OVERVIEW OF AIR 5.x

The Automated Image Registration (AIR) package is primarily designed to solve several different registration problems that arise in tomographic data sets:

- intrasubject, intramodality registration (e.g., PET to PET or MRI to MRI)
- intrasubject, intermodality registration (e.g., PET to MRI)
- intersubject registration (e.g. subject to subject or subject to atlas template)

Intrasubject registration of brain images uses a rigid-body model. Intermodality registration has been validated for some (MRI-PET), but not all modalities. Intersubject registration can be performed using any of a variety of linear or nonlinear models to register different subjects to one another or to an atlas template (for example, an averaged brain in "Talairach space"). These models may also be useful for intrasubject registration of organs that are more deformable than brain or for tracking intrasubject developmental changes over time.

In addition to 3D models, AIR includes homologous 2D deformation models that may be useful in selected circumstances (e.g., single slice fMRI data, pure 2D images, etc.).

Once the registration parameters are known, the AIR package allows the data to be resampled to generate a final image using any of the following interpolation models:

- nearest neighbor

- trilinear
- windowed sinc
- mixed linear/windowed sinc
- unwindowed sinc
- chirp-z
- mixed linear/chirp-z

WHAT IS INCLUDED

The AIR package includes automated routines for aligning and reslicing tomographic image data. In addition, a number of utility routines and error checking routines are provided. The routines are invoked as standard UNIX commands and (with one exception) neither expect nor allow user interaction once the command has been issued. In addition, a series of C subroutines which comprise the AIR library are provided. The AIR library can be used to easily incorporate automated image registration into site specific programs adapted to your particular needs.

AIR also includes graphical user interfaces for two of the most frequently used programs. These interfaces are written in Tcl/Tk.

The AIR package is distributed only as source code.

WHAT IS NOT INCLUDED

The AIR package does not provide any graphics or image display capabilities. No file format converters are provided.

REQUIREMENTS

- UNIX workstation, PC, or Macintosh running Mac OS X
- ANSI C compiler
- 16+ megabytes of RAM
- image file format converters
- Tcl/Tk (optional)

The default image file format is compatible with the ANALYZE format developed at the Mayo Clinic. If you currently use 8 bits/pixel or 16 bits/pixel ANALYZE file format on a computer equipped with an ANSI C compiler, have at least 16 Mbytes of RAM, and are familiar with the UNIX operating system, you should be able to install and use the AIR package without additional assistance, even if you know nothing about C programming. However, if any of these conditions is not met, it is likely that you will need the assistance of a C programmer who is familiar with your operating system and C compiler.

Tcl/Tk is required to use the graphical user interface (GUI), but all of the functionality of the AIR package can be accessed without the GUI.

CONFIGURING AND COMPILING AIR 5.x

UNIX (INCLUDING LINUX AND MAC OS X)

1. In the main AIR directory, type './configure'.
   This will execute a shell script that should automatically create three files. The first of these will be in the main AIR directory and is called 'Makefile'. The other two will be in the src/ subdirectory and are called 'Makefile' and 'config.h'.
   If ./configure fails, it should print out some insight into the problem that you may be able to fix. Alternatively, you might consider some of the workarounds listed below
   If you troubleshoot ./configure, you sometimes need to remove ./configure.cache to get ./configure to forget results of earlier runs
   Caution To Developers: the Makefiles that are generated do not contain the level of dependency information needed for efficient software development. For example, changes to a subroutine in the src subdirectory will not cause all main programs that use that subroutine to be rebuilt without any unnecessary rebuilds of other routines. Developers should consider building a shared library from the src directory subroutines and linking to this shared library as a strategy for shifting such dependencies to runtime.
2. In the src/ subdirectory, edit the 'config.h' file to meet any site specific preferences.
   Any text editor can be used (be sure to save any changes as plain text), configuration details are provided below.
3. Optionally, review and edit the two files called 'Makefile'
   Experts may elect to override default optimization levels, etc.
4. In the main AIR directory, type 'make all'.
   This should compile the code
5. If you want the compiled programs moved to the bin subdirectory, type 'make install' in the main AIR directory
   If you ultimately want the programs somewhere else entirely, it is much easier to copy them from /bin, so type 'make install' to move them to /bin first
6. Before rebuilding AIR (e.g., after changing something in src/config.h) you must type 'make clean' in the main AIR directory before typing 'make all' again
   'make clean' removes all of the compiled libraries and programs so that new, internally consistent versions can be built
7. To start over from the beginning at any point, type 'make distclean' in the main AIR directory and return to step 1
   'make distclean' removes all of the compiled libraries and programs as well as the configuration information and Makefiles

Workarounds:

1. If the ./configure script fails and you cannot get it to run correctly, you can simply create a file called src/config.h and copy any configuration lines that you feel are appropriate into it (an empty file called src/config.h is acceptable if you want to use all the default configurations in the AIR.h file)
   In this case, you will need to execute the MAKE.com file as described next rather than running 'make'

2. If the command 'make all' fails, (e.g., if you don't have the executable called 'make'), you can type 'sh MAKE.com' or 'sh MAKE.com bin' (the latter command will install the programs in the bin subdirectory) from the main AIR directory to compile the code.
3. It is also possible to configure AIR manually in the AIR.h file as in older versions of AIR, but this should not generally be necessary.

PC (DOS, WINDOWS AND WINDOWS NT)

Please note that different PC compilers have different requirements and only general recommendations are provided here.

1. In the src subdirectory, copy the file 'config_PC.h' to 'config.h'
2. Edit the 'config.h' file to meet any site specific preferences
   Any plain text editor can be used, configuration details are provided below.
3. In the main AIR directory, edit the files AIRmain.h and HEADERmain.h, removing the lines with UNIX style forward slashes and uncommenting the lines with the PC style backslashes
   After modification, AIRmain.h should read: #include "src\\AIR.h"
   After modification, HEADERmain.h should read: #include "src\\HEADER.h"
   There should not be any spaces or tabs before '#include' in either of these files
4. Using your PC C compiler, compile all of the files in the src subdirctory into a library
   Consult your C compiler documentation for details
5. Using your PC C compiler, compile each of the programs in the main AIR directory. For example, compile the file 'alignlinear.c' to generate the executable file 'alignlinear.exe'. You will need to link in the library compiled in the previous step.
   Consult your C compiler documentation for details
   Note that the files src/AIR.h, src/config.h and src/HEADER.h are required when compiling each of the programs in the main directory.

CONFIGURING THE SRC/CONFIG.H FILE

The following lines shown in green can be included or altered in config.h to provide site specific information: Omitting these lines causes their default values (defined in the AIR.h file) to be used.

#define AIR_CONFIG_AUTO_BYTESWAP 0

If non-zero, AIR will attempt to detect image files (.hdr and .img), .air file, .warp files and .vector files that need to be byte swapped because they were generated on a different machine. If your machine does not conform to the IEEE Standard for Binary Floating Point Arithmetic (ISO/IEEE Std 754-1985), you should set this value to zero because conformance with this standard is assumed when byte swapping. For the same reason, only data generated by machines that conform to this standard can be properly byte swapped for use by AIR.

#define AIR_CONFIG_OUTBITS 16

The only supported values are 8 and 16. If you choose 8, data will be represented internally and saved as 8 bits/pixel. If you choose 16, data will be represented internally and saved as 16 bits/pixel. (Either way, you will be able to read in 8 or 16 bit data, this variable only controls output data). If you are not sure what to do here, the issue of 8 and 16 bit data is discussed elsewhere at length.

#define AIR_CONFIG_REP16 3

This variable is irrelevant if you have OUTBITS set to 8. The acceptable values are 1, 2 and 3. The variable controls the format of 16 bit data saved to disk by the AIR package. If you are not sure what to do here, the issue of 16 bit data types is discussed elsewhere at length.

#define AIR_CONFIG_THRESHOLD1 7000

#define AIR_CONFIG_THRESHOLD2 7000

AIR_CONFIG_THRESHOLD1 and AIR_CONFIG_THRESHOLD2 control the default pixel value thresholds used by the registration programs. If you have set AIR_CONFIG_OUTBITS to 8, only the first set of thresholds is used. If you have set AIR_CONFIG_OUTBITS to 16, only the second set is used. The thresholds should be values that will typically separate brain (suprathreshold) from nonbrain (subthreshold). The values used in the AIR distribution are not particularly likely to be useful in your laboratory. Look at some data to make a reasonable guess, and bear in mind that these thresholds are easily overridden on a per use basis.

#define AIR_CONFIG_VERBOSITY 0

If non-zero, this allows some non-essential information to be printed to the screen when AIR programs are run.

#define AIR_CONFIG_PIX_SIZE_ERR .0001

The variable decides how fussy the AIR package is about deciding that pixel sizes are identical. If you plan to round off pixel sizes only to the nearest .1 mm (not recommended), you need to make this value .1. If you like to type everything out to 15 decimal places, you can make the AIR package equally compulsive.

The following lines may already appear in config.h as generated by ./configure or as copied from config_PC.h. Adding them (if not already present) or altering their pre-existing values could render AIR uncompilable on your machine, so make changes with caution and make back up copies first.

#define AIR_CONFIG_GROUPS 0

This controls the inclusion of a non-ANSI subroutine that checks for a user's membership in secondary groups when testing whether file reading or writing is expected to succeed. A non-zero value should only be used on platforms that support membership in multiple groups and that support the non-ANSI C subroutine getgroups(). Compilation of src/fprobr.c and src/fprobw.c will fail if getgroups() is not supported and this configuration variable is non-zero.

#define AIR_CONFIG_REQ_PERMS 0

This controls whether AIR uses two C functions that are not part of the ANSI C standard, but that provide useful functionality if available. The non-ANSI functions stat() and lstat() make it possible to access UNIX file permissions. By using these functions, AIR can inform a user of specific reasons that a program failed (e.g., that the user lack write permission in the directory where data is supposed to be saved). Some non-UNIX C compilers also implement stat() and/or lstat() (even though the underlying operating system may not support all of the informative functionality). If stat()

and/or lstat() are available, it is recommended that you use them. However, if they are not available, compilation will probably fail unless you set this value to 0. Possible configuration values are:

- 0. AIR makes not use of stat() or lstat().
- 1. AIR will use stat(), but not lstat().
- 2. stat() and lstat() will be used.

#define AIR_CONFIG_PIPES 0

This controls whether AIR will use a non-ANSI C function needed for safe on-the-fly decompression of image files. If non-zero, AIR will use the function popen() to pipe data from a decompression program into AIR. If your compiler does not support popen, you should set this value to zero. If this value is zero, you do not need to worry about AIR_CONFIG_DECOMPRESS, AIR_CONFIG_COMPRESSED_SUFFIX or AIR_CONFIG_DECOMPRESS_COMMAND because they have no effect in the absence of popen().

#define AIR_CONFIG_DECOMPRESS 0

This parameter is only relevant if AIR_CONFIG_PIPES is non-zero. If this value is set to zero, code to decompress images on-the-fly is disabled.
#if AIR_CONFIG_DECOMPRESS

#define AIR_CONFIG_COMPRESSED_SUFFIX ".gz"

This parameter is only relevant if AIR_CONFIG_PIPES and AIR_CONFIG_DECOMPRESS are both non-zero. This suffix identifies compressed header or image files. For example, if the value is ".gz", files ending with .hdr.gz will be recognized by AIR as compressed header files and files ending with .img.gz will be recognized as compressed image files. If you use the UNIX compress routine, you would need to change the suffix to ".Z".

#define AIR_CONFIG_DECOMPRESS_COMMAND "gunzip -c "

This parameter is only relevant if AIR_CONFIG_PIPES and AIR_CONFIG_DECOMPRESS are both non-zero. This must identify a command that is able to decompress files ending with AIR_CONFIG_COMPRESSED_SUFFIX when followed by a file name ending in this suffix. In addition, the command must output to a pipe. If the decompression program is not on a user's search path, AIR may fail to find the command. In this case, you can give a fully qualified path for the decompression program (e.g., "/usr/local/bin/gunzip -c "). Note that a trailing space after the command is required.

UPGRADING FROM AIR 2.0 OR AIR 3.0X

The format for .air files is identical for AIR 2.0 through AIR 5.x, so these files can be used interchangeably. The format for .warp files has been changed in AIR 5.x. The current version of AIR can read .warp files from earlier versions, but the reverse is not true.

All of the AIR 2.0 and AIR 3.0 commands are available in AIR 5.x, and efforts have been made to assure that commands syntax is unchanged. However, certain changes have proved too awkward to avoid:

- In the program alignlinear, the -w flag is no longer suppported.

- In the program align_warp, the -f flag must be followed by a .warp file name, not an ASCII text file and the -g flag is no longer supported.
- In the program combine_warp, the use of 'null' to replace one of the .air files is no longer supported and no longer needed due to greater flexibility of that program.
- In the program zoomer, the use of 'y' to grant permission to overwrite a pre-existing file is supported only so long as no new options are used, but this usage should be avoided in the future.

UPGRADING FROM AIR 1.0

AIR 5.x is fully compatible with any .air files derived using AIR 1.0. However, the reverse is not the case. The old AIR 1.0 programs will not be able to read the new .air files. If you start getting error messages complaining about being unable to read or load .air files, check your path and make sure that an old version of AIR is not being called without your realizing it.

The most difficult adjustment in starting to use AIR 5.x is the fact that a single program, alignlinear, now provides the automated alignment for linear intrasubject, intersubject, and intermodality problems. As a result, you must pay close attention to make sure that you always specify the model, thresholds, and partitions appropriately. You may find it easiest to use the Tcl/Tk graphical user interface to choose the various options.

GETTING STARTED WITH AIR

SYNOPSIS OF THE PROGRAMS

The AIR package includes two programs for automated registration of images. The first of these, alignlinear, includes 2D and 3D variants of all linear spatial transformation models, including rigid-body models, global rescaling models, affine models and perspective models. It support within-modality, across-modality and linear intersubject registration.It generates a file, referred to here as a ".air file", that contains linear transformation parameters for resampling one of the images to match the other. The second alignment program, align_warp, includes 2D and 3D variants of nonlinear polynomial spatial transformation models ranging from first (linear) to twelfth order. It generates a file, referred to here as a ".warp file", that contains nonlinear transformation parameters for resampling one of the images to match the other. The information in first order .warp files can also be represented by a .air file, and such files can be substituted for .air files in any AIR program that uses .air files. Similarly, any .air file that does not describe a perspective deformation can be represented by a .warp file, and such files can be substituted for .warp files in any AIR program that uses .warp files. Aside from these transparent substitutions, programs within the AIR package are generally designed to work with .air files or .warp files, but not both. Most programs that deal with .warp files include _warp at the ends of their names. AIR 5.0AIR provides limited support for a third type of file for describing registration parameters, referred to here as .vector files. These files describe deformation fields and are provided as a mechanism for integrating any deformation into AIR. The distribution version of AIR does not provide any programs for creating .vector files.

The program reslice will use the registration parameters in a .air file to resample the reslice file (which is explicitly identified in the .air file) to match the standard file. Nearest neighbor, trilinear, or various forms of

sinc and chirp-z interpolation can be used. The program reslice_warp provides similar functionality for .warp files, as does the program reslice_vector for .vector files. AIR 5.0 In addition to reslicing images, it is possible to use .air or .warp files to remap lists of points that refer to coordinate locations in the images. The programs reslice_ucf and reslice_warp_ucf will perform this remapping.

The alignment programs depend upon the image headers for certain information about file dimensions and sizes. If you do not already have headers for your images, you can create them using the program makeaheader. The program scanheader will display the relevant information contained within an image header so that you can verify that the information is correct. The program fixheader will allow you to modify the real world sizes that correspond to the voxel sizes of an image if the header information about these sizes is inaccurate.

For use with AIR, individual slices from an image must be concatenated into a single image volume. The programs reunite and separate facilitate conversion between data formats that store each slice in a separate file and the multiimage format required by AIR.

The programs scanair and scan_warp allow you to display the information contained in .air and .warp files. On a more limted basis, the program scan_vector provides similar functionality for .vector files. Several programs are available to make specific modifications to .air and .warp files. The programs mv_air and mv_warp allow you to change the name of the reslice file designated in a preexisting .air or .warp file. This feature allows you to use the same parameters to resample multiple files that you know (or assume) to be spatially equivalent to one another. The programs cd_air and cd_warp allow you to update .air and .warp files to reflect the fact that the reslice file has been moved to some other directory in the file hierarchy. The same effect could be achieved using mv_air and mv_warp, but cd_air and cd_warp do not require you to keep track of the specific name of the reslice file (which is unmodified) and are therefore well suited to safely changing only the directory name. The program invert_air will exchange the reslice and standard files (adjusting the transformation matrix accordingly) so that you can use parameters originally derived for aligning file A to B to align file B to A instead. There is no counterpart program for .warp files due to the fact that nonlinear transformations cannot generally be analytically inverted. Instead of analytic inversion, numerical inversion can be performed on-the-fly by the programs reslice_unwarp and reslice_unwarp_ucf. The program combine_air allows you to combine multiple .air files into a single .air file that will have the same effect as applying the individual files sequentially to a single data set, but without the accumulation of interpolation errors and loss of data outside the field of view that occurs with sequential resampling.The program combine_warp allows a single .warp file to be combined with .air files and is an exception to the general rule that programs do not deal with both .air and .warp files.

Because the .air, .warp and .vector files are stored separately from the image files, an additional level of file identification has been incorporated into these files. In addition to the names of the files, a ten digit identifier is computed based on the data in the file. This identifier is displayed along with the other information shown by scanair, scan_warp and scan_vector. In the event that you are unsure that a file is the file used to derive a set of registration parameters, you can use the program identify to compute the ten digit number that corresponds to a given data file.

Several programs are provided to manipulate the size or orientation of files without changing any voxel values. The program reorient allows you to rotate your data 90 degrees or 180 degrees around the x-, y-, or z- axis and also makes it possible to flip the data (effectively converting it to its mirror image) along any of

these axes. The program resize will shift, clip or pad your data to achieve any desired matrix size. The program crop will allow planes that contain no data to be trimmed from the edges of an image file. The program layout will composite multiple slices from an image into a single 2D file to create illustrations, etc.

Other programs reformat data in specified ways that do alter voxel values. The program zoomer will interpolate a study with anisotropic voxel sizes to generate a volume composed of cubic voxels. The program magnify uses Fourier interpolation to increase the number of voxels along a given image dimension. The program gsmooth will apply a Gaussian smoothing filter to a file.

Several of the programs that reformat data can also optionally save .air files describing the transformation that they perform. These include crop, reorient, resize and zoomer.

The program manualreslice is the only interactive program in the AIR package. This program requires you to specify values for roll, pitch, yaw, x_shift, y_shift, and z_shift, x-axis_scaling, y-axis_scaling and z-axis_scaling. These values can then be stored in an initialization file for use with the linear automated alignment programs. Alternatively, these values can be used to manually generate a .air file or to manually reslice a file according to these parameters without generating a .air file. When used together with the capabilities of combine_air, and reslice, or with combine_warp and reslice_warp, manually created .air files make it possible for you to resample your data consistently with a pixel size and interplane distance of your choosing.

The program definecommon_air will average together a series of .air files to define an "average" common space for data analysis. The program reconcile_air will compare various .air files with redundant, potentially conflicting information and will create new .air files that are more internally consistent with one another.

The program softmean will take missing data into account in generating a mean image from resliced data that can be used for additional subsequent automated image registration.

Several programs are provided to manage binary files. These are useful for saving editing information, regions of interest, etc.To create a binary file from a non-binary file, use the program binarize. Binary files can be combined, intersected, subtracted, etc., from one another using the program binarymath. The program binarymask will apply a binary file to a non-binary file to create a masked non-binary file. The program overlay_mask provides a shortcut for one use of these binary programs, namely, the ability to overlay thresholded regions of one image as 'blobs' on another image. The program determinant_mask will create a binary file from a .warp file, showing locations in which the .warp file has a positive determinant.

The program setheadermax will change the global maximum value in the header of 16 bit images so that they can be converted to 8 bit images with as many significant figures of precision as possible. This program is irrelevant when AIR is compiled as a 16-bit program.

The program sizeof shows the size of various variable types generated by your C compiler and compares them to the corresponding sizes in the AIR development environment.

QUANTITATION AND REGISTRATION

With the AIR package, you should be able to reorient tomographic images of the same subject to match one another. If both images were acquired using the same modality, it is likely that you will want to perform a quantitative comparison of the two images to see if there has been any change between the two images. In making such comparisons, you should be aware of the fact that there are a number of issues that may arise when comparing a pair of reoriented images that do not arise when comparing a pair of images that were acquired in identical positions:

1. Any errors in normalization of values from different imaging planes will lead to artifacts.
2. Any spatial distortions in your imaging system will lead to artifacts.
3. Any errors in measurement of voxel sizes will lead to artifacts.
4. In PET scanning, misalignment of one of the emission studies relative to the transmission study will lead to artifacts.
5. Unless you have truly isotropic image resolution in all three dimensions, the process of reorienting the images will misalign the axes of best and worst resolution leading to artifacts.
6. Differences in partial volume effects related to discrete sampling of the signal will lead to artifacts.
7. Post-reconstruction resampling of the data in the process of reslicing will lead to interpolation artifacts.
8. Reorienting only one of the images can produce systematic biases in your data. If you always reslice condition B to match condition A and then perform some statistical test on the difference, a statistically significant finding could merely reflect the fact that the image from condition B was always reoriented rather than the fact that condition B was truly different from condition A.

Problems 1-3 are particularly amenable to quantitative analysis using phantoms, and it is strongly recommended that you perform such studies and fix the problems if you suspect that they are leading to errors in quantitation. Problems 4-7 are more difficult to deal with, and in some cases may be unavoidable. A particularly difficult situation can arise if some state or task of interest is significantly associated with a systematic variation in head position. In this case, it may not always be possible to separate the errors in quantitation due to problems 4-7 from true changes. For band-limited data, the use of sinc interpolation may help to minimize resampling interpolation artifacts, but most tomographic data is not fully band-limited in three dimensions at present. Problem 8 can generally be avoided by randomizing or pseudorandomizing which image is to be resliced.

You should note that quantitative errors above are equally applicable to any method of analyzing misregistered data (e.g., the use of regions of interest) but that they are particularly evident when the images are analyzed on a pixel-by-pixel basis (which essentially amounts to extremely tiny ROI's).

With the exception of sinc and chirp-z interpolation for band-limited data, the AIR package does not incorporate any features intrinsically designed to deal with the quantitation problems described above. In some situations, on-line registration, which is not discussed or supported in this release but is discussed in the first paper describing the PET-PET registration method, minimizes many of these quantitative errors.

Note that the AIR package completely separates the derivation of the registration parameters from the final resampling of the data. Consequently, you are able to apply the spatial information contained in the .air files at any point in the data analysis that you like. On-line registration is one extreme, with the information being used to make modification even before data is acquired. However, it should also be possible to apply this information post acquisition, but prereconstruction, etc.

METHOD OF INTERPOLATION

The AIR package now supports nearest neighbor, trilinear, and various forms of sinc and chirp-z interpolation.

THE PROBLEM OF MISSING DATA

DATA MISSING DUE TO MISREGISTRATION

When a misregistered file is resliced to create a registered file, a portion of the resulting file will generally be missing because it was outside the field of view in the original image. Particularly if the field of view is limited, the missing data may include voxels that are within the brain on the reference image. The reslicing programs in the AIR package will assign a voxel value of zero to the regions of missing data which will have linear edges for when using linear models but may have curved edges when using nonlinear spatial transformation models. Users unfamiliar with this possibility may be puzzled by the absence of data, especially since even a slight movement may suffice to lose an entire plane of data in the resliced images (AIR will not extrapolate outside the boundaries of the original image, no matter how trivial the excursion outside these boundaries).

If you have software originally developed for analyzing images that did not require realignment, you should review the consequences of assigning values of zero to missing data.

REGISTRATION OF IMAGES WITH MISSING DATA

When performing registration, the AIR package cannot distinguish between voxels within the images that are zero because of missing data and voxels that are truly zero. Consequently, except in special situations, images to be registered with the AIR package should not have missing data within the brain (structures outside the brain such as scalp, skull, and dura may be missing due to recommended editing for MRI-PET registration or for intersubject registration). Consequently, it is generally best to avoid registering images that have already been resliced. Where possible, registration to the original images (which are assured not to include missing data) is preferred. In instances where one of the registration targets needs to be an average of several images that have been coregistered, use of the program softmean to create the average image can help to assure that no voxels have missing data.

One situation where it may be important to edit the data to remove data within the brain is when brain pathology has led to extreme focal changes between acquisitions that are disrupting the registration process. In this situation, it is possible to force the alignlinear algorithm to perform a unidirectional fit by using the -p1 0 or -p2 0 flags. These flags allow one of the images to be edited to exclude areas that should not be included when deriving the registration parameters.

PREFERRED IMAGE ORIENTATION

There is no preferred or required image orientation for the AIR package, so long as you are consistent (or at least able to keep track of your inconsistencies and consistently take them into account when using the programs). The package should not be expected to rotate an image by more than 45 degrees from its initialization parameters, and an increasing failure rate should be anticipated as the necessary alignment

approaches this value. The package is incapable of recognizing that two images are mirror image versions of one another, a problem that is particularly likely to go unrecognized when the two images being registered start out in different formats. Don't forget that two data sets can be three dimensional mirror images of one another by virtue of having their planes ordered in opposite directions (top to bottom versus bottom to top). A utility program called reorient is provided to allow you to perform repositioning and mirror imaging as needed to get the images oriented for automated registration. Additional fine tuning of the initial registration can be provided when necessary by using an initialization file with the program alignlinear or align_warp.

CONSTRAINTS FOR IMAGE DIMENSIONS AND VOXEL SIZES

AIR does not require voxel sizes to be isotropic along any dimension. You should be able to reorient a coronally or sagitally acquired MRI scan and align it to a PET study without having to worry about the fact that the resulting pixel sizes are anisotropic.

The AIR package is designed to be extremely flexible with regard to the dimensions of your image. For practical purposes, you are limited only by the amount of contiguous RAM that is available for loading an image into memory. It is recommended that you not interpolate your original data to generate cubic (or approximately cubic) voxels prior to using the AIR package. The package has internal interpolation capabilities that are quite fast, avoiding the need to use extra disk space to save what is actually redundant information. You will also avoid the quantitation errors introduced by repeatedly reinterpolating the same data. The one exception to this recommendation is that if you are averaging together multiple realigned images with the intent of using the averaged value for subsequent registration, there may be some advantage to reslicing the realigned images to cubic voxels and averaging them as cubic voxels.

PREFERRED NUMBER OF BITS/PIXEL

The AIR package can be compiled to use either 8 or 16 bits/pixel for internal representation of voxels. Regardless of its internal representation, the AIR package will load images of either 8 bits/pixel or 16 bits/pixel. Images are converted to AIR's internal format when they are loaded. The AIR package will only generate output images with the number of bits/pixel that it uses for internal representation. For example, if the AIR package is compiled for 8 bits/pixel, it can register two images that have 16 bits/pixel and generate an appropriate .air file. If this .air file is used by the program reslice compiled for 8 bits/pixel, the original 16 bit file will be loaded and resliced, but the output image will be saved as 8 bits/pixel. Note however, that the same .air file that was generated by an 8 bits/pixel registration program can be utilized by a 16-bit version of reslice to generate a 16 bits/pixel resliced image. You should be aware of the fact that this process will remap the absolute pixel values (in a systematic way). If you are doing absolute quantitation, you will need to take this remapping of pixel values into account. Issues related to 16 bit images are discussed in detail in the technical notes.

The interconversion between different data formats is handled exclusively by the data loading subroutines, and the other components of the package generally do not have any way of determining what the format of the original data actually was. For 16 bit data, an additional complication is imposed by the fact that there are three different ways that 16 bit data may be represented externally. The AIR package uses the header global maximum and global minimum values to determine which of the three data types is being used. This issue is discussed in detail in the technical notes.

PREFERRED IMAGE RESOLUTION

This is an extremely complicated issue, and most of what is said here consists of empirical guidelines that should be adjusted by your own practical experience.

PET data (intrasubject intramodality):
Extremely noisy high resolution PET images generally take longer to register than smoother, lower resolution images. Furthermore, slight (subpixel size) misregistration of data that is actually already perfectly aligned can result in a small amount of additional smoothing due to the trilinear interpolation process used in the AIR package. This additional smoothing may make the slightly misregistered images "look" better aligned to the AIR package than the registered images. This effect is minimized when the images are already fairly smooth (see 1992 reference for further discussion of this problem). Finally, there are some theoretical reasons to suspect that registration may be particularly robust when the image smoothness is isotropic in all three dimensions.

With these generalizations in mind, we currently reconstruct our PET H215O images with the highest possible resolution (~6mm in-plane, 10 mm axially) and smooth them to isotropic resolution (10mm in all directions) either using gsmooth or using the internal smoothing capabilities of the program alignlinear before registering them to each other.

MRI data (intrasubject, intramodality):
Our validation work suggests that slight smoothing of MRI data with a 2 mm Gaussian filter improves the internal consistency of the resulting transformations. It is uncertain whether this also results in an improvement in true accuracy. If you are going to ultimately smooth the data more anyway, there is little reason to avoid smoothing in the registration process. If you plan to keep and analyze the data at high resolution, arguments can be made both for and against smoothing during registration (note that smoothing to derive registration parameters is independent of smoothing during the ultimate resampling of the data using those parameters).

Intermodality registration:
For intermodality (e.g. MRI to PET) registration, there are theoretical arguments that the PET scans should have the highest possible resolution (assuming that the MRI resolution will be even higher still). Consequently, we realign our 6mm resolution PET images (using the registration parameters derived by smoothing them) to match one another and then average them all together using the program softmean to generate a high resolution, moderately noisy mean image which is then registered to the edited MRI scan. Using somewhat smoother images (e.g., FWHM 7-8 mm) still provides excellent results for MRI to PET registration (see 1993 reference). We do not smooth the MRI images.

Intersubject registration:
Smoothed and unsmoothed data give quite similar results for intersubject registration.

OVERWRITING OF FILES

In general, the programs in the AIR package will not overwrite existing files unless you have explicitly granted permission to do so. However, there is an exception to this rule. The programs alignlinear and align_warp will always overwrite a preexisting file with the same name as the output file--practical experience

has shown that the annoyance of having the program refuse to save the results of many minutes or even tens of minutes of iterative computation because a file with that name already existed outweighs the risk of losing data. A few safeguards have been built in: the programs will announce the fact that they intend to overwrite an existing file before starting the iterations, hopefully giving you sufficient time to type control-C to terminate the program if you want. Also the programs will reject any file name containing ".img" or ".hdr" as an output file name. **To avoid any possibility of data loss, it is recommended that you always make a copy of data that could not be easily replaced if overwritten and that you store the data in a safe location in your directory where it cannot be accidentally overwritten by these programs.**

SOME INTRODUCTORY EXAMPLES TO USING AIR PET-PET and MRI-PET

HOW TO VERIFY AND VOXEL SIZES AND FILE DIMENSIONS

Use the program scanheader to review the header information that the AIR package will use to align and manipulate your studies.

Example:

```
scanheader pet1
```

will display the header information for the study pet.img

The AIR package provides a means for directly modifying voxel sizes (see program fixheader) If the values for the file dimensions or bits/pixel are incorrect, the program makeaheader can be used to make a new, corrected header.

HOW TO REGISTER TWO PET STUDIES

1. Verify the voxel sizes and file dimensions of the studies
2. Decide which study you want to reslice--this study will be called "pet2" in this example. The other study will be called "pet1"
3. Use alignlinear to derive a registration parameter file (called pet2.airpet1 in this example).

   If your PET images are not very noisy:

   ```
   alignlinear pet1 pet2 pet2.airpet1 –m
   ```

   If your PET images have voxels that are extremely anisotropic

   ```
   alignlinear pet1 pet2 pet2.airpet1 -m 6 –z
   ```

   If your PET images are moderately noisy:

   ```
   alignlinear pet1 pet2 pet2.airpet1 -m 6 -b1 5.0 5.0 0.0 -b2 5.0 5.0
   0.0
   ```

   If your PET images are extremely noisy with voxels that are extremely anisotropic:

```
alignlinear pet1 pet2 pet2.airpet1 -m 6 -b1 8.0 8.0 0.0 -b2 8.0 8.0
0.0 -z
```

This may take several minutes.

4. Use reslice to reslice one file to match the other. The resliced file will be called rpet2 (or crpet2 if you want it interpolated to cubic voxels).

   If you want the reslice file to have the same voxel z_size and number of planes as the standard_pet file:

   ```
   reslice pet2.airpet1 rpet2 -k
   ```

   If you want the reslice file interpolated to cubic voxels in addition to being resliced:

   ```
   reslice pet2.airpet1 crpet2
   ```

HOW TO INTERPOLATE YOUR STANDARD FILE TO CUBIC VOXELS

Use zoomer to interpolate the standard pet to cubic voxels. The resulting file in this example will be called cpet1

```
zoomer pet1 cpet1
```

HOW TO REVIEW THE CONTENTS OF A .AIR FILE

To review the contents of .air file pet2.airpet1, use scanair:

```
scanair pet2.airpet1
```

HOW TO INVERT A .AIR FILE TO REVERSE THE DIRECTION OF RESLICING

1. Review the contents of the .air file you want to invert

2. Use invert_air to create a new .air file

   ```
   invert_air pet2.airpet1 pet1.airpet2
   ```

3. Use reslice to create the desired file

   ```
   reslice pet1.airpet2 rpet1 -k
   ```

HOW TO ALIGN AND AVERAGE SEVERAL PET STUDIES IN PREPARATION FOR MRI-PET REGISTRATION

1. Choose one of the studies to serve as the standard to which all the others will be registered (pet1) will be chosen here

2. Derive registration parameters to reslice each study to this standard using alignlinear:

   ```
   alignlinear pet1 pet1 pet1.airpet1 -m 6
   alignlinear pet1 pet2 pet2.airpet1 -m 6
   alignlinear pet1 pet3 pet3.airpet1 -m 6
   ```

```
alignlinear pet1 pet4 pet4.airpet1 -m 6
alignlinear pet1 pet5 pet5.airpet1 -m 6
alignlinear pet1 pet6 pet6.airpet1 -m 6
```

(You may want to include the -b option with additional smoothing if the images are noisy or the -z option if the voxels are extremely anisotropic)

3. Reslice each study to match the standard and interpolate them all to cubic voxels at the same time:

```
reslice pet1.airpet1 crpet1
reslice pet2.airpet1 crpet2
reslice pet3.airpet1 crpet3
reslice pet4.airpet1 crpet4
reslice pet5.airpet1 crpet5
reslice pet6.airpet1 crpet6
```

4. Use softmean to create a mean image (called meanpet here) suitable for registration with an MRI

```
softmean meanpet n crpet1 crpet2 crpet3 crpet4 crpet5 crpet6
```

HOW TO REORIENT AN MRI IMAGE IF IT IS UPSIDE DOWN, BACKWARDS, AND/OR MIRROR IMAGED IN COMPARISON TO YOUR PET IMAGES

1. Use your display package to decide what needs to be done.

2. Use reorient to create a new MRI file in the position you want. In this example, the original upside-down and backwards MRI (ubmri) will be rotated 180° around the x-axis to create a properly oriented MRI (mri):

```
reorient ubmri mri xx
```

HOW TO ALIGN A PET STUDY (OR AVERAGED PET STUDY) TO AN MRI STUDY

1. Reorient the MRI image if needed.

2. Use your image editing package to edit the MRI to remove the scalp, skull, and meninges to create an edited MRI file (emri)

3. Decide whether you want to derive parameters to match the MRI to the PET or the PET to the MRI. (You can always use invert_air if you change your mind later, or if you want both).

4. Use alignlinear to align the studies. In this example, the edited MRI (emri) will be aligned to an averaged PET (meanpet) to create a registration parameter file called emri.airmeanpet

```
alignlinear meanpet emri emri.airmeanpet -m 6 -p1 0 -p2 256 -t2 10
```

5. Reslicing of the MRI to match the PET or the PET to match the MRI using the derived registration parameter file is completely analogous to reslicing of PET studies as described above.

HOW TO RESLICE THE ORIGINAL UNEDITED MRI TO MATCH THE PET STUDY USING REGISTRATION PARAMETERS DERIVED WITH AN EDITED MRI

1. If you used alignpettomri to derive the registration parameters, you will need to create an inverted registration parameter file using invert_air:
   ```
   invert_air meanpet.airemri emri.airmeanpet
   ```
2. Copy the registration parameter file to a new file (uemri.airmeanpet in this example).
   ```
   cp emri.airmeanpet uemri.airmeanpet
   ```
3. Use mv_air to change the file to be resliced to the uneditedmri (mri) in the registration parameter file:
   ```
   mv_air uemri.airmeanpet mri
   ```
4. Use reslice to create the new resliced MRI file (rmri) to match your PET file:
   ```
   reslice uemri.airmeanpet rmri –k
   ```
5.

HOW TO RESLICE EACH OF YOUR ORIGINAL PET STUDIES TO MATCH THE MRI STUDY

1. Register each of your original PET studies to the study that will serve as the standard as described above.
2. Average your PET studies to make a mean image as described above.
3. Align the MRI study to the mean image as described above.
4. If you used alignlinear to align the MRI to the PET study, apply invert_air to the resulting registration parameter file to get parameters to align the mean PET to the MRI.
5. Use combine_air to combine the registration parameter file for aligning the mean PET to the MRI (meanpet.airemri) with the registration parameter files for aligning each of the PET studies to the standard PET to create new registration parameter files for directly reslicing the PET studies to match the MRI:
   ```
   combine_air pet1.airemri n meanpet.airemri pet1.airpet1
   combine_air pet2.airemri n meanpet.airemri pet2.airpet1
   combine_air pet3.airemri n meanpet.airemri pet3.airpet1
   combine_air pet4.airemri n meanpet.airemri pet4.airpet1
   combine_air pet5.airemri n meanpet.airemri pet5.airpet1
   combine_air pet6.airemri n meanpet.airemri pet6.airpet1
   ```
   In each case, the program will state that the parameters are valid only if meanpet is spatially equivalent to pet1 . It is if you have followed the examples as outlined here.
6. Use reslice to generate the resliced files (rrpet1, rrpet2, etc.):
   ```
   reslice pet1.airemri rrpet1 –k
   reslice pet2.airemri rrpet2 –k
   reslice pet3.airemri rrpet3 –k
   reslice pet4.airemri rrpet4 –k
   ```

```
reslice pet5.airemri rrpet5 –k
reslice pet6.airemri rrpet6 -k
```

SOME INTRODUCTORY EXAMPLES TO USING AIR (MRI-MRI)

HOW TO CONVERT SLICE DATA INTO VOLUME DATA

1. Remove all header information contained within each slice data file and convert the slice data to either 8 bit or 16 bit integer values. The AIR package does not include file format converters for removing header information or otherwise converting proprietary formats. Some file format converts refer to the desired format as "raw" format. You can verify that the file does not contain any extra header information by computing the single slice data file's size in bytes as (x_dimension*y_dimension*bits/pixel)/8 and comparing this with the actual size of the file in bytes.

2. Change the name of each slice data file so that it contains only a single period followed by img at the end of the file name (e.g., mri01.img).

3. Use the program makeaheader to create a .hdr file corresponding to each .img slice data file. You need to know the x and y dimensions of each image (use a z-dimension of one for single slice data). You also need to specify the voxel sizes along each dimension (the z-dimension should be the interslice distance, which is not necessarily the same as the slice thickness). Finally, you must know the number of bits per pixel, and, for 16 bit data, how the data is stored. All of the files that are to be converted into a single volume must have identical dimensions and voxel sizes, so you can make one .hdr file and just copy it for the other images if you like.

   Example:
   ```
   makeaheader mri01 3 256 256 1 0.8 0.8 3.0
   ```

   will create a file called mri1.hdr for a 256x256 plane of type 3 16 bit data with voxel dimensions of 0.8 mm (i.e., FOV=204.8) and an interslice distance of 3.0 mm.

4. Use the program reunite to combine the individual slices into a single volume.

   Example:
   ```
   reunite mrivolume y mri01 mri02 mri03 mri04 mri05 mri06 mri07 mri08
   mri09 mri10
   ```

   will create files called mrivolume.hdr and mrivolume.img containing the data from the 10 specified slices, mri01.img, mri02.img ... mri10.img. Note that the order in which the file names are entered dictates the ordering of the planes.

HOW TO VERIFY AND VOXEL SIZES AND FILE DIMENSIONS

Use the program scanheader to review the header information that the AIR package will use to align and manipulate your studies.

Example:
```
scanheader mri01
```

will display the header information for the study mri01.img

The AIR package provides a means for directly modifying voxel sizes (see program fixheader) If the values for the file dimensions or bits/pixel are incorrect, the program makeaheader can be used to make a new, corrected header.

HOW TO REGISTER TWO MRI STUDIES

1. Verify the voxel sizes and file dimensions of the studies.

2. Decide which study you want to reslice--this study will be called "mri2" in this example. The other study will be called "mri1".

3. Identify a voxel value in the studies that will reliably distinguish areas outside the body from values in the brain. In this example, a 16 bit value of 7000 will be used for both files.

4. Use alignlinear to derive a registration parameter file (called mri2.airmri1 in this example).

   If you want to use the default smoothing and pre-alignment interpolation:
   ```
   alignlinear mri1 mri2 mri2.airmri1 -m 6 -x 3 -c 0.1 -t1 7000 -t2 7000
   ```

   If your MRI images have voxels that are extremely anisotropic
   ```
   alignlinear mri1 mri2 mri2.airmri1 -m 6 -x 3 -c 0.1 -t1 7000 -t2 7000 –z
   ```

   If you want the registration to be based on slightly smoothed versions of the data:
   ```
   alignlinear mri1 mri2 mri2.airmri1 -m 6 -x 3 -c 0.1 -t1 7000 -t2 7000
    -b1 2.0 2.0 2.0 -b2 2.0 2.0 2.0
   ```

   If you have data where the scalp and skull are prominent and there is substantial initial misregistration you can do a two stage registration with heavy initial smoothing followed by normal smoothing:
   ```
   alignlinear mri1 mri2 mri2.airmri1 -m 6 -x 3 -c 0.1 -t1 7000 -t2 7000
    -b1 8.0 8.0 8.0 -b2 8.0 8.0 8.0 -g mri2.init -gs mri2.inits

   alignlinear mri1 mri2 mri2.airmri1 -m 6 -x 3 -c 0.1 -t1 7000 -t2 7000
    -b1 2.0 2.0 2.0 -b2 2.0 2.0 2.0 -f mri2.init -fs mri2.inits
   ```

5. Use reslice to reslice one file to match the other. The resliced file will be called rmri2 (or crmri2 if you want it interpolated to cubic voxels).

   If you want the reslice file to have the same voxel z_size and number of planes as the standard_mri file:

```
reslice mri2.airmri1 rmri2 –k
```

If you want the reslice file interpolated to cubic voxels in addition to being resliced:

```
reslice mri2.airmri1 crmri2
```

HOW TO INTERPOLATE YOUR STANDARD FILE TO CUBIC VOXELS

Use zoomer to interpolate the standard mri to cubic voxels. The resulting file in this example will be called cmri1

```
zoomer mri1 cmri1
```

HOW TO REVIEW THE CONTENTS OF A .AIR FILE

To review the contents of .air file mri2.airmri1, use scanair:

```
scanair mri2.airmri1
```

HOW TO INVERT A .AIR FILE TO REVERSE THE DIRECTION OF RESLICING

1. Review the contents of the .air file you want to invert
2. Use invert_air to create a new .air file
   ```
   invert_air mri2.airmri1 mri1.airmri2
   ```
3. Use reslice to create the desired file
   ```
   reslice mri1.airmri2 rmri1 –k
   ```

HOW TO CONVERT VOLUME DATA BACK INTO SLICE DATA

Use the program **separate**:

```
separate mrivolume smri y
```

This command will take the volume "mrivolume.img" and create a series of images "smri001.img", "smri002.img", "smri003.img" ... which are raw format data files containing one slice of data per file.

HOW TO REORIENT AN MRI VOLUME IF IT IS UPSIDE DOWN, BACKWARDS, AND/OR MIRROR IMAGED

1. Use your display package to decide what needs to be done.
2. Use reorient to create a new MRI file in the position you want. In this example, the original upside-down and backwards MRI (ubmri) will be rotated 180° around the x-axis to create a properly oriented MRI (mri):

   ```
   reorient ubmri mri xx
   ```

SOME INTRODUCTORY EXAMPLES TO USING AIR (SUBJECT-SUBJECT)

HOW TO VERIFY AND VOXEL SIZES AND FILE DIMENSIONS

Use the program scanheader to review the header information that the AIR package will use to align and manipulate your studies.

Example:

```
scanheader pet1
```

will display the header information for the study pet.img

The AIR package provides a means for directly modifying voxel sizes (see program fixheader) If the values for the file dimensions or bits/pixel are incorrect, the program makeaheader can be used to make a new, corrected header.

HOW TO REGISTER SUBJECTS TO ONE ANOTHER OR TO AN ATLAS USING A LINEAR AFFINE SPATIAL TRANFORMATION

1. If you are using MRI data, manually edit the data to remove nonbrain structures. For PET data with a PET atlas, editing is not required.

2. In this example, the file called atlas can either be an edited image or averaged edited image in a standardized space (e.g., Talairach space) or it can simply be edited data from another subject in its native space.

   Use alignlinear to derive a registration parameter file (called mri01.airatlas in this example)
   ```
   alignlinear atlas mri01 mri01.airatlas -m 12 -x 3 -c 1
   ```

   This can take many minutes.

3. Use reslice to reslice the mri01 to match the atlas. The resliced file will be called rmri01 (or cmri01 if you want it interpolated to cubic voxels).

   If you want the reslice file to have the same voxel z_size and number of planes as the standard_pet file:
   ```
   reslice mri01.airatlas rmri01 –k
   ```

   If you want the reslice file interpolated to cubic voxels in addition to being resliced:
   ```
   reslice mri01.airatlas crmri01
   ```

HOW TO CREATE YOUR OWN ATLAS

1. Pick one subject and register the images of all subjects to the one you have selected (register that subject to their self as well). This example will assume that you have registered three subjects to

subject number one to generate .air files, "mri01.airmri01", "mri02.airmri01", and "mri03.airmri01".

2. Use definecommon_air to create .air files for registering each individual into a space that approximates the average size, shape and orientation of the original images of the subjects:

```
definecommon_air mri .airmri01 mri .aircommon y 01 02 03
```

This will produce output .air files, "mri01.aircommon", "mri02.aircommon" and "mri03.aircommon".

3. Use reslice to resample each original file into the common space:

```
reslice mri01.aircommon crmri01
reslice mri02.aircommon crmri02
reslice mri03.aircommon crmri03
```

4. Use softmean to average the files in the common space:

```
softmean atlas y null crmri01 crmri02 crmri03
```

This will create an averaged file called "atlas.img"

5. If you like, you can repeat this entire process using "atlas" as the target in step 1 (don't register "atlas" to itself). You are unlikely to see much change after one or two iterations.

6. If you want the atlas to have a standardized orientation (e.g., with the AC-PC line horizontal), you can figure out the transformation required to achieve this orientation and then use manualreslice to create a .air file that will allow "atlas" to be resliced to that orientation (use only rigid-body transformations, i.e.no rescaling). Suppose that this file is called "atlas.airstandardspace", you can then create .air files that will allow each individual file to be resliced directly into that standard space using combine_air:

```
combine_air mri01.airstandardspace y atlas.airstandard mri01.airatlas
combine_air mri02.airstandardspace y atlas.airstandard mri02.airatlas
combine_air mri03.airstandardspace y atlas.airstandard mri03.airatlas
```

7. Now you can reslice each individual file directly into the standard space:

```
reslice mri01.airstandard crmri01 –o
reslice mri02.airstandard crmri02 –o
reslice mri03.airstandard crmri03 –o
```

and create an average atlas in the standard space:

```
softmean standardatlas y null crmri01 crmri02 crmri03
```

The file "standardatlas.img" is the final atlas.

HOW TO REGISTER SUBJECTS TO AN ATLAS USING NONLINEAR SPATIAL TRANSFORMATIONS

1. If you are using MRI data, manually edit the data to remove nonbrain structures. For PET data with a PET atlas, editing is not required.

2. In this example, the file called atlas can either be an edited image or averaged edited image in a standardized space (e.g., Talairach space) or it can simply be edited data from another subject in its native space.

   Use align_warp to derive a registration parameter file (called mri01.warpatlas in this example) using a 5th order polynomial fit for MRI data (or a 3rd order polynomial fit for PET data).

   For MRI data:
   ```
   align_warp mriatlas mri01 mri01.warpmriatlas -m 1 5
   ```

   For PET data:
   ```
   align_warp petatlas pet01 pet01.warppetatlas -m 1 3
   ```

   This can take a long time. You may get better results by initializing align_warp with a linear tranformation derived using alignlinear. See the align_warp page for details.

3. Use reslice_warp to reslice the original file to match the atlas:

   ```
   reslice_warp mri01.warpmriatlas rmri01 -k
   ```

HOW TO LINK TOGETHER A SERIES OF .AIR FILES AND A .WARP FILE

Suppose that you have:

- pet1.airmeanpet (.air file for registering scan "pet1" to a mean pet space)
- meanpet.airmri (.air file for registering the mean pet space to the subject's MRI)
- mri.warpatlas (.warp file for registering the subject's MRI to an atlas)
- atlas.airstandardspace (.air file for registering the atlas into some standardized space)

and you want a single file for transforming "pet1" directly into the standardized space.

1. Use combine_air to combine any sequential .air files into a single tranformation:

   ```
   combine_air pet1.airmri y meanpet.airmri pet1.airmeanpet
   ```

2. Use combine_warp to combine .air and .warp files into a single .warp file:

   ```
   combine_warp pet1.warpstandardspace y atlas.airstandardspace
   ```

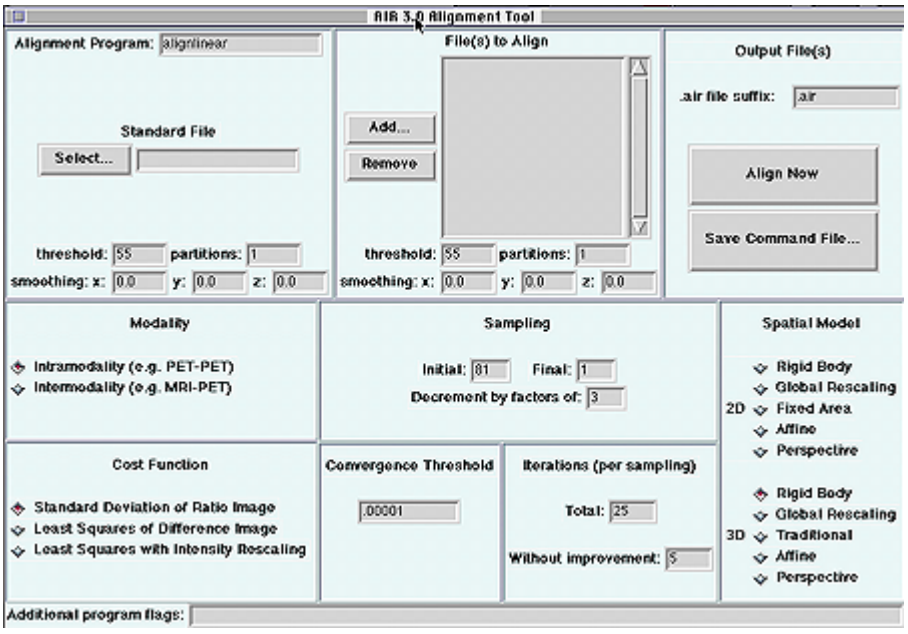
```
mri.warpatlas pet1.airmri
```

3. Use reslice_warp to resample "pet1.img"

```
reslice_warp pet1.warpstandardspace rpet1 -k
```

GRAPHICAL INTERFACES TO AIR

EXAMPLE SCREEN



Here is what the AIR alignment tool looks like. There is also a similar reslice tool.

WHY USE TCL/TK

The programs alignlinear, align_warp, reslice, and reslice_warp form the core of the AIR package. The number of options associated with these programs can make them difficult to use. Consequently, a graphical user interface (GUI) is available using Tcl/Tk. Tcl/Tk is a freely available package for creating window driven front ends to programs on UNIX platforms. You must download, compile, and install Tcl/Tk before you can use the AIR 5.x GUI.

GETTING AND COMPILING TCL/TK

If you do not already have Tcl/Tk, you can download it at: http://www.scriptics.com.

MODIFYING THE TCL/TK AIR SCRIPTS TO WORK PROPERLY

Once Tcl/Tk and AIR 5.x are installed, you need to modify the scripts align.tcl, align_warp.tcl, reslice.tcl and reslice_warp.tcl in the AIR5.x directory before you can successfully use these scripts. The two critical modifications are:

1. Change the first line of each script to identify the full path name for the shell wish (this is the Tk shell). The current first line:
`#!/usr/local/bin/wish4.0`
is only acceptable if you have a Tk shell called wish4.0 in the subdirectory /usr/local/bin on your computer.
2. Change the line near the end of the script that identifies the AIR program that the script will execute. Unless AIR is on your normal search path (so that the programs can be invoked from any directory simply by typing the name of the program), you must supply a full path name to the program (alignlinear or reslice). For example, the current line:
`set program "alignlinear"`
might need to be changed to:
`set program "/usr/local/AIR5.1/alignlinear"`
if you have placed the AIR executable binary files that you have compiled into the subdirectory /usr/local/AIR5.1/.
Similar changes need to be made to align_warp.tcl, reslice.tcl and reslice_warp.tcl

CONFIGURING THE TCL/TK AIR SCRIPTS

All of the default values and colors in the align.tcl, align_warp.tcl, reslice.tcl and reslice_warp.tcl scripts can be configured to better suite your preferences. Colors are located at the begining of the scripts in lines like:

set color1 #fffccceee
These are RGB colors in the form #RRRGGGBBB with values 0123456789abcdef available.
More importantly, thresholds, etc., are configured near the end of the align.tcl script. Default variables that you might want to change are listed here using the alignlinear flags that correspond:

```
-t1 threshold1
-t2 threshold2
-b1 xsmooth1 ysmooth1 zsmooth1
-b2 xsmooth2 ysmooth2 zsmooth2
-x costfxn
-m model
-s initial final decrement
-r iterations
-c converge
```

The program align_warp does not have variables for cost function since only one cost function is implemented.

AIR PROGRAMS ALPABETICALLY

- alignlinear (AIR automated linear registration both within and across subjects)
- align_warp (AIR automated nonlinear registration)

- binarize (converts 8 or 16 bit file to a binary file)
- binarymask (applies a binary mask to a file)
- binarymath (performs binary operations on pairs of binary input files)
- cd_air (changes .air file's target directory)
- cd_warp (changes .warp file's target directory)
- combine_air (combines multiple .air files)
- combine_warp (combines .air and .warp files)
- crop (crops empty planes from images)
- definecommon_air (finds a good common space for data analysis based on .air files)
- determinant_mask (creates a mask where the determinant of a .warp file is positive)
- fixheader (corrects header file's voxel sizes)
- fuse (joins 3D volumes at their edges)
- gsmooth (Gaussian smoothing)
- identify (computes hash value, verifies header and image compatibility)
- invert_air (interconverts .air file's reslice file and standard file)
- layout (creates 2D layout of 3D data)
- magnify (magnifies images using Fourier interpolation)
- makeaheader (creates new header files)
- manualreslice (trial and error registration, creates initialization files)
- mv_air (changes .air file's reslice file)
- mv_warp (changes .warp file's reslice file)
- overlay_mask (superimposes a mask file onto a regular image file)
- reconcile_air (reconciles discrepancies between a series of .air files defining all possible pairwise registrations)
- reorient (rotates file 90 or 180 degrees or creates file's mirror image)
- resize (changes file's matrix size and/or shifts file within matrix)
- reslice (reslices data based on specified .air file)
- reslice_ucf (reslices point list using a .air file)
- reslice_unwarp (reslices data based on numerical inversion of specified .warp file)
- reslice_unwarp_ucf (reslices point list based on numerical inversion of specified .warp file)
- reslice_vector (reslices data based on specified vector field file)
- reslice_warp (reslices data based on specified .warp file)
- reslice_warp_ucf (reslices point list based on specified .warp file)
- reunite (joins multiple 2D files into a single 3D file)
- scanair (displays contents of .air file)
- scanheader (displays contents of header file)
- scan_vector (displays contents of .vector file)
- scan_warp (displays contents of .warp file)
- separate (cuts a 3D volume into multiple 2D files)
- setheadermax (changes header file's maximum)
- sizeof (displays C variable sizes)
- softmean (averages files together compensating for missing data)
- zoomer (interpolates file to cubic voxels)

TECHNICAL NOTES

CREDITS

REFERENCES TO SITE IN PAPERS THAT USE AIR

Primary References

One or more of the following references should be cited in any paper based on data registered using the AIR package or programs utilizing the AIR library.

- Woods RP, Cherry SR, Mazziotta JC. Rapid automated algorithm for aligning and reslicing PET images. Journal of Computer Assisted Tomography 1992;16:620-633.

  The original AIR manuscript validating the intramodality method implemented in AIR 1.0.

- Woods RP, Mazziotta JC, Cherry SR. MRI-PET registration with automated algorithm. Journal of Computer Assisted Tomography 1993;17:536-546.

  The method used for intermodality registration as implemented in AIR 1.0

- Woods RP, Grafton ST, Holmes CJ, Cherry SR, Mazziotta JC. Automated image registration: I. General methods and intrasubject, intramodality validation. Journal of Computer Assisted Tomography 1998;22:139-152.

  Description of AIR 3.0 and validation for intrasubject, intramodality registration of MRI data and intrasubject, intramodality registration of PET data.

- Woods RP, Grafton ST, Watson JDG, Sicotte NL, Mazziotta JC. Automated image registration: II. Intersubject validation of linear and nonlinear models. Journal of Computer Assisted Tomography 1998;22:153-165.

  Validation of AIR 3.0 for intersubject registration.

Secondary references

The following references provide additional validation of AIR.

- Strother SC, Anderson JR, Xu XL, Liow JS, Bonar DC, Rottenberg, DA. Quantitative comparisons of image registration techniques based on high-resolution MRI of the brain. Journal of Computer Assisted Tomography 1994;18:954-62.

  A comparative study using simulations that included AIR 1.0.

- Jiang AP, Kennedy DN, Baker JR, Weisskoff R, Tootell RBH, Woods RP, Benson RR, Kwong KK, Brady TJ, Rosen BR, Belliveau JW. Motion detection and correction in functional MR imaging. Human Brain Mapping 1995;3:224-235.

  Validation of AIR 1.0 in the context of fMRI.

- Black KJ, Videen TO, Perlmutter JS. A metric for testing the accuracy of cross-modality image registration: Validation and application.Journal of Computer Assisted Tomography 1996;20:855-861.

  AIR MRI-PET registration validation in monkeys

- West J, Fitzpatrick JM, Wang MY, Dawant BM, Maurer CR, Kessler RM, Maciunas RJ, Barillot C, Lemoine D. Collignon A, Maes F, Suetens P, Vandermeulen D, van den Elsen P, Napel S, Sumanaweera TS, Harkness B, Hemler PF, Hill DLG, Hawkes DJ, Studholme C, Maintz JBA, Viergever MA, Malandain G, Pennec X, Noz ME, Maguire GQ, Pollack M, Pelizzari CA, Robb RA, Hanson D, Woods RP. Comparison and evaluation of retrospective intermodality brain image registration techniques. Journal of Computer Assisted Tomography 1997;21:554-566.

  AIR 1.0's MRI-PET registration technique in a comparative blinded study.

- Imran MB, Kawashima R, Sat K, Kinomura S, Ito H, Koyama M, Goto R, Ono S, Yoshioka S, Fukuda H. Mean regional cerebral blood flow images of normal subjects using technetium-99m-HMPAO by automated image registration. Journal of Nuclear Medicine 1998;39:203-207.

  Use of AIR for intersubject registration of HMPAO-SPECT images.

ACKNOWLEDGMENTS

Scott T. Grafton, Simon R. Cherry and John C. Mazziotta have been instrumental throughout the development of AIR providing support, encouragement, data, constructive recommendations and feedback. John D.G. Watson, Colin J. Holmes, and Nancy L. Sicotte have been active collaborators in the validation of subsequent versions.

Feedback and encouragement from groups using earlier versions of AIR contributed to the decision to make the software more widely available. John Watson, Ralph Myers, Richard Frackowiak, Jon Heather, Mark Mintun, Tom Nichols, Joel Lee, Tom Zeffiro and Tom Grabowski were especially helpful and supportive.

Aiping Jiang, David Kennedy and their collaborators have played an active role in validating the use of AIR for fMRI data.

The decision to incorporate sinc interpolation in AIR was a direct consequence of Joe Hajnal's important work in this area. Chirp-z interpolation would not have been implemented had Robert Cox not brought this technique to my attention.

Bugs have been identified by Marco Iacoboni, Darren Emge, Kate Fissell, Tom Grabowski, Mark Evans, Greg Ward, Michael Mega, Edward Vessel, William Gandler, Jon Anderson, Hans Johnson, Adolf Pfeffer and others whose names I have misplaced.

Mark Evans provided detailed and helpful suggestions that should make AIR version 3.05 and later portable to PC's and Macintoshes without needing to modify the souce code.
At UCLA, Eric Behnke, George Branch, Rick Cai, Robert Knowlton, Michel Levesque, Larry Pang, Michael Phelps, Ron Sumida, Arthur Toga, Charles Wilson, and Jingxi Zhang all contributed time or resources helpful in the validation of the software.

Mirence Sibomana supports a CTI ECAT wrapper for AIR 3.0.
CTI (Knoxville, TN) provided use of a Sun SPARCstation used previously for software development.
Randy Frank provided example code illustrating the implementation of automatic byteswapping
Validation of the software has been supported by Department of Energy cooperative agreement DE-FC03-87ER 60615, National Institute of Mental Health grant R01-MH37916 and NIH-NINDS grant P01-NS15654, NIH-NINDS grant 1 K08 NS01646 and NCRR grant RR13642. Salary support during the initial rewriting of the software and documentation for widespread distribution was provided by a Fellowship from the Dana Foundation as a Dana Scholar in Neurosciences, and subsequent salary support was provided by NIH-NINDS grant 1 K08 NS01646. Ongoing support for software development and distribution has been provided by The Ahmanson Foundation, the Pierson-Lovelace Foundation and the Brain Mapping Medical Research Organization.

New development work for AIR 5.x has been supported by the NCRR (RR13642)