INSTITUT NATIONAL DE L'INFORMATION GÉOGRAPHIQUE ET FORESTIÈRE

Topographic surveys in MMVII

IGN - 2024

JMM

Introduction

MMVII

MicMac is a free open-source photogrammetry solution developed at (**IGN**) - French Mapping Agency - since 2003. A second version named **MMVII** aimed at facilitating external contributions and being more maintainable in the long term has been in development since 2020.

https://github.com/micmacIGN/micmac

In its global compensation stage, **MMVII** can handle:

- tie points
- ground control points (GCP)
- distortion models
- rigid cameras blocks
- clinometers (in progress)
- and topometric survey measurements!

Let's call topometric survey *topo* for this presentation.

MMVII

Compilation

- latest Windows build:
  https://github.com/micmacIGN/micmac/releases
- latest documentation:
  https://github.com/micmacIGN/micmac/releases/tag/MMVII_Documentation
- compilation instructions:
  https://github.com/micmacIGN/micmac/blob/master/MMVII/README.md

Command line

MMVII is mainly used with command line.

To list existing commands:

```
$ MMVII
Bench => This command execute (many) self verification on MicMac-V2 behav
BlockCamInit => Compute initial calibration of rigid bloc cam
CERN_ImportClino => A temporary command to arrange clino format
ClinoInit => Initialisation of inclinometer
CodedTargetCheckBoardExtract => Extract coded target from images
CodedTargetCircExtract => Extract coded target from images
CodedTargetCompleteUncoded => Complete detection, with uncoded target
CodedTargetExtract => Extract coded target from images
CodedTargetGenerate => Generate images for coded target
CodedTargetGenerateEncoding => Generate en encoding for coded target, acc
CodedTargetRefineCirc => Refine circ target with shape-distorsion using 3
CodedTargetSimul => Extract coded target from images
CompPIB => This command is used compute Parameter of Binary Index
Cpp11 => This command execute some test for to check my understanding of
DM01DensifyRefMatch => Create dense map using a sparse one (LIDAR) with o
...
```

To get the help of a command:

```
$ MMVII ImportGCP
...
   => Import/Convert basic GCP file in MMVII format
 == Mandatory unnamed args : ==
  * string :: Name of Input File
  * string :: Format of file as for ex "SNASXYZSS"
  * string [PointsMeasure,Out] :: Output PointsMeasure
 == Optional named args : ==
  * [Name=NameGCP] string :: Name of GCP set
  * [Name=NbDigName] int :: Number of digit for name,
                 if fixed size required (only if int)
  * [Name=NumL0] int :: Num of first line to read , [Default=0]
  * [Name=NumLast] int :: Num of last line to read
                 (-1 if at end of file) ,[Default=-1]
  * [Name=PatName] std::vector<std::string> :: Pattern
                 for transforming name (first sub-expr)
  * [Name=ChSys] std::vector<std::string> :: Change
                 coordinate system, if 1 Sys In=Out,
                 [Default=[LocalNONE]]
  * [Name=MulCoord] double :: Coordinate multiplier,
                 used to change unity as meter to mm
```

The -**help** argument displays help, whatever is already written on the command line:

```
$ MMVII ImportGCP toto
Level=[UserEr:InsufP]
Mes=[Not enough Arg, expecting 3 , Got only 1]
========= ARGS OF COMMAND ==========
MMVII ImportGCP toto
Aborted

$ MMVII ImportGCP toto -help
...
   => Import/Convert basic GCP file in MMVII format
 == Mandatory unnamed args : ==
  * string :: Name of Input File
  * string :: Format of file as for ex "SNASXYZSS"
  * string [PointsMeasure,Out] :: Output PointsMeasure
 == Optional named args : ==
  * [Name=NameGCP] string :: Name of GCP set
...
```

Use -*Help* for usage examples.

With the correct setup (Linux only, see *MMVII/README.md*), < *tab* > can be used for command line completion.

Example of command line with mandatory and optional parameters:

```
MMVII ImportGCP  2023-10-06_15h31PolarModule.coo NXYZ Std NumL0=14 \
                 NumLast=34  PatName="P\.(.*)" NbDigName=4
```

The command to test MMVII:

```
MMVII Bench 1
```

MMVII Projects

A **MMVII** project root is a directory containing a set of image files or an XML file containing a list of image files names (*SetOfName*).

MMVII will write and read data in a subdirectory named *MMVII-PhgrProj* that will be created automatically when needed.

The file structure is as follows:

```
Project Root
    |-- *.JPG                         <-- image files
    |-- MMVII-LogFile.txt
    +-- MMVII-PhgrProj/
        +-- InitialOrientations
        +-- MetaData                  <-- metadata rules
        |   +-- Std
        +-- Ori                       <-- calib and img ori
        |   +-- InitL93
        |   +-- InitRTL
        |   +-- FinalRTL
        +-- PointsMeasure             <-- 3d and 2d coords
        |   +-- InitL93
        |   +-- InitRTL
        +-- Reports
        +-- RigBlock
        +-- SysCo
        +-- Topo
```

Example:

With initial orientation files *MMVII-PhgrProj/Ori/Init/*.xml*, just give *Init* as command line argument.

Completion works for that!

SysCo

Introduction

The main coordinate systems (SysCo) types supported by MMVII are:

- **Local**: any Euclidian frame, without any geolocalization or vertical direction knowledge
- **GeoC**: geocentric coordinates
- **RTL**: a local Euclidian frame defined by an origin point where Z is normal to ellipsoid and X is on east direction
- **Proj**: any georeferenced system supported by the PROJ library

When SysCo is known, its definition is recorded into the file CurSysCo.xml, in Ori and PointsMeasure directories.

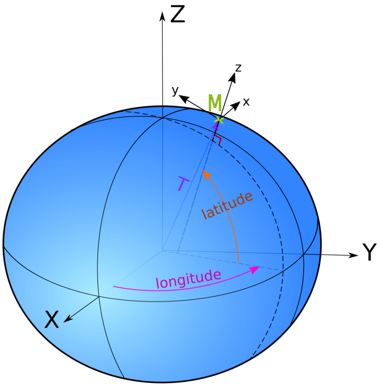
SysCo definition

The SysCo definitions for MMVII commands can be:

- the name of a file in MMVII source subfolder *MMVII/MMVII-RessourceDir/SysCo/* or in project subfolder *MMVII-PhgrProj/SysCo/*, without its extension (e.g., *L93*)
- any PROJ definition (e.g., *EPSG:4326*)
- any string starting with **Local** for a local frame (e.g., LocalAMRules)
- **GeoC** for a geocentric frame

SysCo definition

- a string starting with **RTL**, with the pattern: *RTL*X0*Y0*Z0*Def* (e.g., *RTL*0.675*45.189*0*EPSG:4326*), where you give the origin point coordinates in a certain PROJ system.



Examples

- **SysCo=L93** will set the SysCo to *IGNF:LAMB93*, as defined in *MMVII/MMVII-RessourceDir/SysCo/L93.xml*
- **SysCo=IGNF:LAMB1** will set the SysCo to Lambert I
- **SysCo=LocalPanel** will set the SysCo to a local frame defined as "LocalPanel", that will not be convertible into any other SysCo
- **SysCo=RTL*657700*6860700*0*IGNF:LAMB93** will set the SysCo to a tangent local Euclidian frame, with origin (657700, 6860700, 0) in Lambert 93
- **SysCo=GeoC** will set the SysCo to geocentric coordinates

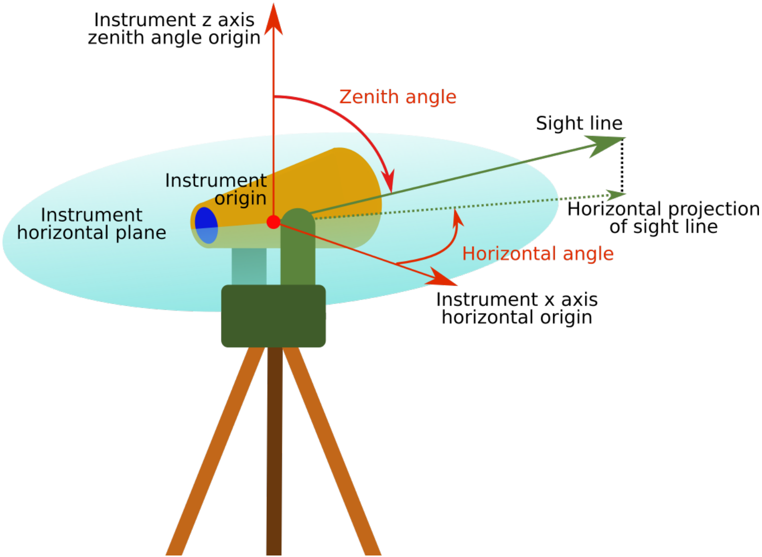
C.F doc chapter 21.

Topo

Station

For now, only station-based topo measurements are available.

These measurements are made from an instrument that is verticalized/plumb or not. The position and orientation of an instrument define a *station*. All the measurements are attached to a station and are expressed in the station frame.


Instrument z axis
zenith angle origin
Zenith angle
Sight line
Instrument
origin
Instrument
horizontal plane
Horizontal projection
of sight line
Horizontal angle
Instrument x axis
horizontal origin

Measurements

The following measurements types are currently supported:

- distances
- horizontal angles
- zenithal angles
- direct Euclidian vectors

Usage

Two MMVII commands can use topo measurements in compensation:

- *OriBundleAdj* via the *TopoFile* option
- *TopoAdj*: when there is no photogrammetry

The topo measurements files can be given as a set of MMVII json or xml files, or in a simplified text format (named *OBS* file) inherited from IGN's Comp3D micro-geodesy compensation software.

All the measurements files must be in the *MMVII-PhgrProj/Topo/[TopoName]* folder.

OBS format

MMVII supports only a subset of Comp3D *OBS* format (https://ignf.github.io/Comp3D/doc/obs.html).

OBS files are text files with fields delimited by any number of spaces or tabs. Blank lines are overlooked. The * character defines a comment that goes up to the end of the line.

A measurement line is composed by:

- code: an integer representing the type of observation
- station name
- target name
- measurement value (meters for distances, gon for angles)
- measurement *a priori* σ (meters for distances, gon for angles)
- anything else is ignored until the end of the line

Example of an *OBS* line describing a measured distance of 100.0000 m, with a σ of 1 mm from *PointA* to *PointB*:

```
3  PointA   PointB   100.0000   0.001  * comment
```

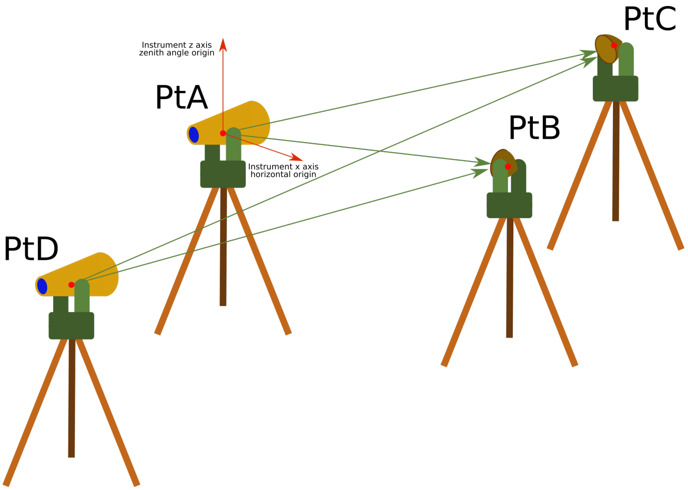
The observations codes are:

- *3*: 3D distance
- *5*: local horizontal (hz) angle
- *6*: local zenithal (zen) angle
- *7*: local horizontal angle for a new station
- *14*: local Δx
- *15*: local Δy
- *16*: local Δz

C.F doc chapter 22.

Example 1

An example of topo dataset can be found in *MMVII/MMVII-UseCaseDataSet/TopoMini/*. It corresponds to this configuration:



3D points file

The initial coordinates of the 4 points, in Lambert 93, are in a simple text file (*inputs/coords.cor*):

```
* 1st column:  0 = free point
1  PtA  657700.000  6860700.000  10.000
0  PtB  657710      6860700      10     * approx
0  PtC  657710      6860710      10     * approx
1  PtD  657700.000  6860690.000  10.000
```

The coordinates of PtA and PtD are supposed known (with a certain precision). The coordinates of PtB and PtC are just for initialization.

How to import this using the *ImportGCP* command?

We give the text format (additional_info, name, x, y, z), the name of the resulting *PointsMeasure* and the coordinates SysCo.

We also specify that the points that have '0' for their additional_info are free points, that the sigma for known points is 0.001m and that lines starting with '*' are comment lines.

```
MMVII ImportGCP inputs/coords.cor ANXYZ InitL93 \
  ChSys=[L93] AddInfoFree=0 Sigma=0.001 Comment=*
```

Here the sigma is given in computation frame, there is no conversion for now.

In the resulting file *MMVII-PhgrProj/PointsMeasure/InitL93/MesGCP-coords.xml*, the points PtA and PtD have an attribute *__Opt__Sigma2* equivalent to $\sigma = 0.001m$, the points PtB and PtC have no *__Opt__Sigma2*, making them free points.

The file *MMVII-PhgrProj/PointsMeasure/InitL93/CurSysCo.xml*, records the SysCo of *InitL93*.

SysCo

A *RTL* SysCo is mandatory to be able to compute a topo compensation. PtA is chosen as RTL origin (tangency point).

What is the SysCo definition?

What is the *GCPChSysCo* command to make the conversion?

SysCo definition:

```
RTL*657700*6860700*0*IGNF:LAMB93
```

SysCo definition:

```
RTL*657700*6860700*0*IGNF:LAMB93
```

GCPChSysCo command:

```
MMVII GCPChSysCo "RTL*657700*6860700*0*IGNF:LAMB93" \
  InitL93 InitRTL
```

The file *MMVII-PhgrProj/PointsMeasure/InitRTL/CurSysCo.xml*, records the SysCo of *InitRTL*.

SysCo

This transfomation can also be done during *ImportGCP*:

```
MMVII ImportGCP inputs/coords.cor ANXYZ InitRTL \
  ChSys="[L93,RTL*657700*6860700*0*IGNF:LAMB93]" \
  AddInfoFree=0 Sigma=0.001 Comment=*
```

SysCo

This transfomation can also be done during *ImportGCP*:

```
MMVII ImportGCP inputs/coords.cor ANXYZ InitRTL \
  ChSys="[L93,RTL*657700*6860700*0*IGNF:LAMB93]" \
  AddInfoFree=0 Sigma=0.001 Comment=*
```

ImportGCP can also automatically create a *RTL* SysCo, with its origin equal to the average of the input coordinates: just give **RTL** as the destination SysCo. This new SysCo will be saved as *MMVII-PhgrProj/SysCo/RTL.xml*, making the SysCo available for every following command as **RTL**:

```
MMVII ImportGCP inputs/coords.cor ANXYZ InitRTL \
  ChSys=[L93,RTL] AddInfoFree=0 Sigma=0.001 Comment=*
```

Measurements

- an instrument on PtA measures hz angle, zen angle and distance to PtB and PtC
- an instrument on PtD makes the same to PtB and PtC

The corresponding *OBS* file (*inputs/meas.obs*) is:

```
7   PtA   PtB     0     0.001
6   PtA   PtB   100     0.001
3   PtA   PtB    10.05  0.005
5   PtA   PtC   -40.62  0.001
6   PtA   PtC   100     0.001
3   PtA   PtC    14.88  0.005

7   PtD   PtB     0     0.001
6   PtD   PtB   100     0.001
3   PtD   PtB    14.88  0.005
5   PtD   PtC   -14.96  0.001
6   PtD   PtC   100     0.001
3   PtD   PtC    22.82  0.005
```

Measurements

This file has to be imported into a subdirectory of *MMVII-PhgrProj/Topo*:

```
MMVII ImportOBS inputs/meas.obs Obs1
```

Unknowns count

Unknowns count

Two verticalized stations, one with its origin on PtA, the other on PtD. Each has an horizontal orientation unknown (G_0) due to the random orientation of the instrument when it has been set.

Unknowns count

Two verticalized stations, one with its origin on PtA, the other on PtD. Each has an horizontal orientation unknown (G_0) due to the random orientation of the instrument when it has been set.

The number of unknowns in this configuration is:

- 3 per point $(x, y, z) \rightarrow$ 12 unknowns
- 1 per station $(G_0) \rightarrow$ 2 unknowns

The number of constraints is:

- 3 per constrained point, PtA and PtD $\rightarrow$ 6 constraints
- 1 per topo measurement $\rightarrow$ 12 constraints

Unknowns count

Two verticalized stations, one with its origin on PtA, the other on PtD. Each has an horizontal orientation unknown (G_0) due to the random orientation of the instrument when it has been set.

The number of unknowns in this configuration is:

- 3 per point $(x, y, z) \rightarrow$ 12 unknowns
- 1 per station $(G_0) \rightarrow$ 2 unknowns

The number of constraints is:

- 3 per constrained point, PtA and PtD $\rightarrow$ 6 constraints
- 1 per topo measurement $\rightarrow$ 12 constraints

Total: 14 unknowns, 18 constraints.

Adjustment

The *TopoAdj* command can perform an adjustment between topo and GCP constraints. It is used as a substitute to *OriBundleAdj* when there is no photogrammetry.

```
== Mandatory unnamed args : ==
 * string [Topo,In] :: Dir for Topo measures
 * string [Topo,Out] :: Dir for Topo measures output
 * string [PointsMeasure,In] :: Dir for points initial coordinates
 * string [PointsMeasure,Out] :: Dir for points final coordinates

== Optional named args : ==
 * [Name=GCPW] double :: Constrained GCP weight factor (default: 1)
 * [Name=DataDir] string :: Default data directories  ,[Default=Std]
 * [Name=NbIter] int :: Number of iterations ,[Default=10]
 * [Name=GCPFilter] string :: Pattern to filter GCP by name
 * [Name=GCPFilterAdd] string :: Pattern to filter GCP by additional in
 * [Name=GCPDirOut] string [PointsMeasure,Out] :: Dir for output GCP
 * [Name=LVM] double :: Levenberg-Marquardt parameter (to have better co
```

Command line ?

In our example, the input topo directory is *Obs1* and the input PointsMeasure is *InitRTL*. We give output directories names for topo and points.

```
MMVII TopoAdj Obs1 InitRTL Obs1_out FinalRTL
```

The final σ_0 value should be around 1 if everything goes well. In this example, $\sigma_{0init} > 5000$, because the initial coordinates of PtB and PtC are approximate, and after 10 iterations it stabilizes at $\sigma_{0final} = 1.7$.

Outputs

The output topo directory contains a single xml file with all the measurements and some output values (residuals, stations orientations...). It can be used as topo input file.

For now, there is no computation of final coordinates uncertainty...

The last step is to convert the RTL coordinates to Lambert 93:

Outputs

The output topo directory contains a single xml file with all the measurements and some output values (residuals, stations orientations...). It can be used as topo input file.

For now, there is no computation of final coordinates uncertainty...

The last step is to convert the RTL coordinates to Lambert 93:

```
MMVII GCPChSysCo L93 FinalRTL FinalL93
```

Orientations

Stations orientation constraints

Each station has orientation constraints that have to be given before the station observations lines in the *OBS* file.

The possible orientation constraints are:

- `#FIX`: the station is axis-aligned, it is verticalized and oriented to north
- `#VERT`: the station is verticalized and only horizontal orientation is free
- `#BASC`: the station orientation has 3 degrees of freedom, meaning non-verticalized and not oriented to north

After a # line (`#FIX`, `#VERT` or `#BASC`), all the following stations have the new orientation constraint until the next # line.

Each `OBS` file starts with an implicit `#VERT`, making the stations verticalized by default.

For now, the vertical is modeled as the Earth's ellipsoid normal. Vertical deflection grids may be added later.

Example

It is possible to say that the station on PtA was not verticalized by using #BASC:

```
#BASC
 7   PtA    PtB      0       0.001
 6   PtA    PtB    100       0.001
 3   PtA    PtB     10.05    0.005
 5   PtA    PtC    -40.62    0.001
 6   PtA    PtC    100       0.001
 3   PtA    PtC     14.88    0.005

#VERT
 7   PtD    PtB      0       0.001
 6   PtD    PtB    100       0.001
 3   PtD    PtB     14.88    0.005
 5   PtD    PtC    -14.96    0.001
 6   PtD    PtC    100       0.001
 3   PtD    PtC     22.82    0.005
```

Several stations on the same point

MMVII automatically creates a new station when a point is used for the first time as the origin of a measurement.

If we have to make a new set of orientation unknowns because two instruments were set on the same point with different orientations, we can:

- use separate `OBS` files
- add a #-line to separate the measurements sets (`#NEW` to keep the orientation constraints)
- use a code **7** instead of **5** for the first measurement

A separate `OBS` files or a #-line closes all current stations. Code **7** only closes the previous station on one point.

Example


```
5  St1  PtA  100.000  0.001 * creates a station on St1
5  St1  PtB  110.000  0.001

5  St2  PtA  200.000  0.001 * creates a station on St2

7  St1  PtA  150.000  0.001 * closes station on St1, creates new on St1
5  St1  PtC  210.000  0.001

5  St2  PtE  250.000  0.001 * uses previous station on St2
```

Example


```
5   St1   PtA   100.000   0.001 * creates a station on St1
5   St1   PtB   110.000   0.001

5   St2   PtA   200.000   0.001 * creates a station on St2

7   St1   PtA   150.000   0.001 * closes station on St1, creates new on St1
5   St1   PtC   210.000   0.001

5   St2   PtE   250.000   0.001 * uses previous station on St2
```

But:

```
5   St1   PtA   100.000   0.001 * creates a station on St1
5   St1   PtB   110.000   0.001

5   St2   PtA   200.000   0.001 * creates a station on St2

#NEW                            * closes all the stations
5   St1   PtA   150.000   0.001 * creates a station on St1
5   St1   PtC   210.000   0.001

5   St2   PtE   250.000   0.001 * creates a station on St2
```

Centering

For two points that should be one above an other:

```
#FIX
14  PtA  PtB  0   0.001 * PtA and PtB have the same
15  PtA  PtB  0   0.001 * horizontal position

16  PtA  PtB  0.1 0.001 * PtB is 10cm above PtA
```

Warning: code 16 is a difference of height only for points with the same horizontal position!

Use the future code 4 for generic height difference.

Application to example 1?

Orientation special case

If a station has no orientation measurements (no hz angle nor dx/dy with a significative distance), it is automatically set as a `#FIX` station.

It simplifies the usage of distances between points when there are no angle measurements (explicit `#FIX` not required).

In the future, distances measurements may exist outside of stations and then have no orientations unknowns.

Example 2

Comp3D to MMVII

Comp3DFigureNoInit dataset: a regular Comp3D computation project.

7 verticalized stations, measuring 78 targets on the ground and aiming at each other.

IGN

Computation frame

There is no geolocalization, but approximative latitude is 44.40°(for ellipsoid curvature).

COR file :

```
1  HLLST0001    100.00000 100.00000 10.00000  0.00100 0.00100 0.00100
```

Only one point, with constraints on its 3 coordinates (code **1**) is given (arbitrary coordinates). Comp3D auto-initialization methods computes the initial coordinates of all points, from **HLLST0001** and one azimuth (in the **OBS** file).

Computation frame

In Comp3D, the computation frame is spherical, with a radius of the total curvature of the ellipsoid at the given latitude.

MMVII works in a RTL frame uses an ellipsoid for vertical modelization.

Thus the results will be different.

RTL frame definition for a frame at latitude 44.40°:

Computation frame

In Comp3D, the computation frame is spherical, with a radius of the total curvature of the ellipsoid at the given latitude.

MMVII works in a RTL frame uses an ellipsoid for vertical modelization.

Thus the results will be different.

RTL frame definition for a frame at latitude 44.40°:

`RTL*0*44.40*0*EPSG:4326`

Import COR

To import a Comp3D **COR** file using only codes **0** and **1**:

Import COR

To import a Comp3D **COR** file using only codes **0** and **1**:

```
MMVII ImportGCP figure.cor ANXYZ InitRTL \
      ChSys=["RTL*0*44.40*0*EPSG:4326"] \
      AddInfoFree=0 Sigma=0.001 Comment=*
```

Sigma can't be read with **ImportGCP**. Many other point codes are used in Comp3D, but only **0** and **1** are supported in MMVII.

Obs file


```
MMVII ImportOBS figure.obs Obs1
```

Obs file


```
MMVII ImportOBS figure.obs Obs1
```

Fails:

```
Reading obs file "./MMVII-PhgrProj/Topo/Obs1/obs.obs"...
 ######################################
Level=[Internal Error]
Mes=[Error reading ./MMVII-PhgrProj/Topo/Obs1/obs.obs at line 1:
    "8 HLLST0001  HLLPI0005  100 0.001 0 0 0"]
```

Obs file

The **OBS** file starts with:

```
8 HLLST0001  HLLPI0005  100 0.001 0 0 0 **un faux gisement
 * mais qui part bien du point HLLST0001 (seul dans le .cor)

*Données réduites

*Tours d'horizon
*Station n°1 HLLST0001 Temperature = 290 Pression = 7609
7 HLLST0001 HLLPI0005     0.0000  0.0008  0.0000 0.0000 0.0000
5 HLLST0001 HLLPI0012   388.7158  0.0008  0.0000 0.0000 0.0000
5 HLLST0001 HLLPI0015   383.9676  0.0008  0.0000 0.0000 0.0000
5 HLLST0001 HLLPI0022   372.7380  0.0008  0.0000 0.0000 0.0000
```

The code **8** is an azimuth constraint, saying that HLLPI0005 is in east direction from HLLST0001. It fixes the orientation ambiguity of the system and kickstarts the initial coordinates estimation.

Obs file

The code **8** is not supported in MMVII, we have to replace it with:

Obs file

The code **8** is not supported in MMVII, we have to replace it with:

```
#FIX
5 HLLST0001  HLLPI0005  100 0.001   * hz orientation
#VERT    * next stations have unknown G_0
*Données réduites

*Tours d'horizon
*Station n°1 HLLST0001 Temperature = 290 Pression = 7609
7 HLLST0001 HLLPI0005     0.0000  0.0008  0.0000 0.0000 0.0000
5 HLLST0001 HLLPI0012   388.7158  0.0008  0.0000 0.0000 0.0000
5 HLLST0001 HLLPI0015   383.9676  0.0008  0.0000 0.0000 0.0000
5 HLLST0001 HLLPI0022   372.7380  0.0008  0.0000 0.0000 0.0000
```

Results


```
MMVII TopoAdj Obs1 InitRTL Obs1_out FinalRTL NbIter=5 | grep "sigma0"
Topo sigma0: 6.5339 (533 obs)
Topo sigma0: 2.08124 (533 obs)
Topo sigma0: 0.896159 (533 obs)
Topo sigma0: 0.896159 (533 obs)
Topo sigma0: 0.896159 (533 obs)
```

Results from Comp3D:

```
sigma0 initial:              85.5105
sigma0 final:                 0.8962
Iterations:                   3
Sphere radius:          6377652.47 m
Total observations number:   538
Active observations number:  536
Parameters:                  262
```

Differences

For this computation, in MMVII:

- no obs code 8 (#FIX + code 5)
- no deactivated obs (skips obs with sigma<0)
- coordinates constraints are not taken into account in statistics (for now)
- better initialization!?
- ellipsoidal model
- refraction coefficient is fixed (for now)
- no sigma exports

Other pros:

- can be adjusted with photogrammetry
- supports unverticalized stations
- one station can have angular and cartesian observations

Other cons:

- lacking many initialization methods
- no 1D or 2D points
- no height differences
- no PPM/target definition
- not vertical deflexion
- sigma0 is computed only on topo obs (no GCP or photo)

Example 3

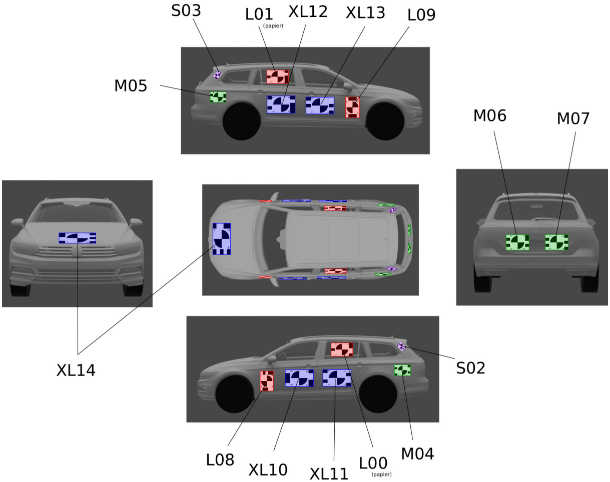
Principle

TopoPrissma9img dataset: find a car trajectory from fixed cameras

Each camera is pre-calibrated.

There are several fixed ground points used to impose an orientation to the cameras:

The car is equipped with several coded targets. The 3D coordinates of the targets in the car frame were measured by topometry.



Cameras orientation

To get the cameras orientation:

- import ground targets 3D coordinates
- import ground targets images coordinates
- add images metadata to make calibration/image links
- import cameras calibrations
- cameras initial resection on ground targets
- adjustment on ground targets

Ground targets

Get 3d coords for ground targets, convert to RTL:

Ground targets

Get 3d coords for ground targets, convert to RTL:

```
MMVII ImportGCP inputs/coord_gnd.cor SNXYZ GND \
   ChSys=[L93,RTL] Sigma=0.001
```

Ground targets

Get 3d coords for ground targets, convert to RTL:

```
MMVII ImportGCP inputs/coord_gnd.cor SNXYZ GND \
   ChSys=[L93,RTL] Sigma=0.001
```

Copy 2d mes for gnd points:

```
cp inputs/gnd_img_V2/* \
   MMVII-PhgrProj/PointsMeasure/GND/
```

Image Metadata


```
# set metadata
MMVII EditCalcMTDI Std ModelCam \
   Modif=['IGN2_.*.jpg',BFS-PGE-161S7M-C,0] Save=1
MMVII EditCalcMTDI Std AdditionalName \
   Modif=['IGN2_.*cam-2234.(...)-.*.jpg','$1',0] \
   Save=1

# focal 8mm for 125 130 138, the others 12mm
MMVII EditCalcMTDI Std Focalmm \
   Modif=['IGN2_.*-cam-223..1(25|30|38)-.*.jpg',8,0]\
   Save=1
MMVII EditCalcMTDI Std Focalmm \
   Modif=['IGN2_.*-cam-.*.jpg',12,1] Save=1
```

Warning: **'$1'** for linux, **"$1"** for windows !

Cameras orientation

```
# copy calibs
mkdir -p MMVII-PhgrProj/Ori/Calib
cp inputs/calibsV2/*.xml MMVII-PhgrProj/Ori/Calib

# initial orientation on GND points
MMVII OriPoseEstimSpaceResection ".*.jpg" GND \
    Calib Init

# adjust ori on gnd points
MMVII OriBundleAdj ".*.jpg" Init Adjusted \
    GCPDir=GND GCPW=[1,0.5] PPFzCal=".*" \
    PoseVisc=[1,1]

MMVII ReportGCP ".*.jpg" GND Adjusted
```

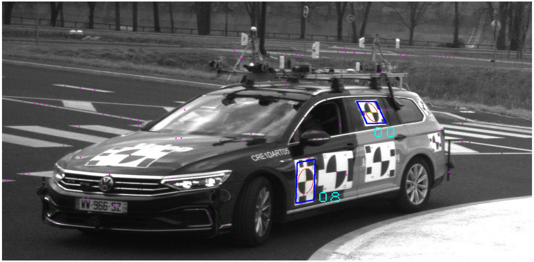
Car frame

To get the car origin from the images :

- fix cameras orientation, position and calibration
- get the coded targets images coordinates
- use the targets coordinates in car sub-frame (as topo obs)
- adjust

Coded targets

```
# extract coded targets
MMVII CodedTargetExtract 'IGN2_.*.jpg' \
   inputs/IGNDroneSym_*_FullSpecif.xml \
   Adjust=1 CC=1 DMD=16 Debug=511 \
   OutPointsMeasure=Targets
# results can be checked in MMVII-PhgrProj/VISU/
```



Prepare topo

The OBS file *inputs/car_xyz.obs* represents the car frame as a set of local euclidian measurements:

```
14 car 00 -0.7180 0.0003 * DX from car to 00 = -0.7180, sigma = 0.0003
15 car 00 -0.7048 0.0003 * DY from car to 00
16 car 00 0.8287 0.0003  * DZ from car to 00
14 car 01 0.6973 0.0003
15 car 01 -0.7168 0.0003
16 car 01 0.8470 0.0003
...
```

Prepare topo


```
# get approximate 3d coords for coded targets,
# as free points
MMVII ImportGCP inputs/coord_approx_car.cor \
   SNXYZ Targets ChSys=[L93,RTL] Sigma=-1

#import topo: coded targets coords in car frame
MMVII ImportOBS inputs/car_xyz.obs BlocCar
```

Adjust car frame


```
# adjust with frozen cameras
MMVII OriBundleAdj ".*.jpg" Adjusted Out \
   GCPDir=Targets GCPW=[1,0.1] TopoDirIn=BlocCar \
   PPFzCal=".*" PatFzCenters=".*" PatFzOrient=".*" \
   TopoDirOut=BlocCarOut GCPDirOut=CarOut NbIter=20

# export coords to L93
MMVII GCPChSysCo L93 CarOut CarOutL93
```

Check :

- *MMVII-PhgrProj/Topo/BlocCarOut/TopoOut.xml* for topo residuals
- *MMVII-PhgrProj/PointsMeasure/CarOutL93/MesGCP-NewGCP.xml* for final coordinates.

Adjust car frame

Use both GCP sets at the same time with AddGCPW:

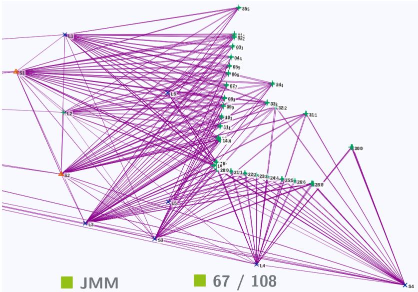
```
MMVII OriBundleAdj ".*.jpg" Adjusted Out2
   GCPDir=GND GCPW=[1,0.5] \
   AddGCPW=[[Targets,1,0.1]] TopoDirIn=BlocCar \
   PPFzCal=".*" PatFzCenters=".*" PatFzOrient=".*" \
   TopoDirOut=BlocCarOut GCPDirOut=CarOut NbIter=20
```

or:

```
MMVII OriBundleAdj ".*.jpg" Adjusted Out2 \
   GCPDir=Targets GCPW=[1,0.1] \
   AddGCPW=[[GND,1,0.5]] TopoDirIn=BlocCar \
   PPFzCal=".*" PatFzCenters=".*" PatFzOrient=".*" \
   TopoDirOut=BlocCarOut GCPDirOut=CarOut NbIter=20
```

Example 4

Polygon K



Polygon K

- two types of coded targets
- topo measurements
- two cameras in a rigid bloc
- no initial calibration
- no initial orientation

Extract both targets types on images:

```
MMVII CodedTargetCircExtract ".*JPG" \
    inputs/CERN_Nbb14_*_FullSpecif.xml \
    DiamMin=8 OutPointsMeasure=TargetsC ZoomVisuEllipse=1

MMVII CodedTargetExtract ".*JPG" \
    inputs/IGNIndoor_Nbb12_*_FullSpecif.xml \
    DMD=30 Debug=1023 Margin=0.3 Tolerance=0.2 \
    OutPointsMeasure=TargetsI
```

Add the camera specs in MMVII/MMVII-RessourceDir/CameraDataBase.xml:

```
<Pair>
   <K>"SONY A6400"</K>
   <V>
      <Name>"SONY A6400"</Name>
      <SzPix_micron> 3.9 3.9 </SzPix_micron>
      <SzSensor_mm> 23.4  15.6 </SzSensor_mm>
      <NbPixels>6000  4000</NbPixels>
   </V>
</Pair>
```

Set metadata:

```
#specify the camera model
MMVII EditCalcMTDI Std ModelCam ImTest=C1_00100.JPG \
    Modif=[.*.JPG,"SONY A6400",0] Save=1

#specify focal length
MMVII EditCalcMTDI Std Focalmm ImTest=C1_00100.JPG \
    Modif=[".*.JPG",16,0] Save=1

#specify groups of images (C1=Camera 1) & (C2=Camera 2)
MMVII EditCalcMTDI Std AdditionalName \
    ImTest=C1_00100.JPG \
    Modif=["(.*)_.*.JPG","\$1",0] Save=1
```

Compute topo:

```
MMVII ImportGCP inputs/coord.cor ANXYZ InitTopoRTL \
    ChSys=[L93,"RTL*657700*6860700*0*IGNF:LAMB93"] \
    AddInfoFree=0 Sigma=0.001 Comment=*

MMVII ImportOBS inputs/polygone.obs TopoObs

MMVII TopoAdj TopoObs InitTopoRTL TopoOut TargetsTopoRTL
```

Initial orientation on GCPs by space resection:

```
#create an initial calibration with default params
MMVII OriCreateCalib ".*JPG" CalibInit Degree=[3,1,1]

# Add 3d coords to extracted 2d coords"
cp MMVII-PhgrProj/PointsMeasure/TargetsTopoRTL/* \
    MMVII-PhgrProj/PointsMeasure/TargetsI/
cp MMVII-PhgrProj/PointsMeasure/TargetsTopoRTL/* \
    MMVII-PhgrProj/PointsMeasure/TargetsC/

#filter to keep only images adapted to space resection
MMVII OriPoseEstimCheckGCPDist ".*JPG" TargetsC

#calibrated space resection
MMVII OriPoseEstimSpaceResection \
    SetFiltered_GCP_OK_Resec.xml \
    TargetsC CalibInit Resec
```

Bundle adjustment:

```
# init block cam
MMVII BlockCamInit SetFiltered_GCP_OK_Resec.xml Resec \
    "(.*)_(.*).JPG" [1,2] RigInit ShowByBloc=1

# use block cam in BA
MMVII OriBundleAdj SetFiltered_GCP_OK_Resec.xml Resec BA \
    GCPDir=TargetsC GCPW=[1,0.5] TopoDirIn=TopoObs \
    BRDirIn=RigInit BRW=[1e-2,1e-5] NbIter=20 \
    GCPDirOut=FinalRTL

# export to L93
MMVII GCPChSysCo L93 FinalRTL FinalL93

# reports
MMVII ReportGCP SetFiltered_GCP_OK_Resec.xml FinalL93 BA
```

Both targets types can be added to bundle adjustment, but 3D coords must be split between both PointsMeasure folders.

```
MMVII OriBundleAdj SetFiltered_GCP_OK_Resec.xml Resec BA \
    GCPDir=TargetsC GCPW=[1,0.5] TopoDirIn=TopoObs \
    BRDirIn=RigInit BRW=[1e-2,1e-5] NbIter=20 \
    AddGCPW=[[TargetsI,1,0.5]]  GCPDirOut=AllPtsOut
```

TODO

Missing features

Vertical:

- stations and targets heights
- height differences
- 2D and 1D points ?

Missing features

Statistics:

- residuals for every constraint
- correct σ_0
- parameters confidence estimation

Misc:

- refraction parameter
- relative sigmas
- more useful error messages
- units choice?

Missing features

New measurement types:

- unknown sub-frame
- rotation axis
- distances equalities
- barycenters

Implementation

Details

See documentation chapter 12

and presentation on the wiki:

https://github.com/micmacIGN/micmac/files/14614598/SerialDeriv.pdf

Automatic derivation

MMVII has its own automatic derivation system, based on c++ templates and compilator interpretation of source code.

Steps:

- write a formula in c++
- register this formula in MMVII source
- run **MMVII GenCodeSymDer** or simply **make full**
- use this formula in MMVII least squares

Implementation: MMVII/src/SymbDerGen/Formulas_Geom3D.h

```
class cDist3D
{
  public :
    cDist3D() {}
    static const std::vector<std::string> VNamesUnknowns() {
        return {"p1_x","p1_y","p1_z", "p2_x","p2_y","p2_z"};
    }
    static const std::vector<std::string> VNamesObs() { return {"D"}; }
    std::string FormulaName() const { return "Dist3D"; }

    template <typename tUk,typename tObs>
             static std::vector<tUk> formula
                  (   const std::vector<tUk> & aVUk,
                          const std::vector<tObs> & aVObs   )
    {
          typedef cPtxd<tUk,3> tPt;
          tPt p1 = VtoP3(aVUk,0);
          tPt p2 = VtoP3(aVUk,3);
          tPt v  = p1-p2;
          const tUk & ObsDist  = aVObs[0];
          return {  Norm2(v) - ObsDist } ;
     }
};
```

Registration: MMVII/src/SymbDerGen/GenerateCodes.cpp

```
// dist3d
template <class Type>
cCalculator<Type> * TplEqDist3D(bool WithDerive,int aSzBuf)
{
  return StdAllocCalc(NameFormula(cDist3D(),WithDerive),aSzBuf);
}

cCalculator<double> * EqDist3D(bool WithDerive,int aSzBuf)
{
  return TplEqDist3D<double>(WithDerive,aSzBuf);
}

...

int cAppliGenCode::Exe()
{
  ...
 for (const auto WithDer : {true,false})
 {
   ...
   GenCodesFormula((tREAL8*)nullptr,cNetWConsDistSetPts(3,true),WithDer)
   GenCodesFormula((tREAL8*)nullptr,cDist3D(),WithDer);
```

Generated: MMVII/src/GeneratedCodes/CodeGen_cDist3DVal.cpp

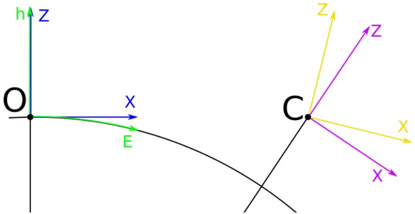
```
  for (size_t aK=0; aK < this->mNbInBuf; aK++) {
// Declare local vars in loop to make them per thread
    double &p1_x = this->mVUk[aK][0];
    double &p1_y = this->mVUk[aK][1];
    double &p1_z = this->mVUk[aK][2];
    double &p2_x = this->mVUk[aK][3];
    double &p2_y = this->mVUk[aK][4];
    double &p2_z = this->mVUk[aK][5];
    double &D = this->mVObs[aK][0];
    double F11_ = (p1_y - p2_y);
    double F10_ = (p1_z - p2_z);
    double F12_ = (p1_x - p2_x);
    double F14_ = (F11_ * F11_);
    double F13_ = (F10_ * F10_);
    double F15_ = (F12_ * F12_);
    double F16_ = (F14_ + F15_);
    double F17_ = (F13_ + F16_);
    double F18_ = std::sqrt(F17_);
    double F19_ = (F18_ - D);
    this->mBufLineRes[aK][0] = F19_;
  }
```

Generated: MMVII/src/GeneratedCodes/CodeGen_cDist3DVDer.cpp

```
for (size_t aK=0; aK < this->mNbInBuf; aK++) {
  double &p1_x = this->mVUk[aK][0];
  double &p1_y = this->mVUk[aK][1];
  double &p1_z = this->mVUk[aK][2];
  double &p2_x = this->mVUk[aK][3];
  double &p2_y = this->mVUk[aK][4];
  double &p2_z = this->mVUk[aK][5];
  double &D = this->mVObs[aK][0];
  double F12_ = (p1_x - p2_x);
  double F31_ = (p2_x - p1_x);
  double F35_ = (p2_y - p1_y);
  double F39_ = (p2_z - p1_z);
  double F11_ = (p1_y - p2_y);
  double F10_ = (p1_z - p2_z);
  double F36_ = (F35_ + F35_);
  double F21_ = (F12_ + F12_);
  double F32_ = (F31_ + F31_);
  double F27_ = (F10_ + F10_);
  double F40_ = (F39_ + F39_);
  double F24_ = (F11_ + F11_);
  double F15_ = (F12_ * F12_);
  double F14_ = (F11_ * F11_);
  double F13_ = (F10_ * F10_);
  double F16_ = (F14_ + F15_);
  double F17_ = (F13_ + F16_);
  double F18_ = std::sqrt(F17_);
  double F20_ = (2 * F18_);
  double F19_ = (F18_ - D);
  double F22_ = (F21_ / F20_);
  double F25_ = (F24_ / F20_);
  double F28_ = (F27_ / F20_);
  double F33_ = (F32_ / F20_);
  double F37_ = (F36_ / F20_);
  double F41_ = (F40_ / F20_);
  this->mBufLineRes[aK][0] = F19_;
  this->mBufLineRes[aK][1] = F22_;
  this->mBufLineRes[aK][2] = F25_;
  this->mBufLineRes[aK][3] = F28_;
  this->mBufLineRes[aK][4] = F33_;
  this->mBufLineRes[aK][5] = F37_;
  this->mBufLineRes[aK][6] = F41_;
}
```

Topo formulas

For Topo stations, the measurements are expressed in the instrument's local frame.



- `Green`: projection SysCo
- `Blue`: adjustment RTL SysCo
- `Purple`: 3D point local vertical frame
- `Yellow`: instrument frame

Each instrument orientation rotation from RTL is computed via the local vertical frame at its origin:

$$R_{RTL \to Instr} = R_{Vert \to Instr} \cdot R_{RTL \to Vert}$$

Where $R_{RTL \to Vert}$ is computed by the SysCo from station origin position and $R_{Vert \to Instr}$ is unknown, with a degree of liberty depending on the station orientation constraint.

It is recorded in `cTopoObsSetStation` as:

```
tRot mRotSysCo2Vert; //rotation between global SysCo and vertical frame

tRot mRotVert2Instr; //current value rotation from vert to instr frame

std::vector<tREAL8> mParams;
    // mRotVert2Instr unknown is recorded as mParams[0..2]
```

The possible orientation constraints are:

- `#FIX`: `mParams` is fixed for x, y and z
- `#VERT`: `mParams` is fixed for x and y
- `#BASC`: `mParams` has no fixed component

The transformation from RTL to instrument local frame is:

$$T_{Instr} = R_{RTL \rightarrow Instr} \cdot (T_{RTL} - S_{RTL})$$

Where:

- T_{Instr}: target point in instrument local frame
- S_{RTL}: station origin point in RTL SysCo
- T_{RTL}: target point in RTL SysCo
- $R_{RTL \rightarrow Instr}$: rotation from RTL to instrument frame

Then for each type of observation (l being the measurement value):

- `cFormulaTopoDX`:

$$residual = T_{Instr_X} - l$$

- `cFormulaTopoDY`:

$$residual = T_{Instr_Y} - l$$

- `cFormulaTopoDZ`:

$$residual = T_{Instr_Z} - l$$

- `cFormulaTopoHz`:

$$residual = \arctan\left(T_{Instr_X}, T_{Instr_Y}\right) - I$$

- `cFormulaTopoZen`:

$$ref = 0.12.\frac{hz\_dist\_ellips\left(T, S\right)}{2.earth\_radius}$$

$$d_{hz} = \left\| T_{Instr_X}, T_{Instr_Y} \right\|$$

$$residual = \arctan\left(d_{hz}, T_{Instr_Z}\right) - ref - I$$

- `cFormulaTopoDist`:

$$residual = \left\| T_{Instr} \right\| - I$$

Angles residuals are in $[-\pi, +\pi]$ interval.

This is implemented like this:

```
class cFormulaTopoHz
{
public :
   std::string FormulaName() const { return "TopoHz";}
   std::vector<std::string>  VNamesUnknowns()  const
   {
     // Instrument pose with 6 unknowns : 3 for center, 3 for axiator
     // target pose with 3 unknowns : 3 for center
     return  Append(NamesPose("Ci","Wi"),NamesP3("P_to"));
   }
   std::vector<std::string>    VNamesObs() const
   {
     // for the instrument pose, the 3x3 current rotation matrix
     // as "observation/context" and the measure value
     return  Append(NamesMatr("mi",cPt2di(3,3)), {"val"} );
   }
```

```
    template <typename tUk>
                std::vector<tUk> formula
                (
                   const std::vector<tUk> & aVUk,
                   const std::vector<tUk> & aVObs
                ) const
    {
      cPoseF<tUk>  aPoseInstr2RTL(aVUk,0,aVObs,0,true);
      cPtxd<tUk,3> aP_to = VtoP3(aVUk,6);
      auto       val = aVObs[9];
      cPtxd<tUk,3>  aP_to_instr = aPoseInstr2RTL.Inverse().Value(aP_to);
      auto az = ATan2( aP_to_instr.x(), aP_to_instr.y() );
      return {  DiffAngMod(az, val) };
    }
};
```

```
template <Type> Type DiffAngMod(const Type & aA, const Type & aB)
{
     auto aDiff = aA - aB;
     if (std::isfinite(aDiff))
     {
         if (aDiff < -M_PI)
         {   int n = (aDiff-M_PI)/(-2*M_PI);
             aDiff += n*2*M_PI;  }
         if (aDiff > 2*M_PI)
         {   int n = aDiff/(2*M_PI);
             aDiff -= n*2*M_PI;  }
     }
     return aDiff;
}

template <Type> Type DerA_DiffAngMod(const Type & aA,const Type & aB)
{     return 1.;     }

template <Type> Type DerB_DiffAngMod(const Type & aA,const Type & aB)
{     return -1.;    }


MACRO_SD_DEFINE_STD_BINARY_FUNC_OP_DERIVABLE( MMVII,
        DiffAngMod, DerA_DiffAngMod, DerB_DiffAngMod )
```

Least squares

MMVII least square system is described in documentation (12.3 and 12.6).

Topo in least squares

The topo classes are in MMVII/src/Topo/:

- `cTopoPoint`: a point used with survey measurements. Keeps a pointer to the unknowns from GCP or Ori.
- `cTopoObs`: an observation corresponding to a formula, between several points.
- `cTopoObsSet`: a set of observations. The set is used to share common parameters between several observations. e.g., `cTopoObsSetStation` adds a rotation corresponding to an instrument setting.
- `cBA_Topo`: the class that handles the least square part. It records all the points and sets.

Topo formulas map:

```
std::map<eTopoObsType, cCalculator<double>*> mTopoObsType2equation =
    {
        {eTopoObsType::eDist, EqTopoDist(true,1)},
        {eTopoObsType::eHz,   EqTopoHz(true,1)},
        {eTopoObsType::eZen,  EqTopoZen(true,1)},
        {eTopoObsType::eDX,   EqTopoDX(true,1)},
        {eTopoObsType::eDY,   EqTopoDY(true,1)},
        {eTopoObsType::eDZ,   EqTopoDZ(true,1)},
    };
```

Unknowns specific to topo are only **cTopoObsSet::mParams** (= rotation unknowns for stations):

```
void cBA_Topo::AddToSys(cSetInterUK_MultipeObj<tREAL8> & aSetInterUK)
{
    for (auto& anObsSet: mAllObsSets)
        aSetInterUK.AddOneObj(anObsSet);
}

void cTopoObsSet::PutUknowsInSetInterval()
{
    if (!mParams.empty())
        mSetInterv->AddOneInterv(mParams);
}

void cTopoObsSetStation::OnUpdate()
{
    auto aRotOmega = getRotOmega();
    aRotOmega = mRotVert2Instr.Inverse(aRotOmega); //see cPoseF comments
    mRotVert2Instr = mRotVert2Instr *
        cRotation3D<tREAL8>::RotFromAxiator(aRotOmega);
    updateVertMat(); // update mRotSysCo2Vert with new station position
    // now this have modified rotation, the "delta" is void:
    resetRotOmega();
}
```

Adding new observations to a **cResolSysNonLinear**:

```
void cBA_Topo::AddTopoEquations(cResolSysNonLinear<tREAL8> & aSys)
{
  for (auto &obsSet: mAllObsSets)
    for (size_t i=0;i<obsSet->nbObs();++i)
    {
      cTopoObs* obs = obsSet->getObs(i);
      auto equation = getEquation(obs->getType());
      aSys.CalcAndAddObs(equation, obs->getIndices(),
                         obs->getVals(), obs->getWeights());
```

Getting unknowns indices for an observation:

```
std::vector<int> cTopoObs::getIndices() const
{
  std::vector<int> indices;
  switch (mSet->getType()) {
  case eTopoObsSetType::eStation:
  {
    cTopoObsSetStation* set = dynamic_cast<cTopoObsSetStation*>(mSet);
    ... // checks

    set->getPtOrigin()->getUK()->PushIndexes(indices);

    set->PushIndexes(indices, set->mParams.data(), 3);

    cObjWithUnkowns<tREAL8>* toUk =
        mBA_Topo->getPoint(mPtsNames[1]).getUK();
    int nbIndBefore = indices.size();
    toUk->PushIndexes(indices);

    break;
  }
  ...
  return indices;
}
```

Getting “observation/context” for an observation:

```
std::vector<tREAL8> cTopoObs::getVals() const
{
  std::vector<tREAL8> vals;
  switch (mSet->getType()) {
  case eTopoObsSetType::eStation:
  {
    cTopoObsSetStation* set = dynamic_cast<cTopoObsSetStation*>(mSet);
    ... // checks

    set->PushRotObs(vals);
    if (mType==eTopoObsType::eZen)
      ...   vals.push_back(ref_cor);
    vals.insert(std::end(vals),std::begin(mMeas),std::end(mMeas));
    break;
  }
  ...
  return vals;
}

void cTopoObsSetStation::PushRotObs(std::vector<double> & aVObs) const
{    // fill aPoseRTL2Instr
    (mRotVert2Instr * mRotSysCo2Vert).Mat().PushByCol(aVObs);
}
```

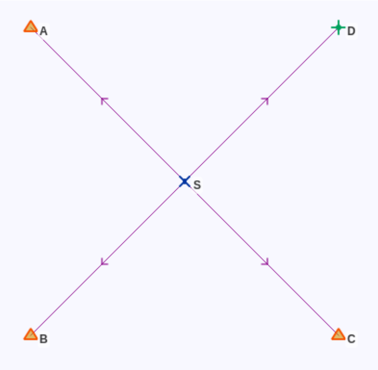
Direct Dev

- refraction parameter
- TopoW parameter
- initializations
  - resection
  - centering
  - basc ori
- new bench
- Code 4
- unknown refraction
- #CAM
- codes 3 and 4 outside of cTopoObsSetStation

Refraction parameter

Resection

To add resection initialisation, start with an example dataset:



Resection


```
* COR
 1 A  90 110 100 0.001 0.001 0.001
 1 B  90  90 100 0.001 0.001 0.001
 1 C 110  90 100 0.001 0.001 0.001

* OBS
 7  S  A    0 0.001
 5  S  B  300 0.001
 5  S  C  200 0.001
 5  S  D  100 0.001

 6  S  A  100 0.001
 6  S  B  100 0.001
 6  S  C  100 0.001
 6  S  D  100 0.001

 3  S  A   14.140 0.001
 3  S  B   14.140 0.001
 3  S  C   14.140 0.001
 3  S  D   14.140 0.001
```

Resection

Resection algorithm:

- find 3 hz obs to an init point from the same vericalized station
- make sure the 3 hz obs are different enought
- find a zen obs to an init point from a vericalized station
- use a complicated formula: https://www.aftopo.org/lexique/relevement-sur-trois-points-calcul-dun/ (RELÈVEMENT BARYCENTRIQUE)
- implement in *MMVII/src/Topo/topoinit.cpp*, call in *cBA_Topo::tryInit()*

Code 4

For ellipsoid height difference observation, the equation is:

$$residual = (H_{to} - H_{from}) - l$$

To convert the points RTL coordinates into ellipsoid heights, the fist step is to convert them to geocentric (X, Y, Z) and then use the formula from *Bowring, 1985, The accuracy of geodetic latitude and height equations*
geodesie.ign.fr/contenu/fichiers/documentation/pedagogiques
/TransformationsCoordonneesGeodesiques.pdf

3.3 Coordonnées cartésiennes (X,Y,Z) ▶ Coordonnées géographiques (λ,φ,h)

3.3.1 Formules de conversion

$$f = 1 - \sqrt{1 - e^2}$$

$$R = \sqrt{X^2 + Y^2 + Z^2}$$

$$\lambda = ATAN2(Y, X)$$

$$\mu = arctg\left[\frac{Z}{\sqrt{X^2 + Y^2}} \cdot \left((1-f) + \left(\frac{e^2 a}{R}\right)\right)\right]$$

$$\varphi = arctg\left[\frac{Z(1-f) + e^2 a \sin^3 \mu}{(1-f)\left[\sqrt{X^2 + Y^2} - e^2 a \cos^3 \mu\right]}\right]$$

$$h = \left[\sqrt{X^2 + Y^2} \cdot \cos\varphi\right] + \left[Z \sin\varphi\right] - \left[a\sqrt{1 - e^2 \sin^2 \varphi}\right]$$

Source : Bowring, 1985, The accuracy of geodetic latitude and height equations, *Survey Review, 28, pp202-206 (modifié pour l'expression de λ)*

Where a and e are constants from the ellipsoid.

Links

Links

- https://github.com/micmacIGN/micmac/wiki/MMVII-prog-session-2024-03-Satellite-Bundle-Adjustment