

Controlling Precision Stepper Motors in Flight using (Almost) No Parts

David Randall
Jet Propulsion Laboratory
California Institute of Technology
4800 Oak Grove Drive
Pasadena, California 91109-8099
(818) 393-4370
dpr@jpl.nasa.gov

Abstract—This concept allows control of high-performance stepper motors with minimal parts count and minimal flight software complexity. Although it uses a small number of common flight-qualified parts and simple control algorithms, it is capable enough to meet demanding system requirements. Its programmable nature makes it trivial to implement changes to control algorithms both during integration & test and in flight. Enhancements such as microstepping, half stepping, back-emf compensation, and jitter reduction can be tailored to the requirements of a large variety of stepper motor based applications including filter wheels, focus mechanisms, antenna tracking subsystems, pointing and mobility. The hardware design (using an H-bridge motor controller IC) was adapted from JPL’s MER mission, still operating on Mars. This concept has been fully developed and incorporated into the MCS instrument on MRO, currently operating in Mars orbit. It has been incorporated into the filter wheel mechanism and linear stage (focus) mechanism for the AMT instrument. On MCS/MRO, two of these circuits control the elevation and azimuth of the MCS telescope/radiometer assembly, allowing the instrument to continuously monitor the limb of the Martian atmosphere. Implementation on MCS/MRO resulted in a 4:1 reduction in the volume and mass required for the motor driver electronics (100:25 square inches of PCB space), producing a very compact instrument. In fact, all of the electronics for the MCS instrument are packaged within the movable instrument structure. It also saved approximately 3 Watts of power. Most importantly, the design enabled MCS to meet very its stringent maximum allowable torque disturbance requirements.¹²

TABLE OF CONTENTS

| | |
|--|-----------|
| 1. BACKGROUND | 1 |
| 2. DESIGN CHALLENGES | 2 |
| 3. FLIGHT HARDWARE | 3 |
| 4. FLIGHT SOFTWARE | 5 |
| 5. ACCELERATION | 5 |
| 6. MICROSTEPPING | 6 |
| 7. IMPROVING PERFORMANCE | 8 |
| 8. PROGRAMMING THE SYSTEM | 9 |
| 9. CONCLUSION | 10 |
| ACKNOWLEDGEMENTS | 10 |
| REFERENCES | 10 |
| BIOGRAPHY | 10 |

1. BACKGROUND

Spacecraft carry instruments. Instruments have mechanisms. Controlling these mechanisms without affecting either the operation of the instrument or the stability of the spacecraft can sometimes be challenging.

The Mars Climate Sounder (MCS) on the Mars Reconnaissance Orbiter (MRO) posed such a challenge. While relatively small (9.8 kg), it contains two actuators that allow the entire instrument to scan continuously while it orbits Mars. Data collection occurs at two-second intervals, is also continuous, and has to be synchronized with actuator motion starts and stops. Stringent requirements were placed on the instrument with regard to the amount of torque it could impart to the spacecraft.

¹ 978-1-4244-3888-4/10/\$25.00 ©2010 IEEE

² IEEEAC paper #1065, version 2, updated Friday, November 13, 2009

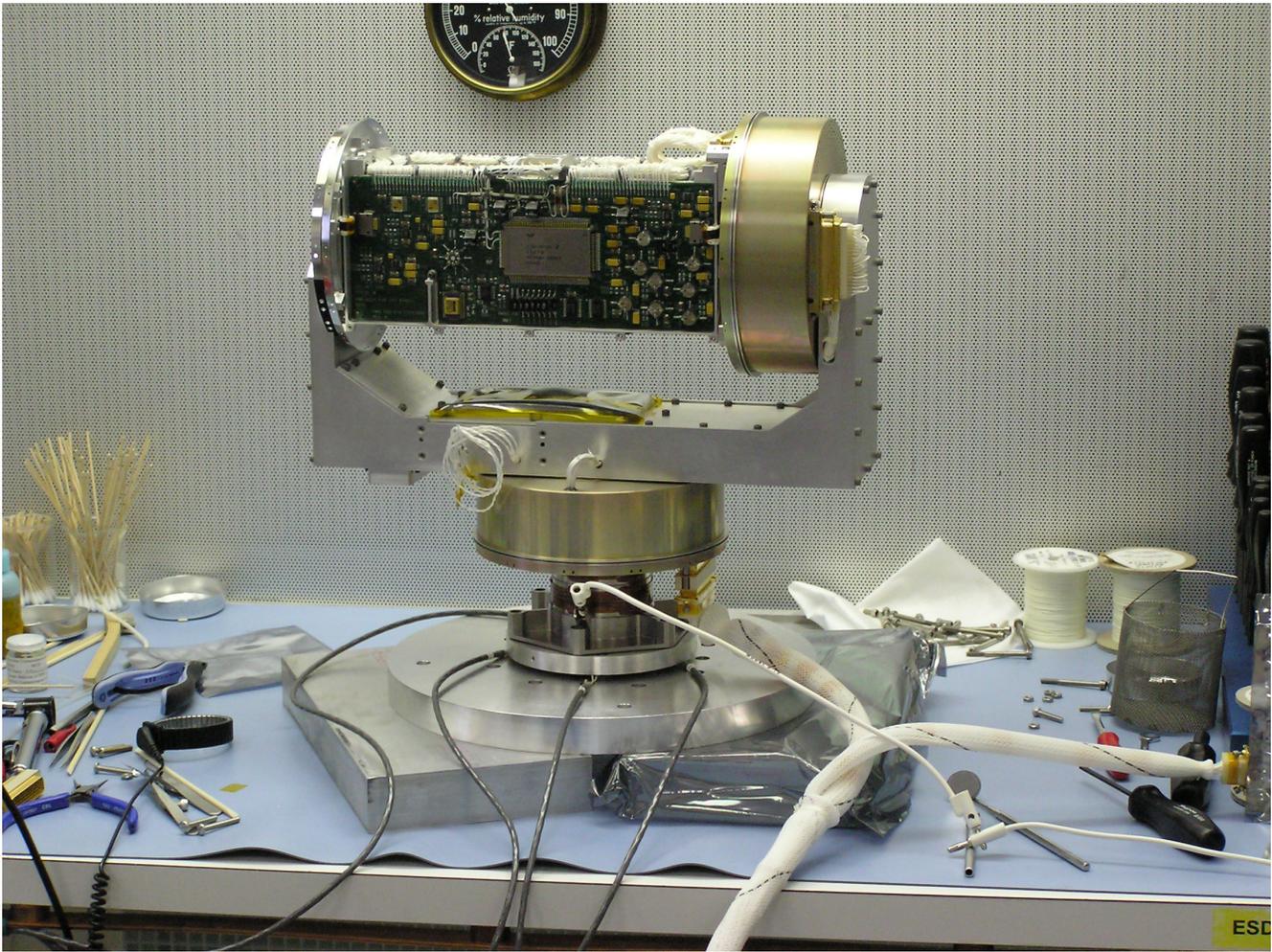


Figure 1 - The MCS instrument, with thermal blanketing, telescope baffles, and covers removed

In figure 1, the MCS instrument is undergoing azimuth torque disturbance testing. The azimuth actuator (bottom) rotates the yoke (u-shaped structure). The yoke holds the elevation actuator (right side) that rotates the radiometer (rectangular structure).

2. DESIGN CHALLENGES

To obtain maximum system performance, it is desirable to use the actuators to slew the MCS instrument as fast as possible, maintaining our torque margins without violating torque disturbance requirements. Of course, we also want to use as little power as possible. These design challenges fall into three broad categories.

Bulk (DC) torque requirements:

- o The actuators have to be accelerated and decelerated at the maximum rates that do not violate the spacecraft's torque requirements. The azimuth maximum allowable torque disturbance is 0.04 Nm. The elevation maximum allowable torque disturbance is 0.02 Nm.

- o The maximum allowable rates are different for the elevation actuator (which only rotates the 4.2 kg radiometer) and the azimuth actuator (which rotates 7.7 kg, including the radiometer, yoke, and elevation actuator).

- o The actuator movements are arbitrary, from 1 step to 1,390 steps. The algorithms have to work regardless of the number of steps.

- o The actuators have a top speed that is less than the speed we could attain in $\frac{1}{2}$ the total travel distance. If the motion is long enough, the actuator has to stop accelerating and run at top speed until it is time to decelerate.

Dynamic (AC) torque requirements:

- o Stepper motors have inherent granularity. The MCS actuators have 30° (12-pole) motors, driving a 3.33:1 planetary gearbox, followed by an 89.1:1 harmonic drive. Each step of the motor produces only a 0.1° movement of the instrument. If driven by conventional means, even this seemingly small motion would far exceed the maximum

allowable dynamic torque requirement of 0.005 Nm from 0 to 125 Hz.

- o Microprocessor-driven control circuitry also has inherent granularity. Differing execution times of various parts of the control algorithm will introduce additional jitter. There is also an absolute boundary to the time resolution that can be achieved, based upon the microprocessor clock. MCS has a time resolution of about 1.5 microseconds.

- o If we accelerate through the natural frequency of the system (57 Hz), it will cause the system to ‘ring’, introducing additional jitter.

Power requirements:

- o The actuators are power hungry and operate almost continuously. They need to use +28V spacecraft bus power (solar cells/battery) for the stepper motors rather than MCS regulated secondary power. This avoids the approximate 75% efficiency penalty of the DC/DC converters used to regulate the MCS power. All the control circuitry (that uses MCS regulated secondary power) must be electrically isolated from the stepper motor drivers (that use spacecraft bus power).

- o Spacecraft bus voltage can vary from +22V to +36V. In reality, it is unlikely that the spacecraft would be conducting normal operations (i.e. allowing MCS to move its actuators) if the battery voltage were to drop below 26V (it would be load shedding to save itself). Even so, a 26V to 36V range produces a 2:1 range in voltage required to generate the same amount of current (and power and torque) in the stepper motor windings.

- o The motor winding resistance varies with temperature.

- o As the motor spins, it generates back-emf (electromotive force). The motor acts like a generator, producing a bucking voltage that decreases the apparent voltage applied to the windings.

3. FLIGHT HARDWARE

The original plan was to design constant current drivers for the motors, which would always apply the same amount of power to the motor windings regardless of operating conditions. This posed a lot of problems in terms of parts count, difficulty of isolating the control interface, and circuit complexity, and did not properly address the torque disturbance requirements. A much simpler hardware solution was to drive the motors with adjustable voltage, and design algorithms to adjust the voltage as operating conditions changed to maintain a constant current. A microprocessor, a field-programmable gate array (FPGA) containing some simple logic, and a few parts for the driver circuitry are all that is required.

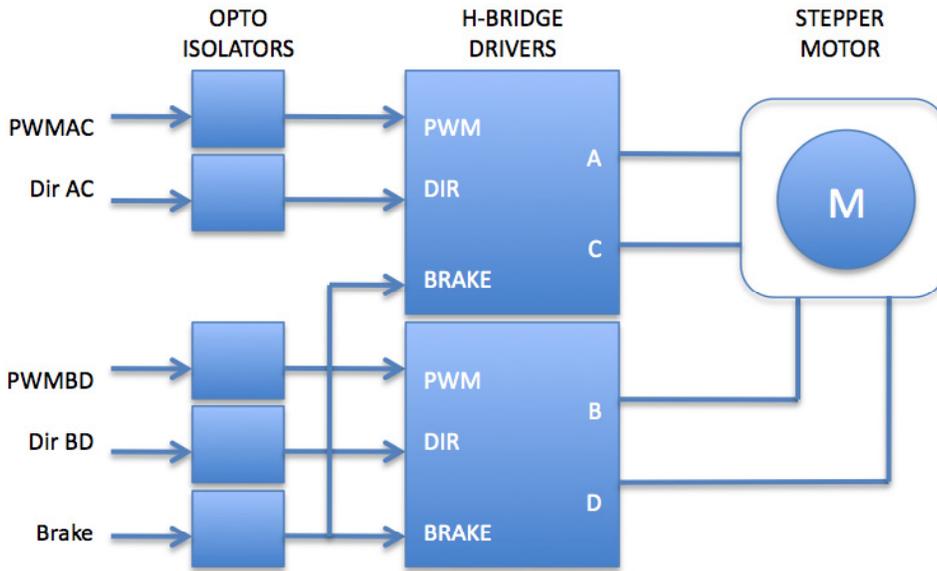


Figure 2 - A block diagram of drive circuitry for one actuator

The control interface in figure 2 is very simple, requiring only five digital control inputs from the FPGA (on the left). Optoisolators are used to isolate MCS (FPGA) power from +28V spacecraft (and motor) power. Brake, PWMAC, and PWMBD control the application of current through the A-C and B-D windings of the bipolar stepper motor in the actuator. DirAC and DirBD set the direction of current through each of the windings.

For each of the motor winding drivers the control inputs work as follows:

- o BRAKE HI and PWM LO: motor winding is disconnected from the driver
- o BRAKE LO and PWM LO: motor winding is shorted (both ends to +28V)

- o BRAKE LO and PWM HI and DIR LO: current flows from C to A (or D to B)
- o BRAKE LO and PWM HI and DIR HI: current flows from A to C (or B to D)

By generating pulse width modulated signals (in the FPGA) for the PWM inputs, the average amount of voltage (and current) applied to each of the windings can be controlled. The direction of current through each winding can also be controlled independently.

The lowest spacecraft bus voltage at which we are required to operate is 22V. The stepper motor winding resistance is directly proportional to temperature. As the motor spins, it generates back-emf. The stepper motor must be selected such that 22V applied to the windings at the maximum allowable flight temperature while stepping at the maximum rate (i.e. the worst case) gives sufficient torque margin.

4. FLIGHT SOFTWARE

| MCS FPGA Memory-Mapped I/O | | | | | | | | | | 30-Jan-2004 DPR |
|----------------------------|------|---|---|---|---|-------|---|-------|-------|-----------------------------------|
| Gate Array | Addr | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
| direction | 7F01 | | | | | BRAKE | | DirBD | DirAC | brake bit and stepper motor phase |
| PWMAC | 7F12 | | | | | | | | PWMAC | motor voltage for A-C winding |
| PWMBD | 7F13 | | | | | | | | PWMBD | motor voltage for B-D winding |

Figure 3 - The microprocessor interface to the FPGA control circuitry

The interface to flight software (figure 3) is also very simple. The BRAKE, DirAC, and DirBD bits go directly to the driver circuitry. The PWMAC and PWMBD registers each produce a 1-bit output, a 64KHz pulse-width-modulated square wave with a duty cycle of 0/127 to 127/127. The direction register contains DirAC and DirBD as its two least significant bits. Together, these are the stepper phase, which can be incremented or decremented in a grey code sequence to cause movement:

00, 01, 11, 10, 00, ... causes movement in one direction

00, 10, 11, 01, 00, ... causes movement in the other direction

The required motor voltage value can be calculated knowing the spacecraft bus voltage, stepper motor winding temperature, and step rate (which determines the amount of back-emf). Then, by bringing BRAKE LO, and writing this calculated value into the PWM registers, we can increment or decrement the direction register at a constant step rate (as described above) to implement conventional stepping.

The motor current can also be monitored real-time, using the current sense output available on the motor driver. Using a comparator, a 1-bit status can be generated (motor current too low/too high). This can be used to adjust the PWM motor voltage values in real-time, if enough processing power is available. This approach was explored, but not used, on MCS (other than to report this status in telemetry).

This circuitry and simple control algorithm minimizes the power used by the stepper motors by using unregulated spacecraft bus power to drive the motor windings, and by adjusting the motor voltage via PWM to always supply the correct calculated motor current and maintain the minimum required torque. However, as stated above, stepping the actuators in this manner would severely violate the maximum allowable torque disturbance requirements of the MCS instrument. Fortunately, the system has a microprocessor, and we can use its capability to implement enhancements to this basic scheme.

5. ACCELERATION

The step rate does not have to be constant. Acceleration can be implemented by continuously decreasing the time interval between changes to the direction register.

Table 1 - Acceleration

| 42 =accel | t time | t dist | t vel | STEPS |
|------------|----------|--------|---------|-------|
| 26.3 =vmax | 0 | 0 | | |
| | 0.069007 | 0.1000 | 2.8970 | 1.00 |
| | 0.097590 | 0.2000 | 4.0983 | 2.00 |
| | 0.119523 | 0.3000 | 5.0194 | 3.00 |
| | 0.138013 | 0.4000 | 5.7962 | 4.00 |
| | 0.154303 | 0.5000 | 6.4871 | 5.00 |
| | 0.169031 | 0.6000 | 7.0990 | 6.00 |
| | 0.182574 | 0.7000 | 7.6678 | 7.00 |
| | 0.195180 | 0.8000 | 8.1973 | 8.00 |
| | 0.207020 | 0.9000 | 8.6946 | 9.00 |
| | 0.218218 | 1.0000 | 9.1744 | 10.00 |
| | 0.228869 | 1.1000 | 9.6122 | 11.00 |
| | 0.239046 | 1.2000 | 10.0396 | 12.00 |
| | 0.248807 | 1.3000 | 10.4496 | 13.00 |
| | 0.258199 | 1.4000 | 10.8441 | 14.00 |
| | 0.267261 | 1.5000 | 11.2248 | 15.00 |
| | 0.276026 | 1.6000 | 11.5929 | 16.00 |
| | 0.284521 | 1.7000 | 11.9497 | 17.00 |
| | 0.292770 | 1.8000 | 12.2962 | 18.00 |
| | 0.300793 | 1.9000 | 12.6332 | 19.00 |
| | 0.308607 | 2.0000 | 12.9814 | 20.00 |
| | 0.316228 | 2.1000 | 13.2811 | 21.00 |
| | ... | | | |
| | 0.624881 | 8.2000 | 26.2449 | 82.00 |
| | 0.628684 | 8.3000 | 26.3000 | 83.00 |
| | 0.632487 | 8.4000 | 26.3000 | 84.00 |
| | 0.636289 | 8.5000 | 26.3000 | 85.00 |

Table 1 shows the elapsed time at each step while accelerating the elevation actuator at 42°/sec/sec to a top speed of 26.3°/sec (at step 83).

Acceleration is implemented in flight software (FSW) by programming time delays into a hardware timer that causes an interrupt. When the interrupt occurs, pre-calculated values for motor voltage and direction are output to the motor, the time delay (until the next interrupt) is programmed into the timer, and new voltage and direction values are calculated (for the next interrupt).

Because back-emf is directly proportional to motor speed, acceleration complicates the calculation of the required motor voltage (PWM) values. Back-emf at each step during acceleration will be different, so the PWM values have to be

adjusted at each step to change the voltage and maintain a constant current through the motor windings.

We also need to decelerate the actuator smoothly, and we need to support an arbitrary number of steps. This is accomplished by breaking each actuator movement into three parts. We use the time delays, moving down through the table, for the first part, acceleration. We continue this until we have reached either the midpoint of the movement or step 83 (where we reach the top speed of the actuator). For short movements (less than or equal to 166 steps), we begin decelerating immediately. For long movements (greater than 166 steps), we run at top speed until it's time to begin decelerating (83 steps before the end of the movement). In either case, we decelerate by moving back up the table (starting where we left off after acceleration) until the movement is complete. The last step is always the same length as the first step.

6. MICROSTEPPING

But, as the saying goes, "Watch out for that first step!" Using the acceleration table, that first step is 69 ms long. Unfortunately, because we have a lot of torque margin in the system, all of the physical motion occurs in the first few milliseconds. The actuator accelerates and decelerates very quickly, and we just wait around at 0 velocity until it's time for the next step.

Using acceleration alone, we still violate the dynamic torque requirement at the beginning and end of each actuator movement, when the step period (1/step rate) is large. Once we accelerate to where the step period decreases to a few milliseconds things smooth out, because the physical movement closely matches the commanded movement (which is theoretically perfect). We need to break the slow steps at the beginning and end of each actuator movement into microsteps, so that the physical movement always closely matches the commanded movement.

A stepper motor can be made to operate smoothly like a brushless DC motor simply by applying analog cosine and sine waves to its windings. We can use our built-in ability to control motor voltage (and current) to do just that. We apply $\frac{1}{4}$ of a cosine wave (in discrete microsteps) to decrease the voltage (from the calculated value to 0) in the currently energized motor winding at a controlled rate, while simultaneously applying $\frac{1}{4}$ of a sine wave to increase the voltage (from 0 to the calculated value) in the other winding. By doing this, the motor armature will smoothly move the 30° from one pole to the next (rather than snapping to there in a few milliseconds), and the instrument (at the other end of the gearbox) will smoothly rotate 0.1° .

Microstepping solves another problem. The natural frequency of the system is 57 Hz. We want to keep the rate of commanded movements well above that (greater than 200 Hz) to avoid causing resonance, so the period between commanded movements should always be less than 5 ms. Microstepping allows this regardless of the step period. On the other hand, the MCS microprocessor is slow. The practical lower limit of command period is about 1 ms, due to the overhead in the interrupt service routine. The only way to keep the command rate within this narrow range while implementing the full range of acceleration is to microstep.

Table 2 - Microstepping

| | | t time | t dist | t vel | STEPS |
|--------|----------|----------|--------|--------|-------|
| 42 | =accel | | | | |
| 26.3 | =vmax | 0 | 0 | 0 | |
| 11.8 | =vinit | 0.003753 | 0.0003 | 0.1576 | 0.00 |
| 0.0033 | =bemf | 0.007507 | 0.0012 | 0.3153 | 0.01 |
| 26.5 | =vbus | 0.011260 | 0.0027 | 0.4729 | 0.03 |
| 0.00% | =detent | 0.015014 | 0.0047 | 0.6306 | 0.04 |
| 0 | =offset | 0.018767 | 0.0074 | 0.7882 | 0.07 |
| | | 0.022521 | 0.0107 | 0.9459 | 0.11 |
| 25 | AZ accel | 0.026274 | 0.0145 | 1.1035 | 0.15 |
| 42 | EL accel | 0.030028 | 0.0189 | 1.2612 | 0.19 |
| | | 0.033781 | 0.0240 | 1.4188 | 0.24 |
| | | 0.037535 | 0.0296 | 1.5765 | 0.30 |
| | | 0.041288 | 0.0358 | 1.7341 | 0.36 |
| | | 0.045042 | 0.0426 | 1.8917 | 0.42 |
| 24 | usteps | 0.048795 | 0.0500 | 2.0510 | 0.50 |
| 12 | usteps | 0.052705 | 0.0583 | 2.2108 | 0.58 |
| 8 | usteps | 0.056344 | 0.0667 | 2.3641 | 0.67 |
| 6 | usteps | 0.059761 | 0.0750 | 2.5080 | 0.75 |
| 4 | usteps | 0.062994 | 0.0833 | 2.6441 | 0.83 |
| 2 | usteps | 0.066069 | 0.0917 | 2.7735 | 0.91 |
| | | 0.069007 | 0.1000 | 2.8970 | 1.00 |
| 186 | f min | 0.071824 | 0.1083 | 3.0155 | 1.09 |
| 555 | f max | 0.074536 | 0.1167 | 3.1295 | 1.17 |
| | | 0.077152 | 0.1250 | 3.2395 | 1.25 |
| | | 0.079682 | 0.1333 | 3.3458 | 1.33 |
| | | 0.082134 | 0.1417 | 3.4489 | 1.42 |
| | | 0.084515 | 0.1500 | 3.5490 | 1.50 |
| | | 0.086831 | 0.1583 | 3.6463 | 1.58 |
| | | 0.089087 | 0.1667 | 3.7411 | 1.66 |
| | | 0.091287 | 0.1750 | 3.8335 | 1.75 |
| | | 0.093435 | 0.1833 | 3.9238 | 1.83 |
| | | 0.095535 | 0.1917 | 4.0120 | 1.92 |
| | | 0.097590 | 0.2000 | 4.0983 | 2.00 |
| | | 0.099602 | 0.2083 | 4.1829 | 2.08 |
| | | 0.101575 | 0.2167 | 4.2658 | 2.17 |
| | | 0.103510 | 0.2250 | 4.3470 | 2.25 |
| | | 0.105409 | 0.2333 | 4.4268 | 2.34 |
| | | 0.107275 | 0.2417 | 4.5052 | 2.42 |
| | | 0.109109 | 0.2500 | 4.5916 | 2.50 |
| | | 0.111803 | 0.2625 | 4.6951 | 2.63 |
| | | 0.114434 | 0.2750 | 4.8056 | 2.75 |
| | | 0.117006 | 0.2875 | 4.9137 | 2.88 |
| | | 0.119523 | 0.3000 | 5.0194 | 3.00 |
| | | 0.121988 | 0.3125 | 5.1230 | 3.12 |
| | | 0.124403 | 0.3250 | 5.2245 | 3.25 |
| | | 0.126773 | 0.3375 | 5.3240 | 3.37 |
| | | 0.129099 | 0.3500 | 5.4217 | 3.50 |
| | | 0.131385 | 0.3625 | 5.5177 | 3.63 |
| | | 0.133631 | 0.3750 | 5.6121 | 3.75 |
| | | 0.135840 | 0.3875 | 5.7049 | 3.88 |
| | | 0.138013 | 0.4000 | 5.7962 | 4.00 |
| | | 0.140153 | 0.4125 | 5.8861 | 4.12 |
| | | 0.142261 | 0.4250 | 5.9746 | 4.25 |
| | | 0.144338 | 0.4375 | 6.0619 | 4.38 |
| | | 0.146385 | 0.4500 | 6.1479 | 4.50 |
| | | 0.148404 | 0.4625 | 6.2327 | 4.62 |
| | | 0.150396 | 0.4750 | 6.3164 | 4.75 |
| | | 0.152362 | 0.4875 | 6.3990 | 4.88 |
| | | 0.154303 | 0.5000 | 6.4871 | 5.00 |
| | | 0.156854 | 0.5167 | 6.5874 | 5.16 |
| | | 0.159364 | 0.5333 | 6.6929 | 5.34 |
| | | 0.161835 | 0.5500 | 6.7967 | 5.50 |
| | | 0.164268 | 0.5667 | 6.8989 | 5.66 |
| | | 0.166667 | 0.5833 | 6.9996 | 5.83 |
| | | 0.169031 | 0.6000 | 7.0990 | 6.00 |

Table 2 shows just the first six steps of elevation actuator movement using acceleration and microstepping. By specifying the acceleration, top speed, initial voltage, back-emf constant, and spacecraft bus voltage, the table is generated automatically.

The first half of the first step (blue) is divided into 12 microsteps. The time between these microsteps is held constant and the distance of each microstep (the voltage values applied PWM1 and PWM2) implement the acceleration. The first movement the actuator makes is a whopping 0.0012°. After 12 microsteps (½ step), the motor armature is positioned halfway between the two motor poles, 49 ms have elapsed, the actuator has moved 0.05°, and its velocity is about 2°/sec.

After step 0.5 the distance between microsteps is held constant, and acceleration is implemented by decreasing the time interval between microsteps. When that time interval approaches 1 ms, the number of microsteps per step is decreased to keep the time intervals between 1 and 5 ms. Steps 0.5 through 2.5 use 12 microsteps/step (green). The number of microsteps per step decreases to 8 at step 2.5 (yellow), to 6 at step 5 (beige), to 4 at step 10, and to 2 at step 20 (where it remains). At step 20, 309 ms have elapsed, the actuator has moved 2°, and its velocity is up to about 13°/sec. The actuator reaches maximum velocity at step 83 (elapsed time 629 ms, distance 8.3°, velocity 26.3°/sec). This will only happen during actuator movements greater than or equal to 16.6°. For movements less than 16.6°, we start decelerating before reaching the end of the table.

Note that the number of microsteps per step is always even. This is so we can reverse direction through the table at a half-step boundary, to begin deceleration even when the number of steps in the actuator movement is odd. Note also that the actual PWM1 and PWM2 voltage values are assigned to either the PWMAC register or the PWMBD register, depending on the current motor phase.

Two other control features are included in the actuator table generator. A detent percentage was implemented that will move the microsteps adjacent to the integral step boundaries further away (earlier or later) from those points in time. An offset was incorporated that can be added to all the PWM1 and PWM2 values that are non-zero. These features were explored in an attempt to remove the small amount of jitter that occurs at the step boundaries, related to the detent torque built into the motor. Neither of these features made a significant impact on torque disturbance. They were set to 0 when the final versions of the MCS actuator tables were generated.

Commanding the actuator to move one single step is very different when microstepping is used. Instead of issuing one commanded movement (snapping the actuator to the next phase) and waiting 69 ms, it now requires 12 commanded microsteps (12 to accelerate to step 0.5, 12 to decelerate to step 1) and 98ms to complete that single step.

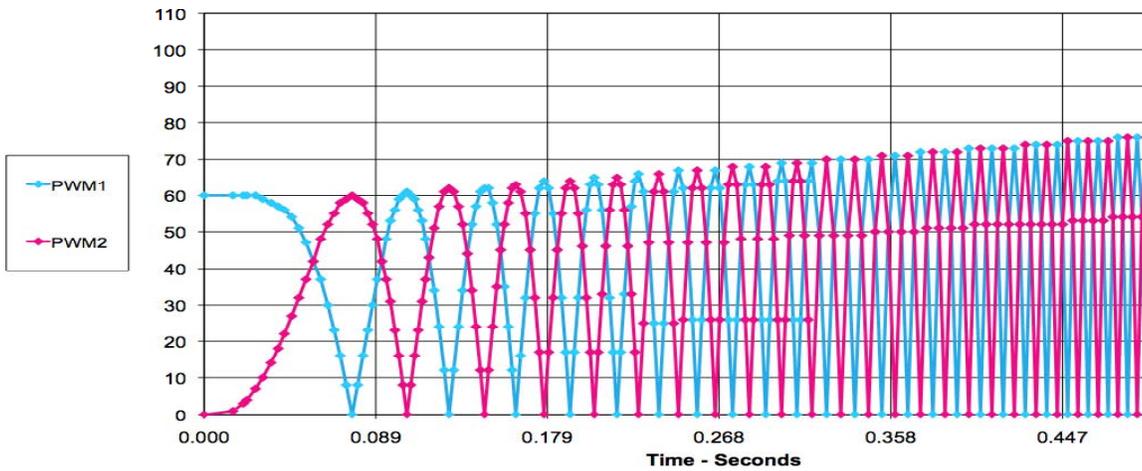


Figure 4 - Calculated MCS PWM1 and PWM2 values

Figure 4 is a chart of calculated MCS PWM1 and PWM2 voltage values v.s. time. You can see the number of microsteps per step decreasing from 24 to 2 (at step 20), and the applied voltage increasing (to compensate for back-emf).

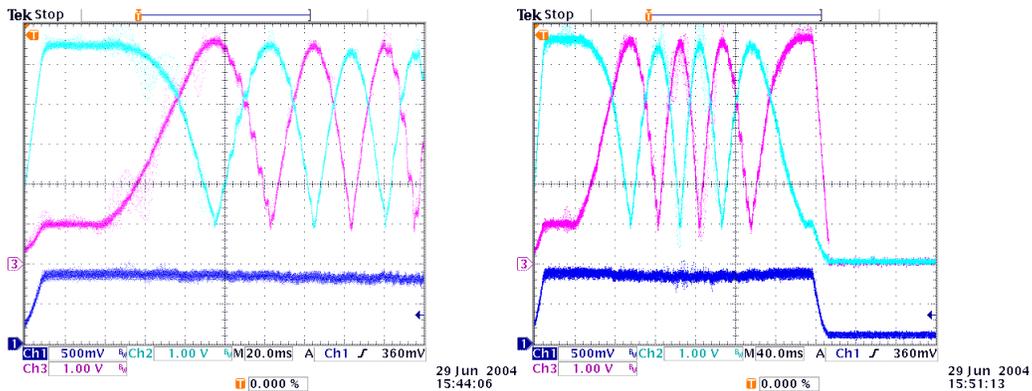


Figure 5 - Oscilloscope photos

The oscilloscope photos in figure 5 show the actual motor current applied to the MCS elevation actuator. Compare the photo on the left to the chart above. Note that, although the applied voltage increases with the number of steps, the measured current (bottom trace) remains constant. On the right is a 7-step elevation actuator movement, showing both acceleration and deceleration.

In the MCS flight software (FSW), the time between microsteps is determined by a hardware counter that causes an interrupt. In the interrupt service routine (ISR), it's important to always have a constant time (number of machine cycles) to when the hardware registers are loaded with values to implement the next microstep, and to when the timer is initialized for the next timing interval. Otherwise, the control algorithm will introduce additional jitter into the system. It is also important to subtract the number of machine cycles between the occurrence of the interrupt and the writing of the next time interval into the timer, from the value of the next time interval written. This is so we don't inadvertently add the overhead of the ISR to each time interval.

7. IMPROVING PERFORMANCE

So now we should have a smooth, jitterless system that meets all requirements. Unfortunately, it is a digital system and things are quantized, which still introduces error that causes jitter... enough jitter so that we still don't meet our maximum allowable dynamic torque requirement.

The problem is in the PWM generators. We can't increase the voltage resolution to more than 7 bits (0/127 to 127/127 of spacecraft bus voltage), due to frequency constraints on the driver (at the high end) and the motor (at the low end). Also, a large part of that 7-bit resolution is used to compensate for spacecraft bus voltage and back-emf. The

truth is, our digital cosine and sine waves at low velocities have a lot of distortion (about 3%).

However, we do have a lot of unused time resolution, especially at low velocities near the beginning and end of the actuator movement, which we can use to compensate for this. By adjusting the time intervals between the microsteps, we can move the discrete points of the digital

cosine and sine waves (which have error in the vertical, voltage direction) horizontally (in the time direction) to place them back on the theoretical cosine and sine curves. Our resolution in the time domain is about 0.1%. This technique yields a marked improvement in jitter at low velocities, and finally allows the system to meet all torque disturbance requirements.

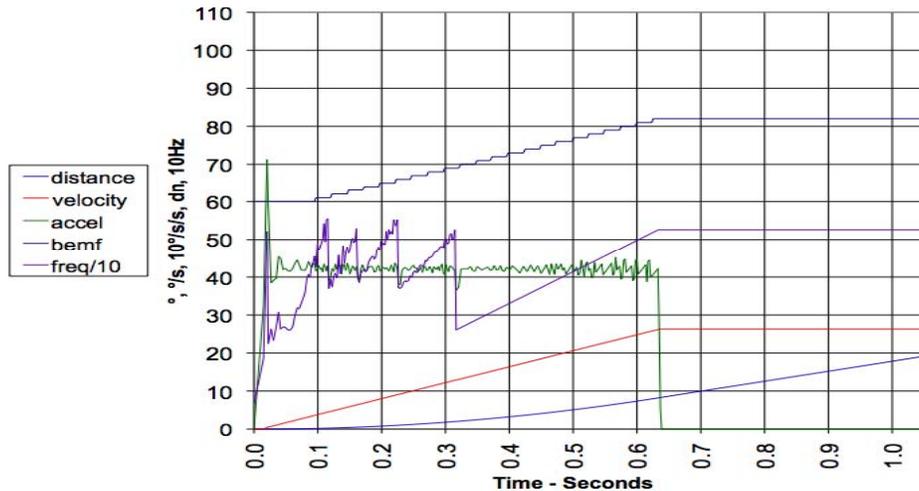


Figure 6 – Distance, velocity, acceleration, back-emf, and frequency during acceleration

In figure 6 (after an artifact at the first microstep), we can see that the acceleration stays constant (with some residual jitter) at 42°/sec/sec until step 83, and that the command frequency always remains above 200 Hz.

8. PROGRAMMING THE SYSTEM

Bear in mind that, for MCS, all the ornate calculations described above are done on the ground. Two acceleration tables are generated (one for elevation, one for azimuth) to be used by the MCS flight software. Each entry in these tables is a PWM1 value, a PWM2 value, and the time delay (in machine cycles) for that microstep. The MCS FSW routine is simple and table-driven. The only real-time in-flight decisions are how to assign PWM1 and PWM2 to PWMAC and PWMBD based on the beginning motor phase, and how to navigate the table based on the number of steps desired in the actuator movement.

In practice, the stepper motor winding temperature is not used in the calculation of the MCS actuator voltages (this would require real-time in-flight calculations). The decision was made to ignore the winding temperature. Because the winding resistance is directly proportional to temperature, this has the advantage of automatically increasing the torque when the actuator (and its bearings, gears, and grease) is cold. The MRO spacecraft bus voltage is very predictable. This is set as a constant when calculating the actuator tables. Should the bus voltage change significantly during the mission, new tables could be generated and uploaded to compensate for the change.

MCS had an anomaly while orbiting Mars. After many months of error-free operation (since initial power on), the elevation actuator reported three (recoverable) position errors, which caused the instrument to safe. The most probable cause was a piece of debris in the planetary gearbox that caused the actuator to stall temporarily and slip 4 steps. The anomaly was intermittent, but persistent... it would disappear for weeks but, when it occurred, errors were spaced at angular distances corresponding to the number of teeth in one of the planetary gears.

Because of the completely programmable nature of this controller architecture, it was a simple matter to generate and upload new actuator tables that increase the torque applied to the actuators. Several different tables were uploaded and used in the investigation of the anomaly. MCS is now operating reliably, using tables with slightly increased torque and a different back-emf constant that further increases the torque at slower velocities where it seems (by experience) to be needed most.

9. CONCLUSION

This controller architecture has been used on several flight projects since MCS with similar, outstanding results. Although it uses a small number of common flight-qualified parts and simple control algorithms, it is capable enough to meet demanding system requirements. Its programmable nature makes it trivial to implement changes to control algorithms both during integration & test and in flight.

ACKNOWLEDGEMENTS

The research described in this paper was carried out by the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration.

Alan Mazer is a Principal Software Engineer at JPL. Alan coded and was responsible for MCS flight software.

David Hykes is a Senior Electronics Engineer at JPL. David provided the hardware design for the MCS motor drivers.

REFERENCES

- [1] MRO/MCS Web site http://mars.jpl.nasa.gov/mro/mission/sc_instru_mcs.html/
- [2] Aerospace Conference Web site <http://www.aeroconf.org/>

BIOGRAPHY



David Randall is a Principal Electronics Engineer, working for Caltech at JPL, NASA's Jet Propulsion Laboratory. David is responsible for the design and delivery of flight electronics for instrument systems. He was the Project Element Manager for MCS electronics, and has recently delivered flight electronics for the CheMin (chemistry and mineralogy) instrument on the Mars Science Laboratory (MSL) rover, scheduled for launch in 2011.