...en Faculty of Mathematics and Natural Sciences

Department of Mathematics and Computing Science

A Graphical User Interface for Automated Image Matching

Rolf Janssen

Advisor:
dr. J.B.T.M. Roerdink

April, 1999

RuG

A Graphical User Interface for Automated Image Matching

Rolf Janssen

Advisor:
Dr. J.B.T.M. Roerdink

13th April 1999

Preface

This thesis discusses the design and implementation of a uniform graphical user interface for image matching algorithms or packages. The program is called "Automated Image Matching" or AIM for short. The reasons for the decision to develop this program are described. Furthermore an introduction into image modalities and image matching in general is given. A design with user recommendations and requirements is given. The implementation is discussed in detail and a comparison is made between the actual implemented features and the design requirements. As appendices the user and technical manuals are also included. The user manual describes the way the program should be used and the technical manual describes the implementation in detail, sometimes even to the code level. The user manual is meant for actual users and the technical manual is mainly meant for people who wish to continue development of the program or who want to know more of the inner workings of AIM. The implementation described here supports two matching packages, Automated Image Registration (AIR) by Roger P. Woods and Multi-Resolution Mutual Information (Alignms) by Alle Meije Wink.

Contents

List of Figures

Chapter 1

Introduction

Medical images are used in hospitals to provide information for making diagnoses and treatment planning. Over the years new digital medical imagery and scanning techniques have become more and more important. The standard X-ray is still the most used technique, but newer scanning techniques are used increasingly, often to visualise phenomena that the normal X-rays are unable to register. These days it is common for patients to be imaged with more than one tomographic radiological imaging modality. The newer scanning techniques are able to create three-dimensional images, instead of the two-dimensional projection provided by an X-ray image. Three of those scanning techniques are:

- **Computed Tomography (CT):** this technique uses X-rays to acquire information. A röntgen tube spins around the patient and takes about a several thousand projections. The X-rays are picked up by an array of sensors. All this collected data is used to calculate a two-dimensional slice of the scanned region. Several slices can be combined to produce a three-dimensional image. The images will show the density of the scanned region; especially the bone structures can be detected by using CT-scanning. These bone structures show up white (high density) on the images. Since this technique uses X-rays, it is limited in time and resolution, in case of living subjects, because of limited allowable radiation. Figure 1.1 shows an example of a CT image.

- **Magnetic Resonance Imaging (MRI):** this technique uses magnetic radiation to acquire information. A MRI unit consists mainly of a large cylindrical magnet, devices for transmitting and receiving radio waves, and a computer. During examination, the patient lies inside the magnet, and a magnetic field is applied to the patient's body. The magnetic field causes magnetic spins in nuclei in certain atoms inside the body to line up. Radio waves are then directed at the nuclei. If the frequency of waves equals that of the spins, a resonance condition occurs. This condition enables the nuclei to absorb the energy of the radio waves. When the radio-wave stimulation stops, the nuclei return to their original state and emit energy in the form of weak radio signals. The strength and duration of these signals depend on various properties of the tissue. A computer then translates the signals into highly detailed cross-sectional images. Soft tissues show up very well on MRI (sometimes called MR) images. Bone tissues show up black in an MRI image, in contrast to CT, where bones are white. Figure 1.2 shows an example of an MRI image.

- **Positron Emission Tomography (PET):** this technique involves injecting a patient with a radio-labeled biologically active compound called a tracer, which decays through the emis-

sion of a positron particle. This particle then annihilates with an electron from the surrounding tissue, emitting two gamma rays which are detected in coincidence by a scintillation gamma camera. Once the data is collected, special algorithms and computer programs produce a 3-D image of the patients anatomic distribution of biological processes. Interpreting these images can be difficult for an untrained person, since most of the PET images do not show anatomic information, such as bones. Image matching with PET images uses a so called water-PET with ^{15}O as a tracer, which shows anatomical structures containing water. Figure 1.3 shows an example of a PET (i.e. a water-PET) image.

Each of these three scanning techniques has specific merits. For example, on CT bone structures are very prominent, but on MR soft tissue is more visible. What is wanted is a way to combine the different types of images (modalities), so all qualities of each type are combined. This cannot be done by just simply adding the images, because the subject hardly ever is in the same position during acquisition in all three scanners. Even between different scans on the same scanner the subject's position is not always the same. If the subject is the same in different scans, then there is only a rigid body transformation (rotation and translation) between the images. Another problem is that the voxel sizes and dimensions of the images are not necessarily the same.

This transformation, which consists of three translations (in the x-, y-, and z-directions) and three rotations ($pitch$, $roll$ and yaw), can be found by manually (on the computer) transforming the image and visually inspecting it to see if the result is correct, or even by holding two images to be matched in front of a light source and comparing the images visually and transforming them mentally. However this can be very inaccurate and difficult, and with the current computers this is not necessary.

This is where "automated image matching" algorithms are used. These algorithms aim to find the transformation between two images "automatically", meaning without user action, which will align the two images in such a way that the corresponding structures are in the same place on both images.

There are various image matching algorithms, both theoretical and actually implemented. Each algorithm has its advantages and disadvantages. Most work only well for specific modalities, therefore it is sometimes necessary to use a specific algorithm for specific modalities.

Since there are several image matching algorithms or packages (this report uses the term algorithms, to indicate both algorithms and packages), there are also a number of interfaces for these algorithms. Many algorithms do not have a Graphical User Interface (GUI) or only have a very simple GUI. Most algorithms simply have a command-line interface. The algorithms that do have a GUI, each have their own way of doing things. This is fine in some cases, it is however not very user friendly, since the user has to learn how to use the interface of each algorithm.

For the two reasons above the program Automated Image Matching (AIM) has been developed. This program provides one uniform graphical user interface for several image matching algorithms.

The current version of AIM supports two different algorithms, but can be extended to support other algorithms. Currently the two supported algorithms are:

- **Automated Image Registration (AIM):** an algorithm developed and implemented by Roger P. Woods. This algorithm was originally intended for PET-PET matching [5] and was later extended to allow for PET-MR matching [6]. See [7] for information about the latest implementation.

- **Multi-Resolution Mutual Information (Alignms):** this algorithm was developed by C. Studholme [3]. It is based on a multi-resolution approach. It uses soft tissue correlation and mutual information to measure the misregistration. The supported implementation is the one made by Alle

Meije Wink [1]. Testing has shown that Alignms (Align Multiple Slices) can match CT-MR, CT-PET and MR-PET very well.

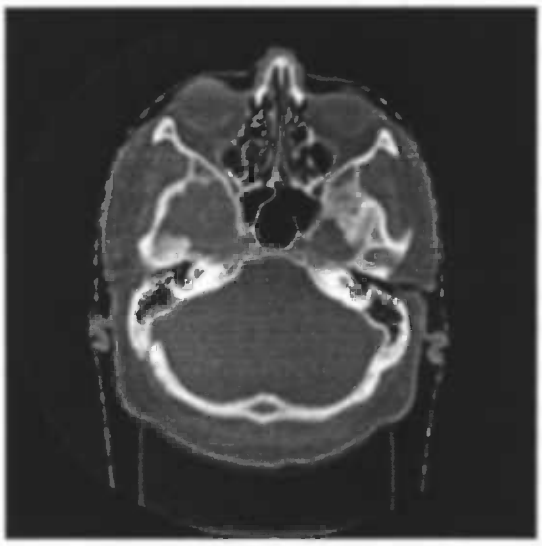

Figure 1.1: CT image

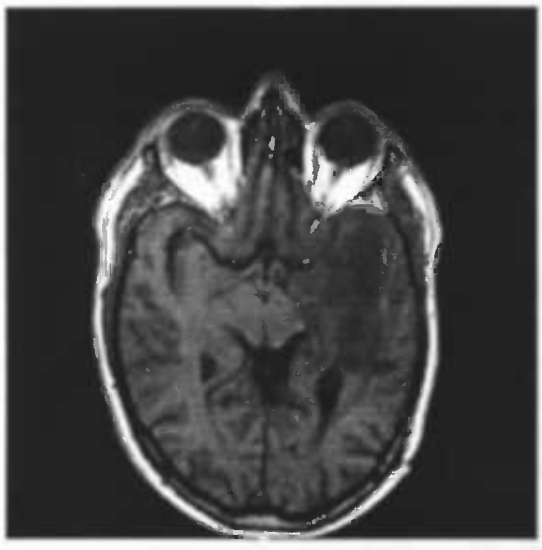

Figure 1.2: MR Image

1.1 Image Matching

What an automated image matching algorithm does is finding a transformation which will align two images. Most algorithms use two important steps for matching images:

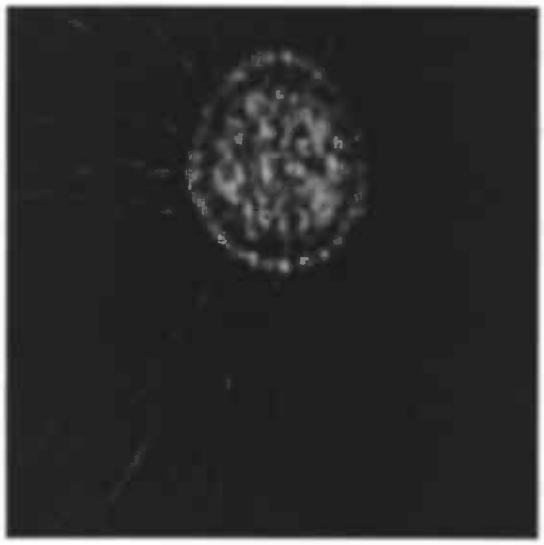

Figure 1.3: PET Image

- The algorithm has a method to measure the "similarity" between two images. An example of such a similarity measure is mutual information [4].
- The algorithm has a method to maximise or optimise this measure. An example is Studholme's multi-resolution method [3].

Most algorithms provide several ways to enhance the correctness of the match by adding additional features, such as thresholds. Thresholds are used to remove unwanted structures in images and which can disrupt a correct match of the images. The simplest way to implement a threshold is setting the gray values of voxels with values below the threshold to zero, but this can cause problems in MRI images, where bones show up black. In CT and PET images this type of thresholds can easily be used. For example in CT images a cushion or a head mask can be seen in some images such as Figure 1.1. These features can be very prominent and make it difficult to get a good match. However choosing a threshold must be done very carefully, because it might also remove relevant information.

Masking is another often used technique to increase the correctness of an image match. If you know that the scanned object is a head, then you could use an ellipsoidal shape around the head to remove any values that lay outside that mask. A mask is like a threshold, it will also make some voxels not count in the match, but it is much more selective of the region that is not used.

However there are some limitations to most algorithms, for example it cannot be expected that an algorithm matches images that are very apart from each other, for example by a rotation of 180 degrees. For this reason most algorithms provide a way to give an initial transformation. This does require user actions, but can increase the correctness of the final match.

A scanning technique is called a "modality", for example CT, MR and PET. There are two main types of matching:

- **Intramodality:** this is matching between images of the same type, for example: MR-MR, PET-PET and CT-CT.
- **Intermodality:** this is matching between images of different types, for example: MR-PET, PET-CT, CT-MR.

Most image matchings are performed intrasubject, meaning that the images are of the same subject (i.e. patient). Intersubject matchings are not often performed, since these require non-rigid body transformations and this is not supported in most matching algorithms.

Image matching requires two images, one that is used as a reference and another as the one that needs to be transformed. The two images needed for matching are called:

- **Floating Image:** the image that is transformed to match to the reference image. After matching this image is usually resliced to the voxel and image dimensions of the reference image and the transformation that was found by the match is applied (reslicing and transforming are done at the same time).
- **Reference Image:** the image that the floating image is matched to. This image is not transformed.

Sometimes other names are used for these images. An usual name for the floating image is "reslice image" and for the reference image the name "standard" image is also used. The term "resliced" image is used by AIM to indicate a resliced floating image.

The images that are used have several dimensions:

- **Image dimensions.** For a three dimensional image these are $width \times height \times depth$ ($= x \times y \times z$), expressed in integers. The depth is the same as the number of slices. The depth of a two dimensional images is 1, it has just one slice. More than 3 dimensions are also possible, but displaying these images is not supported by AIM. The normal dimensions of a slice are 128×128, 256×256, 320×320, 512×512. Other dimensions are also used. The number of slices varies widely, it usually depends on how big a region has been be scanned.
- **Voxel dimensions.** For three-dimensional images these are $width \times height \times depth$, expressed in millimetre (floats). The depth is the same as the slice distance, which is usually larger then the width and height. For a 2D image the depth does not matter. The voxel dimensions for PET images are usually bigger then those of CT and MR. The PET image used for testing has voxel dimensions $3.129 \times 3.129 \times 3.375$ millimetre. The CT used has voxel dimensions $0.78 \times 0.78 \times 3.0$ mm and the MR has voxel dimensions $0.898 \times 0.898 \times 6.0$ mm.

The applied transformation is a 6-parameter rigid body transformation. Rigid body means that the actual sizes and geometry of the object are not changed. There are 3 translation and 3 rotation parameters. The three translation parameters are called: t_x, t_y and t_z, and are in the x, y and z direction respectively. Usually the unit for these translations is voxels/pixels or millimetres (mm). The three rotation parameters are called: $pitch$ (in YZ plane), $roll$ (in XZ plane) and yaw (in XY plane). The pivot point of these rotations is the centre of the middle slice. The unit for rotations is degrees.

The transformation is applied to the floating image, usually during the same time as it is resliced. Reslicing will change the image and voxel dimensions of a floating image to the values of the reference image. Almost always some sort of interpolation is used to calculate the new voxel values. Reslicing will not change the real-world dimensions of the image, but only the dimensions and voxel sizes. If an object is 10 mm, it will still be 10 mm after reslicing.

Most algorithms assume that the object is rigid and therefore is of the same shape in each image, with the only difference being a translation and/or rotation. For this reason some care should be taken to fixate scanned objects.

Chapter 2

Design

2.1 Introduction

This chapter will describe the design of the program Automated Image Matching (AIM).

2.2 "End-User" Requirements

When designing a program, the possible "end-users" of that program need to be considered.

The end-user' wishes play an important role in program design. If the program does not meet the requirements of the "end-users", it will quickly be left unused or users get annoyed when using it. Of course one cannot accommodate all the end-users wishes, but they should be as much as possible taken into consideration.

We consulted a number of possible "end-users" and asked them the following questions:

1. What programs are used by you that have a GUI and perform image matching ?
2. What are the positive points of those programs ?
3. What are the negative points of those programs ?
4. What do you want in an "image matching" program ?
5. What do you not want in an "image matching" program ?

We contacted people from the "PET Centre", the "Radiotherapy" department and the "Radiology" department, all of the Academic Hospital Groningen (AZG).

2.2.1 Programs

Examples of programs that have image matching capabilities are:

SPM: Statistical Parametric Mapping. This program is widely used in medical research. The program is based upon Matlab (not the Windows version, which does not use Matlab). The GUI is limited to the interface that can be produced with Matlab. The matching algorithm is based on the least squares method. SPM requires knowledge of most parameters, therefore it is not very useful for the inexperienced user.

IMIPS: Integrated Medical Image Processing Systems. This program has many features, which include: 3D rendering, fusion, matching, image import and scalping. A demo version of IMIPS is downloadable from `http://www.imips.com`. The matching algorithm is based on the Woods algorithm, see [5] and [6]. The Woods algorithm has somewhat been modified for speed. The GUI is based on AVS. It is a bit overloaded with all kinds of widgets. Order and/or placement of widgets is not always logical. There only are a few selectable parameters. IMIPS makes a difference between 'functional' and 'structural' images and sets the parameters according to the selected modality.

AIR: Automated Image Registration. Package from Woods, which uses its own algorithm. The GUI is based on Tcl/Tk, which only provides an input option for parameters. The GUI calls the different AIR programs with the correct parameters. The problem with this is that if such a call is not correct, then no feedback is given. The GUI does not give access to all parameters. For validation of AIR see [2] and [1]. For more information on the algorithm see [5] and [6].

Since there are not many "image matching" programs used at the AZG, other non-image matching packages were also looked at. Other programs discussed:

TMS: Treatment Management System. A planning system for patients. This program is used at the radiotherapy department. For information on how it works see §2.2.2. Matching is limited to manually defining a transformation. The GUI is complicated, but then again it is a complex system.

AFNI: Analysis of Functional NeuroImages. A program for different kinds of medical image processing. The GUI is a bit overloaded with widgets.

2.2.2 "End-user" Recommendations/Requirements

Radiotherapy

At the radiotherapy department a system called TMS is used. All patient images are inserted into this system. By default a CT image is made of the complete body of the patient. This CT image, called CT_{rt} (rt=radiotherapy) is (always) used as reference image. The CT_{rt} images are scanned with the slices perpendicular to the scanning table. For diagnostic use, other images are also produced, being CT_d (d stands for diagnostic), MR_d and PET_d. These three diagnostic images must be matched onto the reference CT_{rt} image.

A characteristic of this TMS system is that it is very strict with the images it gets. If there is even the smallest error in the header then it will not accept the image. The system also gets image info from the file name.

Recommendations/requirements:

1. Matching modalities: CT reference and MR, CT and PET as floating images. The reference CT always is an axial CT, with the slices taken perpendicular to the table. The floating images can also be diagonal.

2. Reslicing: after matching the floating image should be resliced to match CT_{rt} parameters.

3. Input/output: by default DICOM should be used. A Philips DICOM type called GECOM is also used, but this format will probably be replaced by DICOM. For PET images the Siemens ECAT format is usually used. Another suggestion was to use the CT_{rt} header for the resliced

floating image. This can be done since the floating image is resliced to CT_{rt} parameters. So the CT_{rt} header can be put in front of the resliced floating volume.

4. Filenames: influence on result filenames must be possible. A selectable prefix (e.g. r for resliced) should suffice.

5. Image viewing: 2D slice by slice via a slider or a complete study in one window. When moving through the reference image, the resliced floating image should also be showing the same slice. Of course this behaviour should be selectable. 3D viewing is not needed.

6. The user should be able to select a starting transformation. This can be done via manually inputting markers in the reference image on recognisable features, e.g. the eyes.

7. After matching it should be possible to view the "correctness" of the match in different ways, for example a checkerboard with the two images, presented in alternating order. This validation is very important, since a match must be correct.

8. It should be possible to do pre- and postmatch operations independent of matching, so one can open the reference image and resliced floating image and then use the 'validation' methods, without matching first.

9. Not all program/matching parameters should be shown to the user. A factory default option is useful. Default settings should be saved/loaded.

10. When saving an image, the slices needed for output should be selectable.

11. In the image viewer the Z distance should be shown. The Z distance is not the same as $slice\_size * slice\_number$, because slice 0 is not always the zero z-coordinate.

12. Care should be taken that the correct axes are used. Images are always seen from the feet of a patient and are dependent of the scanning table.

13. Since only Windows 9x/NT is used at the radiotherapy department, it should be possible to port AIM to that platform.

14. It may be useful that the user can remove parts of the images. Some features might be distracting, therefore they should be removed.

15. Usage of "level of window". This allows the changing of the gray value range. Normally there are 256 gray values available for displaying, the way the original image values are mapped onto these values can be changed. Example: the image uses values 0 to 4000, then instead of mapping all the values onto the available 256 values, only the first thousand (0 to 1000) are used.

PET Centre

The only recommendation, was that the program should have a GUI and a command-line. This way the program can be used with scripts.

Radiology

The following programs were demonstrated:

- Gyroview from Philips. An old X-ray program. Runs on Unix.
- A system from Agfa. An medical image review program. Runs on Unix.
- Applicare (Radworks) from Applicare. An medical image review program. Runs on Windows NT.

There where a number of remarks regarding graphical user interface of these three programs. The most important ones were:

1. A GUI needs to be consistent. All three programs had a different interface for the same function.
2. Icons can be very confusing, when they do not reflect their usage. The program from Agfa uses many icons, which are very confusing. A normal button with text is more useful most of the time.

2.3 Design Requirements

In our design we make a distinction between the functional and GUI requirements. The first concern what functions AIM should have. The second concern how the GUI should be.

2.3.1 Functional Requirements

Based on the "end-users" research and our own wishes, we come to following list of functional requirements:

1. Image Input/Output:
   (a) Different medical image formats are supported for both input and output. Supported formats: Analyze (read/write), DICOM (read/write) and Siemens ECAT (read-only).
   (b) Images can be 2 or 3 dimensional.
   (c) Multi-file 3D images support.
   (d) Different non-medical image formats (e.g. gif) are supported.
   (e) Separate slices of 3D images can be saved as 2D images.
2. Parameters:
   (a) Loading and saving of parameters from and to files.
   (b) Usage of default parameter file(s).
   (c) Default parameter file can be specified/changed.
   (d) Parameters that do not require change are hidden from users.
   (e) Parameters can be set to "factory default".
3. Other functions:

(a) Opened images can be viewed in an image viewer. 3D images are shown slice by slice, with the slice selectable. Resliced image viewer can be connected to Reference image viewer, so they show the same slice with moving only one slider.

(b) A starting transformation can be given, if the matching algorithm supports it.

(c) Matching is started and stopped from within AIM.

(d) AIM gives feedback when an algorithm call is invalid (unlike AIR, which shows nothing in such a case).

(e) After matching, AIM can perform reslicing.

(f) Matching results can be viewed as a resliced/fused image, or only the resulting transformation is shown.

(g) Matching results are written to file: a transformation matrix to ASCII-file and images to their own formats.

(h) Matching multiple "floating" images to one "reference" image can be performed.

Note that performing "image matching" is not one of the functions of AIM. AIM does not perform image matching, but calls the image matching algorithms. How those algorithms/program perform matching is not important to AIM, as long it gets results from them. AIM provides the front-end of matching, not matching itself.

2.3.2 GUI/Program Requirements

Besides the several functions AIM has, there are also different requirements of the GUI/program itself.

1. User interface:

   (a) Easy to use and "intuitive". New users should not have to study large manuals, before they are able to use AIM.

   (b) For flexibility AIM uses different windows for different components. The program is not based on one window, but uses multiple windows. The number of visible windows should be limited to a few. Examples of windows are: image window(s), parameter window(s) and the main window.

   (c) The GUI does not have a distracting interface, meaning that it is not overloaded with all kinds of visual gadgets.

   (d) Consistency in the interface, meaning that the same thing won't be done in different ways. E.g. a dialog always asks "yes" and "no" questions in one order and not one time "yes/no" and another time "no/yes".

   (e) Almost all functions can be performed with mouse or keyboard. The most suitable device can be used for each action.

   (f) AIM gives feedback. The user will be informed about any changes to the program. Also the user will be informed if some action goes wrong.

   (g) Indication of progress. If the users see nothing that indicates that the program is actually doing something, they tend to get confused. During matching this indication of progress is left up to the algorithm.

(h) AIM has a (context-sensitive) help-system. This way the user doesn't need to search in the manual if he gets stuck.

(i) AIM is also usable from the command-line, without any graphical components.

2. Program:

   (a) Robust. AIM handles "strange" user actions, without crashing or deleting results.

   (b) Memory efficient. AIM uses as little memory as possible, because the matching algorithm will use lots of memory themselves.

   (c) AIM has an adequate response time to user actions. This is partly machine-dependent.

   (d) Independent of matching algorithms. This means that AIM is not made specifically for one matching algorithm, everything will be kept as general as possible.

   (e) New algorithms can be added easily, without having to change any existing code.

   (f) Existing files are not to be overwritten, unless explicitly specified by the user.

3. Matching Parameters:

   (a) Input of parameters inside GUI via various edit boxes/sliders/etc.

   (b) Matching parameters are presented in a logical way, e.g. alphabetical or in order of importance.

   (c) Parameters that do not require change, should not be shown to the user. These parameters should only be available on request.

Note that looking pretty is not a requirement of AIM. Although it should not look "ugly" if possible, it's not a primary requirement. Everything should be focused on functionality.

2.3.3 Normal usage

Normal use of AIM is:

1. Program start.
2. User opens reference image.
3. User opens floating image.
4. User selects matching algorithm.
5. User sets the necessary parameters for the selected matching algorithm or load parameters from file.
6. User starts matching program.
7. After matching is done, the user can inspect the results, can do any of the previous items again or continue with the next item.
8. AIM saves parameters, results and images if needed.
9. Program end.

Chapter 3

Implementation

This chapter describes the implementation of AIM. The different components or windows are described in detail, but not to the level of the actual code. For more technical descriptions see appendix B. The goal was to design and implement a graphical user interface for different image matching algorithms/packages, therefore the main focus of the implementation was on two issues: the GUI and algorithms/packages support.

3.1 Graphical User Interface

The graphical user interface is an important part of AIM, since the goal was to create a graphical user interface that can use different image matching algorithms/packages. Therefore some care has been taken to implement a robust and consistent GUI.

The first thing that had to be decided was how this GUI would be build. When considering this question for an X-Windows/Unix program you will quickly come to choice of a GUI toolkit and programming language. A GUI toolkit provides higher level commands to make a GUI than the direct X-Windows commands provide. Many different toolkits are available, but after some time the decision was made to use the Qt toolkit, which can be downloaded from the Internet at `http://www.troll.no`. Qt uses C++, therefore there is no choice for a programming language. Some reasons for the choice of Qt and C++:

- Qt's structures (i.e. classes) are very well defined and structured. From all class members it is clear what the function is.
- Qt's documentation is very good. All different components are described in detail and many example programs are available. Qt's documentation is written in HTML, which has hyperlinks to all of Qt's members and is therefore very easy to use.
- Qt is programmed in C++. It uses different classes for all components. This gives the advantage that object oriented languages have, for example reuse and subclassing.
- Qt is not difficult to use, it does not take long to learn how to use all the different classes and event handling.
- Qt is free for the Unix/X-Windows platform. An MS-Windows version is available, but this is not free.

3.1.1 Main window

Figure A.1 shows a screen shot of the main window. All image loading/saving is done from here, images can be shown or hidden, reslicing can be started, the matching algorithm can be chosen and the matching can be started and stopped. For the images two notations are used, one with capitals (e.g. Reference Image) and one without (e.g. reference image). With the first a GUI component is meant and with the latter the actual image is meant. The main window consists of the following parts:

- **Reference Image:** this is the image that will serve as "reference image". The file name of the loaded image is shown in the edit box, which can be selected but not directly edited. It cannot be edited since if it was editable the user would assume that he/she can input the file name and that that image then would be loaded. This is however not the case. Images can only be selected via the file dialog. With the "load" button an image can be loaded. After pressing the load button the file dialog will pop up and the user can select an image to be loaded. During image loading a progress bar is shown, which indicates visually how far AIM is with loading this file. The same progress bar is used during file saving. The user has the choice to load the image as little or big endian. By default the host (computer) endian is used, but sometimes the file and host endian are not identical. With the "view" button, which shows the text "view" or "hide" depending on whether or not the image is hidden or shown, the visibility of the image window can be changed. With the "info" button the image information is shown. The info dialog shows all information that has been stored about the image, including all dimensions, patient's name and scan date/time. Some information is not stored with certain image formats, so these values are set to default values or in case of a string value, set to "unknown".

- **Floating Image:** similar to **Reference Image**.

- **Resliced Image:** similar to **Reference Image**, except that there is no "view" button (this function is available in the main menu), but it has a **"Reslice Floating"** button. The "Reslice Floating" button will start the reslicing of the floating image. The transformation shown in the current parameter window is applied during reslicing. After reslicing this resliced file is loaded into the "Resliced Image". The option to reslice after a match can be turned on or off with the "After Match Reslice Flo. Image" checkbox, but it is not useful to turn this option off, since all matching information can be lost after the match. The "Resliced Image" does not have to be an actual resliced image, but this is its primary use. It also doesn't have to be a matched (and also resliced) file, but it is used to display the matched and resliced image.

- **Matching Algorithm/Package:** with the Matching Algorithm combo box the algorithm to be used can be selected. After selecting an algorithm, the parameter window belonging to that algorithm is shown. Of course only one algorithm at a time can be selected. The selection of the algorithm can also be done from the main menu.

- **Matching Parameters:** there are two buttons, through which the user can load and save the parameters from the currently selected algorithm. Each algorithm has its own parameters, so for each algorithm a different file should be chosen. Items in a parameter file that are not available for an algorithm are not loaded. Parameters for an algorithm, which do not exist in a parameter file, are set to their factory default value.

- **Match Validation:** these options are for the validation of matches. Validation needs two images and with the two radio buttons the choice which images to use can be made: the reference image

with the resliced or floating image. The two images must have the exact same image and voxel dimensions. After selecting which two images to use, the "Validate" button can be pressed to show or hide the fusion window.

We have presented these components in normal order of use, meaning that the user first loads two images, the reference and floating images, then selects the matching algorithms, and then the parameters are loaded or saved. The last thing the user should do is start the match or preview the result of the initial transformation by reslicing and using the fusion window. The image windows are also placed below the corresponding image group boxes.

All functions performed by the buttons on the main window, can also be done from the main menu. This menu also has some extra items, for example: saving of images and repositioning the image windows. These extra functions do not have buttons in the GUI, but are only available from the main menu, to prevent an increase the number of buttons in the GUI, which can be distracting.

3.1.2 Image/Fusion window

An image window is used to show one image. Figure A.2 shows an example of the image window. A fusion window shows two images, therefore it has some extra components. Figure A.3 shows an example of the fusion window. The components that are common to both image windows are:

- **The image:** this is a 2D image taken from the dataset, from a certain depth (i.e. slice) and view direction. The fusion window needs two images.
- **Slice slider:** with this slider the slice from the image that is to be shown can be selected. Where this slice is taken from depends on the selected plane. Clicking left or right of the slider pointer, will increment or decrement the slice number by one slice.
- **View:** with this combo box the displayed view direction can be selected. There are three view directions, if an image is 3D, being: XY, XZ and YZ. The slice to be chosen by the slice slider depends on which view direction is shown. If the view direction is XY then the slice slider slides along the z-dimension, but if the view direction is YZ then the slice slider slides along the x-dimension. The same goes for the XZ view direction, this slides along the y-dimension. This means that the "depth" is always the direction not shown in the view direction name.
- **Zoom:** with this a zoom factor can be chosen, which will increase or decrease the image size depending on the chosen zoom factor. The zoom factors ranges from 0.25 to 3.0. By default the value 1.0 is used, which shows the real size. Choosing a zoom factor of 2.0 on a 320×320 image will increase the image size to 640×640. The actual image stays unchanged, only the image that is displayed is resized (uninterpolated).

As said above, the image and fusion windows both can show three-dimensional images in three view directions, being the XY, XZ and YZ view directions. Figures 3.1, 3.2 and 3.3 are slices taken from different view directions, from one image.

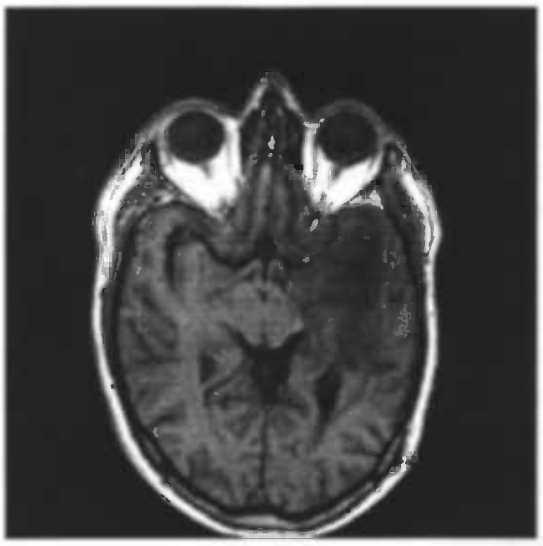

Figure 3.1: MRI image, view direction XY

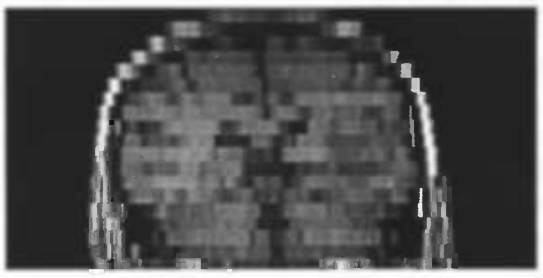

Figure 3.2: MRI image, view direction XZ

Figure 3.3: MRI image, view direction YZ

If the image window is displaying the floating image, the displayed image can be translated and rotated. This rotation and translation shows up on the initial transformation parameters of the parameter window. If the user enters transformation parameters into the parameter window then these values are applied to the floating image.

When right-clicking (i.e. clicking the right mouse button) on the displayed image a menu will pop up, through which some extra features can be accessed. These features are:

- **Save Slice to File:** this saves the current displayed image slice to file. This file will be bmp format and has the name "aim_savedslice_x.bmp" (except the first image, which is called "aim_savedslice.bmp"), with 'x' being a number that will start from '1'. If '1' exists then '2' is used, and this number is increased until a number is found that is not yet used. The image that is written has the same image dimensions as the image that is displayed. To be sure, there are two types of images, one that is displayed and one that is originally read from file and which stays unchanged and from which slices are taken to display. An example: if the original image has a dimension of 256×256 and a zoom factor of 2.0 is used then the saved image will be 512×512.

- **Correct Image:** this applies a function that will do some operations on the original dataset. For example: the real minimum and maximum are calculated and set, the bytes are swapped[1] if needed and/or negative values are made positive. Sometimes it is needed to apply this function before matching or reslicing, especially with AIR. This function can also be used when the header and data are not of the same endian and therefore the image looks really terrible.

Fusion window

The fusion window can display two images at a time. It is necessary that the two images are of the same image and voxel dimensions. Because it displays two images, it has some extra components which are:

- **Percentage/Place slider:** this slider is used to determine the *division line* or *blending percentage* for the fusion functions (see below).

- **Function:** the different fusion functions. Three fusion functions are implemented:

  - **Half/Half:** this function divides the image in two parts. The left side always is the reference image and the right side is the floating or resliced image. The place of division between these two parts can be changed with the slider below the image. This slider always has the same width as the image. In figure A.3 the fusion window can be seen, using the half/half function. Figure A.4 also shows an example of this function.
  - **Quarter/Quarter:** this function divides the image into 4 parts, with upper left and lower right being the reference image and the upper right and lower left being the resliced or floating image. The division lines go from upper left to lower right and are determined by the position of the slider. Figure A.5 shows and example of this function.
  - **Blend:** this function will blend the two images, with a certain blending percentage. The slider below the image now serves as percentage slider, with far left being 100% reference and 0% the other image and the far right being 0% reference and 100% the other image. Figure A.6 shows an example of this function.

3.1.3 Matching Parameter windows

The parameter windows are also an important part of the program. These windows are used to display and change the parameters.

Each algorithm has its own parameter window, therefore each has different components, i.e. parameters, but some parameters can be the same.

[1] A 16-bit values has two bytes, these two bytes must be swapped depending on computer endian type.

Currently two matching algorithms are supported by AIM and as such it has two parameter windows. The two parameter windows are shown in figure A.7.

The parameter windows always have a "Set Default" button and a "Factory Default" button. These two buttons will save the current settings as default and set the settings to the factory default respectively.

As can be seen in figure A.7 the two parameter windows both have "initial transformation" and "threshold" settings. Both settings are directly visible in the image windows. The initial transformation is applied to the floating image and the thresholds to the corresponding image.

3.2 Non-Graphical User Interface

The second part of AIM consists of non-GUI components. These components provide the underlying or extra features, these include: input/output, internal storage of the images and the external execution of matching packages. These components are not visible to the user, but are only used by the GUI components.

3.2.1 Matching

Without the ability to execute matching algorithms this program would be useless, therefore care has been taken to make sure that this is done correctly, from getting all the right parameters to actually starting/stopping the algorithm.

The matching algorithms/packages are independent programs, so the algorithm source code is not included into AIM, they are executed as external programs. This method has several advantages, including:

- AIM is independent of the packages.
- The packages are still usable outside the AIM program.
- The packages need to be executable only, no source code is needed.

Disadvantages are:

- The packages must support a command line interface, this is the way they are called by AIM.
- The packages should not require user input. For example the package must not stop processing and wait for a user action. It should just start and run completely (to its end), without user actions.
- The package must support a way to reslice or return the transformation parameters in a file, so that these can be used by AIM to reslice the floating image.
- AIM must support the image formats needed by the packages.
- AIM must know all package parameters. Normally AIM overrides all parameters.

When the user starts a match the following actions are performed by AIM:

1. The reference and floating image are written to Analyze format as two temporary files. AIM will not overwrite existing files, because it will try to find random temporary file names, which are unused

2. The matching package parameters are gathered and the correct command line is created from those parameters.

3. The matching program is started as an external program. During the time this algorithm is busy, AIM will wait and only handle messages once a second. If AIM would handle messages continuously, then it would use to much processing power.

4. If the match is ready, AIM will check if the program has created output (for example a resliced file or .air file).

5. If the program has created a resliced file, then this file is loaded into the resliced image, else the floating image is resliced by AIM (which uses a routine from AIR for reslicing) and that resliced file is loaded.

6. All temporary files are deleted. The resliced file is of course not deleted.

During matching the external matching program can be stopped.

The resliced file has the same name as the floating image, but with a 'r' prefix. If this 'r' prefixed file already exists, then another 'r' is prefixed and this is repeated until a file name is created which does not yet exist.

3.2.2 Input/Output

For the input and output of AIM, existing code was used. This code comes from the program MedCon, made by Erik Nolf. MedCon can be downloaded from `http://petaxp.rug.ac.be/ nolf` The code of MedCon has been (legally) used for the purpose of reading and writing of medical images. The two main reasons why existing code was used are:

1. Implementing the input and output of several (medical) image formats takes a lot of time, but this time was not available. Therefore it was decided this was outside the scope of the implementation of AIM.

2. MedCon can handle different types of image formats, with most different subtypes (i.e. pixel types such as 8-bit/16-bit/etc.) and little and big endian.

The choice was made to limit the supported file formats to the following:

- **Analyze:** this is a format originally developed by Mayo Clinic and used in the package "Analyze". It has a separate header and image file. The storage of the image data is fairly straightforward. The header provides several fields to store image type and size, voxel dimensions and more. This format can be read and written. Currently this format is the only one for which write support is available.

- **Ecat:** an image format used for PET-scanners. Implemented read-only.

- **DICOM:** the future standard medical image format. Implemented read-only.

3.3 Implemented requirements

3.3.1 Functional Requirements

In section 2.3.1 several functional requirements were presented. In this section the same list is presented, but now with comments whether or not a certain requirement was implemented and a possible reason why. Lack of implementation of requirements was mostly due to time constraints, but some just weren't possible to implement.

1. Image Input/Output:
   (a) DICOM and ECAT were only implemented read-only. Analyze was implemented both read and write. Actually the only write format for complete images implemented is Analyze.
   (b) Implemented, although not all algorithms support 2D images.
   (c) Not implemented.
   (d) Not implemented.
   (e) Implemented. Right-clicking on an image presents a pop-up menu, where an option to save the current slice to bmp-format is available.
2. Parameters:
   (a) Implemented. Each matching algorithm can read/write its own parameters. An algorithm only reads parameters relevant to itself from the file. If an algorithm reads a parameter file from another algorithm, then non-existent parameters are set to factory default values.
   (b) Implemented. If a default file for an algorithm exists then it will be loaded at the program startup.
   (c) Implemented. The current settings can be saved as default, by pressing the "Set Default" button. Confirmation is asked before actually writing the default values.
   (d) Not implemented. Currently all algorithm parameters are shown and are split into two parts, using tab sheets. Currently there are two tab sheets, being "General" and "Advanced" settings. These "Advanced" settings are those that can be hidden from a user, but currently they are always displayed.
   (e) Implemented. The values which are used are those specified by the authors of the algorithms themselves.
3. Other functions:
   (a) Implemented. 3D images are presented one slice at a time. The connection of two image windows has not been implemented, for this purpose the fusion window can be used if the image and voxel dimensions of two images are the same.
   (b) Implemented. The result of initial transformation can also be seen in the floating image window. It is also possible to reslice the floating image with that transformation applied.
   (c) Implemented.
   (d) Implemented. If the algorithm doesn't produce any output then an error is generated.

(e) Implemented. The user has a choice to turn this on or off, but it is not useful to turn this feature off, since matching results can be lost. For example, the resulting air-file (made by AIR) is removed after matching.

(f) If the "reslice after matching" option is turned on then the floating image is resliced and transformed and this image is loaded into the "resliced image". With the fusion window, this resliced image can be compared with the reference image. The actual resulting parameters are not shown. With AIR only a transformation matrix is generated, which is difficult to interpret and get the 6 transformation parameters from. The result transformation of Alignms (= Multi-resolution MI) is written to file, but reading this file has not been implemented.

(g) Not implemented. See previous item.

(h) Not implemented.

3.3.2 GUI/Program Requirements

1. User interface:

   (a) This has been tried to achieve by aligning and order the different components in a logical way. The components are ordered in the same way text is read (by western people), from left to right and top to bottom. Features that are not available at a certain time, because certain conditions are not met, are disabled until they can be used. New windows are placed and positioned automatically, for example the reference image is placed directly below the reference image group box and the resliced image below the resliced image group box.

   (b) The program uses three types of windows, being the main window, image/fusion window and the parameter windows. These windows are placed automatically, but can be moved if wanted.

   (c) The number of GUI components has been reduced by moving some not often used features to the main menu, for example saving of images is only available in the menu.

   (d) This has been implemented by consistent programming.

   (e) This is handled by the GUI toolkit. Almost all everything can be done by both keyboard and mouse.

   (f) Feedback is given by displaying messages on the main window status bar.

   (g) During file reading and writing a progress bar is shown, which shows the percentage read or written. During matching it is almost impossible to know how much time a algorithm needs, for this reason the verbose mode is by default turned on for each algorithm. This verbose mode is often implemented by algorithms and shows what the algorithm is doing. During matching the "Stop" button in the AIM GUI is enabled and this indicates that an external program is busy.

   (h) Not implemented.

   (i) Not implemented.

2. Program:

   (a) This is difficult to achieve, but has been attempted in the implemented.

(b) Unfortunately this has not been achieved completely. In order to display images, they need to be loaded into memory and for 3D images, the amount needed can be a lot. This memory is occupied the whole time the program is active.

(c) The only place where the program can influence this is when an algorithm is processing. During this time AIM will be active only once a second to handle messages (button pressing, etc). If AIM is set to handle message continuously then it will take too much processing power. The 1 second delay seems acceptable.

(d) Most parts are independent of the algorithms. Of course some parts are not. The main matching classes are dependent on the algorithms, but these are separated from the main GUI parts.

(e) Adding new algorithms is possible, but not very easy. It requires the implementation of several new classes and also some modifications to an existing class are needed. See §B.3 on how to add algorithms.

(f) AIM tries hard not to overwrite existing files. Temporary files which do not exist are created to be used in different parts. After reslicing, the resulting file name is that of the floating image, with an 'r' in front of it. If that file also exists, then an 'r' in put in front of that name, and that is repeated until a file name is found which does not exist yet.

3. Matching Parameters:

(a) This is implemented for each algorithm.

(b) Currently only the parameters are split into two groups, being "General" and "Advanced" parameters.

(c) Currently all parameters are shown.

Appendix A

User Manual

A.1 Introduction

Automated Image Matching (AIM) is a program that provides a graphical user interface supporting different image matching algorithms or packages.

An image matching algorithm will align two images in such a way that corresponding structures are in the same position in both images. This is done by transforming, by translation and rotation, the floating image to match the reference image.
Two images are needed for matching:

- **Reference Image:** this image will serve as the reference image, so it will not be transformed.
- **Floating Image:** this image will be transformed to match the reference image and it will also be resliced to match the image and voxel dimensions of the reference image.

Reslicing of the floating image is performed after matching and involves applying the transformation that was found by a match, but reslicing can also be performed without matching or without applying a transformation.

A transformation consists of 6 parameters, corresponding to 3 rotations and 3 translations. The rotations are performed around the centre of the image volume, and are called $pitch$, $roll$ and yaw. The translations are done in the x, y and z directions. Pitch is a rotation in the YZ view direction, roll is a rotation in the XZ view direction and yaw is a rotation in the XY view direction.

An image scanning technique is called a modality. Examples of modalities are Computed Tomography (CT), Magnetic Resonance Imaging (MRI) and Positron Emission Tomography (PET). Matching images from the same modality to each other is called "intramodality matching", and matching images from two different modalities is called "intermodality matching".

Matching is usually done intrasubject, meaning that both images are from the same subject (i.e. patient). Intersubject matching, which uses images from two different subjects, is not often performed and also not supported by most matching algorithms.

A.2 AIM

AIM consists of three main parts:

1. **Main window:** from this window the complete program can be controlled. Some features can be accessed via buttons on the main window. All features can be accessed via the main menu.

2. **Image window:** this is a window that displays a 3-D image. This image can be shown from three view directions and different slices can be shown.

3. **Parameter window:** each algorithm has its own parameter window, which can be used to set all parameters for that algorithm. Settings can be saved to and loaded from file.

A.2.1 Mainwindow

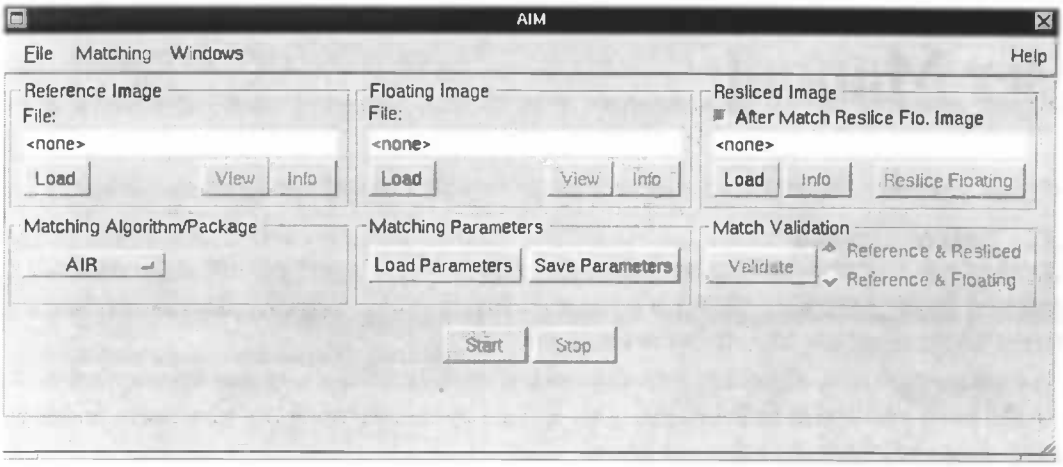

Figure A.1: Main window

The main window, shown in figure A.1, consists of the following parts:

- **Reference Image:** this image can be loaded, viewed or hidden and information about the image can be shown. The loaded image file name is shown. If no file has been loaded, then "<none>" is shown as filename. Via the main menu the image can also be saved to a file.

- **Floating image:** this is the image that will be transformed to match to reference image. The features for this image are the same as those available for the **Reference Image**, but then applied to the floating image.

- **Resliced Image:** this image can be one of three types:

  1. It can be a previously resliced (matched or not) image, which is loaded to use it as a fusion image.
  2. After a match the resliced floating image is loaded into the "Resliced Image".
  3. The floating image can be resliced and also transformed with the current initial transformation parameters, by pressing the **Reslice Floating** button, after which the resliced image is loaded into the **Resliced Image**.

- **Matching Algorithm/Package:** an algorithm can be selected via the combo box. When an algorithm is chosen, then the parameter window belonging to that algorithm is shown.

- **Matching parameters:** all parameters of the current algorithm can be saved to and loaded from file. Each algorithm has its own parameters, therefore it will only save and load those

parameters. Only parameters particular to an algorithm are loaded from file, parameters that are not present in the parameter file are set to (factory) default values. This can happen when a parameter file from another algorithm is read; only parameters that both have in common are read and all others are set to default. For this reason, parameter files should not be shared among algorithms, although it cannot do any harm.

- **Match validation:** after a match it is useful to check whether or not the match was correct. For this the fusion window was created. The fusion window displays two images (see §A.2.2). One of the two images always is the reference image, the other image can be the floating or resliced image. The option to use the floating image can be used to preview how the two images relate to each other before matching. The option to use the resliced image can be used to view how accurate a match was. There is however a limitation to the validation method, which is that both images must have exactly the same image and voxel dimensions.

- **Start and Stop:** with the start button a match can be started and with the stop button the match can be stopped. These buttons only become active when they can be used, so the start button is only active when the floating and reference image are loaded. During matching the start button becomes inactive and the stop button becomes active. The stop button will stop the match, but it should be noted that any intermediate matching results might be lost, depending on the implementation.

Main menu

Some functions in AIM can be executed with buttons, but these and other functions can also be executed from the main menu.

- **File:**
  - Load:
    * Reference Image: this loads the reference image. A file dialog will pop up, in which the image can be selected. In the file dialog the file name can be selected from the list or it can be typed into the edit box. An extension filter can also be applied, this will filter out any files with another extension then the selected extension. The correct file endian can also be chosen. Normally the file endian can be left to its default value (the host endian), but when a file does not load correctly a different setting should be tried.
    * Floating Image: same as Reference Image, but then for the floating image.
    * Resliced Image: same as Reference Image, but then for the resliced image.
  - Save:
    * Reference Image: this will save the reference image to Analyze format. A file name can be chosen for the file, this file must not already exist. AIM will not overwrite an existing file. The file will always be written to the host (computer) endian.
    * Floating Image: same as Reference Image, but then for the floating image.
    * Resliced Image: same as Reference Image, but then for the resliced image.
  - Quit: by selecting this the program will stop.
- **Matching:**

- Algorithm: this has a submenu from which an algorithm can be chosen. The items in this submenu depend on the supported algorithms. The functionality is the same as in the algorithm combo box.
- Start Match: same as the Start button.
- Stop Match: same as the Stop button.

- **Windows:**
  - View/Hide Reference Image: view or hide, depending on current visibility, the reference image.
  - View/Hide Floating Image: view or hide, depending on current visibility, the floating image.
  - View/Hide Resliced Image: view or hide, depending on current visibility, the resliced image.
  - View/Hide Fusion Image: view or hide, depending on current visibility, the fusion image.
  - (Re)Position Windows: position or reposition all image windows. This will put all image windows in their default position.

- **Help:**
  - About AIM: shows an "about" box with some information about AIM.
  - About Qt: shows an "about" box with some information about the toolkit Qt.

A.2.2 Image window

The image window displays the images, one slice at the time. The shown slice and view direction can be selected. The image can also be zoomed in. For each image (reference, floating and resliced) there is an image window. The image windows are positioned automatically, below the corresponding image group box (the box that is labelled with the image type name and contains the file name). With the **"(Re)Position Windows"** option from the main menu all image windows can be repositioned to their default positions.

The different components in the image window are:

- **Slice:** with this slider the shown slice can be chosen. The maximum number of slices depends on the view direction that is selected and the number of slices of that view direction.

- **View:** for a three dimensional image there are three view directions, being: XY, XZ and YZ. When the XY view direction is used then the number of slices is the size of the z-dimension. When the XZ view direction is shown then the number of slices is the size of the y-dimension. When the YZ view direction is shown then the number of slices is the size of the x-dimension.

- **Zoom:** a zoom factor between 0.25 and 3.0 can be chosen to decrease or increase the size of the image shown. The image is not interpolated, but only a fast image resize is applied. The actual image is not resized, only the displayed image slice is resized.

Since the z-dimension (depth) is usually very small compared to the x (width) or y (height) dimensions, the voxel sizes are taken into account when displaying the image. For example, when an image has 20 slices of 2 mm thickness and the image width is 256 voxels with distance 1 mm, then the

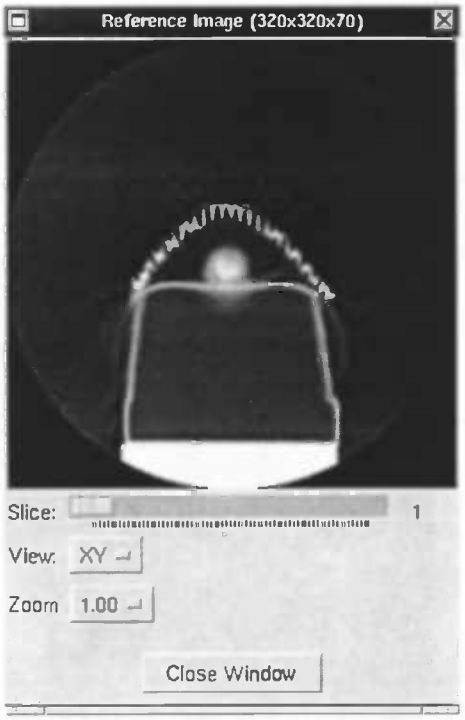

Figure A.2: Image window

displayed image height (view direction XZ) is 40 pixels. This way the displayed image reflects the real world dimensions.

There also are some extra options, which can be reached by right-clicking on the image with the mouse. The options are:

- **Save Slice to File:** this will save the currently displayed image to a bmp-format file called "aim_savedslice_x.bmp" with x a number. The number is increased if a file already exists. The used file name is not changeable.
- **Correct Image:** this will apply a function to the image, which will correct some image parameters, including minimum/maximum values and the voxel endian.

Floating Image Features

When the image window displays the floating image, some extra features are available. These features are rotation and translation. They are used with the mouse and the control key.

- Pressing the **left mouse button** in combination with the **control key**, will allow the **rotation** of the image. Until the control key and mouse button are released all mouse movements to the left or right result in a rotation of the image.
- Pressing the **right mouse button** in combination with the **control key**, will allow the **translation** of the image. Until the control key and the mouse button are released all mouse movements are translated result in a translation of the image.

Fusion window

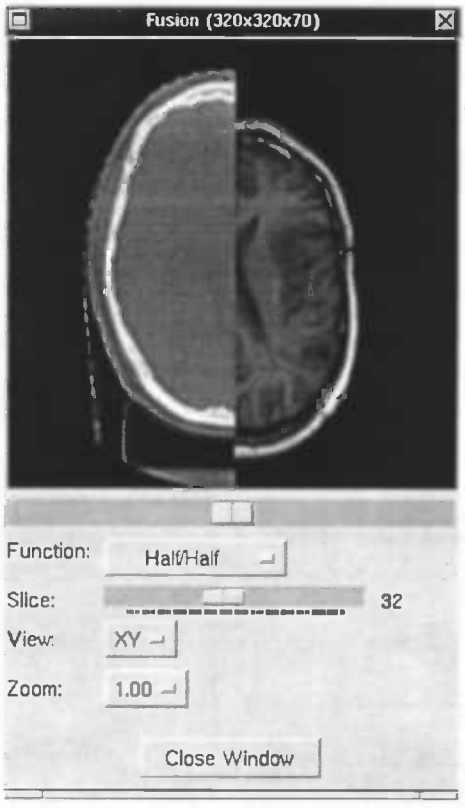

Figure A.3: Fusion window

This window is basically an image window, but with the addition of the ability to display two images, which can be used to validate a match. The left image is always the reference image and the right image is the floating or resliced image.

The two images must have exactly the same image and voxel dimensions in order to be shown. When one of the two images changes (for example by loading another reference image), the fusion window will be disabled when the image and voxel dimensions are not the same, an error message is also shown.

In addition to the components of a normal image window, the fusion window has some extra components, which are:

- **Position/Percentage Slider:** this slider is used to determine the division position or blending percentage, depending on the chosen fusion function. (See below).
- **Function:** used to choose the function to be used for fusion. There are three functions implemented:
    1. Half/Half: this will divide the image in two parts, the left image corresponding to the reference image and the right to the floating or resliced image. With the position/percentage slider the division line can be changed. An example is shown in figure A.4.

2. Quarter/Quarter: this will divide the image in four parts, the upper left and lower right corresponding to the reference image and the lower right and upper left to the other image. The centre of the four-way split can go from upper left to lower right by moving the position/percentage slider. An example is shown in figure A.5.
3. Blend: with this function the two images are blended into each other with a certain percentage. The position/percentage slider now acts as percentage indicator. Completely left is 100% of the reference image and 0% of the the other image, completely to the right is 0% of the reference image and 100% of the other image. An example is shown in figure A.6.

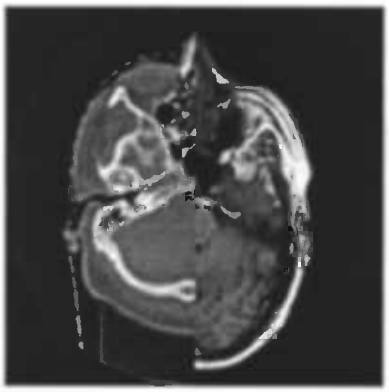

Figure A.4: The "Half/Half" fusion function

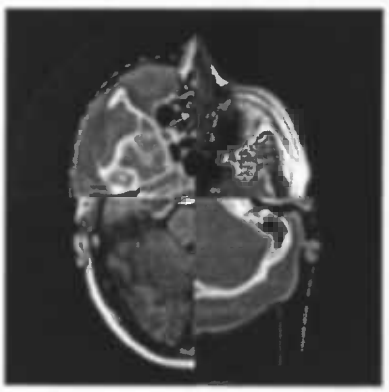

Figure A.5: The "Quarter/Quarter" fusion function

A.2.3 Parameter windows

An image matching algorithm usually has lots of parameters which can be used. The parameter windows facilitate display and modification of these parameters.

Each algorithm has its own parameters, but as can be seen in figure A.7 there are also some parameters that algorithms have in common.

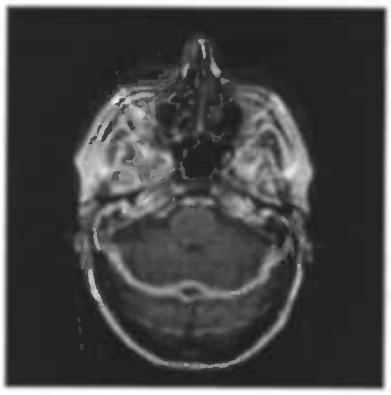

Figure A.6: The "Blend" fusion function

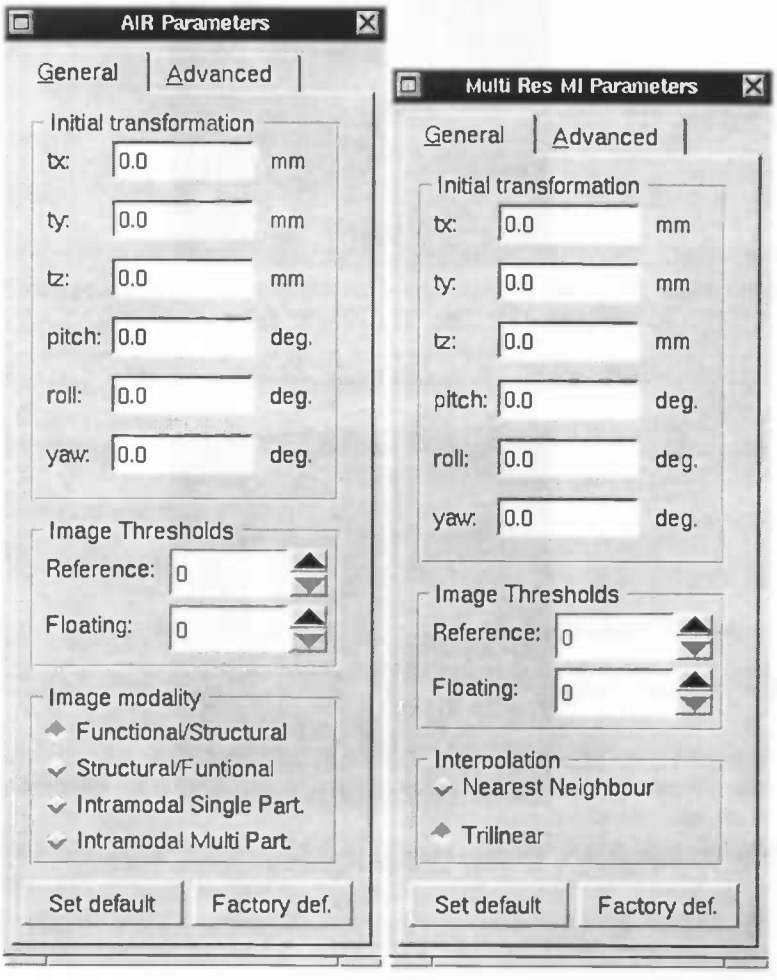

Figure A.7: Left: AIR parameter window. Right: MI parameter window

The parameters are split into two groups, being the general and advanced settings. The general settings include initial transformation parameters or thresholds. The advanced settings are parameters

that require knowledge about the algorithms. Currently all parameters are shown, however some parameters that do no require change will probably be hidden in a later version.

The two currently supported algorithms both have the following two parameters:

- **Initial transformation:** this is the transformation the matching algorithm will use at the start. These parameters can be determined by directly rotating or translating the floating image before matching and visually inspecting the correctness of the initial transformation. It is useful to check this initial transformation, because moving the match a bit in the correct direction can improve the final match. The three translations (tx, ty and tz) use millimetres (mm) as unit, while the three rotations ($pitch$, $roll$ and yaw) use degrees as unit.

- **Image Thresholds:** with these parameters the thresholds applied to the reference and/or floating image can be selected. Thresholds are usually implemented by setting voxels with values below the thresholds to 0 (= black). Some care should be taken when choosing these thresholds, because low values in MR images can also indicate bone structures. The effect of both thresholds can be seen in the displayed floating and reference images.

Matching parameters can be loaded from and saved to file, via the **"load parameters"** and **"save parameters"** buttons in the main window or via the main menu. The settings apply to the currently selected algorithm. Via the **"Set Default"** button the current settings are saved to file and used as default values. Via the **"Factory Def."** button the settings can be reset to the factory (built-in) default values.

Automated Image Registration

Automated Image Registration (AIR) is a matching package developed by Roger P. Woods. This algorithm was initially made for PET-PET [5] matching and later extended to allow PET-MRI [6].

The parameters, besides initial transformation and thresholds, for AIR are:

- **Image Modality:** with this parameter the correct modalities for the reference and floating images can be chosen and this will set some parameters to default values for these modalities. These parameters are the reference and floating partitions. The four choices are:

  1. Functional/Structural: the reference image is functional and the floating image is structural. A functional image is PET or fMRI and a structural image is CT or MR.
  2. Structural/Functional: the reference image is structural and the floating image is functional.
  3. Intramod. Single Part.: intramodality single partition. This can be used for intramodality matching with a single partition. Preferably used for PET-PET matching.
  4. Intramod. Multi Part.: intramodality multi partition. This can be used for intramodality matching with multiple partitions. Preferably used for MR-MR or CT-CT matching.

  For more information about partitions read the AIR package documentation and [6].

- Sampling: these settings determine the sampling rates used by AIR. The settings are: initial and final sampling frequency and the decrement ratio.

- Iterations: these settings determine the number of iterations used by AIR. There are three parameters: Repeated, Halt and Alternate.

- Conv. Thr.: convergence threshold.
- Cost Function: there are three cost functions:
    1. Std. Dev.: standard deviation of ratio image.
    2. Least Sqr.: least squares.
    3. Least Sqr. ir: least squares with intensity rescaling.
- Partitions: these are automatically set by the "Image Modality", but they can be overridden, by setting these values directly. After overriding these settings, the image modality should not be changed, because the partitions will then be turned to default values.

For more information about the exact meaning of all parameters, the AIR manual/documentation can be consulted.

Multi-Resolution MI

This algorithm uses the Multi-Resolution Mutual Information method developed by C. Studholme [3]. The supported implementation is one made by Alle Meije Wink [1] and the program is called *Alignms*.
The parameters for this algorithm are:

- Interpolation: the interpolation method used for reslicing. The two choices are: **Nearest Neighbour** and **Trilinear**. The first one is faster, but is less precise. The latter method is slower, but has a higher precision.
- Cube Size: the size of the cubic voxels used during the match.
- Max # of bins: the maximum number of bins.
- Res. P. Depth: the depth of the resolution pyramid.
- Subv. Opt.: the number of subvoxel optimisations.
- Bin Reduction Factors: these are 5 parameters for the bin reduction factors.

For the exact meaning of all parameters, please read the documentation supplied with *Alignms*.

A.3 Input/Output

AIM supports several image formats for input/output. There are three input formats and one output format. Formats are:

- Input:
    1. Analyze: a format developed by Mayo clinic and used in the Analyze package. This format uses two files, one for the header (extension .hdr) and one for the image data (extension .img). The header contains information of the images, like voxel and image dimensions. The data file contains only raw image data. To open an Analyze image, the header file should be selected.

2. Ecat64: a format used by PET-scanners.
3. DICOM: the standard medical image format.

- Output: Analyze only. AIM will only write files to the Analyze format.

A limitation to the current implementation, is that multi-file images are not read. DICOM almost always uses one file per slice, but AIM can only read one file. Other formats put multiple slices into one file and therefore AIM can read all slices of those images.

A.4 Command-line

AIM doesn't support command line only usage. This means that it must always use the GUI. There are however some command line parameters, which are part of the underlying toolkit. These command line parameters are:

- `-display display`, sets the X display (default is $DISPLAY).
- `-geometry geometry`, sets the client geometry of the main widget.
- `-fn` (or `-font`) `font`, defines the application font.
- `-bg` (or `-background`) `colour`, sets the default background colour and an application palette (light and dark shades are calculated).
- `-fg` (or `-foreground`) `colour`, sets the default foreground colour.
- `-name name`, sets the application name.
- `-title title`, sets the application title (caption).
- `-style= style`, sets the application GUI style. Possible values are motif and windows
- `-visual TrueColor`, forces the application to use a TrueColor visual on an 8-bit display.
- `-cmap`, causes the application to install a private colour map on an 8-bit display.

A.5 Normal usage

The following list shows the normal procedure to use AIM.

1. Load a reference image.
2. Load a floating image.
3. Select an algorithm or package.
4. Load parameters from file, if they were previously saved.
5. Set parameters to the desired values.
6. Save parameters to file, if they are needed for later use.

7. If needed set the initial transformation in the parameter window or directly in the floating image.
8. Start matching.
9. When matching is ready, validate the match by viewing the reference and resliced image in the fusion window.
10. If the match is not satisfactory, then restart from item 3.

A.6 Miscellaneous

A.6.1 Tips

- Clicking to the left or right of the pointer of a slider, will make it decrease or increase by one step. This is easier than keeping the mouse button pressed and decreasing or increasing it one step by moving the mouse left or right.
- When the fusion window is displayed, it is easier to first hide (close) this window before loading another image in the reference image or the other image used in the fusion window. If for example another reference image is loaded and that new one has different image dimensions, it cannot be used by the fusion window and this will give and error message. If the fusion window is not shown, then this error message is also not shown.
- With a program called Osiris (for Windows), Analyze files can be read as raw data and the image can then be written to DICOM (one slice per file) format. In this way, AIM has almost has complete multi-file DICOM to DICOM support. Only multi-file DICOM has to be implemented.
- Double clicking on an edit box will result in the selection of the contents of that edit box. When the text is selected and text (or numbers) are typed in, then the selected text is replaced.

A.6.2 AIM initialisation file

AIM also has an inifile, which it uses to store some settings. Unfortunately there is no interface to this in AIM. The first time AIM is started and closed, the file "aim.ini" is written to the location of the AIM executable. After this file is written it can be modified with a text editor. The settings available:

- **FileEndian:** the current file endian, please do not change this setting.
- **ResetZoom:** this will reset the zoomfactor, when a new image is loaded in an image. 1 is on and 0 is off.
- **DefaultAlgorithm:** this selects the default algorithm. This is the most useful option. Set this to 0 for AIR, which is also the default value, and to 1 if *Alignms* (Multi-Res. MI) is the desired default.
- **TempDir:** the name of the temporary directory, in which all temporary files are written. This must be a directory, without a trailing "/".
- **bmpPath:** the location where the saved slices are stored. By default this is the AIM path, but it can be modified. Value is a directory without a trailing "/".

Appendix B

Technical Manual

B.1 Introduction

This appendix describes the technical aspects of the program Automated Image Matching (AIM). It is mainly intended for people who are interested in extending and/or improving the program.

AIM is written in the programming language C++ and uses several of the features that it offers, including: exception handling, classes and templates.

B.1.1 Conventions

To make the program readable several conventions were used, including variables names, class names and error handling.
The source code is divided in three parts:

- **Graphical User Interface (GUI)**: all classes that are a part of the GUI are in this group and therefore are located in the subdirectory `GUI/`. Examples: `CMainWindow`, `CImageWindow`, `CFusionWindow`, etc.
- **Non-GUI**: all classes that are not part of the GUI or are non-visible are part of this group. These classes can use Qt signal handling or can be subclasses of Qt classes, but are not visible. These classes cannot do input/output of images. They are located in the subdirectory `NONGUI/`. Examples of non-GUI classes: `CImage`, `CMatch`, etc.
- **Input/Output**: all classes that deal with the input and output of images are i/o classes. The actual i/o is performed by the "MedCon" part of AIM (see §B.1.2), but there are several i/o classes which are wrappers for the MedCon i/o. The i/o classes are located in a subdirectory called `IO/`. Examples of i/o classes: `CAnalyzeIO`, `CImageIO`, `CEcatIO`, etc.

The following variables/classes naming conventions are used:

- **Classes:** class names begin with a 'C' to distinguish them from variables and other (non-AIM) classes. The separate words of a class name begin with a capital letter. Examples: `CMainWindow` and `CImageWindowImage`
- **Functions:** function names begin with a non-capital letter and the other parts of the function name start with a capital. Functions that can set and get information for class variables, should start with "set" and "get" respectively. Examples: **getThreshold(), setThreshold(), calculateSomething()**.

- **Variables:** for variables the only convention that should be followed is that they should not look like classes or functions. The functionality of a variable should be clear from its name.

B.1.2 MedCon

An important part of AIM is the input and output of images. In the design of AIM, there were several requirements for the image i/o. Several image formats needed to be implemented. However implementing all these image formats from scratch was not really feasible and therefore the program MedCon was used.

MedCon is a program from Erik Nolf, which can convert several medical images types between each other. Supported image types are: ACR/Nema, Analyze, DICOM (read-only), Ecat64, GIF and Raw Binary/ASCII. MedCon is released under the GNU General Public License, which makes the program free of use and more importantly parts of the code can be freely used, with some limitations.

All MedCon code that was used is separated from the other code and is located in the subdirectory `medcon/`. The following parts of MedCon are used:

- **Input/Output:** the reading and writing of the supported formats is handled by MedCon. MedCon can read both big and little endian files. Files are always converted to the host endian. Not all image formats are used, the actual formats implemented for AIM are: Analyze, DICOM (read-only) and Ecat (read-only).

- **Internal Image data structures:** the image structure, all header information and the actual data, are stored in a structure (**FILEINFO**) that is part of MedCon. This structure is used in the `CImage` class, which is the main image class. MedCon provides several functions that get slices from the image data and these are used for the display of the images, but several extra functions were added to extend this functionality. Instead of only being able to display slices from the XY view direction, AIM can also display slices from the YZ and XZ view directions. Several fusion functions were implemented, these functions get slices from two datasets, instead of just one and they fuse the two images using a certain function.

B.1.3 Qt

The graphical user interface is implemented with a toolkit called Qt. This is a free (for Unix) graphical user interface toolkit, which is very well structured and therefore very useful. Qt can be downloaded from `http://www.troll.no`.

It would lead too far to give extensive information about how Qt works, but some aspects will be briefly explained.

Widgets

The base-class for all objects in Qt is `QObject`, all Qt classes are derived from this class. To make use of non-GUI Qt features, such as signals, a class must be a subclass of `QObject`. For example the the `CMatch` class is a subclass of `QObject`. Qt classnames all begin with a 'Q'.

Graphical user interface components are subclasses of class `QWidget`. A widget is a graphical component (button, combo box, etc), a window or just a non-visual component that holds other components (frames, placeholders, etc). If one makes a widget without a parent, then it is a window (the window manager is the parent in this case), but if it does have a parent then it is just a component which is placed inside a window.

Signals/Slots

Qt uses signals and slots to send and handle messages. A signal is a message that is sent and a slot is a signal handlers which receives and handles message (signals). Messages are send by GUI components such as buttons, combo boxes, sliders, windows, etc. These signals and slots need to be defined in the class description, by adding them into two new sections in the class definition called: **signals** and **public slots**. A slot should be an implemented function. A signal is only sent and is not an actual implemented function.

Signals and slots can be connected to each other with the **connect()** function. Also some special pre-processing has to be applied to a class that uses signals and slots. A special Qt program called "moc" needs to be applied to the class definition (usually in the header file).

A signals/slots example: The class `CImage` stores all information about images, but also acts as a messenger between the image windows and the parameter windows. If the floating image is rotated by the user then the parameters need to be updated in the parameter window. If the user enters a transformation parameter in the parameter window, then the image window needs to be updated. This requires the `CImage` class to send and receive message to and from two classes, being `CImageWindow` and `CMatchXXWindow` (with XX being a matching algorithm).

The class definition of `CImage` has the following:

```
public slots:
  //  called by image windows
  void setRot(int plane, double r);
  void setTrans(int plane, double tx, double ty);
  // called by matching algorithms
  void setTransformation(double tx, double ty, double tz,
                         double pitch, double roll, double yaw);
signals:
  // transmitted to matching algorithms
  void newRot(double pitch, double roll, double yaw);
  void newTrans(double tx, double ty, double tz);
  // transmitted to ImageWindows
  void newTransformation();
```

The functions **setRot()** and **setTrans()** handle the signals sent from the image windows to the `CImage` class. When these functions receive a signal then they will send a signal to the parameter window. These signals are **newRot()** and **newTrans()**.

The function **setTransformation()** handles the signals sent from the parameter windows. When a signal is received, the signal handler will send the signal **newTransformation()** to the image windows.

In the class `CImageWindow` the signals and slots are connected in the following way:

```
QObject::connect(this, SIGNAL(newRot(int, double)),
                 theImage, SLOT(setRot(int, double)));

QObject::connect(this, SIGNAL(newTrans(int, double,double)),
                 theImage, SLOT(setTrans(int, double, double) ));

QObject::connect(theImage, SIGNAL(newTransformation()),
                 this, SLOT(newTransformation()));
```

The first and second calls of the **connect()** function connect the rotation and translation signals from the image window (`CImageWindow`) to the displayed image (`CImageWindowImage`). The third call of the **connect()** function connects signals from the image to the image window.

To make the whole signal/slot thing work, the header definitions also need to be parsed by the Qt program "moc". This can be done by adding some things to the program Makefile.

First you need to actually let the file go through "moc", this can be done by:

```
hCMainWindow.cpp: CMainWindow.h
        $(MOC) CMainWindow.h -o hCMainWindow.cpp
```

This will do the actual mocification of the header file. Moc creates a file called "hCMainWindow.cpp" and this file needs to be compiled and linked with the final program. How this is done, is dependent on the structure of the Makefile.

Adding a class in AIM, which has signals and slots, to the Makefile is simple:

- Add the new class header filename to the `MOCSRCGUI` or `MOCSRCNONGUI` items, depending on it being a GUI or non-GUI component.
- Add the mocification line to the bottom of the Makefile. The easiest way is to use an existing line and replace the filenames.

Example: We add the class `CMatchSomeWindow` to AIM. Then the `MOCSRCGUI` item in the Makefile becomes:

```
MOCSRCGUI  = CImageWindowImage.h CMatchAIRWindow.h CMatchMSWindow.h \
             CFusionImage CImageWindow.h CMainWindow.h \
             CMatchSomeWindow.h
```

At the bottom of the Makefile we add the following:

```
$(AIM_GUI)/hCMatchSomeWindow.cpp: $(AIM_GUI)/CMatchSomeWindow.h
        $(MOC) $(AIM_GUI)/CMatchSomeWindow.h \
               -o $(AIM_GUI)/hCMatchSomeWindow.cpp
```

In addition to the above, the source file of the new class also needs to be added to the `GUIFILES` section, since it is a GUI component. Classes that use no signals or slots do not have to be added to the moc parts, but only to the `GUIFILES`, `IOFILES` or `NONGUIFILES`.

B.1.4 Window Layout

Qt does provide classes that automatically position and size widgets on a window, but unfortunately these classes have some shortcomings. These layout classes are very awkward in use and sometimes do not behave as desired. The method that AIM uses is positioning the widgets in the source code directly. The widgets are positioned relative to each other and the font size is also taken into account. To move and position widgets, the following two functions are used:

- **move(int left, int top)**: this function moves the widget `left` pixels to the left and `top` pixels from the top. The position is relative to the parent.
- **setFixedSize(int width, int height)**: this will set the widget size to a fixed size of $width \times height$.

To determine the size of a widget, the function **sizeHint()** can be used. This function returns the size that should be used for the widget. For example: for a button the returned size is big enough to fit the displayed text on it.

Usually the largest widget is used as reference point for all other components. An example how widgets are placed:

```
tx->setFixedSize( 90, tx->sizeHint().height());
l4->setFixedSize( l4->sizeHint().width(), tx->size().height());

top = 20;

l1->move(INIT_LEFT, top);
l1->setFixedSize(l4->size());
tx->move(l4->width()+10, top);
mm1->move(tx->x()+tx->width()+10, top);

top += l1->height() + 12;

l2->move(INIT_LEFT, top);
l2->setFixedSize(l4->size());
ty->setFixedSize(tx->size());
ty->move(l4->width()+10, top);
mm2->move(mm1->x(), top);

top += l1->height() + 12;

l3->move(INIT_LEFT, top);
l3->setFixedSize(l4->size());
tz->setFixedSize(tx->size());
tz->move(l4->width()+10, top);

mm3->move(mm1->x(), top);
top += l1->height() + 12;
```

The above example is taken from the AIR parameter windows and sizes and positions the initial transformation parameters. The height of the edit boxes is set to a fixed width and height. The height depends on the value returned by **sizeHint()** and this will take into account the font size. Since widget `l4` (a text label) is the biggest one, all other text labels are given the same size as this biggest widget. The edit box `tx` is placed to the left of label `l1` and with an offset of 10 pixels. The unit label (`mm1`) is placed to the left of the edit box `tx`. The position of the next row is determined by increasing the `top` value. Then the second row is placed and all is repeated until all widgets are placed.

B.2 Classes

Since AIM is a C++ program it uses classes to separate the different parts of the program. There are three types of classes, being: GUI, Non-GUI, Input/Output. Besides these classes, several MedCon files are part of AIM.

B.2.1 GUI

`CMainWindow`

See figure A.1 for a screen shot of this window. This is the main window, which controls the whole program. The most important user actions are performed from this class. This class is a subclass of the Qt class `QMainWindow`, which provides several default items: a main menu, a status bar and a main widget. The main menu provides access to all features available in AIM. Some features are only available in the main menu, for example the image saving. The status bar is used to display short messages. Things like "Matching ready..." and "Reslicing ready...". When displaying a message, a time in milliseconds can be provided, after which the message will be removed. The main widget is where all other widgets are put into, it occupies the complete client area of the main window. In case of AIM, this main widget first has a `QFrame` as its primary widget. This frame draws a border around the main widget. The frame contains all other widgets, like the six group boxes, the start and stop buttons and the progress bar.

The advantage of using group boxes is that a name can be chosen, which is displayed in the upper-left corner of that group box. This is very easy for identifying the different parts of AIM. The main window's geometry is fixed, so it cannot be resized. It can be moved though, but it uses a fixed size, because the widgets it holds do not change size or position.

The `CMainWindow` class only has one instance, which is the **FMainWindow** object. This object can be used by other classes throughout the program, but care should be taken when using this object, since incorrect use can create unexpected results.

`CImageWindow`

See figure A.2 for a screen shot of this window. This window displays the actual image on screen. The displayed image is a widget of the `CImageWindowImage` class. Care has to be taken to position all widgets on the right place. A function, called **positionWidgets**, has been implemented to take care of this. The other widgets, like a `QLabel`, `QComboBox` and `QSlider` are fairly straightforward. The label, which displays the slice number can be also an edit box, but then a signal-handler must be implemented to handle the messages from this component.

The image window has a fixed size, although it is resized by AIM, depending on the displayed image size, and the zoom factor. When this window is displaying the floating image, then it will turn the rotation and translation option in the `CImageWindowImage` class on, otherwise it is off.

`CFusionImageWindow`

The fusion window is basically a `CImageWindow`, with the addition of some extra components and the possibility of displaying two images. After combining the two images using a certain fusion function, AIM can display the final image.

`CImageWindowImage`

This is a widget that displays an image of the class `QPixmap`. It gets a pixel map from its parent and displays it with a certain scaling factor. This scaling factor is given by its parent. This class has no knowledge about the actual image, everything must be calculated by its parent.

The rotation and translation of the images can be turned on and off. By default this is off and it is only turned on for the floating image. By using the mouse and the control key the translation and

rotation can be changed. When this is done, it will emit a message with its new values.

To avoid screen flickering, which occurs when the image is redrawn, this class uses double buffering. Double buffering involves drawing the image first in a non-visible backbuffer and then drawing this backbuffer to screen. A widget calls a function **paintEvent()** when it wants to (re)draw the widget. This function can be overloaded and this is done in `CImageWindowImage` to implement the transformation and double buffering. The object `thePixmap` is a `QPixmap` which holds the actual image, which is provided by its parent.

```
void CImageWindowImage::paintEvent(QPaintEvent *p)
{
  QPainter paint;   // an object which has several painting functions.
  QWMatrix matrix;  // the transformation matrix.

  // make sure that we actually have something to display.
  if (!thePixmap->isNull()) {
    backbuffer->resize(this->size());

    // first we draw the backbuffer.
    paint.begin(backbuffer);
      paint.fillRect(this->rect(), QBrush(black));  // clear backbuffer
      if (transenabled) {
        // lets translate and rotate the image.
        matrix.translate(ctx, cty);
        // first move the imagecentre to (0,0)
        matrix.translate(size().width()/2, size().height()/2);
        // rotate
        matrix.rotate(rotation);
        // move it back.
        matrix.translate(-size().width()/2, -size().height()/2);
      }
      // Scale the image. This is dependent on the pixelsize scaling
      // and the window zoomfactor.
      matrix.scale( pixel_dim_scale_x*WindowZoomFactor,    // width
                    pixel_dim_scale_y*WindowZoomFactor ); // height
      // Now we apply the transformation matrix to the image.
      paint.setWorldMatrix(matrix);
      // Finally we draw thePixmap into the backbuffer, this is now
      // automatically transformed by the above transformation.
      paint.drawPixmap(0, 0, *thePixmap);
    paint.end();

    // now we draw the actual visible image.
    paint.begin(this);
    paint.drawPixmap(0,0, *backbuffer);
    paint.end();
  }
```

As can be seen above this all is fairly straightforward and involves no difficult constructions. The

`CImageWindowImage` also provides an option to save the shown image to file. What it actually does is saving the backbuffer to file as bmp-format. It uses the backbuffer to save, since this is also the one that is transformed and scaled.

`CInfoDlg`

This is a fairly simple window class that will put all items from a `FILEINFO` struct into a non-editable `QMultiLineEdit`. This is one of the few places, a Qt layout class has been used to put all components on their correct places.

`CLineEdit`

This is a class that has been implement for convenience. It is a subclass of the `QLineEdit` class, and it provides a non-editable, but selectable line edit. It is used to display the image file names. The user might want to select these names, for copy and paste usage, but it should not be editable.

`CNumberEdit`

This class provides a line edit, that accepts floats or integers. It will not accept letters, only numbers. A selection can be made whether or not it should accept negative (the character '-') and/or floating numbers (the character '.').

`CImageFileDlg`

This is a wrapper class for the default file open and save dialogs. It will add the correct extensions and it adds the file endian checkbox.

`CMatchMSWindow`

This is the class that shows all parameters for the Alignms (Multi-Resolution MI) algorithm. It uses several types of widgets, including `CNumberEdit`, `QSpinBox` and `QRadioButton`.

The tab sheet construction in Qt is somewhat difficult, but Qt does provide a dialog window that uses tab sheets. This tab dialog (class `QTabDialog`), is actually meant to be used as a separate window for program configuration, therefore it has buttons like "Apply" and "OK". The "OK" button always exists and will close the tab dialog when pressed. We wanted to use this dialog, but it should not have some of the buttons which it provides and also should not be used as a window, but only as a widget.

For these reasons we used a nice feature that exists in Qt, which is that a widget's type (window or widget) depends on whether or not it has a parent. If one makes an object of class `QTabDialog`, without specifying a parent, then it is a window, but if we specify a parent, then it is a component or widget inside another window. The only thing that prevented the normal uses of this `QTabDialog`, was the "OK" button, which cannot be removed and always will close (i.e. hide) the widget when pressed. This problem was solved by making sure that the window in which the `QTabDialog` is placed was just big enough to fit the widget in such a way that the "OK" button cannot be seen. With this work-around, the `QTabDialog` could be used in the same way normal tab sheets should work. If a new algorithm is implemented, it is recommended to study this class carefully, especially the parts on how to put all widgets on the window.

`CMatchAIRWindow`

The same as the `CMatchMSWindow` class, but then for all parameters of the AIR algorithm.

B.2.2 Non-GUI

There are various non-GUI classes used by AIM. These classes cannot be seen by the user, but they provide the underlying structure of the program.

`CImage`

This is the internal storage class for images. All operations done on images must go through this class, for example: loading, saving, applying thresholds, etc. It stores images in a `FILEINFO` struct (from MedCon), which holds the actual image, with all its information (dimensions, etc). It is not recommended for any class to modify the image directly, but this can nevertheless be done, since this class can return a pointer to the `FILEINFO` struct. At the start of AIM, there are two objects made from this class, being the Floating Image and the Reference Image. Other objects are made as they are needed.

This class can receive and send two types of messages for transformations, being:

1. From parameter windows to image windows.
2. From image windows to parameter window.

Incoming signals from one object are immediately emitted to the other. What a receiving object does with the signal is implementation dependent, for example an algorithm may not support an initial transformation, in that case the received signals are discarded. Threshold messages are also sent from a parameter window to an image.

`CMatch`

This class is responsible for the correct calling and handling of the different algorithms. This class is dependent on the supported algorithms, since it must do several things that are dependent of the algorithms, including:

- Add the algorithm names to the main window menu and algorithm combo box.
- Create objects for the matching algorithm classes (`CMatchMS` and `CMatchAIR`).
- It must handle the call of the functions **match()** and **reslice()** of the matching classes, which will match and reslice respectively. These two functions must use the correct algorithm.

The two main functions, **main()** and **reslice()**, will match images and reslice images respectively. In order to add an algorithm, some code must be changed in this class and added to it.

`CMatchMS`

This class handles the matching and reslicing of images with the Alignms (Multi-Resolution MI) algorithm. It will start an external process which executes the matching program. In order to call this algorithm, it needs the correct parameters, which it gets from the parameter window belonging to this class. The `CMatchMS` class has an object from the `CMatchMSWindow` class, which is the

parameter window for this algorithm. The `CMatchMS` class is the only class which is allowed to make an instance of the `CMatchMSWindow` class.

There are two functions implemented which control the parameter window: **deleteWindow** and **createWindow**. The first one will delete the window, which is done when another algorithm is chosen. The current implementation does not actually delete the window, but only hide it. This way all settings are kept, for further use. If it were deleted, then the settings should be saved when deleted and read when shown/created. This is not efficient, therefore we simply hide it.

The **createWindow** function will create a new parameter window. If a window already exists and is hidden, then it is simply shown. If a window does not already exists then it is created.

`CMatchAIR`

This class functions in the same way as the `CMatchMS` class, but then for the AIR algorithm and `CMatchAIRWindow`.

`CConfig`

This class holds the configuration for the AIM program. There is one instance (global variable **theConfig**), which can be used throughout the program. There are several settings available for AIM, which are read from and written to the "aim.ini" initialisation file. This class also checks the environment settings relevant to AIM. If one of the mandatory variables does not exist, then AIM will stop.

This class has several items which can be retrieved or set, for example: the temporary directory, Alignms executable, etc. New settings can be added, by implementing a "set" and "get" function for the settings. Also a variable needs to be added to the `private` section of the class definition. This variable must be initialised in the class constructor.

`CInifile`

This class will read and write variables from and to file. The usage of this class is very easy, the following sequence should be followed:

1. Make a new instance of the `CInifile` class by doing the following:
   ```
   CInifile *ini = new CInifile(file);
   ```
   Where "file" is a string, preferably a `QString` object.

2. Open the ini file with the correct read or write status by using:
   ```
   ini->openFile(INI_Write);
   ```
   This will open the initialisation file for writing only. If "INI_Read" is used then it can be used for reading.

3. Read or write the needed variables. A write function accepts two parameters, the first being the variable name and the second being the variable. A read function accepts one parameter, being the variable name. The variable name should be the same as used when writing the variable. A read function return the values belonging to that variable name. The variable name must be a string, but the variable can be of the type used in the read/write function name. The following write functions are available: **writeInt**, **writeBool**, **writeString** and **writeFloat**. The following read functions are available: **readInt**, **readBool**, **readString** and **readFloat**. The read and

write function call must be enclosed in try/catch structures, because exceptions are given when a variable does not exist or some other error occurs.

4. Close the file with the **closeFile()** function.

5. Reclaim memory by deleting the object, with: `delete ini;`

Example:

```
void CConfig::loadConfig()
{
  QString file = getConfigFileName();
  CInifile *ini = new CInifile(file);

  try {
    ini->openFile(INI_Read);
  } catch (...) {
    delete ini;
    return;
  }

  try {
    FileEndian = ini->readInt("FileEndian");
  } catch (...) { FileEndian = HostEndian; }

  try {
    defaultalgorithm = ini->readInt("defaultalgorithm");
  } catch (...) { defaultalgorithm = ALG_AIR; };
}

CConfig::saveConfig()
{
  QString file = getConfigFileName();
  CInifile *ini = new CInifile(file);

  try {
    ini->openFile(INI_Write);
  } catch (...) {
    delete ini;
    return;
  }

  try {
    ini->writeInt("FileEndian", FileEndian);
  } catch (...) { ; }

  try {
    ini->writeString("TempDir", TempDir);
```

```
  } catch (...) { ; }

  ini->closeFile();
}
```

`CException`

When AIM encounters an error, it will throw an object of the `CException` class. Several items are available, for example: an error message, line number and file name. An object which catches the `CException` object, can get all the information given by the exception thrower. Example:
`throw CException("An error occurred");`

B.2.3 Input/Output

`CImageIO`

This class should be used for all image i/o, but no instance of this class should be made, since all functions are defined `static`. This class can be used for three things, being: image read, image write and getting the image format. These functions will use the correct image format classes, depending on the file format.

`CAnalyzeIO`

This class can load and write Analyze images. Actually this class is only a wrapper for the MedCon i/o functions. No instance should exist of this class, all functions are static and can therefore be called without an actual object.

`CDicomIO`

This class can be used to load DICOM images. Like the `CAnalyzeIO` class this class is also a wrapper and also no instance should exist.

`CEcatIO`

This class can be used to load ECAT images. Like the `CAnalyzeIO` class this class is also a wrapper and also no instance should exist.

MedCon

The subdirectory `medcon/` contains several files, which are part of the MedCon program. Some files were modified in order to make it better usable and to extent some of the capabilities. The graphical and non graphical user interface parts of MedCon were removed, only the i/o and internal structures are used. AIM makes extensive use of the `FILEINFO`, in which the image and its information are stored.

MedCon has a function, that can extract a 2D slice from an image, but it can only get a slice from the XY view direction. Some extra functions were implemented to make it possible to get 2D slices from the XZ and YZ view directions. The i/o code also has been modified to reduce the number of supported image formats to the three formats now supported (Analyze, DICOM, Ecat).

B.3 Adding Algorithms/Packages

Currently there are two image matching algorithms supported, but this can be extended. This section describes a step-by-step guide how to actually implement a new algorithm. We will go over each file that has to be modified and also the functions that have to changed.

Modifications to CConfig

AIM needs to know the location of the algorithm. This is is dependent on how the algorithm is implemented, for example the Multi Resolution MI algorithm only has one executable, therefore it is only needed to know the location of this executable, but AIR has several programs and therefore the location of the directory needs to be known. The way AIM retrieves these locations is by using environment variables.

The following in the `CConfig` class needs to be changed:

- **CConfig()**, the constructor. This is the place where the environment variable should be read and set. The following code must be added (XXX = the new algorithm name).

```
s = getenv("AIM_XXX_PATH");
if (s == NULL) {
   ErrorMsg("Error in environment",
   "Environment variable \"AIM_XXX_PATH\" not set !\n
    Please set it to XXX_Path = "";
   XXX_enabled  = false;
 } else {
   XXX_Path = s;
   XXX_enabled  = true;
}
```

  This is in case the environment variable points to a directory. In case of a single executable, one can use instead of "Path" "EXE". The variable **XXX_Path** is a `QString` and the variable **XXX_enabled** is a `bool` (boolean) and both need to be added to the `private` section in the class definition.

- Function **XXXEnabled()** needs to be implemented, this will return the value of "XXX_enabled".

```
bool
CConfig::XXXEnabled()
{
   return XXX_enabled;
}
```

- Function **getXXXPath()** (or **getXXXExe()**) needs to be implemented. This function return the algorithm path or executable.

```
QString
CConfig::getXXXPath()
{
```

```
    return XXX_Path;
}

void
CConfig::setXXXPath(QString aValue)
{
    XXX_Path = aValue;
}
```

Implementation of CMatchXXXWindow

Each algorithm has its own parameters, therefore a parameter window must be made for each algorithm. A parameter window must display parameters and allow modification of the them. It should also try to exclude parameters that are not relevant for the algorithm.

The parameters window can use different widgets for different types of parameters, the following list shows some of the possibilities:

- Integer value: for this a spin box (class `QSpinBox`) can be used. This will allow direct entry of the value, but also change by pressing an up or down button. The `CMatchMSWindow` class contains an example of how to make the spin box exclude certain values.
- Floating point number: a number edit box of the `CNumberEdit` class, can be used for this.
- Boolean: an on or off value can make use of a check box (class `QCheckBox`).
- Multiple options selection, meaning that there is a choice between a limited number of possibilities, for example the choice between three types of interpolation. These kind of parameters can use a combo box (class `QComboBox`).

For other types of parameters a correct GUI representation should be found.

If an algorithm supports initial transformations, then the code used in the two existing parameter windows can be used for the same purpose. If an algorithm does not support this, then there does not have to be a representation in the GUI, but the functions that deal with transformations should be implemented, it doesn't matter if they just return zeros or do nothing. The same holds for the thresholds.

The distinction between the two parameter types (general and advanced) should be used, but it is not obligatory. If more parameters are needed, then more tab sheets may be added. Reasonable names for these extra tab sheets must be used.

The files `CMatchTemplateWindow.cpp` and `CMatchTemplateWindow.h` can be used as a basis for a new algorithm. The word "Template" should be changed to the algorithm name. Inside the two files, there are comments on the implementation and also all functions that have to be implemented.

Implementation of CMatchXXX

This class handles all the matching and reslicing for a certain matching algorithm. The files `CMatchTemplate.cpp` and `CMatchTemplate.h` can be used as basis for a new algorithm. These files contain all the basic code needed for an algorithm. They contain also several comments on what to do.

CMatch

Some changes need to be made to the header file of the `CMatch` class, being:

- Add the algorithm to the algorithms list:

```
// Algorithms
#define ALG_AIR 0
#define ALG_MS  1
// nr of algs.
#define ALG_MAX 2
```

  A new "#define" must be added and the algorithms maximum needs to be increased. Example: `#define ALG_XXX 2` and the algorithm maximum becomes: `#define ALG_MAX 3`.

- Add a variable of the new algorithm matching class to the `private` section of the class definition. Example: `CMatchXXX *matchXXX;`.

The following functions need to be changed:

- **CMatch()**: the class constructor. An instance must be made of the new matching object, which is defined in the class definition. This can be done by using **new**. After an object has been created, then the pointer must be checked, this is done with the macro **CHECK_PTR**.

- **setWindow()**: this function sets the parameter window to the correct window. A new item in the switch statement must be added for the new algorithm and this must do the same as the two other algorithms do.

- **AlgorithmComboActived()**: this function is activated when a new algorithm is chosen. In the code it can easily be seen that a construction which checks for the enabled value of the algorithm should be implemented.

- **setAlgorithmCombo()**: this function adds the algorithm to the algorithm combo box. This is done with the **insertItem** function. Example: `combo->insertItem("XXX", 3);`

- **setMainMenu()**: this function adds the algorithm to the main menu. Example: `aMenu->insertItem("XXXX", 3);`

- **match()**: this algorithm start the matching of images. For each algorithm an item is added to the switch statement. For each algorithm there can be some difference in how the matching is performed, for example AIR uses an air-file, but Alignms does not. It is easiest to use the code from one of the two implemented algorithms and make minor changes to it.

- **reslice()**: this algorithm is used in two ways, the first way is reslicing after match and the second way is reslicing without matching. For a new algorithm only the first way should be implemented and this is done by adding it to the switch statement. When an algorithm reslices after a match, then the reslice only consists of getting the file name of that resliced file. If the algorithm does not reslice then the image must be resliced some other way. If the reslice function is used in the second way (reslicing without matching) then for a new algorithm only the initial transformation should be retrieved.

- **loadSettings()**: add a call to the **loadSettings()** function of the algorithm's matching class for the new algorithm.
- **saveSettings()**: as **loadSettings()**.
- **newTransformation():** simply add the call to **getTransformation()** for the new algorithm to the switch statement.

B.4 Future Versions/Recommendations

In this section some recommendations are made of features that can or should be implemented in the future.

1. Thoroughly test AIM. This is mainly to find possible errors or mistakes.
2. Implement Multi-file DICOM read. This will read DICOM images that have one file per slice, which is the normal way DICOM files are saved. If this is implemented then AIM can be used for DICOM input and Analyze output. The Analyze output can be converted to multi-file DICOM files with for example Osiris.
3. Implement the connection between the shown slice in the reference image window and floating image, resliced image or fusion image. When sliding through one image the other image will display the same slice.
4. Implement the direct input of slice number into number edit, instead of only having a label.
5. Implement a graphical user interface to the configuration settings. The current way, via an initialisation file, is not satisfactory.
6. Add more program configuration settings. If a GUI is made for the configuration settings, then the number of settings can be increased. For example: resliced file prefix, turning on or off of automatic window placement and more.
7. Implement level of window. This should be done in the MedCon file "m-alg.c".
8. Implement more sensible names for the saved slices. They should get a name which combines the slice, view direction and filename.
9. Hiding of parameters. Some algorithm parameters should be hidden. This can be done, but some research should be done on which parameters can be hidden.
10. Improve the execution of external programs.
11. Implement tool-tips. These the yellow pop up tips when holding a mouse over a certain button.
12. Implement a help-system.
13. Decrease memory consumption. Currently AIM holds both the floating and reference image in memory during matching, at which time the algorithm itself also has both images in memory. Before matching starts the two images should be removed from memory and reloaded after matching.

14. Increase robustness. Some error-handling and error catching has not been fully tested and implemented.

15. Improve the validation methods. For example the division point in the Quarter/Quarter function should have two dimensions of freedom, instead of only one. This can be done by tracking the mouse over the image and positioning the division point at the position of the mouse pointer.

16. Add new fusion functions.

Appendix C

INSTALL.TXT

```
Automated Image Matching (AIM)
Installation Manual

Author: Rolf Janssen

1. Introduction

This document describes the installation of the program
Automated Image Matching (AIM).


2. Prerequisites

This version can only be installed on a Unix computer
with X-Windows.

Before installing AIM the following packages need to
be present on the system:
- X11R6
  The X11 library. It's recommended that revision 6 is used,
  since lower versions are known to give problems with Qt.
  X11R6 is usually installed in /usr/X11R6, if not then
  X11LIB and X11INC need to be changed in the AIM Makefile.

- Qt 1.41 or Qt 1.42.
  This is the toolkit used by AIM.
  It can be downloaded from http://www.troll.no.

- GNU make (gmake).
  This must be GNU make and cannot be another make program.
  Sometimes this version is present on a computer system, but
  must be invoked with "gmake", instead of the normal "make".
  To check which version is present use "make -v" or
```

"gmake -v".

- A c++ compiler, which can use exception handling.
  Recommended are gcc 2.8.1 (or higher) or egcs 1.0.3
  (or higher). To check which compiler is present, use: "g++ -v".

3. Image matching Algorithms/Packages

To actually use a matching algorithm, this algorithm must be installed. The current version works with two algorithms, being AIR and Alignms. It's recommended that both algorithms are installed in order to use AIM correctly.

4. Installation

We will go over the installation step by step.

- The AIM package (aim.tgz) has the following structure:
  - aim/, the AIM program.
  - vtdicom/jpeg-6a/, jpeg library.
  - vtdicom/vtdicom/, vtdicom library.

- Before compiling AIM, the vtdicom (in subdirectory vtdicom) must be compiled.
  - Go to subdirectory "vtdicom/jpeg-6a" and invoke "make" (or "gmake"), this should compile the jpeg library.
  - Go to subdirectory "vtdicom/vtdicom" and invoke "make" (or "gmake"), this should compile the vtdicom library.

  The jpeg library is modified to compile with g++ and the normal jpeg library present on most system cannot be used. VTDicom is also a modified version, which can compile with g++. This VTDicom package is therefore not the same as can be found on the website of Erik Nolff (medcon).
- Before AIM can be compiled, it is first to be pointed to the exact location of Qt.
  There are two ways of doing this, ie:
  1) Open the AIM "Makefile" (aim/Makefile) in a texteditor and lookup the QTDIR item. Normally this is defined as "../qt". This entry can be changed to point to the full path of the Qt package. Care should be taken that the "QTLIB" and "QTINC" items are correct. If the complete package of Qt is installed on one place and not split over different directories, then simply pointing QTDIR to the uppermost Qt directory will suffice.

If this is not the case then QTLIB and QTINC should also be changed to point to the Qt library and Qt include files.

2) The easier way is to make a softlink to the Qt directory in the main installation directory (not in the AIM source directory). Example: qt is installed in "/usr/lib/qt" then make a link with "ln -s /usr/lib/qt qt" in the main installation directory.

- After all this "make" can be invoked from the AIM source directory. If all locations are correct then the program should compile without problems. The installation was tested on two computers (a HP-Unix and a PC with Linux) and worked correctly.

Installing the matching packages is left to the user.

5. Environment variables

In order to use AIM some environment variables should be set:

- AIM_PATH, the path where the AIM executable resides. The main program settings file is stored at this location.
- AIM_SETTINGS_PATH, the path where all AIM algorithm settings are stored.
- AIM_AIR_PATH, the path to the AIR package binaries. AIR must be present the current version. Without AIR the program will not start.
- AIM_ALIGNMS_EXE, the full path (and filename) of the Alignms binaries (executable). This settings is also obligatory.

Environment variables can be set with the "export" or "setenv" command.

The path must without a trailing "/", example:
AIM is installed in "/opt/AIR3.08/", then "AIM_AIR_PATH" will become "/opt/AIR3.08"
The variable can be set with the following command:
"export AIM_AIR_PATH=/opt/AIR3.08"
(NOTE: no spaces around the '=').

6. X-Windows Window Managers

Since AIM is a graphical program it also has to deal with the abundance of different Window Managers that exist for X-Windows. Some of them are not to usable with AIM, but this has not been extensivly tested. The best window manager for AIM is KWM, which is part of the K Desktop Environment (KDE). KDE is based on

Qt and therefore works best with Qt programs (such as AIM).

7. Summary

1) Get aim.tgz, and uncompress and untar it to some location.
2) Compile Jpeglib.
3) Compile VTDicom lib.
4) Set correct Qt location, by modifying the AIM Makefile or making a softlink to Qt.
5) Compile AIM.
6) Install matching algorithms/packages.
7) Set environment variables to the correct locations.
8) That's it, if everything is done correctly then AIM can be used. The executable is located in the AIM-sourcepath.

Rolf Janssen,
April 26, 1999

Bibliography

[1] Alle Meije Wink, *Automated medical image matching by maximisation of mutual information*, Master's thesis, 1999.

[2] Alle Meije Wink, *Woods' algoritme en het AIR pakket*, internal AZG/GNIP report, 1999.

[3] C. Studholme, D.L.G. Hill and D.J. Hawkes, *Automated 3-D registration of MR and CT images of the head*, Medical Image Analysis (1996) Volume 1, number 2, pp 163-175.

[4] F.M. Reza, *An Introduction to Information Theory*. Dover. New York. 1994

[5] Roger P. Woods, Simon R. Cherry and John C. Mazziotta, *Rapid Automated Algorithm for Aligning and Reslicing PET Images*, J Comput Assist Tomogr 16(4):620-633, 1992.

[6] Roger P. Woods, Simon R. Cherry and John C. Mazziotta, *MRI-PET Registration with Automated Algorithm*, J Comput Assist Tomogr 17(4):536-546, 1993.

[7] Roger P. Woods, Scott T. Grafton, Colin J. Holmes, Simon R. Cherry and John C. Mazziotta, *Automated Image Registration: I. General Methods and Intrasubject, Intramodality Validation*, J Comput Assist Tomogr 22(1):129-152, 1998.