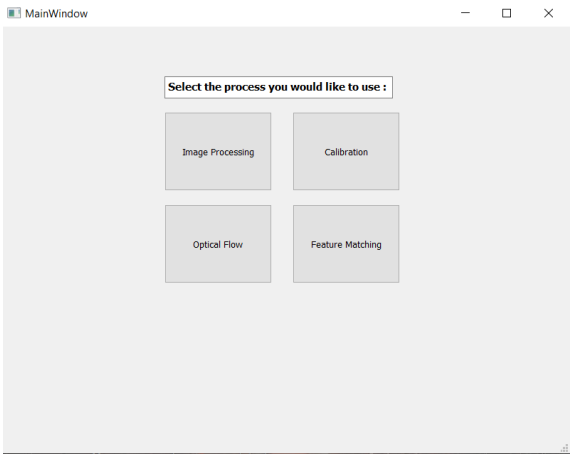
OpenCV Report

This report includes the background and foreground (user interface) explanations of OpenCV GUI contains different properties about Image Processing, Calibration, Optical Flow and Feature Matching.

This project was created using OpenCV 3.2.0 on QT Creator with C++.

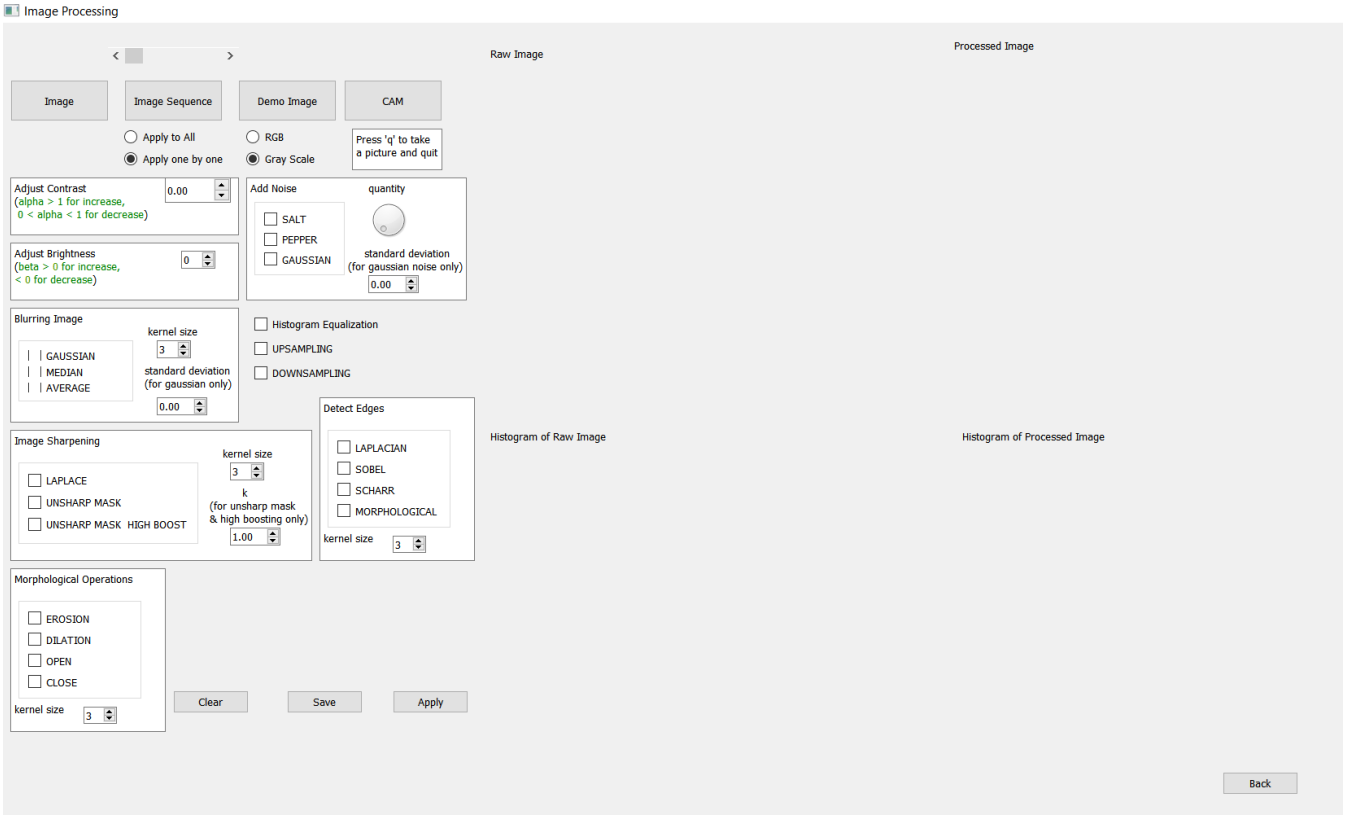
User Interface



Main screen where user can continue selecting one of these 4 type of operations.

Image Processing

By left clicking on Image Processing button, the user will see the following main window.

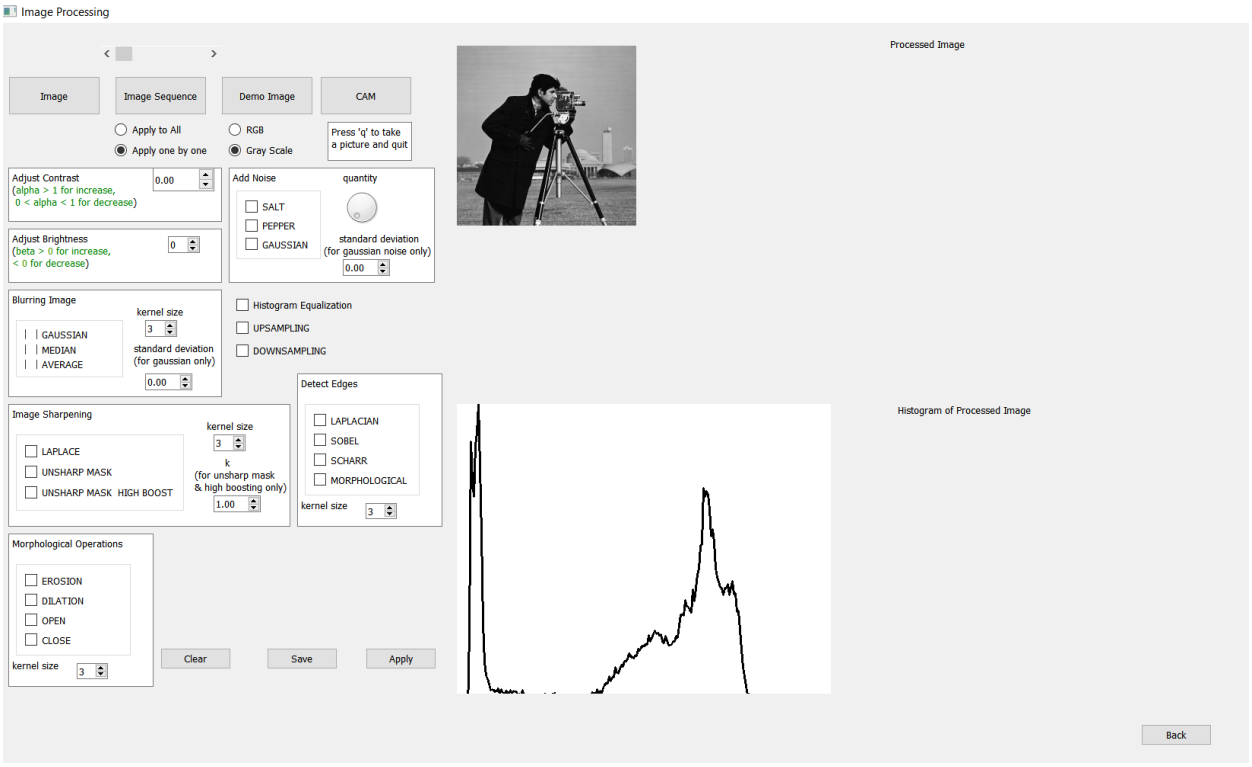


The user can select Demo Image to use a default image coming from App and start using the features. For this option, the user can select whether they want to use a Gray Scale default image or RGB default image.

Any change can be applied but clicking Apply button and the result image will be seen at the right side with result histogram.

Working with Demo Images

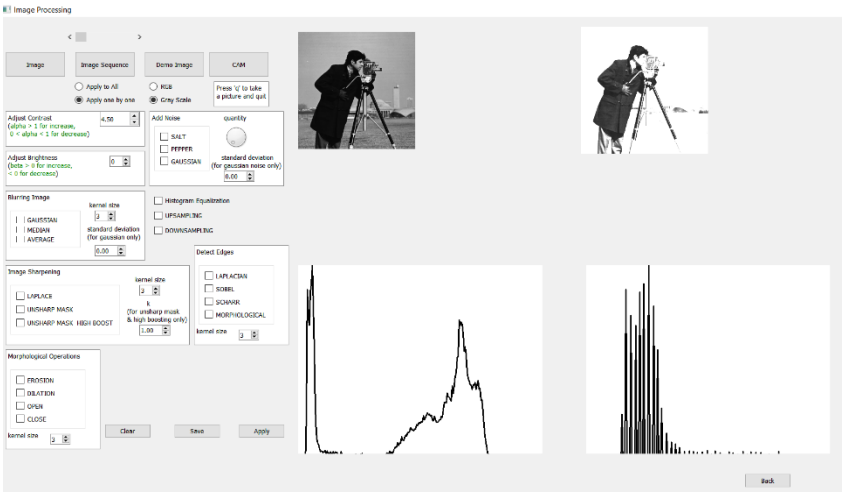
- Gray Scale



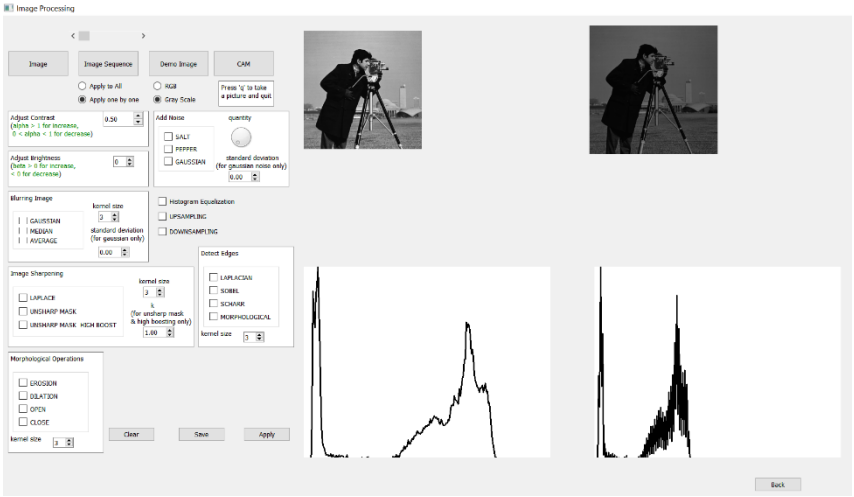
By choosing Gray Scale demo image, we see cameraman.png and its initial histogram. Now we can apply any process between

- Adjust Contrast: The value of alpha can change with the arrows or simply any value can be written in the box at the right side in adjust contrast plane. Increasing alpha will increase the contrast and decreasing between 0 and 1 will decrease the contrast. Each step taken with arrows are having size 0.25. At the background, the applying functions is as follow:

```
new_image (i, j) = image(i, j) * alpha
```



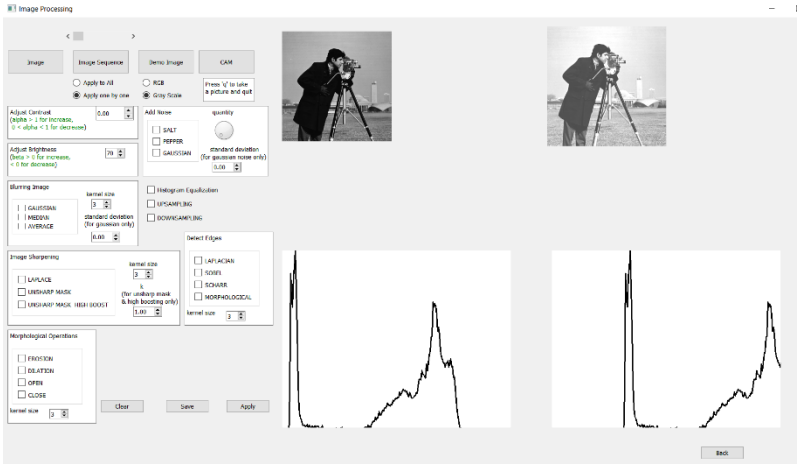
Sample usage of Adjust Contrast with alpha = 4.50



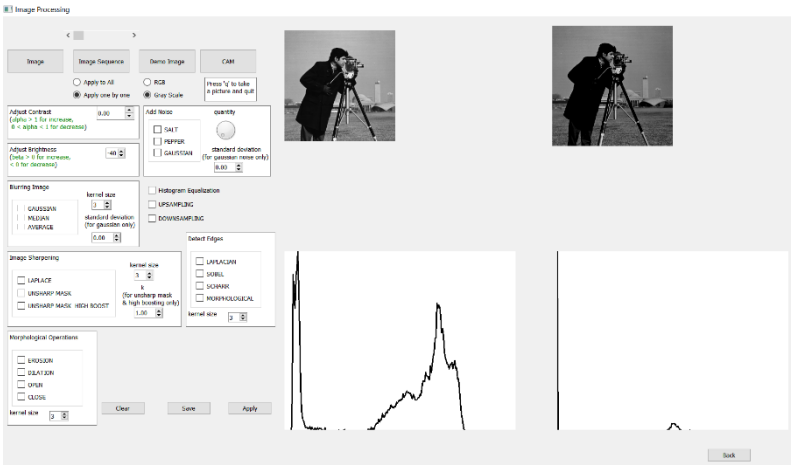
Sample usage of Adjust Contrast with alpha = 0.50

- Adjust Brightness: The value of beta can change with the arrows or simply any value can be written in the box at the right side in adjust brightness plane. Increasing beta will increase the brightness being beta > 0 while decreasing beta will decrease the brightness being beta < 0. Each step taken with arrows are having size 10. At the background, the applying functions is as follow:

```
new_image (i, j) = image(i, j) + beta
```

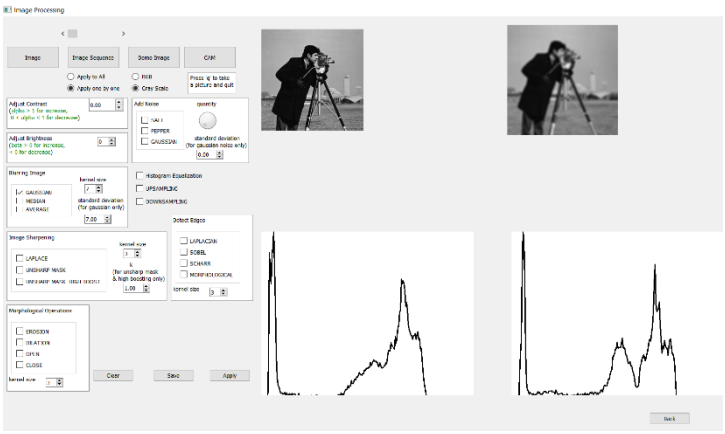


Sample usage of Adjust Brightness with beta = 70

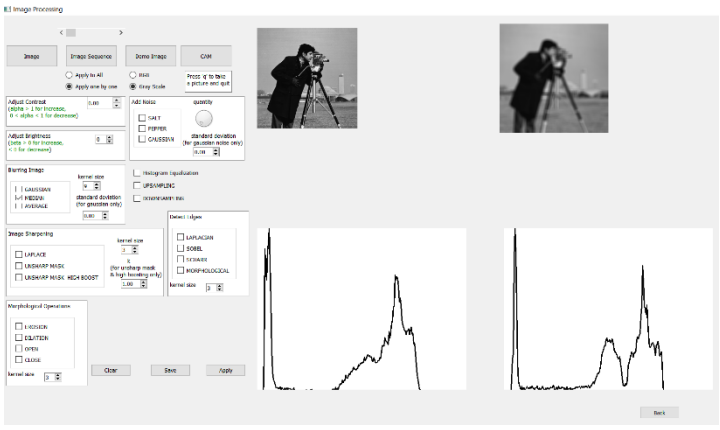


Sample usage of Adjust Brightness with beta = -40

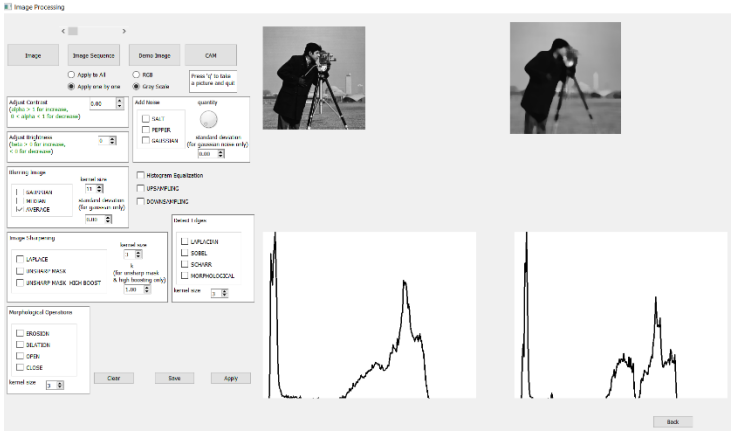
- Blurring Image: The user can blur the image selecting one of Gaussian, Median, Average filters and can arrange the kernel size which has 3 as **default** and **minimum** value. Each kernel size step is 1. In case of choosing Gaussian filter, standard deviation can be determined with the small box at the right side.



Sample usage of Blurring Image with Gaussian Filter kernel size = 3, standard deviation = 7



Sample usage of Blurring Image with Median Filter kernel size = 9

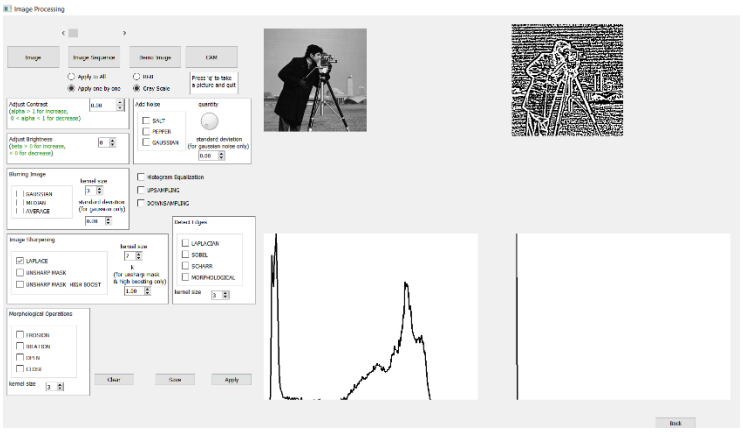


Sample usage of Blurring Image with Average Filter kernel size = 11

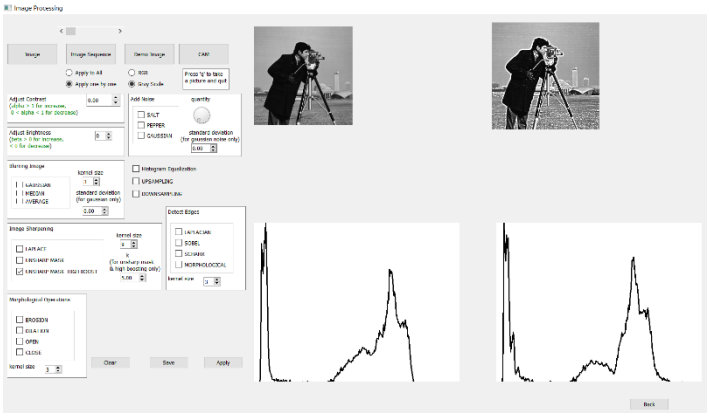
- Image Sharpening: Similar to Blurring Image operation, the user can select one of the sharpening filters between Laplace, Unsharp Mask and Unsharp Mask & High Boost by determining the kernel size. Unlike image blurring, each step increases the kernel size by 2. The reason for that is sharpening kernels require an odd value size and gives run time error otherwise. Additionally, in case of Unsharp Mask & High Boost, the value of k, which is 1 by default, can be given by the user.
  k is the coefficient which determines the effect of unsharp mask m, which is obtaining by a subtraction of a blurred version of image from the input image.

$$m(x,y) = f(x,y) - f_b(x,y)$$

$$g(x,y) = f(x,y) + k * m(x,y)$$

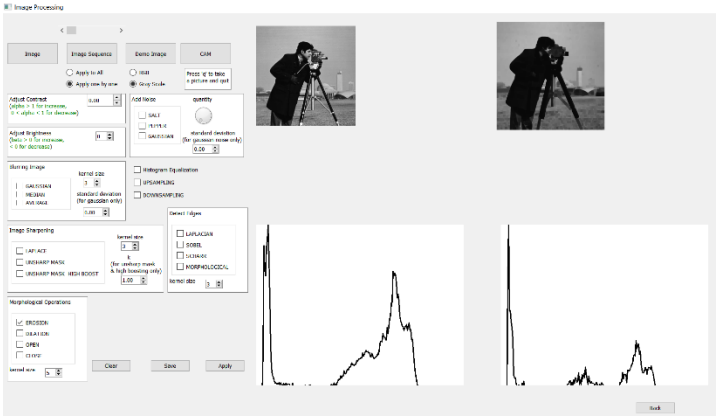


Sample usage of Image Sharpening with Laplace Filter kernel size = 7

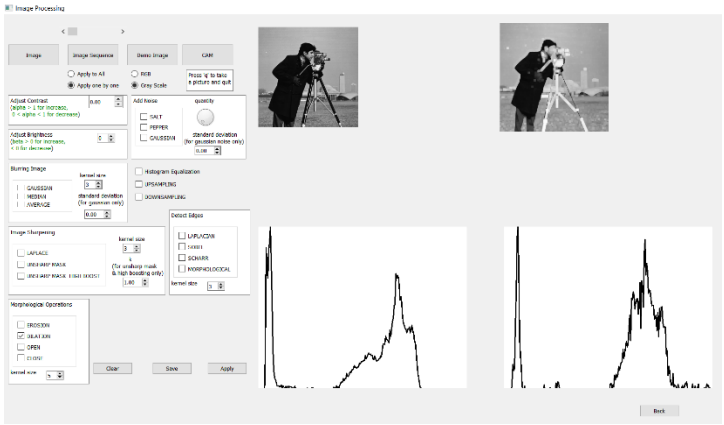


Sample usage of Image Sharpening with Unsharp Mask & High Boost kernel size = 9, k = 5

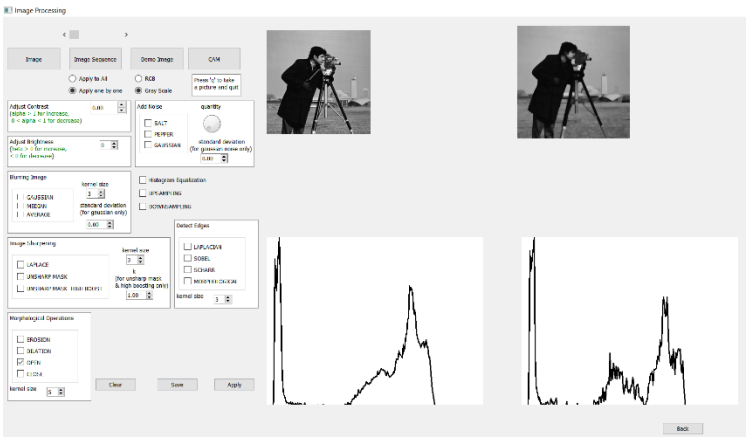
- Morphological Operations: The user can apply Dilation, Erosion, Opening or Closing with chosen kernel size.



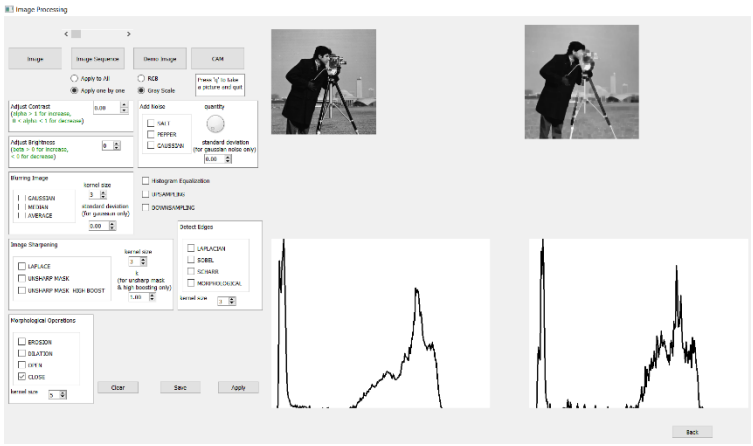
Sample usage of Morphological Operations with Erosion and kernel size = 5



Sample usage of Morphological Operations with Dilation and kernel size = 5

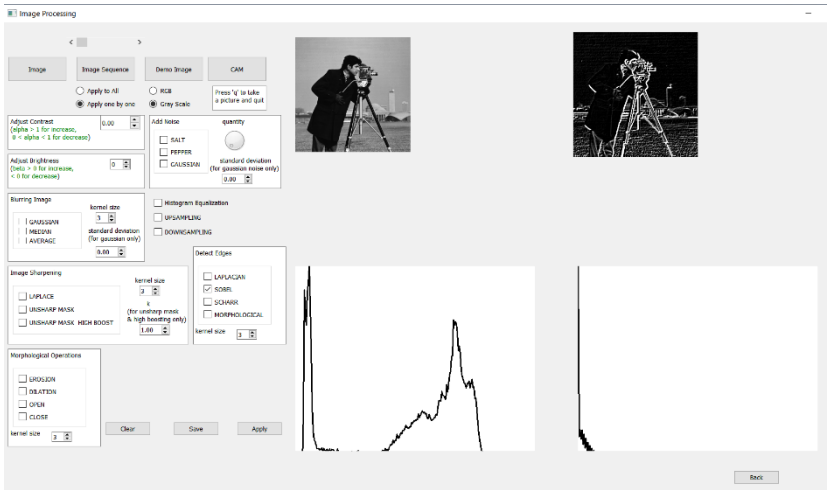


Sample usage of Morphological Operations with Open and kernel size = 5

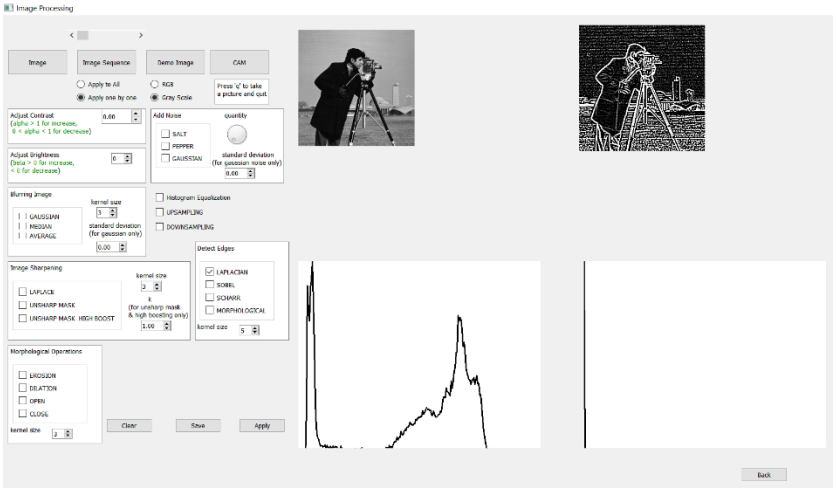


Sample usage of Morphological Operations with Close and kernel size = 5

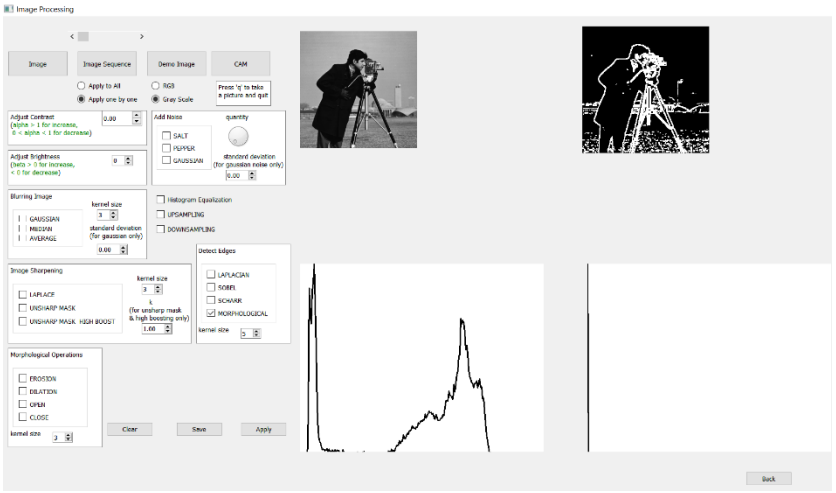
- Detect Edges: The user can detect edges using Laplacian, Sobel, Scharr or Morphological filters by adjusting kernel size which is 3 as default and minimum value.



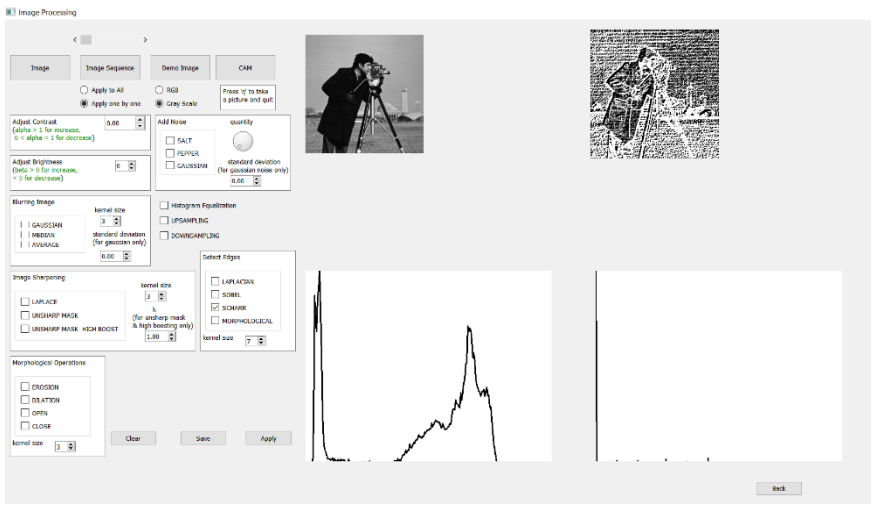
Sample usage of Edge Detection with Sobel filter and kernel size = 3



Sample usage of Edge Detection with Laplacian filter and kernel size = 5

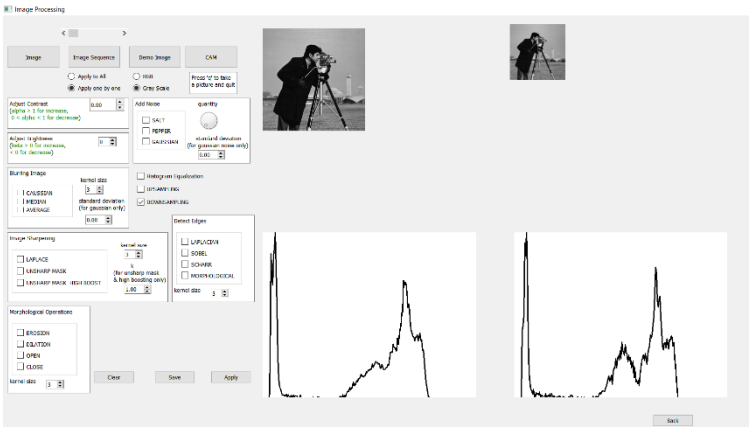


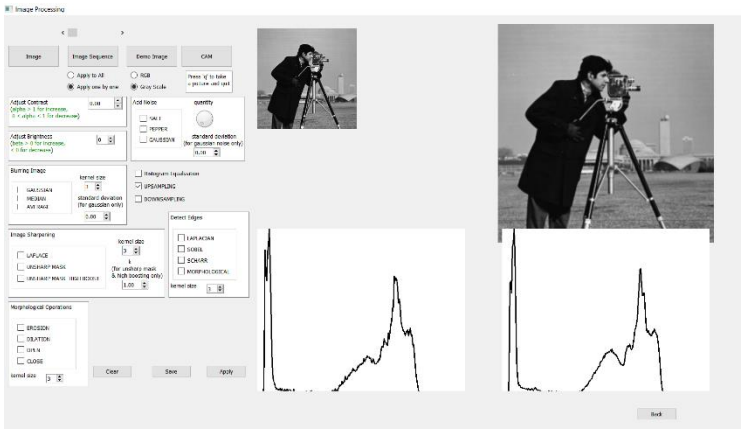
Sample usage of Edge Detection with Morphological operators and kernel size = 5



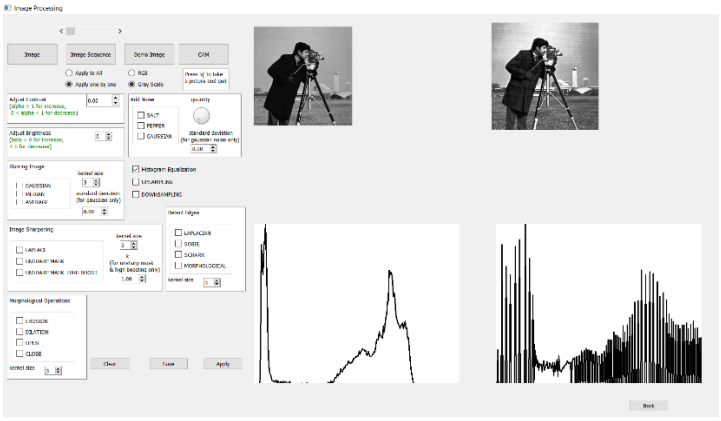
Sample usage of Edge Detection with Scharr filter and kernel size = 5

- Upsampling and Downsampling: We can up or down sample the image selecting upsampling or downsampling boxes. Upsampling will resize the image by %200 and downsampling will resize the image by %50



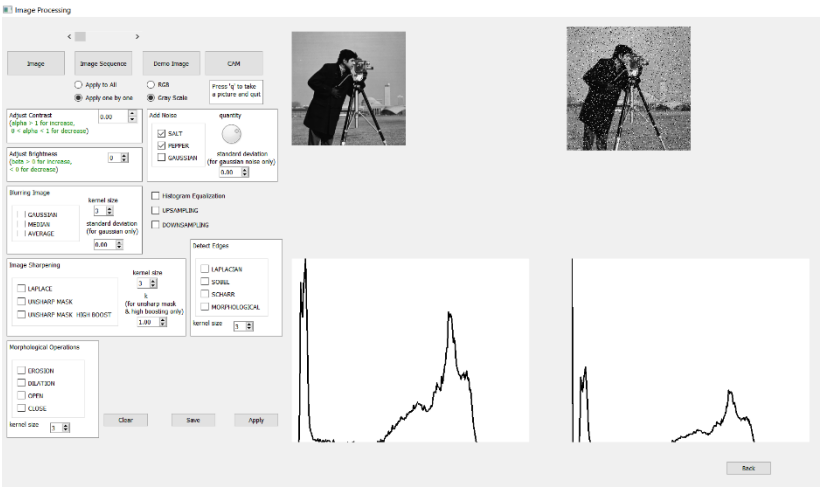


Histogram Equalization: By selecting Histogram Equalization we obtain the equalized image and histogram.

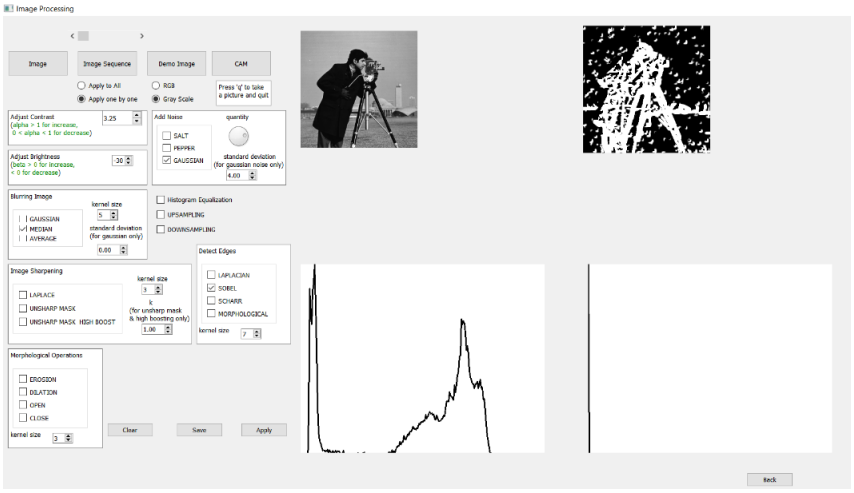


- Adding Noise:

The user can add Pepper, Salt or Gaussian noise selecting the related box. The quantity of noise can be determined by using the circle shaped button at right. This is the quantity of pixels to which the noise will be added. More than one selection is enabled, so for a salt and pepper noise it is sufficient to choose both pepper and salt noise. For gaussian noise, the user can determine standard deviation too.



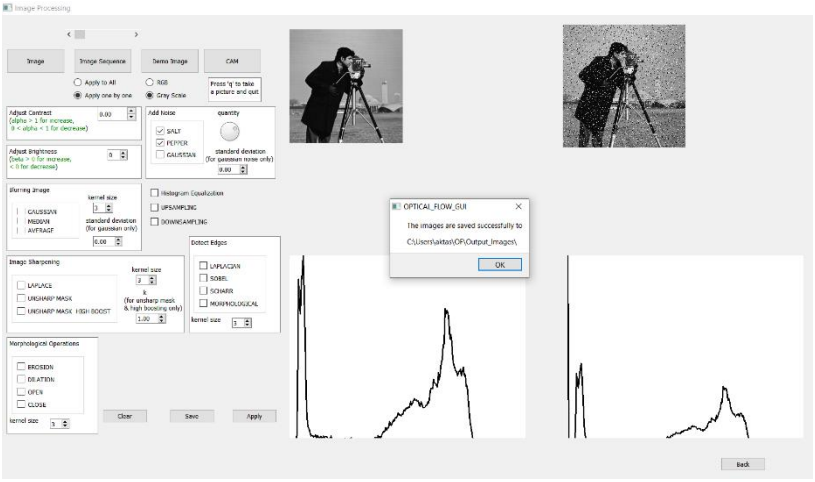
Although I applied each operation one by one to show the output, the effect of the operation, it is possible to choose more operations at a time and apply them in sequence.



Sample usage of choosing different operations with different parameters at a time.

!!! For image sharpening, blurring, edge detection and morphological operations, although it is not logical to choose more than one filter type at the same type, it is not prevented. Because to have this property, the GUI settings do compulsory at least one chose. That means the user would be forced to apply these operations with 1 selected kernel.

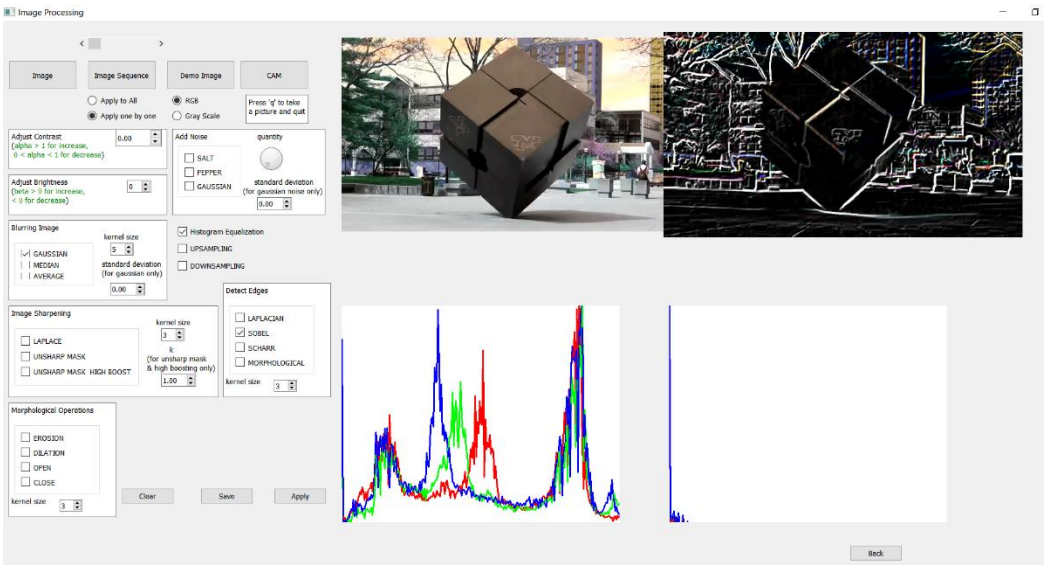
- Save Button: The user can save any output image by clicking save button. That will save the output image to the default directory and send a message to main window.



- Clear Button: After selecting many different operations with different parameter values, the user may want to turn default values. This button clears all the choices.

- RGB

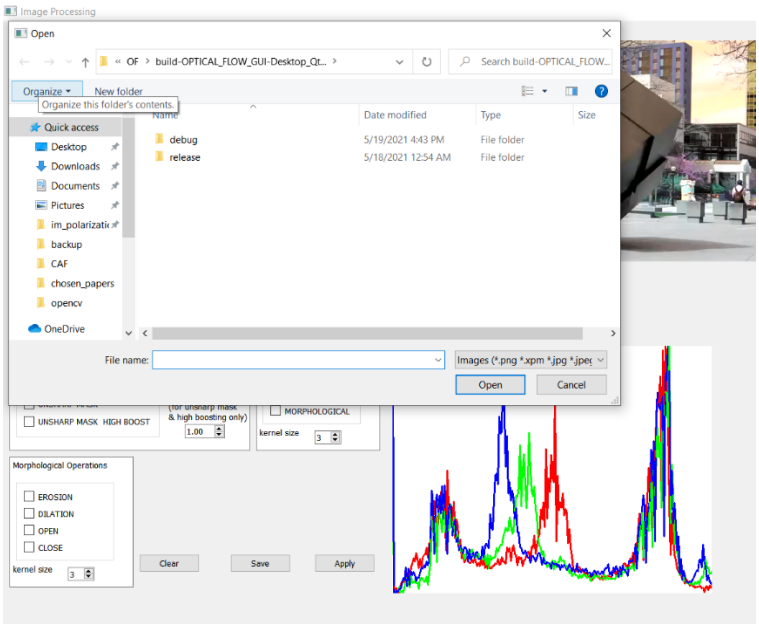
Every single operation shown in Gray Scale section is available for RGB images too. Clicking Demo button with RGB button checked will give the following default image. We see a sample output obtained after applying histogram equalization, edge detection, image blurring and contrast adjusting.



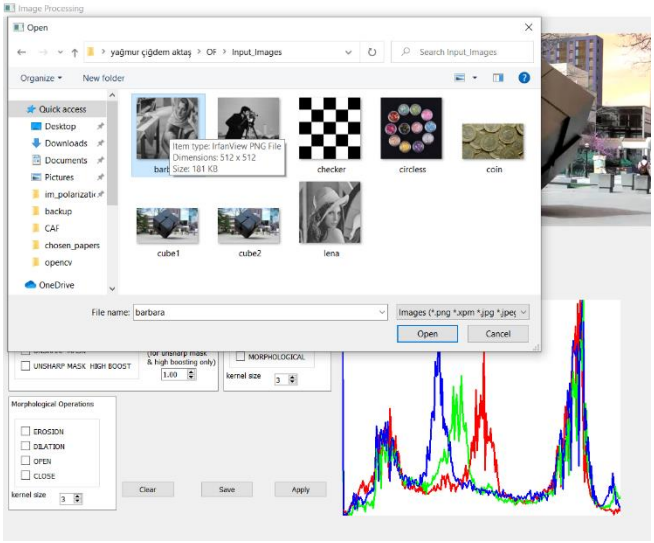
Working with User Images

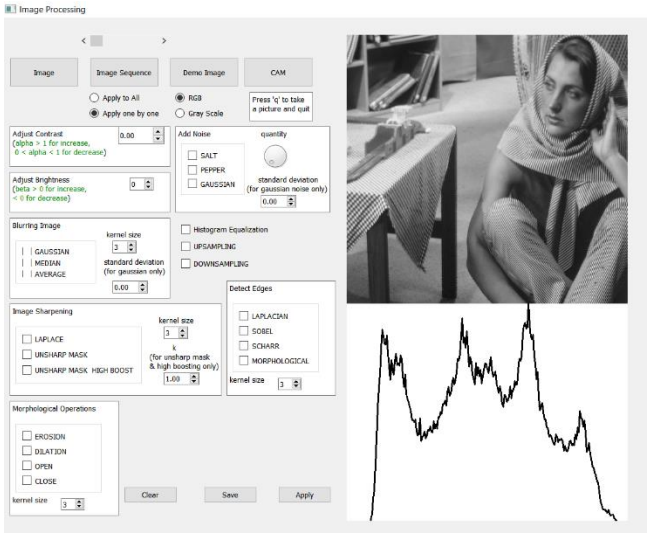
- Image Button

  By clicking, a file explorer window appears and the users can select any image from their local.



As an example, when I select barbar.png from my local folder, we see the selected image uploads on the main window.

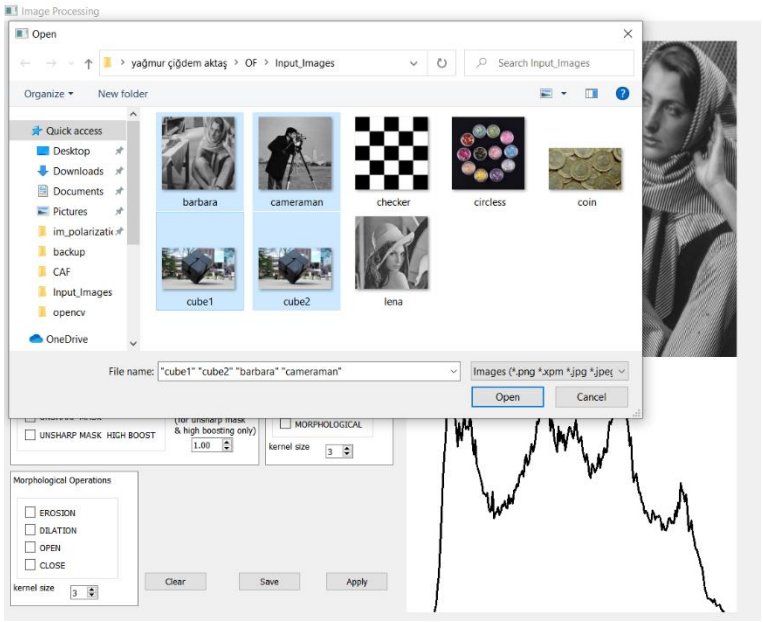




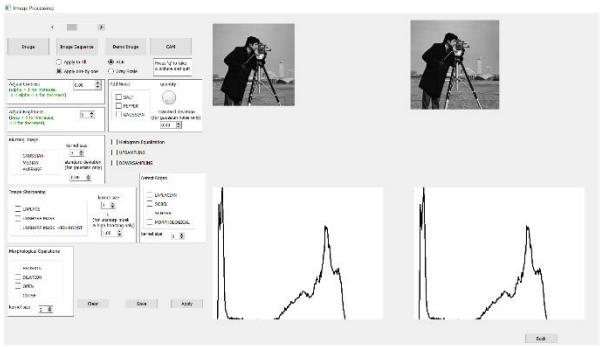
Now any operation we did in Demo section can be applied.

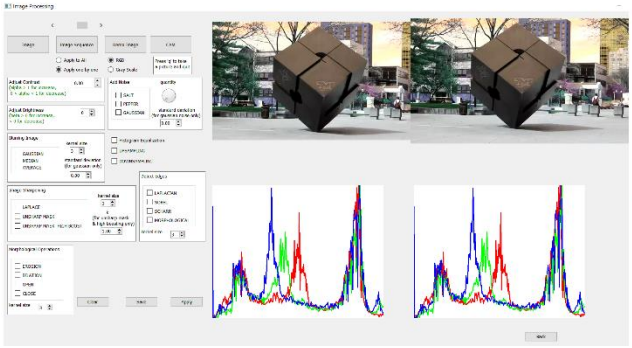
- Image Sequence Button

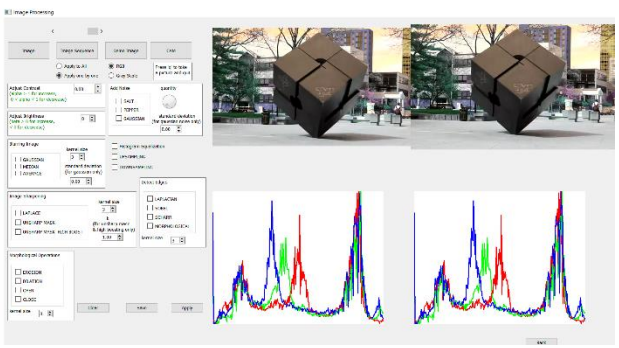
  This button provides user to upload more than 1 image, apply the changes on all the images uploaded at a time. By selecting “apply to all” or “apply one by one”, the user can change the decision any time whether the operations will apply on all the images or just the current image.



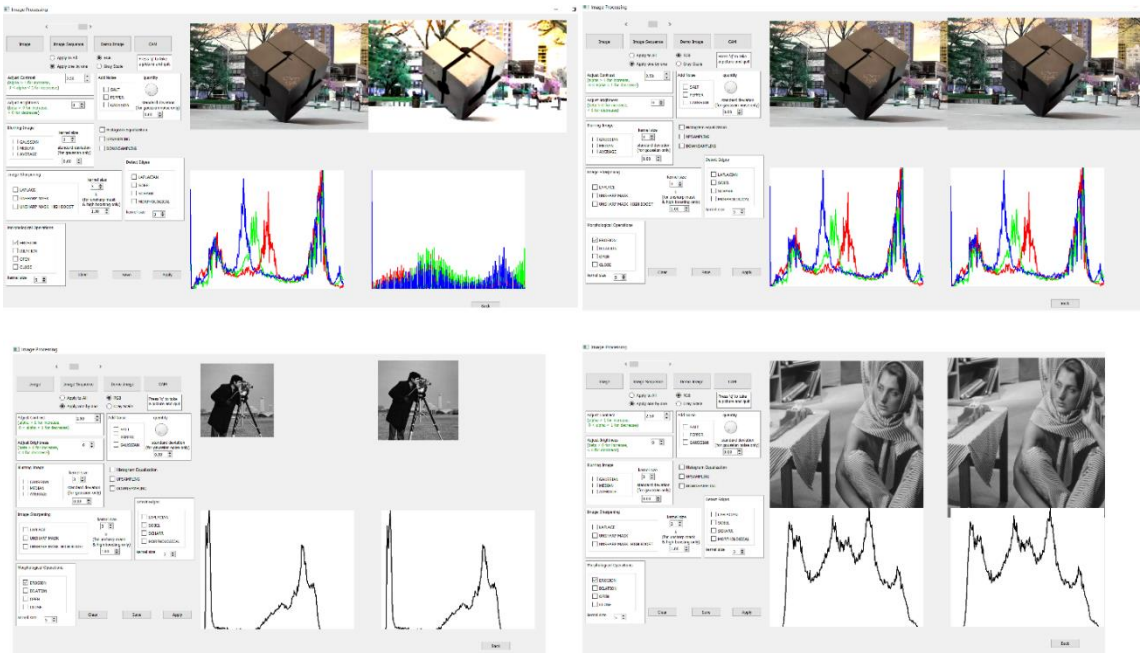
As an example, when we select the following 4 different images, we see each image is uploaded and we can go forward and back by using red marked slider.



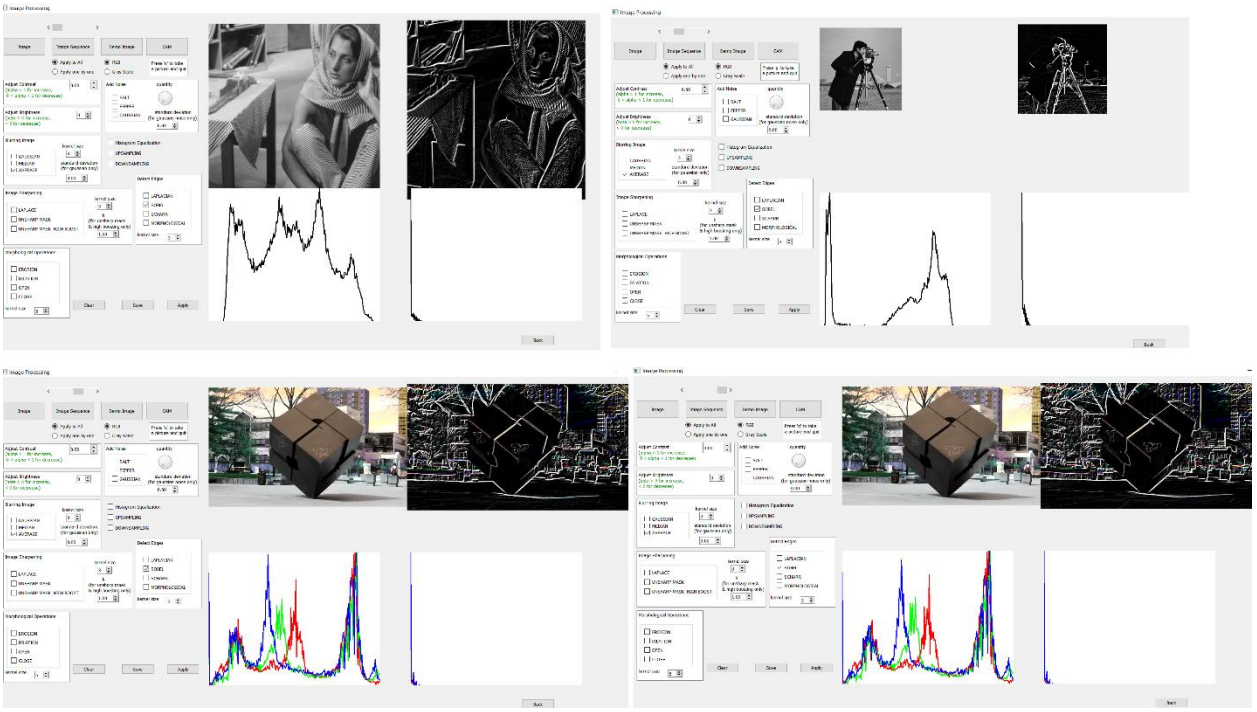




As an example of “apply one by one” selection, when we apply contrast adjusting and Erosion having the last image (cube2) on the current screen, we obtain a processed output image for cube2 while for other images we any change doesn’t apply. The result is shown below:



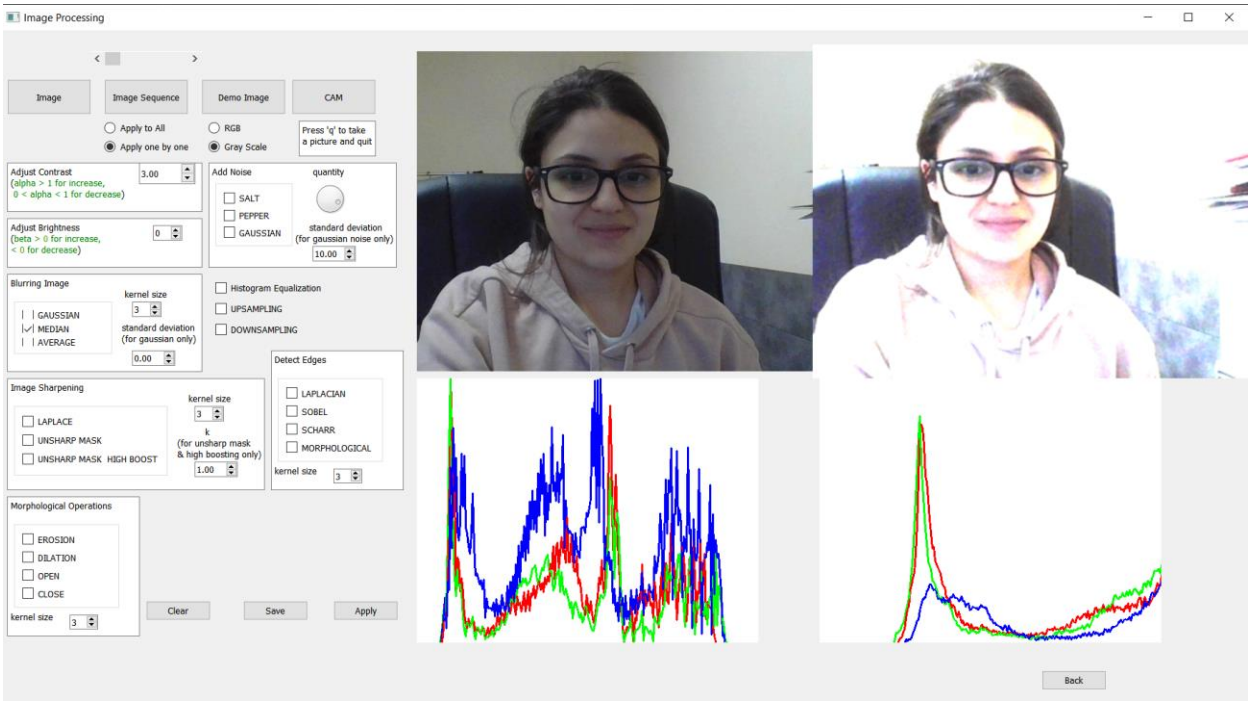
As an example of “apply to all” selection, when we apply image blurring and edge detection having the first image (barbara) on the current screen, we obtain a processed output image for all the images. The result is shown below:



!!! The reason we see the initial images and histograms at both input and processed image places, when we first upload image sequence is something I should have implemented to handle a bug related to this sequence feature.

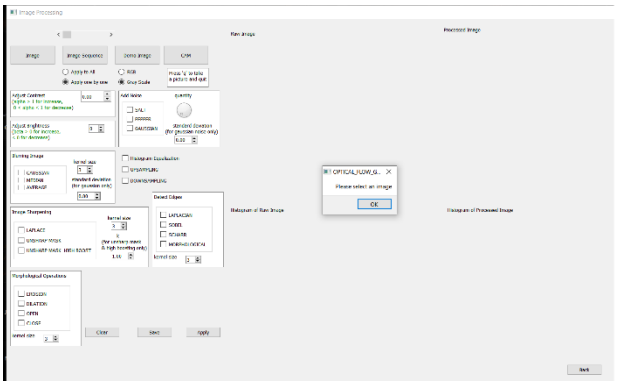
- CAM Button

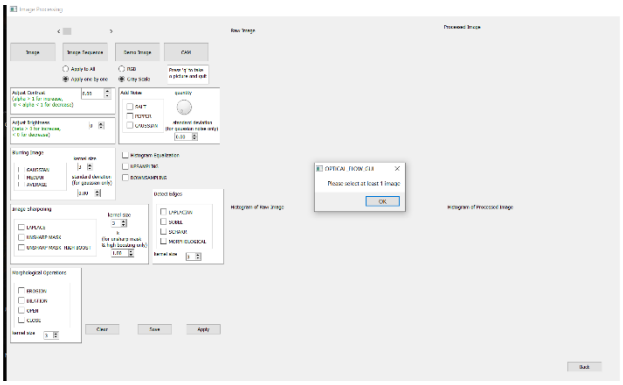
  Using this button, it is possible to use local webcam, take a picture and use it as initial image. The picture is taken when the user press ‘q’ and it is uploaded to input image place. After this step, any operation can be applied.



Measures Taken for This Section

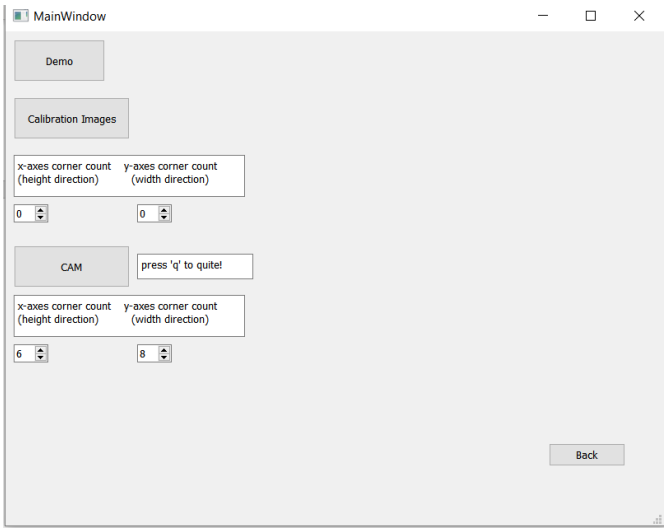
1) If the user tries to give a pair value by not using the arrows but typing in the box, the kernel size is increased by 1 to get an odd value at the background. This is done for edge detection and image sharpening filters to prevent a run time error which occurs with pair kernel size.
2) Additionally, a kernel size can have value between 3 and 31. The limit is determined at the background to prevent any invalid kernel size selection which occurs run time error.
3) If the user clicks Image or Image Sequence button but forget to select any image or cancel the window, a warning message appears in the main window and the error is catching at the background. The user can continue and select any option without any problem.





Calibration

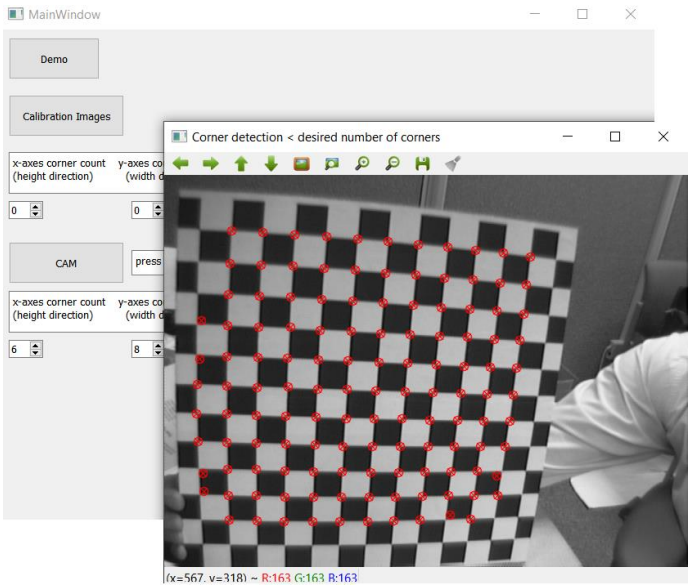
By clicking calibration button on the main window, we can reach Calibration main window.
There are 3 different usage of this section and each of them uses Zhang Calibration with checkerboard pattern images.

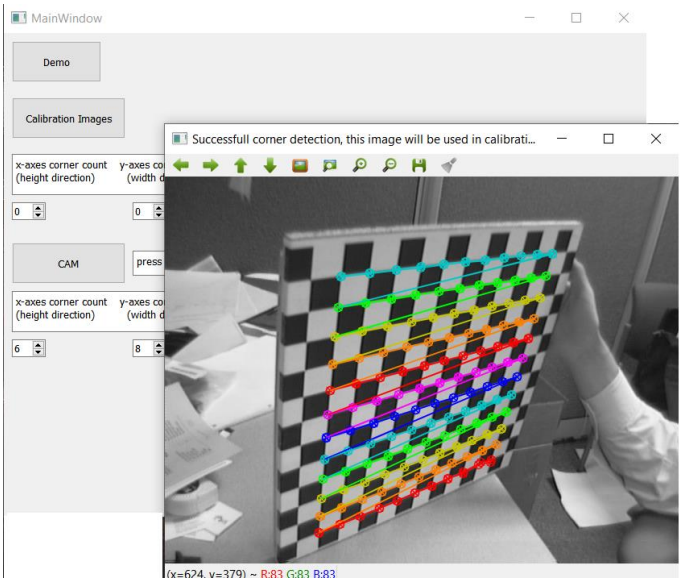


- Demo

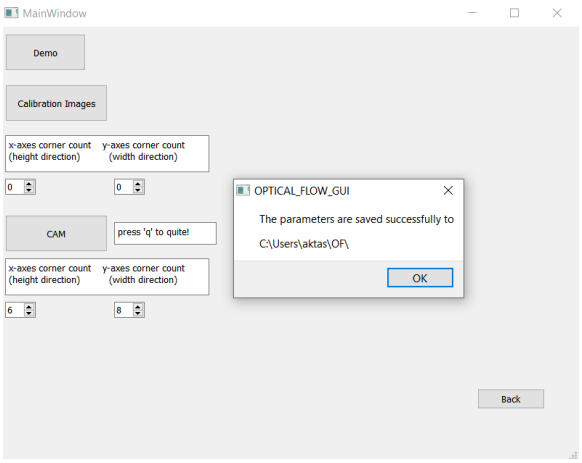
  By clicking Demo button, the user can use default checkerboard pattern images to see how it is working. Each image with corner detections drawn on them, are showing. If expected number of corners can be detected, it is a successful detection. This type of images are shown with a message “Successful corner detection, this image will be used in calibration” and if less than expected corners are detected, the

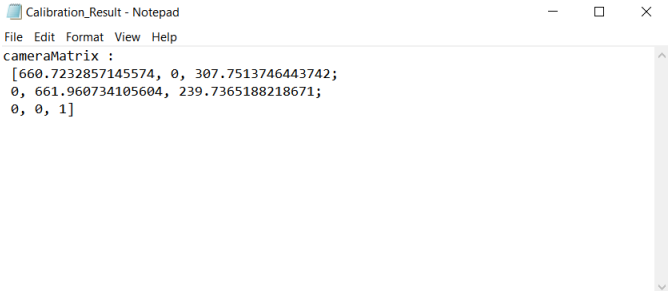
image is shown with a message “Corner detection < expected number of corners”.





Afterwards, only the successful images are used for calibration and we get another message if calibration is done without any problem. The message shows the directory where the calibration results are saved in a file. By looking the mentioned file, we see following result:

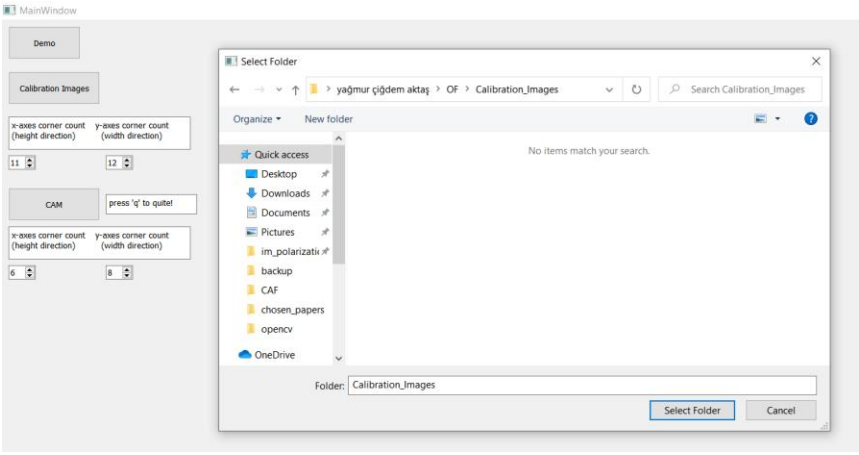


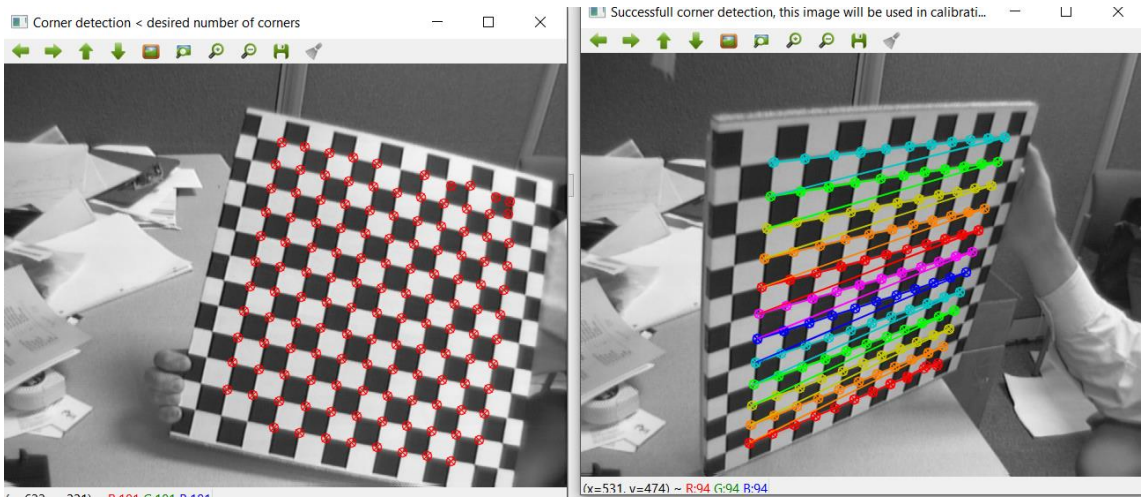


- Calibration Images

  By clicking this button, the user can select the folder where his own calibration images are. Before selecting this folder, the user should correctly determine the number of corners to detect in x and y axes. After that, the calibration process will start automatically.

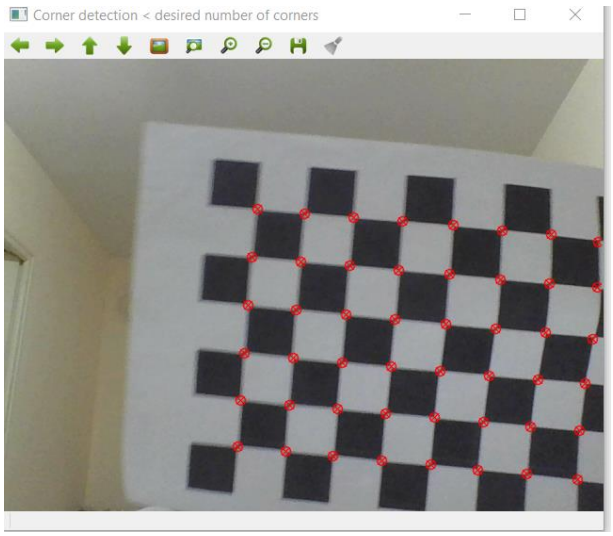
  I determine the number of corners then select the folder where my calibration images are and we see the same process works.

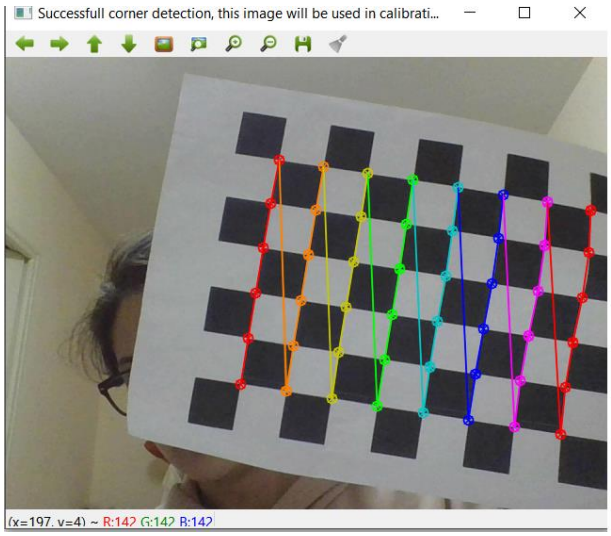




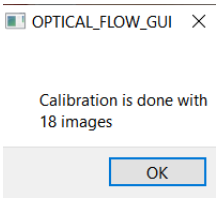
- CAM

It is also possible to calibrate cameras via webcam. The user can click on CAM button right after determining expected corner counts and show the checkerboard images to the camera. The same process will work. To quit, the user should press ‘q’.

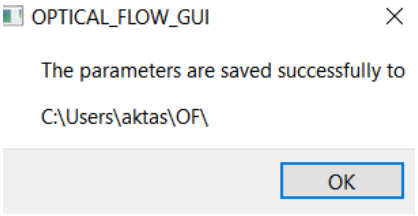


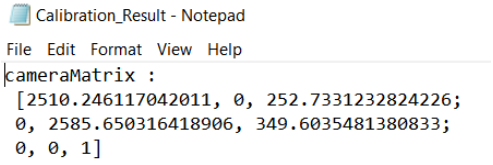


For CAM option, there is only an additional message is sent to interface to declare how many images are used for the calibration. For how many images, corner detection was successful. Because it can get a little confusing to see if the image were good or not trying to show the checker pattern to the camera.



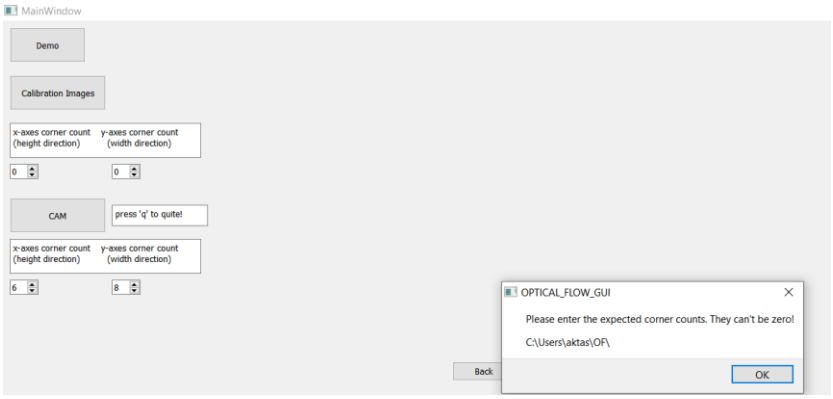
Further process is the same:



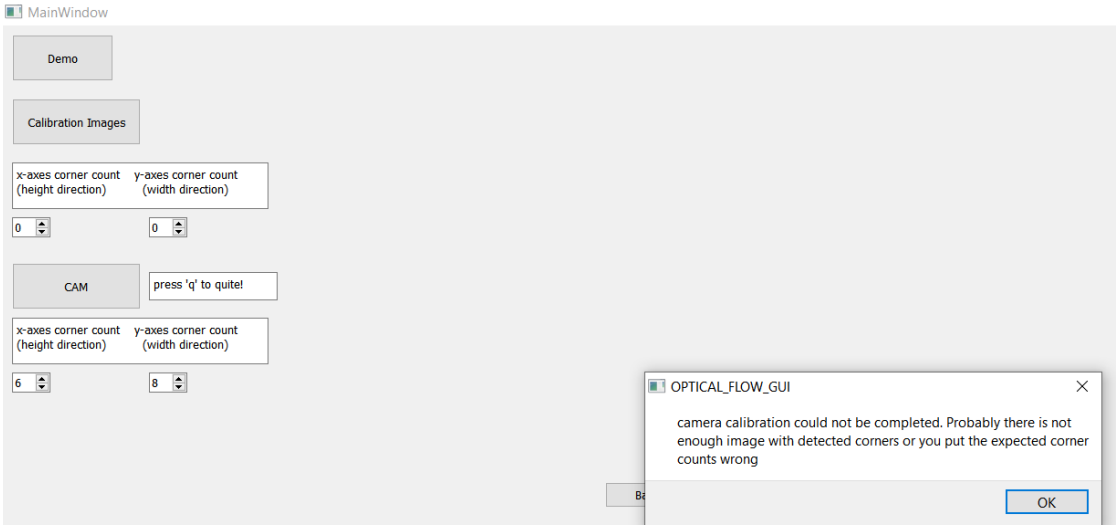


Measures Taken for This Section

1) If user forgets to determine corner the calibration will fail. So instead of opening a file explorer, a warning message is sent.



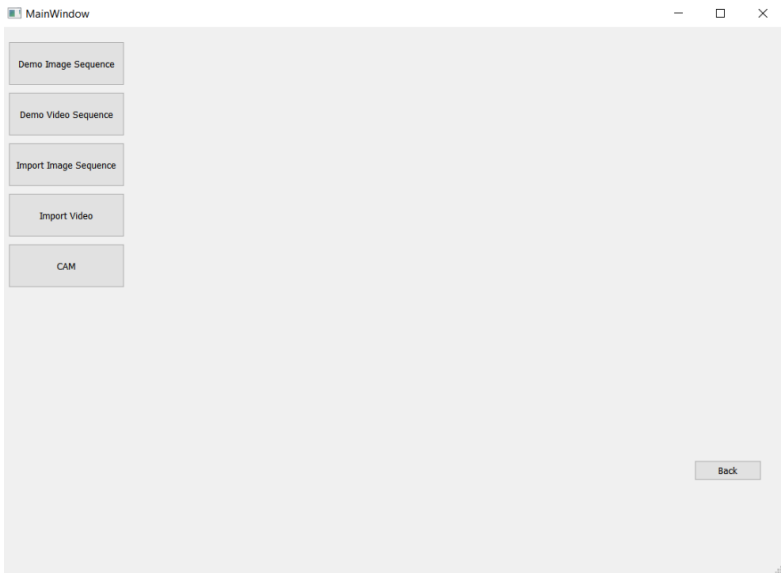
2) The calibration may not be successful every time. Two main reason for that is not having enough successful images. Either the expected corner quantity is given wrong or there are small number of images. For example, the user may press ‘q’ too early in CAM case. In that case, to warn the user that the calibration is not performed, a message is sent.



3) These problems cause run time errors so they are cached at the background too.

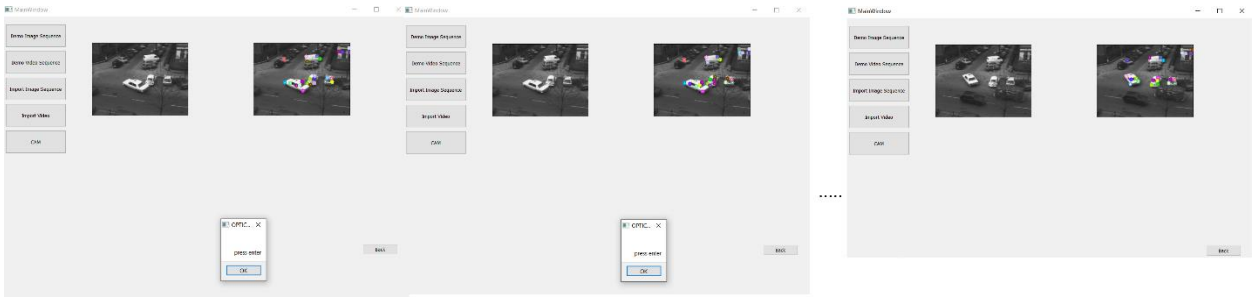
Optical Flow

By clicking Optical Flow button on the main window, the user can start using Optical Flow features. For each option, Lucas Kanade method is implemented.



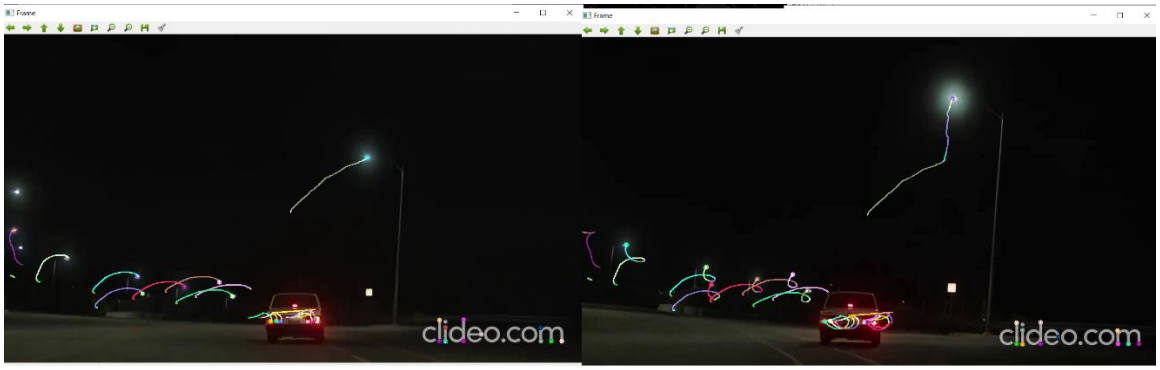
- Image Sequence

  By clicking Import Image Sequence, the user should select the folder where the images are. Then for each image sequence, optical flow will be computed and drawn. At the left side we see the current image of sequence, whereas at the right side we see optical flow arrow. After each result, a message telling “press enter” is sent. It is necessary to pass next image.



- Video

  By selecting a video, we see the scenes with optical flow arrows.



- CAM

  It is possible to open the webcam and see optical flow arrows between webcam frames.

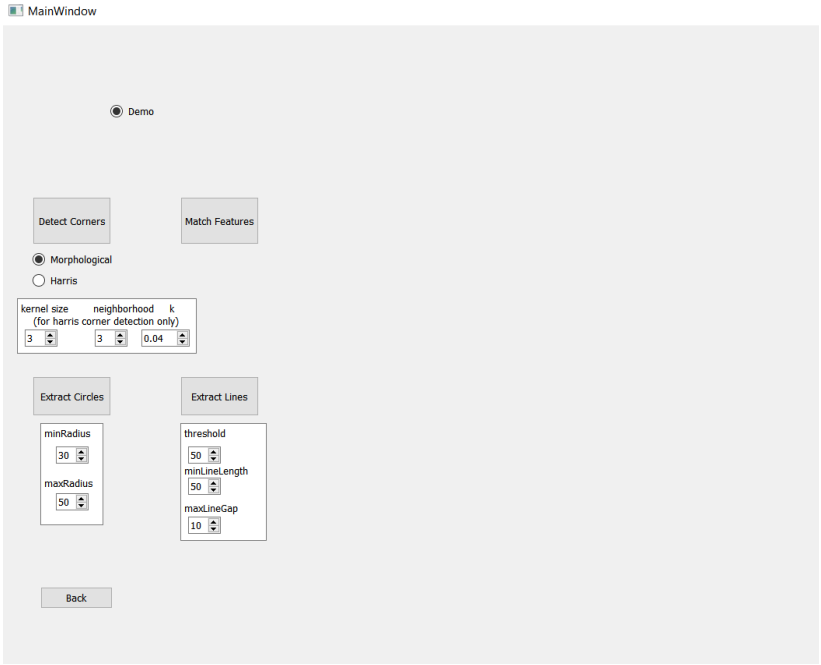
  Like previous sections, the user can test or see the outputs using Demo Image Sequence and Demo Video buttons with default sequences.

Measures Taken for This Section

1) In case there is no optical flow, no difference between frames coming from webcam, the program gives a run time error with -251 code which means in calcOpticalFlowPyrLK(old_gray, gray, p0, p1, status, err,

cv::Size(15,15), 2, criteria); function, p0 is 0 instead of any expected value. So the input array which represents previous image's points are zero. To handle it, I decided to not assign the frame as old-previous image when p0 comes zero from goodFeaturesToTrack(old_gray, p0, 100, 0.3, 7, cv::Mat(), 7, false, 0.04); function. It does the performance worse than earlier and a little bit more noisy to track the optical flow of features. But it solved the run time error, so I decided to apply it for now.

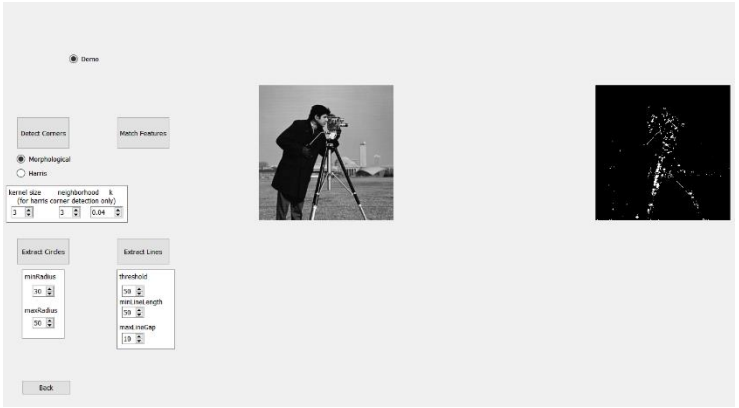
Feature Matching



If Demo box checked, the process will be done with default image/images. If not, for detect corners, extract circles and extract lines, 1 image will be required. For match features, 2 images will be required. In case any image didn't selected, a warning image is sent, in case only 1 image selected for Match Features, same image will be used as left and right images.

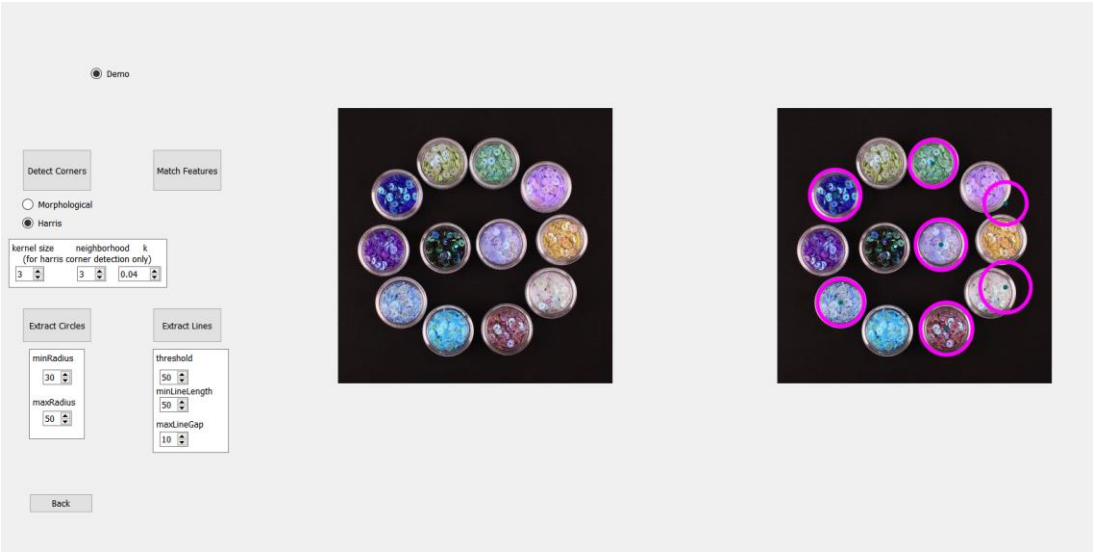
- Detect Corners

Harris Corner or Morphological operators can be selected for that process. For Harris Corner, the user can define kernel size, neighborhood and k. We see demo image results for both:

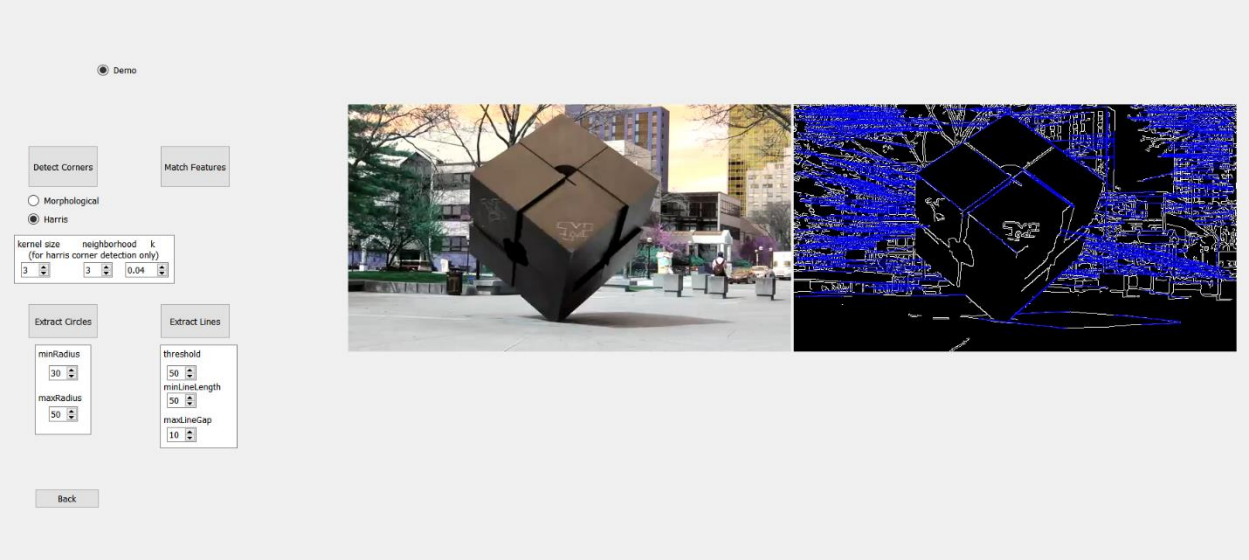


- Extract Circles

  For extract circles, the user can define minimum and maximum radius of circles. Any circle being outside of this range won't be extracted. We see demo image results.



- Extract Lines
  The user can define threshold, minimum line length and maximum line gap.



Back Button : This is a button finding in each submain window(Image Processing, Calibration, Optical Flow, Feature Matching) which provides user to go back to the main window.

Background Implementation

1. Image Processing

   ### Image_Processing.cpp & Image_Processing.hpp :
   Main files contains image processing functions

Image_Processing.hpp | <Select Symbol>

```
typedef enum
{
    GRAYSCALE,
    RGB,
    NOT_DETECTED,
}IMAGE_TYPE;

typedef enum
{
    UINT8,
    UINT16,
    INT8,
    INT16,
}PIXEL_TYPE;

typedef enum
{
    GAUSSIAN,
    AVERAGE, // Mean Filter as the 2. name
    MEDIAN,
}BLUR_TYPE;

typedef enum
{
    LAPLACE,
    UNSHARP_MASK,
    UNSHARP_MASK_AND_HIGH_BOOSTING,
}SHARPENING_TYPE;


typedef enum
{
    SALT,
    PEPPER,
    SALT_AND_PEPPER,
    GAUSSIAN_NOISE,
}NOISE_TYPE;

typedef enum
{
    UPSAMPLING,
    DOWNSAMPLING,
}SAMPLING_TYPE;

typedef enum
{
    EROSION,
    DILATION,
    OPEN,
    CLOSE,
}MORPHOLOGIE_TYPE;

typedef enum
{
    SOBEL,
    PREWITT,
    SCHARR,
    LAPLACIEN,
    MORPHOLOGICAL,
}EDGE_DETECTION_TYPE;
```

Blur and sharpening operations has different kernels to use. According to the kernel selected by user from GUI, it is decided which kernel – which opencv function to use in the related function. Image and pixel type is not an automatic assignment coming from GUI, instead, by looking the chosen image, these types are determined by Check_Image_Type(). Pixel type is important when iterate over the image planes for detect if the image is RGB or a 3 channel – GrayScale in Check_Image_Type for example. On the other hand, Histogram_Plot, Histogram_Equalization, Add_Noise functions requires the info if the image is Grayscale or RGB.

```
cv::Mat Change_Contrast(cv::Mat, double); // alpha > 1 for increase, 0 < alpha < 1 for decrease the contrast
cv::Mat Change_Brightness(cv::Mat, int); // beta > 0 for increase, < 0 for decrease the brightness
cv::Mat Image_Blur(cv::Mat, BLUR_TYPE, uint8_t kernel_size=3, double std_deviation= 0); // std_deviation is just for Gaussi
cv::Mat Image_Sharpening(cv::Mat, SHARPENING_TYPE, uint8_t , double k=1);
cv::Mat Histogram_Plot(cv::Mat);
cv::Mat Add_Noise(cv::Mat, NOISE_TYPE, int, float std_deviation=5.0); // last parameter is the std deviation and is just fo
cv::Mat Sampling(cv::Mat, SAMPLING_TYPE);
cv::Mat Morphological_Operations(cv::Mat, MORPHOLOGIE_TYPE, uint8_t kernel_size = 3);
cv::Mat Edge_Detection(cv::Mat, EDGE_DETECTION_TYPE, uint8_t kernel_size=3);
std::vector<float>  Get_Histogram_Info(cv::Mat);
IMAGE_TYPE Check_Image_Type(cv::Mat);
cv::Mat Histogram_Equalization(cv::Mat);
```

ipwindow_ui & ipwindow.cpp & ipwindow.h:

Main files contains Image Processing GUI window class and member functions of this class.

ipmainwindow.h <Select Symbol>

```
    ~IPmainwindow();

private slots:
    void on_pushButton_clicked();

    void on_pushButton_2_clicked();

    void on_pushButton_3_clicked();

    void on_sequence_iterator_valueChanged(int value);

    void on_alpha_valueChanged(double arg1);

    void on_Apply_clicked();

    void on_kernel_size_valueChanged(int arg1);

    void on_std_dev_valueChanged(double arg1);

    void on_kernel_size_2_valueChanged(int arg1);

    void on_k_valueChanged(double arg1);

    void on_std_dev2_valueChanged(double arg1);

    void on_noise_quant_valueChanged(int value);

    void on_edge_kernel_valueChanged(int arg1);

    void on_morpho_kernel_size_valueChanged(int arg1);

    cv::Mat Apply_Changes(cv::Mat);

    void on_pushButton_4_clicked();

    void on_pushButton_5_clicked();

    void on_pushButton_6_clicked();

    void on_pushButton_7_clicked();

    void on_beta_valueChanged(int arg1);
```

Each button in the Image Process mainwindow has a background function as a member function of IPmainwindow class. Clicking the button triggers related "_clicked()" function or change the value of any box (kernel size, noise quantity etc), triggers related "_valueChanged()" function and the new value is assigned to the related global variable.

2. Calibration

Calibration.cpp & Calibration.hpp:

Main files contains calibration operations. They contain only 1 function with Camera_Calibration name, which choose the type of input data (Image or Cam) and the path where to load images according to this choice. For webcam case, `Camera_Calibration(CAM, Main_Folder, x_corners,y_corners);` function call is used and Main_Folder actually has the default Calibration images path. So here, this argument is only given to be able to call the function with filling required parameters.

Calibrationmainwindow.ui & calibrationmainwindow.cpp & calibrationmainwindow.h

Main files contains Calibration GUI window class and member functions of this class having same type of implementation with Image Processing background.

3. Optical Flow

Optical_Flow.hpp & Optical_Flow.cpp

Main files contains optical flow operations. They contain only

```
bool    Lucas_Kanade(IO_TYPE IO_type,std::vector<cv::Mat>
&,std::vector<cv::Mat> &,std::vector<cv::Mat> &,std::string pat =
InputImage_Folder);
```

function which takes input data type as first parameter (Image Sequence, Video, CAM), a vector to hold optical flow arrows, a vector to hold optical flow colors, a vector to hold input images, a string to hold the path of input images. This path can refer to Video folder too. In case of using CAM, this function is called with default folder which won't be used and only put to call the function properly.

I implemented this function to give results both for arrows and colors representation but I removed the color representation since it didn't seem to me a better visualization as arrows. But I didn't change the function definition. This is the reason of having a vector for optical flow color representation which is not used.

ofmainwindow.ui & ofmainwindow.h & ofmainwindow.cpp

Main files contains Optical Flow GUI window class and member functions of this class having same type of implementation with Image Processing background.

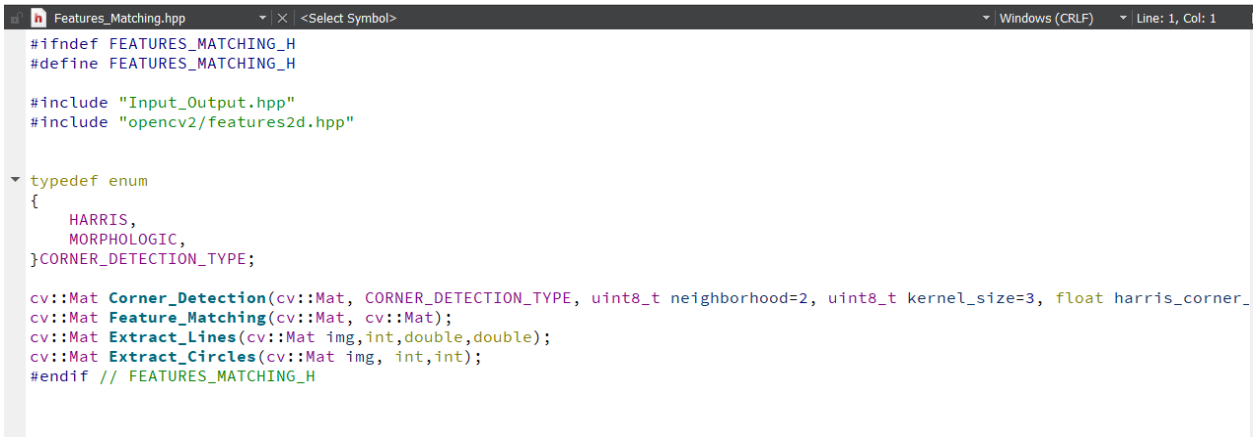
ofmainwindow.h | ofMainWindow

```
#ifndef OFMAINWINDOW_H
#define OFMAINWINDOW_H

#include <QMainWindow>

namespace Ui {
class ofMainWindow;
}

class ofMainWindow : public QMainWindow
{
    Q_OBJECT


public:
    explicit ofMainWindow(QWidget *parent = nullptr);
    ~ofMainWindow();

private slots:
    void on_pushButton_clicked(); // demo image sequence

    void on_pushButton_2_clicked(); // demo video

    void on_pushButton_5_clicked(); // CAM

    void on_pushButton_3_clicked(); // User Image Sequence

    void on_pushButton_4_clicked();  // User Video

    void on_pushButton_6_clicked(); // back button

private:
    Ui::ofMainWindow *ui;
};

#endif // OFMAINWINDOW_H
```

4. Feature Matching

Feature_Matching.cpp & Feature_Matching.hpp:

Main files contains feature matching operations.



fmmainwindow.ui & fmmainwindow.h & fmmainwindow.cpp

Main files contains Feature Matching GUI window class and member functions of this class having same type of implementation with Image Processing background.

Bugs

During creation of this project, I handled a lot of bugs or run time errors. As a matter of fact, there are still some bugs I detected but could not solve. Here is the list of these bugs:

- When saving CAM processed image in Image Processing part, it is saved as BGR for some reason. The CAM images were uploading as BGR on the GUI window and I fixed it. After converting this image, I don't know why the processed image is saved as BGR again.
- When applying Downsampling to the RGB images, the downsampled image is uploading to GUI as grayscale. There is not

any conversion implemented for this step and the problem doesn't occur for Upsampling. So, I couldn't find out the reason.

- When using Histogram Equalization for Image Sequence with "apply all" option, when we go forward and back between the images, back click cause the program shut down.

This project gave me an opportunity to work on a GUI with QT creator which was in my "to learn" list for a long time but I could not have been able to find time for that. I'm very appreciated to have a project that much useful and want to thank you!

Source code is available on
https://github.com/YCAyca/image_processing_of_calibration_GUI

Yağmur Çiğdem Aktaş