Integration of an Arm Kinematics Hot Patch Onboard the Curiosity Rover

Arturo Rankin, Alexandra Holloway, Joseph Carsten, Mark Maimone
Jet Propulsion Laboratory, California Institute of Technology
4800 Oak Grove Dr.
Pasadena, CA 91109
(818) 354-9269
arankin@jpl.nasa.gov

***Abstract*—NASA's Mars Science Laboratory (MSL) mission has updated the Curiosity rover's flight software multiple times since landing on Mars on August 6, 2012. The most common patching method has been a hot patch, in which running flight software is modified after being copied into RAM from its persistent storage. The latest hot patch to be installed on Curiosity fixed an issue in the robotic arm software that computes generalized inverse kinematics. Additional unit testing performed since the start of the surface mission revealed that this software can sometimes produce erroneous solutions.**

The cause was identified as numerical instability in a quartic root finder. When the inputs to that solver are not well conditioned, floating-point numerical issues can cause erroneous roots to be reported. In theory, this could result in the robotic arm turret instruments being commanded to unintended positions, for example, below the terrain surface. Out of approximately 3.7 million unit test cases, 97.2% of the position errors were below 5 mm. However, there were 16 test cases where the position error was greater than 20 cm, and the maximum position error was 1.2 meters.

The patch was uploaded to Curiosity on sol 2642 (January 11, 2020) after the solution was developed, re-implemented as a hot patch, and validated and verified using Earth-based Curiosity testbeds. A checkout test of the patch was performed on Curiosity on sol 2657, and nominal use of the patch began on sol 2658. In this paper, we describe the steps that led to integrating the arm kinematic hot patch into Curiosity's flight software, from the discovery of the bug to the nominal use of the patch in flight.

TABLE OF CONTENTS

1. INTRODUCTION

Many space missions have benefited from the ability to update their software remotely: Earth-orbiting satellites [1], solar system-spanning explorers [2][3], and past NASA Mars Rovers [4], to name a few. In this paper we will describe one of the latest software updates made to the NASA Mars Science Laboratory (MSL) Curiosity Mars rover.

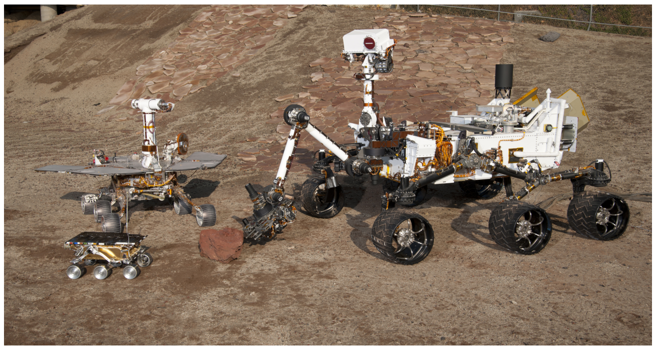

Figure 1. Sojourner (left), MER (middle), and MSL (right) flight-like testbed rovers in the Jet Propulsion Laboratory (JPL) Mars Yard. The MSL Vehicle System Test Bed (VSTB) rover has its robotic arm unstowed with an instrument in close proximity to a rock target. Image Credit: NASA/JPL-Caltech.

On August 6, 2012, the Curiosity rover landed on Mars and began the surface phase of its mission. A Mars solar day (sol) is approximately 39 minutes and 35 seconds longer than the 24 hour Earth day. Curiosity's design lifetime was to survive at least one Mars year (687 Earth days). During its lifetime, Curiosity has mission goals to drive at least 20 kilometers (km), operate its arm joints with specific goals given in Table 1, and acquire at least 27 drill samples.

August 6, 2020 (sol 2857) marked the eight-year anniversary of the landing of the Curiosity rover on Mars. Over those eight years, Curiosity had driven 23,319.0 meters [5] and acquired 28 drill samples, thus far exceeding its design lifetime by 6.1 Earth years, its 20 km mobility objective by 16.6%, and its drill sample objective by 3.7%. The arm joint with the most use has been the turret, which has reached 35.3% of its expected lifetime number of revolutions.

The other NASA spacecraft that have landed successfully on the Martian surface have been the Mars Pathfinder (MPF) Sojourner rover (July 4, 1997) [6], the twin Mars Exploration Rovers (MER) Spirit and Opportunity (January 3 and January 24, 2004, respectively) [7], the Phoenix lander (May 25, 2008) [8], and the Interior Exploration using Seismic Investigations, Geodesy and Heat Transport (InSight) lander (November 26, 2018) [9]. NASA's Mars 2020 (M2020) Perseverance rover launched from Cape Canaveral, Florida on July 30, 2020 and is scheduled to land on Mars on February 18, 2021.

978-1-7281-7436-5/21/$31.00 ©2021 IEEE

Table 1. MSL robotic arm required joint revolutions over its lifetime, and actual joint revolutions over the first 8 years of the mission.

Joint	Required Revs	Actual Revs
1	9,095	1,422.9
2	9,025	1,276.9
3	9,095	2,035.4
4	11,543	3,186.1
5	11,543	4,070.7

Table 2. MSL key robotic arm specifications.

Specification	Value
Degrees of freedom	5
Arm length (from the base to its turret center)	2.2 m
Mass without turret instruments	67 kg
Mass of turret instruments	24 kg
Operating temperature range	–110°C to +50°C
Non-operating temperature range	–128°C to +50°C

All of these spacecraft have a robotic arm except for the Sojourner rover. MSL's robotic arm is the fourth to operate on the surface of Mars [10][11]. The Perseverance robotic arm is kinematically nearly identical to the Curiosity arm, having five degrees of freedom, similar link lengths, and the same actuator layout. Figure 1 illustrates the relative rover size of the three rover missions that have operated on the surface of Mars.

The fact that the rover arms (MER, MSL, and M2020) have five degrees of freedom generally means that their inverse kinematics algorithms are more complex than for the lander arms (Phoenix and Insight) which have only four degrees of freedom. While the arm flight software used for MSL was inherited from MER, MSL's mission requirements necessitated a new ability to control the turret orientation relative to the gravity vector.

The more complex generalized inverse kinematics were implemented to facilitate this capability. And while Perseverance does not require significant gravity relative commanding, not all tools have been physically mounted such that they meet the assumptions of the standard tool inverse kinematics, and thus these generalized inverse kinematics are required there for tool placement as well.

The goal of the MSL mission is to explore and quantitatively assess the habitability and environmental history of the Gale crater field site, which includes the landing ellipse and the adjacent lower portion of Mount Sharp [12]. Proximity science campaigns are performed in between drive campaigns using the Sample Acquisition, Processing, and Handling (SA-SPaH) system.

A key mechanism in the SA-SPaH system is the 5 degree-of-freedom robotic arm, which is responsible for accurately placing five turret-mounted instruments on targets selected by the Science Operations team (see Figure 2). The five instruments on the rotating turret are a drill capable of capturing rock samples, a Mars Hand Lens Imager (MAHLI) camera, a Dust Removal Tool (DRT), an Alpha Particle X-Ray Spectrometer (APXS), and a Collection and Handling for In-Situ Martian Rock Analysis (CHIMRA) device. MSL key robotic arm specifications are shown in Table 2.

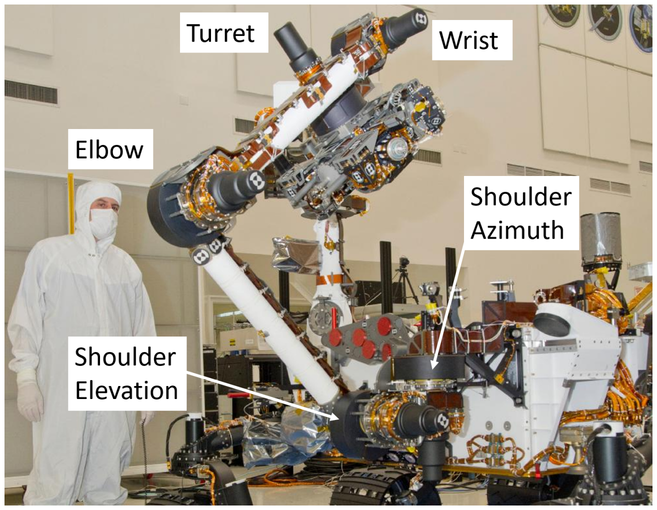

Figure 2. A picture of Curiosity inside the JPL Spacecraft Assembly Facility during pre-launch testing on June 3, 2011, illustrating the five degrees of freedom provided with the robotic arm shoulder azimuth, shoulder elevation, elbow, wrist, and turret joints. Image Credit: NASA/JPL-Caltech.

Table 3. Flight software full updates and cold patches for surface phase on Curiosity through sol 2903.

Release	First Use	Update Type
R9.4.7	Sol 1	Full Update (Cruise)
R10.5.7	Sol 5	Full Update (Cruise)
R10.5.8	Sol 217	Cold Patch
R10.6.4	Sol 264	Full Update (Surface)
R11.0.4	Sol 446	Full Update (Surface)
R11.0.5	Sol 772	Cold Patch
R12.0.3	Sol 875	Full Update (Surface)
R12.0.4	Sol 2808	Cold Patch

Proximity science decisions are guided by MSL's long-term strategic science plan and are subject to changes due to near-term science objectives and engineering constraints arising from terrain features not evident when planning from orbital images. Over the first eight years, robotic arm motion has been commanded on 1310 sols out of 2857 sols (45.9%).

Updates to Curiosity's flight software are rare and typically respond to critical hardware or software defects, or introduce new capabilities helpful for mission success. Thus far during the mission, Curiosity's flight software has been fully upgraded five times, patched and saved into nonvolatile storage three times (as cold patches), and twelve hot patches have been installed to provide incremental improvements where a full release was not justified. All updates that were saved into nonvolatile storage are summarized in Table 3. The most common patching method, however, is a *hot patch*, which is applied to flight software in RAM after the primary Rover Compute Element (RCE) board has booted up [13].

Hot patches can vary in complexity, from simply replacing the value of a single variable to replacing an existing capability with a new capability requiring substantial amounts of new code. An example of a complex hot patch was the replacement of constant wheel speed control with a terrain-adaptive wheel speed control algorithm in 2017, in an effort to reduce the wheel damage rate [14].

A hot patch to fix an issue in the robotic arm generalized inverse kinematics was developed, tested, and flown in 2019. After the start of the surface mission, it was discovered that the robotic arm generalized inverse kinematics could produce erroneous solutions. The root cause is believed to be numerical instability in the quartic root finder used when solving inverse kinematics. In theory, this could result in the robotic arm instruments being commanded to unintended positions, for example, below the terrain surface.

After the development of the arm kinematics hot patch and a Validation and Verification (V&V) test campaign on Curiosity testbeds, the patch was uploaded to Curiosity on sol 2642. A checkout test of the patch was performed on Curiosity on sol 2657, and nominal use of the patch began on sol 2658. In the following sections, we describe the steps in integrating the arm kinematic hot patch into Curiosity's flight software, from the discovery of the bug to its nominal use in flight.

2. Discovery of the Bug

The Curiosity robotic arm software module employs a variety of inverse kinematics functions. These inverse kinematics functions compute the arm joint angles necessary to place the desired tool at a given pose. Most of the inverse kinematics work is calculated by a standard tool inverse kinematics function. The generalized inverse kinematics function is only used in cases where the assumptions made by the standard tool inverse kinematics cannot be met.

During software development of the Perseverance robotic arm software module, which uses the same generalized inverse kinematics code as the Curiosity rover, new unit tests were written to validate the generalized inverse kinematics function. While executing the new unit tests on the Perseverance arm module, there were failures which exposed a potential Curiosity rover vulnerability to robotic arm collisions during arm motions that use the generalized inverse kinematics.

Each unit test starts with a given set of joint angles and a tool control point (TCP). The forward kinematics are run to compute the end effector pose corresponding to those joint angles and TCP. Next, the inverse kinematics are computed using the TCP and calculated end effector pose. This generates sets of joint angles that should result in the specified end effector pose. To verify the inverse kinematics are working correctly, three checks are performed:

1. The inverse kinematics finds at least one solution. We know a valid solution exists because we started with a set of joint angles that produces the end effector pose given as input to the inverse kinematics.
2. The original joint angles are reproduced by the inverse kinematics. It is possible that the inverse kinematics locates alternate solutions (there can be up to 8 solutions for a given end effector pose), but does not find the original joint angles we started with. If this happens, the inverse kinematics is not finding all valid solutions.
3. All reported joint angle solution sets result in the specified end effector pose. For this check, each joint angle set is run through the forward kinematics to compute the end effector pose. This should match the end effector pose given as input to the inverse kinematics.

Table 4. Number of violations that occurred during 3,749,460 unit test cases.

Violation	Linux Workstation	VSTB Testbed
The inverse kinematics DOES NOT find at least one solution	20,167 (0.54%)	20,168 (0.54%)
The original joint angles are NOT reproduced by the inverse kinematics	113,857 (3.04%)	113,854 (3.04%)
All reported joint angle solution sets DO NOT result in the specified end effector pose	105,315 (2.81%)	105,310 (2.81%)

During generalized inverse kinematics unit testing, the user can specify how frequently one of the five joint angles should be varied. An increment of 18° was selected for varying all five joint angles between their min and max values, one joint at a time, which resulted in 937,365 unique sets of joint angles. Four TCPs were selected and each TCP was tested with the 937,365 unique sets of joint angles for a total of 3,749,460 unit test cases.

The Perseverance generalized arm kinematics unit test was initially performed on a Linux workstation and then subsequently on the Curiosity Vehicle System Test Bed (VSTB), a rover testbed containing the same computing avionics hardware as Curiosity. The Linux workstation contains a Pentium central processing unit (CPU) and the VSTB RCEs contain a RAD750 PowerPC CPU. As shown in Table 4, violations of all three of the checks were found. The tolerance used for the missing joint angles check was 0.974° for each joint. The tolerance used for the erroneous solution check was 5 mm in distance and a 0.974° orientation difference in end effector pose.

Violations of the first two checks are unfortunate, but not a safety concern. If a solution to the inverse kinematics cannot be found, the activity will simply fault out (and this is generally caught in simulation before commands are uplinked to the rover). However, violations of the 3rd check are more concerning. In theory, this could result in the arm being commanded to unintended positions, for example, below the terrain surface. The generalized inverse kinematics unit test resulted in 2.81% of the 3,749,460 cases failing with erroneous solutions. The number of violations encountered on the Linux workstation and the VSTB are slightly different due to different floating-point hardware on the different CPUs.

Figure 3 illustrates the distribution of position error observed during the unit test. Of the 113,494 reported solutions with a position error greater than 5 mm, most (89%) resulted in a position error of less than 2 cm. However, there were 16 test cases where the position error was greater than 20 cm, and the maximum position error was 1.2 meters. Figure 4 illustrates the distribution of orientation error observed during the unit test. Of the 367,526 reported solutions with an orientation error greater than 0.974 degrees, most (81.8%) resulted in an orientation error of less than 5°. However, 0.3% of the orientation errors were greater than 20° and the maximum orientation error was 54.8°.

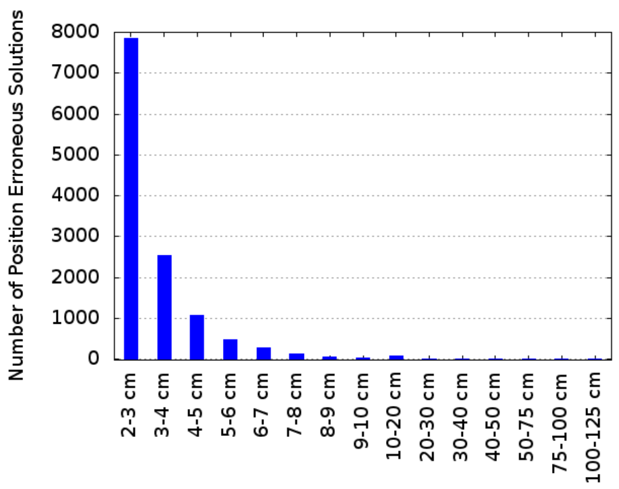

Figure 3. Distribution of the magnitude of position error observed during the unit test. Most errors were less than 2 cm, but the max error was 1.21 meters.

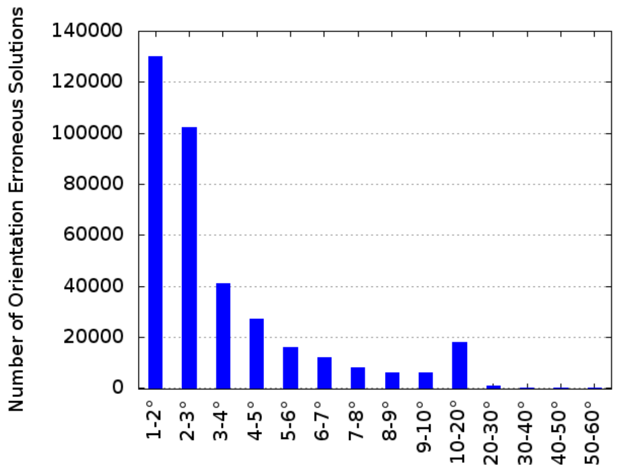

Figure 4. Distribution of the orientation error observed during the unit test. The max orientation error was 54.8°.

3. Risk of Occurrence on Curiosity

To understand the risk of an erroneous solution occurring on Curiosity, the operations team determined the number of commands that can call generalized inverse kinematics (from a review of the source code), the number of those commands that have been executed on Curiosity at least once, and the total number of those commands that have been executed over the first eight years of the mission. The numbers are listed in Table 5.

As each of those commands may call generalized inverse kinematics many times in order to construct a Cartesian trajectory, it's non-trivial to determine the total number of calls that have been made to generalized inverse kinematics function over this period. The 2,769 total number of executed commands that made any use of the generalized inverse kinematics is a lower bound that likely significantly undercounts the actual number of calls to the generalized inverse kinematics function.

The unit test was re-run using a position error tolerance of 2 cm and an orientation error tolerance of 5°. Of the 3,749,460 unit test cases, 26,066 cases resulted in either a position error greater than 2cm or an orientation error of greater than 5°, which is a rate of occurrence for this unit test of one per 143.8 calls to the generalized inverse kinematics. It should be noted, however, that not every bad solution would be used; a single command that makes use of the generalized inverse kinematics might return multiple solutions, many of which need not have a significant error.

Table 5. Generalized inverse kinematics usage during the first 8 years of the mission.

Usage	Number
Available commands that can call generalized inverse kinematics	5
Available commands that have called generalized inverse kinematics	4
Executed commands that have called generalized inverse kinematics	2,769

This analysis is conservative as it makes no assumption about the trajectory generation process. The majority of the uses of the generalized inverse kinematics have been to construct Cartesian trajectories, where the end effector is moved in an approximate straight line from the starting pose to the goal pose. Cartesian trajectories are generated as a series of via points on the Cartesian straight line between starting and ending poses (which requires computing the inverse kinematics at every via point). Via points are added to the trajectory until the deviation from the straight line is below a parameterized threshold at the mid-point between each and every via point. It is very unlikely that a Cartesian trajectory would be successfully computed if the inverse kinematics produces erroneous solutions. In that case, additional via points would likely be added up to the maximum number allowed without constructing a trajectory that meets the straight-line criteria. Therefore, as a side effect of this process, erroneous solutions will often result in a fault before any arm motion is commanded.

The operations team also made an assessment of whether flight-software fault protection could protect against all erroneous solutions. There are general fault protection mechanisms designed to protect the robotic arm hardware in the event of inadvertent contact with rover structure or terrain. Fault protection scenarios were as follows:

1. The onboard collision model, which is designed to prevent collisions between the robotic arm and the rest of the vehicle, runs the forward kinematics on the actual computed joint angles. So if erroneous solutions were to result in collisions between enabled collision objects, these will be caught during pre-check and the move will result in a fault prior to any motion occurring.
2. Unexpected turret instrument contact-switch trips will result in a fault.
3. Unexpected loads measured by the force sensor will result in a fault.
4. Pose dependent current limits are designed to limit the joint torques in the case of a motor stall (which is what would eventually happen if no other safety checks tripped and the arm were to make inadvertent contact).

The consensus among robotic arm subject-matter experts is that there is no guarantee that these fault protection mechanisms are sufficient to prevent damage in all situations, but

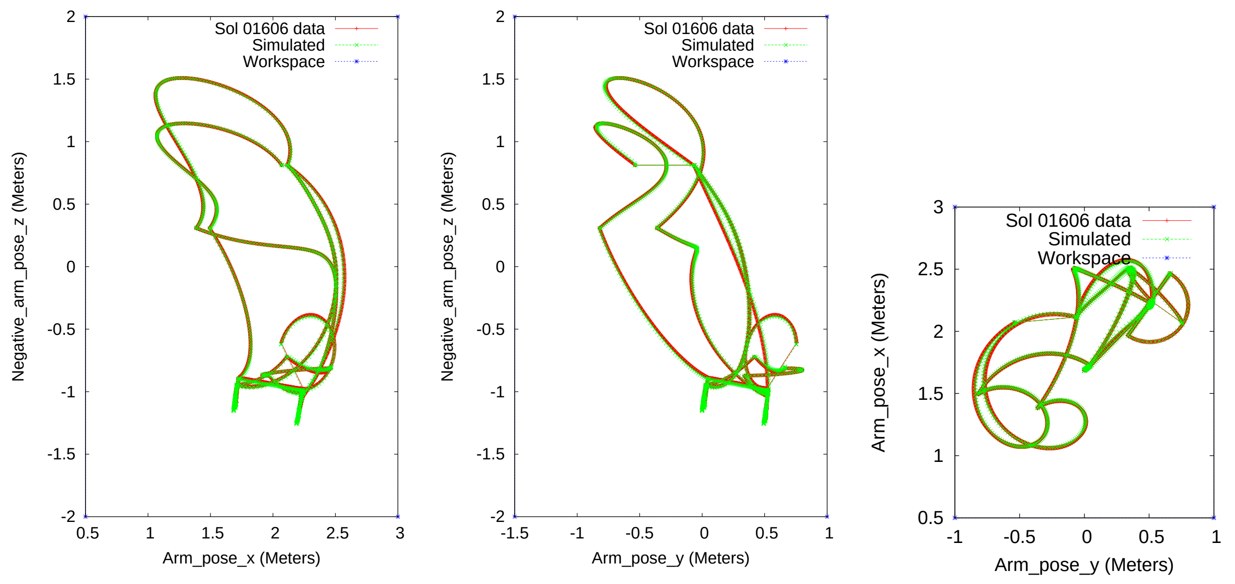

Figure 5. Comparison of planned (simulated) motion of the TCP vs the actual motion from sol 1606 arm activities. The X, Y and Z values are expressed in Rover Mechanical Frame with X forward, Y right, Z down. The left plot is X vs negative Z (viewed from right of rover), the middle plot is Y vs negative Z (viewed from rover deck), and the right plot is Y vs X (viewed from overhead). Thin green lines indicate Tool Frame changes; there is no corresponding turret motion, the TCP simply moves to a different tool on the turret. The ”workspace” label in the plots just indicates the min and max bounds for each plot.

they are designed to minimize the damage potential to the extent possible. Given an increasing risk of experiencing a large position error over time, the project decided to implement a hot patch to eliminate that risk.

To date, the operations team has not identified a robotic arm erroneous solution on Curiosity. That's not to say that Curiosity has never encountered one, but if it did there were not any adverse ramifications. Theoretically, it should be possible to reconstruct the commanded poses and TCPs used for all Curiosity robotic arm moves commanded by generalized inverse kinematics and then see if the resulting joint angles make sense. However, such a task has not been undertaken because it would require significant effort and impact resources for higher priority mission tasks. Figure 5 shows one potential avenue of comparison, comparing the planned vs actual trajectory of the TCP in 3-D over the course of a whole sol's activities.

4. Mitigation

The root cause of this issue is believed to be numerical instability in the quartic root finder utilized when solving the inverse kinematics. When the inputs to that solver are not well conditioned, floating-point numerical issues are believed to cause erroneous roots to be reported. This issue was mitigated in two ways using a hot patch.

First, iterative root polishing using Newton's method was enabled. The idea here was to correct any erroneous roots using an iterative method to search for a true root in the neighborhood of the erroneous one. Root polishing code already existed in the robotic arm flight software, but was disabled by default. Enabling root polishing is controlled by setting the value of a global variable to non-zero, which is trivial to perform in a hot patch. However, during unit testing, two bugs were identified in the heretofore unused implementation of the root polishing code. First, it was discovered that a conditional expression incorrectly used $\geq$ (greater or equal) but should have used $\leq$ (less than or equal). Secondly, it was discovered that a call to a function containing Newton's method had an incorrect integer argument value. Both of the bugs could also be fixed in a hot patch by poking corrected assembly code instructions into the flight software binary image memory locations where the errors exist.

Second, a check of the solutions before returning them was added. Source code was added to the end of an existing generalized inverse kinematics function which checks the results of the generalized inverse kinematics. Every set of joint angles is run through the forward kinematics to generate the corresponding end effector pose. This pose is then compared to the end effector pose given as input to the inverse kinematics. If the two are different, that solution is discarded. This only addresses the erroneous solution violations, causing the kinematics to fail in that case rather than providing a bad solution. This was implemented as a hot patch by making a copy of the function with a different function name, adding the check to the bottom of the new function, and modifying jumps to the old function to call the new function instead.

These two fixes work in tandem. The first fixes the vast majority of the violations, and the second acts as a belt-and-suspenders check to guarantee the arm is never commanded to an unintended position. Fixes 1 and 2 were made in Perseverance robotic arm source code. Unit tests were executed on a Linux workstation with only fix 1, only fix 2, and combined fix 1 and 2 enabled.

Table 6. Number of violations that occurred during 3,749,460 unit test cases after implementing fix 1.

Violation	**Linux Workstation**	**VSTB Testbed**
The inverse kinematics DOES NOT find at least one solution	0	0
The original joint angles are NOT reproduced by the inverse kinematics	29	24
All reported joint angle solution sets DO NOT result in the specified end effector pose	0	0

Table 7. Number of violations that occurred during 3,749,460 unit test cases after implementing fix 2.

Violation	**Linux Workstation**	**VSTB Testbed**
The inverse kinematics DOES NOT find at least one solution	139,650	142,610
The original joint angles are NOT reproduced by the inverse kinematics	23,175	21,561
All reported joint angle solution sets DO NOT result in the specified end effector pose	0	0

The number of violations that occurred during the 3,749,460 unit test cases on a Linux workstation with fix 1, fix 2, and combined fix 1 and 2 are shown in the middle column of Table 6, Table 7, and Table 8, respectively. For fix 2, the thresholds used to reject joint angles that did not result in the specified end effector pose was 0.1 mm and 0.1°. As expected, the erroneous solutions were eliminated, essentially being shifted into more failures to find any solutions.

The results for fix 1 and combined fix 1 and 2 were identical. But because fix 1 does not guarantee complete absence of any erroneous solutions and fix 2 does, both fixes were implemented in a single hot patch for Curiosity. On February 6, 2018, the MSL Change Control Board approved an Engineering Change Request for the flight software team to develop this hot patch.

During development testing of the hot patch, the unit testing performed on a Linux workstation was repeated on the VSTB. The number of violations that occurred during the 3,749,460 unit test cases on the VSTB with fix 1, fix 2, and combined fix 1 and 2 are shown in the rightmost column of Table 6, Table 7, and Table 8, respectively. The number of violations encountered on the Linux workstation and the VSTB are slightly different due to different floating-point hardware on the different CPUs.

5. Integration of the Patch into Mission Operations

Flight software updates on a remote spacecraft require careful planning, testing and review. Flight team members always keep in mind lessons learned from prior missions, such as the fact that missions can be lost due to incorrect updates. For instance, the Viking 1 lander was lost due a software update

Table 8. Number of violations that occurred during 3,749,460 unit test cases after implementing fix 1 and 2.

Violation	**Linux Workstation**	**VSTB Testbed**
The inverse kinematics DOES NOT find at least one solution	0	0
Missing original joint angles	29	24
All reported joint angle solution sets DO NOT result in the specified end effector pose	0	0

inadvertently overwriting some code that was still needed to point its antenna back to Earth. And the Mars Global Surveyor mission was eventually lost due to an error in a memory address written during a parameter update [15].

Updating a complete flight software image requires substantial project resources. Any new capabilities must be tested in a flight-like manner, and because the entire system is being redeployed, even unrelated capabilities must be regression-tested. The results of tests must be reviewed and interactions with other systems considered. And even once the new software has been successfully validated, operational plans must be made to uplink the new software, sometimes requiring months of logistical work to arrange sufficient uplink bandwidth using the most capable Deep Space Network antennas.

In contrast, hot patches require far less effort. The software changes are guaranteed to impact only a small number of functions, eliminating the need to regression-test the whole system. Uplink bandwidth is typically far less than that required for a full system update, eliminating the need for specialized uplink planning. And even if a problem should occur that reboots the rover into safe mode, the hot patch is guaranteed not be installed after the reboot, leaving the rover in a known and consistent state; sequenced commands are not executed in safe mode, so no patch will be applied.

Arm Kinematics Patch

Once development of this particular software source code update was completed, it was compiled into a single object file following standard protocols for Curiosity flight software hot patches [13]. The arm kinematics hot patch files consist of a single object file containing the new functionality, and a shell command script containing VxWorks operating system commands to load the object file to VxWorks, set the values of global variables, and poke new address offsets into memory locations that called the original function. This command script is executed in each bootup period that requires the arm kinematics hot patch.

During the Validation and Verification test campaign, the arm kinematics hot patch was installed on the VSTB and the robotic arm and fault-protection flight software modules were regression tested. In addition, a overall systems regression test was performed called Sol-in-the-Life, which includes common command and control activities during a typical sol.

Robotic arm testing was also performed with additional patches under development installed, to verify their compatibility with the arm kinematics hot patch. This test included some drive commands. At the time this testing was performed, the VSTB only had a single RCE installed, so the portion of the overall fault-protection regression test that requires two RCEs was performed on the Mission System Testbed (MSTB), a platform with two RCEs but no actuators.

The arm kinematics patch is hundreds of times smaller than a full flight software load would be, and the files that comprise it were able to be completely uploaded on sol 2642. During a checkout test of the hot patch on sol 2657, the shell command script was executed to install the hot patch and arm motion was commanded to exercise the generalized inverse kinematics. The checkout test was successful and the patch was approved for nominal use on sol 2658. Since then, the arm kinematics hot patch has been applied to Curiosity's flight software on all sols that contain robotic arm activities.

6. SUMMARY

The robotic arm on the M2020 Perseverance rover is kinematically nearly identical to the robotic arm on the MSL Curiosity rover, having five degrees of freedom, similar link lengths, and the same actuator layout. In addition, Perseverance inherited much of Curiosity's arm flight software, including the generalized inverse kinematics.

During unit testing of the Perseverance robotic arm flight software module on a Linux workstation with 3,749,460 unique sets of joint angles, it was discovered that the generalized inverse kinematics function can generate erroneous solutions that cause position and orientation error. The average rate of occurrence of position error greater than 2 cm or orientation error greater than $5^\circ$was one per 143.8 calls to the generalized inverse kinematics, and the max position error was 1.21 meters. The unit test results were reproduced on the VSTB engineering model of the Curiosity rover with minor differences in the number of violations due to different floating-point hardware between the Linux workstation and the VSTB.

Curiosity has not shown evidence of having encountered a robotic arm erroneous solution – though one may have occurred without drastic consequences. However, in theory, an erroneous solution could result in a turret instrument being commanded to an unintended position, for example, below the terrain surface. Although there are general fault protection mechanisms designed to protect the robotic arm hardware in the event of inadvertent contact with rover structure or terrain, there is no guarantee that these fault protection mechanisms are sufficient to prevent damage in all situations.

A code correction was made to the Perseverance source code, which inherited its robotic arm codebase from Curiosity. The Perseverance bug fix was then implemented as a hot patch for Curiosity. The hot patch was uploaded to Curiosity on sol 2642, and after a checkout test on Mars on sol 2657, it was approved for nominal use on sol 2658.

During the 199 sols between when the patch was approved for nominal use and the 8 year anniversary of the Curiosity landing (sol 2857), robotic arm motion has occurred on 93 sols (46.7%) and generalized inverse kinematics has been called 488 times. The arm kinematics hot patch is applied to Curiosity's flight software on all sols that contain robotic arm activities, eliminating the risk of erroneous solutions from generalized inverse kinematics. The Curiosity SA-SPaH team will continue to monitor robotic arm performance and flight software improvements will be implemented as needed as Curiosity continues to explore as-yet-unseen terrains.

ACKNOWLEDGMENTS

The research described in this paper was performed at the Jet Propulsion Laboratory, California Institute of Technology, under contract with the National Aeronautics and Space Administration. The authors would like to thank the Mars Science Laboratory Program for supporting this research, Igor Uchenik for assistance debugging patch assembly code, Neel Patel for providing SA-SPaH trending data, and Doug Klein and Anna Boettcher for help documenting arm and drill mission requirements and performance.

REFERENCES

[1] A. Calder and P. Kutt, "Flight software design guidelines for enhancing on-orbit maintenance," in *AIAA Infotech Aerospace Conference*, Seattle, Washington, USA, Apr. 2009.

[2] J. Wenz, "Why NASA needs a programmer fluent in 60-year old languages," *Popular Mechanics*, Oct. 2015.

[3] J. H. Webmaster, "New horizons gets a new years workout," http://pluto.jhuapl.edu/news_center/news/20130110.php, Jan. 2013, accessed: 2020-10-03.

[4] G. Reeves and T. Neilson, "The Mars rover Spirit flash anomaly," Big Sky, Montana, Mar. 2005.

[5] A. Rankin, M. Maimone, J. Biesiadecki, N. Patel, D. Levine, and O. Toupet, "Driving Curiosity: Mars rover mobility trends duing the first seven years," in *IEEE Aerospace Conference*, Big Sky, Montana, USA, Mar. 2020.

[6] D. Bickler, "Roving over Mars," *ASME Mechanical Engineering Magazine*, vol. 120, no. 4, pp. 73–77, Apr. 1998.

[7] E. Tunstel, M. Maimone, A. Trebi-Ollennu, J. Yen, R. Petras, and R. Willson, "Mars Exploration Rover mobility and robotic arm operational performance," in *IEEE International Conference on Systems, Man and Cybernetics*, Waikoloa, Hawaii, USA, Oct. 2005.

[8] R. Bonitz, L. Shiraishi, M. Robinson, J. Carsten, R. Volpe, A. Trebi-Ollennu, R. Arvidson, P. C. Chu, J. J. Wilson, and K. R. Davis, "The Phoenix Mars lander robotic arm," in *IEEE Aerospace Conference*, Big Sky, Montana, USA, Mar. 2009.

[9] A. Trebi-Ollennu, W. Kim, K. Ali, O. Khan, C. Sorice, P. Bailey, J. Umland, R. Bonitz, C. Ciarleglio, J. Knight, N. Haddad, K. Klein, S. Nowak, D. Klein, N. Onufer, K. Glazebrook, B. Kobeissi, E. Baez, F. Sarkissian, M. Badalian, H. Abarca, R. G. Deen, J. Yen, S. Myint, J. Maki, A. Pourangi, J. Grinblat, B. Bone, N. Warner, J. Singer, J. Ervin, and J. Lin, "Insight Mars lander robotics instrument deployment system," *Space Science Reviews*, vol. 214, no. 93, pp. 1–18, Jul. 2018.

[10] M. Robinson, C. Collins, P. Leger, W. Kim, J. Carsten, V. Tompkins, A. Trebi-Ollennu, and B. Florow, "Test and validation of the Mars Science Laboratory robotic arm," in *8th International Conference on System of Systems Engineering*, Maui, Hawaii, USA, Jun. 2013, pp. 184–189.

[11] M. Robinson, C. Collins, P. Leger, J. Carsten, V. Tompkins, F. Hartman, and J. Yen, "In-situ operations and planning for the Mars Science Laboratory robotic arm: The first 200 sols," in *8th International Conference on System of Systems Engineering*, Maui, Hawaii, USA, Jun. 2013, pp. 153–158.

[12] J. Grotzinger, J. Crisp, A. Vasavada, R. Anderson, C. Baker, R. Barry, D. Blake, P. Conrad, K. Edgett, B. Ferdowski, R. Gellert, J. Gilbert, M. Golombek, J. Gómez-Elvira, D. Hassler, L. Jandura, M. Litvak, P. Mahaffy, J. Maki, M. Meyer, M. Malin, I. Mitrofanov, J. Simmonds, D. Vaniman, R. Welch, and R. Wiens, "Mars Science Laboratory mission and science investigation," *Space Science Reviews*, vol. 170, pp. 5 – 56, Sep. 2012.

[13] E. Benowitz and M. Maimone, "Patching flight software on Mars," in *Workshop on Spacecraft Flight Software*, Laurel, MD, USA, Oct. 2015.

[14] O. Toupet, J. Biesiadecki, A. Rankin, A. Steffy, G. Meirion-Griffith, D. Levine, M. Schadegg, and M. Maimone, "Terrain-adaptive wheel speed control on the Curiosity Mars rover: Algorithm and flight results," *Journal of Field Robotics*, vol. 37, no. 5, pp. 699–728, Aug. 2020.

[15] G. Webster and D. Brown, "NASA report reveals likely causes of Mars spacecraft loss," https://www.nasa.gov/mission_pages/mgs/mgs-20070413.html, Apr. 2007, accessed: 2020-10-03.

BIOGRAPHY

Arturo Rankin *received his Ph.D. in Mechanical Engineering at the University of Florida in 1997. He is currently a Robotic Systems Engineer in the Robot Operations group at the Jet Propulsion Laboratory, and the M2020 Robot Operations deputy lead. Prior to that, he was the MSL Mobility/Mechanisms team lead and the MSL Flight Software team lead.*

Alexandra Holloway *received her Ph.D. in Computer Science at the University of California, Santa Cruz, in 2015. Specializing in human-centered design and storage systems, she now leads the MSL Flight Software team in building new capabilities for the Curiosity rover. Previously, Alexandra designed user interfaces for the operators of the Deep Space Network.*

Joseph Carsten *earned his M.S. degree in robotics from Carnegie Mellon University in 2005. Since that time he has worked at the Jet Propulsion Laboratory as a flight software developer and Mars mission operator. He has developed mobility, arm, and sampling flight software for the Mars Exploration Rover (MER), Mars Science Laboratory (MSL), and Mars 2020 missions. He has also served on the MER, MSL, Phoenix lander and M2020 robotic operations teams.*

Mark Maimone *is a Robotic Systems Engineer in the Robotic Mobility group at the Jet Propulsion Laboratory. 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 and Rover Planner teams. He holds a Ph.D. in Computer Science from Carnegie Mellon University.*