University of Burgundy

MsCV

Visual Perception Module

Report for

Computer Vision Toolbox

using OpenCV and C++

by

Gopikrishna Erabati

Supervisor: Dr. Abd El Rahman SHABAYEK

1. Introduction

Computer vision is the science that aims to give a similar, if not better, capability to a machine or computer. Computer vision is concerned with the automatic extraction, analysis and understanding of useful information from a single image or a sequence of images. Computer vision is an interdisciplinary field that deals with how computers can be made for gaining high-level understanding from digital images or videos. From the perspective of engineering, it seeks to automate tasks that the human visual system can do.

Computer vision tasks include methods for acquiring, processing, analyzing and understanding digital images, and extraction of high-dimensional data from the real world in order to produce numerical or symbolic information, *e.g.*, in the forms of decisions.

So, to do all this pre-processing using image processing tools and other tools is preferred. So, we aim at building a toolbox where we can do all sort of image processing and computer vision processes in one interface.

1.1 About OpenCV

OpenCV (*Open Source Computer Vision*)[1] is a library of programming functions mainly aimed at real-time computer vision. OpenCV 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.

The library good number of optimized algorithms, which includes a comprehensive set of both classic and state-of-the-art computer vision and machine learning algorithms. These algorithms can be used to detect and recognize faces, identify objects, classify human actions in videos, track camera movements, track moving objects, extract 3D models of objects, produce 3D point clouds from stereo cameras, stitch images together to produce a high resolution image of an entire scene, find similar images from an image database, remove red eyes from images taken using flash, follow eye movements, recognize scenery and establish markers to overlay it with augmented reality, etc [2]. It has C++, C, Python, Java and MATLAB interfaces and supports Windows, Linux, Android and Mac OS. OpenCV is written natively in C++ and has a templated interface that works seamlessly with STL containers.

So, we can say that OpenCV is a powerful computer vision toolbox, which we want to explore.

I have used openCV 3.2 and Qt as IDE and written code in C++.

2. Building ToolBox

As Qt is a free and easy to learn open source cross-platform IDE, I have chosen to build my project in it. I have used Qt GUI feature to build the graphical user interface for the toolbox. As I programmed in C++, I had extensively used the object oriented programming concepts to build my toolbox efficiently.

According to logical classification of tasks, I used tabs in Qt GUI to group the tasks accordingly.

As Opencv has 'mat' data type, it's easy to think in terms of processing which is very similar data type in Matlab, where we are most used to!

2.1 General Architecture of toolbox

In the GUI, we have tabs to select the particular group of logical tasks and in each tab we have different features to select. The input and output data is shown in two panels. There are push buttons to load, process and save data as shown in Fig.1

For any task to perform, the steps are pretty simple.

1. Load the image using 'Load image' push button.

2. Select the tab and select the action to perform on the input image.

3. Enter the parameters to get user choice result.

4. Click 'Process' button to process the selected task.

5. Processed data will be shown under output panel.

6. Click 'save image' button to save the current output.

In Qt, we have a 'MainWindow' class from where all the tasks are being executed according to selection. I have written all the tasks in according to logical classification into classes. So, the MainWindow class access all these classes to perform the required operation accordingly. For a better understanding the class diagram of the toolbox is given in Fig. 2.

I have implemented modular programming approach, where in all tasks are divided according to the modules and solved.

Opencv reads images as 'mat' format, but I faced initial difficulty to show the images on QLabel. So, firstly we need to convert image to QImage format and then we can show the image on QLabel using pixmap.

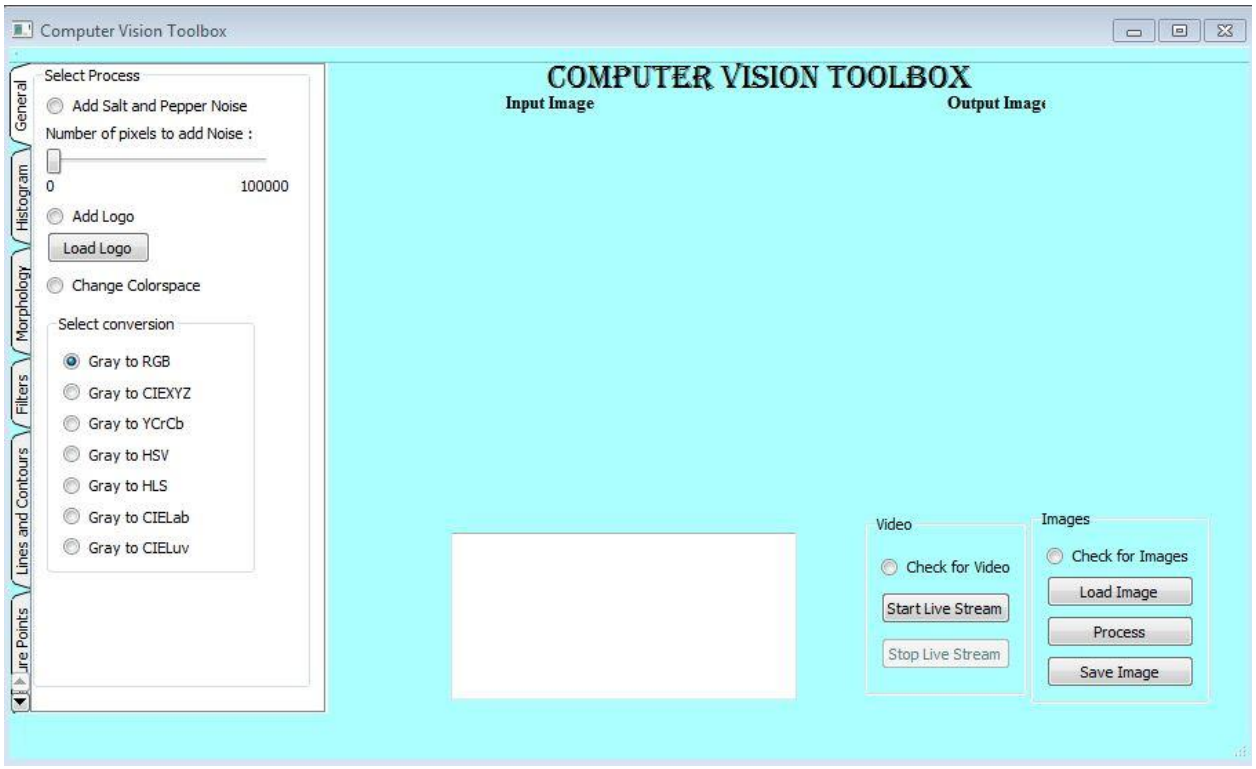

Fig.1 The GUI of tool box

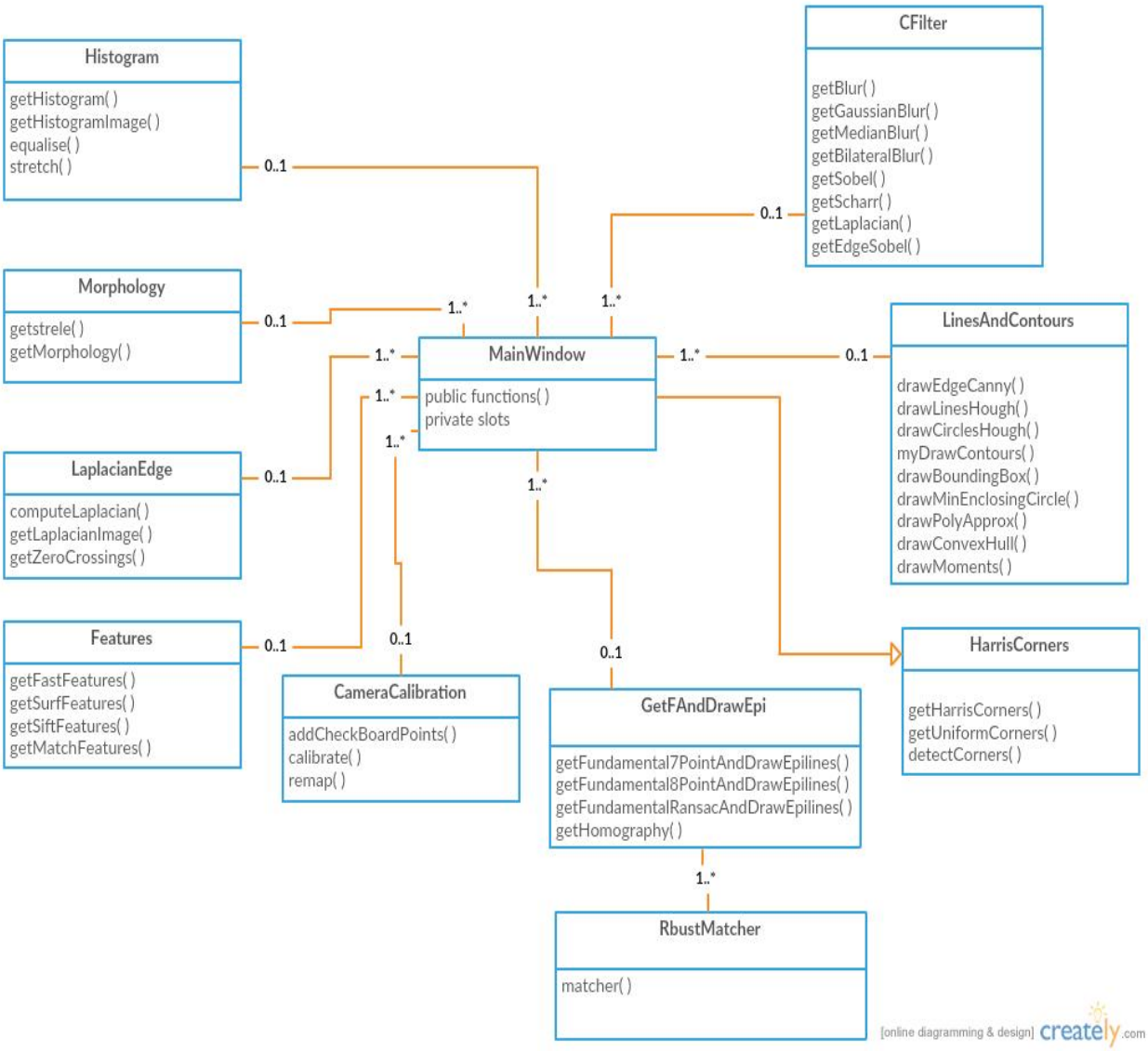

Fig. 2 Class Diagram of toolbox

2.2 Processing Tasks for Images

2.2.1 General Tab

2.2.1.1 Add noise

I have implemented function to add salt and pepper noise. the result is as shown in Fig.3

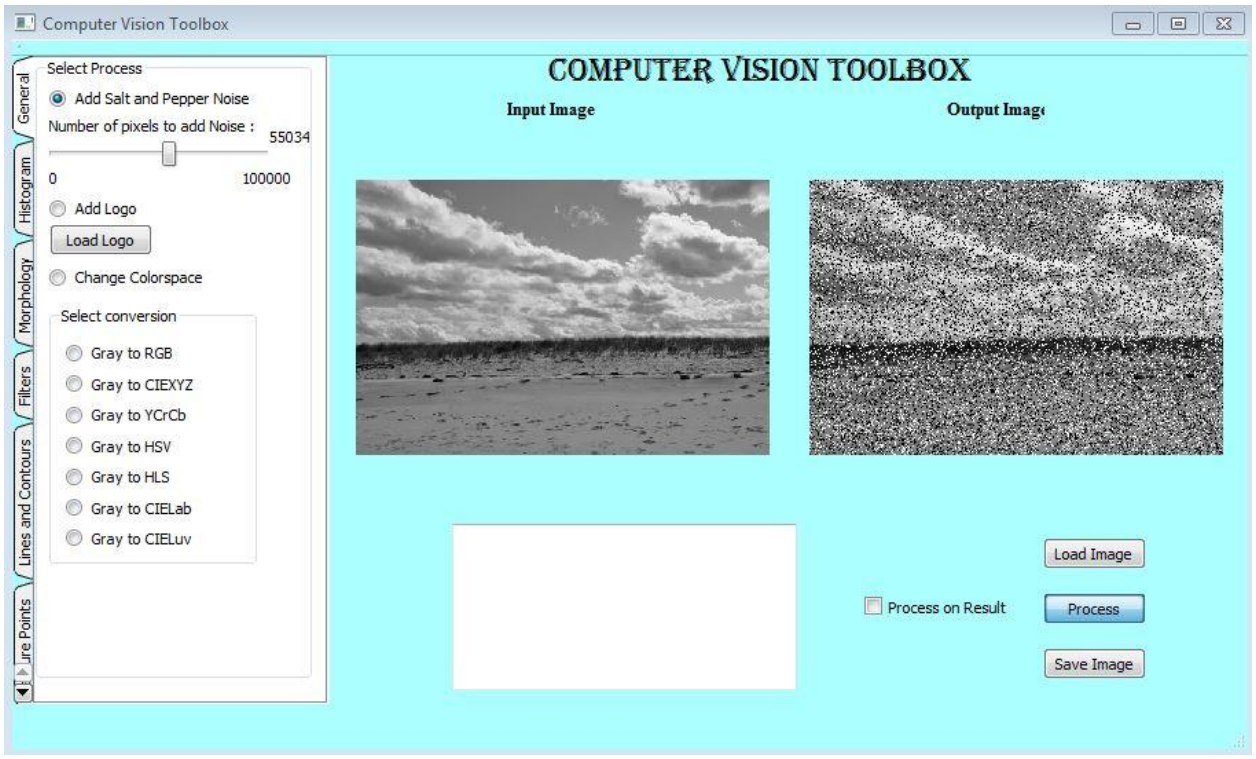

Fig. 3 Salt and pepper noise

2.2.1.2 Add Logo

We need to provide logo by clicking 'add logo' push button , where we can select the logo we want to add to the image. The result is as shwon in Fig. 4.

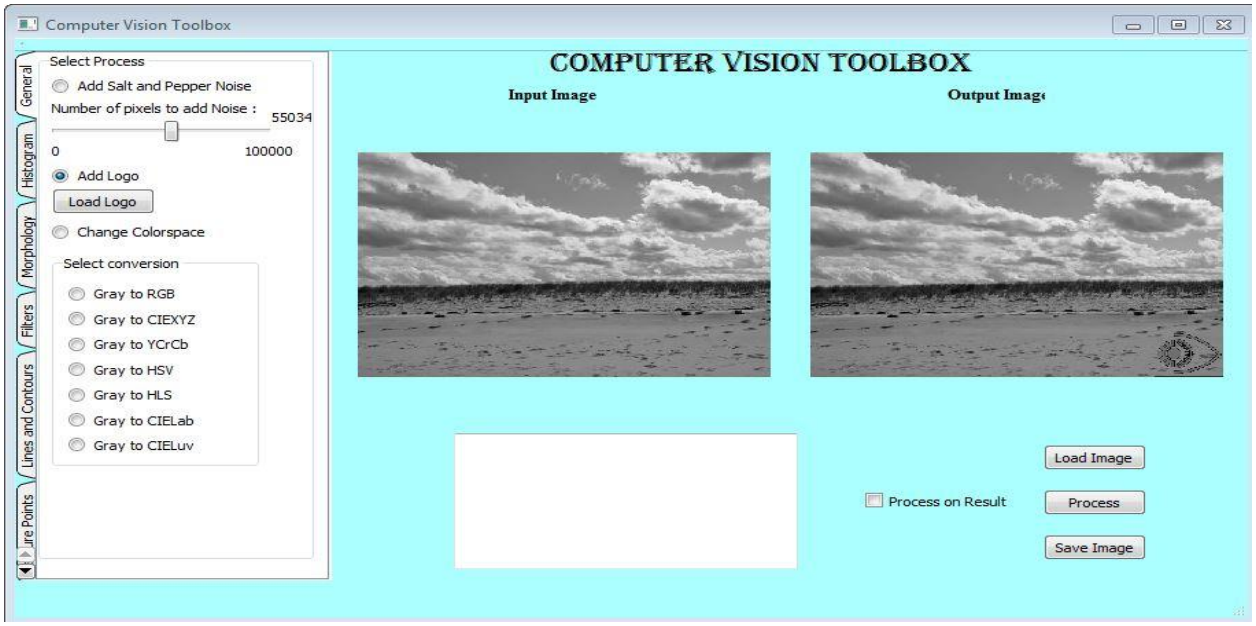

Fig. 4 Add logo to image

2.2.1.3 Change Color Space

We can change color spaces in opencv easily using 'cvtColor'. Seven colorspace conversions are implemented as seen below. The result is as shown in Fig. 5.

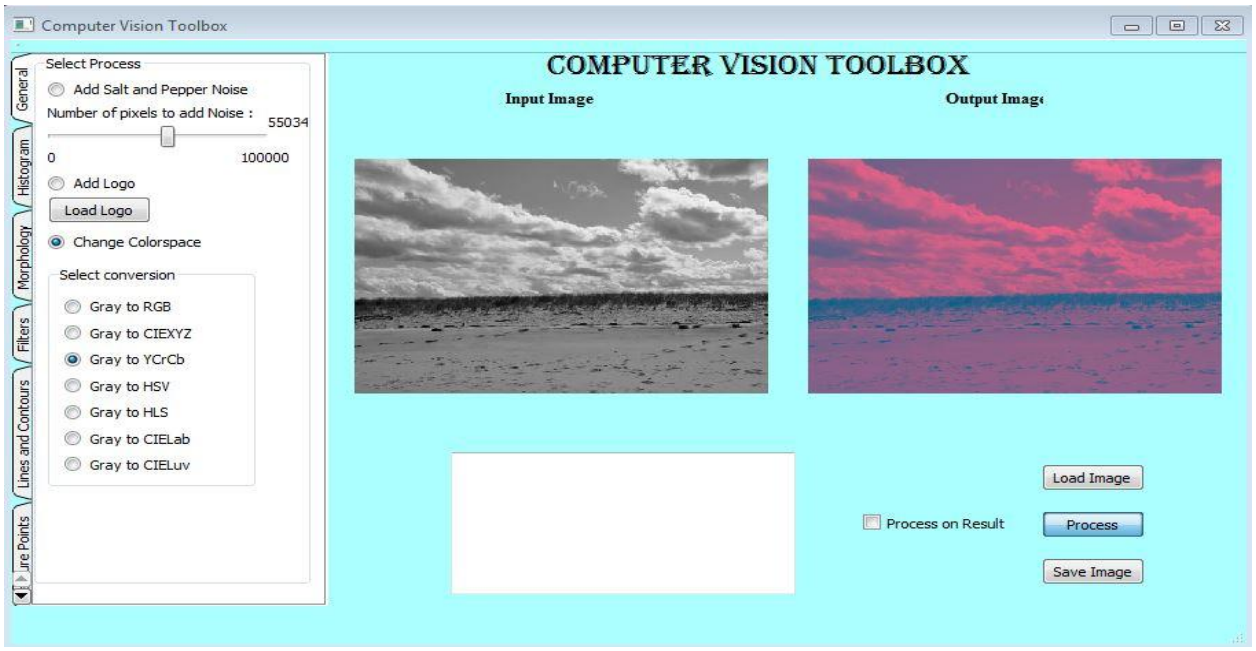

Fig. 5 Gray2YcrCb

2.2.2 Histogram

The operations on images to get histograms are implemented in Histogram class.

The results for plotting histogram of image and equalizing and stretching histogram is as shown in Fig. 6,7,8.

Fig. 6 histogram of an image

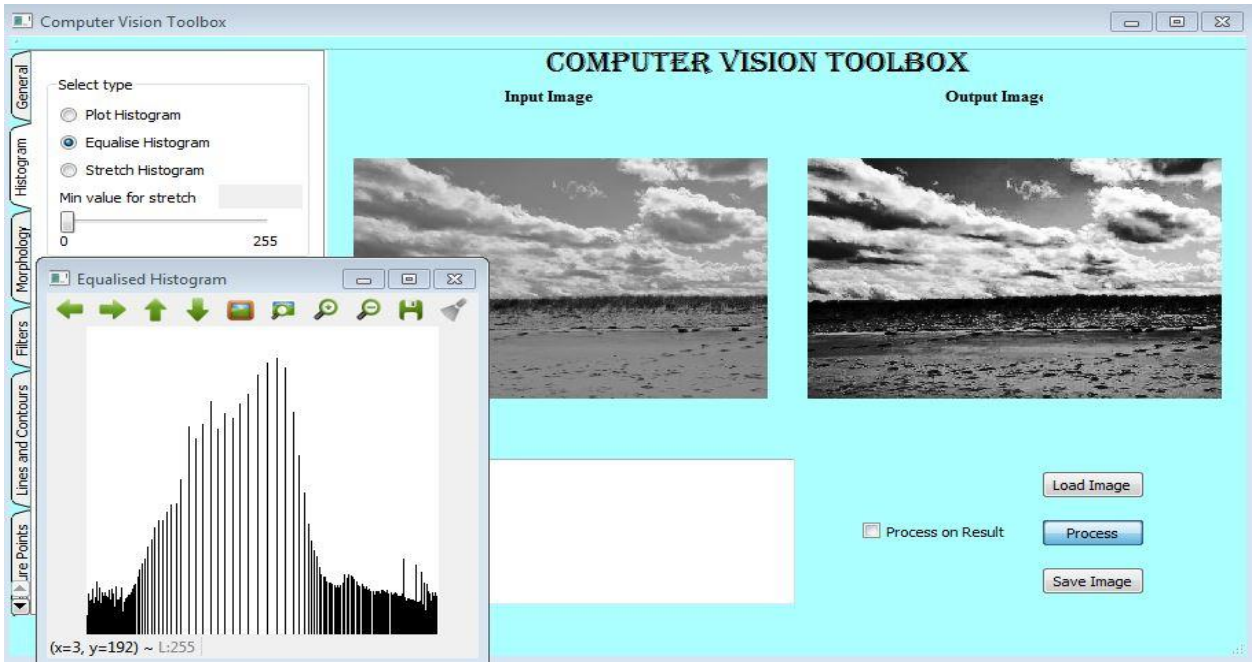

Fig. 7 Equalised histogram

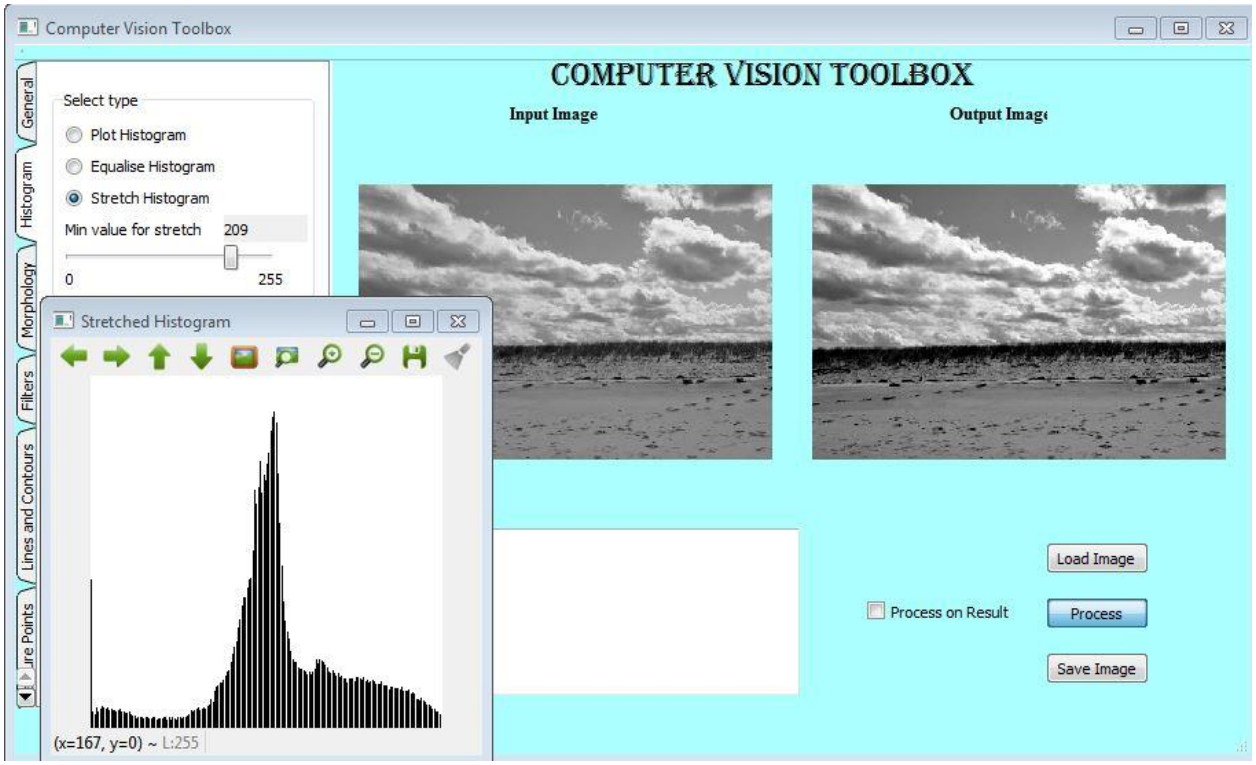

Fig.8 Stretched histogram

2.2.3 Morphology

The operations for morphological processing are implemented in Morphology class. We have options to select for structuring element and also to select different morphological operations and also number of times operations needed to be performed.

The result are as shown in Fig. 9, 10 and 11.

Fig. 9 Morphological gradient

Fig. 10 Morphological gradient on binary image

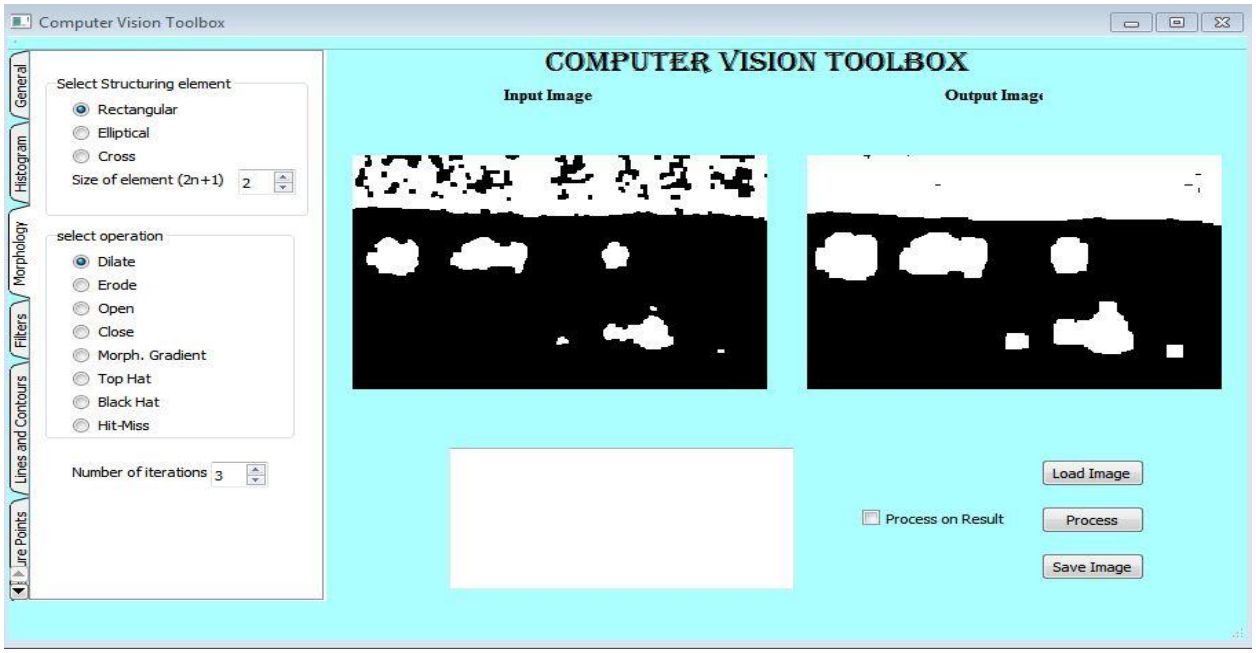

Fig. 11 Dilation on binary image

2.2.4 Filtering.

The filtering actions are implemented under CFilter class, which includes both LPF and HPF actions. LPF such as average filter, Gaussian filter, median filter, bilateral filters are implemented. HPF such as laplacian, sobel, etc are implemented.

The results are as shown in fig. 12.

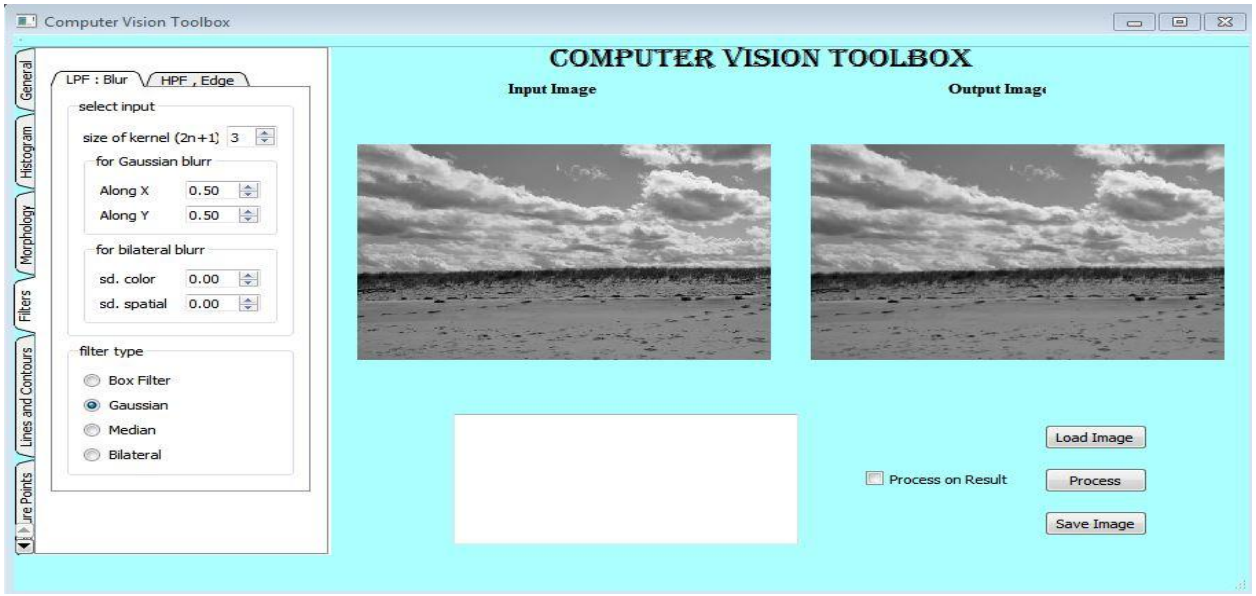

Fig. 12 Gaussian Blur

2.2.5Lines and Contours

All the tasks under lines and contour tab are implemented under LinesAndContours class.

2.2.5.1 Edges

The result of edge detection is as shown in Fig.13

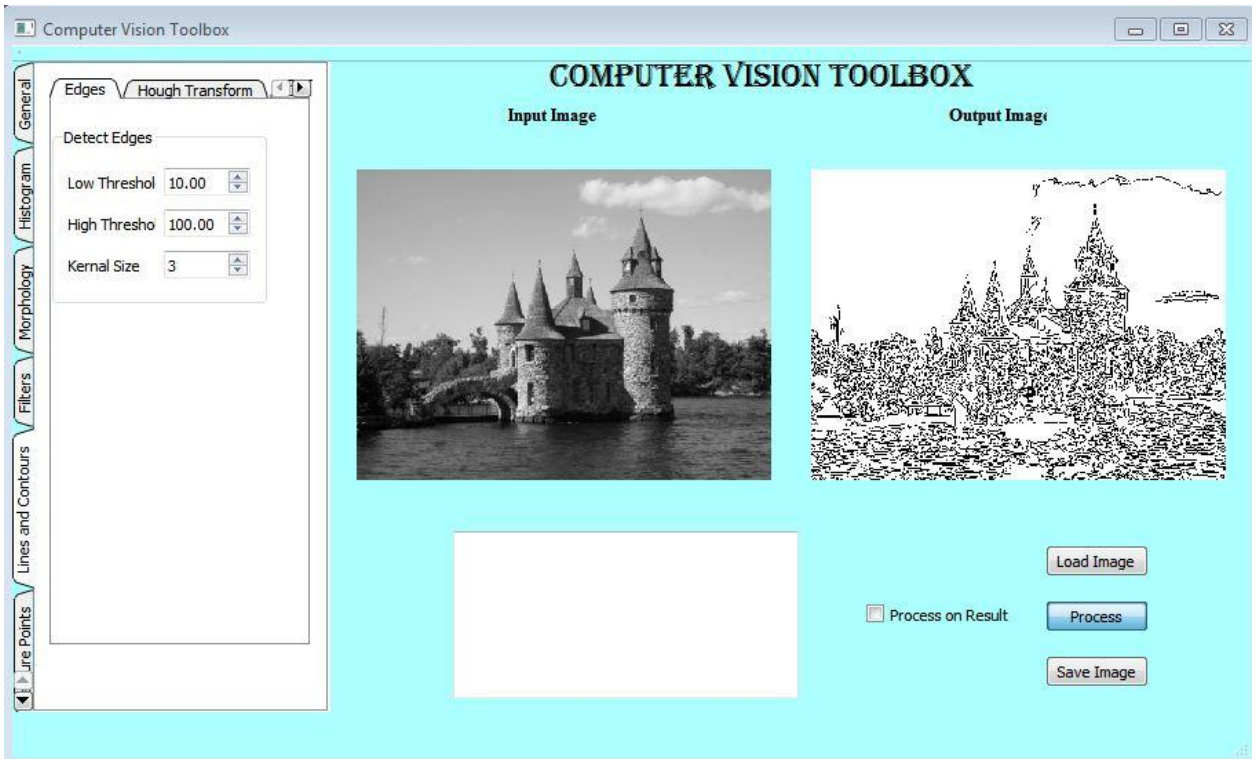

Fig. 13 Edge detection

2.2.5.2 Hough transform

Hough transform is used to detect lines and circles in an image.

the result is as shown in figs. 14,15.

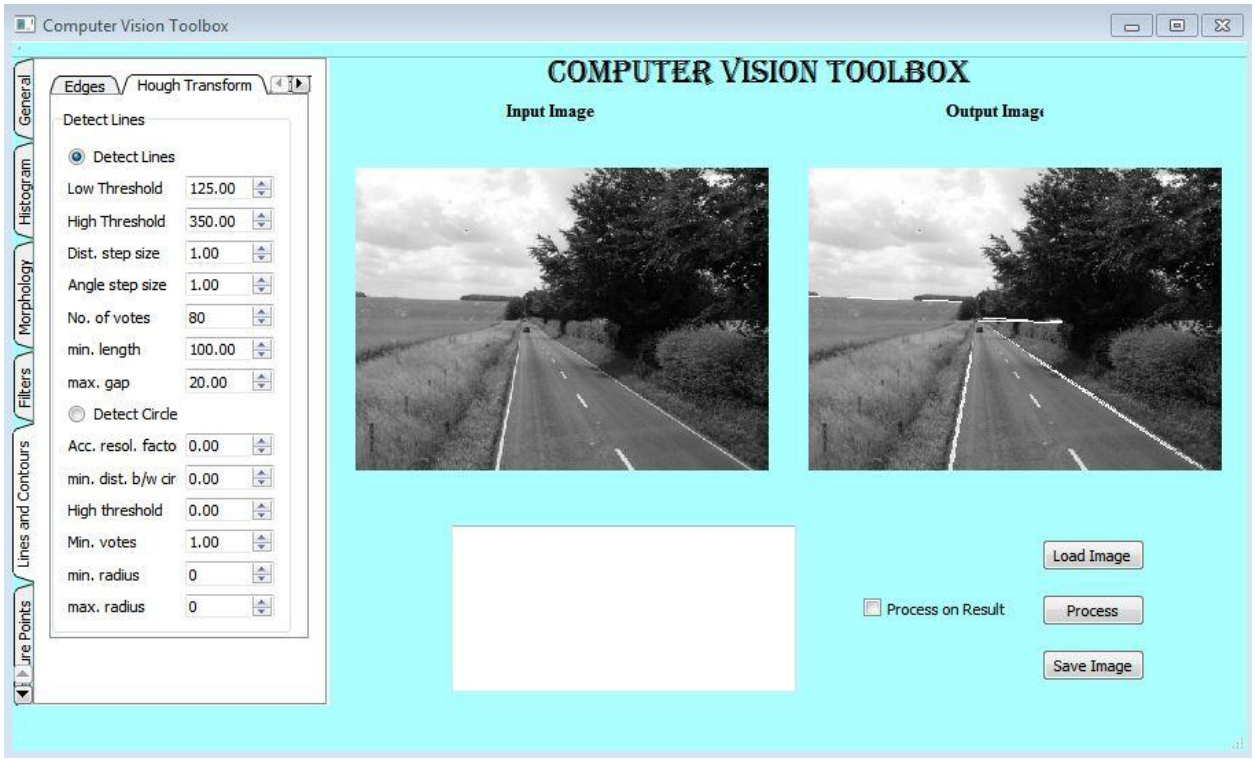

Fig. 14 Hough lines

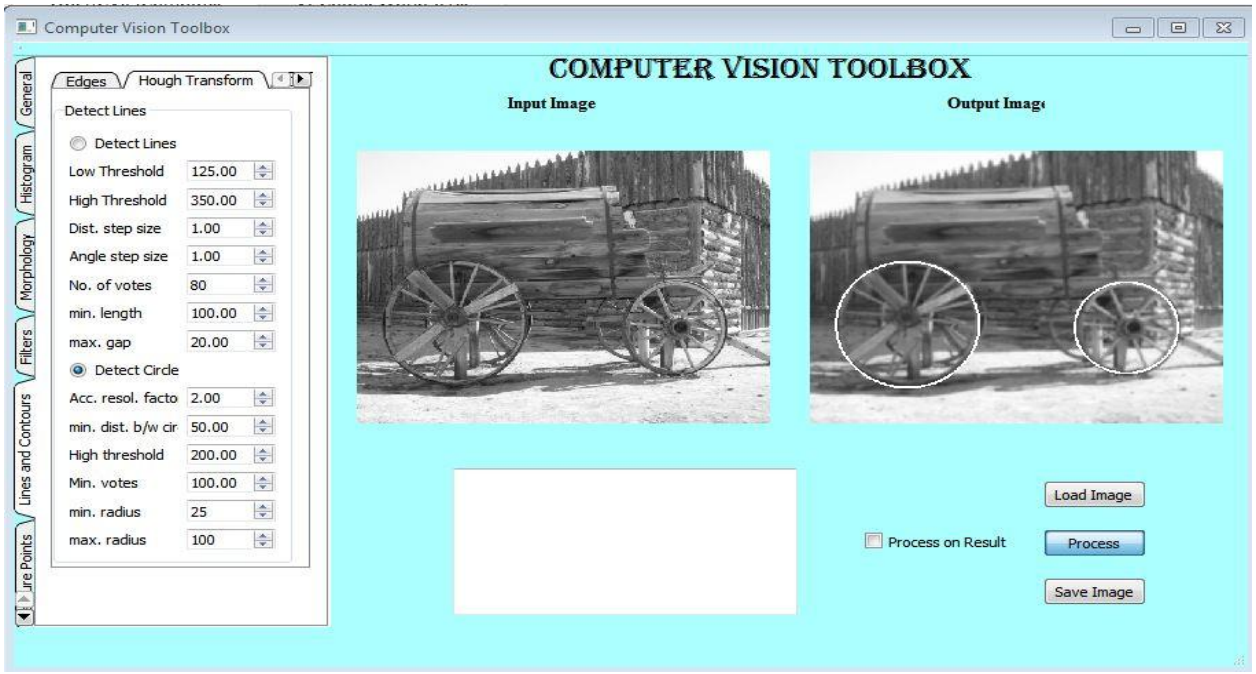

Fig. 15 Hough Circles

2.2.5.3 Contours

Contours and drawn on binary images with minimum and amximum contour length as parameters.

The result is as shown in fig.16.

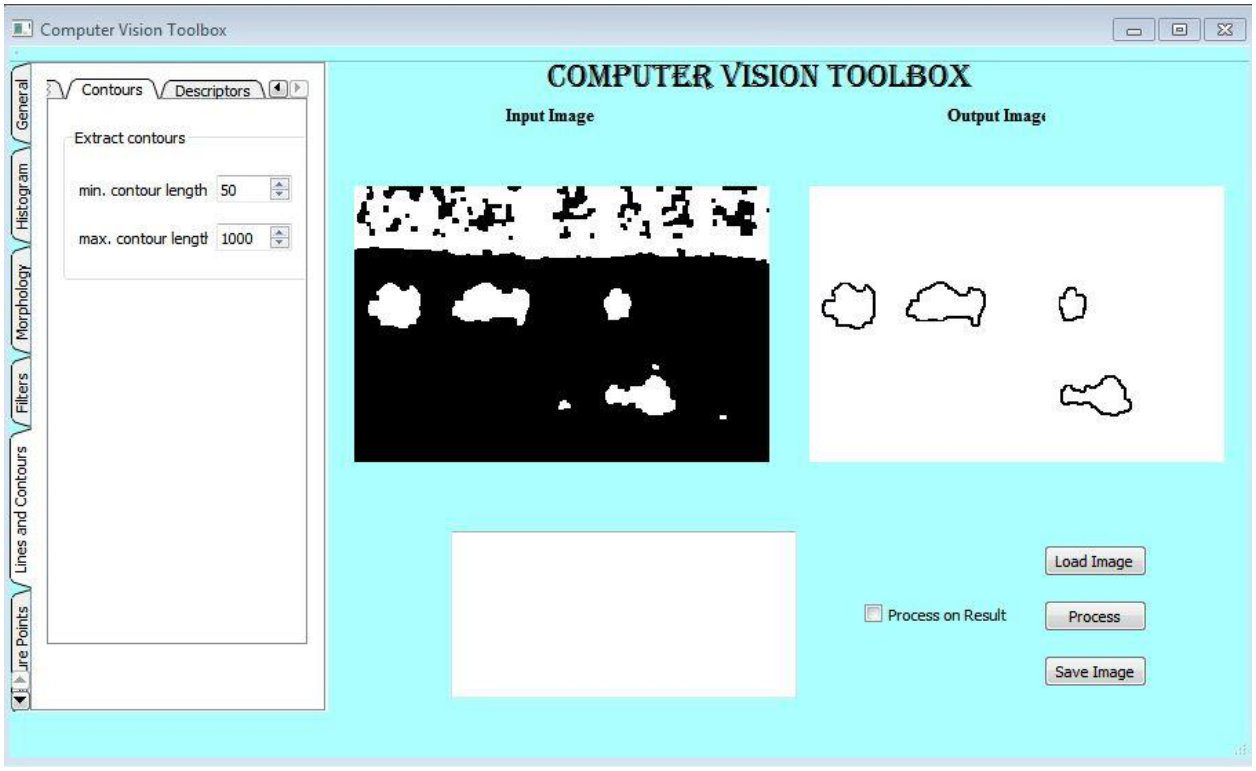

Fig. 16 Contours

2.2.5.4 Shape descriptors

Different types of shape descriptors are impended such as bounding box, minimum enclosing circle, polygon enclosing, convex hull and centroids.

The results are as shown in Figs. 17, 18 and 19.

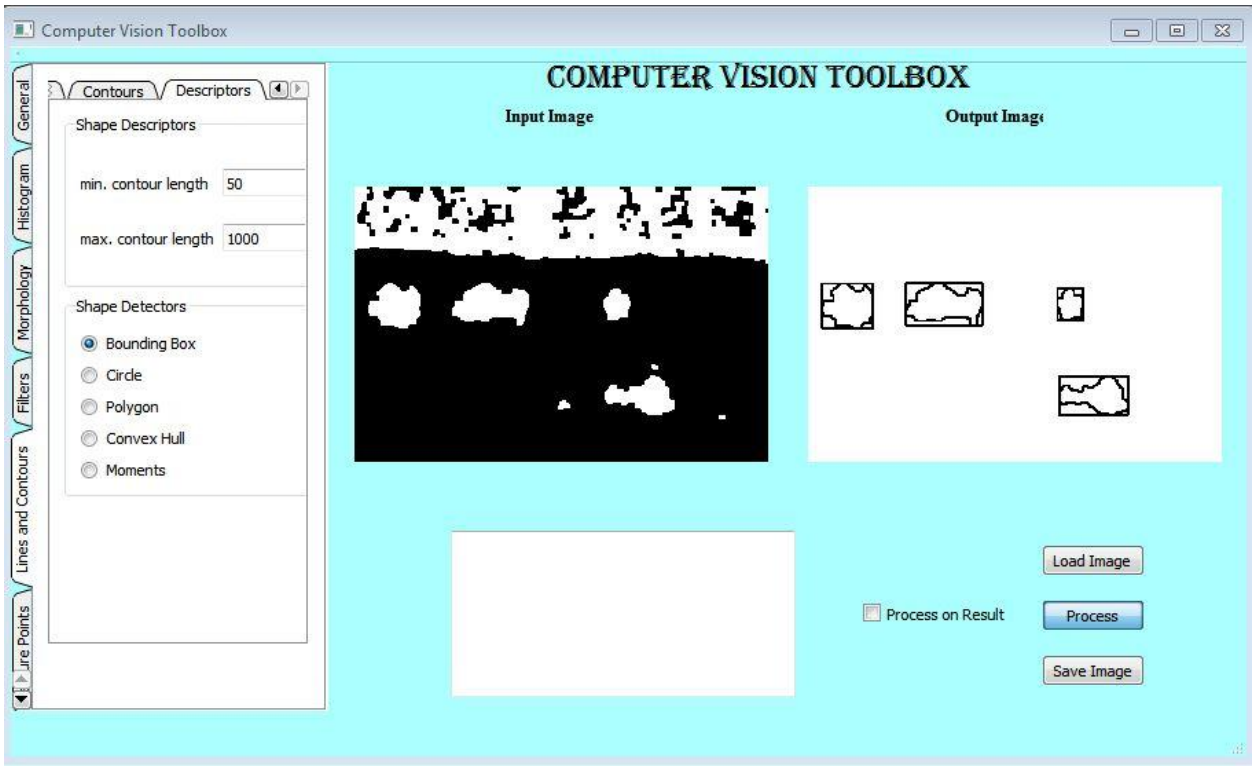

Fig. 17 bounding box

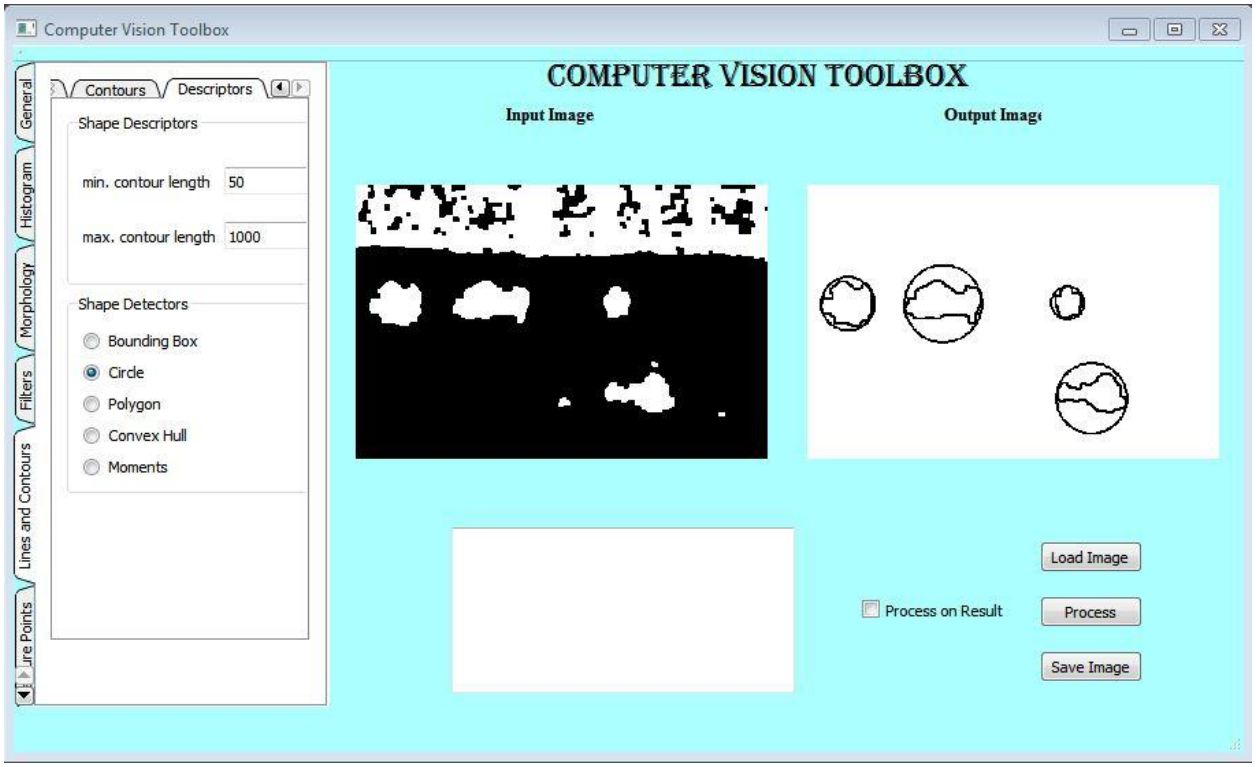

Fig. 18 Minimum enclosing circle

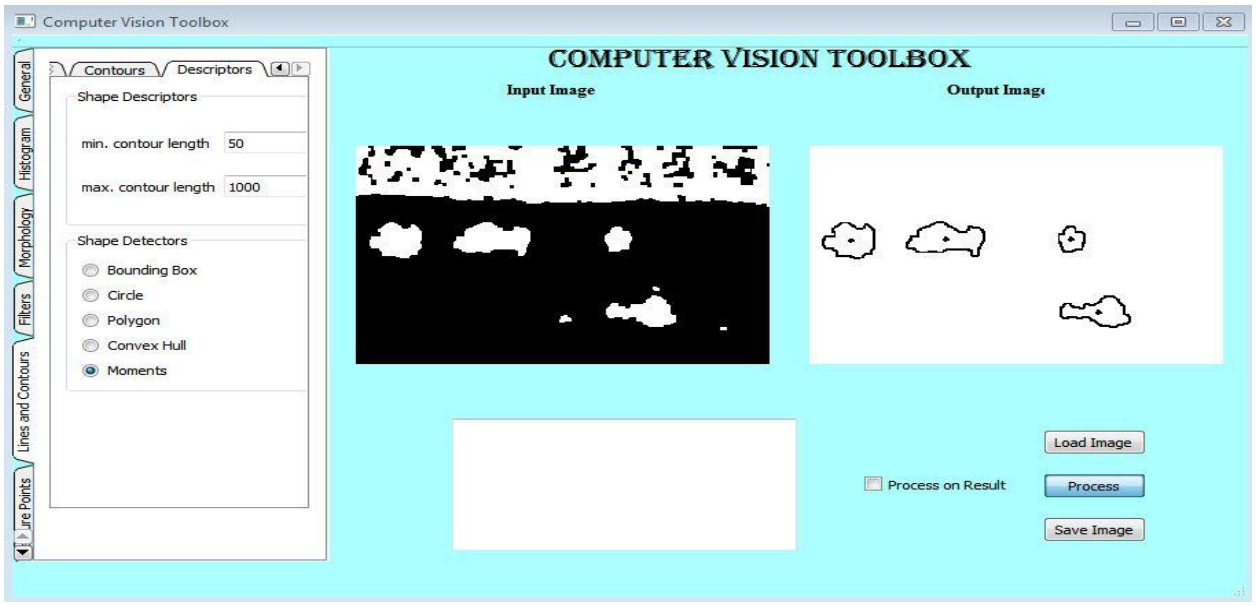

Fig. 19 Centroids of shapes.

2.2.6 Features points

2.2.6.1 Corner points

The corner point detection is implemented under HarrisCorner class.

The result of corner detection is as shown in Fig. 20.

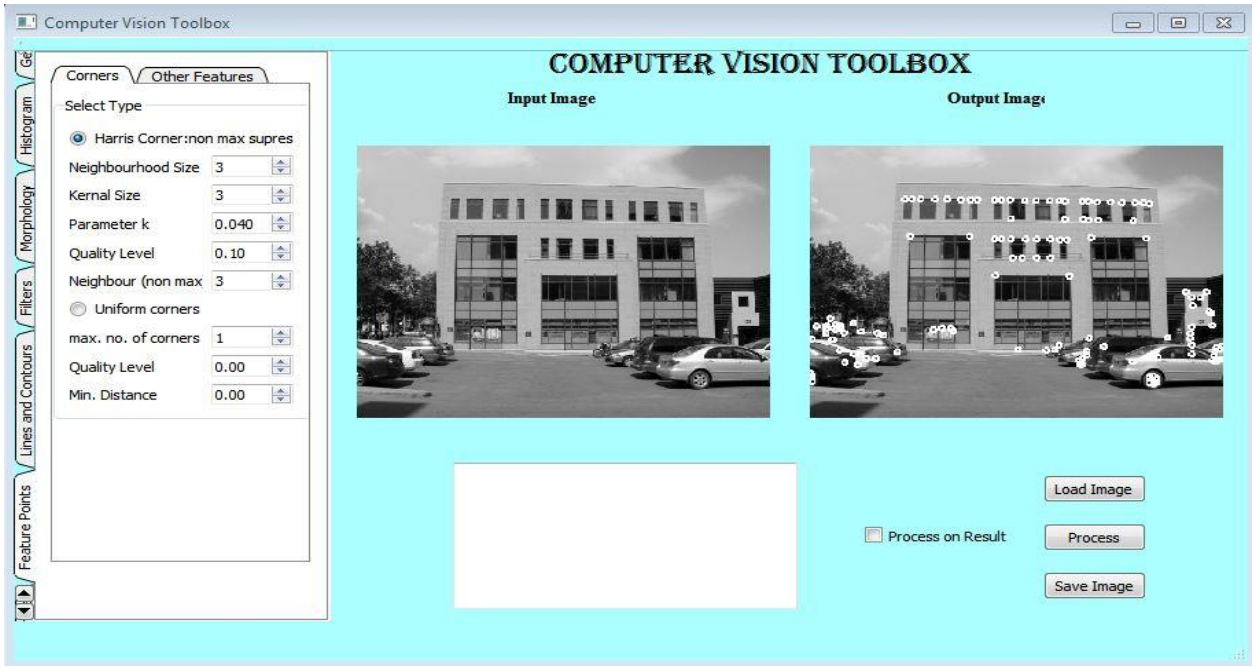

Fig. 20 Corners using Harris corner detector

2.2.6.2 Other features (FAST,SURF, SIFT)

The feature detection and description plays an important role in finding matches or corresponding points which plays significant role in computer vision. The feature are computed under Features class.

The result are as shown in Fig.21, 22 and 23.

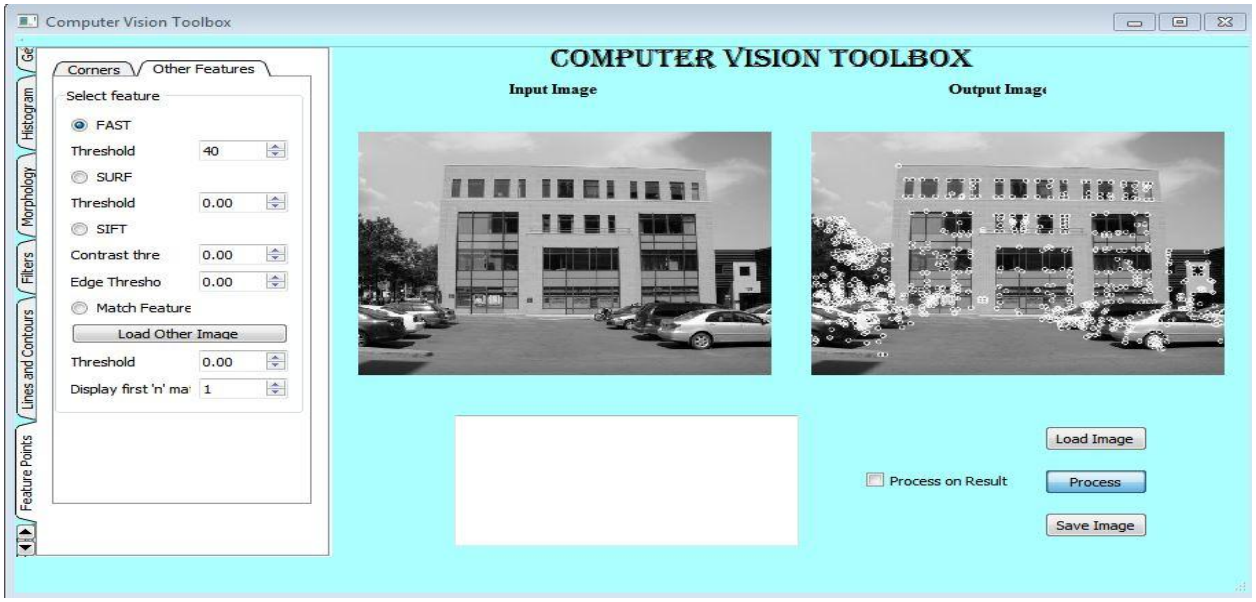

Fig. 21 FAST Features

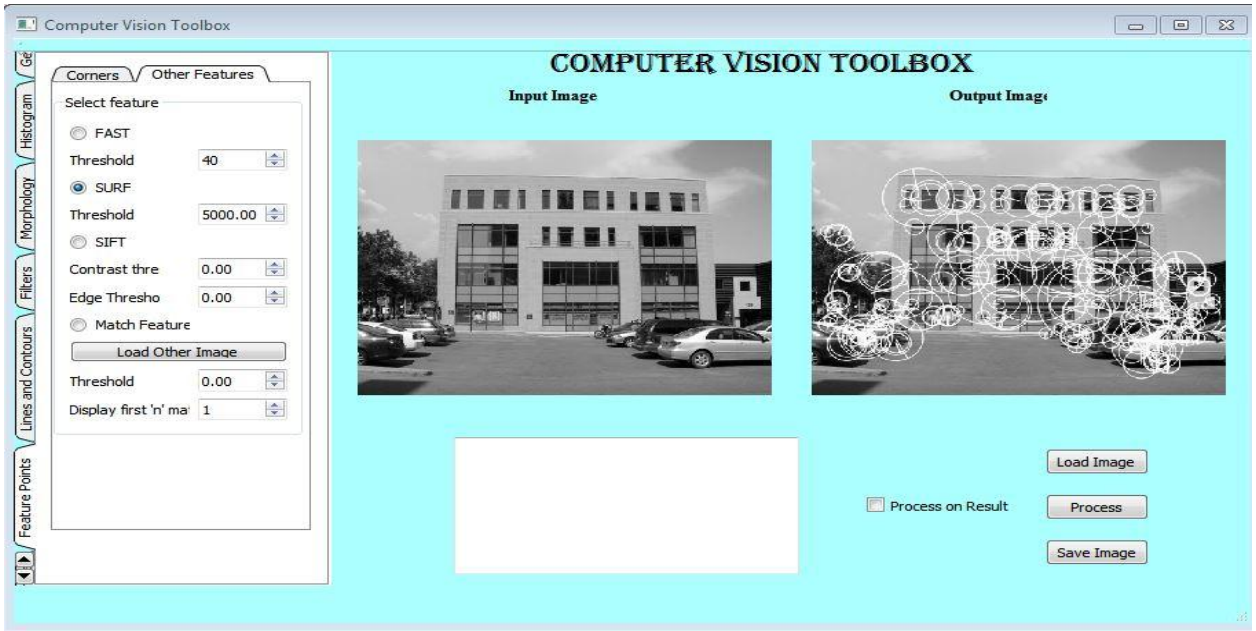

Fig. 22 SURF Features

Fig. 23 SIFT Features

With Surf or Sift features, we can load other image and compute matches between two images.

The result as shown in Fig. 24.

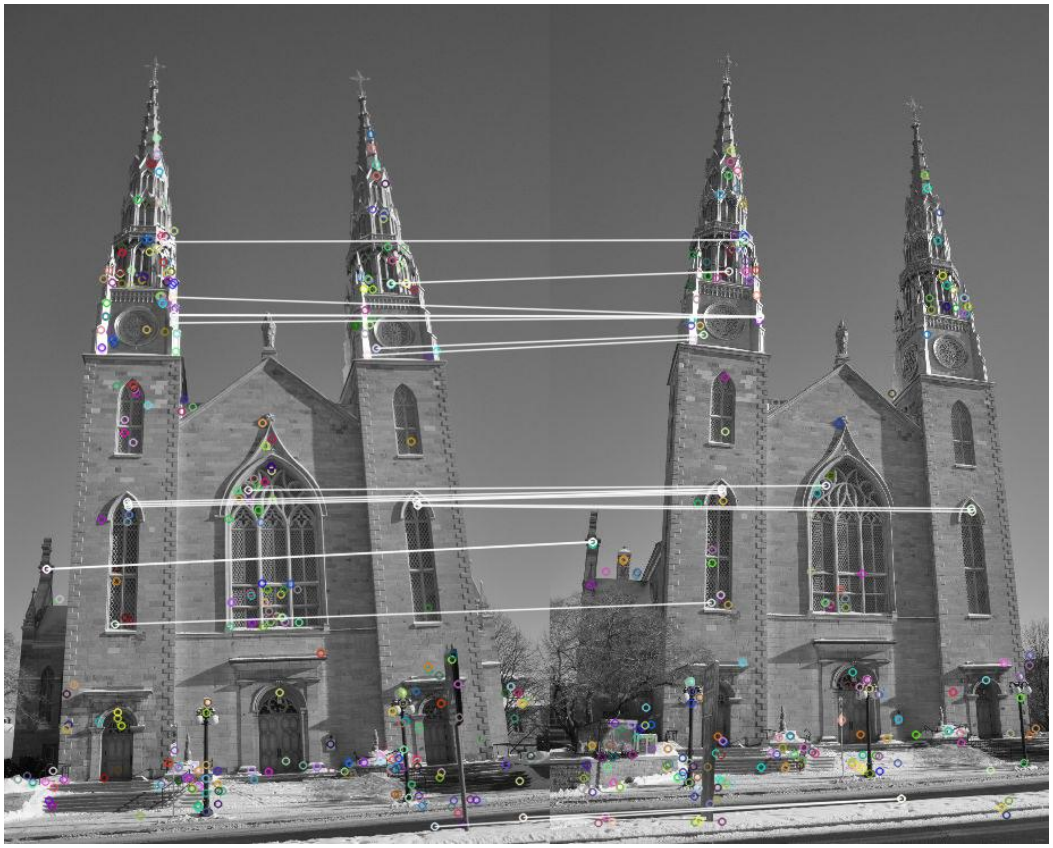

Fig. 24 matches using Surf features

2.2.7 Camera and other

2.2.7.1 Camera calibration

Camera calibration is an important task in computer vision. A calibration pattern is taken and images are captured using a camera to calibrate the camera. The images are processed accordingly to find the camera calibration parameters. These parameters can be used to undistort any distorted images taken from the camera.

The results is as shown in fig. 25.

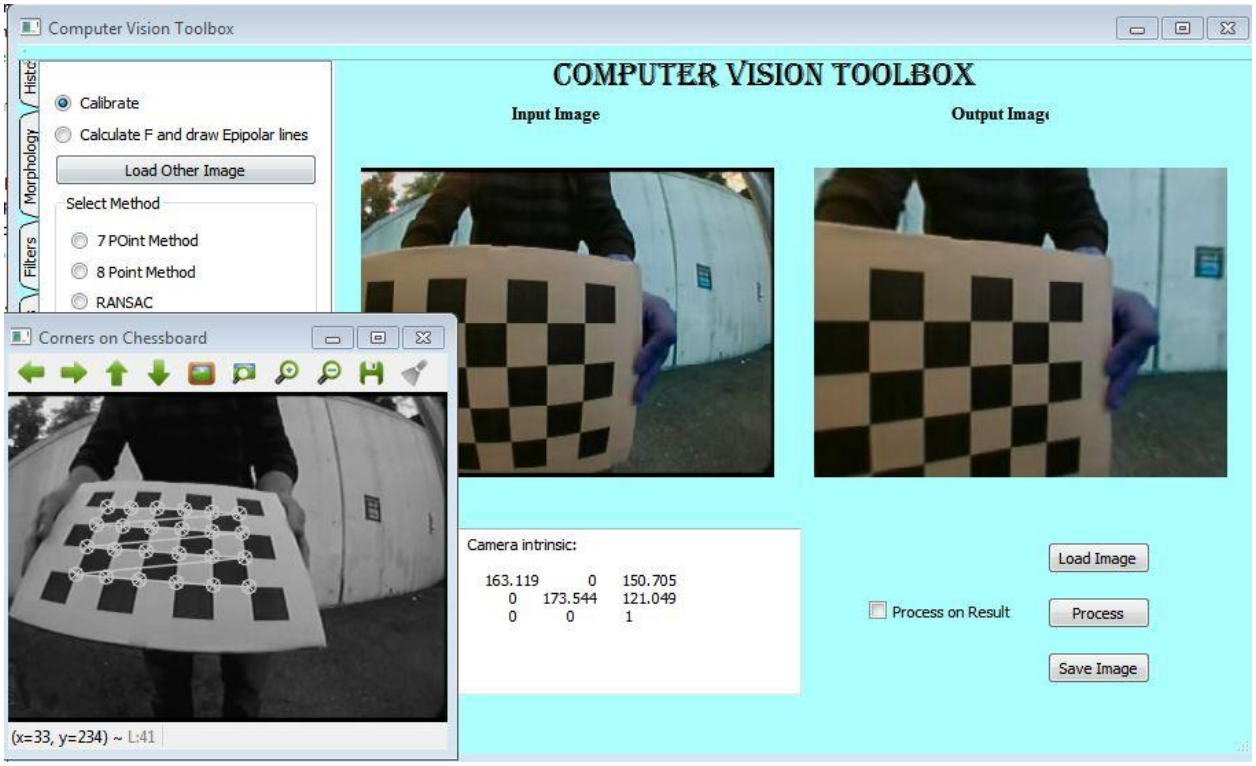

Fig. 25 Camera calibration and undistorted image using calibration

2.2.7.2 Fundamental matrix

By finding the matching points between two images using Surf or Sift, we can make use of the matching points to get the fundamental matrix between two views.

I have implemented three types : 7-point, 8-point and Ransac.

By computing the fundamental amtrix, we can check the accuracy by drawing epipolar lines. From epipolar geometry, the matching point of first image in second image should lie on the epipolar line of the point in first image.

The result is as shown in Fig. 26.

Fig. 26 Epipolar lines in firsta nd second images

2.2.7.3 Homography

From the corresponding points we can also calculate the homography matrix and use the homogrpahy to mosaic two images.

The results area s shown in Fig. 27-30.

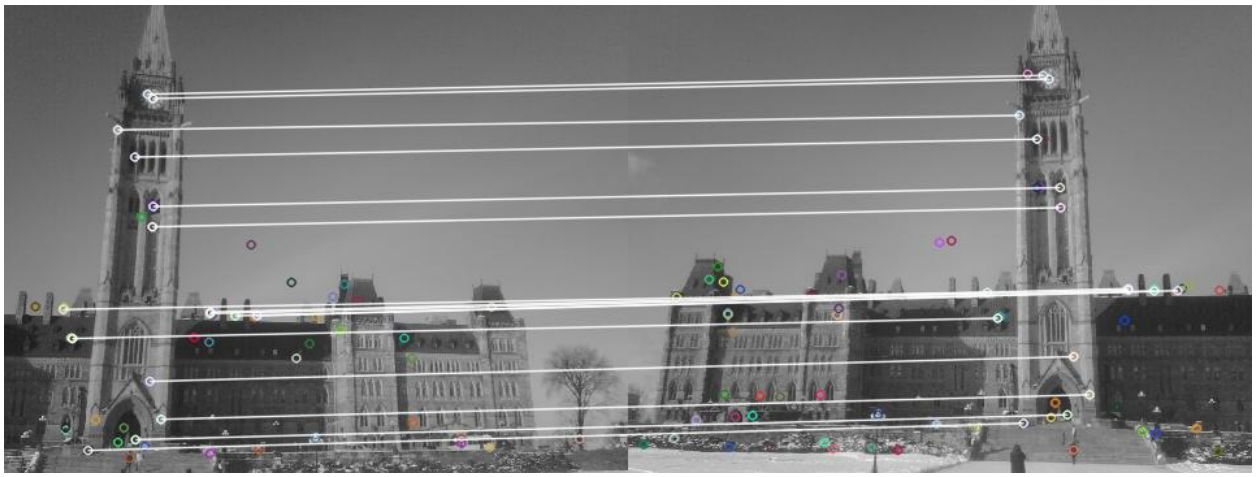

Fig. 27 matching points

Fig. 28 Homography Points in two images.

Fig. 29 Warping after homography (mosaic)

2.3 Processing Tasks for Live Stream

I thank my supervisor for motivating me to do the tasks on live stream.

The following tasks are implemented on live stream data.

Add salt and pepper noise, add logo, change color spaces, plot histogram, morphological operations, low pass and high pass filtering operations like, box filter, gaussian filter, median filter, sobel filter, detect edges using sobel, laplacian filters.

In the gui, there is check box before starting live stream, which disables all the options not available for live stream.

3. Conclusion

This project has given me a great exposure of the OpenCV as a powerful computer vision and image processing tool. This has also given me a great experience to learn many new things regarding GUI and epipolar geometry.

Future Work

I would work on this toolbox further to develop it more sophisticated way and including more features like a history of processes and undo and redo a process like we do in commercial toolboxes.

Acknowledgement

I sincerely thank my supervisor, Dr. Shabayek for motivating me to work on live stream video. The cookbook (I consider as Bible) for OpenCV "OenCV 2 Computer Vision Application programming Cookbook" has helped me a lot to complete this toolbox. I sincerely thank Robert Laganiere and others for their contribution to this book, which is helping lot of budding engineers like me!

References

[1] OpenCV, wikipedia

[2] openCV , official website

[3] OenCV 2 Computer Vision Application programming Cookbook

[4] Opencv forums

[5] Stackoverflow forum