MMDPS文档

# 概述

这是MMDPS的文档，MMDPS代表Multimodal MRI Data Processing System，多模态磁共振影像数据处理系统。

本文档分成几个部分。文件结构、程序结构、环境配置。

# 数据处理步骤

## 数据导入 ZXJ-PC

1. 光盘DICOM/目录下的PA0文件夹拷贝到电脑F:/MMDPDatabase/data\_dicom/<name>\_<time>/文件夹下，将同一批<name>\_<time>写入一个文本文件namelist.txt
2. 打开终端，运行python xxx/mmdps/tools/import\_changgung.py，指定namelist.txt
3. 运行F:/MMDPDatabase/nginx/nginx.exe
4. 查看本机IP地址并做记录

## 数据处理 工作站

1. 将ZXJ-PC的xxx/mmdps/tools/import\_changgeng/mmdpdb.db拷贝到本地xxx/mmdps/server/apiserver/下。根据ZXJ-PC的IP地址，修改rootconfig.py的server.storage地址，将namelist.txt拷贝到本地。
2. 运行xxx/mmdps/server/run\_apiserver.bat
3. 运行tools/start\_main.bat
4. 在main UI中选择BatchGet，指定namelist.txt，将BOLD拷贝到E:/DataProcessing/BOLDPreprocess/Data/FunRaw/中，T1与DWI拷贝到E:/DataProcessing/T1或DWI文件夹中
5. 在main UI中选择RunPara，选择xxx/mmdps/pipeline/T1/configpara\_atlased\_t1.json，注意更改该文件中的FolderList为namelist.txt路径。处理完毕后再执行configpara\_unatlased\_t1.json
6. 重启切换到Debian系统，启动main UI后，选择RunJob，执行xxx/mmdps/pipeline/DWI/configjob\_main\_dwi.json，需要修改configpara\_atlased\_dwi.json和configpara\_dwi.json里边的FolderList为namelist.txt路径。
7. 切换回windows，执行DPARSFA预处理，还需执行export\_pBOLD.py将预处理后的数据导出到E:/DataProcessing/BOLD/下边。
8. 在main UI中选择RunPara，执行xxx/mmdps/pipeline/BOLD/configpara\_bold.json，同样需要修改FolderList

# 文件结构

这是整个系统的文件夹的结构，简述了每个文件的作用。

atlas：脑图谱数据文件夹，有aal、bnatlas、brodmann\_lr、brodmann\_lrce。包括配置文件和多种分辨率的NIFTI文件。

data：其他数据文件，有bnv（BrainNetViewer）需要的默认配置文件，一个MNI模板NIFTI。

matlab：Matlab函数。这里目前要加入Matlab的路径里。startup.m会在启动时被执行。这实际上是不好的，还没想到更好的办法。

mmdps：主package。

common：空

rootconfig.py：非常重要，根配置。各种路径、地址、工具。

dms：Data Management System需要的各种东西。

apicore.py：基于requests，方便对Service API的访问。

batchget.py：批量下载MRI数据。

converter.py：数据导入前的格式转换，DICOM转成NIFTI，已经转换成NIFTI文件到底是哪个模态的匹配。

dbgen.py：根据MRI信息表格，评分表格和分组表格，建立一个数据库，目前是SQLite数据库。

exporter.py：信息导出，现在是根据文件的有无导出一个MRI各模态有无的表格。

importer.py：把MRI数据导入存储、并且更新数据库（目前是覆盖）。这是在converter之后执行的。

mridata.py：相当于Static Data Server的接口，Service API会使用这里面的接口。

storage.py：存储接口，封装了对Static Data Server的访问，便于使用。

tables.py：基于SQLAlchemy定义的实体关系，可以直接用于数据库的初始化、查询、插入等操作。

gui：Graphical User Interface，图形接口。

configfield.py：在Config和Field之间的相互转换。

field.py：Field定义。Field是一棵树，这棵树里存着配置值，可以直接转成对应的GUI，对此GUI里的值的修改又会同步回Field里的配置值。

guiframe.py：tkinter的MainWindow，可以根据需要自动加入滚动条。也可以不要滚动条。使用时，里面的根控件是mainframe。

tktools.py：tkinter工具。把tkinter常用的东西封装了下。创建控件时，会返回控件、绑定的tkvar，整体frame。如果没有，对应的项就是None。

proc：Processing，处理计算框架。

atlas.py：脑图谱工具。最好是使用里面的几个get\*函数来创建，会自动找到atlas数据文件夹里。

dataexport.py：特征导出工具。可以按照每个mriscan导出，也可以按照每个mriscan的每个模板导出。导出后作为特征库被Fusion和Feature服务器使用。

fusion.py：数据融合框架。配置好特征库和评分表，可以直接访问所有数据。

job.py：计算框架里的Job，还包括了BatchJob。

jobconfigfield.py：针对Job的Config和Field的转换。

loader.py：Fusion使用的各种数据的加载器。

netattr.py：非常重要，有Net和Attr，自带了atlas，并且可以建立子网络、子属性。

para.py：计算框架里的Parallel，并行计算的高级接口。

parabase.py：一堆关于并行计算的原始接口，使用很方便。

paraconfigfield.py：针对Para的Config和Field的转换。

util：Utilities，实用工具。

clock.py：时间，包含各种时间格式的转换，当前时间获取。

dataop.py：data operation，矩阵数据的操作。

dwi.py：针对DWI数据的处理的工具函数。

fileop.py：对文件的处理，编辑、压缩解压缩等。

loadsave.py：很重要，对各种格式的文件的加载和保存。

mattool.py：matrix的计算，统计工具。

path.py：路径操作工具。包含了环境变量中设置的搜索路径的使用方式。

run.py：相当于subprocess.run的几个工具函数。

toolman.py：工具管理。

vis：可视化工具。

attrprocs.py：对Attr的默认处理。

bnv.py：BrainNetViewer图的绘制。

heatmap.py：连接热图绘制。

line.py：折线图绘制。

matprocs：对numpy矩阵的处理。

netprocs：对Net的默认处理。

report.py：对Attr和Net的默认报告。

pipeline：对每种模态的处理。

BOLD：对BOLD的处理。输入数据是预处理以后的。

DWI：对DWI的处理，目前只能在装了FSL的Linux上做。

T1：对T1的处理。

Export：导出计算得到的特征到特征库。

Fusion：数据融合，这里对所有特征画了图。

server：服务器。

staticserver：MRI原始数据的静态数据服务器，ngnix。

apiserver：Service API服务器，包含了对数据库中的所有信息的访问。下载MRI数据，下载Feature。

webappserver：Web App服务器，就是网站。

tests：一堆测试用的脚本，可以不看。

tools：工具，这个没有在mmdps package里。每个工具都是独立的。

check\_fullfile.py：看能不能在当前路径下找到参数给出的文件。

gz\_unzip.py：解压.gz文件。

gz\_zip.py：压缩文件为.gz格式。

hdrimg\_to\_nii.py：把hdr/img格式转成nii格式。

nii\_to\_hdrimg.py：把nii格式转成hdr/img格式。

import\_changgung.py：非常重要，长庚数据的导入。

mmdpdb.db：生成的SQLite数据库文件。

mmdpsvarsall.bat：Windows下环境变量设置，执行后要在同一个cmd里跑别的命令。

mmdpsvarsall.rc：Linux下的环境变量设置，source mmdpsvarsall.rc。

mrfseg.exe：MRF分割程序。

runatlased.py：跑多个图谱的脚本。比如basicreport只针对一个图谱，使用这个可以跑多个图谱。基于环境变量MMDPS\_CURATLAS，在被运行的脚本中使用atlas.getbyenv。

runjob.py：非常重要，跑一个Job。

runpara.py：非常重要，跑一个Parallel。

ui\_main.py：GUI，主工具界面。使用start\_main打开。

start\_main.bat：非常重要，主界面，并且配置好了环境变量。Windows下。

tools\_config.json：toolman的配置，就是一堆可视化工具。

ui\_batchget.py：GUI，批量下载MRI数据。

ui\_configjob.py：GUI，配置BatchJob。

ui\_configpara.py：GUI，配置Parallel。

ui\_dmsapp.py：GUI，使用Service API查询和下载数据库中的信息。

ui\_import.py：GUI，原始数据转换和导入。

ui\_runjob.py：GUI，跑一个Job。

ui\_runpara.py：GUI，跑一个Parallel。

# 程序结构

这是对使用系统程序组织结构的基本说明。

## 数据管理

数据导入。几个服务器。数据下载工具。

## 计算流程

核心是计算脚本。先不考虑并行，相当于在算法的开发阶段，只看一个人的一个模态。比如T1。那么计算的脚本要写在这个扫描的文件夹中，像这样：

xiaoming\_20101118

T1.nii.gz

calculate.py

这里的calculate.py，应该以模态原始数据（这里是T1.nii.gz）为输入，经过计算，生成一些处理后的结果。比如，进行灰质分割，得到grey.nii.gz。则直接执行calculate.py，也就是当前路径为工作路径（working directory），得到这样的结果：

xiaoming\_20101118

T1.nii.gz

grey.nii.gz

calculate.py

不使用图谱的计算应该这样。其余的东西，也就是对多个扫描的处理可以用系统提供的工具。

再看这个模态的针对某个图谱的计算。比如计算灰质密度，那么应该把计算脚本放在图谱名的文件夹中，像这样：

xiaoming\_20101118

T1.nii.gz

grey.nii.gz

calculate.py

aal

calcgmd.py

直接在aal路径下执行calcgmd.py，也就是以calcgmd.py所在的路径为工作路径，应该生成针对aal图谱的结果，放在同样目录的gmd.csv中，像这样：

xiaoming\_20101118

T1.nii.gz

grey.nii.gz

calculate.py

aal

calcgmd.py

gmd.csv

对多个人多个图谱的处理，就可以用系统提供的工具直接计算了。计算输入的数据需要用相对路径，比如这里需要用到grey.nii.gz。那么就这样用：greyniipath = ‘../grey.nii.gz’。这里有个要求，就是calcgmd.py需要对所有图谱通用。应该直接使用路径名来得到当前的图谱，然后利用提供的图谱工具进行实际的计算。也就是说，整个calcgmd.py里不应该出现“aal”、“brodmann\_lr”等字样，而一律用atlasobj代替。这个atlasobj可以使用atlas.getbywd()来获取，利用的就是当前工作目录的信息。比如在aal文件夹下，返回的是aal的图谱对象，在brodmann\_lr文件夹下，返回的是brodmann\_lr图谱对象。

调好了无图谱的脚本和有图谱的脚本，下一步是创建Job配置json文件。每个脚本算一个Job，这里对calculate.py和calcgmd.py都是PythonJob。配置好后，应该用runjob.py测试一下。使用--config configjob\_\*.json参数设置Job的配置，使用--folder path\_to/xiaoming\_20101118或--folder path\_to/xiaoming\_20101118/aal分别跑两个Job。应该做到不出问题。实际上，json文件里的Job更可能是个BatchJob，里面包含了一串线性执行的Job。注意，带图谱和不带图谱的Job不能混着。必须只含有一种。甚至可能是BatchJob里还有BatchJob。建议对带图谱和不带图谱的Job配置文件命名成configjob\_atlased\_\*.json和configjob\_unatlased\_\*.json。

然后考虑同时计算多次扫描。这里也分成两个情况，对每个扫描并行和对每个扫描的每个图谱并行。这两种都用到了Parallel。运行的工具是tools里的runpara.py，也需要一个configpara\_\*.json配置文件。

对每个扫描并行，也就是无图谱的，Parallel的配置大概这样：

configpara\_unatlased\_t1.json

{

"name": "T1Proc",

"MainFolder": "F:/BiShe/example/MRIScanProc/T1",

"FolderList": "F:/BiShe/example/MRIScanProc/scanlist.txt",

"Second": false,

"SecondList": "",

"JobConfig": "F:/BiShe/mmdps/pipeline/T1/configjob\_unatlased\_t1.json",

"RunMode": "Parallel"

}

可以看到，Second是false，就是只有一层需要并行的东西。JobConfig是Job的配置文件。执行runpara.py --config configpara\_unatlased\_t1.json，Parallel会在FolderList里的每个文件夹里并行执行runjob.py --config configjob\_unatlased\_t1.json。

对每个扫描的每个图谱并行，Parallel的配置大概这样：

configpara\_atlased\_t1.json

{

"name": "T1Proc",

"MainFolder": "F:/BiShe/example/MRIScanProc/T1",

"FolderList": "F:/BiShe/example/MRIScanProc/scanlist.txt",

"Second": true,

"SecondList": "F:/BiShe/example/MRIScanProc/atlaslist.txt",

"JobConfig": "F:/BiShe/mmdps/pipeline/T1/configjob\_atlased\_calcgmd.json",

"RunMode": "Parallel"

}

可以看到，Second是true，而且有个SecondList选项。比如scanlist.txt的内容是

xiaoming\_20101118

xiaoming\_20101220

atlaslist.txt的内容是

aal

brodmann\_lr

那么，跑这个Parallel的时候，也就是执行runpara.py --config configpara\_atlased\_t1.json时，Parallel会在

xiaoming\_20101118/aal

xiaoming\_20101118/brodmann\_lr

xiaoming\_20101220/aal

xiaoming\_20101220/brodmann\_lr

这4个文件夹下分别运行runjob.py --config configjob\_atlased\_calcgmd.json。当然也是并行运行的。假如FolderList里有m项，SecondList里有n项，那么一共会并行运行m\*n个Job。

两种情况都可以选择串行执行应该执行的所有Job。

而runpara.py又可以作为一个简单的PythonJob，这就构成了一个树。比如一个BatchJob，里面有两个runpara，第一个是unatlased的预处理，第二个是atlased的对每个图谱的属性计算，就可以只用runjob来跑这一整个的流程了。

Job的配置文件起名为configjob\_\*.json，Parallel的配置文件起名为configpara\_\*.json，这样单击Edit时会使用内置的GUI编辑器，否则会使用配置的texteditor。

所有的Job都会有log，这非常重要，排错要用到里面的信息。每个Job都会在当前的工作目录下建立一个log文件夹，里面是按照时间排序的log文件，文件名最后是Job的name属性。

## 数据融合

首先介绍封装提取的特征的类，Net和Attr。

Net是网络特征，包含了实际的numpy数组和对应的atlasobj。

Attr是属性特征，包含了实际的numpy数组和对应的atlasobj。

这样就实现了自动的对应，因为Net和Attr总是要对应一个图谱的。画图时也需要输入Net或Attr的对象，而不是原始数据。虽然大多数情况下会浪费空间，不过从原理上来说就是一个指针。

Fusion里有各种Loader，对应的源代码是fusion.py和loader.py。配置Fusion使用了4个json文件，看fusion\_config文件夹里的内容，分别是group、mainconf、netattr、score。

可以使用NetLoader加载网络，使用AttrLoader加载属性。这里，对不同的特征，是使用特征名来选择的。就是说，fusion对象里只有一个NetLoader，nets；一个AttrLoader，attrs。加入要load功能网络，则fu.nets.load(‘xiaoming\_20101118’, ‘bold\_net’)。

对Group和Score的加载器都是多个放在dict里的。比如要拿到一个代表group的对象，则应该fu.groups[‘healthy’].mriscans。fu.scores[‘strokescore’].loadvstack(mriscans)。

提供了loadvstack函数，这里是加载若干个数据，拉成长条，vstack成一个二维矩阵。这样便于向量计算，不需要一个一个的加载了。

如果数据融合内部也需要并行，可以把要干的活封装到函数对象里或者有.run函数的类里，弄成一大串，再使用parabase里的工具直接运行。

建议对一个Fusion，仅对一个图谱操作。使用atlas.getbyenv(‘aal’)来获取图谱对象。这里的aal是指，如果没有找到MMDPS\_CURATLAS环境变量，就使用这个。写好使用tools里的runatlased.py就可以同时跑多个图谱的Fusion了，默认是串行，加入--parallel可以实现并行。

## 模态处理

对不同模态的处理在pipeline文件夹下。

### T1

输入是原始T1，T1.nii.gz。

对unatlased：betnorm，一个大的matlab脚本，配准和制作只有灰质、白质、脑脊液的像。

对atlased：calcgmd，计算灰质密度。

### DWI

使用了FSL，所以只能在Linux下使用。

看configjob\_main\_dwi.json，分别运行了两个Parallel：configpara\_dwi.json和configpara\_atlased\_dwi.json，具体是configjob\_dwi.json和configjob\_atlased\_dwi\_gen\_net.json。

configjob\_dwi.json里面有两个Job，分别是基本的预处理、dtifit、纤维素追踪。

configjob\_atlased\_dwi\_gen\_net.json里分别是生成个体空间和标准空间的脑图谱、计算结构网络、计算每个脑区的FA和MD属性值。

注意，各种sh脚本里是如何得到当前的图谱名和需要的数据的路径的，需要Linux shell编程知识。

### BOLD

输入是预处理之后的BOLD像，pBOLD。

没有unatlased的处理，只有atlased，分别是gen\_net和calc\_attr。calc\_attr是用MatlabJob算的。内置的没有计算intra。

# 环境配置

环境的配置是必须的，因为系统要调用各种外部工具、取得各种数据、取得各种计算脚本和配置参数的文件内容等等。有几个要配置的地方。

rootconfig.py。主要的需要配置的部分如下：

matlab\_bin：matlab.exe所在的路径。

path.texteditor：一个文本编辑器的路径，推荐Visual Studio Code。

path.niiviewer：一个NIFTI查看程序的路径，推荐mricron。

path.dcm2nii：一个DICOM转NIFTI的程序，对不同的程序需要写不同的具体接口，因为获得具体的模态的图像是要通过文件名来看的。这里推荐dcm2niix，可以直接搜索下载mricrongl，里面有dcm2niix。

server.api：Service API所在的http地址。

server.storage：MRI数据HTTP服务器所在地址。

server.featurestorage：特征数据库HTTP服务器所在地址。

dms.folder\_working：好像没用到。

dms.folder\_dicom：原始DICOM所在的目录，数据导入需要这个。

dms.folder\_rawnii：DICOM到NIFTI转换的目录，也可以直接放入NIFTI文件。

dms.folder\_mridata：Storage里实际数据的目录。是重命名以后的。

mmdpsvarsall.bat。配置环境变量。Windows。使用系统前要在cmd里先执行这个。

MMDPS\_ROOTDIR：mmdps所在的目录，是tools、atlas等文件夹所在的目录。

MMDPS\_BUILTINPATH：内置目录，搜索配置文件或脚本文件时会在这些路径里找。

mmdpsvarsall.rc。同上，不过是Linux下需要的。使用系统前要在term里先source mmdpsvarsall.rc。

Matlab路径配置。

把系统的matlab文件夹加到Matlab的Path里。这里面有个startup.m会在Matlab启动时执行，这样不太好虽然。