PROYECTO DE GRADO

Presentado ante la ilustre UNIVERSIDAD DE LOS ANDES como requisito final para
obtener el Título de INGENIERO DE SISTEMAS

Image Processing (IP) Cemisid 1.0: Biblioteca de Funciones para Procesamiento de Imágenes basada en Armadillo C++ y OpenCV C++

Por

Tec. M. Inf. Jormar S. Turizo A.

Tutor: Prof. Juan M. Ramírez

Junio 2017

INGENIERIA
UNIVERSIDAD DE LOS ANDES
MERIDA - VENEZUELA

Escuela de Ingeniería de Sistemas

PROYECTO DE GRADO
CALIFICACIÓN FINAL

Título del Proyecto de Grado: **"Image Processing (IP) Cemisid 1.0: Biblioteca de Funciones para Procesamiento de Imágenes basada en Armadillo C++ y Open CV C++"**

Bachiller: **JORMAR SIKIU TURIZO ABREU** C.I. **19.086.414**

1) Calificación del (de la) Profesor(a) Tutor(a):	20	(20%)
2) Calificación del manuscrito final:	20	(40%)
3) Calificación de la defensa oral:	20	(40%)
Calificación final:	**20**	(puntos)

Los suscritos miembros del Jurado asignan como calificación final del Proyecto de Grado la nota de:

20 (puntos) **Veinte** (puntos)

Números Letras

Observaciones: Los Miembros del Jurado calificador sugerimos que el presente trabajo sea accesible a la comunidad Universitaria en general, a través de la publicación de un artículo científico, donde se destaquen los principales resultados de este trabajo. En consecuencia le otorgamos **MENCIÓN PUBLICACIÓN.**

Prof. Juan Marcos Ramírez
Tutor

Prof. Alejandro Mujica
Jurado

Prof. Domingo Hernández
Jurado

Mérida, 08 de Junio de 2017.

Image Processing (IP) Cemisid 1.0: Biblioteca de Funciones para Procesamiento de Imágenes basada en Armadillo C++ y OpenCV C++.

Tec. M. Inf. Jormar Sikiu Turizo Abreu

Proyecto de Grado - Sistemas Computacionales - 66 páginas

Resumen: A través de los años distintos mecanismos de captación de imágenes digitales han generado contaminación de ruido impulsivo (pérdida de información o píxeles) en las imágenes, y progresivamente se ha demostrado que estos ruidos pueden ser clasificados. Por ende, al ello generarse se hacen necesarios métodos efectivos que logren removerlos.

Este proyecto se centra en la creación de una Biblioteca *Image Processing* Cemisid 1.0 que simula mediante funciones, los métodos para contaminar las imágenes con ruidos impulsivos (Gaussiano, Sal y Pimienta, Impulsivo Uniforme y Pérdida de Píxeles), incluyendo, métodos de filtrado y aproximación por bloques basados en cálculos de media y mediana, los cuales permiten recuperar los sectores perdidos de las imágenes que han sido afectadas. Estos métodos son implementados mediante las bibliotecas Armadillo y OpenCV de C++ en el manejo de cálculos complejos de algebra lineal y la visualización de imágenes mediante ventanas.

Estas herramientas permitieron crear una biblioteca que mejora los tiempos de cómputo, el manejo de memoria, con documentación en Doxygen e incluye un método de detección facial de OpenCV C++ que daría un aporte hacia otras áreas de estudio. La biblioteca fue implementada bajo la metodología SOFTENG Agile orientada en fases para una ejecución efectiva del proyecto permitiendo lograr los objetivos propuestos.

En conclusión, se demostró con respecto a la remoción de ruido que los métodos de filtrado de imágenes por mediana propuestos dan mejores resultados que los filtrados por media y en la reconstrucción la aproximación robusta por bloques solapados provee una mejor calidad que la aproximación de bloques solapados. Las valoraciones de métricas de calidad de imagen del PSNR, MAE y MSE demostraron que tan efectivos eran los filtrados y las aproximaciones por bloques.

Palabras claves: Biblioteca de funciones, Armadillo C++, OpenCV C++, Doxygen, Procesamiento de Imágenes, Eclipse.

Dedicatoria

A Dios y a la Virgen

Creadores de conciencia infinita de belleza mágica espiritual, que nos enseña que el poder creativo del universo está dentro de cada uno, con su luz y energía fluyendo cada día sin detenerse.

A mi madre Maritza

Por su eterno amor al ser el pilar fundamental de mi vida, ofreciéndome educación, valores y aptitud ante la vida. Apoyándome en todo momento, mediante ejemplo, consejos, esfuerzo, esperanza y motivación constante, además de compartir excelentes momentos de discusión y aprendizajes durante el desarrollo de la tesis, poniendo su sello personal que no olvidaré y reconoceré toda mi vida, por lo manifestado "Bendita mi madre por siempre".

A mi Padre Jorge

Quien desde la distancia y por condiciones de vida, me ha demostrado que en ocasiones existe el cariño.

A mi Familia

A mis tíos y tías por ser ejemplo de tenacidad para enfrentar cada escalón de su vida, al estar pendiente de mí y tomarme en cuenta en cada situación, a mis primos por compartir buenos y malos momentos. En especial a mis abuelos **Aura** y **Américo**, los cuales a pesar de haberse despedido de este plano terrenal, yo estoy segura que siempre están cuidándome y guiándome desde el cielo.

A mis Amigos

Que con su apoyo incondicional me han ayudado con su bella energía, mediante un consejo o palabra de aliento para continuar, por su lealtad y por abrirme las puertas de sus corazones.

Índice

Índice de figuras

Agradecimientos

Primeramente, agradezco a Dios, ser maravilloso que me diera fuerza y fe diariamente para culminar esta meta como parte de mi proyecto de vida. A mi familia por su apoyo incondicional y por estar a mi lado en cada momento.

Agradezco a los profesores de la Universidad de los Andes, quienes me aportaron conocimientos, destrezas y habilidades en mi formación como profesional a lo largo de la carrera de Ingeniería de Sistemas, exigiéndome excelencia y calidad en cada paso.

En especial agradezco sinceramente a mi tutor, Ing. Juan Marcos Ramírez por su esfuerzo y dedicación, aportándome conocimientos, orientaciones, persistencia y motivación durante todo el desarrollo de este proyecto de grado y servir de guía satisfactoriamente en esta área de estudio, hasta lograr culminar esta etapa importante de mi carrera.

Agradezco además al Ing. Gerard Páez, por el apoyo incondicional y confianza que en conjunto con el Ing. Juan Marcos Ramírez, tuvieron en proponerme el desafío de desarrollar este proyecto de grado, lo cual agradezco enormemente.

Mi agradecimiento también va dirigido al Centro de Microcomputación y Sistemas Distribuidos (CEMISID), por ser una unidad de investigación que apoya la innovación en la Facultad de Ingeniería y los espacios para el desarrollo de sistemas con altos estándares de calidad.

Y para finalizar, también agradezco a todos los que fueron mis compañeros de clase durante todos los semestres de la Universidad, ya que gracias al compañerismo, la amistad y el apoyo moral logre superar cada nuevo reto y seguir adelante durante nuestra formación académica.

Capítulo 1.

Introducción

El Procesamiento de Imágenes y sus aplicaciones.

Uno de los campos de la modelación matemática en los que actualmente se está publicando y haciendo investigación es el procesamiento digital de imágenes.

Dentro de este campo, un área que ha incrementado el interés es la restauración de imágenes, debido a la posibilidad de abordar el problema desde diferentes conceptos, y metodologías nuevas.

Las imágenes se adquieren por medio de métodos fotoelectrónicos o fotoquímicos, estos dispositivos y sensores pueden degradar la calidad de las imágenes al introducir ruido impulsivo, deformaciones geométricas o borrosas debido al movimiento o desenfoque de la cámara.

La restauración de imágenes es significativa desde el punto de vista tanto matemático como comercial. En las industrias del cine y la publicidad es frecuente la necesidad de restaurar imágenes que por el paso del tiempo o falta de cuidado, se han deteriorado. Dichas imágenes pueden tener un valor histórico. Algo similar pasa en el estudio del arte, en el cual es frecuente la necesidad de restaurar determinadas obras. En este sentido existe una discusión abierta entre quienes consideran que una obra de valor artístico se debe restaurar y quienes dicen que solo se debe conservar.

Como su nombre lo indica, la restauración es el proceso de reconstruir partes perdidas o deterioradas de imágenes. Una de las preocupaciones planteadas para procesamiento digital de imágenes es aumentar la calidad y moderar la degradación introducida por los sensores y dispositivos de adquisición. Las técnicas de restauración de imágenes están interesadas en recobrar una imagen que ha sido degradada. Para el caso de las imágenes de tiempo atrás desde el punto de vista técnico, la restauración puede ser realizada en forma intuitiva por un artista experto. Sin embargo, en el mundo de las imágenes digitales este concepto ha cambiado, pues se han generado algoritmos matemáticos de cierta complejidad que permiten hacer lo mismo mediante un software.

En general, el procesamiento y visualización de imágenes es realizado usando herramientas de software que contienen una amplia gama de funciones que permiten el desarrollo y la implementación de algoritmos complejos. Los métodos que han sido elaborados en esta área de estudio tienen

limitaciones en la velocidad de compilación y ejecución, además, un alto consumo en el manejo de memoria, siendo evidente esto en sistemas basados en algoritmos implementados en Matlab[1].

Actualmente, está a disposición Armadillo C++ una biblioteca que provee funciones de algebra lineal basada en C++ y posee una sintaxis similar a Matlab. Los algoritmos de procesamiento de imágenes son adaptables a esta herramienta, no obstante esta biblioteca no incluyen ventanas de interfaz, por lo tanto el uso de la biblioteca OpenCV puede resolver el manejo de la visualización de imágenes.

El propósito de esta investigación sugiere la creación de una biblioteca de procesamiento de imágenes basada en estas aplicaciones: Armadillo C++ y OpenCV C++ llamada *Image Processing* Cemisid 1.0, diseñada como una herramienta innovadora que superará las limitaciones que todavía conservan los sistemas actuales.

[1] MATLAB (siglas del inglés MATrix LABoratory, "laboratorio de matrices") es una herramienta de software matemático que posee un entorno de desarrollo integrado.

1.1. Justificación.

El procesamiento de imágenes consiste en manipular una imagen digital mediante un arreglo matricial de puntos o píxeles, donde las técnicas generalmente están basadas en el desplazamiento de una ventana que captura una región de la imagen de entrada. Un área importante en el procesamiento de imágenes son los métodos de restauración, los cuales se enfocan en mejorar la calidad de las imágenes que han sido contaminadas por ruido o que exhiben pérdida de píxeles. Para lograr la restauración de las imágenes, se han desarrollado un importante número de algoritmos que permiten la recuperación de las regiones dañadas o pérdidas a partir de la información disponible en su entorno.

Además, recientemente existen métodos de restauración de imágenes que incorporan operaciones de álgebra lineal complejas tal como; la descomposición de valores singulares (SVD, siglas en inglés de Singular Value Decomposition) de una matriz, pero en general exhiben un alto costo computacional y el manejo ineficiente de memoria, lo que limita su uso en la ejecución de los métodos propuestos. Adicionalmente a esto, la mayoría de las bibliotecas de procesamiento de imágenes no incorporan funciones para la visualización inmediata de los resultados, lo que retrasa el análisis visual de las imágenes de salida.

Por tanto, es necesaria la creación de una herramienta de software libre que permita la integración de funciones que realicen operaciones sobre matrices de forma eficiente, y que además ofrezcan la incorporación de funciones complejas de álgebra lineal que completen su ejecución en un tiempo razonable; es fundamental que esta herramienta manipule imágenes en su forma matricial y permita la visualización de las imágenes resultantes en una interfaz gráfica y esencialmente admita el manejo de algoritmos complejos de procesamiento de imágenes.

La creación de una herramienta de esta naturaleza tiene el propósito de ser utilizado en disciplinas relacionadas con: la visión por computador, el análisis de imágenes y la inspección visual automatizada, que abarcan numerosas áreas como Ingeniería en general, Robótica, Astronomía, Detección Remota, Meteorología, Medicina (Scanner Magnético, Ultrasonido, Radiología), Biología, Biomedicina, Control industrial, Automatización, Telecomunicaciones, entre otras.

En el presente trabajo denominado *Image Processing* Cemisid, proponemos la creación de esta herramienta para mejorar el procesamiento de imágenes y los métodos de reconstrucción, mediante el desarrollo de una Biblioteca de funciones, utilizando las bibliotecas Armadillo y OpenCV. Las cuales tienen dos funciones distintas, la biblioteca Armadillo C++ se compone de funciones matemáticas de algebra lineal que permiten manipular imágenes en su forma matricial y la biblioteca OpenCV C++, la

cual mediante una interfaz gráfica permite visualizar el resultado de los métodos algorítmicos aplicados sobre las imágenes.

1.2. Antecedentes

En los últimos años se han presentado artículos e investigaciones, cada uno con aportes valiosos para la conceptualización del procesamiento de imágenes usando entornos como Matlab y variadas bibliotecas que han permitido ir manipulando imágenes. Estos aplican filtros a imágenes que han sido afectadas por medio de píxeles perdidos y de ruido impulsivo (desconocimiento total de dicha información), por lo cual, admiten mejorar la calidad de una imagen.

Tschumperlé (2012) define en su investigación una biblioteca de procesamiento de imágenes de C++ simple, fácil de usar y capaz de procesar imágenes. Tiene el objetivo de ayudar a los desarrolladores a implementar nuevos algoritmos desde cero en el área del procesamiento de imágenes. Propone un conjunto mínimo de cuatro clases, todas definidas en un único archivo de cabecera C++ CImg.h, que posee muy pocas dependencias ajustables a bibliotecas de terceros. Por consiguiente, CImg demuestra ser una herramienta pequeña, portátil y práctica para el tratamiento de imágenes.

La website de Gebel, R. (2012) plantea: el KL1p un framework portátil de C ++ para manejar la recuperación escasa de problemas inversos de sistemas lineales subdeterminados, como la técnica de muestreo comprimido (CS, en inglés *compressive sensing*). Es multiplataforma y puede utilizarse en un gran número de sistemas con un compilador compatible con C++. Los algoritmos CS más comunes se implementan a través de la combinación de operadores. Estos operadores desempeñan el mismo papel que las matrices en el álgebra lineal, pero se ejecuta bajo funciones equivalentes y eficientes (por ejemplo, la matriz de Fourier implementada con función FFT).

Los algoritmos de muestreo comprimido actualmente implementados en este framework son: Búsqueda Simple (en inglés Basis Pursuit), Búsqueda de Similitud Ortogonal (OMP[2]), Búsqueda de Similitud Ortogonal Regularizada (ROMP[3]), Búsqueda de Similitud para Muestreo Comprimido (CoSaMP[4]), Búsqueda de Subespacios (en inglés Subspace Pursuit), Suavizados L0 (SL0[5]), Paso Aproximado de Mensajes (AMP[6]), Propagación de Crecimiento de Maximización de Expectativas

[2] OMP: siglas en inglés de Orthogonal Matching Pursuit
[3] ROMP: siglas en inglés de Regularized. Orthogonal Matching Pursuit
[4] CoSaMP: siglas en inglés de Compressed Sampling Matching Pursuit
[5] SL0: siglas en inglés de Smoothed L0
[6] AMP: siglas en inglés de Approximate Message Passing

(EMBP[7]). El uso de los métodos como la matriz de Fourier y OMP son algoritmos muy útiles para la manipulación de imágenes digitales.

Indiscutiblemente, se han planteado diversos software para el procesamiento de imágenes. Seguí (2012) diseña una versión mediante el entorno Matlab para crear una interfaz que determine y elimine el ruido impulsivo en las imágenes. La creación de una GUI (en inglés Graphical User Interface), permite reunir los algoritmos de degradación y filtros de recuperación de imágenes como el filtro PGFM, el filtro de difusión, el filtro coseno y el filtro de ruido gaussiano. Como resultado, se diseña una GUI bajo Matlab que no da muchas opciones de adaptabilidad al software, en lo cual concluye; que es una interfaz visualmente sencilla. La aplicación demostró cual era el comportamiento de cada uno de los filtros implementados cuando intervienen distintos parámetros.

El presente estudio tiene como propósito contribuir con este proyecto, aportando una visión general del tratamiento de imágenes, enfocando hacia el ruido impulsivo y el posible uso de métodos de procesamiento de imágenes, que ya han dado importantes resultados. Se demostró que estos algoritmos no pueden realizarse de manera perfecta, pero se pueden lograr un mejoramiento significativo en la calidad de la información contenida en una imagen.

Este proyecto de grado requería una mayor complejidad en la manipulación de imágenes, por lo tanto, el uso de la biblioteca *The CImg* es limitada, dado el origen elemental de sus funciones internas. Lamentablemente, la biblioteca *KL1p* ha perdido soporte y tiene escasa documentación, por lo cual, su instalación es complicada y ocasiona errores.

Programas como el Seguí (2012) y otros de procesamiento de imágenes creados bajo entornos Matlab son aplicaciones limitadas a las características básicas de esta herramienta, poseen diversas desventajas dado que es un Software Privativo/Propietario, empleando interfaces de usuario (GUI) estándares, limitando al software al no contener la documentación de las funciones, las aplicaciones deben ser programadas en Lenguaje M lo cual impide la integración de módulos o API's para su ampliación y además, tiene un coste computacional alto, dado que el entorno tiene un manejo deficiente de memoria al trabajar con imágenes, usando un mayor tiempo de ejecución en las aplicaciones.

[7] EMBP: siglas en inglés de Expectation Maximization Belief Propagation

1.3. Objetivos

1.3.1. Objetivos Generales

Desarrollar una biblioteca de funciones para procesamiento avanzado de imágenes basada en las Bibliotecas Armadillo C++ y OpenCV C++.

1.3.2. Objetivos Específicos

- Desarrollar diversas funciones basadas en la biblioteca Armadillo C++ para el procesamiento de imágenes.

- Implementar funciones de OpenCV (C++) para la visualización de los resultados generados por las funciones del API Armadillo C++.

- Generar la documentación de las funciones desarrolladas usando un entorno Doxygen.

- Valorar el desempeño de las funciones de la biblioteca mediante diversas métricas la calidad de las imágenes.

- Publicar en un repositorio la biblioteca con ejemplos de ejecución de las funciones.

1.4. Resumen de las Contribuciones

Se desarrolló una biblioteca de funciones para el procesamiento de imágenes digitales llamada *Image Processing* Cemisid 1.0, está consta de dos bibliotecas indispensables para su funcionamiento, las cuales son Amadillo C++ para el manejo de matemática lineal y OpenCV C++ para la visualización de las imágenes procesadas. Entre los métodos u operaciones implementadas en la biblioteca se encuentran los Modelos de Ruido en Imágenes, el Filtrado de Ventana y la Aproximación de imágenes.

Los Modelos de Ruido se basan en contaminar imágenes con un tipo específico, los implementados son el Ruido Gaussiano, el Ruido Sal y Pimienta, el Ruido por Pérdida de Píxeles y el Ruido Impulsivo Uniforme.

El Filtrado de Imágenes son métodos básicos que son utilizados para remover ruidos que contaminan las imágenes. Los filtros implementados son el Filtro Promediador (Media) y el Filtro de Mediana. La Aproximación de Bloques son métodos complejos para la remoción de ruido en imágenes, los cuales permiten mejorar y reconstruir mediante bloques de píxeles las áreas deterioradas. Los métodos implementados son la aproximación de imágenes usando bloques solapados, la aproximación de imágenes usando bloques no solapados y la aproximación robusta de imágenes usando bloques no solapados.

Mediante la aplicación de la herramienta Doxygen; un generador de documentación de código, se obtiene la información necesaria de las clases y cada una de las funciones mencionadas. Se incluye además un método de detección de rostros para observar el alcance al que puede llegar esta biblioteca de procesamiento de imágenes. En la visualización de los resultados se manejaron pruebas para cada una de los métodos indicados, los cuales determinan que métodos de filtrado o aproximación mejoran o reconstruyen las imágenes con los diversos tipos de ruido.

En cuanto, al Filtrado Promediador se obtiene un mejor resultado en una imagen contaminada de Ruido Gaussiano, el Filtro de Mediana puede corregir dando un efecto relevante en imágenes contaminadas de Ruidos Sal y Pimienta, Pérdida de Píxeles y Ruido Impulsivo Uniforme. La Aproximación de Bloques son métodos más complejos pero determinantes en la remoción de ruido, ya que se obtiene una recuperación de áreas perdidas con mayor calidad que los métodos de filtrado.

1.5. Organización del Trabajo de Grado

Este trabajo se organiza a través de cinco capítulos a mencionar:

En el Capítulo 1 se introducen los conceptos básicos del problema de procesamiento de imágenes, se describen los antecedentes necesarios para fundamentar este proyecto, así como se exponen los objetivos generales y específicos para lograr su fin último en la creación de una nueva herramienta de manejo de imágenes.

En el Capítulo 2 se describe las bibliotecas C++ que sirven como apoyo base del proyecto, estas son Armadillo C++ y OpenCV C++, seguidamente se describe la documentación de la biblioteca generada por Doxygen, se presentan una serie de conceptos de los modelos de ruidos en imágenes, filtrados de ventana y aproximaciones de imágenes.

En el Capítulo 3 se presenta la metodología utilizada para la elaboración del proyecto de grado, la cual permite generar una guía de la idea en el desarrollo de una nueva aplicación que sirve para procesar y restaurar imágenes con nuevos métodos algorítmicos.

Posteriormente, en el Capítulo 4 se presentan las funciones finales generadas para la biblioteca con sus respectivos ejemplos que se enfocan en mostrar imágenes contaminadas con ruido, procesadas con filtros y con técnicas de aproximación de bloques no solapados y solapados.

Finalmente, en el Capítulo 5 se presentan algunas conclusiones acerca del desempeño de los métodos propuestos y algunas recomendaciones para mejoras investigaciones posteriores.

Capítulo 2

Marco Teórico

Descripción de Bibliotecas, Ruidos Impulsivos, Filtrados de Ventana y Aproximación de bloques.

2.1. Bibliotecas de C++.

2.1.1. Armadillo C++

Según Sanderson, C. (2010) y Sanderson, C., & Curtin R. (2016):

Armadillo C++ es una biblioteca basada en plantillas para álgebra lineal de código abierto para el lenguaje C ++. Tiene como objetivo un buen equilibrio entre velocidad y facilidad de uso, y aprovecha el lenguaje C ++ debido a las capacidades de integración.

Esta herramienta proporciona un manejo directo del álgebra lineal (matemáticas matriciales). La interfaz de programación de aplicaciones es de alto nivel (sintaxis de funciones) y es deliberadamente similar a los lenguajes de Matlab y Octave, de modo que las operaciones matemáticas pueden expresarse de una manera familiar y natural.

Se distribuye bajo una licencia que es aplicable tanto en software libre como en software propietario. Armadillo admite números enteros, de punto flotante y complejos, así como un subconjunto de funciones trigonométricas y estadísticas. Proporcionan varias descomposiciones de matrices a través de la integración opcional con LAPACK. Se emplea un enfoque de evaluación diferida (durante el tiempo de compilación) para combinar varias operaciones en una y reducir (o eliminar) la necesidad de temporales, esto se realiza mediante la meta-programación de C++. En comparaciones de rendimiento la biblioteca Armadillo C++ es considerablemente más rápida que Matlab y Octave, así como bibliotecas C ++ anteriores como IT++ y Newmat.

2.1.2. OpenCV C++

De acuerdo a las websites OpenCV: Librería de Visión por Computador (2015) y OpenCV (2017):

OpenCV es una biblioteca de funciones de programación dirigidas principalmente a la visión por ordenador en tiempo real. Se libera bajo una licencia de BSD y por lo tanto es libre para el uso académico y comercial. Tiene interfaces para C ++, C, Python, Java y soporta Windows, Linux, Mac OS, iOS y Android. Esta biblioteca está enfocada hacia aplicaciones en tiempo real y diseñado para tener eficiencia computacional. Los usos van desde el arte interactivo hasta la inspección de minas, la construcción de mapas en la web o la robótica avanzada. Contiene más de 500 funciones que abarcan una gran gama de áreas en el proceso de visión, como reconocimiento de objetos (reconocimiento facial), calibración de cámaras, visión etérea y visión robótica.

2.1.3. Documentación de la biblioteca (Doxygen).

Heesch, D. v. (2016) plantea:

Doxygen es una herramienta de programación generadora de documentación para código fuente. La palabra Doxygen es un acrónimo de dox(document) gen(generator) el cual, tiene como finalidad crear una documentación completa y profesional de un código escrito bajo un lenguaje de programación, entre estos es compatible con : C++, C, Java, Objective-C, Fortran, Tcl, Python, IDL (versiones Corba y Microsoft), VHDL y en cierta medida para PHP, C# y D. Esta aplicación funciona en la mayoría de sistemas Unix así como en Windows y Mac OS X. La mayor parte del código de Doxygen está escrita por Dimitri van Heesch. Esta herramienta está desarrollada bajo Mac OS X, Linux, y es compatible también con Windows.

Entre los aportes que ofrece doxygen podemos encontrar:

- Generación de documentación en línea en un navegador (HTML), un manual de referencia fuera de línea (Latex), de un conjunto de archivos fuente documentados. También hay soporte para generar salida en formato RTF (MS-Word), PostScript, PDF con hiperenlaces, HTML comprimido y páginas de manual Unix.

- La extracción de la estructura del código de archivos de origen no documentados. Esto es muy útil para encontrar archivos en grandes distribuciones de fuentes. Permitiendo también, la visualización de las relaciones entre los distintos elementos mediante gráficos de dependencia de inclusión, diagramas de herencia y diagramas de colaboración, que se generan automáticamente.

También puede utilizar Doxygen para crear la documentación normal de texto, como por ejemplo la creación de manuales.

2.2. Operaciones de Procesamiento de Imágenes

2.2.1. Modelos de Ruido en Imágenes.

Una imagen puede ser representada a través de la teoría de las matrices, donde una matriz M x N identifica la dimensión de una imagen y las coordenadas (x,y) representan la ubicación de cada píxel de la imagen. (Véase la Figura 1).

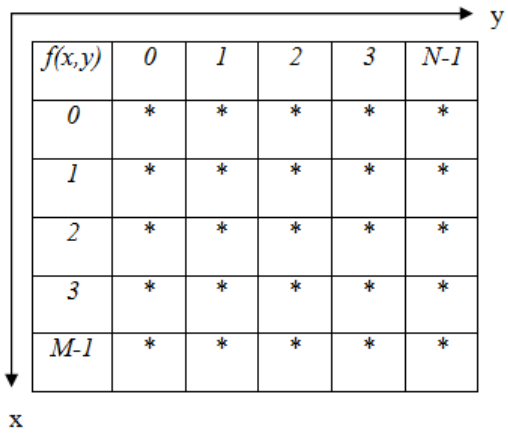


Figura 1 Representación de una imagen digital en un plano

El área de los asteriscos (*) representan una matriz imagen. Cada asterisco es un píxel, y la ubicación de cada píxel. Los valores de las coordenadas del origen son (x,y) = (0,0) y representan la posición de los píxeles a lo largo de la imagen. La notación utilizada para la representación de imágenes digitales, permite escribir la matriz M X N como se muestra a continuación:

$$f(x,y)= \begin{bmatrix} f(0,0) & f(0,1) & \dots & f(0,N-1) \\ f(1,0) & f(1,1) & \dots & f(1,N-1) \\ f(M-1,0) & f(M-1,1) & \dots & f(M-1,N-1) \end{bmatrix}$$

El Ruido en una imagen digital se reconoce cuando una imagen se ve un poco distorsionada. Por definición un ruido es cualquier perturbación que sufre una señal en el proceso de adquisición o transmisión o almacenamiento. Por consiguiente, un ruido es un defecto de información contaminada o degradada en una imagen. De manera que este puede ser modelado como un proceso estocástico o probabilístico.

2.2.1.1. Ruido Gaussiano

Un ruido gaussiano se origina "…debido a que la distribución del ruido es semejante a una distribución Gaussiana de una determinada media y varianza". (Vettorazzi, 2007, p. 46).

Es un ruido blanco y es el más usual, esencialmente distorsiona la imagen original. Si la distribución es normal es llamada Ruido Blanco Gaussiano. El ruido gaussiano tiene una función de densidad dada por la ecuación: (Véase Ecuación 1)

$$p(z) = \frac{1}{\sqrt{2\pi\sigma}} e^{-(z-\mu)^2/2\sigma^2}$$

Ecuación 1 Función de densidad gaussiana.
Fuente: (Vettorazzi, 2007, p. 46).

"En donde z representa el nivel de gris, μ es la media del valor de z y σ es la desviación estándar. A partir de la ecuación anterior, puede deducirse que aproximadamente el 70% de los datos de la imagen se encuentran en el intervalo (μ - σ , μ + σ), mientras que el 95% de los datos se encontrará en el intervalo (μ - 2σ, μ + 2σ)." (Vettorazzi, 2007, p.46).

La gráfica de la distribución gaussiana se muestra en la Figura 2:

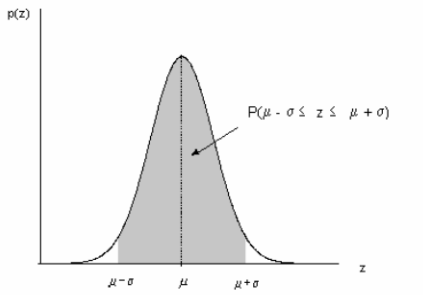


Figura 2 Grafica de la Distribución Gaussiana
Fuente: Walpole & Myers, "Probabilidad y Estadística para Ingenieros, Pearson Education 1999.

Entre las características del Ruido Gaussiano se encuentran:

- Generar variaciones pequeñas en la imagen.
- Disminuir la nitidez de la imagen.
- Aumentar la borrosidad
- Pérdida de detalles
- Suelen producirse por componentes o circuitos electrónicos (sensores, digitalizadores, falta de iluminación, altas temperaturas, entre otros).
- Afecta la imagen completa.
- La intensidad de los píxeles se ve alterada.

2.2.1.2. Ruido Sal y Pimienta

Ciertamente este ruido "Es causado principalmente por fallas en el funcionamiento de los sensores encargados de capturar una imagen o por errores de tiempo, cuando se produce el proceso de digitalización de la imagen." (Vettorazzi, 2007, p.51).

Este ruido tiene una función de densidad dada por la siguiente ecuación:

$$p(z) = \begin{cases} P_a & para \quad z = a \\ P_b & para \quad z = b \\ 0 & \text{en cualquier otro caso} \end{cases}$$

Ecuación 2 Función de densidad ruido sal y pimienta
Fuente: (Vettorazzi, 2007, p.51)

Para lo cual, se establece que si b>a, puntos claros aparecerán en la imagen, si a>b aparecerán puntos oscuros contaminando la imagen y si Pa y Pb son cero, el Ruido es llamado "unipolar". En el caso de una imagen en escala de grises, los puntos claros o blancos pertenecen a los valores cercanos a 255 y los puntos oscuros o negros pertenecen a los valores cercanos a 0. Para una imagen de 8 bits esto significaría que a=0 (negro) y b= 255 (blanco), como se observa en la Figura 3.

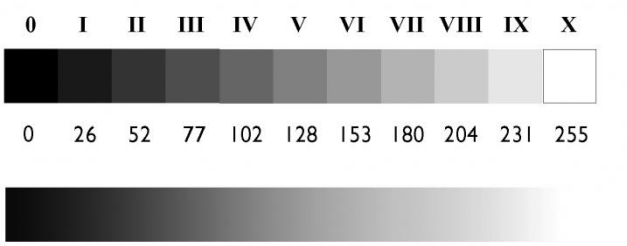


Figura 3 Escala de Grises

"Este tipo de ruido generalmente toma valores extremos cercanos al 0 ó 255 en la imagen debido a que los impulsos de ruido pueden ser negativos o positivos y generalmente, suele suponerse que los valores de a y b se encuentran "saturados", ya sea en sus valores máximos o en sus valores mínimos cuando se digitaliza la imagen". (Vettorazzi, 2007, p.51)

Entre sus características se encuentran:

- El valor de un píxel toma valores muy altos o muy bajos.
- El valor máximo se le llama sal y el mínimo pimienta.
- El ruido se controla mediante un nivel, el cual representa la densidad del ruido.

2.2.1.3. Pérdida de Píxeles

Cuando las imágenes son representadas en escala de grises, cada píxel está representado por un valor numérico que nos indica la intensidad del píxel, el cual estará entre el 0 y el 255. Mientas más bajo sea el valor, más oscuro será el píxel y viceversa. De esta forma, un píxel con valor 0 corresponde a negro y un píxel con valor 255 a blanco. En una imagen existe perdida de píxeles cuando el ruido cercano a 0 (ruido negro o puntos negros) contaminan la imagen. Por ejemplo la representación de una imagen digital 3x3 en escala de grises: (Véase Figura 4)

$$\text{Imagen} = \begin{matrix} 0 & 5 & 104 \\ 168 & 255 & 176 \\ 46 & 200 & 60 \end{matrix}$$

Figura 4 Imagen Digital 3x3

2.2.1.4. Ruido Impulsivo Uniforme

Así mismo se dice que "El ruido uniforme puede utilizarse para generar cualquier otro tipo de distribución de ruido y es comúnmente utilizado con el fin de degradar imagen para la evaluación de algoritmos de restauración de imágenes digitales". (Vettorazzi, 2007, p.49).

La función de densidad de este tipo de ruido está dada por la siguiente ecuación 3:

$$p(z) = \begin{cases} \frac{1}{b-a} & si \quad a \leq z \leq b \\ 0 & \text{en cualquier otro caso} \end{cases}$$

Ecuación 3: Fuente: (Vettorazzi, 2007, p.50).

Donde la media y varianza se calculan con las siguientes ecuaciones 4:

$$\mu = \frac{a+b}{2} \qquad ; \qquad \sigma^2 = \frac{(b-a)^2}{12}$$

Ecuación 4: Fuente: (Vettorazzi, 2007, p.50).

La gráfica de la distribución de ruido uniforme se muestra en la Figura 5:

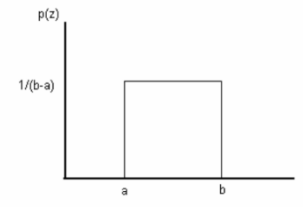


Figura 5 Gráfica de la distribución de ruido uniforme
Fuente: (Vettorazzi, 2007, p.50).

2.2.2. Filtrado de ventana

La aplicación de un filtrado se realiza con el fin de mejorar una imagen mediante la reducción o eliminación de ruido. Se utiliza principalmente para eliminar altas o bajas frecuencias en una imagen, lo cual permite suavizar una imagen, realzar o detectar bordes. Tiene como objetivo aproximar el valor de un píxel, usando información de la propia imagen. Este método consiste en modificar el píxel elegido de un conjunto de píxeles vecinos a él. (Véase la Figura 6).

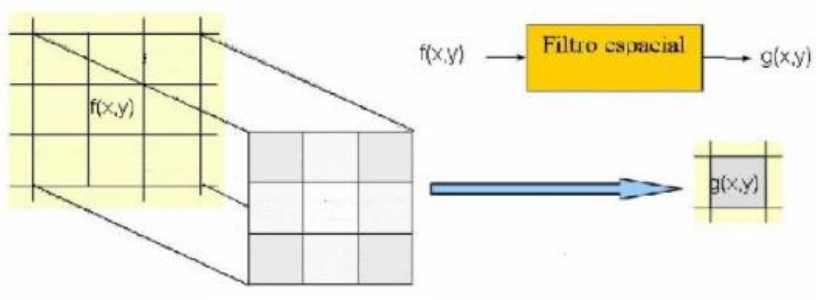


Figura 6 Filtrado de Ventana
Fuente: (Ruido y Filtrado, 2015, p.24).

Una imagen puede ser filtrada mediante el dominio del espacio o el dominio de frecuencia, la primera se trabaja directamente sobre los píxeles de la imagen y en la segunda las operaciones se llevan a cabo con la transformada de Fourier en la imagen.

Nos enfocaremos en los filtros del dominio del espacio lineal y no lineal, explicando lo siguiente: el filtrado de la mediana y el filtrado promedio o media aritmética

El filtrado se realiza mediante la selección de una submatriz que será llamada ventana, la cual está compuesta por un valor de un píxel seleccionado para cambiar y tiene un entorno llamados valores adyacentes o vecinos.

2.2.2.1. Filtro Promediador (media)

Se emplea para disminuir el valor de las altas frecuencias, para suavizar la imagen. Consiste en reducir la cantidad de variaciones de intensidad entre los píxeles adyacentes. La intensidad del píxel seleccionado se sustituye por el valor promedio de los píxeles adyacentes de la ventana. Uno de los principales problemas que se difumina son los bordes y detalles de contraste.

Entre sus características se encuentran:

- Mientras el tamaño de la ventana sea mayor, hay mayor reducción del ruido y difuminación de bordes.
- Es un filtro sensible a los cambios de los valores de la ventana, ya que puede crear intensidades de grises que no aparecían en la imagen.

2.2.2.2. Filtro de Mediana

El filtrado se basa en remplazar el valor de un gris de cada píxel por la mediana de los niveles de grises en un entorno del píxel

Como objetivo se tiene que el píxel seleccionado remplace con un nivel de gris más próximo al valor de sus adyacentes. (Véase la Figura 7)

Se plantea en (Ruido y Filtrado, 2015, p.35) las siguientes características:

- Atenúa el ruido sal y pimienta.
- Elimina efectos engañosos.
- Preserva bordes de la imagen.
- Pierde detalles como puntos y líneas finas.
- Redondea las esquinas de los objetos.
- Desplaza los bordes.

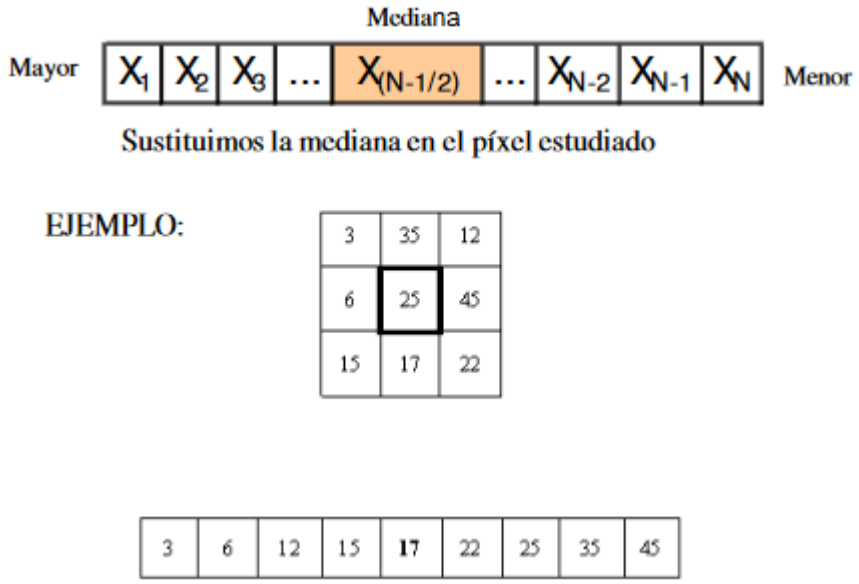


Figura 7 Filtro de mediana

Fuente: (Ruido y Filtrado, 2015, p.35).

2.2.3. Aproximación de Imágenes

Las aproximaciones de imágenes usando bloques son métodos para restaurar imágenes con píxeles perdidos, utilizando ventanas o bloques de píxeles de 8x8 para el procesamiento secuencial de una imagen.

A continuación se explican tres métodos de Aproximación:

- Aproximación usando Bloques no Solapados.
- Aproximación usando Bloques Solapados.
- Aproximación Robusta usando Bloques Solapados.

2.2.3.1. Usando Bloques no Solapados

La aproximación de imágenes usando bloques no solapados consiste, en seleccionar una ventana o bloque de 8x8 píxeles y procesarlo mediante funciones de Transformada de Fourier, las cuales están contenidas en la Biblioteca Armadillo C++. Para efectuar este procedimiento se debe seguir lo siguiente:

- Primero se debe aplicar al bloque de 8x8 la Transformada Rápida de Fourier: fft2(). Este proceso convierte el bloque actual, de una matriz real en una matriz compleja.

- Luego la matriz resultante debe ser umbralizada. El umbral es la magnitud del sexto coeficiente más grande. Los valores que no pasen el umbral se convierten a cero, dado que estos valores tienen mayor posibilidad de ser ruidos o contaminación en la imagen. A este proceso se le conoce como Sparse (también llamado coeficientes dispersos en una matriz en donde los valores relevantes son distintos de cero).

- A continuación, se aplica a la matriz el proceso de la transformada inversa de Fourier: ifft2(). Este proceso permite llevar la matriz compleja al rango de matriz real para ser posteriormente sustituida en la imagen original y continuar con el próximo bloque o ventana de 8x8.

- Para seleccionar el próximo bloque, el algoritmo debe moverse 8 píxeles en la imagen de manera horizontal y obtener el siguiente bloque de 8x8 para que este sea el próximo bloque a ser procesado y de esta manera repetir los procedimientos anteriores.

  Una desventaja en la aproximación de bloques no solapado es que provoca blocking similar al píxelado pero en bloques, dando como resultado que los objetos curvos y las líneas diagonales tengan una apariencia poco natural y más cuadrada.

2.2.3.2. Usando Bloques Solapados

La aproximación de imágenes usando bloques solapados consiste en seleccionar una ventana o bloque de 8x8 píxeles y procesarlo mediante funciones de Transformada de Fourier, las cuales están contenidas en la Biblioteca Armadillo C++. Para efectuar este procedimiento se deben seguir lo siguientes pasos:

- Primero se debe aplicar al bloque de 8x8 la Transformada Rápida de Fourier: fft2(). Este proceso convierte el bloque actual que es una matriz real en una matriz compleja.

- Luego la matriz resultante debe ser umbralizada. El umbral es la magnitud del sexto coeficiente más grande. Los valores que no pasen el umbral se convierten a cero, dado que estos valores tienen mayor posibilidad de ser ruidos o contaminación en la imagen. A este proceso se le conoce como Sparse (también llamado coeficientes dispersos en una matriz en donde los valores relevantes son distintos de cero).

- A continuación se aplica a la matriz el proceso de la transformada inversa de Fourier: ifft2(). Este proceso permite llevar la matriz compleja al rango de matriz real y continuar con el próximo bloque o ventana de 8x8.

- Para seleccionar el próximo bloque, el algoritmo debe moverse 1 píxel en la imagen de manera horizontal y obtener el siguiente bloque de 8x8 para que este sea el próximo bloque a ser procesado.

- Se debe efectuar los anteriores procedimientos en el bloque seleccionado. En este paso podemos notar, que al procesar los bloques solapados los píxeles se vuelven a procesar y se repiten, por lo tanto cada píxel se procesa 64 veces. Esos mismos píxeles son guardados en un vector, al cual se le aplica el promedio o mediana para obtener el mejor valor de él, y de esta manera obtener una mejor aproximación en el bloque.

- Al obtener la mejor aproximación el bloque este es sustituido en la imagen original.

2.2.3.3. Robusta usando Bloques Solapados

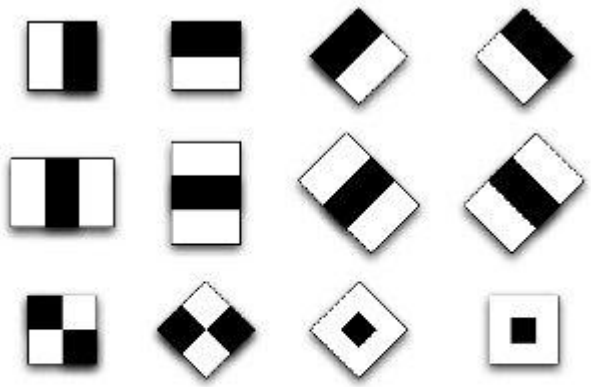
La aproximación robusta de imágenes usando bloques solapados consiste en seleccionar un bloque de la imagen de 8x8 y procesarlo mediante la "Transformada Discreta del Coseno Robusto basada en Mediana Ponderada" basado en los artículos de Ramírez J. & Paredes J. (2014) y Ramírez J. & Paredes J. (2015). Este algoritmo permite realizar los mismos pasos anteriores de la aproximación por imágenes, usando bloques solapados pero tiene como ventaja que el resultado de la reconstrucción es de mejor calidad aunque como desventaja tenga mayor tiempo de cómputo.

2.2.4. Detección de Rostros

Según Detección de rostros. (2017):

OpenCV cuenta con un algoritmo o clasificador Haar (clasificadores en cascada) los cuales están entrenados en detectar rostros y estos son almacenados en archivos XML. El clasificador Haar fue el primer framework de detección de objetos propuesto por Paul Viola y Michael Jones en 2001 que permitía el análisis de imágenes en tiempo real, haciendo uso de una función matemática (Wavelet Haar) propuesta por Alfred Haar en 1909.

Los clasificadores haar, definen regiones rectangulares sobre una imagen en escala de grises (imagen integral) y al estar formada por un número finito de rectángulos, se puede obtener un valor escalar que consiste en sumar los píxeles de cada rectángulo, en base a una serie de clasificadores en cascada. Cada clasificador determina si la subregión se trata del objeto buscado o no. A diferencia de otros algoritmos, este solo invierte capacidad de procesamiento a las subregiones que posiblemente representen un rostro.



Clasificadores Haar

Pasos para detectar rostros:

Para la detención de rostros el algoritmo debe manejar lo siguiente:

- Primero: debemos cargar la imagen
- Segundo: convertir la imagen a escala de grises. Para convertir una imagen a escala de grises o a otro formato, OpenCV cuenta con la función cvtColor y se utiliza del siguiente modo:

cvtColor(imagen, imagen, CV_BGR2GRAY);

- Tercero: aplicar ecualización de histograma a la imagen en grises para estandarizar el contraste y brillo de la imagen, esto para que distintas condiciones de iluminación no afecten la detección del rostro en la imagen, de este modo el algoritmo es más eficaz al detectar las caras presentes en una imagen. (Véase Figura 8).

equalizeHist(imagen, imagen);

Figura 8 Ecualización de histograma

- Al tener la imagen procesada se carga el detector a utilizar, pasándole a la función el nombre del clasificador al método load de la clase CascadeClassifier, los archivos .xml. Estos archivos se encuentran en C:\opencv\data aquí se encuentran varias carpetas que contienes distintos tipos de clasificadores, en la carpeta C:\opencv\data\haarcascades se encuentran varios clasificadores no solo para detectar rostros, sino también para la detección de ojos, boca, nariz, entre otros. Se debe usar de la siguiente forma:

```
CascadeClassifier detector;
if(!detector.load("haarcascade_frontalface_alt.xml"))
    cout << "No se puede abrir clasificador."<< endl;
```

Para detectar rostros de frente se usa haarcascade_frontalface_alt.xml, para detectar cuerpo completo se puede utilizarhaarcascade_fullbody.xml, para detectar ojos se cuenta con haarcascade_eye.xml, existen muchos otros.

- Para detectar los rostros presentes en la imagen, se utilizan coordenadas que se guardan en la variable llamada rect.

```
vector<Rect> rect;
detector.detectMultiScale(dest, rect);
```

- Finalmente se recorren los rectángulos encontrados por el algoritmo y se marca el rostro en la imagen original. (Véase Figura 9)

```
for (Rect rc : rect){
    rectangle(imagen,
            Point(rc.x, rc.y),
            Point(rc.x + rc.width, rc.y + rc.height),
            CV_RGB(0,255,0), 2);
}
```

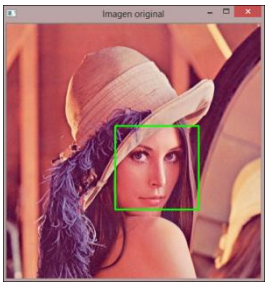


Figura 9 Detección de rostro

2.2.5. Métricas de Calidad de la Imagen

Las métricas PSNR, MSE y MAE, son las más usadas para evaluar la calidad de las imágenes, que han sido reconstruidas mediante los distintos métodos de remoción de ruido.

- El PSNR es la relación señal a ruido de pico, una medida de valoración de la separación cromática (colores) entre imágenes definidas a partir de diferentes paletas. Se calcula con la ecuación 5:

$$PSNR = 10*LOG(10)*\left|\frac{255^2}{(mean(mean((IO-IF)^2)))}\right|$$

Ecuación 5

Donde IO es la matriz o imagen con ruido y IF la imagen filtrada.

- El MSE es el error cuadrático medio. Se calcula mediante la ecuación 6:

$$MSE = mean(mean((IO-IF)^2))$$

Ecuación 6

- El MAE es el error promedio absoluto. Se calcula mediante la ecuación 7:

$$MAE = mean(mean|IO-IF|)$$

Ecuación 7

Capítulo 3
Marco Metodológico.

La Metodología, Fases y Procedimientos.

3.1. Descripción de la Metodología SOFTENG Agile.

La Metodología SOFTENG tiene como objetivo minimizar riesgos, gestionar cambios de forma eficaz, y ofrecer un producto de calidad que cumpla con las expectativas del usuario final.

El marco metodológico está orientado a procesos, guiando al proyecto hacia un objetivo común y claramente definido, por el cual su ejecución se realiza según los plazos y costes previstos.

3.2. Fases de la Metodología.

La factibilidad de Metodología SOFTENG Agile está orientada a seguir fases para la ejecución efectiva del proyecto.

En la Figura 10 se muestra el diagrama de fases y procesos de la metodología:

Figura 10 Fases y procesos de la Metodología SoFteng Agile.

3.3. Descripción de las Fases de la Metodología

Desde la fuente "SOFTENG Agile" (s.f.) se extraen las siguientes fases:

Fase I: Estudio estratégico: establece las bases y el alcance del proyecto, así como los recursos necesarios, timing y costes, sugiere además, que se quiere obtener y se plantea descubrir nuevas oportunidades que logren incrementarlo para conseguir su máximo alcance en el mercado actual.

Fase II: Diseño y arquitectura: clarifica los objetivos del proyecto, plantea la estrategia más adecuada para el desarrollo del mismo, así como describe la funcionalidad a implementar definiendo su alcance.

Esta fase se divide en etapas:

- **Análisis funcional:** precisa cuales son los objetivos a alcanzar y la descripción modular detallada de los requerimientos del proyecto.
- **Análisis tecnológico:** consiste en seleccionar de la tecnología a aplicar, arquitectura, diagrama de objetos, modelo conceptual y lógico de la BD y definición de procesos.
- **Maqueta:** define la línea gráfica de interfaz.
- **Planificación:** fundamenta el plan detallado del proyecto, asignación de recursos y definición de entregables.

A continuación en la Figura 11 se expone el ciclo del diseño y arquitectura de que trata de un proceso que se lleva a cabo mediante ciclos iterativos hasta que el usuario solicitante se encuentre conforme con el producto a desarrollar.

Figura 11 Ciclos del diseño y arquitectura de la Metodología SOFTENG Agile

Fase III: <u>Producción:</u> consiste en el desarrollo del proyecto organizado en hitos y entregables y así facilitar la revisión de la aplicación a medida que se va construyendo.

Consta de la utilización de los siguientes mecanismos:

- Prototipo
- Diseño de interfaz
- Creación de la Base de datos
- Implementación
- Integración y pruebas-testeo

Fase IV: <u>Control de calidad:</u> en esta fase la aplicación ya ha sido desarrollada y testeada con éxito, por lo cual pasará por el control de calidad, el cual se basa en un profundo testeo, tanto funcional (comparándolo con la documentación de requerimientos), como técnico (especialmente de carga y stress, simulando conexiones de usuarios que la usan), finalizando con la aceptación del usuario solicitante.

Fase V: <u>Puesta en marcha:</u> esta fase también se llama fase de despliegue y se divide en cinco etapas cuyo orden y ámbito dependerá del proyecto en cuestión:

- **Instalación del hardware:** consiste en realización de la instalación del servidor o clúster de servidores.
- **Instalación del software:** radica en la instalación y configuración el software y, en general, los requerimientos necesarios en servidor para el funcionamiento correcto de la aplicación.
- **Instalación de la aplicación:** establece la migración desde el servidor de pruebas al servidor definitivo.
- **Migración de datos:** plantea la migración de la información desde el antiguo gestor de base de datos de la organización al nuevo servidor.
- **Formación:** fundamenta toda la documentación necesaria, y capacitación de los usuarios para el uso de la aplicación o gestión de contenidos en el caso de proyectos Web.

Fase VI: Fase de cierre, inicio de la mejora continúa y soporte: establece que se han alcanzado los objetivos del proyecto, durante este periodo se pueden analizar ampliaciones funcionales que aporten más valor añadido al proyecto, o nuevas oportunidades para mejorarlo que desemboquen en futuras colaboraciones.

Fase VII: Gestión del proyecto: constituye en todas la actividades de gestión necesarias para llevar a buen término el proyecto y lograr los objetivos marcados. Estas actividades las lleva a cabo el coordinador de proyecto asignado, y consisten principalmente en el control y coordinación de recursos, costes, tiempos, planificación, entregables y calidad.

3.4. Procedimientos para la realización del Proyecto

A continuación se plantea como se realizó el Proyecto a través de la metodología "SOFTENG Agile".

Fase I: Estudio Estratégico: Los procedimientos de esta fase estuvieron referidos a:

- Establecer las bases necesarias de la biblioteca por medio de la conceptualización del área de ideas en el estudio del procesamiento de imágenes.
- Descubrir el potencial de las herramientas utilizadas para el desarrollo de software que logran la implementación de las funciones.
- Proyectar el alcance de las funciones de la biblioteca, para lograr abarcar la solución de la problemática de forma más eficiente.

Fase II: Diseño y arquitectura: el diseño se efectuará por etapas:

- Análisis funcional: Se procedió por medio de la definición del problema, el objetivo general y los objetivos específicos para la creación de la Biblioteca.
- Análisis tecnológico: Se presentó mediante el planteamiento y aplicación de la tecnología mediante la programación en el lenguaje C++, con las Bibliotecas Armadillo C++, OpenCV C++ y Doxygen. Esta se expone en la siguiente arquitectura: (Véase Figura 12).

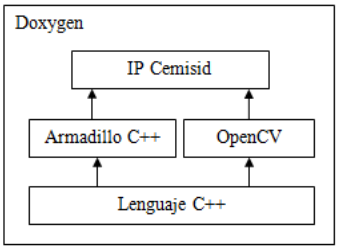


Figura 12 Arquitectura de la Biblioteca IP Cemisid

- Planificación: Consistió en la creación de un cronograma de actividades con el tutor para el desarrollo del proyecto.

Fase III: Producción: Mediante el desarrollo de la Biblioteca IP Cemisid las funciones añadidas a la biblioteca, han sido probadas a medida de que se iban construyendo. Cada revisión se efectuó mediante la interfaz gráfica codificada en OpenCV, que permitió la visualización de cada resultado de las funciones.

Fase IV: Control de calidad: Se efectuó con la valoración de una métrica estadística que evaluó la calidad de las imágenes que han sido reconstruidas mediante los distintos métodos de remoción de ruido.

Fase V: Se dividió en etapas:

- **Instalación del software:** En esta fase se realizó y documento en las secciones la instalación e integración de los distintas herramientas de software y bibliotecas (Eclipse, Armadillo C++, OpenCV C++).

- **Formación:** Se preparó la documentación de cada una de las funciones de la Biblioteca, mediante la integración de Doxygen, la cual da soporte para generar documentos HTML y latex que exponen los métodos y utilización de cada una de las funciones.

- **Fase de cierre, inicio de la mejora continúa y soporte:** Se aplicaron varios archivos de prueba, que podrán ser usados para la ejecución de la biblioteca y además este proyecto cuenta con un repositorio que será accesible para futuras colaboraciones.

Fase VI: Gestión del proyecto: En esta fase se demostró la ejecución de todas las actividades planificadas y el logro de los objetivos planteados.

Capítulo 4
Resultados.

Funciones y Resultados de la versión 1.0 de la Biblioteca Image Processing Cemisid.

4.1. Biblioteca Image Processing Cemisid

La biblioteca cuenta con un archivo cabecera llamado Image.h este archivo contiene el nombre de todos los métodos implementados. Esta cuenta con la clase principal de Image() para manipular todos los métodos internos.

Archivo

- include <Image.h>

En la siguiente sección 4.2 se exponen las Funciones de la Biblioteca IP Cemisid 1.0 detallando la información interna de las funciones. Para más información se puede consultar la documentación generada por Doxygen llamada "Documentación de la Biblioteca IP Cemisid 1.0" que se encuentra anexa a este documento.

4.2. Funciones de la Biblioteca IP Cemisid 1.0

A continuación se presenta la información de los métodos implementados en la Biblioteca, donde se exponen;

- Descripción: Información de que hace el método.
- Función: Se refiere al nombre asignado a la función para acceder al método.
- Prueba: Es el archivo donde se encuentra un ejemplo del método.
- Ejemplo: Es una prueba corta de cómo puede ser utilizada la función.
- Parámetros: Son todos los parámetros que influyen en la función.
- Retorna: Son todos los parámetros que retorna la función.

Descripción	**Constructor de la clase Image, accede a sus métodos internos**
Función	Image ()
Prueba	basic_funcions.cpp
Ejemplo	Image I;
Parámetros	No tiene parámetros
Retorna	No retorna nada

Descripción	**Carga una imagen con formato .pgm al formato mat de Armadillo C++**
Función	arma::mat Image_load (arma::mat, std::string)
Prueba	basic_funcions.cpp
Ejemplo	mat matrix; string ruta= "src/Resources/images/house.256.pgm"); Image_load(matrix, ruta);
Parámetros	• matrix es la matriz que contiene los píxeles de la imagen. • ruta es la ubicación o localización de la imagen.
Retorna	No retorna nada. Carga la imagen a la clase

Descripción	**Constructor parametrizado de la clase Image, guarda la matriz de la imagen con el tamaño de las filas y columnas.**
Función	Image (arma::mat)
Prueba	basic_funcions.cpp
Ejemplo	mat matrix; Image(matrix);

Parámetros	• matrix: matriz que contiene los píxeles de la imagen. • (private) rows: guarda el valor de las filas • (private) cols: guarda el valor de las columnas
Retorna	No retorna nada

Descripción	**Obtiene las dimensiones nxn de la matriz imagen.**
Función	void Dimensión ()
Prueba	basic_funcions.cpp
Ejemplo	Dimensión();
Parámetros	No tiene parámetros
Retorna	Retorna la dimensión de la ímagen nxn

Descripción	**Obtiene el valor de un píxel de la matriz de la imagen original.**
Función	int Get_pixel (int, int)
Prueba	basic_funcions.cpp

Ejemplo	int row=5;
	int col=5;
	Get_pixel(row, col);

Parámetros	• row es el número de filas de la matriz • col es el número de columnas de la matriz
Retorna	Retorna el valor de un píxel

Función	**Imprime todos los píxeles de la matriz imagen**
Descripción	void Get_pixels ()
Prueba	basic_funcions.cpp
Ejemplo	Get_pixels();
Parámetros	No tiene parámetros
Retorna	Retorna todos los píxeles.

Descripción	**Guarda la matriz mat en formato pgm**
Función	void SaveImage (arma::mat, std::string)
Prueba	gaussian_noise.cpp
Ejemplo	mat matrix; string ruta = "src/Resources/images/house.256.pgm"); SaveImage(matrix, ruta);
Parámetros	• matrix es la matriz que contiene los píxeles de la imagen.

	• ruta es la ubicación o localización de la imagen.
Retorna	Retorna si la matriz fue guardada o si hubo problemas al guardar.

Descripción	**Aplica Ruido Gaussiano a una matriz** El Ruido Gaussiano es una matriz de media 0 y con una desviación estándar variable.
Función	arma::mat Gaussian_noise (double)
Prueba	gaussian_noise.cpp
Ejemplo	double nivel=0.1; Gaussian_noise(nivel);
Parámetros	• nivel es el nivel/porcentaje de varianza variable (ruido impulsivo) Parámetros Internos: • A es una matriz gaussiana de dimensiones iguales a la matriz y de media 0 y varianza 1 • B es una matriz donde se almacena la matriz gaussiana de media 0 y el cálculo de una varianza determinada • std es la desviación estándar
Retorna	Retorna la matriz con el ruido aplicado

Descripción	**Aplica el Ruido Pérdida de píxeles a una matriz**
Función	arma::mat Lost_pixels_noise (double)
Prueba	lost_Pixels_noise.cpp
Ejemplo	double nivel=0.1; Lost_pixels_noise(nivel);
Parámetros	• nivel es el porcentaje de ruido
Retorna	Retorna la matriz con el ruido aplicado

Descripción	**Aplica Ruido Sal y Pimienta a una matriz.**
Función	arma::mat Salt_and_pepper_noise (double)
Prueba	salt_pepper.cpp
Ejemplo	double nivel=0.1; Salt_and_pepper_noise(nivel);
Parámetros	• nivel es el porcentaje de ruido
Retorna	Retorna la matriz con el ruido aplicado

Descripción	**Aplica Ruido Impulsivo Uniforme a una matriz**.
Función	arma::mat Impulsive_uniform_noise (double)

Prueba	uniform_impulsive_noise.cpp
Ejemplo	double nivel=0.1; Impulsive_uniform_noise(nivel);
Parámetros	• nivel es el porcentaje de ruido
Retorna	Retorna la matriz con el ruido aplicado

Descripción	**Cálculo del PSNR** Permite calcular el PSNR de una imagen con ruido y una imagen filtrada
Función	double PSNR (arma::mat , arma::mat)
Prueba	filtering_mean.cpp filtering_median.cpp
Ejemplo	mat matrix_r; mat matriz_f; PSNR(matrix_r, matriz_f);
Parámetros	• matrix_r es la matriz con ruido. • matriz_f es la matriz filtrada o recuperada
Retorna	Retorna el valor del PSNR

Descripción	**Cálculo del MAE** Permite calcular el MAE (Error promedio absoluto) de una imagen con ruido y una imagen filtrada
Función	double MAE (arma::mat , arma::mat)
Prueba	filtering_mean.cpp filtering_median.cpp
Ejemplo	mat matrix_r; mat matriz_f; PSNR(matrix_r, matriz_f);
Parámetros	• matrix_r es la matriz con ruido. • matriz_f es la matriz filtrada o recuperada
Retorna	Retorna el valor del MAE

Descripción	**Cálculo del MSE** Permite calcular el MSE (Error cuadrático medio) de una imagen con ruido y una imagen filtrada
Función	double MAE (arma::mat , arma::mat)
Prueba	filtering_mean.cpp filtering_median.cpp
Ejemplo	mat matrix_r; mat matriz_f;

	PSNR(matrix_r, matriz_f);
Parámetros	• matrix_r es la matriz con ruido. • matriz_f es la matriz filtrada o recuperada
Retorna	Retorna el valor del MAE

Descripción	**Aplica un filtrado usando el promedio o la mediana para remover ruido.** El método aplicar un filtrado a una matriz, utilizando el sparse. Se selecciona una ventana de lxl a la cual se le aplica un promedio o la mediana para restaurar la Imagen. Se calcula utilizando el tamaño de la ventana l y el píxel central de la ventana, luego se aplica el promedio o mediana de los píxeles adyacentes al central.
Función	arma::mat Filtering (arma::mat , int , int)
Prueba	filtering_mean.cpp filtering_median.cpp

Ejemplo	mat matrix;
	int l = 5;
	int flag = 0;
	Filtering(matrix, l, flag);

Parámetros	• matrix es la matriz con ruido a la cual se le aplicara el filtrado. • l es el tamaño de la ventana • flag es la bandera que indica si se efectúa el promedio o la mediana. Donde promedio es igual a 0 y mediana igual a 1
Retorna	Retorna la matriz con el filtrado aplicado

Descripción	**Carga dos imágenes en una ventana de opencv con sus respectivas etiquetas (label).**
Función	void Two windows opencv (const char _ , const char _ , std::string , std::string)
Prueba	overlap.cpp noise_remover.cpp idtc_Robusta_mean.cpp
Ejemplo	const char_ imag1= " lady_256_0_1.pgm"; const char_ imag2= " lady_256_1.pgm"; string label1 = "Imagen1"; string label2 = "Imagen2"; Two_windows_opencv(imag1, imag2, label1, label2);
Parámetros	• imag1 es la ruta o ubicación de la imagen1. • imag2 es la ruta o ubicación de la imagen2. • label1 es el nombre de la imagen1. • label2 es el nombre de la imagen2.
Retorna	Abre la ventana opencv con dos imagenes.

Descripción	**Carga cuatro imágenes en una ventana de opencv con sus respectivas etiquetas (label).**
Función	void Four windows opencv (const char _ imag1, const char _ imag2, const char _ imag3, const char _ imag4, const char _ label1, const char _ label2, const char_ label3, const char _ label4)
Prueba	gaussian_noise.cpp filtering_mean.cpp
Ejemplo	const char_ imag1= " lady_256_0_1.pgm"; const char_ imag2= " lady_256_1.pgm"; const char_ imag3= " lady_256_10.pgm"; const char_ imag4= "lady_256_100.pgm"; Four_windows_opencv(imag1, imag2, imag3, imag4, label1, label2, label3, label4);
Parámetros	• imag1 es la ruta o ubicación de la imagen1 • imag2 es la ruta o ubicación de la imagen2 consecutivamente con….Imag3 … Imag4 • label1 es el nombre de la imagen1 • label2 es el nombre de la imagen2 consecutivamente con ….Imag3 … Imag4
Retorna	Abre la ventana opencv con cuatro imágenes

Descripción	**Aproximación por bloques no solapado (ventana 8x8)** Permite remover ruido mediante el uso de una ventana 8x8 no solapado
Función	arma::mat Noise remover (arma::mat , int)
Prueba	noise_remover.cpp
Ejemplo	mat matrix; int image_tam = 256; Noise_remover(matrix, image_tam);
Parámetros	• matrix es la matriz que se le removerá el ruido. • image_tam es el tamaño de la imagen.
Retorna	Retorna la matriz sin ruido.

Descripción	**Aproximación por bloques solapado (ventana 8x8)** Permite remover ruido mediante el uso de una ventana 8x8 solapado
Función	arma::mat Overlap (arma::mat)
Prueba	overlap.cpp
Ejemplo	mat matrix; Overlap(matrix);

Parámetros	• matrix es la matriz que se le removerá el ruido.
Retorna	Retorna la matriz sin ruido.

Descripción	**Aproximación Robusta por bloques solpados (ventana 8x8)** Permite remover ruido mediante el uso de una ventana 8x8 solapado por media (0) y mediana ponderada (1).
Función	mat Image::Idtc_Robusta(mat matrix, int flag)
Prueba	idtc_Robusta_median.cpp idtc_Robusta_mean.cpp
Ejemplo	mat matrix; int flag=0; Idtc_Robusta(matrix, flag);
Parámetros	matrix es la matriz que se le removerá el ruido. flag es la bandera que indica si es 0 se aplica la media y si es 1 la mediana
Retorna	Retorna la matriz sin ruido.

Descripción	**Detección de Rostros**
Función	void Detection Face (string)
Prueba	detection_face.cpp
Ejemplo	string ruta=" lady_256_0_1.pgm"; Detection Face (ruta);
Parámetros	ruta es la ubicación de la imagen
Retorna	Retorna la matriz con rostros detectados

4.3. Ejemplos.

4.3.1. Imágenes contaminadas con ruido.

4.3.1.1. Ruido Pérdida de Píxeles. Se observa en la Figura 13 el porcentaje de ruido 0.1%, 1%, 10% y 100%.

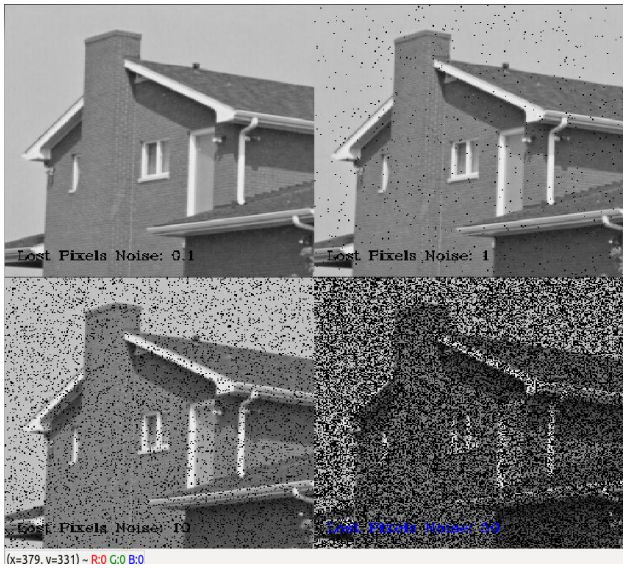


Figura 13 Ruido Pérdida de Píxeles

4.3.1.2. Ruido Gaussiano. Se observa en la Figura 14 el porcentaje de ruido 0.1%, 1%, 10% y 100%

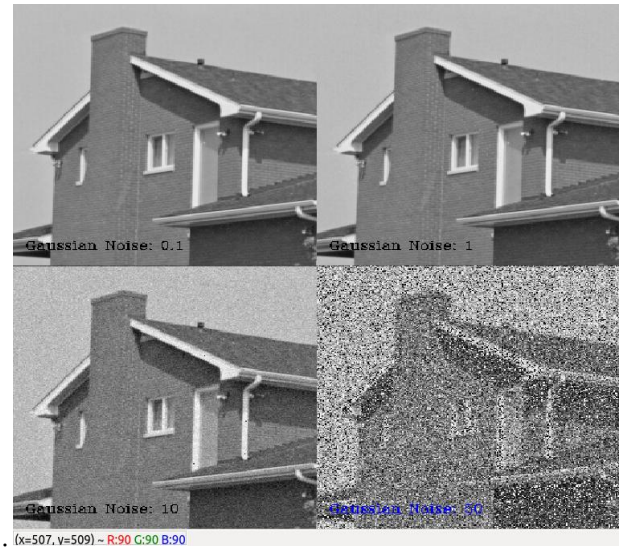


Figura 14 Ruido Gaussiano

4.3.1.3. Ruido Sal y Pimienta. Se observa en la Figura 15 el porcentaje de ruido 0.1%, 1%, 10% y 100%.

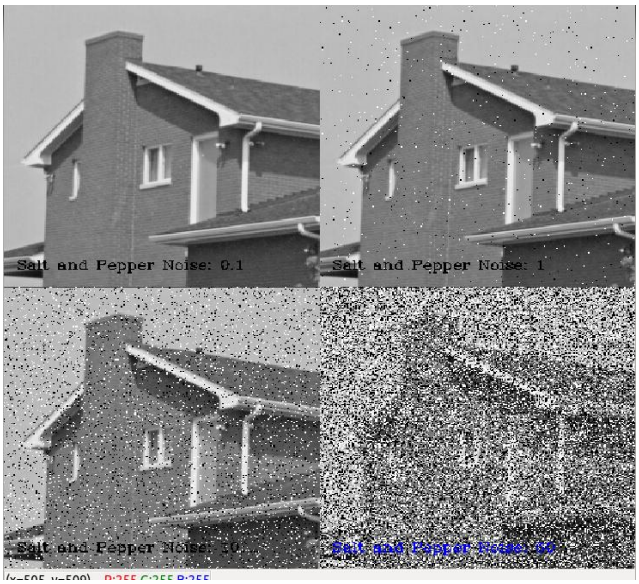


Figura 15 Ruido Sal y Pimienta

4.3.1.4. Ruido Impulsivo Uniforme. Se observa en la Figura 16 el porcentaje de ruido 0.1%, 1%, 10% y 100%.

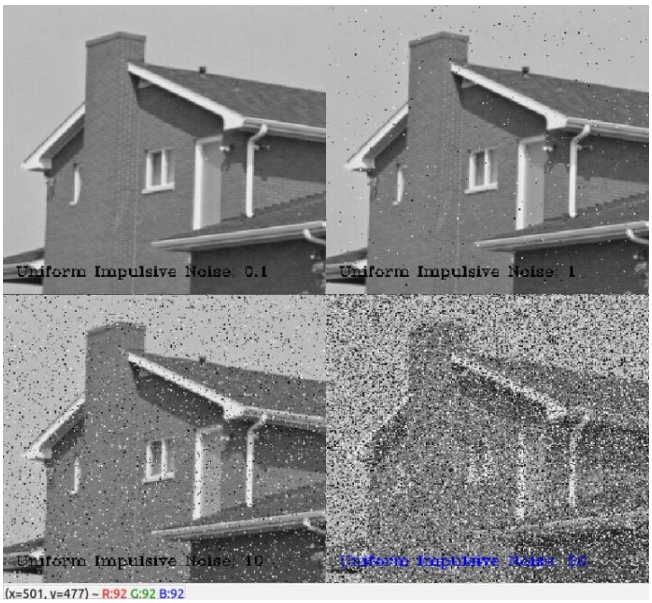


Figura 16 Ruido Impulsivo Uniforme

4.3.2. Imágenes procesadas con el filtro de media (o promediador).

- *Filtrado promedio/media aritmética:* Resultados Óptimos en Ruido Gaussiano : 10% (Véase Figura 17).

Se observa en la figura 17, los valores del PSNR de la imagen con ruido aplicado (IO,R) con el valor de 28.0848 y PSNR de la imagen filtrada con la mediana (IO, F) con el valor de 28.1402, así mismo, los valores de MSE de la imagen con ruido aplicado (IO,R) con 101.064, MSE de la imagen filtrada con la mediana (IO, F) con el valor 99.7842 y finalmente, el MAE de la imagen con ruido aplicado (IO,R) con el valor de 8.02069 y MAE de la imagen filtrada con la mediana (IO, F) con el valor de 6.06782.

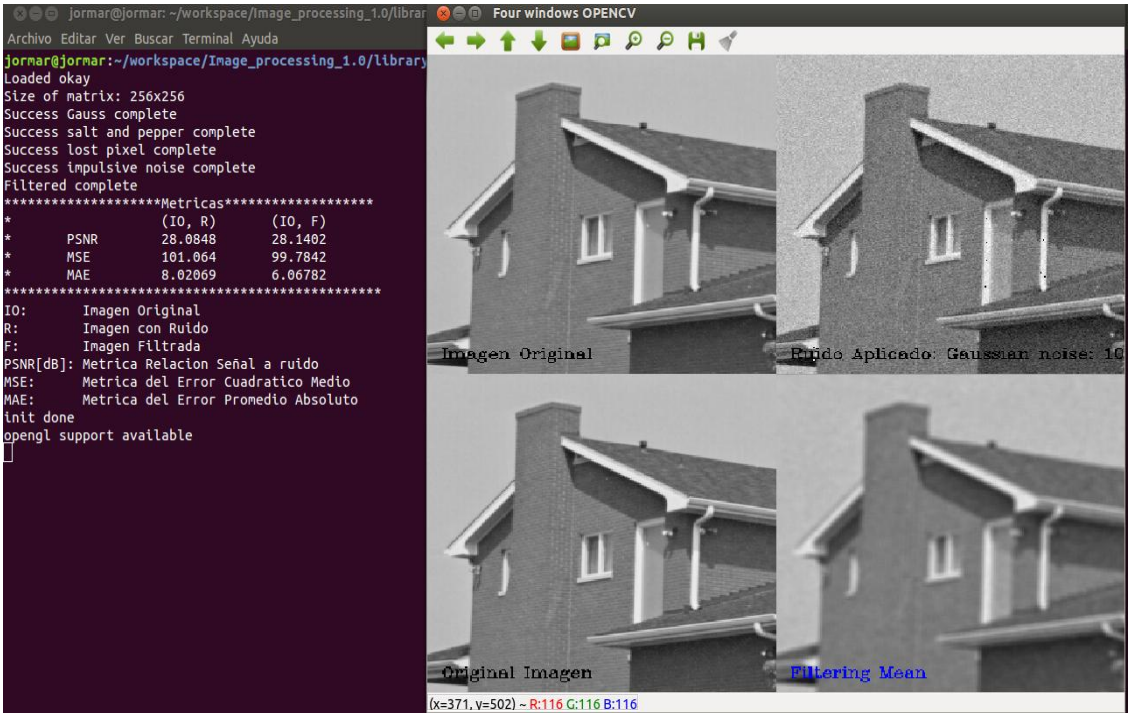


Figura 17 Filtrado promedio/media aritmética

4.3.3. Imágenes procesadas con el filtro de mediana.

- *Filtrado de la mediana:* Resultados Óptimos en Ruido Sal y Pimienta: 5% (Véase Figura 18).

Se observa en la figura 18, los valores del PSNR de la imagen con ruido aplicado (IO,R) con el valor de 18.5228 y PSNR de la imagen filtrada con la mediana (IO, F) con el valor de 30.7731, así mismo, los valores de MSE de la imagen con ruido aplicado (IO,R) con 913.685, MSE de la imagen filtrada con la mediana (IO, F) con el valor 54.4214 y finalmente, el MAE de la imagen con ruido aplicado (IO,R) con el valor de 6.31294 y MAE de la imagen filtrada con la mediana (IO, F) con el valor de 3.81581.

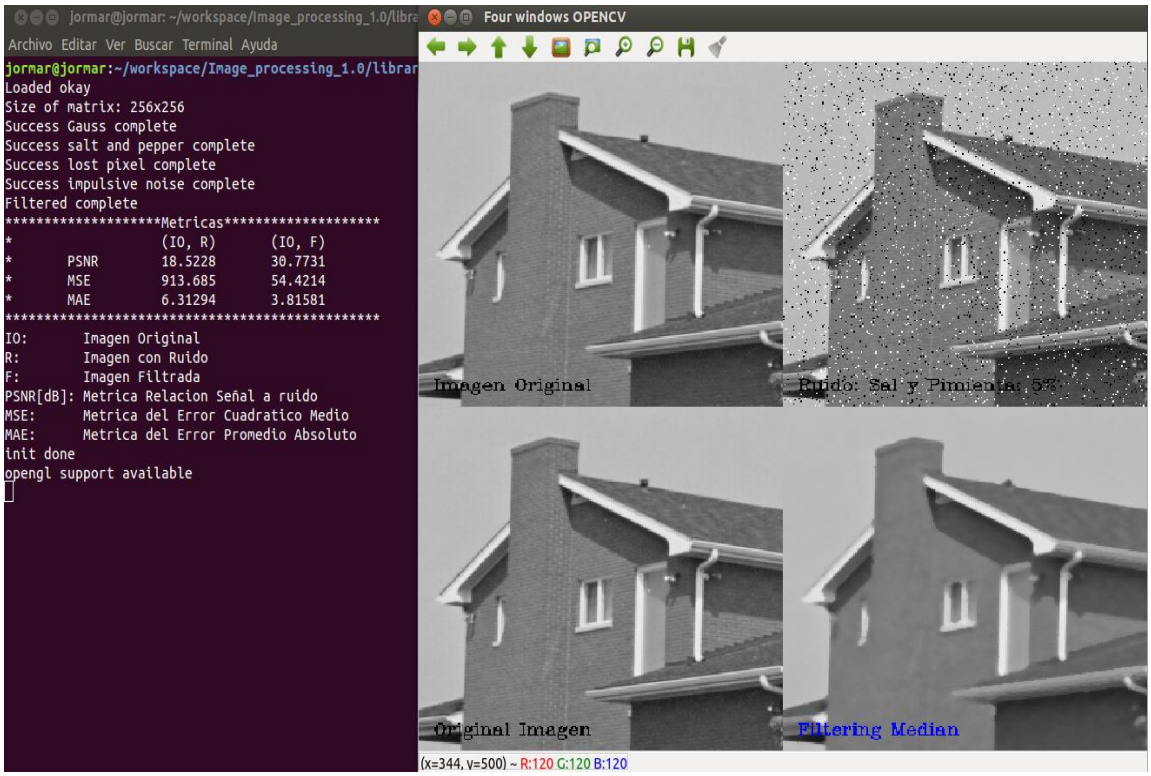


Figura 18 Filtrado de la mediana en Ruido Sal y Pimienta

- *Filtrado de la mediana:* Resultados Óptimos en Ruido Impulsivo Uniforme:5% (Véase Figura 19).

Se observa en la figura 18, los valores del PSNR de la imagen con ruido aplicado (IO,R) con el valor de 22.1686 y PSNR de la imagen filtrada con la mediana (IO, F) con el valor de 30.7828, así mismo, los valores de MSE de la imagen con ruido aplicado (IO,R) con 395.388, MSE de la imagen filtrada con la mediana (IO, F) con el valor 54.3006 y finalmente, el MAE de la imagen con ruido aplicado (IO,R) con el valor de 3.69096 y MAE de la imagen filtrada con la mediana (IO, F) con el valor de 3.85162.

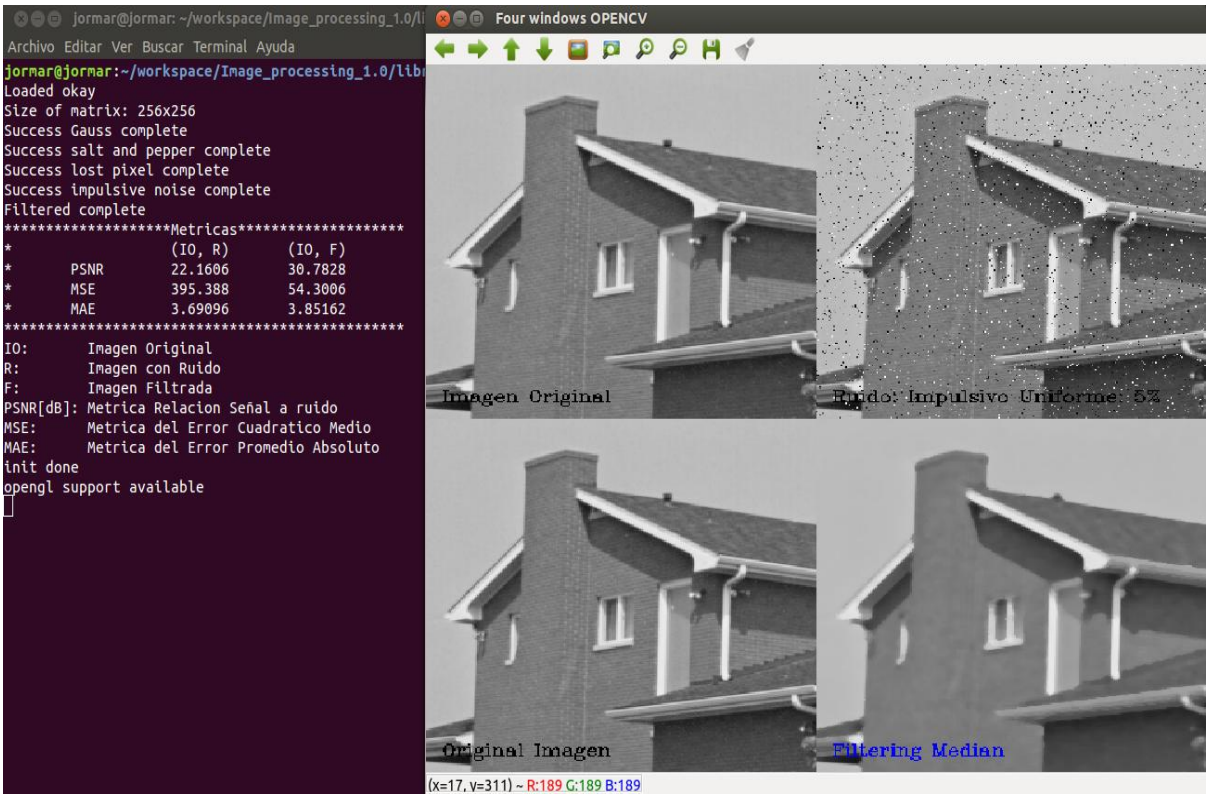


Figura 19 Filtrado de la mediana en Ruido Impulsivo Uniforme

- *Filtrado de la mediana:* Resultados Óptimos en Pérdida de Píxeles: 5% (Véase Figura 20)

Se observa en la figura 20, los valores del PSNR de la imagen con ruido aplicado (IO,R) con el valor de 17.8645 y PSNR de la imagen filtrada con la mediana (IO, F) con el valor de 30.5235, así mismo, los valores de MSE de la imagen con ruido aplicado (IO,R) con 1863.25, MSE de la imagen filtrada con la mediana (IO, F) con el valor 57.6486 y finalmente, el MAE de la imagen con ruido aplicado (IO,R) con el valor de 6.90105 y MAE de la imagen filtrada con la mediana (IO, F) con el valor de 3.98487.

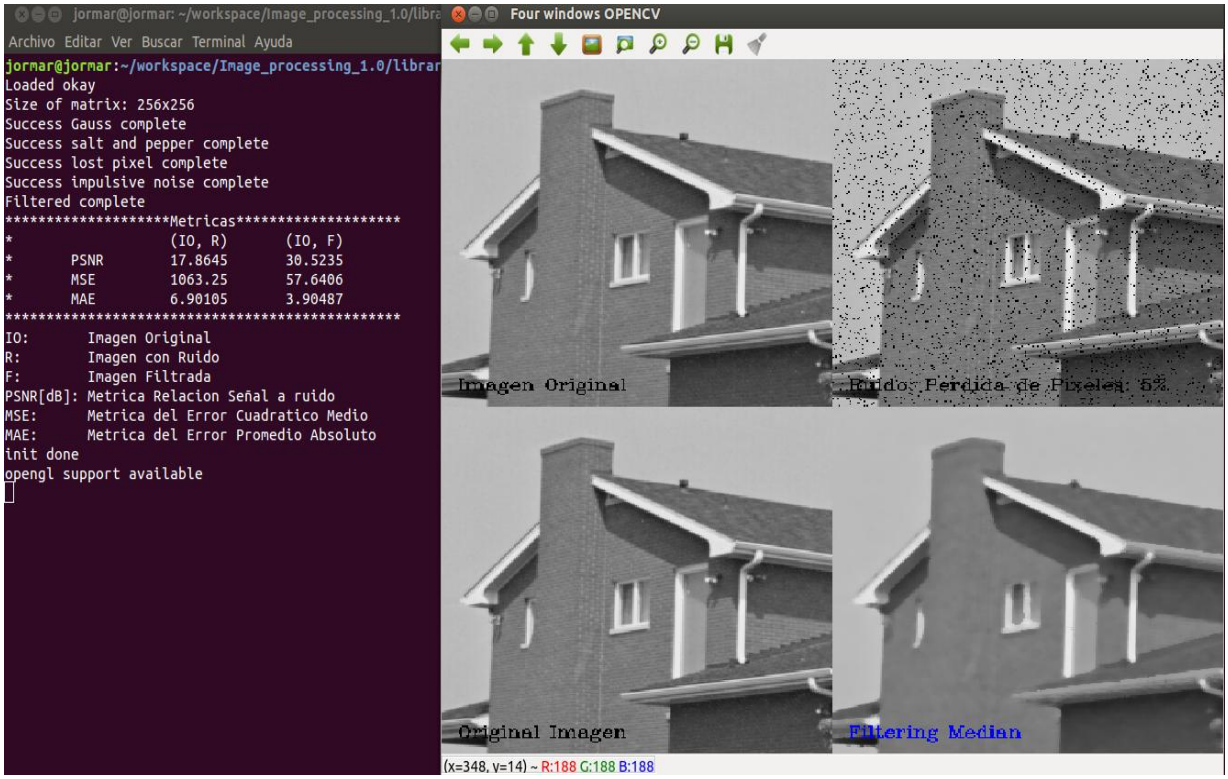


Figura 20 Filtrado de la mediana en Pérdida de Píxeles

4.3.4. Aproximación de bloques no solapados.

En la Figura 21 se observan los Resultados Óptimos en Ruido Gaussiano de 10%

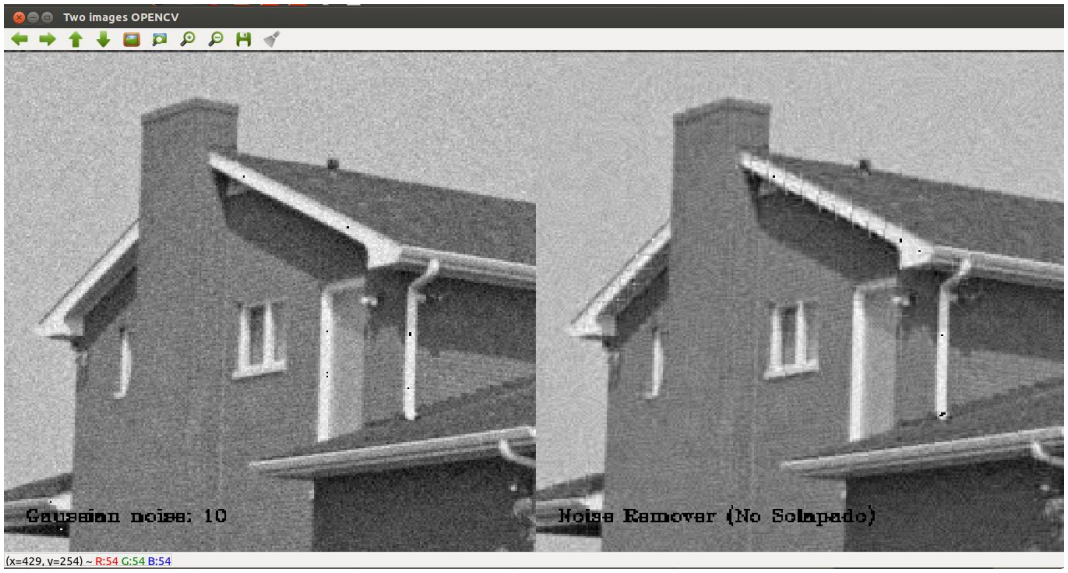


Figura 21 Aproximación de bloques no solapados

4.3.5. Aproximación de bloques solapados.

En la Figura 22 se observan los Resultados Óptimos en Ruido Gaussiano de 10%

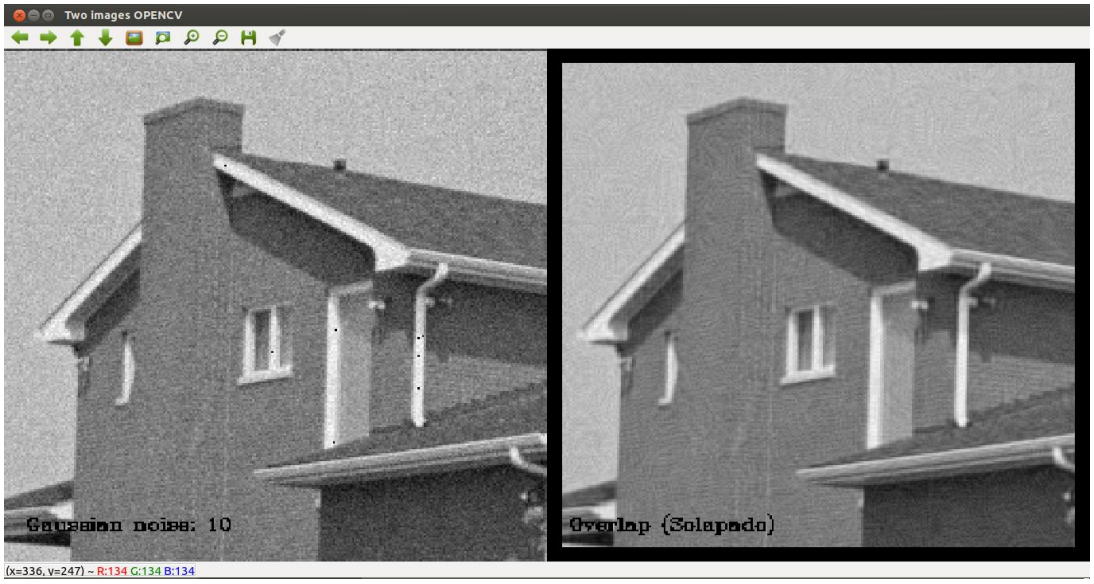


Figura 22 Aproximación de bloques solapados.

4.3.6. Aproximación Robusta de bloques solapados con media

En la Figura 23 se observan los Resultados Óptimos en Ruido Sal y Pimienta de 10%.

Figura 23 Aproximación Robusta de bloque solapados con media

4.3.7. Aproximación Robusta de bloques solapados con mediana

En la Figura 24 se observan los Resultados Óptimos en Ruido Sal y Pimienta de 10%.

Figura 24 Aproximación Robusta de bloque solapados con mediana

4.3.8. Detección de rostros.

Los resultados se observan en la Figura 25.

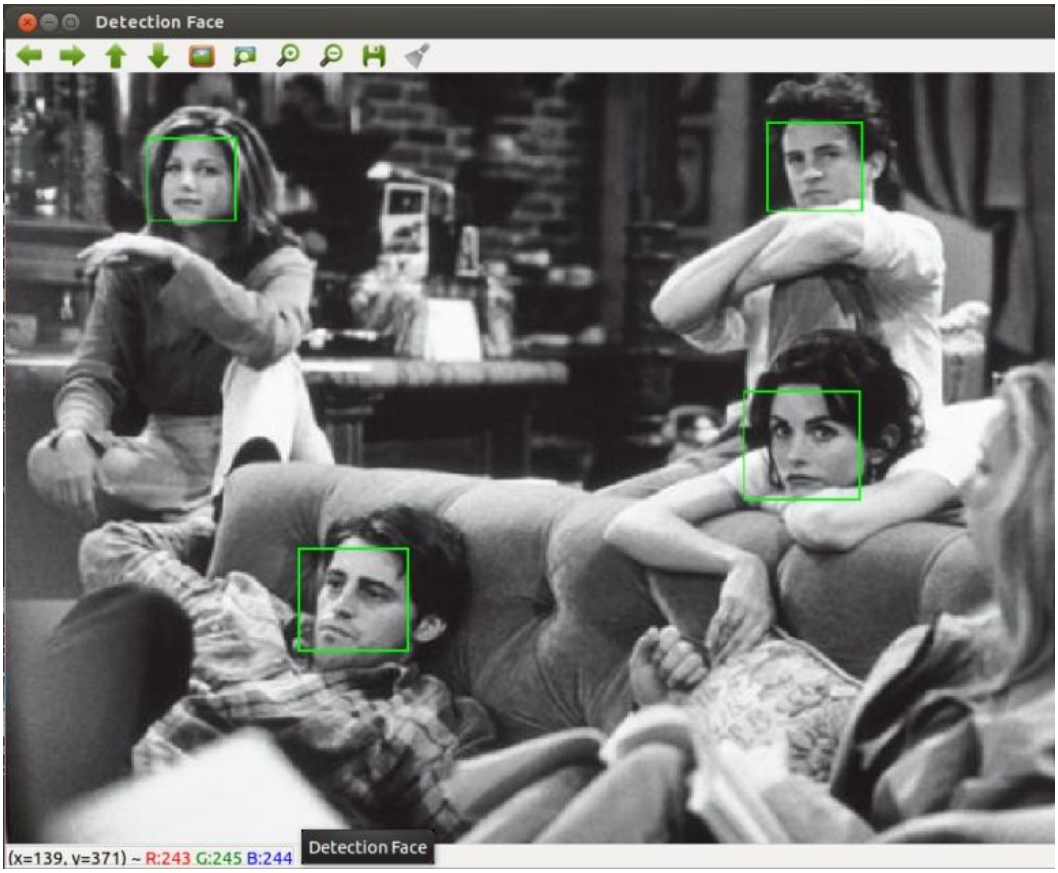


Figura 25 Detección de rostros.

4.4. Mejoras de IP Cemisid 1.0 con respecto a sistemas de procesamiento de imágenes basados en Matlab.

La biblioteca de procesamiento de imágenes IP Cemisid 1.0 demostró ser un avance significativo en comparación con otros sistemas de procesamiento de imágenes basados en Matlab, dado que tiene las siguientes características:

1. Debido a que la biblioteca IP Cemisid 1.0 creada, funciona con software libre, usando Eclipse C++, Armadillo C++, OpenCV C++ y Doxygen.

2. Integra OpenCV C++, proporcionado un manejo de Interfaces de Usuario (GUI[8]) con mejores resultados visuales.

3. Está programada en C++, lo cual mejora el rendimiento computacional y permite a la integración con otras herramientas como Doxygen.

4. La instalación e integración de estas herramientas libres consume menos memoria en disco duro.

5. Demostró mejor eficacia y rendimiento en los métodos de aproximación por bloques, reduciendo valiosamente el tiempo de ejecución y el consumo de memoria.

[8] GUI: Siglas en inglés de Graphical User Interface

Capítulo 5
Conclusiones y Recomendaciones.

5.1. Conclusiones.

- Este proyecto de grado se centró en la creación de una Biblioteca de Procesamiento de Imágenes que incluye métodos que permitieron contaminar imágenes con ruidos impulsivos, filtrarlas con cálculos basados en promedio y mediana, y aproximarlas por bloques no solapados y solapados con cálculos de promedio y mediana para lograr restaurar imágenes afectadas, incluyéndose además un método para la detección de rostros que muestra un alcance hacia otras áreas de estudio.
- En vista de los resultados obtenidos, se concluye que los filtrados basados en cálculos de mediana lograron remover el ruido de imágenes mejor que el filtrado basado en cálculos de promedio (media).

- El Método de Aproximación Robusta Solapado presenta una ventaja sobre el Método de Aproximación Solapado en cuanto a la calidad de imagen pero posee una desventaja de ejecución en el tiempo de cómputo. Ambos métodos son un aporte relevante en el procesamiento de imágenes dado que los sistemas actuales implementados en Matlab tienen un alto costo de cómputo y se quedan sin memoria en la ejecución de cálculos usándose imágenes con menor resolución que las usadas en estas pruebas.
- Los filtrados y aproximaciones evaluados por medio de métricas de calidad de imagen del PSNR, MAE y MSE, demostraron errores reducidos entre píxeles y como consecuencia las imágenes que se obtuvieron son de mejor calidad, porque se aproximaron notoriamente a la imagen original sin ruido.
- Finalmente, la manipulación de la interfaz con OpenCV C++ condujo a una amplia gama de configuración de ventanas que visualizaron las imágenes procesadas.
- Por ende, este proyecto de grado tuvo como fin último documentar con Doxygen todos estos métodos para el manejo y manipulación de las funciones, del mismo modo permitirá nuevos aportes que proveerán mejores soluciones a los efectos de los ruidos contaminantes de las imágenes.

5.2. Recomendaciones.

- Dado que la creada biblioteca IP Cemisid 1.0, funciona con imágenes en escala de grises, sería una excelente contribución extender esta biblioteca con funciones para manejar imágenes a color.

- Elaborar una comparación exhaustiva de rendimiento, mediante tiempos de ejecución entre esta biblioteca IP Cemisid 1.0. y los sistemas de procesamiento de imágenes basados en Matlab.

- Un aporte significativo seria incluir el proceso de Aprendizaje de Diccionarios como KSVD, dentro del método de Aproximación Robusta de Imágenes por Bloques Solapados.

Bibliografía

Berriel R. (2014a). Installing OpenCV 3.0.0 on Ubuntu 14.04. [Website]. Recuperado de: http://rodrigoberriel.com/2014/10/installing-opencv-3-0-0-on-ubuntu-14-04/

Berriel R. (2014b). Using OpenCV 3.0.0 with Eclipse. [Website]. http://rodrigoberriel.com/2014/10/using-opencv-3-0-0-with-eclipse/

Castellanos P. (2016). Libreria Armadillo para Linux (álgebra lineal C++). [Website]. Recuperado de: https://info-2-blog.blogspot.com/2016/02/libreria-armadillo-para-linux-algebra.html?m=1

Detección de rostros. (2017). [Website] Recuperado de: http://acodigo.blogspot.com/2013/06/deteccion-de-rostros.html

Gebel, R. (2012). KL1p: A portable C++ library for Compressed Sensing. [Website] Recuperado de: http://kl1p.sourceforge.net./home.html

Heesch, D. v. (2016). Doxygen: Generate documentation from source code. [Website]. Recuperado de: http://www.stack.nl/~dimitri/doxygen/

How To Install Eclipse Luna on Ubuntu 14.04. (2015). [Website]. Recuperado de: http://idroot.net/tutorials/how-to-install-eclipse-luna-on-ubuntu-14-04/

OpenCV. (2017). [Website]. Recuperado de: http://opencv.org/

OpenCV: Librería de Visión por Computador. (2015). [Website]. Recuperado de: https://osl.ull.es/software-libre/opencv-libreria-vision-computador/

PrinceBalabis. (2014). Integrating Doxygen with Eclipse. [Website]. https://github.com/theolind/mahm3lib/wiki/Integrating-Doxygen-with-Eclipse

Ramírez J. y Paredes J. (2014). Robust Sparse Recovery Base On Weighted Median Operator. *IEEE International Conference on Acoustic, Speech and Signal Processing (ICASSP)*. Department of Electrical Engineering, Universidad de Los Andes, Mérida, Venezuela. Doi: 978-1-4799-2893-4/14/$31.00. Recuperado de: http://www.mirlab.org/conference_papers/International_Conference/ICASSP%202014/papers/p1050-ramirez.pdf

Ramírez J. y Paredes J. (2015). Robust Transforms Based on the Weighted Median Operator. *IEEE Signal Processing Letters,* 22(1), pp. 120 – 124. doi**:** 10.1109/LSP.2014.2349351. Recuperado de: http://ieeexplore.ieee.org/document/6880779/

Ruido y Filtrado. (2015). [Versión electrónica: diapositiva]. Extraído el 10 de noviembre, 2016, de:http://www.dc.uba.ar/materias/t1/2015/c2/archivos/ClaseRuidoFiltrado2C2015.pdf

Sanderson, C. (2010). Armadillo: An Open Source C++ Linear Algebra Library for Fast Prototyping and Computationally Intensive Experiments. *NICTA Technical Report. Australia.* Recuperado de: http://www.conradsanderson.id.au/pdfs/armadillo_nicta_2010.pdf

Sanderson, C., y Curtin R. (2016). Armadillo: a template-based C++ library for linear algebra. *Journal of Open Source Software,* 1, pp. 26. doi: http://dx.doi.org/10.21105/joss.00026 Recuperado de: http://arma.sourceforge.net/armadillo_joss_2016.pdf

Seguí, D. (2012). *Diseño de un interfaz hombre máquina utilizando el entorno MATLAB para la detección y eliminación de ruido impulsivo en imágenes.* (Tesis de pregrado). Universidad Politécnica de Valencia, España. Recuperado: de: https://riunet.upv.es/bitstream/handle/10251/17763/memoria.pdf?sequence=1&isAllowed=y

SOFTENG Agile. (s.f). [Website]. Recuperado de: https://www.softeng.es/es-es/empresa/metodologias-de-trabajo/softeng-agile.html

Tschumperlé, D. (2012). The CImg Library. *IPOL 2012 Meeting on Image Processing Libraries*. Cachan, Francia. Recuperado de: https://tschumperle.users.greyc.fr/publications/tschumperle_ipol2012.pdf

Vettorazzi, J., (2007). *Restauración de imágenes distorsionadas mediante técnicas de procesamiento digital y comparación entre dos métodos de restauración. (*Tesis de pregrado). Universidad de San Carlos de Guatemala. Recuperado de: http://biblioteca.usac.edu.gt/tesis/08/08_0203_EO.pdf

Anexos

Anexo A Repositorio del Proyecto.

En el siguiente enlace https://github.com/jormarsikiu/Image-Processing-Cemisid-1.0- se encuentra alojado el Repositorio del Proyecto: ***Image Processing* Cemisid 1.0** en el sitio web: Github para próximas colaboraciones. (Véase la Figura 26)

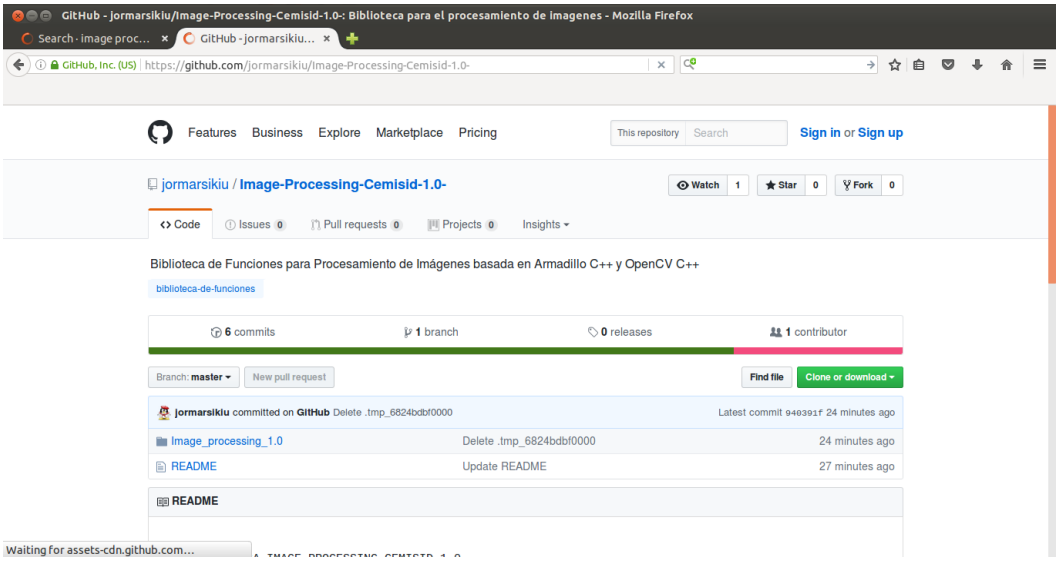


Figura 26 Repositorio del Image Processing 1.0

Anexo B Instalación e Integración de Aplicaciones.

Guías y manuales de uso para el usuario final.

La generación de estos manuales de instalación se efectúa debido a la limitada información en español en la web sobre la integración de Bibliotecas Armadillo C++ y OpenCV C++ en un mismo entorno de programación C++ como es en este caso Eclipse.

1. Biblioteca Armadillo C++ en el entorno Eclipse C++

Instalación e Integración de Eclipse C++. Con Armadillo C++.

1.1. Instalación de Eclipse Neon 4.6 (Ubuntu 15.10)

Según, el sitio web How To Install Eclipse Luna on Ubuntu 14.04 (2015) se puede instalar Eclipse C++ con los pasos que siguen a continuación. (Estos pasos fueron aplicados en las versiones de Eclipse a 4.6 y Ubuntu 15.10).

Entrar al terminal y escriba lo siguiente:

1. $ sudo apt-get update
2. $ sudo apt-get install openjdk-7-jdk
3. Descargar eclipse de la página: http://www.eclipse.org/downloads/eclipse-packages/
4. El paquete: Eclipse IDE for C/C++ Developers
5. Descomprima el archivo de la siguiente manera:
6. $ tar -zxvf -zxvf **eclipse-cpp-neon-1a-linux-gtk.tar.gz** -C /opt
7. Creamos el enlace simbólico para acceder a eclipse:
   $ sudo ln -s /opt/eclipse/eclipse /usr/local/bin/
8. Cree un documento llamado *eclipse.desktop* y guárdelo en */usr/share/applications/*copie dentro lo siguiente:

Archivo eclipse.desktop

```
[Desktop Entry]

Name=Eclipse

Type=Application

Exec=/opt/eclipse/eclipse

Terminal=false

Icon=/opt/eclipse/icon.xpm

Comment=Integrated Development Environment

NoDisplay=false

Categories=Development;IDE;

Name[en]=eclipse.desktop
```

9. Instalamos eclipse en el escritorio:

$ sudo desktop-file-install /usr/share/applications/eclipse.desktop

10. Instalamos el icono de eclipse:

$ sudo cp /opt/eclipse/icon.xpm /usr/share/pixmaps/eclipse.xpm

Nota: Al abrir eclipse y ejecute la actualización de paquetes: plugs-ins está aparece al abrirlo, sino aparece busque: **Help -> Check for updates**

Abrir la pestaña **Help -> Install New Sofware** y añadir

11. Seleccionar lo siguiente **CDT Main Features, CDT Optional Features. (**Véase Figura 27)

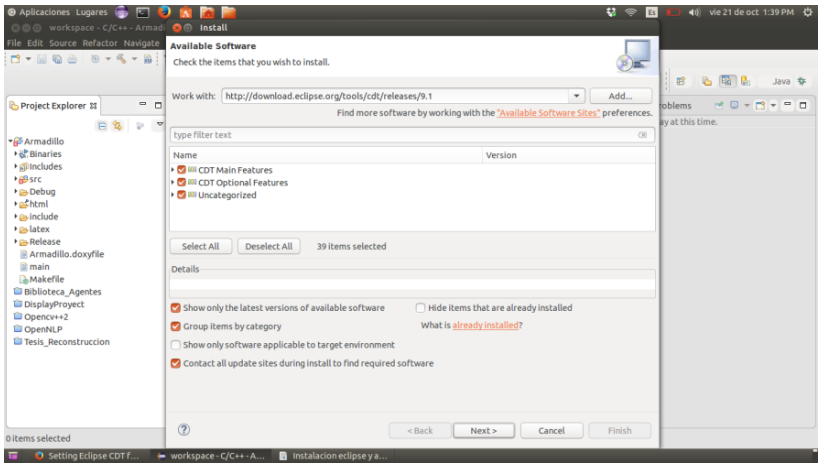
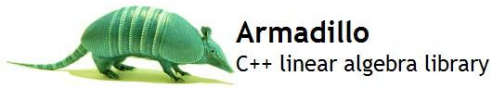


Figura 27 Instalación CDT en Eclipse.

1.2. Instalación de Armadillo C++ 7.800.1 (Ubuntu 15.10):



Entrar al terminal y escribir lo siguiente:

1. $ sudo apt-get install liblapack-dev libblas-dev libboost-dev
2. Descargamos Armadillo C++ de la página oficial de Sanderson, C., y Curtin R. (2016).
3. Descomprimimos el comprimido armadillo-7.800.1.tar.xz
4. $ cd armadillo-7.800.1
5. $ cmake .
6. $ make
7. $ sudo make install

1.3. Integración de Armadillo C++ con Eclipse Neon 4.6

Se extrae de Castellanos P. (2016). Los siguientes pasos:

Paso 1: Abrimos Eclipse:

- Creamos un nuevo proyecto como **"C++ Proyect"**
- Hacemos clic derecho en el nombre del proyecto y seleccionamos **Properties**.
- Buscamos la pestaña ubicada en **C/C++ Build -> Settings -> Includes de GCC C++ Compiler**. (Véase Figura 28).

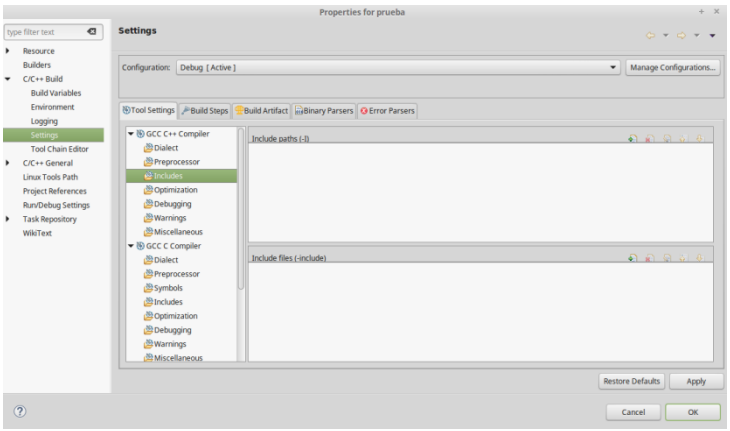


Figura 28 Integración de Armadillo C++

Paso 2: A continuación añadiremos un fichero que contiene la librería de Armadillo:

- Buscamos el apartado "**Include files (-include)**" [parte de abajo]
- Hacemos clic en el icono de añadir fichero y en la ventana tecleamos la siguiente ruta: **/usr/include/armadillo (**Véase Figura 29).

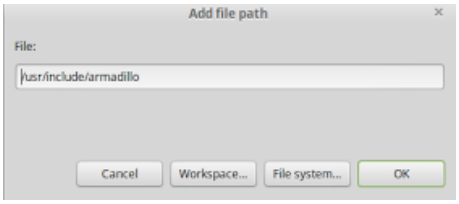


Figura 29 Integración de Armadillo C++

- Al añadirla la veremos en la parte inferior de la siguiente forma: (Véase Figura 30).

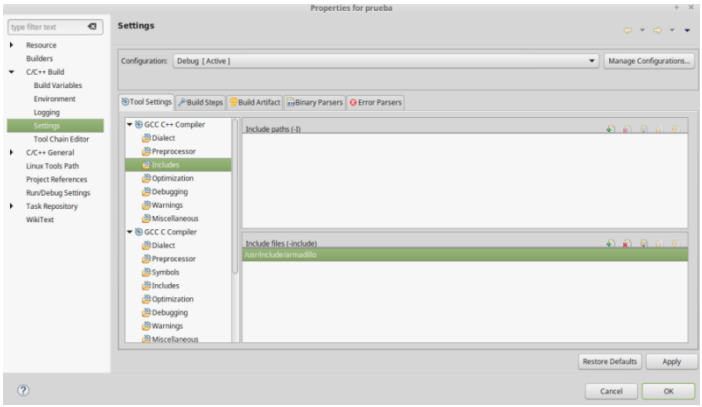


Figura 30 Integración de Armadillo C++

Paso 3:

- Vamos a sección **GCC C++ Linker ->Libraries ->Libraries(-l)**
- Hacemos clic en el icono de añadir y en la ventana tecleamos la palabra: **armadillo** (Véase Figura 31).

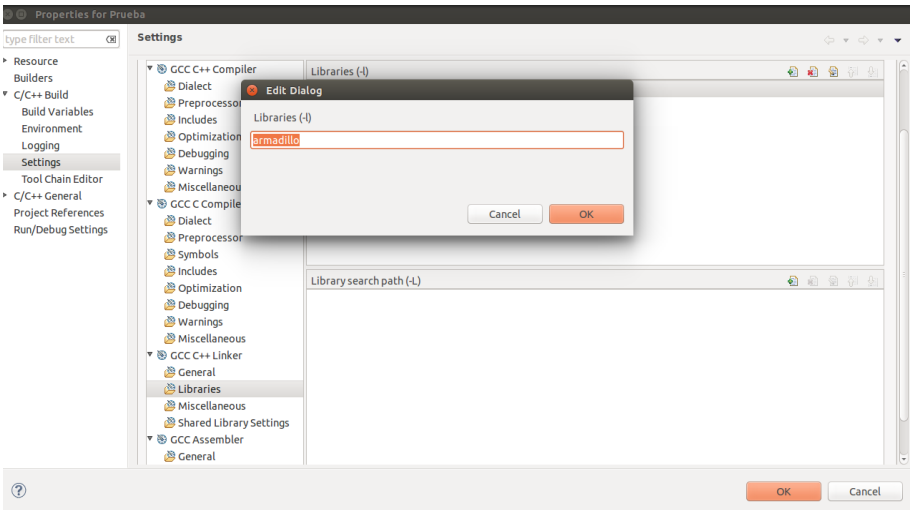


Figura 31 Integración de Armadillo C++

- Al añadirla la veremos en la parte inferior de la siguiente forma: (Véase Figura 32).

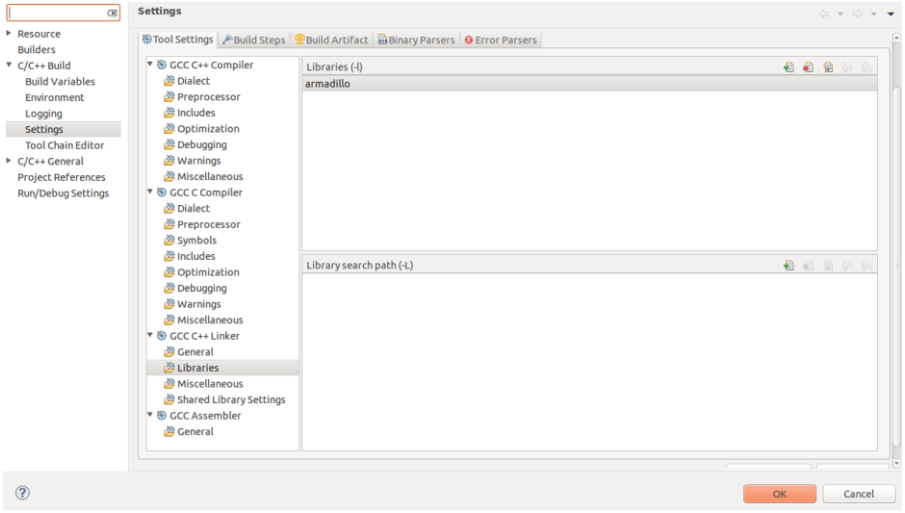


Figura 32 Integración de Armadillo C++

Paso 4: Ir a la pestaña **Optimization**

- Luego donde se encuentra **Optimization Level**
- Cambiar a la opción: **Optimize more (-O2) (**Véase Figura 33).

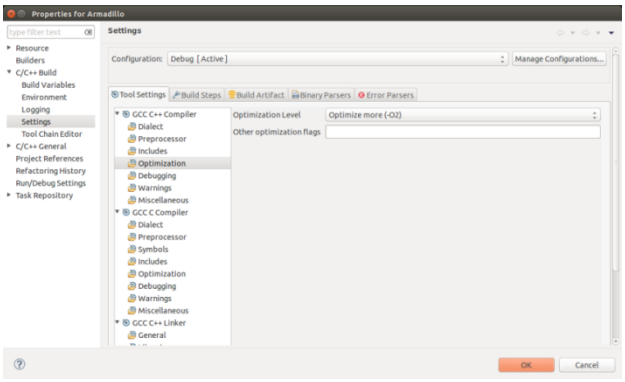


Figura 33 Integración de Armadillo C++

Paso 5

Por último hacemos clic en **Apply y OK.** Ya podemos probar a incluir la librería de Armadillo en nuestro código y compilar nuestro programa para confirmar que se ha indexado correctamente. (Véase Figura 34).

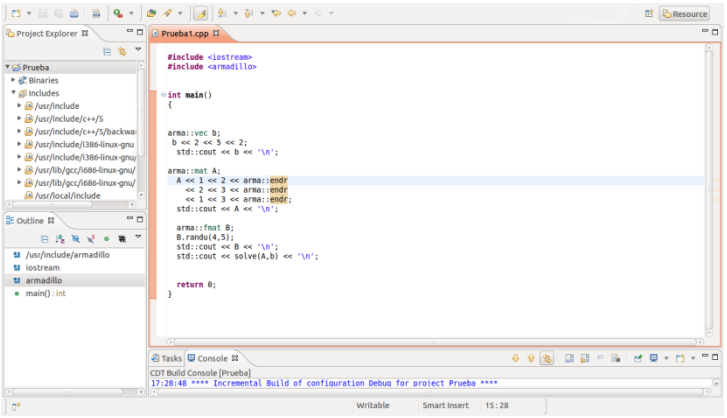


Figura 34 Integración de Armadillo C++

- Compilar por terminal la Biblioteca Armadillo C++

Abrir el terminal y escribimos:

1. $ g++ prog.cpp -o prog -O2 -larmadillo
2. $ ./prog

2. Biblioteca OpenCV C++ en el entorno Eclipse C++

Instalación e integración de OpenCV C++ con Eclipse C++.

2.1. Instalación de OpenCV 3.0.0 (Ubuntu 15.10):

Berriel R. (2014a) en el sitio web Installing OpenCV 3.0.0 on Ubuntu 14.04 propone los siguientes pasos:

Abra el terminal:

1. Actualice el Repositorio:

   $sudo apt update -y && sudo apt upgrade

2. Instale las herramientas para instalar

   $ sudo apt install build-essential cmake

3. Instale los Paquetes de Multimedia necesarios

   $ sudo apt install libjpeg-dev libpng-dev libtiff5-dev libjasper-dev libgdal-dev zlib1g-dev libwebp-dev libv4l-dev libxine2-dev libopencore-amrnb-dev libopencore-amrwb-dev x264 libx264-dev yasm libxvidcore-dev libvorbis-dev libtheora-dev libswscale-dev libavformat-dev libavcodec-dev libdc1394-22-dev libgstreamer-plugins-base0.10-dev libgstreamer0.10-dev libfaac-dev ffmpeg

4. Instale los Paquetes de Parallel frameworks

   $ sudo apt install libtbb2 libtbb2-dbg libtbb-dev libpomp2-dev

5. Instale los Paquetes de Python Support

   $ sudo apt install python-dev python-tk python-numpy python3-dev python3-tk python3-numpy

6. Instale los Paquetes de Oracle JDK

   $ sudo add-apt-repository ppa:webupd8team/java

   $ sudo apt update

$ sudo apt install oracle-java8-installer

$ sudo apt install ant

7. Instale los Extras

   $ sudo apt install libeigen3-dev doxygen libgl1-mesa-dev libglu1-mesa-dev mesa-common-dev qt5-default libatlas-base-dev gfortran

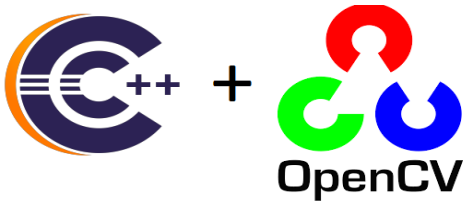
8. Descargar opencv-3.0.0 y descomprimir

   - $ cd **opencv-3.0.0-alpha**
   - $ mkdir build
   - $ cd build
   - $ cmake -D CMAKE_BUILD_TYPE=RELEASE -D CMAKE_INSTALL_PREFIX=/usr/local -D WITH_TBB=ON -D WITH_V4L=ON -D WITH_QT=ON -D WITH_OPENGL=ON ..
   - $ make -j $(nproc)
   - $ sudo make install
   - $ sudo /bin/bash -c 'echo "/usr/local/lib"> /etc/ld.so.conf.d/opencv.conf'
   - $ sudo ldconfig

9. Probar si está instalado correctamente:

   - $ cd /opencv-3.0.0/samples/
   - $ sudo cmake .
   - $ sudo make -j $(nproc)
   - $ cd /opencv-3.0.0/samples/cpp/
   - Buscar en la carpeta ../data/lena.jpg y copiar la imagen en la carpeta cpp
   - Luego correrlo:
   - $./cpp-example-facedetect lena.jpg

2.2. Integración de OpenCV 3.0.0 con Eclipse Neon



Berriel R. (2014b) en el sitio web Using OpenCV 3.0.0 with Eclipproponemos los siguientes pasos:

Crear un proyecto nuevo: C++ en Eclipse

1. **File -> New -> C++ Project**
2. Escoja el nombre del proyecto: **Project Name**
3. Escoja el Executable**: Empty Project** in **Project Type**
4. Escoja: **Linux GCC**
5. Click a Finish;

Enlazar OpenCV con el proyecto

1. Seleccione el proyecto y busque en el menu: Properties (o presione Alt+ENTER)
2. Buscar: C/C++ Build -> Settings
3. Buscar: GCC C++ Compiler -> Includes
4. Escriba en el terminal: pkg-config --cflags opencv
5. Añada en Include paths (-l) del terminal: /usr/local/include/opencv
6. Buscar: GCC C++ Linker -> Libraries
7. Escriba en el terminal: pkg-config --libs opencv
8. Añada a Library search paths (-L) del terminal: /usr/local/lib
9. En GCC C++ Linker -> Libraries
1. Añada las bibliotecas que necesarias para el proyecto en Libraries (-l).
2. Usaremos:
    a. opencv_core
    b. opencv_imgcodecs
    c. opencv_highgui
3. Listo ya está instalado OpenCV

3. Doxygen en el entorno Eclipse C++

Instalación e integración de Doxygen en el entono Eclipse C++

3.1. Instalación de Doxygen en Ubuntu (15.10)

PrinceBalabis (2014) propone en su sitio web llamado Integrating Doxygen with Eclipse los siguientes pasos para la instalación de la aplicación:

1. Descargar la versión Linux Doxygen. Es un archivo .tar.gz , desde la página: http://www.stack.nl/~dimitri/doxygen/download.html

   *A binary distribution for **Linux x86-64.** (Véase la Figura 35).

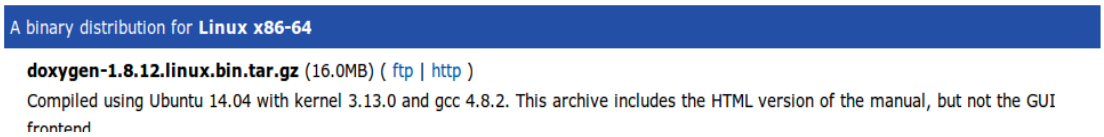


Figura 35 Versión Linux Doxygen.

2. Abra el terminal:

- tar -xzvf doxygen-1.8.12.linux.bin.tar.gz
- cd doxygen-version
- ./configure
- make install
- Si observa el siguiente error:
  /usr/bin/install: cannot stat 'bin/doxytag': No such file or directory
  Puede leer acerca de eso aca:
  http://stackoverflow.com/questions/15020691/installation-of-doxygen-and-error-in-make-command
- Listo ya se encuentra instalado Doxygen

3.2. Integración de Doxygen con Eclipse C++

- Abra Eclipse y busque la pestaña **Help -> Install New Software.** (Véase la Figura 36).

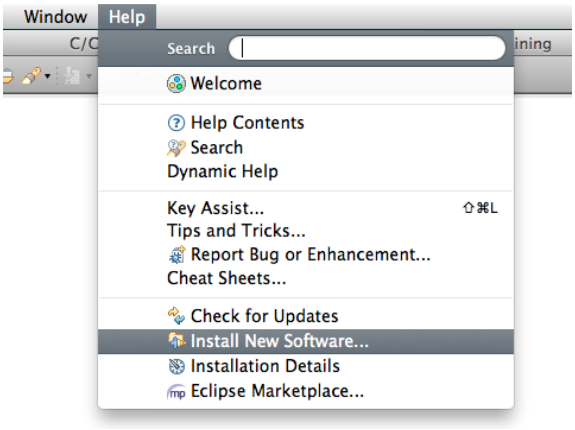


Figura 36 Integración de Doxygen con Eclipse

- **Escriba en la ventana "Work with:"**: y marque "eclox" y presione **"Next".** (Véase la Figura 37).

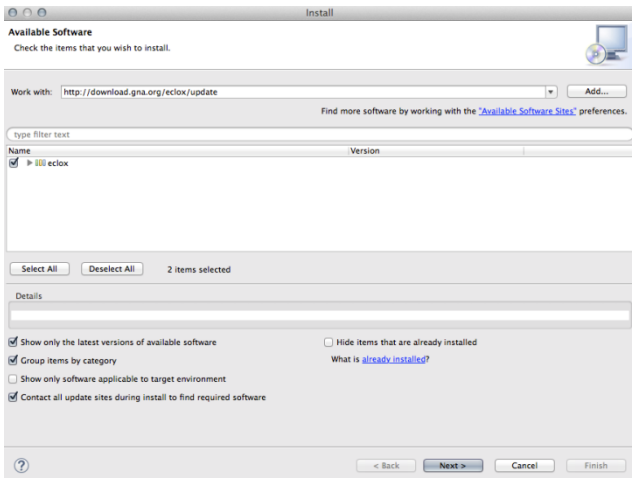


Figura 37 Integración de Doxygen con Eclipse

- Presione **"Next"** y luego**"Finish"**

- Si aparece lo siguiente presione "OK"**.** (Véase la Figura 37).

Figura 38 Integración de Doxygen con Eclipse

- Si sale lo siguiente presione "Yes" para reiniciar eclipse y aplicar los cambios. (Véase la Figura 39).

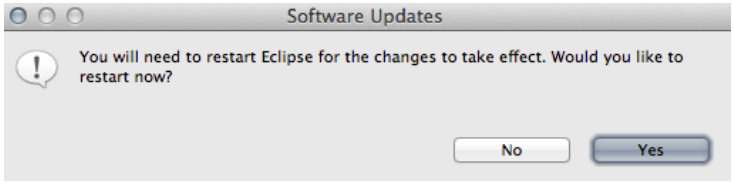


Figura 39 Integración de Doxygen con Eclipse

- De nuevo en eclipse, vaya a la pestaña **"Window -> Preferences -> Doxygen"**
- Agregamos la ubicación de Doxygen. Add "/usr/local/bin"
- Presione añadir directorio, luego**"Apply"**, y despues **"OK"**
- Listo reiniciar eclipse y ya se encuentra integrado Doxygen con Eclipse.

- ## Añadir Doxygen en el Proyecto C++

1. Primero se debe crear un New Proyect en Eclipse: **File -> C++ Proyect**
2. Presione el botón  . Si aparece lo siguiente presione**"Yes"**. Si no omitir este paso. (Véase la Figura 40).

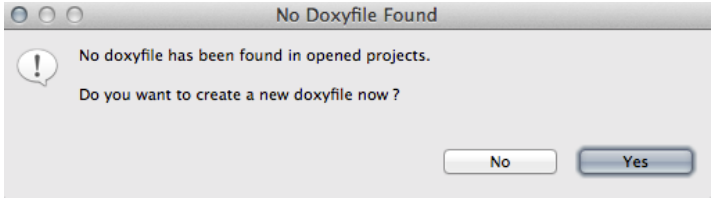


Figura 40 Añadir Doxygen en el Proyecto C++

NOTA: Si abre el proyecto y ya tiene un archivo .doxyfile en su carpeta solo presionar la flecha del icono  y seleccionar el proyecto. Si va a generar uno nuevo siga el paso 3.

3. Se debe configura la ubicación de Doxygen. De la siguiente manera: (Véase la Figura 41).

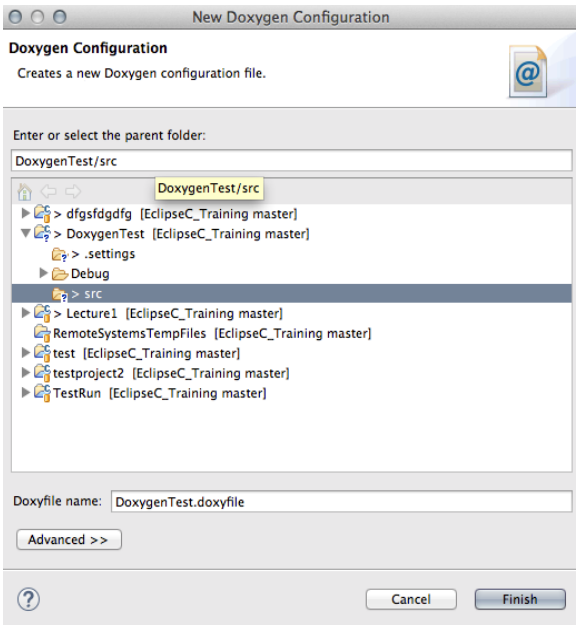


Figura 41 Añadir Doxygen en el Proyecto C++

4. Presione doble click el archivo *DoxygenTest.doxyfile* que ha creado en el proyecto. Observara una ventana como la siguiente. En este paso se configura Doxygen. (Véase la Figura 42).

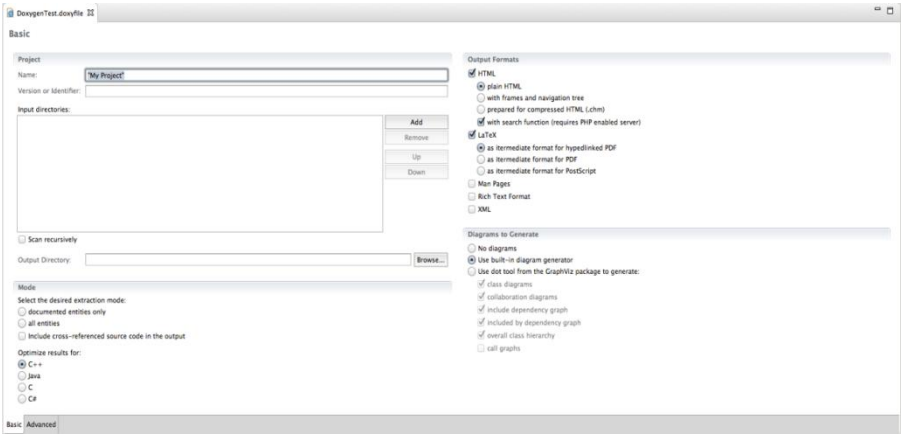


Figura 42 Añadir Doxygen en el Proyecto C++

5. Añada un nombre del documento en *Name y la Version.*(Véase la Figura 43).

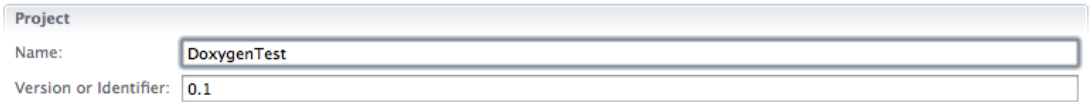


Figura 43 Añadir Doxygen en el Proyecto C++

6. Elija la entrada y salida del directorio del proyecto donde quieres que se ubique el documento "Input Directory" y "Output Directory"Seleccione Add y Browsey añada la ubicación. (Véase la Figura 44).
   *El punto indica que se encuentra en la ubicación actual del proyecto.

Figura 44 Añadir Doxygen en el Proyecto C++

7. Otras opciones:
8. Para generar documentación de todos los archivos seccione:

   Mode -> Select the desired extraction mode ->all entities(Véase la Figura 45).

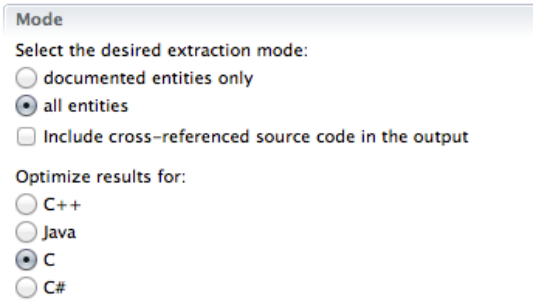


Figura 45 Añadir Doxygen en el Proyecto C++

9. Configuración de la salida de la documentación: "Output Formats". Se puede generar la documentación en HTML-docs, Latex y XML. Seleccione la requerida. (Véase la Figura 46).

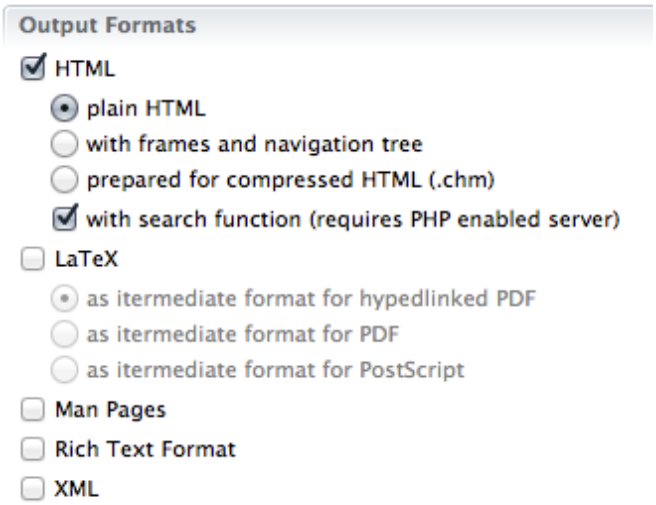


Figura 46 Añadir Doxygen en el Proyecto C++

10. Guarde la configuración y presione el botón  . Elija el archivo .doxyfile y presione "OK". Listo se genero la documentación de Doxygen.

11. Si se seleccionó la documentación "html". Presione el archivo con doble click para abrir la documentación.

12. Presione el botón  para generar o actualizar el documento del proyecto.

4. Compilación y ejecución de la biblioteca IP Cemisid 1.0

Otra manera de compilar y ejecutar sin usar Eclipse es usando el Makefile incluido en el repositorio. Para más información puede referirse al video: https://youtu.be/7nDnuBZ_M1s