Embedding Linux to Track Concealed Weapons

Alexander R. Perry, *Quantum Magnetics*
John F. Sturtz, *Codeweavers*
David Walsh, *Vista Clara*
Brian Whitecotton, *Quantum Magnetics*
alex.perry@qm.com http://www.qm.com
jfs@codeweavers.com http://www.codeweavers.com
davewalsh@vista-clara.com http://www.vista-clara.com
brian.whitecotton@qm.com http://www.qm.com

Abstract

A football-sized array of magnetic sensors can detect blades, knives, handguns and vehicles in the vicinity. Immediate analysis can show the concealed target as a red blob that is superimposed on a video image. This paper reviews why Linux was the only viable platform, describes the software and summarizes the algorithms being used.

1 Introduction

After initial processing by a Digital Signal Processor (DSP), the reduced data rate will fit through an asynchronous serial port at $115\,kb/s$ for full analysis and tracking in the host computer. The host must ...

- reliably retrieve the streaming data from the serial port
- filter and scale the data according to factory calibration data
- analyze the data for object parametrics, position and orientation
- translate the position into user-calibrated video alignment
- mark the threatening weapon's position on the camera image

Figure 1: *System tracking hidden gun*

Some frames, grabbed from the video stream during a demonstration, are shown in figure 1. The demonstrator was holding the gun in his right hand. The blob obscures the actual position of the gun and the red color indicates that the magnetic signature matches a dangerous object category.

For details on the underlying technologies, readers are referred to the literature on magnetic tensor gradiometry[1], magnetic position extraction[2], threat identification[3] and other applications[4]. This paper only summarizes the algorithms to show the computational impact of their numerical implementations.

The software development for this project was initially conducted using Microsoft tools to take advantage of existing source code. The prior project recorded raw data without any calculation and subsequently used modelling software for analysis. The central part of this paper reviews the new Linux-based implementation for this project, including descriptions of the problems that were encountered and shows how each problem was resolved as the code base for the project was migrated to Debian GNU/Linux.

1.1 Background

Most weapon detection systems range in price between $500 and $5000 depending on features. The end user of the system expects to treat it like any other embedded system, where power cycling is a routine activity and the unit is turned off at the end of the day by pulling the plug. As a result, systems need to use journaling filesystems or similar.

Battery powered systems need to run for a full working day, so the choice of a power efficient processor (such as PowerPC and StrongARM) is appropriate for such applications. Few operating systems are both portable and scalable.

In contrast, some situations require high reliability for 24/7 use. Windows NT and 2000, the Microsoft releases recommended for such applications, require considerable processor power, memory and disk space just to run their own infrastructure. The cost of those resources is significant compared to the total bill of materials cost of an installed weapon detection system. We failed to find technical benefits to justify that expense.

2 Object Reporting

Each object detected by the analysis generates a stream of data records. Each record has a timestamp that identifies when the raw data was actually measured, the three dimensional location of the object in polar coordinates, and additional tags. These tags identify the class of object, the orientation and other factors to ascertain whether it is dangerous, and thus what color code should be used for the blob in the video overlay.

The timestamp is naturally always a tiny fraction of a second old, so the software must extrapolate from recent data records to determine where the object is likely to be at the time that the next video frame is generated by the camera. This ensures that the blob is correctly drawn for moving people and objects.

2.1 Image Overlay

In the initial Windows-based software, a frame grabber was used to convert a video tape (from a VCR) into a directory of bitmap files. These bitmaps were matched to the timestamps and the targets marked up by modifying individual pixels. The directory was subsequently emitted to a VCR and also converted into a digital video for computer presentation. The hardware cost of systems to implement this approach in real time was prohibitive.

For real time, instead, a video mixing unit superimposes the computer's display onto the camera image using a 'bluescreen' mode to ensure that most of the computer display was invisible. The mixer output is a video signal that has threat objects marked in, ready for display on a security monitor and/or recording by a VCR for subsequent replay or digitization. The image in figure 1 was grabbed from such a tape.

A small program in Microsoft's Visual Basic painted the entire screen blue, then accepted a sequence of target coordinates and moved a color-selectable blob to the desired location. In operation, we found that the blob could only be moved at a fraction of the video framerate (about 10 fps), so that the motion appeared jerky and was late. This jerkiness made it difficult to match the blob against a crowded room and thereby determine which person is carrying the gun.

The low frame rate was in part due to the use of a Rapid Application Development tool, compared to a C compiler. The scientists and other non-programmers on the team were comfortable with VB or command line GCC, but found the Visual C environment overwhelming and didn't use it.

Under Linux, we implemented a trivial equivalent using the `ncurses` library and its suite of graphics characters. Even an old Pentium 60 was able to achieve three times the video framerate for position updates, so we have not yet optimized from that library to using the underlying `terminfo` directly. Although the positioning accuracy is one character cell, careful choice of the VGA console mode achieves resolution comparable to the video camera alignment.

The text-based Linux version offers the rapid and smooth tracking of the target that is critical to reliable identification.

2.2 Video Alignment

The system will, in general, operate without any need to communicate with the user. However, the configuration procedure at the user site must determine the parameters for converting magnetic unit polar coordinates to the video image screen coordinates.

It is important to make this easy for the user to do. The users need to feel comfortable moving system components around to meet their needs. They also need to check the parameters quickly whenever a component might have been knocked out of alignment.

Although keyboard and mouse have shown themselves to be effective user input devices for a desktop situation, they are limiting when embedded in an open area setting. Keyboards are very hard to use when an ergonomic position cannot be achieved and maintained, as is usually the case if the user cannot sit down. Mice require a flat surface to operate and most variants do not survive a dirty industrial environment well. Thus, a user interface that requires these devices (or emulations thereof) is usually inconvenient to the users.

A standard Linux system does not have a single native user interface model, so we are free to select user input and output devices to suit our environment and needs. A joystick (whether analog or USB) was found to be a sufficient input device, while a VCR-style 'On Screen Display' met our output device needs.

Suitable joysticks are cheap and can be picked up for $5 or so. There is less of a problem with damage, loss or theft compared to handheld keyboards and clip-on mice with $100 prices. The hot-plug capabilities of USB are also helpful.

2.3 Data distribution

Data delivery from the analysis was by function call under Windows, limiting the multitasking that could be achieved with low latency. We usually use TCP sockets under Linux, with one object data record being stored in each packet, so that both the bandwidth and the packet rate are predictable and low. This also permits the computer that draws the video overlay to be physically separate (i.e. next to the camera) from the one that controls the hardware (i.e. next to the sensor).

In some cases there are many data users, such as multiple video camera viewpoints being overlaid or a supervisor station, in which case the data records can be delivered using UDP and the broadcast address for the local ethernet segment. Loss of occasional individual data records does not significantly impact the video overlay, since it will extrapolate from available data.

This use of network protocols empowers the local users to structure their environment to meet specific needs, but it is important to either use an isolated network or a local firewall gateway to ensure that the display system cannot be spoofed by a remote attacker and thereby permit a threatening weapon to pass through undetected.

The built-in IP chains/tables support in the Linux kernel permits the system to safely operate autonomously, yet still be on the local network for delivering status reports and other application specific traffic to a central security monitoring station elsewhere in the facility.

3 Object Detection

The hardware does not directly measure the object position. The contents of each object data record must be deduced by careful calculations from the raw magnetic information that is reported by the magnetic sensors.

3.1 Analysis

Theory says that an individual magnetic sensor pointing in direction $\hat{n} = (n_x, n_y, n_z)$ and observing a magnetic source of strength m oriented in the direction (m_x, m_y, m_z) at a distance $r = (r_x, r_y, r_z)$ will observe a magnetic field B

$$B = \frac{\mu_0}{4\pi|r|^5}\left(3(m \cdot r)(r \cdot n) - (r \cdot r)(m \cdot n)\right) \tag{1}$$

An array of S sensors, numbered $s = 0 \dots S-1$, usually has all the directions n_s known and fixed. The sensors are not all in the same location, so their distances r_s to the magnetic source will not be equal. These distances are the difference $r_s = p - o_s$ of the sensor positions relative to the center of the array o_s and the magnetic source position relative to that center p. Thus,

$$m \cdot r = (p_x - o_x)m_x + (p_y - o_y)m_y + (p_z - o_z)m_z \tag{2}$$

We can write equation 1 for each of the magnetic sensors. We have S equations that relate S known values of B_s with $6S$ known constants n_s, o_s and 6 unknown values $p = (p_x, p_y, p_z)$ and $m = (m_x, m_y, m_z)$. Providing we have $s \geq 6$, a solution is possible. The computation is not merely non-linear due to the dot-products such as equation 2, but especially due to the term $|r|^5$ which expands to

$$\frac{1}{|r|^5} = (r \cdot r)^{-2.5} \tag{3}$$

This can be solved using an iterative descent by brute force, but this has an unpredictable execution time and is thus inappropriate for a system seeking a predictable performance.

A popular method involves separating both p and m into their unit vectors $\hat{p}, \hat{m}$ and magnitudes $|p|, |m|$ and rewriting the equations. Assuming we place our sensors in an array pattern that makes tracking possible, we can compute an intermediate result called the *magnetic field tensor gradient*. This result is nine linear combinations of the B_s values (the actual combinations are dependent on the array pattern) which can be used to deduce $\hat{p}$ and $\hat{m}$ using some nasty calculations[2]. Although nasty, the execution time can be predicted and so this calculation is preferred for a real time system. Once those values are known, a separate computation step uses all the original B_s to compute the m and p.

This entire calculation sequence was prototyped in MATLAB to analyze recorded data files, working from Tucson AZ. The data was usually acquired by personnel in San Diego CA, since someone needed to move the desired target object past the sensor array in a predictable path. For certain kinds of testing, the object could be hung over the array as a pendulum. This could be left swinging for remote algorithm developers to collect additional data whenever they needed it.

Once the algorithm was demonstrably tracking objects, it was translated directly into C and installed on a Linux computer. After verifying that it generated the same tracking results as the MATLAB version, lab staff could stream data from any of the hardware systems to the algorithm development computer over the network (and/or VPN). The track results were delivered to a nearby computer with a 3D graphics and perspective capable display for inspection.

Now that the algorithm is stabilized and reliable, it may be installed locally on all computers that are easily fast enough to perform the floating point math. In general, therefore, Pentium-class and above have the tracking algorithm, while 386 and 486 do not.

3.2 Filtering

The primary purpose of the filter is to eliminate irrelevant frequency bands from the raw data. This band-limits the data stream, which also allows some decimation to be applied without loss of signal. By performing the filter and decimation stages on the data before the numerically intensive analysis, the processor load is minimized so that a lower price computer chassis can be used.

The secondary purpose of the filter module is decoupling, through use of independent input and output fifo structures. The demanding real time operation of the serial port concludes by dumping the raw data into the input fifos.

A separate thread, at a lower priority, retrieves blocks of data V from the input fifo and applies a sequence of FIR filters and decimations before writing smaller blocks of data B into the output fifos. The FIR calculation simply involves multiplying the most recent $i = 0 \ldots n-1$ readings $V(T-i, c)$ on each channel c with a constant array of n coefficents $C(i)$ that were loaded from file. The decimation is implicit by simply neglecting to perform that calculation for the $2 \ldots d$ output readings that will subsequently be discarded.

$$B(j,c) = \sum_{i=0 \ldots n-1} C(i) V(d \times j - i, c) \quad (4)$$

Clearly, the filter calculation takes linear CPU time for the volume of data and is completely predictable. The real time needed for execution depends on the system cache (or worse, paging) performance. The filter thread is important, since it reduces the amount of memory in use and thus improves cache hits. It is also using large amounts of memory to store the historical raw data and so is best batched.

When the filter batch thread doesn't have anything to do, the output thread builds magnetic data records in the format expected by the analysis software and deposits them into a socket for delivery. This will trigger the computational load that was discussed above.

The software infrastructure to implement this scheme was written in St Paul MN and tested on the hardware being developed and located in the San Diego CA labs. After a few weeks, the San Diego people learned not to unplug hardware without checking for remote logins first, and the St Paul people learned to phone and find out whether someone had removed the hardware before blaming their code.

The source code can be compiled and run, without changes, on full x86 computers with linux and also on the Dragonball microcontroller with uClinux on the *UCSIMM* from (now) *Lineo*.

3.3 Serial Port

Initially, the serial data from the hardware was received by a *Pentium II 450* computer, which ran a succession of versions of Windows while we attempted to get the serial port to accept data without regular and frequent data loss. After consistently failing, we switched to Linux on a variety of machines and have had few problems since. What was difficult about the situation that caused Windows to fail, where even a 80386 running Linux was sufficient ?

The raw data stream consists of 72 byte records that arrive 100 times per second over a 115200 baud serial cable with hardware handshaking. Each eight bit character is surrounded by two framing bits, so the link is running at 62% of its theoretical capacity with $3.7\,ms$ of unused time in every $10\,ms$ interval. Most serial port hardware implementations have a $16\,byte$ fifo, so five interrupts are needed to transfer one data record from the wire into memory (unless a nonstandard polling architecture is used).

About $1.4\,ms$ after the interrupt is raised, if the operating system has still not serviced the interrupt, the fifo becomes full and data delivery from the wire is paused by the hardware handshake. We are now consuming that $3.7\,ms$ buffer of unused time, which is shared among five interrupt events. Thus, the operating system must achieve an average interrupt response time of $2.1\,ms$, and even one response time in excess of $10\,ms$ will always cause data loss.

Our reference test used a simple *Visual Basic* program that grabbed data from the serial port as fast as possible, inspected it to determine whether any loss had occurred, then immediately discarded it. No user (or GUI message) communication occurred while the test was in progress. Data loss was observed to occur on average once every couple of minutes with the network cable plugged in, reducing to every ten minutes or so with no network and every normal user configurable item optimized. Touching the mouse or keyboard caused data loss events every ten seconds or so.

We modified the data stream so that it was simply delivering timestamps to the host computer with no other information content. The test program was extended to gather statistics. We found that there were gaps due to missing data in excess of ten seconds duration several times per day. Although we experimented with the advanced configuration choices and considered non-standard hardware installation, these were not viable long term solutions due to the cost and complexity impact on the end user.

On occasion, we still need to run the original Visual Basic software that predates the Linux port. This is now reliably achieved by having it connect to a serial port emulation provided by the *Dialout/IP* client from *Tactical Software*. Under the emulation is a raw TCP/IP connection that goes to a Linux computer, which actually operates the serial port and has a fifo implemented in software that can hold several minutes of data.

An old 80386 class computer (running Debian 2.1) was found to be comfortably sufficient to provide reliable serial I/O and client network communications. On this machine, we benefitted from the `irqtune` utility in the `hwtools` package, which can place the serial port interrupt at the highest priority. When this was done, we could run other software from the 80386 console terminal without impacting serial port operations.

4 DSP development

Low cost single board and single chip computers offer fast hardware I/O with chaining DMA and signal processing features such as multiply-accumulate instructions. Often, these units can be purchased with Linux preinstalled.

At the time that the electronics were designed, several years ago, these options were not available at reasonable cost. Therefore, a dedicated small signal processing chip was designed into the electronics. The *Analog Devices* 21xx series processor has two synchronous serial interfaces, one of which communicates with the sensor electronics and the other with the host processor.

A team containing several programmers developed the code in parallel by segregating the project along functional lines. The Windows-based development tools would get confused when importing changes made by another programmer, such as when a shared header file needed to be modified. Some code loss had occurred during a previous project, so this project utilized the DOS-based tool chain from the vendor. This tool chain made no assumptions about the state of the underlying source files and relied on file timestamps to control the build process, thereby increasing reliability.

Each developer used an account on a Linux computer with samba services to expose their directory to any desktop computer. The tool chain would execute in the directory of the individual's SMB share, resulting in a self-contained development environment. Each user would log into the linux machine to run cvs or take advantage of the more flexible search and indexing tools available at the unix command line.

This allowed rapid parallel development

with no code loss. The Network Time Protocol (NTP) was used to synchronize the clocks of the various computers, but this was unreliable on Windows98 computers because they tended to believe whatever any of the other network servers had most recently told them. Occasionally, this resulted in build errors when cvs timestamps were slightly misaligned, but not sufficiently for *make* to provide a warning.

The download and diagnostic software was implemented on linux machines, using serial and parallel port interfaces. The DOS tools were installed in a virtual hard disk image under *dosemu* and embedded into a Linux makefile, so that all developers could use a single identical execution environment for the tool chain from any computer. In addition to allowing the developers to telecommute, this permitted them to make code changes from whichever computer happened to be closest to the hardware's location.

Thus, typing `make test` can rebuild the DSP code, download and verify it, reboot the DSP, check it wakes up, then run low level regression tests for obvious errors. It then checks the source tree for the protocol drivers in the host, reinstalling them if necessary, before running a high level functional test that verifies reliable acquisition. This kind of cross-tool automation in a 'make' environment is trivial under Unix. Since toolchains are designed by their vendors to be completely self-contained under Windows, this kind of easy project integration is generally impossible.

A single Linux computer exported the directories to the network, ran the tools under the DOS emulator and streamed data to and from local hardware test stations. The Pentium 75 system supported three simultaneous users with $16\,MB$ of RAM and $3\,GB$ of disk space. When two additional users were added, one of which was using *octave* and *gnuplot* for analyzing test results, the memory was doubled to $32\,MB$.

5 Conclusions

Looking forward to the integrated product, the most valuable feature of developing this system under Linux must obviously be the opportunity to move source around. We can develop and test code with fast commodity desktop computers and then migrate it directly into a microcontroller target.

Looking at the development process, we benefitted greatly from the remote access and administration. This allowed developers who work several timezones away to actively test their changes against the hardware in our labs.

In combination, they allow the project manager to select team members from a worldwide pool of skills, instead of being restricted to the engineers who reside in (or can relocate to) the lead company's city.

6 Acknowledgements

This work was funded in part by the National Institute of Justice, supporting development of new technologies to benefit state and local justice services, and the United States Navy, promoting research into new methods of detecting and clearing minefields.

7 Availability

More information about this technology is available from

`http://www.qm.com/development/weapons_detect_body.htm`

The new application software that was developed in support of this system is currently proprietary and closed source. Driver enhancements have been sent upstream, of course. The Open Source development model is not yet useful in this area, due to a shortage of relevant expertise in the technical teams of the collaborating organizations. Hopefully, this will change in the new future ?

References

[1] P.V. Czipott, R. Matthews, A.R. Perry, R.H. Koch and G.I. Allen, "Development of a man-portable room-temperature gradiometer: phase II, portable and fieldable prototype" in *Proc. SPIE 3711 (Information Systems for Navy Divers and Autonomous Underwater Vehicles Operating in Very Shallow Water and Surf Zone Regions)* in press, 1999.

[2] W.M. Wynn, "Detection, localization, and characterization of static magnetic dipole sources," in *Detection and Identification of Visually Obscured Targets, C.E. Baum*, pp. 337-374, Taylor & Francis, Philadelphia, 1999.

[3] A.R. Perry, "Magnetic Localization and Real-Time Tracking of Concealed Threats" in *SPIE conference proceedings* 2001.

[4] Y. Dalichaouch, A.R. Perry, C. Moeller, and P.V. Czipott, "Development of a room-temperature gradiometer system for underground structure detection and characterization," to be published in *Proceedings of SPIE, "Aerospace/Defense Sensing, Simulation and Controls," 24-28 April 2000, Orlando, Florida.*