# Tractography Meta-Pipeline Command Line Tool (TRAMPOLINO) Documentation

***Release 0.1.9***

**Matteo Mancini**

**Apr 05, 2022**

# Contents:

CHAPTER 1

# Installation

## 1.1 Stable release

To install Tractography Meta-Pipeline Command Line Tool (TRAMPOLINO), run this command in your terminal:

```
$ pip install trampolino
```

This is the preferred method to install Tractography Meta-Pipeline Command Line Tool (TRAMPOLINO), as it will always install the most recent stable release. Note that TRAMPOLINO is developed in Python 3, so if your system still has both Python 2 and 3, you probably should be running:

```
$ pip3 install trampolino
```

If you don't have pip installed, this Python installation guide can guide you through the process.

## 1.2 From sources

The sources for Tractography Meta-Pipeline Command Line Tool (TRAMPOLINO) can be downloaded from the Github repo.

You can either clone the public repository:

```
$ git clone git://github.com/matteomancini/trampolino
```

Or download the tarball:

```
$ curl  -OL https://github.com/matteomancini/trampolino/tarball/master
```

Once you have a copy of the source, you can install it with:

```
$ python setup.py install
```

Once again, in case of a system with both Python 2 and 3, the command becomes:

```
$ python3 setup.py install
```

CHAPTER 2

# Usage

TRAMPOLINO is called directly from the command line:

```
trampolino
```

Without any further arguments, this command is equivalent to *trampolino -help* and it shows the general help.

TRAMPOLINO has three subcommand, each of them specific to one of the steps to reconstruct tractography: *recon*, *track* and *filter*. It is possible to show the help for the related subcommand:

```
trampolino recon --help
```

One can provide just one or more of the subcommands depending on the desired results, e.g. just processing the diffusion data or also reconstructing the actual streamlines. In the following paragraphs, some examples are showed for each of the three package interfaces implemented so far. For the sake of showing how detailed a command can be, some examples are quite long. For short-, to-the-point examples, you can have a look at the section about *using the Force!*

# CHAPTER 3

# MRtrix3

Estimation of the ODF using the multi-tissue multi-shell spherical deconvolution approach (mtms-csd):

```
trampolino -n msmt_csd -r example_results recon -i sherbrooke_3shell/dwi.nii.gz -v␣
↪sherbrooke_3shell/bvec.txt -b sherbrooke_3shell/bval.txt mrtrix_msmt_csd
```

Streamlines tracking using the iFOD2 probabilistic algorithm:

```
trampolino -n msmt_csd -r example_results track -o wm.mif -s seed.mif mrtrix_tckgen
```

Both the steps combined:

```
trampolino -n msmt_csd -r example_results recon -i sherbrooke_3shell/dwi.nii.gz -v␣
↪sherbrooke_3shell/bvec.txt -b sherbrooke_3shell/bval.txt mrtrix_msmt_csd track␣
↪mrtrix_tckgen
```

When the steps are combined, there is no need to specify the input of each step: the output from the *recon* step are automatically fed into the *track* step.

The previous steps plus the SIFT filtering:

```
trampolino -n msmt_csd -r example_results recon -i sherbrooke_3shell/dwi.nii.gz -v␣
↪sherbrooke_3shell/bvec.txt -b sherbrooke_3shell/bval.txt mrtrix_msmt_csd track␣
↪mrtrix_tckgen filter mrtrix_tcksift
```

The whole workflow using three angular threshold and two different algorithms (multiple results are generated):

```
trampolino -n msmt_csd -r example_results recon -i sherbrooke_3shell/dwi.nii.gz -v␣
↪sherbrooke_3shell/bvec.txt -b sherbrooke_3shell/bval.txt mrtrix_msmt_csd track --
↪angle 30,45,60 --algorithm iFOD2,SD_Stream mrtrix_tckgen filter mrtrix_tcksift
```

The previous example but doing ensemble tractography over the angles:

```
trampolino -n msmt_csd -r example_results recon -i sherbrooke_3shell/dwi.nii.gz -v␣
↪sherbrooke_3shell/bvec.txt -b sherbrooke_3shell/bval.txt mrtrix_msmt_csd track --
↪angle 30,45,60 --algorithm iFOD2,SD_Stream --ensemble angle mrtrix_tckgen filter␣
↪mrtrix_tcksift
```

(continues on next page)

(continued from previous page)

For the sake of simplicity, the examples for the other software packages directly show the combined commands, but it is in any case possible to run just one of the steps and use the parallel and ensemble features.

# CHAPTER 4

## Diffusion Toolkit

Estimation of the tensor from the diffusion data, streamlines tracking and spline-based filtering:

```
trampolino -n dtk_wf -r dti_results recon -i sherbrooke_3shell/dwi.nii.gz -v -v␣
↪sherbrooke_3shell/bvec.txt -b sherbrooke_3shell/bval.txt dtk_dtirecon track dtk_
↪dtitracker filter dtk_spline
```

# CHAPTER 5

# DSI Studio

Processing the data using GQI and streamline tracking:

```
trampolino -n dsi_wf -r dsi_results recon -i dwi.nii.gz -v bvec.txt -b bval.txt dsi_
↪rec track dsi_trk
```

# CHAPTER 6

# dMRI Trekker

Obtaining a simple tractogram from 10000 seeds using a FOD estimated with MRtrix3:

```
trampolino -n trekker -r trekker_results track -o wm.mif -s mask.mif --opt seed_
↪count:10000 trekker
```

# CHAPTER 7

# TractSeg

Segmenting the corpus callosum with TractSeg given a FOD estimated with MRtrix3:

```
trampolino -n tractseg -r tractseg_results track -o wm.mif -s mask.mif tractseg
```

The same case as before, but including the FOD estimation:

```
trampolino -n tractseg -r tractseg_results recon -i sherbrooke_3shell/dwi.nii.gz -v␣
↪sherbrooke_3shell/bvec.txt -b sherbrooke_3shell/bval.txt mrtrix_msmt_csd track␣
↪tractseg
```

# CHAPTER 8

# Force Mode

If you wish to try out different options for only one of the three stages without bothering too much about the other two or supplying your own data, you might want to use the –force functionality of trampolino. Let's say you just want to try out three different angular thresholds using iFOD2 of MRtrix3, the command you want to use looks as follows:

```
trampolino --force track --angle 30,45,60 mrtrix_tckgen
```

Using the –force flag in your command, trampolino will download an example DWI dataset and produce all required input files, without the need of specifying them yourself. Depending on the interface you specify for the workflow, trampolino will do the required steps using the same interface. In the above case, trampolino will e.g. run the default reconstruction using MRtrix3.

Similar to this, if you only care about e.g. streamline filtering, the following is possible:

```
trampolino --force filter dtk_spline
```

In this case, trampolino would download the example dataset, run the default reconstruction and streamline tractography using the Diffusion Toolkit, and finally execute the filtering options you provided.

This can be a useful tool to rapidly retrieve some sample results for exploration purposes. Consider this example with TractSeg, where a corpus callosum is obtained just launching:

```
trampolino --force track tractseg
```

# CHAPTER 9

# Conversion

TRAMPOLINO provides a conversion subcommand for tractography, so one is able to easily go from TRK to TCK and viceversa. Converting from TRK to TCK is immediate:

```
trampolino convert -t track.trk trk2tck
```

Converting from TCK to TRK requires a reference volume:

```
trampolino convert -t track.tck -r meanb0.nii.gz tck2trk
```

Finally, the conversion subcommand can be concatenated as the others:

```
trampolino track -o wm.mif -s brainmask.mif mrtrix_tckgen convert -r meanb0.nii.gz␣
↪tck2trk
```

# CHAPTER 10

# Containers

It is possible to directly run a given workflow in a container from TRAMPOLINO. This may be desirable in several scenarios, for example:

1. the desired software package is not installed;
2. the software package is not _installable_ (e.g. trekker on macOS);
3. there may be software conflicts on the host machine (!).

To run the workflows in a container, it is necessary to install both Docker (see [these instructions](https://docs.docker.com/get-docker/)) and the Docker API:

```
pip install docker
```

Once you have installed it, you need an image for a suitable container. The one I created (the *Dockerimage* is available in the codebase, folder *containers*) can be pulled directly from DockerHub with:

```
docker pull ingmatman/trampolino
```

Otherwise, you can build it locally:

```
docker build -t ingmatman/trampolino $TRAMPOLINO_PATH/containers
```

Once you have built it, you can run for example:

```
trampolino --container --force -n trekker-docker -r docker_results track trekker
```

This will start the workflow inside a container, and will save the results and the logs in the output folder. To keep temporary files, you can add the option *–keep*. Also to use a custom image (i.e. with a different tag from *ingmatman/trampolino*), you can pass it with the *–image* option. An example of both these options:

```
trampolino --container --name my_image --keep -n msmt_csd -r example_results track -o␣
↪wm.mif -s seed.mif mrtrix_tckgen
```

So far, the *ingmatman/trampolino* includes MRtrix3 (*3.0.0*) and Trekker (*0.7*). More tools are coming soon!

CHAPTER 11

# Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

## 11.1 Types of Contributions

### 11.1.1 Report Bugs

Report bugs at https://github.com/matteomancini/trampolino/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

### 11.1.2 Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with "bug" and "help wanted" is open to whoever wants to implement it.

### 11.1.3 Implement Features

Look through the GitHub issues for features. Anything tagged with "enhancement" and "help wanted" is open to whoever wants to implement it.

### 11.1.4 Write Documentation

Tractography Meta-Pipeline Command Line Tool (TRAMPOLINO) could always use more documentation, whether as part of the official Tractography Meta-Pipeline Command Line Tool (TRAMPOLINO) docs, in docstrings, or even on the web in blog posts, articles, and such.

### 11.1.5 Submit Feedback

The best way to send feedback is to file an issue at https://github.com/matteomancini/trampolino/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

## 11.2 Get Started!

Ready to contribute? Here's how to set up *trampolino* for local development.

1. Fork the *trampolino* repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/trampolino.git
```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

```
$ mkvirtualenv trampolino
$ cd trampolino/
$ python setup.py develop
```

4. Create a branch for local development:

```
$ git checkout -b name-of-your-bugfix-or-feature
```

   Now you can make your changes locally.

5. When you're done making changes, check that your changes pass flake8 and the tests, including testing other Python versions with tox:

```
$ flake8 trampolino tests
$ python setup.py test or py.test
$ tox
```

   To get flake8 and tox, just pip install them into your virtualenv.

6. Commit your changes and push your branch to GitHub:

```
$ git add .
$ git commit -m "Your detailed description of your changes."
$ git push origin name-of-your-bugfix-or-feature
```

7. Submit a pull request through the GitHub website.

## 11.3 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.
3. The pull request should work for Python 2.7, 3.4, 3.5 and 3.6, and for PyPy. Check https://travis-ci.org/matteomancini/trampolino/pull_requests and make sure that the tests pass for all supported Python versions.

## 11.4 Tips

To run a subset of tests:

```
$ py.test tests.test_trampolino
```

## 11.5 Deploying

A reminder for the maintainers on how to deploy. Make sure all your changes are committed (including an entry in HISTORY.rst). Then run:

```
$ bumpversion patch # possible: major / minor / patch
$ git push
$ git push --tags
```

Travis will then deploy to PyPI if tests pass.

CHAPTER 12

# Credits

## 12.1 Development Lead

- Matteo Mancini <ingmatteomancini@gmail.com>

## 12.2 Contributors

- Bastian David

CHAPTER 13

# Indices and tables

- genindex
- modindex
- search