Building a registration quarantine

This is a tutorial on how to build a MINC2-based registration platform on a linux based platform.

The most easy way (as of December 2011)

Install procedure

The compilation of some of the code fails when using a version of g++ higherthan 4.4, so it is important to use g++-4.4 for the entire build procedure. To get the compiler:

On Ubuntu Lucid 10.04 LTS:
> sudo apt-get install g++-4.4

And to make sure that the correct version of g++ is used:

> export CXX=g++-4.4

On Ubuntu Natty 11.04:

> sudo apt-get install gfortran gcc-4.4 g++-4.4 build-essential
> sudo update-alternatives --install /usr/bin/gcc gcc /usr/bin/gcc-4.5 40 --slave /usr/bin/g++ g++ /usr/bin/g++-4.5
> sudo update-alternatives --install /usr/bin/gcc gcc /usr/bin/gcc-4.4 60 --slave /usr/bin/g++ g++ /usr/bin/g++-4.4
> export CXX=g++-4.4

Please verify that the following

> g++ --version
> gcc --version
> gfortran --version

results in g++ and gcc reporting version 4.4.5, gfortran reporting 4.5.2

1. Dependencies
   apt-get the following before you start on Ubuntu Lucid 10.04 LTS:

```
> sudo apt-get install build-essential realpath zlib1g-dev bison flex libx11-dev glutg3-dev libxmu-dev
byacc libsoqt4-dev libdbi-perl automake libtool libblas-dev liblapack-dev libblitz0-dev libreadline-dev
bzr bzrtools octave3.2 imagemagick swig
```

   apt-get the following before you start on Ubuntu Lucid 11.04 LTS:

```
> sudo apt-get install build-essential realpath zlib1g-dev bison flex libx11-dev glutg3-dev libxmu-dev
byacc libsoqt4-dev automake libtool libdbi-perl libblas-dev liblapack-dev libblitz0-dev libreadline-dev
bzr bzrtools octave3.2 imagemagick swig
```

2. Get the modified Makefile (The basis for the quarantine comes from the Makefile that Vladimir Fonov created: original_Makefile_created_by_Vladimir).

   https://github.com/mcvaneede/Image-Registration-Quarantine-Makefile/blob/master/Makefile

3. To install all software:

   ```
   make full-MICe-quarantine
   ```

   or example with options:

   ```
   make full-MICe-quarantine INSTALL_DIR=/realpath/install/dir  BUILD_DIR=/realpath/build/dir
   PARALLEL_BUILD=-j2
   ```

   Some libraries that are used by the registration software have unclear copyright issues, so we can not have these packages happily sit somewhere on the internet. You can request them from us, and after you have copied the tarballs over to the source (/realpath/build/dir/src) directory, you can complete the installation:

   ```
   make MICe-fuzzy INSTALL_DIR=/realpath/install/dir BUILD_DIR=/realpath/build/dir PARALLEL_BUILD=-j2
   ```

4. Create an environment script:

   ```
   make environment
   ```

   In a bash shell, you can now set the environment for the quarantine as follows:

   ```
   . environment.sh
   ```

Included packages

Package and version information from August 24, 2011:

Package	version	in Vladimir's Makefile	done
arguments	0.2.1	yes	✅
bicInventor	0.3.1		✅
bicpl	1.4.6	yes	✅
brain-view2	0.2 01 sept 2011		✅
classify	1.1.0	yes	✅
cmake	2.8.4	yes	✅
coin3d	3.1.3		✅
conglomerate	1.6.6	yes	✅
Display	1.5.0	yes	✅
ebtks	1.6.4	yes	✅
ezminc	92519b5	yes	✅
fftw	3.2.2	yes	✅
fltk	1.3.x-r7725	yes	✅
Getopt-Tabular	0.3	yes	✅
glim_image	1.2	yes	✅
gsl	1.15	yes	✅
hdf5	1.8.7	yes	✅
ITK	3.20.0	yes	✅

inormalize	1.0.2	yes	✓
laplacian_thickness	1.1.2		✓ (fuzzy*)
MBM	0.6.1		✓ (fuzzy*)
mice-minc-tools	0.1		✓
minc	9e69692 (2.1.01)	yes	✓
mincANTS	1p9_p1		✓
mincblob	1.2.1	yes	✓
mincmorph	1.4	yes	✓
mni_autoreg	0.99.6	yes	✓
mni_perllib	0.08	yes	✓
mouse thickness	0.1		✓
N3	1.12.0	yes	✓
netcdf	4.0.1	yes	✓
netpbm	10.35.74	yes	✓
numpy	1.6.1		✓
oobicpl	0.4.4	yes	✓
pcre++	0.9.5	yes	✓
Perl Test::Files	0.14		✓
PMP	0.7.10		✓ (fuzzy*)
pyminc	0.2		✓
python	2.7.2		✓
Quarter	1.0.0		✓
R	2.13.1		✓
ray_trace	1.0.3	yes	✓
Register	1.4.0	yes	✓
RMINC	latest from Launchpad		✓
scipy	0.9.0		✓
tagtoxfm_bspline	1.0		✓
VTK	5.6.1	yes	✓
xfmavg	1.0.0		

(* fuzzy: these packages have ambiguous copyright issues, so should be requested for)

Testing your quarantine

You can test your quarantine using the with the following files: test_files_28september2012.tar.gz

Step 1, unpack the tarball:

```
> tar xzvf test_files_28september2012.tar.gz
{when you type ls, you should see the following}
>  ls *
mapping_for_striatum.csv
mincANTS_protocol_224_micron.pl
rough_mask.mnc
rough_striatum_segmentation.mnc
test_files_28september2012.tar.gz

initial-model:
native-atlas-large_mask.mnc native-atlas-large.mnc native-atlas_mask.mnc native-atlas.mnc native.mnc native-
small_mask.mnc native-small.mnc native_to_standard.xfm

test_files_striatum_mutants:
mouse_1.mnc mouse_2.mnc mouse_3.mnc mouse_4.mnc mouse_5.mnc mouse_6.mnc
mouse_1_mutant.mnc mouse_2_mutant.mnc mouse_3_mutant.mnc mouse_4_mutant.mnc mouse_5_mutant.mnc mouse_6_mutant.
mnc
```

There are two directories, the test_files_striatum_mutants directory contains sample input images and the initial-model directory contains files that determine the size and orientation of the output files of the registration pipeline. The test files were created such that there is (approximately) a 25% difference in the size of the striatum in the mutant files compared to the originals. To test all the software, you'll run a registration and recover these changes.

Step 2, set your environment:

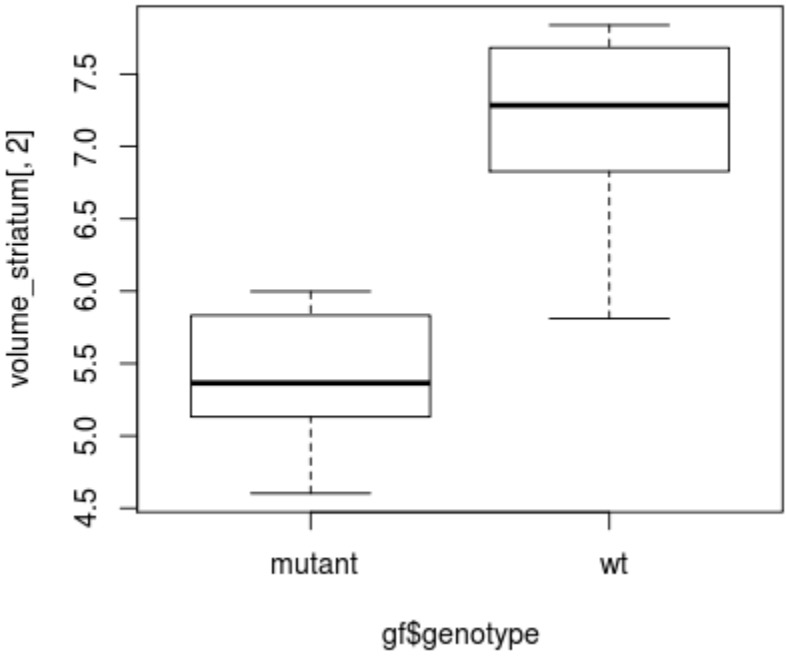
```
{If you ran the "make environment" command when installing the software you can use}
> . environment.sh

{Otherwise you can set your paths manually}
> export PATH=/realpath/install/dir/bin:$PATH
> export PERL5LIB=/realpath/install/dir/perl
> export PYTHONPATH=/realpath/install/dir/python
> export LD_LIBRARY_PATH=/realpath/install/dir/lib:/realpath/install/dir/lib/InsightToolkit
```

Step 3, run the registration (on a single machine should take about 3-5 hours):

```
> MICe-build-model.pl -spawn -pipeline-name test_run_mutants -sinc -no-resample-atlas -no-registration-accuracy
-init-model /where/the/tarball/is/initial-model/native -lsq6 -lsq12 -nlin -nlin-registration-method mincANTS -
nlin-protocol /where/the/tarball/is/mincANTS_protocol_224_micron.pl /where/the/tarball/is
/test_files_striatum_mutants/mouse* -nlin-stats
```

Step 4, recover the change in the striatum using R:

```
# first we will create a comma separated text file that provides the mapping between the Jacobian determinants
we want to examine and the group they belong too:
> echo absolute_volume, genotype > jacobian_determinan_for_absolute_volume.csv; for file in
test_run_mutants_processed/*; do base=`basename $file`; if [[ $base =~ "mutant" ]]; then type="mutant"; else
type="wt"; fi; pwd=`pwd`; echo ${pwd}/${file}/stats-volumes/${base}-log-determinant-scaled-fwhm0.2.mnc, $type;
done >> jacobian_determinan_for_absolute_volume.csv

# this should look something like this, there the ---***--- will point to the directory you have your pipeline
in:
absolute_volume, genotype
---***---/test_run_mutants_processed/mouse_1/stats-volumes/mouse_1-log-determinant-scaled-fwhm0.2.mnc, wt
---***---/test_run_mutants_processed/mouse_1_mutant/stats-volumes/mouse_1_mutant-log-determinant-scaled-fwhm0.2.
mnc, mutant
---***---/test_run_mutants_processed/mouse_2/stats-volumes/mouse_2-log-determinant-scaled-fwhm0.2.mnc, wt
---***---/test_run_mutants_processed/mouse_2_mutant/stats-volumes/mouse_2_mutant-log-determinant-scaled-fwhm0.2.
mnc, mutant
---***---/test_run_mutants_processed/mouse_3/stats-volumes/mouse_3-log-determinant-scaled-fwhm0.2.mnc, wt
---***---/test_run_mutants_processed/mouse_3_mutant/stats-volumes/mouse_3_mutant-log-determinant-scaled-fwhm0.2.
mnc, mutant
---***---/test_run_mutants_processed/mouse_4/stats-volumes/mouse_4-log-determinant-scaled-fwhm0.2.mnc, wt
---***---/test_run_mutants_processed/mouse_4_mutant/stats-volumes/mouse_4_mutant-log-determinant-scaled-fwhm0.2.
mnc, mutant
---***---/test_run_mutants_processed/mouse_5/stats-volumes/mouse_5-log-determinant-scaled-fwhm0.2.mnc, wt
---***---/test_run_mutants_processed/mouse_5_mutant/stats-volumes/mouse_5_mutant-log-determinant-scaled-fwhm0.2.
mnc, mutant
---***---/test_run_mutants_processed/mouse_6/stats-volumes/mouse_6-log-determinant-scaled-fwhm0.2.mnc, wt
---***---/test_run_mutants_processed/mouse_6_mutant/stats-volumes/mouse_6_mutant-log-determinant-scaled-fwhm0.2.
mnc, mutant

# the test in R:
> R
> library(RMINC)
> gf <- read.csv("jacobian_determinan_for_absolute_volume.csv")

# in the tarball with test files there was a file called rough_striatum_segmentation.mnc and a
mapping_for_striatum.csv file that we will now use to calculate the volume of the striatum for all the input
files:

> volume_striatum <- anatGetAll(gf$absolute_volume, atlas="rough_striatum_segmentation.mnc", defs="
mapping_for_striatum.csv")

# the exact numbers in your case might differ slightly from what is stated below, but they should be mostly the
same:
> volume_striatum
 striatum
 [1,] 7.619784
 [2,] 5.997711
 [3,] 7.682343
 [4,] 5.832673
 [5,] 7.838357
 [6,] 5.553919
 [7,] 5.811785
 [8,] 4.602663
 [9,] 6.947825
[10,] 5.175882
[11,] 6.827077
[12,] 5.131377
attr(,"atlas")
[1] "rough_striatum_segmentation.mnc"
attr(,"class")
[1] "anatUnilateral" "anatMatrix" "matrix"
attr(,"anatIDs")
[1] "1"

# you can plot the difference:
> plot(volume_striatum[,1] ~ gf$genotype)
(see image below)
```




```
# the actual difference in my case:
> tapply(volume_striatum[,1], gf$genotype, mean)
 mutant      wt
5.382371 7.121195
> (5.382371 - 7.121195) / 7.121195 * 100
[1] -24.41759
# is about 25%. The result you get should be similar to this.
```

The easy way (before December 2011)

Get a single tarball here:

http://repo.phenogenomics.ca/repo/MICe-software/

Untar that tarball, then compile all the necessary bits as follows, first checking for all dependencies:

```
./waf --prefix=/dir/to/install/into configure --build_python --build_visualization
```

Address any errors pointed out herein. If you get the following ppm related error output:

```
Checking for program gcc or cc           : /usr/bin/gcc
Checking for program cpp                 : /usr/bin/cpp
Checking for program ar                  : /usr/bin/ar
Checking for program ranlib              : /usr/bin/ranlib
Checking for gcc                         : ok
Checking for program g++ or c++          : /usr/bin/g++
Checking for g++                         : ok
Checking for program perl                : /usr/bin/perl
Checking for perl                        : ok 5.8.8
Checking for perl module DBI             : ok
Checking for header ppm.h                : not found     <-----
Checking for library ppm                 : not found     <-----
```

the missing ppm library can be installed using:

```
sudo apt-get install libnetpbm9-dev
```

If you get the following GL/glut.h related error output:

```
Checking for program gcc or cc           : /usr/bin/gcc
Checking for program cpp                 : /usr/bin/cpp
Checking for program ar                  : /usr/bin/ar
Checking for program ranlib              : /usr/bin/ranlib
Checking for gcc                         : ok
Checking for program g++ or c++          : /usr/bin/g++
Checking for g++                         : ok
Checking for program perl                : /usr/bin/perl
Checking for perl                        : ok 5.8.8
Checking for perl module DBI             : ok
Checking for header ppm.h                : yes
Checking for library ppm                 : yes
Checking for header GL/glut.h            : not found     <-----
```

the missing files can be installed using:

```
sudo apt-get install glutg3-dev
```

A fortran related error:

```
Checking for program gcc or cc           : /usr/bin/gcc
Checking for program cpp                 : /usr/bin/cpp
Checking for program ar                  : /usr/bin/ar
Checking for program ranlib              : /usr/bin/ranlib
Checking for gcc                         : ok
Checking for program g++ or c++          : /usr/bin/g++
Checking for g++                         : ok
Checking for program perl                : /usr/bin/perl
Checking for perl                        : ok 5.8.8
Checking for perl module DBI             : ok
Checking for header ppm.h                : yes
Checking for library ppm                 : yes
Checking for header GL/glut.h            : yes
Checking for program f77                 : not found     <-----
Checking for program gfortran            : not found     <-----
Checking for program g77                 : not found     <-----
```

can be remedied by:

```
sudo apt-get install g77
```

Once it finishes successfully, compile the code:

```
./waf --prefix=/dir/to/install/into configure build --build_python --build_visualization
```

Note that you'll need to have python installed for this to work. It's also not been extensively tested yet, so please report any errors in the build process.

Some requirements: you need to have the build tools (C and C++ compiler, etc.) installed, as well as octave, the DBI perl module, and netpbm is highly recommended (the libnetpbm9-dev package on debian/ubuntu). For building python and its numeric libraries you'll also need a fortran compiler.

Once the software is compiled, source the installed environment file to use it:

```
. /dir/to/install/into/environment
```

You might need to edit that environment file to fit your local needs; in particular, if you have the Sun Grid Engine installed (which is highly recommended) in a non-default location, you'll need to add it onto the PATH.

The hard way: source code available from the Brain Imaging Centre at McGill

Many of the source files needed to install the registration platform can be found on the software page of McGill university. There you can find source code for different types of operating systems. The following tutorial was created by installing the software on Ubuntu Hardy.

Main software page:
http://packages.bic.mni.mcgill.ca/

Source code for linux systems:
http://packages.bic.mni.mcgill.ca/tgz/

The list (and order) of packages that are required from that website:

1. netcdf
2. hdf5
3. minc-2
4. mni_autoreg
5. xfmavg (comes from http://packages.bic.mni.mcgill.ca/scripts/)
6. ebtks
7. bicpl
8. conglomerate
9. classify
10. inormalize
11. Register
12. mincblob
13. mincmorph
14. mni_perllib
15. N3
16. ray_trace
17. Display

That website will list multiple versions for each package - grab the highest version number unless indicated otherwise.

General dependencies

There are a number of libraries that the packages above depend on. This list is different for different platforms, and keeps changing, but here is a list that was used for Ubuntu Karmic:

```
netcdfg-dev libhdf5-serial-dev libnetpbm9-dev fftw-dev libgsl0-dev
libgetopt-tabular-perl libmni-perllib-perl libxext-dev glutg3-dev
libsoqt3-dev libxmu-dev libxi-dev imagemagick libtext-format-perl
libpcre++0 libpcre3 libsimage-dev libpcre++-dev
```

General way of installing a package

Given a compressed package: package.tar.gz, located in /usr/local/minc2/src

Open a terminal and cd (change directory) to the parent directory where the files are:

```
> cd /usr/local/minc2/src
```

Decompress the files:

```
> tar -xzvf package.tar.gz
(Messages will appear indicating which files are being decompressed)
```

cd to the directory containing the decompressed files:

```
> cd package
```

Configure the installation procedure. The --prefix= flag will indicate where the package will be installed.

```
> ./configure --prefix=/install/package/here
(Running `configure' takes a while. While running, it prints some messages telling which features it is
checking for.)
```

Compile the package:

```
> make
(Compilation messages will appear)
```

Type `make install' to install the programs and any data files and documentation.

```
> make install
(Installation messages will appear)
```

These commands may need to be preceded by *sudo* to give you the needed permissions to install the packages. For example:

```
> sudo make
(and)
> sudo make install
```

You can remove the program binaries and object files from the source code directory by typing `make clean'. To also remove the files that `configure' created (so you can compile the package for a different kind of computer), type `make distclean'.

MINC2 specific installation procedures

ℹ The examples below will assume that MINC2 will be installed under /usr/local/minc2 and that the packages reside under /usr/local/minc2/src

Prerequisitions

MINC2 relies on the netcdf and hdf5 libraries for its data storage. These two libraries should be present on the system before MINC2 is installed, or the process will fail.

netcdf

MINC2 is based on C-code, and thus a C compiler (e.g., gcc) is needed in order to compile the code. Typically gcc is available from your OS vendor (using apt-get or aptitude/synaptic). Make sure you choose the "-dev" or "-devel" binary package to ensure you have the developer files associated with the library. Since MINC2 is based on C-code, no interface for C++ or FORTRAN is needed. This can be made explict during the installation process by setting the environment variable for those compilers to "":

```
> cd /usr/local/minc2/src/netcdf-3.5.0/src
> export CC=gcc
> export CXX=""
> export FC=""
> export F90=""
> ./configure --prefix=/usr/local/minc2
> make
> make install
```

The commands that start with *export* work for a bash shell. If this command does not work, and you are running a csh/tcsh shell, the environment variables can be set using the command *setenv*:

```
> setenv CC gcc
(and for example)
> setenv FC ""
```

hdf5

The general installation procedure can be followed for hdf5, specifying only the --prefix flag for configure

On to the MINC2 libraries and software

The MINC2 library

For all versions of MINC2 upto 0.14, the configure option --enable-minc2 is needed. Any version 0.15 or later does not need this option anymore. The example below shows the commands for MINC2 version 0.15.

The --with-build-path flag for configure indicates where to look for libraries and include files which are needed for the installation of MINC2 (netcdf and hdf5).

```
> ./autogen.sh
> ./configure --prefix=/usr/local/minc2 --with-build-path=/usr/local/minc2
> make
> make install
```

General way of installing a MINC2 package

```
> ./configure --prefix=/usr/local/minc2 --with-build-path=/usr/local/minc2 --with-minc2
> make
> make install
```

Exceptions

As usual, there are some exceptions to the rule. The packages bicpl and ebtks have slightly different configure options and the package mni_perllib has a different installation procedure all together.

xfmavg

This is a script that is not contained in any of the packages, but can be found here: http://packages.bic.mni.mcgill.ca/scripts/. Right click on xfmavg, and save the link as /usr/local/minc2/bin/xfmavg, in order for the program to reside in the bin directory where all the other executable will be. Then changes the mode of the file to make it executable.

```
> chmod a+x /usr/local/minc2/bin/xfmavg
```

Furthermore, the xfmavg script uses the GNU Octave library. This can be downloaded from their website http://www.gnu.org/software/octave/.

Register and/or Display

The following libraries must be installed before building Register and/or Display. Typically opengl (libGL), and glut (libglut) are available from your OS vendor (using apt-get or aptitude/synaptic). If possible, install the vendor version; otherwise see the websites listed below. Make sure you choose the "-dev" or "-devel" binary package to ensure you have the developer files associated with the library.

- OpenGL or Mesa
  - OpenGL may be included with your OS, or see http://www.opengl.org/
  - Mesa is a replacement for OpenGL, often included in linux distributions if not, see http://www.mesa3d.org/ (there is also a copy of Mesa on our ftp site ftp://ftp.bic.mni.mcgill.ca/pub/register+Display/MesaLib-3.4.tar.gz)
- GLUT
  - a layer on top of OpenGL, may be included in your OS distribution
  - if not, see http://reality.sgi.com/mjk/glut3/glut3.html

In addition make sure you have the header files for the X server (library libX11). This library is also typically available from your OS vendor (using apt-get or aptitude/synaptic). Make sure you choose the "-dev" or "-devel" binary package to ensure you have the developer files associated with the library.

⚠️ It might be that compiling these tools using the `make' command fails. If that happens follow the ./configure part by:

```
make LIBS="-lglut"
(or depending on your settings:)
sudo make LIBS="-lglut"
```

bicpl

In addition to the general configure flags, --with-image-ppm needs to be specified, i.e.,

```
> ./configure --prefix=/usr/local/minc2 --with-build-path=/usr/local/minc2 --with-minc2 --with-image-ppm
```

ebtks

The --with-minc2 configure flag does not need to be specified, i.e.,

```
> ./configure --prefix=/usr/local/minc2 --with-build-path=/usr/local/minc2
```

mni_perllib

```
> perl Makefile.PL PREFIX=/usr/local/minc2/lib INSTALLDIRS=perl
> make
> make test
> make install
```

For a more specific install (example):

```
> perl Makefile.PL PREFIX=/projects/mice/share/arch/linux64/quarantine_shoddy_bastards/
INSTALLSITELIB=/projects/mice/share/arch/linux64/quarantine_shoddy_bastards/share/perl/5.8.8
> make install
```

Post installation

This example tutorial assumes that MINC2 will be installed under /usr/local/minc2. During installation the folder /usr/local/minc2/bin is created and that folder will contain all the executables. The executables that are "known" (the programs you can run) by a shell are the ones it can find in the directories listed in the environment variable PATH. So what is left to do after installing the tools, is update that environment variable in order to find them.

To do this for all users in Ubuntu, update the PATH in the file /etc/environment:

⚠️ example

```
(Use a text editor to open file /etc/environment, it might say something like this:)

LANGUAGE="en_CA:en"

LANG=en_CA.UTF-8
PATH="/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/bin/X11:/usr/games"

(Now, extend the PATH with a colon and the path to the new bin directory:)

LANGUAGE="en_CA:en"

LANG=en_CA.UTF-8
PATH="/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/bin/X11:/usr/games:/usr/local/minc2/bin"
```

For all users in Debian Sarge, update the PATH environment variable in /etc/profile and in /etc/bash.bashrc