OpenPTV User Guide

Release 0.0.05

OpenPTV consortium

Jul 05, 2019
| 1  | Introduction          | 3  |
| 2  | Installation instructions | 7  |
| 3  | Tutorials             | 11 |
| 4  | Information for developers | 27 |
| 5  | How to add/fix documentation | 29 |
| 6  | The OpenPTV graphical user interface | 31 |
| 7  | Additional software projects | 33 |
| 8  | Indices and tables    | 35 |
OpenPTV is the abbreviation for the Open Source Particle Tracking Velocimetry consortium. The core of this software is the 3D-PTV software originally developed at ETH Zurich. The consortium of the academic institutions is working on improving the core algorithms, developing a stand-alone library with a simpler and clear API. We also develop the new user interface in Python, that started at the Tel Aviv University. Read more about the consortium on our website http://www.openptv.net

In the future we would like to allow everyone to add their algorithms to the OpenPTV library, named liboptv and develop several interfaces, combining it with the pre- and post-processing routines using Python/NumPy/SciPy/PIL/Matplotlib/etc. See the existing repositories on http://github.com/OpenPTV

1.1 About 3D-PTV measurement method

3D-PTV in a nutshell

1.1.1 Objectives of the 3D-PTV experimental method

We are convinced that the three dimensional tracking method that provides otherwise inaccessible information about the flow can make an impact in various applications, allowing for the researchers and industry to get a deeper insight into their flows. Most of the flows are highly complex and turbulent and only few of them can get a limited low-dimensional or analytical description that explains the different flow phenomena. Experimental research is inevitable in observing the flow and discovering new phenomena, in addition to assisting to explain the old ones.

1.1.2 Introduction

The 3D Particle Tracking Velocimetry (3D-PTV) offers a flexible technique for the determination of velocity fields in flows. It is based on the visualization of a flow with small, neutrally buoyant particles and a stereoscopic recording image sequences of the particles. During 80’s-90’s the successful research work performed by the Institute of Geodesy and Photogrammetry at ETH Zurich led to an operational and reliable measurement tool used in hydrodynamics and space applications. In cooperation with the Institute of Hydromechanics and Water Resources Management at
ETH Zurich further progress has been achieved in the improvement of the existing hard- and software solutions. Regarding the hardware setup the acquisition system used at the ETH Zurich was upgraded from offline to online image digitization.

1.1.3 Data acquisition

- Seed a flow with tracer particles
- Illuminate a 3-D observation volume inside the flow by a pulsed lightsource
- Image the scene by 2 (or rather 3-4) synchronized
- Length of image sequences depending from imaging rate and storage device

The system used at the ETH Zurich was upgraded from offline to online image digitization. In the previous system, the image sequences were firstly recorded on analogue videotapes and digitized afterwards, while in the new system two frame grabbers (Matrox Genesis) are used to provide online digitization and storage. The length of the recorded digital image sequences is nowadays restricted by the storage device capabilities. The data rate for a 60 Hz full-frame camera with a resolution of 640 x 480 pixels is about 19 MB/sec, and hence in an experiment which lasts for 1 minute four cameras deliver a total amount of about 4.5 GB image data.

The Particle Tracking Velocimetry software performs the following tasks:

- Calibration of the multi-camera system (determination of camera exterior and interior orientations, lens distortion and further disturbances, (e.g. Willneff and Maas, 2000) and the exact geometric modelling (“multimedia geometry” - each beam from a particle to the sensor passes the three optical media water, glass, air with different refractive indices, which leads to a twice broken beam).
- Image preprocessing: perform highpass filtering due to non-uniformities in the background illumination
- Detect particles in the images by a modified thresholding operator, localize particles with subpixel accuracy by a centroid operator
- Establish stereoscopic correspondences
- Determine 3-D particle coordinates
1.1.4 Data Processing

- Image preprocessing: perform highpass filtering due to non-uniformities in the background illumination
- Detect particles in the images by a modified thresholding operator, localize particles with subpixel accuracy by a centroid operator
- Establish stereoscopic correspondences
- Determine 3-D particle coordinates
- Storage of all relevant object and image space information
- Perform tracking in 2-D image and 3-D object space

A crucial point is the handling of ambiguities occurring in different steps of the data processing chain:

- Particles may overlap in the images. For that reason a modified thresholding/centroid operator was developed searching for local maxima in the images and dividing particle images at local minima under certain conditions.
- Due to the fact that particle images cannot be distinguished by features like size, shape or color, the only criterion for the establishment of stereoscopic correspondences is the epipolar line. Ambiguities occur when multiple candidates are found in a search area defined by the epipolar line. These ambiguities can only be solved if a third (or even a fourth) camera is being used.
- Ambiguities may also occur in the tracking procedure. Criteria like local correlation and smoothness of the velocity field are employed to solve these criteria.

Another important issue is an accurate calibration of the system (determination of camera exterior and interior orientations, lens distortion and further disturbances) and the exact geometric modelling (“multimedia geometry” - each beam from a particle to the sensor passes the three optical media water, glass, air with different refractive indices, which leads to a twice broken beam).

1.1.5 Potential

- Truely 3-D technique: all three components of the velocity field are determined in a 3-D observation volume
- Delivers 3-D vector field for Eulerian analysis plus 3-D trajectories for Lagrangian analysis
- A system based on 4 CCD progressive scan cameras (digitized to 640 x 480 pixels) is capable of tracking more than 1000 particles
- The relative accuracy of the velocity vectors is ~ 1:4000 of the field of view

Real time image processing schemes

- Real time image compression using a customized FPGA design,
- Real time image processing using a on-camera FPGA,
1.1.6 Collaboration

- Institute of Environmental Engineering, ETH Zurich
- Institute of Photogrammetry, ETH Zurich
- Turbulence Structure Laboratory, Tel Aviv University
- Technical University Eindhoven, Applied Physics
- Riso National Laboratory
- International Collaboration for Turbulence Research, ICTR
- COST action “Particles in Turbulence”

We organize the PTV benchmarking (open, free and user-friendly, we’ll publish only what you want to be published) in order to make our algorithms validated versus each other and improve our particle tracking abilities worldwide. Write to ‘Alex’ if you want to join with your own version of particle tracking software or with your data test case (e.g. that you find difficult to track or to improve).

1.1.7 See also

- Particle Tracking Velocimetry on Wikipedia

1.1.8 References

- Kreizer Mark, Ratner David and Alex Liberzon Real-time image processing for particle tracking velocimetry, Experiments in Fluids, Volume 48, Issue 1, pp.105-110, ‘http://adsabs.harvard.edu/abs/2010ExFl...48..105K’
2.1 Introduction

The OpenPTV contains of:

1. Core library written in C, called liboptv
2. Python/Cython bindings, shipped together with the liboptv

The Python bindings allow the easy access to the C library. There are two Python GUI packages that are built around the Python bindings to allow the end-user to use it in a more intuitive way:

1. Python 3 with PyQt4 GUIs and command line scripts from Yosef Meller called The Particle Bureau of Investigation or pbi
2. Python 3 based GUI (using TraitsUI, Enthought Chaco, etc.) called PyPTV

2.1.1 The overview

1. if you plan to use C/C++/Fortran/etc. - compile the liboptv from source using cmake and testing it using libcheck library (see below instructions for Linux and Mac OS X).
2. if you plan to use it only from Python (either through pbi command line approach or using pyptv GUI) then you can save time on installing liboptv by pip (see below) or compiling through Python bindings and testing it from Python.

2.1.2 liboptv - a library of the OpenPTV algorithms

This is a library - you can build it and link to it in your own project, e.g. calling functions from your own GUI or command-line software. When the package is installed correctly, you can reference it in your code by including files from the optv directory under the standard include path. For example:

```c
#include <optv/tracking_frame_buf.h>
```
To build your program you also link it with `liboptv`. On `gcc`, one adds the flag `-loptv` to the command line. Other compilers and IDEs have their own instructions for adding libraries; consult your IDE/compiler manual for the details.

The library is using Check framework for the unit tests and Cmake project for the build. We recommend installing both software packages, however it is not obligatory, you may skip the relevant parts if you’re not going to develop or test the library.

### 2.1.3 Installation

If you want to try the software but not really to get a development version, then you can use one of the two options: Virtual Machine appliance for VirtualBox software or Docker:

#### 2.1.4 Virtual Machine

Ubuntu 18.04 with `pyptv`, see here: https://github.com/alexlib/pyptv/wiki/Getting-started-using-VirtualBox-Linux-(Ubuntu-18.04)-image

#### 2.1.5 Try Docker

This method requires to install first: Docker and X Server (for GUI); then, either pull the ready image or build locally the docker image. This installation works on Windows (tested on Win 10), Mac OS X (tested on Mojave 10.14.2) and Linux.

Please follow the detailed instructions on PyPTV Dockerfiles

#### 2.1.6 Building development version and installation

If you want your own copy of the software, compiled and tested on your platform, then you first need to install Python 3. Note that we work on the Python 3 version but it is not ready yet.

### 2.2 Recommended Python distributions

Instead of using the Python 3 that comes with your system we recommend to install one of these distributions - later it will minimize the issues of cross-compiled packages.

1. Anaconda Python distribution for Windows or Linux
2. Canopy (Enthought Python Distribution) for Mac OS X.

#### 2.2.1 Installation of `liboptv` and Python bindings

Install Python 3 (Anaconda or anything else):

```
pip install optv
```

Now you can proceed to Python shell and try:

```
import optv
```
2.2.2 Installation of Python GUI (including \textit{liboptv}) called PyPTV

Install Python 3 (Anaconda or anything else):

\begin{verbatim}
python -m pip install --upgrade pip
pip install numpy
pip install pyptv --index-url https://pypi.fury.io/pyptv --extra-index-url https://pypi.org/simple
\end{verbatim}

2.2.3 Use our test case folder to see PyPTV in action like in video tutorials

Download and run the test case:

\begin{verbatim}
git clone --depth 1 --single-branch https://github.com/OpenPTV/test_cavity.git
pyptv test_cavity
\end{verbatim}

2.3 If nothing works, where I can get help?

Send your build logs, description of the problem and details of the operating system, Python version, etc. to our Google group or forum: <https://groups.google.com/forum/#!forum/openptv>
3.1 Detailed documentation

- Thesis of Jochen Willneff that explains most of the theory behind the 3D-PTV [Link]
- Multi-plane calibration manual by Lorenzo del Castello, TU/e. [Link]
- Dumbbell (or wand) calibration, dumbbell_calibration
- PTV file system description: briefly about rt_is and ptv_is files <PTV_files> or detailed [Link]
- Technical aspects of 3D PTV [Link]
- Python Bindings to PTV library [Link]
- Tutorial written by Hristo Goumnerov [Link]
- Report and tutorial by Dominik Bauer, *Particle Tracking Velocimetry with OpenPTV*, [Link]

3.2 How to use the custom image segmentation and target files

Sometimes there is a need for a sophisticated particle (or any other object) identification that is not possible using standard OpenPTV tools (highpass with edge detection and particle center identification, see `liboptv` for details). One of such examples is our dumbbell calibration - one needs to identify two (and only two) bright spots of relatively large objects that could not be implemented using OpenPTV. Therefore, we implement the object identification in Python and write per each image the `_target` file in the same folder as the images (i.e. if `/img/img.10001` we add `/img/img.10001_target`). Then we tell the OpenPTV-Python not to use the `liboptv`, but instead use the existing `_targets` files. There is a checkbox to be checked in in the `Main Parameters`. 
3.2.1 Step 1

Run your image processing routine, e.g. https://github.com/alexlib/alexlib_openptv_post_processing/blob/master/Python/dumbbell.ipynb and save the identified objects into the files, see an example of the writing subroutine:

```python
def write_dumbbells(filename, centers, radii, indices): counter = 0 with open(filename,’w’) as f:
    f.write(‘%dn’ % 2) for idx in indices:
        x, y = centers[idx] r = radii[idx] f.write(‘%4d %9.4f %9.4f %5d %5d %5d %5d %5dn’ % (counter,y,x,r**2,2*r,2*r,r**2*255,-1)) counter+=1
```

3.2.2 Step 2

Check in the `use existing_target_files` in the `Main Parameters`

All the rest should work as usual: `Sequence -> Tracking`.

Please, see our screencasts for the quick overview and the step-by-step tutorial:

- Tutorial 1: <http://youtu.be/S2fY5WFsFwo>
- Tutorial 2: <http://www.youtube.com/watch?v=_JxFxwVDSt0>
- Tutorial 3: <http://www.youtube.com/watch?v=z1eqFL5JIJc>

If you want to practice, install the software and download the necessary files from downloads page.

3.3 Tutorial

New experiment:

- For a new set of experiments open a new folder. The file should contain the following sub-folders: cal: for calibration, parameters, img and res. For example, a clean file for example (copy it and rename the file) is in ptv/fresh_test.
3.3.1 Calibration files:

- The `cal` folder contains: calibration images, one for each camera, e.g. `cam1.tif`, `cam2.tif` and so on, orientation files `cam1_ori`, `cam2_ori`..., and a `calblock.txt` file that contains the x,y,z coordinates of the calibration target.

- `ori` files: camera’s orientation files:

```
10.0 10.0 300.0
0.01 0.05 0.0002
1.0 0.0 0.0
0.0 1.0 0.0
0.0 0.0 1.0
0.0 0.0
80.0
0.0001 0.0001 100.0000
```

- First row: x,y,z of the camera sensor from the calibration target origin (0,0,0)
- Second row: the angles [radians], the first is around x axis, then y axis and the third is the angle of rotation around z direction which coincides with the imaging axis of the camera (the line that connects the sensor and the target)
- The next three rows is the rotation matrix
- Next 2 parameters are the `xp`, `yp` positions of the pinhole in respect to the image center in millimeters. if the camera imaging axis is at 90 deg. to the sensor, then `xp=yp=0.0`.
- Next parameter is the back-focal distance, typically called `f`. For example, if we have a ratio of world image to chip image of 500 mm to 65 mm (384 pixels is therefore corresponding to 17 microns), e.g. 1:8. The distance from lens to calibration target is about 800 mm. Hence the focal distance is about 100 mm.
- Last row with the 3 parameters is the the position of the glass in respect to to the origin and in the coordinate system of the calibration target (x is typically from left to right, y is from bottom to top and z is by definition the positive direction looking at the camera. so if the glass wall is perpendicular to the imaging axis and parallel to the calibration target, and the distance in water is about 100 mm the last row is 0.0 0.0 100.0. Since division by zero is not recommended we sugest to use a very tiny deviation from 0.0, e.g. 0.0001

Calibration best practice:

In the first run- choose reasonable parameters according to the cameras positions in the experiment.

- Gain 4 calibration pictures, one for each camera, and copy them to the cal file.
- right click on the current run. choose calibration parameters:
  1. Image data:

Fill in the name of the four calibration pictures ,four orientation data pictures and file of coordinates on plate.
2. Calibration data detection:
Different parameters in order to detect the dots on the calibration target.

3. Manual pre-orientation:
Fill in the numbers of four points on the calibration target. The numbers should be set as chosen in manual orientation.

4. Calibration orientation parameters:
The lens distortion is modeled with up to five parameters :k1,k2,k3+ p1,p2
Affin transformation: scx, she
Principle distance: xp, yp

In the first calibration process don’t mark those parameters. After establishing the calibration, the different parameters can be marked in order to improve the calibration.

• In the upper toolbar choose: calibration and create calibration
• load/show images*: shows the calibration images

\textit{detection}: detect the calibration dots on the calibration image. check that all the dots were identified correctly and marked in blue, and that there aren’t any extra dots.

mark the four points from the manual pre-orientation in each camera and press manual orient. This creates the man_ori.dat. Next time, skip this stage and press detection and then orient with file.

\textit{show initial guess}: The yellow dots show where the dots from the calibration plane would end up on your images if the initial guess would be correct.

If the yellow dots aren’t in the right location, change the ori files - edit ori files and press show initial guess again to see the change, do it until the yellow and blue dots match.

Check that the position of each camera according to the ori files is also reasonable according to the cameras position in reality.

\textit{sort grid}: situates all the dots in their positions. Check that all dots were found and marked correctly.

\textit{orientation}: creates the orientation.

In order to improve the orientation: mark some of the Calibration orientation parameters and press orientation again.

\textbf{3.3.2 Dumbbell calibration}

Sometimes it is inconvenient to position a calibration target. Either because there is something in the way, or because it is cumbersome to get the entire target again out of the observation domain. It would be much easier to move a simple object randomly around the observation domain and from this perform the calibration.

This is what \textbf{Dumbbell calibration} is doing. The simple object is a dumbbell with two points separated at a known distance. A very rough initial guess is sufficient to solve the correspondence problem for only two particles per image. In other words, the tolerable epipolar band width is very large: large enough to also find the correspondence for a very rough calibration, but small enough so as not to mix up the two points. From there on, calibration optimizes
the distances by which the epipolar lines miss each other, while maintaining the detected distance of the dumbbell points.

Unlike previous calibration approaches, Dumbbell calibration uses all camera views simultaneously.

Required input

![Diagonal rendering of user interface for OpenPTV User Guide, Release 0.0.05](image)

Somehow, an object with two well visible points has to be moved through the observation domain and recorder. The dumbbells points should be separated by roughly a third of the observation scale.

Note that the accuracy by which these dumbbell points can be determined in 2d, also defines the possible accuracy in 3d.

Processing:

- Copy at least 500 images of the dumbbell (for each camera) as a tiff file to a new file Prepare target files using matlab code: `tau\dumbbell\_detection\_db\_v3b`. Every target file should contain only 2 points.

- Right click on the current run: choose main parameters.

Main parameters:

write the name of the first dumbbell image, and the name of the calibration images you want to use.

![Main Parameters for OpenPTV User Guide, Release 0.0.05](image)

Particle recognition: * since there are ready target files, mark `use existing\_target\_files`. 
Sequence processing:

Fill in the numbers of the first and last picture in the sequence processing, and the base name for every camera.

Criteria for correspondences:

min corr for ratio nx: min corr for ratio ny: min corr for ratio npix: sum of gv: min for weighted correlation: tol band: The number that defines the distance from the epipolar line to the possible candidate [mm].
Processing of a single time step:

- In the upper toolbar choose: start and then pre-tracking, image coordinate, after that the two points of the dumbbell are detected. Then choose pre-tacking, correspondence. This establish correspondences between the detected dumbbell from one camera to all other cameras.

- You can press one point of the dumbbell in each camera and to see the epipolar lines.

- The processing of a single time step is necessary to adjust parameters like grey value thresholds or tolerance to the epipolar line.
• In the upper toolbar choose: sequence, sequence without display
• In the upper toolbar choose: tracking, detected particles. Then tracking, tracking without display and then show trajectory.
• Right click on the current run. choose calibration parameters:
  1. Dumbbell calibration parameters:
**Eps [mm]**: It is the tolerable bandwidth by which epipolar lines are allowed to miss each other during calibration. should be the same number as the tol. band in **Criteria for correspondences**. **Dumbbell scale [mm]**: distance between the dumbbell points. It is quite important Since the algorithm optimizes two targets, the epipolar mismatch and the scale of the dumbbell particle pair. **Gradient descent factor**: if everything would be linear then a factor of 1 would converge after one step. Generally one is a bit instable though, so a more careful, but slow, value is 0.5. **Weight for dumbbell penalty**: this is the relative weight that is given to the dumbbell scale penalty. With one it is equally bad to have dumbbell scale of only 24mm and to have epipolar mismatch of 1mm. After rough converge this value can be reduced to 0.01-0.2, since it is difficult to precisely even measure this scale.

**Step size through sequence**: it is step size. It could be different then 1 when the dumbbell recording is very long with successive images that are almost identical, then step size of 10 or so might be more appropriate.

In the upper toolbar choose : calibration and create calibration. choose orient with dumbbell.

1. **Shaking calibration**:

2. **Processing of a single time step**

2. **Main parameters**:
Write the name of the first image, and the name of the calibration images you want to use.

1. Particle recognition:

Don’t mark use existing_target_files. fill the particle recognition parameters in order to find the particles.

- Press start in the upper toolbar. the four picture images from main parameters, general will appear.
• Under Pretracking the processing of a single time step regularly starts with the application of a highpass filtering (Highpass). After that the particles are detected (Image Coord) and the position of each particle is determined with a weighted grey value operator. The next step is to establish correspondences between the detected particles from one camera to all other cameras (Correspondences).
The processing of a single time step is necessary to adjust parameters like grey value thresholds or tolerance to the epipolar line.

1. Sequence:

   After having optimized the parameters for a single time step the processing of the whole image sequence can be performed under Sequence.

   • Under main parameters, Sequence processing. Fill in the numbers of the first and last picture in the sequence processing, and the base name for every camera.

   • In the upper toolbar choose sequence with or without display of the currently processed image data. It is not advisable to use the display option when long image sequences are processed. The display of detected particle positions and the established links can be very time consuming.

   • For each time step the detected image coordinates and the 3D coordinates are written to files, which are later used as input data for the Tracking procedure.

1. Tracking:

   1. Tracking parameters:

      Before the tracking can be performed several parameters defining the velocity, acceleration and direction divergence of the particles have to be set in the submenu Tracking Parameters. The flag ‘Add new particles position’ is essential to benefit from the capabilities of the enhanced method. To derive a velocity field from the observed flow.
1. Tracking, Detected Particles displays the detected particles from the sequence processing.

1. Choose tracking, tracking without display. Again it is not advisable to use the display option if long sequences are processed. The tracking procedure allows bidirectional tracking.

2. Tracking, show Trajectories displays the reconstructed trajectories in all image display windows.
OpenPTV need developers. Your support, code and contribution is very welcome and we are grateful you can provide some. Please send us an email to openptv@googlegroups.com to get started, or for any kind of information.

We use Git for development version control, and we have our main repository on Github.

## 4.1 Development workflow

This is absolutely not a comprehensive guide of git development, and it is only an indication of our workflow.

1. Download and install `git`. Instruction can be found [here](#).
2. Set up a github account.
3. Clone OpenPTV repositories using:

   ```bash
   git clone http://github.com/openptv/openptv.git
   ```

4. create a branch `new_feature` either in `liboptv` where you implement your new feature.
5. Fix, change, implement, document code, …
6. From time to time fetch and merge your master branch with that of the main repository.
7. Be sure that everything is ok and works in your branch.
8. Merge your master branch with your `new_feature` branch.
9. Be sure that everything is now ok and works in your master branch.
10. Send a pull request.
11. Create another branch for a new feature.
4.2 Programming languages

As a general rule, we use ANSI C for the liboptv library and Python for the interface. You are welcome to use Python for the core algorithms as well if it does not make any difference with code speed. In those situations where Python speed is the bottleneck, we have some possibilities, depending on your skills and background. If something has to be written from scratch use the first language from the following which you are comfortable with: Cython, C, C++, Fortran. If you have existing, debugged, tested code that you would like to share, then no problem. We accept it, whichever language may be written in!

4.3 Things OpenPTV currently needs, (in order of importance)

1. Move all the core algorithms into liboptv, clean and tested
2. Documentation
3. Cython wrappers for C algorithms, see pybind directory.
4. Flow field filtering and validation functions, see post-ptv repository
5. Better graphical user interface design, e.g. Qt, QML, …
How to add/fix documentation

We decided to use Github to host our documentation and Sphinx to generate it. Sphinx allows us to create automatic docs in HTML, LaTeX, or PDF from the docstrings of Python and C code. In addition, it uses a relatively simple ASCII test format (reST) that can be easily edited on any platform, yet create a good documentation website.

The source files of the documentation are in the openptv-python/docs/source directory, and the respective images are in docs/images directory.

If you want to add/fix documentation, then:

1. fork the openptv-python repository
2. add the document using reST http://sphinx-doc.org/rest.html#lists-and-quote-like-blocks
3. add images to images directory and downloads to downloads directory

If you wish to see the result in HTML, then

1. download and install Sphinx http://sphinx-doc.org/latest/install.html
2. run ‘‘ make html ‘‘ from openptv-python/docs directory to generate your local copy of the documentation. Use your browser to see ../..//docs/html/index.html

For example:
The result may look like:

5. When the documentation is ready - please submit your pull request and the group will review the submission.

Eventually, using the same setup we will regenerate the HTML and push it to the documentation repository under http://www.openptv.net/docs (see for example http://alexlib.github.io/docs)
CHAPTER 6

The OpenPTV graphical user interface

• Python Bindings to PTV library [Link]
• Tutorial written by Hristo Goumnerov [Link]
Additional software projects

The command line version of the 3D-PTV is the only derivative that is distributed under the same license. Other packages will be slowly adopted by the 3D-PTV project but these are developed by the independent authors. Please, pay attention to their license files

- Streaks tracking software by Matthias Machacek
- Command line version of the 3D-PTV software from the University of Plymouth
- Post-processing software package by Beat Luethi
- Real Time Particle Tracking Velocimetry using FPGA on camera particle identification
- Two-frame tracking from University of Rome, written by Luca Shindler

API:

{% if not READTHEDOCS %}
code
{% endif %}

{% endif %} api_reference downloads_page
CHAPTER 8

Indices and tables

- genindex
- modindex
- search