FULLY CONTROLLING MARS RECONNAISSANCE ORBITER CONTEXT CAMERA IMAGES AND PRODUCING COSMETICALLY STABLE MOSAICS: METHODS. S.J. Robbins[*,1], M.R. Kirchoff[1], R.H. Hoover[1]. [*]stuart@boulder.swri.edu, [1]Southwest Research Institute, 1050 Walnut Street, Suite 300, Boulder, CO 80302.

Introduction: The Context Camera (CTX) aboard NASA's *Mars Reconnaissance Orbiter* (*MRO*) [1] has been returning high-resolution (5–6 mpp) and -quality data of Mars' surface for over a decade. As of PDS release 55 (March 2021, including data through August 2020), the instrument has returned >115,000 images that cover ~99% of the planet in good quality. However, images often have ~100s meter offsets from each other and a controlled ground source, resulting in seam mismatches when mosaicking and poor matches to other, high-resolution datasets.

Over the last several years, we developed and improved upon an efficient, accurate workflow within *ISIS* (USGS's *Integrated Software for Imagers and Spectrometers*), driven by Python scripts, to automate much of the control process that can create a fully controlled CTX dataset. We demonstrated the viability of this workflow by producing a mosaic of Mare Australe ("MC-30"), covering south of –65°N, or 4.7% of Mars' surface [2]. We have also done other regions of Mars, totaling >50% of its surface.

Over the past year, we have further improved the efficiency and speed, which have allowed us to create fully controlled networks for entire Mars Charts ("MC") in about one week (~3% of Mars, ~3000 images, ~2.0 TB of data). In this abstract, we discuss our new method and why it is faster than our previous approach. In a companion abstract to the 2021 Geologic Mappers' Meeting, we discuss our current products including methods of cosmetic control [3].

Previous Automated Control Network Workflow [2]: For context, we first describe our previous method of creating cartographic control networks, since it is easier to understand and therefore compare our new workflow against. After standard calibration in *ISIS*, we begin the control process by finding polygons where at least two images overlap. We originally did this using *ISIS*'s `FINDIMAGEOVERLAPS` (all *ISIS* programs in this abstract are in fixed-width font), but we found that the software was incredibly slow when the number of images was large, and it was single-threaded. We created an alternative version in Python that we could multi-thread, and we went further and created an alternative to `AUTOSEED`, which creates a grid of candidate points, with a set spacing, where those overlap polygons exist. We set the reference image ("measure") to be that image with the minimum emission/slew angle, the idea being that was the most likely to match to any random emission angle image.

Our code then proceeded to try to register (match) the measures for each point, using `POINTREG`. It used templates of different sizes to give a range of feature patterns to try to match, and the code required that a match be made in at least two different templates for the match to be saved. Any surviving points were merged into a single control network.

That single network was solved (`JIGSAW`), and errors were propagated through. Our code then invoked a "skimmer," where the largest-residual points were extracted ("skimmed") and attempts were made to try to register them again, using larger template sizes. Any that registered were placed back into the network, which was solved again, and residuals examined. Any points previously skimmed that were still large residuals were discarded. This process looped until the 99.99th percentile of the point residuals was <1.0 pix.

Next, the code examined the convex hull ratios (CHRs) of images. CHR is the area of a convex polygon that encompasses all points, divided by the area of the image. It does not guarantee a good distribution of points, but it is a reasonable indicator of what images might need more points. The image was then divided into 25 equal latitude-longitude regions, and a sample `AUTOSEED` grid with half the spacing of the original was examined. Any points in the new grid that were in a 1/25th region of an image that would significantly increase the CHR and point count of that region were saved. After all areas of all images were examined, those saved points were registered via `POINTREG`, and the process begun three paragraphs earlier was repeated through four successive grid-size halvings.

The entire process for 1/480th of Mars (1/16th MC) took ~1 day, though areas of significant repeat coverage (landing sites) or the poles took much longer. The manual effort required to examine and fix remaining issues was minimal, ~5–15 minutes each, except polar regions. It should be noted that this code was also tested and worked with MDIS images of Mercury, ISS images of Saturn's mid-sized moons, and LORRI images of Pluto and Charon.

New Automated Control Network Workflow: Our new workflow grew from looking at the common issues we had with the previous code's product and a desire to increase the speed. A common issue that also affects speed is that too many sample points were produced and registered, wasting time. Even with a thinning code applied later, we still wasted time creating and matching those points we would later remove. For example, the code might produce hundreds of points on two images that overlap and match well, but that many points is much more than is needed. We also found that on images that were large relative to the body (such as ISS of Saturn's moons), the grid approach could miss some images entirely.

To mitigate these issues, we started from a different premise: Instead of a grid, we create a list of all unique image overlaps. That means for a three-circle Venn Diagram, there would be four unique overlaps: Each

combination of two circles, and a fourth for all three overlapping. By looking at individual overlaps, we can (a) make sure each overlap has a *maximum* number of points, while also (b) ensuring that sometimes critical links between images can be attempted.

Our code sorts overlaps by number of images on them, and it calculates what the CHR and area coverage would be *if* that overlap were completely filled by points. If it significantly improves either, the code adds that overlap to a "To Do" list. This check means we can ensure we include small overlaps that may cover many small images, while excluding potentially large overlaps in the middle of an even larger image where more points would not benefit the network.

Points are made randomly on overlaps in the reduced list. We create a minimum of 2 points in each overlap, up to a maximum of 30 points based on area. We make several versions of each point, each version with a different reference image. The code tries to register them (`POINTREG`) with multiple templates, requiring at least *three* matches to within 1 pixel for it to be saved. After `POINTREG` is done for an overlap, the code examines the same point made with different references and keeps that with the most measures. If that best point is missing some measures, the code goes through remaining versions of the point and saves the one(s) needed to include those missing images, if it can. The code next looks at the saved points in the overlap and if it is less than a threshold, it will try to create more, repeating this paragraph.

After finishing all overlap areas, the networks are merged, solved, and the iterative skimmer is run, this time requiring the 99.999th percentile of the points be <1.0 pix. Then, the CHRs and area coverage of each image are again analyzed. The code tries to create more points on any overlap area that includes any image whose CHR or area coverage of points falls below a certain threshold from what they could have been, based on the original overlaps selection. It then does the entire overlaps selection again, lowering the threshold for what could be a CHR or area coverage increase. If any image's registered points fall below a threshold from the new maximum possible, then the overlaps covering that image are saved to a new "To Do" list. The code takes that list and repeats the process from the previous paragraph, up through the skimmer step in this paragraph.

The manual effort required to fix remaining issues, for a full MC region, is on the order of ~10–20 hours. While this might seem to be arduous, it is not bad when considering that that would be 15 weeks' time for *all of Mars at 6 m/pix*. We are also actively working on code improvements to reduce this time.

New Automated Code Benefits: (a) The new code is faster than the old code in our tests, completing a benchmark in 29 minutes versus 65 minutes, on the same hardware. Even though the CPU time put into making and registering each point is longer, that is more than made up for in the many fewer points made and tested. We are able to move from 1/16th MCs up to *full* MCs, controlling the entire Tharsis Chart (MC-09) on a 36-core PC in only one week.

(b) The new code builds more efficient networks than the old code, even though some points are duplicated (but offsets in NASA's SPICE means duplicated points are offset by 100s m). From our old code, a typical 1/16th MC made a network ~70–250 MB in size. From our new code, a full MC is ~50–110 MB, meaning these networks are ~10–20× smaller in file size.

(c) The new code builds networks with better CHR coverage and network depth than the old code. The average CHR is ~5–10% more in these networks compared with those from the previous, almost certainly an effect of emphasizing coverage in each unique overlap rather than a grid. Also, because we emphasize more measures per point, the average number of measures per point ("network depth") is much higher.

Other Bodies: The old code had specific alternative methods for some parts to make it work with other instruments on other bodies, and the improvements in the new code were not "made aware" of those different methods. We tested the new code on a small benchmark region of Mercury, with 145 MDIS images spanning 20–500 m/pix and the full range of solar incidence angles, and we found, without any adjustments, the new code easily produced a control net for all images, and a mosaic showed no offsets at the seams.

Standards: Our work uses the community-standard *ISIS* software, meaning that all tracking of uncertainties and other types of output produced by this software are maintained. Our Python wrapper uses standard libraries, and Python is a free compiler that can be run on almost any computer. We use native Python libraries to divide the work for each region into multiple areas to take advantage of modern high-core-count computers, allowing it to scale well, even up to a cluster. Only a few tasks truly need to be done in serial, on one processor (*e.g.*, `JIGSAW` network solver).

Future Work: While we are still making some code improvements, at this stage we are mostly applying our code to cartographic control problems, focusing on CTX coverage of Mars, and optimizing it for that work. However, the code works well on other bodies, to which we plan to apply it, and we are also happy to discuss any collaboration (please e-mail the corresponding author!).

References: [1] Malin et al. (2007). "Context Camera Investigation on board the Mars Reconnaissance Orbiter." doi:10.1029/2006JE002808. [2] Robbins et al. (2020). "Fully Controlled 6 meters per pixel Mosaic of Mars' South Polar Region." doi:10.1029/2019EA001053. [3] Robbins et al. (2021). "Fully Controlling Mars Reconnaissance Orbiter Context Camera Images and Producing Cosmetically Stable Mosaics: Products." Planetary Geologic Mappers' Meeting, Abstract 7003.

Funding: This work was funded internally by Southwest Research Institute.