

IMAGE IMPROVEMENT USING CRIMMINS SPECKLE REDUCTION OF NOISE INTENSITY

**Submitted in partial fulfillment of the requirements
for the award of Bachelor of Engineering degree
in Computer Science and Engineering**

by

MANITEJA REDDY TOORPU (Reg. No. 37110430)

GUNDALA VISHNU VARDHAN (Reg. No. 37110253)



**DEPARTMENT OF COMPUTER SCIENCE AND
ENGINEERING SCHOOL OF COMPUTING**

SATHYABAMA

**INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)**

**Accredited with Grade "A" by NAAC JEPPIAAR NAGAR, RAJIV
GANDHI SALAI, CHENNAI – 600 119**

MARCH 2021



SATHYABAMA

**INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)**

Accredited with “A” grade by NAAC

Jeppiaar Nagar, Rajiv Gandhi Salai, Chennai – 600 119

www.sathyabama.ac.in



DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

BONAFIDE CERTIFICATE

This is to certify that this project report is the bonafide work of Maniteja Reddy Toorpu (Reg. No. 37110430) and Gundala Vishnu Vardhan (Reg. No. 37110253) who carried out the project entitled “Image Improvement Using Crimmins Speckle Reduction Of Noise Intensity” under my supervision from December 2020 to March 2021.

Internal Guide
Dr. R.M. GOMATHI, M.E., Ph.D.,

Head of the Department

Submitted for Viva voce Examination held on _____

Internal Examiner

External Examiner

DECLARATION

I Maniteja Reddy Toorpu hereby declare that the Project Report entitled “Image Improvement Using Crimmins Speckle Reduction Of Noise Intensity” is done by me and my partner Gundala Vishnu Vardhan under the guidance of Dr. R.M.Gomathi, M.E., Ph.D., Department of Computer Science and Engineering at Sathyabama Institute of Science and Technology is submitted in partial fulfillment of the requirements for the award of Bachelor of Engineering degree in Computer Science and Engineering.

DATE:

PLACE: CHENNAI

SIGNATURE OF THE CANDIDATE

ACKNOWLEDGEMENT

I am pleased to acknowledge my sincere thanks to Board of Management of SATHYABAMA Sathyabama Institute of Science and Technology is for their kind encouragement in doing this project and for completing it successfully. I am grateful to them.

I convey my thanks to Dr. T. Sasikala, M.E., Ph.D., Dean, School of Computing, Dr. S. Vigneshwaril, M.E., Ph.D., and Dr. L. Lakshmanan, M.E., Ph.D., Heads of the Department of Computer Science and Engineering for providing me necessary support and details at the right time during the progressive reviews.

I would like to express my sincere and deep sense of gratitude to my Project Guide Dr. R.M. Gomathi, M.E., Ph.D., for her valuable guidance, suggestions and constant encouragement paved way for the successful completion of my project work.

I wish to express my thanks to all Teaching and Non-teaching staff members of the Department of Computer Science and Engineering who were helpful in many ways for the completion of the project.

ABSTRACT

Image noise is an undesirable by-product of image capture that obscures the desired information. The noise in an image can be reduced by using various techniques. Techniques like mean filtering, median filtering & Gaussian filtering reduces the noise, but they increase the blurriness of the image. Techniques like conservative smoothing and crimmins speckle removal, reduce the noise with out much blurriness in the image. Canny edge removal is more useful for identifying the nerves by removing the noise in medical images. This project aims at applying all the above mentioned noise removal techniques, and, observe how these techniques works. The project comprises of four modules. The first module deals with the application of mean & median filters. The second module deals with Gaussian filter and conservative smoothing. The third module deals with Crimmins Speckle Reduction and final module deals with the nerves identification using Canny edge removal. Image filters can be used to reduce the amount of noise in an image and to enhance the edges in an image. There are two types of noise that can be present in an image: speckle noise and salt-and-pepper noise. Speck noise is the noise that occurs during image acquisition while salt-and-pepper noise (which refers to sparsely occurring white and black pixels) is caused by sudden disturbances in an image signal. Enhancing the edges of an image can help a model detect the features of an image.

CONTENTS

CHAPTER NO.	TITLE	PAGE NO.
	ABSTRACT	V
	LIST OF FIGURES	VIII
1	INTRODUCTION	1
	1.1 INTRODUCTION	1
	1.2 SYSTEM ANALYSIS	1
	1.2.1 HARDWARE REQUIREMENTS	1
	1.2.2 SOFTWARE REQUIREMENTS	2
	1.2.3 FEASIBILITY ANALYSIS	2
	1.2.4 ECONOMICAL ANALYSIS	2
	1.2.5 TECHNICAL ANALYSIS	2
	1.3 ADVANTAGES	3
	1.3.1 TECHNICAL ADVANTAGES	3
2	AIM AND SCOPE	4
	2.1 AIM OF THE PROJECT	4
	2.2 SCOPE	4
	2.3 OBJECTIVE	4
3	SYSTEM DESIGN	5
	3.1 SYSTEM DESIGN	5
	3.2 SCREENSHOT AND DESCRIPTION	5
	3.3 USE CASE DIAGRAM	7
	3.3.1 DESCRIPTION OF USE CASE DIAGRAM	7
	3.4 SEQUENCE DIAGRAM	7
	3.4.1 DESCRIPTION OF SEQUENCE DIAGRAM	8
	3.5 MEAN FILTERING	8
	3.5.1 ALGORITHM DESCRIPTION	8
	3.6 GAUSSIAN FILTERING	9
	3.7 CRIMMINS SPECKLE REMOVAL	10
	3.8 CANNY EDGE REMOVAL	12
	3.8.1 DEVELOPMENT OF CANNY ALGORITHM	12
	3.8.2 PROCESS OF CANNY EDGE EDETECTION ALGORITHM	13

4	IMPLEMENTATION	14
	4.1 FLOW DIAGRAM	14
	4.2 PYTHON	15
	4.2.1 FEATURES AND PHILOSOPHY	15
	4.3 MODULE IMPORTED	15
	4.3.1 OS MODULE	15
	4.3.2 OS FUNCTIONS	15
	4.3.3 STS MODULE	17
	4.4 NUMPY	17
	4.5 TOOLS USED	19
	4.5.1 QPUSH BUTTON	20
	4.5.2 QLINEEDIT	22
	4.5.3 QMAINWINDOW	23
	4.5.4 CREATING MENU	24
	4.5.5 CREATING TOOLBAR	25
	4.6 SOFTWARE TESTING	26
	4.6.1 VERIFICATION	26
	4.6.2 VALIDATION	26
	4.6.3 BLACK BOX TESTING	26
	4.6.4 WHITE BOX TESTING	26
	4.6.5 TYPES OF TESTING	26
5	RESULTS AND DISCUSSION	28
	5.1 MEAN FILTERING	28
	5.2 GAUSSIAN FILTERING	28
	5.3 CRIMMINS SPECKLE REMOVAL	30
	5.4 CANNY EDGE	31
6	CONCLUSION AND FUTURE WORK	32
	6.1 CONCLUSION	32
	6.2 FUTURE WORK	32
	REFERENCES	33

LIST OF FIGURES

FIGURE No.	FIGURE NAME	PAGE No.
3.1.1	DATA FLOW DIAGRAM	5
3.2.1	SCREEN SHOTS SHOWING THE FILES CREATED DURING THE PROJECT	5
3.2.2	SIX PUSH BUTTONS	6
3.3.1	UML DIAGRAM	7
3.4.1	SEQUENCE DIAGRAM	8
3.6.1	SHAPE OF THE IMPULSE RESPONSE OF A TYPICAL GAUSSIAN FILTER	10
4.1.1	FLOW DIAGRAM	14
4.5.1	DESIGNER SCREENSHOT 1	19
4.5.2	DESIGNER SCREENSHOT 2	20
4.5.3	UI FILES	20
4.5.3.1	QMAIN WINDOW LAYOUT DIAGRAM	23
5.1.1	MEAN FILTERING	28
5.2.1	GAUSSIAN FILTERING	29
5.3.1	CONSERVATIVE SMOOTHING	30
5.4.1	CRIMMINS SPECKLE REMOVAL	31
5.5.1	CANNY EDGE	31

CHAPTER 1

INTRODUCTION

1.1 INTRODUCTION

Image noise is an undesirable by-product of image capture that obstruct the desired information or Image noise is random variation of brightness or color information in images, and is usually an aspect of electronic noise. It can be produced by the image sensor and circuitry of a scanner or digital camera. Image noise can also originate in film grain and in the unavoidable shot noise of an ideal photon detector. Noise in an image can be reduced by using various techniques. Techniques like mean filtering, median filtering & Gaussian filtering reduces the noise, but they increase the blurriness of the image. Techniques like conservative smoothing and crimmmins speckle removal, reduce the noise with out much blurriness in the image.

There are various techniques for reducing noise they are:

1. Mean filtering.
2. Mean filtering and Gaussian filtering .
3. Conservative smoothing
4. crimmmins speckle removal.
5. Canny edge.

The mean filter is used to blur an image in order to remove noise. It involves determining the mean of the pixel values within a $n \times n$ kernel. The pixel intensity of the center element is then replaced by the mean. This eliminates some of the noise in the image and smooths the edges of the image. The Gaussian Filter is similar to the mean filter however it involves a weighted average of the surrounding pixels and has a parameter sigma. The kernel represents a discrete approximation of a Gaussian distribution. While the Gaussian filter blurs the edges of an image (like the mean filter) it does a better job of preserving edges than a similarly sized mean filter.

1.2 SYSTEM ANALYSIS

1.2.1 HARDWARE REQUIREMENTS

1. It requires a minimum of 2.16 GHz processor.
2. It requires a minimum of 4 GB RAM.
3. It requires 64-bit architecture.
4. It requires a minimum storage of 500GB.

1.2.2 SOFTWARE REQUIREMENTS

1. It requires a 64-bit Ubuntu Operating System.
2. Python Qt Designer for designing user interface.
3. MY SQL server for storing database Entities.
4. Pyuic for converting the layout designed user interface (UI) to python code.
5. Python language for coding.

1.2.3 FEASIBILITY ANALYSIS

As the name implies, a feasibility study is used to determine the viability of an idea, such as ensuring a project is legally and technically feasible as well as economically justifiable. It tells us whether a project is worth the investment—in some cases, a project may not be doable.

1.2.4 ECONOMICAL ANALYSIS

This assessment typically involves a cost/ benefits analysis of the project, helping organizations determine the viability, cost, and benefits associated with a project before financial resources are allocated. It also serves as an independent project assessment and enhances project credibility. Our project is economically feasible because in this we have used “UBUNTU”, “PYTHON”, “PYQT” designer tool and “PYUIC” which are all available as an open source.

1.2.5 TECHNICAL ANALYSIS

This assessment focuses on the technical resources available to the organization. It helps organizations determine whether the technical resources meet capacity. Technical feasibility also involves evaluation of the hardware, software, and other technology requirements of the proposed system. A prototype of the tool was developed to verify the technical feasibility. The prototype is working successfully and hence the project is feasible.

1.3 ADVANTAGES

1. An image pre-processing step can improve the accuracy of machine learning models. Pre-processed images can help a basic model achieve high accuracy when compared to a more complex model trained on images that were not pre-processed.
2. Very useful to the police personnel in improving the quality of the CCTV images, so that culprits can be accurately identified.
3. This project finally leads to the improvement of quality of the investigation of criminal cases.

1.3.1 TECHNICAL ADVANTAGES

1. Using latest Python's Image Filter tool to reduce the noise, and to improve the quality of the images.
2. Using Python, which is chosen as the best programming language, by the Programming Community.
3. More Functionality can be implemented with less no. of lines of code in Python.
4. PyQt tool is used to create the Graphical User interfaces.
5. All the Front end code is generated automatically by PyUIC.

CHAPTER 2

AIM AND SCOPE

2.1 AIM OF THE PROJECT

The project aims at developing a tool for Image improvement using Crimmins Speckle Reduction of noise intensity, using four modules

- (1) Mean filtered image, Median filtered image.
- (2) Gaussian Filtered image, Image using conservative smoothing.
- (3) Output image of Crimmins Speckle Reduction.
- (4) Nerves image produced using canny edge removal.

2.2 SCOPE

As of now, the system is currently working with the downloaded images. The applicability of this project, in the real life scenario, using CCTVs, is yet to be explored. Very useful to the police personnel in improving the quality of the cctv images, so that culprits can be accurately identified.

2.3 OBJECTIVE

1. A restoration application developed for medical images and analysis over parameters for performance of various filters.
2. Speckle: This is granual noise that intensity exists and degrades the quality of active radar ,synthetic aperture radar,medical images.
3. This tool is developed by using python along with its layout toolkit PyQt,PyUIC and image filter.

CHAPTER 3

SYSTEM DESIGN

3.1 SYSTEM DESIGN

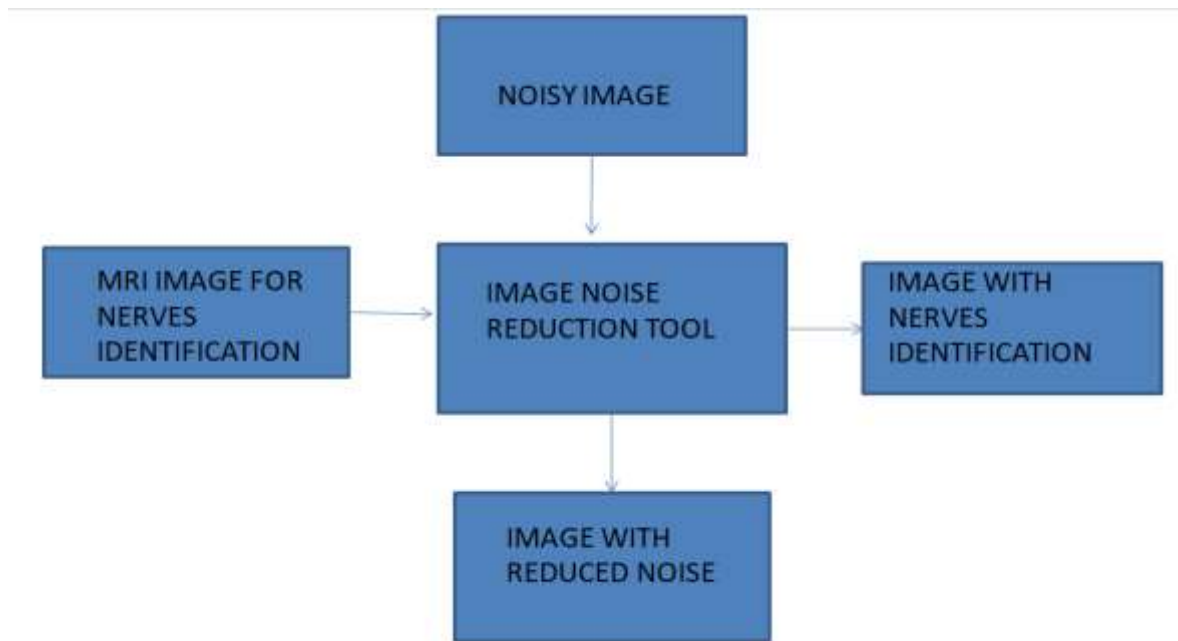


Fig.3.1.1 Data flow diagram

Inputs like Noisy Image as well as the MRI image for nerves identification are to be provided as Input to the system, using the corresponding user interfaces. The following section describe the screen shots of these user interfaces.

Fig.3.1 Data flow diagram

3.2 SCREENSHOT AND DESCRIPTION



Fig.3.2.1 Screen shot showing the files created during the project

The above screen shot shows different files created during this projects. There are four different types of files: (1) .py files (2) .ui file (3).txt file and (4).png file

1. .ui files are the user interface files, created by using PyQt layout editor
2. .py files are python program files created either manually, or automatically. For instance, each .ui file has a corresponding .py file that is created automatically by using the PyUIC tool. .txt files contains the generic useful information about the project.
3. crimmins11.py, is the entry program for this project. Execution of this python program leads to the entry screen as follows:

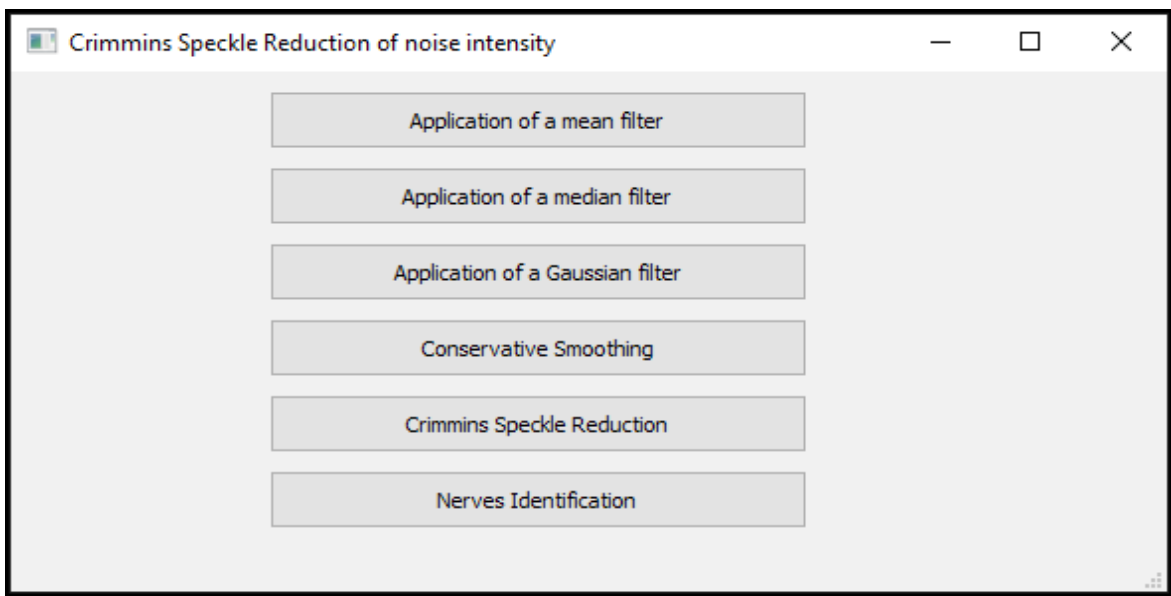


Fig.3.2.2 Six Push Buttons

This entry screen consists of six push buttons. Upon clicking the first button, crimm1.py program is instantiated resulting in the mean filtering of the image, showing the mean filtered image along with the original image. The second button leads to the invoking of crimm2.py program, resulting in the median filtering of the image, showing the median filtered image along with the original image. Upon clicking the third button, crimm3.py program is instantiated resulting in the Gaussian filtering of the image, showing the gaussian filtered image along with the original image. Upon clicking the fourth button, Conservative filtering is applied on the given image. The fifth button leads to the

Crimmins Speckle removal, and the sixth button leads to the canny edge detection of nerve identification.

3.3 USE CASE DIAGRAM

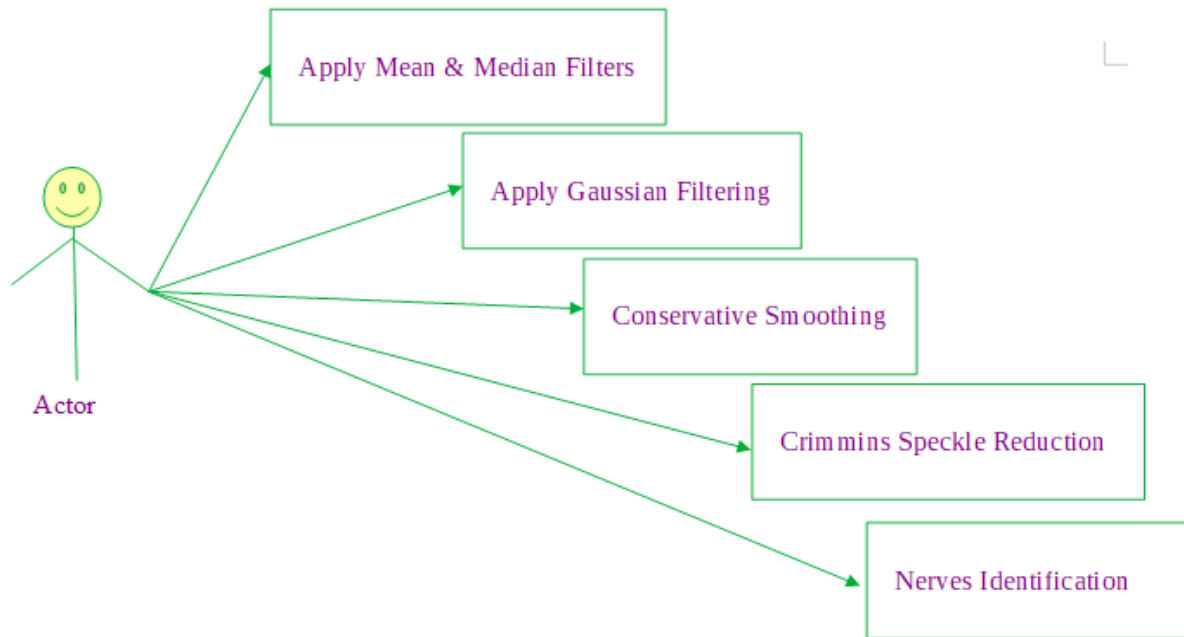


Fig.3.3.1 UML Diagram

3.3.1 DESCRIPTION OF USE CASE DIAGRAM

Use case diagrams are usually referred to as behavior diagrams used to describe a set of actions (use cases) that some system or systems (subject) should or can perform in collaboration with one or more external users of the system (actors). A use case diagram at its simplest is a representation of a user's interaction with the system that shows the relationship between the user and the different use cases in which the user is involved. As we can see the user is interacting with system by a UI through which the customer can perform above mentioned operations like mean, median & Gaussian filters, conservative smoothing, Crimmins Speckle Reduction and nerve Identification.

3.4 SEQUENCE DIAGRAM

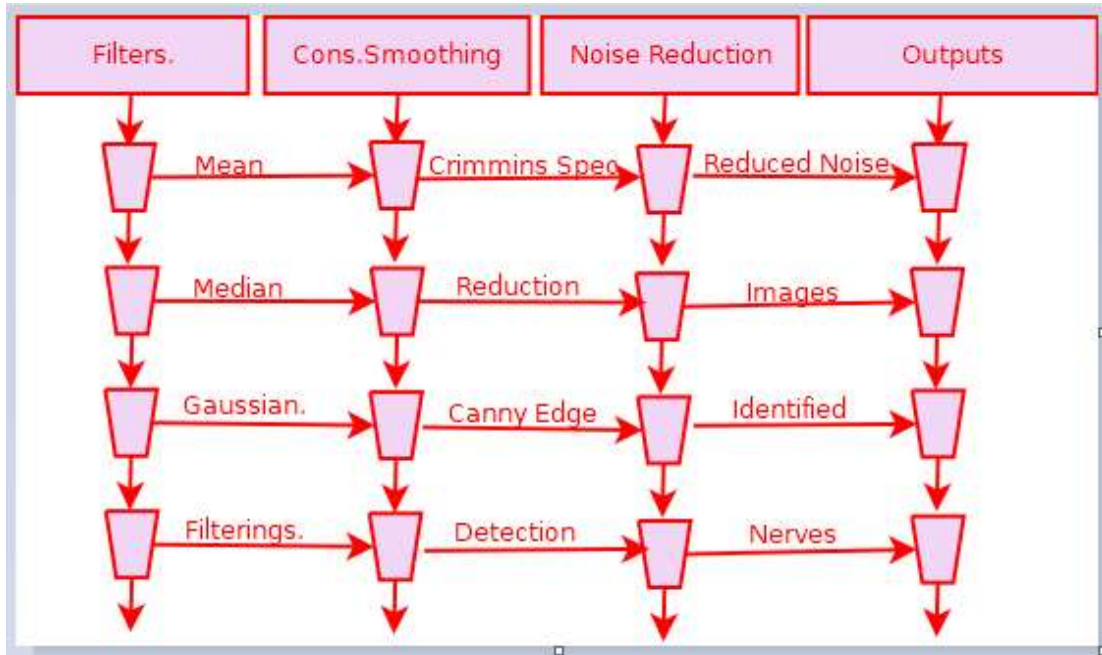


Fig.3.4.1 Sequence Diagram

3.4.1 DESCRIPTION OF SEQUENCE DIAGRAM

A sequence diagram is an interaction diagram that shows how objects operate with one another and in what order. It is a construct of a message sequence chart. A sequence diagram shows object interactions arranged in time sequence. From above mentioned sequence diagram we have to go in sequence: Apply the mean, median and Gaussian filtering techniques, conservative Smoothing, Crimmns Speckle Reduction & Canny edge detection for nerves Identification.

3.5MEAN FILTERING

The **median filter** is a non-linear digital filtering technique, often used to remove noise from an image or signal. Such noise reduction is a typical pre-processing step to improve the results of later processing (for example, edge detection on an image). Median filtering is very widely used in digital image processing because, under certain conditions, it preserves edges while removing noise (but see the discussion below), also having applications in signal processing.

3.5.1 ALGORITHM DESCRIPTION

The main idea of the median filter is to run through the signal entry by entry, replacing each entry with the median of neighboring entries. The pattern of neighbors is called the

"window", which slides, entry by entry, over the entire signal. For one-dimensional signals, the most obvious window is just the first few preceding and following entries, whereas for two-dimensional (or higher-dimensional) data the window must include all entries within a given radius or ellipsoidal region (i.e. the median filter is not a separable filter).

3.6 GAUSSIAN FILTERING

In electronics and signal processing, a Gaussian filter is a filter whose impulse response is a Gaussian function (or an approximation to it, since a true Gaussian response is physically unrealizable as it has infinite support). Gaussian filters have the properties of having no overshoot to a step function input while minimizing the rise and fall time. This behavior is closely connected to the fact that the Gaussian filter has the minimum possible group delay. It is considered the ideal time domain filter, just as the sinc is the ideal frequency domain filter.^[1] These properties are important in areas such as oscilloscopes^[2] and digital telecommunication systems.

The one-dimensional Gaussian filter has an impulse response given by

$$g(x) = \sqrt{\frac{a}{\pi}} \cdot e^{-a \cdot x^2}$$

and the frequency response is given by the Fourier transform

$$\hat{g}(f) = e^{-\frac{\pi^2 f^2}{a}}$$

with the ordinary frequency. These equations can also be expressed with the standard deviation as parameter

$$g(x) = \frac{1}{\sqrt{2\pi} \cdot \sigma} \cdot e^{-\frac{x^2}{2\sigma^2}}$$

and the frequency response is given by

$$\hat{g}(f) = e^{-\frac{f^2}{2\sigma_f^2}}$$

$$\sigma \cdot \sigma_f = \frac{1}{2\pi},$$

where the standard deviations are expressed in their physical units, e.g. in the case of time and frequency in seconds and hertz, respectively. In two dimensions, it is the product of two such Gaussians, one per direction:

$$g(x, y) = \frac{1}{2\pi\sigma^2} \cdot e^{-\frac{x^2+y^2}{2\sigma^2}}$$

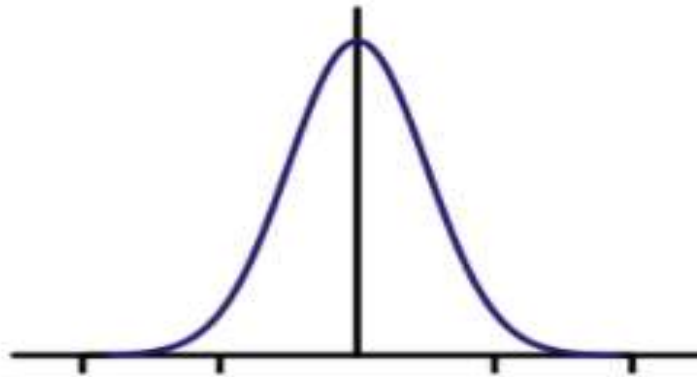


Fig 3.6.1. Shape of the impulse response of a typical Gaussian filter

3.7 CRIMMINS SPECKLE REMOVAL

Crimmins Speckle Removal works by passing an image through a speckle removing filter which uses the *complementary hulling technique* (raising pixels that are darker than their surrounding neighbors, then complementarily lowering pixels that are brighter than their surrounding neighbors) to reduce the speckle index of that image. The algorithm uses a non-linear noise reduction technique which compares the intensity of each pixel in an image with those of its 8 nearest neighbors and, based

upon the relative values, increments or decrements the value of the pixel in question such that it becomes more representative of its surroundings. The noisy pixel alteration (and detection) procedure used by Crimmins is more complicated than the ranking procedure used by the non-linear median filter. It involves a series of pairwise operations in which the value of the 'middle' pixel within each neighborhood window is compared, in turn, with each set of neighbors (N-S, E-W, NW-SE, NE-SW) in a search for intensity spikes.

The operator has a simple form. Suppose that at any time the three consecutive pixels (e.g. on a N-S column) that are being examined are a , b , c . The algorithm is:

1. For each iteration
 1. Do dark pixel adjustment: For each of the 4 directions
 1. Process whole image with: **if $a \geq b + 2$ then $b = b + 1$**
 2. Process whole image with: **if $a > b$ and $b \leq c$ then $b = b + 1$**
 3. Process whole image with: **if $c > b$ and $b \leq a$ then $b = b + 1$**
 4. Process whole image with: **if $c \geq b + 2$ then $b = b + 1$**
 2. Do light pixel adjustment: For each of the 4 directions
 1. Process whole image with: **if $a \leq b - 2$ then $b = b - 1$**
 2. Process whole image with: **if $a < b$ and $b \geq c$ then $b = b - 1$**
 3. Process whole image with: **if $c < b$ and $b \geq a$ then $b = b - 1$**
 4. Process whole image with: **if $c \leq b - 2$ then $b = b - 1$**

Note that, over several iterations, the effects of smoothing in this way propagate out from the intensity spike to infect neighboring pixels. In other words, the algorithm smoothes by reducing the magnitude of a locally inconsistent pixel, as well as increasing the magnitude of pixels in the neighborhood surrounding the spike. It is important to notice that a *spike* is defined here as a pixel whose value is more than 2 intensity levels different from its surroundings. This means that after 2 iterations of the algorithm, the immediate neighbors of such a spike may themselves become *spikes* with respect to pixels lying in a wider neighborhood.

3.8 CANNY EDGE REMOVAL

The **Canny edge detector** is an edge detection operator that uses a multi-stage algorithm to detect a wide range of edges in images. It was developed by John F. Canny in 1986. Canny also produced a *computational theory of edge detection* explaining why the technique works.

3.8.1 DEVELOPMENT OF CANNY ALGORITHM

Canny edge detection is a technique to extract useful structural information from different vision objects and dramatically reduce the amount of data to be processed. It has been widely applied in various computer vision systems. Canny has found that the requirements for the application of edge detection on diverse vision systems are relatively similar. Thus, an edge detection solution to address these requirements can be implemented in a wide range of situations. The general criteria for edge detection include:

1. Detection of edge with low error rate, which means that the detection should accurately catch as many edges shown in the image as possible
2. The edge point detected from the operator should accurately localize on the center of the edge.
3. A given edge in the image should only be marked once, and where possible, image noise should not create false edges.

To satisfy these requirements Canny used the calculus of variations – a technique which finds the function which optimizes a given functional. The optimal function in Canny's detector is described by the sum of four exponential terms, but it can be approximated by the first derivative of a Gaussian.

Among the edge detection methods developed so far, Canny edge detection algorithm is one of the most strictly defined methods that provides good and reliable detection. Owing to its optimality to meet with the three criteria for edge detection and the

simplicity of process for implementation, it became one of the most popular algorithms for edge detection.

3.8.2 PROCESS OF CANNY EDGE DETECTION ALGORITHM

The Process of Canny edge detection algorithm can be broken down to 5 different steps:

1. Apply Gaussian filter to smooth the image in order to remove the noise
2. Find the intensity gradients of the image
3. Apply gradient magnitude thresholding or lower bound cut-off suppression to get rid of spurious response to edge detection
4. Apply double threshold to determine potential edges
5. Track edge by hysteresis: Finalize the detection of edges by suppressing all the other edges that are weak and not connected to strong edges.

The Canny algorithm is adaptable to various environments. Its parameters allow it to be tailored to recognition of edges of differing characteristics depending on the particular requirements of a given implementation. In Canny's original paper, the derivation of the optimal filter led to a Finite Impulse Response filter, which can be slow to compute in the spatial domain if the amount of smoothing required is important (the filter will have a large spatial support in that case). For this reason, it is often suggested to use Rachid Deriche's infinite impulse response form of Canny's filter (the Canny–Deriche detector), which is recursive, and which can be computed in a short, fixed amount of time for any desired amount of smoothing. The second form is suitable for real time implementations in FPGAs or DSPs, or very fast embedded PCs. In this context, however, the regular recursive implementation of the Canny operator does not give a good approximation of rotational symmetry and therefore gives a bias towards horizontal and vertical edges.

CHAPTER 4

IMPLEMENTATION

4.1 FLOW DIAGRAM

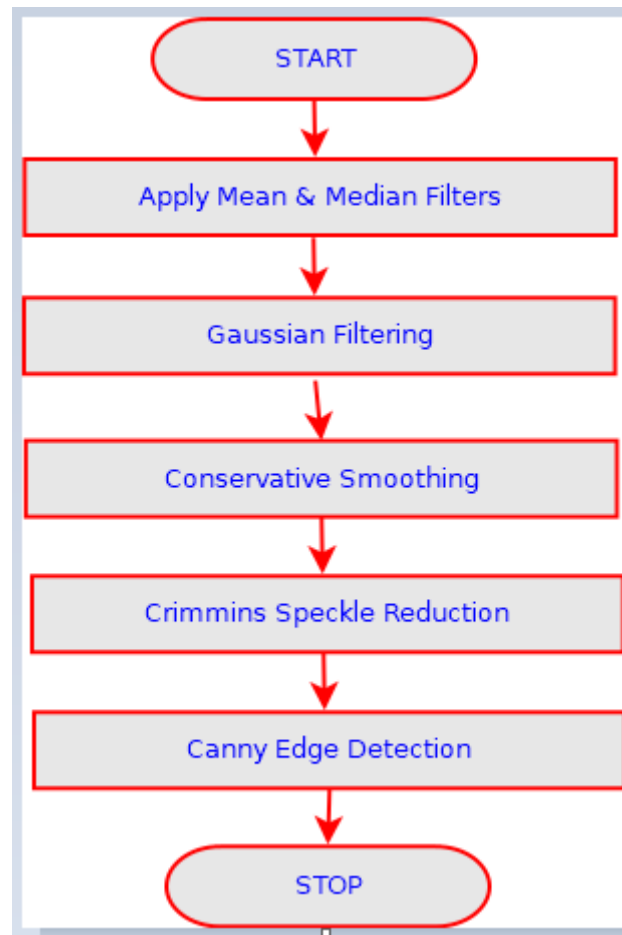


Fig.4.1.1 Flow Diagram

This diagram is basically a flow chart to represent the flow from one activity to another activity. The activity can be described as an operation of the system. So, the control flow is drawn from one operation to another. In activity diagram we can see that first apply mean, median & Gaussian filetrings, perform conservative smoothing & Crimmins Speckle Reduction and finally execute the canny edge detection for nerves identification.

4.2 PYTHON

Python was conceived in the late 1980s, and its implementation began in December 1989 by Guido van Rossum at Centrum Wiskunde & Informatics (CWI) in the Netherlands as a successor to the ABC language (itself inspired by SETL) capable of exception handling and interfacing. Python is an interpreted high-level programming language for general-purpose programming. Created by Guido van Rossum and first released in 1991, Python has a design philosophy that emphasizes code readability, notably using significant whitespace. It provides constructs that enable clear programming on both small and large scales. Python features a dynamic type system and automatic memory management. It supports multiple programming paradigms, including object oriented, imperative, functional and procedural, and has a large and comprehensive standard library. Python interpreters are available for many operating systems. CPython, the reference implementation of Python, is open source software and has a community-based development model, as do nearly all of its variant implementations.

4.2.1 FEATURES AND PHILOSOPHY

Python is a multi-paradigm programming language. Object-oriented programming and structured programming are fully supported, and many of its features support functional programming and aspect-oriented programming (including by meta programming and meta objects (magic methods)). Many other paradigms are supported via extensions, including design by contract and logic programming. Python uses dynamic typing, and a combination of reference counting and a cycle detecting garbage collector for memory management. It also features dynamic name resolution (late binding), which binds method and variable names during program execution.

4.3 MODULES IMPORTED

4.3.1 OS MODULE

The OS module in Python provides a way of using operating system dependent functionality. The functions that the OS module provides allows you to interface with the underlying operating system that Python is running on – be that Windows, Mac or Linux.

4.3.2 OS FUNCTIONS

Executing a shell command

`os.system()`

Get the users environment

`os.environ()`

Returns the current working directory.

`os.getcwd()`

Return the real group id of the current process.

`os.getgid()`

Return the current process's user id.

`os.getuid()`

Returns the real process ID of the current process.

`os.getpid()`

Set the current numeric umask and return the previous umask.

`os.umask(mask)`

Return information identifying the current operating system.

`os.uname()`

Change the root directory of the current process to path.

`os.chroot(path)`

Return a list of the entries in the directory given by path.

`os.listdir(path)`

Create a directory named path with numeric mode mode.

`os.mkdir(path)`

Recursive directory creation function.

`os.makedirs(path)`

Remove (delete) the file path.

`os.remove(path)`

Remove directories recursively.

`os.removedirs(path)`

Rename the file or directory src to dst.

`os.rename(src, dst)`

Remove (delete) the directory path.

`os.rmdir(path)`

4.3.3 SYS MODULE

The `sys` module provides information about constants, functions and methods of the Python interpreter. `dir(system)` gives a summary of the available constants, functions and methods. Another possibility is the `help()` function. Using `help(sys)` provides valuable detail information. The module `sys` informs e.g. about the maximal recursion depth (`sys.getrecursionlimit()`) and provides the possibility to change (`sys.setrecursionlimit()`). The current version number of Python can be accessed as well by using this module. Lots of scripts need access to the arguments passed to the script, when the script was started. `argv` (or to be precise `sys.argv`) is a list, which contains the command-line arguments passed to the script. The first item of this list contains the name of the script itself. The arguments follow the script name. Every serious user of a UNIX or Linux operating system knows standard streams, i.e. input, standard output and standard error. They are known as pipes. They are commonly abbreviated as `stdin`, `stdout`, `stderr`. The standard input (`stdin`) is normally connected to the keyboard, while the standard error and standard output go to the terminal (or window) in which you are working. These data streams can be accessed from Python via the objects of the `sys` module with the same names, i.e. `sys.stdin`, `sys.stdout` and `sys.stderr`. The standard output (`stdout`) can be redirected e.g. into a file, so that we can process this file later with another program. The same is possible with the standard error stream, we can redirect it into a file as well. We can redirect both `stderr` and `stdout` into the same file or into separate files.

4.4 NumPy

NumPy's main object is the homogeneous multidimensional array. It is a table of elements (usually numbers), all of the same type, indexed by a tuple of positive integers. In NumPy dimensions are called axes. NumPy is module for Python. The name is an acronym for "Numeric Python" or "Numerical Python". It is an extension module for Python, mostly written in C. This makes sure that the precompiled mathematical and numerical functions and functionalities of Numpy guarantee great execution speed. Furthermore, NumPy enriches the programming language Python with powerful data structures, implementing multi-dimensional arrays and matrices. These data structures guarantee efficient calculations with matrices and arrays. The implementation

is even aiming at huge matrices and arrays, better known under the heading of "big data". Besides that the module supplies a large library of high-level mathematical functions to operate on these matrices and arrays. SciPy (Scientific Python) is often mentioned in the same breath with NumPy. SciPy needs Numpy, as it is based on the data structures of Numpy and furthermore its basic creation and manipulation functions. It extends the capabilities of NumPy with further useful functions for minimization, regression, Fourier-transformation and many others. Both NumPy and SciPy are not part of a basic Python installation. They have to be installed after the Python installation. NumPy has to be installed before installing SciPy. NumPy is based on two earlier Python modules dealing with arrays. One of these is Numeric. Numeric is like NumPy a Python module for high-performance, numeric computing, but it is obsolete nowadays. Another predecessor of NumPy is Numarray, which is a complete rewrite of Numeric but is deprecated as well. NumPy is a merger of those two, i.e. it is built on the code of Numeric and the features of Numarray. NumPy's array class is called ndarray. It is also known by the alias array. Note that numpy.Array is not the same as the Standard Python Library class array.array, which only handles one-dimensional arrays and offers less functionality. The more important attributes of an ndarray object are:

ndarray.ndim

The number of axes (dimensions) of the array.

ndarray.shape

The dimensions of the array. This is a tuple of integers indicating the size of the array in each dimension. For a matrix with n rows and m columns, shape will be (n,m). The length of the shape tuple is therefore the number of axes, ndim.

ndarray.size

The total number of elements of the array. This is equal to the product of the elements of shape.

ndarray.dtype

An object describing the type of the elements in the array. One can create or specify dtype's using standard Python types. Additionally NumPy provides types of its own. numpy.int32, numpy.int16, and numpy.float64 are some examples.

ndarray.itemsize

The size in bytes of each element of the array. For example, an array of elements of type float64 has itemsize 8 ($=64/8$), while one of type complex32 has itemsize 4 ($=32/8$). It is equivalent to `ndarray.dtype.itemsize`.

ndarray.data

The buffer containing the actual elements of the array. Normally, we won't need to use this attribute because we will access the elements in an array using indexing facilities.

4.5 TOOLS USED

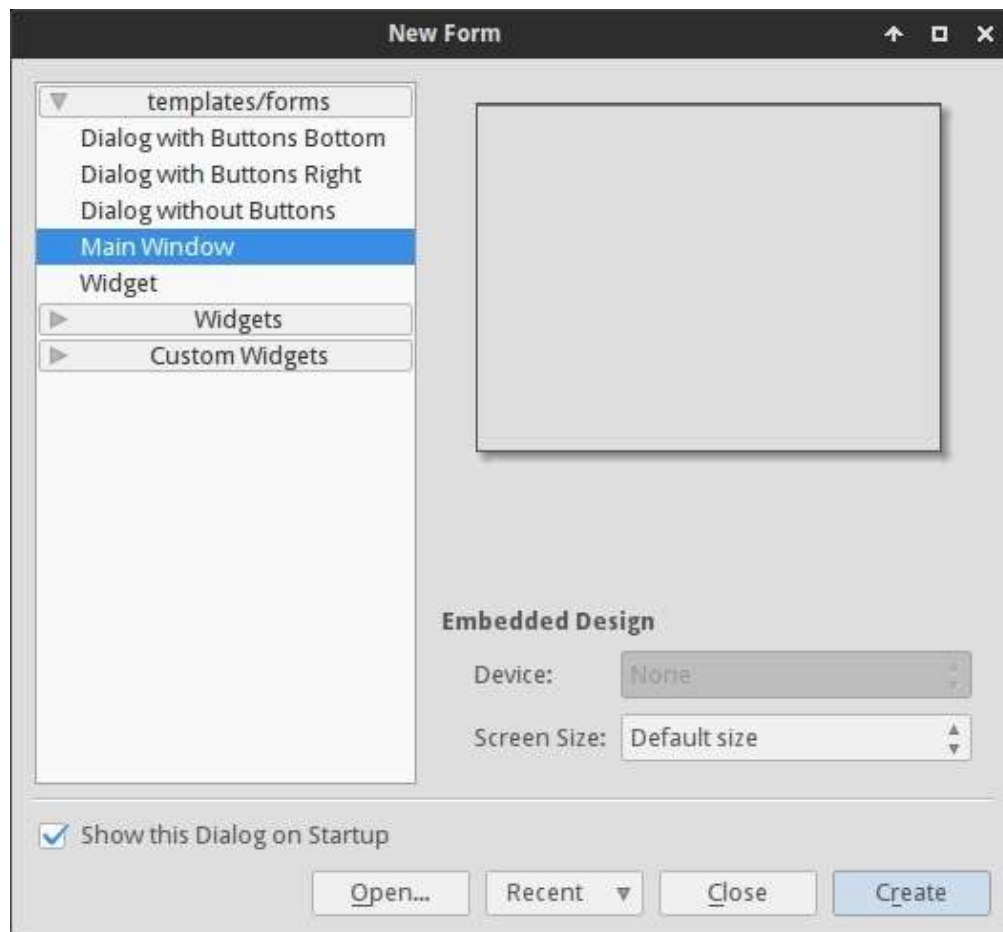


Fig.4.5.1 QT Designer Screenshot 1

You can then drag and drop required widgets from the widget box on the left pane. You can also assign value to properties of widget laid on the form.

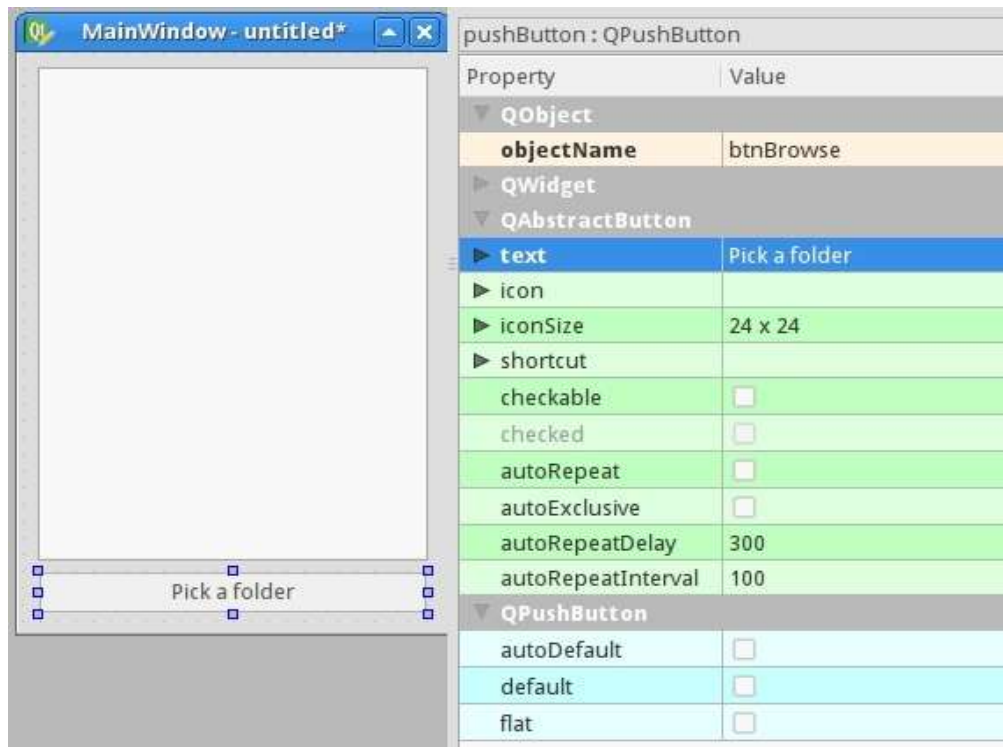


Fig.4.5.2 QT Designer Screenshot 2

The designed form is saved as demo.ui. This ui file contains XML representation of widgets and their properties in the design. This design is translated into Python equivalent by using pyuic4 command line utility. This utility is a wrapper for uimodule. The usage of pyuic4 is as follows



Fig.4.5.3 Ui files

4.5.1 QPushButton

The push button, or command button, is perhaps the most commonly used widget in any graphical user interface. Push (click) a button to command the computer to perform some action, or to answer a question. Typical buttons are OK, Apply, Cancel, Close,

Yes, No and Help. A command button is rectangular and typically displays a text label describing its action. A shortcut key can be specified by preceding the preferred character with an ampersand in the text. For example:

```
QPushButton *button = new QPushButton("&Download", this);
```

In this example the shortcut is Alt+D. See the QShortcut documentation for details (to display an actual ampersand, use '&&'). Push buttons display a textual label, and optionally a small icon. These can be set using the constructors and changed later using `setText()` and `setIcon()`. If the button is disabled the appearance of the text and icon will be manipulated with respect to the GUI style to make the button look "disabled". A push button emits the signal `clicked()` when it is activated by the mouse, the Spacebar or by a keyboard shortcut. Connect to this signal to perform the button's action. Push buttons also provide less commonly used signals, for example, `pressed()` and `released()`. Command buttons in dialogs are by default auto-default buttons, i.e. they become the default push button automatically when they receive the keyboard input focus. A default button is a push button that is activated when the user presses the Enter or Return key in a dialog. You can change this with `setAutoDefault()`. Note that auto-default buttons reserve a little extra space which is necessary to draw a default-button indicator. If you do not want this space around your buttons, call `setAutoDefault(false)`. Being so central, the button widget has grown to accommodate a great many variations in the past decade. The Microsoft style guide now shows about ten different states of Windows push buttons and the text implies that there are dozens more when all the combinations of features are taken into consideration. The most important modes or states are:

1. Available or not (grayed out, disabled).
2. Standard push button, toggling push button or menu button.
3. On or off (only for toggling push buttons).
4. Default or normal. The default button in a dialog can generally be "clicked" using the Enter or Return key.
5. Auto-repeat or not.
6. Pressed down or not.

As a general rule, use a push button when the application or dialog window performs an

action when the user clicks on it (such as Apply, Cancel, Close and Help) and when the widget is supposed to have a wide, rectangular shape with a text label. Small, typically square buttons that change the state of the window rather than performing an action (such as the buttons in the top-right corner of the `QFileDialog`) are not command buttons, but tool buttons. Qt provides a special class (`QToolButton`) for these buttons. If you need toggle behavior (see `setCheckable()`) or a button that auto-repeats the activation signal when being pushed down like the arrows in a scroll bar (see `setAutoRepeat()`), a command button is probably not what you want. When in doubt, use a tool button. A variation of a command button is a menu button. These provide not just one command, but several, since when they are clicked they pop up a menu of options. Use the method `setMenu()` to associate a popup menu with a push button. Other classes of buttons are option buttons (see `QRadioButton`) and check boxes (see `QCheckBox`). In Qt, the `QAbstractButton` base class provides most of the modes and other API, and `QPushButton` provides GUI logic. See `QAbstractButton` for more information about the API.

4.5.2 `QLineEdit`

A line edit allows the user to enter and edit a single line of plain text with a useful collection of editing functions, including undo and redo, cut and paste, and drag and drop. By changing the `echoMode()` of a line edit, it can also be used as a "write-only" field, for inputs such as passwords. The length of the text can be constrained to `maxLength()`. The text can be arbitrarily constrained using a `validator()` or an `inputMask()`, or both. When switching between a validator and an input mask on the same line edit, it is best to clear the validator or input mask to prevent undefined behaviour. A related class is `QTextEdit` which allows multi-line, rich text editing. You can change the text with `setText()` or `insert()`. The text is retrieved with `text()`; the displayed text (which may be different, see `EchoMode`) is retrieved with `displayText()`. Text can be selected with `setSelection()` or `selectAll()`, and the selection can be `cut()`, `copy()`ied and `paste()`d. The text can be aligned with `setAlignment()`. When the text changes the `textChanged()` signal is emitted; when the text changes other than by calling `setText()` the `textEdited()` signal is emitted; when the cursor is moved the `cursorPositionChanged()` signal is emitted; and when the Return or Enter key is pressed the `returnPressed()` signal is emitted. When editing is finished, either because the line

edit lost focus or Return/Enter is pressed the `editingFinished()` signal is emitted. Note that if there is a validator set on the line edit, the `returnPressed()/editingFinished()` signals will only be emitted if the validator returns `QValidator.Acceptable`. By default, `QLineEdit`s have a frame as specified by the Windows and Motif style guides; you can turn it off by calling `setFrame(false)`. The default key bindings are described below. The line edit also provides a context menu (usually invoked by a right mouse click) that presents some of these editing options.

4.5.3 QMainWindow

A main window provides a framework for building an application's user interface. Qt has `QMainWindow` and its related classes for main window management. `QMainWindow` has its own layout to which you can add `QToolBars`, `QDockWidgets`, a `QMenuBar`, and a `QStatusBar`. The layout has a centre area that can be occupied by any kind of widget. You can see an image of the layout below

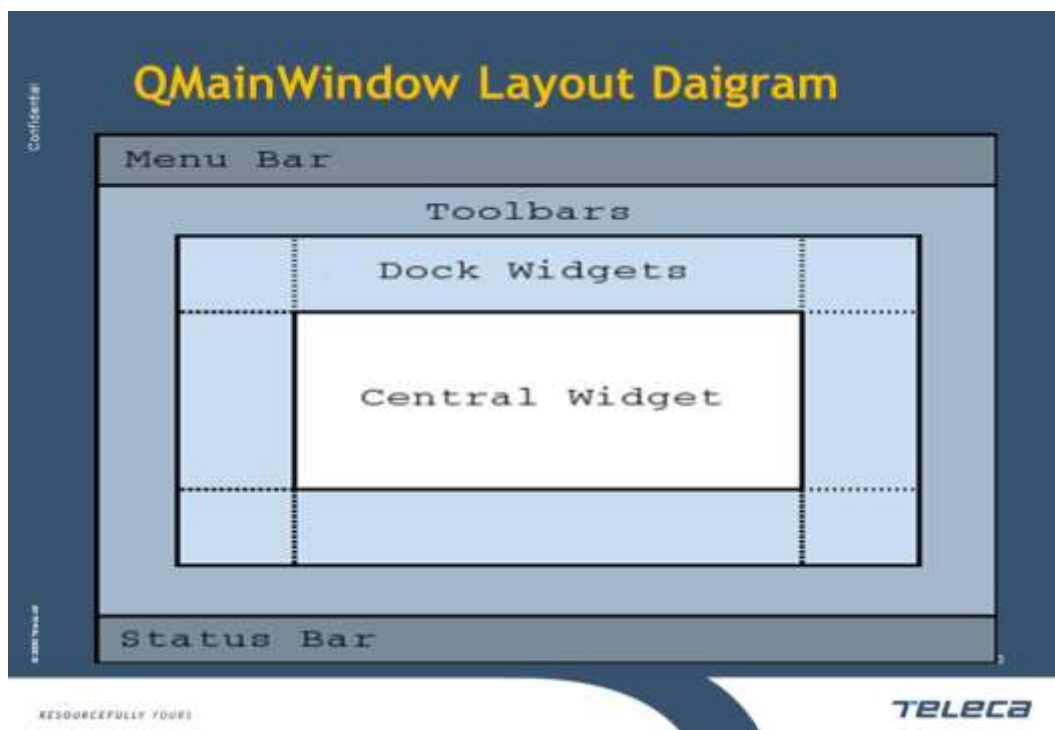


Fig 4.5.3.1 QMainWindow Layout Diagram

Note: Creating a main window without a central widget is not supported. You must have a central widget even if it is just a placeholder.

Creating Main Window Components

A central widget will typically be a standard Qt widget such as a QTextEdit or a QGraphicsView. Custom widgets can also be used for advanced applications. You set the central widget with setCentralWidget(). Main windows have either a single (SDI) or multiple (MDI) document interface. You create MDI applications in Qt by using a QMdiArea as the central widget. We will now examine each of the other widgets that can be added to a main window. We give examples on how to create and add them.

4.5.4 CREATING MENU

Qt implements menus in QMenu and QMainWindow keeps them in a QMenuBar. QActions are added to the menus, which display them as menu items. You can add new menus to the main window's menu bar by calling menuBar(), which returns the QMenuBar for the window, and then add a menu with QMenuBar::addMenu(). QMainWindow comes with a default menu bar, but you can also set one yourself with setMenuBar(). If you wish to implement a custom menu bar (i.e., not use the QMenuBar widget), you can set it with setMenuWidget().

Creating Main Window Components

A central widget will typically be a standard Qt widget such as a QTextEdit or a QGraphicsView. Custom widgets can also be used for advanced applications. You set the central widget with setCentralWidget(). Main windows have either a single (SDI) or multiple (MDI) document interface. You create MDI applications in Qt by using a QMdiArea as the central widget. We will now examine each of the other widgets that can be added to a main window. We give examples on how to create and add them.

An example of how to create menus follows:

```
void MainWindow.createMenus()
{
    fileMenu = menuBar()->addMenu(tr("&File"));
    fileMenu->addAction(newAct);
}
```

```
fileMenu->addAction(openAct);
```

```
fileMenu->addAction(saveAct);
```

The `createPopupMenu()` function creates popup menus when the main window receives context menu events. The default implementation generates a menu with the checkable actions from the dock widgets and toolbars. You can reimplement `createPopupMenu()` for a custom menu.

4.5.5 CREATING TOOLBAR

Toolbars are implemented in the `QToolBar` class. You add a toolbar to a main window with `addToolBar()`. You control the initial position of toolbars by assigning them to a specific `Qt.ToolBarArea`. You can split an area by inserting a toolbar break - think of this as a line break in text editing - with `addToolBarBreak()` or `insertToolBarBreak()`. You can also restrict placement by the user with `QToolBar.setAllowedAreas()` and `QToolBar.setMovable()`. The size of toolbar icons can be retrieved with `iconSize()`. The sizes are platform dependent; you can set a fixed size with `setIconSize()`. You can alter the appearance of all tool buttons in the toolbars with `setToolButtonStyle()`.

An example of toolbar creation follows:

```
void MainWindow.createToolBars()
{
    fileToolBar = addToolBar(tr("File"));
    fileToolBar->addAction(newAct);
```

`isWidgetType()` returns whether an object is actually a widget. It is much faster than `inherits("QWidget")`. Some `QObject` functions, e.g. `children()`, `objectTrees()` and `queryList()` return a `QObjectList`. A `QObjectList` is a `QPtrList` of `QObject`s. `QObjectList`s support the same operations as `QPtrList`s and have an iterator class, `QObjectListIt`. To convert the design file to python code saved as `design.py`, use `cd` command to change to the directory holding the `design.ui` file and simply run:

```
$ pyuic4 design.ui -o design.py
```

If you want to specify full path for either input or output file you can do that like this:

```
$ pyuic4 path/to/design.ui -o output/path/to/design.py
```

4.6 SOFTWARE TESTING

Software testing is the process of evaluation a software item to detect differences between given input and expected output. Testing assesses the quality of the product. Software testing is a process that should be done during the development process. In other words, software testing is a verification and validation process.

4.6.1 VERIFICATION

Verification is the process to make sure the product satisfies the conditions imposed at the start of the development phase. In other words, to make sure the product behaves the way we want it to.

4.6.2 VALIDATION

Validation is the process to make sure the product satisfies the specified requirements at the end of the development phase. In other words, to make sure the product is built as per customer requirements.

4.6.3 BLACK BOX TESTING

Black box testing is a testing technique that ignores the internal mechanism of the system and focuses on the output generated against any input and execution of the system. It is also called functional testing.

4.6.4 WHITE BOX TESTING

White box testing is a testing technique that takes into account the internal mechanism of a system. It is also called structural testing and glass box testing.

Black box testing is often used for validation and white box testing is often used for verification.

4.6.5 TYPES OF TESTING

There are many types of testing like

1. Unit Testing

Unit testing is the testing of an individual unit or group of related units. It falls under the class of white box testing. It is often done by the programmer to test that the unit he/she has implemented is producing expected output against given input.

2. Integration Testing

Integration testing is testing in which a group of components are combined to produce output. Also, the interaction between software and hardware is tested in integration testing if software and hardware components have any relation. It may fall under both white box testing and black box testing.

3. Functional Testing

Functional testing is the testing to ensure that the specified functionality required in the system requirements works. It falls under the class of black box testing.

4. System Testing

System testing is the testing to ensure that by putting the software in different environments (e.g., Operating Systems) it still works. System testing is done with full system implementation and environment. It falls under the class of black box testing.

5. Regression Testing

Regression testing is the testing after modification of a system, component, or a group of related units to ensure that the modification is working correctly and is not damaging or imposing other modules to produce unexpected results. It falls under the class of black box testing.

CHAPTER 5

RESULTS AND DISCUSSION

5.1 MEAN FILTERING

Mean filtering is a simple, intuitive and easy to implement method of smoothing images, i.e. reducing the amount of intensity variation between one pixel and the next. It is often used to reduce noise in images.

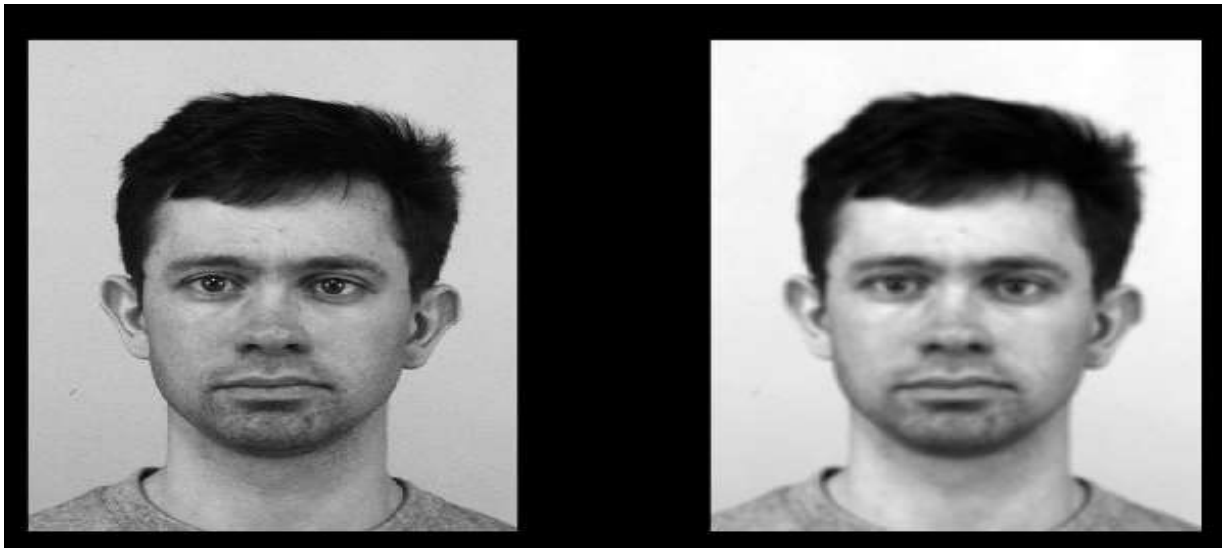


Fig.5.1.1 Mean Filtering

5.2 GAUSSIAN FILTERING

The Gaussian blur is a type of image-blurring filters that uses a Gaussian function (which also expresses the normal distribution in statistics) for calculating the transformation to apply to each pixel in the image. The formula of a Gaussian function in one dimension is

$$G(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{x^2}{2\sigma^2}}$$

In two dimensions, it is the product of two such Gaussian functions, one in each dimension:

$$G(x, y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$



Fig.5.2.1 Gaussian Filtering

5.3 CONSERVATIVE SMOOTHING

Conservative smoothing is a noise reduction technique that derives its name from the fact that it employs a simple, fast filtering algorithm that sacrifices noise suppression power in order to preserve the high spatial frequency detail (e.g. sharp edges) in an image. It is explicitly designed to remove noise spikes --- i.e. isolated pixels of exceptionally low or high pixel intensity (e.g. salt and pepper noise) and is, therefore, less effective at removing *additive noise* (e.g. Gaussian noise) from an image.



Fig 5.3.1 Conservative smoothing

5.4 CRIMMINS SPECKLE REMOVAL

Crimmins Speckle Removal works by passing an image through a speckle removing filter which uses the complementary hulling technique (raising pixels that are darker than their surrounding neighbors, then complementarily lowering pixels that are brighter than their surrounding neighbors) to reduce the speckle index of that image. The algorithm uses a non-linear noise reduction technique which compares the intensity of each pixel in an image with those of its 8 nearest neighbors and, based upon the relative values, increments or decrements the value of the pixel in question such that it becomes more representative of its surroundings. The noisy pixel alteration (and detection) procedure used by Crimmins is more complicated than the ranking procedure used by the non-linear median filter. It involves a series of pair wise operations in which the value of the 'middle' pixel within each neighborhood window is compared, in turn, with each set of neighbors (N-S, E-W, NW-SE, NE-SW) in a search for intensity spikes.

For each iteration:

a) Dark pixel adjustment: For each of the four directions

- 1) Process whole image with: if $a \geq b+2$ then $b = b + 1$
- 2) Process whole image with: if $a > b$ and $b \leq c$ then $b = b + 1$
- 3) Process whole image with: if $c > b$ and $b \leq a$ then $b = b + 1$
- 4) Process whole image with: if $c \geq b+2$ then $b = b + 1$

b) Light pixel adjustment: For each of the four directions

- 1) Process whole image with: if $a \leq b - 2$ then $b = b - 1$
- 2) Process whole image with: if $a < b$ and $b \geq c$ then $b = b - 1$

3) Process whole image with: if $c < b$ and $b \geq a$ then $b = b - 1$

4) Process whole image with: if $c \leq b - 2$ then $b = b - 1$



Fig.5.4.1 Crimmis Speckle Removal

5.4 CANNY EDGE

Canny edge detection is a image processing method used to detect edges in an image suppressing noise.

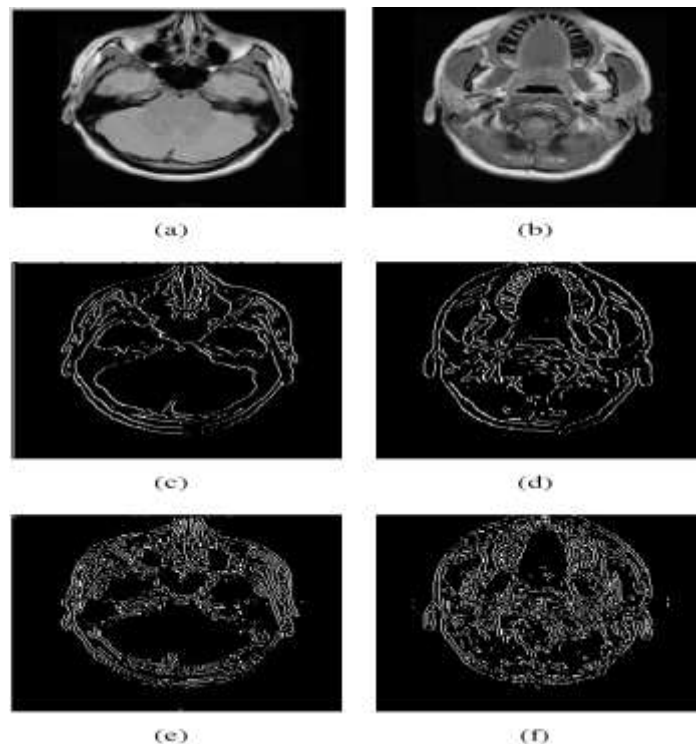


Fig.5.5.1 Canny Edge Removal

CHAPTER 6

CONCLUSION AND FUTURE WORK

6.1 CONCLUSION

This project entitled “Image improvement using Crimmins Speckle Reduction of noise intensity.” is useful to the police personnel in improving the quality of the cctv images, so that culprits can be accurately identified. An image pre-processing step can improve the accuracy of machine learning models. Pre-processed images can help a basic model achieve high accuracy when compared to a more complex model trained on images that were not pre-processed. This project finally leads to the improvement of quality of the investigation of criminal cases.

6.2 FUTURE WORK

As of now, the system is currently working with the downloaded images. The applicability of this project, in the real life scenario, using CCTVs, is yet to be explored.

REFERENCES

- [1] Radhika Chandwadkar,Saurabh Pramod Dhole: Comparison of Edge Detection Techniques, Conference paper,Research Gate
- [2] Shivani Chourasia,Anand Vardhan Bhalla: Review of Segmentation and DT-DWT Techniques,International Journal of Emerging Technologies in Engineering Research,Volume 7, Issue 4
- [3] Priyanka Shivhare, Vinay Gupta,"Review of Image Segmentation Techniques Including Pre & Post Processing Operations", International Journal of Engineering and Advanced Technology (IJEAT) ISSN: 2249 – 8958, Volume-4 Issue-3
- [4] <https://www.python.org/>
- [5]<https://github.com/baoboa/pyqt5/blob/master/pyuic/uic/pyuic.py>
- [6] <https://www.numpy.org/>
- [7] <https://riverbankcomputing.com/software/pyqt/intro>

