RP-check: An Architecture for Spaceflight Command Sequence Validation

Mark W. Maimone, Scott Maxwell, Jeffrey J. Biesiadecki, Stirling Algermissen

Jet Propulsion Laboratory
California Institute of Technology
Pasadena, CA USA
Mark.Maimone@jpl.nasa.gov

***Abstract*—NASA Mars Rover operations are planned anew every day, in contrast to traditional deep space missions where operations can be planned weeks or months in advance. Verification of the rovers' spacecraft commands must be performed quickly, implemented robustly, and must consider multiple possible outcomes. In this paper we discuss the architecture behind one component of our system, a software tool named RP-check that evaluates command sequences created by the Rover Planner (RP) team. RP-check is one component of the Rover Sequencing and Visualization Program (RSVP), our primary tool for commanding and validating the mobility, arm, and turret subsystems on the rovers.**

RP-check is the spacecraft sequencing equivalent of programming language analyzers like *lint* [6]. It automates the analysis of sequences of commands, evaluating their conformance to best practices described by mission-spanning Flight Rules and Rover Planner team-specific recommendations. It checks hundreds of rules and typically completes its assessment in less than 30 seconds, enabling Rover Planners to continuously validate and refine their command sequences many times a day while planning activities.

RP-check owes its success to a simple framework that enables it run quickly and makes it easy to add new rules. Built and maintained by members of the Rover Planner team, new rules and test cases can be validated and deployed quickly to ensure that any issues of concern that arise during operations will be addressed in all future plans.

TABLE OF CONTENTS

1. INTRODUCTION

Every day offers a new experience for NASA's Mars Rovers Curiosity and Opportunity. Human operators plan each whole day of rover activities by creating new sequences of commands that must be reviewed, verified and possibly corrected before being uplinked to the spacecraft. A complete description of the Mars Science Laboratory (MSL) operations planning model is beyond the scope of this paper, see [4], [10], [1], [8] for more details. In short, on a given planning day the mobility, arm and turret plans spanning one or more days are written into command sequences and reviewed tactically by the Rover Planner team (two or more people) using the Rover Sequencing and Visualization Program (RSVP, [19]). Initial versions of their sequences are verified throughout the day by multiple software tools, and the final versions are also reviewed by the whole operations team (approximately twenty people).

Most commands sent by Rover Planners are event-driven (i.e., each activity must complete before the next will begin), but the spacecraft also supports time-constrained commanding. All commands sent to Mars are either simulated or checked by hand against a database of Flight Rules. These are constraints on mission operations that are typically levied either as a result of lessons learned while creating the spacecraft, or from problems or idiosyncrasies discovered during mission operations. Flight rules are expressed in English prose, and the operations team enforces them daily either by manual review or via automated checkers like RP-check or those listed in the next section. The flight rule database is not static; the nature of extended flight operations is that flight rules are created or re-assessed fairly frequently. RP-check is a rule-based checker, and was designed to support easy incorporation of new and updated flight rules.

Other MSL Validators

SeqGen is the command validator that checks every MSL sequence planned for execution on the rover [11]. As used on MSL, it evaluates a subset of flight rules on all uplinked commands and produces reports of known or possible rule violations. All such potential violations must be resolved manually before commands are authorized for use in flight. Some known limitations of SeqGen are that it does not model off-nominal branches (e.g. "else" commands are typically not evaluated), does not model most mechanisms explicitly, can only track a limited number of parameters, and can only apply Flight Rules to a single sequence at a time. As currently implemented, many false positive warnings are generated, it takes tens of minutes to assess command sequences on a busy day, and the duration of the process required to update its software database by adding or modifying rules can be weeks or longer.

The Surface Flight Software Simulation (SSIM) component of RSVP performs a high-fidelity simulation of the nominal rover mobility, arm and turret command sequences. It uses detailed 3D meshes generated from stereo imagery collected by the rover to inform its position and attitude estimates during its simulation, and it runs much of the same Flight

This work was performed while Scott Maxwell was an employee of JPL, Caltech.
978-1-5386-2014-4/18/$31.00

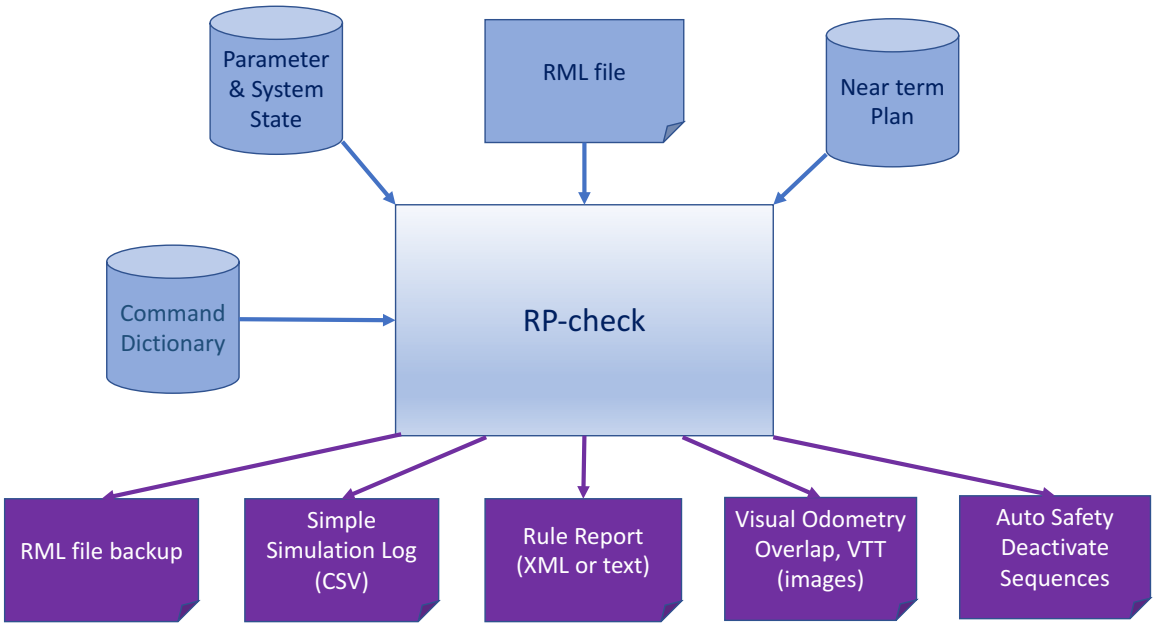

Figure 1. Basic workflow of data passed through RP-check

Software that controls the actual rover. SSIM also enables sensitivity analysis of arm and drive operations by expanding safety zone boundaries using simulation "halos", and allows varying the nominal drive behavior by changing the amount of slip predicted for the terrain. But although SSIM provides a very high fidelity prediction of the execution of a single plan and its duration, it cannot evaluate how well the plan adheres to flight rule constraints, nor does it assess commands in off-nominal branches (e.g., "else" clauses that never execute).

Related Work

The need to quickly check spacecraft commands with a lint-like tool extends beyond RSVP-enabled projects like the Mars Exploration Rover (MER), Phoenix, and MSL missions. Cassini's Imaging Science Subsystem (ISS) and Visual and Infrared Mapping Spectrometer (VIMS) teams used a similar tool called the Remote Constraint Checker to validate sequences [3]. During operations this enabled instrument teams to receive early feedback when they submitted problematic parameters in their commands, and ultimately enabled them to deliver error-free sequences. In contrast to RP-check, Cassini's Remote Constraint Check uses a client-server based model where a remote user can submit uplink commands and parameters to a server for validation. This was due to the ISS and VIMS teams being remote to JPL. Both Cassini's Remote Constraint Checker and RP-check are used in addition to SeqGen, which also remotely validates sequences.

[12] abstracts over many approaches to model-based checking of autonomous systems, giving some examples from the Remote Agent controller for the Deep Space One mission. [15] provides more details about Remote Agent's validation of the Planner, Executive and other component plans using first-order predicate logic to confirm temporal and other constraints, and a Deep Space One uplink constraint checker that validated flight rule conformance. Unlike RP-check which focuses primarily on event-driven command transitions, [13] describe the language and capabilities of a TOPEX/Poseidon validator designed to use timed linear logic constraints.

2. General Architecture

The Rover Planner sequence and Flight Rule checker (RP-check) assesses how well the planned sequence of commands obeys relevant flight rules. It has access to different knowledge about the current state of the vehicle than the SeqGen validator. For instance, RP-check tracks the current values of tens of thousands of parameters stored in Non-volatile Parameter Memory (NPM), many of the 41 motor mechanisms on the rover, and reads the MSlice tool's current high level mission operations plan outlining the general timing of all activities being scheduled [14]. See Figure 1 for the overall workflow. RP-check does not model any knowledge about the terrain or surrounding imagery; it does model rover pose and motor positions, but only using a simplified flat-world model, nothing like the detailed simulation performed by SSIM. In practice these have not been significant limitations however, because Flight Rules are typically written to describe inter-dependencies between commands, not specific terrain-related behaviors.

RP-check owes its success to a simple framework that enables it to run quickly and makes it easy to add new rules. It generates a single execution trace though all commands, expanding any nested sequence calls into a serialized list of commands. Certain configuration changes like pose estimates that result from driving and arm operations are modeled once, and their results are cached and made available for later analysis by multiple rules (see Table 1 for a complete list).

Adding a new rule is as simple as determining what parts of the flight system are relevant to the rule, modeling them in variables, and writing code to step through the sequence trace and evaluate only those commands and arguments relevant to the new rule.

RP-check's speed was explicitly one of its design goals: a tool that completes its analysis in only 30 seconds is considerably more useful – and more likely to be used frequently during the planning day – than one that takes ten minutes or more. The goal was to make RP-check something that could easily be invoked when the Rover Planners had even a few spare moments, so that they could catch and fix errors early and often. Considerable engineering effort therefore went into making the tool fast.

RP-check accepts as input a single Rover Markup Language (RML) file containing one of more command sequences that are planned to execute across one or more Martian Solar Days (sols). RML is a dialect of XML developed for spacecraft commanding. It was developed for the MER mission and later reused and adapted for the Phoenix lander and MSL missions. Using an XML-based language was very helpful in developing RP-check and its predecessor, FR-check, as XML parsers are readily available for nearly all computer languages; this was a helpful leg up in getting both scripts up and running.

RP-check presumes that the RML file is syntactically valid, a property which is ensured by the Rover Sequence Editor (RoSE) component of RSVP. RP-check constructs an execution trace of the RP sequences starting with the main sequence of the file, typically the DEFAULT sequence. It then serializes all the included commands, recursively expanding out any invocations of sub-sequences into a single full list of commands. Obscure cases such as sequences that recursively invoke each other (so called "ping-pong" sequences) are detected and cut off at the first level of nesting, to prevent an infinite command expansion.

RP-check achieves 100% code coverage over command sequences by making an unusual simplifying assumption. When a conditional is found in a sequence, it is treated simply as another command, not as place where the control flow of sequence execution sequence might change. That has the side effect of causing *both* branches of each conditional to be evaluated. While that is not a realistic representation of how the sequence will be run, it does allow RP-check to review *all* sequenced commands, not just those in the nominal path of execution. An ideal validator would consider all possible control flows through a sequence, honoring conditional branches and evaluating only those commands relevant to each possible outcome. However, the number of possible paths is exponential in the number of conditionals and Rover Planner sequences typically involve many conditionals, so that was not a generally viable option. This simplifying assumption has worked so well that only one RP-check rule has required a more nuanced understanding of control flow to achieve its purpose.[2] This is in contrast to SeqGen and SSIM, both of which follow a single branch at a time (often via "magic comments" as explained below).

[2]There was one rule whose sequencing framework *required* explicit modeling of branches, so the capability to model multiple control paths was added. But it only models conditionals where one of its branches explicitly commands termination of the current sequence. That simplifying assumption reduced millions of different traces down to a more manageable tens of thousands of traces, enabling the overall assessment to complete in a reasonable amount of time.

One key to RP-check's success was its rule writers' diligent effort to eliminate false positives. In contrast to tools such as SeqGen, which assume that their user has weeks to respond and can therefore afford to over-warn, RP-check is used in an environment where its users might be able to spare only minutes to process its output – if that. It was therefore critical to ensure that its output was tightly focused; if it issues a warning, its user can safely assume the warning is worthwhile. This avoided the possibility of accidentally training the team to ignore its output, and saves them from having to tediously search for the bits of wheat hidden among the chaff, a time sink they certainly cannot afford.

RP-check also allows sequence authors to disable certain warnings by adding "magic comments" to commands. This capability allows the sequence author to eliminate false positive RP-check warnings, while at the same time preserving a visible affirmation of their existence directly in the RML file. Because Rover Planner sequences are reviewed several times prior to uplink, the comments provide context to reviewers that a warning was issued, and gives them an opportunity to ask the author about it.

Other magic comments serve to explain complex or obscure command arguments. For instance, given a command argument expressed as `0.785 radians`, RP-check will look for a comment such as `angle=45/deg`. Whenever it finds such a comment, it will actively ensure that the command argument matches the comment's value after accounting for unit conversion. Similar annotations are used to document arm target names and locations, command durations, expected distance traveled, assumed TRUE/FALSE values for conditionals (used by SeqGen and/or SSIM), steering and driving angles, arbitrary parameter values, command sequence names, sol numbers, and memory offsets not documented explicitly in the Command Dictionary. Any inconsistencies between command arguments and the explanations given in magic comments will result in a warning message; a typo in the name of the magic comment also produces a warning, thus averting any incorrect expectation that a mistyped rule is being checked. Magic comments also help enforce sequencing author expectations, expressing constraints on parameters as described later in this section.

RP-check offers its output in two formats, selectable when the script is run: as plain (though optionally color-coded) text output directly to the terminal, or as XML. This latter, machine-friendly format is useful when running RP-check as a child process of the RSVP sequence editor, RSVP-RoSE: RSVP-RoSE can open a modeless dialog box showing the interpreted output of RP-check, which it runs automatically and asynchronously as soon as the user makes any change to the sequence (see Figure 2). When the user has been idle long enough, RSVP-RoSE "catches up" and displays RP-check's latest output to the user automatically. References to commands are translated to hyperlinks; when the user clicks a link, RSVP-RoSE highlights and scrolls to the corresponding command. The intent of this feature is both to encourage running RP-check and to make it even easier to interpret and act on its warnings.

Early on, RP-check was officially integrated into the script Rover Planners run to deliver their final sequences in preparation for uplink; it is therefore effectively mandatory to run it at least once during the planning day. Results of this final verification step are automatically posted to the planning page and are available for team review.

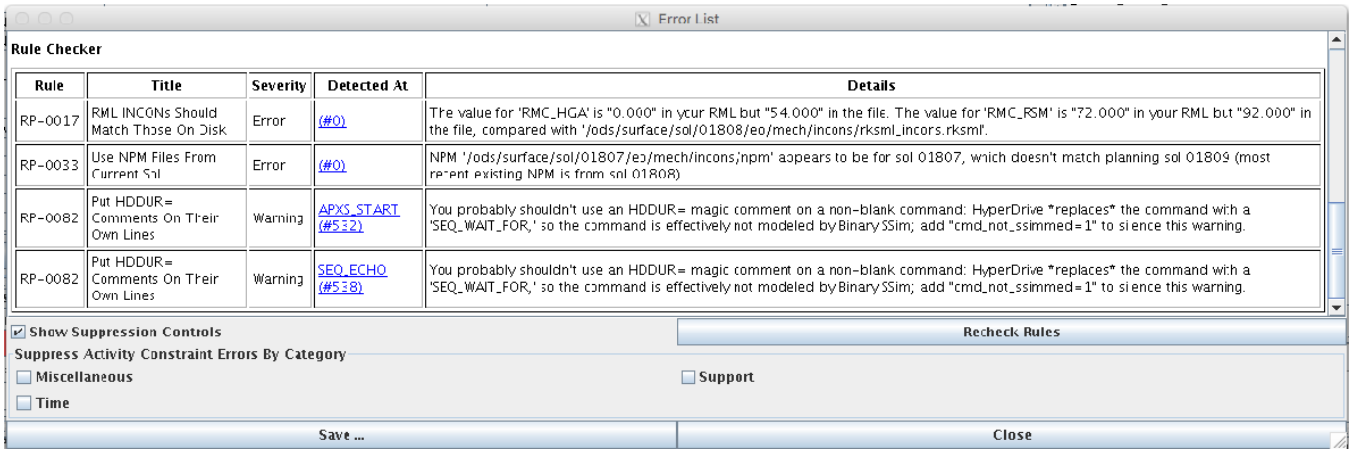

Figure 2. Example of RP-check's XML output as rendered by the RSVP/RoSE GUI

RP-check also serves as a valuable self-training device. When new Rover Planners trainees construct their own sequences at their own pace, RP-check provides immediate feedback on how well they are doing without requiring any homework grading turnaround time.

Sophistication of Models

RP-check does not typically employ a very detailed model of spacecraft instruments, nor of the world around it. It uses just as much detail as is needed to implement individual rules.

Consider the drill instrument. Although RP-check maintains a model of the rover's drill feed motor position, it cannot predict when the drill bit will impact the terrain. So any commands that are expected to impact the terrain result in the current feed position being marked "unknown". Position knowledge is restored by any command that moves it to a fixed location, e.g. a drill stow command. Although that model seems quite coarse and less detailed than what SSIM can provide, it has proven sufficient to validate and find errors in even complex drill test sequences.

For mobility, RP-check has no knowledge of the shape of the 3D terrain surrounding the rover. Instead, it relies on a flat-world assumption, using only X,Y and yaw to model what should be in reality a full 6 degree-of-freedom rover position and attitude. Yet even that simple model is sufficient to express many of the constraints on driving, especially when supplemented by "magic comments" expressing the expected distance driven.

RP-check is not a tool for resolving fine positioning details due to varying amounts of slip; SSIM does that job very well. But RP-check *does* verify that when drives are planned in high-slip environments (determined by checking the state of certain mobility parameters), command sequences will include enough margin to account for the maximum amount of slip being predicted. E.g., if 80% slip is expected during a drive, then the sequence must either include five times as many primitive ARC commands (compared to no-slip), or must allow five times the duration of higher level GO_TO commands that drive toward a specific location on the terrain.

Preprocessing Annotations

Certain computations are known to be needed for more than one rule. For instance, estimating the rover's position and orientation at each command is needed by all rules that take distance traveled into account. So a number of quantities are all precomputed prior to evaluating any individual rule. Table 1 shows the pre-annotated values provided by RP-check. Having all these data readily available often simplifies the construction of new rules, helping preserve our ability to turn around new rules quickly.

Typical Rule Framework

A typical rule is implemented as a subroutine call. Rules are written directly in a standard scripting language, not in an abstracted mini-language, to provide the most flexibility in implementation. Each rule is passed a standard set of inputs, including the expanded and serialized trace of all commands in the RML file more-or-less in order (the exceptions being the non-modeling of conditionals, and the breaking up of recursive calls).

Each rule will typically loop independently through all the planned commands, and will often choose to ignore most of them. Only those commands relevant to the rule at hand will be explicitly intercepted and dealt with. This strategy allows multiple developers to work independently, as each rule functions independently of the others. It also enables RP-check to run even faster by distributing its rule checks across multiple cores or CPUs in parallel.

Flight rule needs are encoded as succinctly as possible. Most of the rules are implemented using nothing more than short perl functions.

But it is important to provide enough capability to implement complex rules as well. For example, the rover's Visual Odometry capability [9], [5] enables it to measure unexpected pose changes (e.g., translational slip) by comparing pairs of stereo images. But those images must overlap; so RP-check provides a rule that tracks the pointing direction parameters, and calls out to several helper functions to render a ray-traced scene illustrating the overlap between viewing angles. A warning is issued and a report is created if adjacent images do not overlap by a minimum percentage (see Figure 3). As another example, when commanding Visual Target Tracking (VTT) to track targets across images acquired between drive steps, RP-check also generates a report documenting the tracked feature's location, as in Figure 4. The flexibility that enables experts to encode the right amount of detail for each rule is critical.

Table 1. Annotations that are cached and available for use within each rule.

Field	Description
Arm	
ARM_IN_CONTACT	Boolean true when the arm is expected to be in contact with terrain
ARM_OVERDRIVE	Amount of overdrive used during arm placement (meters)
ARM_TARGET_DIST	Distance to/beyond the current Arm Target (meters)
ARM_TOOL	Current Arm Tool
ARM_TOOL_DESC	Description of command that most recently set the Arm Tool
MAHLI Imager	
MAHLI_OPEN	Boolean true when MAHLI cover is open
MAHLI_OPEN_DESC	Description of command that opened MAHLI
MAHLI_OPEN_ELT	Command that opened MAHLI
DURATION	Coarse estimate of duration of this command (details only for imaging)
ACCUMULATED_DURATION	Coarse estimate of overall command durations
Drill	
BIT_IN_ROCK_ELT	Command expected to have placed the drill in contact
DRILL_STOW_DESC	Description of command that stowed the drill
DRILL_FEED_POS_MM	Drill Feed location post-command (millimeters)
DRILL_FEED_POS_MM_WAS	Drill Feed location pre-command (millimeters)
PRELOAD_N	Applied Preload Force (Newtons)
PRELOAD_PRELOADED	Boolean true when this command preloaded the arm
PRELOAD_UNLOADED	Boolean true when this command unloaded the arm
BATTLE_SHORT_DESC	Description of command that enabled Battle Short
PERCUSS_OR_RAMP	Boolean true when this command runs Percussion
Dust Removal Tool	
DRT_ACTIVE_IN_THIS_CMD_ELT	Boolean true when DRT active
DRT_RUNNING_ELT	Command that turned the DRT on
DRT_TURNED_ON_HERE	Boolean true when this command turns the DRT on
Inlet Covers	
IIC_OPEN	Mapping from Inlet Cover name to Description of Open command (if open)
Basic Driving	
SIMPLE_SIM_DELTA_H	Heading change expected from this command (radians)
SIMPLE_SIM_DIST	Distance driven by this command (meters)
SIMPLE_SIM_START_H	Pre-command Initial Heading (radians)
SIMPLE_SIM_START_X	Pre-command Initial Site Frame X position (meters)
SIMPLE_SIM_START_Y	Pre-command Initial Site Frame Y position (meters)
SIMPLE_SIM_START_VO_AZ	Pre-command Visual Odometry Azimuth angle (RNAV radians)
SIMPLE_SIM_START_VO_EL	Pre-command Visual Odometry Elevation angle (RNAV radians)
SIMPLE_SIM_H	Post-command Current Heading (radians)
SIMPLE_SIM_X	Post-command Current Site Frame X position (meters)
SIMPLE_SIM_Y	Post-command Current Site Frame Y position (meters)
SIMPLE_SIM_VO_AZ	Post-command Visual Odometry Azimuth angle (RNAV radians)
SIMPLE_SIM_VO_EL	Post-command Visual Odometry Elevation angle (RNAV radians)
Detailed Driving	
DOES_VO	Does this command perform a Visual Odometry Update
NAV_GOAL	Current Navigation Goal X, Y with Frame, Frame Index
VTT_TARGET	Current Visual Target Tracking X, Y, Z with Frame, Frame Index
DRIVING	Boolean true when this command could cause motion
TRCTL	Boolean true when Traction Control is enabled
STEERS	Boolean true if this command could steer the wheels
STEER_DESC	Description of most recent steering command
STEER_LF	Current LF steering position (radians)
STEER_RF	Current RF steering position (radians)
STEER_RR	Current RR steering position (radians)
STEER_LR	Current LR steering position (radians)
Internals	
NOSTACK_PARALLELS	List of sequences that might be running in parallel
COND_INSIDE	Conditional nesting depth count
PLAN_SOL	Sol on which this command will execute
STACK_DEPTH	Sequence nesting depth count
MOT_15000	Boolean true when this sequence was run without being cleaned up yet
MOT_15002	Boolean true when this sequence was run without being cleaned up yet

When warnings are generated, they will automatically include a link to the command at which the violation was found. The best messages are those that not only point out a problem, but also link back to any prior information that will be helpful in resolving the warning (e.g. "Command #244: You opened the camera dust cover in command #200, but failed to close it by the end of the backbone here."). See Table 2 below for more example warnings.

Using and Updating System State

After each communication from the spacecraft, our downlink team automatically documents the state of the spacecraft in files made available to SSIM and RP-check. These files contain the values of over 24,000 ground-defined parameters and 369 pieces of system state as of September 2017 (e.g., current motor positions, current vehicle position). This information is stored in Non-volatile Parameter Memory (NPM) flash on

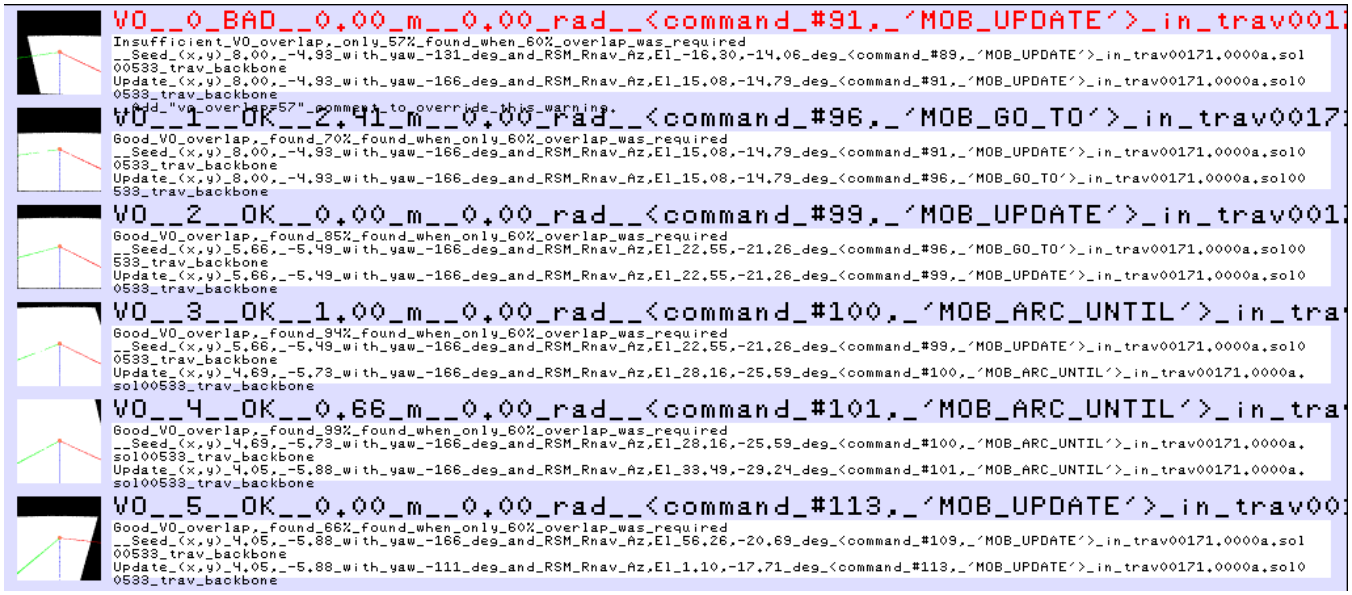

Figure 3. Example report created by the Visual Odometry Overlap Rule

Figure 4. Example report created by the Visual Target Tracking (VTT) rule on Sol 1822.

the spacecraft, and provides the initial state used to predict future spacecraft behavior.

RP-check provides a framework that not only reads NPM to initialize its state, it can also track changes made to parameters throughout Rover Planner command sequences. This allows rule implementers to validate constraints even as parameter values change, and when a constraint is violated to report exactly which parameter-changing command caused it.

Rover Planners can also express two forms of parameter constraints in their sequences; snapshots and direct comparisons. If the Magic Comment `restore_seqid=s1,s2,...` is used in an RML file, RP-check will snapshot the parameter state before any of those sequences is run, and keep track of all parameter assignments made while any of the named sequences is active. Then, when all of the sequences become inactive, it will compare all the named parameters against their initial NPM values. You'll get a warning if there are any differences from the initial state. This helps ensure that diagnostic routines undo any of their parameter changes.

Direct comparisons of scalar parameters to constant values can be enforced as well. If Magic Comments `parm_EQ(parm,val)` (or related forms with `NE, LE, LT, GE, GT`) are found, the named parameter will be checked to ensure that its current value satisfies the given constraint; if not, a warning will be given. Any number of these may appear in comments, they will be evaluated as a conjunction of constraints, and a warning will be generated for each one that fails to satisfy its constraint.

3. IMPLEMENTATION

RP-check is primarily written in Perl [18], using object-oriented syntax available in perl v5.

The MSL Command Dictionary is an XML file shared by all ground tools that encapsulates the legal syntax for spacecraft commands. Whenever a new Command Dictionary is released, RP-check developers preprocess it to create string-initialized perl constants to represent each command opcode /and argument name. By convention, RP-check developers refer to command and argument names *only* using these auto-generated constants. Doing so enables the perl compiler itself to ensure there are no typos or outdated references to nonexistent commands or argument names in the validator.

Helper Tools

RP-check makes use of multiple helper programs and scripts to perform its modeling. It uses the freeware ray-tracing renderer *Rayshade* [7] to generate images used in the Visual Odometry computation, and other helpers developed by JPL (e.g., some tools in [2]) to assist with the following functions:

- Query the current state of the Activities being planned, e.g. start times, sequence names.
- Check the onboard status of Sample Processing Acquisition Manager (SPAM) scripts
- Convert between multiple time formats
- Verify geometric camera model external 3D pointing parameters
- Backup files
- Prune rover state files down to a more manageable size
- Render a ray-traced scene of overlapping 3D camera frusta for Visual Odometry overlap computation
- Compute the percentage of pixels in common for Visual Odometry overlap assessment
- Annotate images with text and graphics
- Compute Double Ackermann steering angles
- Compare RAM offsets to motor status fields
- Compare hex addresses against the symbol table

4. VALIDATION

A validator is only as useful as tests prove it to be. RP-check includes a detailed unit test framework that allows us to ensure that any new changes have not broken its existing capabilities.

An RP-check test case is simply an RML file designed to check the outcome or evaluation of a single rule. Each RML file follows a particular naming convention.

rulename.(pass,fail).###.rml

A "pass" file should pass the named rule (its outcome under other rules is irrelevant for the purpose of this test). A "fail" file should fail the named rule; often there are a greater number of negative test cases because there are usually more ways to be wrong than right. Test files may also be provided in a plain ASCII "rseq" format, as supplemental batch processing will convert them to the standard RML format prior to testing. Test files may be as short as a single command, or as detailed as the full contents of complete multi-sol plan RML file. Table 2 shows some simple examples.

The unit test framework will run the current development version of RP-check over each test file, and log the outcome of each check to stdout. We use GNU Parallel [16] to efficiently evaluate RP-check's performance over a thousand RML files in under ten minutes. Full test logs are compared to prior logs to ensure any unexpected changes will be discovered and corrected prior to release.

In addition to these unit tests, developers often run tests comparing the outputs of old and new versions of RP-check on many of the flight-proven RML files already sent to Curiosity.

5. USAGE DURING MISSION OPERATIONS

Rover Planners can run RP-check dozens of times each day. They rely on the internal RP-only checks and the MSL project's Flight Rule checks to help ensure the safety of their sequences. Results of the MSL Flight Rule checks are posted for review by the entire uplink team every day.

As of September 2017, the MSL version of RP-check implements 242 explicit rules, and has the potential to generate 534 different Warnings and 542 Error messages. It has been run over 1,114 RML files, and validated individual sequences that have executed on 1,139 different sols.

RP-check has been used to validate Mars Rovers' command sequences for Curiosity since 2012, and its predecessor (Flight Rule check, or FR-check) for Spirit and Opportunity since 2004. As such, it is also directly relevant to the 2020 Mars Rover mission, which will use the MSL operations model as a starting point.

Automated Generation of Safety Deactivate Sequences

Over the years some of RP-check's modeling became so detailed, it became possible to not only validate certain sequences, but also *generate* some using the tool. In particular, MSL took advantage of this to automate the creation of "Safety Deactivate" sequences in 2016. That automation eliminates the need for manual creation and review of those sequences each day, saving a few minutes for simple plans, and tens of minutes on complex planning days.

Rover Planners create a single "backbone" command sequence for each period of mobility, arm or turret activities. To ensure that those backbones and any associated helper sequences do not exceed their allocated duration, a corresponding "safety deactivate" sequence is constructed for each backbone. This sequence of contingency commands is run automatically after the planned duration has elapsed to stop any backbones or helpers that overrun their expected durations, and clean up any interrupted commands.

From 2004 until 2016, human rover planners had to construct these safety deactivate sequences by hand each day, and review them several times prior to uplink. RP-check has validated the contents of these sequences for years, so MSL undertook an Operations Improvement Initiative to fully automate their creation. RP-check already understood when commands were missing from those sequences, so in 2016 it was updated to also *construct* such sequences on its own. This automated construction of safety deactivate sequences now provides even more safety: more checking is done than ever before, and the sequences are re-checked just before the backbones are finally delivered for uplink to ensure that nothing was forgotten.

Table 2. Some of RP-check's simpler unit tests. Actual commands have been replaced with pseudocode. These illustrate some of the differences between RP-check and SeqGen. (1) RP-check supports using magic comments to disable warnings. (2) RP-check processes all sequences within a sol, not just a single sequence. (3) RP-check provides access to over 24,000 system parameter values.

Test Sequence	Result
Rule RP-0150: Warn when Traction Control is not being used	
`trav0: Drive 1 meter forward`	Warning (detected at command #0 in trav0): Hey, why aren't you using traction control? Please use "traction_control_disabled=1" to override.
`trav0: Drive 1 meter forward` *# traction_control_disabled=1*	passed
`trav0: Turn on Traction Control` `trav0: Drive 1 meter Forward`	passed
Rule RP-0016: Explicitly Power Off IMU After Driving	
`trav0: Drive 1 meter forward`	Warning (detected at command #0, in trav0: You didn't power off the IMU after driving

`trav0: Drive 1 meter forward`
`trav0: Run trav1`
`trav1: Turn off the IMU`
	passed

Rule RP-0136: Assert parameter values	
`trav0: Set Drive parm susp_diff_min to -0.517` *# parm_GE(drive_susp_diff_min,-0.463)*	Error (detected at command #1 in trav0): FAILED Parameter constraint parm_GE (drive_susp_diff_min, -0.463), current value "-0.517" does not match
`trav0: Set Drive parm susp_diff_min to 0` *# parm_GE(drive_susp_diff_min,-0.463)*	passed

Operational Impacts

As RP-check has become increasingly trusted by the project, it has taken on additional roles. In addition to providing the Automated Safety Deactivate Sequences, it is also a typical component of the Corrective Actions recommended in response to Incident Surprise Anomaly reports. Investigations often conclude with a recommendation to implement a new RP-check rule, which can usually be accomplished swiftly. This not only reflects well on the project's confidence in the tool, it also has helped to keep the Rover Planners' process from becoming unsustainably cumbersome. Instead of a growing patchwork of rules that must be manually checked or new processes that must be followed, we have an equivalent fast, automatic check – one that never becomes tired or loses focus, as a human might. Some examples appear below.

Keeping management of the RP-check tool in the hands of Rover Planners allows us to make, test, and release changes quickly. If a commanding error is discovered, or a strange new system behavior is understood for the first time, it can take time to communicate all aspects of that problem, its implications and its workaround to the entire team. We minimize the chance of repeating earlier mistakes by updating this software validation tool quickly.

Problems and Resolutions

Here are a few examples illustrating how nimble RP-check development activities have led to quick and effective problem mitigations.

- **Checking the wrong Master Sequence ID** Rover Planners begin each of their primary "backbone" sequences with a check to ensure that it is being called by the expected "master" sequence. Master sequences are the primary command sequences that orchestrate the rover's daily activities, activating each backbone and Safety Deactivate after the backbone is due to finish. These are typically named for the sol on which they will execute.

On sol 1247, the wrong check was sequenced in an arm backbone; it checked for "247" in the sequence name, but the prior sol's "246" master was still running. Handover from one sol's master sequence to the next typically happens around 10am Mars Local time, and this backbone was unusual in that it was scheduled to run prior to that handover.
RP-check was quickly updated to check the MSlice plan to learn precisely when the handover to a a new master sequence would occur, and now ensures that RP sequences will test for the correct one.

- **Bug counting motions between VO updates** Rover flight software keeps track of how many motions take place between Visual Odometry (VO) updates. Rover Planners can then use that knowledge to schedule a new update only if the rover has been commanded to move since the last one.
On sol 1753, we noticed that counter would sometimes increment even though no motion was commanded. We eventually realized it was due to a simple flight software bug that always incremented the counter during certain types of primitive motion commands, even if the command decided no actual motor motion was necessary (e.g., given a command to straighten wheels when they are already straight).
Changing flight software takes a lot of resources. But once we understood the nature of the bug, we were able to quickly update RP-check to model it. Now RP-check issues a warning if a sequence attempts to use that counter after any number of primitive motion commands has been issued. This has enabled us to change our current sequences, and be confident that future sequences will avoid tripping this bug again, without requiring a resource-costly flight software update.
- **Traction Control is running** New Traction Control software was uplinked in 2017, and has been in use on Curiosity since sol 1646 [17]. This capability was designed to reduce wear and tear on the wheels, so the project recommends it be used for most driving activities.
RP-check was updated to track the state of this software, incorporating a new TRCTL annotation to indicate that status (documented in Table 1). Given that annotation and the existing "is this a DRIVING command" annotation, adding

a new rule to ensure that Traction Control is enabled for all driving commands was straightforward (see Table 2 for related unit test outputs).

Future Directions

Looking ahead to future missions, several enhancements suggest themselves.

- **Parallel Processing of Rules** Although we have already split the evaluation of hundreds of rules across multiple CPU cores, we have not yet optimized their distribution. We should realize more gains by actively choosing how to distribute rule evaluations based on their predicted durations.
- **Better Terrain Modeling** RP-check's simplified terrain model has been sufficient for validating existing flight rules. But if inputs to RP-check were expanded to include pose estimates derived from detailed terrain modeling under multiple slip assumptions, RP-check could add new types of constraints to ensure resource constraints are met even in slippery terrains under wide variations in the amount of predicted slip.
- **Better Visual Odometry Overlap** RP-check's current Visual Odometry overlap computation assumes the world is flat and well-textured with visible features. But Rover Planners fairly often have to point cameras directly at features of interest to make up for feature-sparse areas like sand patches, or point at hillsides when climbing. Adding knowledge of terrain shape and texture could make the automated checking even more robust.
- **Tighter coupling with traditional programming language static analyzers** The MER and MSL sequencing model is quite restricted, supporting subroutine calls and simple conditionals but only limited use of variables and no recursion. There are likely additional benefits to be gained, especially in modeling of conditionals, so long as such integration does not slow down the verification process too much.

As missions move in the direction of placing more autonomy onboard the spacecraft, validation tools like this become even more important. Although high fidelity simulators like SSIM can generate detailed, precise individual traces by emulating flight software in a simulated environment, you still need tools that can address the uncertainty inherent in exploring and interacting with new terrains.

6. Conclusion

Static analysis of spacecraft command sequences' conformity to mission Flight Rules and Rover Planner team best practices helps ensure operational safety. Our RP-check software has performed this task for mobility, arm and turret command sequences quickly and robustly for years. The ability to quickly deploy new rules developed by experts keeps spacecraft operations safe and effective. Future missions can benefit from making similar architectural choices.

Acknowledgments

FR-check was developed by Scott Maxwell in collaboration with the MER RSVP and Rover Planner operations team in 2004. Scott also provided the initial MSL RP-check implementation for Curiosity in 2012. It is continually evolving with new rule requests coming from the Flight Rule Working Group and Rover Planner teams. Originally written and maintained by Scott, more recent developers include Jeffrey J. Biesiadecki, Mark W. Maimone, and Stirling Algermissen. Thanks to many of the Rover Planner team for their help and suggestions over the years, especially Joseph Carsten, John Michael Morookian and Doug Klein for proposing and validating many arm and turret rules and Oliver Toupet for validating the safety deactivate arm and turret capabilities. Thanks to Vandi Verma for her work on SSIM and helpful comments on this paper.

The work described in this paper was performed at the Jet Propulsion Laboratory, California Institute of Technology, under contract with NASA. Copyright 2017 California Institute of Technology. U.S. Government sponsorship acknowledged.

References

[1] Jeffrey J. Biesiadecki, Eric T. Baumgartner, Robert G. Bonitz, Brian K. Cooper, Frank R. Hartman, P. Christopher Leger, Mark W. Maimone, Scott A. Maxwell, Ashitey Trebi-Ollenu, Edward W. Tunstel, and John R. Wright. Mars Exploration Rover surface operations: Driving Opportunity at Meridiani Planum. In *IEEE Conference on Systems, Man and Cybernetics*, pages 1823–1830, The Big Island, Hawaii, USA, October 2005.

[2] Jeffrey J. Biesiadecki, Robert Liebersbach, and Mark W. Maimone. Mars exploration rover mobility and IDD downlink analysis tools. In *International Symposium on Artificial Intelligence, Robotics, and Automation for Space (i-SAIRAS)*, Los Angeles, California, USA, February 2008.

[3] Thomas Burk and David Bates. Cassini attitude control operations: Flight rules and how they are enforced. In *AIAA Guidance, Navigation and Control Conference and Exhibit*, Honolulu, Hawaii, August 2008. https://arc.aiaa.org/doi/abs/10.2514/6.2008-6808.

[4] Debarati Chattopadhyay, Andrew Mishkin, Alicia Allbaugh, Z Nagin Cox, Steven W Lee, Grace H Tan-Wang, and Guy Pyrzak. The mars science laboratory supratactical process. In *Space Ops*, Pasadena, California, May 2014.

[5] A. Johnson, S. Goldberg, Y. Cheng, and L. Matthies. Robust and efficient stereo feature tracking for visual odometry. In *International Conference on Robotics and Automation*, Pasadena, CA, USA, May 2008.

[6] S. C. Johnson. Lint, a C program checker. In *COMP. SCI. TECH. REP*, pages 78–1273, 1978.

[7] Craig E Kolb. General *rayshade* information. Freeware ray tracer http://graphics.stanford.edu/ cek/rayshade/info.html, 1992.

[8] Chris Leger, Ashitey Trebi-Ollenu, John Wright, Scott Maxwell, Bob Bonitz, Jeff Biesiadecki, Frank Hartman, Brian Cooper, Eric Baumgartner, and Mark Maimone. Mars Exploration Rover surface operations: Driving Spirit at Gusev Crater. In *IEEE Conference on Systems, Man and Cybernetics*, pages 1815–1822, The Big Island, Hawaii, USA, October 2005.

[9] Mark Maimone, Yang Cheng, and Larry Matthies. Two years of visual odometry on the Mars Exploration Rovers. *Journal of Field Robotics*, 24:169–186, 2007.

[10] Andrew H. Mishkin, Daniel Limonadi, Sharon L. Laubach, and Deborah S. Bass. Working the martian night shift: The MER surface operations process. *IEEE*

Robotics and Automation Special Issue (MER), pages 46 – 53, 2006.

[11] Alexander Patrikalakis and Taifun O'Reilly. Advances in discrete-event simulation for MSL command validation. In *IEEE/ACM International Symposium on Distributed Simulation and Real Time Applications*, Delft, Netherland, Oct 2013.

[12] Charles Pecheur. Verification and validation of autonomy software at nasa. Technical Report 20010000882, NASA Ames Research Center, April 2000.

[13] J.F. Peters and S. Ramanna. Verifying command sequences for satellite systems. *IEEE Aerospace and Electronic Systems Magazine*, 7(10):14–21, October 1992.

[14] Mark W. Powell, Khawaja S. Shams, Michael N. Wallick, Jeffrey S. Norris, Joseph C. Joswig, Thomas M. Crockett, Jason M. Fox, Recaredo J. Torres, James A. Kurien, Michael P. McCurdy, Guy Pyrzak, Arash Aghevli, and Andrew G. Bachmann. MSLICE science activity planner for the Mars Science Laboratory mission. Technical report, JPL, September 2009. http://www.techbriefs.com/component/content/article/5707.

[15] Ben Smith, William Millar, Julia Dunphy, Yu-Wen Teng, Pandu Nayak, Ed Gamble, and Micah Clark. Validation and verification of the remote agent for spacecraft autonomy. In *IEEE Aerospace Conference*, Big Sky, Montana, US, March 1999.

[16] O. Tange. Gnu parallel - the command-line power tool. *;login: The USENIX Magazine*, 36(1):42–47, Feb 2011.

[17] Olivier Toupet, Jeffrey Biesiadecki, Arturo Rankin, Amanda Steffy, Gareth Meirion-Griffith, Dan Levine, Maximilian Schadegg, and Mark Maimone. Traction control design and integration onboard the mars science laboratory curiosity rover. In *IEEE Aerospace Conference*, Big Sky, Montana, March 2018.

[18] Larry Wall. *Programming Perl*. O'Reilly & Associates, Inc., Sebastopol, CA, USA, 3rd edition, 2000.

[19] John Wright, Frank Hartman, Brian Cooper, Scott Maxwell, Jeng Yen, and Jack Morrison. Driving on mars with RSVP. *IEEE Robotics and Automation Special Issue (MER)*, pages 37 – 45, 2006.

BIOGRAPHY

Mark Maimone *is a Robotic Systems Engineer in the Robotic Mobility group at the Jet Propulsion Laboratory. He has supported MER and MSL active mission operations since 2004. Mark designed and implemented the GESTALT self-driving surface navigation Flight Software for MER and MSL missions; during MSL operations served as Deputy Lead Rover Planner, Lead Mobility Rover Planner and Flight Software Lead; developed downlink automation tools for MER and MSL; and is now a member of the Mars 2020 Rover FSW development team. He holds a Ph.D. in Computer Science from Carnegie Mellon University.*

Scott Maxwell *A graduate of the University of Illinois at Urbana-Champaign, Scott Maxwell was for several years the solar system's only Mars Rover Driver Team Lead. His most egregious nerdery centers around tiny computers, martial arts, and Shakespeare. He lives in Pasadena, CA with a wife who's way too good for him and their infant son.*

Jeffrey Biesiadecki *has been a software engineer at NASA's Jet Propulsion Laboratory since 1993, after completing his Master's degree in Computer Science at the University of Illinois, Urbana-Champaign. He designed and implemented the core motor control and non-autonomous mobility flight software for the Mars Exploration Rovers (MER) and MSL, and was a rover driver for MER and MSL, responsible for command sequences that tell the rover where to drive and how to operate its robotic arm on the surface of Mars. He is presently leading the development of the Mars 2020 rover Sampling and Caching Subsystem flight software.*

Stirling Algermissen *is a MSL Rover planner certified for mobility, arm, and sampling operations. This role involves developing RP-check based on new flight rules and safety checks. He has also participated in MSL operations as: OPGS imaging shift lead and tactical analyst, Engineering Camera Payload Uplink and Downlink Lead, and a Payload Downlink Coordinator. Stirling develops software for all of these operations roles including MSL's image telemetry processing software, science instrument trending, and data processing pipeline. He is a software developer for MGSS Instrument Data Subsystem's AMMOS-PDS Pipeline Service, DataDrive, and Marsviewer software development tasks, and also for M2020 and InSight's IDS teams.*