Performance Tests of a 12-Channel Real-Time GPS L1 Software Receiver

B.M. Ledvina, A.P. Cerruti, M.L. Psiaki, S.P. Powell, and P.M. Kintner
College of Engineering, Cornell University

BIOGRAPHIES

Brent M. Ledvina is a Postdoctoral Associate in the School of Electrical and Computer Engineering at Cornell University. He recently received his Ph.D. in Electrical and Computer Engineering from Cornell University. He has a B.S. in Electrical and Computer Engineering from the University of Wisconsin at Madison. His areas of interest cover estimation, plasma irregularities in the equatorial and mid-latitude ionosphere, and GPS technology.

Alessandro P. Cerruti is a Ph.D. student in the School of Electrical and Computer Engineering at Cornell University. His interests include development of dual-frequency software/hardware GNSS receivers. He has earned both his B.S. and M.Eng. degrees in Electrical and Computer Engineering at Cornell University.

Mark L. Psiaki is an Associate Professor of Mechanical and Aerospace Engineering at Cornell University. He received a B.A. in Physics and M.A. and Ph.D. degrees in Mechanical and Aerospace Engineering from Princeton University. His research interests are in the areas of estimation and filtering, spacecraft attitude and orbit determination, and GPS technology and applications.

Steven P. Powell is a Senior Engineer with the Space Plasma Physics Group in the School of Electrical and Computer Engineering at Cornell University. He has been involved with the design, fabrication, testing, and launch activities of many scientific experiments that have flown on high altitude balloons, sounding rockets, and small satellites. He has M.S. and B.S. degrees in Electrical Engineering from Cornell University.

Paul M. Kintner, Jr. is a Professor of Electrical and Computer Engineering at Cornell University. His interests as a rocket scientist range from exploring the northern lights to understanding space weather. His recent work with GPS receiver design and scintillations led NASA to appoint him chair of the Living With a Star Geospace Mission Definition Team.

ABSTRACT

A 12-channel real-time GPS software receiver has been tested in order to determine its accuracy and tracking performance under dynamic conditions. The motivation for this work is to demonstrate that this receiver's performance is equivalent to a hardware receiver despite the fact that it implements all of its base-band mixing, correlation, and accumulation operations in software in a general-purpose PC. Such receivers offer an attractive route to the development of products that exploit new signals that will be available in the future such as the L2 civilian signal and Galileo signals. A software receiver can be developed to track such signals without the need to develop a new hardware correlator chip. All that needs to be done is to adjust the parameters of the RF front-end to receive the new carrier frequencies and upgrade software

to calculate correlations using the new spread-spectrum codes.

The real-time software receiver is reviewed, and its operation is compared to that of receivers which use hardware correlators. Also discussed is an upgrade to the original implementation that uses processor-specific x86 assembly code (MMX). This upgrade provides a 25% increases in processing speed. Another aspect that is discussed is the use of 1-bit RF front end data and the related speed-up. The software receiver is tested under static and dynamic conditions and is compared to a hardware receiver in terms of its tracking stability and the accuracy of its observables. The dynamic test conditions include a rocket scenario that is generated using a GPS signal simulator and road vehicle motion that is tested using car-mounted receivers.

INTRODUCTION

A real-time software receiver architecture can provide GPS user equipment with operational flexibility that will prove more and more useful as time goes by. The current GPS system is slated to expand its capabilities to include new civilian codes on the L2 frequency and a new L5 frequency. A receiver that uses a hardware correlator will require hardware modifications in order to use these new signals. In the near term, a receiver designer will be faced with a complex trade-off in order to decide whether the extra complexity is worth the improved performance that will accrue gradually as new GPS satellites replace older models. A software receiver can use new signals without the need for a new correlator chip. New frequencies and new pseudo-random number (PRN) codes can be used simply by making software changes. Thus, software receiver technology will lessen the risks involved for designers during the period of transition to the new signals. Furthermore, a software receiver could be reprogrammed to use the Galileo system, which provides an added benefit from the use of a software radio architecture. Thus, there are good reasons to develop practical real-time software GPS receivers.

A GPS software receiver differs from a hardware receiver by performing correlations in software running on a general purpose microprocessor. A software receiver can be broken down into various components (see Figure 1). First, an antenna, possibly followed by a pre-amp, receives the L-band GPS signals. After the antenna comes a RF section that filters and down converts the GHz GPS signal to an intermediate frequency in the MHz range. The RF section also digitizes the signal, outputing a binary bit-stream. The next section is a buffering and data acquisition section that reads that data into the microprocessor's memory. The final section is the general purpose microprocessor that runs software to perform the remaining GPS receiver functions. The software correlator separates the signal into different channels allocated to each satellite. For each satellite, the correlator mixes the Doppler shifted intermediate frequency signal to baseband and correlates it with a local copy of the satellite's PRN code. The final components of the receiver involve standard software routines that track the signals, demodulate the navigation message, and compute the navigation solution.

GPS software receivers have been around for several years. In the recent past, GPS software receivers have been developed that either post-process stored signals or operate in real-time. Previous real-time software receivers function with 12 channels running on a high-end PC [*Thor et al.*, 2002, *Ledvina et al.*, 2003a] or 8 channels on a DSP chip [*Akos et al.*, 2001b]. The work presented in this paper improves upon these previous works by demonstrating that a software receiver can operate in a dynamic environment. Previous works show only static receiver operation.

The remaining portions of this paper describe the software receiver, the experiments, and present performance results for this system. The second section describes the hardware including the RF front end and the PC. The third section reviews the software receiver's bit-wise parallel software correlation process. Section 4 describes the static performance of the receiver. Section 5 investigates the receiver's dynamic navigation performance. Section 6 gives a summary and concluding remarks.

SYSTEM CONFIGURATION

Central to the software GPS receiver is a personal computer (PC). The current system consists of a PC with a

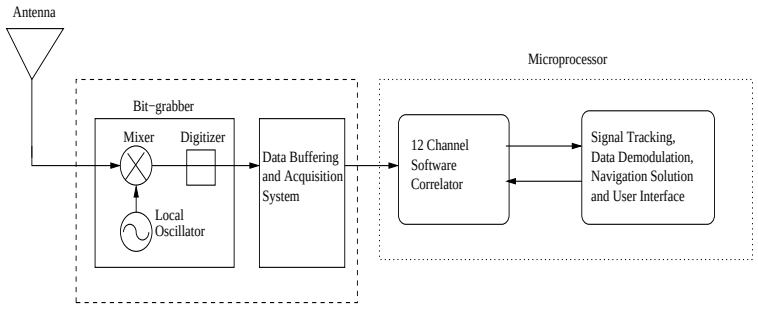

Figure 1: *A typical GPS software receiver showing the separation between special purpose hardware and general hardware.*

1.73 GHz AMD Athlon processor running the RT-Linux operating system. RT-Linux is a hard real-time variant of Linux implemented as a set of patches to the standard Linux kernel. Due to its real-time optimized design, RT-Linux provides low latency interrupt responsiveness along with the ability to execute threads at regular intervals. This translates into a highly efficient and responsive operating system that reliably executes time critical code. An additional feature of RT-Linux is that it keeps the functionality of Linux by running the kernel as the lowest priority thread. By retaining the functionality of Linux, it is easy to develop, test, debug, and run real-time software. Another benefit of using Linux is that tools such as drivers, a C complier, and text editors are readily and freely available.

The next component of the software receiver is the digitizing RF front end, also known as a bit-grabber. The heart of the bit-grabber is a Mitel/Zarlink GP2015 RF front-end. The front-end down converts the nominal 1.57542 GHz GPS L1 signal to an intermediate frequency of $(88.54/63) \times 10^6$ Hz $\cong$1.40539 MHz and then performs analog-to-digital conversion. The resultant, digitized signal has two binary bits per sample corresponding to a sign and a magnitude. The possible values for the digitized signal are ± 1 and ± 3. The two binary bits are available as outputs from the bit-grabber. In order to provide accurate timing, the sign and magnitude bits are synchronized to a $(40/7) \times 10^6$ Hz $\cong$ 5.714 MHz clock signal, which is the third output from the bit-grabber card.

A data acquisition system reads the digitized sign and magnitude bits from the bit-grabber into the PC. To make the process of reading data into the PC more efficient and to prepare for efficient correlation calculations, the DAQ card reads 32 bits of buffered samples at a time. The 32 bits consist of 16 sign bits and 16 magnitude bits. A series of shift registers buffer the data, packing the sign and magnitude bits into separate 16-bit words. A divide-by-16 counter converts the 5.714 MHz clock down to 357.14 KHz, which provides a signal indicating when the buffer is full.

The data acquisition system consists of a PC card and driver software. The card is a National Instruments PCI-DIO-32HS digital I/O card. Pertinent features of this card are the 32 digital input lines, direct memory access (DMA) and availability of a driver for RT-Linux. A suite of open source drivers and application interface software for DAQ cards known as COMEDI (COntrol and MEasurement and Device Interface) is freely available. COMEDI provides Linux/RT-Linux support for nearly one hundred DAQ cards spanning numerous manufacturers.

The software receiver is written entirely in ANSI C code using tools available from standard Linux distributions. To promote portability of the software, no processor-specific assembly language or special instructions are used.

REVIEW OF THE SOFTWARE CORRELATOR IMPLEMENTATION

The software receiver tested here uses bit-wise parallel Boolean logic computations to calculate correlations. This approach differs from other software receivers that perform integer-based computations in either the time domain or frequency domain. Bit-wise parallel correlation

processes 32 RF output data samples at a time. This reduces the number of instructions by a factor of 2 or more [*Ledvina et al.*, 2003a]. Bit-wise parallel correlation is similar to how a hardware correlator functions. The bit-wise nature of this correlation method is aided by how the RF front end output data is read into the PC's memory. 32-bit words, representing 16 samples, are synchronously read into the PC's memory. There is no translation to an integer representation of the signal. Boolean logic operates on 32-bit words, each of which represents the sign or magnitude of various signals at 32 successive samples.

In order to reduce the computation load during correlation, the code and carrier replicas are pre-computed and stored in memory. Prompt and early-minus-late C/A codes are pre-computed for each PRN number at code phases that allow for 1.8-meter measurement accuracy. These pre-computed codes require 930 kilobytes of storage. In-phase and quadrature carrier signals are computed on a coarse frequency grid with 175 Hz spacing at a 0 initial phase offset. These replicas require 320 kilobytes of storage. The correlator compensates for the use of carrier replica signals of the incorrect frequency and phase. The in-phase and quadrature accumulations are rotated by an angle that represents the average phase difference between the available base-band signal and the intended signal at the correct frequency and phase. This procedure is thoroughly explained in *Ledvina et al.*, [2003a] and *Ledvina et al.*, [2003b].

Base-band mixing is performed by computing the bit-wise exclusive-or of the RF data sign words and carrier replica sign words over a C/A code period. The RF data magnitude words and carrier replica magnitude words are not operated on, and are simply passed to the next stage. The resultant base-band signal has a 3-bit representation consisting of a 32-bit sign word, a 32-bit low magnitude word, and a 32-bit high magnitude word for 32 successive samples. Code mixing also involves a sequence of bit-wise exclusive-or operations. To mix the base-band signal with the C/A code, the base-band sign words are exclusive-ored with the code replica's sign words. The resultant signal also has a 3-bit representation. Accumulation involves summing the resultant signal over the accumulation interval, which is nominally the C/A code period. Accumulation is performed by separating the eight different combinations of the 3-bit signal and then using a look-up table to count the instances of each of the eight combinations.

COMPUTATIONAL PERFORMANCE

The computational performance of a software receiver is an important issue in determining its real-world applicability. Previous real-time software receivers require large computational speeds, limiting their usefulness. The receiver presented in *Akos et al.* [2001a] requires nearly 100% utilization of a 1 GHz PC to run only 6 channels. The improved performance of the bit-wise parallel receiver described in *Ledvina et al.* [2003a] requires 50% of a 1.73 GHz PC to run 12 channels.

Two types of computational performance enhancements have been tested. First, assembly language instructions provide a method to optimize the inner loop performance of the software receiver to a particular microprocessor. The drawback to this method is that the correlation algorithms must be custom tailored to particular processors. Second, a reduction in the number of bits in the RF data stream can provide increased computational performance. This reduction reduces the carrier-to-noise ratio. For example, switching from a 2-bit RF data stream to a 1-bit stream with no change in sampling frequency causes a 2 dB reduction in C/N_0 [*Van Dierendonk*, 1996]. Regardless of the drawbacks, these increases in computational efficiency may be useful in certain applications.

The computational performance of the software receiver described in *Ledvina et al.* [2003a] has been tested with these two enhancements. In order to perform the testing, a non-real-time version of the software receiver has been used. This receiver performs correlation, tracking, data demodulation, and computes a navigation solution. The receiver processes 24 seconds of data on 4 channels. The processing time is then multiplied by 3 and divided by 24 to represent a 12-channel receiver's operation during 1 second. This test excludes the computational requirements from the data acquisition system. These latter requirements are typically small because of the use of DMA data transfers.

The receiver has been tested in four different configura-

tions. All four configurations utilize the same 1.73 GHz PC and 5.714 MHz RF sampling rate. The first configuration is with a 2-bit RF front end data stream. The second configuration replaces the inner loop code mixing and sections of the accumulation process with x86-specific MMX assembly language instructions. MMX instructions perform 128-bit integer-based computations instead of 32-bit computations. The third configuration uses a 1-bit RF front end data stream. The fourth configuration uses 1-bit RF front end data with MMX instructions.

The results from these tests are shown in Figure 2. It should be noted that the nominal receiver computational requirement is 40%, which has been reduced from the 50% stated in *Ledvina et al.* [2003a]. This reduction is primarily due to performance tuning and profiling that helped reduce the inner loop processing requirements. The use of MMX instructions decreases the computational requirements by 25% in the 2-bit case and 0% in the 1-bit case. The lack of reduction in the 1-bit case is thought to be due a single factor. The MMX instructions were hand-coded which may not represent the most efficient execution on a pipelined processor.

A computational decrease of 50% is seen when going from a 2-bit RF front end data stream to a 1-bit data stream. This corresponds well with the reduction by a factor of 1/2 in the number of instructions [*Ledvina et al.*, 2003a]. Measured changes in C/N_0 indicate a decrease of approximately 2 dB when using a 1-bit RF data stream.

STATIC NAVIGATION PERFORMANCE

The tracking and navigation performance of the GPS software receiver is first investigated using a stationary receiver. Two performance comparisons are made. First, the software receiver's tracking performance is compared to that of an off-line smoother. Second, the software receiver's pseudorange errors and navigation performance are compared to those of a hardware receiver.

The tracking loop performance of the software receiver code has been evaluated by comparing it to a software receiver implemented in MATLAB that uses smoother-based carrier and code tracking loops and that operates on the same data in an off-line mode. Figure 3 compares

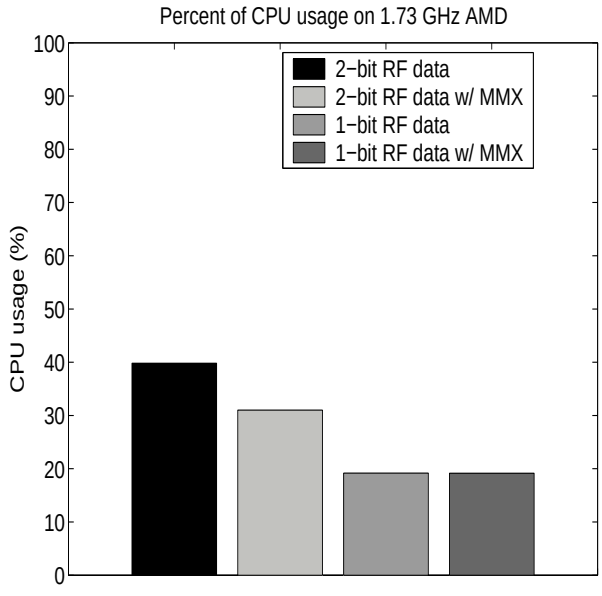

Figure 2: *Percent CPU usage of the software receiver in various configurations.*

the Doppler shift of the carrier from the real-time software receiver with that of the MATLAB smoother. The mean frequency error deviation after the transient period is less than 2Hz. Thus, the real-time software receiver's FLL operates properly with the software-computed accumulations.

The static horizontal navigation accuracy of the receiver is shown in Figure 4. This figure shows the horizontal navigation solution error every second for a duration of 15 minutes. This plot demonstrates that the software receiver nominally performs as well as a typical GPS L1 hardware receiver.

Next, the static navigation performance of the software receiver is compared with a hardware receiver. The hardware receiver and software receiver are identical, except for their implementations of the correlator. The hardware receiver uses a Mitel/Zarlink GP2021 digital correlator chip, while the software receiver uses a software correlator. All other components of the receiver, including the RF front end and tracking and navigation software are identical. The two receivers share a single roof-mounted antenna. The signal is split by a passive splitter and is connected to both receivers. The receivers were operated concurrently during this test.

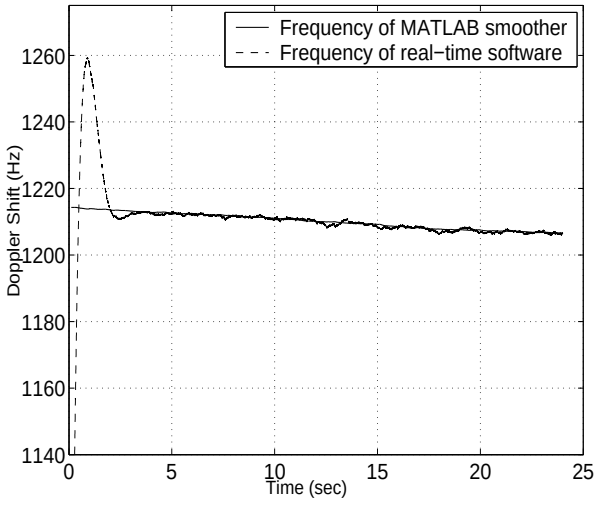

Figure 3: *Doppler shift estimate of the real-time software receiver's FLL compared with that of a MATLAB smoother [Ledvina et al., 2003].*

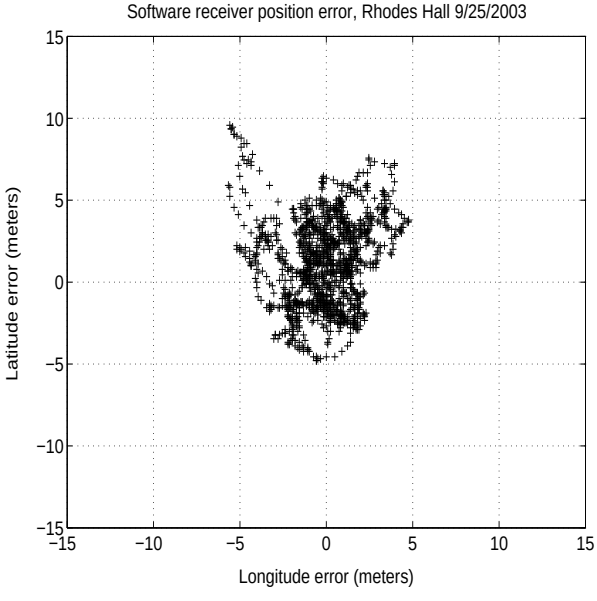

Figure 4: *Horizontal navigation error of the software receiver.*

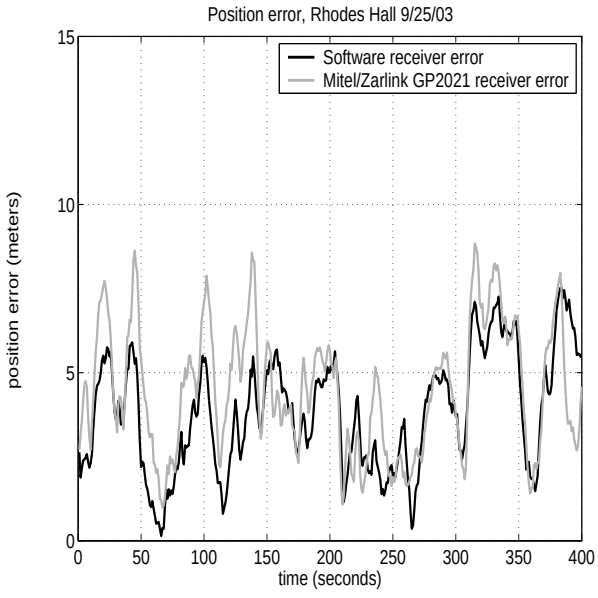

Figure 5: *Norm of the position error for both the hardware receiver and the software receiver.*

Figure 5 shows a comparison of the static navigation performance of both receivers. This figure shows the norm of the error with respect to the surveyed antenna location. The two plots are similar, showing good correlation in navigation errors between the two receivers. The differences between the two error plots are due to two factors. First, the receivers did not sample pseudoranges at the exact same times. This leads to minor discrepancies between the receivers when computing the navigation solution. The second difference is due to performance differences of the correlators. Since neither plot shows a tendency for a larger error, it is reasonable to assume that both correlators perform similarly.

Figure 6 shows the residual pseudorange error for a high elevation satellite, PRN 2, from both the hardware and software receiver for the same data set as Figures 4 and 5. The residual pseduorange error is calculated by taking the difference between the measured pseudorange and the true range to the satellite. Corrections for the receiver clock offset, the ionospheric delay, and tropospheric delay are made. The result is an estimate of the receiver-dependent pseudorange error.

These residual pseudorange errors are typical for a GPS

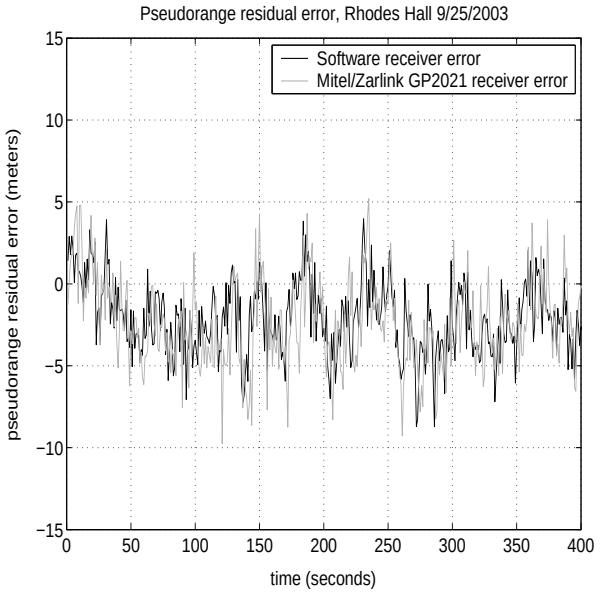

Figure 6: *Residual pseudorange error for PRN 2 for both the software receiver and the hardware receiver.*

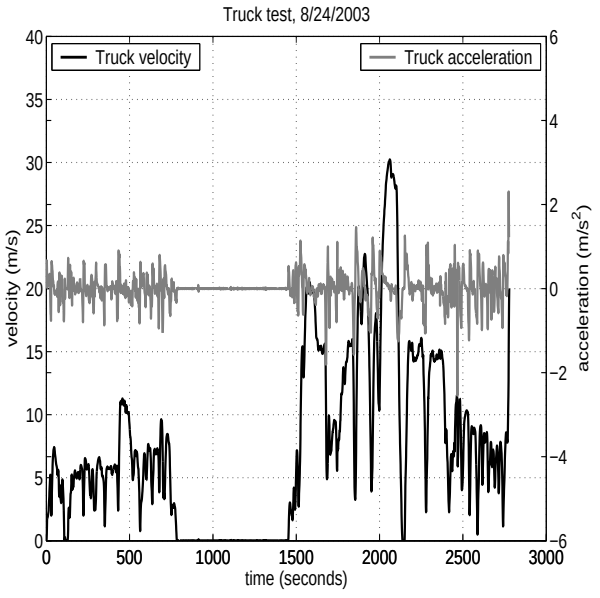

Figure 7: *Vehicle velocity and acceleration profile used to test the software receiver.*

L1 receiver. These two receivers have similar error time histories, which are mostly the result of multipath effects and antenna noise, which are common to both receivers. Differences in these residual errors are due to the lack of synchronization when sampling the pseudoranges, the receivers' different thermal noise, and differences in the receivers' correlators. The similarity between these residual errors reinforces the notion that this software receiver functions as well as a hardware equivalent.

DYNAMIC NAVIGATION PERFORMANCE

The software receiver has been tested in two different dynamic scenarios. The first scenario demonstrates typical automobile dynamics. This scenario has been performed using a truck-mounted software receiver and a Magellan GPS PROMARK X handheld receiver. The second scenario tested the software receiver during a simulated rocket flight.

For the first test, the software receiver was placed in the back of a truck, and its antenna was mounted on the truck's roof. A second GPS receiver, a Magellan handheld receiver, was held situated just outside the passenger window. The separation between the two receivers' antennas was about 2 meters. The truck was driven for roughly an hour, during which both receivers collected 1-second-sample pseudorange and Doppler shift data.

The time history of the truck's position is required in order to determine the accuracy of the software receiver. Additionally, the time history of the vehicle velocity and acceleration are of interest. The truck's position, velocity, and acceleration have been derived from the Doppler shift measurements of the Magellan receiver. These Doppler shift measurements have been used to compute the receiver's velocity using post-processing software. The receiver velocity and the numerically differentiated acceleration are shown in Figure 7. These velocity and acceleration profiles represent typical automobile dynamics. The maximum velocity is 30 m/s and the maximum acceleration is 2 m/s^2.

The "truth" receiver position time history has been determined by integrating the Magellan's velocity solution time history from a best-fit initial point. This position time history is more precise than the pseudorange-derived position time history from either receiver, which is why it has been designated as the "truth" position time history.

Figure 9 shows the time history of the norm of the difference between the software receiver's position and the

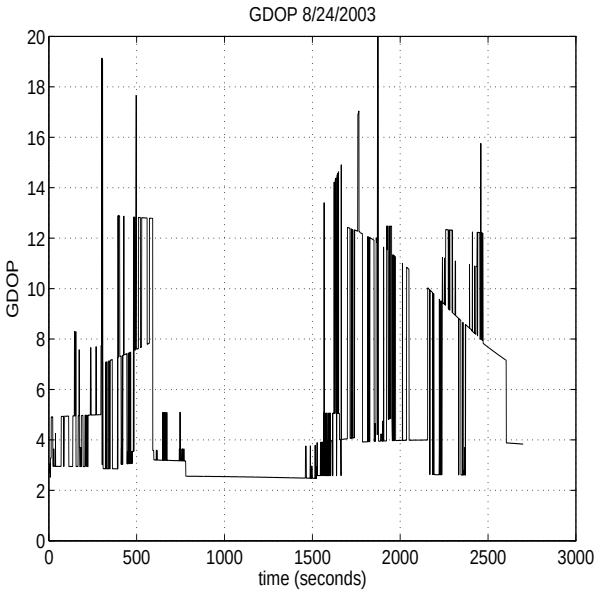

Figure 8: *GDOP during the automobile test.*

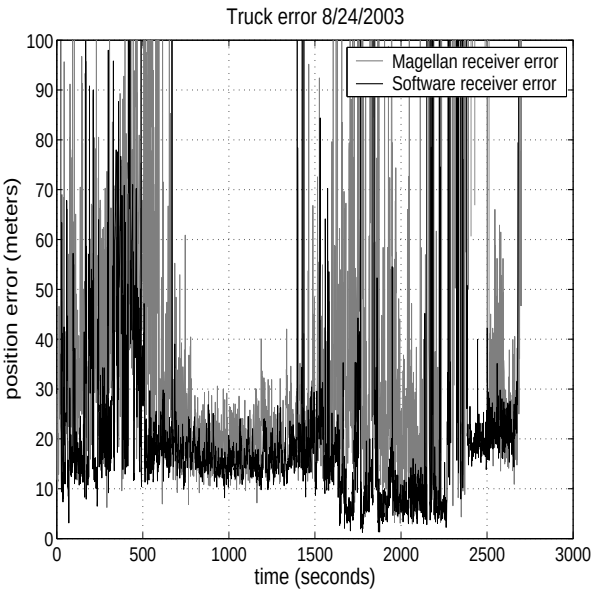

Figure 9: *Position error of the software receiver and a hardware receiver during typical automobile dynamics.*

"truth" truck position. This figure also shows the time history of the norm of the difference between the Magellan receiver's pseudorange-derived position and the "truth" truck position. Both navigation solutions have been computed by post-processing pseudorange observables using the same set of satellites in order to make for an equal comparison.

Figure 8 shows the GDOP time history. This time history has been computed using the "truth" truck position and the mutually available satellites. Because of obstructions, such as buildings and trees, only 4 or 5 satellites were mutually visible to the receivers during certain times. These times coincide with increases in GDOP that are seen during the beginning and end of the test. These increases in GDOP correspond to the increases in position error as seen in Figure 9.

Post-processing of pseudorange observables for both receivers has been used to generate navigation solutions for this case solely because this technique allows one to ensure that both receivers' solutions use the same satellites, which causes them to have the same GDOP. This constraint makes the accuracy comparison a fair 'apples-to-apples' comparison. The receivers both successfully generated navigation solutions in real time, but these solutions have not been used because of the lack of a guarantee that they would always use the same satellites. In actuality, had the real-time solutions been used rather than post-processed solutions, significant differences in GDOP would have occurred only a few times and only temporarily because both receivers were generally able to track the same satellites for most of the time.

Both receivers performed similarly in this test. Differences between the two receivers can be attributed to three sources. First, the pseudoranges were not sampled at the exact same times. With a 1-second sampling rate, the relative sample times may differ by as much as 0.5 seconds. Second, the receivers used different antennas separated by 2 meters. This produces a 2-meter bias in the software receiver's position error. Third, one receiver has a hardware correlator and the other has a software correlator. This test demonstrates that a software receiver can perform as well as a hardware receiver in low dynamic conditions.

The second dynamic test scenario involves a simulation of a rocket flight. This simulation was performed on a 12-channel single-frequency simulator. Figure 10 shows the velocity and acceleration profiles of the simulated test flight. The maximum velocity is 300 m/s and the maximum acceleration is 50 m/s^2. An acceleration limit of 5

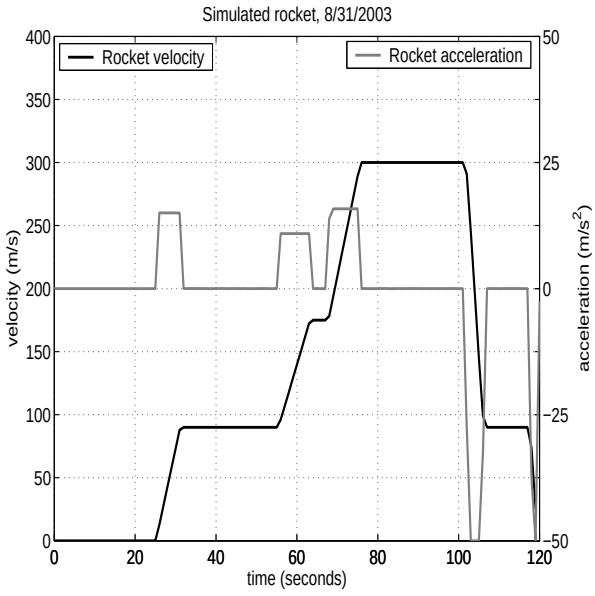

Figure 10: *Simulated rocket dynamics used to test the software receiver.*

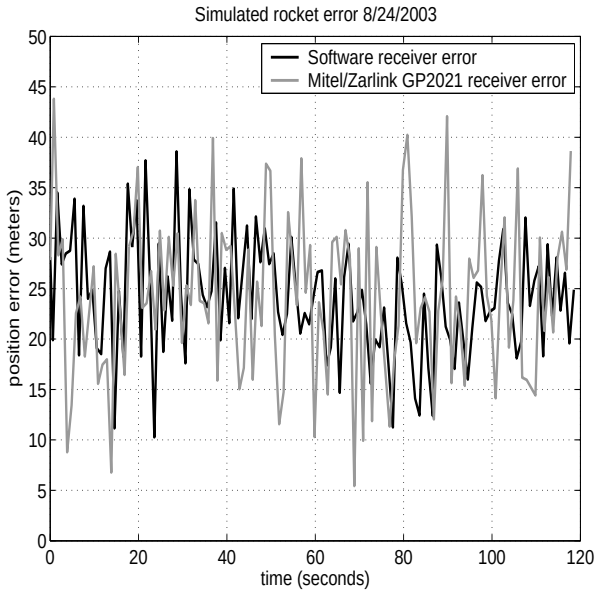

Figure 11: *Position error of the software receiver and a hardware receiver during a simulated rocket test.*

g's has been used because it does not require re-tuning of the receivers' tracking loops.

The hardware receiver in this test is the same one used in the static test scenario. This receiver is similar to the software receiver, except for its use of a digital hardware correlator. This test was run twice; once with the software receiver and once with the hardware receiver.

Figure 11 shows the position errors from the software receiver and the hardware receiver. The errors have been computed by taking the norm of the difference between the real-time receiver position time history and the simulated position time history. During the test, both receivers remained locked on to all of the satellites in view. The errors for both receivers are similar in magnitude. The relatively large error bias is most likely due to the average GDOP of 5.1. These results further the notion that a software receiver performs as well as a hardware receiver in a dynamic environment.

SUMMARY AND CONCLUSIONS

A 12-channel real-time software GPS L1 receiver that runs on a common PC has been tested. The hardware consists of a RF front end bit-grabber card, a data acquisition system, and a PC with a 1.73 GHz AMD Athlon processor running RT-Linux. The software consists of data acquisition code, a software correlator, and GPS software that provides the typical GPS functions such as navigation and tracking.

The computational requirements of the software receiver have been demonstrated with of 1-bit and 2-bit RF data sample sizes and processor-specific assembly language (MMX) instructions. Depending on the combination, the 12-channel real-time software receiver consumes 19-40% of a 1.73 GHz CPU's processing power.

The navigation performance of the software receiver has been shown to be similar to that of a hardware receiver. This has been shown in both a static scenario and two dynamic scenarios. In the static scenario it has been shown that the measured pseudoranges for both receivers are similar and that navigation accuracy is on the order of 5-10 meters. In dynamic tests, the software receiver also has navigation solution errors that are similar to those of hardware receivers. These tests demonstrate that a software receiver can be used in dynamic situations with no loss in performance.

ACKNOWLEDGEMENTS

Research at Cornell University was funded by the Office of Naval Research under grant N00014-92-J-1822.

REFERENCES

1. Akos, D.M., P-L. Normark, P. Enge, A. Hansson, and A. Rosenlind, Real-Time GPS Software Radio Receiver, *Proc. of the Institute of Navigation National Technical Meeting*, Long Beach, CA, January 22–24, pp. 809–816.

2. Akos, D.M., P.-L. Normark, A. Hansson, A. Rosenlind, C. Stahlberg, and F. Svensson, Global Positioning System Software Receiver (gpSrx) Implementation in Low Cost/Power Programmable Processors, *Proc. of the Institute of Navigation National Technical Meeting*, Salt Lake City, UT, September 11–14, 2001, pp. 2851–2858.

3. Ledvina, B.M., M.L. Psiaki, S.P. Powell, and P.M. Kintner, A 12-Channel Real-Time GPS L1 Software Receiver, *Proc. of the Institute of Navigation National Technical Meeting*, Anaheim, CA, January 22–24, 2003.

4. Ledvina, B.M., M.L. Psiaki, S.P. Powell, and P.M. Kintner, Bit-Wise Parallel Algorithms for Efficient Software Correlation Applied to a GPS Software Receiver, *IEEE Trans. Wire. Comm.*, in press.

5. Thor, J., P.-L. Normank, and C. Stahlberg, A high-performance real-time GNSS software receiver and its role in evaluating various commercial front end ASICs, in *Proc. of the Institute of Navigation GPS*, Portland, OR, Sept. 24–27, 2002, pp. 2554–2560.

6. Van Dierendonck, A.J., "GPS Receivers," in *Global Positioning System: Theory and Applications, Vol. I*, Parkinson, B.W. and Spilker, J.J. Jr., eds., American Institute of Aeronautics and Astronautics, (Washington, 1996), pp. 329-407.