SSC24-S3-10

Automation as a Mindset: An Approach to Streamlining Spacecraft Development and Operations

Annalisa Vilaysing, Terrance Yee, Elizabeth Nelson
Malin Space Science Systems, Inc.
5580 Pacific Center Blvd, San Diego, CA 92121; 858-552-2650 x215
vilaysing@msss.com

ABSTRACT

With limited opportunities and funding for deep space missions, there is increased pressure to build better, faster, and cheaper space-borne payloads. Smaller teams and limited tools provide a unique challenge for such development. This paper reviews the approach the Micro-Mission Systems (MMS) group at Malin Space Science Systems (MSSS) took in the development of the Mars Synchronous Orbiter (MSO) to streamline and automate various tasks so team members could focus their efforts on solving more difficult problems and minimize user induced errors.

The team utilized free tools to automate labor intensive and manual processes, run and monitor tests with little to no user intervention, and build complex flight operation scripts and timetables. Some of these tools were also integrated with commercial flight software systems to deepen the level of automation that could be accomplished and allow for a very lean staffing plan.

The integration of automation at every stage of development culminates within the spacecraft itself in two forms: an autonomous Fault Detection, Isolation, and Recovery (FDIR) system and on-board image processing in the infrared (IR) and visible range. These processes are not unprecedented in spacecraft; however, MSO pushes the envelope on the robustness of these processes in the application of deep space.

In the first autonomous process, the FDIR subsystem monitors key parameters of the spacecraft's health, reboots upon detection of major faults, and proceeds with its primary operations and science objectives on its own. It will only seek ground intervention if it cannot recover from faults autonomously. This subsystem allows routine reboots to be scheduled into nominal operations so that minor errors have less of a chance to accumulate into major errors.

In the second autonomous process, on-board payload image processing was designed to reduce the volume of downlinked data while still delivering products sufficient for science analysis. MSO implements multiple algorithms to improve signal-to-noise ratios for the IR products and to construct high-dynamic range (HDR) visible images that still retain accurate radiometric data. Compression using JPEG2000 results in a data reduction factor up to or exceeding 1000. Raw payload data and lossless final products are generally available - downlink permitting - allowing for a robust delivery system of imaging products.

1 INTRODUCTION

Spacecraft development is a multi-faceted process requiring expertise from different disciplines and coordination of many moving parts. While it's preferable for responsibilities to be compartmentalized within a team, limited funding for deep space missions require team members to wear multiple hats and be more efficient with less manpower, time, and tools. Limitations may breed innovation, but it also demands automation.

During the development of the Mars Synchronous Orbiter (MSO), the Micro-Mission Systems (MMS) group at Malin Space Science Systems (MSSS) fluctuated between 6 to 9 members at any given point in time and their total responsibilities ranged from the spacecraft's flight software to imaging systems and hardware design to functional testing. Automation systems were necessary to streamline and optimize particular hand-heavy and laborious processes in order to free up resources to focus on other complex issues and development. This paper will discuss three automation tools/systems that were used to execute a complex set of equations to model MSO's battery performance, execute and monitor spacecraft test procedures, and build detailed flight operation scripts and timetables. These tools and systems were built using free software such as Python, Grafana, and Google Apps Script provided by the Google Suite. Some of these tools were also integrated with RocketLab USA's MAX flight software system or Ansys Government Initiative's (ASI) Satellite

Tool Kit (STK) to deepen the level of automation that could be performed on the development system with minor oversight, allowing for a very lean staffing plan.

Approaching automation as a mindset for the entire team and the development of the spacecraft became reflected in the operating ideology of the spacecraft itself. MSO benefits from three major automation processes which allow it to run autonomously and efficiently: a Fault Detection, Isolation, and Recovery (FDIR) subsystem, a robust on-board processing algorithm for infrared (IR) image processing, and an on-board processing algorithm for visible image processing that produces high-dynamic range products. These systems are not uncommon in spacecraft; however, the design of these systems grant MSO more autonomy in what it can do, especially in the application of deep space operations.

This paper will discuss the logic behind these subsystems and the particular challenges solved by them. The FDIR system monitors key parameters of the spacecraft's health and takes action independently to keep the spacecraft performing nominal operations. This system grants MSO the ability to perform a system reboot upon detecting a major fault and return to pre-planned scripts and sequences to continue its primary science and communication relay tasks. A major benefit of this system also allows routine weekly reboots to be commanded in nominal operations so that an accumulation of minor errors have less of a chance to become major errors. This drastically reduces the amount of downtime when faults are detected and the need for ground support for the spacecraft to perform its objectives.

The on-board processing algorithms for the IR and visible imaging products were implemented because MSO does not nominally have the downlink budget to deliver the volume of desired raw data to ground. These algorithms allow for more data to be sent without compromising the downlink rates required to send that data or the scientific accuracy of the data. IR imaging products generated by MSO have greater signal-to-noise ratio (SNR) improvements through real-time multi-point non-uniformity correction, the averaging of hundreds to thousands of image acquisitions, and operations in 16-bit space. High-dynamic range visible imaging products generated by MSO have multiple exposure data constructed into one image through the use of a robust patch-based system and adaptive weighting functions done in Bayer-space. File sizes of the imaging products are further optimized through JPEG2000 compression and are returned to ground progressively which results in a data reduction factor up to or exceeding 1000. MSO also has enough on-board storage capability to hold raw payload data and lossless image products for ground users should data rates permit downlink.

2 OPTIMIZATION OF WORKFLOW

Power Modeling Tool

A major benefit of automation is that tedious, repetitive, and frankly annoying tasks can be executed accurately, indefinitely, and without complaint. Modeling the state of charge of MSO's battery at various flight modes and operational environments generated a lot of complaints within the team and as such, was a prime candidate for automation.

This model was required to simulate the battery's state of charge and the results would be used to properly size and configure set points for the battery. There were a rather lengthy number of input parameters that battery state of charge is dependent on such as battery resistance coefficients, environmental temperature data, component power loads, solar array efficiency factors at different distances from the Sun, and solar incidence angles just to name a few of them. Properly managing all the input data on top of the different edge cases to be tested was difficult and prone to human error. It also took a lot of time to manually define the component duty cycle data for component load data calculations.

The automation of setting up and feeding data into this model was implemented through two software systems: Google Apps Script and AGI's Systems Tool Kit (STK) software. Google Apps Script is a low-code platform that allows tasks to be programmatically executed across products in Google's suite of online applications and was used to write all of the code for automation. STK is a modeling and simulation software for digital mission engineering and systems analysis. Further augmented with SOLIS, a module to model satellites and their systems, a digital recreation of MSO was created to simulate all of the environmental conditions that would be fed into the battery model.

The first stage of the automation code would intake the desired time scale and duration of the test case to be modeled and output a set of blank worksheets that would be automatically formatted with the input parameters and associated equations of the battery model. While the sheets are being generated, a user can adjust the scenario parameters in STK to the same desired time scale and duration and run a simulation that would output data for the time dependent input parameters (component duty cycles, environmental temperature, solar intensity, and solar incidence angles) of the battery model. This output of input parameters was designed to be formatted in a way that can directly interface with the automation code,

allowing the only action from the user to copy and paste the file into a location the automation code can access.

The second stage of the automation code would then parse the time dependent input parameters into the blank worksheets and ask the user for the remaining non-time dependent input parameters such as battery characteristics and expected efficiency margins. Since all worksheets were generated with the required equations, it only takes a few seconds for the data to propagate through all the worksheets and finalize with a graph of the battery's modeled state of charge, total loads, and generated power.

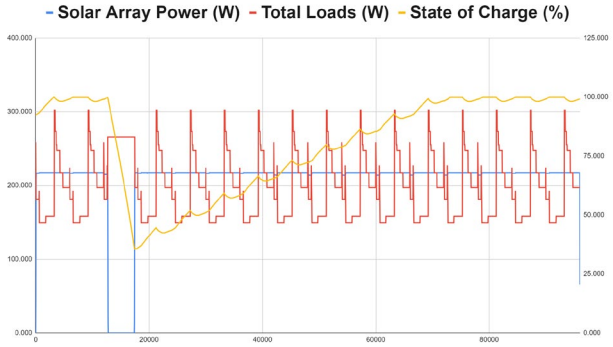

Figure 1: Output of the battery power model automation code for a case where the spacecraft is operating under the coldest expected conditions. The model was generated at a time step of 30 seconds and an analysis duration of 1 day. The blue line graphs the generated solar array power, the red line graphs the total component loads, and the yellow line graphs the battery state of charge.

This automation code turned adjusting time scales for more fine or coarse calculations from a nightmare into a trivial matter, allowed for multiple test cases to be modeled simultaneously since it ran off of a central code base, and helped with determining a solution for the battery's size and set points much more quickly than if this process was done manually.

Hands-Off Testing Procedures

With only a single test engineer on the development team, hands-off testing procedures were necessary to decrease the set up and observation time of testing sequences on the spacecraft. Automation tools allowed for multiple tests to be performed in parallel and adequately managed by one person. The automation of test procedures were implemented in several stages: 1) determining when and where automation scripts were necessary, 2) implementing monitoring scripts and automated shutoff precautions, and 3) utilizing operation dashboards and visualizations of databases that were autonomously fed data from the test environment.

Multiple interconnected tools were necessary to carry out automated testing of the spacecraft. These include:

- Remote control of spacecraft components and ground support equipment;
- Command scripts for RocketLab USA's Ground Data System (GDS) for MAX, an off-the-shelf flight software system;
- Telemetry monitoring and alarm scripts written in Python;
- PostgreSQL databases, an open-source database management system; and
- Grafana, an open-source web application that allows users to visualize and query data.

Environmental testing required functional tests to constantly run in order to verify survival under thermal stress. This posed an opportunity to automate component operations and monitoring. Both the commands being sent and the alarm system setup are included in scripts, which eliminate the need for an operator and attendant during tests which last days to weeks long. Thus, while one automated test is running, another test requiring manual commands can be performed simultaneously, and multiple milestones are met within a short period of the project.

During component level tests, power supplies can be powered on or off remotely using a Python script and General Purpose Interface Bus (GPIB) connection, so long as the power supply has the capability. This allows engineers to power on components and run tests using MAX GDS remotely instead of requiring them to be physically present with the component. For example, during image capture tests, one can remotely turn on the power supply, remotely connect to the flight computer, and remotely actuate the command to capture and immediately analyze a picture.

Python scripts can also be used to alert test personnel in real time as soon as the monitored data becomes out of bounds. Multiple telemetry points of interest can be monitored at the same time, and the script differentiates them to alert the responsible engineer. A simple text message is sent which states exactly which test point is out of bounds and continues to send texts until returned to a nominal state. Multiple team members can be added to the text system to increase reliability and avoid a loss of hardware.

This is useful not only for safety of the component being tested and the environment but also for timekeeping when analyzing data later on. If, for example, the temperature of the hardware approached a dangerously high level, the monitoring script recognizes the issue and

immediately shuts off the power supply before damage has occurred to either the component or the hardware surrounding it. The temperature increase may also correlate to the effects on the hardware's voltage reading, which could be traced using both the time stamp from the alert and the overall trend from the data.

After MAX GDS is launched, a Python script is executed from the command line which immediately opens the necessary telemetry windows. This is an enormous time saver, as the operator no longer has to manually request the windows. MAX GDS can then be connected to a PostgreSQL database to allow for further interfacing with Grafana. Thresholds are added to live feed graphs which alert the operator to correct the spacecraft if moving towards a hazardous state. These thresholds can include high and low values, as well as yellow or red levels of alarm (or any degree of differentiation desired). Grafana dashboards can be both focused and general; a dashboard is created for each specialized operations role. Examples include a dashboard for the attitude determination and control system and a dashboard dedicated to the propulsion system. A separate dashboard exists for a top level view of the most critical components on the spacecraft, such as battery state of charge, ADCS control mode, and operational status - all in the same view.

Additional Grafana dashboards can be used for component level testing. Separate PostgreSQL databases can be created for use with testing that does not require the MAX GDS operational commands and telemetry data. Python scripts can run tests of printed circuit boards in a loop and relay data back to a host on a PC. Multiple dashboards can be run at once from separate databases. Therefore, a telemetry dashboard generated directly from MAX GDS can be viewed in tandem with the dashboard viewing the information from the PC's database. Data collected by the python scripts can also be saved and analyzed with and without the use of Grafana.

Overall, the use of automation has saved the team time and effort by granting the ability to run multiple tests at once without direct oversight, keep spacecraft hardware running within safe levels, and query and view large volumes of data.

Collaborative Flight Operation Plans

Flight operation plans can be written mainly in one of two methods: 1) define a particular set of commands for one period and then repeat that set for the desired amount of time, or 2) define a command for each epoch for the entire duration of the plan. Each method has its pros and cons. The first method is simpler to define and write and produces neater, shorter scripts, but leaves no room for additional commands outside of the routine operation. Furthermore, any unscheduled spacecraft reboots would only start the script at the beginning of a sequence, potentially skipping an operation that otherwise would have been expected from the ground. The second method allows for specific commands or complicated sequences at very precise times and in the event of a spacecraft reset, operations would pick up where it left off or onto the next expected command. However, the scripts using this method can become very long, tedious to write, and difficult to debug if any errors occur during testing.

Then, the complexity of generating these scripts was further increased as conflicting priorities were introduced such as requests for routine orbit maintenance maneuvers, communications relay with ground assets on Mars, or special observation passes of Mars for scientific research interrupt nominal operations. For this reason alone, the first method of generating flight operation scripts is unsuitable for MSO. In order to overcome the challenges of collecting the requests from multiple parties in a unified way, balancing these conflicting priorities, and generating the scripts without headache, automation comes into play.

The first challenge was solved through the usage of Google Sheets, a web-based application that enables multiple users to modify spreadsheets and share the data online in real time. This allows the multiple parties to input their requests as new rows to define their 'event timeframes' which is characterized by the category of the spacecraft maneuver or phenomena, the event start time, and event end time. A new feature of Google Sheets called 'Timeline' can then output a comprehensive timeline of all events to highlight priority event timeframes for ground users. However, this output only serves as a visual aid to find where interruptions to routine operations would occur. The generation of the flight operations script is performed autonomously via Google Apps Script that interfaces with this Google Sheet.

This automation script begins with definition of the hierarchy of priorities that were designed for MSO: 1) orbit maintenance maneuvers, 2) Mars communication relay passes, 3) special observations of Mars phenomena, 4) Earth occultation or if DSN communication is otherwise not available, 5) eclipse periods, and 6) a combination of Earth occultation and eclipse period. In the exclusion of any of these priorities, MSO routine operation sequence is a 15 minute block: an imaging pass of Mars for 2.5 minutes before a communications pass for downlink for 12.5 minutes.

With this information and the initial epoch that the flight operation plan will be executed, the automation script

will then build the flight script using several passes. The orbit maintenance maneuver and communications relay timeframes are placed first into the flight script. These two event timeframes are non-negotiable in the flight script and all remaining priorities and operations will be built around them.

In the second pass, the automation script will fill in the remaining epochs with the routine operation sequence and make modifications if the epoch falls within the remaining four priorities. If the epoch is within the timeframe of a:

- Special observation: then the imaging pass may be extended into the downlink pass;
- Earth occultation or DSN unavailability: then downlink passes will be replaced with a slew to the Sun for power generation;
- Eclipse: then operations may be reduced to only imaging or only communications based on battery state of charge; and
- Simultaneous Earth occultation and eclipse: then the spacecraft will only perform imaging until exiting one of the two priorities.

The resulting script is considered a rough cut of the flight operation plan and proceeds through additional passes for optimization. This initial flight script is written in FlightJAS, the programming language that the flight software uses, which is fed directly into SOLIS to simulate the entire operation plan. Key outputs from this simulation are battery state of charge levels and slew durations for each maneuver which are used for the final pass of the automation script. If the battery state of charge falls below the allowable threshold, then a few cycles of the routine operation sequence will be replaced with power generation. Slew durations are accounted for all maneuvers and can affect durations of other operations. For example, if an orbit maintenance maneuver requires 1 minute of burn and the slew for the burn takes 2 minutes, then the actual duration of the timeframe in the flight script needs to be 3 minutes. A few iterative passes of script adjustment then simulation may be required to resolve battery state of charge levels and timing of slews for operations.

After these passes of the automation script, the final flight operation script has been built and can be used for final simulation, review, and then verification and validation. This final step is performed on a flatsat that will stay on the ground which is a functional copy of the satellite itself using engineering model hardware. Realistic simulation of the spacecraft's behavior is done through another module of MAX, the On-Board Dynamic Simulation System (ODySSy). While SOLIS is a virtual simulation that can be used to validate the entirety of the flight operation plan at accelerated rates, ODySSy on the flatsat is run in realtime and is used to validate a representative day or two only. The telemetry produced by ODySSy, the on-board FDIR subsystem, and GDS are monitored autonomously and any dangerous or unsafe trends are identified in alarms without requiring extensive human data review. After these verification/validation steps the operations script is released for upload at the next scheduled DSN contact.

The use of automation for generating the flight operation script greatly reduces the amount of manhours and tedious planning that would have been required if performed manually and also removes human error that would inevitably occur with managing multiple priorities and different operation sequences. Moreover, the rapid generation of the flight script allows for quicker responses to modify the operation plan to capture special phenomena or respond to emergency events without compromising routine operations.

3 SPACECRAFT AUTOMATION

Fault Detection, Isolation, and Recovery

When Earth and Mars are at the largest distance between them, a signal can take approximately 21 minutes to be sent one-way. At best, it can take an hour for an error to be detected on the spacecraft, send a message for help to Earth, generate a solution, and then send the solution to the spacecraft to be executed. During the most infrequent contact periods, it might be a week before ground operators would even be aware of a problem. This represents the potential loss of an hour's or a week's worth of time to complete mission objectives for an issue that could have been solved by the most common troubleshooting technique: have you tried turning it off and back on again?

This is the core ideology behind the Fault Detection, Isolation, and Recovery (FDIR) subsystem on MSO. The subsystem is designed to detect serious faults and reboot the spacecraft (the turning it off portion) before autonomously returning to full flight operations after successful reboot, safe mode entry, and safe mode exit once telemetry is "green" across the board (the turning it back on portion). This will allow the spacecraft to tolerate transient faults and recover to full operability without the need for operator input. While minor faults that do not affect the spacecraft's performance or persistent faults that haven't been cleared through system reboots will be directed for ground intervention, the main purpose of the FDIR subsystem is to minimize disruptions to the completion of MSO's mission objectives. Leveraging the ability for autonomous return to nominal flight operations allows MSO to deliberately reboot the spacecraft once a week via command in the

flight operations script or through the expiration of a one-week deadman timer within the Master Power Controller. This removes any build up of non-volatile bit errors in memory and ensures a solid method of recovering spacecraft communications if any soft latchups are encountered.

The subsystem has been designed to be executed identically, independent of entry conditions, in order to reduce complexity in design and testing. There are a plethora of potential faults that could be monitored to trigger a spacecraft reset, but the main premise of the FDIR subsystem is that all serious errors inevitably result in one of three main conditions:

- Loss of battery charge, where the battery state of charge or voltage has fallen below allowed threshold levels;
- Loss of attitude control or knowledge, where the attitude determination & control system (ADCS) has begun detecting control errors; or
- Failure to pet one of the watchdogs, where a subsystem in avionics, flight software, or payloads has suffered command loss.

This top-down approach is a crude methodology for fault detection, but allows the spacecraft to continue completing its mission objectives as long as the main subsystems are operational. A more fine grained, component-level method of fault detection may result in more precise and faster subsystem resets instead of a broad and slower system reset; however, component-level fault detection and recovery dramatically increases the complexity of the FDIR system and produces many potential variations in responses, making testing and managing those responses time consuming and expensive, as well as introducing further opportunities for human error. A component fault that does not affect operations is ignored if the spacecraft can still perform its tasks within the allowable thresholds on the three main criteria.

Monitoring of these conditions is configured as a set of telemetered values with an assigned conditional check or set of conditional checks to a persistence threshold as shown in Figure 2. A fault condition is entered once a conditional check returns true for the persistent threshold defined for that fault and once entered, the fault response sequence is executed.

In addition to being triggered by any of these three fault conditions, the FDIR subsystem will also run when MSO performs its routine weekly system reboot. This is a unique capability of the spacecraft, where most spaceborne payloads operate based on a 'run until there's

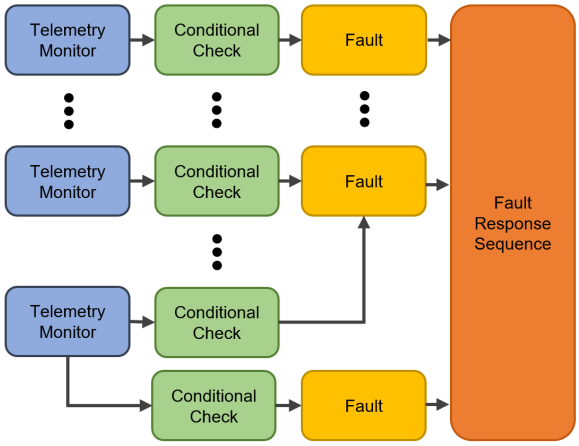

Figure 2: Event-based monitoring architecture

a problem' ideology. The autonomous recovery of the spacecraft through the FDIR subsystem allows MSO to function as robustly as possible, opting for preventative maintenance to continuously keep minor issues under control rather than performing reactive maintenance for an issue that may have gone past the point of repair.

Once triggered by any of the three fault conditions or the weekly reboot, the FDIR subsystem will attempt an orderly shutdown where any write commands will be completed and components not critical for spacecraft operation are turned off. Regardless of whether or not the shutdown was performed cleanly, the Master Power Reset flag on the Master Power Controller will be maintained for a long enough period to allow for bleed off of residual charge to clear any soft latchups. Afterwards, the FDIR system will begin the reboot process and set the spacecraft into one of four following modes:

1. Normal Mode: the system has determined that all fault conditions are above allowable thresholds and will resume the flight operation plan;
2. Safe Mode: the status of fault conditions is undetermined and the spacecraft will focus on staying power positive and command receptive;
3. Emergency Power Mode: the system has determined that power levels are below allowable thresholds and the spacecraft will focus on staying command receptive with thermal and attitude control functions disabled; or
4. Hibernation Mode: the system has determined that the power has drained below the critical level to safely sustain the flight computer and radio communications and will disconnect the battery from primary loads, only autonomously transitioning to Safe Mode when power levels return to an acceptable threshold.

The FDIR subsystem has been designed to be invoked for only one instance at a time, even if there are subsequent faults that follow the original. The FDIR subsystem can only be re-entered once the spacecraft has recovered to a state that has cleared the original fault. This is to ensure that the spacecraft avoids getting stuck in partial check loops and allow watchdog protection against Single Event Effects to be continuous. Additionally, since the fault response to each of the three fault conditions is identical, running through the entire FDIR logic should clear any additional faults anyway. Each time the FDIR fault response has been triggered, the subsystem will take a 'snapshot' of key telemetry and store this information for ground users so that the fault could be investigated at a later time.

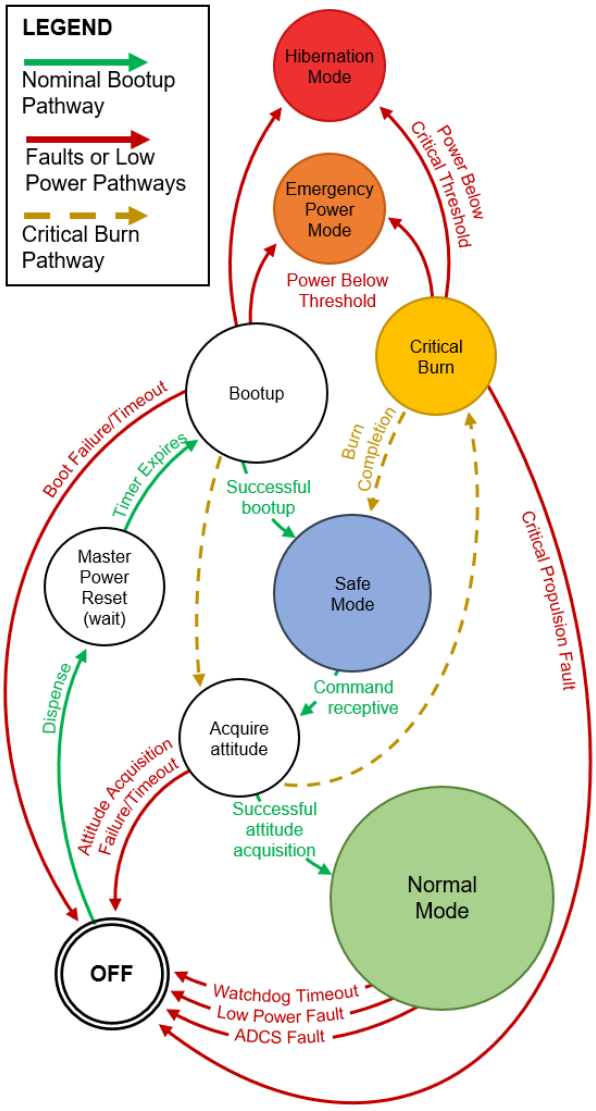

Figure 3: Spacecraft operation modes and entry/exit pathways

In the event that a fault condition has been triggered a certain amount of times within a specified time frame, the fault will then be classified as a persistent fault. The spacecraft will disable autonomous transition into Normal Mode and wait in Safe Mode for ground intervention.

There is one scenario the spacecraft will experience that requires the FDIR subsystem to monitor a specific subset of faults and produce a unique response. That is the event of critical burns, a spacecraft maneuver period that is vital to maintain the trajectory of the spacecraft. There are two executions of these critical burns, the Trans-Mars Injection maneuver that will put the spacecraft on a transfer trajectory to Mars and the Mars Orbit Insertion maneuver that will put the spacecraft into Mars orbit. Critical burns must reach a certain amount of ΔV (change in velocity) within a specified timeframe for maximum efficiency. Therefore, it is crucial that only faults related to maintaining the critical burn are detected and cleared immediately. If the spacecraft is rebooted during the timeframe of a critical burn, the FDIR subsystem will execute an expedited version of the ADCS recovery sequence and then proceed with the critical burn. After the critical burn has completed, the FDIR subsystem will proceed with the remainder of its bootup sequence.

Overall, the FDIR subsystem has been designed to respond to faults in a robust manner, minimize the need for ground user intervention, and authorize on-board autonomy for recovery from Safe Mode. Since, it's impossible to plan for every imaginable possible fault, this design allows the spacecraft to attempt to handle those unpredictable contingencies through a simple system reboot response. Only until faults become persistent does the spacecraft signal for help. The routine weekly resets is also advantageous for the long-term health of the spacecraft so that it can perform its mission objectives as efficiently as possible.

Infrared Imaging

One of the key automation innovations of this mission is moving the majority of the image processing chain on board the spacecraft. This has the primary advantage of minimizing the amount of data that has to be sent back to Earth, resulting in significant savings due to the cost of the Deep Space Network's (DSN) time. Therefore, after initial on-orbit calibration and characterization, only final image products will be routinely sent to Earth.

This also allows the IR instrument to utilize extensive averaging to improve the signal to noise ratio of the final science product. Up to 3600 raw images can be averaged to improve the SNR by a factor of approximately 55. JPEG2000 compression is also employed to allow the

progressively encoded data to be sent to the ground at low resolution shortly after creation, while the large on board storage (2TB) holds the entire mission data at full resolution (final products only). This allows scientists to prioritize which data sets to download at higher resolution and thus investigate interesting phenomena quickly, while allowing the full resolution data to be downloaded much later when Mars and Earth are closer and data rates are highest.

The IR detector must have an "RBS" or "Row-redundant Bolometer" correction applied to remove gross non-uniformities, which are dominated by the image column gain values that would otherwise produce a vertical stripe pattern. The RBS file is not anticipated to change rapidly throughout the mission but can be revised in flight based on in flight calibration if warranted. The correction for the RBS file values is applied in the detector and affects the analog values coming out of the detector.

The other analog correction to non-uniformities is the 2D map of the detector known as the "compensation". This compensation is planned to be derived from a fresh deep space image which should be uniformly cold, although the option to compensate with a view of the "calibration paddle" is also available. This provides a uniformly warm view if on-orbit characterization indicates superior performance. Compensation to the detector will be loaded just prior to each scientific observation (every 15 minutes). Once compensation is loaded, it is applied by the detector to the analog output.

After digitization, the images can be collected and summed over an observation period (currently planned to be up to 30 seconds at 120 Hz for 3600 images). Upon completion of an observation the digital image values are divided by the number of observations to generate an average value.

In addition to collecting digitally averaged observations of the planet, each 15 minutes the instrument will collect deep space and calibration paddle images before the science observation and potentially after as well. These cold and warm uniform images will also be averaged over the same number of observations as the images of the planet. These averaged calibration images are used to apply a two point Non-Uniformity Correction (2-pt NUC) to the averaged science observation which allows not only a final non-uniformity correction to be applied to the detector, but also correct the data to known absolute radiometric values. If on-orbit calibration shows that it is necessary, both pre and post observation 2-pt NUC's can be interpolated and applied to correct for temporal drift as the camera warms up or cools down over time.

During the science observations, additional data on the instrument and spacecraft status are gathered by flight software so they can be associated with the observations in the metadata attached to each image. This metadata includes spacecraft pointing information, position, temperatures of the instruments and spacecraft power levels as well as settings used to create the images such as camera frame rate and number of images averaged.

The final on board process is creating the JPEG2000 image products which include Metadata and make the image components available in a range of resolutions available to download to Earth. These final products are then transferred from the instruments to the flight computer and then stored in non-volatile memory for access by the routine recent image download utility or as high resolution products to be downloaded upon request or at a later date.

The entire process of collecting and processing the IR data is driven at the top level by the flight operations scripts in FlightJAS, which then invoke various utilities to execute the tasks described above. Imaging software is invoked by MAX via a script process called "payload_agent". These programs access raw payload image data in the Unix file system, process raw payload data per ground selected parameters, and move processed image products to the MAX "to_send" downlink directory.

A tool to separate the layers of a JP2 file called "jp2_deconstruct.py" has been developed to allow a JP2 image to be reduced in size and transmitted piecemeal in order to reduce the amount of downlink bandwidth needed. The JP2 pieces can be viewed independently with certain applications and therefore additional image pieces can be downlinked if the initial image passes inspection on the ground and proves interesting.

A script to wait for JP2 files to be created (i.e. wait for a picture to be taken) and take actions upon that file such as deconstructing it is called "jp2_process.sh".

Overall, the IR3C imaging software has to be able to perform the following functions:

- Compensation: Converges all pixel values to a certain target signal value in order to reduce the coarse non-uniformity between the pixels.
- Calibration: Obtain a bias frame correction, dark frame correction, stored as sensor gain & offset information to create a flat-field correction.
- Image averaging: Individual pixels from an arbitrary number of consecutive images are summed on the camera electronics and then averaged before the raw image is sent to the flight computer.

- Compression: Apply lossless JPEG 2000 [18] compression or another lossless progressive compression format.
- Metadata: Attach data to images crucial to understanding the environment in which the images were taken.
- Deconstruction/Construction: Separate JPEG 2000 images into smaller constituent layers for downlink and later on the ground, reconstruct into a lossless image or a lossy portion of an image as needed.

Implementation of these processes are performed through a set of programs that are managed by a Python dictionary to call the appropriate program and feed in the correct parameters.

Program Name	Description
mso_ir3c_cal_flags.py	Set Calibration Flags
mso_ir3c_cals.py	Calibrate Cameras
mso_ir3c_cals.py	Get Calibration Data
mso_ir3c_compensate.py	Compensate Cameras
mso_ir3c_compensate_save.py	Save Compensation
mso_ir3c_compensate_write.py	Write Compensation
mso_ir3c_setups.py	Setup Cameras
ir3c_jp2 mso_ir3c_pics.py	Take JP2 Pictures

Table 1: Python programs used for the IR3C imaging process

These programs work together as follows:

Step One: IR3C camera calibration is performed through the 2-pt NUC process as described earlier in this section.

Step Two: IR3C camera compensation data is saved and written. Compensation data may be written to or read from the BIRD17µ-640 detector on the IR3C camera. The camera may not image during reading or writing of compensation data. Approximately 100 seconds per camera is required to transmit a full compensation image, however, this is only necessary for diagnostics, not routine operations.

Writing compensation image data is performed by:

- Powering up the detector by calling mso_ir3c_setup.py
- Writing the compensation data by calling mso_ir3c_compensate_write.py
- The camera is now ready to capture images.

Prior to reading compensation data it is assumed the camera has been powered up with the detector setup and previously compensated. Reading compensation data from a detector that has not been previously compensated will return undefined / useless image data.

Reading the compensation image data is performed by:

- Reading the compensation data by calling mso_ir3c_compensate_read.py.

Step Three: Before images are acquired, the IR3C cameras go through a few more setup processes.

- First, setup the camera using the mso_ir3c_setups.py command,
- Prepare for compensation using the mso_ir3c_cal_flags.py command to engage the calibration flags,
- Perform compensation by using the mso_ir3c_compensate.py, and then
- Restore to picture taking mode using the mso_ir3c_cal_flags.py command to disengage the calibration flags.

Step Four: A FlightJAS script is invoked to generate an XML file containing the metadata for the IR3C camera that a picture is to be taken with. The FlightJAS script gets the spacecraft clock date/time from MAX, converts it to a string and uses it as an argument to ir3c_metadata_xml.py which creates the .xml file. The rest of the arguments used to create the XML metadata are created by mso_ir3c_metadata_xml.py and added to the argument list. The final XML file generated is placed in the "cooked" imaging file system partition.

Step Five: A picture from the camera is then taken through commanding payload_agent. Raw and JPEG 2000 images are created and placed in the "raw" and "cooked" imaging file system partitions respectively. The jp2_process.sh daemon sees that a new JPEG2000 file has been created and makes a directory based on the JPEG2000 file name in the "cooked" imaging partition. It then moves the JPEG2000 file to that newly created directory, attaches the XML metadata sitting in the cooked partition to it using opj_jpip_addxml, deletes the XML metadata file and then deconstructs the JPEG2000 file in the cooked partition directory using jp2_deconstruct.py.

High-Dynamic Range Visible Imaging

The quality of images acquired in visible light conditions is dependent on two factors: the spatial sampling and the magnitude of brightness differences. Spatial sampling is the size of individual sensors within a detector and how the optics perform in illuminating each sample.

Brightness differences, in other words the contrast or modulation, is dependent on the brightness and range of brightness's in the scene as well as the detector's ability to distinguish details in that range. Scene brightness and the detector's ability to capture that range is called *dynamic range*.

Scene dynamic range depends on the nature of the illumination, the ability of the scene to reflect light, and the optical properties within the volume between the scene and the sensor. Scenes with high contrast (daylight and shadow) have high dynamic range; scenes of low contrast (like the diffuse illumination on an overcast day) have low dynamic range. A principal factor in how well scenes can be captured is the levels of fineness into which the range can be sampled and divided. As long as the subject's dynamic range doesn't exceed the camera's dynamic range, then the image can be resolved with all the details. However, more often than not, the subject's dynamic range is higher than what the camera can handle and details can get lost on either side of the extreme.

For global observations of Mars, there are multiple opportunities for large subject dynamic range during the twilight hours where sunlit features on one side of the planet will be drastically brighter than the dim features on the other side or during eclipse periods that could be dotted with auroras or faint lightning produced by dust storms. These periods are wonderful opportunities to capture valuable scientific data, but the limitations of the camera's dynamic range can miss out on getting every detail. A longer exposure would allow more light to reach the camera's sensor to resolve details in the shadow, but the harsh highlights would max out the photocells of the camera's sensor and lose details in the brighter portions of the image. A shorter exposure would limit the amount of light so that the sensor's photocells are not maxed out and details in the highlights can be resolved, but then details in the shadows would be lost. Any exposure in between would lose details from both light and shadow.

MSO addresses the inherent limitations of dynamic range in two ways: it uses a 12-bit analog-to-digital converter (ADC) to encode the dynamic range of the detector and exposure control using an automation software to perform on-board processing that will construct high-dynamic range (HDR) images from a set of low-dynamic range source images for downlink. The process begins with taking an image at a baseline exposure between the light extremes of highlights and shadow. Then 1 or 2 additional images above and below the baseline exposure (for a total of 3 or 5 source images) will be taken to capture image data for details in the highlights and shadows. Finally, the source images will be combined into one with all details, as if it was captured by a camera with high-dynamic range. Unfortunately, MSO does not have the space in its downlink budget to send the 3 or 5 source images for this process to be done on the ground. Therefore, the visible imaging on MSO will be performed on-board by an automation software that will reduce the amount of data to be downlinked while still preserving radiometric information for scientific analysis.

There are multiple challenges that come with this approach including: finding the correct exposure and exposure stops to capture as many details as possible, ensuring that noise and loss factors do not obscure those details, merging each exposure in a way that displays those details without destroying the data about the actual amount of light the imager has measured, and ensuring that the final product is a manageable size for downlink; all to be done autonomously by the spacecraft.

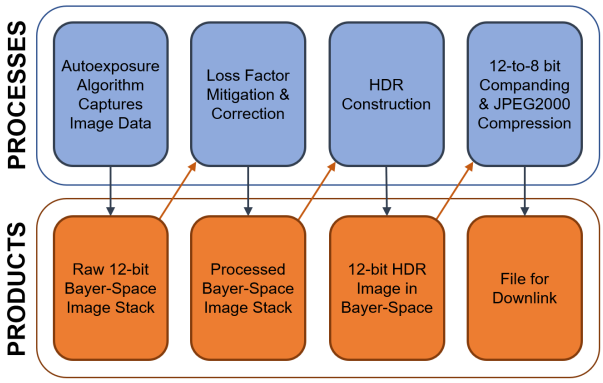

Figure 4: Flowchart of processes and their products that will be covered in this section

The first challenge is finding the correct exposures. The general approach is to determine a "seed" exposure time and then determine the range necessary for HDR. The process can be done in two ways: 1) with a pre-determined method by the operations team who will estimate the exposure from the basic physics of the scene illumination and viewing conditions, and then iterating with ground-in-the-loop or, 2) by using an on-board auto-exposure algorithm.

The first method involves some initial guesswork and iteration. MSO would take a single image for downlink at an exposure setting ground operators have determined would be the most likely to be suitable for a particular time of day. Ground operators would then analyze the image and determine whether a different exposure setting should be used in the next round of imaging. It may take a few rounds of this process to pinpoint an acceptable baseline exposure for each size of subject dynamic range. The empirical method would take some time and hands-on effort, but would produce images that

are acceptable and verified by ground operators. Once the dataset of baseline exposures for different times of day on Mars has been established, the iterative process begins again for the additional exposure stops above and below the baseline. Ground operators can begin with one and two exposure stops and increase as necessary until all light and dark details have been resolved.

The second method proceeds in a similar fashion as the first method where a baseline exposure must be determined first, but this initial step can be performed autonomously. Autoexposure algorithms have already been developed and used for deep space applications such as the Imager for Mars Pathfinder and imagers on the Mars Exploration Rover. These prior algorithms form the basis of the algorithm used on MSO. This algorithm involves viewing the image's histogram of pixel values or Digital Number (DN). There are four parameters to the algorithm: a lower and upper target DN threshold, an allowable percentage of pixels to exceed either DN threshold (pixel fraction), and a termination value. The measured upper and lower DN thresholds are calculated by counting the number of pixels on the right and left sides of the histogram respectively, until the number of pixels equals the commanded pixel fraction. If the difference between both DN thresholds from their local maxima (e.g. for an 8-bit image maxima are 0 for lower and 255 for upper) are within the termination value, then the algorithm terminates and the image is accepted as the baseline exposure. Otherwise, the process iterates by generating a new exposure time (t_{new}) by multiplying the current exposure time ($t_{current}$) by the ratio of the lower to upper DN threshold.

$$t_{new} = t_{current} \times \frac{DN(lower)}{DN(max) - DN(upper)} \quad [1]$$

This autoexposure algorithm was designed to center the histogram as much as possible so that low- and high-contrast images could be accommodated and the baseline exposure would only capture the midtones of the subject.

As for the remaining under- and over-exposure times, the algorithm repeats similarly except the right side of the histogram for the upper DN threshold and left side of the histogram for the lower DN threshold are evaluated separately. Different pixel fractions are defined for the final two exposures on either side of the baseline. The method for calculating these exposure times is identical to that used on the imagers on MER.

$$t_{new} = t_{current} \times \frac{DN(desired)}{DN(measured)} \quad [2]$$

Both approaches to determining the seed exposure and variances around it will produce a dataset of baseline exposures and appropriate exposure steps above and below the baseline at different times of the day on Mars. This dataset of exposure settings can be stored onboard the spacecraft and used for future reference for the remainder of the mission. A number of pathologic cases, where the histogram has many modes, or is heavily skewed, often require the use of a combination of ground-in-the-loop and autonomous methods to ensure that acceptable images are produced for the next step in the HDR process.

The next challenge is to ensure that noise and loss factors are properly mitigated before the HDR construction

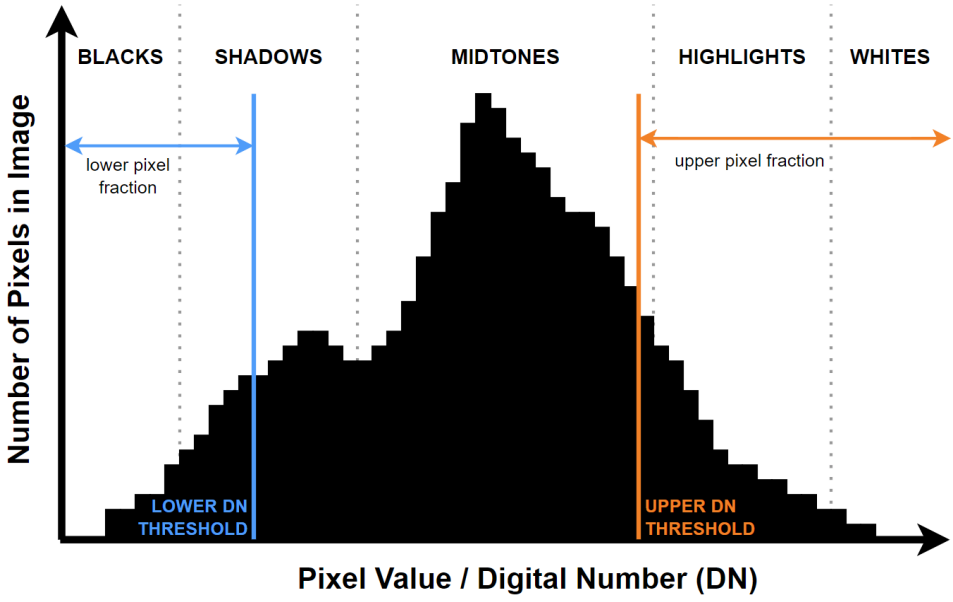

Figure 5: Pictorial representation of the MSO autoexposure algorithm

process since errors would propagate through to the final product. Specific types of loss factors are related to the type of sensor technology used for an imager as well as the image acquisition mode. The camera imaging system used on MSO makes use of a Complementary Metal Oxide Semiconductor (CMOS) digital image sensor and images are taken using a Global Reset Release mode (the digital equivalent to a mechanical global shutter). Four mitigation/correction processes emerge from this configuration: dark current compensation, parasitic light sensitivity (PLS) mitigation, flat-field or non-uniformity correction, and bad pixel mitigation.

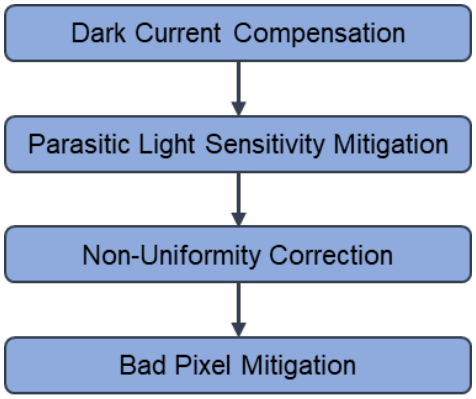

Figure 6: Mitigation and correction processes used to reduce errors before HDR construction process

Dark current is the accumulation of thermal energy within an imager's sensor. The sensor cannot differentiate between these thermal electrons and the photoelectrons generated by light (i.e. the data that is actually desired), resulting in the undesirable "contamination" of the signal. Therefore, dark current must be compensated. Compensation for dark current is a complex issue since it is dependent on both temperature and exposure times. A straightforward method would be to take a reference image at a constant temperature and exposure with a closed shutter and subtract this reference from the image; however, CMOS sensors do not have a mechanical shutter to make use of this method and many spacecraft cameras avoid mechanical shutters owing to their mass, complexity, and cost.

An alternative method to perform the dark current compensation (developed by Abarca and Theuwissen), uses the dual functionality of the CMOS pixel architecture to generate an artificial dark current reference frame. This method leverages use of the CMOS imager pixel as a local temperature sensor, enabling the collection of thermal distribution data which eliminates the temperature dependency and leaves the exposure time to be the only variable. This is particularly useful since multiple exposure times will be used for a single HDR image. Implementation of this method would be to take a source image of Mars at one exposure and then immediately take a temperature image to create the artificial dark current frame at the same exposure time. This will proceed for the remaining exposures, essentially creating a pair of source and temperature images for every exposure.

A second and simpler, but less accurate method, is a two-step process that recognizes the fact that deep space is not photoactive in the general sensitivity range of CMOS detectors. After performing a planet centered imaging pass and collecting all the source images, the spacecraft will begin to slew towards an Earth-centered attitude for its downlink pass. During this slew, calibration images at each exposure will be taken of deep space which will function in a similar manner as a 'mechanical closed shutter'. Comparison of these deep space calibration images with the deep space portion of the image around Mars' disk will yield an approximate value of the dark current generation. This is only approximate because there may be a temperature difference between the calibration and source images.

Use of either method is dependent on the accuracy desired as well as the duration of the imaging passes. Short imaging passes will not be able to utilize the first method since imaging time is doubled.

Parasitic light sensitivity is a phenomenon unique to CMOS sensors using a global shutter mechanism. In order to separate the exposure phase from the readout phase in the sensor, an in-pixel light-shielded storage node is used; however, the information in the storage node is corrupted by further incoming light between the end of exposure and the start of readout. Pixels affected by PLS appear as an after-glow on one side of the final image. The amount of influence PLS will have on an imaging system is fixed by the sensor and cannot be corrected, but there are four factors that can be considered to mitigate its effects. Those factors are: exposure time, sensor readout time, wavelengths of the light source, and the angle of incident light. PLS effects appear to decrease with longer exposure times because their relative contribution in the time-domain of the readout phase decreases, i.e. already high DN values won't visually appear much brighter compared to low DN values. Conversely, shorter sensor readout times also decrease PLS effects since there is less time between exposure and readout for the parasitic light to affect the storage nodes. Light source wavelengths and angle of incident cannot be changed since MSO is designed to image Mars at the same angle, but it is useful to know that PLS effects decrease at shorter wavelengths and shallower angles of incidences. The considerations of

these factors are vital during the construction step of the HDR process.

Another limitation of any imaging system is the uniformity of its brightness response as a function of position within its field of view. The use of color filter arrays closely adjacent to the photosites as well as the color and brightness attributes of the optics create variations of the camera's responses that can be corrected. This correction is easily done by dividing the source image by another corrected image of uniform scene to increase brightness in areas of lower response.

The final process is bad pixel mitigation. Defects in manufacturing, contamination of the sensor, and radiation damage can affect the uniformity in the sensitivity of individual pixel photosites leading to single pixels that are more 'hot' (bright or sensitive) than their neighboring pixels or more 'cold' (dark or less sensitive). Manufacturing defects are mapped out and supplied by the sensor manufacturer. Sensor contamination is mapped out during calibration. Radiation damage occurs throughout the life of the sensor and would be mapped out as they are found throughout the course of its lifecycle. Bad pixel mitigation is performed by replacing the DN value with an average value of its neighboring pixels within the same color channel. Since these defects are often only a single or a few pixels in size, this simple averaging done by the algorithm is sufficient for MSO's purposes.

With the baseline, over-, and under-exposed images scrubbed of as many noise and loss factors as possible, the autonomous HDR processing can now compile the exposures into a single image. Challenges in this part of the process are characterized by noise factors, motion artifacts, and ensuring that radiometric data from each exposure is not lost during this compiling process. Since each exposure is taken at different times, there will inevitably be movement between images. That movement can come from cloud movement over the surface of Mars or the small jitter from the spacecraft itself stemming from the errors in attitude control during Mars tracking. These slight differences between images can appear in the final image as 'ghosts' when the movement is large or blurring when the movement is small. The final image also must be able to track which pixels came from which exposure and appropriately scale the value of each pixel to form a coherent image while still preserving the radiometric data.

The impact of these issues can be reduced by a combination of two methods. The first is proposed by Kang et al that utilizes an adaptive weighting function so that noise and local motions contribute less to the final image. This method can also be leveraged to increase contribution from images with longer exposure times or shorter readout times to reduce the effect of PLS. An added benefit of this method is that it performs the HDR image construction in Bayer space instead of full RGB. This allows the HDR process to use 12-bit raw data from the camera system and decreases the processing power required from the flight computer. The second method (proposed by Sen et al) utilizes a patch-based system for constructing the HDR image which is particularly effective with aligning all of the exposures and extracting accurate radiometric data during the final merge. As the range of DNs created in the final merged image can exceed the initial 12-bit encoding, these images will likely be 16-bit encoded.

Notation	Description
S	Source LDR images
S_N	LDR image of +/- exposure stops
R	Reference source LDR image
I^N	Adjusted source LDR images to N^{th} exposure stop; created using $g^n(S)$ to map an I^{th} image to N^{th} exposure
K_N	Adjusted reference source LDR image to N^{th} exposure stop; initially created using $g^n(R)$ and iteratively optimized using $I^k(H)$
L_K	Reconstructed K image from an iterative optimization cycle
H	Final HDR constructed image using an adaptive weighting function $w(L_k)$

Table 2: Notations used in Figure 7

With the HDR image constructed in Bayer space, the autonomous process will then prepare the image for downlink. It is important to ensure that the method for reducing or compressing the data volume for downlink can recover all information after restoring or decompressing the data. This is defined as lossless (when all data can be recovered) or lossy (when some data is permanently lost) compression. The methods employed for this are a 12-to-8 bit compression table-lookup process or upon receipt an 8-to 12 bit decompression table (the inverse of the compression table), both of which are somewhat lossy and the JPEG2000 compression algorithm (that can be either lossy or lossless, depending on the amount of compression commanded. The compression/expanding (contracted to companding) process is an optional step that is used to further reduce data volume by limiting the effects of photon statistical or shot noise in the image. Since the data volume has already been optimized by using a Bayer space image and JPEG2000 compression, the

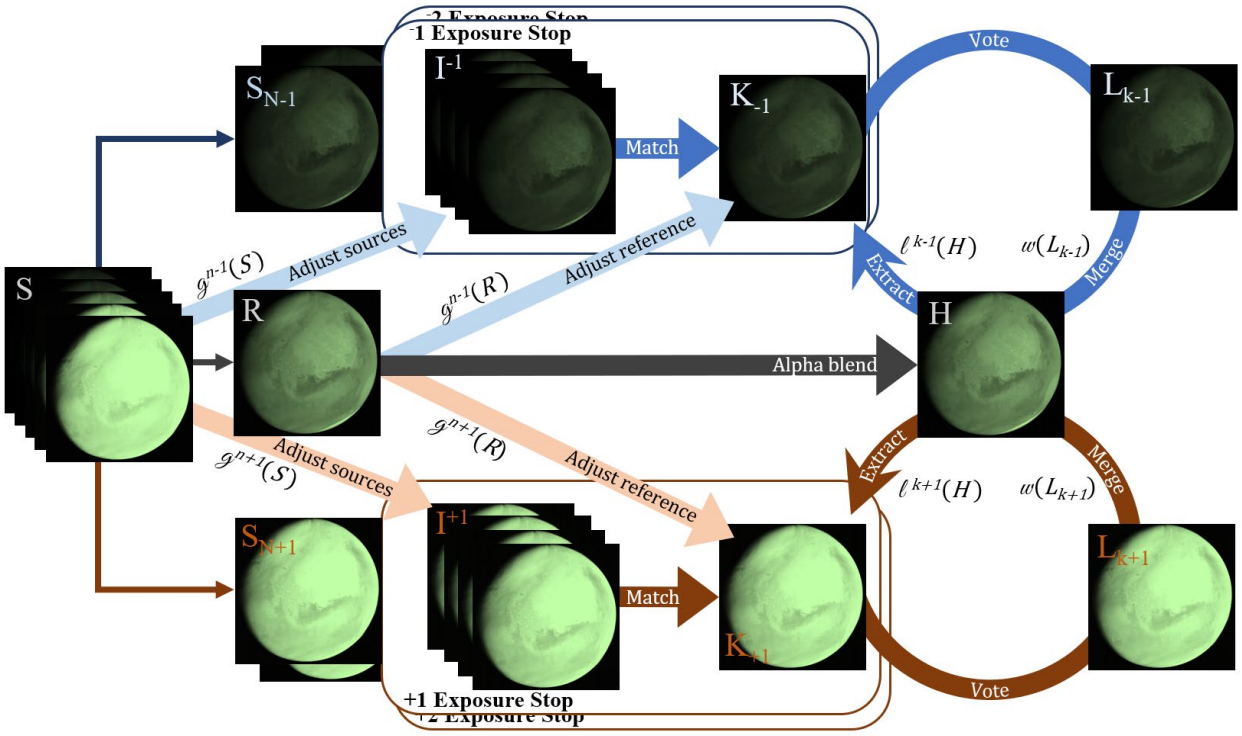

Figure 7: HDR construction process

companding process would only be implemented if downlink rates are below nominal expectations or if the NASA Planetary Data System has an objection to the use of JPEG2000 and insists on another image format.

This concludes the entire autonomous visible imaging process that occurs on-board the spacecraft. Additional steps are performed during ground processing, but are not the subject of this paper. This process will be able to produce high-dynamic range images that can resolve faint phenomena in the dark hemisphere and bright sunlit features on the day side that are sufficient for science analysis. Any of the steps in this process can be omitted should science teams request anything between single exposure, unprocessed, raw image data or the full HDR, corrected, processed image data. This makes the visible imaging system on MSO robust in its delivery.

4 CONCLUSION

This paper has discussed the automation mindset the MMS group has taken in the development of MSO that led to optimized workflows that allowed the small team to perform the plethora of tasks required for such development. The time savings provided by these optimized workflows allowed the team to focus on more complicated problems that ultimately culminated in MSO adopting the same automated operation ideology. The automation procedures on MSO solved problems surrounding deep space operations and limited downlink budgets which helped make the mission feasible in the first place.

Automation has taken the brunt of the brute work required for spacecraft development and is a path forward for small teams with restricted budgets to enter the playing field. It is also a pathway for spacecraft operating in deep space to have more resilient responses to inevitable contingencies and more diversity in its delivery of mission objectives. It is the hope of the MMS group that future spacecraft take advantage of similar highly automated and autonomous solutions to continue exploring and go further into the last frontier.

References

A. Abarca & A. Theuwissen (2023). "A CMOS Image Sensor Dark Current Compensation Using In-Pixel Temperature Sensors." Proceedings of the 2023 International Image Sensor Workshop.

G. Meynants et al. Backside illuminated global shutter CMOS image sensors.

H. Kang, S. Lee, K. Song, M. Kang (2014). "Bayer patterned high dynamic range image reconstruction using adaptive weighting function." Eurasip Journal on Advances in Signal Processing 10.1186

J. N. Maki et al. (2003) "Mars Exploration Rover Engineering Cameras" Journal of Geophysical Research: Planets 108(E12).

J. Warren & M. Malin (1995) Shot Noise Limiting Companding for MSI. https://www.msss.com/http/near_cal/fs_algorithms/companding/companding.html

P. H. Smith et al. (1997) "Results from the Mars Pathfinder Camera" Science 278(5344), 1758-1765.

P. Sen et al. (2012) "Robust Patch-Based HDR Reconstruction of Dynamic Scenes." ACM Transactions on Graphics 31(6), 203:1-203:11