ASIFT Workflow Status Report

Mike MacFerrin, University of Colorado

April 20, 2018

The ASIFT Workflow is a set of combined algorithms made available through the IceBerg project to automatically geo-locate old aerial or satellite imagery using a database of well-know, geo-located and ortho-rectified current imagery.

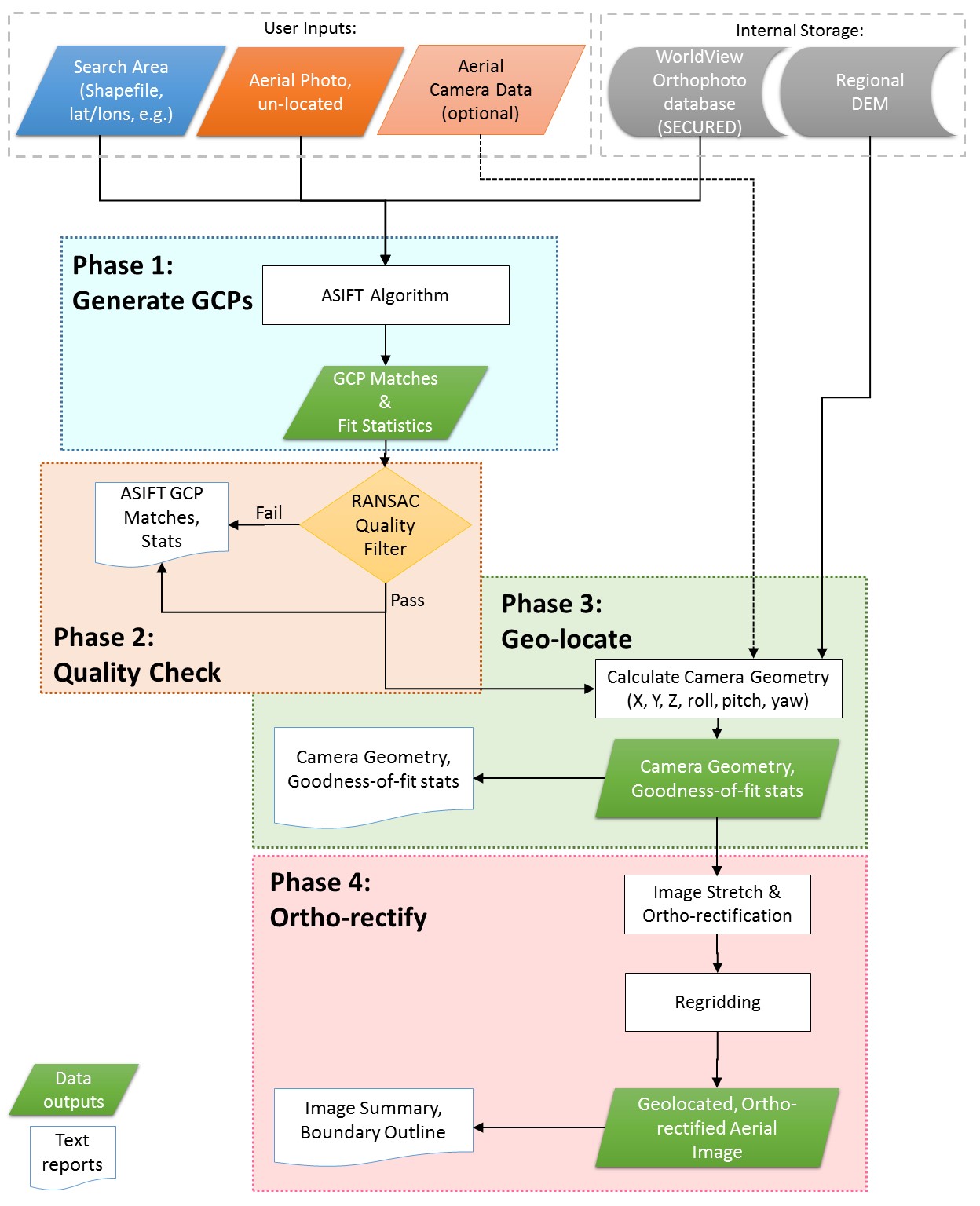
The Goal of the ASIFT Workflow is for a user to take an airborne or satellite image anywhere that we have WorldView Ortho-rectified imagery available, and where the user knows the approximate location of the photo (within ~100 km) but does not have geo-location information on it. The ASIFT Workflow be able to pin down where that photo was taken, automatically determine a likely set of geo-located ground control points for the image, and ortho-rectify the image if needed.

Pre-conditions:

To maximize the chances of success, the photo provided should contain:

* Geographically-distinct and permanent features: There must be distinct, relatively-high contrast features within the image that would create good “keypoints” for the ASIFT algorithm. Mountainous regions are good, for instance. The flat-white featureless interior of an ice sheet is not.
* The keypoints must be invariant through time (seasons to decades): i.e. a stable mountain ridge is a good keypoint. A drifting sand dune, an eroding riverbank, or flat white fields of snow or grass are not good keypoints.
* The user must have a general idea where the image is located (the ASIFT algorithm cannot search the entire world, at least not yet). The user can provide one of the two following pieces of data to fulfill this requirement:
  + A center lat/lon location and a search radius (within 50km of 45.8 N 142.3 W, e.g.)
  + A shapefile in WGS84 (lat/lon) coordinates, containing a single polygon within which to search (i.e. a shapefile outlining the Antarctic Dry Valleys or Baranof Island, Alaska)

ASIFT Workflow Diagram:



ASIFT Status Update, April 20, 2018:

So far, a working prototype of Phase 1 has begun, but is not yet complete. In the “IceBerg-Project\ASIFT” repository, in a folder called “Phase 1 – Keypoint Generation” the following files are contained:

* **ASIFT\_EXECUTABLE\_README.txt**: directions from Morel & Yu (2010) explaining how to compile and run the ASIFT v1.0 executable
* **demo\_ASIFT\_src.zip**: Original source code for the ASIFT executable (Mac & Linux)
* **demo\_ASIFT\_Win.zip**: Original Installer for the ASIFT Executable (Windows)
* **Run\_ASIFT\_Test\_Cases.py**: A Python 2.7 file to run the test cases in their current state.
* **Test\_Case\_Descriptions.csv**: A comma-separated text file providing the meta-data descriptions of the ASIFT test cases, used by “Run\_ASIFT\_Test\_Cases.py” to run the test cases.

Several folders are also contained in the directory:

* **demo\_ASIFT\_src** and **demo\_ASIFT\_Win:** Unzipped & compiled/installed versions of the ASIFT executables, from the zip files. These may be deleted and overwritten when compiling the ASIFT code on a new computer.
* **TEST\_CASE\_IMAGES:** Contains all the images used by the test cases described in “Test\_Case\_Descriptions.csv”
* **TEST\_CASE\_0:** Outputs of Test Case 0: the “adam1” & “adam2” images provided by Morel & Yu (2010), known to work well with the current algorithm.
* **TEST\_CASE\_1:** Overlapping segments of an identical Worldview image.
* **TEST\_CASE\_2:** One WorldView image segment overlapping aother Worldview image segment
* **TEST\_CASE\_3:** A recent airborne image overlapping a WorldView image (not yet implemented)
* **TEST\_CASE\_4:** An historical airborne image overlapping a WorldView image
* **TEST\_CASE\_5:** A recent airborne image overlapping a stack of WorldView images (not yet implemented)
* **TEST\_CASE\_6:** An historical airborne image overlapping a stack of WorldView images.

ASIFT Executable Limitations

The ASIFT Executable currently has several limitations, which we will work around or fix during the remainder of the project.

1. PNG read/write only. Currently the image library used by the “demo\_ASIFT” algorithm can only read PNG images.
   1. The current work-around is to save all GeoTiff images (worldview, etc) to PNG format before executing ASIFT
   2. In future versions, we will modify the ASIFT executable code in order to run on Tiff formats without needing the PNG intermediary.
2. There is a HARD-CODED assertion in the ASIFT executable that does not permit the # of image rows plus the image kernel size to exceed 4000 pixels. No images greater than ~3500 pixels on an edge will currently run.
3. On many images >3000 pixels, the ASIFT executable reaches a recursion depth limit and crashes. Mostly, images containing only 2000 pixels or less on a side should be used. Two potential solutions exist for this problem:
   1. Re-factor the ASIFT code to run iteratively rather than recursively. This will allow the executable to use the full memory heap rather than being limited by the (far smaller) stack limit.
   2. Tile all images (both the worldview and
4. Images with high-contrast work best. Images that are too dark or too light will be

How to Run the ASIFT Executable:

Mac/Linux: compile the code and build it in a “build” directory within the “demo\_ASIFT\_src” folder. The following bash shell commands should do this:

cd demo\_ASIFT\_src

mkdir build

cd build

cmake ..

make

A “demo\_ASIFT” executable should now reside in the “build” directory (on Linux).

To execute “demo\_ASIFT”

> demo\_ASIFT img1.png img2.png imgOutVert.png imgOutHori.png matchings.txt keypoints1.txt keypoints2.txt 0

Paremters:

img1.png & img2.png: Input filenames

imgOutVert.png: an output image with img1 & img2 vertically aligned, with white lines connecting keypoint matches

imgOutHori.png: an output image horizontally aligned, with white lines connecting keypoint matches

matchings.txt: a text file identifying matching keypoints in each image

keypoints1.txt & keypoints2.txt: a text file containing locations and supporting data of each keypoint found in the respective image.

To Run the ASIFT Test