DeWAFF

Generated by Doxygen 1.9.1

1 Implementación y validación de un algoritmo de abstracción de imagen en C++

Repositorio dedicado a avances del proyecto elétctrico.
Estudiante: Isaac F. Fonseca Segura.
Profesor guía: Dr.rer.nat. Francisco Siles Canales.

1.1 Tabla de contenidos

1.2 Pasos seguidos

Esta sección sirve como referencia para la metodología del trabajo escrito.

- Configurar una VM con Ubuntu 22.04 LTS
- Leer el material de `Open CV introduction`
- Seguir la guía de instalación `OpenCV Instalation tutorial - Linux`
- Instalar VSCode y las extensiones necesarias para C++. Además de extensiones para soporte de CMake y Doxygen
- Investigar sobre el framework DeWAFF
- Comenzar implementación en C++
- Hacer un fork the `temporalSegmentation`
- Automatización de compilación con CMake
- Revisión extensiva y corrección de inconsistencias en el fork
- Pruebas con imágenes y revisión de resultados
- Pruebas con vídeo y revisión de resultados
- Documentación del código con Doxygen
- Elaboración del trabajo escrito

1.3 Implementación actual

En la implementación actual se cuenta no sólo con la clase DeWAFF, pero también con clases que asisten al procesado de la imagen como la clase NonAdaptiveUSM. Además se incluyen clases que se encargan del preprocesado de la imagen y la presentación del programa en la terminal. Finalmente hay clases con métodos de asistencia como Timer y Tools.

1.4 Clases

Las clases implementadas y/o adaptadas y sus métodos son las siguientes:

- DeWAFF
  - DeceivedBilateralFilter
  - NonAdaptiveUSMFilter
  - LaplacianKernel
  - GaussianKernel
  - GaussianExponentialFactor
- FileProcessor
  - processImage
  - processVideo
  - processFrame

    - errorExit
- CLI
    - run
    - help
- Timer
    - start
    - stop
- Tools
    - meshGrid
    - getMinMax

Estas clases y funciones se encuentran extensivamente documentadas en formato Doxygen. Para generar la documentación se deben seguir los siguientes pasos

1.5 Generación de la documentación

Se deben introducir los siguientes comandos

```
cd doxygen/
doxygen Doxyfile
```

Después se debe abrir el enlace símbolico en el directorio raíz `DeWAFF-B52786/` del proyecto llamado `index.html`. Esto abrirá en su navegador la documentacion del proyecto. Doxygen debe estar instalado previamente.

1.6 Compilación del proyecto

La compliación de este proyecto se realiza de manera automatizada con `cmake` y `make`. Para crear un ejecutable debe seguir los siguientes pasos en el directorio raíz `DeWAFF-B52786/`. Todas las dependencias de `OpenCV` deben estar instaladas previamente.

```
cmake .
cmake --build .
```

Si hace cambios al código podrá actualizar el proyecto con `make`.

1.7 Ejecución del framework

Para correr el programa se debe haber generado el binario `DeWAFF` en el directorio raíz al haber seguido los pasos de compilación. Una vez con este binario se pueden procesar imágenes y vídeo con la opción de hacer un benchmark.

Para correr el programa use el siguiente comando en el directorio raíz `DeWAFF-B52786/`, o donde desee ubicar el programa

1.7.1 Procesar imagen

```
./DeWAFF -i <ruta del archivo>
```

1.7.2 Procesar imagen y hacer benchmark

```
./DeWAFF -i <ruta del archivo> -b <número de iteraciones>
```

1.7.3 Procesar vídeo

```
./DeWAFF -v <ruta del archivo>
```

1.7.4 Procesar vídeo y hacer benchmark

```
./DeWAFF -v <ruta del archivo> -b <número de iteraciones>
```

El resultado se generará en la `ruta del archivo` escogido y se le agregará el sufijo DeWAFF de forma que el resultado se mostrará de la forma `<nombre_del_archivo>_DeWAFF.<ext>`. En el directorio raíz se encuentran un par de ejemplos en el directorio `img/`.

A los vídeos se les agrega la extensión `.avi` y las imágenes la extensión `.png` por defecto.

1.8 Idea inicial de implementación

La idea original era crear desde cero una implementación de DeWAFF. Más adelante se esperaba partir de una implementación del algoritmo en Matlab, lamentablemente esta se perdió, por lo que se comenzó a implementar el framework desde cero. A medio camino se dio con una implementación de código abierto. Se hizo un fork de esta y se comenzó a trabajar con esto como nueva base. A pesar de tener ciertas inconsistencias tenía las bases necesarias y una implementación del filtro bilateral funcional.

A continuación se muestran las ideas originales del proyecto, las cuales terminaron siendo consistentes con la implementación actual.

1.8.1 Clase DEWAFF

La clase DEWAFF debe contar con un constructor que inicialice los valores necesarios para iniciar el filtrado de una imagen.

1.8.2 Métodos

La clase DEWAFF debe contar con métodos que realicen los siguientes pasos:

- Un método que permita cargar una imágen desde una ruta local. Adquirir el tamaño de la imagen y guardar otros parámetros que sean relevantes. Y finalmente definir el formato de la imágen (color, densidad de pixeles).
- Un método que permita escoger el tipo de filtrado del framework utilizar (es posible pasar un string como parámetro). Este debe llamar a las funciones de filtrado necesarias y pasarles los parámetros que estas necesiten. Una vez terminado debe guardar la imágen con sufijo, ej: "imagen_\<BF>_\<DEWAFF>.png" si se utiliza el filtro bilateral.

1.8.3 Funciones

Se debe implementar una función por cada tipo de operación que el framework implemente. Por ejemplo, se deben implementar los filtros y las funciones que estos requieran.

1.8.4 Pruebas

Con imágenes de formato FHD o HD se debe probar cada configuración del framework para asegurar el correcto funcionamiento del mismo. Se pretende usar imágenes en estas resoluciones para evaluar los tiempos de funcionamiento del framework y verificar visualmente los resultados.

Como pruebas finales se planea procesar imágenes de gran tamaño obtenidas de microoscopios u otros dispositivos.

2 Hierarchical Index

2.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

3 Class Index

3.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

4 File Index

4.1 File List

Here is a list of all documented files with brief descriptions:

5 Class Documentation

5.1 DeWAFF Class Reference

Deceived Weighted Average Filters Framework class It applies a filter which intput and weighting input are decoupled, so it is possible deceive the input and to still use the original input weighting values.

```
#include <DeWAFF.hpp>
```

Inheritance diagram for DeWAFF:

Collaboration diagram for DeWAFF:

Public Types

- enum **FilterType** : unsigned int { **DBF** , **DSBF** , **DNLM** , **DGF** }

Public Member Functions

- DeWAFF ()

  Parameter for the Laplacian deceive.
- Mat DeceivedBilateralFilter (const Mat &inputImage, int windowSize, double spatialSigma, int rangeSigma)

  Apply the Deceived Bilateral Filter to an image. Uses an UnSharp mask as deceiver.
- Mat DeceivedScaledBilateralFilter (const Mat &inputImage, int windowSize, double spatialSigma, int range←Sigma)

  Apply the Deceived Scaled Bilateral Filter to an image. Uses an UnSharp mask as deceiver. Similar to the Deceived Bilateral Filter, but the weighting image is low pass filtered.
- Mat DeceivedNonLocalMeansFilter (const Mat &inputImage, int windowSize, int patchSize, double spatial←Sigma, int rangeSigma)

  Apply the Deceived Non Local Means Filter to an image. Uses an UnSharp mask as deceiver. Computationally demanding algorithm, can take as much as ten times more than the other filters in the framework.
- Mat DeceivedGuidedFilter (const Mat &inputImage, int windowSize, double spatialSigma, int rangeSigma)

  Apply the Deceived Guided Filter to an image. Uses an UnSharp mask as deceiver. The fastest WAF for the DeWAFF yet.

Additional Inherited Members

5.1.1 Detailed Description

Deceived Weighted Average Filters Framework class It applies a filter which intput and weighting input are decoupled, so it is possible deceive the input and to still use the original input weighting values.

5.1.2 Constructor & Destructor Documentation

5.1.2.1 DeWAFF() `DeWAFF::DeWAFF ( )`

Parameter for the Laplacian deceive.

DeWAFF class constructor. Sets the lambda parameter for the Laplacian deceive.

5.1.3 Member Function Documentation

5.1.3.1 DeceivedBilateralFilter()

```
Mat DeWAFF::DeceivedBilateralFilter (
        const Mat & inputImage,
        int windowSize,
        double spatialSigma,
        int rangeSigma )
```

Apply the Deceived Bilateral Filter to an image. Uses an UnSharp mask as deceiver.

Parameters

inputImage	input image
windowSize	processing window size, has to be odd numbered and greater or equal than 3
spatialSigma	spatial standard deviation
rangeSigma	range or radiometric standard deviation

Returns

Mat output image

5.1.3.2 DeceivedGuidedFilter()

```
Mat DeWAFF::DeceivedGuidedFilter (
    const Mat & inputImage,
    int windowSize,
    double spatialSigma,
    int rangeSigma )
```

Apply the Deceived Guided Filter to an image. Uses an UnSharp mask as deceiver. The fastest WAF for the DeWAFF yet.

Parameters

inputImage	input image
windowSize	processing window size, has to be odd numbered and greater or equal than 3
spatialSigma	spatial standard deviation
rangeSigma	range or radiometric standard deviation

Returns

Mat output image

5.1.3.3 DeceivedNonLocalMeansFilter()

```
Mat DeWAFF::DeceivedNonLocalMeansFilter (
    const Mat & inputImage,
    int windowSize,
    int patchSize,
    double spatialSigma,
    int rangeSigma )
```

Apply the Deceived Non Local Means Filter to an image. Uses an UnSharp mask as deceiver. Computationally demanding algorithm, can take as much as ten times more than the other filters in the framework.

Parameters

inputImage	input image
windowSize	processing window size, has to be odd numbered and greater or equal than 3
patchSize	spatial standard deviation
spatialSigma	range or radiometric standard deviation
rangeSigma	output image

Returns

Mat

5.1.3.4 DeceivedScaledBilateralFilter()

```
Mat DeWAFF::DeceivedScaledBilateralFilter (
    const Mat & inputImage,
    int windowSize,
    double spatialSigma,
    int rangeSigma )
```

Apply the Deceived Scaled Bilateral Filter to an image. Uses an UnSharp mask as deceiver. Similar to the Deceived Bilateral Filter, but the weighting image is low pass filtered.

Parameters

inputImage	input image
windowSize	processing window size, has to be odd numbered and greater or equal than 3
spatialSima	spatial standard deviation
rangeSigma	range or radiometric standard deviation

Returns

Mat output image

The documentation for this class was generated from the following files:

- /home/isaac/Desktop/DeWAFF/include/DeWAFF.hpp
- /home/isaac/Desktop/DeWAFF/src/DeWAFF.cpp

5.2 Filters Class Reference

Class containing Weighted Average Filters (WAFs). This implementation relies on padding the original image to fit square odd dimensioned kernels throughout the processing.

```
#include <Filters.hpp>
```

Inheritance diagram for Filters:

Collaboration diagram for Filters:

Public Member Functions

- Mat BilateralFilter (const Mat &weightingImage, const Mat &inputImage, const int windowSize, const double spatialSigma, const int rangeSigma)

  Apply a Bilateral Filter to an image. This is the decoupled version of this filter, this means that the weighting image for the filter can be different from the input image.
- Mat ScaledBilateralFilter (const Mat &weightingImage, const Mat &inputImage, const int windowSize, const double spatialSigma, const int rangeSigma)

  Apply a Scaled Bilateral Filter to an image. This is the decoupled version of this filter, this means that the weighting image for the filter can be different from the input image. The difference between this filter and the not scaled version is that the weighting image is pre scaled through a Gaussian low pass filter to have better performance under heavy AWGN.
- Mat NonLocalMeansFilter (const Mat &weightingImage, const Mat &inputImage, const int windowSize, const int patchSize, const double rangeSigma)

  Apply a Non Local Means Filter to an image. This is the decoupled version of this filter, this means that the weighting image for the filter can be different from the input image. The algorithm used for this filter is very computationally demanding.
- Mat **GuidedFilter** (const Mat &inputImage, const Mat &guidingImage, const int windowSize, const int range↵ Sigma)

Protected Attributes

- Utils **lib**

5.2.1 Detailed Description

Class containing Weighted Average Filters (WAFs). This implementation relies on padding the original image to fit square odd dimensioned kernels throughout the processing.

5.2.2 Member Function Documentation

5.2.2.1 BilateralFilter()

```
Mat Filters::BilateralFilter (
        const Mat & weightingImage,
        const Mat & inputImage,
        const int windowSize,
        const double spatialSigma,
        const int rangeSigma )
```

Apply a Bilateral Filter to an image. This is the decoupled version of this filter, this means that the weighting image for the filter can be different from the input image.

Parameters

weightingImage	image used to calculate the kernel's weight
inputImage	image used as input for the filter
windowSize	processing window size, has to be odd numbered and greater or equal than 3
spatialSigma	spatial standard deviation
rangeSigma	range or radiometric standard deviation

Returns

Mat output image

This filter uses two Gaussian kernels, one of them is the spatial Gaussian kernel $G_{spatial}(U,m,p) = \exp\left(-\frac{||m-p||^2}{2\sigma_s^2}\right)$ with the spatial values from an image region $\Omega \subseteq U$. The spatial kernel uses the $m_i \subseteq \Omega$ pixels coordinates as weighting values for the pixel $p = (x,y)$

The other Gaussian kernel is the range Gaussian kernel $G_{range}(U,m,p) = \exp\left(-\frac{||U(m)-U(p)||^2}{2\sigma_s^2}\right)$ with the intensity (range) values from an image region $\Omega \subseteq U$. The range kernel uses the $m_i \subseteq \Omega$ pixels intensities as weighting values for the pixel $p = (x,y)$ instead of their locations as in the spatial kernel computation. In this case a the input U is separated into the three CIELab weightChannels and each channel is processed as an individual input $U_{channel}$

The two kernels convolve to obtain the Bilateral Filter kernel $\phi_{SBF}(U,m,p) = G_{spatial}(||m-p||)\, G_{range}(||U(m)-U(p)||)$

The Bilateral filter's norm corresponds to $\left(\sum_{m \subseteq \Omega} \phi_{\text{SBF}}(U,m,p)\right)^{-1}$

Finally the bilateral filter kernel can be convolved with the input as follows $Y_{\phi_{SBF}}(p) = \left(\sum_{m \subseteq \Omega} \phi_{SBF}(U,m,p)\right)^{-1} \left(\sum_{m \subseteq \Omega} \phi_{SBF}(U,$

5.2.2.2 NonLocalMeansFilter()

```
Mat Filters::NonLocalMeansFilter (
        const Mat & weightingImage,
        const Mat & inputImage,
        const int windowSize,
        const int patchSize,
        const double rangeSigma )
```

Apply a Non Local Means Filter to an image. This is the decoupled version of this filter, this means that the weighting image for the filter can be different from the input image. The algorithm used for this filter is very computationally demanding.

Parameters

weightingImage	image used to calculate the kernel's weight
inputImage	image used as input for the filter
windowSize	processing window size, has to be odd numbered and greater or equal than 3
rangeSigma	range or radiometric standard deviation. Used to calculate the parameter $h = \frac{\sqrt{(\sigma)}}{2}$

Returns

Mat output image

The discrete representation of the Non Local Means Filter is as follows, $\phi_{\text{NLM}}(U,m,p) = \sum_{B(m) \subseteq U} \exp\left(\frac{||B(m)-B(p)||^2}{h^2}\right)$ where $B(p)$ is a patch part of the window Ω centered at pixel p. $B(m)$ represents all of the patches at Ω centered in each m pixel. The Non Local Means Filter calculates the Euclidean distance between each patch $B(m)$ and $B(p)$ for each window $\Omega \subseteq U$. This is why this algorithm is highly demanding in computational terms. Each Euclidean distance matrix obtained from each patch pair is the input for a Gaussian decreasing function with standard deviation h that generates the new pixel p value

The Non Local Means filter's norm is calculated with $\phi_{NLM}(U,m,p)\left(\sum_{m\subseteq\Omega}\phi_{\text{NLM}}(U,m,p)\right)^{-1}$

The NLM filter kernel is applied to the laplacian image $Y_{\phi_{\text{NLM}}}(p)=\left(\sum_{m\subseteq\Omega}\phi_{NLM}(U,m,p)\right)^{-1}\left(\sum_{m\subseteq\Omega}\phi_{NLM}(U,p,m)\,\hat{f}_{\text{USM}}(m)\right.$

5.2.2.3 ScaledBilateralFilter()

```
Mat Filters::ScaledBilateralFilter (
        const Mat & weightingImage,
        const Mat & inputImage,
        const int windowSize,
        const double spatialSigma,
        const int rangeSigma )
```

Apply a Scaled Bilateral Filter to an image. This is the decoupled version of this filter, this means that the weighting image for the filter can be different from the input image. The difference between this filter and the not scaled version is that the weighting image is pre scaled through a Gaussian low pass filter to have better performance under heavy AWGN.

Parameters

weightingImage	image used to calculate the kernel's weight
inputImage	image used as input for the filter
windowSize	processing window size, has to be odd numbered and greater or equal than 3
spatialSigma	spatial standard deviation
rangeSigma	range or radiometric standard deviation

Returns

Mat output image

This filter uses two Gaussian kernels, one of them is the spatial Gaussian kernel $G_{spatial}(U,m,p) = \exp\left(-\frac{||m-p||^2}{2\sigma_s^2}\right)$ with the spatial values from an image region $\Omega\subseteq U$. The spatial kernel uses the $m_i\subseteq\Omega$ pixels coordinates as weighting values for the pixel $p=(x,y)$

The other Gaussian kernel is the range Gaussian kernel $G_{range}(U,U^s,m,p)=\exp\left(-\frac{||U^s(m)-U(p)||^2}{2\sigma_s^2}\right)$ with the intensity (range) values from an image region $\Omega\subseteq U$. The range kernel uses the $m_i\subseteq\Omega$ pixels intensities as weighting values for the pixel $p=(x,y)$ instead of their locations as in the spatial kernel computation. In this case a the image U is separated into the three CIELab scaledChannels and each channel is processed as an individual image $U_{channel}$

The two kernels convolve to obtain the Scaled Bilateral Filter kernel $\phi_{SBF}(U,U^s,m,p) = G_{spatial}(||m-p||)\,G_{range}(||U^s(m)-U(p)||)$

The Scaled Bilateral filter's norm corresponds to $\left(\sum_{m\subseteq\Omega}\phi_{\text{SBF}}(U,U^s,m,p)\right)^{-1}$

Finally the bilateral filter kernel can be convolved with the input as follows $Y_{\phi_{SBF}}(p)=\left(\sum_{m\subseteq\Omega}\phi_{SBF}(U,U^s,m,p)\right)^{-1}\left(\sum_{m\subseteq\Omega}\phi_{SBF}\right.$

The documentation for this class was generated from the following files:

- /home/isaac/Desktop/DeWAFF/include/Filters.hpp
- /home/isaac/Desktop/DeWAFF/src/Filters.cpp

5.3 GuidedFilter Class Reference

An open source OpenCV guided filter implementation under the MIT license.

```
#include <GuidedFilter.hpp>
```

Public Member Functions

- **GuidedFilter** (const cv::Mat &I, int r, double eps)
- cv::Mat **filter** (const cv::Mat &p, int depth=-1) const

5.3.1 Detailed Description

An open source OpenCV guided filter implementation under the MIT license.

The documentation for this class was generated from the following files:

- /home/isaac/Desktop/DeWAFF/include/GuidedFilter.hpp
- /home/isaac/Desktop/DeWAFF/src/GuidedFilter.cpp

5.4 GuidedFilterColor Class Reference

Inheritance diagram for GuidedFilterColor:

Collaboration diagram for GuidedFilterColor:

Public Member Functions

- **GuidedFilterColor** (const cv::Mat &I, int r, double eps)

Additional Inherited Members

The documentation for this class was generated from the following file:

- /home/isaac/Desktop/DeWAFF/src/GuidedFilter.cpp

5.5 GuidedFilterImpl Class Reference

Inheritance diagram for GuidedFilterImpl:

Public Member Functions

- cv::Mat **filter** (const cv::Mat &p, int depth)

Protected Attributes

- int **Idepth**

The documentation for this class was generated from the following file:

- /home/isaac/Desktop/DeWAFF/src/GuidedFilter.cpp

5.6 GuidedFilterMono Class Reference

Inheritance diagram for GuidedFilterMono:

Collaboration diagram for GuidedFilterMono:

Public Member Functions

- **GuidedFilterMono** (const cv::Mat &I, int r, double eps)

Additional Inherited Members

The documentation for this class was generated from the following file:

- /home/isaac/Desktop/DeWAFF/src/GuidedFilter.cpp

5.7 ProgramInterface Class Reference

In charge of displaying the program and capturing the needed parameters.

```
#include <ProgramInterface.hpp>
```

Public Member Functions

- ProgramInterface (int argc, char **argv)

  Constructor for the ProgramInterface class. Sets all the necessary parameters for the DeWAFF processing, including the ones captured from the user input in the terminal.
- int run ()

  Starts the program execution.

5.7.1 Detailed Description

In charge of displaying the program and capturing the needed parameters.

5.7.2 Constructor & Destructor Documentation

5.7.2.1 ProgramInterface()

```
ProgramInterface::ProgramInterface (
        int argc,
        char ** argv )
```

Constructor for the ProgramInterface class. Sets all the necessary parameters for the DeWAFF processing, including the ones captured from the user input in the terminal.

Parameters

argc	argument count from the terminal
argv	arguments from the terminal

The documentation for this class was generated from the following files:

- /home/isaac/Desktop/DeWAFF/include/ProgramInterface.hpp
- /home/isaac/Desktop/DeWAFF/src/ProgramInterface.cpp

5.8 Timer Class Reference

Class containing the timer methods for the benchmarking of file processing.

```
#include <Utils.hpp>
```

Inheritance diagram for Timer:

Collaboration diagram for Timer:

Public Member Functions

- void start ()
  Starts the timer and resets the elapsed time.
- double stop ()
  Stops the timer and returns the elapsed time.

5.8.1 Detailed Description

Class containing the timer methods for the benchmarking of file processing.

5.8.2 Member Function Documentation

5.8.2.1 stop() `double Timer::stop ( )`

Stops the timer and returns the elapsed time.

Returns

Elapsed time in seconds

The documentation for this class was generated from the following files:

- /home/isaac/Desktop/DeWAFF/include/Utils.hpp
- /home/isaac/Desktop/DeWAFF/src/Utils.cpp

5.9 Utils Class Reference

Useful tools for image processing These tools are statics objects to use them in the lifetime of the program without the need of constinuous instantiation.

```
#include <Utils.hpp>
```

Inheritance diagram for Utils:

Public Member Functions

- void MeshGrid (const Range &range, Mat &X, Mat &Y)

  Generates a meshgrid from X and Y unidimensional coordinates Example: xRange = [0,3[and yRange = [0,3[will return the following X and Y coordinates.
- void MinMax (const Mat &A, double ∗minA, double ∗maxA)

  Gets the global min and max values of a 3 channel Matrix.
- Mat **GaussianFunction** (Mat input, double sigma)
- Mat GaussianKernel (int windowSize, double sigma)

  Computes a spatial Gaussian kernel $G(X,Y) = \exp\left(-\frac{|X+Y|^2}{2\sigma_s^2}\right)$ where X + Y are the horizontal and vertical coordinates on a windowSize $\times$ windowSize 2D plane. The result can be interpreted as looking at a Gaussian distribution from a top view.
- Mat LoGKernel (int windowSize, double sigma)

  Computes a Laplacian of Gaussian kernel. Same as fspecial('log',..) in Matlab. LoG$_{kernel} = -\frac{1}{\pi\sigma^4}\left[1 - \frac{X^2+Y^2}{2\sigma^2}\right]\exp\left(-\frac{X^2+Y^2}{2\sigma^2}\right)$ and normalize it with $\frac{\sum LoG}{|LoG|}$ where $|LoG|$ is the number of elements in LoG so it sums to 0 for high pass filter behavior consistency.
- Mat NonAdaptiveUSM (const Mat &image, int windowSize, int lambda, double sigma)

  Applies a regular non adaptive UnSharp mask (USM) with a Laplacian of Gaussian kernel $\hat{f}_{USM} = U + \lambda\mathcal{L}$ where $\mathcal{L} = l * g$. Here g is a Gaussian kernel and l a Laplacian kernel, hence the name "Laplacian of Gaussian".
- Mat EuclideanDistanceMatrix (const Mat &image, int patchSize)

  Computes an Euclidean distance matrix from an input matrix. This is achieved by calculating the Euclidean distance between a fixed patch at the center of the input image and a patch centered in every other pixel in the input image, mathematically, for an input matrix $A = (a_{ij})$ every element will take the corresponding Euclidean distance value $a_{ij} = d_{ij}^2 = ||x_i - x_j||^2$.

5.9.1 Detailed Description

Useful tools for image processing These tools are statics objects to use them in the lifetime of the program without the need of constinuous instantiation.

5.9.2 Member Function Documentation

5.9.2.1 EuclideanDistanceMatrix()

```
Mat Utils::EuclideanDistanceMatrix (
        const Mat & inputImage,
        int patchSize )
```

Computes an Euclidean distance matrix from an input matrix. This is achieved by calculating the Euclidean distance between a fixed patch at the center of the input image and a patch centered in every other pixel in the input image, mathematically, for an input matrix $A = (a_{ij})$ every element will take the corresponding Euclidean distance value $a_{ij} = d_{ij}^2 = ||x_i - x_j||^2$.

Parameters

inputImage	input image to obtain the patches to compute the matrix
patchSize	size of the used patch. Analog to window size

Returns

Mat Euclidean distance matrix

5.9.2.2 GaussianKernel()

```
Mat Utils::GaussianKernel (
        int windowSize,
        double sigma )
```

Computes a spatial Gaussian kernel $G(X,Y) = \exp\left(-\frac{|X+Y|^2}{2\sigma_s^2}\right)$ where X + Y are the horizontal and vertical coordinates on a windowSize $\times$ windowSize 2D plane. The result can be interpreted as looking at a Gaussian distribution from a top view.

Parameters

windowSize	2D plane dimension
sigma	standar deviation for the Gaussian distribution

Returns

Mat A Gaussian kernel

Pre computates the spatial Gaussian kernel Calculate the kernel variable $S = X^2 + Y^2$

5.9.2.3 LoGKernel()

```
Mat Utils::LoGKernel (
        int windowSize,
        double sigma )
```

Computes a Laplacian of Gaussian kernel. Same as fspecial('log',..) in Matlab. $\text{LoG}_{kernel} = -\frac{1}{\pi\sigma^4}\left[1 - \frac{X^2+Y^2}{2\sigma^2}\right]\exp\left(-\frac{X^2+Y^2}{2\sigma^2}\right)$ and normalize it with $\frac{\sum \text{LoG}}{|\text{LoG}|}$ where $|\text{LoG}|$ is the number of elements in LoG so it sums to 0 for high pass filter behavior consistency.

Pre computates the spatial Gaussian kernel Calculate the kernel variable $S = X^2 + Y^2$

5.9.2.4 MeshGrid()

```
void Utils::MeshGrid (
        const Range & range,
        Mat & X,
        Mat & Y )
```

Generates a meshgrid from X and Y unidimensional coordinates Example: xRange = [0,3[and yRange = [0,3[will return the following X and Y coordinates.

X =
[0, 0, 0;

1, 1, 1;
2, 2, 2]
Y =
[0, 1, 2;
0, 1, 2;
0, 1, 2]

Wich would form the following mesh grid

(X, Y) =
[(0,0) (0,1), (0,2);
(0,1) (1,1), (2,1);
(2,0) (2,1), (2,2)]

Parameters

range	range to form the 2D grid
X	x axis values for the mesh grid
Y	y axis values for the mesh grid

5.9.2.5 MinMax()

```
void Utils::MinMax (
        const Mat & A,
        double * minA,
        double * maxA )
```

Gets the global min and max values of a 3 channel Matrix.

Parameters

A	Input matrix
minA	Minimun value of the matrix A
maxA	Maximun value of the matrix A

5.9.2.6 NonAdaptiveUSM()

```
Mat Utils::NonAdaptiveUSM (
        const Mat & image,
        int windowSize,
        int lambda,
        double sigma )
```

Applies a regular non adaptive UnSharp mask (USM) with a Laplacian of Gaussian kernel $\hat{f}_{USM} = U + \lambda\mathcal{L}$ where $\mathcal{L} = l * g$. Here g is a Gaussian kernel and l a Laplacian kernel, hence the name "Laplacian of Gaussian".

Parameters

image	Image to apply the mask

Returns

Filtered image

The documentation for this class was generated from the following files:

- /home/isaac/Desktop/DeWAFF/include/Utils.hpp
- /home/isaac/Desktop/DeWAFF/src/Utils.cpp

6 File Documentation

6.1 /home/isaac/Desktop/DeWAFF/include/DeWAFF.hpp File Reference

```
#include "opencv2/core/core.hpp"
#include "opencv2/imgproc/imgproc.hpp"
#include "opencv2/highgui/highgui.hpp"
#include "Utils.hpp"
#include "Filters.hpp"
```

Include dependency graph for DeWAFF.hpp:

6.2 /home/isaac/Desktop/DeWAFF/include/Filters.hpp File Reference

```
#include <omp.h>
#include <opencv2/opencv.hpp>
#include "opencv2/core/core.hpp"
#include "opencv2/imgproc/imgproc.hpp"
#include "opencv2/highgui/highgui.hpp"
#include "GuidedFilter.hpp"
#include "Utils.hpp"
```

Include dependency graph for Filters.hpp: This graph shows which files directly or indirectly include this file:

Classes

- class Filters

  Class containing Weighted Average Filters (WAFs). This implementation relies on padding the original image to fit square odd dimensioned kernels throughout the processing.

6.2.1 Detailed Description

Author

Isaac Fonseca (isaac.fonsecasegura@ucr.ac.cr)

Date

2022-11-06

Author

Manuel Zumbado

David Prado (davidp)

Juan Jose Guerrero

Date

2015-08-29

6.3 /home/isaac/Desktop/DeWAFF/include/GuidedFilter.hpp File Reference

Guided filter implementation from `https://github.com/atilimcetin/guided-filter`.

```
#include <opencv2/opencv.hpp>
```
Include dependency graph for GuidedFilter.hpp: This graph shows which files directly or indirectly include this file:

Classes

- class GuidedFilter

  An open source OpenCV guided filter implementation under the MIT license.

Functions

- cv::Mat **guidedFilter** (const cv::Mat &I, const cv::Mat &p, int r, double eps, int depth=-1)

6.3.1 Detailed Description

Guided filter implementation from `https://github.com/atilimcetin/guided-filter`.

Author

Atılım Çetin

Nikolai Poliarnyi

Date

2020-06-1

2022-11-17

6.4 /home/isaac/Desktop/DeWAFF/include/ProgramInterface.hpp File Reference

```
#include <string>
#include <cstdio>
#include <iomanip>
#include <unistd.h>
#include <iostream>
#include "Utils.hpp"
#include "DeWAFF.hpp"
```
Include dependency graph for ProgramInterface.hpp:

Classes

- class ProgramInterface

  In charge of displaying the program and capturing the needed parameters.

6.4.1 Detailed Description

Author

Isaac Fonseca (isaac.fonsecasegura@ucr.ac.cr)

Date

2022-11-06

Author

David Prado (davidp)

Date

2015-11-05

6.5 /home/isaac/Desktop/DeWAFF/include/Utils.hpp File Reference

```
#include <iostream>
#include <algorithm>
#include <sys/time.h>
#include "opencv2/core/core.hpp"
#include "opencv2/imgproc/imgproc.hpp"
#include "opencv2/highgui/highgui.hpp"
```

Include dependency graph for Utils.hpp: This graph shows which files directly or indirectly include this file:

Classes

- class Utils

  Useful tools for image processing These tools are statics objects to use them in the lifetime of the program without the need of consinuous instantiation.
- class Timer

  Class containing the timer methods for the benchmarking of file processing.

6.5.1 Detailed Description

Author

Isaac Fonseca (isaac.fonsecasegura@ucr.ac.cr)

Date

2022-11-06

Author

David Prado (davidp)

Date

2015-11-05

Index