Improving real-time communication between host and motion system in a HWIL simulation

Howard S. Havlicsek, Larry Zana

Acutronic USA, Inc., 640 Alpha Dr., Pittsburgh, PA 15238

ABSTRACT

The effectiveness of a HWIL facility for developing missile guidance and targeting systems is limited by the quality of the elements that simulate the continuous physical processes. A motion simulator, which stimulates the inertial measurement and targeting sensors, must produce motion that is consistent with the actual physical processes. The effectiveness of a HWIL simulation is progressively degraded by each non-ideal element or process in the loop. The interface between the simulation computer and the motion system is traditionally a problematic link that is resolved once and for all in the Acutrol3000 Motion Control instrumentation.

This paper focuses on issues relating to data synchronization, time skew correction, multi-rate data smoothing, and reduced state motion vectors. Concepts are addressed and results are presented.

Keywords: HWIL, motion simulator, real-time data, reflective memory

1.0 INTRODUCTION

Hardware-in-the-loop simulation as a concept has evolved over the years and continues to be a crucial development/test environment for the cost effective deployment of flight vehicle and target engagement systems. Figure 1 shows a typical HWIL architecture for simulating missile flight dynamics. Three principal components of the closed loop system are the missile or components of the missile, the simulation computer, and the physical motion system, a.k.a. the motion simulator or flight table.

The motion system in Figure 1 consists of the physical simulator and the motion controller; however, from the viewpoint of the HWIL simulation specialist, the motion system is ideally viewed as a black box. In practice this box is not so black, and may require calibration and/or accommodation of the non-ideal behaviors. Recent Acutronic papers, "Improvements to Transient Fidelity of HWIL Flight Tables Using Acceleration Feedback"[1] and "Improvements in Flight Table Dynamic Transparency for Hardware-in-the-Loop Facilities"[2] address performance issues related to closed loop control of a flight table. This paper attempts to expand on the topic of motion system performance by addressing the data communication and the real-time processes that transform command states of the simulation computer to actuator commands of the motion system.

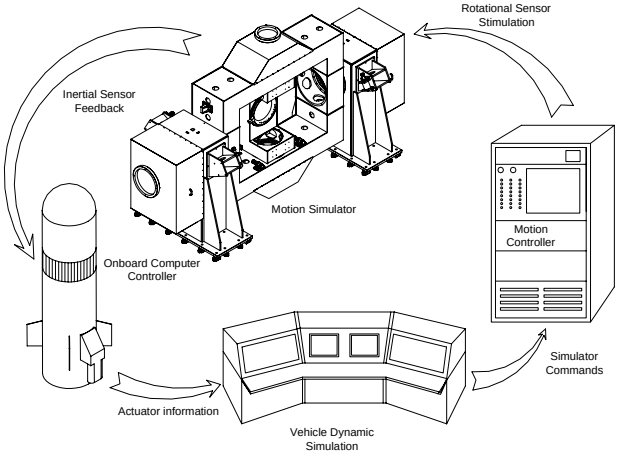


Figure 1: Hardware-in-the-loop architecture and data flow

2.0 MOTION SYSTEM INTERFACES

In the Hardware-in-the-Loop environment the continuous and discrete time components of an integrated flight vehicle are removed from the compact aerodynamic package and are constructed in the distributed hardware and processing systems of the laboratory. The simulation environment often contains subsystems that are supplied by vendors who specialize in supporting fields such as simulation computers, image projectors, and motion flight tables. Acutronic is in the business of designing and supplying high performance motion systems that replicate the dynamic motion that a missile experiences in flight.

Mounting the missile/components to the flight table requires custom interfaces and fixtures. The interface between the table and the control instrumentation strives to be transparent to the end user. The interface to the simulation computer(s) is usually implemented using COTS interface hardware and provides real-time command and monitor of Flight table (missile body) motion.

2.1 Hardware interface options

The interface hardware that connects the simulation computer with the motion control system generally has no relation to the internal missile interfaces; consequently, the preferred hardware and communication protocol is decided by a combination of the facility requirements and the cost/performance assessment of available options for each subsystem.

Commercial interfaces that are traditionally used can be categorized into two basic groups; the parallel interface protocol, and the reflective memory (RM) protocol. The parallel interface, once the interface of choice, has been used for many years with a modest advancement in performance; it is now losing popularity to competing technologies. Over the last 5 years RM interfaces have proven to be higher performance, more versatile, and easier to implement. Many facilities strive to reduce the dedicated point-to-point (parallel) interfaces, opting for the concurrency of reflective memory, which allows many subsystems to simultaneously use or source data.

2.1.1 Parallel interface

he parallel interface is generally a TTL buffered bus capable of transferring 1 to 4 bytes of data for each transfer cycle. Data words must be a minimum of 32 bits to provide the resolution and dynamic range required for motion states. A standard interface such as the NI-653x board offered by National Instruments is capable of transferring a 32-bit word in one direction requiring two boards for bi-directional communication. Alternately, one board can be used to communicate simultaneously in both directions using 16-bit words, but requiring two transfers per word. The most popular transfer protocol is the direct handshake (see Fig. 2.1), which uses REQ and ACK lines for control. Direct memory access (DMA) is used on the Acutrol3000 side, and is optionally used on the customer side depending on preferences. If DMA is not used, then each word must be polled or be serviced by an interrupt; either case reduces the data transfer rate. Interface drivers are available for some platforms, but generally have to be customized for the application.

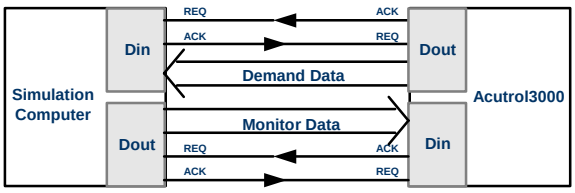


Figure 2.1 Parallel Interface Hardware

2.1.2 Reflective memory interface

A Reflective Memory (RM) interface is based on the replicated or shared memory concept, which uses a high-speed serial-ring network to transfer data in real-time between several computers (see Fig. 2.2). Data written to RM is automatically sent to the same memory location in all nodes on the network. Reflective memory appears as conventional shared memory in the host computer and a dual-port controller allows simultaneous R/W by the network. Data packets are passed by a “message insertion” protocol and ultimately removed by the originating node after all other nodes have been updated. Each board has a unique Node ID address for packet identification. This interface architecture generally provides a method for

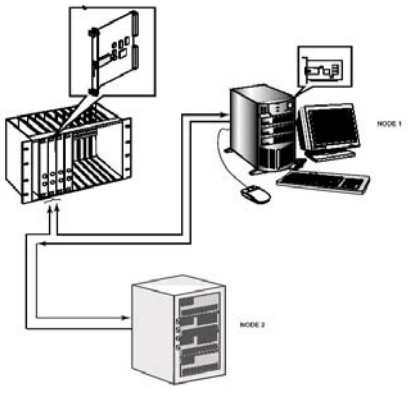


Figure 2.2 Reflective Memory

interrupt generation based on data transfers to specific memory addresses or nodes. Otherwise, semaphores can be used to regulate data usage.

Each computer platform has a custom RM interface board that is easily installed as add-on memory and does not require a custom driver. The network protocol that mirrors data between all of the nodes is transparent to the operating systems and the user application code. Interrupt service routines for the RM are dedicated to supporting the objectives of the application and not to transferring data between computers.

The interconnection medium can either be copper or fiber-optic and the distance between nodes can be measured in kilometers.

2.2 Interface performance

Comparing interfaces is not a straightforward task because of the fundamental differences of the protocols. Each interface family has various modes of operation, which have different features and performance levels. Factors such as cable length, software drivers, data formats, data word/block size, and the number of nodes, all have an effect on the usable data bandwidth. The table below summarizes the peak performance for several commercial interfaces and should not be interpreted as absolute data transfer performance.

Specifications	**Reflective Memory**	**Reflective Memory**	**Parallel**	**Parallel**
Board	Scramnet SC150e	VMIC 5565	NI-6533	NI-6534
Data Transfer Rate	14.7 Mbyte/sec	47 Mbyte/sec	6.7 Mbyte/sec	19 Mbyte/sec
Memory Size	2/4/8 MByte	64/128 MByte	N.A.	32 Mbyte
Node Distance	300/3500 m (Single/Multi)	300/10000 m (Single/Multi)	1/100 m	1/100 m
Network Transfer Rate	150 M	2.12 G	N.A.	N.A.

All of the interfaces identified above are capable of providing adequate performance for the typical real-time data communication between the host simulation computer and the motion system. The data that needs to be transferred is quite limited in comparison to other general-purpose digital interfaces. The typical data blocks consists of the following 32-bit words:

Data Type	**Demand Block**	**Monitor Block**
Motion Vector	P,R,A for each axis = 9	P,R,A for each axis = 9
Control Word	1 X 3 axes = 3	
Sync ID (semaphore)	1 per block = 1	1 per block = 1
Totals	13	10

Ideally the update rate is at least ten times the highest frequency that must be simulated. At one-tenth the Nyquist frequency, the linear phase delay of one frame is 18° and begins to compromise the fidelity of the simulation.

2.3 Identification of communication pitfalls

When motion demand or monitor data are transferred between computers, the integrity of the data may be compromised because of the following reasons:

- The interface does not perform deterministically (missed data or time jitter).
- Communication delays, which add phase lag to the overall simulation.
- Omission of one or more states from the motion demand vector.
- Scaling errors result in an incoherent motion vector.
- Asynchronous transfer results in beat frequencies.
- Sub-harmonic transfer rate results in discontinuous motion states.

3.0 ACUTROL3000 REAL-TIME DATA PROCESSING

3.1 Data processing algorithms

Acutronic has developed real-time data processing algorithms that permit asynchronous multi-rate transfer of motion demand vectors; the following topics are addressed in this paper:

- Asynchronous read and write by the Host simulation computer.
- Relative transfer rates Host/Acutrol from < 10% to > 200%.
- Tracking of the Host frame timing.
- Tracking of the Acutrol frame timing and local time scaling.
- Tracking of Host demand data to ensure coherent data.
- Synthesis of missing states in the demand vector.
- User tuning of demand and monitor transfer latencies.
- Multi-rate demand translation (time skew correction).
- Demand coherency validation (Acutrol frame rate).
- Demand State Limiting.

3.2 Asynchronous data transfer

The HWIL simulation is almost always treated as a linear time invariant process, which requires the Host simulation frame to be fixed during the time of the simulation. It is customary that the host frame period be determined empirically by measuring the worst-case execution time for a scenario. Consequently, it is not convenient for both simulation and control computers to run at the same fixed frame rate and even harder to require absolute synchronization of data transfers.

An important objective of the Acutrol real-time data processing algorithms is to minimize computation delays and to compensate for latencies that result from sequencing data through a multi-rate interface. To this end, the Acutrol3000 controller maintains two real-time processing loops that run simultaneously and asynchronously. This allows host data to be transferred synchronous to the controller and also provides a periodic time basis that host demands can be conditioned for multi-rate translation. By having two loops in the same processing environment the controller has more freedom to define and manage the asynchronous data translation protocols.

This section discusses asynchronous data transfer from an operational viewpoint and introduces the concepts presented in this paper. First we look at the process from the host side of the interface:

- Demand data is output at the end of the simulation frame and assumed to be accurate at that time; writing to a sync semaphore provides a trigger to the controller to process the data.
- Monitor data is read at the beginning of the next frame and assumed to be accurate at that time; a sync semaphore identifies that the data is new, and coherent.
- Done.

The activity in the motion controller is a bit more complicated to accommodate the simplicity of the simulation side:

- A semaphore signals new demand data is available from the host simulation computer.
- When Acutrol is “on-line”, a simulation scenario may start or stop as required by the host.
- The beginning of a scenario will be detected and the controller host loop will be initialized.
- When the scenario stops, it will be detected and the flight table will switch to a safe mode of control.
- During a scenario, demand vectors will be periodically received and time tagged.
- The time tag is compensated for Interrupt latency, transfer latency, and jitter.
- Data is buffered and a new demand flag is set.

Figure 3.2.1 details the write operation. At the end of the M-1 host frame data has been written and an interrupt (↓) is generated. Acutrol frame N is in process and has no chance to use the new data. At the beginning of the N+1 controller frame, it is recognized that there is new data, with a new time stamp. The data will be used at the end of frame N+1 and must be time-skew corrected by the time-skew interval. At the beginning of the N+2 frame, new data is not indicated; consequently, the old demand data will be corrected to the end of the N+2 frame. New demand data is received at the end of host frame M, which in this case occurs sufficiently before controller frame N+3; therefore, the cycle starts over.

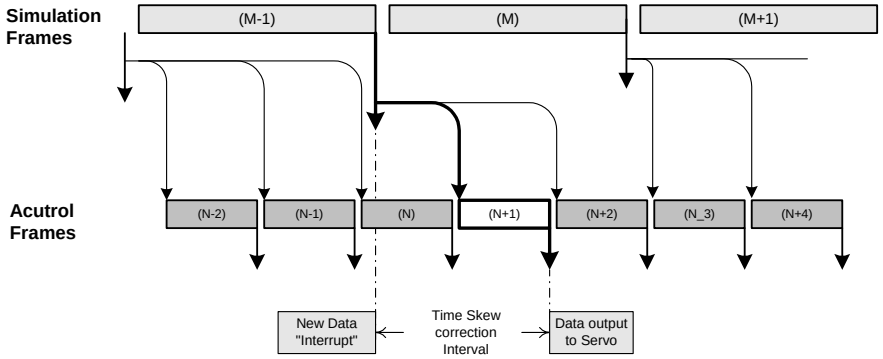


Figure 3.2.1 Asynchronous Write to Motion Controller

- The motion states are measure on each frame and are time tagged to the beginning of the frame.
- The time tag is compensated for Interrupt latency and jitter.
- The time to the next host frame time is computed.
- Monitor data is time skew corrected and made available for reading.

Figure 3.2.2 details the read operation of the flight table motion states. Motion data from frame N+1 is time skew corrected to the beginning of host frame M+1. If the time to updating the monitor data is within the lock-out-zone of the host frame time, the update is skipped; this ensures that all the data is from the same frame. Frames N and N+2 correct their motion data to the same host frame.

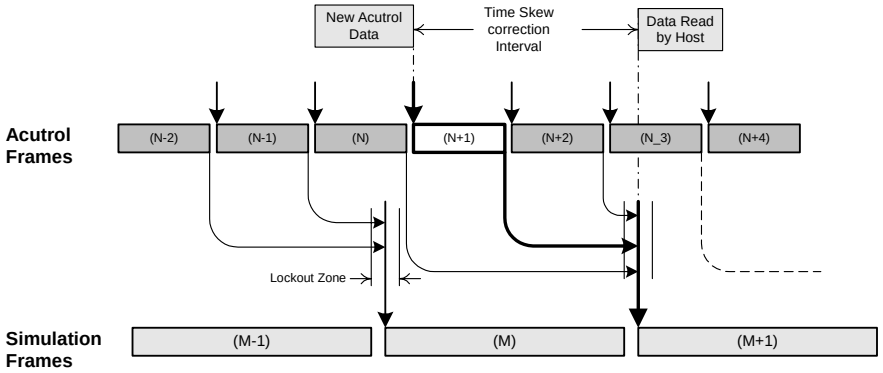


Figure 3.2.2 Asynchronous Read from Motion Controller

3.3 Data processing topology

A simulation frame begins as the host computer inputs scenario and feedback data. The simulation algorithms are processed and the dynamic states of the simulation are predicted to the end of the current frame. The flight table demand vector is output prior the beginning of the next host simulation frame and this signals the beginning of the controller host frame. A diagram of the Acutrol3000 real-time data processing is presented in Figure 3.3.1. Note that once the real-time date is processed through the host interface (blocks 1 & 2), it is immediately placed into shared

memory along with other variables that are accessed by the various real-time routines. The completion of the host demand data transfer generates an interrupt in the controller. The host interrupt is the highest-priority interrupt service routine (ISR) in the Acutrol application code and runs all of the host frame routines to completion. The ISR begins by running the host frame time tracker (11), which captures the CPU time stamp counter (TSC) and updates the host frame times. Next, the host demand tracker (12) filters the host demand vector (4) and updates the filtered demand vector (6); the new demand semaphore (5) is set TRUE. Still in the ISR, the host monitor routine (13) time-skew corrects the current Acutrol monitor vector (8) to be accurate at the current host monitor time $t_H(n)$ and places the new host monitor vector (9) in memory. Finally the monitor semaphore is set TRUE and the ISR is done.

As time continues, either another host frame interrupt is generated, or more likely, a controller interrupt is generated by the servo control hardware (16). The controller ISR captures the TSC, performs other required tasks, and sets a semaphore that launches the real-time thread after returning. In the controller thread, the Acutrol frame time tracker (14) is executed and the frame time variables are updated. The new demand flag (5) is tested and if true, the multi-rate demand translator (15) is refreshed; otherwise, the old host demand vector (4) is used to update the Acutrol demand vector (7). The Acutrol demand vector is used by the servo controller (16) algorithms to generate motion table (18) commands. The controller monitors the table position sensors and produces an Acutrol monitor vector (8), which is time skew corrected to the predicted host frame time $t_H(n+1)$ by the host frame monitor (17). The monitor semaphore is set TRUE and the real-time thread blocks until the next Acutrol frame interrupt.

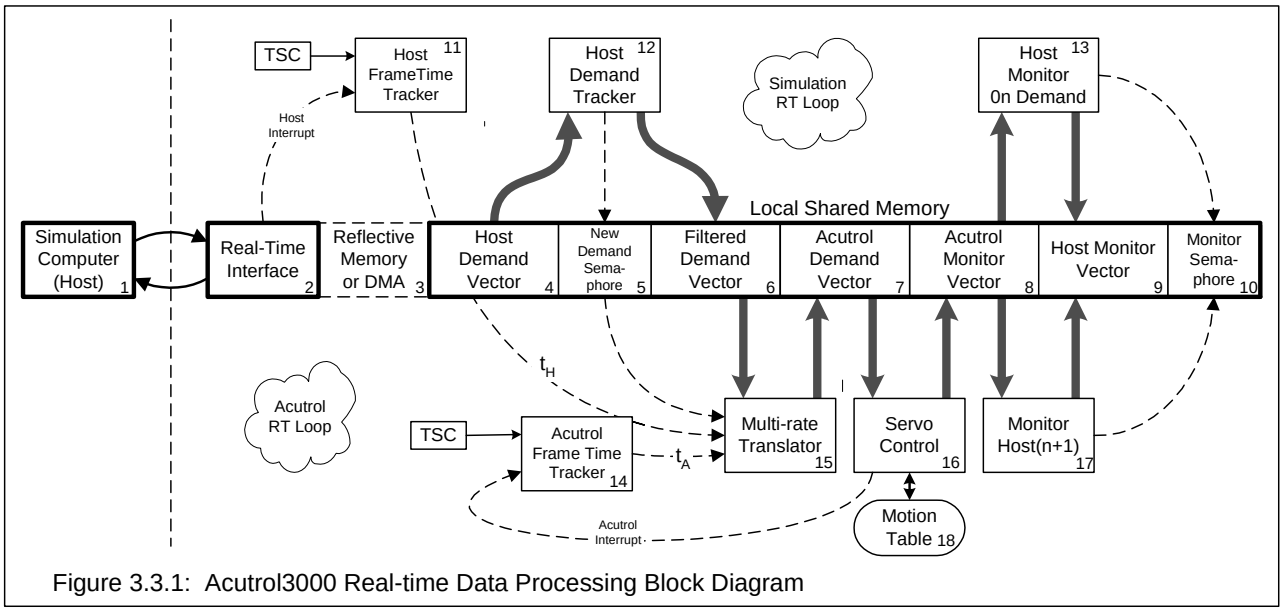


Figure 3.3.1: Acutrol3000 Real-time Data Processing Block Diagram

3.3.1 Frame Time Tracker

The *frame time tracker* (FTT) is used in both the host and the controller loops to condition the measurements of the Pentium TSC (time-stamp-count), which provide the relative timing required for multi-rate translation of motion data. Priority tasks performed in the corresponding interrupt service routines include reading the TSC, running the FTT, and updating the frame time variables.

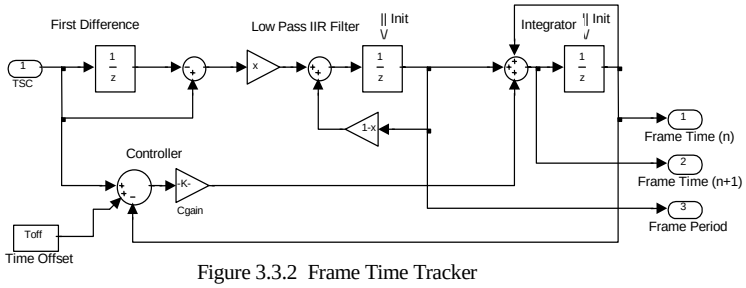


Figure 3.3.2 Frame Time Tracker

The first difference of the TSC is low pass filtered and provides a measure of the average frame period. The TSC is measured in CPU clock cycles and the Acutrol frame period is precisely known; thus, the time scale factor can be computed for all time measurements based on the

TSC. The estimated frame time t(n+1) is computed by integrating the current frame time t(n) by the sum of the average period and the loop tracking error. The controller gain sets the dynamic tracking response and the phase of the frame time is adjusted to compensate for data transfer and interrupt latencies. The estimate of the next Acutrol frame time $t_A(n+1)$ defines the next time that the compensated servo command will be output to the table actuation system. Demand data must be translated to this time from the last host frame time $t_H(n)$. The current frame time $t_A(n)$ of the controller indicates the instant that the position measurement of the motion table was accurate. The time from $t_A(n)$ to the estimate of the next host time $t_H(n+1)$ represents the latency of the monitor data that must be time skew corrected.

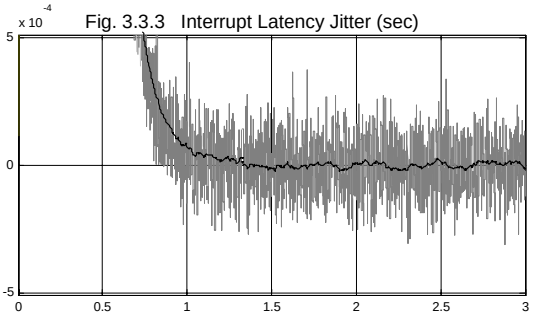


Fig. 3.3.3 Interrupt Latency Jitter (sec)

Figure 3.3.3 shows the effectiveness of the FTT in minimizing the uncertainty of the interrupt latency and thus the noise of the timing measurements.

3.3.2 Host Demand Tracker

The *host demand tracker* serves to ensure that the states of the demand vector are coherent (integrally related and discrete time continuous). The *tracker* is executed each time that new demand data is transferred from the simulation computer to the Acutrol motion controller. Ideally, the *tracker* receives a demand vector that contains position, rate, and acceleration motion states. If the simulation computer only sends a partial demand vector then the tracker serves the additional function of estimating the missing states; a full demand vector is always generated.

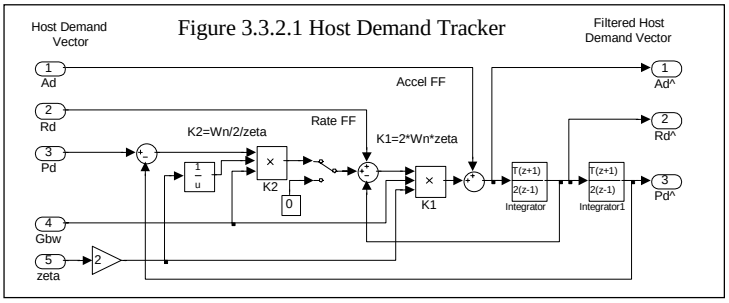


Figure 3.3.2.1 Host Demand Tracker

The *tracker* shown in Figure 3.3.2.1 has programmable bandwidth and damping to tailor the dynamic response, which rejects the non-coherent character of the demand data. Setting a low bandwidth allows the acceleration to dominate the behavior of the filtered demand vector; whereas, a high bandwidth will give the position state more authority. With a high bandwidth, the *tracker* attempts to minimize the error between the input demand and the output from the previous frame. Consequently, the *tracker* generates a demand vector that is predicted by one host frame. This effect, shown in Figure 3.3.2.2, can be used to facilitate the process of extrapolating the current host demand vector over multiple Acutrol frames until the next time that host data is received.

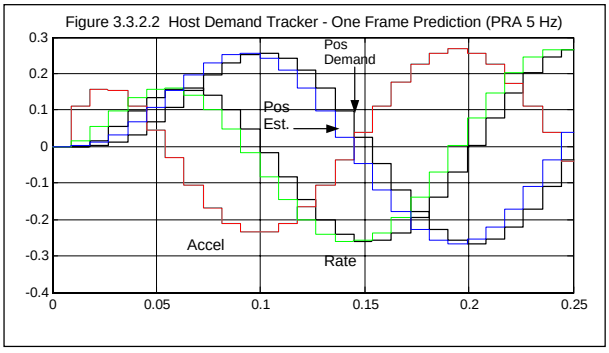
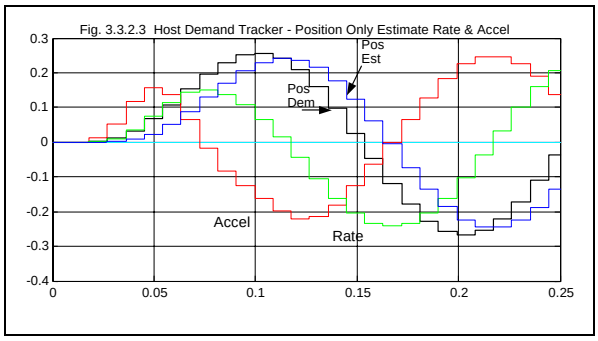
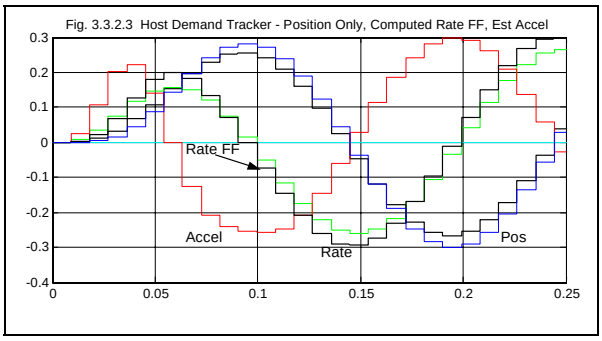


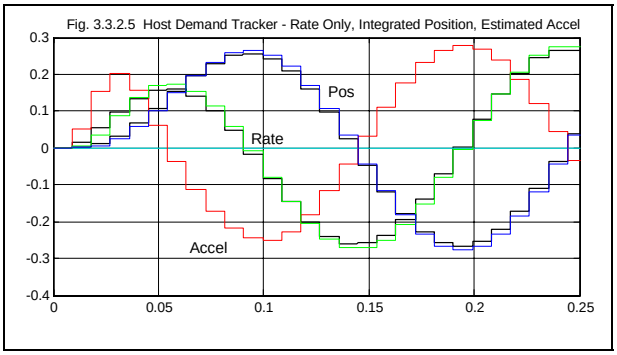
Figure 3.3.2.2 Host Demand Tracker - One Frame Prediction (PRA 5 Hz)

Because of the finite *tracker* bandwidth, the omission of motion states generally results in a phase delay of the filtered demand vector. In practice, alternate *host demand tracker* configurations may be selected which employ more sophisticated methods to estimate the missing states and thus minimize the phase and amplitude errors. Figure 3.3.2.3 has only position as an input and the demand tracker outputs a full coherent demand vector that has an effective delay of approximately 1.5 host frames. This corresponds to approximately 24° of phase shift at 5 Hz. The phase shift is reduced to about 5° by using an alternate *tracker*, which reconstructs a feed forward rate demand from the position.





Another simulation scheme sends only rate commands to the motion table; this results in piece wise continuous rate demands and the host simulation loop is obliged to manage the position of the motion system. The *tracker* estimates position and acceleration and in this case, the bandwidth is used to trade off tracking (phase lag) for a reduction of acceleration transients. Sending both rate and acceleration results in virtually no tracking error; however the true position of the flight table can only be known by reading the position.



3.3.3 Demand Translator

The *Demand Translator* (Fig.3.3.3.1) functions as a bridge to pass real-time data from the host simulation frame to the Acutrol frame. Data sequencing in the two processing environments is assumed to be asynchronous and the relative update frequency may vary widely from one application to another. In general, the Acutrol update rate of 2.5 to 5 kHz is faster than the average simulation rate used in HWIL laboratories. The multi-rate *translator* is able to function effectively for simulation update rates between one tenth and two times the Acutrol update rate. If the simulation rate is greater than the Acutrol frame rate, then the *host tracker* will simply update the filtered host demands at a faster rate.

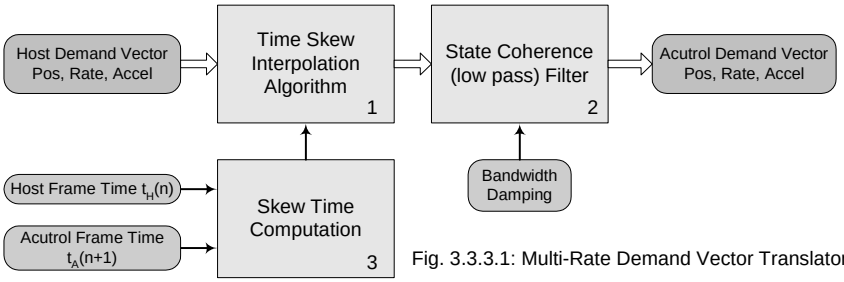


Fig. 3.3.3.1: Multi-Rate Demand Vector Translator

The *translator* runs synchronous to the Acutrol frame and uses the most recent data available from the *host tracker*. The most recent host frame time $t_H(n)$ corresponds to the time tag for this data and is referenced to the Acutrol time base. The time $t_A(n+1)$ corresponds to the end of the current Acutrol frame, which is the instant that the actuator commands are output. In Figure 3.3.3.1, block 3 computes the time skew between input and application of the demand

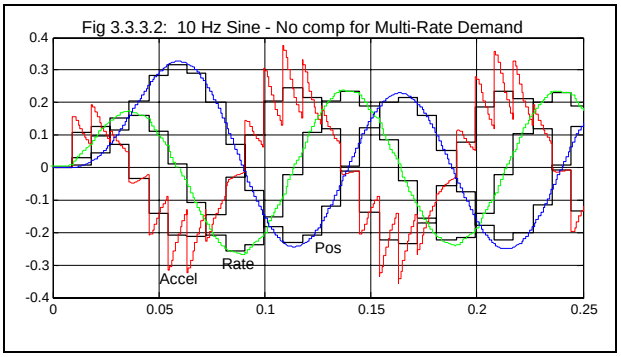
data, and block 1 uses this time skew to translate the demand vector. The coherence filter does not affect the dynamic behavior of the demand data because the full state demand vector naturally integrates to the translated time as long as the translated states maintain a coherent relationship (rate is the integral of acceleration, and position, the integral of rate). As with most of the RT modules in the Acutrol3000, the user can choose between several versions of the *translator* based on different extrapolation, interpolation, and filter algorithms.

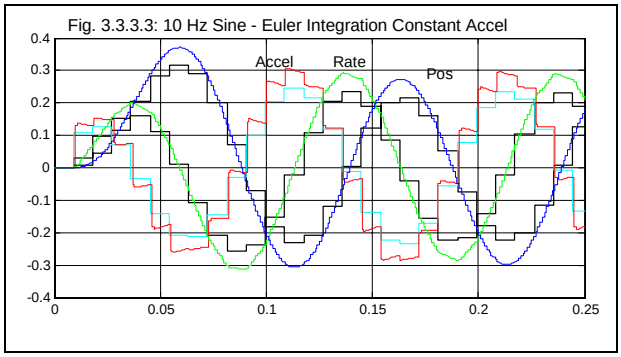
The heart of the *demand translator* is the time skew interpolation block (1). Variations of the translator are driven by one or more of the following features/objectives:

- Host frame prediction
- Smooth motion states (fidelity)
- Coherency of motion states
- Noise rejection

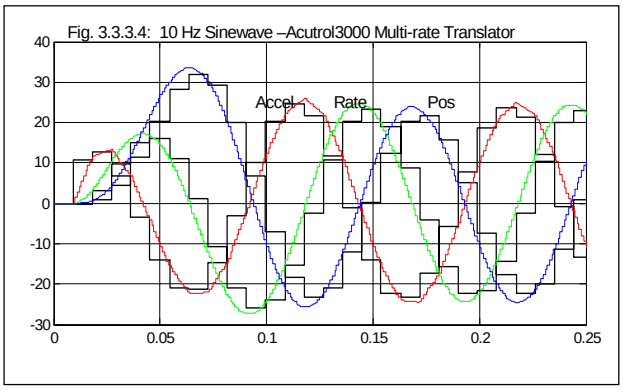
The simulated plot in Figure 3.3.3.2 represents the worst case handling of multi-rate data, i.e. no data translation. In this case, host demand data asynchronously becomes available and the controller simply uses the most recent data as the motion system demands for the entire host frame interval. In a severe instance, the acceleration feed forward will saturate the servo control loops; consequently, this arrangement results in a system that has relatively large phase lag and the dynamic motion is contaminated with large beat frequency disturbances.

Figure 3.3.3.3 demonstrates using Euler integration of the host demands over one host frame period. The integration is done at the controller frame rate to produce controller servo commands. If the simulation computer can maintain sub harmonic synchronization, then beat frequencies will be eliminated. The resultant commands are much smoother but the step acceleration produces piece wise continuous ramping of the rate motion state. Acceleration (torque) steps at the host frame rate may still cause significant stimulation of the structural modes of the motion system.





The final case shown in Figure 3.3.3.4 is a simulation of the Acutrol3000 translator response. Note that the host acceleration is transformed into a piece wise continuous ramp (constant jerk) and becomes the (feed forward) acceleration command in the controller. The rate and position command states are very smooth because the coherency filter on the output ensures correct integral scaling of the motion states. Also notice that the Acutrol rate and position commands are predicted to the beginning (leading edge) of the corresponding host demand. The net phase shift from the host demands to the Acutrol3000 controller commands is zero; this is true for all frequencies in the simulation bandwidth as long as a host sends a full demand vector.



4.0 SUMMARY MEASURED DATA

In this section data is presented for various test cases to demonstrate the effectiveness of the real-time data processing algorithms incorporated in the Acutrol3000 motion controller. Data was collected using built-in data logging capability and the data was post processed and plotted using Excel.

Figure 4.1 is a plot of the servo command (torque) that drives the table. The host sends motion demands at one tenth of the controller sampling rate and synthesizes a sine wave at a frequency of 0.5 Hz. Two signals are plotted together; the indicated signal, that exhibits periodic transients, has the translation algorithm disabled. The transients occur at the sub-harmonic beat frequency, and are completely eliminated when the *translator* is in use.

Fig. 4.1 Servo Cmmd 0.5 Hz
(update ratio = 0.1)

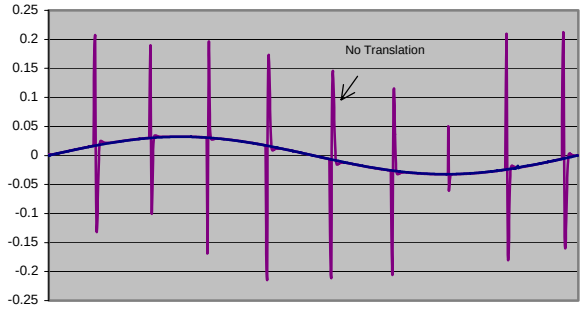


Fig. 4.2 Servo Cmmd 10 Hz
(update ratio = 0.999)

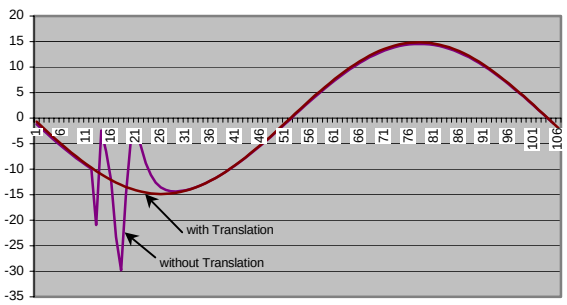
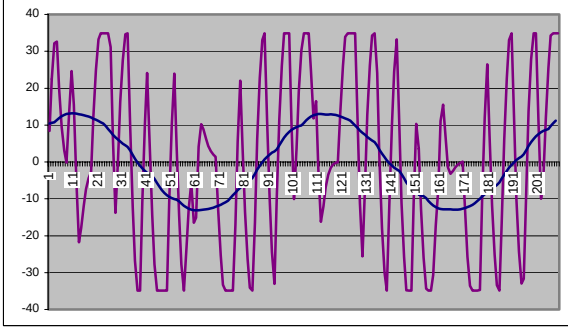


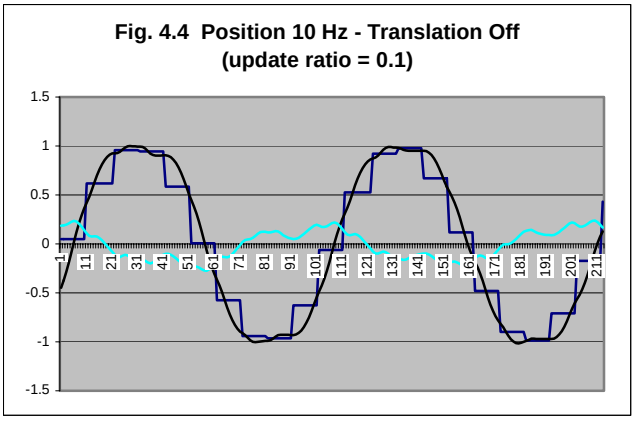
Figure 4.2 is a plot of the servo command of a 10 Hz sine wave where the host update rate is nearly the same as the controller. As the sample time of one system overtakes the other a discontinuity of the transferred motion states causes a transient in the table motion. Again this is totally eliminated when the *translator* is active.

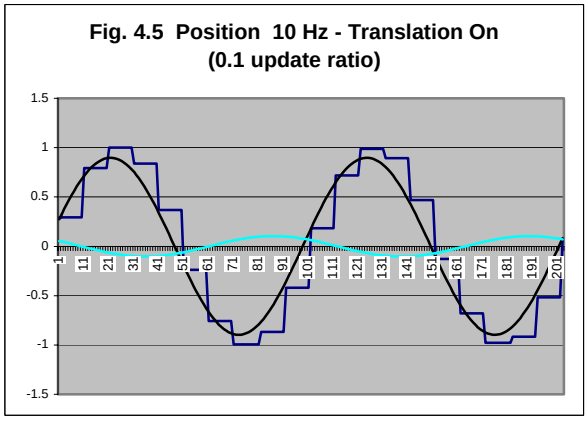
Figure 4.3 is a 10 Hz sine wave with a host update of approximately one-tenth the controller. The elimination of the beat frequency disturbance is nearly 100% with the *translator*. The noise of this servo command is rich in harmonics and could physically be felt and heard in the table.

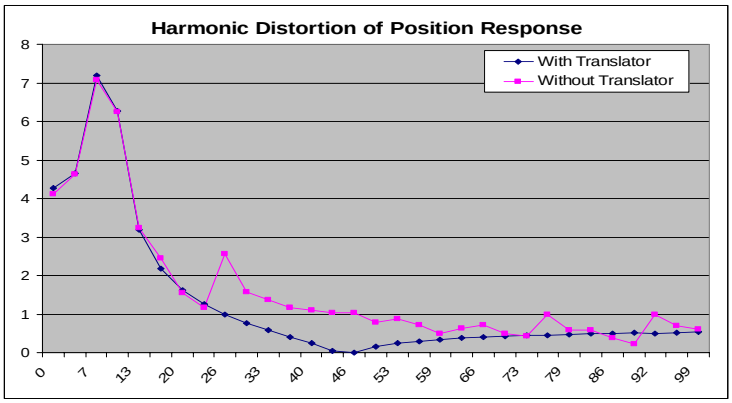
Figures 4.4 and 4.5 are plots of position corresponding to the servo commands of Figure 4.3. The three signals plotted are position demand, (updated at 100 Hz), position feedback from the test table (updated at 1 kHz), and the position error. The improvement of the position response is significant in two areas. First, the phase of the position is more representative of the host position command at the instant that it was sent. In a HWIL simulation, this position will track with less following error and will contribute less phase delay to the total simulation. The position fidelity is improved as is seen in the amplitude spectrum plotted in Figure 4.6 and the resulting simulation is not contaminated by undesirable systematic noise.

Fig. 4.3 Servo Cmmd 10 Hz
(update ratio = .1)









References

[1] Michael Swamp, Colin Stevens, Peter Hoffstetter, “Improvements to Transient Fidelity of HWIL Flight Tables Using Acceleration Feedback”, Proceedings of SPIE Vol. 4717 (2002), presented at Aerosense 2002, 1-2 April 2002.

[2] Louis A. DeMore, Paul Mackin, Michael Swamp, Roger Rusterholtz, “Improvements in Flight Table Dynamic Transparency for Hardware-in-the-Loop Facilities”, Proceedings of SPIE Vol. 4027 (2000), presented at Aerosense 2000, 24-26 April 2000.