PyNTA Documentation

Release 0.1

Aquiles Carattino

Jun 19, 2019

Contents:

Nanoparticle tracking analysis refers to a technique used for characterizing small objects optically. The base principle is that by following the movement of nanoparticles over time, it is possible to calculate their diffusion properties and thus derive their size.

PyNTA aims at bridging the gap between experiments and results by combining data acquisition and analysis in one simple to use program.

PyNTA is shipped as a package that can be installed into a virtual environment with the use of pip. It can be both triggered with a built in function or can be included into larger projects.

CHAPTER 1

Installing

The best place to look for the code of the program is the repository at https://github.com/nanoepics/pynta. In short, if you want to install PyNTA you can run the following command:

```
pip install git+https://github.com/nanoepics/pynta
```

If you need further assistance with the installation of the code, please check *Installing*.

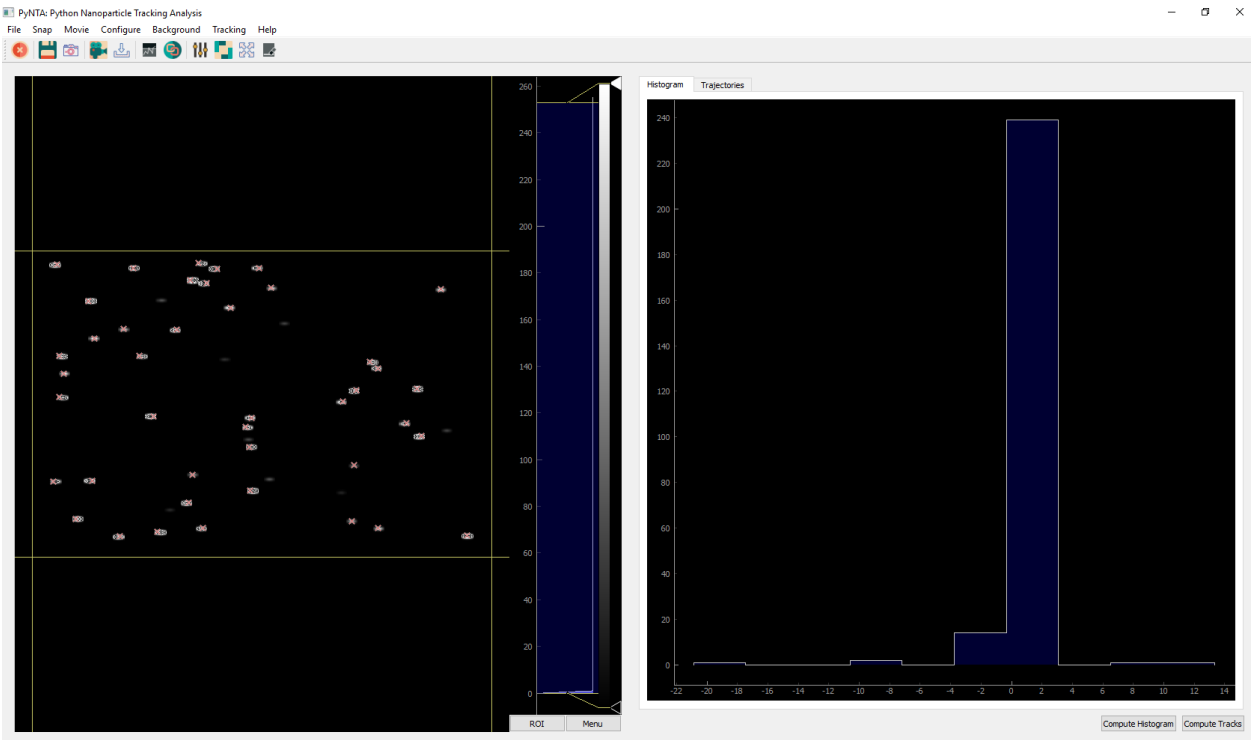
CHAPTER 2

Start the program

After installing, the program can be started from the command line by running the following:

```
python -m pynta -c config.yml
```

Remember that config.yml needs to exist. To create your own configuration file, you can start with the example provided in the examples folder. Once the program starts, it will look like the following:



CHAPTER 3

Contributing to the Program

The program is open source and therefore you can modify it in any way that you see fit. You have to remember that the code was written with a specific experiment in mind and therefore it may not fulfill or the requirements of more advanced imaging software.

However the design of the program is such that would allow its expansion to meet future needs. In case you are wondering how the code can be improved you can start by reading *How to Contribute Code*, or directly submerge yourself in the documentation of the different classes *PyNTA API*.

If you want to start right away to improve the code, you can always look at the *List of Todo's*.

CHAPTER 4

Acknowledgements

This program was developed by Aquiles Carattino with the support of funding from NWO, The Netherlands Scientific Organization, under a Vici grant from Prof. Allard Mosk. This work was carried on at Utrecht University in the months between June 2018 and December 2018.

4.1 Installing

Pynta can be installed directly with pip. The first step is to create a virtual environment on your machine in order to avoid clashes with the versions of the dependencies. Virtual Environments are must-have tool, regardless of what you are doing. You can read a discussion about them directly on Python For The Lab. You can also see how to create a virtual environment *Setting up a Python Virtual Environment*.

To install PyNTA you can run the following command:

```
pip install git+https://github.com/nanoepics/pynta
```

This will get you the latest stable version of Pynta. If you, however, would like to test new features, you can download the development branch of the program:

```
pip install git+https://github.com/nanoepics/pynta@develop
```

Especially if you want to try the development version, you should install it in a virtual environment. We can't guarantee that the development will be future compatible, i.e. that the program stays compatible with itself over time. One of the highest risks is that an upgrade on the development branch may brake your config files or customizations you have done.

Moreover there is the risk of requiring dependencies that are not fully supported or that are later dropped.

4.1.1 Dependencies

By default, when you install PyNTA, the following dependencies will be installed on your computer:

- trackpy

- pyqt5<5.11
- numpy
- pyqtgraph
- pint
- h5py
- pandas
- pyyaml
- pyzmq
- numba

However, not all dependencies are mandatory for the program to work. For instance, if you are not interested in the GUI but are planning to run the program from the command line, you are free to skip PyQt5, Pyqtgraph.

Numba is used because it accelerates the tracking of particles with **trackpy**. But if it is not available on the computer, the program will run anyways.

Trackpy

Trackpy is instrumental for the program to work correctly. This package is able to detect particles on an image based on few parameters but also to link the particles together, building up single-particle traces. PyNTA uses trackpy to perform all the detection and analysis in real time.

One of the constrains of trackpy is that it depends heavily on Pandas, which is a great tool while working in combination with Jupyter notebooks, but is not that great for user interfaces. Which require to transform from Pandas Data Frames to numpy arrays all the time.

PyQt5 and Pyqtgraph

The user interface is built on PyQt5 in combination with PyQtGraph. PyQt5 versions newer than 5.11 fail at installing through the setup process (but they do work if installed directly with pip). If this bug is resolved, the constrain on the version of PyQt to use should be lifted. Moreover, it is desirable to switch to PySide2 as soon as the project is mature.

4.1.2 Operating System Support

PyNTA was tested both on Linux and Windows machines. However, the main environment for PyNTA to run is Windows 10. There are some very fundamental differences on how processes are started between Linux and Windows that have mutual drawbacks. For example, on Windows processes are spawned, meaning that classes are re-imported and not instantiated. Therefore, processes don't start with a shared state. This prevents, for example, to start a new process for a method of a class.

This forced the architecture of the program to rely heavily on functions and not methods, making the code slightly more convoluted than what was desirable. The approach works on Linux also, but the performance may not be optimal.

4.2 The Config File

To start the program, it is necessary to define a configuration file. You can `get a config file here`. However, the best place to find the latest examples of config files is on the Github Repository. The idea behind the config file

is that it makes it transparent both to the end user and to the developer the different settings available throughout the program.

The example config files only show the minimum possible contents. You are free to add as many entries as you would like. However, they are not going to be displayed in the GUI, nor will be used automatically. They will, however, be stored as metadata together with all the files. You could use the config file in order to annotate your experiments, for example.

4.2.1 The Format

The config file is formatted as a YAML file. These files are very easy to transform into python dictionaries and are very easy to type. So, for example, to change how the tracking algorithm works, one would change the following lines:

```
tracking:
  locate:
    diameter: 11  # Diameter of the particles (in pixels) to track, has to be an odd␣
↪number
    invert: False
    minmass: 100
```

Note that for the file to make sense, it has to be indexed with 2 spaces. When reading it, it automatically converts some data types. For example, diameter will be available as `config['tracking']['locate']['diameter']` and will be of type integer. `invert` will be a boolean, etc. If the data type is not clear, the default is a string. So, for example:

```
camera:
  exposure_time: 30ms # Initial exposure time (in ms)
```

Will generate a `config['camera']['exposure_time']` of type string, that will need to be transformed to a quantity later on. Note also that comments are ignored (after the # nothing is read).

4.3 Getting Started

In order to familiarize yourself with the program, the best idea is to start with simulated data. In this way you avoid all the problems arising from interfacing with a real instrument and you can see the limitations of the program. The example config available on the repository is already configured to work with a simulated camera. It is recommended that you copy the contents of the file into a folder on your own computer.

4.3.1 Opening the Program

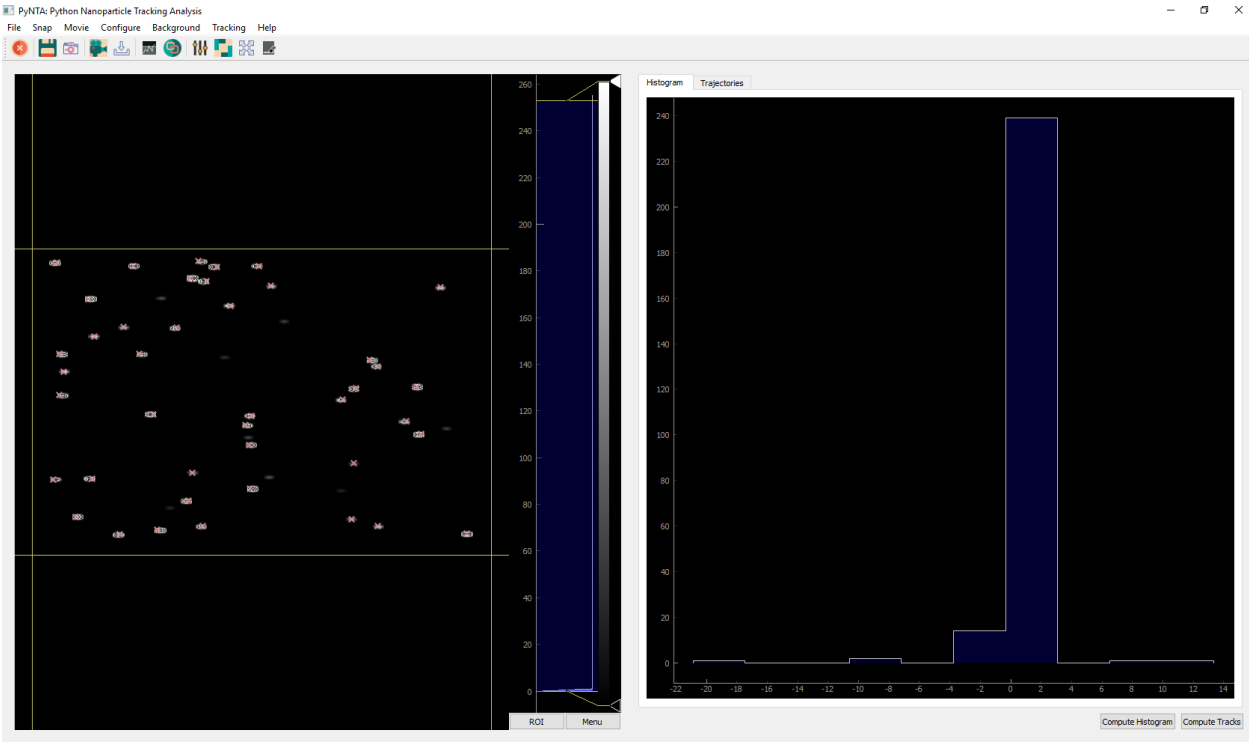
After **installing_** PyNTA, you can trigger it from the command line. We are assuming that you are within the folder that contains the configuration file and that its name is `config.yml`. To start the program, you simply type:

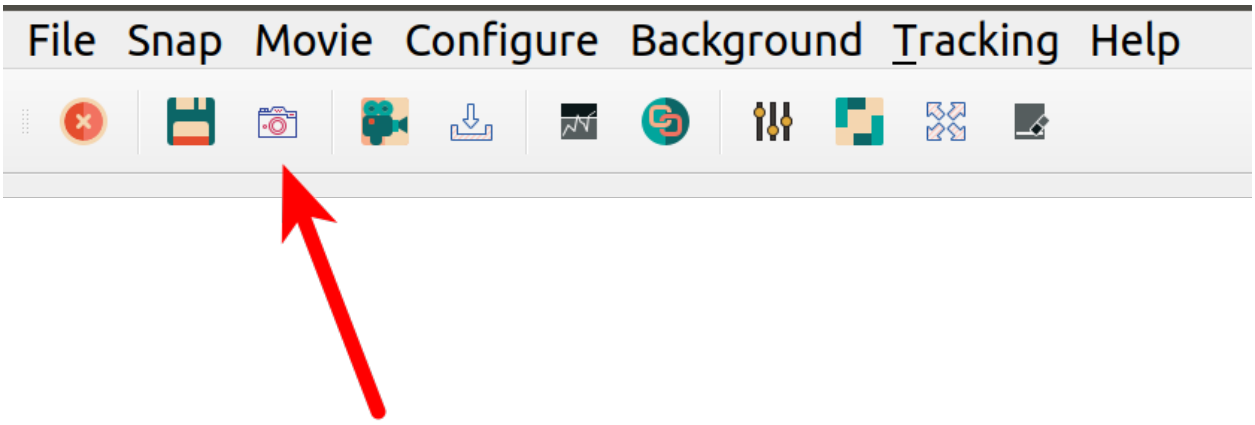
```
python -m pynta -c config.yml
```

After a few moments, a screen like the one below will welcome you to the program.

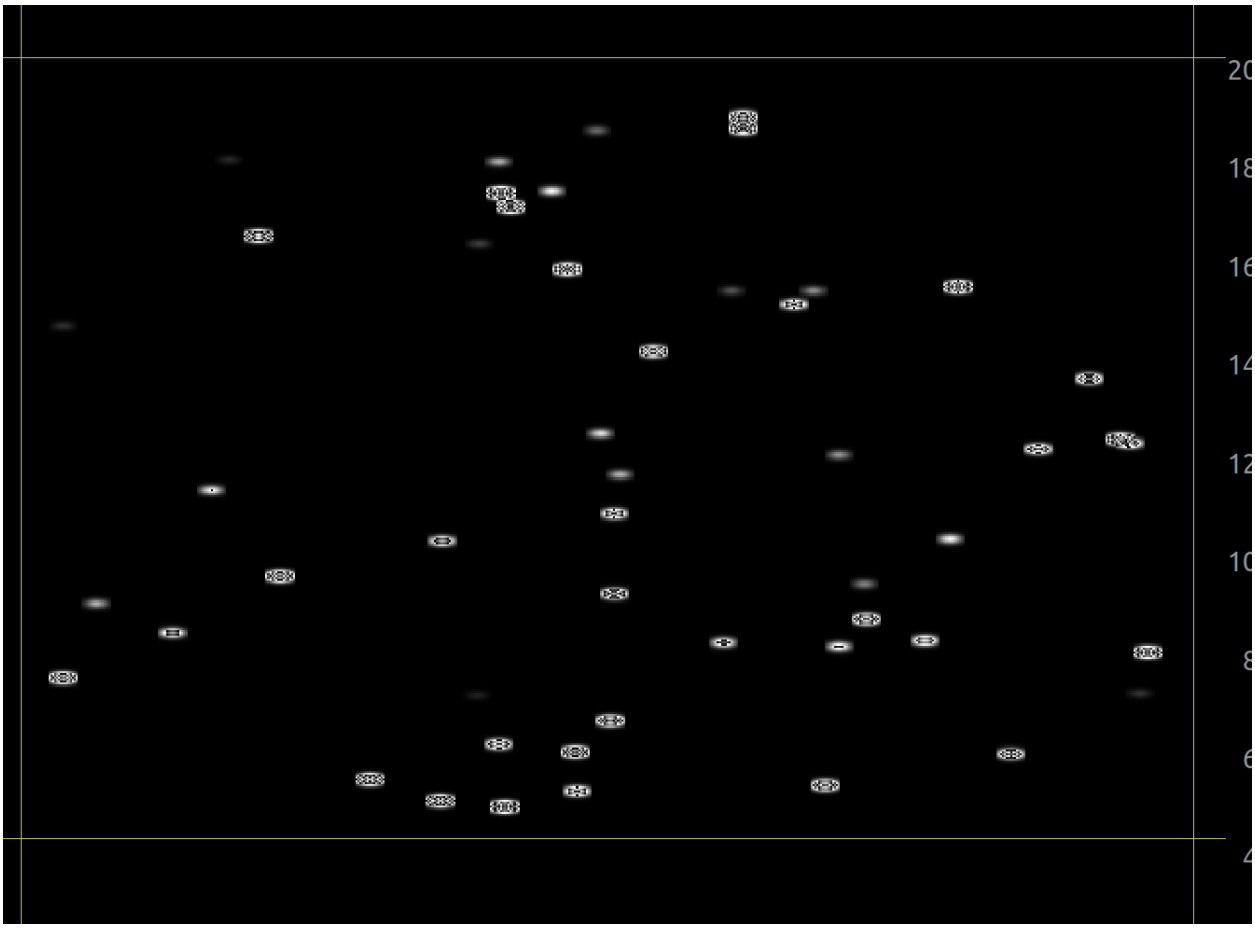
4.3.2 The Tools

Most of the options were designed to be self-explanatory. However it is important to give a short discussion in order to speed the introduction to the tool. After initializing, normally one would like to snap a photo in order to see what is being recorded by the camera. You can achieve it by clicking the button as shown in the image below:


PyNTA: Python Nanoparticle Tracking Analysis
File Snap Movie Configure Background Tracking Help
Histogram
Trajectories
ROI
Menu
Compute Histogram
Compute Tracks


File Snap Movie Configure Background Tracking Help

This will record a single image from the camera and will be displayed on the space right below:



The image can be zoomed-in and out by scrolling with the central wheel of the mouse. Dragging allows to move around the image. In order to return to the full view, it is possible to right-click on the image and select `View All`. The histogram on the right of the image shows the levels for displaying. You can adjust the minimum and maximum as well as the color scale. Right clicking on the image allows you to do an `Auto Range`, i.e. to adjust the levels such that the maximum and minimum correspond to those of the data being displayed.

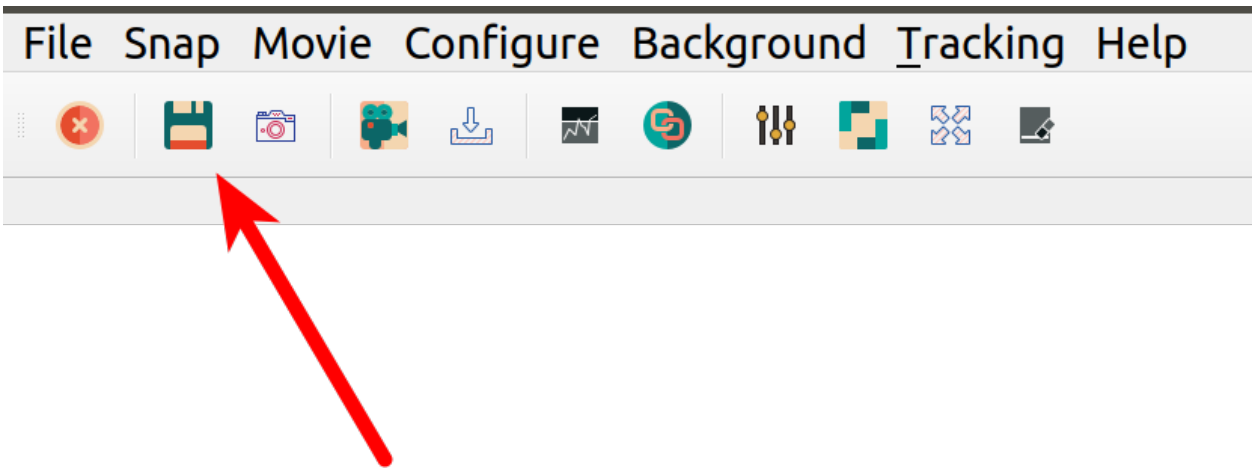
If you want to save the image you can click on the icon for saving, as shown below:

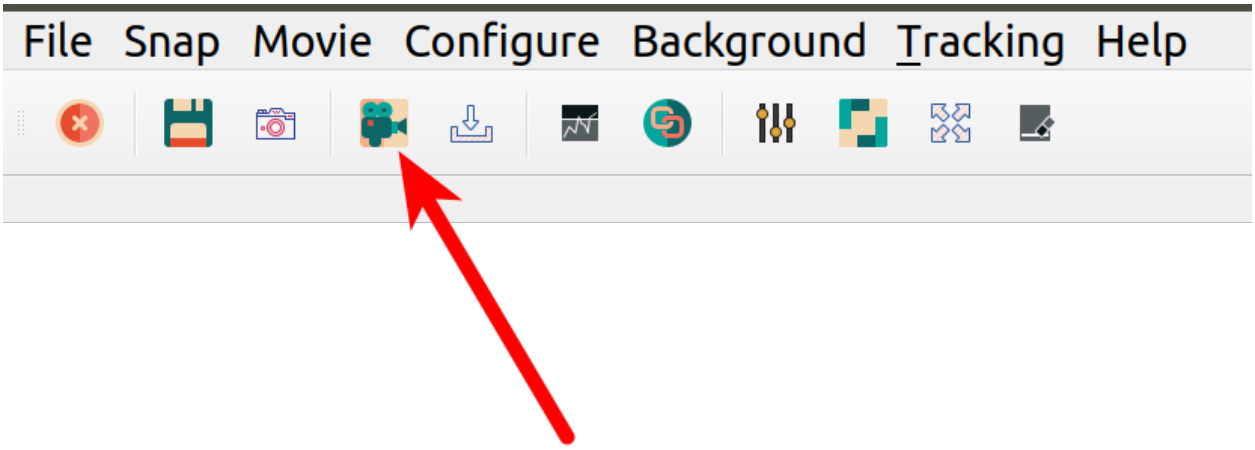
PyNTA also allows you to acquire continuous images, by clicking on the icon highlighted below. The exact behaviour will depend on the camera employed. For example, if a frame-grabber is available, the exact timing between frames can be guaranteed. Cameras without a buffer, however, will have a timing that depends on the computer ability to read from them. The communication with the camera happens in a separate thread, trying to guarantee the maximum reliability of the timing.

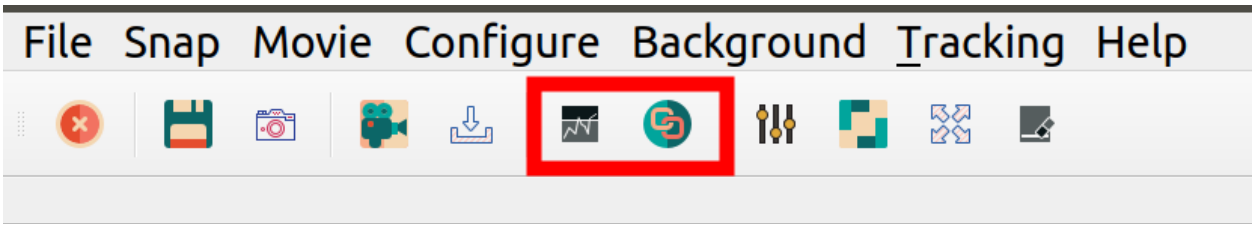
Another feature is the continuous saves option, which is right next to the start movie button. The continuous saves streams all the available frames to a file on the hard rive. The location of the file is determined in the config file or, as we will see later, can be set in the configuration on the User Interface. In case of acquiring at high frame rates, not all frames are displayed to the user, but all of them will be saved.

4.3.3 Tracking and Linking

The feature that really makes PyNTA unique is the ability to identify and track nanoparticles on a video in real time. The procedure for tracking and analysis requires of two steps. First, you have to start identifying the particles, with


File Snap Movie Configure Background Tracking Help


File Snap Movie Configure Background Tracking Help



the button called `start tracking`. You will see red crosses appearing on the particles in the image. It takes a few instants to setup the linking procedure, during which the movie may seem to freeze.

If you are satisfied with how the identification of particles works, you can start linking the positions. Linking is a procedure that identifies whether locations in consecutive frames belong to the same particle or not. This procedure can be computationally expensive and requires fine tuning of the parameters. Linking also happens in a separate process, and in parallel to the acquisition and identification of particles.

4.4 How to Contribute Code

Rules for contributing code

4.5 Setting up a Python Virtual Environment

This guide is thought for users on Windows that want to use virtual environments on their machines.

1. Run:

```
pip.exe install virtualenv
```

   At this point you have a working installation of virtual environment that will allow you to isolate your development from your computer, ensuring no mistakes on versions will happen. Let's create a new working environment called Testing

7. Run:

```
virtualenv.exe Testing
```

   This command will create a folder called Testing, in which all the packages you are going to install are going to be kept.

8. To activate the Virtual Environment, run:

```
.\Testing\Scripts\activate
```

(The `.` at the beginning is very important). If an error happens (most likely) follow the instructions below. Windows has a weird way of handling execution policies and we are going to change that.

Open PowerShell with administrator rights (normally, just right click on it and select run as administrator) Run the following command:

```
Set-ExecutionPolicy RemoteSigned
```

This will allow to run local scripts. Go back to the PowerShell without administrative rights and run again the script activate.

9. When you are inside a virtual environment, you should see the name between `()` appearing at the beginning of the command line.

   Now you are working on a safe development environment. If you run:

   ```
   pip freeze
   ```

   You will see a list of all the packages currently installed in your environment. The list should be empty.

10. To deactivate the virtual environment just run:

    ```
    deactivate
    ```

11. If you run `freeze` again, you will see all the packages installed in the computer:

    ```
    pip freeze
    ```

4.6 PyNTA API

List of methods of PyNTA, for developers

4.7 List of Todo's

CHAPTER 5

Indices and tables

- genindex
- modindex
- search