Flight Software Case Study:
Spacecraft Telemetry

Joseph Carsten

Jet Propulsion Laboratory, California Institute of Technology

1 Author's Note

The information contained here is highly biased by the author's experience. This is experience is dominated by Mars rover development and surface operations. The author's flight software development experience spans the areas of autonomous driving, robotic arm control, antenna pointing, and high-level activity coordination. The author has served as a rover driver for the Mars Exploration Rovers (MER), Mars Science Laboratory (MSL), and Mars 2020. He has also been a robotic arm operator for these missions as well as the Mars Phoenix lander.

2 Introduction

Welcome to the unglamorous but critically important world of spacecraft telemetry. This is an area that often doesn't get a huge amount of design attention and specification. However, telemetry is the only visibility into what a spacecraft is doing. No one is standing next to the vehicle watching and listening to it as it executes commanded activities. The information that gets put into telemetry is the only insight we have. This is how we understand both hardware and software performance, as well as diagnose and respond to anomalies. Insufficient reporting can mean early warning signs of hardware degradation might be missed. Sparse reporting can also result in more ground tool development to help reconstruct an understanding of what the vehicle is doing from the limited information made available.

Anomaly investigations represent one of the key events in which the available telemetry is heavily scrutinized and the timeliness and clarity of that telemetry is most important. For simple anomalies, having clear telemetry that is downlinked as soon as possible can facilitate a quick diagnosis and recovery. Telemetry that is sparse, confusing, or delayed will likely result in operational stand downs to enable operators enough time to understand and gather more data, thus consuming spacecraft life that could have been devoted to the science mission. For more complex anomalies, the anomaly resolution is unlikely to be reached in the planning cycle following the anomaly, and the investigation can span many planning cycles. In these cases, operators may be hunting for every scrap of data they can find that would help lead to a root cause. Diagnostic activities may be executed onboard the vehicle to amass additional data, often gathering more detailed telemetry than is nominally collected. As a flight software developer, it is useful to have these scenarios in mind. Have you left enough breadcrumbs behind to diagnose unexpected behavior of the algorithms in your module or problems with the hardware your module interacts with? For any software related problems, it could very well be you that gets the call when there is an anomaly. What data do you want to have in order to diagnose issues that may come up during testing or operations? Determining the part of the system responsible for an issue is often the first step in the investigation. This may not always be immediately obvious, and you may end up devoting significant effort looking into an issue only to eventually discover there is no problem with your code and the issue lies in some part of the code that someone else owns. It can be useful to

think about telemetry that makes it easier to determine which side of an interface might be responsible for problems so the investigation can be routed to the correct developer more expeditiously.

There are three main classes of spacecraft telemetry: Event Reporting (EVR), Engineering, Housekeeping, and Accountability (EHA), and data products.

3 Event Reporting (EVR)

This section coming later!

4 Engineering, Housekeeping, and Accountability (EHA)

This section coming later!

5 Data Products

Data products form the final class of spacecraft telemetry. Data products are the most general form of telemetry and can contain nearly arbitrary data. They shine at reporting large pieces of data like images or high rate motor telemetry, but can report small chunks of data as well. When a data product is recorded, the application writing the product assigns it a downlink priority. This priority determines the order in which it will be transmitted back to Earth. Generating a data product generally requires asynchronous message passing, making it more complicated to implement than EVRs or EHA which can be generated through synchronous interfaces.

5.1 Data Product Size

It is important to be cognizant of the size of the data products you generate. Generally the bandwidth between Earth and the spacecraft is heavily limited. Any routinely generated engineering data means there is less bandwidth left over for science, and science is likely the primary reason the mission exists. This issue has a larger impact on missions with more limited telecommunication performance and those that have instruments that are capable of generating very large data sets.

Another nuance to consider is the amount of *decisional* data volume available. Decisional data is the data necessary to decide what to do next with the spacecraft. For example, when driving a rover to a new location, post-drive imaging of the area around the rover is often decisional. This imagery is likely needed to plan science observations near the rover or pick the next drive target. Without this imagery, the observations that can be planned are limited and likely suboptimal. While a mission may have very good data return overall, it could be that the decisional data volume is much more of a limiting factor. For a Mars surface mission, decisional data may often be limited to a single afternoon communication pass relayed through an orbital asset. If the data you are generating is likely to be decisional, there is extra incentive to spend more time optimizing for size.

Limitations of spacecraft storage should also be considered. It's possible the spacecraft avionics are limited and there isn't a huge amount of space to store data products.

Another consideration is the impact of spacecraft safing on telecommunication rates. For certain classes of anomalies, the spacecraft may transition into safe mode. This mode may be accompanied by non-standard activities that can severely limit the available bandwidth. The communication bit rate may be reduced in order to facilitate more reliable communication. Or the spacecraft may turn to a suboptimal

communication attitude in order to optimize other factors. If data you are generating might be needed to decide how to recover from a safe mode event, it can be important to optimize for data product size.

5.1.1 Optimizing Data Product Size

There are a variety of strategies that can be used to optimize data product size.

5.1.1.1 Data Types

Paying special attention to the size of the type used to report a given telemetry field can pay sizeable dividends. The payoff will be highest for fields that are reported over and over again (like a motor position in a motion history). For instance on MSL and Mars 2020, floating point values are largely stored in 64 bit fields. This has utility when performing mathematical operations as wider bit fields do a better job of minimizing floating point round off issues. However, that level of precision is often overkill for telemetry reporting. Simply converting those 64 bit values into 32 bit ones when populating telemetry channels can cut the data volume in half. You do need to be sure that 32 bits of precision is sufficient for the information being reported, but in most cases it will be. Another way to achieve even more savings is to convert the value into a 16 bit integer. For this approach you'll often want to apply a scale factor first in order to report as much precision as possible. For instance, if you have a value that varies between 0 and 500, multiplying the value by 100 before rounding it to an integer value will give a couple more decimals of precision while still keeping the values within what can be represented by a 16 bit unsigned number (0 to 65535). Note that this approach will mean that the scaling will need to be undone on the ground side to get back to the original value.

Another optimization that can be made applies to enumerated types. On MSL and Mars 2020, enumerated values were stored as 32 bit integers. Often the number of unique values is fairly limited and could easily fit into a much smaller data type like an 8 bit integer for telemetry transmission.

In some cases, data volume can be optimized by bit packing the telemetry fields. If you have a bunch of Boolean flags for instance, they could be combined into a single telemetry field where each bit represents one of the flags. Wider data can also be bit packed. For instance if you have pieces of state with 6 distinct values, each state can be represented with 3 bits. You could pack 10 of these states into a single 32 bit integer. Note that bit packing will often require special decoding on the ground side to turn the telemetered values into something intelligible. It's good to make sure you have strategy for handling this in a reasonable way. Ideally the ground data system will already have provisions for this type of decoding, but that is not always the case.

5.1.1.2 Data Collection Options

Another way of managing the data volume generated is to provide data collection options that can be modified by operators. The idea is to allow operators to turn on or off certain pieces of telemetry, or to tailor the fidelity of the reported telemetry. For certain operations there may be a desire for increased telemetry, and for others a smaller set of data may be deemed sufficient. Having knobs to increase the data collected can be extremely useful for anomaly investigations and diagnostic activities.

When implementing various data collection options, it's good to keep in mind the operational scenarios. Ideally the system would be constructed such that operators don't have to make many changes to telemetry configuration when executing a nominal set of activities. Constructing the system such that operators are forced to constantly micromanage the available telemetry collection options is generally not the most preferred approach. Striving for a system with some telemetry collection parameters that

can be tuned and then largely left alone is generally a better approach. For example, on MSL the rate at which motor telemetry is recorded during arm motion is parameterized. The arm stow and unstow activities are long enough that the motor telemetry would overflow the internal data collection buffers if collected at the standard rate. Instead of having to toggle a single data collection rate parameter around every arm stow and unstow, the data collection rate was parameterized across two parameters. One governed the data collection rate when stowing and unstowing the arm, and the other controlled the rate for all other arm motion. This architecture resulted in cleaner sequencing and less things for operators to directly manage on daily basis.

5.1.1.3 Summary Data

Generating high-level summary data reported independently from the more detailed data products is another way of optimizing returned telemetry. A small sized summary will generally be easy to downlink on a decisional basis. This summary can be used to determine if everything performed roughly as expected. Outliers in the summary data can trigger operators to request more detailed telemetry covering the operation in question. If the summary data looks nominal, operators could choose to simply delete the corresponding large, detailed data products. Data included in these sorts of summaries is often statistical in nature. This might include minimum, maximum, and average values across an operation. There may also be small algorithm specific quantities that are useful to report. For instance, a visual odometry algorithm might report the number of features tracked and a measure of their spread across the image.

5.1.1.4 Compression

Compression is another way to reduce data volume. Lossless compression can be used to decrease the size of a data product for "free". However, compression generally takes computing resources, so in CPU limited environments using compression may come with non-trivial system costs of which to be aware. Compressed data products will need to be uncompressed before they can be further processed and interpreted, so ensure this is taken into account by the ground data system.

Traditionally, image data products have supported compression (both lossless and lossy), but compression of other types of data products is not as common. When considering data product compression, its good to leverage any compression related infrastructure available on the system. There may be common compression libraries or services that can be utilized. Both MSL and Mars 2020 provided a great feature in which compression was built into the data product infrastructure. This allowed any data product to be compressed simply by enabling it through a parameter. No extra work was necessary on the part of the data product creator to enable this functionality. One lesson learned from both MSL and Mars 2020 is to enable this type of data product compression early in the test campaign. On both of those missions, it took significant effort to actually enable data product compression for various products on the flight vehicle since the verification and validation campaigns for the various systems rarely made use of this feature. The primary concern was the additional CPU loading needed to perform the compression, and the fear that it would negatively impact the activity being performed. Having compression enabled for test campaign obviates most of that concern and makes compression the default standard going into operations.

5.1.2 Very Small Products

There are some potential nuances that are particularly relevant to very small data products. Every data product likely comes with a set of standard metadata. On Mars 2020 this metadata reported things like

rover position and the sequence ID generating the data product. This metadata was around 111 bytes. For very small data products, this metadata may comprise a significant fraction of the overall data volume.

In addition to storage size constraints, the system may also have constraints on the number of data products that can be stored. Generating many, many small data products may result in consumption of a problematic number of these available slots (on MER the total number of data products was limited to something on the order of a couple thousand).

Lots of small products may also have a negative impact on other aspects of system performance. One example of this was the Mars 2020 packetizer. The packetizer converts data products into the packets that are actually transmitted over the radio. Encountering a large chunk of very small products would cause the packetizer to fall behind and fail to keep up with the transmission rate. This would result in parts of the pass being wasted as there was no data available to send.

Instead of generating lots of very small products, you might consider alternatives. Perhaps the information could be better communicated through EVRs. Or maybe instead of generating a data product for each of these small pieces of information, many of the pieces could be accumulated into a single data product.

5.2 Anomaly Investigation and Resolution

For any non-trivial anomaly, data products will likely play a critical role in understanding the problem and planning a recovery.

5.2.1 Data Timeliness

Providing operators with data relevant to anomaly diagnosis as quickly as possible only helps speed the fault recovery process. The criticality of a timely recovery is mission and fault dependent. For an anomaly that puts a spacecraft in an unsustainable state (for instance being power negative), quick recovery (at least into a sustainable state) is imperative. For very short duration missions, recovery timeliness is also likely important. The shorter the mission, the more impactful slow fault recoveries are likely to be. But for any mission, time spent performing anomaly recoveries generally takes away time that could have been used to perform other mission objectives.

One method used to help quickly provide data relevant to a fault diagnosis is an anomaly data product priority. This was used extensively in the mechanisms flight software modules on both MSL and Mars 2020. Parameters specifying anomaly data product priorities were provided, and these priorities were generally set to high priority bins. Upon encountering a fault, data products covering the time period of the anomaly would be created at the anomaly downlink priority rather than the usual priority. This allows the anomaly products to “cut in line” and come down before other data. The idea being that recovering from the fault is likely to be the highest priority activity for the next uplink, and therefore it is desirable to get the data necessary for that recovery as soon as possible. Best case, this can save the ground-in-the-loop cycle that operators would need in order to perform a data product management activity to retrieve desired data from lower priority bins.

5.2.2 Unrecoverable Data

Another important fault consideration is preserving unrecoverable data that could be useful for anomaly diagnosis. This might include software or hardware state that is not usually telemetered, or could

include higher rate recording of standard data. Often this is implemented through ring buffers that continuously record important state. These buffers are sized to hold a small set of recent history and older data is continuously overwritten. When a fault is encountered, the contents of these buffers are transferred to a data product for downlink.

5.2.3 Detailed Diagnostics

Some anomalies are easier to diagnose than others. Even if detailed data covering the anomaly is immediately available, there are a certain class of anomalies that cannot be quickly resolved. These are often the result of hardware degradation or failure. Identifying the exact failure mode usually entails performing additional tests on the suspect hardware to narrow down root cause. Operators are often looking for any insight they can get, and may be interested in telemetry that is not generally collected or collecting at rates much higher than usual. This situation should be considered when architecting the data products that will be provided. If a hardware measurement can be made by the system, it's generally a good idea to have some way to telemeter the results even if they are uninteresting in the nominal case.

6 Case Studies

6.1 Mars 2020 Motor Telemetry

6.1.1 Background

This case study focuses on Mars 2020, however the motor control software architecture used on Mars 2020 was inherited from MSL and the two are extremely similar. Figure 1 shows the subset of flight software modules involved in controlling the actuators attached to the Rover Motor Control Assembly (RMCA). The arrow direction indicates command flow. Telemetry flows in the opposite direction. The mechanism modules shown in blue implement high level motion control behaviors specific to a given actuator group. For instance, the ARM module implements Cartesian trajectories used to move the end effector in a straight line. All of these mechanism modules make requests to the MOT module in order to move actuators. The MOT module provides generic interfaces used to command actuator motion. This includes features like coordinated motion (moving a requested set of actuators such that all actuators start and end motion at the same time) which are implemented by MOT. The MOT module sends these motion requests down to the MCA module. MCA is responsible for generating and decoding the 1553 traffic required to communicate with the RMCA. Finally, the BCMGR module is responsible for managing the 1553 bus and dispatching requests to the RMCA where the low-level, high-rate motor control is performed. The 1553 bus rate is 64 Hz, and thus this limits the control rate that can be achieved by software running on the Rover Compute Element (RCE).

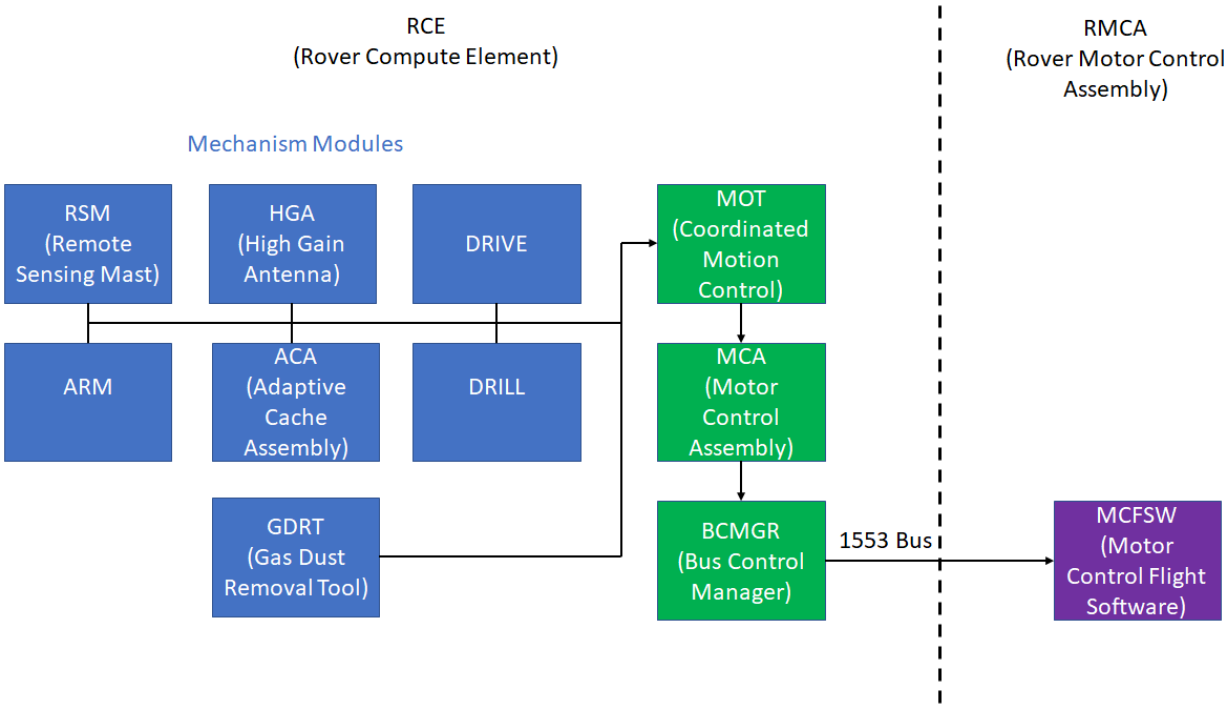


Figure 1: Mars 2020 Motor Control Module Context Diagram

All motion requests for a given actuator flow through its associated mechanism module (i.e. those modules "own" their actuators). This can be through commands dispatched directly to that module, or through requests from higher level modules (for instance the modules responsible for managing the mast mounted imagers will make a request to the RSM module to point the camera head before capturing an image). The exception to this is a set of low-level commands dispatched directly to the MOT module which can be used to move any actuator. These commands are generally used for diagnostic data collection and in off-nominal situations where standard commanding is insufficient.

In addition to standard motor telemetry like encoder-based motor positions and current measurements, there are a few other classes of sensors connected to the RMCA. These include resolvers used to measure the positions of both active (like the arm actuators) and passive joints (like the rover suspension), contact switches, and force sensors. These sensors can be monitored during motion requests and used for control and fault protection.

6.1.2 Standard Motor Telemetry

The mechanism modules shoulder most of the responsibility for creation and management of data products that report motor related telemetry. This includes fairly generic actuator telemetry as well as mechanism specific telemetry (like arm force torque sensor data). Of course telemetry reporting for low-level commands dispatched directly to MOT is handled by the MOT module. The system is architected such that the MOT module can write generic actuator and sensor telemetry to the mechanism module owned data product buffers during motion. This set of standardized telemetry is common across actuators and sensors of a given type. This data collection infrastructure is used by all mechanism modules, as well as MOT, to record basic actuator and sensor telemetry. This architecture has several nice features.

- Less code duplication and implementation effort. Instead of every mechanism module defining its own set of actuator and sensor telemetry, and developing code to extract and write out that telemetry, that code was written once and contained in the MOT module.
- Consistency. Instead of having different data reported for each mechanism group, with various data resolution and data product field naming and ordering, a consistent set of telemetry is generated across the system. This makes the system easier to manage and understand.
- Data product processing is easier. Developing ground tools to process and display actuator telemetry is streamlined. A custom tool for each mechanism group and data product type is unnecessary. Because much of the motor telemetry reported across the system has the same format, it can be processed using the same tools and libraries.

Motor telemetry provided by MOT for actuators with encoders is shown in Table 1. Clients can choose between two different records. Verbose records include all of the fields shown in Table 1 for a total of 40 bytes per sample (the data below plus a 4 byte record header). Brief records contain a subset of the data for a total of 24 bytes per sample.

Field	Type	Units	Description	Included in Brief Record
mid	8-bit unsigned integer		Motor ID	Yes
field	8-bit unsigned integer		Field position register	No
state	8-bit unsigned integer		Motor/brake controller state	Yes
stopmode	8-bit unsigned integer		Stop/fault response mode	No
cstatus	16-bit unsigned integer		Current motor status	Yes
tripped	16-bit unsigned integer		Bitwise-OR of tripped safety checks	Yes
enc_count	32-bit signed integer	Counts	Encoder odometer	Yes
pos	32-bit float	Radians	Joint position	Yes
rate	32-bit float	Radians per second	Joint rate	No
cmotor	16-bit unsigned integer	Milliamps	Motor current	Yes
cbus	16-bit unsigned integer	Milliamps	Bus current	Yes
vavg	16-bit signed integer	Millivolts	Motor voltage	Yes
tprt1	16-bit signed integer	Centi-degrees Celsius	Temperature measured by PRT 1	No
tprt2	16-bit signed integer	Centi-degrees Celsius	Temperature measured by PRT 2	No
cbrake	16-bit signed integer		Brake current (raw DN)	No
pwmbrake	16-bit unsigned integer		Brake duty cycle (raw DN)	No
home_err	16-bit signed integer		Accumulated home pulse errors	No

Table 1: Mars 2020 Motor Telemetry Records

Motor telemetry natively comes across the 1553 bus at 64 Hz making that the highest data collection rate possible. MOT provides clients the ability to throttle the data collection rate by specifying the number of samples to skip between writing out data. Setting this option to zero means all sample are collected. Using a value of three means that 16 Hz data would be collected (one sample saved and three

discarded). Client modules generally expose the choice between brief and verbose records as well as the data collection rate to the operators though parameters.

The data fields in these records are designed with data efficiency in mind since these records are likely to be recorded many times per second during motor actuation. Special attention was paid to the width of each field. Fields that could fit within a single byte were allocated only that byte. For instance, there are 46 motors managed by the RMCA and thus the mid field easily fits within a single byte. The tripped field is a bit-packed reporting of fifteen different safety checks, more than one of which may be triggered at a given time. Bit-packing saves significant data volume in comparison to allocating a byte for each of the individual safety checks. Other fields that might traditionally be reported as floating point values (temperatures, currents, and voltages) are stored as 16-bit integers rather than 32-bit floats to save data volume. Rather than simply rounding the values to the nearest integer, they are first scaled to fill the 16 bits available and thus preserve reasonable precision. For instance note that the tprt1 and tprt2 values are reported in centi-degrees C, the native value having been multiplied by one hundred before reporting.

6.1.3 High Rate Motor Telemetry

The motor control flight software (MCFSW) running on the RMCA is responsible for the low-level motor control functions. MCFSW is responsible for gathering motor telemetry at its 512 Hz execution rate, converting it to 64 Hz telemetry (generally by averaging), and providing it to the RCE across the 1553 bus. This telemetry rate conversion is necessary because the 1553 bus runs at 64 Hz. This limits the RCE control rate and thus telemetry rate used by control algorithms does not need to be any higher. In addition, the 1553 bus bandwidth is insufficient to support real-time 512 Hz telemetry transmission in the general case. That being said, there are cases where this very high rate telemetry is extremely useful. The system provides a couple different ways for this data to be recorded and telemetered.

6.1.3.1 Detailed Motor Telemetry Recording

One way of recording motor telemetry at rates above 64 Hz is through the detailed telemetry recording functionality (or DET recording for short). This service is provided by the MCA module and works as follows. The operator configures a set of parameters that specifies the actuators for which to record higher rate telemetry, options for what data to recorded, as well as the desired recording rate. The recording rate is specified as the number of samples to skip between each sample recorded. This is same method used to manage the recording rate for standard motor telemetry, but in this case the base rate is 512 Hz rather than 64 Hz data. A skip of zero indicates collection of 512 Hz data and one would yield 256 Hz data.

Once the desired parameters have been configured, the MCA_DET_RECORD command is used to signal that the detailed telemetry should now be collected. For any subsequent commands that move actuators for which detailed recording has been configured, MCA will request that MCFSW send back that additional telemetry during motion. MCA then records this additional data into a special buffer. This buffer is split into a log section (recording stops when the buffer fills up) and a ring section (new data overwrites old when the buffer fills). How much of the buffer is allocated to the log section versus the ring is user configurable. Until one of these sections fills up, they both contain identical data. For motions short enough that all the additional high rate telemetry can be recorded, the buffer is generally configured to be either 100% log or 100% ring (either of which will give equivalent behavior until the buffer is filled). For longer motions that cannot be captured in their entirety, the user can decide

whether the beginning or end of motion is of more interest. Increasing the log percent will cover more of the start of motion and increasing the ring percent will cover more of the end of motion.

The MCA_DET_RECORD command does not actually generate any data products. It simply initiates the data logging process. The internal buffer storing the extra data is written out to a data product by the MCA_DET_DMP command. This command also turns off the additional telemetry collection.

One nicety of this implementation is that it enables very high rate motor telemetry to be recorded for any RMCA controlled actuator without any implementation overhead in the many mechanism modules. Because this feature is implemented in the low-level MCA module, the higher level modules don't even know whether or not the recording is being done, and don't have to implement any sort of support for it.

DET recording does not provide many frills. It does not support autonomously dumping data products when the recording buffer fills up. It is up to the operator sprinkle the MCA_DET_RECORD and MCA_DET_DMP commands in the appropriate places to avoid loss of data when recording across multiple motions that will fill the DET buffer. It is also up to the operator to be aware of 1553 bandwidth limitations and ensure the additional telemetry requested does not exceed the available bandwidth. Finally, the buffer used to record the additional telemetry is shared with the anomaly ring readout (described in section 6.1.3.2 below). Therefore both of these capabilities cannot be used simultaneously. If an anomaly ring readout is in progress and the MCA_DET_RECORD command is sent, the MCA_DET_RECORD command simply fails. It would be far better if it just waited for the anomaly readout to finish and then starting the recording. The main reason these limitations exist is driven by the fact that DET recording was designed to facilitate special case diagnostics. It is not used all that frequently (although it is very useful in the niche it serves), and thus spending additional implementation time to make the commanding more operable was deemed low priority.

6.1.3.2 Anomaly Ring Readout

Autonomous anomaly ring readout is an extremely useful capability for the diagnosis of motor related faults. Whenever actuators are being moved, MCFSW writes 512 Hz motion history data into internal ring buffers. These buffers are sized to store approximately the last ten seconds worth of motion for each actuator. In the event that a motor related fault occurs, the MCA module requests that the contents of these buffers be transferred to the RCE. It can take several minutes to read the data across the 1553 bus, but once the data has been transferred, it is written out to a high priority data product. This means it should be very quickly available for operators to inspect and use formulate recovery plans. This autonomous ring readout captures data that would be otherwise unrecoverable, and also has a high probability of containing data of interest.

6.1.3.3 MSL Drill Feed Brake Anomaly Example

On sol 1536 the MSL rover's drill feed mechanism experienced a motor stall when attempting to extend the bit toward the surface for drilling. A many month investigation ensued which made extensive use of high rate DET recording (see ISA 60503 for more details). Figures 2 and 3 show plots of 512 Hz motor telemetry collected during a diagnostic drill feed motion. These plots show the motor current and motor rate as a function of position of the motor (before the gear train). The DN values around the perimeter of the plot are the motor encoder count (there are 144 counts per motor revolution).

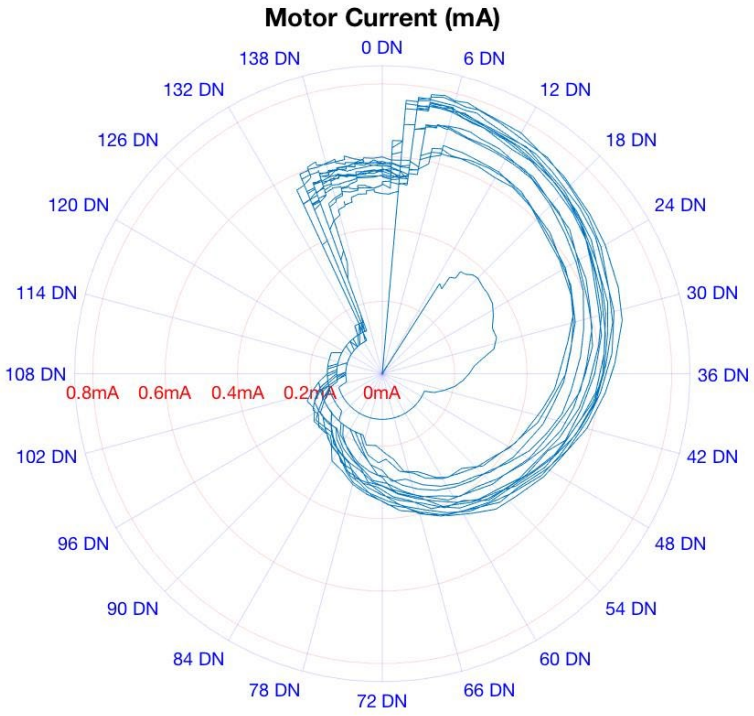
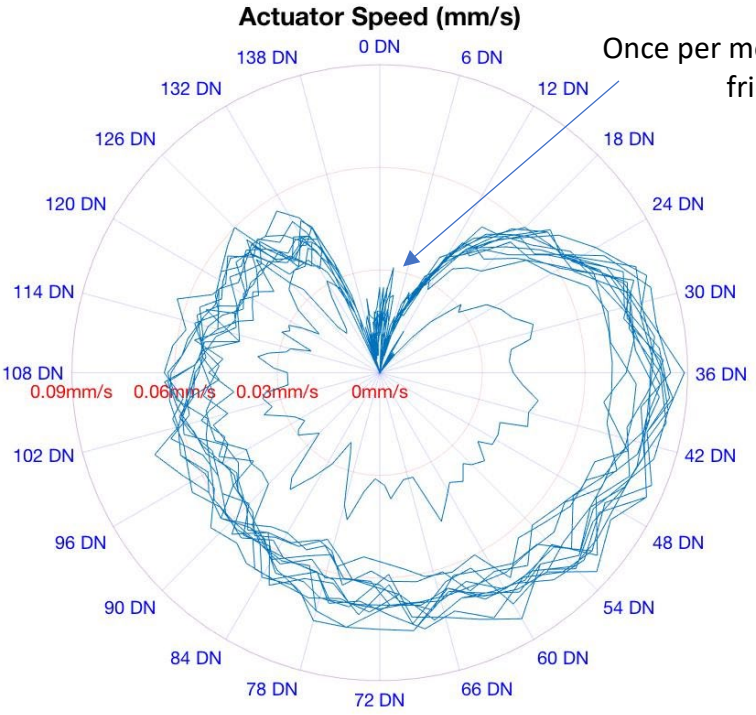


Figure 2: High Rate MSL Drill Feed Diagnostic Move Motor Current



Reviewed and determined not to contain CUI.

Figure 3: High Rate MSL Drill Feed Diagnostic Move Motor Rate

Looking carefully at Figure 3, an actuator speed of 0.06 mm/s at the output of the gear train is 1350 DN/s at the motor. At this motor rate, 512 Hz data collection provides one sample approximately every 2.6 DN (6.6 degrees of motor rotation per sample). At this sampling rate, the dramatic one per motor revolution slowdown is very easy to see. Without using DET recording, the maximum data collection rate would be 64 Hz, which is one sample every 21 DN (52.7 degrees of motor rotation per sample). The standard nominal data collection rate is often set to 8 Hz which would provide less than a single sample per motor revolution. Lower telemetry collection rates lead to more aliasing and would make picking out the underlying signature much more challenging for this anomaly investigation.

6.1.4 Rover Compute Element Warm Reset

One of the most feared motor control fault cases is a reboot of the rover compute element (RCE) during mechanism motion. The flight software implementation contains literally thousands of state assertion checks. If any of these checks fails, the RCE is rebooted. Bugs in these assertions are one way in which the RCE might reset during motion (see MSL ISA 212950). There are a wide variety of other events that might trigger an RCE warm reset as well.

An RCE reset in the middle of mechanism motion means that the RCE will no longer be putting motor control commands onto the 1553 bus and will not be recording any of the motor telemetry the RMCA places onto the bus. On the first 64 Hz control cycle where the RCE fails to communicate with the RMCA, the RMCA will initiate a fault response that brings all actuators to a stop. This prevents any runaway motion due to the fact that the high-level control system is no longer in the picture. The discussion here focuses on strategies designed to ensure that ground operators have motor telemetry in the period leading up to, and following the warm reset event.

6.1.4.1 History Buffer RAM Readout

Rather than writing motion history telemetry directly to the non-volatile file system during motion, the data is instead written into a buffer in RAM. Once the motion has completed, the data in that RAM buffer is written out to a data product on the non-volatile file system. RAM access is much faster than accessing the file system directly, so this strategy supports higher data collection rates, and isolates any performance issues of the onboard file system such that they do not impact the high rate control loop. However, in the event of an RCE reset, any data stored only in RAM is at risk of being lost. In the event of a warm RCE reset, it is possible to potentially recover motion history telemetry. During a warm reset, the RCE remains powered throughout the reset event. This means that data stored in volatile RAM is not actually forgotten. This is in contrast to a cold reset where the RCE is powered off before rebooting. In the case of a cold reset, data in RAM is very unlikely to be retained. Luckily most of the unexpected RCE reset events are very likely to be of the warm variety.

Mars 2020 uses the following strategy to attempt to recover telemetry after a warm reset event. First, the motion history data is always written to RAM buffers at fixed memory addresses. Placing the data at fixed addresses means that the location where that data should reside is always known. Second, as part of the boot behavior after a warm reset event, the data in these addresses is examined. If the data there matches the format of motion history records, that data is written out to non-volatile data products. This enables motion history telemetry to be retained even in the event of an unexpected warm reset event.

6.1.4.2 RMCA Data Transfer

While recovery of the motion history telemetry stored on the RCE in the event of an RCE warm reset is useful, the data stored in those motion histories will be missing one of the most critical time periods. As soon as the reset event occurs, obviously no additional motion history data will be recorded by the RCE. This means the very end of the motion will be missing, which records the final position of the mechanisms that were in motion at the time of the reset. Luckily the RMCA should still be functioning nominally throughout the RCE reset event. The RMCA brings the motors to a graceful stop and tracks the final mechanism positions.

One of the actions that occurs during RCE flight software initialization is to check the power state of the RMCA. During a normal RCE shutdown, the RMCA is always powered off. Therefore, there is no nominal scenario in which the RMCA should be on at RCE boot time. If the RMCA is on, it is assumed that an unexpected RCE reset occurred with motors possibly in motion. This triggers readout of RMCA state, including the internal 512 Hz motion history ring buffers (the buffers discussed in section 6.1.3.2). The data in these ring buffers is then written out to data products. This data will contain the final ramp down and the ultimate positions of all motors that were being actuated during the RCE reset event.

6.1.5 Active Hold

While the majority of the motor control telemetry system was well designed, there were some areas that could have been better. One of these was telemetry recording for active hold. Active hold is a capability that keeps the brakes open and the motor holding position for a parameterized duration after the completion of a motion request. If another motion request is made during the active hold period, then no brake actuation is necessary as the brakes are already disengaged. This capability was designed to reduce the overall number of brake cycles. This functionality was implemented by the MOT module. The higher level mechanism modules could simply finish their requests, and then if configured, MOT would take over and independently hold position for the parameterized period. This architecture was primarily driven by the extensive resource sharing within the motor control avionics. Motor drivers, brake drivers, and sensor channels were all oversubscribed and shared through multiplexing. Managing active hold at the MOT level meant that if a request were to be made for a conflicting actuation (i.e. one that required resources being used for active hold), MOT could simply end the active hold early and immediately service the other request.

In the end, this implementation choice resulted in suboptimal motion history data product management. Standard motion history data products are managed by the higher level mechanism modules (not by MOT). At the end of a motion request, the mechanism module often writes out the motion history data to a data product. This clears out the RAM buffer temporarily storing that data, making room for telemetry from the next motion request. Data cannot both be written to and read from that RAM buffer at the same time. In order for motor telemetry generated during active hold to be written into the standard motion history RAM buffer, there would need be tight coordination between the mechanism modules and MOT so that writing that RAM buffer to data products could be deconflicted with any active hold period. Either that or MOT would have to maintain its own motion history buffers specifically for recording active hold telemetry. Rather than add additional complication, motion history data generated during active hold is simply not recorded at all in the nominal case. If there is a motor fault during active hold, the buffers stored on the RMCA will be written out as usual (see section 6.1.3.2) and will cover the active hold period. But in the nominal case there is no active hold telemetry.

6.1.6 Anomaly Examples

While the exact details of the motor control architecture discussed here are specific to Mars 2020, this design was inherited with from MSL with little modification. Therefore, all of the concepts here are applicable to MSL as well. Table 2 contains a list of motor related Incident, Surprise, Anomaly (ISA) reports form the MSL mission (which has been in operation longer than Mars 2020, and thus had more time to accumulate anomalies). Motor telemetry played an important role in resolving all of these ISAs.

ISA Number	Description
57835	Drill percussion short
59287	CHIMRA tunnel door stall
60503	Drill feed brake anomaly
60832	Drill chuck brake anomaly
203137	Arm turret brake A failure
209103	Arm wrist brake A failure
212950	Fatal during arm hold

Table 2: Selected MSL Motor Related ISAs

6.2 Mars 2020 Vision Compute Element Images

The Mars 2020 rover carried a set of avionics called the Vision Compute Element (VCE). The VCE was designed to perform image processing at high rate. It was designed to be used during entry decent and landing as well as during autonomous navigation on the surface. The interface with the main Rover Compute Element (RCE) was through the 1553 bus, as well as a high-speed serial interface used to transfer image data to the VCE. The VCE generated telemetry which was stored internally, most of which had to be transferred to the RCE for downlink using VCE data management commands. For the surface phase of the mission, the data stored on the VCE included a non-volatile ring buffer configured to hold roughly the most recent 540 images (270 stereo pairs) used by the VCE. This enabled ground operators to retrieve the images corresponding to interesting portions of a drive (for instance regions in which the autonomous navigation system had to avoid hazards).

Specification of the images to be transferred from the VCE and downlinked was done using a relative indexing scheme. The most recently captured image was at index zero, the next most recent image at index 1, and so on. This image identification scheme was useful in some situations, for instance automatically retrieving the image pair likely to show the terrain underneath the rover at the end of a drive. However, this scheme did impose some limitations. Because the index needed to retrieve a given image changed each time a new image was taken, this meant that any images selected by ground operators had to be retrieved before doing any activities that would add new images to the VCE buffer (effectively any mobility activities). Images could be transferred from the VCE at a rate of about four stereo pairs per minute, so the duration of image retrieval could be non-trivial. There were cases where the flexibility to perform the image retrieval after a subsequent drive would have been useful. This could have been fairly easily supported by also allowing images to be referenced by an unchanging absolute index into the ring buffer. To be fair, this image ring buffer was a bonus capability that was not in the baseline specification, making it understandable that the interfaces for it were fairly rudimentary.

6.3 Acknowledgements

The research was carried out at the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration (80NM0018D0004).