INSTITUT NATIONAL
DE L'INFORMATION
GÉOGRAPHIQUE
ET FORESTIÈRE

Python API for MMVII

IGN - 2024

JMM

Introduction

MMVII is easily scriptable thanks to its command line interface.

How to read/write MMVII files for a custom usage?

How to use MMVII classes and functions?

You can add features to MMVII by modifying its sources (the doc helps!):

- it's in C++
- fork or pushing right to MMVII repository
- quality and reasonable follow-up should be ensured
- adding commands for some very specific cases makes MMVII more complex for all users

You can make your own C++ project using the *libP2007.a* library. . .

Standalone C++: source (read Ori)


```cpp
#include <MMVII_PCSens.h>
namespace MMVII { void CloseRandom(); }
int main()
{
  MMVII::cMMVII_Appli::InitMMVIIDirs(
    std::string(getenv("HOME"))+"/micmac/MMVII/" );
  MMVII::InitStandAloneAppli("mini2007");
  std::string oriPath =
    "Ori-PerspCentral-IMGP4168.JPG.xml";
  MMVII::cSensorCamPC *aCam;
  aCam = MMVII::cSensorCamPC::FromFile(oriPath);
  std::cout<<"Center: "<<aCam->Center()<<"\n";
  delete aCam;
  MMVII::CloseRandom();
  return 0;
}
```

Standalone C++: CMakeLists


```
cmake_minimum_required(VERSION 3.15)
project(mini2007 VERSION 0.1.0)

set(CMAKE_CXX_STANDARD 17)
set(CMAKE_CXX_STANDARD_REQUIRED ON)
set(MICMAC_PATH $ENV{HOME}/micmac)
set(MMVII_SOURCE_DIR $ENV{HOME}/micmac/MMVII)
set(mmv2_include_dir "${MMVII_SOURCE_DIR}/include")
set(mmv2_external_include_dir
    "${MMVII_SOURCE_DIR}/ExternalInclude")
set(EIGEN3_INCLUDE_PATH
    "${mmv2_external_include_dir}/eigen-3.4.0")
```

Standalone C++: CMakeLists


```
add_executable(${CMAKE_PROJECT_NAME} main.cpp)

include_directories(${mmv2_include_dir}
    ${mmv2_external_include_dir}
    ${EIGEN3_INCLUDE_PATH})
target_link_libraries(${PROJECT_NAME}
    ${MICMAC_PATH}/MMVII/bin/libP2007.a)
target_link_libraries(${PROJECT_NAME}
    ${MICMAC_PATH}/lib/libelise.a)
target_link_libraries(${PROJECT_NAME}
    ${MICMAC_PATH}/lib/libANN.a)
target_link_libraries(${PROJECT_NAME}
    pthread X11 stdc++fs -fopenmp)
```

. . . or use the Python API!

MMVII Python API (aka *apib11*) is based on *pybind11* (https://github.com/pybind/pybind11):

> *pybind11 is a lightweight header-only library that exposes C++ types in Python and vice versa, mainly to create Python bindings of existing C++ code.*

A selection of MMVII classes and functions is made usable in Python, with some adjustments:

- differences between C++ and Python synthax (e.g.: no overloading, no templates. . .)
- simplification if possible
- pythonization:
  - define *__repr__()* etc.
  - automatic conversion from lists or *np.array* into MMVII points objects etc.
- memory management: *return value policy*

Other tools

Other tools to generate a Python API:

- *SWIG*: universal, can make APIs for many languages, but uses a specific synthax and not easy to use with modern C++. Used in an unofficial Python API for MM3D.
- *Boost.Python*: close to *pybind11*, but depends on *Boost*...

Why pybind11?

- rather simple C++ syntax
- good documentation
- home-made automatic integration of *Doxygen* comments into Python doc

Documentation

- this presentation
- *MMVII/apib11/README.md*
- MMVII documentation chapter 17
- examples in *MMVII/apib11/examples*

Compilation

Sources

All sources are in *MMVII/apib11/*.

The main files are:

- *py_MMVII.cpp* / *py_MMVII.h*: initialization, closing and error handling, calling all the other files
- *MMVII.py*: Python-side initialization
- *makedoc.py*: automatic C++ *Doxygen* comments conversion into Python doc
- *setup.py*: description of the MMVII module and its compilation

Sources

Some MMVII C++ class bindings:

- *py_MMVII_Matrix.cpp*, *py_MMVII_Geom3D.cpp*: matrix, 3d rotation and isometry
- *py_MMVII_Images.cpp*, *py_MMVII_Image2D.cpp*: data from images
- *py_MMVII_MeasuresIm.cpp*: 2D and 3D measures, sets of measures
- *py_MMVII_PCSens.cpp*, *py_MMVII_Mappings.cpp*: cameras and mappings

Building

The build system is based on a *makefile* calling *Setuptools*.

For now, it only works on GNU/Linux.

Extract from *MMVII/apib11/README.md*:

```
Dependencies
    sudo apt install python3-pip doxygen
    pip3 install pybind11 wheel

First, compile MMv1 and MMv2.
Then, in 'apib11' directory:
    make

Installation:
    make install
```

Usage

Installation

The *wheel* file *dist/MMVII-*.whl*, created at compilation, can be distributed to machines with the same OS, architecture, python version...

- it contains all the necessary files to run the module: MMVII does not have to be installed on the machine to use the python module.
- it can be installed with:

```
pip3 install MMVII-*.whl
```

Import


```
>>> import MMVII
MMVII path: /home/Toto/.local/MMVII/MMVII
MMVII initialized.
>>> MMVII.
MMVII.AimeDescriptor(      MMVII.Mes1GCP(
MMVII.AimePCAR(            MMVII.MesIm1Pt(
MMVII.Box2dr(              MMVII.PerspCamIntrCalib(
MMVII.Box3di(              MMVII.Rect1(
MMVII.Box3dr(              MMVII.Rect2(
MMVII.DataIm2Df(           MMVII.Rect3(
MMVII.DataIm2Di(           MMVII.Rotation3D(
MMVII.DataIm2Dr(           MMVII.SensorCamPC(
MMVII.DataIm2Duc(          MMVII.Set2D3D(
...
```

Example

Let's read a 2D measurements file, correct the image coordinates from distortion and export them.

The data is in:

```
path = 'MMVII/MMVII-TestDir/' \
       'Input/Saisies-MMV1/MMVII-PhgrProj/'
```

- the 2D measurements file:

PointsMeasure/ Saisies_MMVII/MesIm-IMGP4167.JPG.xml

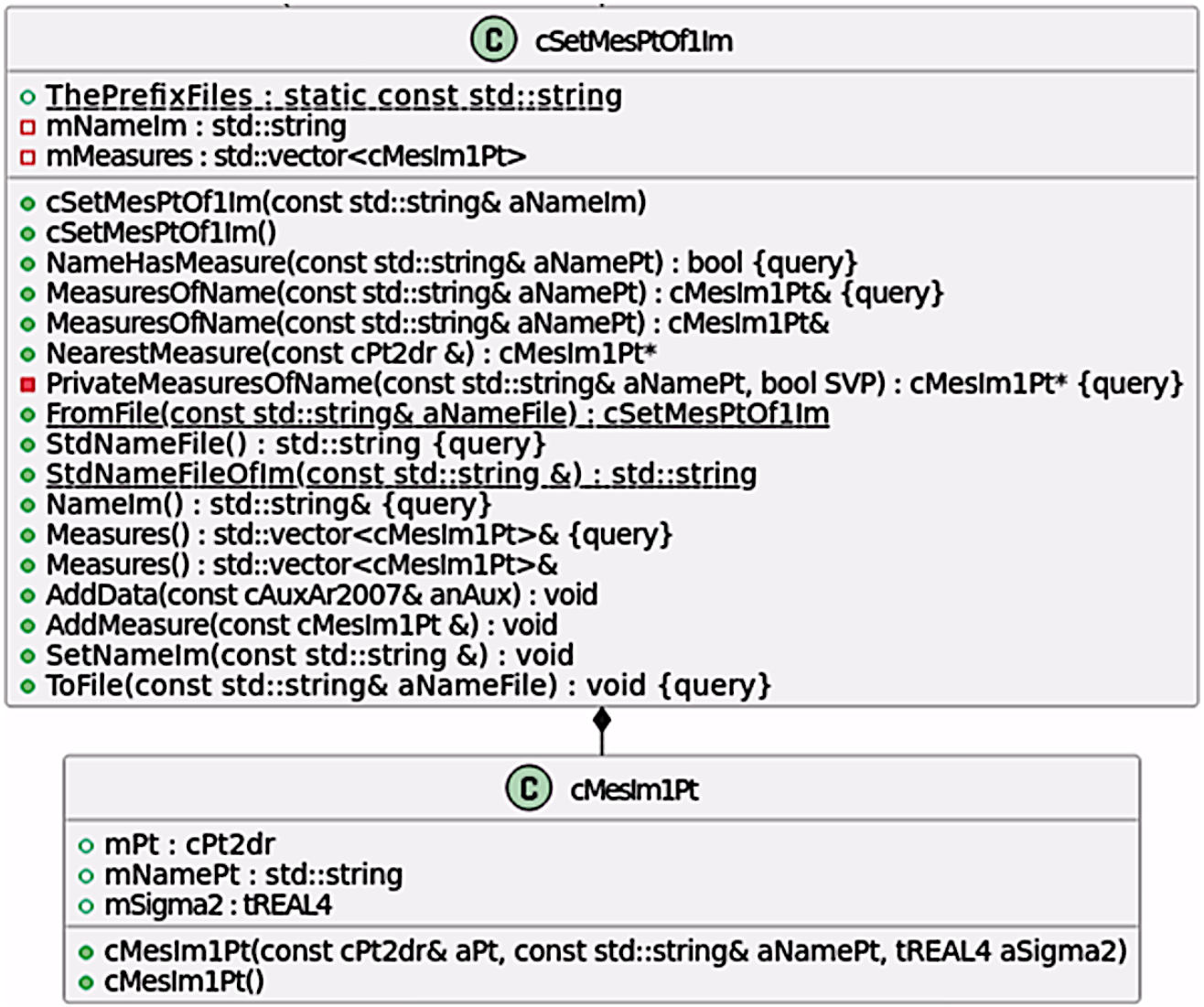
- the calibration file:

Ori/toto/ Calib-PerspCentral-Foc-28000_Cam-PENTAX_K5.xml

2D measurements file


```
<Root>
    <Type>"MMVII_Serialization"</Type>
    <Version>"0.0.0"</Version>
    <Data>
        <SetMesIm>
            <NameIm>"IMGP4167.JPG"</NameIm>
            <Measures>
            <el>
               <Name>"Stone-6"</Name>
               <Pt>644.2863 232.6833</Pt>
               <Sigma2>1 0 1</Sigma2>
            </el>
            [...]
            </Measures>
        </SetMesIm>
    </Data>
</Root>
```

C++ classes diagram



Reading the file


```
pt2dSet = MMVII.SetMesPtOf1Im.fromFile(path
              +'PointsMeasure/Saisies_MMVII/'
              +'MesIm-IMGP4167.JPG.xml')
```

```
>>> pt2dSet
SetMesPtOf1Im MesIm-IMGP4167.JPG.xml
Stone-6 644.28631 232.68332
Stone-7 1265.3681 735.26883
Grille 322.0761 766.81187
```

```
>>> dir(pt2dSet)
['AddMeasure', '__class__', [...], '__str__',
    '__subclasshook__', 'fromFile', 'measures',
    'measuresOfName', 'nameHasMeasure', 'nameIm',
    'nearestMeasure', 'stdNameFile', 'toFile']
```

Python integration


```
>>> type(pt2dSet.measures())
<class 'list'>
>>> type(pt2dSet.measures()[0])
<class '_MMVII.MesIm1Pt'>
>>> type(pt2dSet.measures()[0].pt)
<class 'numpy.ndarray'>
```

```
>>> for mes in pt2dSet.measures():
...     print(mes.namePt, mes.pt)
...
Stone-6 [644.286308   232.68331645]
Stone-7 [1265.36807144  735.2688294 ]
Grille [322.07609729 766.81186618]
```

Errors handling


```
>>> MMVII.SetMesPtOf1Im.fromFile('xxx.xml')


############### Python API error handler ######

Level=[UserEr:OpenFile]
Mes=[Cannot open file : xxx.xml in mode read]
Traceback (most recent call last):
    File "<stdin>", line 1, in <module>
RuntimeError: UserEr:OpenFile Cannot open file :
    xxx.xml in mode read
>>>
```

Errors handling


```
>>> try:
...     tmp = MMVII.SetMesPtOf1Im.fromFile('no.xml')
...     print('read OK')
... except:
...     tmp = 79
...     print('error when reading')
...

################ Python API error handler ######

Level=[UserEr:OpenFile]
Mes=[Cannot open file : error.xml in mode read]
error when reading
>>> tmp
79
```

Calibration

Get the perspective camera internal calibration object:

```
pcIntrCalib = MMVII.PerspCamIntrCalib.fromFile(path
 +'Ori/toto/'
 +'Calib-PerspCentral-Foc-28000_Cam-PENTAX_K5.xml')
```

```
>>> dir(pcIntrCalib)
[[...], 'dir_Dist', 'dir_DistInvertible',
    'f', 'fromFile', 'infoParam', 'invProjIsDef',
    'inv_Proj', 'mapPProj2Im', 'name', 'pp',
    'szPix', 'toFile', 'value', 'values']

>>> pcIntrCalib.pp
array([856.86700874, 577.61365093])
>>> pcIntrCalib.f
2112.2179520972604
```

Mappings

Mapping between photogrammetric/PP and pixel/image frames:

$$Q_{Im} = PP + F * Q_{PP}$$

```
pp2i = pcIntrCalib.mapPProj2Im()

i2pp = pp2i.mapInverse()
```

```
>>> pt = pcIntrCalib.pp
>>> print(pt, ' -> ', i2pp.value(pt))
[856.8670 577.6136]  ->  [0. 0.]
```

```
>>> pt = pcIntrCalib.pp + [1000, 100]
>>> print(pt, ' -> ', i2pp.value(pt))
[1856.8670  677.6136]  ->  [0.4734 0.0473 ]
```

Distorsion

Distorsion is also a mapping:

```
dist = pcIntrCalib.dir_Dist()
```

But not easily invertible:

```
>>> type(pcIntrCalib.dir_Dist())
<class '_MMVII.DataMapping2D'>

>>> type(pcIntrCalib.dir_DistInvertible())
<class '_MMVII.DataInvertibleMapping2D'>
```

```
inv_dist = MMVII.DataInvertOfMapping2D(
                  pcIntrCalib.dir_DistInvertible())
```

Do not use *pcIntrCalib.dir_DistInvertible()* directly!

```
>>> for mes in pt2dSet.measures():
...     pt = mes.pt
...     print('Pt', mes.namePt, 'im:', pt)
...     print(' -> central with disto: ',
            i2pp.value(pt))
...     print(' -> central no disto: ',
            inv_dist.value(i2pp.value(pt)))
...     print(' -> lig/col no disto: ',
            pp2i.value(inv_dist.value(i2pp.value(pt))))
...
Pt Stone-6 im: [644.286308   232.68331645]
  -> central with disto:  [-0.10064335 -0.16330243]
  -> central no disto:  [-0.10100147 -0.1638833 ]
  -> lig/col no disto:  [643.52988797 231.45641165]
[...]
```

Full script


```
import MMVII

pt2dSet = MMVII.SetMesPtOf1Im.fromFile(
    'MesIm-XXXX.JPG.xml')
pcIntrCalib = MMVII.PerspCamIntrCalib.fromFile(
    'Calib-PerspCentral-Foc-28000_Cam-PENTAX_K5.xml')

pp2i = pcIntrCalib.mapPProj2Im()
i2pp = pp2i.mapInverse()
inv_dist = MMVII.DataInvertOfMapping2D(
                pcIntrCalib.dir_DistInvertible())

for mes in pt2dSet.measures():
    mes.pt = pp2i.value(
                inv_dist.value(
                    i2pp.value(mes.pt)))

pt2dSet.toFile('out_no_dist.xml')
```

Development

Documentation

The basic documentation:

https://pybind11.readthedocs.io/en/stable/basics.html

Classes manipulation documentation:

https://pybind11.readthedocs.io/en/stable/classes.html

How to bind *cMesIm1Pt*

cMesIm1Pt class is declared in *MMVII/include/MMVII_MeasuresIm.h*:

```
class cMesIm1Pt
{
     public :
        cMesIm1Pt(const cPt2dr & aPt,
                  const std::string & aNamePt,
                  tREAL4 aSigma2);
        cMesIm1Pt();
        cPt2dr          mPt;
        std::string     mNamePt;
        tREAL4          mSigma2[3];  // xx xy yy
};
```

The Python-accessible version is in *MMVII/apib11/py_MMVII_MeasuresIm.cpp*:

```
void pyb_init_MeasuresIm(py::module_ &m) {
    py::class_<cMesIm1Pt>(m, "MesIm1Pt",
                          DOC(MMVII_cMesIm1Pt))
      .def(py::init<>(),
           DOC(MMVII_cMesIm1Pt,cMesIm1Pt))
      .def(py::init<const cPt2dr &,
                    const std::string &,tREAL4>(),
           DOC(MMVII_cMesIm1Pt,cMesIm1Pt))
      .def_readwrite("pt", &cMesIm1Pt::mPt,
           DOC(MMVII_cMesIm1Pt,mPt))
      .def_readwrite("namePt", &cMesIm1Pt::mNamePt,
           DOC(MMVII_cMesIm1Pt,mNamePt))
```

The *DOC* part is the doxygen comment.

```
    .def_property("sXX",
       [](const cMesIm1Pt& m)
          {return m.mSigma2[0];},
       [](cMesIm1Pt& m, tREAL8 sXX)
          { m.mSigma2[0] = sXX;},
       "Sigma2 of x coordinate")
    [...]
    .def("__repr__",
       [](const cMesIm1Pt &m) {
         std::ostringstream ss;
         ss.precision(8);
         ss << "MesIm1Pt " << m.mNamePt << " "
            << m.mPt << ", sigma2 (xx,xy,yy): "
            << m.mSigma2[0] << ", "
            << m.mSigma2[1] << ", "
            << m.mSigma2[2] << ")";
         return ss.str();
       })
  ;
```

SetMesPtOf1Im


```
class cSetMesPtOf1Im : public cMemCheck
{
public :
  cSetMesPtOf1Im(const std::string & aNameIm);
  cSetMesPtOf1Im();
  static cSetMesPtOf1Im FromFile(const std::string&);
  void AddMeasure(const cMesIm1Pt &);
  void AddData(const  cAuxAr2007 & anAux);
  void ToFile(const std::string & aNameFile) const;
  [...]
  const std::vector<cMesIm1Pt> &  Measures() const;
  std::vector<cMesIm1Pt> &  Measures() ;
  [...]
private :
  [...]
  std::string               mNameIm;
  std::vector<cMesIm1Pt>  mMeasures;
};
```

SetMesPtOf1Im::Measures() returns a reference to the *mMeasures* attribute. The *cSetMesPtOf1Im* object must not be destroyed while the reference is still used.

This function *return value policy* must be adjusted to avoid crashes and memory corruption:
pybind11.readthedocs.io/en/stable/advanced/functions.html

```
py::class_<cSetMesPtOf1Im>(m, "SetMesPtOf1Im",
                           DOC(MMVII_cSetMesPtOf1Im))
  [...]
  .def("measures",
    py::overload_cast<>(&cSetMesPtOf1Im::Measures),
    py::return_value_policy::reference_internal,
    DOC(MMVII_cSetMesPtOf1Im,Measures))
  [...]
```

py::overload_cast is mandatory since *Measures()* is overloaded.

Future developments

- Distribution:
  - modernize *setup.py* project
  - integrate in main *cmake* build system
  - automatize compilation on several targets
  - integrate in *github actions*
  - distribute on *Python Package Index* (https://pypi.org/)
- API design:
  - based on users needs
  - with users help: tests, documentation, development

Examples of API improvements

- fix *SetMesGCP.__repr__()*, points name missing
- add *cPerspCamIntrCalib.Undist()*
- fix *SetMesPtOf1Im.measuresOfName()* which crashes with an error message beyond understanding
- add *measuresOfName()* to *SetMesGCP*
- add *cBlocOfCamera* class