MIRACL Documentation

Release 2.2.5

Maged Goubran

Apr 19, 2023

CONTENTS:

ABOUT

Find important information about MIRACL here. Use the `Next` button at the bottom of the page or jump to a topic directly by chosing it from the sidebar menu or TOC.

1.1 MIRACL in a nutshell

MIRACL (**M**ulti-modal **I**mage **R**egistration **A**nd **C**onnectivity anaLysis) is a general-purpose, open-source pipeline for automated:

1. Registration of mice CLARITY data to the Allen Reference Atlas

2. Segmentation and feature extraction of mice CLARITY data in 3D (Sparse and nuclear staining)

3. Registration of mice multimodal imaging data (MRI & CT, in-vivo & ex-vivo) to Allen Reference Atlas

4. Tract or label specific connectivity analysis based on the Allen Connectivity Atlas

5. Comparison of diffusion tensort imaging (DTI)/tractography, virus tracing using CLARITY and Allen Connectivity Atlas

6. Statistical analysis of CLARITY and imaging data

7. Atlas generation and Label manipulation

1.1.1 Program structure

MIRACL is structured into *Modules* and *Workflows*.

Modules

The pipeline is comprised of different `Modules` depending on their respective functionality. Functions for each module are grouped together:

Module	Functionality
connect	Connectivity
conv	Conversion (Input/Output)
reg	Registration
seg	Segmentation
lbls	Labels
utilfn	Utilities
sta	Structure Tensor Analysis
stats	Statistics

An example of using a module would be to run the `clar_allen` function which performs a CLARITY whole-brain registration to Allen Atlas on a `nifti` image (down-sampled by a factor of five):

```
$ miracl reg clar_allen -i niftis/SHIELD_05x_down_autoflor_chan.nii.gz -o ARI -m␣
→combined -b 1
```

The above command uses the `-i` flag to select the nifti file, `-o` to specify the orientation of the image, `-m` to register to both hemispheres and `-b` to include the olfactory bulb.

Workflows

The workflow (*flow*) module combines multiple functions from the above modules for ease of use to perform a desired task.

For example, a standard reg/seg analysis could look like this:

First perform registration of whole-brain CLARITY data to ARA:

```
$ miracl flow reg_clar -h
```

Then perform segmentation and feature extraction of full resolution CLARITY data:

```
$ miracl flow seg -h
```

Or structure tensor analysis:

```
$ miracl flow sta -h
```

1.2 Licence

MIRACL is licensed under the terms of the GNU General Public License v3.0.

MIRACL is distributed in the hope that it will be useful, but **WITHOUT ANY WARRANTY**; without even the implied warranty of **MERCHANTABILITY** or **FITNESS FOR A PARTICULAR PURPOSE**. You should have received a copy of GNU General Public License v3.0 along with HippMapp3r.

The code is released for academic research use only. For commercial use, please contact .

1.3 Citing MIRACL

Citation guidelines for using MIRACL in your work.

1.3.1 MIRACL publication

If you use **MIRACL** in your work please cite our paper:

APA

Goubran, M., Leuze, C., Hsueh, B., Aswendt, M., Ye, L., Tian, Q., Cheng, M.Y., Crow, A., Steinberg, G.K., McNab, J.A., Deisseroth, K., and Zeineh, M. (2019). Multimodal image registration and connectivity analysis for integration of connectomic data from microscopy to MRI. Nature communications, 10(1), 5504.

BibTeX

```
@article{goubran2019multimodal,
  title={Multimodal image registration and connectivity analysis for integration of
→connectomic data from microscopy to MRI},
  author={Goubran, Maged and Leuze, Christoph and Hsueh, Brian and Aswendt, Markus and
→Ye, Li and Tian, Qiyuan and Cheng, Michelle Y and Crow, Ailey and Steinberg, Gary K
→and McNab, Jennifer A and Deisseroth, Karl and Zeineh, Michael},
  journal={Nature communications},
  volume={10},
  number={1},
  pages={5504},
  year={2019},
  publisher={Nature Publishing Group UK London}
}
```

1.3.2 Tools used by MIRACL

Some of our functions build on or use these tools (please cite their work if you are using them):

Tool	Function
Allen Regional Atlas	Atlas registration
Allen Connectivity Atlas	Connectivity analyses
ANTs	Registration
Fiji	Segmentation
c3d	Image processing
FSL	Diffusion MRI processing
MRtrix3	Tractography
TrackVis/DTK	Tractography

1.4 MIRACL in research

A selection of publications that used or referenced **MIRACL**:

Hsueh, B., Chen, R., Jo, Y., Tang, D., Raffiee, M., Kim, Y. S., . . . Goubran, M., & Deisseroth, K. (2023). Cardiogenic control of affective behavioural state. Nature, 1-8.

Qu, L., Li, Y., Xie, P., Liu, L., Wang, Y., Wu, J., . . . & Peng, H. (2022). Cross-modal coherent registration of whole mouse brains. Nature Methods, 19(1), 111-118.

Georgiadis, M., Schroeter, A., Gao, Z., Guizar-Sicairos, M., Liebi, M., Leuze, C., . . . & Rudin, M. (2021). Nanostructure-specific X-ray tomography reveals myelin levels, integrity and axon orientations in mouse and human nervous tissue. Nature communications, 12(1), 2941.

Wang, X., Zeng, W., Yang, X., Zhang, Y., Fang, C., Zeng, S., . . . & Fei, P. (2021). Bi-channel image registration and deep-learning segmentation (BIRDS) for efficient, versatile 3D mapping of mouse brain. Elife, 10, e63455.

Boillat, M., Hammoudi, P. M., Dogga, S. K., Pages, S., Goubran, M., Rodriguez, I., & Soldati-Favre, D. (2020). Neuroinflammation-associated aspecific manipulation of mouse predator fear by Toxoplasma gondii. Cell reports, 30(2), 320-334.

Pallast, N., Wieters, F., Nill, M., Fink, G. R., & Aswendt, M. (2020). Graph theoretical quantification of white matter reorganization after cortical stroke in mice. NeuroImage, 217, 116873.

Ito, M., Aswendt, M., Lee, A. G., Ishizaka, S., Cao, Z., Wang, E. H., . . . & Steinberg, G. K. (2018). RNA-sequencing analysis revealed a distinct motor cortex transcriptome in spontaneously recovered mice after stroke. Stroke, 49(9), 2191-2199.

1.5 Acknowledgements

Huge thank you to:

- Vanessa Sochat (@vsoch) for creating the Docker & Singularity containers for the pipeline

- Newton Cho, Jordan Squair and Stéphane Pagès for helping optimize the segmentation workflows & troubleshooting

- The members of AICONS Lab for their feedback and contributions

GitHub contributers (ordered by number of commits):

- vsoch

-

ishita1988

-

chrisroat

PR's:

- dariocalvarez

1.6 AICONS Lab

The **A**rtificial **I**ntelligence and **CO**mputational **N**euro**S**ciences (AICONS) Lab is located at the Sunnybrook Research Institute of the University of Toronto and is part of the Black Centre for Brain Resilience and Recovery, Harquail Centre for Neuromodulation, and Temerty Centre for AI Research and Education in Medicine.

Our work combines technical and translational research, focusing on the development of novel AI, computational and imaging tools to probe, predict and understand neuronal and vascular circuit alterations, and model brain pathology in neurological disorders, including *Alzheimer's disease, stroke and traumatic brain injury*.

For more information visit our official webpage.

CHAPTER

TWO

INSTALLING AND RUNNING MIRACL

We provide instructions on how to install and run **MIRACL** using either of the following methods:

Important: **Docker** is our recommended method for running **MIRACL** on local machines and servers. We recommend **Singularity** to run **MIRACL** in a cluster environment (e.g. Compute Canada).

Attention: Support for installing **MIRACL** locally (i.e. on your host system directly without using **Docker** or **Singularity**) will be phased out in future versions of the software

Docker

We provide a build script to automatically create a **Docker** image for you that can be run using **Docker Compose**. This method does not require a manual installation of **MIRACL** and works on Linux, macOS and Windows (using WSL 2).

Tip: This is our recommended method for running **MIRACL** on local machines and servers

Docker is well suited if you want to run **MIRACL** on a local machine or local server. If you need to run **MIRACL** on a cluster, see our instructions for installing **Singularity**. If you don't have Docker installed on your computer, do that first. Make sure your installation includes **Docker Compose** as it is required to run the build script we provide. Note that **Docker Compose** is included as part of the **Docker Desktop** installation by default.

First, it is important to understand how the container is built. There is a base image in the docker folder that installs **Python** and dependencies. Then the `Dockerfile` in the base of the repository builds the `mgoubran/miracl` image from that base. When the build happens, it cats the `version.txt` file in the repository to save a versioned base, but then the build uses the tag revised-base-latest that is always the latest base. The base container is built from this folder and pushed manually, while the main container is built and pushed automatically via the **CircleCI** Recipe. Thus, if you want to update the base, you will need to see the `README.md` in that folder and push new images.

This will build a **Docker** image of **MIRACL** based on its latest version using our default naming scheme. For custom names and specific versions see below for our `Additional build options` section.

Clone the **MIRACL** repo to your machine:

```
$ git clone https://www.github.com/mgoubran/MIRACL
$ cd MIRACL
```

Build the latest **MIRACL** image using the build script we provide:

```
$ ./build.sh
```

> **Error:** Make sure that the script can be executed. If it can't and you are the owner of the file, use `chmod u+x build.sh` to make it executable. Prefix with `sudo` if you are not the owner of the file or change permissions for `g` and/or `o`.

Once the image has successfully been built, run the container using **Docker Compose**:

```
$ docker compose up -d
```

> **Note:** Note that the **Docker Compose** syntax is different if you installed it using the standalone method. **Compose standalone** uses the `-compose` syntax instead of the current standard syntax `compose`. The above command would thus be `docker-compose up -d` when using **Compose standalone**.

> **Note:** Note that the **Docker Compose** syntax is different if you installed it using the standalone method. **Compose standalone** uses the `-compose` syntax instead of the current standard syntax `compose`. The above command would thus be `docker-compose up -d` when using **Compose standalone**.

The container is now running and ready to be used.

Interactively shell inside:

```
$ docker exec -it miracl bash
```

Files that are saved while using **MIRACL** should be saved to volumes mounted into the container in order to make them persistent. To mount volumes, just add them to the `docker-compose.yml` in the base directory under volumes.

> **Danger:** Do not delete the volume that is already mounted which mounts your `.Xauthority`! This is important for X11 to work correctly.

Example:

```
volumes:
    - '/home/mgoubran/.Xauthority:/home/mgoubran/.Xauthority'
    - '/home/mgoubran/mydata:/home/mgoubran/mydata'
```

Exit your container and navigate to your **MIRACL** folder. Use **Docker Compose** to stop the container:

```
$ docker compose down
```

> **Note:** Note that the **Docker Compose** syntax is different if you installed it using the standalone method. **Compose standalone** uses the `-compose` syntax instead of the current standard syntax `compose`. The above command would thus be `docker-compose up -d` when using **Compose standalone**.

> **Note:** Note that the **Docker Compose** syntax is different if you installed it using the standalone method. **Compose standalone** uses the `-compose` syntax instead of the current standard syntax `compose`. The above command would

thus be `docker-compose up -d` when using **Compose standalone**.

Naming is done automatically when using our build script which includes a default naming scheme. By default, the image is named `mgoubran/miracl:latest` and the container is tagged with `miracl`.

You can easily change the defaults if your usecase requires it by running our build script with the following options:

```
$ ./build -i <image_name> -c <container_name>
```

Options:

```
-i, Specify image name (default: mgroubran/miracl)
-c, Specify container name (default: miracl)
```

Example:

```
$ ./build -i josmann/miracl -c miracl_dev_version
```

Tip: Use `./build -h` to show additional options

By default, **Docker** images will be built using the latest version of **MIRACL**. If you need to build a **Docker** image based on a specific version of **MIRACL**, do the following:

1. Clone the **MIRACL** repository and navigate to the **MIRACL** folder:

```
$ git clone https://www.github.com/mgoubran/MIRACL
$ cd MIRACL
```

2. Cloning the repository will download all tags/versions. List them with:

```
$ git tag -l
```

Example output:

```
v1.1.1
v2.2.1
v2.2.2
v2.2.3
v2.2.4
v2.2.5
```

3. Decide which tag/version of **MIRACL** you want to use and check it out as a new branch:

```
$ git checkout tags/<tag_name> -b <branch_name>
```

Example:

```
$ git checkout tags/v2.2.4 -b miracl_v2.2.4
```

4. If you are reverting to a version of MIRACL >= 2.2.4, you can build the image for your chosen version by running the build script with the `-t` flag:

```
$ ./build.sh -t
```

Note: If you want to build an image for a version of MIRACL <= 2.2.4 either follow the build instructions of the particular version or download the latest build script using e.g. `wget https://raw.githubusercontent.com/AICONSlab/MIRACL/master/build.sh` (overwrites current build script if present) and run it with the `-t` flag.

5. From here you can follow our instructions for building **MIRACL** from scratch starting with `docker compose up -d`. Our script will automatically detect the version of the branch you checked out and tag the image accordingly.

Singularity

Unlike **Docker**, **Singularity** is well suited to run in a cluster environment (like Sherlock at Stanford or Compute Canada). We provide the latest version of **MIRACL** as a **Singularity** container that can be conveniently pulled from cloud storage.

Tip: This is our recommended method for running **MIRACL** in a SLURM cluster environment such as Compute Canada or Sherlock @ Stanford

First, log in to the cluster:

```
$ ssh -Y <username>@<cluster>
```

`<cluster>` could be `sherlock.stanford.edu` or `cedar.computecanada.ca` for example

Once logged in, change the directory to your scratch space and pull (download) the **Singularity** container:

```
$ cd $SCRATCH
$ singularity pull miracl_latest.sif library://aiconslab/miracl/miracl:latest
```

Attention: `singularity pull` requires **Singularity** version `3.0.0` or higher. Please refer to our *Troubleshooting section* ("Can I build a Singularity container from the latest MIRACL image on Docker Hub") if you are using an older version of **Singularity**.

To shell into the container use:

```
$ singularity shell miracl_latest.sif bash
```

Use the `-B` flag to bind a data directory to the container:

```
$ singularity shell -B /data:/data miracl_latest.sif bash
```

See also:

For running functions on clusters please check our **Singularity** tutorials for Compute Canada and Sherlock

Local

Warning: Support for this installation method will be discontinued in future versions of **MIRACL**. We recommend to use **Docker** or **Singularity** instead.

Steps to setup/run **MIRACL** on a Linux/macOS machine:

```
$ git clone https://github.com/mgoubran/MIRACL.git miracl
```

Tip: Alternatively, you can download the zip file containg the repo and uncompress it

---

Next, change directories into the newly created `miracl` folder:

```
$ cd miracl
```

Create your virtual **MIRACL** environment and activate it:

Attention: To setup a virtual environment you need **Anaconda** for **Python 2.7**. It can be downloaded from their official website

---

```
$ conda create --name miracl python=3.7.4 pip
$ conda activate miracl
```

Install dependencies:

```
$ pip install -e .
```

Next, download the `depends` folder from our Dropbox link and place it either inside the `linux_depends` or `mac_depends` folder:

```
$ mv ~/Downloads/depends.zip miracl/.
$ cd miracl
$ unzip depends.zip
$ rm depends.zip
```

This folder contains compiled versions of **ANTS** and **c3d** for Linux or Mac OS. Before continuing, make sure to change the permissions.

This can be done by running:

```
$ chmod -R 755 <path/to/depends>/*
```

In order to run the pipeline, some symbolic links must be added to access certain commands. Inside the `miracl` folder, run:

```
$ sudo ln -s <path/to/depends>/ants/antsRegistrationMIRACL.sh /usr/bin/ants_miracl_clar &
↪& chmod +x /usr/bin/ants_miracl_clar
$ sudo ln -s <path/to/depends>/ants/antsRegistrationMIRACL_MRI.sh /usr/bin/ants_miracl_
↪mr && chmod +x /usr/bin/ants_miracl_mr
```

Make sure <path/to/depends> is replaced with the directory path that leads to the `depends` directory.

Place the `atlases` folder (which got downloaded together with the `depends` folder) inside the `miracl` folder:

```
$ mv ~/Downloads/atlases.zip miracl/.
$ cd miracl
$ unzip atlases.zip
$ rm atlases.zip
```

This folder contains the Allen Atlas data needed for registration and connectivity analysis.

First, download **Fiji/ImageJ** from their offical website.

Then do:

```
$ cd depends
$ wget https://downloads.imagej.net/fiji/latest/fiji-linux64.zip
$ unzip fiji-linux64.zip
$ rm fiji-linux64.zip
```

Next, install additional plugins by going to `Help -> Update` and clicking on the `Manage update sites` button.

Choose the following update sites:

- 3D ImageJ Suite: http://sites.imagej.net/Tboudier

- Biomedgroup: https://sites.imagej.net/Biomedgroup

- IJPB-plugins: http://sites.imagej.net/IJPB-plugins

Download FSL and install it:

```
$ wget https://fsl.fmrib.ox.ac.uk/fsldownloads/fslinstaller.py
$ sudo python fslinstaller.py
```

For the visualization of nifti files and labels we recommend ITKSNAP or the nifti plugin for **Fiji/ImageJ**.

If you have diffusion MRI data download and install MRtrix3:

```
$ sudo apt-get install git g++ python python-numpy libeigen3-dev zlib1g-dev libqt4-
→opengl-dev libgl1-mesa-dev libfftw3-dev libtiff5-dev
$ git clone https://github.com/MRtrix3/mrtrix3.git
$ cd mrtrix3
$ ./configure
$ ./build
$ ./set_path
```

To end a **MIRACL** session, deactivate your virtual environment:

```
$ conda deactivate
```

To update **MIRACL**, navigate into your **MIRACL** base folder (e.g. `$ cd miracl`) and run:

```
$ git pull
```

You should be good to go!

Windows

> **Warning:** Support for installing **MIRACL** locally in the WSL will be discontinued in future versions of **MIRACL**. We recommend to use **Docker** or **Singularity** instead.

To install **MIRACL** on your Windows system, Windows Subsystem for Linux (WSL) must be installed. WSL2 is preferred. From there, the usual steps to install **MIRACL** on a Linux based system will be used with a few tweaks.

> **Hint:** Follow the below steps if you want to install **MIRACL** in your WSL instance locally. If you prefer to use **Docker** to run **MIRACL** on Windows follow our installation instructions for **Docker** instead.

The Windows Subsystem for Linux (WSL) creates an environment that allows users to run versions of Linux without having to set up a virtual machine or a different computer.

To install WSL, users can follow the instructions from Microsoft. More comprehensive instructions can be found here. Upgrading from WSL 1 to WSL 2 is recommended, due to WSL 2's benefits.

Note: You may ignore this step if you have a preferred Linux distribution that is already installed in your WSL2

A Linux distribution (distro), like Ubuntu, is an operating system based on the Linux kernel.

Now that WSL (either 1 or 2) is installed, the **Ubuntu 22.04** distro can be installed. To install Ubuntu, open the Windows Store app, search for "**Ubuntu 22.04**", and select the Get button. You could also use this link.

The Ubuntu distro should have **Python 3** installed. To ensure that this is the case, update all packages installed in the WSL:

```
$ sudo apt update
$ sudo apt -y upgrade
```

We can see which version of **Python 3** is installed by typing:

```
$ python3 -V
```

The output in the terminal window will show the version number.

pip is required to install software packages in **Python**. It can be installed by running the following command:

```
$ sudo apt install -y python3-pip
```

You could also use **Anaconda** to install the packages but we found that installing and using pip was more straightforward.

To actually install **MIRACL**, follow the local installation instructions for Linux and macOS.

To use **MIRACL's** graphical user interface (GUI), **Xming** must be installed. **Xming** is a display server for Windows computers, that is available for use by anyone. It can be downloaded from SourceForge.

Before running **MIRACL's** GUI, run Xming. In the terminal window where **MIRACL's** GUI will be run, input the following command:

```
$ export DISPLAY=$DISPLAY:localhost:0
```

Now that everything is installed, **MIRACL** can be run via the WSL. To run:

1. Open WSL via terminal

2. Navigate to the folder where you would like to run MIRACL from

3. Activate the environment containing `miracl`:

```
$ source activate miracl
$ miraclGUI
```

TUTORIALS

The tutorial structure generally matches the module/function structure of **MIRACL**. Refer to the *Legend section* for syntax information.

Choose the tutorial for the function you are interested in from its respective module in the sidebar/TOC or go to our *Getting started section* for a tutorial on **MIRACL's** general usage.

Note that not all functions have tutorials yet. . . we are working on it!!!

3.1 Legend

In the docs/tutorials, code examples are written as follows:

```
$ miracl -h
```

Code blocks look like this:

```
usage: miracl [-h] {connect,conv,flow,lbls,reg,seg,sta,stats,utils} ...

positional arguments:
  {connect,conv,flow,lbls,reg,seg,sta,stats,utils}
    connect             connectivity functions
    conv                conversion functions
    flow                workflows to run
    lbls                label manipulation functions
    reg                 registration functions
    seg                 segmentation functions
    sta                 structure tensor analysis functions
    stats               statistical functions
    utils               utility functions

optional arguments:
  -h, --help            show this help message and exit
```

Inline code is marked as: `$ miracl -h`

Admonitions are displayed as colored text boxes. This is an example of what a 'tip' admonition would look like:

Tip: The above `-h` flag can be used with each of MIRACL's modules/functions

---

We use brackets to denote text as follows:

- {}: Used for variabels.

    - Example: niftis/downsample{factor}x.nii.gz

- <>: Used for placeholder text in examples that you need to replace with your own information.

    - Example: $ ssh <username>@cedar.computecanada.ca

- [ ]: Placeholders for flag arguments used in command-line scripting.

    - Example: $ miracl flow sta -f [Tiff folder] -o [output nifti]

- []: Denotes flags in the command-line help menus.

    - Example: $ miracl [-h]

Files and directories (or generally paths) are denoted like this: example_dir/example_file.nii.gz

Names of exectutable programs are marked as follows: **MIRACL**

Lastly, links are highlighted in blue (purple when clicked): link to MIRACL's README

3.2 Getting started

MIRACL can be run in the command-line or by using its GUI.

3.2.1 Command-line

Note: If you have installed **MIRACL** locally, run source activate miracl first to start its virtual environment.

To look at available modules, invoke the help menu:

```
$ miracl -h
```

The following menu should be printed to the terminal:

```
usage: miracl [-h] {connect,conv,flow,lbls,reg,seg,sta,utils} ...

positional arguments:
  {connect,conv,flow,lbls,reg,seg,sta,utils}
    connect             connect functions
    conv                conv functions
    flow                workflows to run
    lbls                Label manipulation functions
    reg                 registration functions
    seg                 segmentation functions
    sta                 STA functions
    sta                 STA functions
    utils               Utils functions


optional arguments:
  -h, --help            show this help message and exit
```

If you want information about a particular module you can call it with -h. Let's use the conv module as an example. Invoke its help menu using:

```
$ miracl conv -h
```

You should get:

```
usage: miracl conv [-h] {tiff_nii,nii_tiff,set_orient,gui_opts} ...

positional arguments:
  {tiff_nii,nii_tiff,set_orient,gui_opts}
    tiff_nii            convert Tiff stacks to Nii
    nii_tiff            convert Nii volume to Tiff stack
    set_orient          Set orientation tag with GUI
    gui_opts            GUI options


optional arguments:
  -h, --help            show this help message and exit
```

For accessing the help menu of a specific function in the `conv` module, say `tiff_nii`, type:

```
$ miracl conv tiff_nii -h
```

You should get:

```
usage:  Converts Tiff images to Nifti

A GUI will open to choose your:

    - < Input CLARITY TIFF dir >


----------

For command-line / scripting

Usage: miracl conv tiff_nii -f [Tiff folder]

Example: miracl conv tiff_nii -f my_tifs -o stroke2 -cn 1 -cp C00 -ch Thy1YFP -vx 2.5 -
↪vz 5

required arguments:
  -f dir, --folder dir  Input CLARITY TIFF folder/dir

optional arguments:
  -d , --down           Down-sample ratio (default: 5)
  -cn , --channum       Chan # for extracting single channel from multiple channel data␣
↪(default: 0)
  -cp , --chanprefix    Chan prefix (string before channel number in file name). ex: C00
  -ch , --channame      Output chan name (default: eyfp)
  -o , --outnii         Output nii name (script will append downsample ratio & channel␣
↪info to given name)
  -vx , --resx          Original resolution in x-y plane in um (default: 5)
  -vz , --resz          Original thickness (z-axis resolution / spacing between slices)␣
↪in um (default: 5)
  -c  [ ...], --center  [ ...] Nii center (default: 0,0,0 ) corresponding to Allen atlas␣
↪nii template
```

(continues on next page)

```
-dz , --downzdim      Down-sample in z dimension, binary argument, (default: 1) => yes
-pd , --prevdown      Previous down-sample ratio, if already downs-sampled
-h, --help            Show this help message and exit
```

To run the function with an input dir called `input_tiff_dir`, a down-sampling factor of 5 and an output called `test`, you would type:
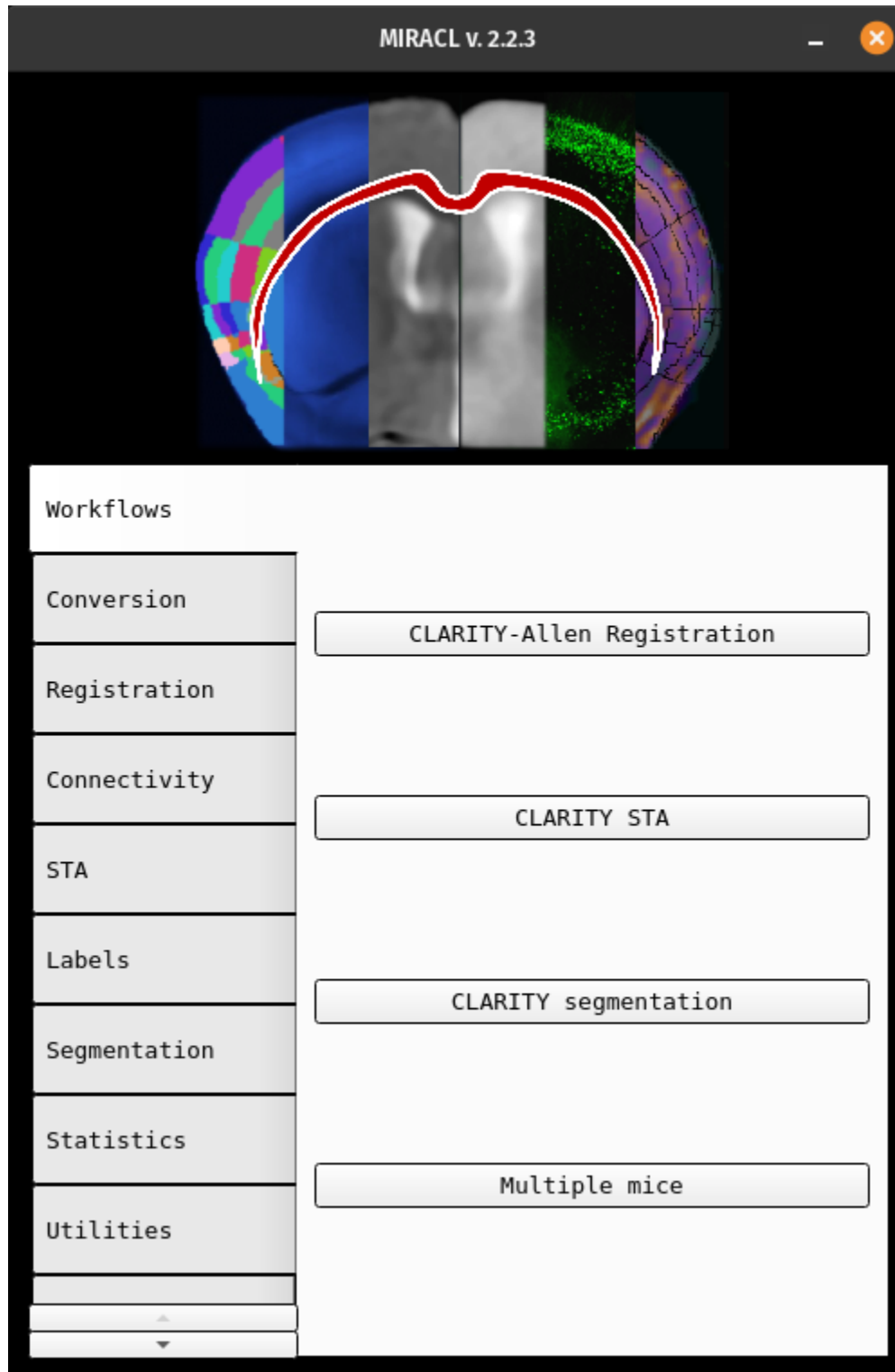
```
$ miracl conv tiff_nii -f input_tiff_dir -d 5 -o test
```

3.2.2 GUI

To run the main GUI:

```
$ miraclGUI
```

The GUI should open:

To get the GUI of a specific function, run it without arguments, for example:

```
$ miracl conv tiff_nii
```

Not all functions have GUIs yet. . . we are working on it!!!

See also:

Check the rest of the tutorials for more detailed documentation on modules and functions

3.3 Workflows

The workflow (`flow`) module combines multiple functions from various modules for ease of use to perform a desired task.

For example, a standard reg/seg analysis could look like this:

First perform registration of whole-brain CLARITY data to ARA:

```
$ miracl flow reg_clar -h
```

Then perform segmentation and feature extraction of full resolution CLARITY data:

```
$ miracl flow seg -h
```

Or structure tensor analysis:

```
$ miracl flow sta -h
```

Note that not all functions have tutorials yet. . . we are working on it!!!

3.3.1 CLARITY whole-brain registration to Allen Atlas

The registration workflow relies on an autofluorescence channel input (tiff files), and can perform whole-brain or hemi-sphere registrations to the Allen Atlas.

This workflow performs the following tasks:

1. Sets orientation of input data using a GUI

2. Converts TIFF to NII

3. Registers CLARITY data (down-sampled images) to Allen Reference mouse brain Atlas

4. Warps Allen annotations to the original high-res CLARITY space

5. Warps the higher-resolution CLARITY to Allen space (if chosen)

6. Test data

7. A test dataset (CLARITY autofluorescence channel) for registration is found here under data: https://www.dropbox.com/sh/i9swdedx7bsz1s8/AABpDmmN1uqPz6qpBLYLtt8va

Main outputs

File	Description
`reg_final/clar_allen_space.nii.gz`	CLARITY data in Allen reference space
`reg_final/clar_downsample_res{vox}um.nii.gz`	CLARITY data downsampled and oriented to 'standard'
`reg_final/annotation_hemi_{hemi}_{vox}um_clar_downsample.nii.gz`	Allen labels registered to downsampled CLARITY
`reg_final/annotation_hemi_{hemi}_{vox}um_clar_vox.tif`	Allen labels registered to oriented CLARITY
`reg_final/annotation_hemi_{hemi}_{vox}um_clar.tif`	Allen labels registered to original (full-resolution) CLARITY

GUI

Run:

```
$ miraclGUI
```

and choose CLARITY-Allen Registration from the Workflows tab:
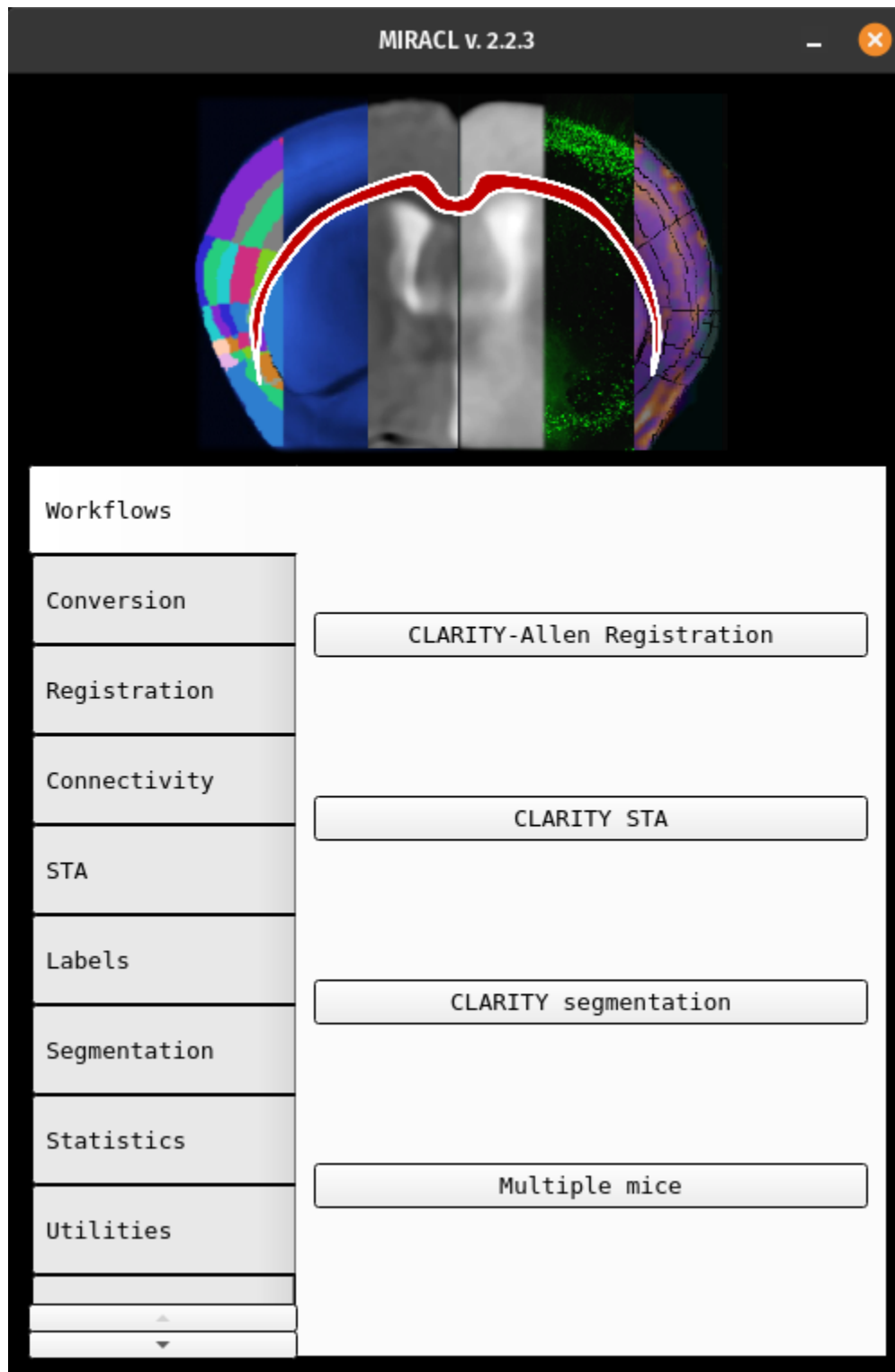
Or run:

```
$ miracl flow reg_clar
```

Choose the input tiff folder with the auto fluorescence channel from the pop-up menu:
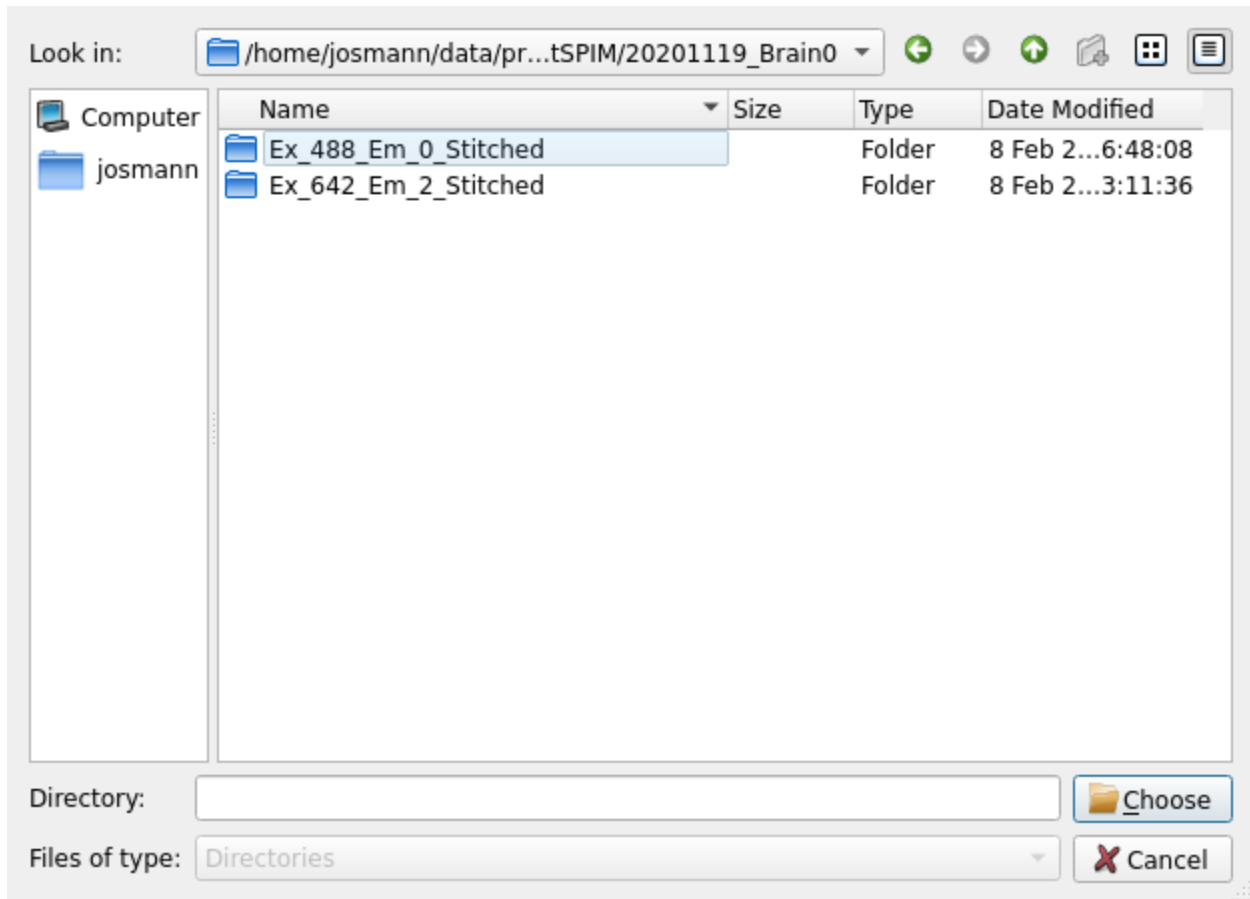
The following GUI will appear which opens the data and lets you set its orientation manually:
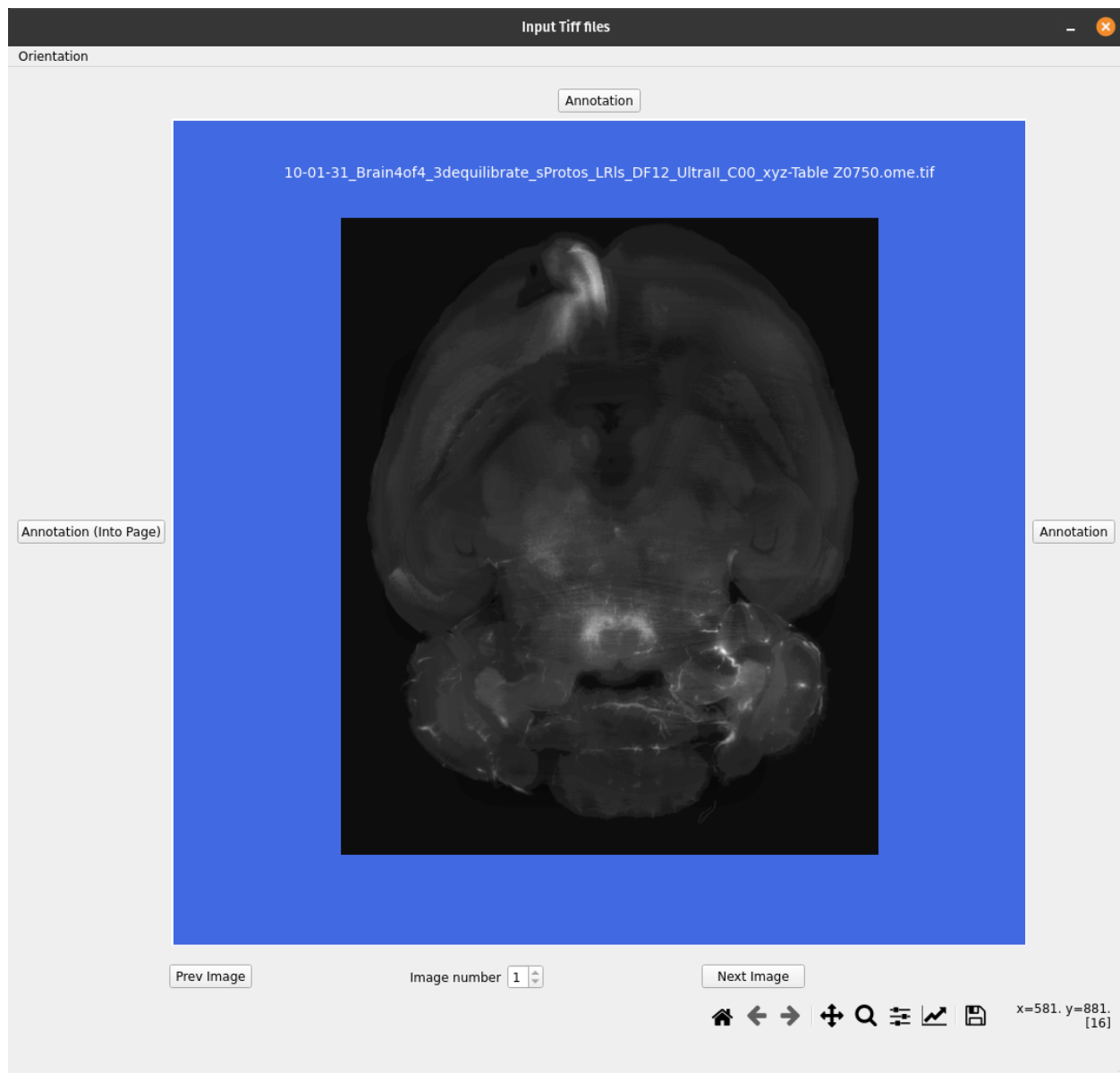
You can navigate through the data using the bar bellow, by specifying the slice number or using the arrows:
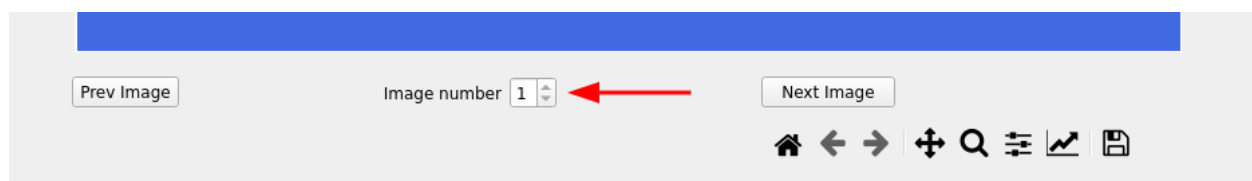
First choose the data plane (axial, coronal or sagittal):
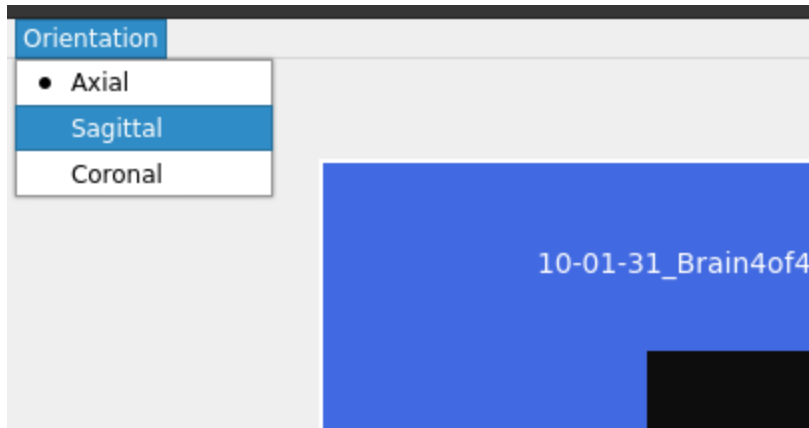
Then choose the orientation at the top and right of the image:
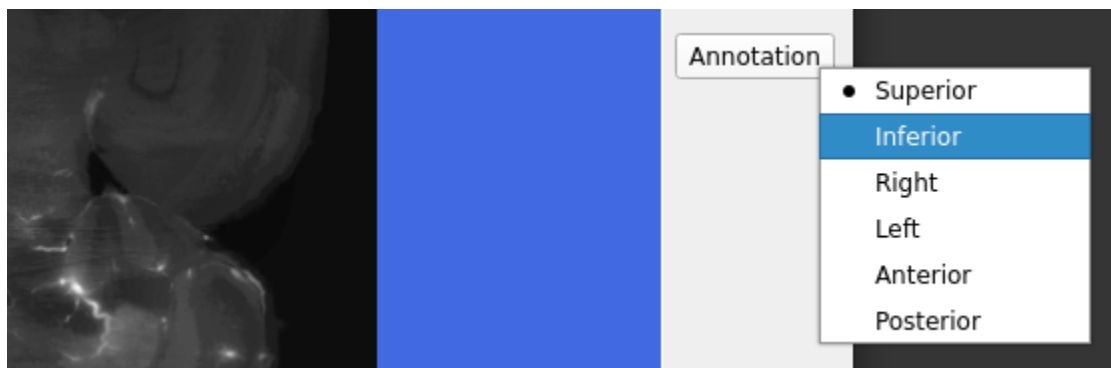
Next, choose the orientation for scrolling through the slices (going into the page), can confirm the orientation by changing the image number at the bottom (enter higher number and press `Enter`), or using the `Next` or `Prev` image buttons:
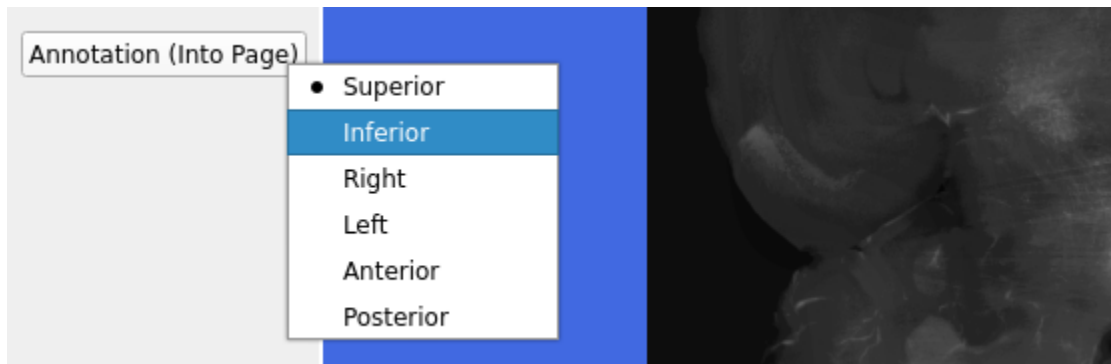
Finally close the GUI:
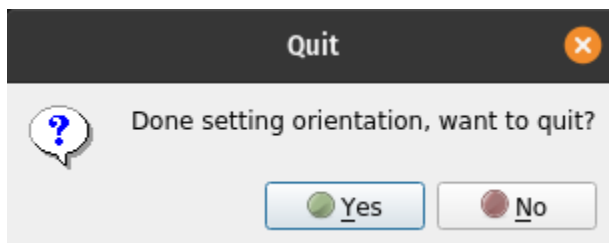
Next, set the tiff conversion parameters:

Conversion parameters description:

Parameter	Description Default	
out nii	Output nifti name	`clarity`
downsample ratio	Downsample factor for conversion	5
channel #	Number for extracting single channel from multiple channel data (leave blank if single channel data/tiff files)	`0`
channel prefix	String before channel number in file name (leave blank if single channel). For example, if tiff file name has `_C001_.tif` for channel 1 and `_C002_.tif` for channel 2, to choose channel 1 if it's the auto fluorescence channel: • Chan number would be: `1` • Chan prefix would be: `C00`	Channel prefix not invoked if not provided
channel name	Output channel name	`eyfp`
in-plane res	Original resolution in x-y plane in um	5
z res	Thickness (z-axis resolution/spacing between slices) in um	5
center	Center of nifti file	`0 0 0`

Next, choose the registration options:

Registration parameters description:

Parameter	Description	Default
Hemi	Warp Allen labels with hemisphere split (Left labels are different from Right labels) or combined (Left and Right lables are the same i.e. mirrored). Accepted inputs are: • `combined` • `split`	`combined`
Labels resolution [vox]	Voxel size/resolution of labels in um accepted inputs are: `10`, `25` or `50`	`10`
olfactory bulb	If olfactory bulb is included in the dataset. Accepted inputs are: `0` (not included) `1` (included)	`0`
Side	**Only if registering hemisphere, else leave blank.** Accepted inputs are: `rh` (right hemisphere) `lh` (left hemisphere)	None

Command-line

Usage:

```
$ miracl flow reg_clar -f [ Tiff folder ]
```

Example:

```
$ miracl flow reg_clar  -f my_tifs -n "-d 5 -ch autofluo" -r "-o ARS -m combined -v 25"
```

Arguments (required):

```
f. Input Clarity tif dir/folder
```

Optional arguments (remember the quotes):

```
Conversion to nii (invoked by -n " "):

d.  [ Downsample ratio (default: 5) ]
cn. [ chan # for extracting single channel from multiple channel data (default: 0) ]
cp. [ chan prefix (string before channel number in file name). ex: C00 ]
ch. [ output chan name (default: eyfp) ]
vx. [ original resolution in x-y plane in um (default: 5) ]
vz. [ original thickness (z-axis resolution / spacing between slices) in um (default: 5)␣
↪]
c.  [ nii center (default: 5.7 -6.6 -4) corresponding to Allen atlas nii template ]

Registration (invoked by -r " "):

o. Orient code (default: ALS)
   to orient nifti from original orientation to "standard/Allen" orientation
m. Warp allen labels with hemisphere split (Left different than Right labels) or␣
↪combined (L & R same labels / Mirrored)
   accepted inputs are: <split> or <combined>  (default: combined)
v. Labels voxel size/Resolution of labels in um
   accepted inputs are: 10, 25 or 50  (default: 10)
l. image of input Allen Labels to warp (default: annotation_hemi_split_10um.nii.gz -␣
↪which are at a resolution of 0.01mm/10um)
   input could be at a different depth than default labels
   If l. is specified (m & v cannot be specified)
```

Visualize results

Run:

```
$ miracl reg check
```

Usage:

```
$ miracl reg check -f [ reg_final_folde r] -v [ visualization_software ] -s [ reg_space_
↪(clarity_or_allen) ]
```

Example:

```
$ miracl reg check -f reg_final -v itk -s clarity
```

Arguments:

```
Required:

  -f, Input final registration folder

Optional:
```

```
-v, Visualization software: itkSNAP 'itk' (default) or freeview 'free'
-s, Registration Space of results: clarity (default) or allen
```

3.3.2 STA workflow

Run the structural tensor analysis (STA) workflow for fiber quantification and tracking.

> **Attention:** Run workflow after running the CLARITY-Allen registration first

Workflow for STA:

1. Converts Tiff stack to nii incl. down-sampling

2. Uses registered labels to create a seed mask at the depth (ontology level) of the desired label (seed)

3. Creates a brain mask

4. Runs STA analysis using the seed and brain masks

5. Computes virus intensities for all labels at that depth

Executes:

```
conv/miracl_conv_convertTIFFtoNII.py
lbls/miracl_lbls_get_graph_info.py
lbls/miracl_lbls_generate_parents_at_depth.py
utils/miracl_extract_lbl.py
utils/miracl_create_brainmask.py
sta/miracl_sta_track_primary_eigen.py
lbls/miracl_lbls_stats.py
```

Main Outputs

File	Description
`clarity_sta_{label}_seed/dog{dog}_gauss{gauss}/` `filter_ang{angle}.trk`	Tract file
`virus_signal_stats_depth_{depth}.csv`	Virus stats csv
`sta_streamlines_density_stats_depth_{depth}.csv`	Streamline density stats csv

GUI

From the main GUI (invoked with: `$ miraclGUI`), select `Workflows -> CLARITY STA`:
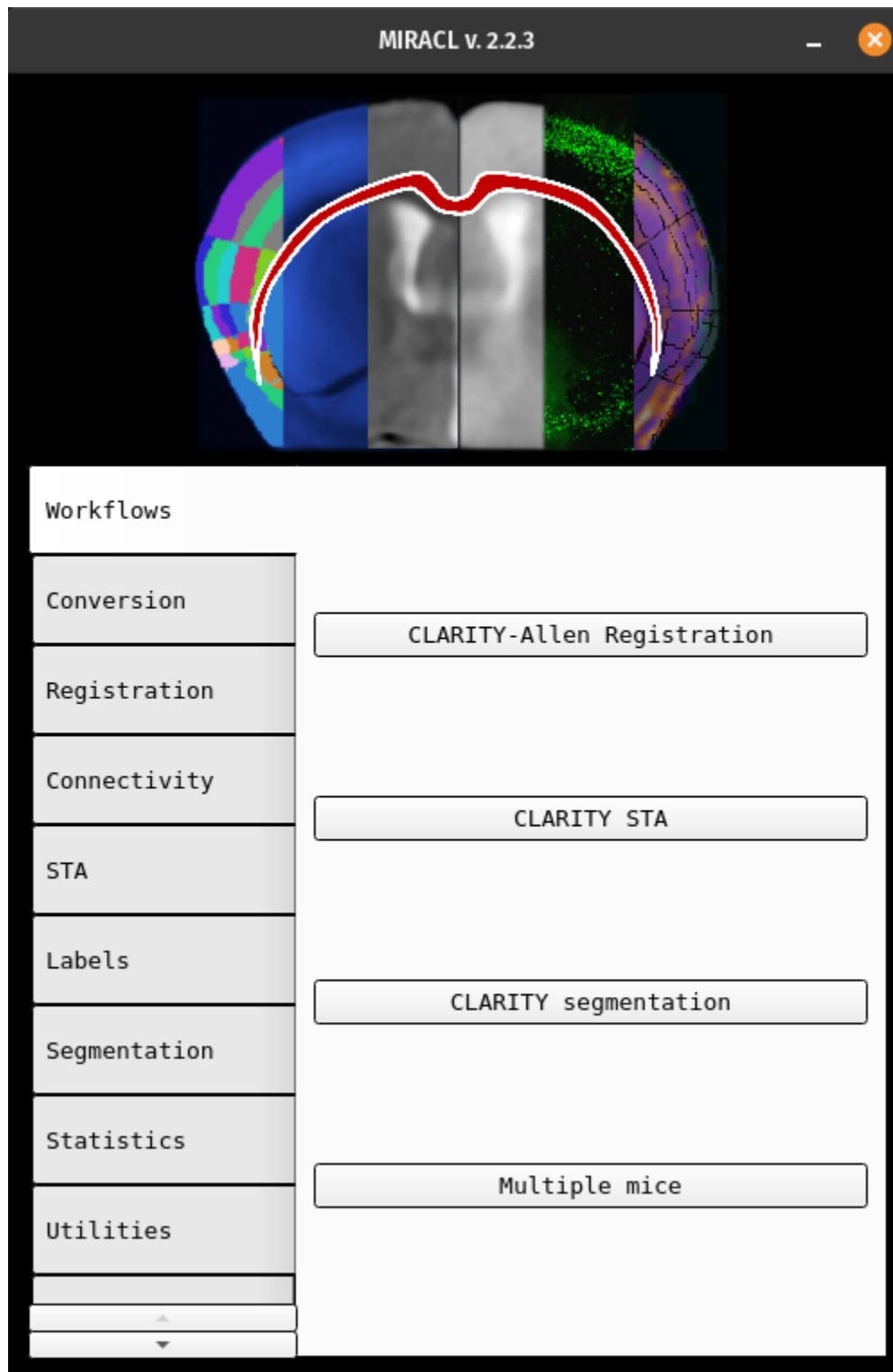
The following window will appear:

STA workflow	
No file selected	Select Brain mask (optional)
No file selected	Select Seeding label mask (optional)
No Dir selected	Select Input tiff folder
No Dir selected	Select CLARITY final registration folder
Out nii name (def = clarity)	
Seed label abbreviation	
hemi (combined or split)	
Derivative of Gaussian (dog) sigma	
Gaussian smoothing sigma	
Tracking angle threshold	
Downsample ratio (def = 5)	
chan # (def = 1)	
chan prefix	
Out chan name (def = AAV)	
Resolution (x,y) (def = 5 um)	
Thickness (z) (def = 5 um)	
Downsample in z (def = 1)	
Dilation factor for x axis (def = 0)	
Dilation factor for y axis (def = 0)	
Dilation factor for z axis (def = 0)	
Step length (def 0.1)	
Use 2nd order range-kutta method for tracking (def 0)	
Output directory	
Help	Enter
	Run

Hint: To open the STA workflow menu directly use: `$ miracl flow sta`

Click on `Select Input tiff folder` and choose the folder that contains the virus channel from the dialog window.

Then choose the registered Allen labels inside the final registration folder (`reg_final`) from the dialog window by clicking on `Select CLARITY final registration folder`.

Next choose the output file name (`Output nii name`), e.g. `Mouse05`. Our script will automatically append downsample ratio and channel info to the given name.

Set the tracking parameters:

Parameter	Description	Default
Seed label abbreviation	From Allen atlas ontology, for the seed region. Examples: Combined hemispheres: • CP for Caudoputamen • PL for Prelimbic Area Right hemisphere: • RCP for Right Caudoputamen • RPL for Right Prelimbic Area	Required. Function will exit with error 1 if not provided.
hemi	Labels hemisphere. Accepted inputs are: • *combined*` (both) • `split` (left or right)	`combined`
Derivative of Gaussian (dog) sigma	Example: `1`	`0.5,1.5`
Gaussian smoothing sigma	Example: `0.5`	`0.5,2`
Tracking angle threshold	Example: `35`	`25,35`
Use 2nd order runge-kutta method for tracking	Use 2nd order runge-kutta: • `0` (don't use) • `1` (use)	`0`

And the tiff conversion parameters:

Parameter	Description	Default
Downsample ratio	Set the downsample ratio.	5
chan #	For extracting single channel from multiple channel data.	1
chan prefix	String before channel number in file name. Example:	`C00`
Resolution (x,y)	Original resolution in x-y plane in um.	5
Thickness	Original thickness (z-axis resolution/spacing between slices) in um.	5
Downsample in z	Downsample in z dimension. Binary: • `0` (no) • `1` (yes)	1

Users can also input their own brain mask, as well as their own seeding mask. Both masks would respectively replace the automatically generated brain mask and regional mask used for the tractography. Users also have the option to dilate the seed mask across any of the three dimensions, by a value (indicated by the `Dilation factor` fields).

Attention: Note that the following parameters are required:

- `tiff folder`

- `output nii name`

- `Seed label abbreviation`

- `CLARITY final registration folder`

- `hemi`

- `Derivative of Gaussian`

- `Gaussian smoothing sigma`

- `Tracking angle threshold`

After choosing the parameters, first press `Enter` to save them and then `Run` to start the workflow.

Command-line

Usage:

```
$ miracl flow sta -f [ Tiff folder ] -o [ output nifti ] -l [ Allen seed label ] -m [␣
↪hemisphere ] -r [Reg final dir] -d [ downsample ratio ]
```

Example:

```
$ miracl flow sta -f my_tifs -o clarity_virus -l PL -m combined -r clar_reg_final -d 5 -
↪c AAV g 0.5 -k 0.5 -a 25
```

Or for right PL:

```
$ miracl flow sta -f my_tifs -o clarity_virus -l RPL -m split -r clar_reg_final -d 5 -c␣
↪AAV -g 0.5 -k 0.5 -a 25
```

Arguments:

```
arguments (required):
  -f: Input Clarity tif folder/dir (folder name without spaces)
  -o: Output nifti
  -l: Seed label abbreviation (from Allen atlas ontology)
  -r: CLARITY final registration folder
  -m: Labels hemi
  -g: Derivative of Gaussian (dog) sigma
  -k: Gaussian smoothing sigma
  -a: Tracking angle threshold

optional arguments:
  -d: Downsample ratio (default: 5)
  -c: Output channel name
  -n: Chan number for extracting single channel from multiple channel data (default: 0)
  -p: Chan prefix (string before channel number in file name). ex: C00
  -x: Original resolution in x-y plane in um (default: 5)
  -z: Original thickness (z-axis resolution/spacing between slices) in um (default: 5)
```

```
-b: Brain mask (to replace brain mask automatically generated by workflow)
-u: Seed mask (in place of regional seed mask generated by workflow)
-s: Step length
--downz: Downsample in z
--dilationfx: Dilation factor for x (factor to dilate seed label by)
--dilationfy: Dilation factor for y (factor to dilate seed label by)
--dilationfz: Dilation factor for z (factor to dilate seed label by)
--rk: Use 2nd order range-kutta method for tracking (default: 0)
--out_dir: Output directory
```

3.3.3 CLARITY whole-brain segmentation

There are multiple segmentation functions for different data (stains/channels):

- `virus`

- `cFos`

- `sparse`

- `nuclear`

The segmentation workflow relies on an output from the registration workflow, but the segmentation wrapper function can be run without running the registration workflow.

This workflow performs the following tasks:

1. Segments neurons in cleared mouse brain of sparse or nuclear stains in 3D

2. Voxelizes segmentation results into density maps with Allen Atlas resolution

3. Computes features of segmented image and summarizes them per label

It executes:

```
seg/miracl_seg_clarity_neurons_wrapper.sh
seg/miracl_seg_voxelize_parallel.py
seg/miracl_seg_feat_extract.py
```

Main outputs

File	Description
segmentation/seg.{tif,mhd} or seg_nuclear.{tif,mhd}	Segmentation image with all labels (cells)
segmentation/seg_bin.{tif,mhd} or seg_bin_nuclear.{tif,mhd}	Binarized segmentation image
voxelized_seg.{tif,nii}	Segmentation results voxelized to ARA resolution
voxelized_seg_bin.{tif,nii}	Binarized version
clarity_segmentation_features_ara_labels.csv	Segmentation features summarized per ARA labels

Hint: Results can be opened in Fiji for visualization

GUI

Select from the main GUI menu (invoked from the cli: `$ miraclGUI`) or run:

```
$ miracl flow seg
```
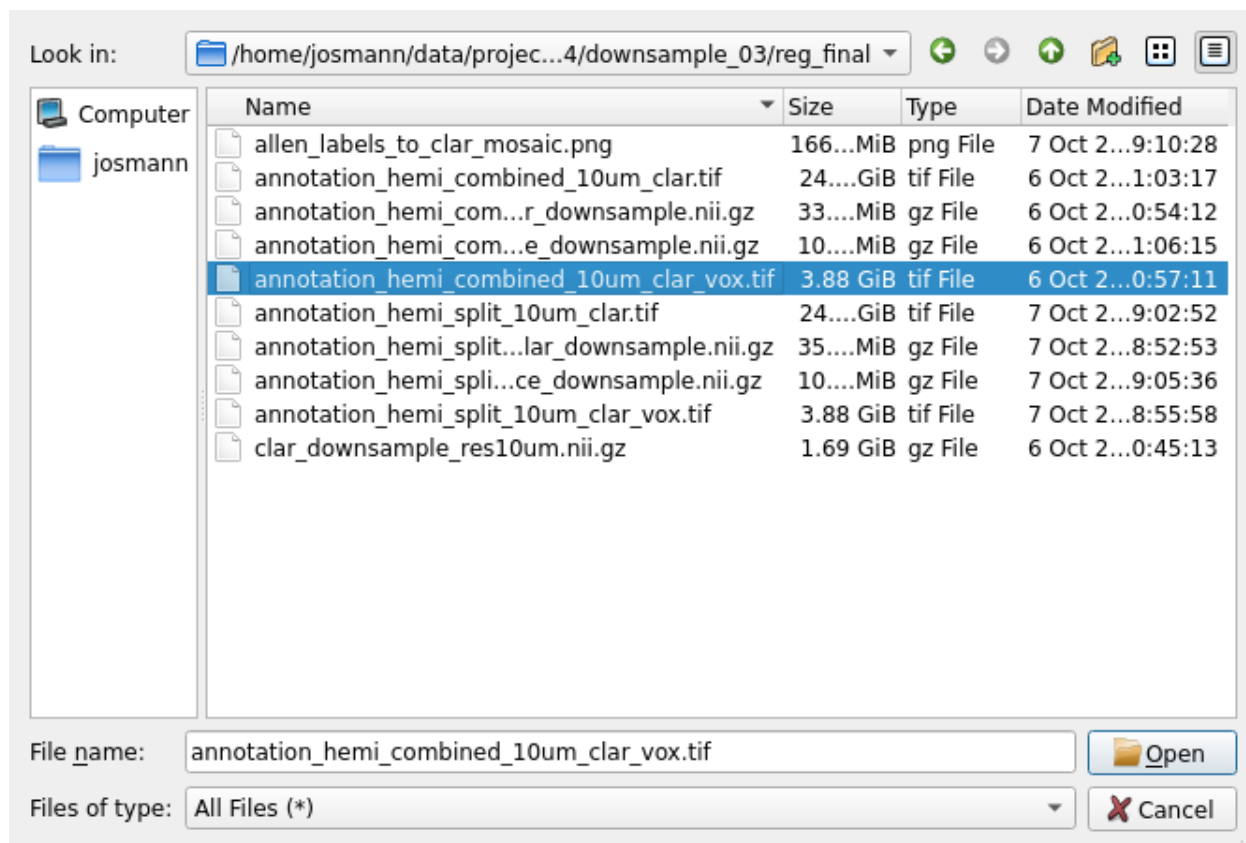
The following window will appear:

Click on `Select registered labels (..clar_vox.tif) in the reg_final dir` to choose the registered labels `annotation_hemi_{side}_**um_clar_vox.tif` to summarize segmentation features where:

- `{side}` -> `combined` or `split`
- `**` is the resolution -> `10`, `25` or `50`

The following window will appear:

Next, click on `select input tiff dir` to select folder with Thy1-YFP or other channel:
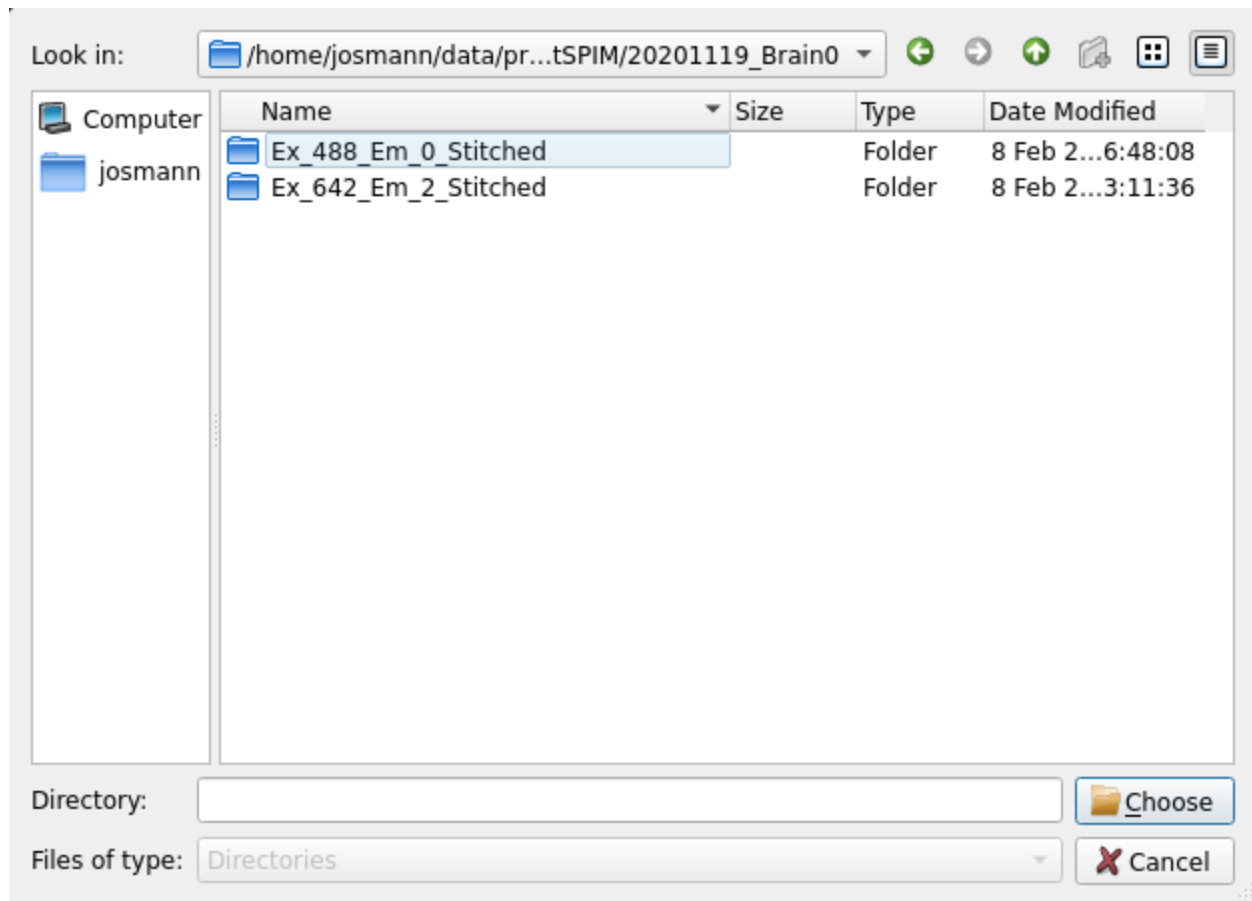
Lastly set the segmentation parameters:

Parameter	Description	Default
seg type	Channel type: • `virus` • `cFos` • `sparse` (like `Thy1 YFP`) • `nuclear` (like `PI`)	`virus`
channel prefix	Channel prefix and number if multiple channels. Example: `Filter0001`	`None`
labels voxel size	Registered labels voxel size in um: • `10` • `25` • `50`	`10`

Click `Enter` and `Run` to start the segmentation process.

Command-line

Usage:

```
$ miracl flow seg -f [ Tiff_folder ]
```

Example:

```
$ miracl flow seg -f my_tifs -t nuclear -s "-p C001" -e "-l reg_final/annotation_hemi_
→combined_25um_clar_vox.tif"
```

Arguments:

```
arguments (required):

  f. Input Clarity tif folder/dir [folder name without spaces]
  t. Channel type: sparse (like Thy1 YFP) or nuclear (like PI)

optional arguments (don't forget the quotes):

  Segmentation (invoked by -s " "):
    p. Channel prefix & number if multiple channels (like Filter0001)

  Feature extraction (invoked by -e " "):
    l. Allen labels (registered to clarity) used to summarize features
        reg_final/annotation_hemi_{hemi}_{vox}um_clar_vox.tif
```

3.4 Conversion

Functions to convert between image file formats e.g. converting tiff image files to nifti format.

Note that not all functions have tutorials yet. . . we are working on it!!!

3.4.1 Tiff to Nifti

Function to convert Tiff images to Nifti format for analysis or visualization.

GUI

Run:

```
$ miracl conv tiff_nii
```

The following window will open:

Click on `Select Input tiff folder` to choose the input tiff folder:
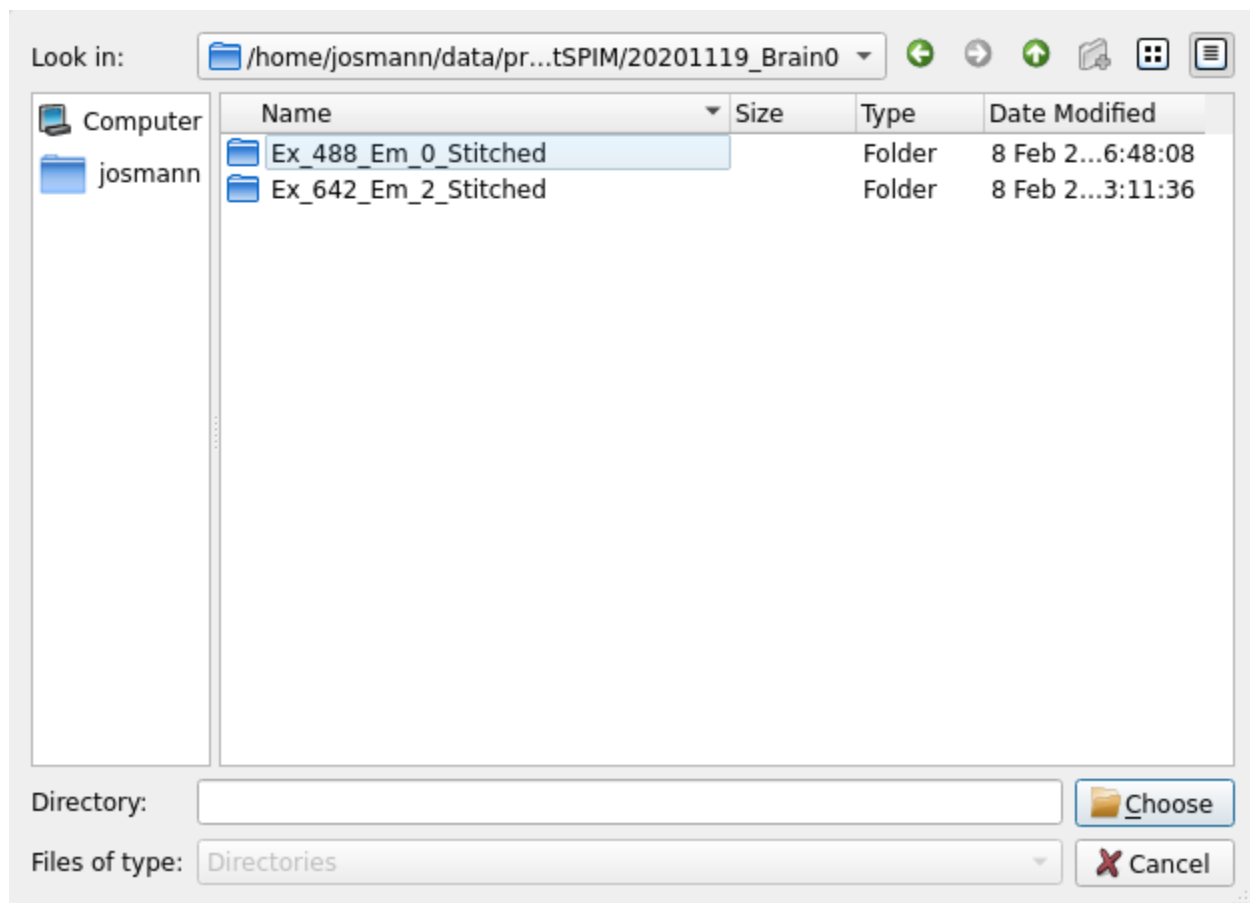
Next set the desired parameters or use the default by leaving the fields blank:

Parameters	Description	Default
Out nii name	Output file name.	`clarity`
Downsample ratio	Downsample factor for nifti images.	5
chan # and prefix	Use if tiff files have more than one channel. For example, given the names `10-04-06_R923_06R1ca_647_100p350_na144_UltraII_**C001**_xyz-Table Z1284.ome.tif` for channel 1 and `10-04-06_R923_06R1ca_647_100p350_na144_UltraII_**C002**_xyz-Table Z1284.ome.tif` for channel 2 with the latter being the desired channel for conversion, `chan #` would be `2` and `chan prefix` would be `C00`.	Not invoked if not provided
Out chan name	Output channel name.	`eyfp`
Resolution (x,y)	Original resolution in x-y plane in um.	5
Thickness	Original thickness (z-axis resolution/spacing between slices) in um.	5
center	Center of Nifti file.	`0 0 0`
Downsample in z	Down-sample in z dimension. Binary: • `0` (no) • `1` (yes)	1
Prev Downsampling	Previous downsample ratio if already downsampled. Accepted inputs are: • `0` (downsampled) • `1` (not downsampled)	1

After choosing the parameters press `Enter` to save them then `Run` to start the conversion process.

Tip: After the conversion is done, nifti (`nii/nii.gz`) files can be visualized using the **ITKsnap** software

---

Command-line

Usage:

```
$ miracl conv tiff_nii -f [ Tiff_folder ]
```

Example:

```
miracl conv tiff_nii -f my_tifs -o stroke2 -cn 1 -cp C00 -ch Thy1YFP -vx 2.5 -vz 5
```

Required arguments:

```
-f dir, --folder dir   Input CLARITY TIFF folder/dir
```

Optional arguments:

```
-d, --down             Down-sample ratio (default: 5)
-cn, --channum         Chan # for extracting single channel from multiple channel data
↪(default: 1)
-cp, --chanprefix      Chan prefix (string before channel number in file name). ex: C00
-ch, --channame        Output chan name (default: eyfp)
-o, --outnii           Output nii name (script will append downsample ratio & channel info
↪to given name)
-vx, --resx            Original resolution in x-y plane in um (default: 5)
-vz, --resz            Original thickness (z-axis resolution / spacing between slices) in
↪um (default: 5)
-c [ ...], --center    [ ...]
                       Nii center (default: 0,0,0 ) corresponding to Allen atlas nii
↪template
-dz, --downzdim        Down-sample in z dimension, binary argument, (default: 1) => yes
-pd, --prevdown        Previous down-sample ratio, if already down-sampled
-h, --help             Show this help message and exit
```

3.5 Registration

Registration functions for e.g. registering data (down-sampled images) to Allen Reference mouse brain atlas.

Note that not all functions have tutorials yet. . . we are working on it!!!

3.5.1 CLARITY-Allen registration

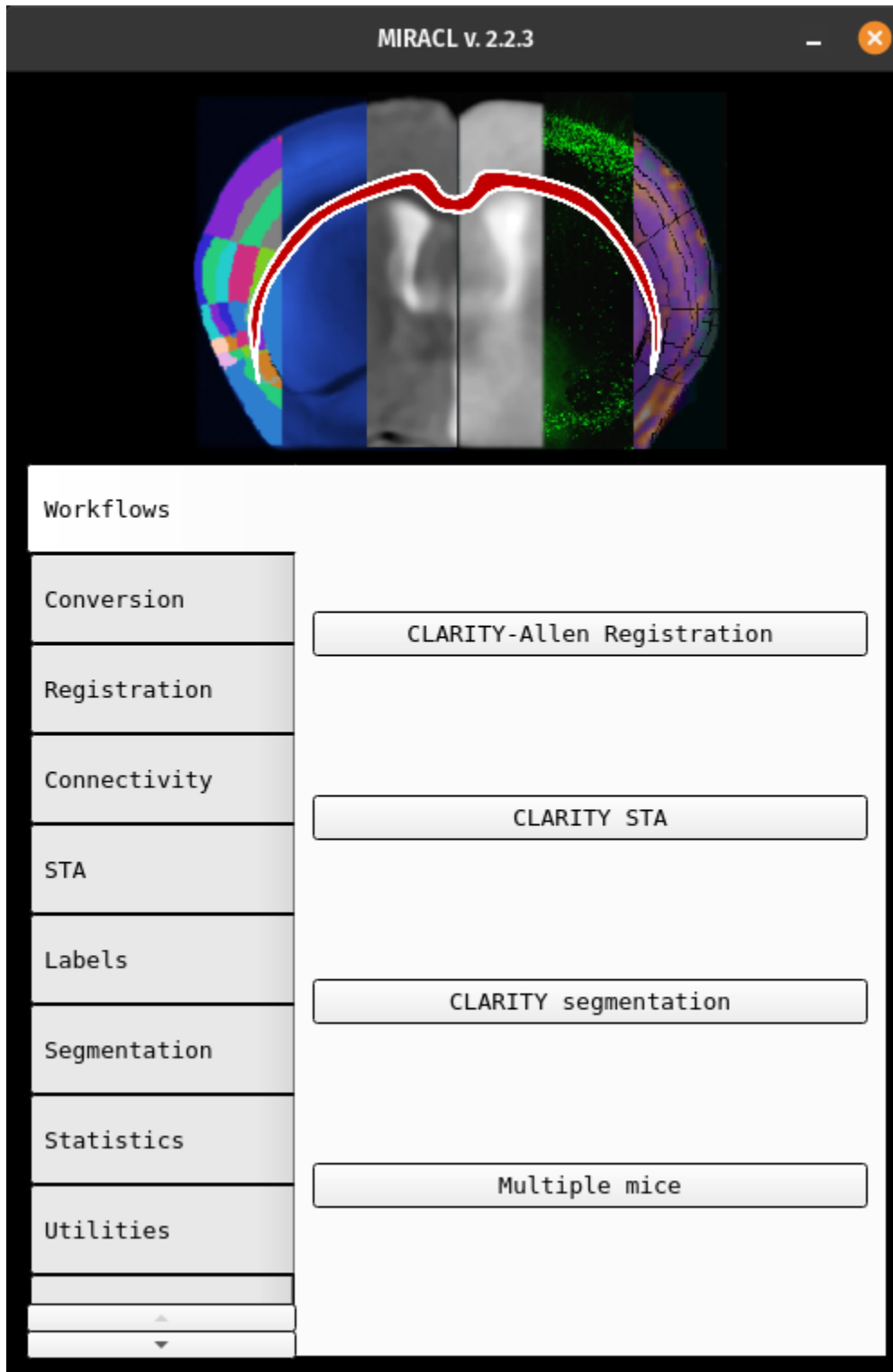This function will do the following:

1. Registers CLARITY data (down-sampled images) to Allen Reference mouse brain atlas

2. Warps Allen annotations to the original high-res CLARITY space

3. Warps the higher-resolution CLARITY to Allen space

GUI

To open the main registration menu, open MIRACL's main menu first by running:
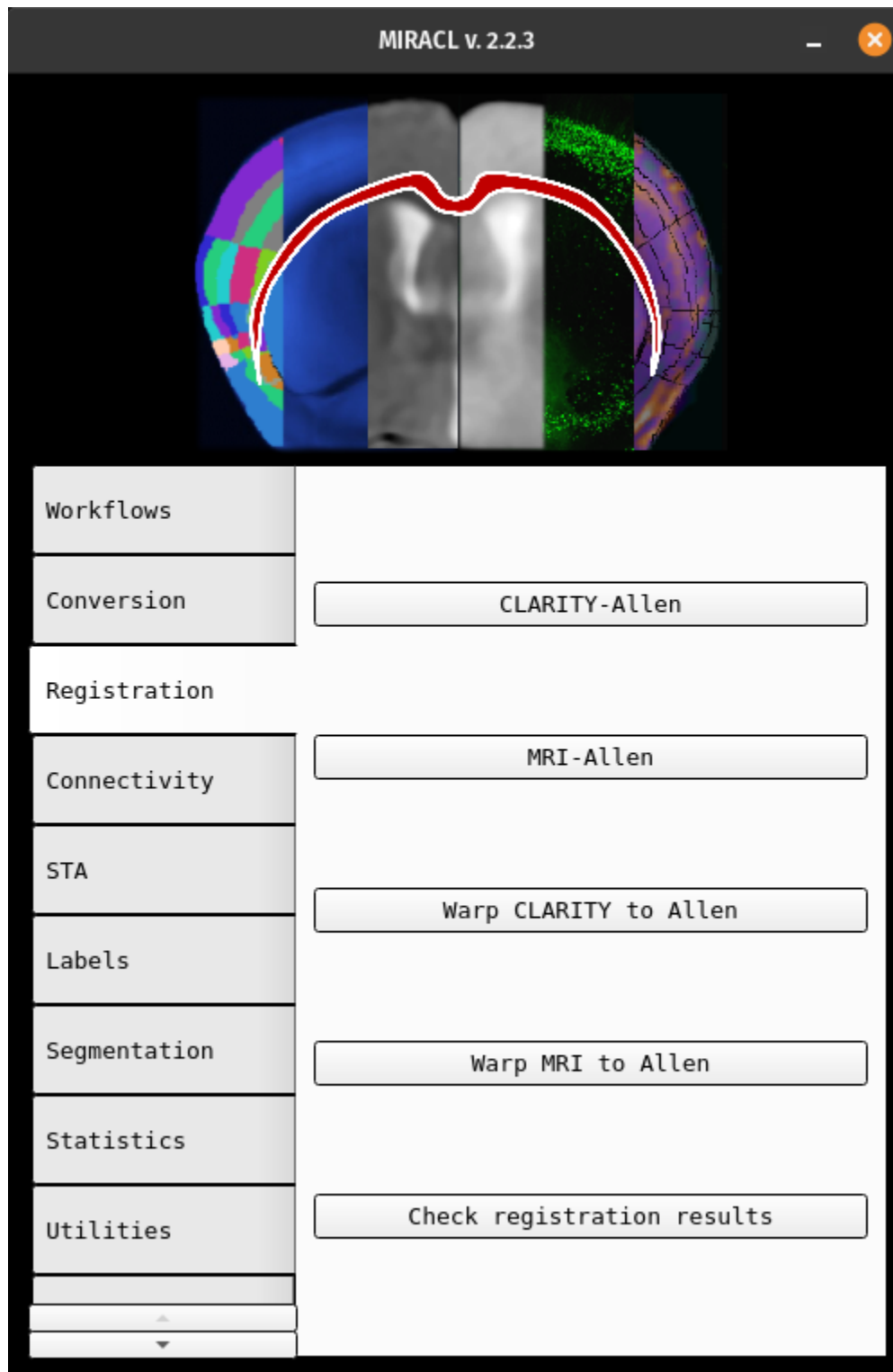
```
$ miraclGUI
```

MIRACL's main menu will open:

Select the `Registration` tab on the left for the main registration menu.

The main registration window will look like this:

From here you can select `CLARITY-Allen` to start the registration. The `Reg options` menu will open:

Tip: To open the above Reg options menu directly, run: `$ miracl reg clar_allen`

The registration will be run on downsampled CLARITY Nii images. You can provide the folder containing these files in the first field. **This parameter is required to run the registration**. You can use `MIRACL's` conversion methods to create the downsampled files if you do not have them yet.

Flag	Parameter	Description	Default
-i	Input down-sampled CLARITY Nii	Preferably auto-fluorescence channel data (or Thy1_EYFP if no auto chan). File name should have `**x_down` like `05x_down` (meaning 5x downsampled). Example: • `combined` • `split`	Required. Script exits with error 1 if not provided.

All remaining parameters are optional. If left blank, their respective default values will be used:

Flag	Parameter	Description	Default
-r	Output directory	Directory the output (results) will be written to.	`<working_directory>/` `reg_final`
-o	Orient code	Code to orient nifti from original orientation to 'standard/Allen' orientation.	`ALS`
-m	Labels hemi	Chose to register to one or both hemispheres. Warps Allen labels with hemisphere split (L differ from R labels) or combined (L and R have the same labels i.e. are mirroed). Accepted inputs are: • `combined` • `split`	`combined`
-v	Labels resolution [vox]	Labels voxel size/resolution in um. Accepted inputs are: • `10` • `25` • `50`	`10`
-b	Olfactory bulb included	Specify whether the olfactory bulb is included in brain. Accepted inputs are: • `0` (not included) • `1` (included)	`0`
-s	Side	Provide this parameter if you are only registering one hemisphere instead of the whole brain. Accepted inputs are: • `rh` (right hemisphere) • `lh` (left hemisphere)	`None`
-p	Extra int correct	If utilfn intensity correction has already been run, skip correction inside registration. Accepted inputs are: • `0` (don't skip) • `1` (skip)	`0`

After providing the parameters click `Enter` to save them and `Run` to start the registration process.

Once the registration is done the final files will be located in the output folder (default: `<working_directory>/`

reg_final). Files created in intermediate steps will be located in a folder called <working_directory>/
clar_allen_reg.

Command-line

The command-line version has additional functionality that is not included in the GUI version:

```
-l, input Allen labels to warp: input labels could be at a different depth than default␣
↪labels.
    -m and -v flags cannot be used if this parameter is specified manually (default:␣
↪annotation_hemi_combined_10um.nii.gz)
-a, input custom Allen atlas: for example for registering sections
-f, save mosaic figure (.png) of Allen labels registered to CLARITY (default: 1).
-w, warp high-res clarity to Allen space (default: 0).
```

> **Attention:** Note that the above listed `-i` parameter (`input down-sampled CLARITY Nii`) is also required for
> the command-line

Usage:

```
$ miracl reg clar_allen -i [ input_clarity_nii_folder ] -o [ orientation_code ] -m [␣
↪hemispheres ] -v [ labels_resolution ] -l [ input_labels ] -s [ side_if_hemisphere_
↪only ] -b [ olfactory_buld_included ]
```

Example:

```
$ miracl reg clar_allen -i downsampled_niftis/SHIELD_03x_down_autoflor_chan.nii.gz -o␣
↪ARI -m combined -b 1
```

3.5.2 MRI whole-brain registration to Allen Atlas

This registration method performs the following tasks:

1. Registers in-vivo or ex-vivo MRI data to Allen Reference mouse brain Atlas

2. Warps Allen annotations to the MRI space

GUI

Invoke with `$ miraclGUI` and select from main menu or run:

```
$ miracl reg mri_allen_nifty
```

The following window will open:

Click on `Select In-vivo or Ex-vivo` MRI and choose the input MRI nii (preferable T2-w) using the dialog window. Then set the registration options:

Parameter	Description	Default
Orient code	Orient nifti from original orientation to 'standard/Allen' orientation.	RSP
Labels Hemi	Warp allen labels with hemisphere split (Left different than Right labels) or combined (Left and Right labels are the same/mirrored). Accepted inputs are: • `split` • `combined`	`combined`
Labels resolution [vox]	Labels voxel size/resolution in um. Accepted inputs are: • `10` • `25` • `50`	`10`
Olfactory bulb included	Specify whether the olfactory bulb is included in brain. Accepted inputs are: • `0` (not included) • `1` (included)	`0`
skull strip	Strip skull. Accepted inputs are: • `0` (don't strip) • `1` (strip)	`1`
No orient	No orientation needed (input image in 'standard' orientation). Accepted inputs are: • `0` (orient) • `1` (don't orient)	`0`

Click `Enter` and `Run` to start the registration process.

Command-line

Usage:

```
$ miracl reg mri_allen_nifty -i [ input invivo or exvivo MRI nii ] -o [ orient code ] -m␣
→[ hemi mirror ] -v [ labels vox ] -l [ input labels ] -b [ olfactory bulb ] -s [ skull␣
→strip ] -n [ no orient needed ]
```

Example:

```
$ miracl reg mri_allen_nifty -i inv_mri.nii.gz -o RSP -m combined -v 25
```

Arguments:

```
arguments (required):
```

(continues on next page)

```
  i.   input MRI nii
       Preferably T2-weighted

optional arguments:

  o.   orient code (default: RSP)
       to orient nifti from original orientation to "standard/Allen" orientation
  m.   hemisphere mirror (default: combined)
       warp allen labels with hemisphere split (Left different than Right labels) or␣
→combined (L & R same labels / Mirrored)
       accepted inputs are: <split> or <combined>
  v.   labels voxel size/Resolution in um (default: 10)
       accepted inputs are: 10, 25 or 50
  l.   input Allen labels to warp (default: annotation_hemi_combined_10um.nii.gz )
       input labels could be at a different depth than default labels
       If l. is specified (m & v cannot be specified)
  b.   olfactory bulb included in brain, binary option (default: 0 -> not included)
  s.   skull strip or not, binary option (default: 1 -> skull-strip)
  f.   FSL skull striping fractional intensity (default: 0.3), smaller values give larger␣
→brain outlines
  n.   No orientation needed (input image in "standard" orientation), binary option␣
→(default: 0 -> orient)
```

3.6 Utilities

A collection of utility functions.

Note that not all functions have tutorials yet. . . we are working on it!!!

3.6.1 Intensity correction

Intensity correction for data with inhomogeneity issues. Performs correction on CLARITY tiff data in parallel using N4.

1. Creates a downsampled nifti from the tiff data

2. Runs N4 'bias field'/intensity correction on the nifti file

3. Up-samples the output bias field and applies it to the tiff data

Command-line

Usage:

```
$ miracl utils int_corr -f [ input_tiff_folder ] -od [ output_folder ] -s [ shrink_
→factor] -cn [ channel_num ] -cp [ channel_prefix ] -p [ power ]
```

Example:

```
$ miracl utils int_corr -f tiff_folder -od bias_corr_folder
```

Required arguments:

Flags	Description	Default Default
-f, –folder	Input CLARITY TIFF folder/dir	No default. Input folder must be provided by user.

Optional arguments:

Flags	Description	Default
-od, --outdir	Output folder name	`int_corr_tiffs`
-cn, --channum	Chan # for extracting single channel from multiple channel	`data 1`
-cp, --chanprefix	Chan prefix (string before channel number in file name). Example: `C00`	`None`
-ch, --channame	Output chan name	`AAV`
-on, --outnii	Output nii name (script will append downsample ratio & channel info to given name)	
-vx, --resx	Original resolution in x-y plane in	`um 5`
-vz, --resz	Original thickness (z-axis resolution/spacing between slices) in um	`5`
-m, --maskimg	Mask images before	`correction 1`
-s, --segment	Perform level-set seg using brain mask to get a dilated	`one 0`
-d, --down	Downsample/shrink factor to run bias corr on downsampled	`data 5`
-n, --noise	Noise parameter for histogram sharpening - deconvolution	`0.005`
-b, --bins	Histogram bins	`200`
-k, --fwhm	FWHM for histogram sharpening - deconvolution	`0.3`
-l, --levels	Number of levels for	`convergence 4`
-it, --iters	Number of iterations per level for convergence	`50`
-t, --thresh	Threshold per iteration for	`convergence 0`
-p, --mulpower	Use the bias field raised to a power of `p` to enhance its effects	`1.0`

3.7 HPC/SLURM clusters

MIRACL was built with HPC/SLURM clusters in mind. We recommend **Singularity** as it is well suited to run in a cluster environment. We provide a **Singularity** container of **MIRACL's** latest version that can be pulled to a node directly from our online repo.

We provide tutorials on how to use **MIRACL** on Compute Canada and Sherlock (supercomputer at Stanford university) but the principles explained here will be similar to other SLURM clusters.

Note: If you would like to add a tutorial for a particular cluster that you are working with that is missing here, we invite you to add it to this section by submitting a PR (note that we write Sphinx documentation in `.rst` format) through our official GitHub.

3.7.1 Running MIRACL on Compute Canada

This tutorial highlights the registration workflow but a similar approach applies to other commands.

When using Compute Canada, `MIRACL` can be used as a `Singularity` container. The following instructions are based on the steps provided on the Compute Canada Wiki.

Copy your data to Compute Canada

For example, to copy a folder called `input_clar` containing tiff files you want to register to the Allen Atlas use:

```
$ scp -r input_clar niagara.computecanada.edu:/scratch/<username>/.
```

or

```
$ rsync -avPhz input_clar <username>@niagara.computecanada.edu:/scratch/<username>/.
```

Log in to Compute Canada server

Log in to the Compute Canada server you copied your data to:

```
$ ssh -XY <username>@niagara.computecanada.edu
```

Setting up and using MIRACL

Load the specific Singularity module you would like to use (e.g. Singularity 3.5):

```
$ module load singularity/3.5
```

Note: Not specifying the version will load the latest version available on your node

Since `MIRACL` will take up a significant amount of space, it is recommended to download and work with the `MIRACL` `Singularity` container in the scratch directory. First, navigate there:

```
$ cd $SCRATCH
```

Then pull (download) the `Singularity` container:

```
$ singularity pull miracl_latest.sif library://aiconslab/miracl/miracl:latest
```

Attention: `singularity pull` requires `Singularity` version `3.0.0` or higher. Please refer to our *Troubleshooting* section ("Q: Can I build a Singularity container from the latest MIRACL image on Docker Hub") if you are using an older version of Singularity.

Note: If you have a particular Singularity container of `MIRACL` that you want to use on Compute Canada, just copy it to the servers directly using e.g. `scp` or `rsync` instead of pulling (downloading) the latest version of `MIRACL` from the `Singularity` registry

Start the **MIRACL** **Singularity** container with the default folders mounted:

```
$ singularity shell miracl_latest.sif bash
```

Singularity will automatically mount your scratch folder to your container. If you need to mount a specific directory into a specific location, use the following:

```
$ singularity shell -B <location_outside_container>/<source_mount>:<location_in_
→container>/<target_mount> miracl_latest.sif bash
```

Once you are logged in to the container, load the GUI from the shell:

```
$ miraclGUI
```

Note: Please consult our *Troubleshooting* section on **Singularity** if you experience problems with opening **MIRACL's** GUI on Compute Canada

Or use **MIRACL** from the command line. For example, run **MIRACL's** CLARITY registration workflow on the folder that you copied over previously:

```
$ miracl flow reg_clar -f input_clar -n "-d 5 -ch autofluo" -r "-o ARS -m combined -v 25"
```

Note: If you have a particular Singularity container of **MIRACL** that you want to use on Compute Canada, just copy it to the servers directly using e.g. `scp` or `rsync` instead of pulling (downloading) the latest version of **MIRACL** from the **Singularity** registry

3.7.2 Running MIRACL commands on Sherlock (Stanford supercomputer)

This tutorial highlights the registration workflow but a similar approach applies to other commands.

Setting up MIRACL (first time)

Log in to Sherlock:

```
$ ssh -Y username@sherlock.stanford.edu
```

Start an interactive session:

```
$ sdev
```

Move to your scratch folder:

```
$ cd SCRATCH
```

Pull (download) Singularity container:

```
$ singularity pull miracl_latest.sif library://aiconslab/miracl/miracl:latest
```

> **Attention:** `singularity pull` requires **Singularity** version `3.0.0` or higher. Please refer to our Troubleshooting section ("Q: Can I build a **Singularity** container from the latest MIRACL image on Docker Hub") if you are using an older version of **Singularity**.

> **Tip:** If you have a particular **Singularity** container of **MIRACL** that you want to use on Sherlock, just copy it to the servers directly using e.g. **scp** or **rsync** instead of pulling (downloading) the latest version of **MIRACL** from the **Singularity** registry

---

Copying your data to Sherlock

Copy a folder called, e.g. `input_clar` with tiff files that you want to register to the Allen Atlas using **scp**:

```
$ scp -r input_clar sherlock.stanford.edu:/scratch/users/<username>/clarity_registration/
↪.
```

or **rsync**:

```
$ rsync -avPhz input_clar sherlock.stanford.edu:/scratch/users/<username>/clarity_
↪registration/.
```

> **Attention:** Make sure to replace <username> with your Sherlock username

---

Running MIRACL in an interactive session

For quick jobs that don't require much resources you can login to Sherlock:

```
$ ssh -Y username@sherlock.stanford.edu
```

Move to your scratch folder:

```
$ cd SCRATCH
```

Start interactive session:

```
$ sdev
```

Start **Singularity** with binded data:

```
$ singularity shell miracl_latest.sif bash
```

Within the shell, load the GUI:

```
$ miraclGUI
```

Or use the command-line:

```
$ miracl lbls stats -h
```

Note: Please consult our Troubleshooting section if you experience problems with opening **MIRACL**'s GUI on Sherlock

---

Running SBATCH jobs

If you want to run jobs with specific resources for larger, longer jobs (e.g. running the registration workflow) you can do the following:

First get the data orientation (please check the registration tutorial for setting orientation):

```
$ miracl conv set_orient
```

After setting the orientation, a file called `ort2std.txt` will be created that might look like this:

```
$ cat ort2std.txt
tifdir=/scratch/users/username/clarity_registration/input_clar
ortcode=ARS
```

Use that orientation code (`ARS`) in your registration workflow.

First check the workflow arguments:

```
$ miracl flow reg_clar -h
```

Assuming you wanted to run this command with the following arguments, for example on your data:

```
$ miracl flow reg_clar -f input_clar -n "-d 5 -ch autofluo" -r "-o ARS -m combined -v 25"
```

Create an sbatch script named, for example `reg_job.sbatch` and paste the following lines:

```
#!/bin/bash
#SBATCH --job-name=clar_reg
#SBATCH --ntasks=1
#SBATCH --time=05:00:00
#SBATCH --cpus-per-task=12
#SBATCH --mem=32G

module load singularity

singularity exec ${SCRATCH}/miracl_latest.sif miracl flow reg_clar -f ${SCRATCH}/clarity_
→registration/input_clar -n "-d 5 -ch autofluo" -r "-o ARS -m combined -v 25"
```

Attention: Note that the `miracl` function call comes after invoking the **Singularity** call `singularity exec ${SCRATCH}/miracl_latest.sif` and that full file paths were used for the `.sif` container and the input data

---

This sample job (called: `clar_reg`) asks for 5 hours, 12 cpus and 32G of memory on one node. Adjust the requested resources based on the job you are submitting.

Next submit the sbatch script:

```
$ sbatch reg_job.sbatch
```

To check on the status of your submitted job use:

```
$ squeue -u $USER
```

See also:

For more resources on SLURM sbatch jobs check Stanford's tutorials on submitting and running jobs on Sherlock

TROUBLESHOOTING

Choose an troubleshooting issue from the sidebar menu or TOC.

If you cannot find an answer to your problem here, please open an issue on our offical GitHub page.

4.1 Docker

4.1.1 ⚠ MIRACL's GUI (miraclGUI) is not working

Access control might be enabled on your host machine. Change your `xhost` access control on your host machine (not within your **Docker** container). Exit the running **Docker** container and type:

```
xhost +
```

Log back in to your container. Reset `xhost` after you are done with using the **MIRACL** GUI using:

```
xhost -
```

The above will enable/disable access for all users. If this is not the desired behavior, access can be granted in a more fine grained manner.

Example:

```
xhost +SI:localsuer:<yourusername>
```

Note: Replace <yourusername> with the actual user name of your host system

4.1.2 ⚠ The GUI worked before but does not work anymore

Navigate to the directory that contains your `docker-compose.yml` and restart the container with `docker compose down` followed by docker `compose up -d`. The GUI should work again.

4.1.3 ⚠ I cannot run X or Y with Docker because of permission denied errors

If you have not set up a **Docker** user you might need to run **Docker** commands with sudo. While this should work, setting up a **Docker** user is the prefered.

4.1.4 ⚠ Processes that require TrackVis or Diffusion Toolkit are not working

Because of their respective licenses, we could not include **TrackVis** or **Diffusion Toolkit** in our **Docker** image directly. Please download and install them on you host machine using their installation guide. After they have been successfully installed, mount a volume to your **MIRACL Docker** container that contains the binary folder for **TrackVis** and **Diffusion Toolkit** and add the binaries to your $PATH within your **MIRACL Docker** container using the mounted volume.

4.1.5 ⚠ STA workflow fails when trying to create tracts

Make sure that the **TrackVis** and **Diffusion Toolkit** binaries are available to **MIRACL**. See the previous question for details.

4.1.6 ⚠ I need to install or make changes to apps in the MIRACL container but my user is not authorized to do so

The user in the container is the user of your host machine. This is done to avoid issues with the **MIRACL** GUI and X11. If you need to make changes that require sudo privileges, just log out of your container and log back in as root:

```
$ docker exec -it -u root miracl bash
```

After making your changes, log out and log back in with your regular user:

```
$ docker exec -it miracl bash
```

> **Attention:** Always remembert that changes to the container are not persistent. If you need to make permanent changes as sudo, add them to the Dockerfile instead.

4.1.7 ⚠ MIRACL's GUI (miraclGUI) does not work anymore after my ssh connection has been broken

Assuming you logged back in with ssh and are in the container, exit the container and restart it using docker compose down and docker compose up -d.

Shell back into the container and the GUI should work again.

> **Note:** Note that the **Docker Compose** syntax is different if you installed it using the standalone method. **Compose standalone** uses the -compose syntax instead of the current standard syntax compose. The above command would thus be docker-compose up -d when using **Compose standalone**.

4.1.8 ⚠ I do not want to create the image using the provided script

You can build the image yourself, not using the script we provide. However, the build script makes sure that the GUI version of **MIRACL** works with **Docker** and it is therefore recommended to use it. Build the image with:

```
$ docker build -t mgoubran/miracl .
```

Note: Do not forget the . at the end of the command. It is required to point `docker build` to the `Dockerfile`.

To run the container use:

```
$ docker run -it mgoubran/miracl bash
```

Attention: If you make changes in the container that are not stored on a volume, make sure to use the same container the next time you run **MIRACL** as changes made to a container will only apply to this specific container. If you run **MIRACL** again from the same image using `docker run -it mgoubran/miracl bash`, a new container will be created that does not contain the changes you made to the first container.

Warning: The **MIRACL** GUI will be unlikely to work out-of-the-box but you can try the troubleshooting steps in the following section to make it work.

4.1.9 ⚠ I get either or both of the following errors whenever I try to run the GUI from within a Docker container that was build without the provided build script

```
Authorization required, but no authorization protocol specified
qt.qpa.xcb: could not connect to display :1
```

Note: The number for the display could be different in your case

```
qt.qpa.plugin: Could not load the Qt platform plugin "xcb" in "" even though it was␣
→found.
This application failed to start because no Qt platform plugin could be initialized.␣
→Reinstalling the application may fix this problem.

Available platform plugins are: eglfs, linuxfb, minimal, minimalegl, offscreen, vnc,␣
→wayland-egl, wayland, wayland-xcomposite-egl, wayland-xcomposite-glx, webgl, xcb.
```

Exit your running **Docker** container and run the following to mount an X11 socket from the host system in a new **Docker** container:

```
docker run -it -e DISPLAY=$DISPLAY -v /tmp/.X11-unix:/tmp/.X11-unix mgoubran/miracl bash
```

If you still receive the above error, you may have to change your xhost access control. See previous troubleshooting step above.

4.2 Singularity

4.2.1 ⚠ Can I build a Singularity container from the latest MIRACL image on Docker Hub

Absolutely! To do so, however, you will need to grab a development node after logging in to the cluster. If you try pulling from the login node, you will use a ton of memory building the SIF image, and the process will be killed (in other words, it won't work).

```
$ sdev
```

or

```
$ salloc
```

Once you have your node, you can then build the container:

```
$ cd $SCRATCH
$ singularity build miracl_latest.sif docker://mgoubran/miracl:latest
```

4.2.2 ⚠ Processes that require TrackVis or Diffusion Toolkit are not working

Because of their respective licenses, we could not include **TrackVis** or **Diffusion Toolkit** in our **Docker** image directly. Please download and install them on you host machine using their installation guide. After they have been successfully installed, mount a volume to your **MIRACL Docker** container that contains the binary folder for **TrackVis** and **Diffusion Toolkit** and add the binaries to your $PATH within your **MIRACL Docker** container using the mounted volume.

4.2.3 ⚠ I get the following error whenever I try to run the GUI from within the Singularity container on Compute Canada

```
qt.qpa.plugin: Could not load the Qt platform plugin "xcb" in "" even though it was␣
↪found.
This application failed to start because no Qt platform plugin could be initialized.␣
↪Reinstalling the application may fix this problem.

Available platform plugins are: eglfs, linuxfb, minimal, minimalegl, offscreen, vnc,␣
↪wayland-egl, wayland, wayland-xcomposite-egl, wayland-xcomposite-glx, webgl, xcb.
```

We do not recommend trying to make X11 forwarding work directly from the terminal. You should use VNC instead. Follow the instructions here. Once you are connected to your login or compute node with VNC, you will see a desktop environment. Open a terminal there and follow *our tutorials* on how to use **MIRACL** with **Singularity** on clusters.

If you for some reason need to run the **MIRACL** GUI directly in the terminal, using a **Singularity** container and X11, try the following workarounds:

Login Nodes

Exit your **Singularity** container and start a VNC server (for 3600sec or more as required) on your login node:

```
vncserver -MaxConnectionTime 3600
```

The first time the VNC server is started you will prompted for a password (do not leave this blank). Once done, check if a X11 socket is available for your username:

```
ls -la /tmp/.X11-unix/
```

Note: If no socket is available for your username, log out and log back in to your login node

Start another **Singularity** container and try to run `miraclGUI` again from within it.

Compute Nodes

Exit your **Singularity** container and set an environment variable on your allocated compute node:

```
export XDG_RUNTIME_DIR=${SLURM_TMPDIR}
```

Start a VNC server:

```
vncserver
```

Start another **Singularity** container and try to run `miraclGUI` again from within it.

4.3 Local installation

4.3.1 ⚠ I get the following error whenever I try to run the GUI: qt.qpa.plugin: Could not load the Qt platform plugin "xcb" in "{anaconda path}/envs/miracl_merge/lib/python3.7/site-packages/cv2/qt/plugins" even though it was found

If you know the path to the environment name, try running the following line to remove the specific file in question:

```
rm "{path_to_environment_name}/lib/python3.7/site-packages/cv2/qt/plugins/platforms/
↪libqxcb.so"
```
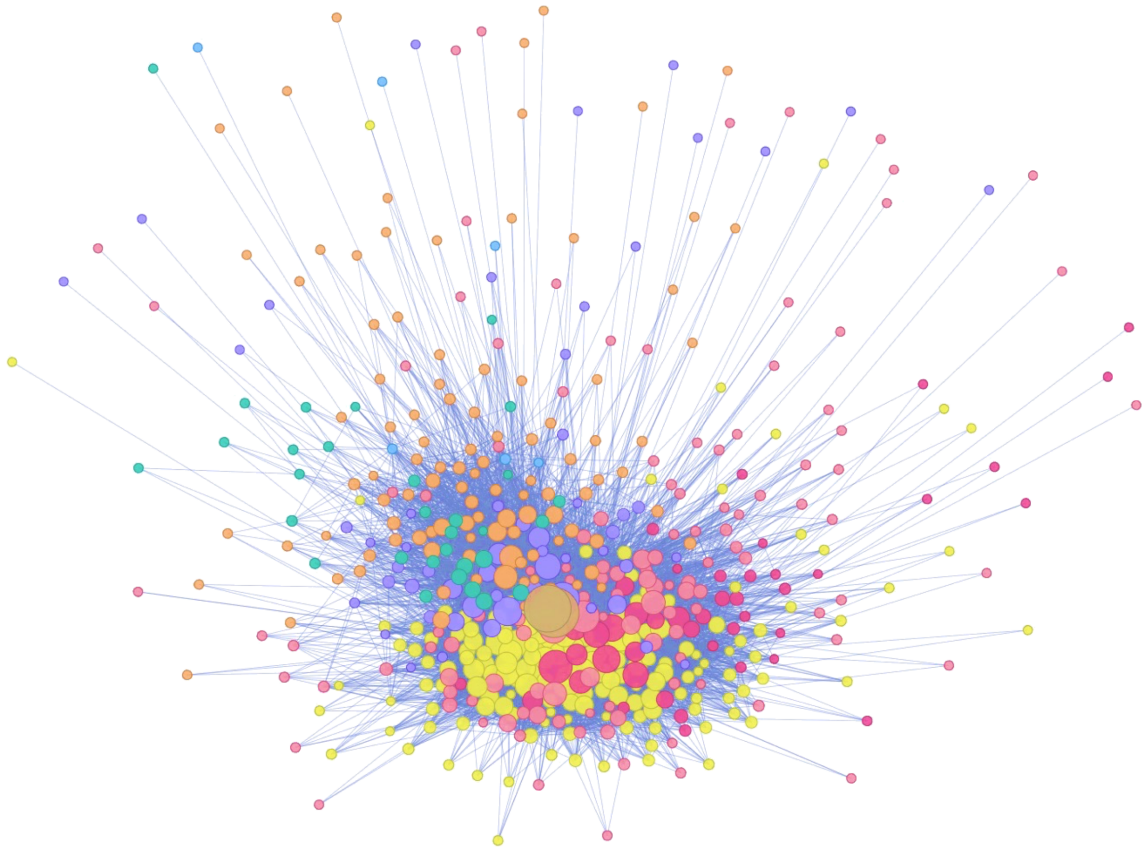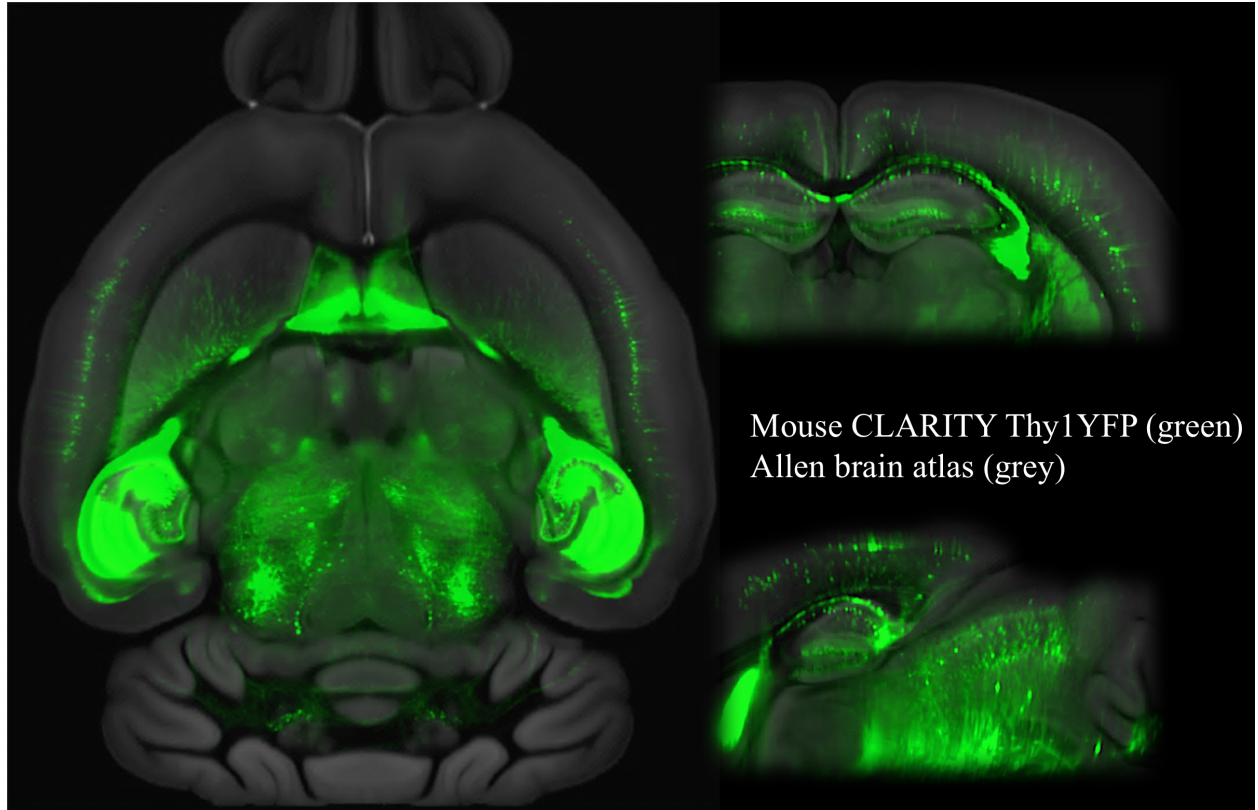
FIVE

GALLERY

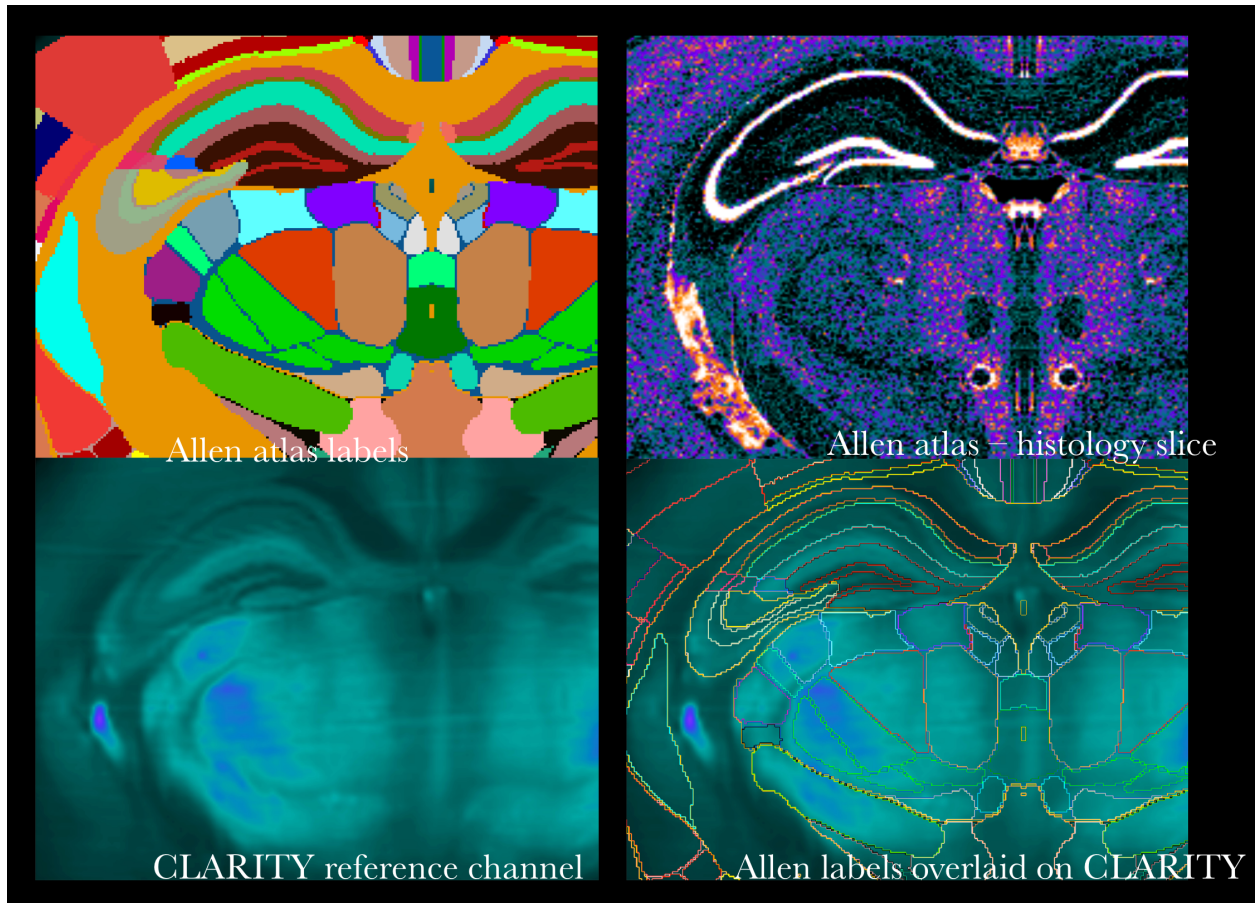Here is some representative work!

5.1 Graphical User Interface (GUI)
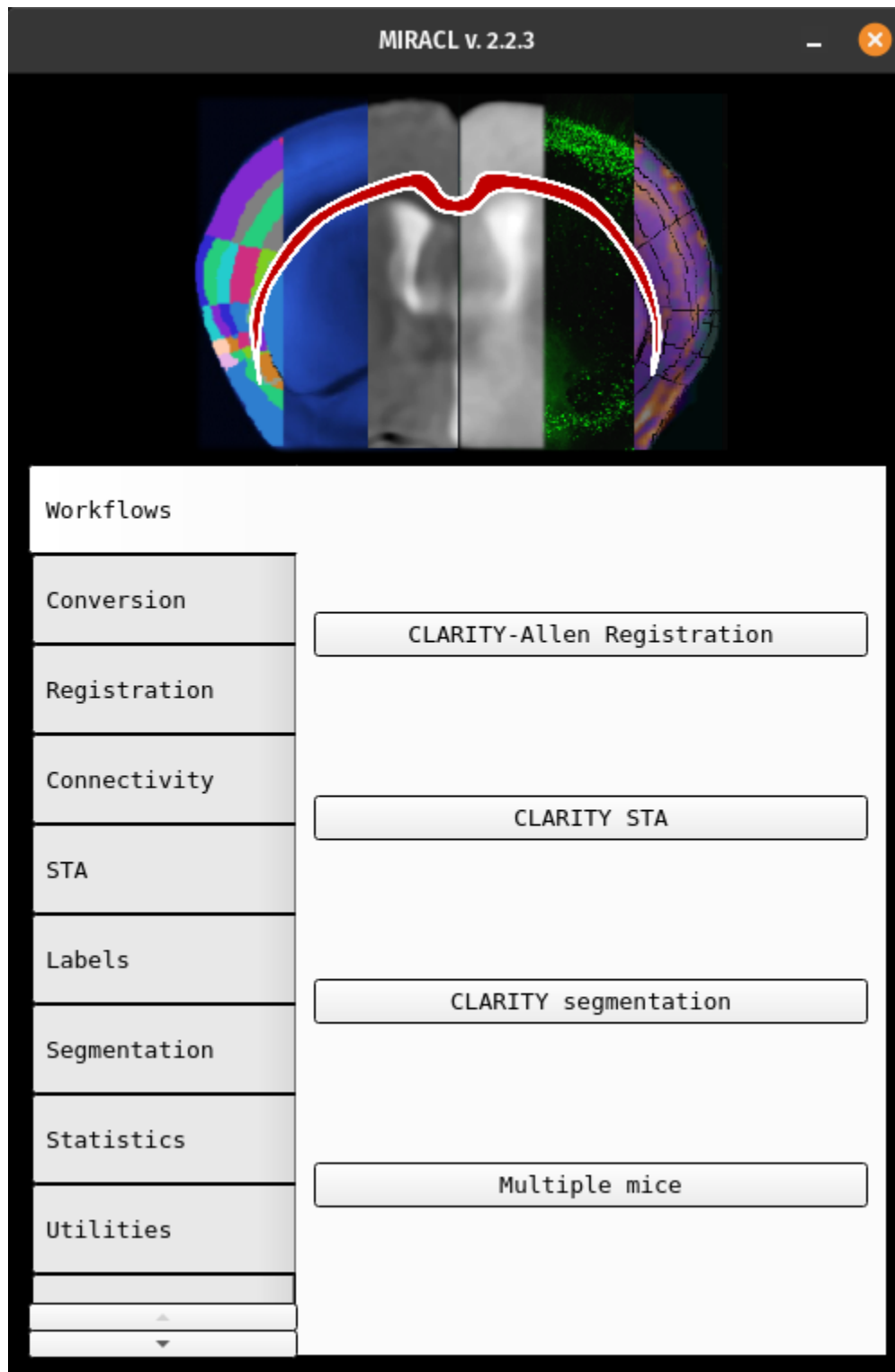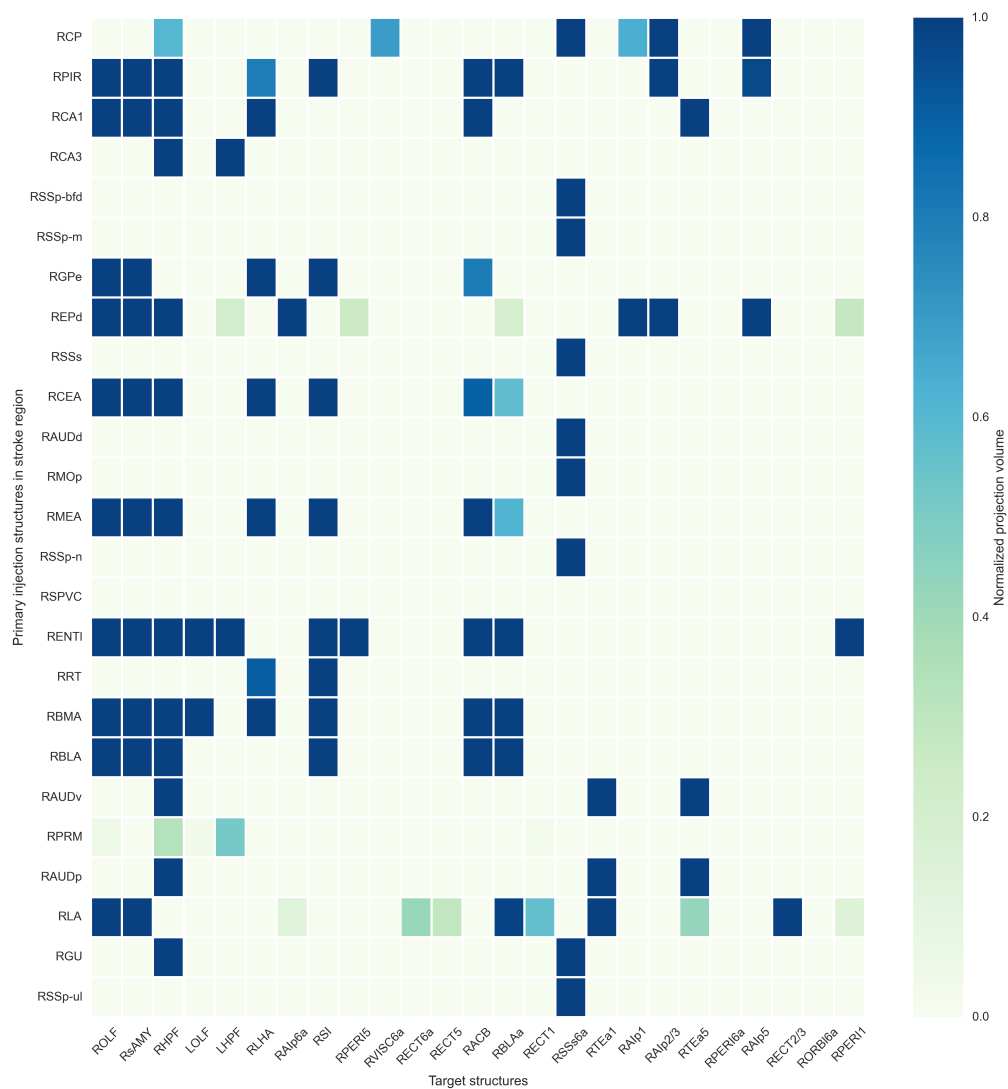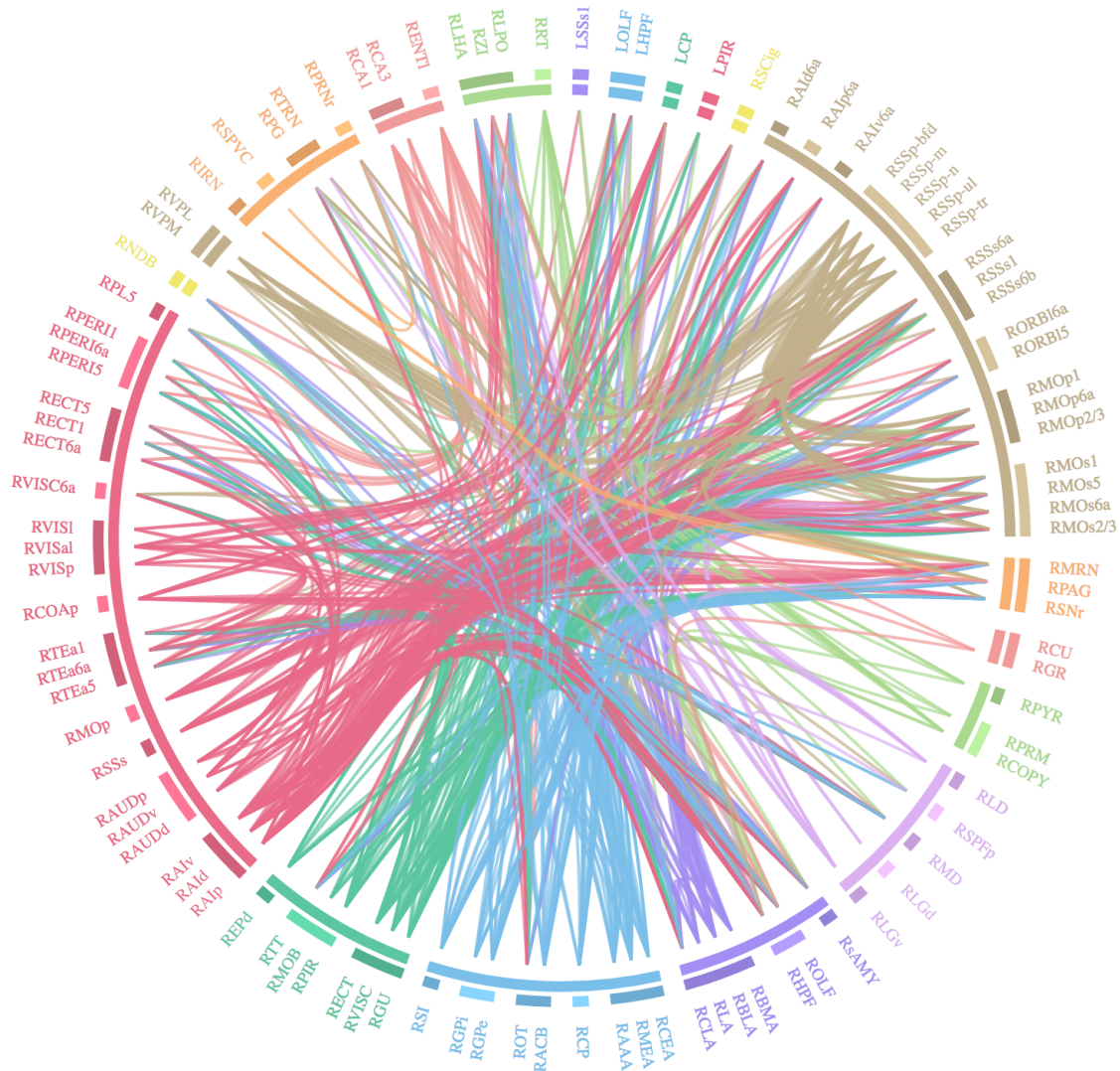
5.2 Brain Graph
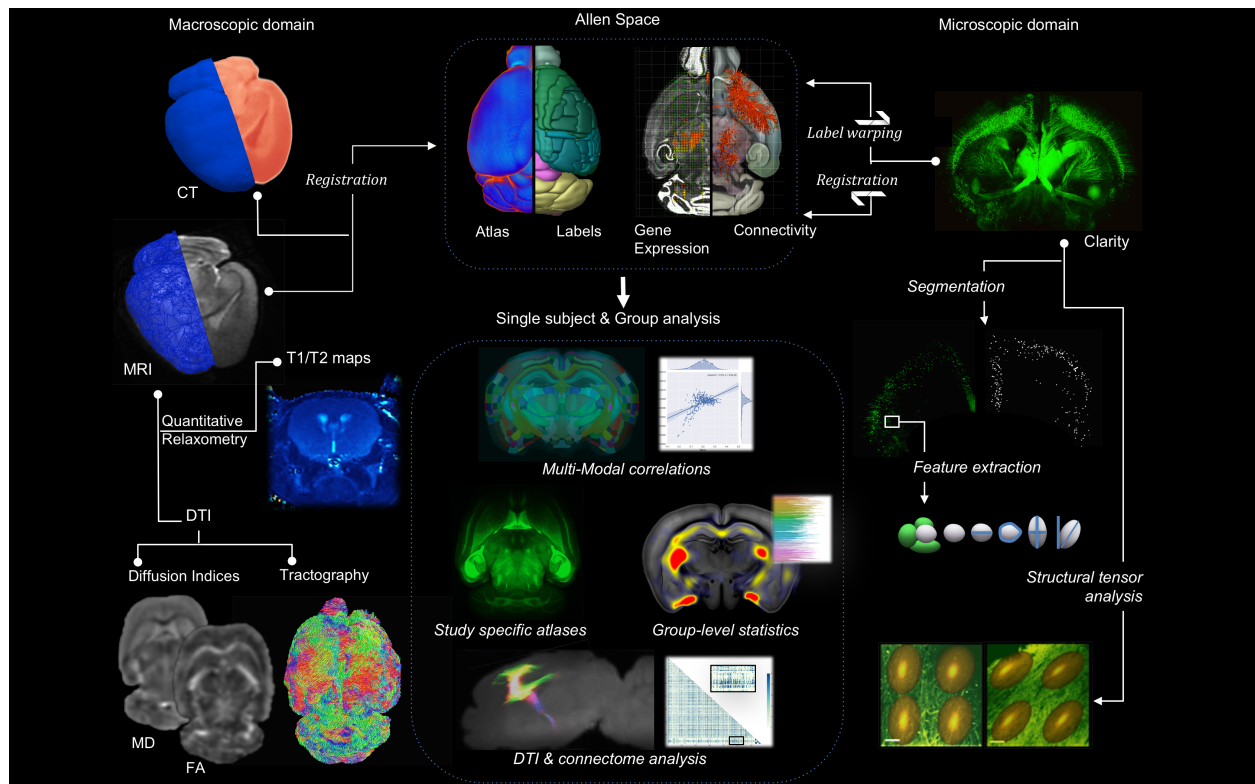
5.3 Clarity Registration

Mouse CLARITY Thy1YFP (green)
Allen brain atlas (grey)

Allen atlas labels

Allen atlas — histology slice

CLARITY reference channel

Allen labels overlaid on CLARITY

Fig. 1: Connectivity matrix heat map with 25 labels

Fig. 2: Connectogram grouped by parent ID with 50 labels

Fig. 3: Density along connectivity graph

5.4 Connectivity

5.5 Pipeline
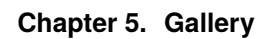
5.6 Registration and Segmentation
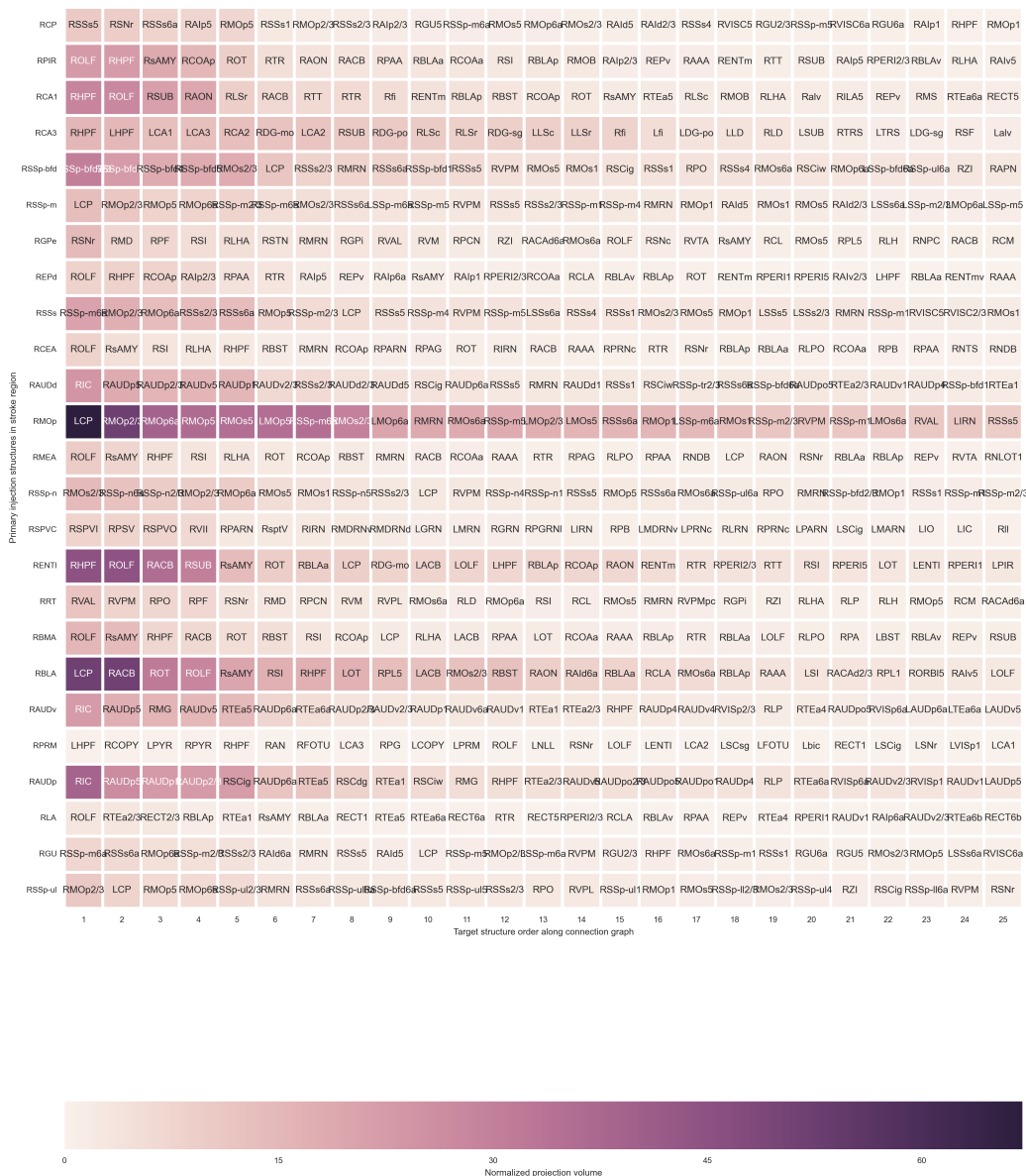
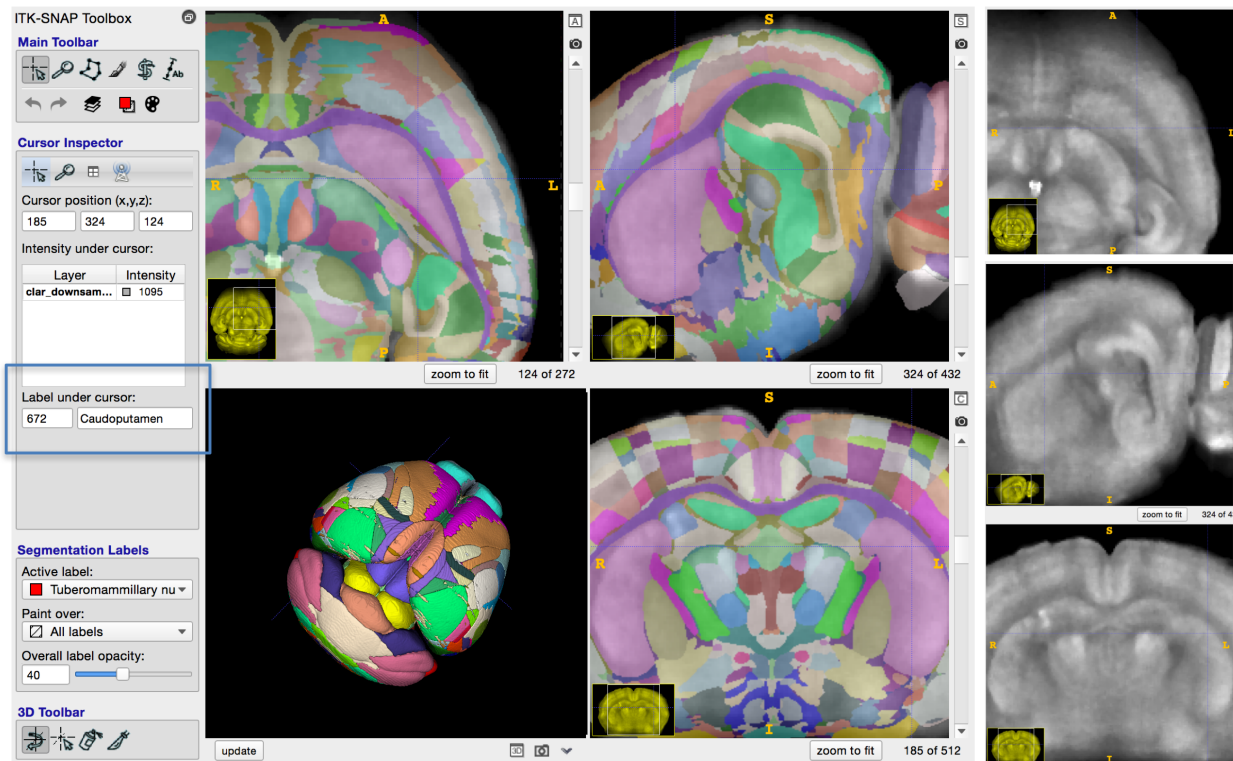Fig. 4: Projection map along graph with 25 labels

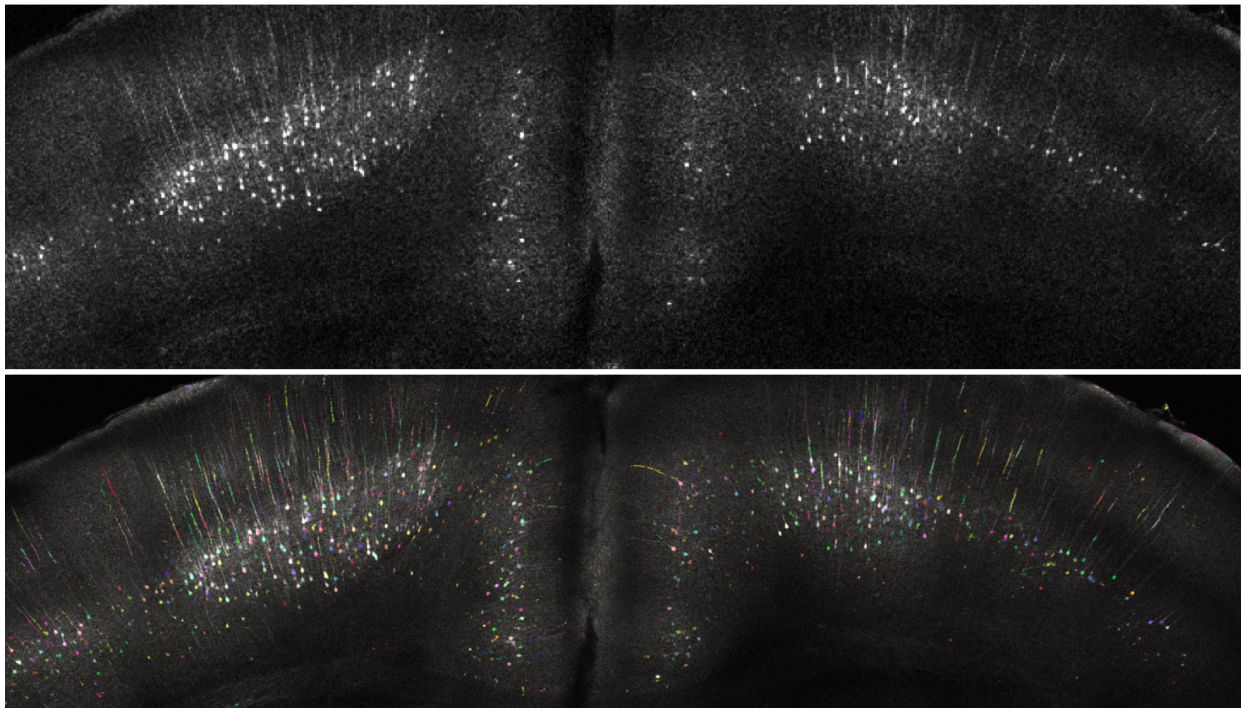Fig. 5: Registration result visualized in ITK-SNAP

Fig. 6: Segmentation
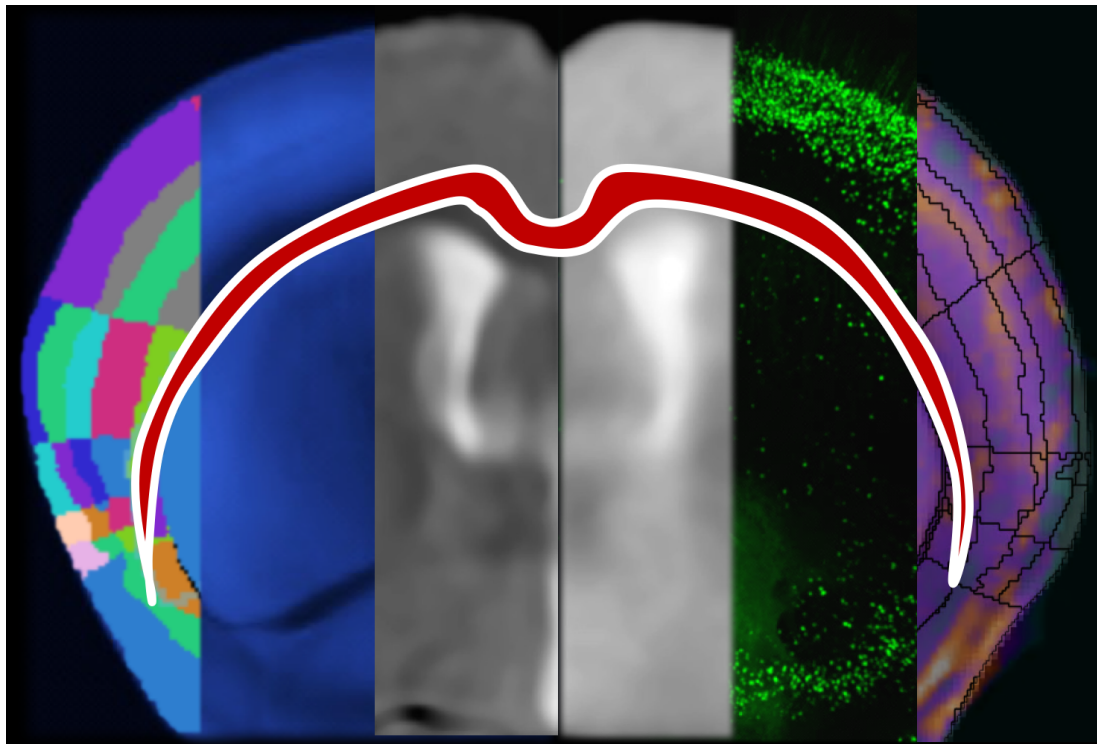
EXAMPLE AND ATLASES DATA

Sample data you can use to test **MIRACL** and the data for the atlases used by **MIRACL** can be found here:

Dropbox Data Link

The `Atlases` folder contains templates, annotations, histology, ontology graph info and LUT/label description of the Allen Reference Atlas (ARA).

The `Data` folder contains test data with example inputs and outputs for the registration and segmentation modules.

For a detailed description and input parameters please check the respective help or *tutorial* of each module.

MIRACL (**M**ulti-modal **I**mage **R**egistration **A**nd **C**onnectivity anaLysis) is a general-purpose, open-source pipeline for automated:

1. Registration of mice CLARITY data to the Allen Reference Atlas

2. Segmentation and feature extraction of mice CLARITY data in 3D (Sparse and nuclear staining)

3. Registration of mice multimodal imaging data (MRI & CT, in-vivo & ex-vivo) to Allen Reference Atlas

4. Tract or label specific connectivity analysis based on the Allen Connectivity Atlas

5. Comparison of diffusion tensort imaging (DTI)/tractography, virus tracing using CLARITY and Allen Connectivity Atlas

6. Statistical analysis of CLARITY and imaging data

7. Atlas generation and Label manipulation

We recommend using MIRACL with the Docker or Singularity containers we provide but it can also be installed locally. See our *installation instructions* for more information.

Copyright © 2023 @ AICONS Lab.

All Rights Reserved.