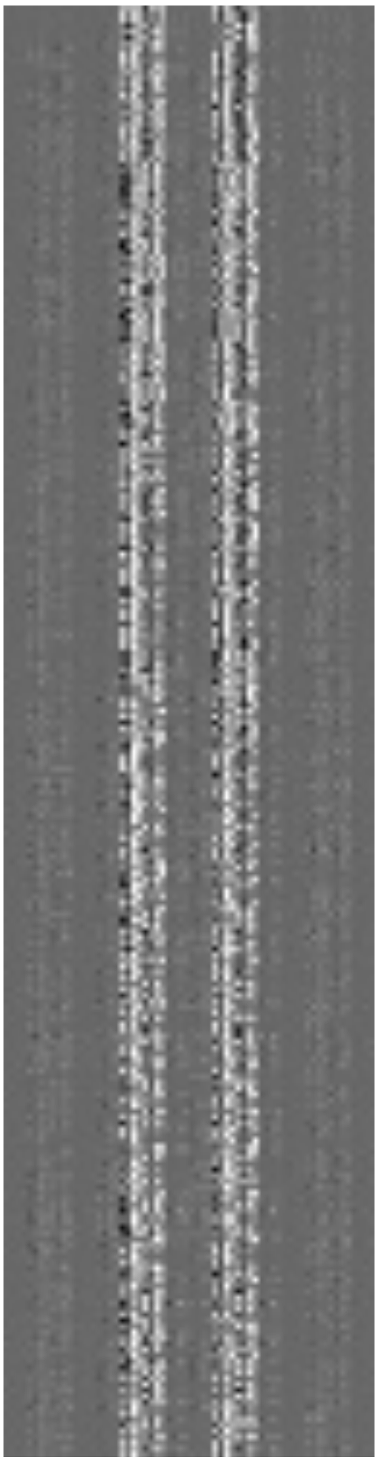
G. Saving SURF Descriptors as images

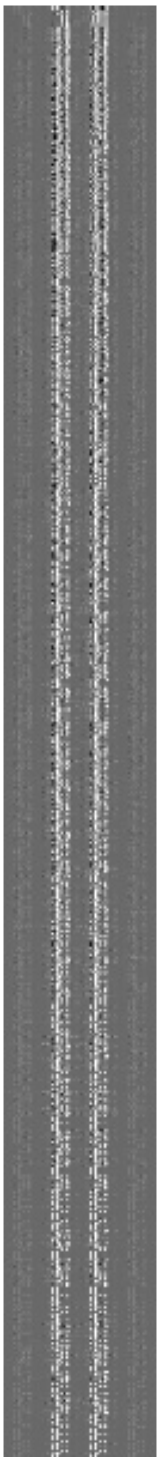
After descriptors have been calculated in SURF (Code F), following code can be used to save the descriptors as image (JPEG) files.

```
//-- Save descriptors in file

     FileStorage store("keypoints_surf.yml", 
FileStorage::WRITE);

     write(store, "keypoints1", keypoints1);

     write(store, "descriptor1", descriptors1);

     store.release();



     //-- saving descriptors as images ----- scale the 
descriptor va;ues and then save

     //normalize(descriptors1, descriptors1, 0, 1, 
NORM_MINMAX);

     double min, max;

     minMaxLoc(descriptors1, &min, &max);

     cout << min << endl << max;

     descriptors1 += abs(min);

     descriptors1 *= 255;

     imwrite("images/descriptor_1.jpeg", descriptors1);


     minMaxLoc(descriptors2, &min, &max);

     cout << min << endl << max;

     descriptors2 += abs(min);
```

```
descriptors2 *= 255;

imwrite("images/descriptor_2.jpeg", descriptors2);
```

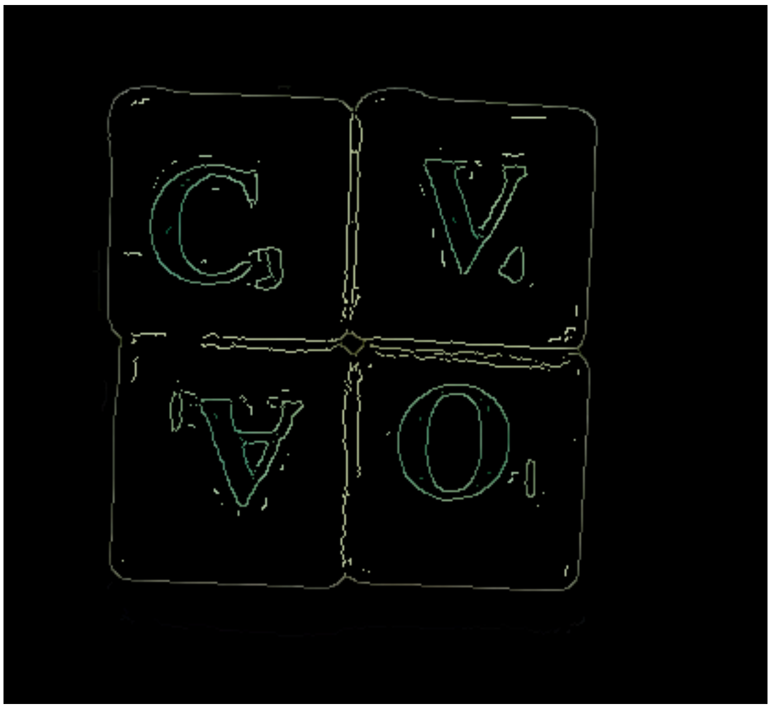
APPENDIX B: DESCRIPTOR IMAGES


DESCRIPTOR 1

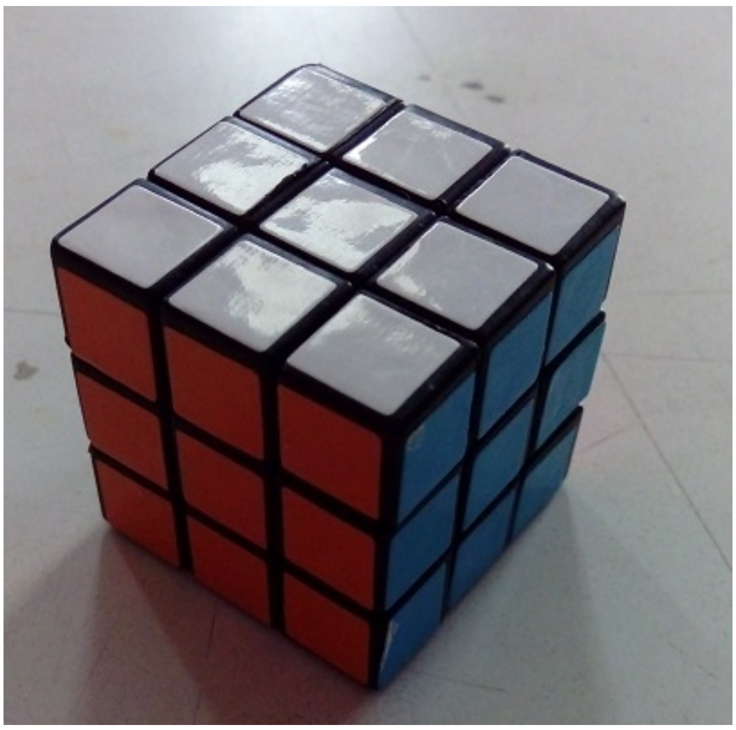

DESCRIPTOR 2

APPENDIX C: IMAGES OF IMPLEMENTATIONS

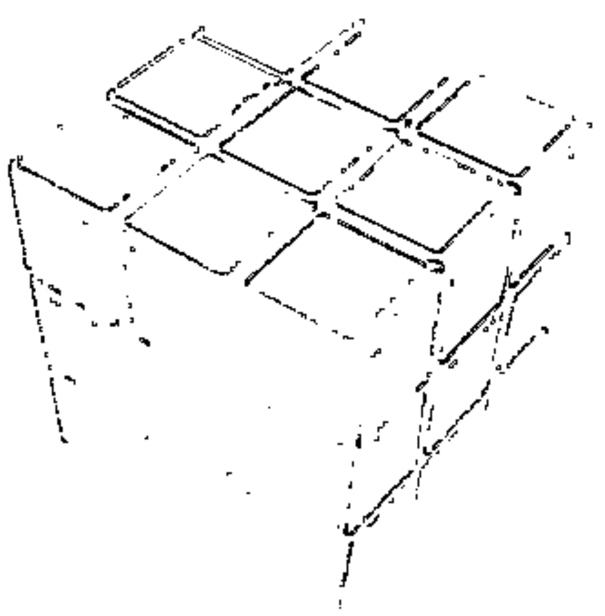
ORIGINAL IMAGE



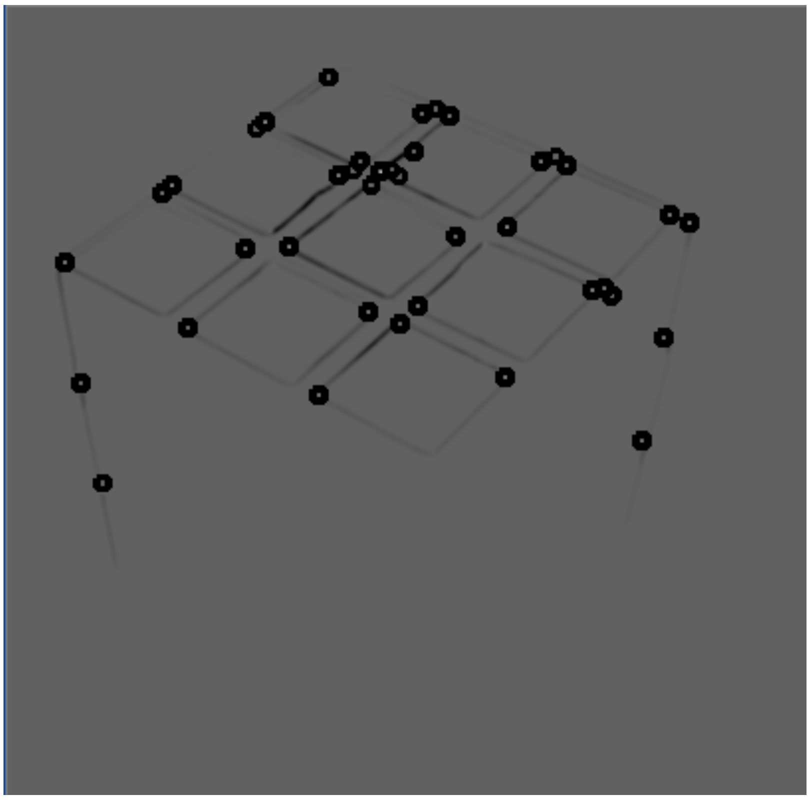
OUTPUT OF CANNY EDGE DETECTOR



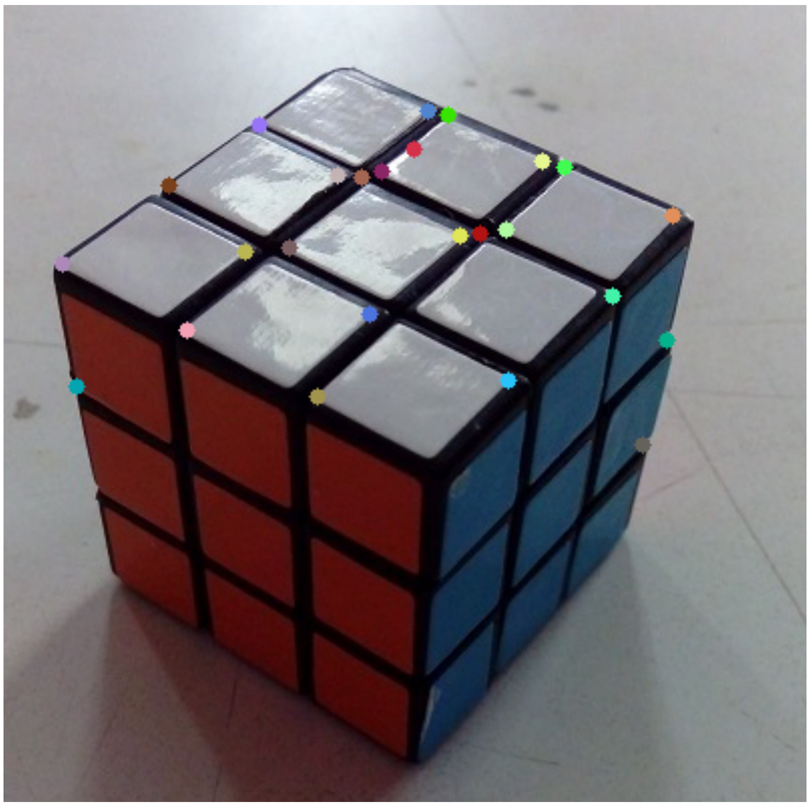
ORIGINAL IMAGE



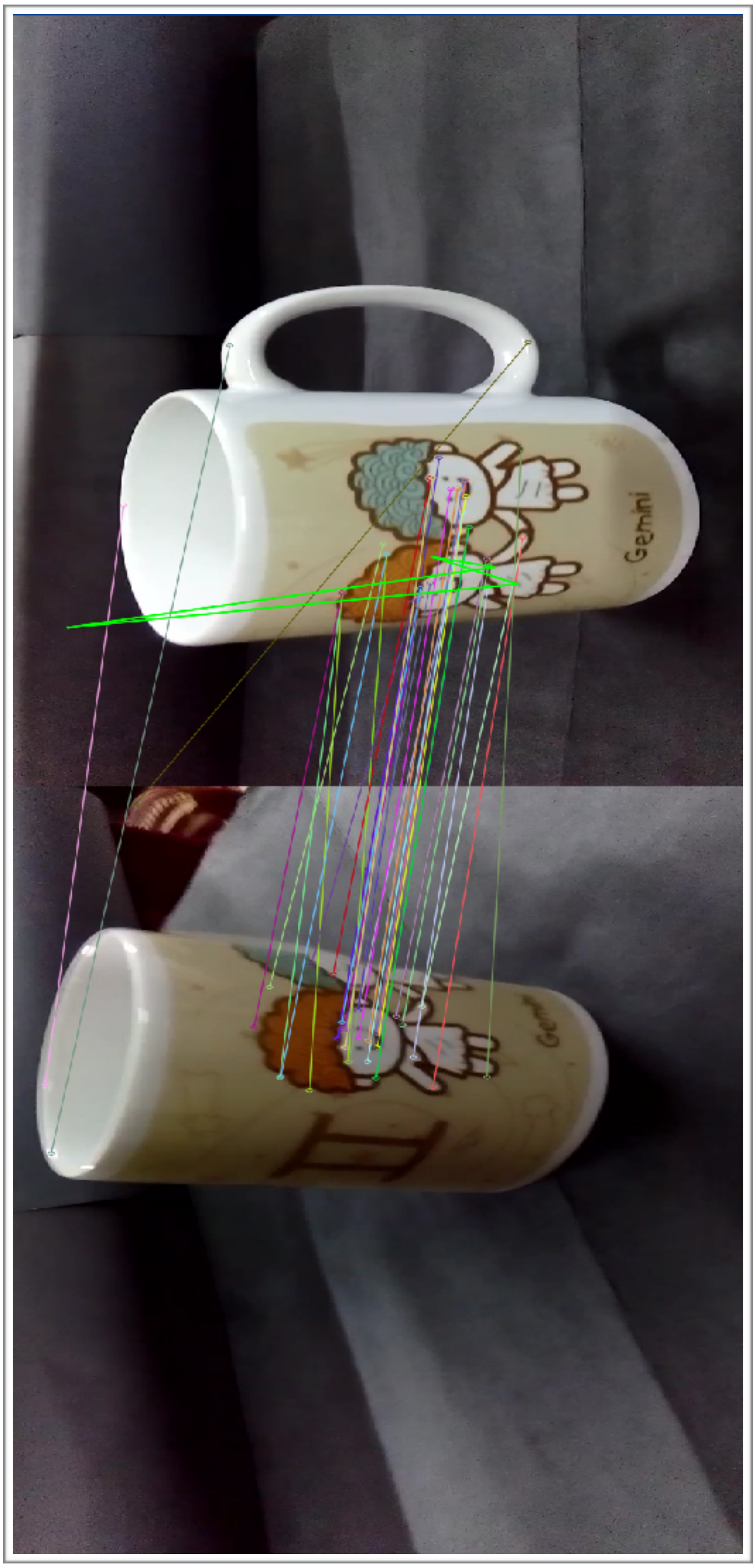
OUTPUT OF MARR HILDRETH EDGE DETECTOR



OUTPUT OF HARRIS CORNER DETECTOR



OUTPUT OF SHI-TOMASI CORNER DETECTOR



OUTPUT OF SURF ALGORITHM

- END OF THESIS -