Computer Vision Course Tasks

In this Repository we present a variety of Image Processing Techniques implemented from scratch using `Python` with help of some helpful packages.

Table of contents

Installation

To install the required libraries and dependencies, open your terminal in the repository directory and run this command:

```
pip install -r requirements.txt
```

Script's Components

requirements.txt contains the versions of each libraries, if already installed the installation will be skipped:

- Scipy
- Numpy
- Pyqtgraph
- PyQt5
- opencv-python
- Pillows
- Matplotlib
- scikit_learn

Usage

The **GUI** is composed of many tabs; each tab contains some push buttons, combo boxes or sliders, input texts and some widgets to view the images.

Each category of implemented Algorithms is displayed in a separate tab in the GUI.

Simply you could load the image you want to apply the algorithm on via push buttons, adjust the required parameters then apply the selected algorithm.

Here's the view of the UI tabs without loading any images or applying any algorithms.

▶ Details

Image Processing

In this section we present some implementations such as adding noise to image, filtering the added noise, viewing different types of histograms, applying threshold to image and hybrid images.

1. Adding Noise To Image

We implemented 3 types of noise: `Uniform`, `Gaussian` and `Salt & Pepper`. In each type, you could adjust some parameters such as **Signal-To-Noise Ratio (SNR)** and **Sigma** to show different outputs.

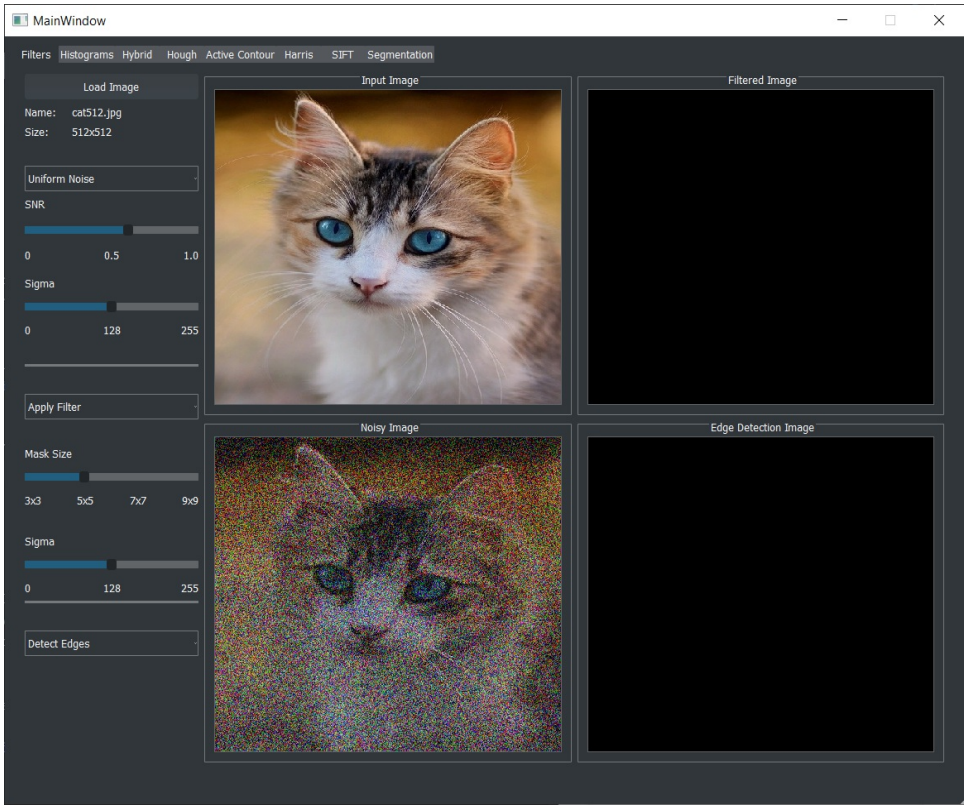
The results below were taken with the following setup:

Noise parameters:

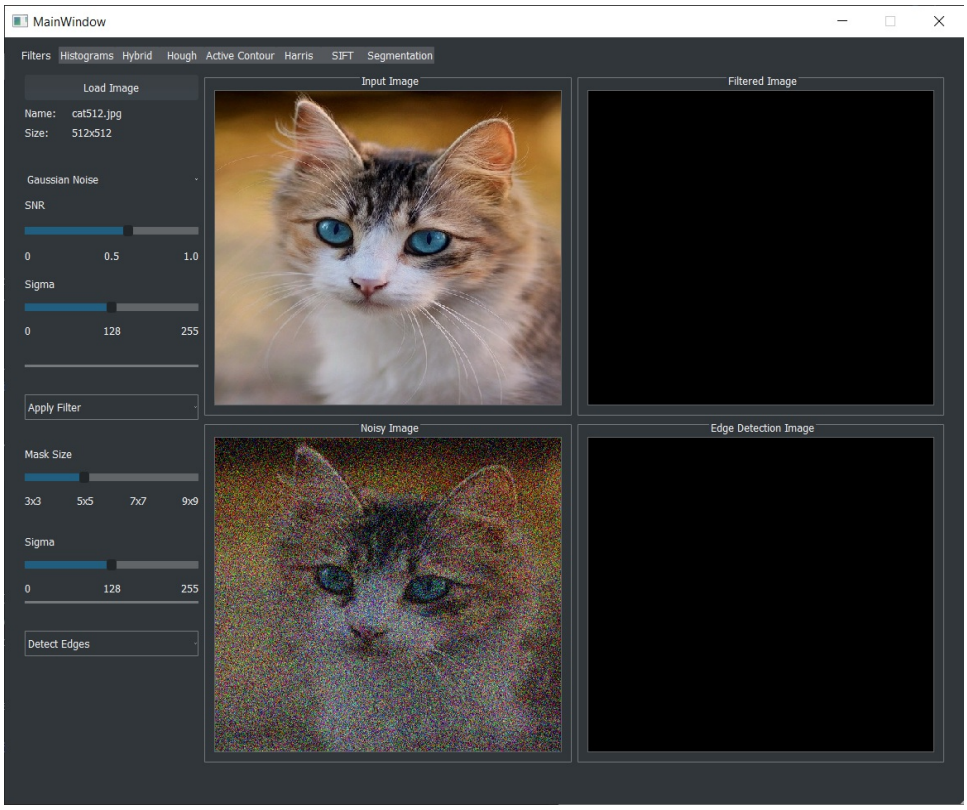
- `SNR` = 0.6
- `Sigma` = 128 (For Gaussian Noise Only)

The whole GUI is displayed to show you the difference between the original and the noisy image.

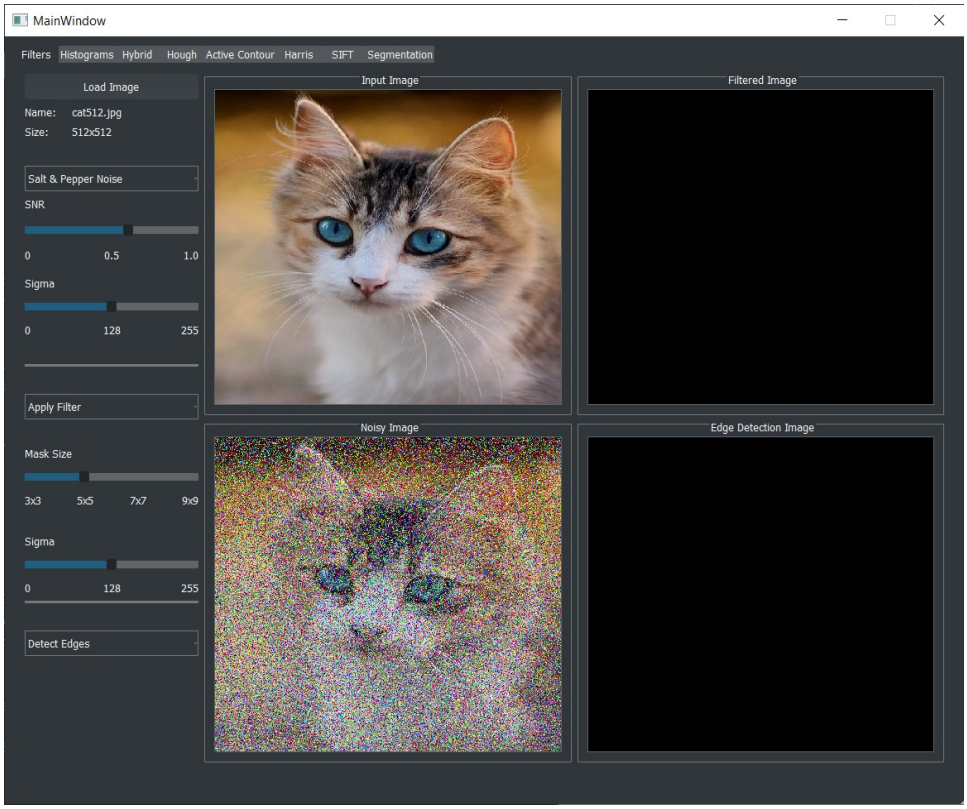
1.1 Uniform Noise



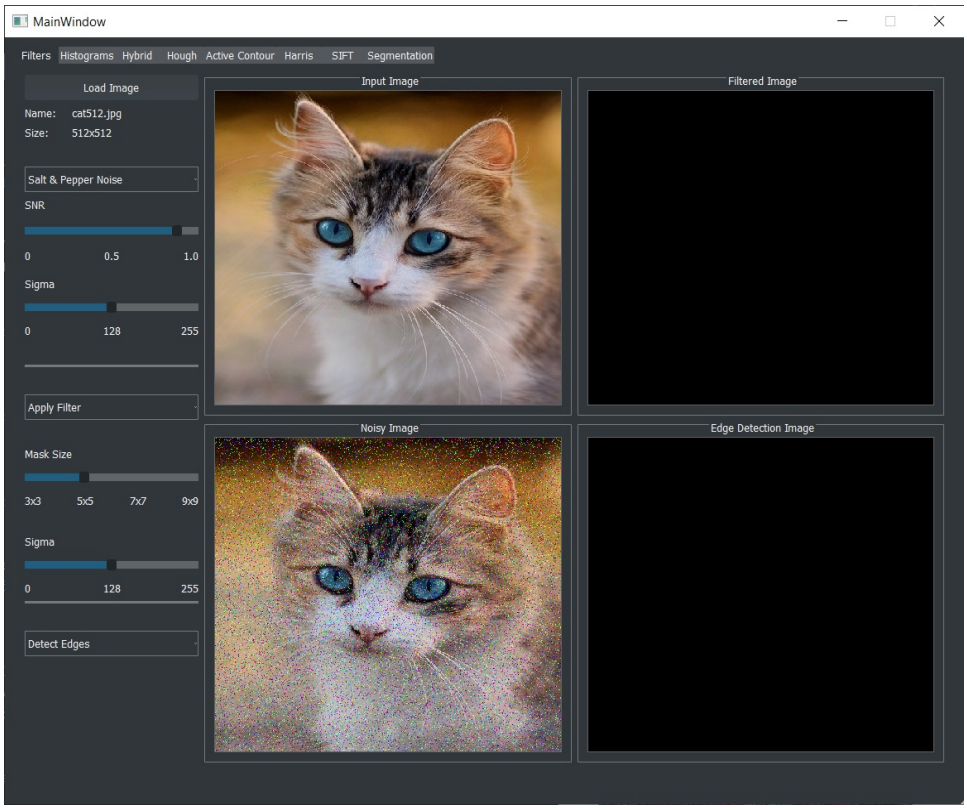
1.2 Gaussian Noise



1.3 Salt & Pepper Noise



To decrease amount of noise, move the SNR slider a little, and this would be the new output with `SNR = 0.9`, which means only 10% of the image is noise.



2. Image Filtering

We implemented 3 types of Filters: `Average`, `Gaussian`, and `Median` filter. In each filter, you could adjust some parameters such as **mask size** and **Sigma** to show different outputs.

The results below were taken with the following setup:

Noise parameters:

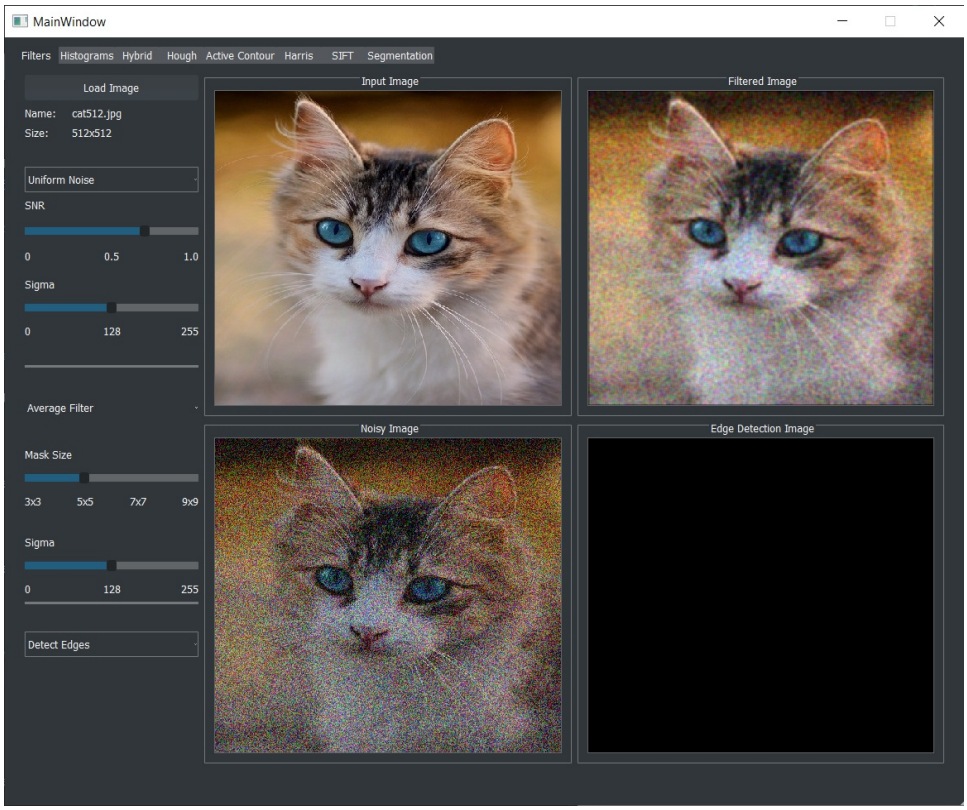
- `SNR` = 0.6
- `Sigma` = 128 (For Gaussian Noise Only)

Filter Parameters:

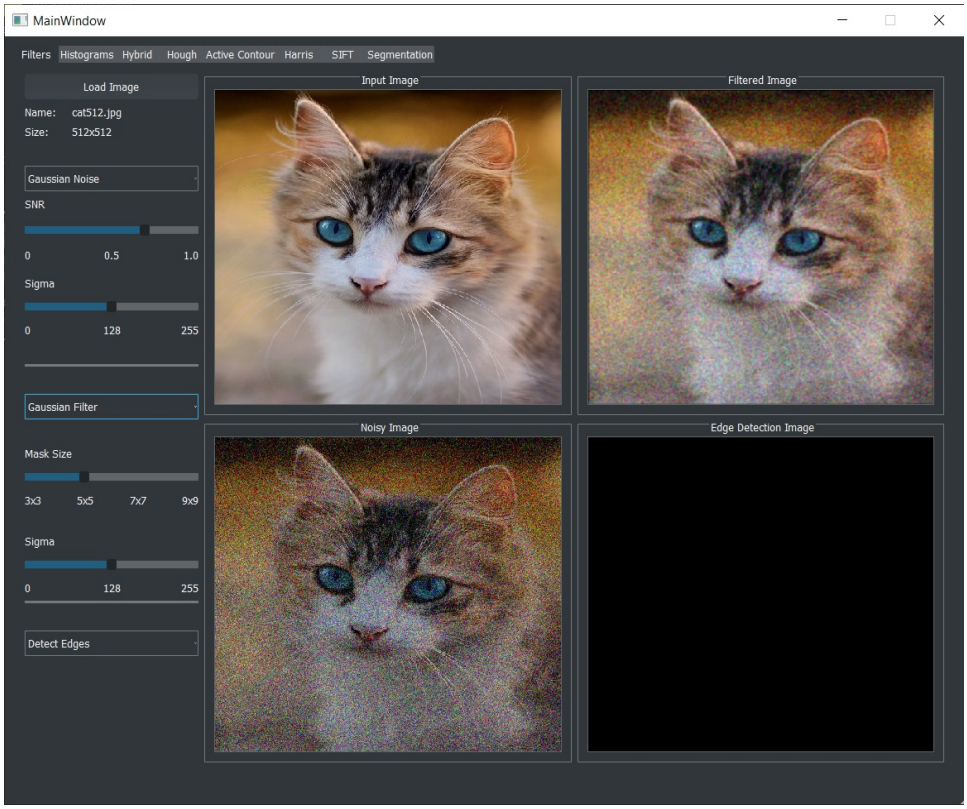
- `Mask Size`: 5x5
- `Sigma` = 128 (For Gaussian Filter Only)

The whole GUI is displayed to show you the difference between the noise and the filtered image.

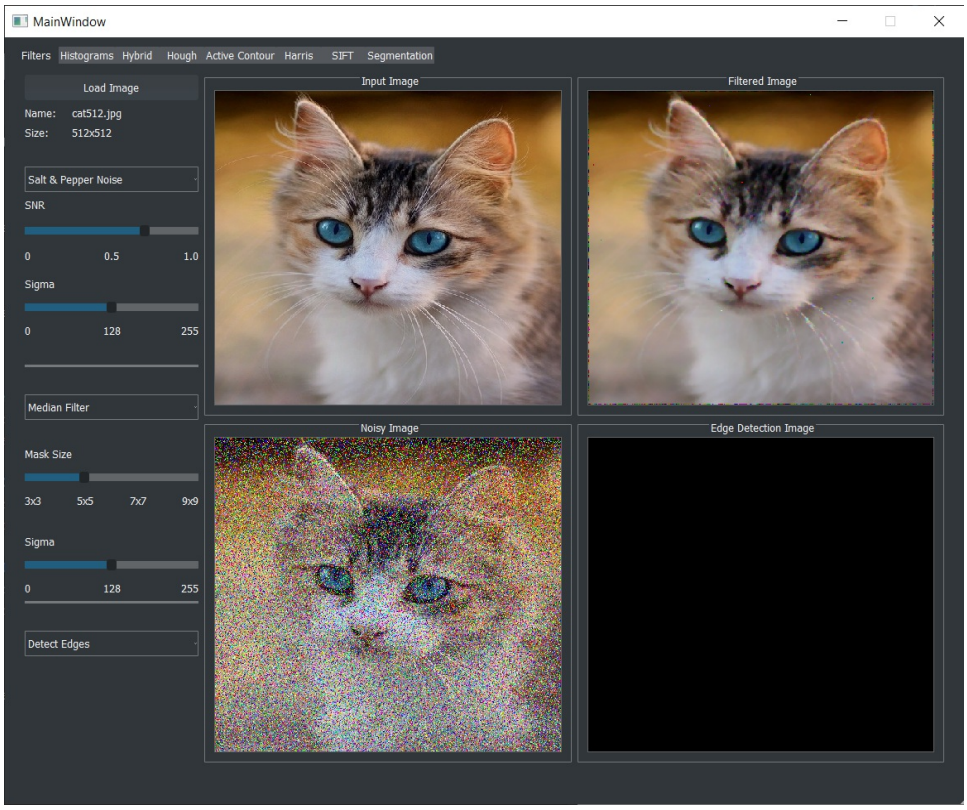
2.1 Average Filter Applied on Uniform Noise



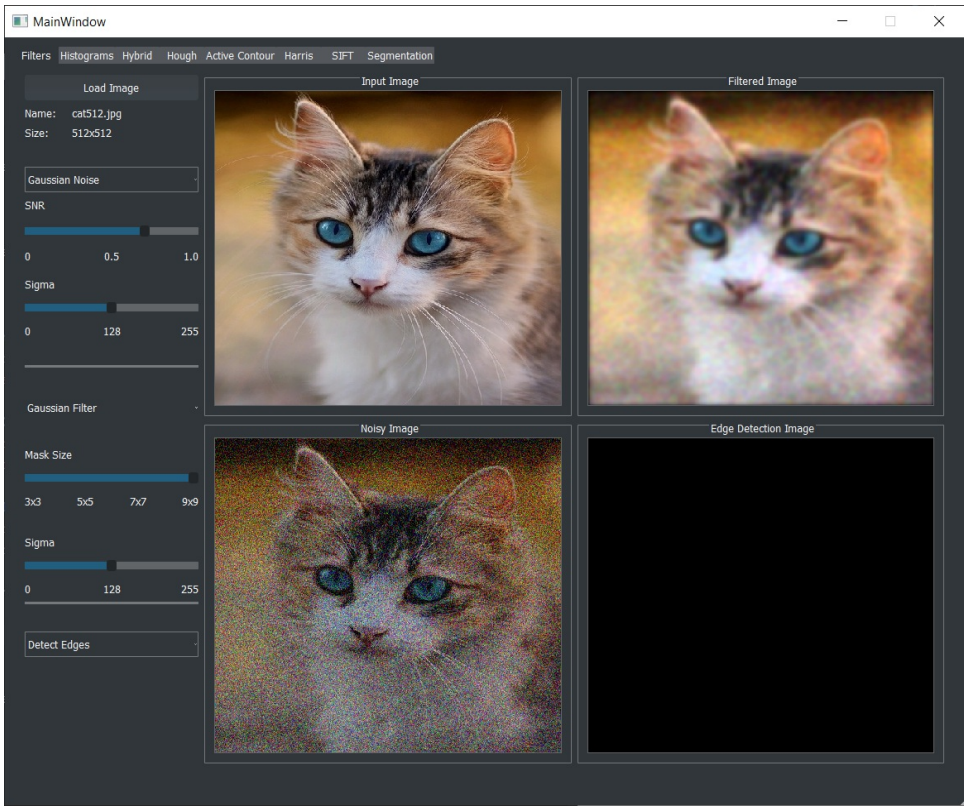
2.2 Gaussian Filter Applied on Gaussian Noise



2.3 Median Filter Applied on Salt & Pepper Noise



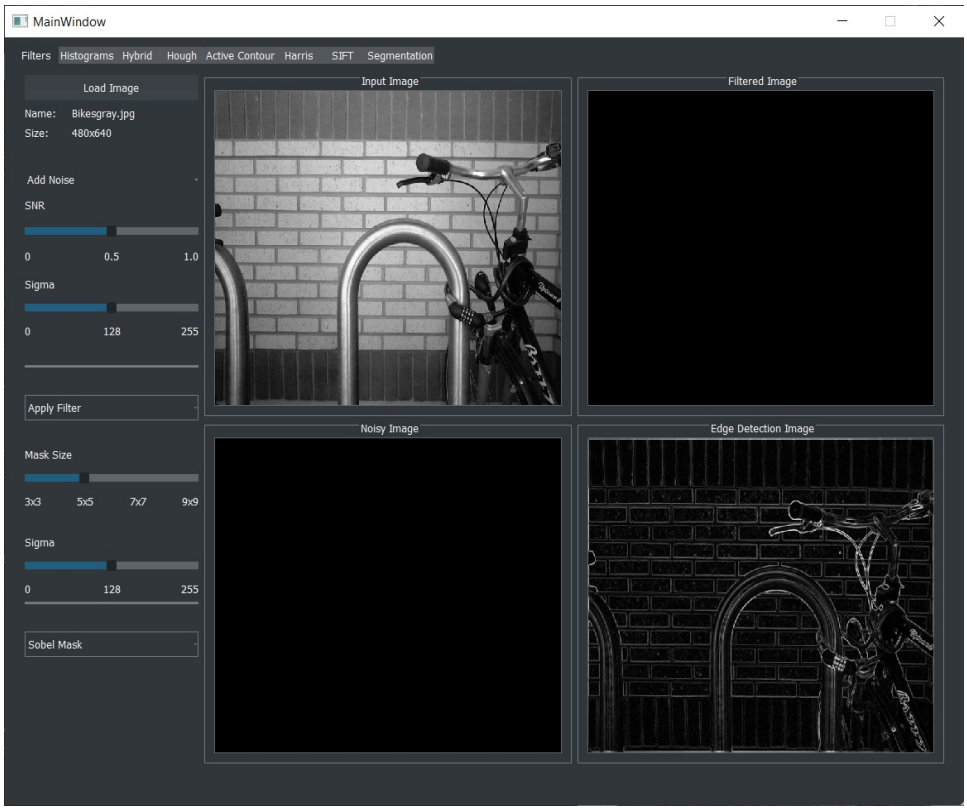
To increase the blurring effect, increase mask size, and this would be the new output with `mask size = 9x9`.



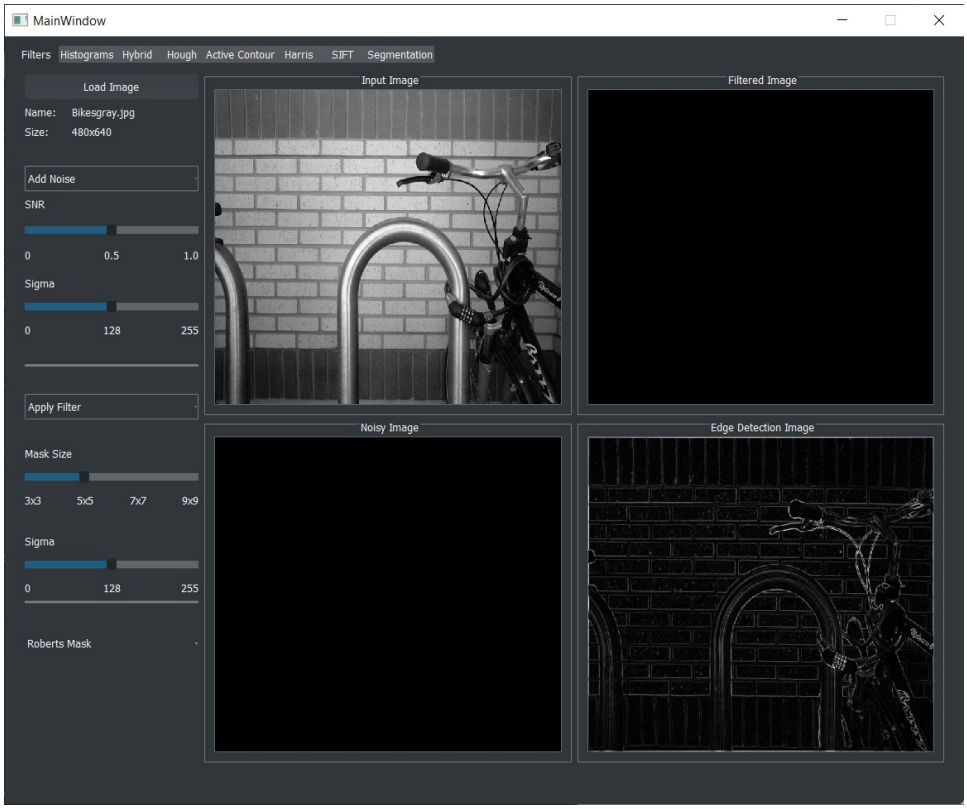
3. Edge Detection

We implemented 4 types of Edge Detection Techniques (Masks): `Prewitt`, `Sobel`, `Roberts` and `Canny`.

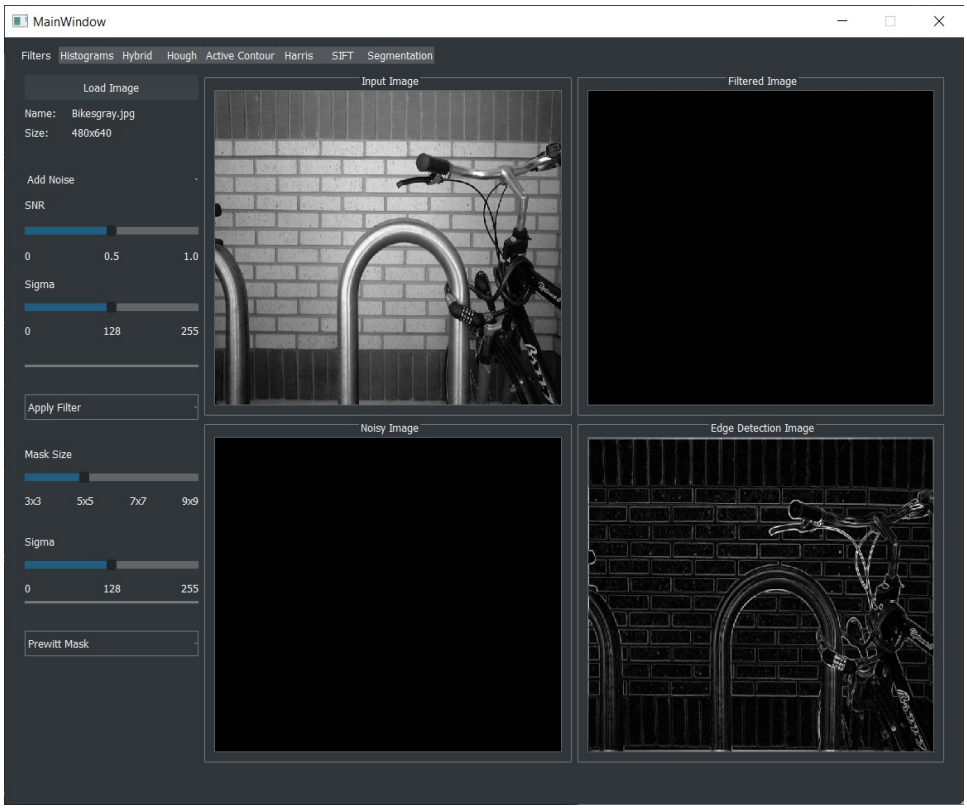
3.1 Sobel Mask



3.2 Roberts Mask

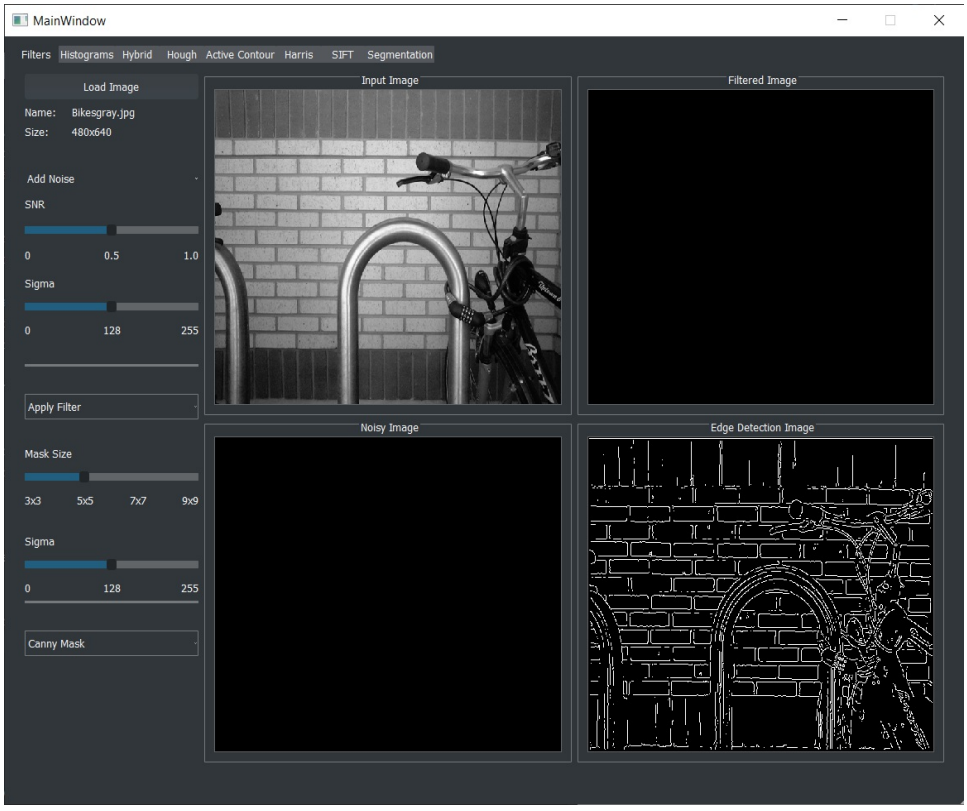


3.3 Prewitt Mask



3.4 Canny Mask

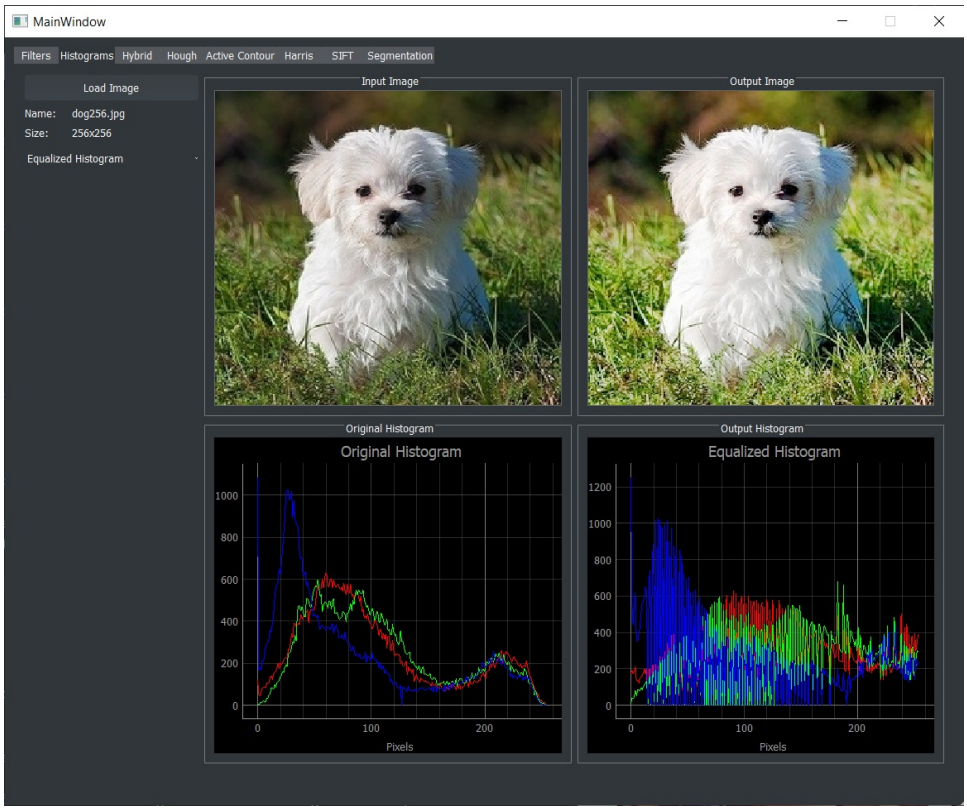
The Canny edge detector is an edge detection operator that uses a multi-stage algorithm to detect a wide range of edges in images.



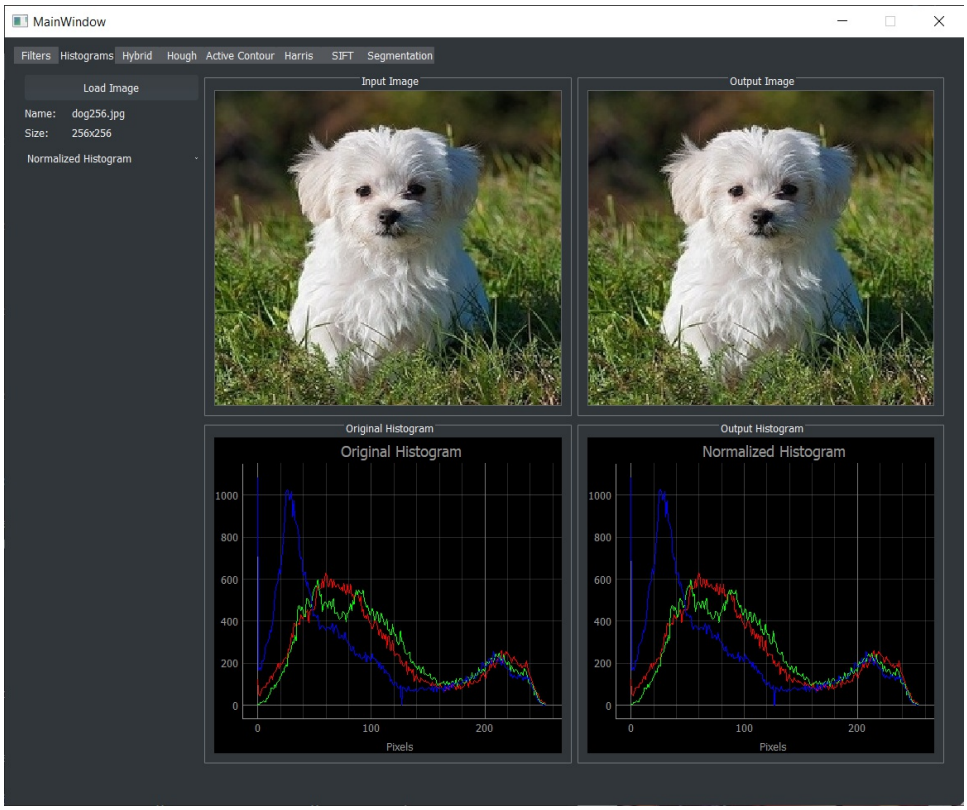
4. Image Histogram and Thresholding

We applied `Histogram Equalization and Normalization`, each algorithm is used for specific problem. In addition to `Convert RGB to Gray Image`. We also applied `Local and Global Thresholding` to differentiate between objects in the image and display specific area of interest. .

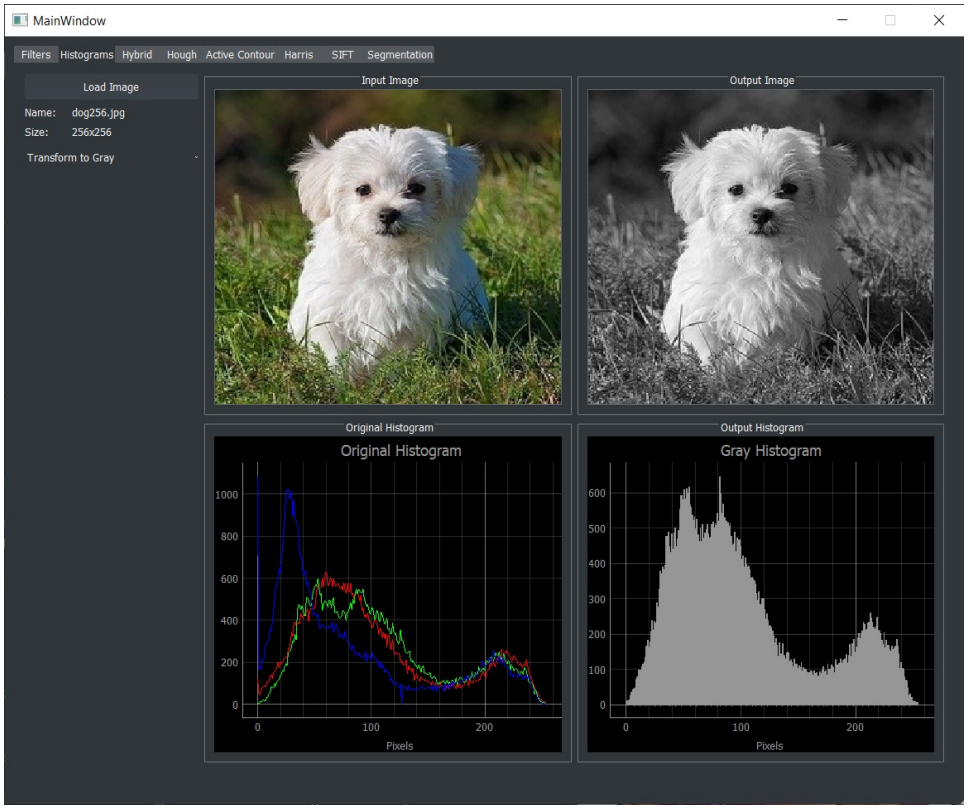
4.1 Histogram Equalization



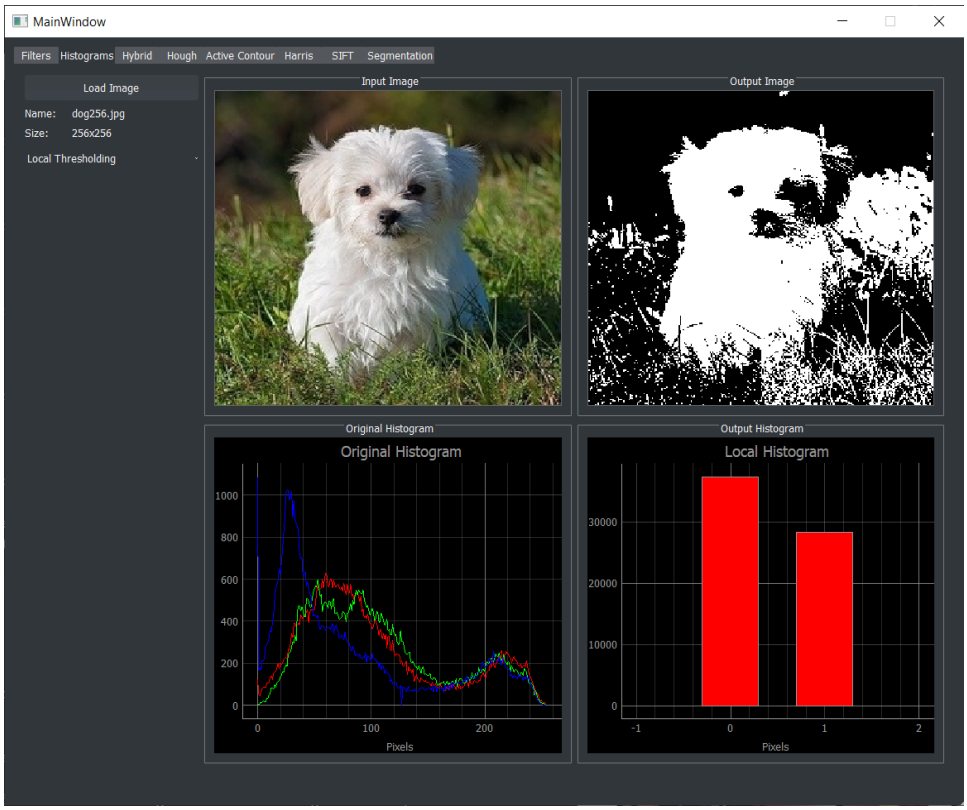
4.2 Histogram Normalization



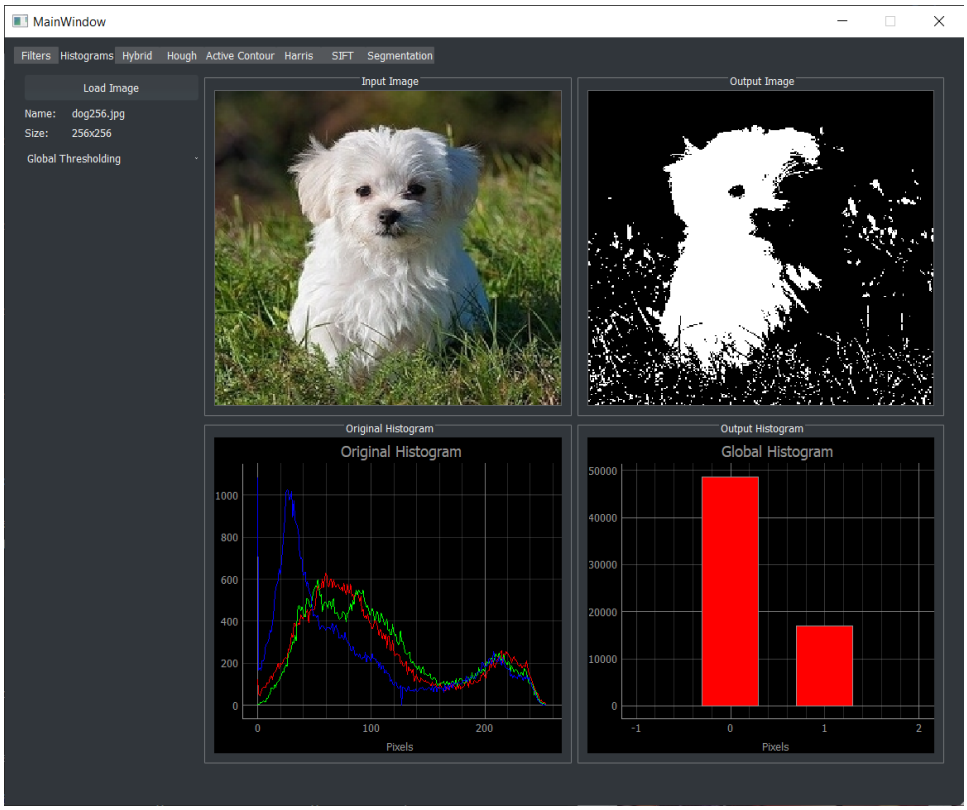
4.3 RGB To Gray



4.4 Local Thresholding



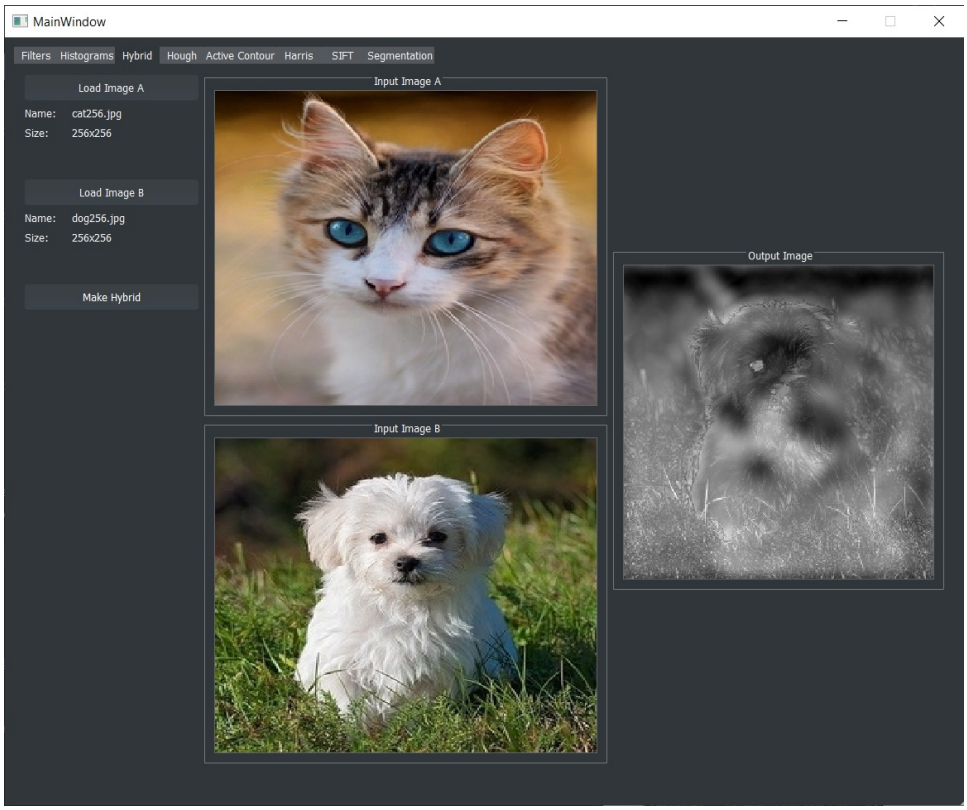
4.5 Global Thresholding



5. Hybrid Images

Given 2 images, we apply a Low Pass Filter to the 1st image, and a High Pass Filters to the 2nd image, both in Frequency Domain, and mix the two images to see the output.

5.1 Low Pass Filter With High Pass Filter



If you zoomed in the image you would see more details from the dog, if you zoomed out the image you would see more details from the cat.

Boundary Detection

In this section we present 2 algorithms implementations; `Hough Transformation` and Active Contour Model, aka `'Snake Algorithm'`.

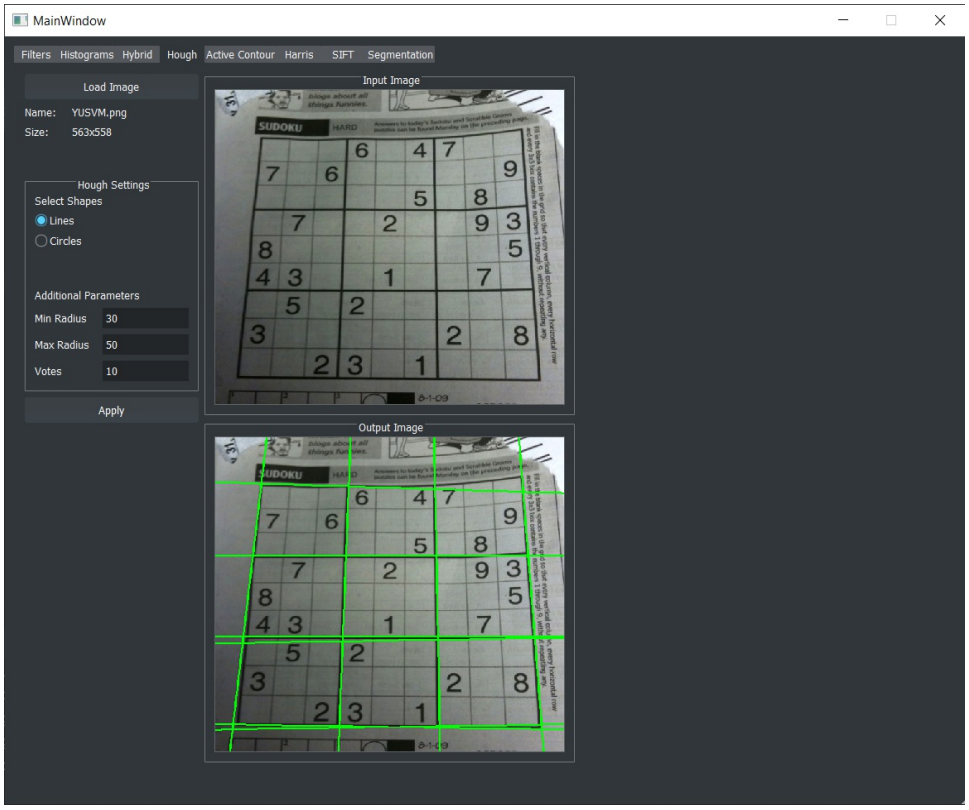
1. Hough Transformation

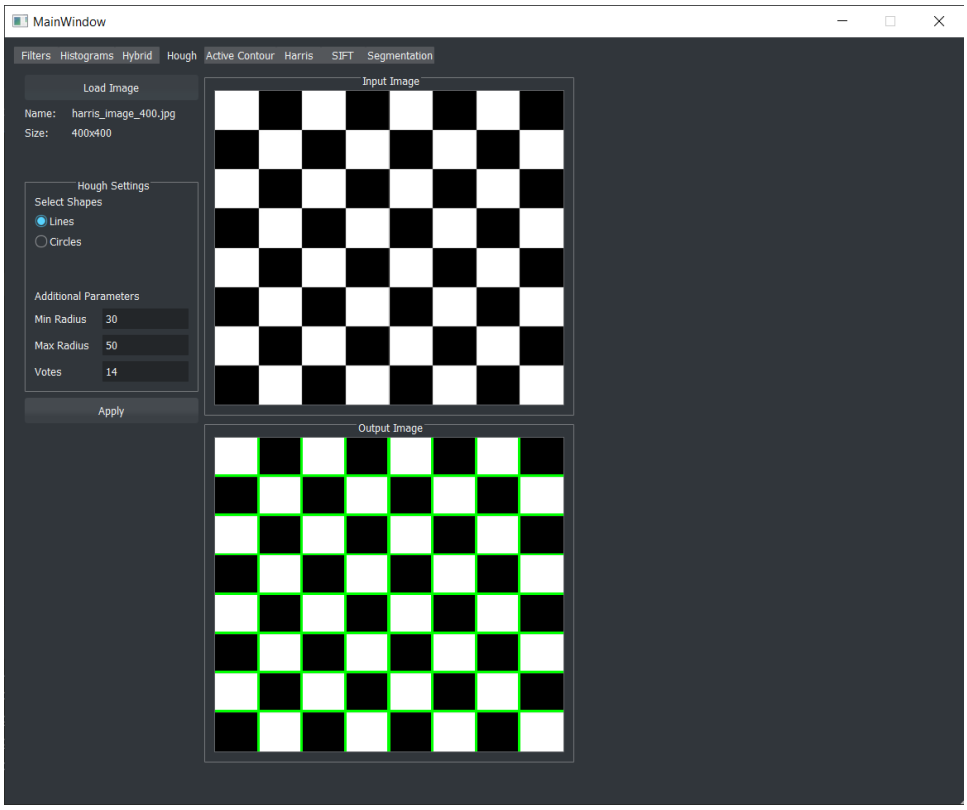
The Hough transform is a technique that locates shapes in images. In particular, it has been used to extract lines, circles and ellipses if you can represent that shape in mathematical form.

The results below were taken with the following setup:

1.1 Line Detection

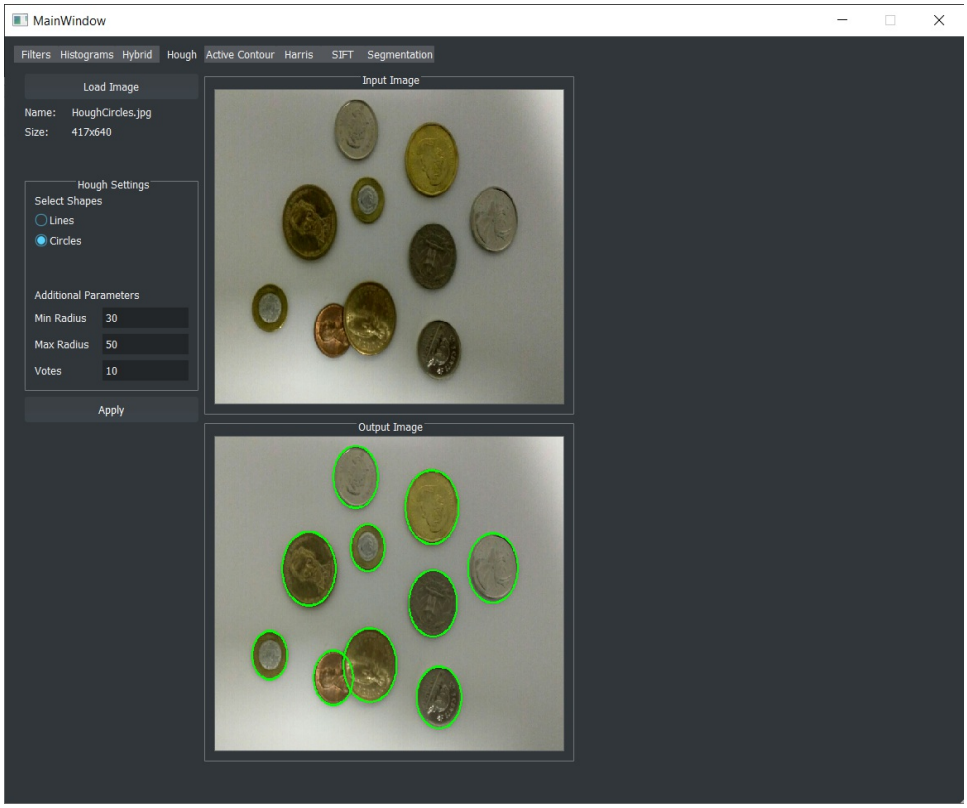
The `Votes` number is basically responsible for determining the amount of output lines. More votes means more detected lines, but this doesn't mean that **5** votes should equal **5** lines, it's not working in that way.

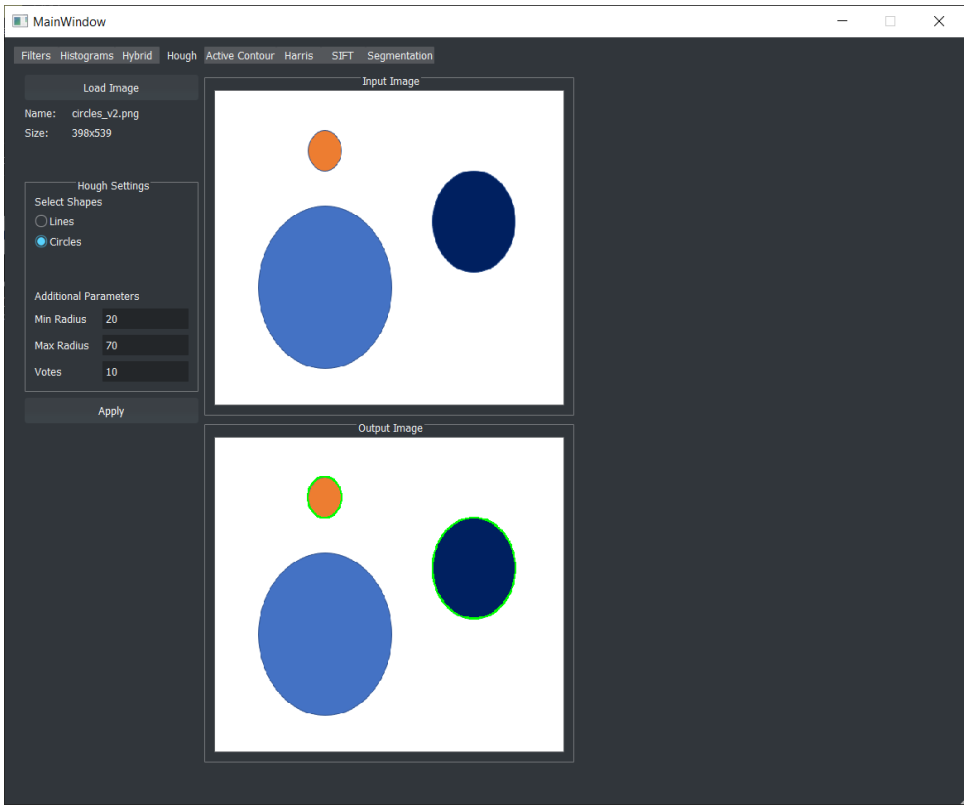




1.2 Circle Detection

Here there is an option to choose the range of radius you want to detect, minimum and maximum range.





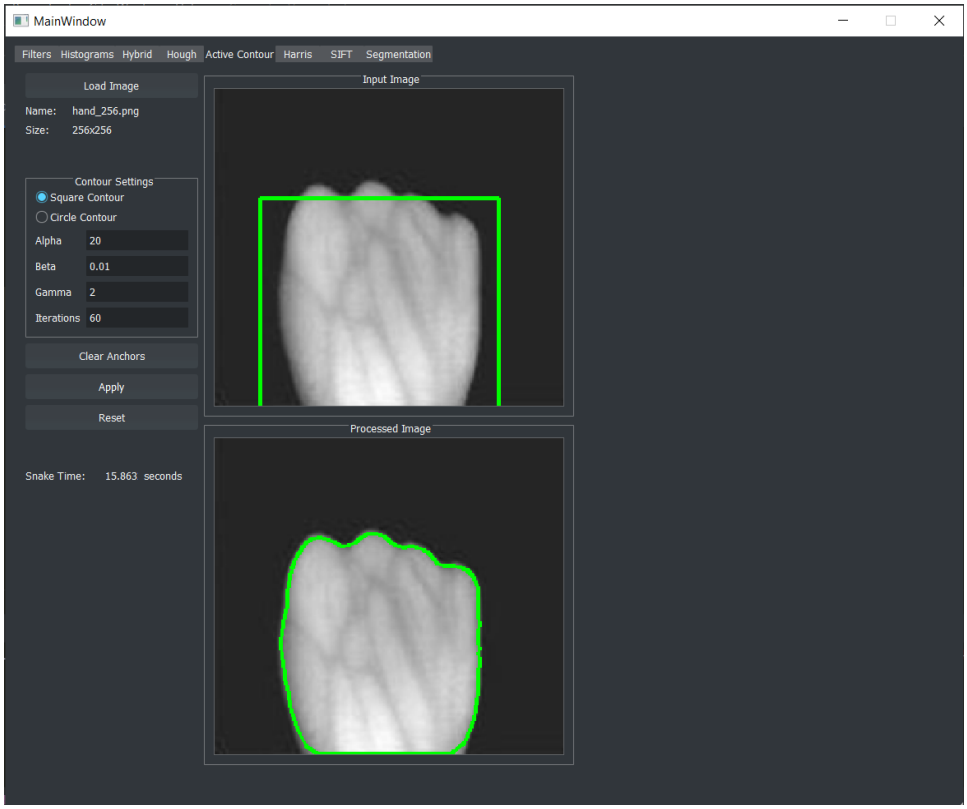
In the 2nd image the maximum radius is less than the bigger circle, so it wasn't detected.

2. Active Contour Model

Active contour is one of the active models in segmentation techniques, which makes use of the energy constraints and forces in the image for separation of region of interest.

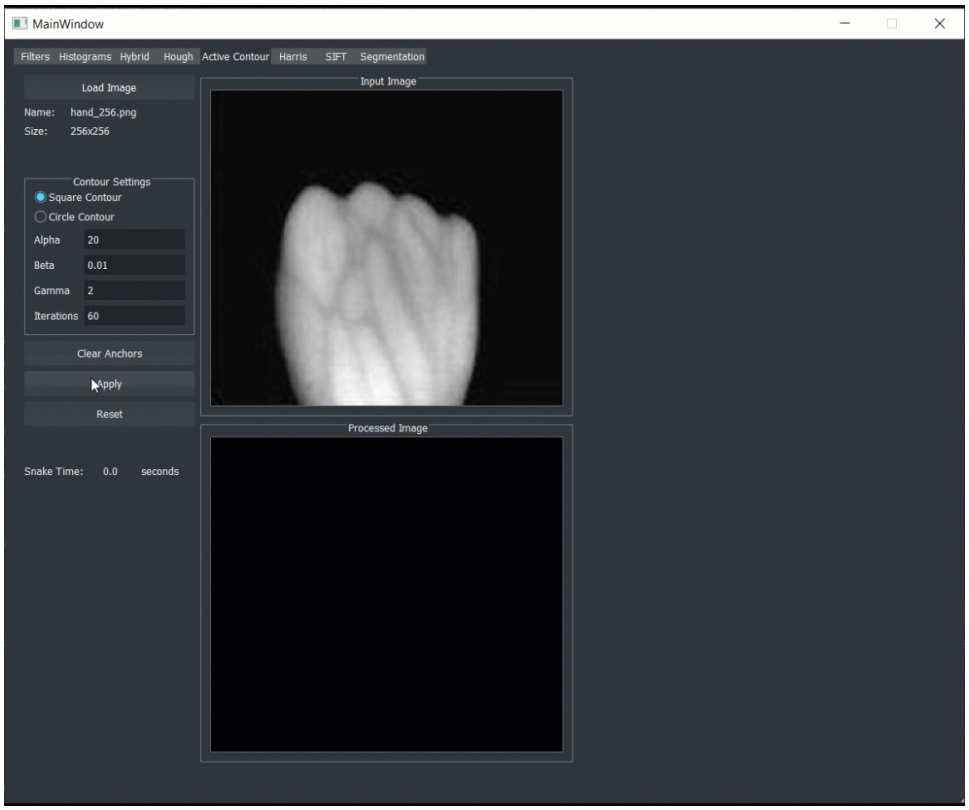
Active contour defines a separate boundary or curvature for the regions of target object for segmentation. This implementation is based on `Greedy Algorithm`.

2.1 Result of applying Snake Model on a hand image

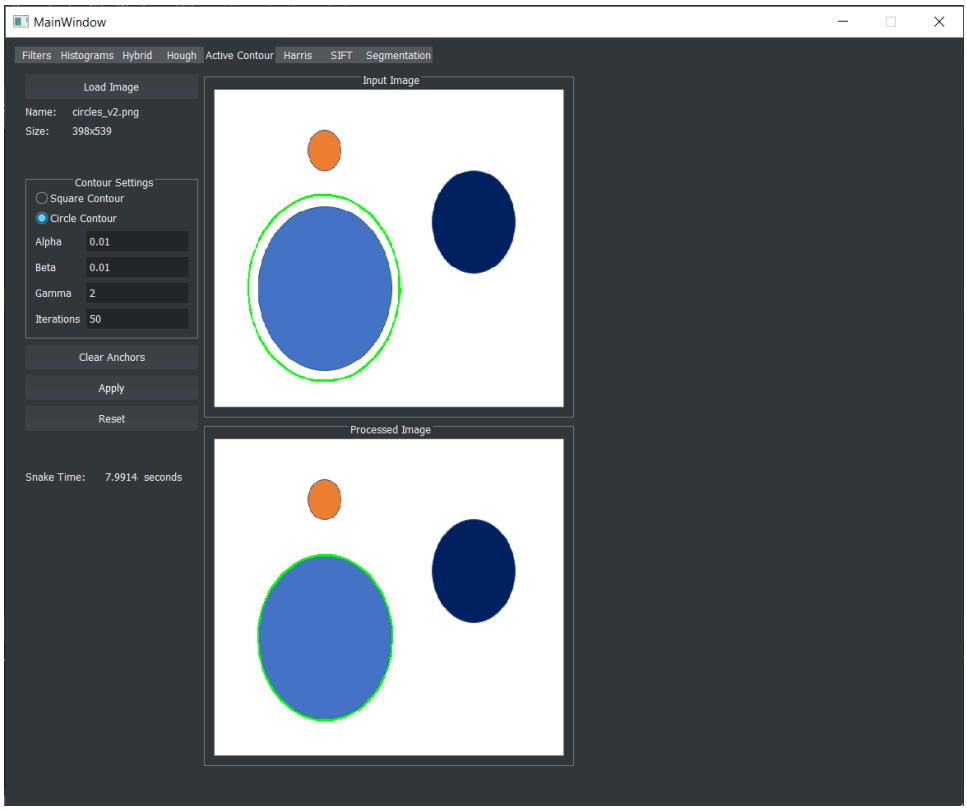


The parameters' values of `alpha`, `beta`, `gamma` and `number of iterations` are selected by trial and error approach.

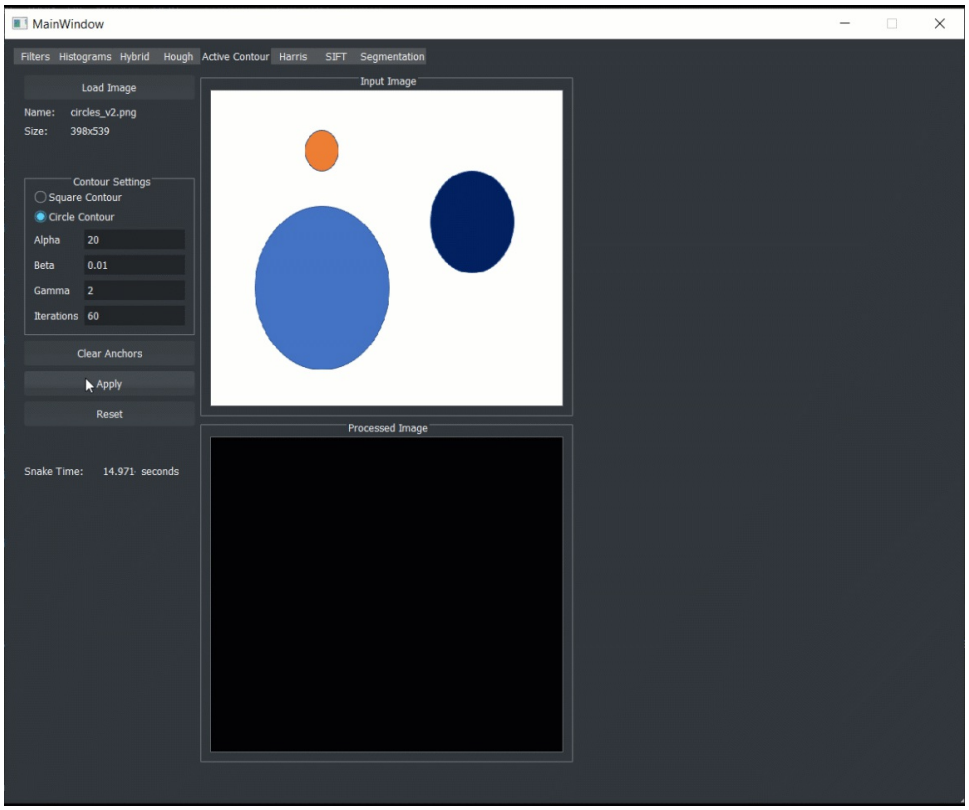
This GIF shows the process in a better way


MainWindow
Filters
Histograms
Hybrid
Hough
Active Contour
Harris
SIFT
Segmentation
Load Image
Name: hand_256.png
Size: 256x256
Contour Settings
Square Contour
Circle Contour
Alpha 20
Beta 0.01
Gamma 2
Iterations 60
Clear Anchors
Apply
Reset
Snake Time: 0.0 seconds
Input Image
Processed Image

2.2 Result of applying the algorithm on circles image



This GIF shows the process in a better way



Features Detection and Image Matching

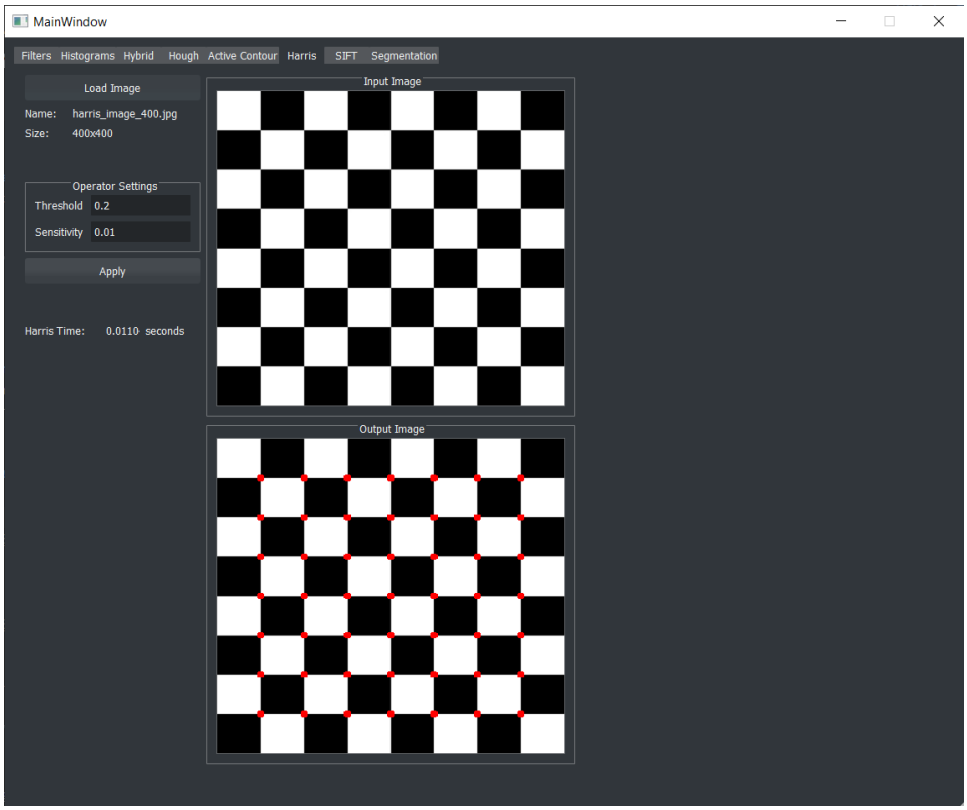
In this section we present 3 algorithms implementations; `Feature Extraction Using Harris Operator`, `Scale Invariant Features (SIFT)` and `Feature Matching`.

1. Extract The Unique Features In All Images Using Harris Operator

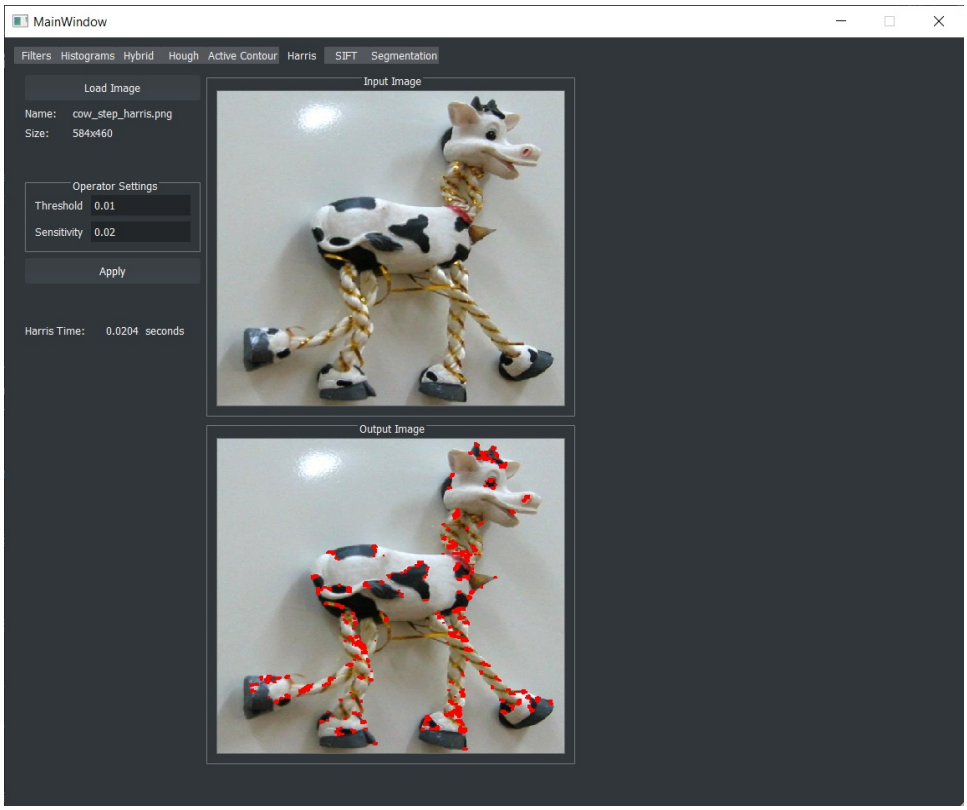
There are mainly 2 parameters in Harris Detector:

- `Threshold`: Value used computing local maxima (Higher threshold means less corners)
- `Sensitivity`: Sensitivity factor to separate corners from edges. (Small values result in detection of sharp corners).

1.1 Harris Corners with `0.2` Threshold and `0.01` Sensitivity



1.2 Harris Corners with `0.1` Threshold and `0.02` Sensitivity



The processing time is barely noticeable, it only took about `0.01 second` to detect all the corners in the first image and `0.02 second` in the second image.

2. Feature Descriptors Using Scale Invariant Features (SIFT) Algorithm

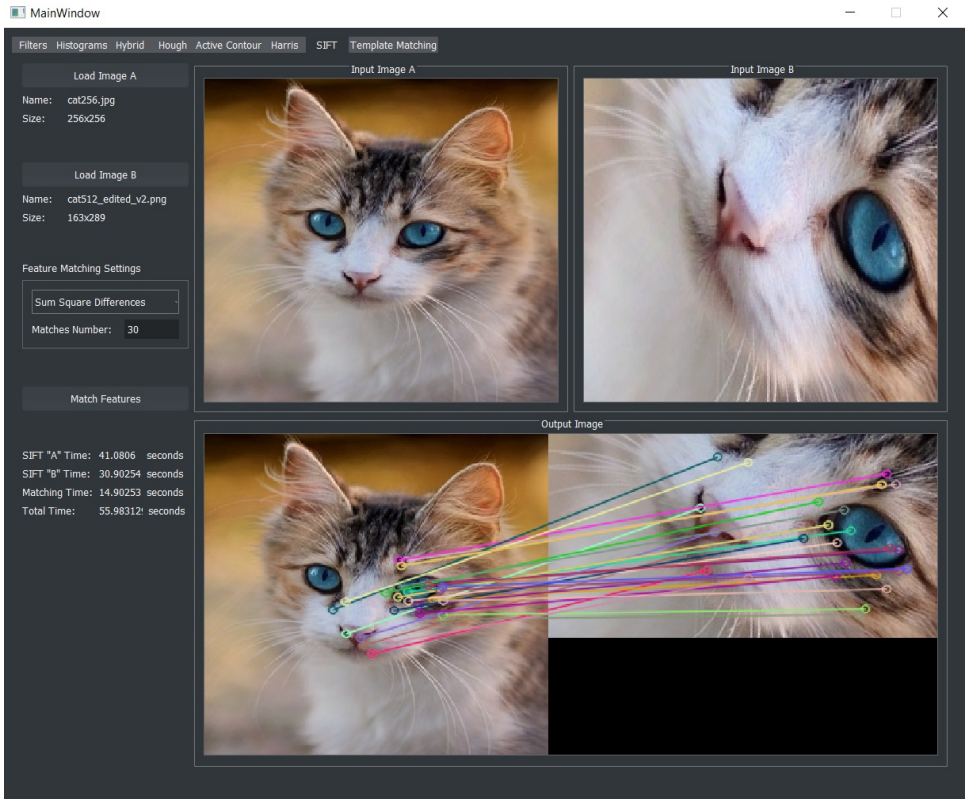
Applying SIFT Algorithm to generate features descriptors to use them in matching images with different techniques.

It is not necessary to show the output of SIFT algorithm, the final output is shown in the matching step.

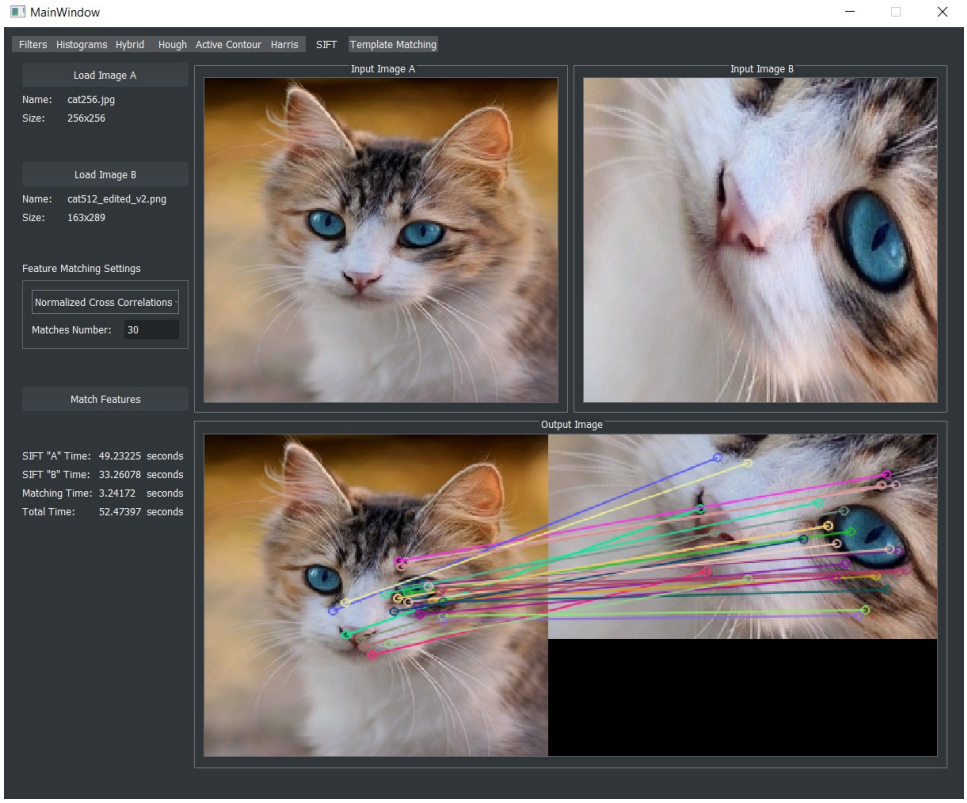
3. Matching the Image Set Features

We applied two Matching Algorithms, Sum Of Squared Differences `(SSD)` and Normalized Cross Correlations `(NCC)`.

3.1 Feature Matching Using Sum of Squared Differences (SSD)



3.2 Feature Matching Using Normalized Cross Correlations (NCC)



The computations in this algorithm are heavily and extreme, so as you see it took around `1 minute` to finish the whole process on a small image.

Note:

In the above results, each SIFT Algorithm applied was running on a separate thread for faster and better experience, and to avoid GUI freezing problem.

Image Segmentation

In this section we present some algorithms implementations for Image Segmentation; `Using Thresholding and Clustering Methods`.

1. Segmentation Using Thresholding

We implemented 3 types of Thresholding, Local and Global for each threshold:

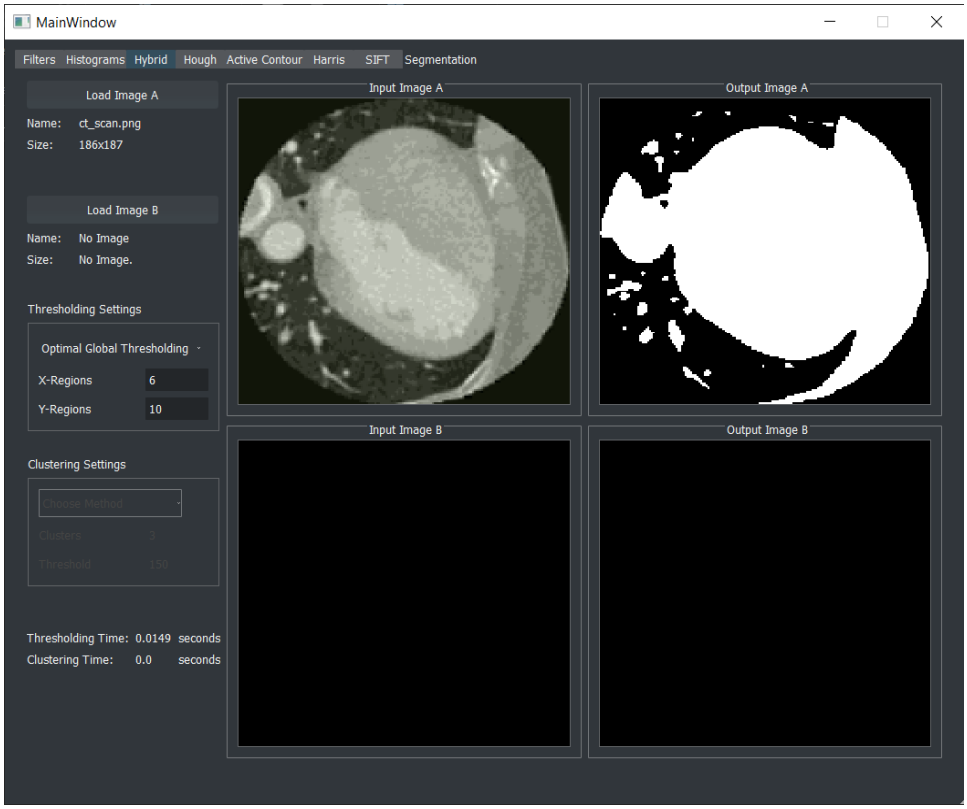
- `Optimal Thresholding`
- `Otsu Thresholding`
- `Spectral Thresholding (More than 2 modes)`

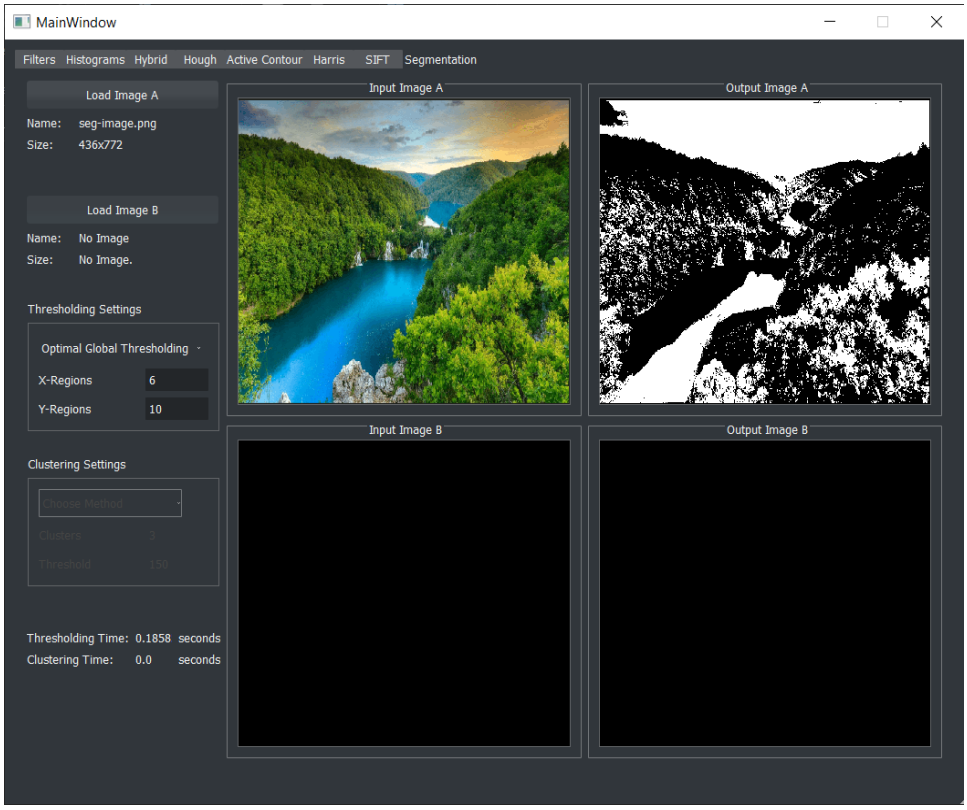
There are mainly 1 parameter used in Local Thresholding:

- `X-Regions`: indicates how many regions we want to divide in x-direction
- `Y-Regions`: indicates how many regions we want to divide in y-direction

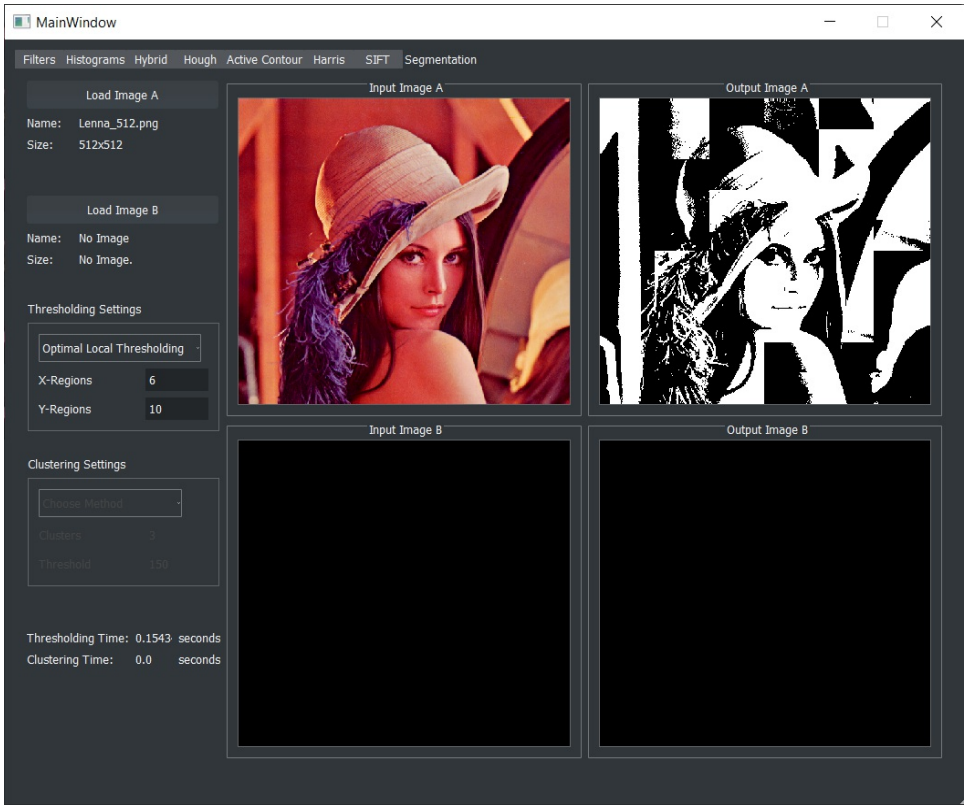
1.1 Optimal Thresholding

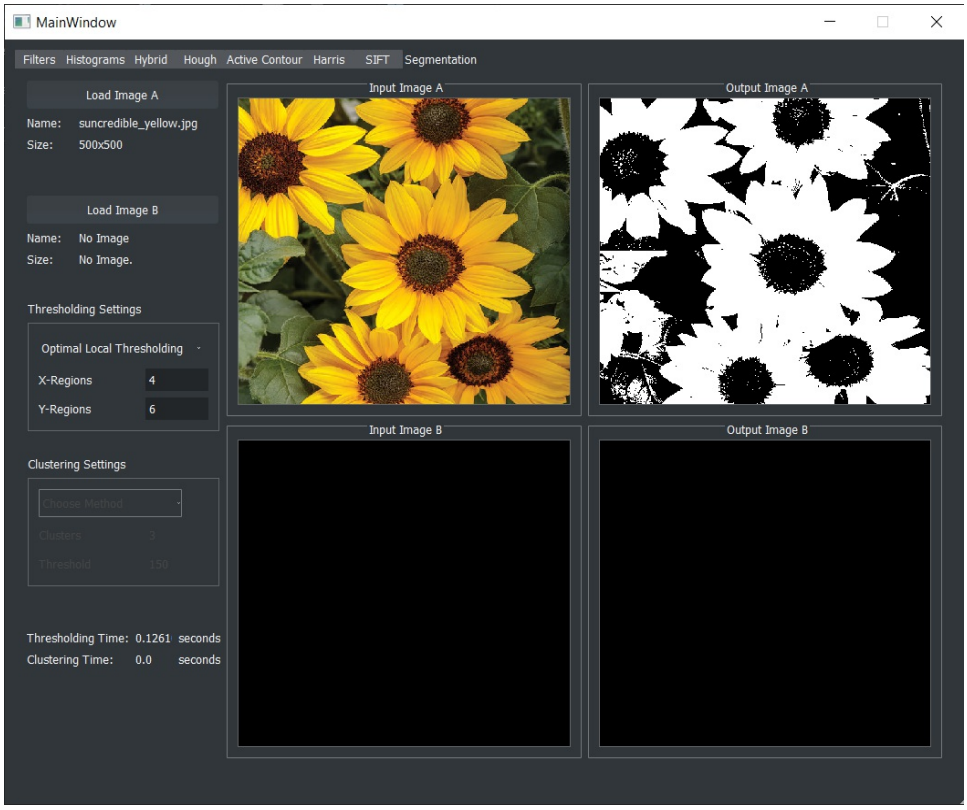
Using Global Thresholding





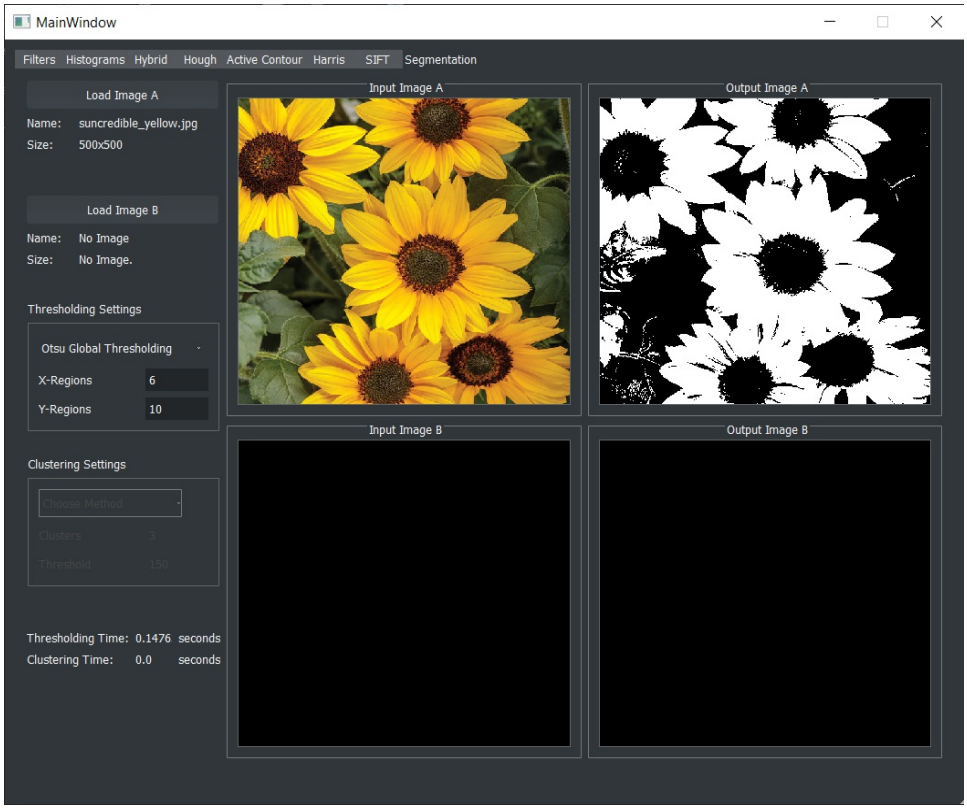
Using Local Thresholding

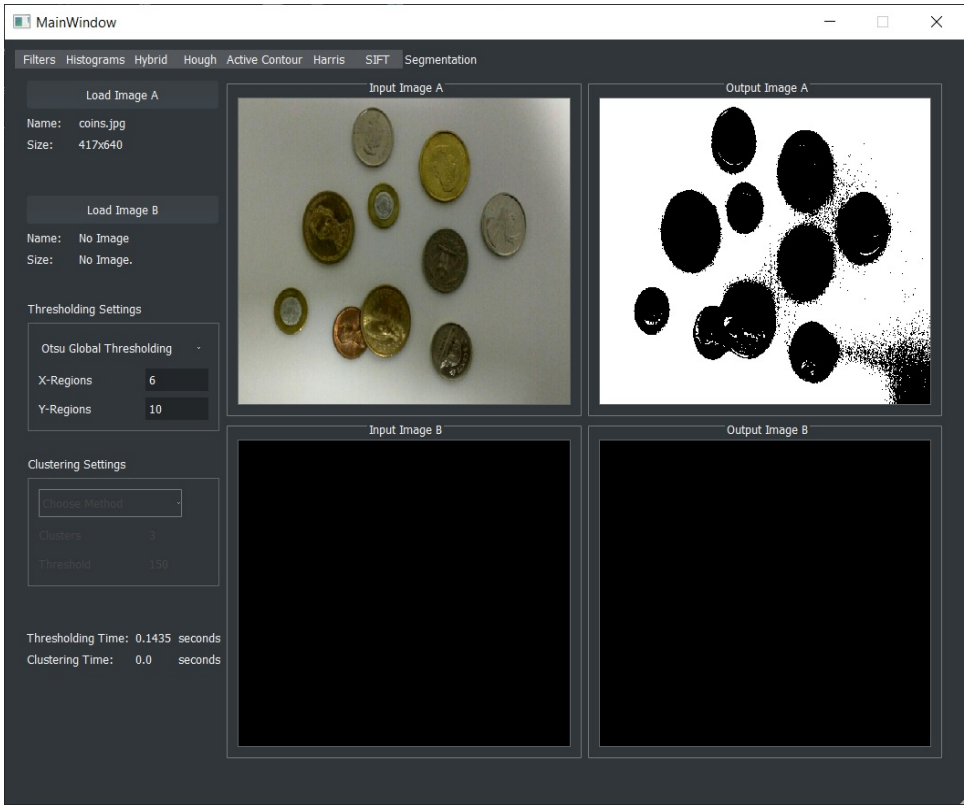




1.2 Otsu Thresholding

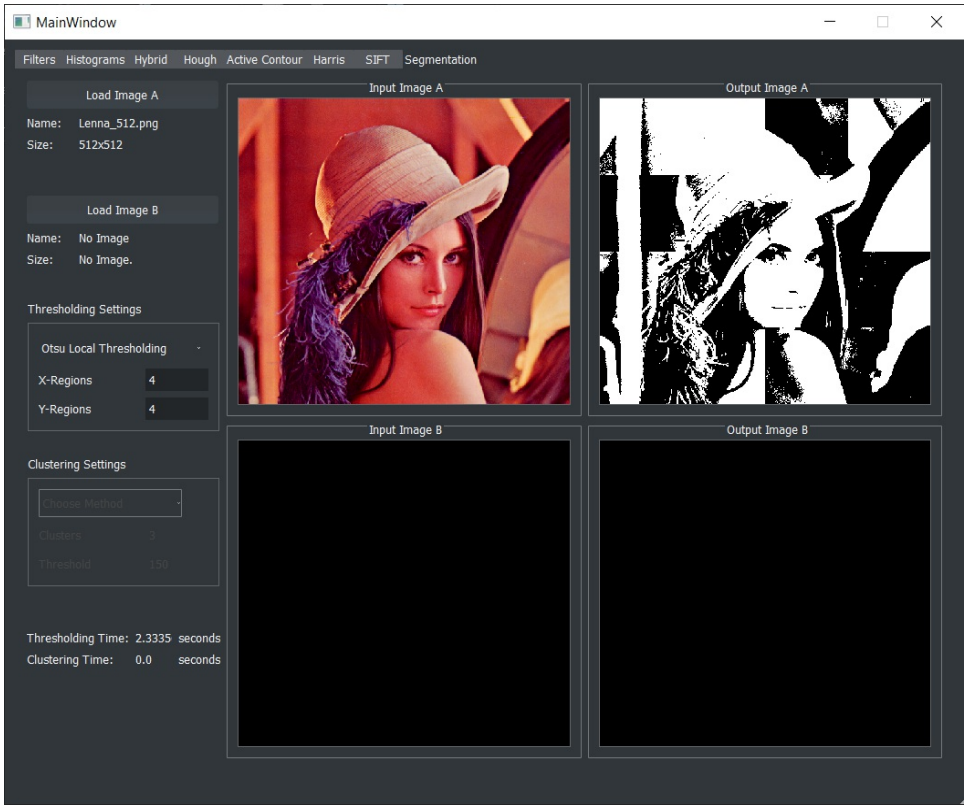
Using Global Thresholding





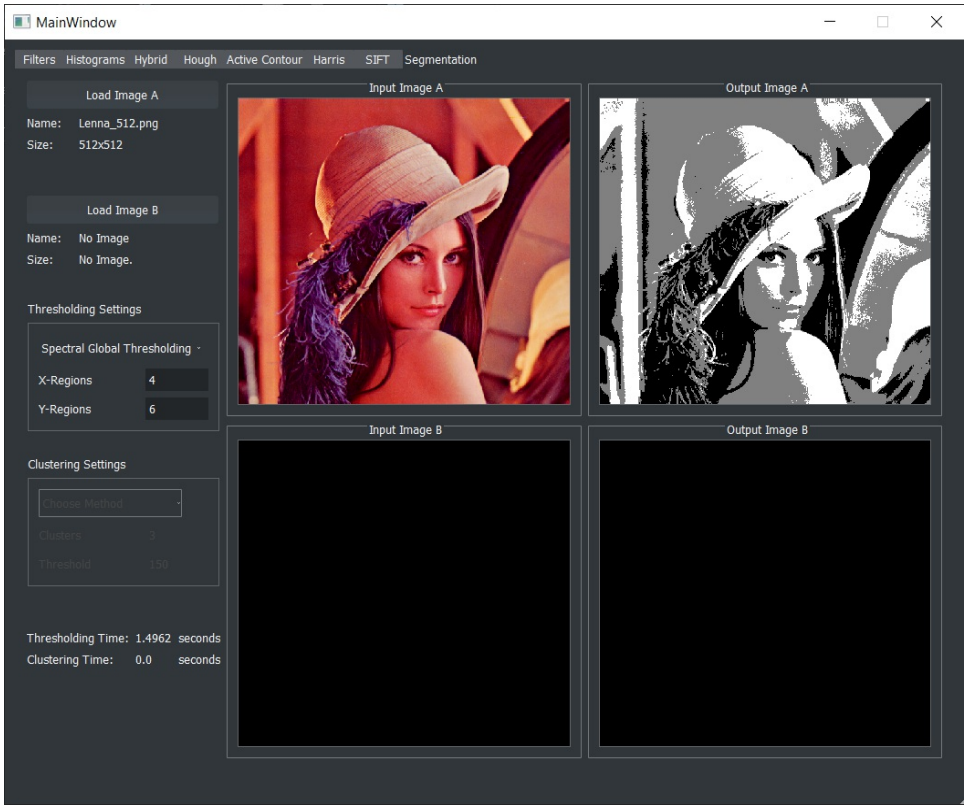
Note: There is some noise in the image which affects the output a little.

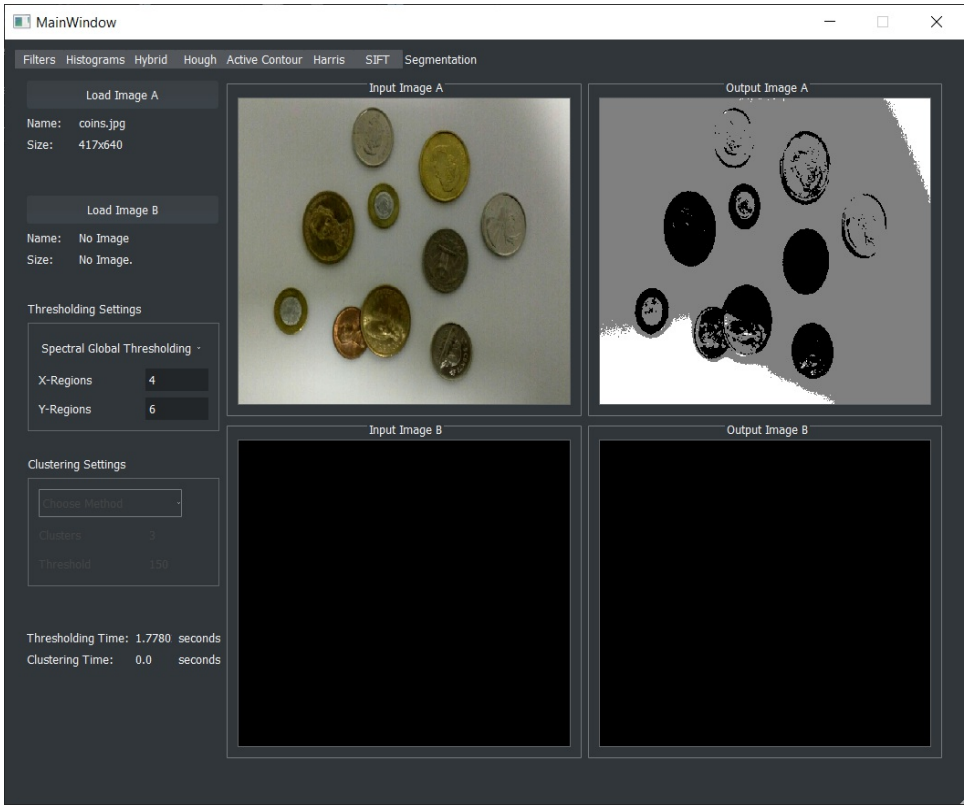
Using Local Thresholding



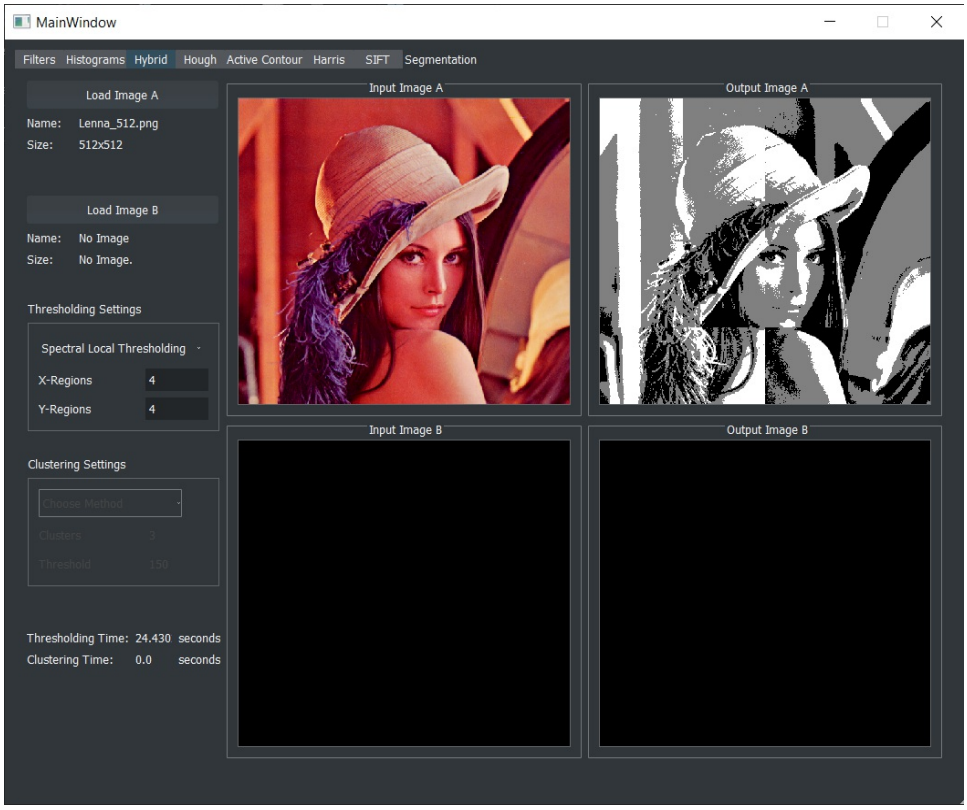
1.3 Spectral Thresholding

Using Global Thresholding





Using Local Thresholding



2. Segmentation Using Clustering

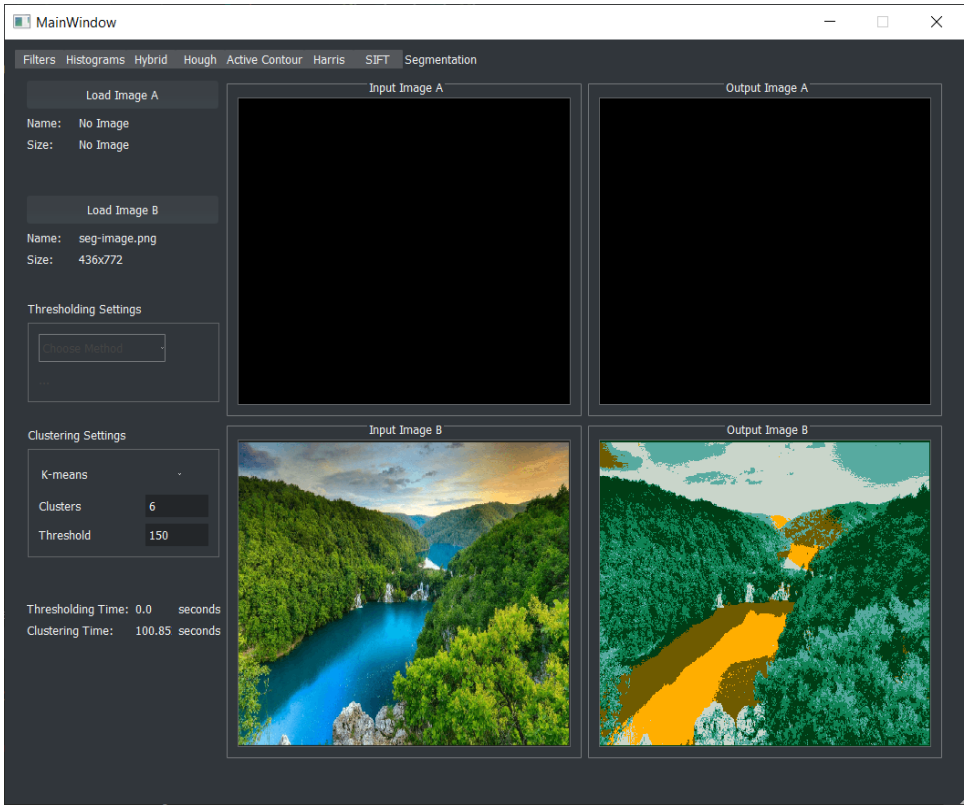
We implemented 4 Clustering methods:

- `K-Means`
- `Region Growing`
- `Agglomerative Clustering`
- `Mean-Shift`

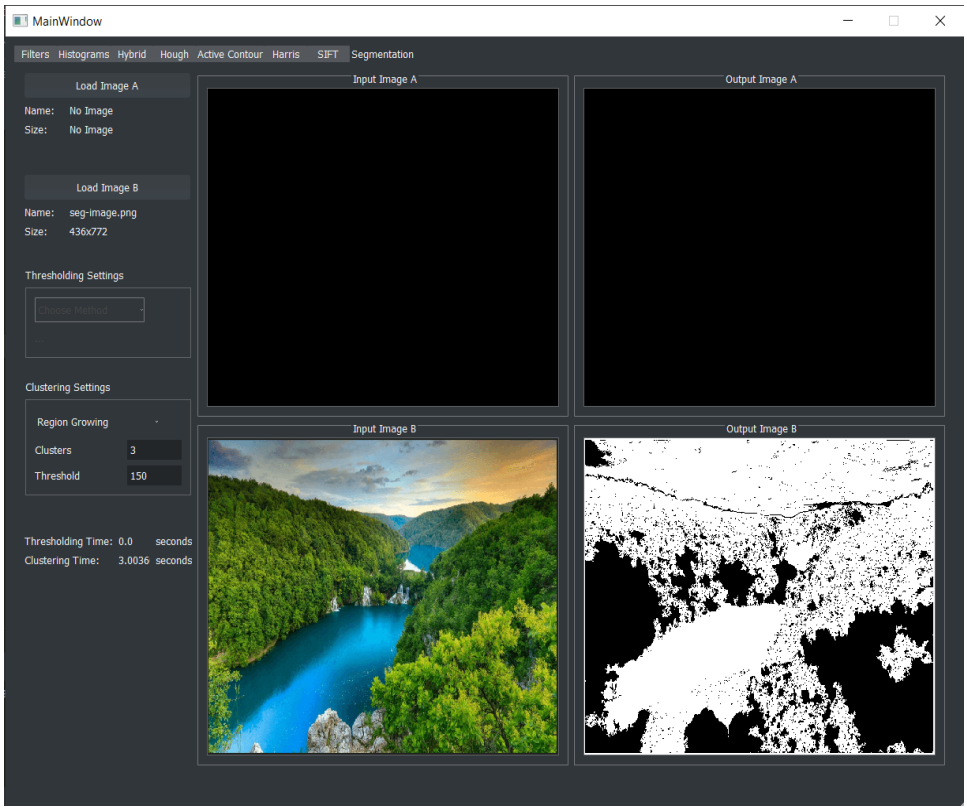
There are mainly 2 parameters in some Clustering methods:

- `Number of Clusters`: to specify how many clusters you need in the output image.
- `Threshold`: to threshold the output image in specif level in some methods.

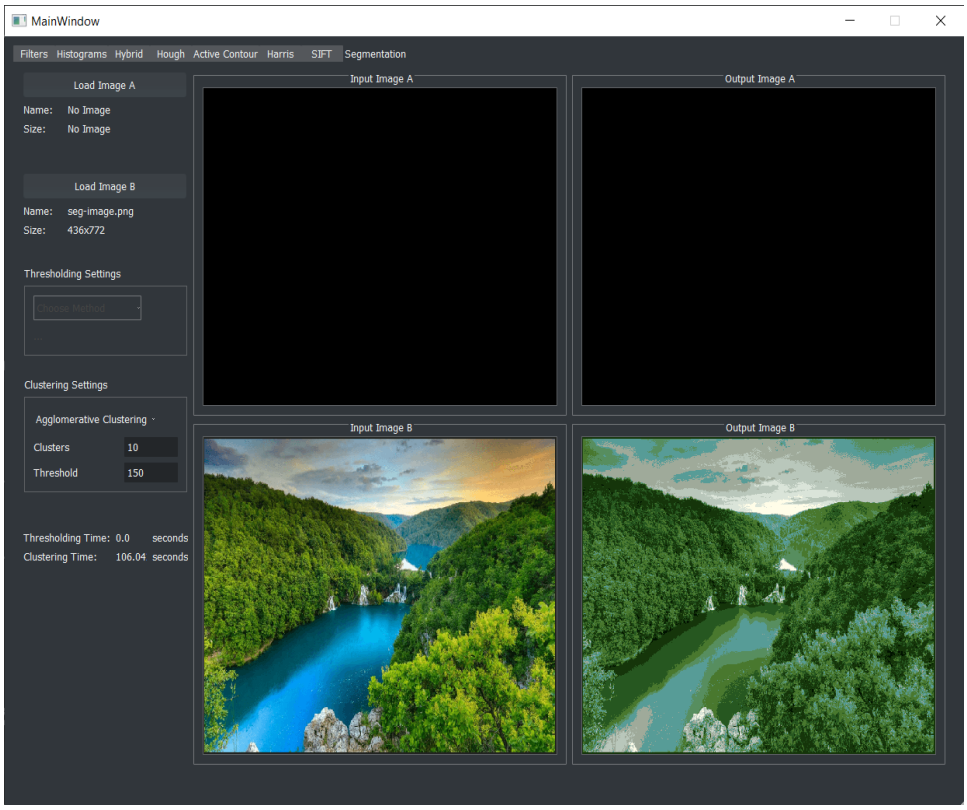
2.1 K-Means with `6` Clusters



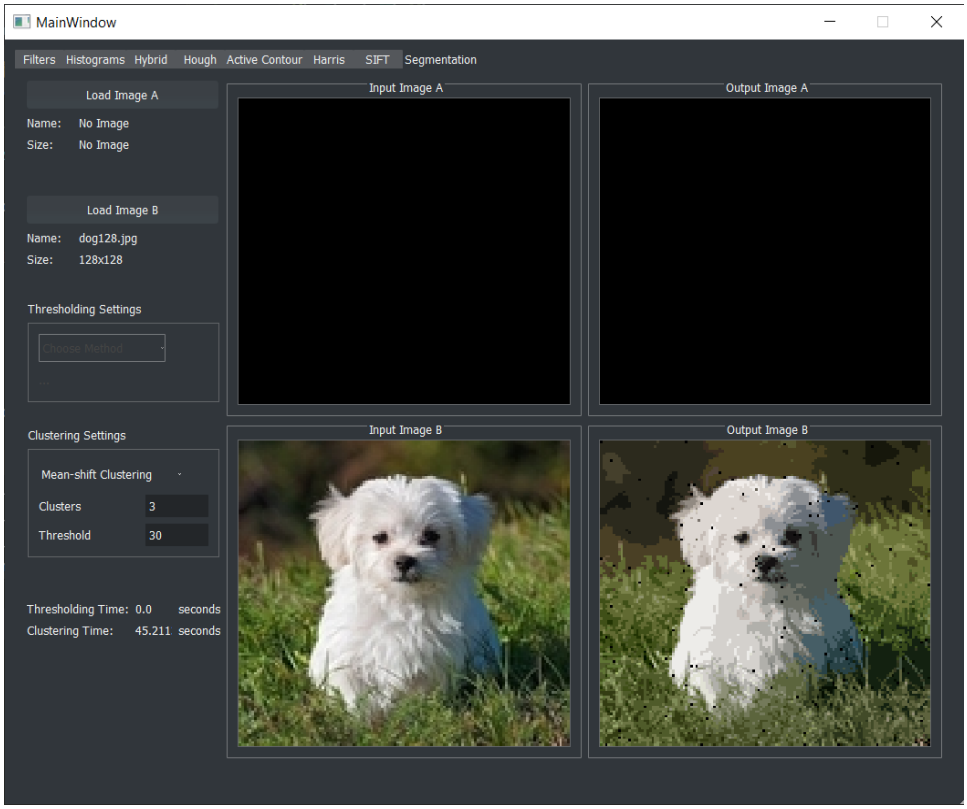
2.2 Region Growing with `3` Clusters



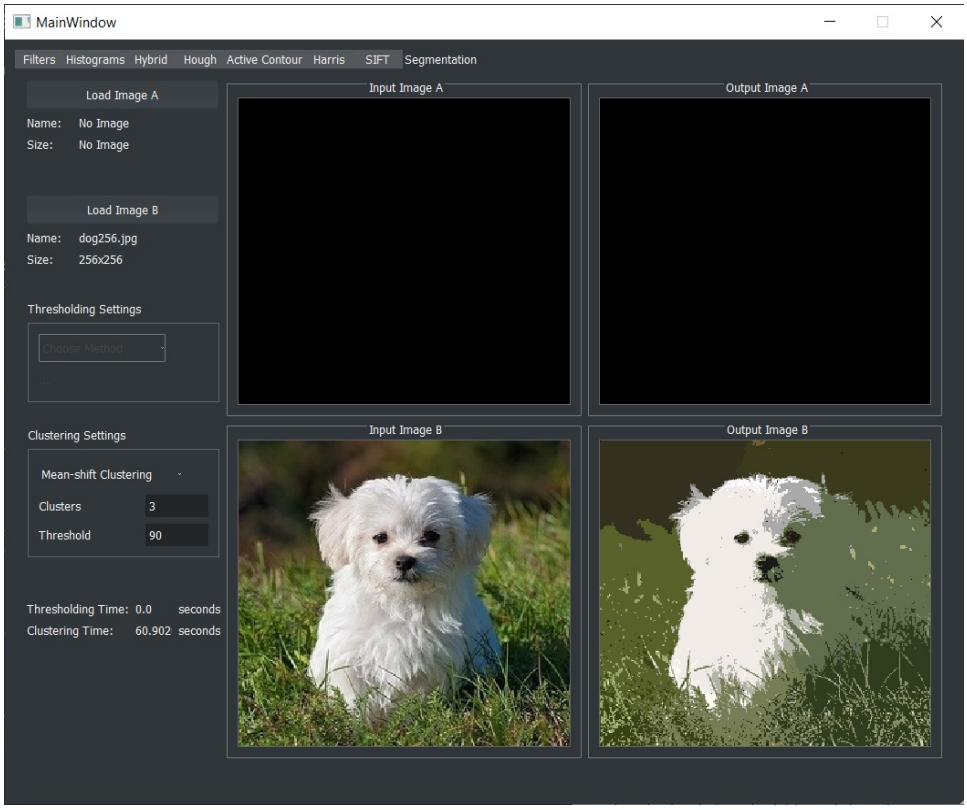
2.3 Agglomerative Clustering with `10` Clusters



2.4 Mean-Shift with `30` Threshold



Mean-Shift with `90` Threshold



The output image is changed whenever you change the threshold. You could choose the desired threshold by trial and error to know what value fits.

Face Detection and Recognition

In this section we present Face Detection and Recognition implementations; `Using PCA/Eigenfaces Analysis`.

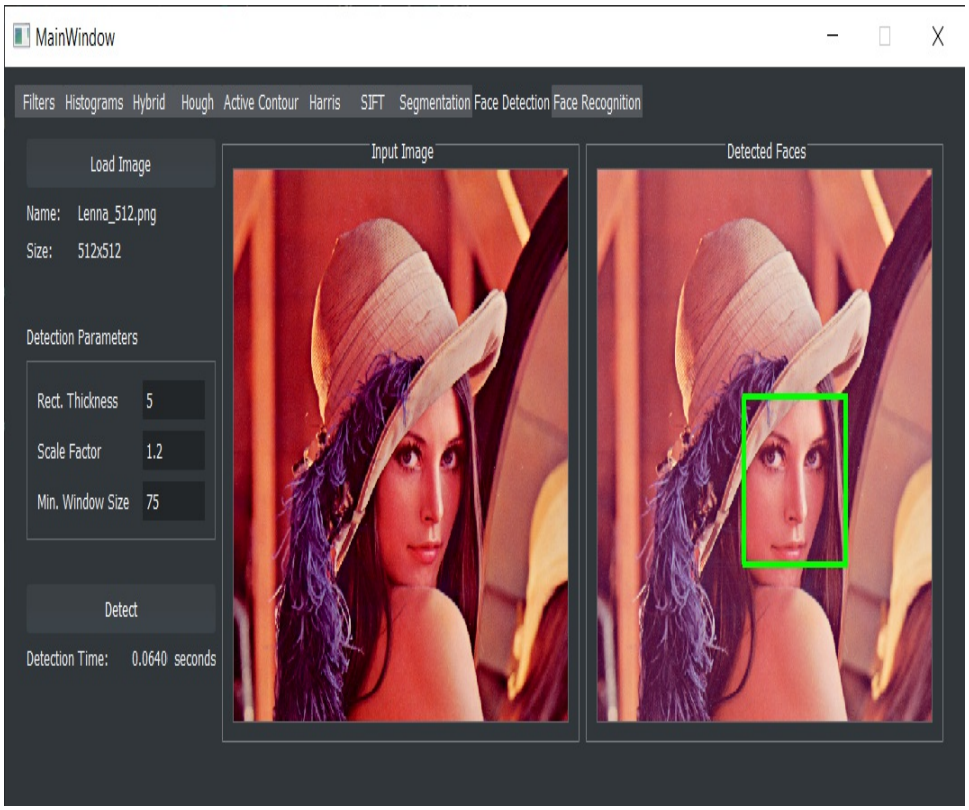
1. Face Detection

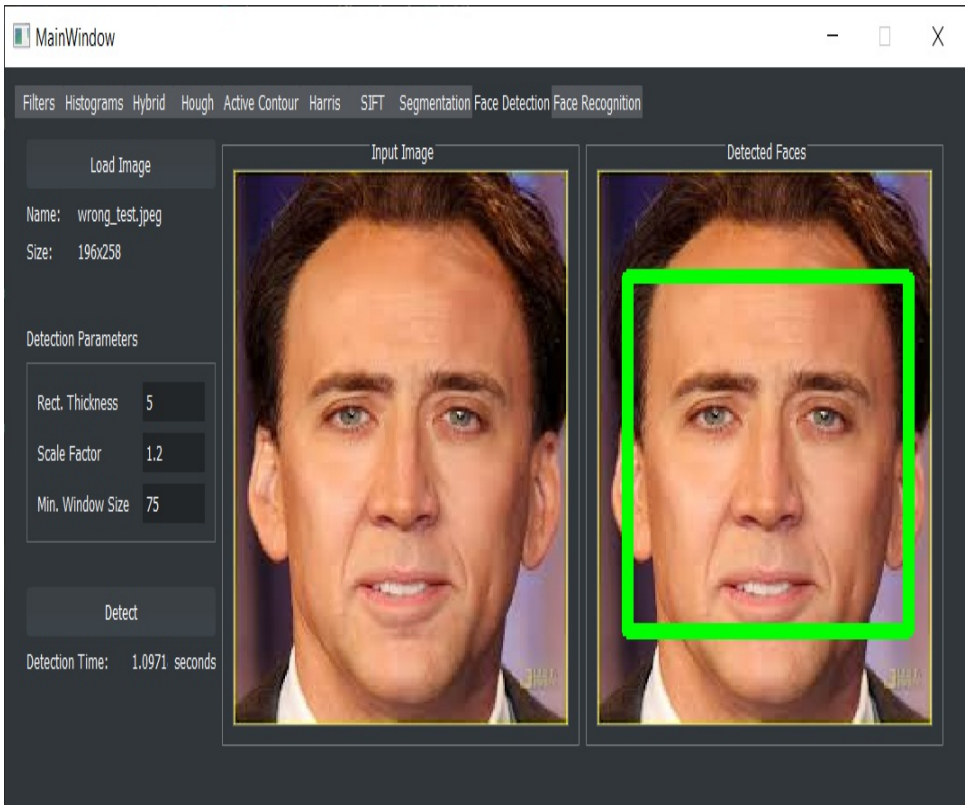
This is implemented using openCV library, using `CascadeClassifier` which contains OpenCV data used to detect objects.

There are mainly 2 parameter you can adjust in Face Detection:

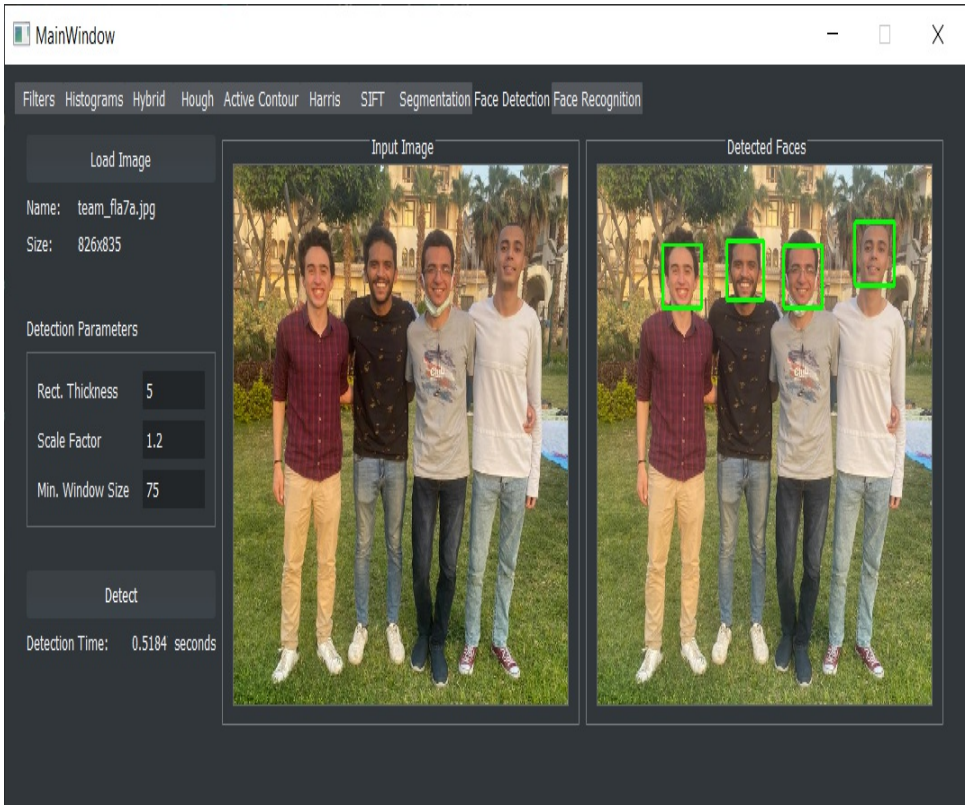
- `Scale Factor`: Since some faces may be closer to the camera, they would appear bigger than the faces in the back. The scale factor compensates for this.
- `Minimum Window Size`: size of each moving window that that algorithm to detect objects.

Face Detection of one person





Face Detection of our team



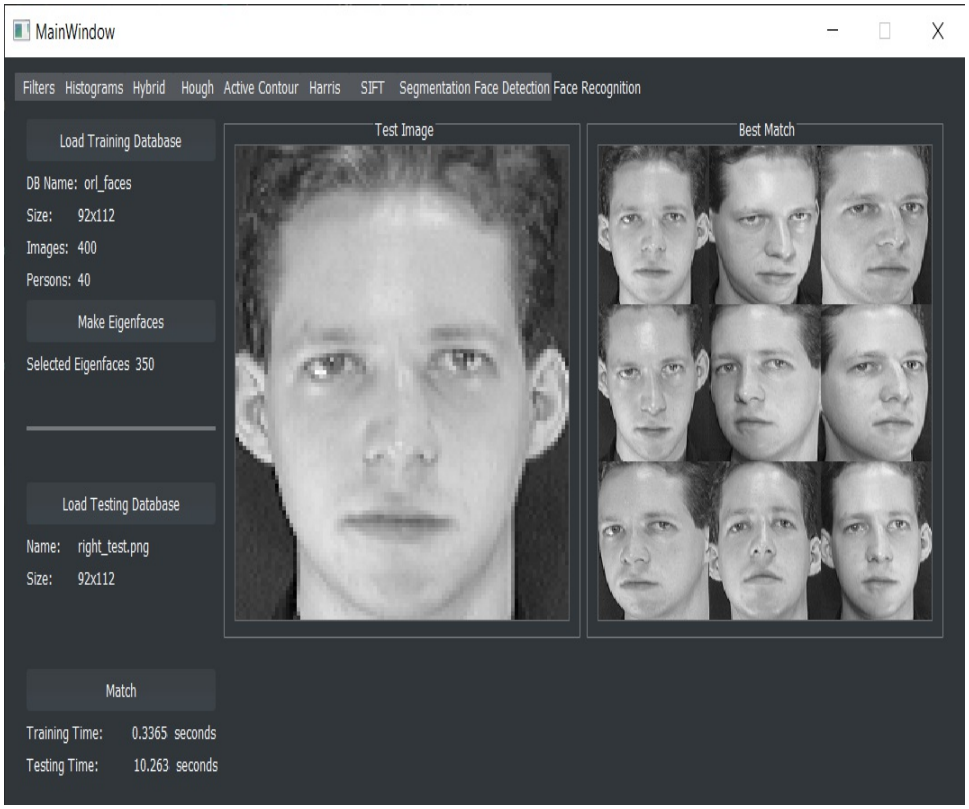
2. Face Recognition

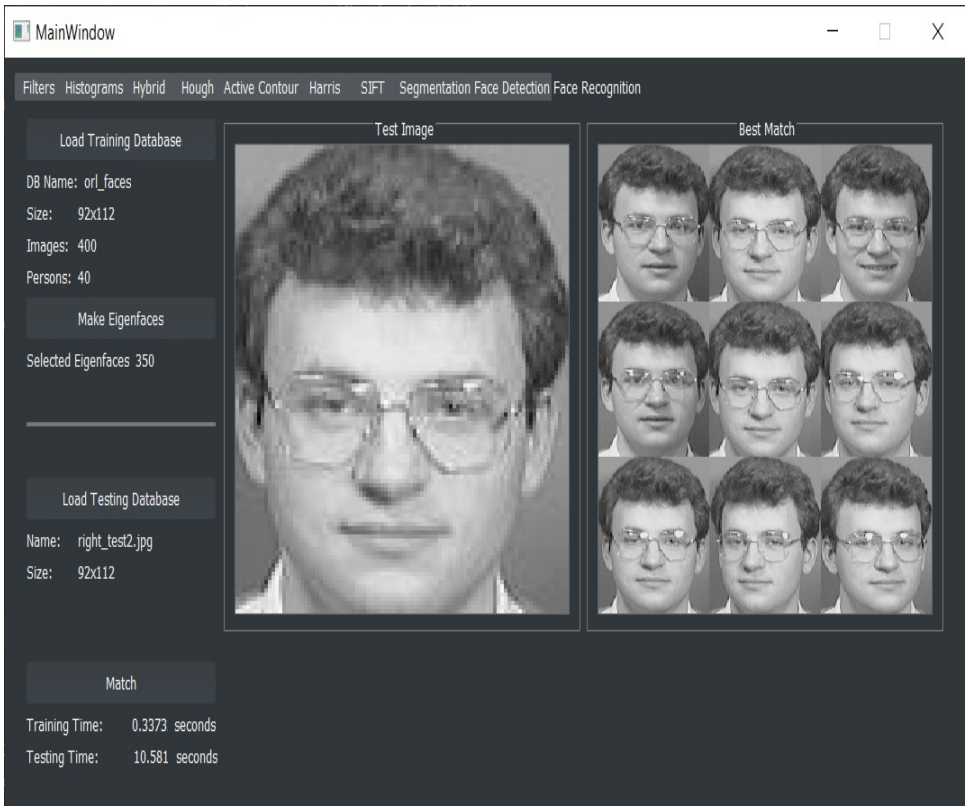
This implementation is based on `PCA/Eigenfaces Analysis`, it's implemented from scratch with the help of some useful libraries.

Quick Description

- First you need to load the training dataset which consists of 40 class (folder), each class represents one person. Each class has 10 images, taken in different positions.
- Create Eigen-faces matrix for the dataset, which will be used later to compare with any new test image.
- Load your test image then start the matching (recognition) process.
- This function runs in a separate QThreads to ensure best quality and prevent GUI from freezing as it may take some seconds to finish.

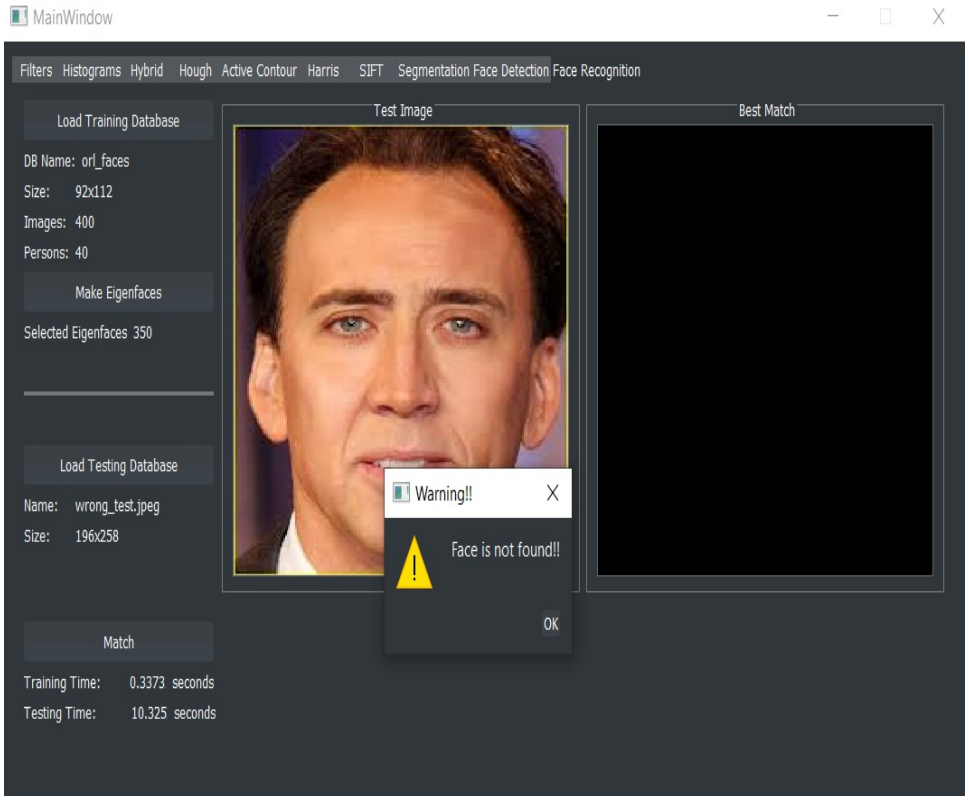
Face Recognition with correct test





The output image `Best Match`, is just a combination of all the class images in the database, just for displaying purposes to make it clear to the user.

Face Recognition with wrong test



This repository is created by a group of 4 students in Biomedical Engineering Department, Cairo University. ©

Name	Section	B.N Number
Ahmed Salah El-Dein	1	5
Ahmad Abdelmageed Ahmad	1	8
Ahmad Mahdy Mohammed	1	9
Abdullah Mohammed Sabry	2	7