A COMPONENT–BASED ARCHITECTURE FOR MEDICAL IMAGE PROCESSING

A COMPONENT–BASED ARCHITECTURE FOR MEDICAL IMAGE PROCESSING

By
CHRISTOPHER LAMBACHER, B.Sc.

A Thesis
Submitted to the School of Graduate Studies
in Partial Fulfilment of the Requirements
for the Degree
Master of Science

McMaster University

© Copyright by Christopher Lambacher, 2005

Master of Science (2005) McMaster University
(Computer Science) Hamilton, Ontario

TITLE: A Component–Based Architecture for Medical Image Processing

AUTHOR: Christopher Lambacher, B.Sc. (McMaster University)

SUPERVISORS: Dr. W.F.S. Poehlman and Dr. M.V. Kamath

NUMBER OF PAGES: x, 70

Abstract

This thesis discusses the development of a component-based architecture for the processing of medical imaging data. Such an architecture enables a wider range of analysis techniques as applied to Magnetic Resonance Imaging (MRI) data than is currently the norm. Differences in image segmentation results on images obtained using 1.5T vs. 3T MRI machines were investigated using the architecture and tools presented in this thesis. This thesis presents an analysis of the design, implementation, and sample results of such an architecture processing actual MRI data. Studies on images acquired from a test object (phantom) and the brain of a human subject reveal that a component-based architecture can assist in developing and testing novel algorithms in the computational laboratory, and serve as a platform for examining the effects of various image processing procedures in a radiology suite.

Acknowledgments

Funding for this project was given in part by NSERC. Computational facilities were generously funded by DeGroote Foundation grants to Dr. Markad V. Kamath. Imaging facilities were provided by St. Joseph's Healthcare Hamilton. Guidance on the MRI portions of this thesis was provided by Dr. Mike Noseworthy. Countless hours of guidance were provided by my supervisors, Dr. Markad V. Kamath and Dr. WSF Poehlman. Thank you all. This thesis would not have been possible without you.

I would also like to thank my parents and my wife for all their love and support.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background

Since the earliest use of X-Rays in medicine over a century ago, medical imaging technology has enhanced the diagnostic ability of doctors and clinicians [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]. Modern physicians now have access to many techniques for creating images of the internal structures of the human body without invasive surgery. Current technology can create high-resolution 2- and 3-dimensional images of internal body structures. Most prominent among such technologies are computerized tomography (CT scan) using X-Rays, magnetic resonance imaging (MRI), single photon emission computed tomography (SPECT) and positron emission tomography (PET) [2, 6]. Each technology has applications that focus on a specific disease [10], novel intervention [3], anatomical structure [11], or physiological function [5, 9]. MRI scanning gives good tissue differentiation and is non-ionizing to tissues; it is therefore often the modality of choice for monitoring anatomy and pathology [7, 11]. Further, since MR images show anatomy in great detail, the scans are ideally suited for quantifying the volume of anatomical structures. Using radiochemistry, novel pharmacological compounds have been formulated to study specific molecules and/or their activity through PET technology [9]. Such complementary tools extend the applications of imaging in medicine [2].

1.2 Motivation, Goals and Objectives

Although imaging technology provides vast opportunities for retrieving valuable information about a patient's internal condition, the volume and formatting of the information produced by the technology may not be easy to interpret directly by the physician or clinician. This brings about the motivation to develop assistive technologies that can facilitate presentation and interpretation of the data generated by specific imaging methods. In particular, processing which helps to automate the isolation of individual anatomical structures and other areas of interest can accelerate the diagnostic process.

This thesis describes a system for managing a type of processing on medical imaging data called image segmentation. Image segmentation techniques attempt to identify structures by separating contiguous sections of an image with similar characteristics. The system designed for this project uses the k-Means [12, pp.529–533] clustering algorithm for performing image segmentation. See Appendix A for the algorithm details.

The design for the system, called the Modular Image-Processing Architecture (MIPA), provides a component-based model that allows for multiple methods of processing and viewing image data. The components of the system operate independently, making the architecture ideal for comparing multiple segmentation algorithms and techniques. In this context components are plugable software modules that allow for the system to be extended by a third party after it has been released. Section 4.5 discusses reasons for this architecture.

Components fall into several categories including:

1. Input (file-format reader)
2. Proximity measure (method to measure similarity of two points)
3. Feature selection (pre-processing)
4. Segmentation (main processing)
5. Integration (post-processing)
6. User interface

This primary goal of the research for this project is the exploration of a component-based system to facilitate further research efforts in image processing techniques. This goal is achieved through the development and testing of each component listed above. In order to examine the feasibility of the concept and subsequent

design, multiple algorithms for component types one, two, four and six were developed and subjected to rigorous evaluation. Different algorithms allowed the possibility of increasing temporal performance by changing algorithm details and facilitated system testing by creating algorithms only used for testing.

A secondary goal of the system is its evaluation as a clinical/medical research tool. Towards this goal, the MIPA system described in this thesis is used to evaluate and compare the performance of a commonly used pattern classification algorithm (k-Means) on images acquired from multiple MRI scanners. The test sources for this thesis were 2- and 3-dimensional images produced from 1.5 and 3 Tesla Magnetic Resonance Imaging (MRI) machines, located at St. Joseph's Healthcare Hamilton.

1.3 Novelty of the Research Presented in this Thesis

As will be discussed in Section 2, many publications discuss the processing of medical images and seek improved algorithms, methods and procedures. However, literature investigations conducted during the course of this research were unable to locate other instances of a modifiable architecture that can enhance and optimize the processing involved in medical imaging context. It is such an architecture, where components can be tailored to individual image processing tasks, which is being proposed and implemented in this thesis. Therefore, research presented in this thesis is novel in a number of ways:

1. There is no known software or platform that permits both clinical researchers and algorithm-oriented investigators to experiment with various paradigms. For example, a physician can set up a script for testing various paradigms such as sequential comparison of images taken over several months, or the effect of therapy. Alternately, an algorithm-oriented researcher can set up individual components of various algorithms using tools provided in the MIPA system described in this thesis.

2. An innovative feature of the system design, in the field of MRI image processing, is that modifications to algorithms or procedures for viewing an image or sets of images can be speedily implemented and propagated to users and published. These modifications can be done and distributed by third parties.

3. It is also a feature of the MIPA system design that it has the capacity in the future to serve as a platform to integrate several complex image-processing

steps, such as algorithms from pattern recognition, artificial intelligence techniques and similar mathematical procedures. For the clinician, the system can act as a test bed to verify hypothesis as to which sets of steps or procedures will generate the best outcome for patient diagnosis and other clinical analysis.

4. McMaster University's facilities at St. Joseph's Healthcare Hamilton house two MRI machines: one with 1.5 Tesla, and the other with 3 Tesla magnetic fields. Both MRI scanners were made available for gathering the images reported in this thesis. This permitted comparison of processing on images from both scanners. Other comparisons between segmentation results on a 1.5 vs. 3 Tesla MRI machine did not appear in any reported literature.

1.4 Overview of the Thesis

Chapter 2 provides an overview of MRI imaging technology and the literature available on various segmentation procedures. Chapter 3 examines the factors that motivated this research; it defines the problem to be addressed, and some of the issues contributing to the formulation of the design. Chapter 3 also defines the scope and limitations of the research, and which aspects of the problem are directly addressed by this research. Chapter 4 provides design criteria and describes how individual features are implemented. Chapter 5 documents the process of testing individual modules, and the overall system. The MIPA system was tested on images generated from a manufactured test object (phantom) and the brain of a human subject. Chapter 5 also documents the clinically relevant results obtained from studying the results of processing these images produced under different controls (1.5 and 3 Tesla, and a combination thereof). Chapter 6 discusses implications of this research in the context of an improved software platform for both clinical users and computer scientists.

1.5 Summary

The principal objective of this thesis is the design of a novel, modular image processing architecture suitable for analyzing magnetic resonance images. The benefits of developing such a system include facilitation of research into algorithms and clinical aspects of image processing.

Chapter 2

Background

2.1 Introduction

The history of imaging for clinical purposes has centred on the transmission of radiation through the human body for most of the twentieth century. While computed tomography in itself was a major breakthrough in thinking of how to assemble the images to minimize radiation, the use of magnetic field at radio frequencies set the tone for novel imaging modalities in the last 30 years. In this chapter, a high-level overview of magnetic resonance imaging and a survey of the literature on segmentation techniques are presented.

2.2 Magnetic Resonance Imaging

Magnetic Resonance Imaging (MRI) produces high-definition 2- and 3-dimensional raster images of internal body structures by using a strong magnetic field and radio frequency (RF) waves.

Felix Blotch theorized that a spinning charged particle, such as a proton, creates a magnetic field [13]. Magnetic fields have an orientation just like a bar magnet. The North and South poles of a magnet, or magnetic field, naturally attract one another; within a charged particle, this results in an alternating alignment of the particle's protons. Where the number of protons is evenly matched (paired), the field orientation is nullified and undetectable. However, in certain atoms with an unmatched proton (such as Oxygen, Nitrogen, and Hydrogen) the unpaired proton results in a measurable magnetic field with a specific orientation. It is this field which MRI uses to produce its images.

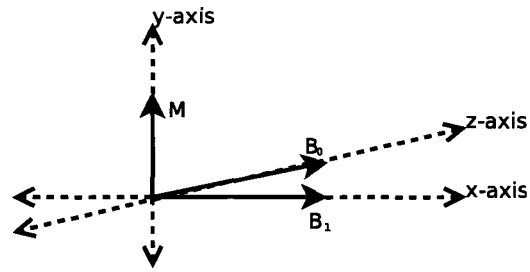


Figure 2.1: Magnetic Forces

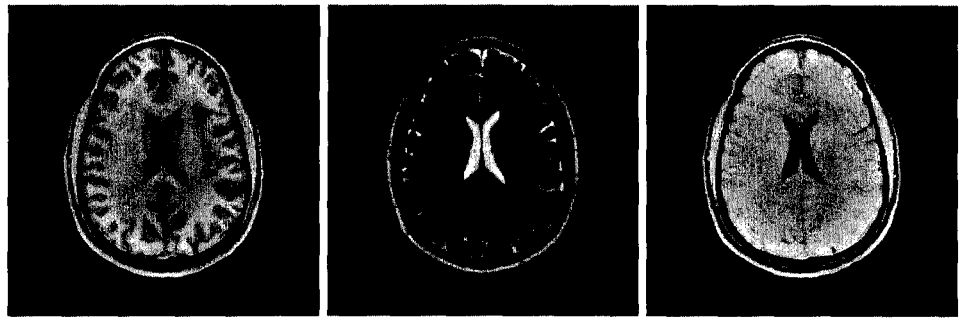
When a magnet is placed in the magnetic field of another magnet, the original magnet will attempt to align itself to the other field. This same phenomenon causes a compass to align itself with the earth's magnetic field. In MRI imaging, the subject is placed in a high strength magnetic field, called the B_0. The unpaired protons in the subject's body will align themselves such that their field is in the direction of B_0. The alignment of the fields with another magnetic field is called magnetization.

To generate an image, low-frequency radio waves are directed at the area to be imaged. These waves are reflected back by the magnetized spins of the protons within the object. Certain frequencies of radio waves will change the alignment of the spins; this is because the Radio Frequency (RF) pulse creates a smaller localized magnetic field called B_1. The introduction of the B_1 field from the RF pulse will actually cause the magnetization to be perpendicular to B_0 and B_1. If we say the z-axis is parallel to B_0 and the RF pulse is along the x-axis, then magnetization M will be deflected along the y-axis (Figure 2.1).

Once the RF pulse is turned off, the component of magnetization in the x-y plane will decrease and the component of magnetization along the z-axis will increase. This process is called relaxation, because the magnetization will relax to its equilibrium state in alignment with B_0.

The magnetization along the z-axis can be expressed as a function of time, t since the RF pulse was turned off, and of the final equilibrium magnetization, M_0, as $M_z(t) = M_0(1 - e^{\frac{-t}{T_1}})$. T_1 is a constant that depends on the specific material being magnetized and the strength of B_0. As B_0 decreases, so does T_1.

The magnetization along the x-y plane can be expressed as a function of t, time since turning off the RF pulse, and of the final equilibrium magnetization M_0 as $M_{xy}(t) = M_0 e^{\frac{-t}{T_2}}$. T_2 is a constant that depends on the specific material being



(a) 3T T_1 (b) 3T T_2 (c) 3T Proton Density

Figure 2.2: Image of the same region of the brain captured with 3 different MRI modalities.

magnetized.

The magnetization of the subject will also depend on the density of protons. An object with fewer protons to magnetize, will have a lower magnetic field and hence magnetization. This Proton Density (PD) is denoted as $N(H)$.

Different types of MRI images reflect these different properties, T_1, T_2 and PD. Figure 2.2 shows how T_1, T_2 and PD images differ. Table 2.1 shows how different materials in the body appear for T_1 and T_2 pulse sequences. The values obtained for an image by a particular pulse sequence are dependent on the B_0 field strength and the material being scanned. If the magnetic field is changed and the same material is scanned, the resultant values for the image will be different. The magnetic field of an MRI is generally not changeable and permanently switched on.

2.3 Image Segmentation

An MR image provides discrete values for an area or a volume of the anatomy being studied. These values for areas and volumes are called pixels and voxels respectively. Segmentation is the process of identifying sections of the image as representing distinct anatomical structures. Segmentation plays an important role in the diagnostic process because of the need to accurately measure the volume of structures such as tumours. Researchers can use these measurements to fur-

Table 2.1: Intensity appearance of structures from various MR sequences [14].

Structure	T_1–weighted	T_2–weighted
Fat	White	Light grey
Air	Black	Black
Bone	Black	Black
White matter	Light grey	Grey
Grey matter	Grey	Very light grey
CSF in ventricles	Dark grey	Light grey or white
Malignant tissue	Dark grey	Light grey or white

ther the understanding of diseases like cancer, and even mental diseases such as Alzheimer's disease or schizophrenia [15]. In a MRI image of the brain, the process of identifying distinct segments of an image and specific structures such as grey matter or white matter follows segmentation procedures. Research in the area of image segmentation is quite extensive, therefore only representative but relevant papers are reviewed in this section.

Image segmentation can be done manually, semi-automatically or automatically. Partial or full automation in segmentation is normally accomplished using pattern recognition algorithms. These algorithms normally make use of some derived information about the image that may or may not include the image itself; this derived information is termed a feature of the image. Processing the image may require several different pattern recognition algorithms, or multiple passes with the same algorithm. These algorithms are designed for processing images of specific areas of the body, such as the head [15].

Pattern recognition techniques may use many types of features (derived information) extracted from the images in question. Features could be the intensity value for the pixel, the intensity value after applying filters to reduce noise, the maximum or minimum intensity value in the general area of the pixel being studied [14], or any other combination of information about the pixel and its neighbours.

The most effective choice of pattern recognition technique for any given image will depend on the pulse sequence used, the region being imaged, the resolution of the scan and many other factors [14].

Saeed [14] identifies the following five key image characteristics that are em-

ployed for segmentation of MRI images: grey tone distribution, texture properties, edge detection, region growing and contour following. Grey tone distribution uses a threshold to distinguish important structures from the background based on image intensity values. The texture properties characteristic attempts to identify regions of similar texture, such as smoothness, coarseness, and regularity. Edge detection determines the location of abrupt changes in grey level that occur at the edge of a structure. Region growing algorithms start with a seed of 8 pixels (for 2-D structures) and 26 voxels (for 3-D MRI) and neighbouring pixels are compared for similar properties. When there are no acceptable neighbours of similar properties, the growth process is terminated. Contour following is particularly robust for determining the boundary of binary structures. Each pixel or voxel has dimensional and shape information attached to it. A start of the boundary is identified and path of the contour is mapped out using specific rules [16]. Binary-based contouring procedures are particularly helpful for extracting outer scalp boundaries [14]. Segmentation procedures based on the properties listed above often provide less than optimum segmentation and are often not suitable for automated segmentation algorithms. Higher-level segmentation procedures described in the literature incorporate algorithms for recognition and interpretation [14].

Much recent research has focused on fuzzy clustering methods such as Fuzzy c-Means. Clark et al. [17] used Fuzzy c-Means to identify tumours in the brain. They used a multi-step approach where they initially searched for more clusters than necessary and then performed a second clustering step. The initial step assures that a particular cluster will not erroneously contain pixels from more than one structure. The second pass groups the clusters from the first pass together.

Chang et al. [18] developed a two-stage system using Fuzzy c-Means and a fuzzy rule-based system. The first stage uses the fuzzy rule based system to provide an initial pixel filter which aims to do a good first approximation, while the second stage uses Fuzzy c-Means to classify the remaining unclassified pixels.

Xue et al. [19] use Fuzzy c-Means in conjunction with various filtering techniques to provide accurate segmentation even with noisy images. The technique first removes noise with a wavelet-based filter. The images then go through preliminary segmentation using minimum-error thresholding. Next, segmentation is done with Fuzzy c-Means followed by a process called adaptive enhancement.

Kwon et al. [20] used a modified form of Fuzzy c-Means called hierarchical Fuzzy c-Means. They first performed pre-processing for noise reduction and to remove the skull from the image. Next, they use their hierarchical Fuzzy c-Means to do the actual segmentation. To do this, they divide the image into hierarchical volumes and calculate the membership for each volume.

Poon and Braun [21] present a contour model that incorporates region analysis for segmentation. Their deformable model uses an iterative method to minimize an energy function for N contours corresponding to N+1 regions.

Boone et al. [22] have developed a multivariate statistical model of MRI images and have applied it to segment images in 3D space. Their solution for segmentation uses the 3D relative distribution of the pixel-intensities to form a probabilistic model that is built using the properties of the image in all directions of the neighbourhood for each pixel. Boone et al. report accuracies of 79.4% and 78.1% when identifying grey matter and white matter respectively.

2.4 State-of-the-Art in Modular Based Image Processing Systems

From the review in the previous section it can be stated that few researchers have thought of developing an architecture suitable for implementing any number of myriad of algorithms, from basic building components of computing and develop an architecture that optimizes resources (memory, CPU), processing steps and combines them for diagnostic or testing purposes. While there are libraries that provide some of the functions needed for image segmentation, no description of component / modular-based systems was found in the literature for performing a specific procedure such as feature extraction or segmentation. For example, certain toolkits, such as VTK [23], VolVis [24] can perform visualization but do not provide any other capability. Both VTK and VolVis provide 3D data visualization and rendering. Other toolkits provide data access routines and some low level processing functions. Examples of this include ITK [25, 26], ImLib3D [27] and OpenVL [28, 29]. However, a comprehensive software framework to support a modularized system suitable for medical image processing could not be found during a focused literature search and review.

2.5 Summary

In this chapter an outline of magnetic resonance imaging is provided. A review of segmentation methods is also presented. Most of these methods focus on efficient use of algorithms but very few publications articulate the need for modifiable architecture to optimally process images. There are currently no papers that focus on modular image processing systems, the subject of this thesis.

Chapter 3

Motivation for a Modular-Based Architecture for Image Processing

3.1 Introduction

Medical Imaging is a rapidly expanding field. MRI in particular can assist a physician in arriving at an accurate diagnosis, observing changes in a patient's condition over time, and predict a prognosis for certain types of pathologies. The needs of end-users who operate on MR images recorded in a clinical environment are well documented [14]. An inquiry into such documentation can give greater insight into the users' needs and identify areas where a software designer must focus. This chapter examines these requirements and proposes that a modular component-based architecture can enhance the quality of processing in terms of its efficiency and quality. While the MIPA architecture can be used with a variety of image processing algorithms, segmentation has been selected as the focus for this research.

3.2 Possible System Architectures

Several architectures are possible for an image processing system. This paper proposes using a component–based system. Two other options are a modular but monolithic system or a library–based system.

The most basic architecture is the monolithic system. The software itself may be written in a modular fashion simplifying modification, however each addition to the system would require a complete recompile and the entire system would

need to be replaced on the end user's computer.

A slightly more configurable approach would be a library–based system. The library system would be allow the creation of the system in layers, each layer building on lower level libraries. New libraries could be added at any level to extend functionality and bug fixes could be applied to individual libraries as needed without requiring the end user to install all libraries for a particular update. However, adding new algorithms as new libraries would require changes to the higher level parts of the system.

A component based system is the most configurable. Components are discoverable at runtime and describe their capabilities to the core system. In this architecture, algorithms may be added piecemeal and changes can be provided to the end user as individual components. This also allows a greater degree of freedom for the collection and integration of third party functionality.

3.3 Justification of MIPA for Segmentation

Segmentation is one of the most important procedures in image processing and is needed for a large variety of applications. Therefore, segmentation was chosen to serve as a vehicle/tool for implementing MIPA and to evaluate its performance with one specific algorithm (k-Means clustering). Techniques proposed in this thesis are not confined to segmentation, as they can be extended to other image processing techniques as well.

3.4 Justification for the MIPA System in Image Processing Systems

Although there exist algorithms and techniques for processing images and performing qualitative analysis on them, there does not yet exist a system that integrates these techniques. Because of such a limitation in the technical domain, each new research project in an imaging facility must reinvent the entire processing pipeline, even if the research is only concerned with one particular part of the pipeline. This makes it necessary for the researcher to spend extra time on designing elements of a system that are not central to their work or expertise.

3.5 MR Facilities at St. Joseph's Healthcare Hamilton

During the course of the research reported in this thesis, it was discovered that there were two scanners at the facility: one with 1.5 Tesla field strength and another with a 3 Tesla field strength (B_0). It was therefore possible to obtain a set of images from each of these two scanners for the same anatomical structure or test object (phantom). Since automatic image segmentation results are heavily dependent on the source images used, recording several different images from the same anatomical structure can improve these results. For example, use of T_1 weighted and T_2 weighted images together is very common. As stated in Section 2.2, the values for T_1 and T_2 are not only dependent on the material being scanned but also the magnetic field of the scanner (B_0). St. Joseph's Healthcare Hamilton provided a unique opportunity to compare segmentation results from two MRI machines with differing magnetic fields.

3.6 Identification of End Users and their Needs

There are two categories of users of the system described in this thesis. The first category of user is the algorithm researcher who is primarily interested in design, test and evaluation of algorithms, and the performance of those algorithms on given images. This type of user is interested in modifying algorithms, using new sources of data, defining new methods of feature extraction and generally altering the processing of data and its flow through the system. Algorithm researchers usually have some computer programming background.

The second category of users are interested in clinical medicine. This category of user includes medical researchers, physicians and clinicians. End users are interested in using specific (usually previously recorded) images, features and algorithms. They may or may not have some computer programming background. Some of these users can be categorized as advanced users who have a deeper understanding of the system and have experienced the benefits of sophisticated image processing in their clinical practice. These advanced users may want to experiment and extend the boundaries of what image processing can do to treat their patients.

The algorithm researchers will identify and explore new methods of using the system for a variety of algorithms, while clinical end users will use those techniques to perform medical research or diagnosis. When algorithm researchers

have created a new extension to the system, a method of publication should allow the extension to be made available to the medical specialist who can benefit from their colleagues' work. This method of publication should allow piecemeal addition of extensions and not require recompilation of the system. Preferably, extensions should not be dependent on system version.

Segmentation and image processing in general has several interconnected steps, the first of which is data loading. The second step is data processing. Further data processing steps may use the original data and/or the output of a previous processing step. Finally, data visualization is required for the original input data and for data output at any of the processing steps.

A graphical user interface should make it simple to access the extensions and be flexible enough to adapt to the requirements of the extension. The extension itself may determine part of its own interface. The interface should provide a mechanism for specifying what data loading, data processing and visualization steps to perform, as well as how the data is fed from each processing step into the next step.

Finally, in order for this system to become accepted by a wide variety of users, it should run on all computing platforms that are currently being used for medical imaging research and diagnosis. These include Microsoft Windows, Linux, common Unix variants and Mac OS X.

3.7 Limitations

The requirements as outlined in Section 3.6 are very broad; implementation of such a system is a major undertaking that could extend to several years. For this project, it was decided that the processing pipeline should be limited to five separate tasks, namely, data input, feature extraction, segmentation, post processing and visualization. It was also decided that only original data and segmentation results would be displayed (none of the intermediate results or post processing data, such as histogram, would be displayed). These limitations reduce the complexity of the design and implementation to a manageable level, while retaining the core functionality necessary to prove the usefulness of a system of this type. The tasks selected for implementation still provide significant flexibility for future segmentation research.

3.8 Summary

Identification of end users and their needs are critical while designing any software system. In this context, the needs of two diverse groups of users were examined to arrive at a solution that can stand alone and serve its intended purpose. Therefore, this chapter delved into operational requirements of two groups of specialized users of the proposed system. In addition, justification was provided as to why segmentation was made the key tool for testing the modular architecture based system.

Chapter 4

Design of the Proposed System with Component-Based Architecture

4.1 Introduction

This chapter describes the design and development of a configurable system. The system consists of three distinct layers, namely: the user interface layer, a core layer which acts as the glue to keep the system together, and finally the component layer which provides the ability to modify the computational process. This chapter describes the design and implementation of the above layers, and discusses the justification for and use of various tools.

4.2 Component-Based Architecture

4.2.1 Configurability through Component Use

To allow users and developers to easily extend the system, most of the functionality of the system is implemented as components that are detected at runtime. Components can provide file input, feature extraction, distance (dissimilarity) calculation, clustering or post-processing functionality. The user interfaces are also modifiable. Core functionality is separated from the user interface; any number of user interfaces can be developed to interact with the software interface of the core system.

4.2.2 System Layers

The system is designed in a series of layers as shown in Figure 4.1. The top and bottom layers, user interface and components respectively, can have an arbitrary number of implementations. The system interface and the component manager compose the core of the system. The following sections describe the design of the layers in the system.

Core Layer

The core of the system is made up of the system interface and the component manager. It provides a library with which a user interface interacts. The system interface connects to the component manager. The component manager enumerates the components in the component layer and provides a method of querying the information collect at enumeration time to the system interface. The system interface in turn provides an interface abstraction for interacting with the components to other layers. The core layers provide services to the components that are necessary for system operation. The core specifies the method of image representation that is internally used by the system; there is no defined object for this, instead an array used where each value in the array is a pixel value. Nothing stops components from using another representation internally to itself, however all image data is moved through the system in the format used bu the core.

User Interface Layer

Multiple user interfaces can be designed to serve different purposes. The user interfaces use information provided by the system interface layer to determine at run time which actions can be performed. Actions are scripted through the system interface in the core layer. The core layer also provides information back to the user interfaces, as feedback, about the status of the activities initiated from the user interfaces.

Component Layer

The component layer provides information about particular component instances through registration of various objects to the component manager. The components are divided into several types depending on their function (See Section 4.3). Further, the function of each component determines the software interface and how the component manager will use it.

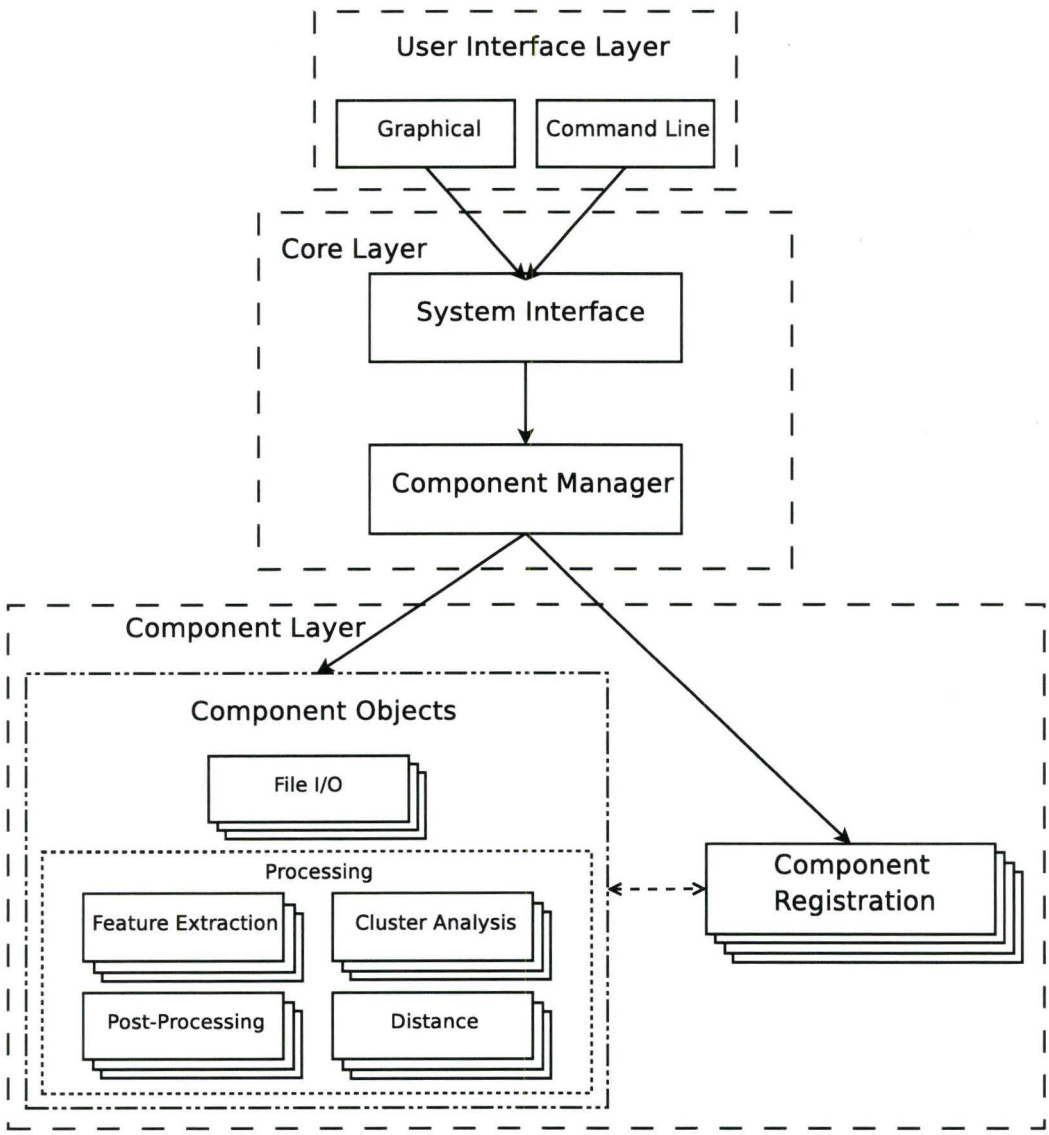


Figure 4.1: System Layers

4.3 Implementation

The design described in this thesis was implemented in the Python programming language and the GTK+ Graphical User Interface (GUI) toolkit. Selection considerations for choosing Python are described below in Section 4.4. A core set of Python libraries and user interface elements provide the basic infrastructure, while additional functionality is added through a component infrastructure. Internally, both image and feature data are represented as one row in a 2-dimensional array using the Numeric Python library. The Numeric Python library is also used for numeric processing. Numeric Python provides MATLAB-like matrix operations in Python.

Components fall into several categories depending on the functionality they provide. Categories include services that load data files, do feature extraction, calculate distances, compute clustering metrics, and finally perform post-processing.

Components use the registration interface to provide information about themselves to the system at runtime. The information provided includes the component label, a human understandable label and the names and types of the parameters that the component is expecting. Valid component parameters are integers, floating point numbers, strings, file names, lists of file names, integer ranges, floating point number ranges, and slices. Slices are a concept borrowed from Python and are not to be confused with image slices. A slice, in this context, refers to a selection of a range of values from a list. This is useful for specifying dimensions to use for feature extraction.

Processing tasks are driven by a list of commands to be executed, along with their arguments. Each command is mapped to a component that performs the specified action. The command-processing engine always processes components in a specific order, first by component type, and then in the order they were given. Processing by order of type is done to satisfy dependencies between component types. Data file loading is always first, followed by the registration of the distance measure. Next, feature extraction is performed, followed by segmentation and finally post processing. A component is used when an associated command is executed.

Two user interfaces were developed for this thesis: a command line interface and a GTK+ based GUI interface (See Appendix C). The command line interface makes it possible to execute commands from a scripting language such as the Bourne Shell, which is useful for testing new components. The GUI interface provides new users an easier method for selecting the commands they want to run, and provides a method for viewing results adjacent to original images.

4.4 Programming Languages and Development Tools

Python was used as the primary programming language because of the ease with which applications can be developed. Python is a high level, object-oriented, interpreted language. In many ways it is similar to Perl, but also has many distinctive features and a more legible syntax. Python natively provides many useful structures such as lists and dictionaries, and handles memory considerations. This makes it easy for a new programmer to start using Python.

Python provides an extensive range of basic library functions as well as having many freely available third party libraries for more specialized tasks. Python and most of the core libraries are portable to Unix, Mac OS, Windows, and other systems. Python also allows libraries to be built in C or C++; this is important for cases where it would be useful to add the functionality of a C or C++ library as a component to the system. This also allows components to be completely written in C or C++, allowing greater control over memory usage and the speed of computation in circumstances when the Python language is too slow.

The Numeric Python library was employed for internal representation and manipulation of data. Numeric Python was used because it provides optimized matrix calculations. Numeric Python matrix manipulations and semantics are very similar to MATLAB, which is commonly used in large-scale matrix data manipulation. Python and the Numeric Python library should be easy to read and modify for anyone with programming experience and an understanding of matrix computation.

GTK+ was used as the widget set (user interface components) for the GUI because of its cross platform nature and mature, freely available, Python language bindings. The development model for GTK+ with Python is very rapid. GUI layout is done with the GLADE development tool (`http://glade.gnome.org/`) and the user interface specification is saved to an XML file which generates the user interface elements when loaded at runtime. Since the XML files generated are lengthy and not intended to be hand modified, a sample is omitted. Once an initial GUI project framework has been laid out, the programmer proceeds by defining event handlers to react to user initiated events such as button clicks.

All libraries used, and Python itself, are open source projects and are available to anyone for any use with no restrictions. This was an important factor in choosing the tools and libraries used. For instance, Qt – an alternative to GTK+ – was rejected because it was necessary to pay for a development license or accept the GPL license. The GPL license may have negative implications for component developers. See Appendix D for The Component Developer's Guide.

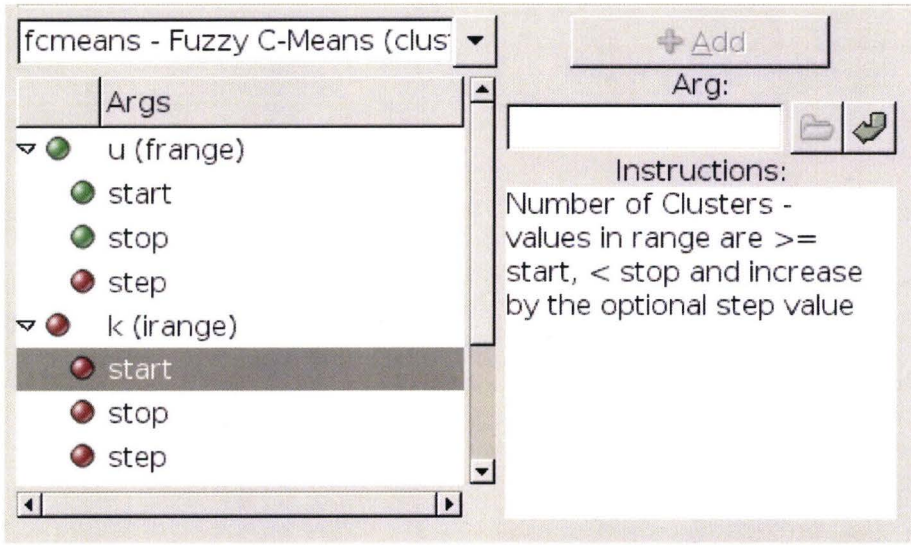
4.5 Details of the Architecture

As discussed in section 4.2, the system was designed with layered interfaces and components providing the majority of the functionality. This structure was used because it provided the maximum amount of flexibility and ease of extensibility. Designing the system with layered software interfaces allows future developers to create new kinds of user interfaces to the system that may better integrate with existing systems. For example, to integrate the system with the software on a particular imaging device, the user interface elements alone could be changed, without having to re-implement the component architecture and command-processing engine.

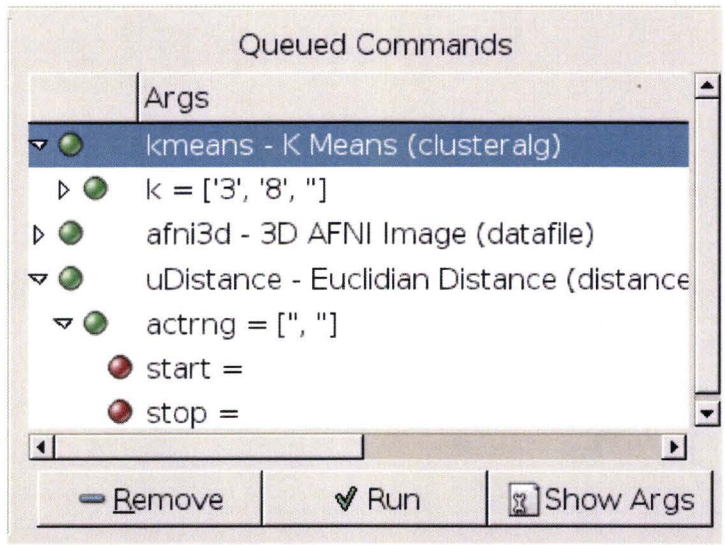
A component-based architecture was used for the majority of the system because it facilitates adding new data file loaders and processing algorithms. This extensibility is important so that researchers and developers can share their progress, without the need to allocate resources to integrating new software components into the core system. The inspiration for these design considerations was the Vim editor project [30], which has an on-line repository of scripts providing various forms of functionality. This model has been very successful in allowing the developers of the editor to focus on the core product, while providing a framework that makes user- contributed extensions available to everyone. A similar approach is taken by the Mozilla Project [31], which provides user interface extensions to their web browser and email clients at `http://update.mozilla.org/`.

4.6 Human Computer Interaction

The interface for configuring a processing run is divided into two sections: adding arguments to the command queue, and reviewing them before sending them to be processed. All of the information used in the interface for adding commands is provided by the component registration process (See Figure 4.2(a)). The top left of the interface provides a drop down box for selecting the desired command. Once selected, the arguments that can be entered are added to the tree located below the drop down box. Initially all arguments are flagged as incomplete with a red dot. Once an argument is filled out using the entry location on the right, the argument will be flagged as complete with a green dot. As can be seen in Figure 4.2(a), argument components are broken down in the tree. Once all necessary components of an argument are completed, the argument is flagged as completed, while the subcomponents which are still incomplete, are flagged as



(a) Adding Arguments



(b) Reviewing Arguments

Figure 4.2: Processing Setup

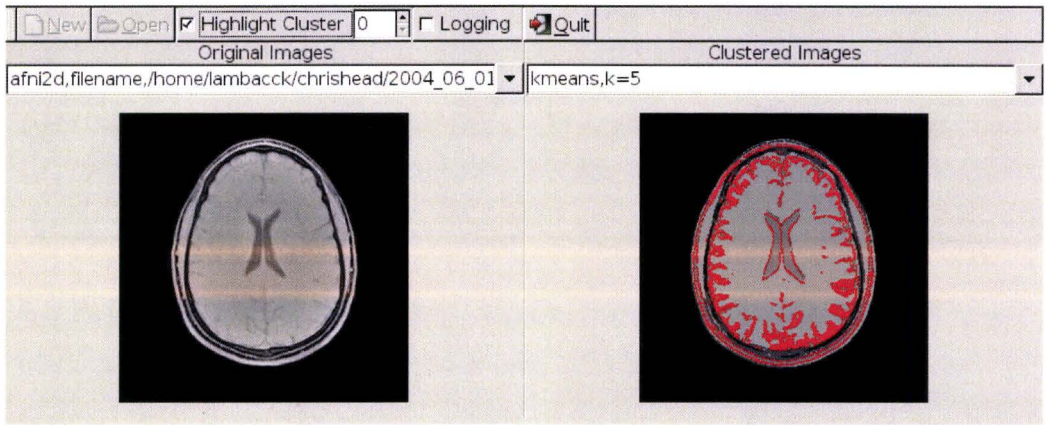


Figure 4.3: Image and Result Visualization

such. The open button on the right is enabled when appropriate, and the green arrow button allows the user to indicate their acceptance of the argument value. A description of the argument or component is given in the bottom right. When all necessary arguments are completed, the add button in the top right is enabled.

The interface for reviewing the command queue, as shown in Figure 4.2(b), consists mainly of a tree of commands in the queue. The tree expands to show arguments of the commands and their subcomponents. The same green and red dots show which arguments were given values. Buttons across the bottom allow the user to remove the selected command, accept the queued commands and send them to be processed, or have the commands be printed to the log window. The log window outputs in a format that can be used with the command line interface. The run button is only enabled when there are enough commands in the queue for an execution run successfully. A successful run requires at least one data source, a distance function, and a processing algorithm.

Once a command queue has been set to run, or when a previously run queue is opened, the visualization interface is enabled (See Figure 4.3). As results are returned from processing algorithms, they become available to be viewed. The clustered results take their colour pallet from the average value of the cluster associated with the pixel. Since each input image provides a possible colour average, the colour used to display the clustered result is taken from the input image currently being displayed on the left side of the screen. When a 3D image is being visualized, a slider bar allows the user to select the slice they would like to view. A check box and spin control at the top of the window allow the user to highlight a

cluster so that it is easily identifiable in the visualization results. Figure 4.3 shows a cluster being highlighted in red.

4.7 Summary

This chapter provides architectural details of the component-based system. It was the intention of the design to incorporate as many open source tools and libraries as possible, with the hope that it would make maintenance easy and manageable.

Chapter 5

Evaluation of the Component-Based Architecture for Image Processing

5.1 Introduction

Following the design and implementation of the proposed system, an evaluation of its performance is presented in this chapter. In order to assess software functionality and quality a biomedical study was conducted using the system. The results from the evaluation of the software functionality and quality are discussed in this chapter, while the results of the biomedical study are discussed in Appendix B.2.

The goal of the assessment of the software functionality and quality is to determine whether the system performs as expected and whether it is generally useful.

5.2 Evaluation of the Overall System

5.2.1 Testing of Individual Components

Unit testing was performed on each software module in order to ascertain that the basic expectations were met. Some modules were trivial to assess. The image-loading modules were tested in conjunction with the image display module. The data path though the clustering algorithm was the most difficult to test. In order to confirm conformance, a dummy algorithm was developed that assessed each pixel as its own cluster, resulting in an output display with a 1:1 pixel map from original to output image. When the image was displayed and confirmed to be displaying

correctly, the data path was determined to be valid.

5.2.2 Performance Evaluation

Two categories of performance evaluation were done: evaluation of the k-Means algorithm [12, pp.529–533], and evaluation of the time taken to process the data.

The k-Means algorithm was used in this thesis because it is well documented in pattern recognition literature, allowing a focus on the architecture itself. The details of the algorithm are presented in Appendix A for completeness.

Evaluation of the k-Means Algorithm

The k-Means algorithm was originally developed as a working implementation using MATLAB, and then ported to the Python language for use within the system. Test data containing 2, 3, and 4 clusters were correctly classified. Testing was also done using the MATLAB implementation and MRI data, which was useful for direct comparison to the output from the Python implementation.

Data Processing Time

Execution time was tested with the Python implementation of the k-Means algorithm. The results indicated that execution would take too long to be practically useful to researchers. The python profiler found that most of the execution time was being spent in the interface between the Distance function and the k-Means algorithm. Based on these observations, a C implementation for the k-Means algorithm was created with the Pyrex tool. The Pyrex tool allows developers to write modules in a Python-like language that is then translated into a C language file; this C language file is next compiled as a C language extension for Python. Only minor modifications were necessary to convert the existing Python implementation of k-Means to C, using Pyrex.

Tables 5.1 and 5.2 show the time taken, in seconds, for the C Language and Pure Python implementations for both the 2-D data and 3-D data with 56 slices. Speed improvements of an order of magnitude or better were obtained, in general.

The component based architecture of the system allowed easy extension of the system to include the alternative k-Means algorithm implementation and also allowed easy comparison of the two implementations. Addition and comparison of completely different algorithms would not be any more difficult.

Table 5.1: Time taken – in seconds – with Pure Python and C Language implementations of k-Means algorithm for 2-D images.

Clusters	Pure Python	C Language
3	109	29
4	824	53
5	1806	93
6	1682	154

Table 5.2: Time taken – in seconds – with Pure Python and C Language implementations of k-Means algorithm for 3-D images.

Clusters	Pure Python	C Language
3	40260	1723
4	212894	2895
5	140550	5031
6	250059	8051

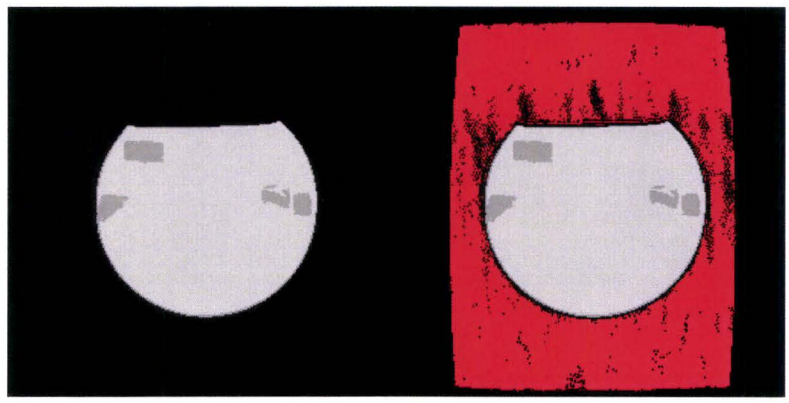


Figure 5.1: Two views of the same resultant image: the left version has no cluster highlighting; the one on the right has a cluster highlighted (in red) that was not visible without highlighting.

This comparison also showed that the system can just as easily be extended in C as in native Python.

Visualization of Results

Clustered results are shown with a palette taken from the original image being shown on the left of the display. Sometimes clusters do not appear visible with this mechanism, because the original colours in the area surrounding the cluster are too similar to the cluster itself to be easily distinguishable to the human eye. This is the reason behind the *highlight cluster* option. One cluster at a time can be highlighted with the highlight colour. For the purpose of this project, the colour selected was red; however, this should be a configurable option in a publicly released version of this system, as red may not be suitable for some users. Figure 5.1 shows a clustered image without highlighting turned on and the same result image with highlighting turned on. Such highlighting greatly enhances the ability for the user to analyze and appreciate these results.

5.3 Summary

The system proved its ability to be used for segmentation research. Adding new functionality is relatively easy and further research into segmentation tech-

niques could be aided by using this system. The system implemented successfully fulfils the goals set out in Chapters 3.6 and 4.2.

Extensibility was demonstrated by the implementation of components that were used entirely for testing as well as components designed to provide performance enhancement over previously developed components. The ability to perform Algorithm comparison is shown through the development of two different implementations for the k-Means algorithm which provided vastly different performance from each other.

Chapter 6

Discussion and Suggestions for Future Work

6.1 Introduction

Having designed, developed and tested a component-based image processing system, it is essential to examine the relevance of the system in a computational laboratory or radiology suite. This chapter identifies the above issues and suggests some possible avenues for future research.

6.2 General Advantages of the MIPA System

One of the direct benefits of the system is its portability between platforms. Many tools only work on one or a small number of operating systems. The system presented in this thesis is portable to any operating system that supports the Python language. Since it is possible to create new user interfaces, an operating system that supports Python but not the GTK+ graphical toolkit can still be supported by creating a new user interface using a different graphical toolkit.

6.3 Advantages of the MIPA System for Algorithm Research

In order to effectively compare several different (but similar-purpose) algorithms, it is necessary to minimize differences between the test cases. The MIPA

system allows researchers to isolate the algorithm of interest from the rest of the system through the compartmentalization of components, making it easier to directly correlate differences in results to the algorithms themselves. A computer scientist can focus his efforts on a new image-processing algorithm, without developing a new user interface or core system pipeline.

If a MIPA-like system is adopted as a standard for research, new algorithms can be easily compared against existing algorithm implementations. The MIPA system makes it easy for algorithm implementations to be shared between researchers.

6.4 Advantages of the MIPA System in a Radiology Suite

If a component-based system such as the MIPA system became a standard tool in the Radiology Suite, researchers and clinicians would be able to create and exchange new diagnostic tools and algorithms. Some examples of how the system would accomplish this goal include:

- A clinician can adopt the latest, most efficient components for image processing or core system work, without having to relearn the entire system interface
- A user-interface developer can design one or more interfaces optimized to particular types of users, without needing direct knowledge of how the underlying system layers operate.

6.5 Suggestions for Future Work

The most important addition to the system would be 3-D visualization. This would present obvious advantages for the clinician in his diagnostic efforts, by allowing the additional perspectives on structures that are an integral part of a 3-dimensional image.

Performance gains could be obtained from various opportunities for parallel processing. The simplest type of parallel processing applicable to the system would see processing steps that are not dependent on one another performed simultaneously. This kind of scheduling, while not trivial, is a well-known problem space. Solutions for tasks of this kind exist for database operations, paral-

lel compilation/build activities, and other scheduling-intensive problem domains. Another type of parallel computation applicable would in the processing of the algorithms that comprise the various components of the system. Parallelization of algorithm processing is a more difficult task, as research into parallelizable algorithms will need to be performed.

A great deal of flexibility can be obtained by fully implementing the requirements from Section 3.6; more diverse types of data and processing could then be supported. With this accomplished, even pre-processing tasks such as those described in Section B.2.2 could be integrated into the system.

6.6 Summary

This chapter presents the multiple uses and application of component-based architecture in both for algorithm research and a radiological context. Suggestions for future work are also listed.

Chapter 7

Conclusions

This thesis has presented a component-based architecture to assist research into segmentation of radiological images in general and MRI images in particular. At a time when hardware costs are diminishing rapidly, an architecture – such as the one described herein – can decrease software development costs and provide relatively easy maintenance. The results described in this thesis demonstrate the feasibility of component-based image processing in a radiological context, something not yet done in this field. Testing of the system on images obtained from a test object (phantom) and a human brain revealed the superiority of 3T MRI system compared to a 1.5T system for image segmentation. A system of the type examined here can support experimental research for a wide range of users including the computer scientist, the biomedical researcher, and the diagnosing physician.

Appendix A

k-Means Algorithm

The k-Means algorithm, sometimes called the Isodata or c-Means algorithm groups a data set into a predefined number of clusters, k. It is called a hard or crisp clustering algorithm because each data point belongs exclusively to a single cluster [12, pp.529–533].

k-Means or c-Means or Isodata algorithm

- Choose arbitrary initial estimates of the centre for each cluster
- while there is no change in the cluster centre
  - For $i = 1$ to N, were N is the number of data points
    - find the nearest cluster centre for data point i, (called cluster j)
    - assign data point i to cluster j
  - For $j = 1$ to k
    - update the cluster centres as the mean of the data points assigned to each cluster.

Appendix B

Bio-Medical Study

B.1 Introduction

The goal of the biomedical study was to evaluate the practicality of differentiating between the processing (segmentation) of images obtained from a 1.5T MRI machine vs. images obtained from a 3T machine. An additional point of comparison was through the combination of images from both 1.5T and 3T machines. Qualitative evaluation was performed to determine if features obtained from a combined image gave superior segmentation results to those of either machine alone.

B.2 Bio-Medical Evaluation

B.2.1 Data Acquisition

Data for testing was acquired at St. Joseph's Healthcare Hamilton. Images from two separate sources were obtained and subjected to segmentation: a test object (phantom) and the author's brain. A phantom was used in order to be able to test the accuracy of the segmentation algorithms by looking for known volumes. The author's brain was used in order to test the algorithms on real anatomical data. Radiology department of the St. Joseph's Healthcare Hamilton generously provided access to their 1.5T and 3T MRI machines for the work reported in this thesis. All images were acquired on June 1, 2004.

The phantom was composed of honeydew melon and avocado chunks suspended in gelatin. The materials were picked at the suggestion of Dr. Mike Nose-

worthy, directory of MRI research facility at St. Joseph's Healthcare Hamilton, such that they could be distinguished when viewed as an MRI image. Unfortunately, only one modality showed the difference between avocado and melon. This was the 1.5T T_1 modality, seen in Figure B.1. The light coloured block on the right side of the image is avocado, while all other squares are melon chunks. The lower piece on the left hand side is also melon. Volumes for the melon and avocado were determined by measuring the amount of water the chunks displaced when submerged. The volume of gelatin was not recorded. The volume measurement was taken so that quantitative analysis of the accuracy of the segmentation algorithm could be performed.

Three sets of images per subject from each MRI machine were acquired, each of the three sets using a different modality. The modalities used were T_1 weighted, T_2 weighted and PD weighted.

B.2.2 Pre-Processing

In order to improve the bio-medical results, some pre-processing was done on the images using the AFNI (Analysis of Functional NeuroImages) tools. AFNI is a set of C language tools that facilitate processing and display of MRI and fMRI image data (`http://afni.nimh.nih.gov/afni/`) in a MRI facility. It is recommended that AFNI algorithms be applied to enhance the quality of images obtained for a study. For example, the AFNI registration algorithm modifies the images so that all pixels in each of the images are in line with one another. Such processing effectively removes differences in orientation between images caused by subject movement. It also aligns the images that were taken on the different MRI machines, since it is impossible to place the subjects in exactly the same orientation after moving between machines.

Figures B.2 and B.1 show the resultant images for the human brain and phantom respectively. In both the human brain and the phantom, slice 33 was used for 2-D testing; this slice is presented here.

The pre-processing discussed here was performed externally to the MIPA system. Section 6.5 discusses the potential inclusion of these pre-processing steps in the MIPA system.

B.2.3 Design of Test Cases

Three test cases were run for each of the two test objects. Test case number one used all three modalities from the 1.5T MRI machine. Test case number two

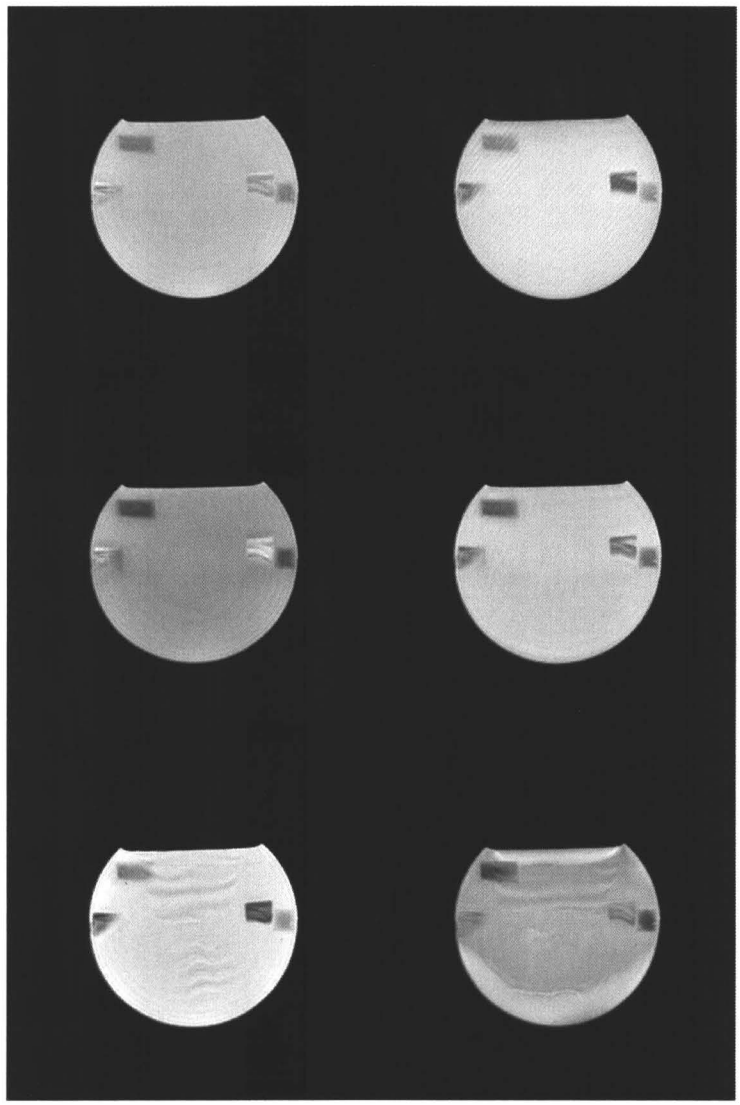


Figure B.1: Original phantom images: left column 1.5T, right column 3T. Top to bottom: Proton Density, T_1, T_2.

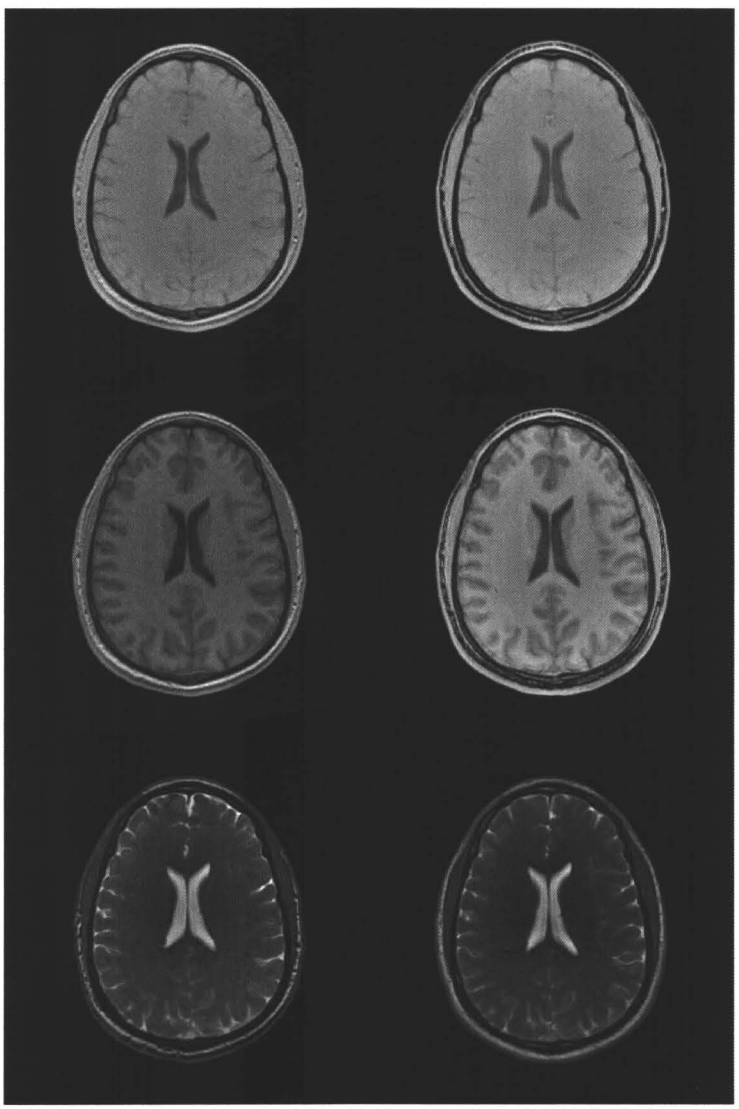


Figure B.2: Original human images: left column 1.5T, right column 3T. Top to bottom: Proton Density, T_1, T_2.

used all three modalities from the 3T MRI machine. Test case number 3 used all three modalities from both the 1.5T and the 3T MRI machines.

The test cases each performed k-Means clustering in search of 3, 4, 5, 6, 7, and 8 clusters. After the clustering, a histogram was calculated for each of the results. The histogram was computed so that the volumes of the clusters could be calculated and compared with the volumes recorded for the phantom.

Sections B.2.4 and B.2.5 discuss the results obtained from the test runs on the phantom and human test objects respectively.

B.2.4 Identification of Disparate Clusters in a Phantom

As described in Section B.2.1 the only image (Figure B.1) where the avocado was visually distinguishable from the melon was the 1.5T T_1 image. 1.5T image processing provided a segmentation of melon and avocado in separate clusters while 3T image processing did not. Figure B.3 shows the original 1.5T and 3T T_1 images as well as the most reasonable results for 1.5T image processing in 2-D and 3-D. The 3-D processing provided far superior results over 2-D processing.

Volume comparisons using histograms were not attempted because it could be visually determined that the results were not what were expected, i.e. classification based on pixel intensities were not successful, and therefore volume comparisons would not provide rational results. Because the differentiation between melon and avocado is difficult for humans, these results are not indicative of a failure of the software. Better materials for construction of the material need to be researched. Section B.3 discusses this further.

B.2.5 Results from Processing of Images of the Human Brain

Test cases using less than 5 clusters did not provide any comparable results, and so were discarded. Figure B.4 shows all the results obtained for test cases using 5 clusters. 5 cluster segmentation shows demarcation of various anatomical structures. Raw T_1 images are also shown in 1.5T and 3T to provide a basis for comparison.

Obtaining Useful Results

As can be seen from Figure B.5 images from the 1.5T machine, the 3T machine and the combination were able to provide good results in at least one test case. The 1.5T case was able to provide useful results only when the processing

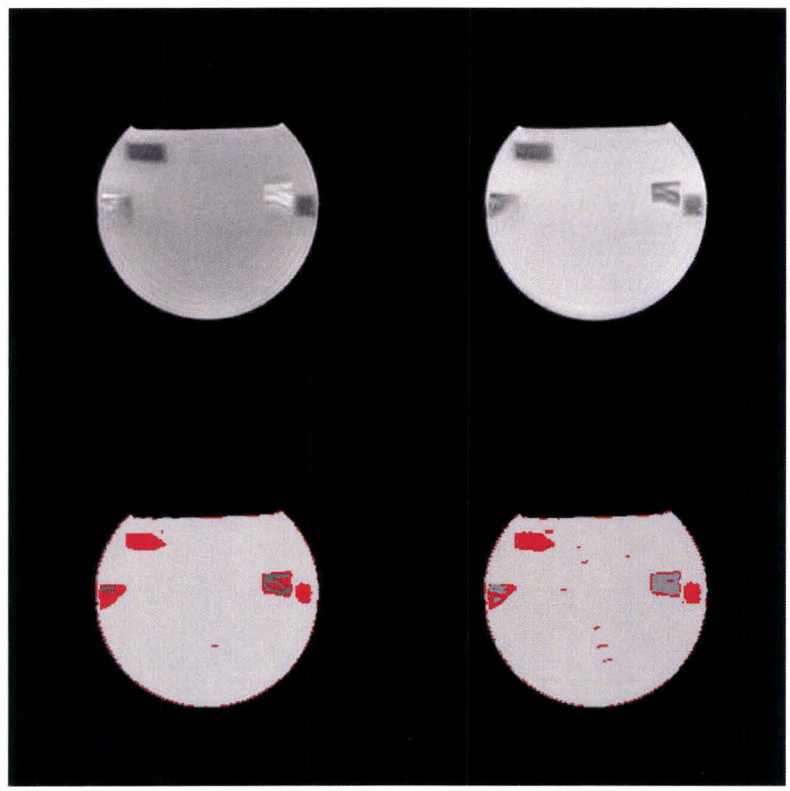


Figure B.3: Phantom results where melon and avocado are identified as different structures: top left original 1.5T T_1, top right 3T T_1, bottom left 1.5T 2-D 7 clusters, bottom right 3T 3-D 8 clusters.

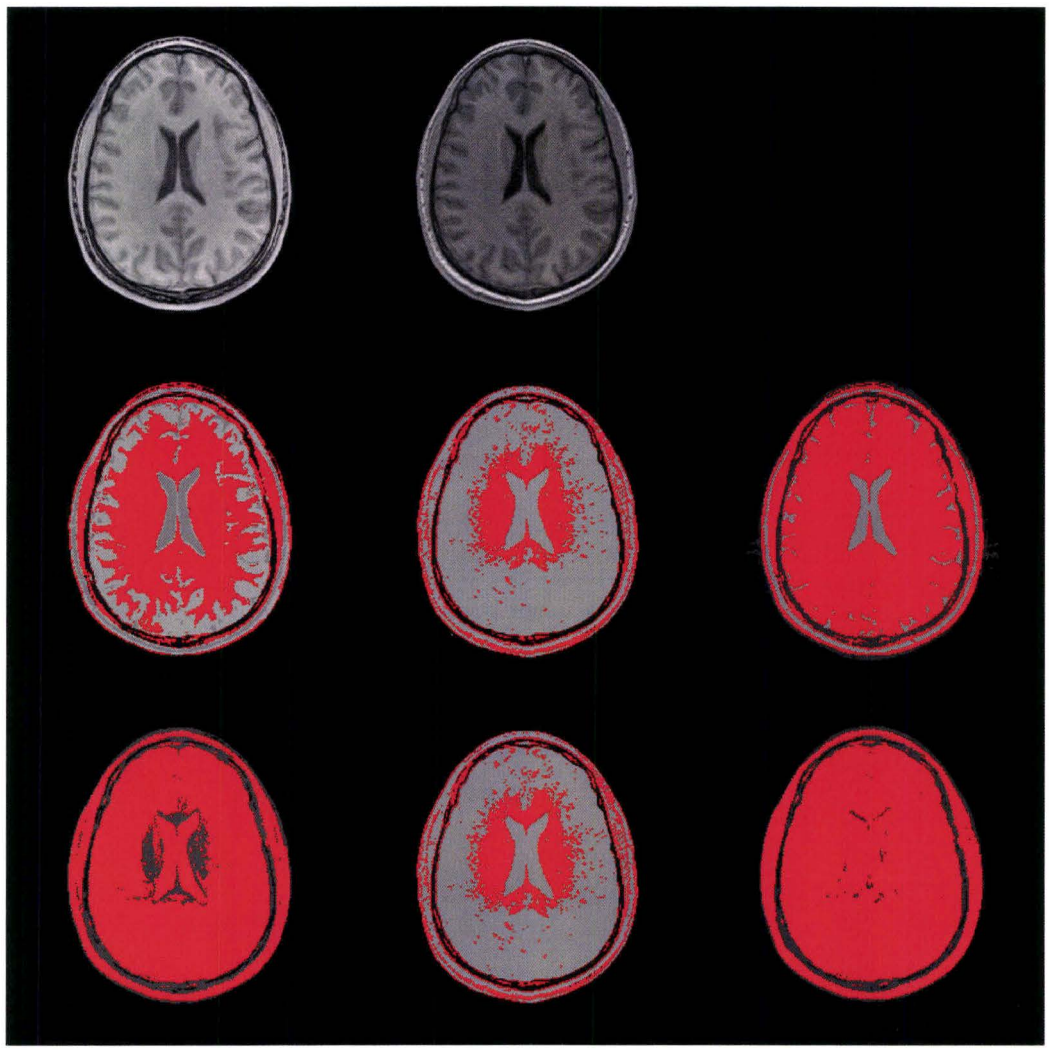


Figure B.4: All images segmented with 5 clusters: top row are original T_1 weighted images, middle row are 2-D and bottom row are 3-D. The left column is 1.5T; the middle column is 3T; the right column is a combination of 1.5T and 3T.

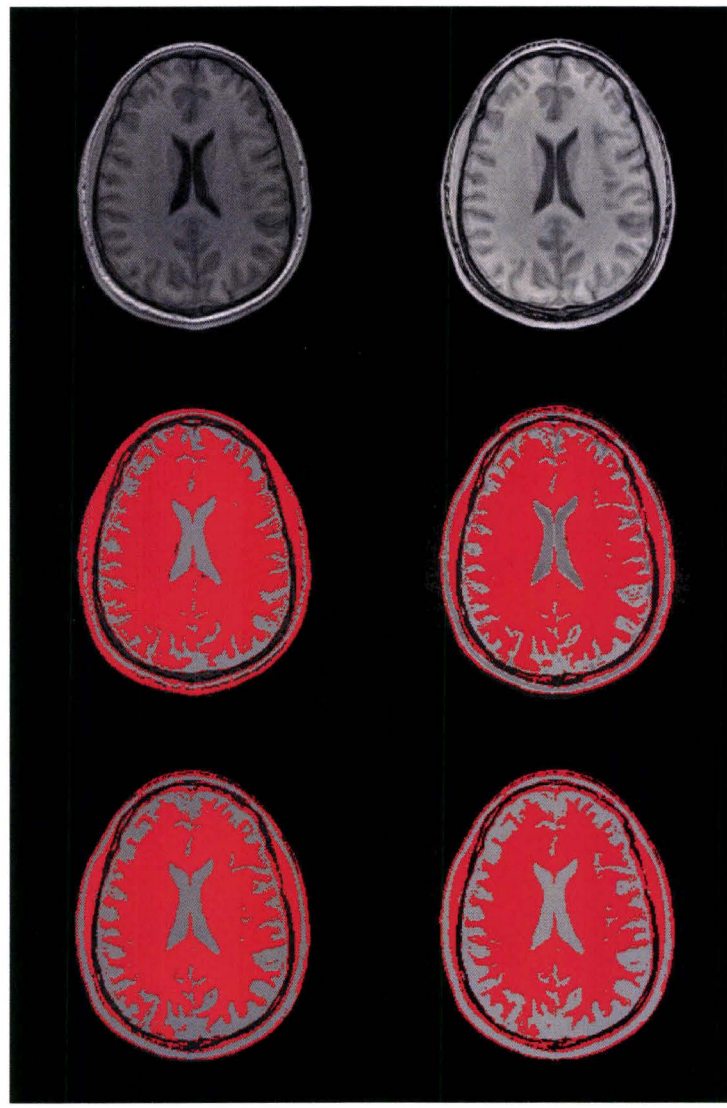


Figure B.5: Reasonable human results: top left to bottom right: 1.5T T_1 weighted original, 3T T_1 weighted original, 1.5T 3-D 6 clusters, 3T 2-D 7 clusters, both 3-D 7 clusters, both 2-D 6 clusters

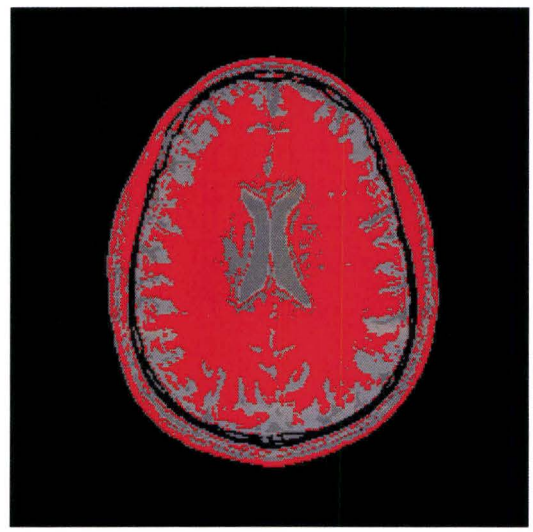


Figure B.6: Human results for 1.5T in 2-D with 8 clusters.

acted on the entire 3-D image, rather than just a single slice. The best single slice image for the 1.5T results can be seen in Figure B.6. The 3T case was only able to produce useful results when acting on a single slice at a time. Figure B.7 shows the best result of processing the entire 3-D 3T image set. Using 3T and 1.5T images together seemed to produce the best results overall. Using both, reasonable results were returned for 2-D and 3-D image processing; also, the segmented image most closely matched the input. As was expected, the results from processing 3T images alone appear slightly better than the results of processing 1.5T images, but the comparison is probably invalidated because the valid results for 1.5T are based on 3-D processing while 3T results are based on 2-D processing. It is important that the attending physician make the conscious decision regarding the optimum number of clusters bases on his clinical expertise.

B.3 Discussion

The use of a phantom did not fully emulate the human brain as effectively as was hoped. The gelatin vibrated during scanning which probably added to the noise found in these images. In addition, the percentage of gelatin should have been lower to increase the role of melon and avocado in the images. Selecting ob-

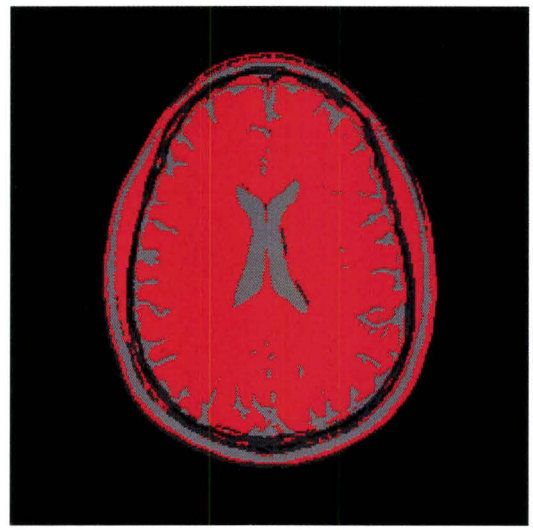


Figure B.7: Human results for 3T in 3-D with 7 clusters.

jects with greater differentiation than avocado and melon would have also helped to improve the results.

Reasonable results were obtained from processing images from the human brain. We can clearly show an improvement in using both the 1.5T images and the 3T images together. One hypothesis for the result is that the images captured at each of the two field strengths have different deficiencies, allowing them to act in a complimentary fashion when analyzed together.

Pixel location did not play any part in the image processing. All results are based solely on pixel intensity. Adding locality information features such as those used by Matsui, Suganami and Kosugi [15] would probably improve results dramatically. These results demonstrate, however, that even when using a simplistic approach to segmentation, useful results can be achieved.

Appendix C

User's Guide

C.1 Graphical User Interface

C.1.1 Starting and Stopping the Program

To start the application run the following from the distribution directory:

```
lambacck $ python clusterviewer.py
```

You will get a window that looks like Figure C.1.

Click on the quit button or the window manager's close button to exit. If the application is currently doing image processing, it may take some time to interrupt this process and close successfully.

C.1.2 Modes

The program has two operational modes. The first mode is the processing setup mode. The second mode is the reviewing results mode.

Processing setup mode is entered by clicking on the new button located in the top left corner of the window. This causes a file chooser dialog (Figure C.2) to open, allowing the user to select the location for saving processing results. After selecting an empty folder, the program will be ready to configure a new processing session (Section C.1.3).

If processing was already conducted in another program session, the results can be loaded by clicking the open button at the top left. The *open saved results dialog* (Figure C.3) will allow the user to select the folder containing the results to be reviewed. After selecting the saved results to be reviewed, the program will be ready to review processing results (Section C.1.4).

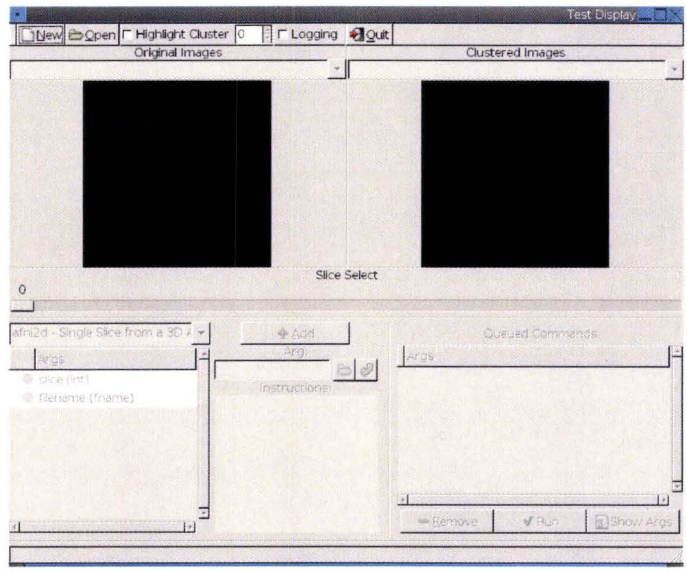


Figure C.1: Initial screen presented to user.

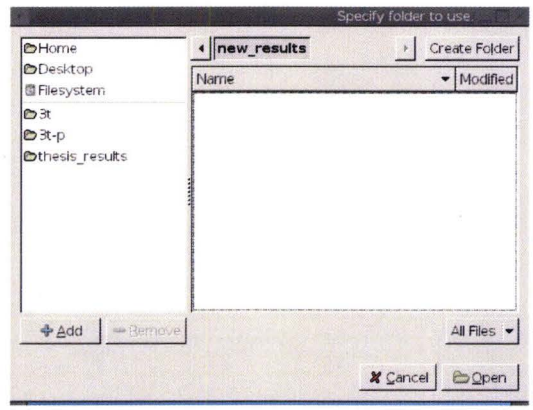


Figure C.2: New processing task output folder selection dialog.

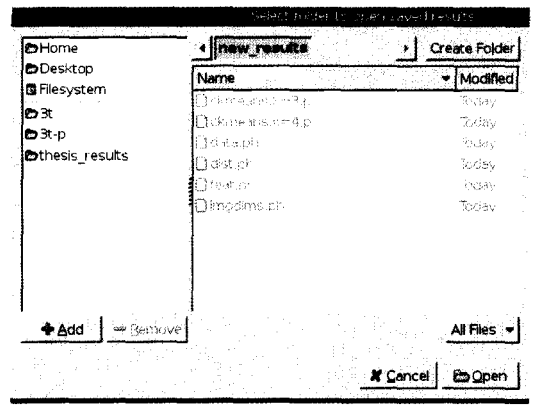


Figure C.3: Open Saved Results Dialog

C.1.3 Configuring a New Processing Session

After selecting a folder to save results, as described in Section C.1.2, the program will be ready for the user to configure a new processing session (Figure C.4). Defining a processing session consists of three tasks, Adding Commands (Section C.1.3), Reviewing Commands (Section C.1.3), and Generating Results (Section C.1.3).

Processing sessions are command driven. Commands are divided into several types: image loading, distance calculation, feature selection, main processing, and post processing. Any image processing session requires an image loading command, a distance calculation command, and a processing command in order to run. Command execution order is determined first by type, in the order given above, and then by order given by the user.

Adding Commands

The bottom left portion of the screen is the command definition area. In this area available commands can be chosen from the command drop down box (Figure C.5. Arguments for the command are shown in the argument list below the command drop down. To the right of the command drop down and argument list are the add button, argument editing area and argument instruction area. The add button is disabled until all necessary command arguments are given.

The argument list shows completed arguments with a green dot, while uncompleted arguments have a red dot. Arguments are edited by selecting the argument

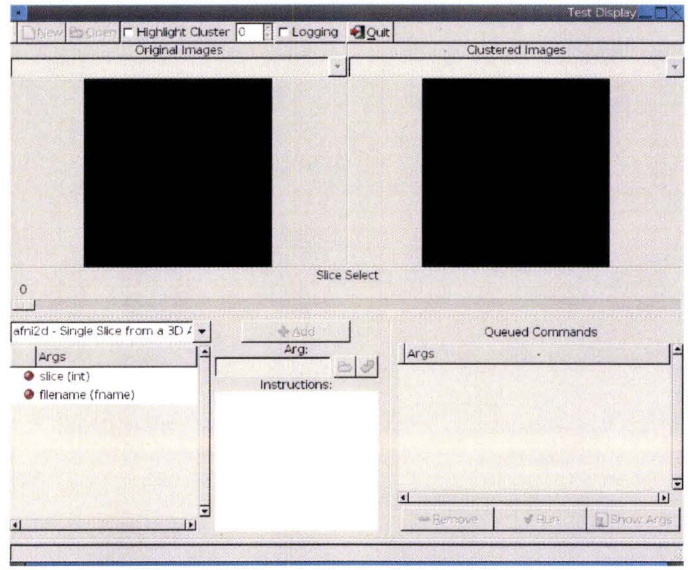


Figure C.4: Ready to define a processing session.

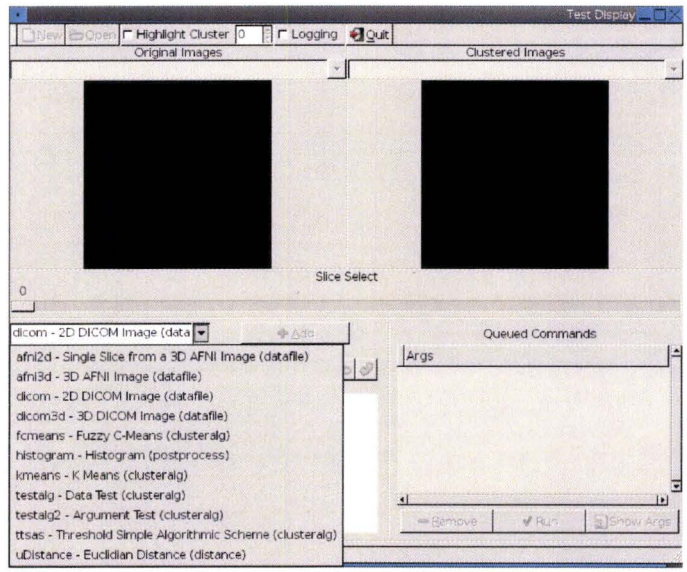


Figure C.5: Available commands are selected from drop down box.

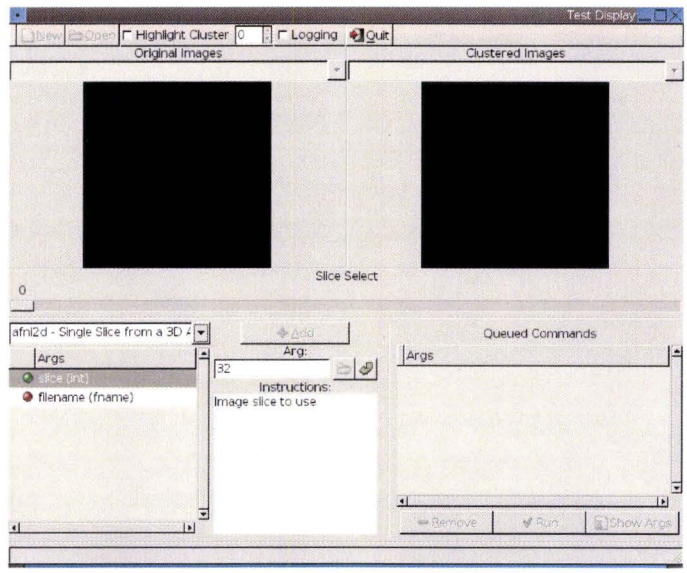


Figure C.6: Status indicators for completed arguments turn red.

from the argument list and typing in the argument edit box. Selecting another argument or clicking the accept button will save the argument value. Once an argument is saved, its completed indicator will change from red to green (Figure C.6.

If the argument requires one or more filenames, the file selector button will become enabled (Figure C.7) and clicking it will provide the file selector dialog (Figure C.8). By default the file selector will limit the visible files to those of the type desired by the command argument. Figure C.8 shows a file selector dialog limited to AFNI files.

Some arguments have sub arguments (Figure C.9). These sub arguments expand out in a tree to allow each sub argument to be completed. Each individual sub argument also has a completed indicator. When all necessary sub arguments have been defined, the completed indicator for the main argument will become green (Figure C.9).

Once all necessary arguments are completed, the add button will be enabled (Figure C.10). Clicking the add button will add the completed command the Queued Commands list (See Section C.1.3).

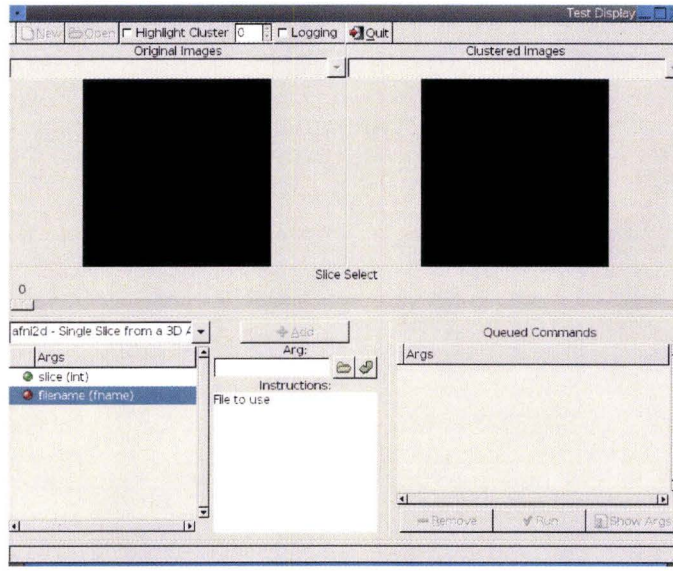


Figure C.7: The file selector button is enabled when editing an argument that expects a file name.

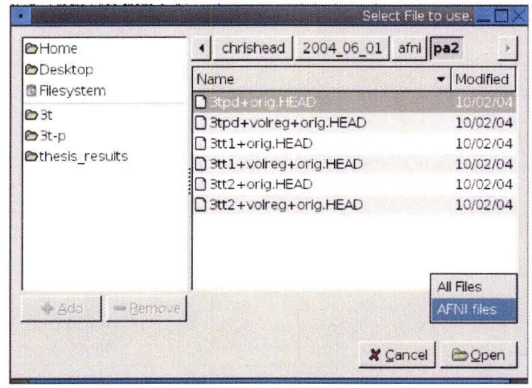


Figure C.8: File Selection Dialog

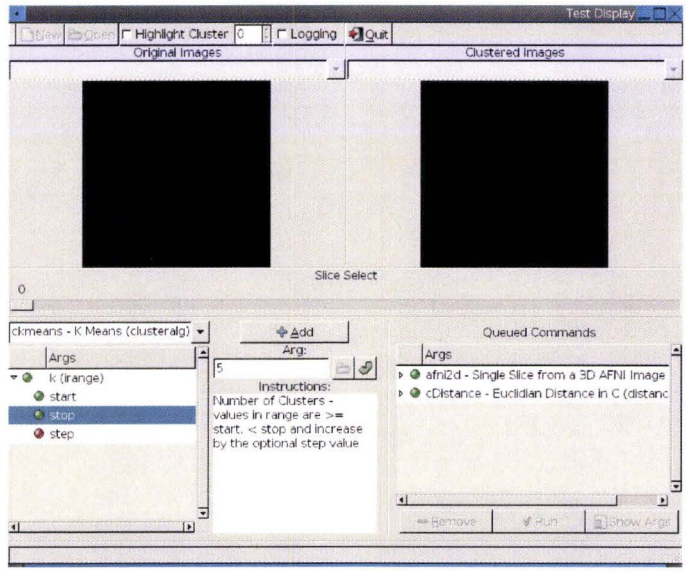


Figure C.9: Some arguments are optional.

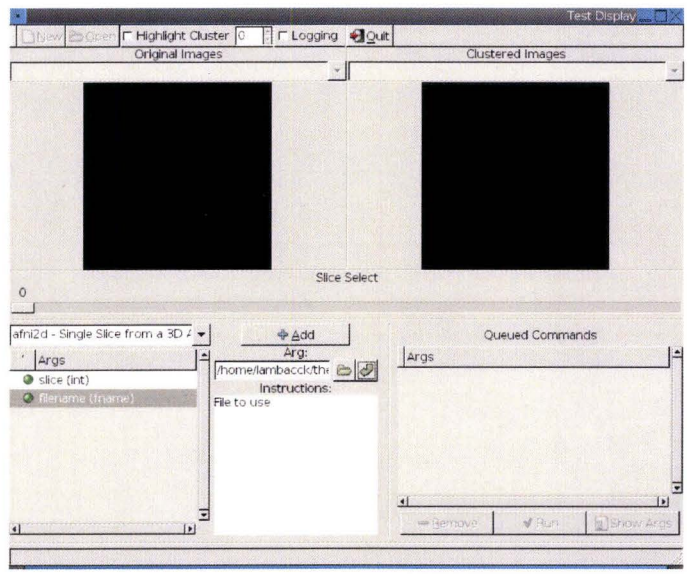


Figure C.10: Command is ready to be added.

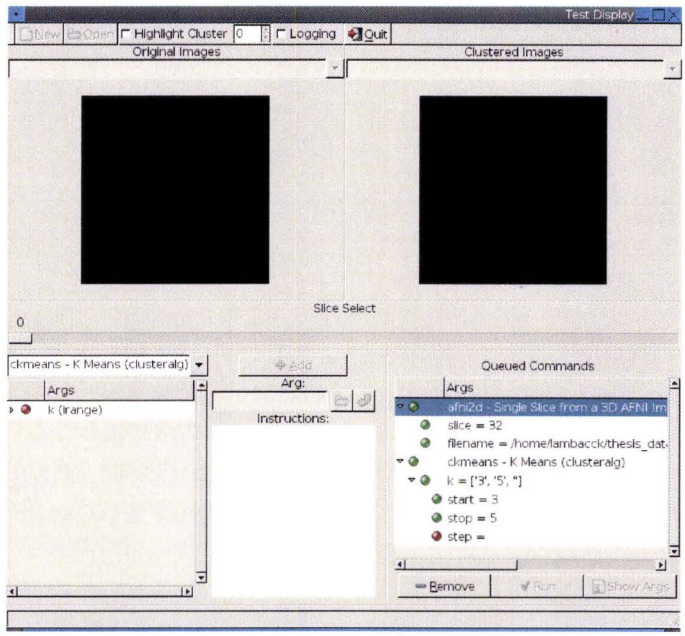


Figure C.11: Commands Being Reviewed

Reviewing Commands

The bottom right side of the display shows the queued commands (Figure C.11). Commands are added to the queue by clicking the add button on the left (See Section C.1.3). Commands expand out in tree style to show their argument values and sub-arguments. Arguments that are not required and not filled out are marked with a red dot in the status area. Commands can be removed from the queue by selecting the command to be removed and then clicking the remove button.

Once at least one data source command, distance calculation command, and processing command have been added to the queue, the run button and the show args button will become enabled (Figure C.12). Clicking the run button will make the program begin to process results (See Section C.1.3). Clicking on the show args button will show print the commands out to the log window (See Section C.1.5) in a format suitable for use with the command line interface to the system (See Section C.2).

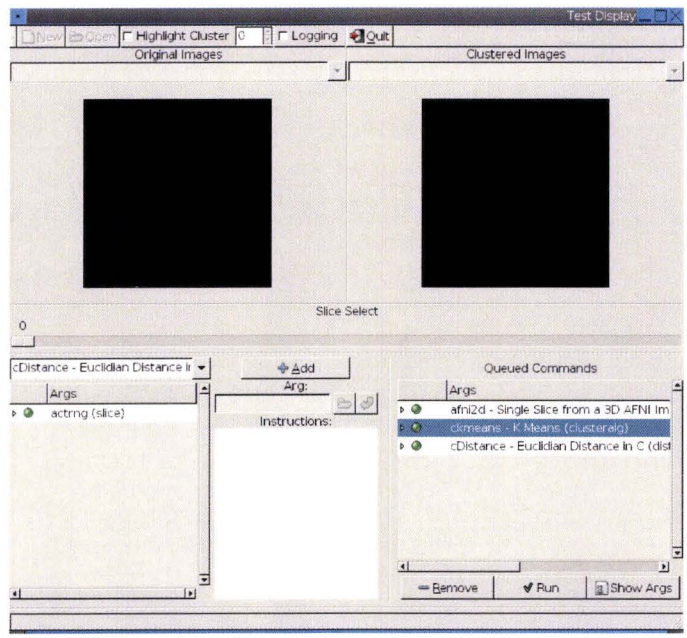


Figure C.12: Sufficient commands are defined to enable the run button.

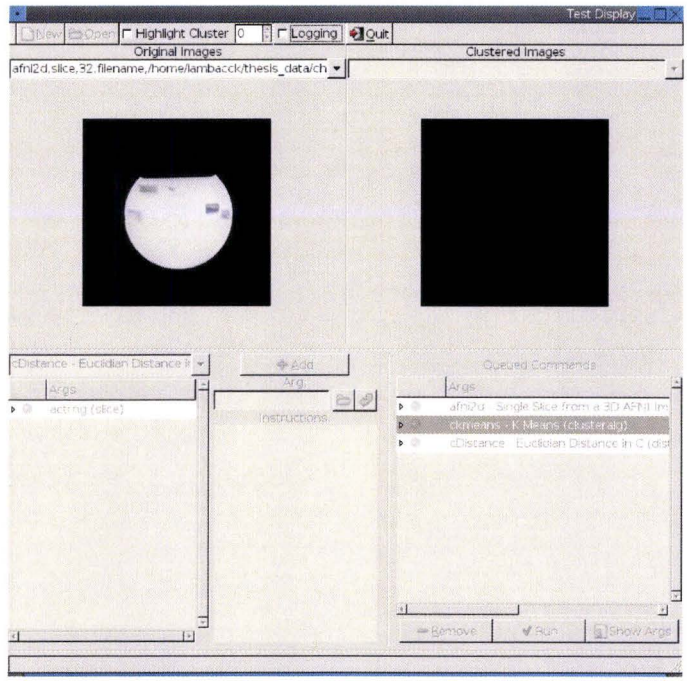


Figure C.13: Source images are shown after processing has started.

Generating Results

When the desired commands have been added to the queued command list (See Section C.1.3), clicking the run button will cause the system to start generating results. The user interface will then be placed in results mode (Section C.1.4) so that results can be reviewed as they are returned by the system (Figure C.13). Initially only the source images will be available for review. As results are generated they will also be made available for review (Figure C.14). As results are generated, some performance data will be printed to the log window (See Section C.1.5).

C.1.4 Reviewing Results

The Graphical User Interface allows the user to compare original images against processed images when in result review mode (Figure C.14). This mode is entered by clicking the open button when the application is first started (See

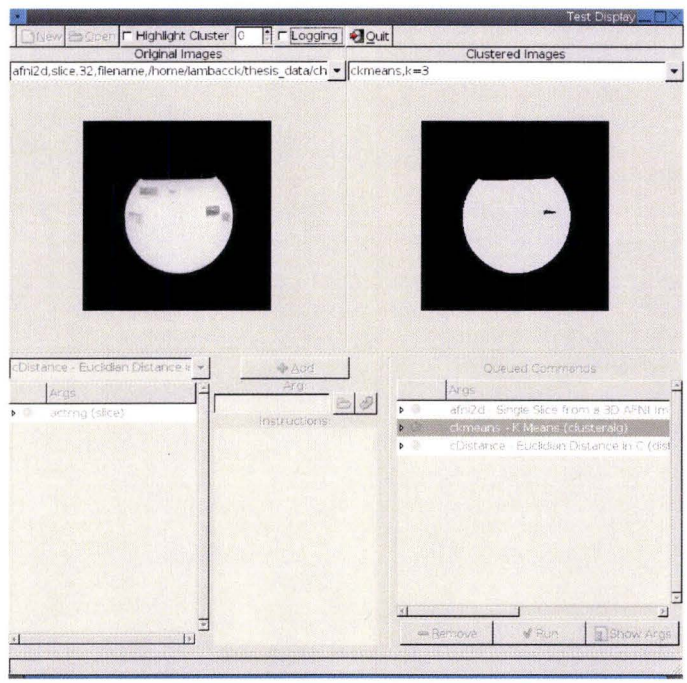


Figure C.14: Results are shown as they become available.

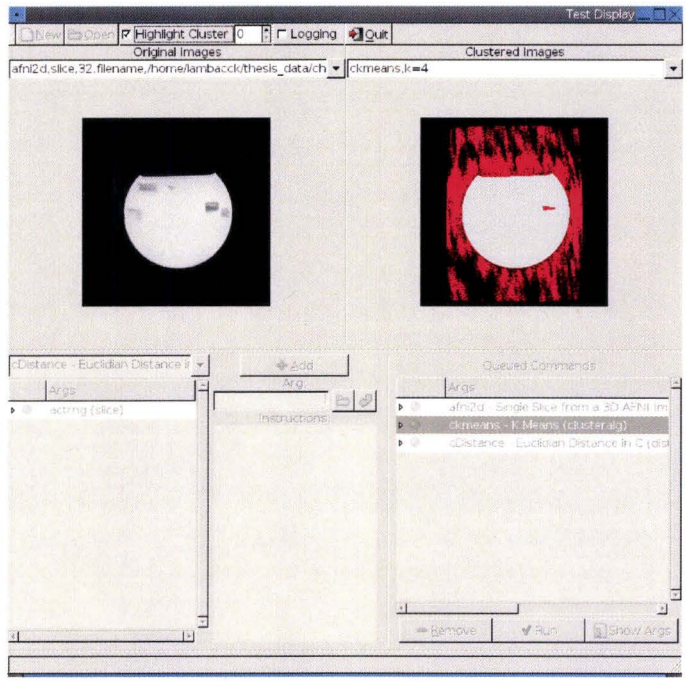


Figure C.15: A result image with cluster zero being highlighted.

Section C.1.2) or by clicking the run button to generate results (See Section C.1.3).

The left hand side of the screen shows the source image selected by the user from the original image drop down box. The right hand side of the screen shows the resulting processed images, which are selected from the clustered images drop down.

Often clustering results are not obvious. The highlight cluster features allows the user to select a cluster for the software to highlight. This feature is enabled by clicking the highlight cluster check-box at the top of the screen (Figure C.15). The number next to the highlight cluster check-box indicates which cluster is currently highlighted and allows the user to select the cluster to highlighted. Figure C.15 shows cluster zero being highlighted while Figure C.16 shows cluster two being highlighted.

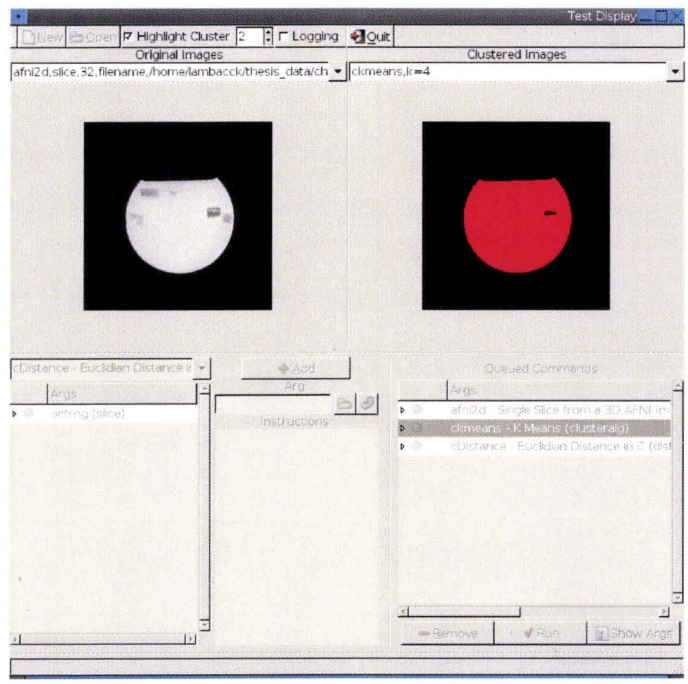


Figure C.16: A result image with cluster two being highlighted.

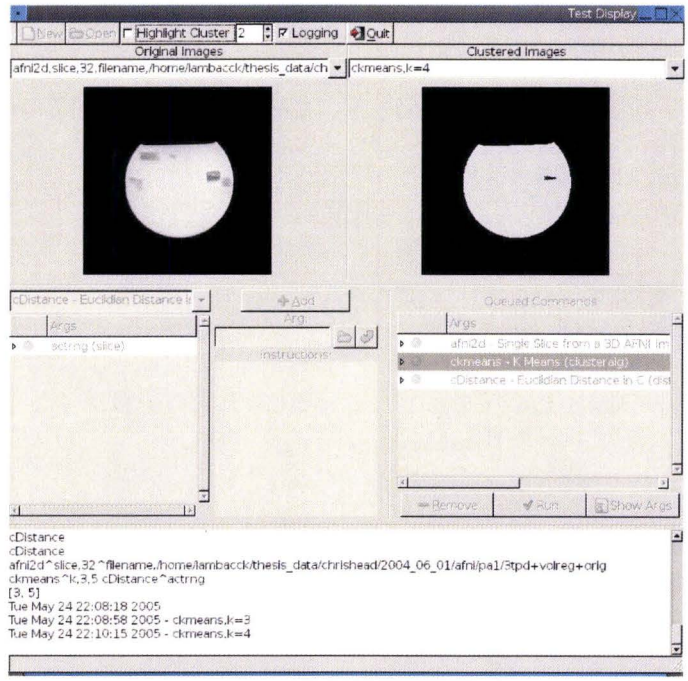


Figure C.17: The log window is enabled.

C.1.5 The Log Window

The Graphical User Interface provides a log window which can be shown or hidden at the user's request. By default it is not shown. The log window can be toggled by clicking the log window check box at the top of the screen. Figure C.17 shows the log window containing some performance data and output generated by the show args button (See Section C.1.3).

C.2 Command Line Interface

The command line interface is implemented by the cmdline.py script. cmdline.py is executed as follows:

```
python cmdline.py targetdir command
```

where targetdir is the directory to save results to and command is one or more space separated commands for the system to process. The format for command is the name of the command and the arguments for the command separated by

carat (‘^’) characters. The arguments consist of the argument name an its parameters separated by commas (‘,’). An example is “`python cmdline.py ~/results "afni2d^filename,~/data/input^slice,33" "cDistance^actrng" "ckmeans^k,3,8"`”

Performance data will be output to the console at runtime. Results can be viewed in the Graphical User Interface (Section C.1).

C.3 Implemented Commands

Several commands are available for use. This section outlines what they do and what arguments they take.

C.3.1 Input Commands

afni2d

Input a particular slice from a 3-D AFNI file.

Argument	Description
filename	The name of the file to use
slice	the image slice number to use

afni3d

Input a 3-D AFNI file.

Argument	Description
filename	The name of the file to use

dicom

Input a 2-D DICOM file.

Argument	Description
filename	The name of the file to use

dicom3d

Compose several 2-D dicom files into a 3-D image.

Argument	Description
filename	The names of the files to use

C.3.2 Distance Commands

uDistance

Euclidean distance.

Argument	Description
actrng	the input range to calculate the distance on (comma separated start, stop pair, one or both are optional)

cDistance

Euclidean distance implemented with Pyrex.

Argument	Description
actrng	the input range to calculate the distance on (comma separated start, stop pair, one or both are optional)

C.3.3 Processing Commands

kmeans

k-Means clustering algorithm.

Argument	Description
k	number of clusters to generate (comma separated start, stop, step triple; step is optional)

ckmeans

k-Means clustering algorithm implemented in Pyrex. Use with cDistance.

Argument	Description
k	number of clusters to generate (comma separated start, stop, step triple; step is optional)

C.3.4 Post Processing Commands

histogram

Calculate histogram on cluster results.

Argument	Description
f	filename to write the histogram to

Appendix D

Developer's Guide

D.1 Overview

Components in the MIPA system are Python modules that expose the MIPA defined component interface. Any valid Python module, either a C extension or a text based script, will work. The MIPA system will attempt to register all Python modules placed in the plugins directory. Registration allows the component to tell the system what commands it provides.

See the Python Programming Language documentation (`http://www.python.org/doc/`) for information on programming in Python and creating Python modules. Image data is represented as Numeric Python Arrays. See the Numeric Python documentation for information on using Numeric Python (`http://numeric.scipy.org/`).

D.2 Command Registration

Each component (Python module) must define a function called `register` (See Listing D.1) which returns information about the commands that the component supports. This information is returned as a list of `registration.RegistrationInfo` objects. Listing D.1 gives an example defining 2 commands, cmd and filecmd.

Line 1 brings the RegistrationInfo class from the registration module into the current name-space. Line 3 begins the definition of the register function.

On Lines 4 and 7, the RegistrationInfo objects are created. The first argument to the constructor is the name of the command; a class of this name will also need to be defined in the module. The second argument to the constructor is a short

Listing D.1: The registration function.

```
1  from registration import RegistrationInfo
2
3  def register():
4      c = RegistrationInfo('cmd','Command Desc','clusteralg')
5      c.addParam('arg', 'Arg Desc', 'irange')
6
7      f = RegistrationInfo('filecmd','FileCmd Desc','postprocess')
8      fargs = {'filter': [('*.txt', 'Text Files')]}
9      f.addParam('filename', 'Filename Desc', 'fname', fargs)
10     f.addParam('i', 'I Desc', 'int')
11
12     return [c, f]
```

Table D.1: Command Types

Command Type	Description
datafile	File Loading
distance	Distance Calculation
feature	Feature Extraction
clusteralg	A clustering algorithm (main processing)
postprocess	Post-processing of main processing output

description of the command. This should clearly describe what the command does, for instance a command for the k-Means algorithm might be called `'kmeans'` and have a description like `'k-Means Clustering Algorithm'`. The third and final argument to the constructor is the type of command. Command types are shown in Table D.1.

On Lines 5, 9 and 10 arguments are added to the commands with the `addParam` method. The first argument to `addParam` is the name of the argument. The second argument to addParam is a description, which should clarify the purpose of the argument being added. The third parameter is the argument type. Table D.2 lists the argument types and the values they accept. Line 9 shows the use of the optional fourth argument to `addParam`, filename args. This argument is only valid with an argument type of `'fname'` or `'fnamel'` and provides extra information for file

Table D.2: Argument Types

Argument Type	Description
str	A string argument
int	An integer argument
float	A floating point argument
fname	A filename argument
fnamel	A list of filenames
irange	A range of integers denoted as a start, stop, step triple where step is optional
frange	A range of integers denoted as a start, stop, step triple where step is optional
slice	A start, stop pair where both start and stop are optional positive integers. If none are given, whole range is to be used, if one is given it is taken as stop and start is taken as 0

handling to the user interface. This argument takes a python dictionary, in this case defined on Line 8. Table D.3 lists of dictionary keys and their function.

D.3 Command Definition and Semantics

Each command type requires that an object be defined with a particular interface. Each command takes some arguments as defined during registration. The arguments are passed as a dictionary where the keys are the argument names and the values are the values provided by the user. Some commands take these arguments at construction, while others take the arguments during a method call. The following sections describe interfaces.

D.3.1 File Input: datafile

File input is performed by commands of type datafile. The constructor for a datafile command is passed the argument dictionary. The only other interface that a command of type datafile must export is the getArray method which takes no arguments and returns a one dimensional Numeric Python Array object. Each

Table D.3: File Argument Dictionary Keys and Values

Dictionary Key	Description
strip	A list of suffixes that should be removed from the file name before use
filter	A list of tuples defining file filters. The first element in the tuple is a wildcard that matches the file type, the second element in the list is a name for the type of files this matches

element in the Array object represents one 16 bit pixel value organized moving left to right, top to bottom.

D.3.2 Distance Calculation: distance

Distance calculation is performed by commands of type distance. The constructor for a distance command is passed the argument dictionary. The only other interface that a command of type distance must export is the calculate method which takes two arguments, x and y. x and y are one dimensional Numeric Python Array objects representing vectors. The calculate method returns the result of the distance calculation as either a floating point number or an integer.

D.3.3 Feature Extraction: feature

Feature extraction is performed by commands of type feature. The constructor for a command of type feature is passed the argument dictionary. The only other interface that a command of type feature must export is the getArray method which takes no arguments and returns a one dimensional Numeric Python Array object. Each element int he Array object represents one 16 bit pixel value organized moving left to right, top to bottom.

D.3.4 Main Processing: clusteralg

Main processing is performed by commands of type clusteralg. The constructor for a command of type clusteralg is passed the distance command object. A command of type clusteralg must also define the execute method which takes an argument dictionary and the data to process.

The data is a Numeric Python Array where the first dimension is the pixel position (y * width + x) and the second dimension is each individual feature (original data or extracted feature).

The argument dictionary is handled a little bit differently because irange and frange arguments cause execute to be called once for every combination of argument and values in the range. For instance if a command defines arguments i and j of type irange and the user indicates that i should have a start and a stop 1 and 3 while j should have a start and stop 0 and 2, execute will be called 4 times. The values taken by (i, j) will be $(1,0)$, $(1,1)$, $(2,0)$, and $(2,1)$.

Each time execute is called, must return a pair. The first element of the pair is a one dimensional Numeric Python Array object of the same length as the first dimension of the data Array object passed to execute. The Array object represents the cluster that the image pixel belongs to. The second element in the pair is a list of one dimensional Numeric Python Array objects that define the middle of each cluster. each Array object in the list must be of the same length of the second dimension of the data Array object passed to execute.

D.3.5 Post Processing: postprocess

Post processing is performed by commands of type postprocess. The constructor for a command of type postprocess is passed the distance command object. A command of type postprocess must also define the execute method which takes an argument dictionary and the data that was processed. The cluster membership of each clustering run is passed in the argument dictionary as a two dimensional Numeric Python Array object. The first dimension is the pixel location and the second dimension is the cluster run.

Arguments given by the user are passed in the argument dictionary in the same way that they are for the commands of type clusteralg.

The execute method must return some value, but the value is arbitrary, i.e. the system does not care what the value is.

Bibliography

[1] Terri S. Armstrong, Marlene Z. Cohen, Jeffrey Weinberg, and Mark R. Gilbert. Imaging techniques in neuro-oncology. *Seminars in Oncology Nursing*, 20(4):231–239, November 2004.

[2] Jerrold T. Bushberg, J. Anthony Seibert, Edwin M. Leidholdt Jr., and John M. Boone. *The Essential Physics of Medical Imaging*. Williams and Wilkins, 2nd edition, 2001.

[3] Jeong-Ho Chae, Ziad Nahas, Mikhail Lomarev, Stewart Denslow, Jeffrey P. Lorberbaum, Daryl E. Bohning, and Mark S. George. A review of functional neuroimaging studies of vagus nerve stimulation (VNS). *Journal of Psychiatric Research*, 37(6):443–455, November–December 2003.

[4] Federico Giove, Girolamo Garreffa, Giovanni Giulietti, Silvia Mangia, Claudio Colonnese, and Bruno Maraviglia. Issues about the fmri of the human spinal cord. *Magnetic Resonance Imaging*, 22(10):1505–1516, December 2004.

[5] E. Mark Haacke, Norman Y.C. Cheng, Michael J. House, Qiang Liu, Jaladhar Neelavalli, Robert J. Ogg, Asadullah Khan, Muhammad Ayaz, Wolff Kirsch, and Andre Obenaus. Imaging iron stores in the brain using magnetic resonance imaging. *Magnetic Resonance Imaging*, 23(1):1–25, January 2005.

[6] Karl Herholz and W. D. Heiss. Positron emission tomography in clinical neurology. *Molecular Imaging and Biology*, 6(4):239–269, July–August 2004.

[7] Dae-Shik Kim and Michael Garwood. High-field magnetic resonance techniques for brain research. *Current Opinion in Neurobiology*, 13(5):612–619, October 2003.

[8] Venkata S Mattay and Terry E Goldberg. Imaging genetic influences in human brain function. *Current Opinion in Neurobiology*, 14(2):239–247, April 2004.

[9] Rosa Maria Moresco, Cristina Messa, Giovanni Lucignani, Giovanna Rizzo G., Sergio Todde, Maria Carla Gilardi, Adelmo Grimaldi, and Ferruccio Fazio. Pet in psychopharmacology. *Pharmacological Research*, 44(3):151–159, September 2001.

[10] Reginald F. Munden Mylene T. Truong and Benjamin Movsas. Imaging to optimally stage lung cancer: Conventional modalities and pet/ct. *Journal of the American College of Radiology*, 1(12):957–964, December 2004.

[11] Susan S. Spencer, William H. Theodore, and Samuel F. Berkovic. Clinical applications: MRI, SPECT, and PET. *Magnetic Resonance Imaging*, 13(8):1119–1124, 1995.

[12] Sergios Theodoridis and Konstantinos Koutroumbas. *Pattern Recognition*. Academic Press, 2nd edition, 2003.

[13] Ray H. Hashemi, William G. Bradley, Jr, and Christopher J. Lisanti. *MRI: The Basics*. Lippincott Williams and Wilkins, second edition, 2004.

[14] N. Saeed. Magnetic resonance image segmentation using pattern recognition, and applied to image registration and quantitation. *NMR in Biomedicine*, 11(7):157–167, 1998.

[15] Kazuhrio Matsui, Yusuke Suganami, and Yukio Kosugi. Feature selection by genetic algorithm for MRI segmentation. *Systems and Computers in Japan*, 30(7):69–78, 1999.

[16] Alberto Martelli. Application of heuristic search methods to edge and contour detection. *Communications of the ACM*, 19(2):73–83, February 1976.

[17] Matthew C Clark, Lawrence O Hall, Dmitry B Goldof, Laurence P Clarke, Robert P Velthuizen, and Martin S Silbiger. MRI segementation using fuzzy clustering techniques. *IEEE Engineering in Medicine and Biology Magazine*, 13(5):730–742, 1994.

[18] Chih-Wei Chang, Gilbert R Hillman, Hao Ying, Thomas A Kent, and John Yen. A two-stage human brain MRI segmentation scheme using fuzzy logic.

In *IEEE International Conference on Fuzzy Systems*, volume 2, pages 649–654, March 1995.

[19] Jing-Hao Xue, Aleksandra Pizurica, Wilfried Philips, Etienne Kerre, Rik Van De Walle, and Ignace Lemahieu. An integrated method of adaptive enhancement for unsupervised segmentation of MRI brain images. *Pattern Recognition Letters*, 24(15):2549–2560, 2003.

[20] M. J. Kwon, Y. J. Han, I. H. Shin, and H. W. Park. Hierarchical fuzzy segmentation of brain MR images. *International Journal of Imaging Systems and Technology*, 13(2):115–125, 2003.

[21] Colin S Poon and Michael Braun. Image segmentation by a deformable contour model incorporating region analysis. *Physics in Medicine and Biology*, 42(9):1833–1841, 1997.

[22] John M. Boone, Greg S. Hurlock, J. Anthony Seibert, and Richard L. Kennedy. Automated recognition of lateral from PA chest radiographs: Saving seconds in a PACS environment. *Journal of Digital Imaging*, 16(16):345–349, 2003.

[23] Kitware Inc. The visualization toolkit home page. `http://public.kitware.com/VTK/`, April 2005.

[24] Rick Avila and Lisa Sobierajski. The VolVis project. `http://www.cs.sunysb.edu/~vislab/volvis_home.html`, April 2005.

[25] T.S. Yoo, M. J. Ackerman, W. E. Lorensen, W. Schroeder, V. Chalana, S. Aylward, D. Metaxes, and R. Whitaker. Engineering and algorithm design for an image processing API: A technical report on ITK – the insight toolkit. In J. Westwood, editor, *Medicine Meets Virtual Reality*, pages 586–592. IOS Press Amsterdam, 2002.

[26] The ITK Project. National library of medicine insight segmentation and registration toolkit. `http://www.itk.org/`, April 2005.

[27] Marcel Bosc and Torbjrn Vik. The ImLib3D homepage. `http://imlib3d.sourceforge.net/`, April 2005.

[28] Sarang Lakare and Arie Kaufman. OpenVL: the open volume library. In *Proceedings of the 2003 Eurographics/IEEE TVCG Workshop on Volume graphics*, pages 69–78. ACM Press, 2003.

[29] Sarang Lakare. The OpenVL home page. http://openvl.sourceforge.net/, April 2005.

[30] The VIM Editor Project. The vim editor home page. http://www.vim.org/, April 2005.

[31] The Mozilla Organization. Mozilla orgainization home page. http://www.mozilla.org/, April 2005.