

CFTM 用户手册

版本 1.0

CFTM User Manual Version 1.0

李鑫沅, 丁明涛 编

更新日期 2024.06.24

地学与卫星大数据研究中心

Big Data Center for Geosciences and Satellites

长安大学

Chang'an University

Copyright © 2024, 地学与卫星大数据研究中心, 长安大学

期刊论文: Xinlong Li, Mingtao Ding*, Zhenhong Li*, Peng Cui, 2024, Common-Feature-Track-Matching approach for multi epoch UAV photogrammetry co-registration, ISPRS Journal of Photogrammetry and Remote Sensing (under review).

专利号: ZL 202311672166.3

在线文档 (中文版): <https://blog.csdn.net/LXLng/article/details/136598055>

在线文档 (英文版): <https://blog.csdn.net/LXLng/article/details/134613468>

Github 项目: <https://github.com/lixinlong1998/CoSfM>

技术支持: 李鑫沆, Tel: +86 15583308860, Email: xinlong.li@chd.edu.cn

简介

CFTM 是一个用于多时相无人机摄影测量高精度配准的工具。

CFTM 通过 Agisoft Metashape Python API 和 COLMAP (可选的) 实现。

CFTM 是一种互运动恢复结构 (CoSfM) 方法，可以在进行摄影测量的同时，配准多期航测像片集。利用我们新颖的匹配策略——“特征轨迹匹配”和特殊的优化算法，它能够生成密集且分布均匀的共连接点 (Common Tie Points, CTPs)，从而实现航测像片的高精度互配准。CFTM 的输出是优化后的像片位姿，这意味着后续生成的地形产品 (例如数字高程模型 (Digital Elevation Model, DEM)、数字正射影像 (Digital Orthophoto Map, DOM) 和三维网格模型) 直接对齐在同一个坐标系下，非常便于后续的高精度时空分析。

如果你在研究中使用了本工具，请引用文献:

```
@Article{XinlongLi2024CFTM,  
  author      = {Xinlong Li, Mingtao Ding, Zhenhong Li, Peng Cui},  
  title       = {Common-Feature-Track-Matching approach for multi epoch UAV  
photogrammetry co-registration},  
  journal     = {ISPRS Journal of Photogrammetry and Remote Sensing},  
  number      = {X},  
  volume     = {XX},  
  month      = {XXX},  
  year       = {2024},  
  url        = {https://XXX/}  
}
```

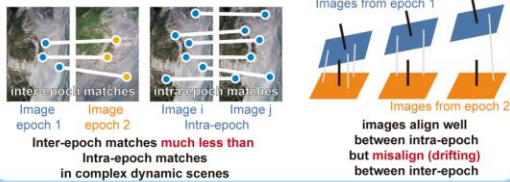
论 文 摘 要

Automatic co-registration of multi-epoch Unmanned Aerial Vehicle (UAV) image sets remains challenging due to the radiometric differences caused by complex dynamic scenes. Specifically, illumination changes and vegetation variations usually lead to insufficient and spatially unevenly distributed common tie points (CTPs), resulting in under-fitting of co-registration near the areas without CTPs. In this paper, we propose a novel Common-Feature-Track-Matching (CFTM) approach for UAV image co-registration, to alleviate the shortage of CTPs with complex dynamic scenes. Instead of matching features between multi-epoch images, we first search correspondences between multi-epoch feature tracks (i.e., groups of features corresponding to the same 3D points), which avoids the removal of matches due to unreliable estimation of the relative pose between inter-epoch image pairs. Then, the CTPs are triangulated from the successfully matched track pairs. Since the even distribution of CTPs is crucial for robust co-registration, a block-based strategy is designed, as well as for parallel computation. Finally, an iterative optimization algorithm is developed to gradually select the best CTPs to refine the poses of multi-epoch images. We assess the performance of our method on two challenging datasets. The results show that CFTM can acquire adequate and evenly distributed CTPs in complex dynamic scenes, thus achieving comparatively high co-registration accuracy approximately four times higher than the state-of-the-art in challenging scenario. Our code is available at <https://github.com/lixinlong1998/CoSfM>.

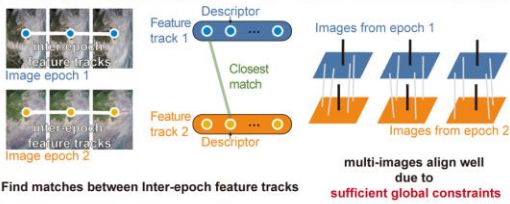
论文图形摘要

Common-Feature-Track-Matching approach for multi epoch UAV photogrammetry co-registration

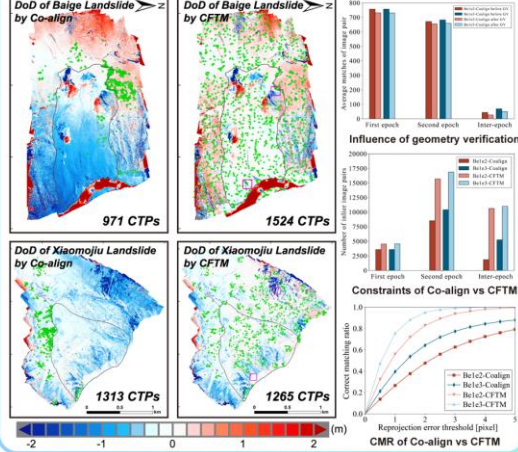
Challenging on co-registration of UAV images



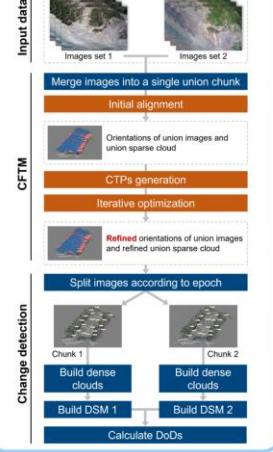
Novel strategy: Matching on feature tracks



Results: Co-align vs CFTM (Ours)



Pipeline of applications



目 录

1 下载资源.....	1
2 安装与配置.....	2
2.1 安装 Agisoft Metashape Pro v1.8.5.....	2
2.2 添加 Metashape.exe 到环境变量.....	3
2.3 安装第三方 python 库	4
2.4 安装 COLMAP (可选但推荐)	8
3 运行示例.....	9
3.1 数据集.....	9
3.2 方法流程.....	10
3.3 程序设置.....	10
3.4 运行步骤.....	12
步骤 1 创建工程文件.....	12
步骤 2 粗配准.....	16
步骤 3 制作检查点（可选的）	17
步骤 4 配置参数.....	18
步骤 6 运行脚本.....	24
步骤 7 评估报告.....	25
4 示例结果.....	26
5 分析与评估.....	27
5.1 分析 CFTM 中的匹配点.....	27
5.2 对比使用 CFTM 之前和之后的匹配点.....	28
6 处理报错.....	30
ERROR: Database: <colmap_database_path> is not found!	30
ERROR: this script could only deal with pairwise co-align!.....	30
ERROR: COLMAPindex should be 'identifier' or 'image_path'!.....	30
7 变量结构.....	31

1 下载资源

源代码链接: <https://github.com/lixinlong1998/CoSfM>

文档 (英文版): <https://blog.csdn.net/LXLng/article/details/134613468>

文档 (中文版): <https://blog.csdn.net/LXLng/article/details/136598055>

数据集:

百度网盘: <https://pan.baidu.com/s/1pQaQGsnx3l-Th9ohO8zUGQ?pwd=yieb>

Google Drive:

2 安装与配置

CFTM v1.0 在 Windows 环境下基于 Agisoft Metashape Pro v1.8.5 软件的 Python API 开发。由于 Metashape Pro v1.8.5 中没有开放特征点和原始特征匹配结果的接口，特征点的提取通过开源的第三方库 OpenCV(默认的)或 COLMAP(可选的)的特征提取功能实现，结果分析借助了 COLMAP 中的特征匹配功能。此外，部分功能依赖第三方库函数，尽管 Metashape Pro v1.8.5 自带的 python3.8 环境预置了部分第三方库函数，但还需要额外安装和/或升级一些第三方库函数。

2.1 安装 Agisoft Metashape Pro v1.8.5

如果您是第一次接触 Metashape Pro, 请注意该软件是一款付费的摄影测量软件，下载是免费的，但永久使用需要购买许可证。Agisoft 公司提供了教育优惠版本的许可证 (Educational License)，详情查看官网购买说明。以下是一些相关链接，便于您快速了解和查阅 Metashape：

- 1 Agisoft 官网: <https://www.agisoft.com/downloads/installer/>
- 2 购买许可: <https://www.agisoft.com/buy/licensing-options/>
- 3 新手教程: <https://www.agisoft.com/support/tutorials/>
- 4 论坛: <https://www.agisoft.com/forum/>
- 5 知识库: <https://agisoft.freshdesk.com/support/solutions>

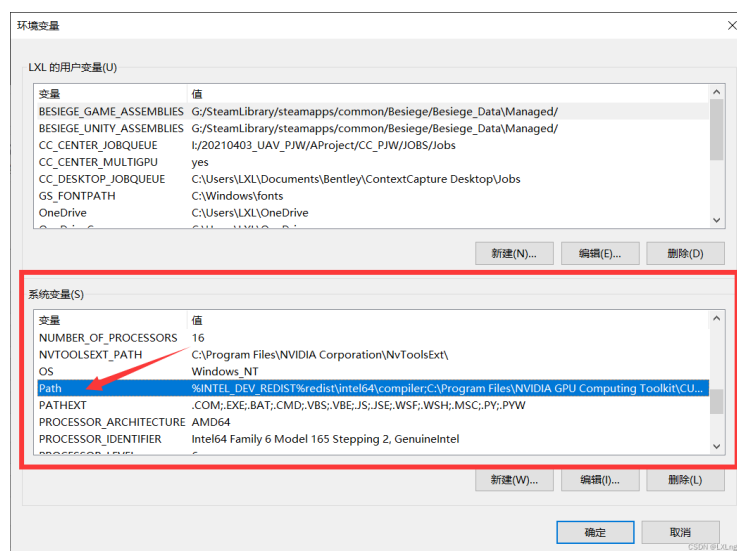
- 访问 Agisoft 官方网站并下载 Metashape Pro v1.8.5 安装程序。注意，官网会随版本更新而下架以往版本，用户可 google 搜索历史版本或者购买许可后联系官方提供的技术支持。
- 以管理员身份运行下载的安装程序，并按照安装向导的指示进行安装。
- 在安装过程中，您可能需要接受许可协议、选择安装路径以及其他选项。我们建议除了安装路径以外的其他选项保持默认即可。

2.2 添加 Metashape.exe 到环境变量

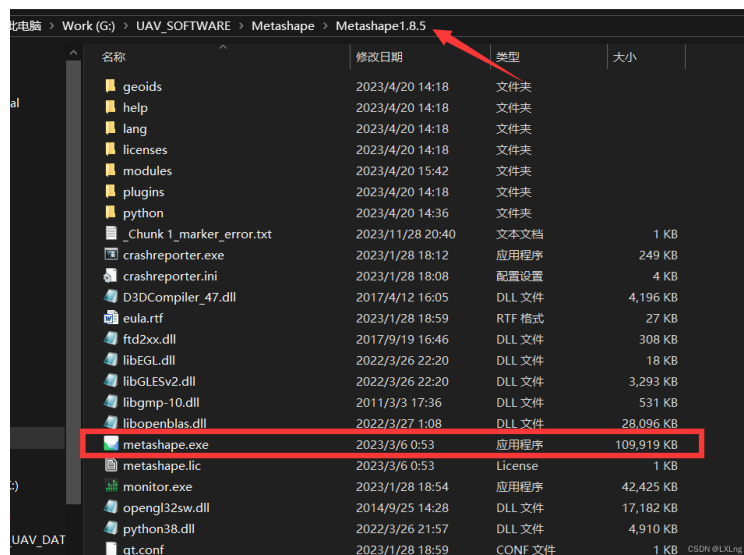
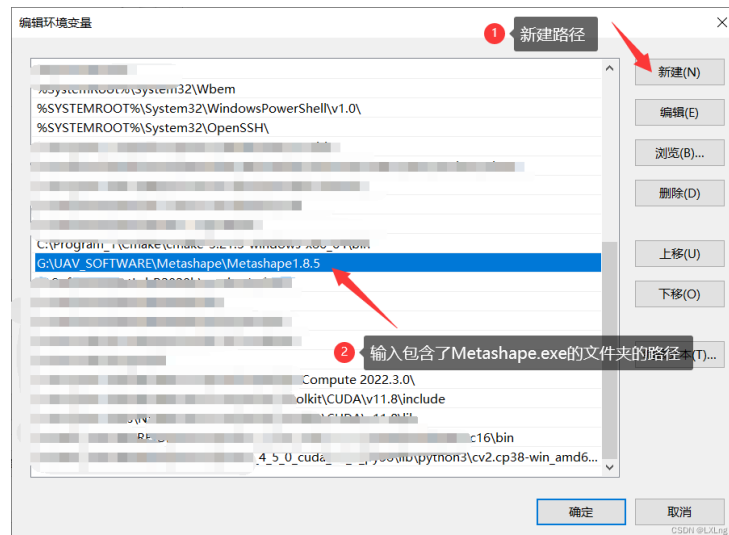
- “此电脑”右键菜单选择“属性”，找到“高级系统设置”，打开面板



- 打开环境变量面板，在“系统变量”中选择“Path”，然后点“编辑”。



- 点击“新建”，然后输入安装路径(即包含了 Metashape.exe 的文件夹的路径)，依次“确定”所有面板，即可完成环境变量设置。



2.3 安装第三方 python 库

官方提供了安装 python 库的教程：[How to install external Python module to Metashape Professional package](#)。我们在这提供 2 种在 Windows10 系统下的指定 python 环境中安装第三方库函数的方法（以 numpy 为例）：

第一种，从源中自动获取：

- 1 Win+R，输入 cmd，回车进入命令行 cmd.exe
- 2 输入 `cd /d G:\UAV_SOFTWARE\Metashape\Metashape1.8.5\python`（注意替换为你的安装路径）
- 3 输入 `python -m pip install numpy` 安装 numpy 库

4 如果找不到库的话，可以尝试在后面指定源网址，例如使用清华源，则输入 `python -m pip install numpy -i https://pypi.tuna.tsinghua.edu.cn/simple/`

第二种，手动下载 whl:

- 1 在网站上下载版本合适的轮子< `numpy-1.24.2-cp38-cp38-win_amd64.whl` >
- 2 Win+R，输入 `cmd`，回车进入命令行 `cmd.exe`
- 3 输入 `cd /d G:\UAV_SOFTWARE\Metashape\Metashape1.8.5\python`,注意替换为你的安装路径
- 4 执行命令 `python -m pip install <轮子存放路径>`

这里列出了所需要安装的第三方库函数，这些库的版本仅在 Metashape Pro v1.8.5 中测试过是合适的。

库名称	版本	推荐方式
numpy	1.23.5	自动获取
scipy	1.10.1	自动获取
pandas	2.0.0	自动获取
networkx	3.1	自动获取
GDAL	GDAL-3.4.3-cp38-cp38-win_amd64.whl	手动下载
opencv-python	4.7	自动获取
opencv-contrib-python	4.7	自动获取
matplotlib	3.7.1	自动获取
python-dateutil	2.8.2	自动获取
pytz	2023.3	自动获取

- 开始安装第三方 python 库。GDAL 需要采用第二种方法安装，在 google 中搜索下载或者在我们 github 项目的 libs 文件中下载 GDAL 的 whl 文件。准备好 GDAL 的安装包后，Win+R 打开命令行，逐行执行下面的命令来安装第三方库 (如果您在安装过程中遇到提示 pip 损坏的相关问题，您可以参考这个解决办法¹⁾)。

- 1 `cd /d G:\UAV_SOFTWARE\Metashape\Metashape1.8.5\python`
- 2 `python -m pip install --upgrade pip`

1 关于 Metashape 自带 python 环境中 pip 损坏后的修复方式:

Win+R 打开命令行，在命令行中逐行运行如下命令，注意将路径替换为你的路径

```
cd /d <G:\UAV_SOFTWARE\Metashape\Metashape1.8.5\python>
```

```
python <...CFTM_v1.0\toolbox\get-pip.py>
```

```
python -m pip install --upgrade pip
```

然后就可以正常使用 pip 了，但是再次打开 Metashape 时，Metashape 会自动更新并恢复至原来的 pip;

如果不希望这样，你可以尝试关掉 Metashape 中的自动更新选项（工具→偏好设置→杂项→程序启动时检查更新）

```

3 python -m pip install numpy==1.23.5
4 python -m pip install scipy==1.10.1
5 python -m pip install pandas==2.0.0
6 python -m pip install matplotlib==3.7.1
7 python -m pip install networkx==3.1
8 python -m pip install python-dateutil==2.8.2
9 python -m pip install pytz==2023.3
10 python -m pip install opencv-python==4.7.0.72 -i
    https://pypi.tuna.tsinghua.edu.cn/simple/
11 python -m pip install opencv-contrib-python==4.7.0.72 -i
    https://pypi.tuna.tsinghua.edu.cn/simple/
12 python -m pip install
    G:\UAV_SOFTWARE\Metashape\Python3Module\GDAL-3.4.3-cp38-cp38-
    win_amd64.whl

```

- 安装完成后，Win+R 打开命令行，在命令行中逐行输入下面的命令，可以查看当前 Metashape 的 Python 环境中所安装的库函数，您可以对照参考库函数列表进行检查。

```

1 cd /d G:\UAV_SOFTWARE\Metashape\Metashape1.8.5\python
2 python -m pip list

```

库	版本号	库	版本号
attrs	23.1.0	backcall	0.2.0
click	8.1.3	colorama	0.4.3
ConfigArgParse	1.5.3	contourpy	1.0.7
cycler	0.11.0	dash	2.9.3
dash-core-components	2.0.0	dash-html-components	2.0.0
dash-table	5.0.0	decorator	4.4.2
fastjsonschema	2.16.3	Flask	2.2.3
fonttools	4.39.3	GDAL	3.4.3

importlib-metadata	6.5.0	importlib-resources	5.12.0
ipykernel	5.3.4	ipython	7.16.1
ipython-genutils	0.2.0	ipywidgets	8.0.6
itsdangerous	2.1.2	jedi	0.17.2
Jinja2	3.1.2	jsonschema	4.17.3
jupyter-client	6.1.6	jupyter-core	4.6.3
jupyterlab-widgets	3.0.7	kiwisolver	1.4.4
llvmlite	0.39.1	MarkupSafe	2.1.2
matplotlib	3.7.1	nbformat	5.7.0
networkx	3.1	numba	0.56.4
numpy	1.23.5	open3d	0.17.0
opencv-contrib-python	4.7.0.72	opencv-python	4.7.0.72
packaging	23.1	pandas	2.0.0
parso	0.7.1	pexpect	4.8.0
pickleshare	0.7.5	Pillow	9.5.0
pip	24.0	pkgutil_resolve_name	1.3.10
plotly	5.14.1	prompt-toolkit	3.0.5
ptyprocess	0.6.0	Pygments	2.6.1
pyparsing	3.0.9	pyrsistent	0.19.3
PySide2	5.15.2.1	python-dateutil	2.8.1
pytz	2023.3	pywin32	228
pyzmq	19.0.1	qtconsole	4.7.5
QtPy	1.9.0	scipy	1.10.1
setuptools	49.2.0	shiboken2	5.15.2.1
shiboken2-generator	5.15.2.1	six	1.15.0
tenacity	8.2.2	tornado	6.0.4
traitlets	4.3.3	tzdata	2023.3
wcwidth	0.2.5	Werkzeug	2.2.3
wheel	0.29.0	widgetsnbextension	4.0.7
zipp	3.15.0		

2.4 安装 COLMAP (可选但推荐)

CFTM v1.0 的特征点提取通过使用开源的第三方库 OpenCV(默认的)或 COLMAP(可选的)的特征提取功能实现，结果分析借助了 COLMAP 中的特征匹配功能。如果您想使用更快的 SIFT-GPU 功能，则需要安装 COLMAP 3.6 版本（本工具与其他 COLMAP 版本的兼容性未经测试），使用时请注意遵守 COLMAP 的许可协议。

安装 COLMAP 的官方安装教程：[官方网站](#)，[GitHub Releases](#)。其他安装教程：[Windows10 下 Colmap 的安装与调试、三维重建实践及中间结果输出](#)。

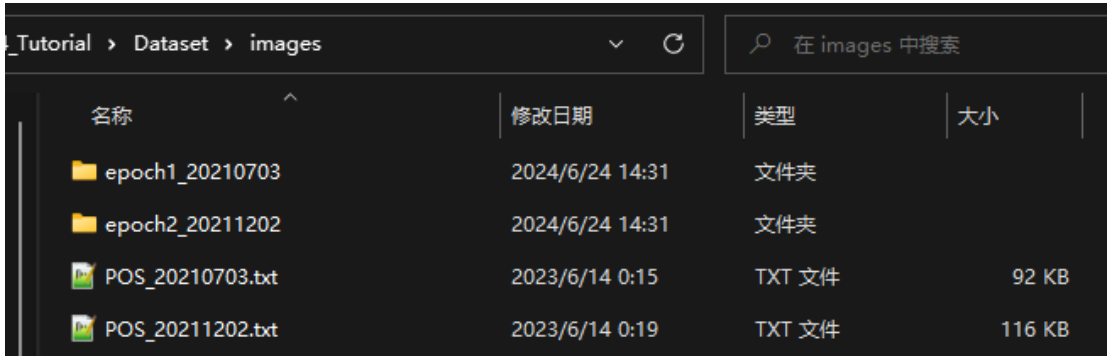
直接下载 COLMAP 3.6 Release: <https://github.com/colmap/colmap/releases/tag/3.6>



点击 COLMAP-3.6-windows-cuda.zip 或 COLMAP-3.6-windows-no-cuda.zip（取决于你的设备是否支持 NVIDIA GPU）进行下载。下载后，你将获得一个压缩文件，然后解压即可。接着，双击 "RUN_TUTORIALS" 以测试环境。如果没有问题，说明 COLMAP 已成功安装，你可以直接双击 "COLMAP" 进入软件的图形界面。请记录 COLMAP.bat 的路径，因为后续配置将需要它。

3 运行示例

解压 Tutorial 压缩包后，可以看到如下图所示的文件。其中，Dataset 中存放了示例数据，Example 为本教程运行后得到的示例结果，供读者直接查看，TryHere 提供了执行 co-alignment 后的工程文件（相当于完成了步骤 1 至 3），以便于更快地直接地体验 CFTM 算法。

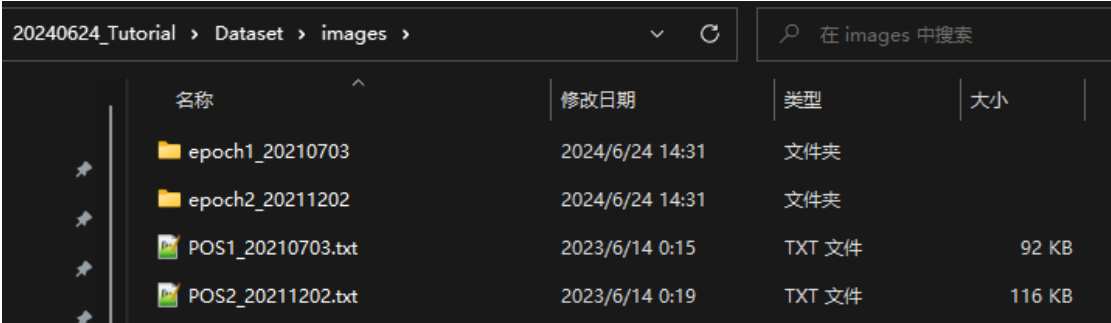


名称	修改日期	类型	大小
epoch1_20210703	2024/6/24 14:31	文件夹	
epoch2_20211202	2024/6/24 14:31	文件夹	
POS_20210703.txt	2023/6/14 0:15	TXT 文件	92 KB
POS_20211202.txt	2023/6/14 0:19	TXT 文件	116 KB

在运行执行本章的示例前，请确认你已经完成了前面所述的安装程序。

3.1 数据集

准备同一测区在 2 个不同时相采集的像片集，我们记为 epoch1 和 epoch2；在光学航测相机拍摄影像时，机载 GNSS/IMU 定位系统同时获取相机的光心地理定位和相机姿态角，合称为“POS 数据”，我们记为 POS1 和 POS2。注意，有些无人机厂商（例如大疆 DJI）将 POS 信息直接写入像片的 EXIF，而有些厂商（例如飞马 Feima）则以独立的 csv 文件存放 POS 信息，此处我们以后者为例。

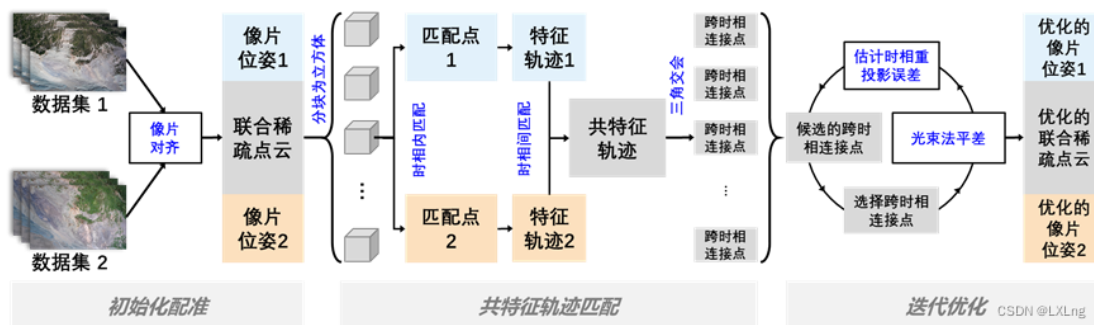


名称	修改日期	类型	大小
epoch1_20210703	2024/6/24 14:31	文件夹	
epoch2_20211202	2024/6/24 14:31	文件夹	
POS1_20210703.txt	2023/6/14 0:15	TXT 文件	92 KB
POS2_20211202.txt	2023/6/14 0:19	TXT 文件	116 KB

3.2 方法流程

这里我们简要叙述一下整套方法的工作流程，具体细节请看论文的“Section 3 Methodology”，简要流程如下：

- 1 获取多个不同时相的像片数据集；
- 2 对多个不同时相的像片数据集中的像片进行粗配准，得到概略像片位姿和联合稀疏点云；
- 3 将所述联合稀疏点云分割为固定大小的网格；
- 4 计算每个网格的外包立方体，将所述外包立方体投影到对应的像片数据集中得到掩膜多边形，以提取掩膜多边形的特征点子集；
- 5 根据所述特征点子集构建每个网格中不同时相的特征轨迹；
- 6 对两个不同时相的特征轨迹进行特征匹配，将距离最近的同名特征作为共特征轨迹；
- 7 通过三角交会，从所述共特征轨迹中构建共连接点；
- 8 计算所述共连接点的时相重投影误差 ERE，并进行迭代，直至迭代收敛，则得到配准后的像片位姿。



3.3 程序设置

配置代码路径，打开 CFTM 代码文件夹，找到“config.py”双击打开。首先，我们需要在 SETUP 区域设置路径：

- 1 `PATH_CODE = CFTM 代码文件夹的路径`
例如：`PATH_CODE = r'D: \CoSfM\Release\CFTM_v1.0'`

2 `PATH_COLMAP_BAT = COLMAP.bat` 的路径

例如: `PATH_COLMAP_BAT = r'D:\Software\COLMAP\COLMAP-3.6-windows-cuda\COLMAP.bat'`

下面两个路径仅限于您的设备有 GPU 且需要启用 OpenCV 的 SIFT-GPU 功能时才需要设置:

3 `PATH_CUDA = CUDA` 的路径 ()

例如: `PATH_CUDA = r'C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v11.8\bin'`

4 `PATH_OPENCV = Windows` 下将 CUDA 绑定到 OpenCV 时所用的路径

例如: `PATH_OPENCV = r'G:\OpenCV\opencv_4_5_0_cuda_11_1_py38\install\x64\vc16\bin'`

随后分别打开 `..\CFTM_v1.0\toolbox` 中的脚本文件:

1 `CheckPoints_Analyse.py`

2 `CheckPoints_Import.py`

3 `ICTPs_Analyse.py`

4 `ICTPs_Delete.py`

5 `Process_SplitChunk.py`

检查每个脚本文件的 `import` 部分, 将 `sys.path.append()` 中的路径设置为 CFTM 代码文件夹的路径。例如:

1 `sys.path.append(r'D:\Research\20221223_CoSfM\Release\CFTM_v1.0')`

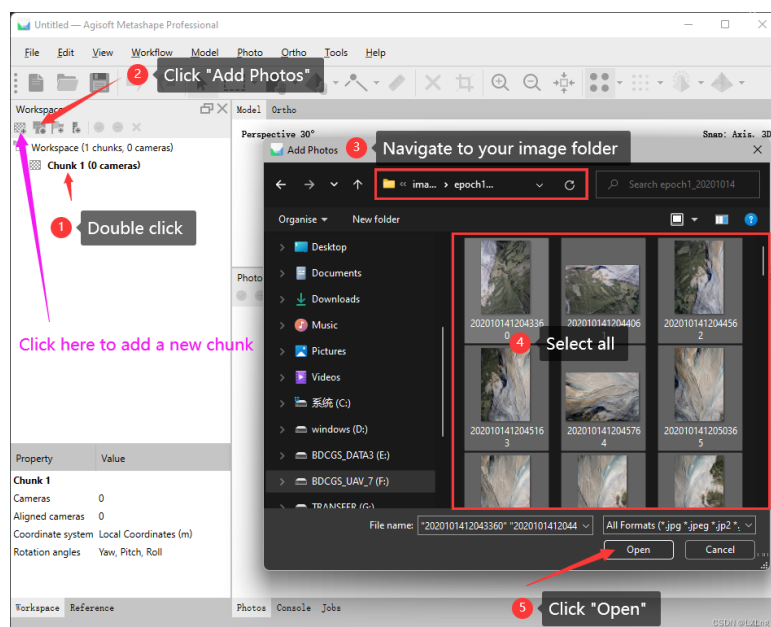
如果您使用 Pycharm 打开 CFTM 项目, 您可以使用全局替换功能来替换这些路径。此外, Pycharm 项目的 python 解释器需要设置为 Metashape 安装文件下的 `python.exe`, 例如 “`D:\Software\Metashape\python\python.exe`”。

3.4 运行步骤

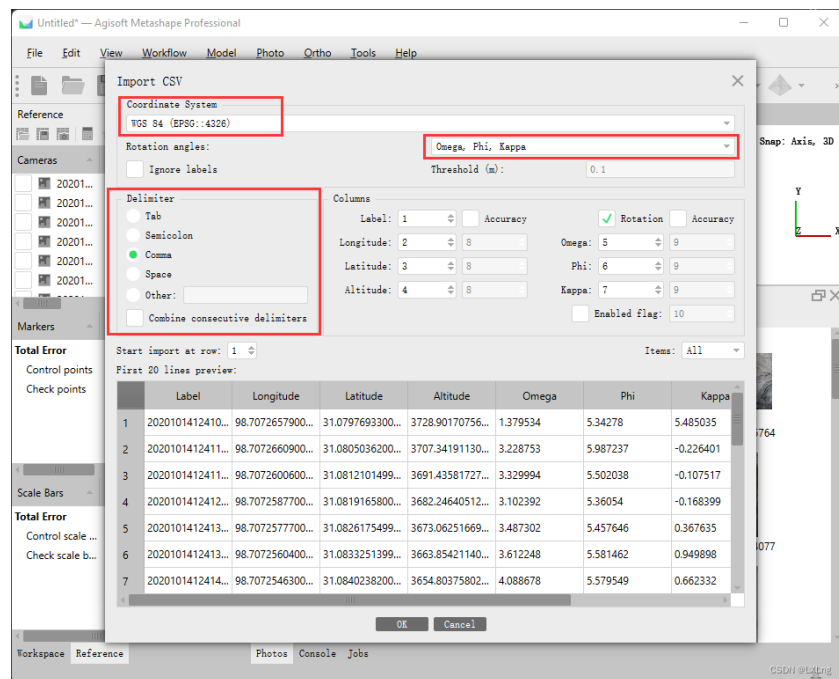
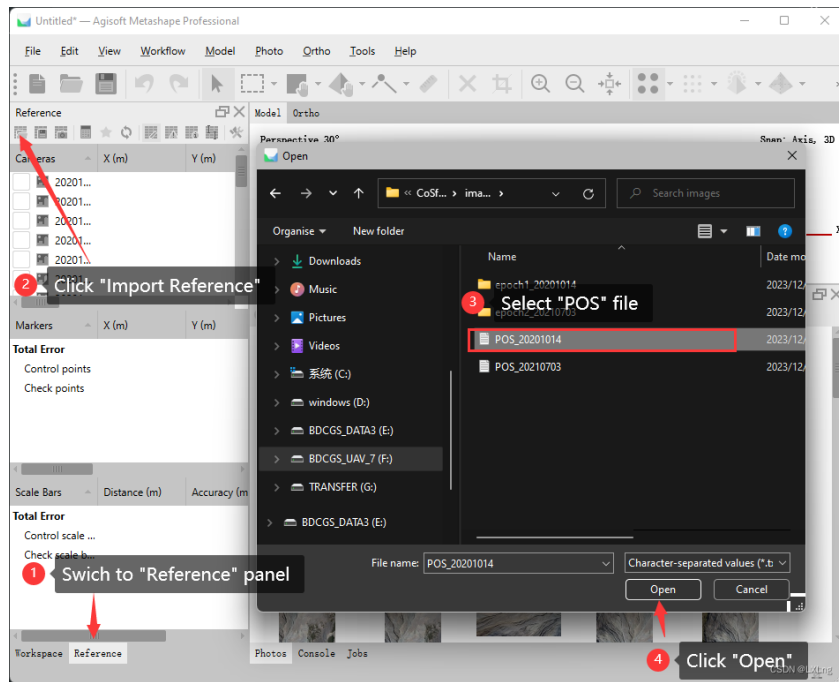
步骤1 创建工程文件

打开 Metashape.exe，我们需要创建工程文件，并将两期数据导入到工程中。在工作区双击 chunk1 选中堆块，点击添加照片按钮。在弹出的文件管理器中，转到“...\Tutorial\Dataset\images\epoch1_20201014”，Ctrl+A 选择所有像片，点击打开，等待照片加载。

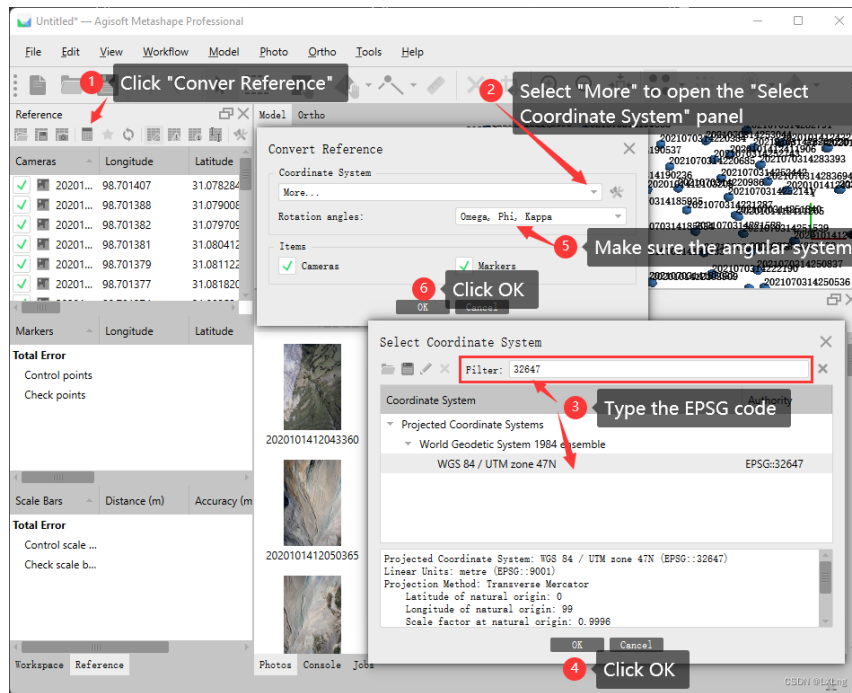
对于第二期数据，我们点击添加堆块按钮，然后双击 chunk 2 选中堆块后，以相同的方法添加第二期像片，文件路径是“...\Tutorial\Dataset\images\epoch2_20210703”。



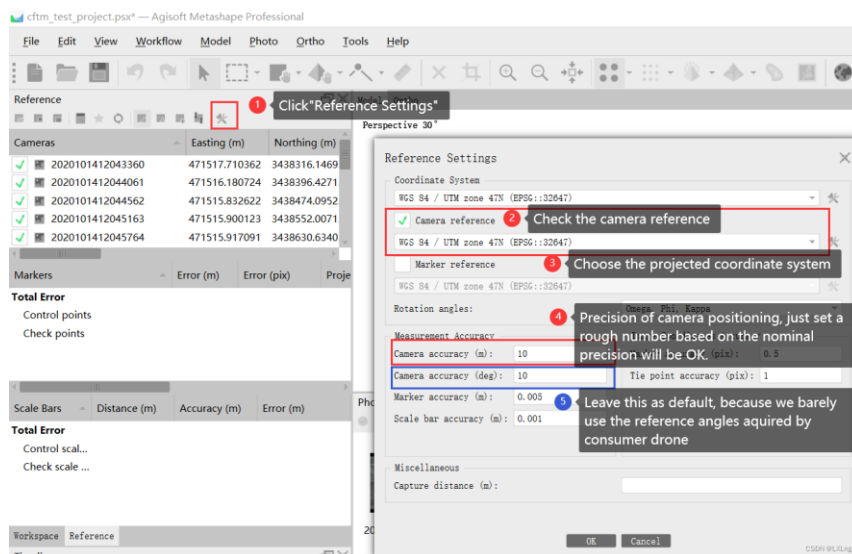
完成照片导入后，需要为每期像片导入 POS 数据。下面以 Chunk1 为例，在工作区中，先双击选中 Chunk1，然后将工作区面板切换至参考面板。点击导入参考，选择对应日期的 POS 数据，在弹出的选项框中按下图指示选择坐标系统、转角系统、分隔符、标签和精度。



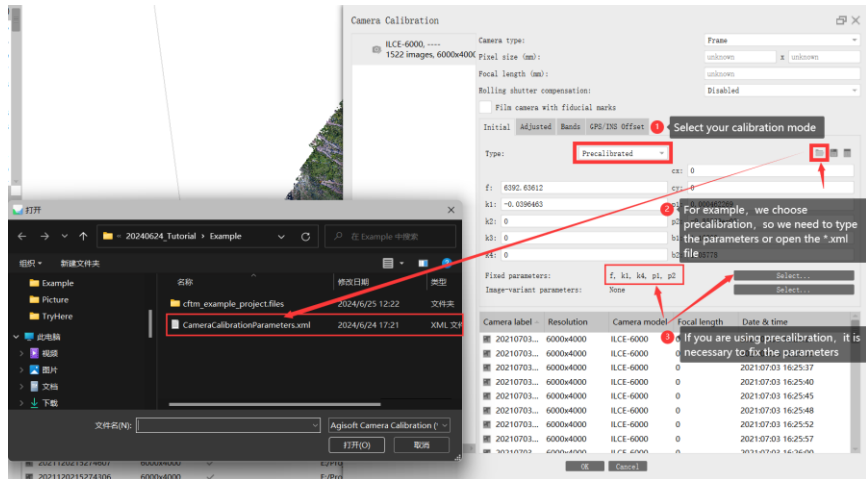
为了便于计算，需要将 chunk 的坐标系设置为投影坐标系。按照下图所示，点击“转换参考坐标系”，在弹出的选项框中选择目标坐标系。此处根据经度换算，我们选择 WGS 84/UTM zone 47N 投影坐标系。其余设置保持默认。



设置参考参数和精度参数。如下图所示，点击参考设置，勾选 camera reference，参考坐标系选择 WGS 84/UTM zone 47N。相机精度设置为 10，这是无人机机载 GPS 的标称精度，其余值保持默认即可。

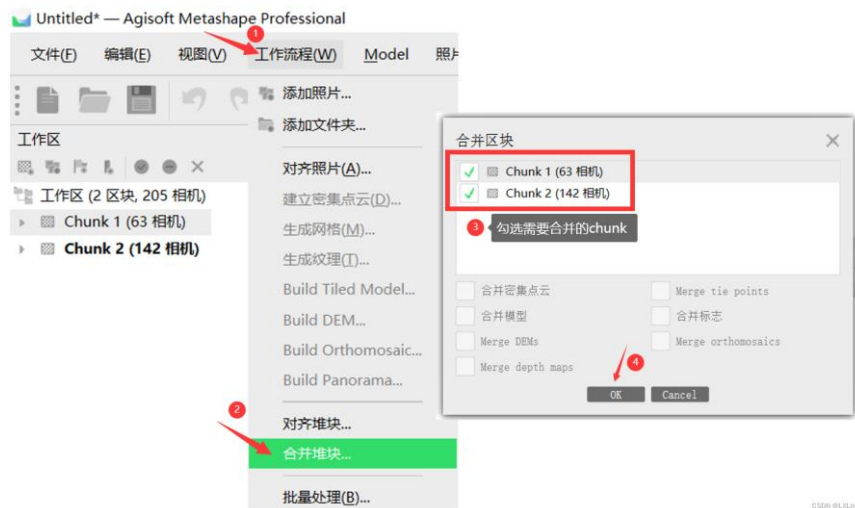


在主菜单中选择 Tools，选择 Camera Calibration...，在弹出的选项框中配置相机的检校参数。首先，选择检校类型。若选择自检校，则只需要选择需要固定的参数（即保持为 0 的参数项，这意味着不检校该参数）；若选择预检校（本示例为这种），则需要手动输入参数或者打开参数文件(在..\Tutorial\Example\下)，然后固定相应参数，防止过拟合。

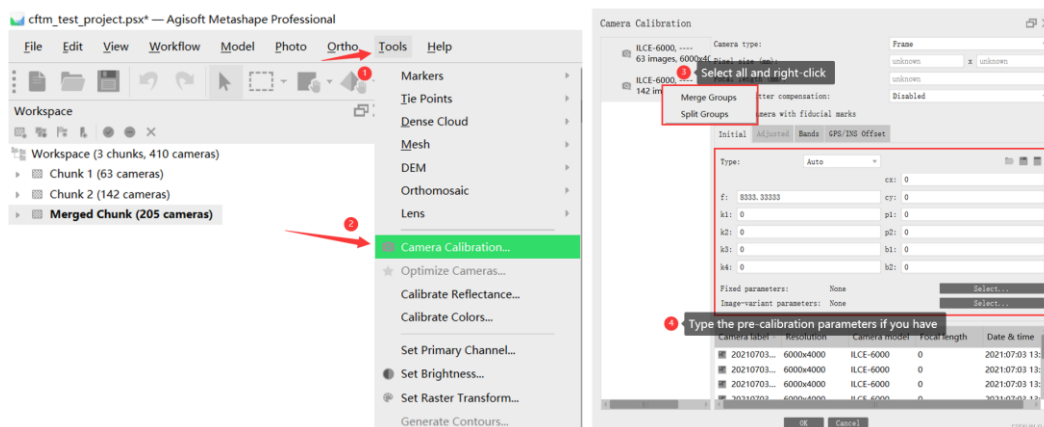


至此，我们完成了 Chunk1 的数据导入，重复这些步骤，完成 Chunk2 的数据导入。完成后，将工程文件保存到“...\Tutorial\Example”（这是本示例的工作空间文件夹，后续 CFTM 的过程文件会存储在与工程文件同目录的文件夹下），命名为“cftm_example_project.psx”。

若你的单期数据中有多个架次需要分别添加，可以先将每个架次的数据单独添加到 chunk 中，然后合并为一个 chunk。具体步骤是在主菜单栏中点击工作流程，选择合并堆块...，在弹出的选项框中，勾选要合并的 chunk，然后点击确定。

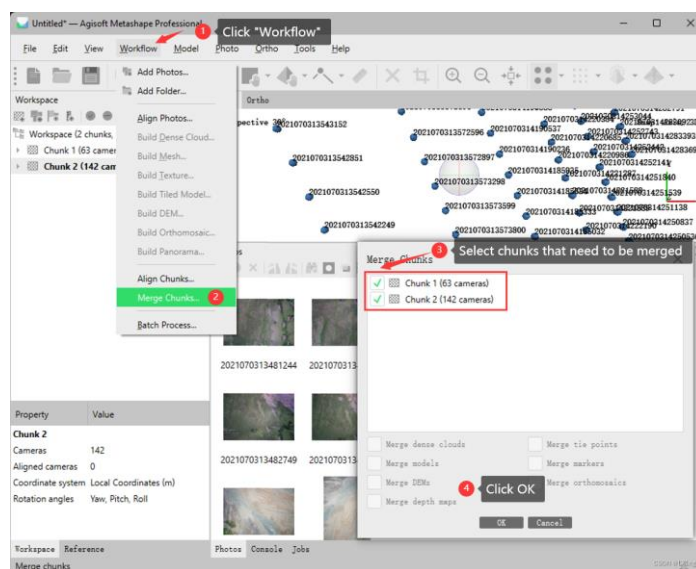


需要注意的是，多个 chunk 合并时 Tools→Camera Calibration...里会有多个相机模型，可以将其全部选中后，右键菜单选择 Merge Groups，这样合并的 chunk 就只对应一个相机模型。

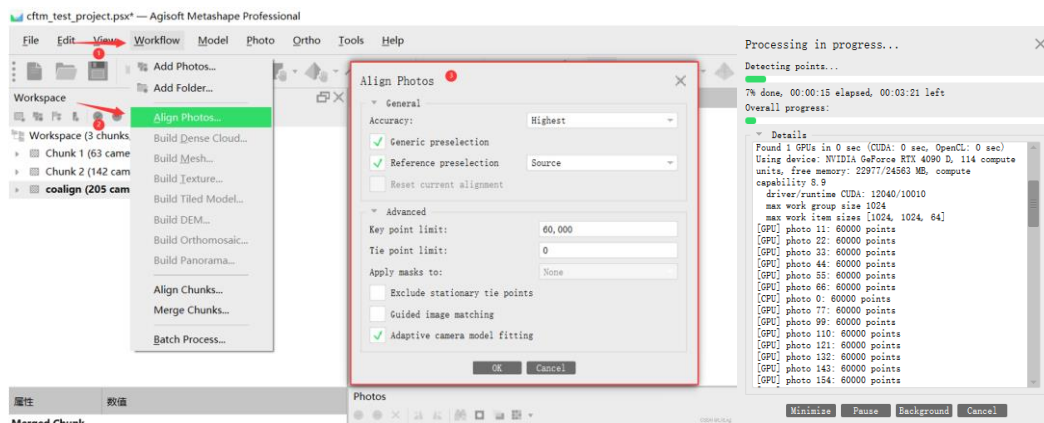


步骤2 粗配准

使用 Co-alignment 算法进行配准，需要先将两个堆块合并为一个堆块，然后执行 SfM 算法。具体步骤是在主菜单栏中点击工作流程，选择合并堆块...，在弹出的选项框中，勾选 Chunk1 和 Chunk2，然后点击 OK，这样我们就得到了名为 Merge 的堆块（可以重命名为"coalign"）。



如下图所示，在主菜单中的 Workflow 中选择 Align Photos...，在弹出的选项框中根据下图设置参数，即精度为最高、启用预选择、关键点数量限制 60000，无连接点限制，启用自适应的相机模型拟合。点击 OK 后开始对齐照片。



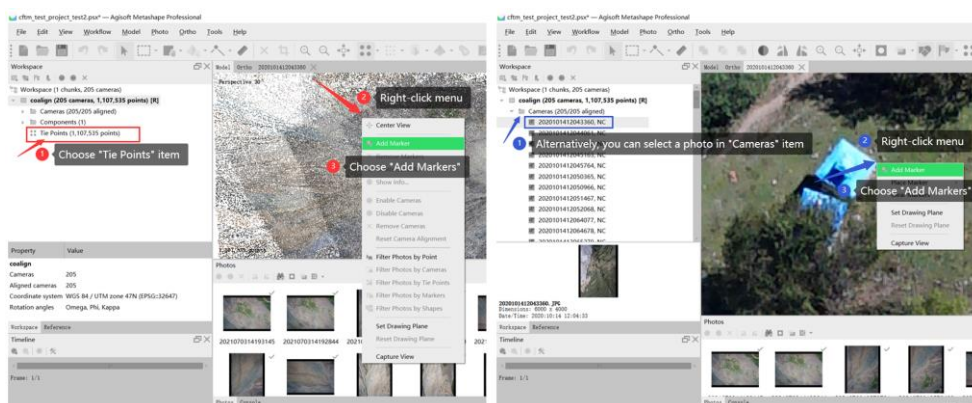
待运行完成后，我们实现了利用 co-alignment 对数据进行粗配准，其结果是合并的稀疏点云和像片位姿。

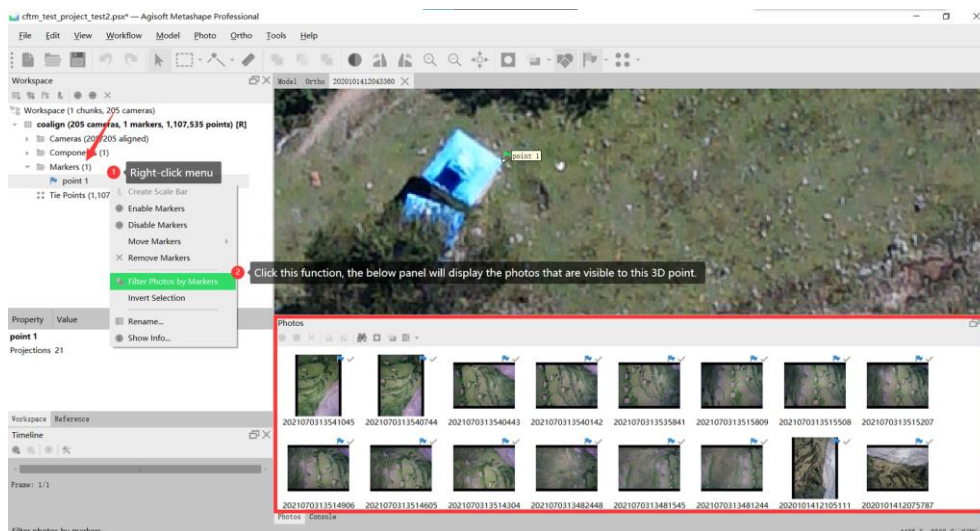
步骤3 制作检查点 (可选的)

在稀疏点云或者照片集中寻找稳定的地面特征点，并在特征点的位置上右键菜单，选择“Add Markers”。添加的 marker 可以在左侧工作区的“Markers”分组中找到。

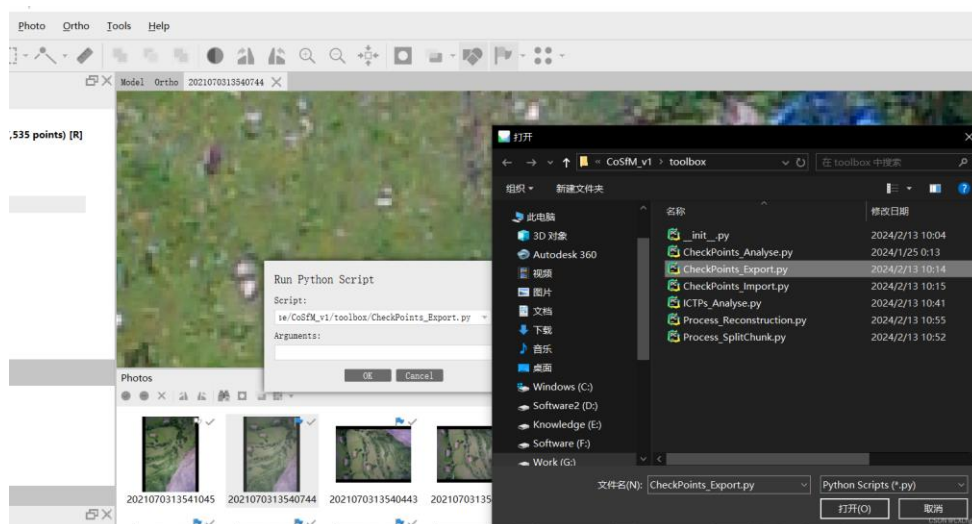
我们需要进行精准刺点，选中新添加的 marker（例如此处名为 point 1），右键菜单中选择“Filter Photos by Markers”后，在 Photos 面板中会出现该点对应的可视像片。双击第一张像片，然后在放大的面板中拖动标签到准确的位置，小旗子的颜色从蓝色变为绿色（在 Metashape 中，Marker 作为控制点时，绿色和蓝色分别表示在优化时启用和不启用该图像点；但是作为检查点时，只有右键菜单选择 Remove Marker 后，该图像点才会失效，否则会被视为有效图像点参与计算）。然后继续对下一张像片进行刺点，尽可能保证两个时相中有大致相等数量的像片被刺点。

重复上述检查点制作步骤，目标是在稳定区域中均匀地选择不少于 5 个检查点。





Ctrl+R 弹出“Run Python Script”运行脚本对话框，选择 CFTM 代码包下的…
 \toolbox\CheckPoints_Export.py 程序，点击 OK，随后检查点数据文件将导出在工程文件的同目录下（例如此处名为 cftm_test_project_CPsdatabase.txt）



最后，由于 Metashape 中光束法平差时 Marker 会被视为 tie point 参与计算，为了避免影响自动配准结果，我们需要移除刚刚创建的这些检查点 Markers（不用担心，我们已经保存了检查点数据文件，后续可以使用 CFTM 代码包下的…
 \toolbox\CheckPoints_Import.py 程序来将这些点重新导入回工程）。

步骤4 配置参数

我们需要配置 Run.py 中的“SETUP”区域的参数。在 Run.py 中，大部分参数按照下图保持默认，而 CFTM_args 参数需要根据实际情况设置。在 CFTM_args 中，

需要设置的关键参数包括: `gird_size_1`, `gird_size_2`, `num_nominated`, `num_selected`, `num_max_iterations`, `pool_size`, `threshold_Distance`。

- 1 **`gird_size`** 是指将初始稀疏点云按此大小分割为方格。**`gird_size_1`** 设置的方格是为了约束匹配范围, 加速匹配效率, 所以不宜太大, 可保持默认; **`gird_size_1`** 设置的方格是为了保证空间均匀性, 但是需要防止算法为了满足空间均匀而选择低质量的点, 建议 **`grid_size_2`** 大于 **`grid_size_1`**。
- 2 算法会先在全局选出 ERE 最小的 **`num_nominated×m×n`** 的 CTPs 作为候选点, 然后为每个网格尝试从候选点集中选出 **`num_selected`** 个 ERE 最小的 CTPs。因此最终得到的 CTPs 是小于 **`num_selected×m×n`** 的。
- 3 **`num_max_iterations`** 是最大迭代次数。根据我们的经验, 如果粗配准效果相对较好, 那么迭代次数可能是个位数, 一般而言, 我们建议设为 30。
- 4 **`pool_size`** 需要根据你的 CPU 核心数量来设置; 例如 Intel i7-10875H 有 16 个核心, 那么此处的 **`pool_size`** 最大可以设为 16, (如果出现内存不足或者阻塞了其他程序, 您可以降低 **`pool_size`** 的数值)。
- 5 **`threshold_Distance`** 是指如果一对跨时相特征轨迹各自对应的三维点之间的距离超过了该阈值, 那么算法会忽略该对跨时相特征轨迹, 也即初始配准最大偏移量的预估值。根据我们的经验, **`threshold_Distance`** 应该至少大于粗配准精度。

```
# Open a project file
workspace_path = r"...\Tutorial\Example"
project_name = "cftm_example_project.psx"
images_path = r'...\Tutorial\Dataset\images'
check_points_path =
r"...\Tutorial\Dataset\CPsDatabase.txt"

# please make sure your major version is:
compatible_major_version = "1.8"
```

```

# Process arguments
process_args = {
    # Choose the processing options for CFTM.
    "process_coalign": False,
    "process_feature_extraction": True,
    "process_CTPs_generation": True,
    "process_iterative_optimization": True,
    "process_analysis": True,
    "process_analysis_matches": False,
    "process_compare_matches": False,

    # Choose the processing options for reconstruction.
    "process_reconstruction": True,
    "process_build_DSM": True,
    "process_build_DOM": False,
    "process_build_TiledModel": False,
    "process_export_DSM": True,
    "process_export_DOM": False,
    "process_export_TiledModel": False,
    "process_white_balance": False,

    "save_project_each_step": True,
}

# Select the mode for dividing epochs.
# If it's "DATE", it will divide them by day based on
the dates of the photos.

```

```

# If it's "FOLDER", it will divide them based on the
folders where the photos are stored.
epoch_mode = 'DATE' # or 'FOLDER'

# Which feature extraction mode do you want to use?
# 0 : use OpenCV to extract SIFT feature without cuda
(default but slow);
# 1 : use OpenCV to extract SIFT-GPU feature with cuda
(need additional manual compilation);
# 2 : use COLMAP to extract SIFT-GPU feature
(recommended).
feature_extraction_mode = 2

# Whether to continue the run from the last breakpoint?
begin_from_breakpoint = True

# Specify which chunk to process; leave empty will
automatically select the chunk which has maximum images.
coalign_chunk_name = ''

# Generate CTPs for all grids or only for those grid
without CTPs.
for_all_grids = True

# If CTPs are generated for all grids, do you what to
reuse the CTPs generated by co-alignment?
reuse_ICTPs = True

```

```

# Whether to use *.shp to mask the unstable areas?
unstable_areas_mask_path = ''

# Common Feature Track Matching
CFTM_args = {
    "gird_size_1": 32,
    "gird_size_2": 64,
    "scoring": [0, 0, 0, 0, 0, 0, 2, 4, 8, 16],
    "num_nominated": 10,
    "num_selected": 5,
    "num_max_iterations": 20,
    "pool_size": 16, # the number should in line with
your CPU cores
    "criterias_1": [2, 3],
    "criterias_2": [1, 3],
    "ratio_inlier_init": 0.25,
    "confidence": 0.9,
    "threshold_RepError": 0.5,
    "threshold_Distance": 3,
    "skip_edge": False,
    "threshold_Termination": 0.001,
    "num_inertia": 3,
}

# Make sure the coordinates of the chunk are in the
projected coordinate system
reference_args = {
    "PJCS_EPSG": 32647,

```

```

    "camera_accuracy_m": 2.0,
    "camera_accuracy_deg": 10.0,
    "marker_accuracy_m": 20.0,
    "marker_accuracy_pix": 0.5,
    "tiepoint_accuracy_pix": 1,
    "camera_reference_enabled": True,
    "marker_reference_enabled": False,
    "rotation_system": 'opk'
}

# Bundle Adjustment Arguments
bundle_adjustment_args = {
    "fit_f": True,
    "fit_cx": True, "fit_cy": True,
    "fit_b1": False, "fit_b2": False,
    "fit_k1": True, "fit_k2": True,
    "fit_k3": True, "fit_k4": False,
    "fit_p1": True, "fit_p2": True,
    "fit_corrections": False,
    "adaptive_fitting": True,
    "tiepoint_covariance": True
}

# Photogrammetry arguments
photogrammetry_args = {
    # Align photos
    "quality_SPC": 1, # 1(UltraHigh), 2(High), 4(Medium)
    "keypoint_limit": 60000,

```

```

    "keypoint_limit_Per_Megapixel": 1000,
    "tiepoint_limit": 0,

    # Error reduction
    "reprojection_error": 0.3,
    "reconstruction_uncertainty": 15,
    "projection_accuracy": 5,
    "max_ratio_remove_points": 0.3,

    # Dense matching
    "quality_DPC": 2,
    "dense_cloud_FilterMode": 'Mild',

    # Build tiled model
    "tiled_model_TiledModelFormat": 'Cesium',
    "tiled_model_pixel_size": 0.3, # metres
    "tiled_model_face_count": 20000,
    "tiled_model_ModelFormat": 'None',
    "tiled_model_ImageFormat": 'JPEG',
}

```

步骤6 运行脚本

键盘输入 Win+R 打开 cmd.exe, 输入下面命令来运行脚本, 注意将路径替换为您的 Run.py 路径, 回车后开始运行 CFTM。

```
1 metashape.exe -r D:\CoSfM\Release\CFTM_v1.0\Run.py
```

CFTM 具有断点记录功能。若发生意外情况, 在修正后, 您可以使用上述命令来继续运行 CFTM, CFTM 会从上一个断点处继续运行。您也可以通过手动修改... Tutorial\Example\cftm\state.json 文件中的“NO”和“DONE”来单独控制每个环节是否已经运行完成; 如果为“NO”则会继续运行该部分, 如果为“DONE”则

会跳过该部分。

... Tutorial\Example\cftm\state.json 文件示例:

```
{"coalign": "NO", "CFTM_creat_task": "DONE", "feature_extraction": "DONE",  
"CFTM_run_task": "DONE", "IO_mergeCTP": "DONE", "IO_iterater": "DONE",  
"IO_implementCTPs": "DONE", "analysis_CTPs": "DONE", "analysis_CPs":  
"DONE", "analysis_matches": "NO", "compare_matches": "NO", "reconstruction":  
"NO", "finished_grids": [], "finished_iterations": []}
```

步骤7 评估报告

在 CFTM 程序结束后，您可以在 `cftm_example_project.psx` 同目录下的“...\\Tutorial\\Example\\cftm”文件夹中看到所有过程文件和结果。

您可以在...\\cftm\\Reports 文件夹中找到生成的 GCTPs 及其质量指标，其中“_GCTPsQua.txt”是 GCTPs 的数据列表，“_GCTPsQuaSta.txt”是基于 GCTPs 数据的统计指标。

若您在之前制作了检查点，并且在 `Run.py` 中配置了检查点文件路径，那么您可以在...\\cftm\\Reports 文件夹中找到 CPs 质量指标，其格式与 GCTPs 相同。

4 示例结果

5 分析与评估

5.1 分析 CFTM 中的匹配点

如果在 Run.py 中将 process_analysis_matches 设置为 True，则 CFTM 会在…
\cftm\Reports 文件夹下生成 MatchesReport_CFTM.txt 文件

文件结构及解释如下：

- [Epoch1_PairNum, Epoch2_PairNum, Epoch1_MatchNum, Epoch2_MatchNum]
时相 e 中的像对数量，时相 e 中的匹配数量
- [Epoch1_PairMatcNum_AVG, Epoch2_PairMatcNum_AVG]
时相 e 中平均每个像对的匹配数量
- [Epoch1_PairInlierMatcRatio_AVG, Epoch2_PairInlierMatcRatio_AVG]
时相 e 中平均每个像对的有效匹配数量
- [Common_PairNum, Common_MatchNum1, Common_MatchNum2, Common_MatchPortion, Common_PairMatcNum_AVG]
跨时相的像对数量，跨时相匹配数量，跨时相匹配数量，跨时相匹配在所有有效匹配中的占比，平均每个跨时相像对的正确匹配数量
- [Epoch1_TrackNum, Epoch2_TrackNum, Epoch1_InlierTrackNum, Epoch2_InlierTrackNum]
时相 e 中的特征轨迹数量，时相 e 中的有效特征轨迹数量
- [Common_TrackNum, Common_TrackCrosMatcNum_AVG]
跨时相的特征轨迹数量，平均每个跨时相特征轨迹中的匹配数量

5.2 对比使用 CFTM 之前和之后的匹配点

如果在 Run.py 中将 process_compare_matches 设置为 True，则 CFTM 会在...
\cftm\Reports 文件夹下生成 MatchesReport_Coalign.txt 文件
文件结构及解释如下：

- [Epoch1_PairNum, Epoch2_PairNum, Common_PairNum]
像对数量
- [Epoch1_MatchNum, Epoch2_MatchNum, Common_MatchNum]
匹配数量
- [Epoch1_PairMatchNum_AVG, Epoch2_PairMatchNum_AVG, Common_PairMatchNum_AVG]
平均每个像对的匹配数量
- [Epoch1_PairInlierMatchNum_AVG, Epoch2_PairInlierMatchNum_AVG, Common_PairInlierMatchNum_AVG]
平均每个像对的有效匹配数量
- [Epoch1_PairInlierMatchRatio_WAVG, Epoch2_PairInlierMatchRatio_WAVG, Common_PairInlierMatchRatio_WAVG]
加权平均每个像对的有效匹配数量
- [Epoch1_PairInlierMatchRatio_AVG, Epoch2_PairInlierMatchRatio_AVG, Common_PairInlierMatchRatio_AVG]
平均每个像对的有效匹配率（忽略有效率为 0 的像对）
- [Epoch1_MatchPortion, Epoch2_MatchPortion, Common_MatchPortion]

时相 1 匹配在所有匹配中的占比，时相 2 匹配在所有匹配中的占比，跨时相匹配在所有匹配中的占比。

■ [Epoch1_InlierMatchPortion, Epoch2_InlierMatchPortion, Common_InlierMatchPortion]
平均每个像对的有效匹配率

■ [Epoch1_TrackNum, Epoch2_TrackNum, Common_TrackNum]
特征轨迹数量

■ [Epoch1_ValidTrackNum, Epoch2_ValidTrackNum, Common_ValidTrackNum]
有效特征轨迹数量

■ [Epoch1_ERValidTrackNum, Epoch2_ERValidTrackNum, Common_ERValidTrackNum]
误差剔除后，剩余的有效特征轨迹数量

■ [Epoch1_ValidTrackPortion, Epoch2_ValidTrackPortion, Common_ValidTrackPortion]
特征轨迹有效率

■ [Epoch1_ERValidTrackPortion, Epoch2_ERValidTrackPortion, Common_ERValidTrackPortion]
误差剔除后特征轨迹有效率

6 处理报错

ERROR: Database: <colmap_database_path> is not found!

SOLUTION: 没有找到 COLMAP 数据库，这一步是在 Compare Matches 时，没有找到在 Feature Extraction 中创建的 COLMAP 文件，您可能在运行 CFTM 的过程中删除或者移动了这部分文件。您可以通过修改 `...\\cftm\\state.json`，将 `"feature_extraction": "DONE"` 改为 `"feature_extraction": "NO"`，然后重新运行 CFTM；或者将 `process_compare_matches` 设置为 `False`。

ERROR: this script could only deal with pairwise co-align!

SOLUTION: CFTM 目前只能处理 2 个时相数据之间的配准，请确保输入的数据来自于两个时相。关于时相的划分模式(变量“`epoch_mode`”), 若您选择"DATE", 请检查是否每个时相中的像片日期为同一天；若您选择"FOLDER", 请检查变量“`images_path`”文件夹下的子文件夹(即每个时相对应的像片文件夹)数量为 2。

ERROR: COLMAPindex should be ‘identifier’ or ‘image_path’!

SOLUTION: 请确认 Metashape 中存储的像片路径是否与 COLMAP 中存储的像片路径是否一致，这通常发生在硬盘重新连接时，盘符发生了变化的情况。

7 变量结构

- Cameras = {camera_id:Camera,...}
 - camera_id = int
 - Camera = [transform, reference, covariance, sensors_id, state, path, identifier]
 - transform= [camera_transform, camera_loc, camera_rot, camera_transform PJCS]
 - reference = [camera_loc_Ref,location_accuracy,camera_rot_Ref,rotation_accuracy]
 - covariance = [camera_location_covariance, camera_rotation_covariance]
 - states = [label,camera_loc_Ref,camera_rot_Ref, selected, camera_orientation]
- Points = {point_id:Point,...}
 - point_id = int
 - Point = [Coordinates, Covariance, StandardDeviation, Quality, selected, valid, Track_id]
 - Coordinates=[[X, Y, Z],[X, Y, Z]] # in CLCS & PJCS respectively
 - Covariance = matrix # with 3x3 covariances in mm
 - StandardDeviation = [stdX, stdY, stdZ] # in mm
 - Quality = [RE, RU, PA, IC]
- Tracks = [Track,...] # index=Track_id, length=len(chunk.point_cloud.Tracks)
 - Track = [view,...] # index=view_id, length=number of views
 - view = [camera_id, project_info]
 - project_info = [projection.coord[0], projection.coord[1], projection.size, projection.track_id]
- Sensors = {sensor_id:Sensor,...}
 - sensor_id = int
 - Sensor = [calibration_params, calibration_matrix]
 - calibration_params = [f, cx, cy, b1, b2, k1, k2, k3, k4, p1, p2, p3, p4, height, width]
 - calibration_matrix = matrix # nxn covariance matrix, where n is the number of estimating parameters.
- Projections = {identifier:Camera_Projections,...}
 - Camera_Projections= [projection_info,...]
 - projection_info = [u, v, size, Track_id]
- CoordinateTransform = [M, LSE, T, R, S]
 - M = ndarray # 4x4 ndarray
 - LSE = ndarray # 4x4 ndarray
 - T = ndarray # 4x4 ndarray
 - R = ndarray # 3x3 ndarray
 - S = float
- CoordinateAttribute = [crs_name, crs_PJCS_wkt, crs_GCCS_wkt, crs_GDCS_wkt]
 - crs_PJCS_wkt = string # Projection coordinate system definition in WKT format.
 - crs_GCCS_wkt = string # Geocentric coordinate system definition in WKT format.
 - crs_GDCS_wkt = string # Geodetic coordinate system definition in WKT format.
- identifier = int
 - unique number for image, consist of photographing time and image name (e.g. an image taking at '2021:04:04 13:40:13' with name 'DSC00009.JPG', then its identifier is 2021040413401309).
- cameraPaths = {camera.photo.path:identifier,...}
 - camera.photo.path = string
 - identifier = int
- camera_ids = {identifier:camera_id,...}
 - identifier = int
 - camera_id = int
- point_ids = {Track_id :point_id,...} # if the track faild in built a point, the point_id will be -1
 - Track_id = int
 - point_id = int
- validTracks = [Track,...] # index=validTrack_id
 - Track = [view,...] # index=view_id, length=number of views
 - view = [camera_id, project_info]
 - project_info = [projection.coord[0], projection.coord[1], projection.size, projection.track_id]
- validTracks_ids = {validTrack_id:Track_id,...}
 - validTrack_id= int
 - Track_id= int
- pair_ids = {pair_id:match state,...}

- pair_id = (int, int)
- match_state = [all, valid, invalid]
 - all = int # the total matches number
 - valid = int # the valid matches number
 - invalid = int # the invalid matches number
- match_ids = {pair_id:[valid_matches, invalid_matches],...}
 - pair_id = (int, int) # (camera_id1, camera_id2)
 - valid_matches = [Match,...]
 - invalid_matches = [Match,...]
 - Match = ([proj_u,proj_v,size,track_id],[proj_u,proj_v,size,track_id])
- chunkKeypoints = {identifier:Keypoints,...}
 - identifier = int
 - Keypoints = ndarray # dtype=float32, shape=(numKeypoints, 6)
 - Keypoint = [u,v,affinity1, affinity2, affinity3, affinity4]
- chunkDescriptors = {identifier:Descriptors,...}
 - identifier = int
 - Descriptors = ndarray # dtype=uint8, shape=(numKeypoints, 128)
- Boundary_CLCS = [boundary_X_MAX,boundary_X_MIN,boundary_Y_MAX,boundary_Y_MIN]
 - the boundary of sparse point cloud in XY plane, in meter
 - boundary_X_MAX = float
 - boundary_X_MIN = float
 - boundary_Y_MAX = float
 - boundary_Y_MIN = float
- Boundary_PJCS = [boundary_X_MAX,boundary_X_MIN,boundary_Y_MAX,boundary_Y_MIN]
 - the boundary of sparse point cloud in XY plane, in meter
 - boundary_X_MAX = float
 - boundary_X_MIN = float
 - boundary_Y_MAX = float
 - boundary_Y_MIN = float
- epochs = [datelist,...]
 - datelist = [date,...] # a datalist contains the date for an epoch.
 - date = 'YYYY:MM:DD'
 - for example: epochs = [['2020:10:14'], ['2021:07:03'], ['2021:12:03'], ['2021:12:06']]
- epochNum = [ei,...] # index=epoch_id, length=epochs number
 - ei = int # the number of images from the specified epoch
 - for example: epochNum = [e1, e2, e3, e4]
- signal = [si, ...]
 - si = int # Using 0 or 1 to represent whether there is image from the specified epoch
 - for example: signal = [0, 0, 1, 1]
- CamerasEpoch = [epoch_id,...] # index=camera_id, length=cameras number
 - epoch_id = int # assign which epochs dose camera belongs to
- TracksEpoch = [epochNum,...] # index=Track_id, length=Tracks number
 - epochNum = [e1, e2, e3, e4] # the counts for images of a certain epochs in a Track.
- EpochTracks = [Track,...] # index=EpochTracks_id (accompanied by EpochTracks_ids)
 - extract the Track that contains images from different epochs based on the TracksEpoch
 - Track = [view,...] # index=view_id, length=number of views
 - view = [camera_id, project_info]
 - project_info = [projection.coord[0], projection.coord[1], projection.size, projection.track_id]
- EpochTracks_ids = [Track_id,...] # index=EpochTracks_id
 - Since the EpochTracks extract from the Track_ids, the index of EpochTracks is different from Track_ids. So the original Track_id index of it is recorded in this associated variable.
- CommonTiePoint_IdList = [point_id, ...] # index=CommonTiePoint_id, length=CommonTiePoints number
 - analysis Common Tie Points (CTPs) based on TracksEpoch and extract the index of epoch tie points to this list.
 - point_id = int
- CommonTiePoint_Signal = [signal,...] # index=CommonTiePoint_id, length=CommonTiePoints number
 - analysis common tie points(CTPS) based on TracksEpoch and assign the signal for each common tie point.
 - signal = list
- CTPsCoord_CLCS = [Coordinates,...] # length=CommonTiePoints number
 - Coordinates = [X, Y, Z] # in CLCS
- CTPsCoord_PJCS = [Coordinates,...] # length=CommonTiePoints number
 - Coordinates = [X, Y, Z] # in PJCS
- EpochPairs
- EpochMatches

- PointsGrid = {grid_id: Point_IdList}
 - point_id list of each cell
 - grid_id = (r, c)
 - Point_IdList = [point_id,...]
- CTPsGrid = matrix # shape=(r, c), index=r, c
 - number of CTP in each cell
 - CTP number = int
- PointsGrid_NoCTPs = {grid_id: Point_IdList}
 - point_id list of the cell that contains no CTPs
 - grid_id = (r, c)
 - Point_IdList = [point_id,...]
- GridDots
- Cubes = { grid_id: conners}
 - grid_id = (r, c)
 - conners = [topLeft, topRight, leftBottom, rightBottom, upper, lower]
- CubesPolygon
- features_cube_ei = {grid_id: conners}
 - similar to Cubes, but only contains the camera from ei
 - grid_id = (r, c)
 - conners = [topLeft, topRight, leftBottom, rightBottom, upper, lower]
- features_cube_camera_id = [keypointList,...] # index=keypoints+1
 - keypointList = [identifier, keypoint_id, keypoint_id, ..., keypoint_id]
 - identifier = int
 - keypoint_id = int
- CameraMask_cube = {identifier: polygon,...}
 - identifier = int
 - polygon = ndarray # 2x2 the 2D coordinates of conners in the image plane
- matches_cube_e1 = [matches_cube_pair_id,...]
 - matches_cube_pair_id = list
- MatchesReport_cube_e1 = [pair,...]
 - pair = [(camera_id1, camera_id2), Matches, inlierMatches]
 - Matches = np.array([[keypoint_id, keypoint_id, distance], ...])
 - inlierMatches = np.array([[keypoint_id, keypoint_id, distance], ...])
- TracksReport_cube_e1 = [TrackReport,...]
 - TrackReport = [views, edges]
 - views = np.array([[camera_id, keypoint_id], ...])
 - edges = np.array([[view_id, view_id, distance], ...])
- OriginTracksReport_cube_e1 = [TrackReport,...]
 - TrackReport = [views, edges]
 - views = np.array([[camera_id, keypoint_id], ...])
 - edges = np.array([[view_id, view_id, distance], ...])
- OriginTracksReport_cube_e1 = [matches_cube_pair_id,...]
 - matches_cube_pair_id = list
- matches_cube_pair_id = [item,...] # length = matches+1
 - item = [(camera_id1, camera_id2), match, match, ..., match]
 - match = [feature, feature, matchQuality]
 - matchQuality = [distance, point3D]
 - feature = [camera_id, proj_info, keypoint_id]
 - proj_info = [proj_u, proj_v]
- matches_cube_pair_verified = [item,...] # length = matches+1
 - item = [(camera_id1, camera_id2), match, match, ..., match]
 - match = [feature, feature, matchQuality]
 - matchQuality = [distance, point3D]
 - feature = [camera_id, proj_info, keypoint_id]
 - proj_info = [proj_u, proj_v]
- epochMatches = [match,...] # length = matches
 - match = [feature, feature]
 - feature = [camera_id, proj_info, keypoint_id]
 - proj_info = [proj_u, proj_v]
- epochMatchesQuality = [distance,...] # index = by epochMatches
 - record the Euclidean distance between the matched two descriptors.
 - distance = float
- epochTracksMatches_cube = [epochTracksMatches,...]
 - the descriptors of common feature track.
 - epochTracksMatches = [view_id,...]

- view_id = (camera_id, keypoints_id)
- epochTracksMatches_cube_Quality = [epochMatchesQuality,...] # index = by epochTracksMatches
 - the matches and similarity measures of common feature track.
 - epochMatchesQuality= [distance ,...]
 - distance = float
- Tracks_ViewId = {view_id,...}
 - record the Euclidean distance between the matched two descriptors.
 - view_id = (camera_id, keypoints_id)
- Camera_IdList = [camera_id,...]
 - The visible camera list for a certain cube.
 - camera_id = int
- Camera_IdList_cube = {grid_id:Camera_IdList}
 - The visible camera list for each cube.
 - grid_id = int
 - Camera_IdList = [camera_id,...]
- Cameras = {camera_id:Camera,...}
 - camera_id = int
 - Camera = [transform, reference, covariance, sensors_id, state, path, identifier]
 - transform= [camera_transform, camera_loc, camera_rot, camera_transform_PJCS]
 - reference = [camera_loc_Ref,location_accuracy,camera_rot_Ref,rotation_accuracy]
 - covariance = [camera_location_covariance, camera_rotation_covariance]
 - states = [label,camera_loc_Ref,camera_rot_Ref, selected, camera_orientation]
- Points = {point_id:Point,...}
 - point_id = int
 - Point = [Coordinates, Covariance, StandardDeviation, Quality, selected, valid, Track_id]
 - Coordinates=[[X, Y, Z],[X, Y, Z]] # in CLCS & PJCS respectively
 - Covariance = matrix # with 3x3 covariances in mm
 - StandardDeviation = [stdX, stdY, stdZ] # in mm
 - Quality = [RE, RU, PA, IC]
- Tracks = [Track,...] # index=Track_id, length=len(chunk.point_cloud.Tracks)
 - Track = [view,...] # index=view_id, length=number of views
 - view = [camera_id, project_info]
 - project_info = [projection.coord[0], projection.coord[1], projection.size, projection.track_id]
- Sensors = {sensor_id:Sensor,...}
 - sensor_id = int
 - Sensor = [calibration_params, calibration_matrix]
 - calibration_params = [f, cx, cy, b1, b2, k1, k2, k3, k4, p1, p2, p3, p4, height, width]
 - calibration_matrix = matrix # nxn covariance matrix, where n is the number of estimating parameters.
- Projections = {identifier:Camera_Projections,...}
 - Camera_Projections= [projection_info,...]
 - projection_info = [u, v, size, Track_id]
- CoordinateTransform = [M, LSE, T, R, S]
 - M = ndarray # 4x4 ndarray
 - LSE = ndarray # 4x4 ndarray
 - T = ndarray # 4x4 ndarray
 - R = ndarray # 3x3 ndarray
 - S = float
- CoordinateAttribute = [crs_name, crs_PJCS_wkt, crs_GCCS_wkt, crs_GDCS_wkt]
 - crs_PJCS_wkt = string # Projection coordinate system definition in WKT format.
 - crs_GCCS_wkt = string # Geocentric coordinate system definition in WKT format.
 - crs_GDCS_wkt = string # Geodetic coordinate system definition in WKT format.
- identifier = int
 - unique number for image, consist of photographing time and image name (e.g. an image taking at '2021:04:04 13:40:13' with name 'DSC00009.JPG', then its identifier is 2021040413401309).
- cameraPaths = {camera.photo.path:identifier,...}
 - camera.photo.path = string
 - identifier = int
- camera_ids = {identifier:camera_id,...}
 - identifier = int
 - camera_id = int
- point_ids = {Track_id :point_id,...} # if the track faild in built a point, the point_id will be -1
 - Track_id = int
 - point_id = int

- validTracks = [Track,...] # index=validTrack_id
 - Track = [view,...] # index=view_id, length=number of views
 - view = [camera_id, project_info]
 - project_info = [projection.coord[0], projection.coord[1], projection.size, projection.track_id]
- validTracks_ids = {validTrack_id:Track_id,...}
 - validTrack_id= int
 - Track_id= int
- pair_ids = {pair_id:match_state,...}
 - pair_id = (int, int)
 - match_state = [all, valid, invalid]
 - all = int # the total matches number
 - valid = int # the valid matches number
 - invalid = int # the invalid matches number
- match_ids = {pair_id:[valid_matches, invalid_matches],...}
 - pair_id = (int, int) # (camera_id1, camera_id2)
 - valid_matches = [Match,...]
 - invalid_matches = [Match,...]
 - Match = ([proj_u,proj_v,size,track_id],[proj_u,proj_v,size,track_id])
- chunkKeypoints = {identifier:Keypoints,...}
 - identifier = int
 - Keypoints = ndarray # dtype=float32, shape=(numKeypoints, 6)
 - Keypoint = [u,v,affinity1, affinity2, affinity3, affinity4]
- chunkDescriptors = {identifier:Descriptors,...}
 - identifier= int
 - Descriptors = ndarray # dtype=uint8, shape=(numKeypoints, 128)
- Boundary_CLCS = [boundary_X_MAX,boundary_X_MIN,boundary_Y_MAX,boundary_Y_MIN]
 - the boundary of sparse point cloud in XY plane, in meter
 - boundary_X_MAX = float
 - boundary_X_MIN = float
 - boundary_Y_MAX = float
 - boundary_Y_MIN = float
- Boundary_PJCS = [boundary_X_MAX,boundary_X_MIN,boundary_Y_MAX,boundary_Y_MIN]
 - the boundary of sparse point cloud in XY plane, in meter
 - boundary_X_MAX = float
 - boundary_X_MIN = float
 - boundary_Y_MAX = float
 - boundary_Y_MIN = float
- epochs = [datelist,...]
 - datelist = [date,...] # a datalist contains the date for an epoch.
 - date = 'YYYY:MM:DD'
 - for example: epochs = [['2020:10:14'], ['2021:07:03'], ['2021:12:03'], ['2021:12:06']]
- epochNum = [ei,...] # index=epoch_id, length=epochs number
 - ei = int # the number of images from the specified epoch
 - for example: epochNum= [e1, e2, e3, e4]
- signal = [si, ...]
 - si = int # Using 0 or 1 to represent whether there is image from the specified epoch
 - for example: signal = [0, 0, 1, 1]
- CamerasEpoch = [epoch_id,...] # index=camera_id, length=cameras number
 - epoch_id = int # assign which epochs dose camera belongs to
- TracksEpoch = [epochNum,...] # index=Track_id, length=Tracks number
 - epochNum = [e1, e2, e3, e4] # the counts for images of a certain epochs in a Track.
- EpochTracks = [Track,...] # index=EpochTracks_id (accompanied by EpochTracks_ids)
 - extract the Track that contains images from different epochs based on the TracksEpoch
 - Track = [view,...] # index=view_id, length=number of views
 - view = [camera_id, project_info]
 - project_info = [projection.coord[0], projection.coord[1], projection.size, projection.track_id]
- EpochTracks_ids = [Track_id,...] # index=EpochTracks_id
 - Since the EpochTracks extract from the Track_ids, the index of EpochTracks is different from Track_ids. So the original Track_id index of it is recorded in this associated variable.
- CommonTiePoint_IdList = [point_id, ...] # index=CommonTiePoint_id, length=CommonTiePoints number
 - analysis Common Tie Points (CTPs) based on TracksEpoch and extract the index of epoch tie points to this list.
 - point_id = int
- CommonTiePoint_Signal = [signal,...] # index=CommonTiePoint_id, length=CommonTiePoints number

- analysis common tie points(CTPs) based on TracksEpoch and assign the signal for each common tie point.
- signal = list
- CTPsCoord_CLCS = [Coordinates,...] # length=CommonTiePoints number
 - Coordinates = [X, Y, Z] # in CLCS
- CTPsCoord_PJCS = [Coordinates,...] # length=CommonTiePoints number
 - Coordinates = [X, Y, Z] # in PJCS
- EpochPairs
- EpochMatches
- PointsGrid = {grid_id: Point_IdList}
 - point_id list of each cell
 - grid_id = (r, c)
 - Point_IdList = [point_id,...]
- CTPsGrid = matrix # shape=(r, c), index=r, c
 - number of CTP in each cell
 - CTP number = int
- PointsGrid_NoCTPs = {grid_id: Point_IdList}
 - point_id list of the cell that contains no CTPs
 - grid_id = (r, c)
 - Point_IdList = [point_id,...]
- GridDots
- Cubes = { grid_id: conners}
 - grid_id = (r, c)
 - conners = [topLeft, topRight, leftBottom, rightBottom, upper, lower]
- CubesPolygon
- features_cube_ei = {grid_id: conners}
 - similar to Cubes, but only contains the camera from ei
 - grid_id = (r, c)
 - conners = [topLeft, topRight, leftBottom, rightBottom, upper, lower]
- features_cube_camera_id = [keypointList,...] # index=keypoints+1
 - keypointList = [identifier, keypoint_id, keypoint_id, ..., keypoint_id]
 - identifier = int
 - keypoint_id = int
- CameraMask_cube = {identifier: polygon,...}
 - identifier = int
 - polygon = ndarray # 2x2 the 2D coordinates of conners in the image plane
- matches_cube_e1 = [matches_cube_pair_id,...]
 - matches_cube_pair_id = list
- MatchesReport_cube_e1 = [pair,...]
 - pair = [(camera_id1, camera_id2), Matches, inlierMatches]
 - Matches = np.array([[keypoint_id, keypoint_id, distance], ...])
 - inlierMatches = np.array([[keypoint_id, keypoint_id, distance], ...])
- TracksReport_cube_e1 = [TrackReport,...]
 - TrackReport = [views, edges]
 - views = np.array([[camera_id, keypoint_id], ...])
 - edges = np.array([[view_id, view_id, distance], ...])
- OriginTracksReport_cube_e1 = [TrackReport,...]
 - TrackReport = [views, edges]
 - views = np.array([[camera_id, keypoint_id], ...])
 - edges = np.array([[view_id, view_id, distance], ...])
- OriginTracksReport_cube_e1 = [matches_cube_pair_id,...]
 - matches_cube_pair_id = list
- matches_cube_pair_id = [item,...] # length = matches+1
 - item = [(camera_id1, camera_id2), match, match, ..., match]
 - match = [feature, feature, matchQuality]
 - matchQuality = [distance, point3D]
 - feature = [camera_id, proj_info, keypoint_id]
 - proj_info = [proj_u, proj_v]
- matches_cube_pair_verified = [item,...] # length = matches+1
 - item = [(camera_id1, camera_id2), match, match, ..., match]
 - match = [feature, feature, matchQuality]
 - matchQuality = [distance, point3D]
 - feature = [camera_id, proj_info, keypoint_id]
 - proj_info = [proj_u, proj_v]
- epochMatches = [match,...] # length = matches

- match= [feature, feature]
 - feature = [camera_id, proj_info, keypoint_id]
 - proj_info = [proj_u,proj_v]
- epochMatchesQuality = [distance,...] # index = by epochMatches
 - record the Euclidean distance between the matched two descriptors.
 - distance = float
- epochTracksMatches_cube = [epochTracksMatches,...]
 - the descriptors of common feature track.
 - epochTracksMatches = [view_id,...]
 - view_id = (camera_id, keypoints_id)
- epochTracksMatches_cube_Quality = [epochMatchesQuality,...] # index = by epochTracksMatches
 - the matches and similarity measures of common feature track.
 - epochMatchesQuality= [distance ,...]
 - distance = float
- Tracks_ViewId = {view_id,...}
 - record the Euclidean distance between the matched two descriptors.
 - view_id = (camera_id, keypoints_id)
- Camera_IdList = [camera_id,...]
 - The visible camera list for a certain cube.
 - camera_id = int
- Camera_IdList_cube = {grid_id:Camera_IdList}
 - The visible camera list for each cube.
 - grid_id = int
 - Camera_IdList = [camera_id,...]