ASAP-Stereo

Release 0.3.1

Andrew M. Annex

Aug 22, 2023

CONTENTS:

CHAPTER

ONE

INTRODUCTION

ASAP-Stereo (Ames Stereo Automated Pipeline) is a Python API that orchestrates the NASA Ames Stereo Pipeline (ASP) for HiRISE and CTX stereo image processing. ASAP implements functions that use various command-line tools from GDAL, ASP, and USGS ISIS to produce Digital Elevation Models (DEMs) following a workflow of steps described by Mayer and Kite 2016 in their tool asp_scripts. The workflows generally follow a procedure of first downloading and calibrating the data, producing stereo products using ASP, and then finally aligning the stereo products with reference elevation data and the geoid. The commonality in workflow procedures and the number of steps involved mean that users will significantly benefit from the workflow's automation. ASAP also provides both a command-line interface for the whole API and Jupyter Notebooks, accessible through the command-line interface, that executes the workflows for CTX and HiRISE (described below). By providing all three interfaces, users can execute workflows, and then re-run individual steps to improve stereo product quality and run the ASAP workflows in a semi-automatic fashion.

1.1 ASAP Core Functionality

ASAP uses the sh python library to access external command-line tools as if they were Python functions without any direct subprocess management code in ASAP. Although using Python to call other subprocesses is not new, the sh project makes it remarkably easy to do this with minimal code. Using sh enables ASAP to reuse existing command-line tools and to use Python for tasks it is best suited.

1.2 ASAP's Commandline Interface (CLI)

ASAP's CLI is automatically created from the Python classes and functions using the python-fire project, requiring no code changes to the Python API. Python-fire allows ASAP to simultaneously provide both the API and CLI with no additional code to maintain. For example, using python-fire, it is possible to enable tab-completion for ASAP's CLI or drop into interactive python sessions. The python-fire GitHub page is an excellent resource to learn more about the project.

1.3 ASAP Jupyter Notebook Workflows

ASAP provides Jupyter Notebook template workflows to users, and they are the easiest way to use ASAP. ASAP uses the Papermill project to execute the notebook workflows, that project can turn any Jupyter Notebook into a parameterized command-line tool. Papermill uses the workflows provided by ASAP as templates that are then updated to use the user's specified images and parameters. When executed, the notebooks capture all the logs from the various steps of ASAP and the underlying tools into the notebook file. In addition to the logs, the notebooks also include images generated during the workflow's steps for quick quality inspection by the user. Users can share notebooks, allowing for reproducibility and traceability in their scientific data. More details about the notebook workflows will be provided in the tutorials section of the documentation.

CHAPTER

TWO

INSTALLATION

This guide assumes that the user has already installed anaconda with conda version newer than 4.9.

ASAP requires that both Ames Stereo Pipeline and ISIS are available to the user in the PATH, and the user will need to have downloaded the relevant ISIS data folders.

To start it is necessary to:

1. Create a dedicated ISIS conda environment following the instructions at https://github.com/DOI-USGS/ISIS3
2. Download and extract the pre-compiled Ames Stereo Pipeline binary following the first step in Ames Stereo Pipeline Installation

Now we will make a 2nd conda environment dedicated for ASAP using the following conda yml file environment specification (for background on this subject please read the related conda documentation).

In the variables section at the bottom, you will need to update the parameters for your environment. The ISISROOT directory is the conda environment folder for your working ISIS installation, for example "~/anaconda/envs/isis/", it can also be found by activating the environment and printing the $CONDA_PREFIX variable. The two ISIS data directories are explained in the ISIS installation documentation. The ASPROOT directory similarly points to the unzipped archive of the Ames Stereo Pipeline.

```
name: asap_0_3_0
channels:
  - conda-forge
  - defaults
dependencies:
  - python>=3.11
  - ale
  - black=19.10b0
  - pvl
  - nb_conda_kernels
  - jupyterlab
  - requests
  - fire
  - tqdm
  - papermill
  - rasterio
  - pyproj
  - shapely
  - sh
  - pip
  - pip:
    - moody>=0.2.0
    - asap_stereo=0.3.1
variables:
```

(continues on next page)

(continued from previous page)

```
  ISISROOT: /path/to/your/isis/conda/env/
  ISISDATA: /path/to/your/isis/data/dir/
  ISISTESTDATA: /path/to/optional/isis/test/data/dir/
  ASPROOT: /path/to/your/precompiled/StereoToolkitDirectory/
```

Once the above yml file has been updated, create the environment using the command:

```
conda env create --file asap.yml
```

This will create a new conda environment called "asap_0_3_0". The environment however is not ready to use yet. We need to update the PATH for this conda environment to ensure ASP and ISIS programs are available for ASAP to use. To do this run the following commands:

```
conda activate asap_0_3_0
conda env config vars set PATH=$PATH:$ASPROOT/bin:$ISISROOT/bin
conda deactivate
```

This will update the PATH variable in the conda environment to include both ISIS and ASP bin folders. At this point you are ready to start using asap!

To test if asap is installed correctly, run each of the following after activating the new environment:

```
asap
asap ctx
asap hirise
asap common
```

Each of these commands should show the help page for the asap modules and shouldn't print any errors.

CHAPTER

THREE

ASAP-STEREO

3.1 ASAP

class asap_stereo.asap.**ASAP** (*https=False*, *datum='D_MARS'*)

Bases: object

ASAP Stereo Pipeline

asap_stereo (0.3.1)

Github: https://github.com/AndrewAnnex/asap_stereo Cite: https://doi.org/10.5281/zenodo.4171570

ctx_one (*left*, *right*, *cwd=None*)

Run first stage of CTX pipeline

This command runs steps 1-3 of the CTX pipeline

Parameters

- **left** – left image id
- **right** – right image id
- **cwd** (Optional[str]) – directory to run process within (default to CWD)

ctx_three (*max_disp=None*, *demgsd=24*, *imggsd=6*, *cwd=None*, ***kwargs*)

Run third and final stage of the CTX pipeline

This command runs steps 13-15 of the CTX pipeline

Parameters

- **max_disp** (Optional[float]) – Maximum expected displacement in meters
- **demgsd** (float) – GSD of final Dem, default is 1 mpp

- **imggsd** (float) – GSD of full res image
- **cwd** (Optional[str]) – directory to run process within (default to CWD)
- **kwargs** –

Return type
None

ctx_two (*stereo*, *pedr_list*, *stereo2=None*, *cwd=None*)

Run Second stage of CTX pipeline

This command runs steps 4-12 of the CTX pipeline

Parameters

- **stereo** (str) – ASP stereo config file to use
- **pedr_list** (str) – Path to PEDR files, defaults to None to use ODE Rest API
- **stereo2** (Optional[str]) – 2nd ASP stereo config file to use, if none use first stereo file again
- **cwd** (Optional[str]) – directory to run process within (default to CWD)

Return type
None

hirise_one (*left*, *right*)

Download the EDR data from the PDS, requires two HiRISE Id's (order left vs right does not matter)

This command runs step 1 of the HiRISE pipeline

Parameters

- **left** – HiRISE Id
- **right** – HiRISE Id

hirise_three (*max_disp*, *ref_dem*, *demgsd=1*, *imggsd=0.25*, ***kwargs*)

Given estimate of max disparity between reference elevation model and HiRISE output, run point cloud alignment and produce the final DEM/ORTHO data products.

This command runs steps 10-12 of the HiRISE pipeline

Parameters

- **max_disp** – Maximum expected displacement in meters
- **ref_dem** – Absolute path the reference dem
- **demgsd** (float) – GSD of final Dem, default is 1 mpp
- **imggsd** (float) – GSD of full res image

Return type
None

hirise_two (*stereo*, *mpp=2*, *bundle_adjust_prefix='adjust/ba'*, *max_iterations=50*)

Run various calibration steps then: bundle adjust, produce DEM, render low res version for inspection This will take a while (sometimes over a day), use nohup!

This command runs steps 2-9 of the HiRISE pipeline

Parameters

- **stereo** – ASP stereo config file to use

- **mpp** – preview DEM GSD, defaults to 2 mpp
- **bundle_adjust_prefix** – bundle adjust prefix, defaults to 'adjust/ba'
- **max_iterations** – number of iterations for HiRISE bundle adjustment, defaults to 50

Return type
None

info()

Get the number of threads and processes as a formatted string

Returns
str rep of info

class asap_stereo.stereo_quality.**ImgInfo**(*path*)

Bases: object

emission_quality()

Return type
float

incidence_quality()

Return type
float

phase_quality()

Return type
float

qualities()

Return type
tuple

asap_stereo.stereo_quality.**area_overlap**(*geom1*, *geom2*)

Returns a ratio of areas which represent the area of the intersection area of *geom1* and *geom2*, divided by the union of areas of of *geom1*, and *geom2*.

Return type
float

asap_stereo.stereo_quality.**delta_solar_az_quality**(*az1*, *az2*)

Returns a quality score based on the two solar azimuth values, in degrees.

In practice, Shadow-Tip Distance alone does not guarantee similar illumination. The absolute difference in solar azimuth angle between stereo pairs can be optionally constrained.

Becker et al. (2015) indicates: - Limits: 0° to 100°. - Recommended: ≤ 20°

Return type
float

asap_stereo.stereo_quality.**dp**(*emission1*, *gndaz1*, *emission2*, *gndaz2*, *radar=False*)

Returns the Parallax/Height Ratio (dp) as detailed in Becker et al.(2015).

The input angles are assumed to be in radians. If *radar* is true, then cot() is substituted for tan() in the calculations.

Physically, dp represents the amount of parallax difference that would be measured between an object in the two images, for unit height.

asap_stereo.stereo_quality.**dsh**(*incidence1*, *solar_az1*, *incidence2*, *solar_az2*)

Returns the Shadow-Tip Distance (dsh) as detailed in Becker et al.(2015).

The input angles are assumed to be in radians.

This is defined as the distance between the tips of the shadows in the two images for a hypothetical vertical post of unit height. The "shadow length" describes the shadow of a hypothetical pole so it applies whether there are actually shadows in the image or not. It's a simple and consistent geometrical way to quantify the difference in illumination. This quantity is computed analogously to dp.

asap_stereo.stereo_quality.**emission_quality**(*emission_angle*)

Returns a quality score based on the value of *emission_angle*, which is expected to be in decimal degrees, zero being normal to the surface.

Return type
float

Becker et al. (2015) indicates: - Limits: Between 0° and the complement of the maximum slope

(conservatively 45°, greater for smoother terrains) for optical images. Greater than the slope (≥15° even for smooth surfaces) for radar.

- Recommended: No recommendation

asap_stereo.stereo_quality.**get_report**(*file1*, *file2*)

Generate the report of the stereo quality :type file1: :param file1: :type file2: :param file2: :rtype: str :return:

asap_stereo.stereo_quality.**gsd_quality**(*gsd1*, *gsd2*)

Returns a quality score based on the two ground sample distances, *gsd1* and *gsd2*.

Image pairs with GSD ratios larger than 2.5 can be used but are not optimal, as details only seen in the smaller scale image will be lost. If required, images with ratios greater than ~2.5 should be resampled to the GSD of the lower scale image (Becker at al., 2015).

Return type
float

asap_stereo.stereo_quality.**illumination_quality**(*shadow_tip_distance*)

Returns a quality score based on the Shadow-Tip Distance (*dsh*).

Becker et al. (2015) indicates: - Limits: 0 to 2.58. - Recommended: 0

Return type
float

asap_stereo.stereo_quality.**incidence_quality**(*incidence_angle*)

Returns a quality score based on the value of *incidence_angle*, which is expected to be in decimal degrees, zero being normal to the surface.

Return type
float

Becker et al. (2015) indicates: - Limits: Between 40° and 65° depending on smoothness

(shadows to be avoided).

- Recommended: Nominally 50°

asap_stereo.stereo_quality.**parallax**(*emission1*, *gndaz1*, *emission2*, *gndaz2*)

Returns the parallax angle between the two look vectors described by the emission angles and sub-spacecraft ground azimuth angles.

Input angles are assumed to be radians, as is the return value.

Return type
float

asap_stereo.stereo_quality.**phase_quality**(*phase_angle*)

Returns a quality score based on the value of *phase_angle*, which is expected to be in decimal degrees, zero being normal to the surface.

Becker et al. (2015) indicates: - Limits: Between 5° and 120°. - Recommended: ≥ 30°

Return type
float

asap_stereo.stereo_quality.**quality**(*value*, *ideal*, *low*, *high*)

Return a quality value based on *value*.

The value of *ideal* would be a perfect value of *value*, but if *value* is between *low* and *high* it is acceptable. *ideal* must be between *low* and *high*, otherwise a ValueError is raised.

A quality value of one indicates that *value* is *ideal*. Quality values between zero and one indicate that *value* is between *low* and *high* (the closer to *ideal*, the higher the quality score). Values less than zero are beyond the acceptable range of *low* and *high*.

If *ideal* is a two-tuple, then this indicates that all values between and including those values are "ideal" values. Again, these two values must be between *low* and *high*

Return type
float

asap_stereo.stereo_quality.**stereo_overlap_quality**(*area_fraction*)

Returns a quality score based on the stereo area overlap.

Becker et al. (2015) indicates: - Limits: Between 30% and 100%. - Recommended: 50% to 100%.

Return type
float

asap_stereo.stereo_quality.**stereo_strength_quality**(*parallax_height_ratio*)

Returns a quality score based on the Parallax/Height Ratio (*dp*).

Becker et al. (2015) indicates: - Limits: Between 0.1 (5°) and 1 (~45°). - Recommended: 0.4 (20°) to 0.6 (30°).

Return type
float

3.2 ASAP (module level)

asap.**cmd_to_string**()

Converts the running command into a single string of the full command call for easier logging

Parameters
command (RunningCommand) – a command from sh.py that was run

Return type
str

Returns
string of bash command

`@asap_stereo.asap.`**`rich_logger`**(*func*)

rich logger decorator, wraps a function and writes nice log statements

Parameters
`func` (`Callable`) – function to wrap

Returns
wrapped function

`asap.`**`par_do`**(*all_calls_args*)

Parallel execution helper function for sh.py Commands

Parameters

- **`func`** – func to call
- **`all_calls_args`** – args to pass to each call of the func

Returns
list of called commands

3.3 CommonSteps

`class` `asap_stereo.asap.`**`CommonSteps`**

Bases: `object`

ASAP Stereo Pipeline - Common Commands

___ _____ ___ ____

/ | / ___// | / __

/ /| | __ / /| | / /_/ /

/ ___ |___/ / ___ |/ ____/

/_/ |_/____/_/ |_/_/ *S T E R E O*

asap_stereo (0.3.1)

Github: https://github.com/AndrewAnnex/asap_stereo Cite: https://doi.org/10.5281/zenodo.4171570

`bundle_adjust`(**vargs*, *postfix='_RED.cub'*, *bundle_adjust_prefix='adjust/ba'*, *camera_postfix='.json'*, ***kwargs*)

Bundle adjustment wrapper

#TODO: make function that attempts to find absolute paths to vargs if they are files?

Parameters

- **`vargs`** – any number of additional positional arguments (including GCPs)
- **`postfix`** – postfix of images to bundle adjust
- **`camera_postfix`** – postfix for cameras to use

- **bundle_adjust_prefix** – where to save out bundle adjust results
- **kwargs** – kwargs to pass to bundle_adjust

Return type
RunningCommand

Returns
RunningCommand

static cam_test(*cub*, *camera*, *sample_rate=1000*, *subpixel_offset=0.25*)

Return type
str

check_mpp_against_true_gsd(*path*, *mpp*)

Get the GSD of the image, and warn if it is less than 3 * the gsd

Parameters

- **path** – path to image
- **mpp** – proposed mpp for image

compute_footprints(**imgs*)

for each footprint generate a vector footprint :type imgs: :param imgs: gdal rasters with nodata defined :return:

static create_stereodirs()

static create_stereodirs_lis()

static create_stereopair_lis()

static create_stereopairs_lis()

crop_by_buffer(*ref*, *src*, *img_out=None*, *factor=2.0*)

use gdal warp to crop img2 by a buffer around img1

Parameters

- **ref** – first image defines buffer area
- **src** – second image is cropped by buffer from first
- **factor** – factor to buffer with

```
defaults_ps1 = {'--bundle-adjust-prefix': 'adjust/ba', '--processes': 1,
'--stop-point': 5, '--threads-multiprocess': 1,
'--threads-singleprocess': 2}

defaults_ps2 = {'--bundle-adjust-prefix': 'adjust/ba', '--entry-point':
5, '--processes': 2, '--threads-multiprocess': 1,
'--threads-singleprocess': 2}

defaults_ps_s0 = {'--bundle-adjust-prefix': 'adjust/ba', '--entry-point':
0, '--processes': 1, '--stop-point': 1, '--threads-multiprocess': 1,
'--threads-singleprocess': 2}

defaults_ps_s1 = {'--bundle-adjust-prefix': 'adjust/ba', '--entry-point':
1, '--processes': 1, '--stop-point': 2, '--threads-multiprocess': 1,
'--threads-singleprocess': 2}
```

```
defaults_ps_s2 = {'--bundle-adjust-prefix': 'adjust/ba', '--entry-point':
2, '--processes': 1, '--stop-point': 3, '--threads-multiprocess': 1,
'--threads-singleprocess': 2}

defaults_ps_s3 = {'--bundle-adjust-prefix': 'adjust/ba', '--entry-point':
3, '--processes': 1, '--stop-point': 4, '--threads-multiprocess': 1,
'--threads-singleprocess': 2}

defaults_ps_s4 = {'--bundle-adjust-prefix': 'adjust/ba', '--entry-point':
4, '--processes': 1, '--stop-point': 5, '--threads-multiprocess': 1,
'--threads-singleprocess': 2}

defaults_ps_s5 = {'--bundle-adjust-prefix': 'adjust/ba', '--entry-point':
5, '--processes': 2, '--threads-multiprocess': 1,
'--threads-singleprocess': 2}
```

static drg_to_cog(*img*, *scale_bound=0.001*, *gdal_options=None*)

estimate_max_disparity(*ref_dem*, *src_dem=None*)

Estimate the absolute value of the maximum observed displacement between two point clouds, and the standard deviation of the differences

if not applying an initial transform to pc_align, use the max_d value if expecting to apply a transform first and you are interested in the maximum displacement after an initial transform, then use the std_d returned (likely 3X it)

estimate_median_disparity(*ref_dem*, *src_dem=None*)

static gen_csm(**cubs*, *meta_kernal=None*, *max_workers=2*)

Given N cub files, generate json camera models for each using ale

generate_csm(*postfix='_RED.cub'*, *camera_postfix='_RED.json'*)

generate CSM models for both images :type postfix: :param postfix: :type camera_postfix: :param camera_postfix: :return:

geoid_adjust(*run*, *output_folder*, ***kwargs*)

Adjust DEM to geoid

Run geoid adjustment on dem for final science ready product :type run: :param run: :type output_folder: :param output_folder: :type kwargs: :param kwargs:

static get_cam_info(*img*)

Get the camera information dictionary from ISIS using camrange

Parameters
img – path to image as a string

Return type
Dict

Returns
dictionary of info

get_geo_diff(*ref_dem*, *src_dem=None*)

static get_image_band_stats(*img*)

Parameters
img –

Return type
dict

Returns

static get_image_gsd(*img*, *opinion='lower'*)

Return type
float

static get_img_bounds(*img*)

Get the bounds of the image

uses rasterio :type img: :param img: path to image :return: bounds tuple

static get_img_crs(*img*)

Get CRS of the image

uses rasterio :type img: :param img: path to image :return: CRX of image

static get_map_info(*img*, *key*, *group='UniversalGroundRange'*)

Return type
str

static get_mpp_postfix(*mpp*)

get the mpp postfix

Parameters
mpp (Union[int, float, str]) – mpp value

Return type
str

Returns
postfix as a string

get_pedr_4_pcalign_common(*postfix*, *proj*, *https*, *pedr_list=None*)

Return type
str

get_pedr_4_pcalign_w_moody(*cub_path*, *proj=None*, *https=True*)

Python replacement for pedr_bin4pc_align.sh that uses moody and the PDS geosciences node REST API

Parameters

- **proj** – optional projection override
- **https** – optional way to disable use of https
- **cub_path** – path to input file to get query geometry

Return type
str

static get_srs_info(*img*, *use_eqc=None*)

Return type
str

static get_stereo_quality_report(*cub1*, *cub2*)

Get the stereo quality report for two cub files The cub files must be Level1 images (Spiceinit'ed but not map-projected).

The quality values reported by this program are based on the recommendations and limitations in Becker et al. (2015). They have a value of one for an ideal value, between zero and one for a value within the acceptable limits, and less than zero (the more negative, the worse) if the value is beyond the acceptable limit. # TODO refactor into more granular bits :type cub1: :param cub1: path :type cub2: :param cub2: path :rtype: `str` :return:

mapproject_both(*refdem=None*, *mpp=6*, *postfix='.lev1eo.cub'*, *camera_postfix='.lev1eo.json'*, *bundle_adjust_prefix='adjust/ba'*, ***kwargs*)

Mapproject the left and right images against a reference DEM

Parameters

- **refdem** – reference dem to map project using
- **mpp** – target GSD
- **postfix** – postfix for cub files to use
- **camera_postfix** – postfix for cameras to use
- **bundle_adjust_prefix** – where to save out bundle adjust results

static parse_stereopairs()

point_cloud_align(*datum*, *maxd=None*, *refdem=None*, *highest_accuracy=True*, *run='results_ba'*, *kind='map_ba_align'*, ***kwargs*)

point_to_dem(*mpp*, *pc_suffix*, *just_ortho=False*, *just_dem=False*, *use_proj=None*, *postfix='.lev1eo.cub'*, *run='results_ba'*, *kind='map_ba_align'*, *output_folder='dem'*, *reference_spheroid='mars'*, ***kwargs*)

projections = {'IAU_Mars': '+proj=eqc +lat_ts=0 +lat_0=0 +lon_0=0 +x_0=0 +y_0=0 +a=3396190 +b=3396190 +units=m +no_defs', 'IAU_Mercury': '+proj=eqc +lat_ts=0 +lat_0=0 +lon_0=0 +x_0=0 +y_0=0 +a=2439700 +b=2439700 +units=m +no_defs', 'IAU_Moon': '+proj=eqc +lat_ts=0 +lat_0=0 +lon_0=0 +x_0=0 +y_0=0 +a=1737400 +b=1737400 +units=m +no_defs'}

rescale_and_overwrite(*factor*, *postfix='.lev1eo.cub'*)

Rescale the left and right images

Parameters

- **factor** – factor to reduce each dimension by
- **postfix** – file postfix

rescale_cub(*src_file*, *factor=4*, *overwrite=False*, *dst_file=None*)

rescale an ISIS3 cub file using the 'reduce' command given a factor, optionaly do not overwrite file

Parameters

- **src_file** (`str`) – path to src cub file
- **factor** – reduction factor (number [lines, samples] / factor)
- **overwrite** – if true overwrite the src file
- **dst_file** – destination file name, append *rescaled_* if not specified

stereo_asap(*stereo_conf*, *refdem=''*, *postfix='.lev1eo.cub'*, *camera_postfix='.json'*, *run='results_ba'*, *output_file_prefix='${run}/${both}_ba'*, *posargs=''*, ***kwargs*)

parallel stereo common step

Parameters

- **run** – stereo run output folder prefix
- **output_file_prefix** – template string for output file prefix
- **refdem** (str) – optional reference DEM for 2nd pass stereo
- **posargs** (str) – additional positional args
- **postfix** – postfix(s) to use for input images
- **camera_postfix** – postfix for cameras to use
- **stereo_conf** (str) – stereo config file
- **kwargs** – keyword arguments for parallel_stereo

transform_bounds_and_buffer(*img1*, *img2*, *factor=2.0*)

Get bounds of img2 based on centroid of img1 surrounded by a buffer the size of the maximum dimension of img1 (scaled by a factor)

ie if img1 is hirise and img2 is ctx, we find the center point of img1 in img2 then create a bounding box that is buffered (in radius) by the height of the hirise image technically the buffered box would be 2x the height of the hirise which is fine

Parameters

- **img1** – img to find the bounds in img2 space
- **img2** – crs we are interested in finding the expanded bounds of img1 in
- **factor** – how big we want it (radius is longest dim in img1)

Returns

xmin_img2, ymin_img2, xmax_img2, ymax_img2

3.4 CTX

class asap_stereo.asap.**CTX**(*https=False*, *datum='D_MARS'*, *proj=None*)

Bases: object

ASAP Stereo Pipeline - CTX workflow

asap_stereo (0.3.1)

Github: https://github.com/AndrewAnnex/asap_stereo Cite: https://doi.org/10.5281/zenodo.4171570

generate_ctx_pair_list(*one*, *two*)

get_ctx_emission_angle(*pid*)

get_ctx_order(*one*, *two*)

get_first_pass_refdem(*run='results_ba'*)

Return type
str

get_full_ctx_id(*pid*)

static notebook_pipeline_make_dem(*left*, *right*, *config1*, *pedr_list=None*, *downsample=None*, *working_dir='./'*, *config2=None*, *dem_gsd=24.0*, *img_gsd=6.0*, *max_disp=None*, *step_kwargs=None*, *out_notebook=None*, ***kwargs*)

First step in CTX DEM pipeline that uses papermill to persist log

this command does most of the work, so it is long running! I recommend strongly to use nohup with this command

Parameters

- **out_notebook** – output notebook log file name, defaults to log_asap_notebook_pipeline_make_dem.ipynb
- **config2** (Optional[str]) – ASP config file to use for second processing pass
- **working_dir** – Where to execute the processing, defaults to current directory
- **config1** (str) – ASP config file to use for first processing pass
- **pedr_list** (Optional[str]) – Path to PEDR files, defaults to None to use ODE Rest API
- **left** (str) – First image id
- **right** (str) – Second image id
- **max_disp** – Maximum expected displacement in meters, use None to determine it automatically
- **step_kwargs** – Arbitrary dict of kwargs for steps following {'step_#' : {'key': 'value}}
- **downsample** (Optional[int]) – Factor to downsample images for faster production
- **dem_gsd** – desired GSD of output DEMs (4x image GSD)
- **img_gsd** – desired GSD of output ortho images
- **kwargs** – kwargs for papermill

step_1(*one*, *two*, *cwd=None*)

Download CTX EDRs from the PDS

Parameters

- **one** (str) – first CTX image id
- **two** (str) – second CTX image id
- **cwd** (Optional[str]) –

Return type
None

step_10 (*stereo_conf*, *refdem=None*, *posargs=''*, *postfix='.ba.map.tif'*, *camera_postfix='.lev1eo.json'*, ***kwargs*)

Second stereo first step

Parameters

- **stereo_conf** –
- **refdem** – path to reference DEM or PEDR csv file
- **posargs** – additional positional args
- **postfix** – postfix for files to use
- **camera_postfix** – postfix for cameras to use
- **kwargs** –

step_11 (*stereo_conf*, *refdem=None*, *posargs=''*, *postfix='.ba.map.tif'*, *camera_postfix='.lev1eo.json'*, ***kwargs*)

Second stereo second step

Parameters

- **stereo_conf** –
- **refdem** – path to reference DEM or PEDR csv file
- **posargs** – additional positional args
- **postfix** – postfix for files to use
- **camera_postfix** – postfix for cameras to use
- **kwargs** –

step_12 (*pedr_list=None*, *postfix='.lev1eo'*)

Get MOLA PEDR data to align the CTX DEM to

Parameters

- **postfix** – postfix for file, minus extension
- **pedr_list** – path local PEDR file list, default None to use REST API

step_13 (*run='results_map_ba'*, *maxd=None*, *refdem=None*, *highest_accuracy=True*, ***kwargs*)

PC Align CTX

Run pc_align using provided max disparity and reference dem optionally accept an initial transform via kwargs

Parameters

- **run** – folder used for this processing run
- **highest_accuracy** – Use the maximum accuracy mode
- **maxd** (`Optional[float]`) – Maximum expected displacement in meters
- **refdem** – path to pedr csv file or reference DEM/PC, if not provided assume pedr4align.csv is available
- **kwargs** –

step_14 (*mpp=24.0*, *just_ortho=False*, *run='results_map_ba'*, *output_folder='dem_align'*, *postfix='.lev1eo.cub'*, ***kwargs*)

Produce final DEMs/Orthos

Run point2dem on the aligned output to produce final science ready products

Parameters

- **run** – folder used for this processing run
- **mpp** –
- **just_ortho** –
- **output_folder** –
- **postfix** – postfix for cub files to use
- **kwargs** –

step_15 (*run='results_map_ba'*, *output_folder='dem_align'*, ***kwargs*)

Adjust DEM to geoid

Run geoid adjustment on dem for final science ready product :type run: :param run: folder used for this processing run :type output_folder: :param output_folder: :type kwargs: :param kwargs:

step_2 (*with_web=False*)

ISIS3 CTX preprocessing, replaces ctxedr2lev1eo.sh

Parameters
with_web – if true attempt to use webservices for SPICE kernel data

step_3 ()

Create various processing files for future steps # todo: deduplicate with hirise side

step_4 (**vargs*, *bundle_adjust_prefix='adjust/ba'*, *postfix='.lev1eo.cub'*, *camera_postfix='.lev1eo.json'*, ***kwargs*)

Bundle Adjust CTX

Run bundle adjustment on the CTX map projected data

Parameters

- **vargs** – variable length additional positional arguments to pass to bundle adjust
- **bundle_adjust_prefix** – prefix for bundle adjust output
- **postfix** – postfix for cub files to use
- **camera_postfix** – postfix for cameras

Return type
`RunningCommand`

step_5 (*stereo_conf*, *posargs=''*, *postfix='.lev1eo.cub'*, *camera_postfix='.lev1eo.json'*, ***kwargs*)

Parallel Stereo Part 1

Run first part of parallel_stereo asp_ctx_lev1eo2dem.sh

Parameters

- **postfix** – postfix for cub files to use
- **camera_postfix** – postfix for cameras # TODO: use .adjusted_state.json?

step_6(*stereo_conf*, *posargs=''*, *postfix='.lev1eo.cub'*, *camera_postfix='.lev1eo.json'*, ***kwargs*)

Parallel Stereo Part 2

Run second part of parallel_stereo, asp_ctx_lev1eo2dem.sh stereo is completed after this step

Parameters

- **postfix** – postfix for cub files to use
- **camera_postfix** – postfix for cameras # TODO: use .adjusted_state.json?

step_7(*mpp=24*, *just_ortho=False*, *run='results_ba'*, *postfix='.lev1eo.cub'*, ***kwargs*)

Produce preview DEMs/Orthos

Produce dem from point cloud, by default 24mpp for ctx for max-disparity estimation

Parameters

- **run** – folder for results
- **just_ortho** – set to True if you only want the ortho image, else make dem and error image
- **mpp** – resolution in meters per pixel
- **postfix** – postfix for cub files to use

step_8(*run='results_ba'*, *output_folder='dem'*)

hillshade First step in asp_ctx_step2_map2dem script

Parameters

- **output_folder** –
- **run** –
- **mpp** –

step_9(*refdem=None*, *mpp=6*, *run='results_ba'*, *postfix='.lev1eo.cub'*, *camera_postfix='.lev1eo.json'*)

Mapproject the left and right ctx images against the reference DEM

Parameters

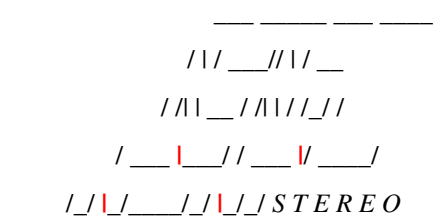
- **run** – name of run
- **refdem** – reference dem to map project using
- **mpp** – target GSD
- **postfix** – postfix for cub files to use
- **camera_postfix** – postfix for cameras to use

3.5 HiRISE

class asap_stereo.asap.**HiRISE**(*https=False*, *datum='D_MARS'*, *proj=None*)

Bases: object

ASAP Stereo Pipeline - HiRISE workflow



asap_stereo (0.3.1)

Github: https://github.com/AndrewAnnex/asap_stereo Cite: https://doi.org/10.5281/zenodo.4171570

generate_hirise_pair_list (*one*, *two*)

Generate the hirise pair.lis file for future steps

Parameters

- **one** – first image id
- **two** – second image id

get_hirise_emission_angle (*pid*)

Use moody to get the emission angle of the provided HiRISE image id

Parameters
pid (str) – HiRISE image id

Return type
float

Returns
emission angle

get_hirise_order (*one*, *two*)

Get the image ids sorted by lower emission angle

Parameters

- **one** (str) – first image
- **two** (str) – second image

Return type
Tuple[str, str]

Returns
tuple of sorted images

static notebook_pipeline_make_dem (*left*, *right*, *config*, *ref_dem*, *gcps=''*, *max_disp=None*, *downsample=None*, *dem_gsd=1.0*, *img_gsd=0.25*, *max_ba_iterations=200*, *alignment_method='rigid'*, *step_kwargs=None*, *working_dir='./'*, *out_notebook=None*, ***kwargs*)

First step in HiRISE DEM pipeline that uses papermill to persist log

This command does most of the work, so it is long running! I recommend strongly to use nohup with this command, even more so for HiRISE!

Parameters

- **out_notebook** – output notebook log file name, defaults to log_asap_notebook_pipeline_make_dem_hirise.ipynb
- **working_dir** – Where to execute the processing, defaults to current directory
- **config** (str) – ASP config file to use for processing
- **left** (str) – first image id
- **right** (str) – second image id
- **alignment_method** – alignment method to use for pc_align
- **downsample** (Optional[int]) – Factor to downsample images for faster production
- **ref_dem** (str) – path to reference DEM or PEDR csv file
- **gcps** (str) – path to gcp file todo: currently only one gcp file allowed
- **max_disp** (Optional[float]) – Maximum expected displacement in meters, specify none to determine it automatically
- **dem_gsd** (float) – desired GSD of output DEMs (4x image GSD)
- **img_gsd** (float) – desired GSD of output ortho images
- **max_ba_iterations** (int) – maximum number of BA steps to use per run (defaults to 50 for slow running hirise BA)
- **step_kwargs** – Arbitrary dict of kwargs for steps following {'step_#' : {'key': 'value}}

pre_step_10 (*refdem*, *run='results_ba'*, *alignment_method='translation'*, *do_resample='gdal'*, ***kwargs*)

Hillshade Align before PC Align

Automates the procedure to use ipmatch on hillshades of downsampled HiRISE DEM to find an initial transform

Parameters

- **run** –
- **do_resample** – can be: 'gdal' or 'asp' or anything else for no resampling
- **alignment_method** – can be 'similarity' 'rigid' or 'translation'
- **refdem** – path to reference DEM or PEDR csv file
- **kwargs** –

pre_step_10_pedr (*pedr_list=None*, *postfix='_RED.cub'*)

Use MOLA PEDR data to align the HiRISE DEM to in case no CTX DEM is available

Parameters

- **pedr_list** – path local PEDR file list, default None to use REST API
- **postfix** – postfix for cub files to use

Return type
str

step_1 (*one*, *two*, *cwd=None*)

Download HiRISE EDRs

Download two HiRISE images worth of EDR files to two folders

Parameters

- **one** – first image id
- **two** – second image id
- **cwd** (Optional[str]) –

step_10 (*maxd*, *refdem*, *run='results_ba'*, *highest_accuracy=True*, ***kwargs*)

PC Align HiRISE

Run pc_align using provided max disparity and reference dem optionally accept an initial transform via kwargs

Parameters

- **run** –
- **maxd** – Maximum expected displacement in meters
- **refdem** – path to reference DEM or PEDR csv file
- **highest_accuracy** – use highest precision alignment (more memory and cpu intensive)
- **kwargs** – kwargs to pass to pc_align, use to override ASAP defaults

step_11 (*mpp=1.0*, *just_ortho=False*, *postfix='_RED.cub'*, *run='results_ba'*, *output_folder='dem_align'*, ***kwargs*)

Produce final DEMs/Orthos

Run point2dem on the aligned output to produce final science ready products

Parameters

- **run** –
- **postfix** – postfix for cub files to use
- **mpp** – Desired GSD (meters per pixel)
- **just_ortho** – if True, just render out the ortho images
- **output_folder** – output folder name
- **kwargs** – any other kwargs you want to pass to point2dem

step_12 (*run='results_ba'*, *output_folder='dem_align'*, ***kwargs*)

Adjust DEM to geoid

Run geoid adjustment on dem for final science ready product

Parameters

- **run** –
- **output_folder** –
- **kwargs** –

step_2 ()

Metadata init

Create various files with info for later steps

step_3 ()

Hiedr2mosaic preprocessing

Run hiedr2mosaic on all the data

step_4 (*postfix='*.mos_hijitreged.norm.cub'*, *camera_postfix='_RED.json'*)

Copy hieder2mosaic files

Copy the hiedr2mosaic output to the location needed for cam2map4stereo

Parameters
postfix – postfix for cub files to use

step_5 (*refdem=None*, *gsd=None*, *postfix='_RED.cub'*, *camera_postfix='_RED.json'*, *bundle_adjust_prefix=None*, ***kwargs*)

todo this no longer makes sense for step 5, needs to run after bundle adjust but before stereo # todo need cameras by this point, currently done in BA Map project HiRISE data for stereo processing

Note this step is optional.

Parameters

- **bundle_adjust_prefix** –
- **camera_postfix** –
- **postfix** – postfix for cub files to use
- **gsd** (`Optional[float]`) – override for final resolution in meters per pixel (mpp)

step_6 (**vargs*, *postfix='_RED.cub'*, *camera_postfix='_RED.json'*, *bundle_adjust_prefix='adjust/ba'*, ***kwargs*)

Bundle Adjust HiRISE

Run bundle adjustment on the HiRISE map projected data

Parameters

- **postfix** – postfix for cub files to use
- **camera_postfix** – postfix for cameras to use
- **vargs** – variable length additional positional arguments to pass to bundle adjust
- **bundle_adjust_prefix** –

Return type
`RunningCommand`

step_7 (*stereo_conf*, *postfix='_RED.cub'*, *camera_postfix='_RED.json'*, *run='results_ba'*, *posargs=''*, ***kwargs*)

Parallel Stereo Part 1

Run first part of parallel_stereo

Parameters

- **run** – folder for results of run
- **postfix** – postfix for cub files to use
- **camera_postfix** – postfix for cameras to use

step_8 (*stereo_conf*, *postfix='_RED.cub'*, *camera_postfix='_RED.json'*, *run='results_ba'*, *posargs=''*, ***kwargs*)

Parallel Stereo Part 2

Run second part of parallel_stereo, stereo is completed after this step

Parameters

- **run** – folder for results of run
- **postfix** – postfix for cub files to use
- **camera_postfix** – postfix for cameras to use

step_9(*mpp=2*, *just_dem=True*, *postfix='_RED.cub'*, *run='results_ba'*, ***kwargs*)

Produce preview DEMs/Orthos

Produce dem from point cloud, by default 2mpp for hirise for max-disparity estimation

Parameters

- **run** – folder for results of run
- **postfix** – postfix for cub files to use
- **just_dem** – set to True if you only want the DEM and no other products like the ortho and error images
- **mpp** –

3.6 Georef

class asap_stereo.asap.**Georef**

Bases: object

ASAP Stereo Pipeline - Georef Tools

asap_stereo (0.3.1)

Github: https://github.com/AndrewAnnex/asap_stereo Cite: https://doi.org/10.5281/zenodo.4171570

add_gcps(*gcp_csv_file*, *mobile_file*)

Given a gcp file in csv format (can have Z values or different extension) use gdaltranslate to add the GCPs to the provided mobile file by creating a VRT raster

create_gcps(*reference_image*, *match_file_csv*, *out_name=None*)

Given a reference image and a match file in csv format, generate a csv of GCPs. By default just prints to stdout but out_name allows you to name the csv file or you can pipe

find_matches(*reference_image*, **mobile_images*, *ipfindkwargs=None*, *ipmatchkwargs=None*)

Generate GCPs for a mobile image relative to a reference image and echo to std out #todo: do we always assume the mobile_dem has the same srs/crs and spatial resolution as the mobile image? #todo: implement my own normalization :type reference_image: :param reference_image: reference vis image :type mobile_images: :param mobile_images: image we want to move to align to reference image :type ipfindkwargs:

:param ipfindkwargs: override kwargs for ASP ipfind :type ipmatchkwargs: :param ipmatchkwargs: override kwarge for ASP ipmatch :return:

get_common_matches(*ref_left_match*, *ref_right_match*)

returns coordinates as column row (x, y). rasterio xy expects row column

get_ref_z(*common_ref_left_crs*, *ref_dem*)

im_feeling_lucky(*ref_img*, *mobile_image*, **other_mobile*, *ipfindkwargs=None*, *ipmatchkwargs=None*, *gdal_warp_args=None*)

Georeference an mobile dataset against a reference image. Do it all in one go, can take N mobile datasets but assumes the first is the mobile image. If unsure normalize your data ahead of time

make_ba_gcps(*ref_img*, *ref_dem*, *ref_left_match*, *ref_right_match*, *left_name*, *lr_left_name*, *right_name*, *lr_right_name*, *eoid='+proj=longlat +R=3396190 +no_defs'*, *out_name=None*)

make_gcps_for_ba(*ref_img*, *ref_dem*, *left*, *right*, *eoid='+proj=longlat +R=3396190 +no_defs'*, *out_name=None*, *ipfindkwargs=None*)

Given a reference image and dem, and two images for a stereopair,

automatically create GCPs for ASP's BA by finding ip match points common between the reference image and the left and right images for the new pair and sampling the z values from the reference DEM.

note that this will create several vrt files because we want to make normalized downsampled images to find a good number of matches and to save time between images of large resolution differences

match_gsds(*ref_image*, **images*)

matches_to_csv(*match_file*)

Convert an ASP .match file from ipmatch to CSV

normalize(*image*)

ref_in_crs(*common*, *ref_img*, *cr=True*)

transform_matches(*match_file_csv*, *mobile_img*, *mobile_other*, *outname=None*)

Given a csv match file of two images (reference and mobile), and a third image (likely a DEM) create a modified match csv file with the coordinates transformed for the 2nd (mobile) image This works using the CRS of the images and assumes that both mobile images are already co-registered This is particularly useful when the imagery is higher pixel resolution than a DEM, and permits generating duplicated gcps

static warp(*reference_image*, *mobile_vrt*, *out_name=None*, *gdal_warp_args=None*, *tr=1.0*)

Final step in workflow, given a reference image and a mobile vrt with attached GCPs use gdalwarp to create a modified non-virtual file that is aligned to the reference image

CHAPTER

FOUR

TUTORIAL

4.1 ASAP CTX Walkthrough

4.1.1 Overview

There are multiple ways to use ASAP to produce DEMs. We will start with making a CTX DEM. The primary intended way to use asap is to use the notebook-based workflow commands through the command-line interface. These notebook-based workflow commands take a few parameters and will run a dozen or so individual steps serially. They are long-running commands for a single node, so using the nohup command is recommended to execute the program in the background. As the notebook workflow runs, various log files will appear in the directory and an output Jupyter notebook file containing all the logs when the program finishes. If the user is curious, a full_log.log file can be "tailed" by the user.

4.1.2 just run: asap ctx notebook_pipeline_make_dem

We assume the user has already activated their conda environment and are in a new empty directory. To start we need to select two CTX image IDs that we want to make a DEM of. All the steps to find and select CTX image IDs are outside of the scope of this tutorial for now. But in any case we will use a stereo pair of CTX data over Sera Crater in Arabia Terra, Mars.

Our image Ids are B03_010644_1889_XN_08N001W and P02_001902_1889_XI_08N001W. Let's go ahead and take a look at them to see if there are any hazes or other issues.

They look great!

We will also need a stereo.default file, I will use one provided with the Ames Stereo Pipeline for a CTX image pair in their documentation: stereo.nomap. Selecting the correct parameters for a given pair is outside of the scope for this tutorial, but the Ames Stereo Docs are a good place to start to learn more about the various parameters.

Now that we have all of our components we are ready to run the ASAP notebook workflow:

```
nohup asap ctx notebook_pipeline_make_dem B03_010644_1889_XN_08N001W P02_001902_1889_
↪XI_08N001W ./stereo.nonmap &
```

Again, we use the *"nohup ... &"* to run our command in the background, on a 8 core machine this step can take half an hour or so. If you want to produce a DEM faster, you can choose lower resource parameters for ASP by editing the stereo.default file. Alternatively, you can also tell ASAP to make a lower-resolution DEM by specifying a downscale parameter, ie '–downscale 4' to reduce the images by a factor of 4 in resolution. This one command will run several steps, replicating the workflow of the asp_scripts project. A notebook file, which default to be named "log_asap_notebook_pipeline_make_dem.ipynb" will also be created which we will explore later below.

Fig. 1: B03_010644_1889_XN_08N001W

4.1.3 Inspecting outputs

The notebook workflow produces a number of visualization products, some can be viewed using various ASP tools, some are produced by ASAP for viewing directly in the output notebook logs.

The *Good Pixel Maps* help quickly visualize if ASP is able to make sense of the provided images.

Hillshaded products quickly convey the textural quality of the DEMs.

The final data products for CTX are located in the diretory: *ID1_ID2/results_map_ba/dem_align/* where ID1 & ID2 are our CTX image ids. The final science-ready DEM file ends with the extension *_DEM-adj.tif*, the "adj" indicates the file is adjusted to the geoid. The final orthorectified images end with *_DRG.tif*. By default, for CTX images, ASAP will produce 6-meter orthorectified images and 24 meter DEMs. CTX data generally has better-than 6 meter GSD (~5.5 Meters per pixel), but the GSD is rounded up to produce conservative results for safety. This behavior can be overridden by the users at runtime using the 'dem_gsd' and 'img_gsd' command-line arguments. For more information regarding the various output files from Ames Stereo, please see their docs.

Congratulations, you have just made a DEM with ASAP!

Fig. 2: P02_001902_1889_XI_08N001W

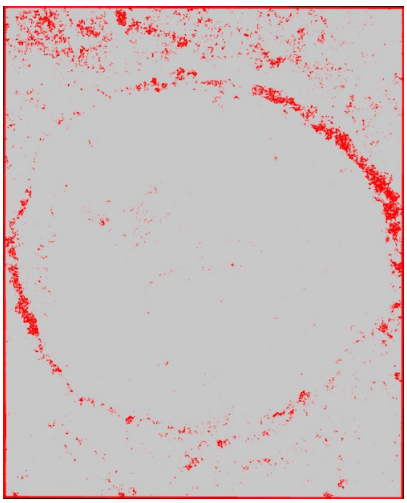


Fig. 3: Good Pixel Map from first pass DEM from step 1.

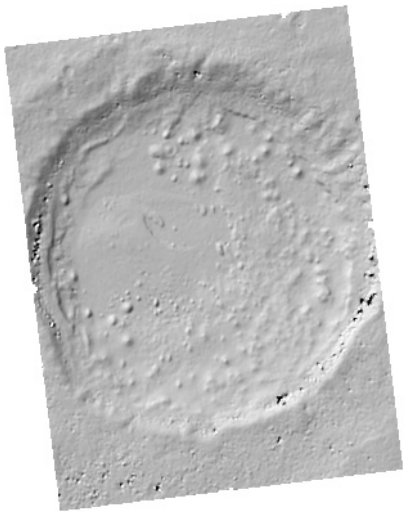


Fig. 4: Hillshaded low-resolution DEM from step 1.

CHAPTER

FIVE

NOTEBOOK WORKFLOW STEP-BY-STEP FOR CTX

Now that we have run the Jupyter Notebook based workflows through the command line interface, we can look at each step that was run and describe what happened in more detail. Note that the function docstrings are also available to describe the parameters of a given step, and what that step does. Below is an export of all the codeblocks in the notebook workflow, additional markdown cells are included in the files but are not important to reproduce here. This workflow replicates the same workflow used by the asp_scripts project.

5.1 Part 1: notebook_pipeline_make_dem

First define all the parameters for the notebook for papermill. The notebook includes a cell metadata tag for papermill to allow these parameters to be defined at runtime. First we need the left and right image ids, the left image typically has the lower emission angle. ASAP will check the metadata of the images to ensure the correct order is provided. The config1 and config2 parameters are paths to stereo.default files the user has to configure the Ames Stereo Pipeline. The first config file is the only required parameter, config2 gives you to use higher quality parameters for the 2nd pass CTX DEM. The "dem_gsd" and "img_gsd" parameters control the number of pixels per pixel the final DEM and orthorectified images have. These default to 24 and 6 meters per pixel which works for generally any CTX image pair. Generally, most CTX images are captured at around 5.5 meters per pixel (GSD) so we pick 6 mpp as a reasonable default. By convention, the DEM post spacing should be at least 3X the image GSD. ASAP defaults to 4X the image GSD to be a bit more conservative, resulting in 24 meters per pixel. Output_path is typically left blank to default to the current working directory. The maxdisp parameter controls the maximum expected disparity (distance) between the intermediate CTX DEM and the reference topography. Leaving this as 'None' will allow ASAP to estimate the disparity for you. The downsample parameter allows you to downsample the imagery by a factor of the value to reduce processing times, a downsample of 4 will reduce the number of pixels by a factor of 4. The pedr_list variable points to the local copy of a file containing a list of all the paths to all of the MOLA PEDR data. By default this is set to None to use the ODE REST API to grab the necessary PEDR data, which is much faster anyways.

```
left  = None
right = None
config1 = None
config2 = None
dem_gsd  = 24.0
img_gsd  = 6.0
output_path = None
maxdisp = None
downsample = None
pedr_list = None
```

Check if config2 was defined, if it was not just use the first config file again

```
if config2 == None:
    config2 = config1
```

For next two lines, print out the config into the notebook file.

```
!cat {config1}
```

```
!cat {config2}
```

Import a few python things to use later on, including the Image function to render images out into the notebook

```
from IPython.display import Image
from pathlib import Path
from asap_stereo import asap
import math
```

If the user did not specify a output directory, make one. Note this step only does something if the output_path is explicitly set to None. By default from the command-line interface ASAP will use the current working directory.

```
default_output_dir = '~/auto_asap/ctx/'
left, right = asap.CTX().get_ctx_order(left, right)
if output_path == None:
    output_path = default_output_dir + f'a_{left}_{right}'
```

Make that directory if needed.

```
!mkdir -p {output_path}
```

Make sure the notebook is now running in that directory

```
%cd {output_path}
```

5.1.1 Step 1: Download images

Now we are getting to the heart of the notebook workflow. First use step-one to download our left and right images using the moody tool. At the end of the command you can see we are using standard bash to redirect stdout and stderr to two log files, the first a log just for this step, the second a cumulative log file for the whole job.

```
!asap ctx step-1 {left} {right} 2>&1 | tee -i -a ./1_download.log ./full_log.log
```

5.1.2 Step 2: Preprocessing through ISIS

Now we replicate the preprocessing from the asp_scripts project/ames stereo pipeline using ISIS commands. This step will run these steps in the following order: mroctx2isis, spiceinit, spicefit, ctxcal, ctxevenodd.

```
!asap ctx step-2  2>&1 | tee -i -a ./2_ctxedr2lev1eo.log ./full_log.log
```

5.1.3 Step 3: Metadata init

Now we create a number of metadata files used by the asp_scripts project to simplify future command calls. We also copy our preprocessed CTX cub files into a new working directory where all the stereo products will be computed. This new directory name uses both image IDs joined by an underscore '{left_id}_{right_id}', for example: "B03_010644_1889_XN_08N001W_P02_001902_1889_XI_08N001W".

```
!asap ctx step-3
```

5.1.4 Step 4: Bundle adjustment

We will use the parallel_bundle_adjust command from Ames Stereo Pipeline to refine the spacecraft position and orientation. The user can later re-run this step with more advanced options or GCPs if so desired.

```
!asap ctx step-4 2>&1 | tee -i -a ./2_bundle_adjust.log ./full_log.log
```

5.1.5 Step 5: Stereo first run (steps 1-3 of stereo in ASP)

Now we can start making our first dem, we pass in the stereo config file to parallel_stereo. We split this into two parts (step 5 & 6) as we may want to run each part with slightly different parameters or give us a chance to inspect the outputs before the final step which can be long running. In the future Step 5 & & maybe reconfigured into the 4 sub-steps for further improvement to the workflow.

```
!asap ctx step-5 {config1}  2>&1 | tee -i -a ./3_lev1eo2dem.log ./full_log.log
```

5.1.6 Step 6: Stereo first run (step 4 of stereo in ASP)

Run step 4, see step 5 above for more information.

```
!asap ctx step-6 {config1}  2>&1 | tee -i -a ./3_lev1eo2dem.log ./full_log.log
```

5.1.7 Step 7: Produce low resolution DEM for map projection

We have made a point cloud, but it is preliminary so we will use it to make a 100 mpp DEM to map-project the CTX images to, to produce a better 2nd pass DEM.

```
!asap ctx step-7 --mpp 100 --just_dem True --dem_hole_fill_len 50 2>&1 | tee -i -a ./
↪4_make_100m_dem.log ./full_log.log
```

5.1.8 Step 8: Make GoodPixelMap and Hillshade Previews

We make image previews of the DEM using the next few steps to check for issues with our first pass DEM. First we will render out the good pixel map image and then the hillshade of the DEM to look for issues with the topography.

```
!asap ctx step-8
```

Use some python to specify a new file name for the png version

```
both = f'{left}_{right}'
img = f'./{both}/results_ba/{both}_ba-GoodPixelMap.tif'
out = img.replace('.tif', '.png')
```

Use gdal_translate to produce a png version of the hillshade image.

```
!gdal_translate -of PNG -co worldfile=yes {img} {out}
```

Display the image in the notebook.

```
Image(filename=out)
```

Now again for the hillshade

```
both = f'{left}_{right}'
img = f'./{both}/results_ba/dem/{both}_ba_100_0-DEM-hillshade.tif'
out = img.replace('.tif', '.png')
```

Convert to a png file again.

```
!gdal_translate -of PNG -co worldfile=yes {img} {out}
```

Display the image in the notebook.

```
Image(filename=out)
```

5.1.9 Step 9: Mapproject ctx against 100m DEM

We now map-project our ctx images against our low resolution DEM to reduce image distortion for our 2nd pass DEM.

```
!asap ctx step-9 --mpp {img_gsd} 2>&1 | tee -i -a ./5_mapproject_to_100m_dem.log ./
↪full_log.log
```

5.1.10 Step 10: Stereo second run (steps 1-3 of stereo in ASP)

Same as step 5, just using the new map projected images this time.

```
!asap ctx step-10 {config2} 2>&1 | tee -i -a ./6_next_level_dem.log ./full_log.log
```

5.1.11 Step 11: Stereo second run (step 4 of stereo in ASP)

Same as step 6, just using the new map projected images this time.

```
!asap ctx step-11 {config2} 2>&1 | tee -i -a ./6_next_level_dem.log ./full_log.log
```

5.1.12 Step 7&8 again: create preview DEMs and Hillshade

We have made our second point cloud, so we should export some visuals as before. The parameter '–folder' just specifies that we are saving things into a different directory this time around.

```
!asap ctx step-7 --folder results_map_ba
```

```
!asap ctx step-8 --folder results_map_ba
```

5.1.13 Step 12: Get PEDR Shots for PC alignment

The final important step in the make_dem workflow is to get the MOLA PEDR data for the region we care about. Again, our data is not completely done until it has been aligned to the MOLA topography. If we had GCPs in the bundle adjust stage this would not be as big of an issue, but since it is relatively easy to align to MOLA we don't need to go through the process of producing GCPs.

```
!asap ctx step-12 {pedr_list}  2>&1 | tee -i -a ./7_pedr_for_pc_align.log ./full_log.
↪log
```

5.1.14 Make Final GoodPixelMap and Hillshade Previews

Nothing too surprising here, just export PNG versions of the images we care about to see the DEM at this stage of the processing.

```
both = f'{left}_{right}'
img = f'./{both}/results_map_ba/{both}_ba-GoodPixelMap.tif'
out = img.replace('.tif', '.png')
```

```
!gdal_translate -of PNG -co worldfile=yes {img} {out}
```

```
Image(filename=out)
```

```
both = f'{left}_{right}'
img = f'./{both}/results_map_ba/dem/{both}_ba_24_0-DEM-hillshade.tif'
out = img.replace('.tif', '.png')
```

```
!gdal_translate -of PNG -co worldfile=yes {img} {out}
```

```
Image(filename=out)
```

One additional bit here, for the MOLA data, show the PEDR2TAB template if created and the amount of PEDR data we have to align to. If the final line is less than a few hundred we could be in a bad situation.

```
!cat ./{left}_{right}/PEDR2TAB.PRM
```

```
!cat ./{left}_{right}/{left}_{right}_pedr4align.csv | wc -l
```

Now that we have finished the first half of the workflow we can inspect the output products for issues before moving forwards. If there are issues noted in the log or after a particular step, that step can be re-run with different parameters until a good solution is found.

At this point, we have a completed DEM! However, it's absolute position in space maybe off from the correct position. Therefore, we must now perform a point cloud alignment to align our DEM with reference topography, in this case MOLA PEDR data to correct the position of the CTX DEM. In older versions of ASAP, this point is the dividing line between the make_dem and align_dem pipelines.

The "maxdisp" parameter in particular deserves attention. It is the number passed to pc_align's –max-displacement parameter in the Ames Stereo Pipeline. Basically, it is the value of the distance you expect to move the CTX DEM to become aligned to your reference DEM (in this case, the PEDR data). It is generally worth estimating this number using a GIS to sample points in both the DEM and reference file, and seeing how far away they are from each other. But, CTX can be well behaved with ASP, so we pick a default of 500 meters which can be large enough for many situations.

5.1.15 Step 13: Align the DEM to MOLA

This is the most important step in the 2nd half of the workflow as all the remaining steps are just producing final science products and visuals for the logs. This step runs pc_align using the provided max displacement (aka disparity). If the logs indicate a larger displacement was observed than the user provided value it will need to be re-run using larger values or with other advanced parameters. If users see issues it is generally easyier to re-run the pipeline at this step repeatedly in the command line or inside the Jupyter notebook.

```
!asap ctx step_13 {maxdisp} 2>&1 | tee -i -a ./8_pc_align.log ./full_log.log
```

5.1.16 Step 14: Make the final CTX DEM

After the previous step everything after is simple and easy as we now have a final aligned point cloud from which DEMs and ortho images can be made. That is all the rest of the steps do, they generate final DEMs with the geoid adjustment to produce science ready DEMs and ortho images for mapping.

```
!asap ctx step_14 --mpp {demgsd}  2>&1 | tee -i -a ./9_dems_orthos.log ./full_log.log
```

5.1.17 Step 15: Adjust final CTX DEM to Geoid (Areoid)

```
!asap ctx step_15 2>&1 | tee -i -a ./10_geoid_adjustment.log  ./full_log.log
```

5.1.18 Make the final CTX Hillshade and Orthos

```
!asap ctx step_8 --folder results_map_ba --output_folder dem_align 2>&1 | tee -i -a ./
↪11_hillshade.log ./full_log.log
```

```
img = './' + str(next(Path('./').glob('./*/results_map_ba/dem_align/*_ba_align_24_0-
↪DEM-hillshade.tif')))
out = img.replace('.tif', '.png')
```

```
!gdal_translate -of PNG -co worldfile=yes {img} {out}
```

```
Image(filename=out)
```

```
!asap ctx step_14 --mpp {imggsd} --just_ortho True  2>&1 | tee -i -a ./12_img_full_
↪ortho.log ./full_log.log
```

CHAPTER

SIX

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

a

INDEX

H

I

M

N

P

Q

R

S

T

W