LA-SUB--93-224

Cost/Benefit Analysis for Video Security Systems

1 Introduction

Dr. Don Hush and Scott Chapman, in conjunction with the Electrical and Computer Engineering Department of the University of New Mexico (UNM), have been contracted by Los Alamos National Laboratories to perform research in the area of high security video analysis. The first phase of this research, presented in this report, is a cost/benefit analysis of various approaches to the problem in question. This discussion begins with a description of three architectures that have been used as solutions to the problem of high security surveillance. An overview of the relative merits and weaknesses of each of the proposed systems is included. These descriptions are followed directly by a discussion of the criteria chosen in evaluating the systems and the techniques used to perform the comparisons. The results are then given in graphical and tabular form, and their implications discussed.

The project to this point has involved assessing hardware and software issues in image acquistion, processing and change detection. Future work is to leave these questions behind to consider the issues of change analysis - particularly the detection of human motion - and alarm decision criteria. This project break point is shown in Figure 1. The criteria for analysis in this report include:

- Cost
- Speed
- Tradeoff issues in moving primative operations from software to hardware
- Real time operation considerations
- Change image resolution
- Computational requirements

2 Competing Architectures

The current system, developed at Los Alamos National Laboratories (LANL) by Steverson, is seen in Figure 2. Ten consecutive frames are digitized and passed directly to the controlling CPU. Here they are averaged to reduce noise before being compared and thresholded.

This has the advantage of reducing noise without loss of resolution. Comparison is done by holding the previous frame and subtracting it from the current image. This leaves an image composed of difference values that are then thresholded. Thresholding involves selecting a critical value. Pixels that have an absolute value of the difference below the threshold are not considered to represent significant change. Pixels above the threshold value are considered to show real change, above simple noise. Change detection is accomplished in this same manner in all three architectures to be compared. The final step is to apply a median filter to further reduce the noise. This system has the advantage of low cost (minimal hardware), simplicity and the fact that it is in place. Its primary disadvantage is that cannot be used to perform real time surveillance because of the temporal averaging.

In Figure 3, the Video Motion Detection (VMD) system (also known as TCATS) is shown. This system was developed at Sandia National Laboratories (SNL) by Dr. Don Hush and Dr. Greg Donohoe. Here, a frame is acquired, digitized and briefly stored. After change detection (and in parallel with subsequent operations) the reference frame is refreshed by combining it with a low-weighted version of the current frame. The reference frame, thus, is an exponentially weighted time average of the input frame. This is a slower, more subtle update of the reference than in either of the other two architectures. The change detection is passed through a Low Pass Filter (I) or a High Pass and Low Pass (II) combination to remove noise, then passed to a controlling CPU. There, it is subsampled and thresholded. Subsampling creates a new image that is smaller than the original by a factor of 64. Each pixel in the smaller image is an average of an 8x8 field of pixels in the original difference image. After thresholding, any pixels that remain active are considered to represent real change. This architecture has two main advantages. First, it can be implemented in real-time and has been shown to work well with continuous real-time data. Secondly, the spatial High Pass Filter is useful in reducing alarms due to changes in illumination (shadows, etc.) A disadvantage is that spatial averaging results in a lower resolution.

The third architecture is seen in Figure 4. Known as MIPS, it was developed at UNM, for SNL, by Dr. Don Hush and Cliff Wood. Here, we begin by moving the Low Pass Filter (and possible High Pass Filter) and image subsampler up front immediately following digitization. The advantage to subsampling as early as possible is that it reduces the number of computations required in all subsequent processing. This spatial averaging reduces resolution, but this is not considered critical in this application, since the concern is for large-scale changes in the scene. The remainder of the processing is as discussed above - difference calculation followed by thresholding. Note that the reference frame is replaced each iteration by the current frame, as it is in system 1.

3 Gain and Offset

The figures for all three architectures show gain and offset adjustments at the analog input signal. These critical terms are intended to adjust the input to make better use of the digitization process. The gain term is a weighting constant that will spread the digitized values over a larger number of the available digital values, resulting in greater resolution in the digitized image. The offset term is intended to shift the adjusted image so that changes in the mean value of the image, introduced in the gain term, will be eliminated. An example of the effect of the gain and offset terms is presented in Figure 5. The inclusion of these terms is critical in that information is lost during quantization and cannot be recovered in subsequent processing. This effect is particularly destructive in attempting to detect change in the scene. Gain and offset terms can reduce the loss in information by maximizing the entropy of the digitization process. Most digital image acquisition hardware allows for programmable control of gain and offset.

Not shown in these figures is the computation of offset and gain for the images being processed. Both terms are highly dependent on the input images. When using a camera as the analog video source it is common to adjust the amplitude and offset of the analog signal by adjusting the iris setting on the lens. The iris controls the amount of energy (light) incident on the sensor array (CCD array). Typically the iris is adjusted by hand so that the image "looks good" on the monitor. Also, many cameras come with an "auto-iris" mechanism which automatically adjusts the iris to maintain a specified average power level in the video signal. In either case the video signal produced by the camera is not always appropriate for digitization, even though it may "look good" on the monitor.

There are numerous possible techniques for adjusting the offset and gain. A detailed study of these techniques is currently being performed by Professor Donohoe and his students at UNM, but for now let us describe a rather simple and straigthforward technique that has proven to work well in the systems in Figures 4 and 5. This technique searches for a setting of the offset and gain that will yield a digitized image with a desired mean and variance. The search is performed iteratively: first the offset is adjusted to achieve a desired mean, and then the gain is adjusted to achieve the desired variance. This process is repeated until both the mean and variance are within a specified tolerance of their desired values. Typically settings for the desired mean and variance are 127 and 3800 respectively.

4 Analysis

The comparison metrics chosen for this analysis are cost and speed. Both of these issues are more complex than the simple comparison of the three architectures presented. The primary impact on both cost and speed is the fact that the analysis of the proposed systems does not hold the hardware/software boundary constant, but allows it to move from as far left as possible in, Figures 2 - 4, to as far right as possible. The graphs shown in Figures 7 - 9 show that moving the boundary from left to right increases speed (decreases processing time) at the expense of increased cost.

Two other factors relate directly to speed and are presented in the analysis. The primary one is the question of whether to subsample the image - that is, to perform a spatial average of each 8x8-pixel section to reduce the image from 512x512 to 64x64. This has the double benefit of increasing processing speed in subsequent stages, while reducing the storage required to hold the image. The speed increase is only evident in the software domain, and so the hardware/software boundary should always adjoin the subsampling operation. The subsampling itself may occur in hardware or software. The disadvantage of subsampling is that resolution is lost. Therefore, this process should always be preceeded by a low pass filter to smooth out the data.

A more minor speed issue relates to the subtraction performed when the current frame is compared to the reference image. If this is performed in software, then the issue of signed verses unsigned subtraction has a fairly dramatic effect on the processing speed. The abs() function takes a significant amount of time. See the results below for further clarification. The analyses performed below consider only the unsigned values since the absolute change value must be examined at some point in the change detection.

Beyond the question of subsampling, the cost metric is affected by the variety of hardware available. This analysis is concerned only with the image processing hardware and does not include direct comparison of CPU's or software platforms. Two manufacturers were included in the comparison - DataCUBE and Imaging Technologies. Both are leaders in the field of real-time image acquisition and processing. The hardware available from these manufacturers are directly comparable in features, applications and price. Prices quoted in the tables below are based on catalog data acquired in January 1992 and are considered to be estimates subject to change.

Many other factors will go into the choice of an architecture for the high security environment. A few include the importance of low pass and high pass filtering, the merits of spatial vs temporal averaging, possible parallelism in the architecture, and slow vs immediate update of the reference frame. These contributors are not dismissed, but do not contribute directly to this analysis except as they effect the cost and speed of processing.

5 Procedures and Results

The speed measurements shown below centered primarily on those operations that are to be implemented in software. The primitives that make up the operations were defined and coded as distinct programs to measure their timing. (A sample program is given in Figure 10.) Each primitive except subsampling was performed on both a 512x512 and a 64x64 pixel data array. (The results are over-reported in that the LPF will always precede subsampling.) In order to avoid additional overhead and look solely at the primitive in question, data arrays were simply malloced. (As seen in Figure 6, the malloc() operation did not contribute significantly to the execution times of these primatives.) They were not initialized (except as noted below) and no image was read in. The values present in memory at the time of allocation were the values worked on. In order to eliminate the possiblility of overflow, data arrays were declared to be of type integer. The operations were run on Sun 4 Workstations during low-use hours. Execution times were measured using the UNIX 'profile' utility. Each task was profiled five times and the results averaged. Two sample profile runs are shown in Figure 6.

Table 1 presents the execution times as measured. In Table 2, the cost and features of the hardware required to perform the various operations are shown. Table 3 applies these results to each architecture over a variable hardware/software boundary and considering the issue of subsampling. Since the primitives displayed in Table 1 only apply to software implementation, all hardware operations are considered to occur at 30 frames per second (FPS). (This is the maximum rate at which these operations will occur.) Table 3 also takes into account the fact that all operations performed in hardware may occur in parallel as the available hardware is extended.

Note that the offset and gain operations are performed on the analog image as it is acquired and are not represented in Table 1. The primitives measured are:

- Subsample - A reduced 64x64 pixel image is formed by copying the pixel from every eighth row and eighth column (every 64th pixel) from the original image into the reduced image. The gains in software processing speed and data storage are thought to outweigh the loss in image resolution, particularly since large-scale change is what is being investigated in this project.

- Superpixel - Each pixel in a 512x512-pixel array is replaced by the average over an 8x8 array of pixels to produce a 64x64 image. This operation has the advantage of increasing software speed while decreasing storage. It should be the first operation in software or the last in hardware. A disadvantage - not considered to be critical here - is that resolution is lost in this spatial averaging.

- Subtract - Appears in all architectures when the current image is compared to a reference image to detect changes. Both a signed and an unsigned version of the subtraction are presented. The unsigned version is slower, but represents the sort of comparison that should be made in detecting change.

- Add - In the current LANL architecture, 10 consecutive frames are acquired and averaged. The addition of these frames will occur in parallel with the acquisition of the next frame. If the image is not subsampled, this operation is slower than the acquisition and therefore Table 3 considers only a single acquisition time followed by nine summations.

- Divide-by-Const - This primitive also occurs in the temporal averaging of architecture 1. The constant, here, is 10. A single divide is performed for each averaging operation.

- Threshold - The decision process in all proposed architectures. This operation involves a comparison and a value substitution for each pixel in the difference image. The result is a binary-valued array with pixels of sufficiently large change flagged.

- Convolve - The LPF and HPF filters are performed by replacing each image pixel with the weighted sum of its nearest neighbors. In this case, the weighting is unity and the number of nearest neighbors is 64, with a 8x8 square area providing the summation zone. This 8x8 kernel moves across the image, which has been placed in a framing image that is 7 pixels larger than the original in every direction. The borders of this frame are initialized to zero. It is the lower right pixel of the kernel that is replaced by the summation. As the kernel moves along it produces and image that is 7 pixels larger than the original both toward the right and towards the bottom.

- Median Filter - This filter serves well in cleaning up isolated pixels that differ widely from the general image. That is, speckle patterns and the like are filtered by this technique. Here, the center pixel of a 3x3 array of pixels is replaced by the median value of the array. The process involves sorting the array for each pixel that is replaced. A quicksort was chosen as it has the fastest average sorting time of known sorts. Nevertheless it is this sorting that makes this a very slow operation. An alternative that may speed things up is to sort only the kernel around the first pixel to be operated on. The values that will not contribute to the next pixel are known and can be flagged. These can be discarded in the next iteration and the three incoming pixels can be inserted in the proper order. This more sophisticated coding has not yet been tried, and it is not clear that the possible gains will be significant. Note that this operation also requires a framing image initialized to zero. This frame extends one pixel wide around the entire original image.

The cost analysis was performed by identifying the specific board from each manufacturer that would perform each specific primative. The prices of these boards were recorded. Table 2 presents, by function, the applicable hardware of both Imaging Technologies and DataCUBE, comparing price, functionality and storage. The analysis of the cost of the hardware is also presented in Table 3.

6 Other Issues

A variety of additional issues are of concern to the client. Not all relate directly to the cost/benefit analysis phase of this project, but all will be considered throughout the project.

- The software concerns involve data formats and compatibility, portabillity, object-oriented programming and designing for modular and flexible solutions. These issues will become more critical as the project moves away from the cost/benefit analysis stage.

- The issue of system networking will also be addressed throughout the project. The perceived advantages of proper networking are multifold. First, the computational load could be distributed to all the processors on the network, speeding up the image processing tasks by using otherwise underworked processors. Secondly, a cost reduction could be seen by placing all the required image processing hardware in a single machine for use by all stations. A third important advantage is in fault tolerance. If any single station fails, the tasks that it performs can be quickly distributed among the remaining stations without loss of continuity. Finally, it is simply easier to keep all systems current, in a network environment, through the systemwide distribution of updates.

- As the project moves from analysis to design, software will be developed using an object-oriented organization. This will provide an improved environment for the engineering of modularity, flexibility and extendability into the software.

7 Conclusion

The video security system that we are working towards in this project should be capable of monitoring several sites simultaneously. The system will be comprised of a host computer (probably a SUN workstation) that will reside at a location physically removed from the sites that are to be monitored. The question then arises as to how much of the processing should be done on site, and how much should be done by the host. One of the original purposes of this cost/benefit analysis was to provide information that would help in making this decision. The most cost efficient solution, and the one recommended here, is to do as much of the processing at the host as possible. This will prevent duplication of resources at each of the remote sites. At a minimum, the remote sites must be equipped with a camera and a transmitter (or some mechanism for transferring the video signal to the host). All of the remaining processing can be performed by the host (as long as it is able to process the information from each site in time to move on to the next site). Our analysis shows how hardware can be added to the host to help enhance its processing capabilities (if needed), and the computational cost incurred as a result. Aside from being the most cost efficient solution, performing all the processing at the host has other advantages. System upgrades are much easier since the only part of the system that will need to be modified is the host. System development is also easier if it is concentrated on one machine. Using a workstation (like a SUN) as a host makes it possible to take advantage of the numerous software tools available for program development, data display, and user interface development. The host should reside on high-speed network to make for easy communication with other hosts. This would have the added advantage of allowing a second host to take over if the primary host fails.

Operation	*Image Size*	*Time*	*Comments*
Subsample	512x512	60 ms	512x512 to 64x64 subsample
Superpixel	512x512	264 ms	512x512 to 64x64 spatial average
Subtract (signed)	512x512	640 ms	
Subtract (unsigned)	512x512	946 ms	
Subtract (signed)	64x64	10 ms	
Subtract (unsigned)	64x64	10 ms	
Add	512x512	676 ms	
Add	64x64	8 ms	
Divide-by-Const	512x512	428 ms	
Divide-by-Const	64x64	12 ms	
Threshold	512x512	94 ms	
Threshold	64x64	2 ms	
Convolve	512x512	17.78 sec	8x8 Convolution Kernel - all coefficients = 1
Convolve	64x64	346 ms	8x8 Convolution Kernel - all coefficients = 1
Median Filter	512x512	27.74 sec	3x3 Kernel
Median Filter	64x64	434 ms	3x3 Kernel

Table 1: Software Execution Times of System Primitives - These figures were derived by performing the UNIX service routine 'Profile' on all computational primatives. Each operation was coded minimally - see figure 10 - and executed 5 times on a Sun 4 work station at low use hours. The average time of execution for each primitive is represented here. Maximum resolution of the profile utility is 10 ms. Higher resolution terms given here are a function of the averaging process.

Operation	*DataCUBE*	*Imaging Technologies*
Digitization	DigiMAX $2450.00 30 FPS - 10 MHz 8-Signal Multiplexing Gain and Offset Control	ADI-150 $2995.00 30 FPS - 10 MHz 4-Input Programmable Gain and Offset
Storage	Framestore $3850.00 3 512x512 8-bit images Pan and Scroll	FB-150 $2995.00 2 512x512 images 8-bit (1 MByte) Pan, Scroll and Zoom or FB-150-1k 3495.00 8 512x512 8-bit images (4 MByte)
Convolution	MaxSIGMA $5100 Programmable Kernel Size	ALU-150 $1995 General Purpose IP Board Or RTC-150 $??? Real-Time to 4x4 Kernel Or RTMP w/Convolver $995.00 + $1495.00 General Purpose Computational Module w/Add-on Convolver Up to 8x8 Kernel
Subtraction	MAX SP $1550.00 General Purpose IP Board	ALU-150
Thresholding	MAX SP	ALU-150
Reference Frame	MAX SP, Framestore	ALU-150, FB-150
Median Filter	$3300.00 (Estimate)	$3300.00 (Estimate)

Table 2: DataCUBE/Imaging Technologies Hardware Comparison - Prices given are quoted from catalog information received in January 1992. These prices are intended as references only and are considered to be subject to change at any time.

Arch.	*1*	*2*	*3*	*4*	*5*	*6*	*comments*
I - LANL	$2450.00	$6300.00	$7850.00	$7850.00	$11150.00	N/A*	D.C. Cost
	$2995.00	$5990.00	$7985.00	$7985.00	$11285.00	N/A	I.T. Cost FB-150
	$2995.00	$6490.00	$8485.00	$8485.00	$11785.00	N/A	I.T. Cost FB-150-1K
	34.38 s	29.15 s	28.23 s	28.17 s	470 ms	N/A	Total System Run Time
II - VMD/ TCATS	$6300.00	$7850.00	$12950.00	$12950.00	$12950.00	$12950.00	D.C. Cost
	$5990.00	$7985.00	$10475.00	$10475.00	$10475.00	$10475.00	I.T. Cost FB-150 RTMP w/ Convolver (Only option considered)
	$6490.00	$8485.00	$10975.00	$10975.00	$10975.00	$10975.00	I.T. Cost FB-150-1K
	18.91 s	18.00 s	18.00s	250 ms	230 ms	170 ms	Total System Run Time No HPF
	36.69 s	35.78 s	18.03 s	290 ms	260 ms	200 ms	Total System Run Time With HPF
MIPS	$2450.00	$7550.00	$7550.00	$9100.00	$9100.00	$9100.00	D.C. Cost
	$2995.00	$5485.00	$5485.00	$7480.00	$10475.00	$10475	I.T Cost FB-150 RTMP w/ Convolver
	$2995.00	$5485.00	$5485.00	$7480.00	$10975.00	$10975	I.T Cost FB-150-1k
	18.91 s	1.71 s	1.71 s	1.14 s	230 ms	170 ms	Total System Run Time No HPF
	36.69 s	18.95 s	1.20 s	1.17 s	260 ms	200 ms	Total System Run Time With HPF

Table 3: 3-Architecture Cost/Speed Comparison - This table is a consolidation and reduction of Tables 1 and 2. Software execution times given consider only unsigned subtraction in the change detection operation. This is the slower subtraction, but is necessary to obtain absolute change values. Imaging Technologies costs consider both the 1 MByte and 4 MByte frame buffers, but only the RTMP board with the convolver option is considered. Numbers across the top of the table refer to the hardware/software break points shown in figures 2 through 4.

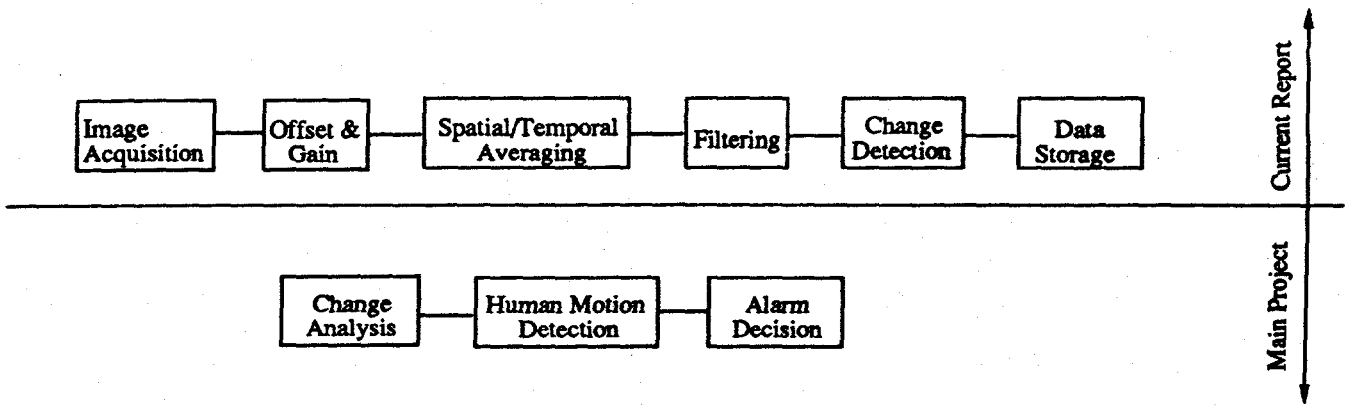
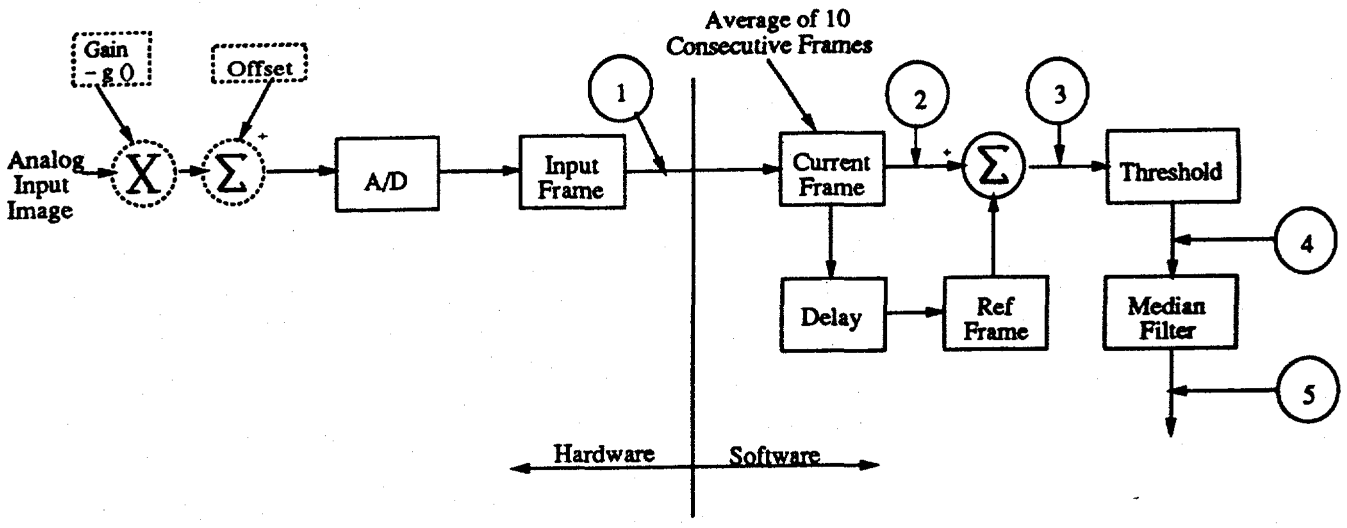
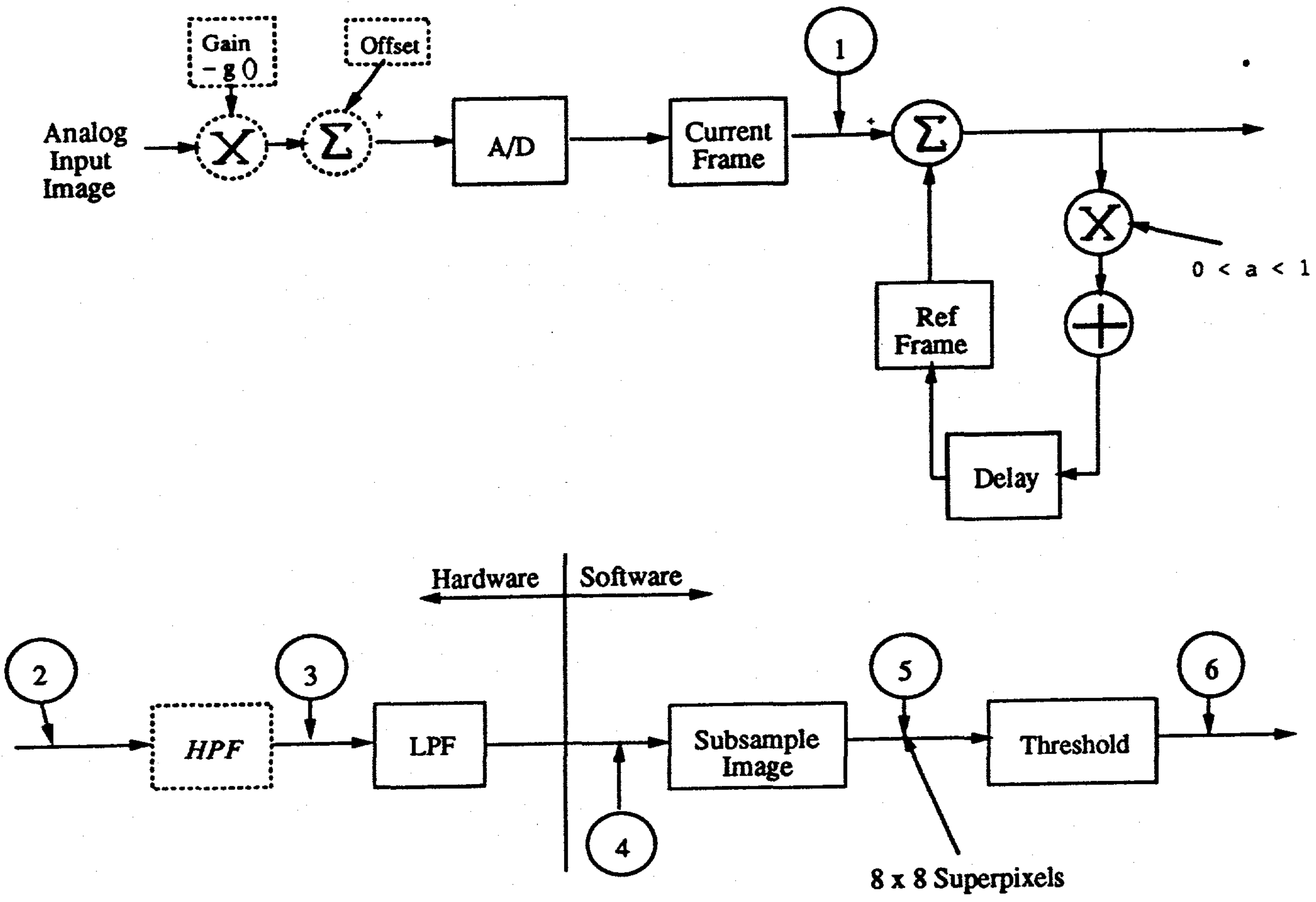


Figure 1: Project Overview



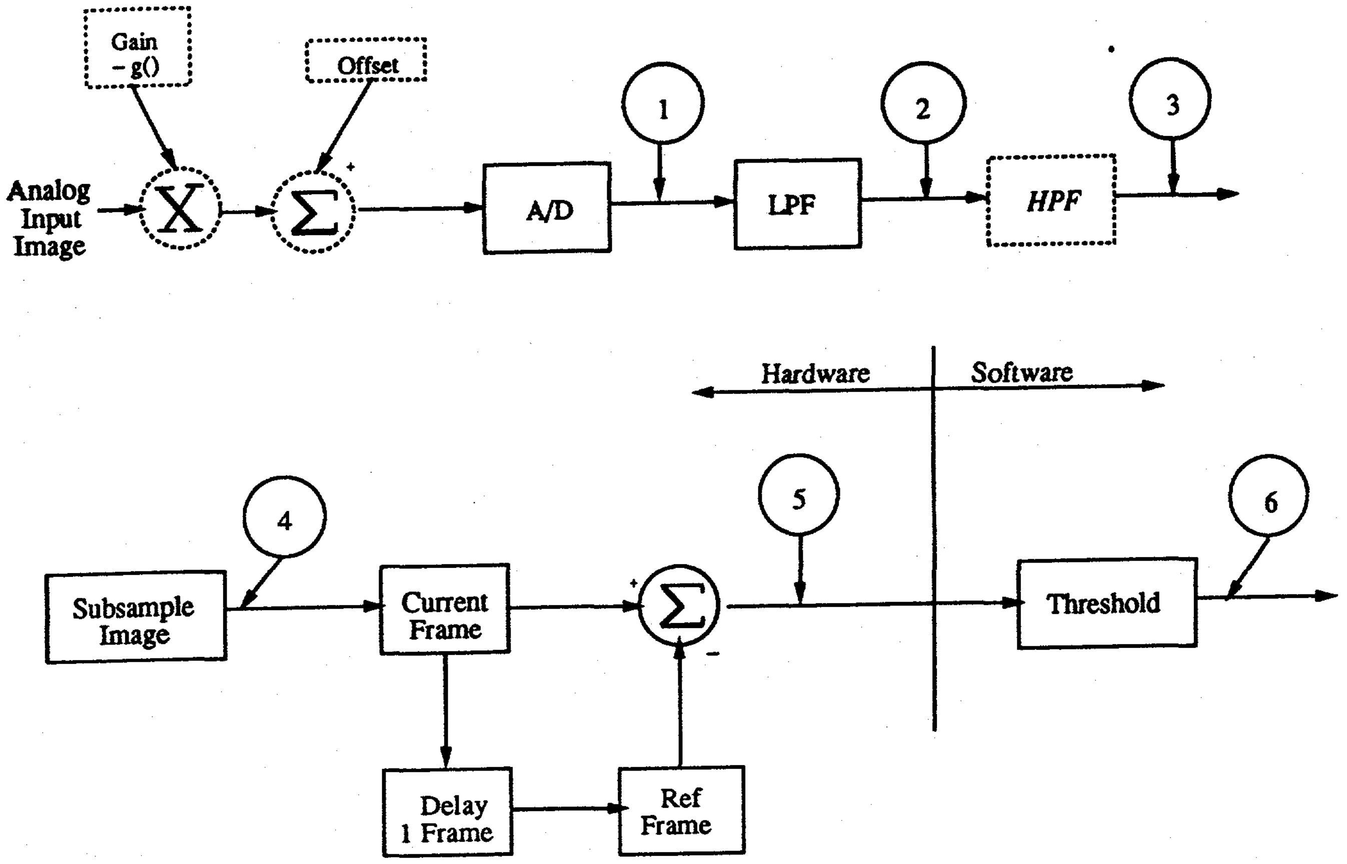
Circled numbers indicate hardware/software break points sampled in table 3.

Figure 2: LANL System



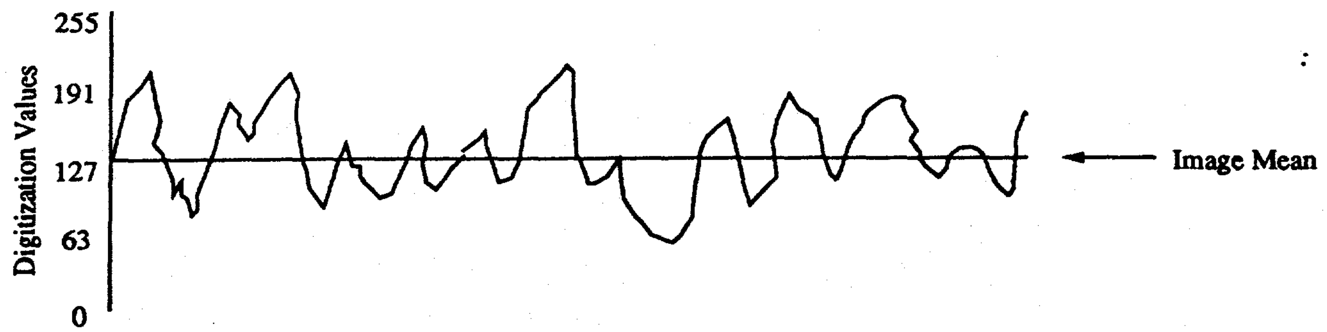
Circled numbers indicate hardware/software boundary test points refered to in table 3.

Figure 3: VMD - Video Motion Detection (TCATS)

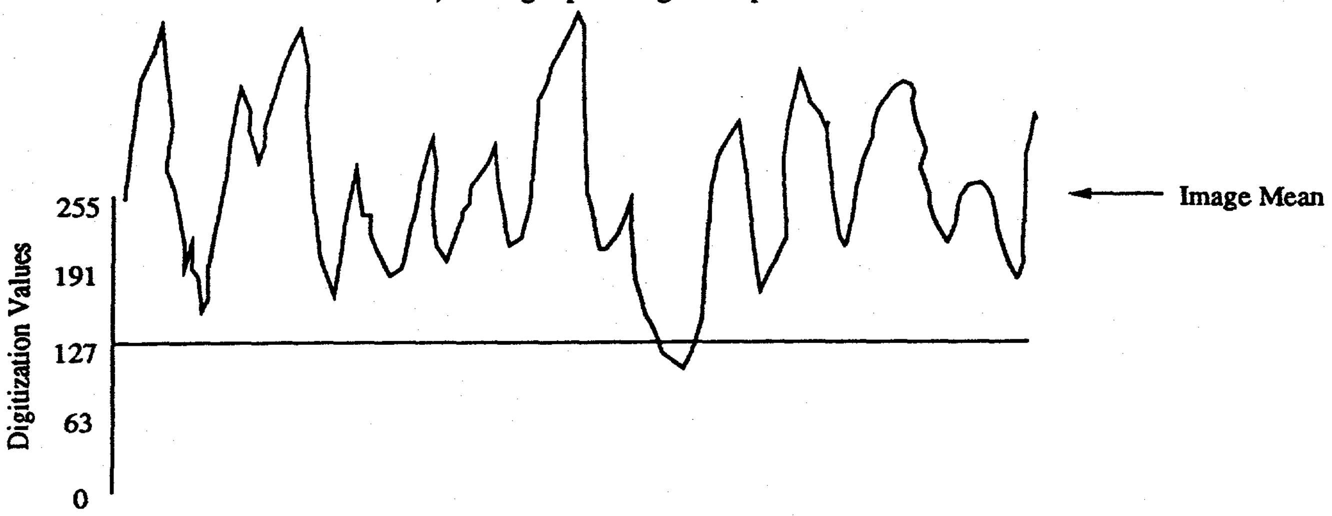


Circled numbers indicate hardware/software boundary test points for table 3.

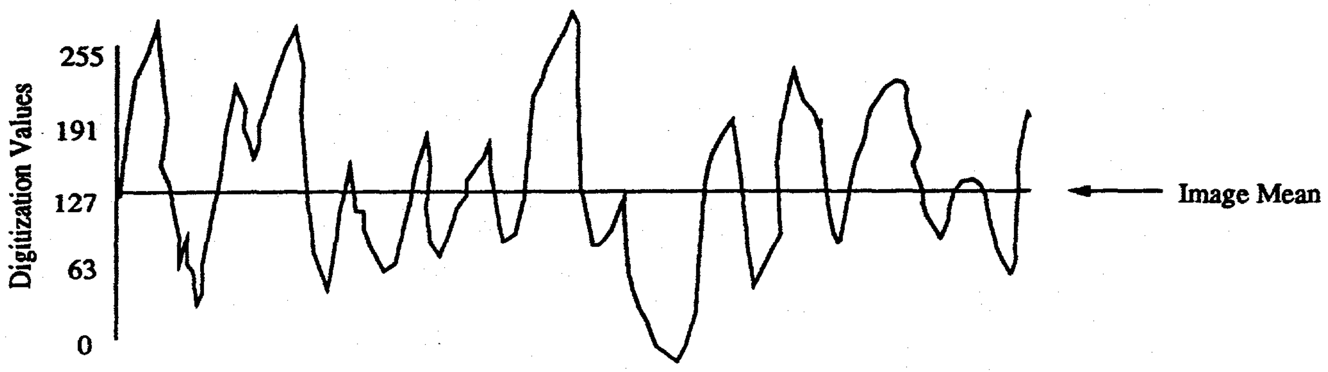
Figure 4: MIPS



a) Analog Input Image – Unprocessed



b) Input Image w/Gain of 2



c) Input Image w/Gain of 2, Offset of −127

Figure 5: Advantages of gain and offset processing. In a) the unprocessed input image covers a small range of A/D values giving low resolution. Gain is applied in b) resulting in a better spread of the image over A/D values, but moving the mean of the image away from its original value. In c) an offset is applied to the gain-processed image to return it to the original mean. The entire process doubles the resolution without distorting the input image.

%time	*cumsecs*	*#call*	*ms/call*	*name*
50.0	0.01	266	0.04	.umul
50.0	0.02	266	0.04	_cfree
0.0	0.02	266	0.00	.udiv
0.0	0.02	1	0.00	_exit
0.0	0.02	266	0.00	_free
0.0	0.02	1	0.00	_main
0.0	0.02	644	0.00	_malloc
0.0	0.02	1	0.00	_on_exit
0.0	0.02	1	0.00	_profil
0.0	0.02	266	0.00	_sbrk

%time	*cumsecs*	*#call*	*ms/call*	*name*
98.4	8.51	1	8510.00	_main
1.0	8.60	794	0.11	_sbrk
0.2	8.62	1561	0.01	_malloc
0.2	8.64			mcount
0.1	8.65	794	0.01	.umul
0.0	8.65	794	0.00	.udiv
0.0	8.65	794	0.00	_cfree
0.0	8.65	1	0.00	_exit
0.0	8.65	794	0.00	_free
0.0	8.65	1	0.00	_on_exit
0.0	8.65	1	0.00	_profil

Figure 6: Example 'profile' runs on Subsample and Convolution Primitives. Operations are arranged, by the utility, in decreasing order of execution time. Maximum resolution is seen to be 10 ms in the cumulative time analysis. It is the cumulative time that was used to determine the tables and figures presented in this report.

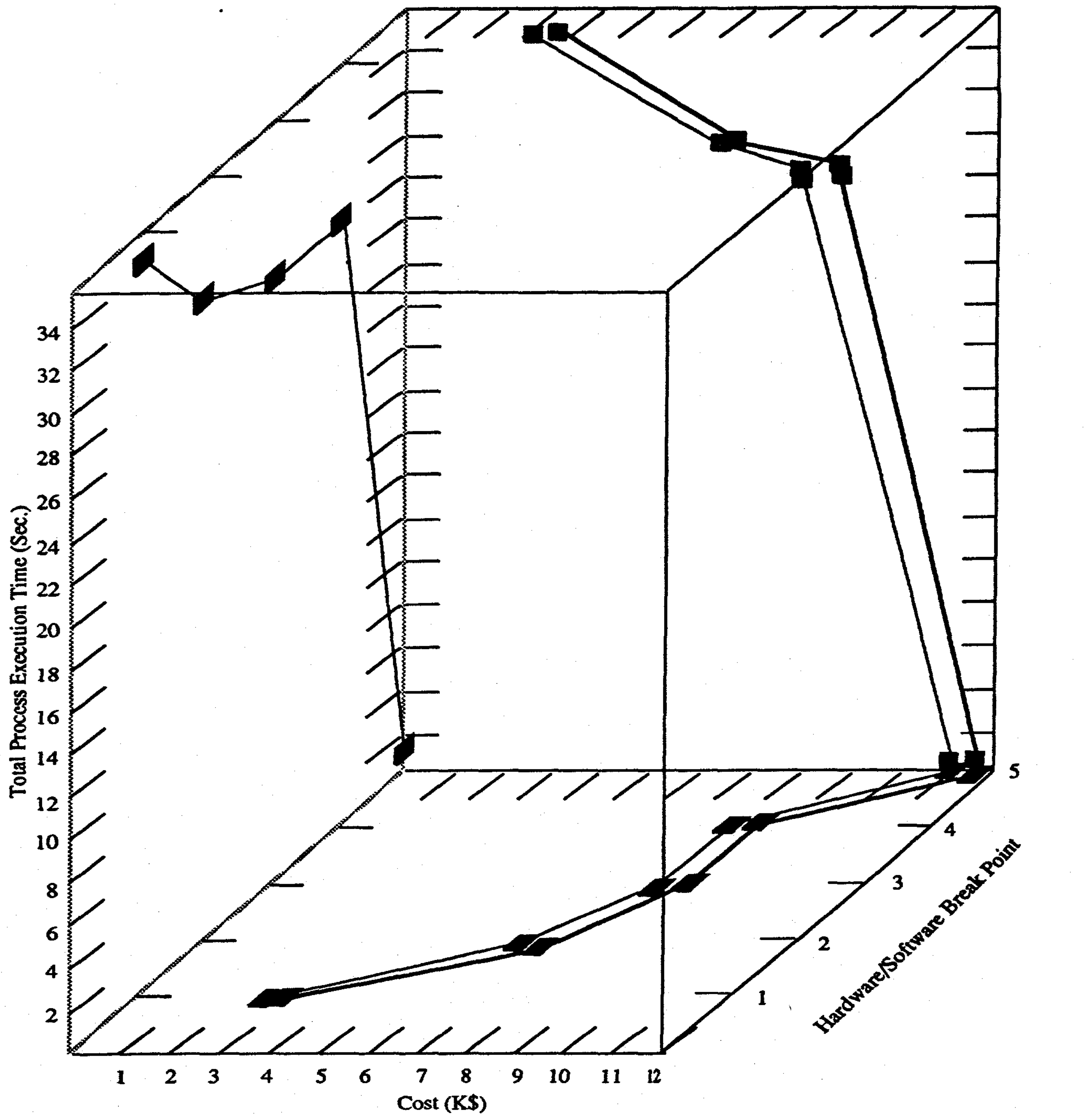


Figure 7: Architecture I - Execution Time/Cost Comparison. 4 MByte I.T. Frame Buffer.

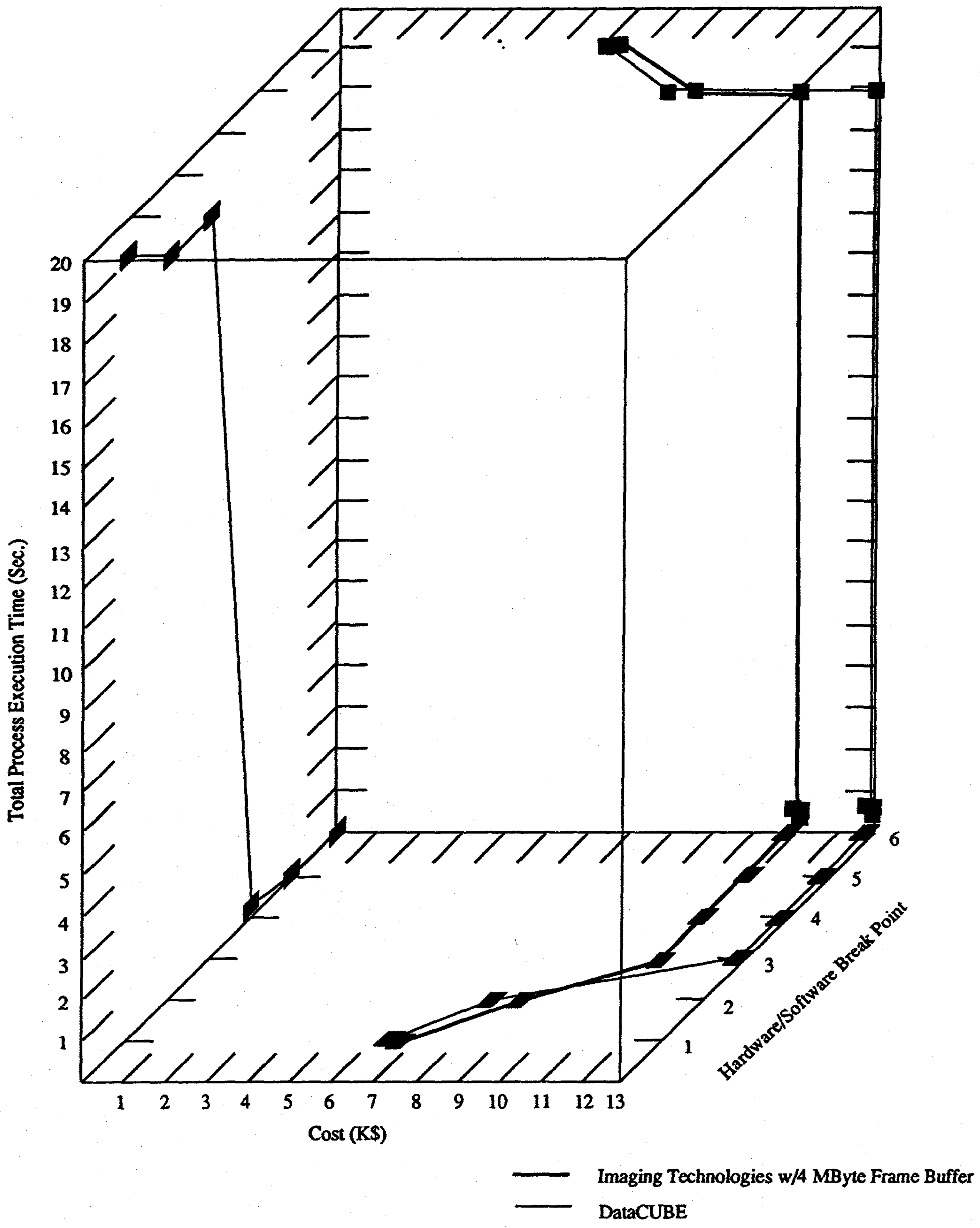


Figure 8: Architecture II - Execution Time/Cost Comparison. 4 MByte I.T. Frame Buffer. No HPF.

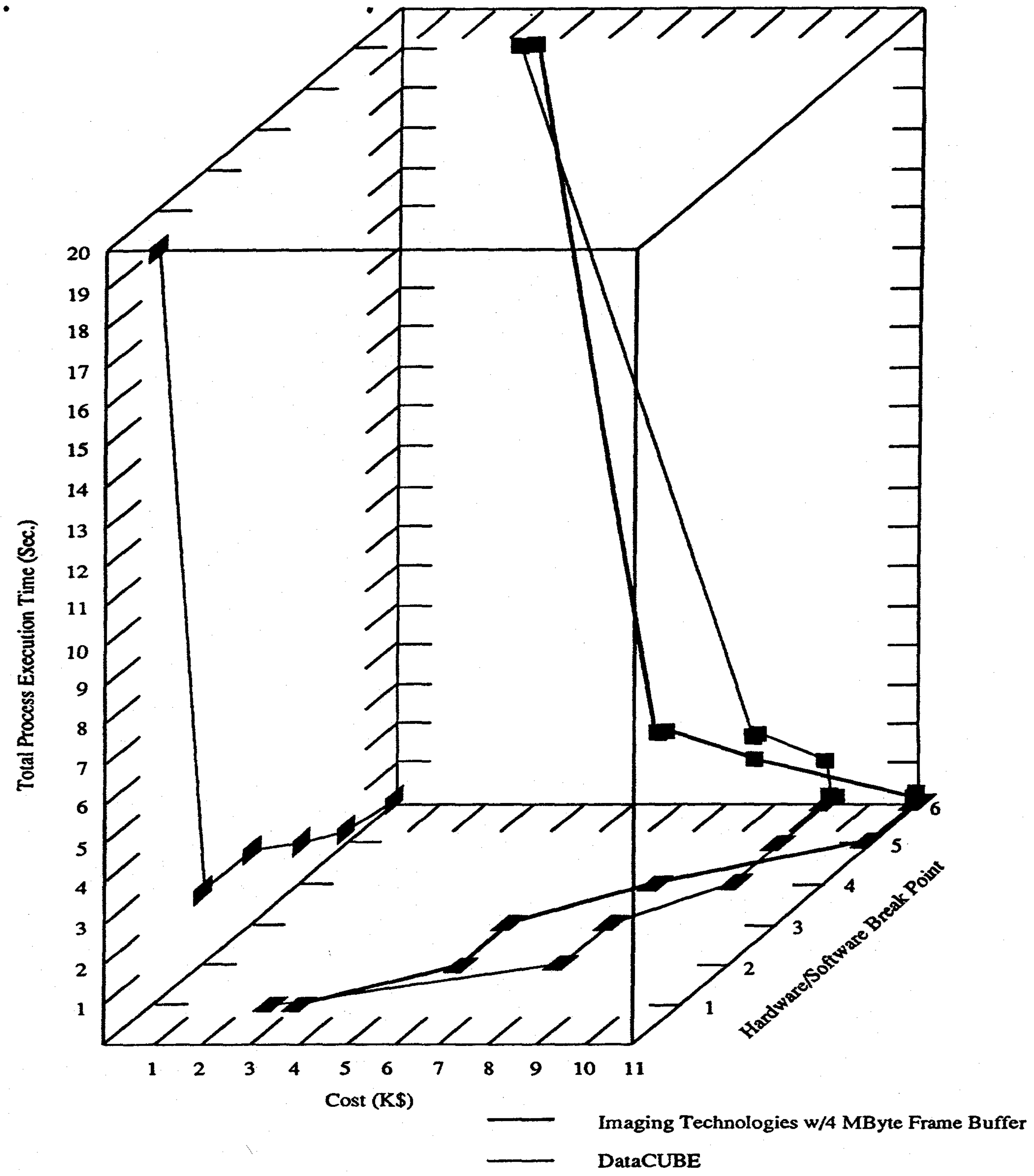


Figure 9: Architecture III - Execution Time/Cost Comparison. 4 MByte I.T. Frame Buffer. No HPF.

```
/*==========================================================================

                Function:               N/A

                File:                   time_test.h

                Programmer:             Scott P. Chapman

                Date:                   5 January 1991

                Project:                LANL High Security Image Processing
                                                - time testing code

                Parameters:             N/A

                Functions Called:       N/A

                Called By:              (Used In:)

                Description:

                        This is the header file contained in the test code for
                        the LANL High Security Image Processing project.  This
                        test code is intended to time the various major function
                        blocks to be executed in the project.  These blocks are:


                                - Spatial filtering by reducing an image from
                                  512x512 to 64x64.  This is achieved by re-
                                  placing every 64th pixel by the average of
                                  itself and its 63 nearest neighbors.  The
                                  pixel to be replaced will be the 3,3 pixel in
                                  an 8x8 (0-7,0-7) block of the original image.
                                  subsequent operations will be time tested on
                                  both the original 512x512 image and the re-
                                  duced 64x64 image.
                                - High and low pass filtering through 8x8
                                  2-dimensional convolution.
                                  [(512x512) or (64x64)] * {[8x8 (additions +
                                  multiplications)] + 1 addition}
                                - Image subtraction - subtracting one image
                                  from another to determine where changes have
                                  occured - 512x512 or 64x64 signed subtrac-
                                  tions.
                                - Image thresholding - comparing the change de-
                                  tection to a threshold value to determine if
                                  detected change is random and arbitrary or
                                  real - 512x512 or 64x64 comparisons.
                                - Temporal image averaging - the filtering
                                  performed by the present system.  Ten images
                                  are stored and then averaged pixel-by-pixel
                                  [(512x512) or (64x64)] * (10 additions + 1
                                  division)

                Modifications:

==========================================================================*/
#include <stdio.h>
#include <strings.h>
#include <ctype.h>
#include <math.h>

/*      BOOLEAN data type                       */

#define FALSE           0
#define TRUE            1
```

```
typedef                 int             BOOLEAN;

/*      Macros                          */
#define MIN(a,b)        ( ( (a) < (b) ) ? (a) : (b) )
#define MAX(a,b)        ( ( (a) < (b) ) ? (b) : (a) )

#define STR_LEN         81              /*  lines 80 characters+1 for NULL  */
#define NO_ANS          -1              /*  gparm answer file   */

typedef                 char            STRING [STR_LEN];

/*   constants defined below represent the row and column lengths and total image
     area for the normal and reduced images (512x512 and 64x64) and for those
     images that have been extended to allow zero padding around their peri-
     pheries to accomadate non-circular convolution with an 8x8 Kernel.
*/

#define                 NORM_X          512
#define                 NORM_Y          512
#define                 NORM_IMAGE      NORM_X * NORM_Y
#define                 RED_X           64
#define                 RED_Y           64
#define                 RED_IMAGE       RED_X * RED_Y
#define                 EXT_NORM_X      526
#define                 EXT_NORM_Y      526
#define                 EXT_NORM_IMAGE  EXT_NORM_X * EXT_NORM_Y
#define                 EXT_RED_X       78
#define                 EXT_RED_Y       78
#define                 EXT_RED_IMAGE   EXT_RED_X * EXT_RED_Y
#define                 KERNAL_X        8
#define                 KERNAL_Y        8
#define                 KERNAL          KERNAL_X * KERNAL_Y
#define                 ABOVE_THOLD     1
#define                 BELOW_THOLD     0

void    main ();
```

```
/*---------------------------------------------------------------------------

                Function:               void main ()

                File:                   subsample.c

                Programmer:             Scott P. Chapman

                Date:                   13 January 1992

                Project:                LANL High Security Image Processing
                                                - time testing code

                Parameters:             None

                Functions Called:       malloc ()

                Called By:              User

                Description:

                        The function reduces a 512x512 image to one that is
                        64x64 by copying pixels from every 8th row and column
                        from the original image to the target image.  To reduce
                        the overhead of the file operations, no image is
                        actually read in, but the values returned by malloc ()
                        are considered acceptable.  As in all primatives
                        measured, image values are declared to be of type
                        integer.

                Modifications:

===========================================================================*/

#include "time_test.h"

void main ()

{
        int             **orig_image,
                        **target_image;

        int             **temp_image;

        int             i, j, k, l;

        /*  build original-sized image and reduced image data arrays  */
        orig_image =
                (int**) malloc (NORM_Y * sizeof (int*) );
        target_image =
                (int**) malloc (RED_Y * sizeof (int*) );
        temp_image =
                (int**) malloc (RED_Y * sizeof (int*) );

        for (i = 0; i < NORM_Y; i++)

        {
                orig_image [i] =
                    (int*) malloc (NORM_X * sizeof (int) );
        }  /*  for  */

        for (i = 0; i < RED_Y; i++)

        {
                target_image [i] =
                     (int*) malloc (RED_X * sizeof (int) );
                    temp_image [i] = (int*) malloc (RED_X * sizeof (int) );
            }  /*  for  */

            for (i = 0; i < NORM_Y; i += KERNAL_Y)

            {

               for (j = 0; j < NORM_X; j += KERNAL_X)

               {
                          target_image [i/KERNAL_Y] [j/KERNAL_X] = orig_image [i] [j];
               }  /*  for  */

            }  /*  for  */

}  /*  main  */
```

```
/*===========================================================================

                Function:           void main ()

                File:               convolve.c

                Programmer:         Scott P. Chapman

                Date:               18 January 1992

                Project:            LANL High Security Image Processing
                                          - time testing code

                Parameters:         None

                Functions Called:   malloc ()

                Called By:          user

                Description:

                          Function performs an 8x8 convolution on an image of size
                          512x512.  The 8x8 convolution kernel consists of weight
                          values all equal to 1.  To reduce overhead, no image is
                          read in.  Rather, the values present when memory is
                          malloced are considered appropriate.  The image is
                          placed in a frame, initialized to zero, that extends 7
                          pixels in all directions around the original image.  As
                          the convolution proceeds, the resulting image is 7
                          pixels longer to the right and taller to the bottom.
                          The actual value of a pixel in the target image is the
                          summation of the pixels of the previous 8 rows and
                          columns.  This value will certainly overflow the 8-bit
                          values of the original image.  For this reason, all
                          image arrays are declared to be of type integer.

                Modifications:

===========================================================================*/

#include "convolve.h"

void main ()

{
        int             **orig_image,
                        **image_frame,
                        **target_image;

        int             **temp_image;

        int             i, j, k, l;

        /*  build original-sized image and reduced image data arrays  */
        orig_image =
                (int**) malloc (NORM_Y * sizeof (int*) );
        target_image =
                (int**) malloc (TRG_NORM_Y* sizeof (int*) );
        image_frame =
                (int**) malloc (EXT_NORM_Y * sizeof (int*) );

        for (i = 0; i < NORM_Y; i++)

        {
                orig_image [i] =
                    (int*) malloc (NORM_X * sizeof (int) );
```

```
    } /*  for  */

    for (i = 0; i < TRG_NORM_Y; i++)

    {
            target_image [i] =
                 (int*) malloc (TRG_NORM_X * sizeof (int) );
    } /*  for  */

    for (i = 0; i < EXT_NORM_Y; i++)

    {
            image_frame[i] =
                 (int*) malloc (EXT_NORM_X * sizeof (int) );
    } /*  for  */

    /*  initialize image frame border to 0  */
    for (i = 0; i < 7; i++)

    {
            for (j = 0; j < EXT_NORM_X; j++)

            {
                    image_frame [i] [j] = 0;
            } /*  for  */

    } /*  for  */

    for (i = 7; i < EXT_NORM_Y; i++)

    {
            for (j = 0; j < 7; j++)

            {
                    image_frame [i] [j] = 0;
            } /*  for  */

    } /*  for  */

    for (i = EXT_NORM_Y - 7; i < EXT_NORM_Y; i++)

    {
            for (j = 0; j < EXT_NORM_X; j++)

            {
                    image_frame [i] [j] = 0;
            } /*  for  */

    } /*  for  */

    for (i = 7; i < EXT_NORM_Y; i++)

    {
            for (j = EXT_NORM_X - 7; j < EXT_NORM_X; j++)

            {
                    image_frame [i] [j] = 0;
            } /*  for  */

    } /*  for  */

    /*  read image into frame  */
    for (i = 7; i < EXT_NORM_Y - 7; i++)

    {
```

Image Change Detection in a Static High Security Environment
Image Compression, Storage and Retrieval

I. Introduction

This document is presented to accompany initial delivery of the image data storage and retrieval subsystem of a high security image change detection alarm system. This work was sponsored by Los Alamos National Laboratories (LANL) and performed by the Electrical and Computer Engineering Department of the University of New Mexico (UNM) (NEED CONTRACT NUMBER??!!). The subsystem performs image acquisition, data compression and storage. It also provides a means for data retrieval and animated data review.

This report begins with a general description of the overall project into which the work fits as a subsystem. The following section gives a more detailed look at the subsystem itself, defining its tasks, implementation and relation to the parent system. Section IV is a user's manual. It presents the configuration management of the software that makes up this project. It proceeds to detail the various procedures an options the user has available. Finally, a conclusion sums up the work do date and gives a brief analysis of system performance. It also suggests directions the project might move over the next year.

II. Image Change Detection Alarm System.

The work presented in this report is being sponsored by LANL in support of an image change detection alarm system. These alarms are to be installed in low level nuclear waste storage facilities. The environment is simulated in figure 1. Nuclear waste is stored in sealed, stacked metal drums. Drums are labled for contents and serialized with high contrast characters. Surveillance is performed by 4 to 6 cameras per storage area, which provide complete video coverage. The environment is completely static in both lighting and motion. This simplifies the matter of alarm generation, since any real change in the scene can be interpreted as an alarm condition. The requirements for this project are that even minor change be detected and recorded. For instance, should a barrel be shifted even slightly, the fact would be noted and the barrel in question identified. Images are acquired from each camera with about a 15 minute period.

The general process of image change detection is shown, along with a simple example, in figure 2. Prior to putting the system into operation, a non-alarm reference image is recorded for each camera. While in operation each camera acquires a (presumably noisy) image that is then processed to produce a noise-reduced image. Current processing includes a median filter, wherein each image pixel is replace by the median value of itself and all its neighbors. This technique is effective at removing random speckle noise. Another filtering technique that is used is to acquire 10 images in rapid sequence and to produce an acquired image that is the average of these images.

Once the image is acquired and processed, it is compared to the reference image by subtracting the reference from the current image pixel-by-pixel. Where there is no change, the absolute difference will be near to zero, while pixels showing real change will have large absolute difference. In general, due to noise, pixels showing no change are not exactly zero, but are within some threshold value of it. For this reason, the change image is clipped by setting pixels of absolute value below the threshold value to zero. The change image is then scanned to determine if real change occured.

If change is determined to have occured, an alarm condition is set and the image is examined to determine if the change involves human activity. Due to the static nature of the environment, human intrusion is simply defined as change that spans a large number of pixels. That is, change in the image of greater than a particular size. If the activity is found to be human, the subsystem described in this report is enabled. That subsystem acquires images at an accelerated rate, compressing and storing them. Whether or not the change is deemed to be human activity, further processing to analyze the scene is performed by the main security system before returning to a no-alarm state.

III. High Speed Image Acquisition, Compression, Storage and Retrieval

The image acquisition and storage system that constitutes the subsystem being presented is intended to photograph the secure area at accelerated rates during human intrusion. Such data can be recovered at a later date for analysis or prosecution. The target frame rate throughout the project has been a maximum of 3 FPS (frames per second). A normal video frame is 512 x 512 pixels of 8-bit grey scale data. Storage of decompressed images at such a frame rate would reduce disc or memory space at a rate of 5.2 MBytes per minute. Extended intrusions, or intrusions in the field of view of multiple cameras can quickly deplete storage assets. For this reason, fast and effective data compression techniques were desired. However, in image data compression, speed is gained at the expense of accuracy and vice versa. To help balance the system, a certain amount of loss of resolution was deemed acceptable.

The primary data reduction technique is simply a 16 times reduction in size of the image. This is accomplished by placing the average of a 4-pixel-by-4-pixel area of the original image into a single pixel of the new image, then moving the averaging window a full window width before repeating. An immediate 16X compression is realized. This technique has the disadvantage of being irreversible. However, given the large scale of the change being detected, the loss in resolution is deemed acceptable.

Even in a dynamic environment, there is a great deal of redundancy in a series of images. For this reason, although this subsystem is not concerned with detecting change, the system was designed to operate on a change image that is the difference between the current image and a reference taken with this same camera. At first glance, this introduces a undesireable factor. The signed difference image has a dynamic range of 9-bits, compared to the current grey scale image whose dynamic range is 8-bits - A decrease in compression efficiency by a factor of 1.13. This becomes a factor of 2.0 when one considers that C does not support a 9-bit data type, causing the system to require a 16-bit short int data type to store the signed difference value. Experimentation showed that the problem could be solved by filtering the difference image by shifting the signed difference value right a single bit and shifting out the low order bit. This will shift the value of a pixel in the reconstructed image by 1, over 50% of the pixels. That is, the shift leaves all pixels even valued. On an average, half the pixels are odd valued and are altered in this process.

The degradation is not visible.

Once the 8-bit difference image is available, it is filtered by clipping all values below a threshold. Once clipped, the thresholded change image lends itself to a second level of data compression. In those areas of the image where no real change is occuring, the change image will contain long strings of zeros. Storing each pixel value over these regions becomes redundant. Instead one need only store a single image value - 0 - and the length of the string of consecutive zeros. An abbreviated example is given in figure 3. Note that the efficiency is slightly reduced in that the length of runs of non-zeros must also be stored. Also, run lengths of greater than 256 are to be hoped for. As a consequence, the run length values must be stored as short int (16-bit). An advantage to this system is that the original difference image that is so compressed is completely recoverable.

A flow diagram of the entire subsystem is presented in figure 4. Once invoked by the parent process, the subsystem runs as a process completely independent of and in parallel with the parent. Communication is through a file read by the subsystem and written to by the parent. This file ("stopfile" in the diagram) is read after processing each image. It contains a single "stop" flag. If true, the process quits. Otherwise, it continues with the next image.

The process begins by taking a single image. The image is subsampled from 512 x 512 to 128 x 128 and stored to its own file without further processing. It is also held in memory to be used as the reference file for future images.

Once the reference image is processed the system begins to acquire images that are fully encoded. An image is received and immediately subsampled. The resultant 128 x 128 image has the reference image subtracted from it creating a 9-bit signed difference image. This image is shifted and clipped. In clipping, values below the threshold are set to zero, while those above it retain their value. The new 128 x 128 8-bit clipped image is then run-length encoded. This produces two files: the 16 bit run-length file and the 8-bit data file. Finally, the stopfile is read to determine if the process should continue.

Data retrieval performs the opposite operation from the encoding process. The reference file is first read from disc. Then the data file and run length file are read and used to reconstruct the difference image. The difference image is shifted left one bit to create a 9-bit (16-bit data type) signed difference image. This image is lost some information in reconstrution in that the low order bit will always be zero due to shifting. This will alter an average of 50% of the pixels by a single numeric value. The retrieved change image is added to the reference image to regain the 128 x 128 image that is represented by the change image. This becomes the retrieved image. No further reconstruction of the original image is possible.

IV. User's Manual

Configuration Management:

This project was constructed under the Khoros software development management tool. It is distributed into the standard Khoros directory tree as described in reference 1. It is highly recommended that anyone who attempts to modify the routines describe herein familiarize themselves with references 1 and 2. The Khoros toolbox is called "learn" and has as its root directory:

```
                /home/tardis/scott/learn
```

Executable files are located in:

```
                /home/tardis/scott/learn/bin
```

Source files each have their own directory in:

```
                /home/tardis/scott/learn/src
```

In addition, the decompress directory holds sample multi-band VIFF images in:

```
        /home/tardis/scott/learn/src/decompress/samples/...
```

Execution is generally tested in:

```
        /home/tardis/scott/learn/src/rundir
```

For Khoros programmers only, the vf routine .form file is in:

```
        /home/tardis/scott/learn/repos/cantata
```

and the panes in:

```
        /home/tardis/scott/learn/repos/cantata/subforms
```

The xv routine decompress has its .form file in

```
        /home/tardis/scott/learn/repos/decompress
```

The _mf file is found in:

```
        /home/tardis/scott/learn/repos/config/src_conf
```

and several .def files are in:

```
        /home/tardis/scott/learn/repos/config/imake_conf
```

For the present, only learn.def should be used.

All the directory configuration given above is in accordance with standard Khors distribution. The .Toolboxes file is located in /home/tardis/scott. Files are set to read and execute for everyone. In addition, write permission will be given to the group on all project files. This slightly dangerous tactic will allow all concerned to modify the code. It is suggested that once the project gains a new developer, that that person remove the global write permission. Write permission within the rundir and src directories is already global to allow interested parties to execute the existing routines.

User's Guide

The functions described below are intended to be executed under the Khoros Cantata visual programming environment. They may also be executed from the command line, though guidance for doing so is not presented. Executing Cantata as follows will allow the "learn" toolbox to appear in the menu system - thereby allowing execution of the non-Khoros routines described below:

```
cantata -form /home/tardis/scott/learn/repos/cantata/learn.form &
```

This discussion generally assumes that the user is familiar with Cantata.

There are three main groups of functions described below. The first of these divides the project into individual glyphs for each step of the compression and image recovery procedure. This is useful for demonstration purposes, as each step can be linked to an image display glyph such as putimage. To arrange such a demonstration, place the glyphs as in figure 5. The demonstration will take a single image through data compression and recovery. Functions in this group include get_image, reduce, subtr, clip, rl and recover.

The second group includes two functions that served as intermediate stages in the development of the project. The functions they serve are useful and should be retained for future availability. record_vf is the full compression routine. However, it takes its input from a multi-band VIFF image rather than from a camera. This may have applications in reducing the size of (then deleting) VIFF images. dsply reverses the process of record_vf by taking compressed image data files and producing a multiband VIFF output.

The final group of routines is the deliverable project. It includes record, for full camera-input image compression. decompress serves as the image recovery, providing and animation of images compressed into a file as discussed below. rec_strt and rec_stp are temporary service routines.

A function-by-function description of the project is given below. Each functional description is accompanied by a figure representing its pane, including the default input values. Figure 6 (not yet available) shows the function selection pane on Cantata - MORE DISCUSSION NEEDED. Note that the images and defaults associated with each function are constructed from an older version of this program and may not reflect the current state of the Cantata panes.

get_image

Figure 7 - Used in the demonstration. Function takes an image from the camera and creates a single VIFF output to be passed to the next function. This project was created on DataCUBE hardware, though LANL uses Videopix equipment. get_image works only with the DataCUBE boards. For this reason, a VIFF image should simply be written to the directory from which the demonstration is running and passed to the reduce function.

Defaults:

Input Board (DataCUBE) - 0
Input Framestore (DataCUBE) - 0
Output File - new_image

reduce

Figure 8 - A demonstration function. reduce takes a 512 x 512 input image and returns a 128 x 128 output image. Each pixel in the output image represents the average of a 4 x 4 block of pixels in the input image. This is the primary and unrecoverable method of data compression in this project.

Defaults:

Input Image - i_image
Output Image - o_image

subtr

Figure 9 - For demonstration purposes, subtr takes two 8-bit grey scale images - generally the current image and the reference - and produces a 9-bit signed (actually 16-bit short) image that is the difference of the inputs. This technique is used to detect changes between two images, or, as in this case, produce an image that is easier to compress and from which the original may be regained.

Defaults:

Reference Input File - r_input
Current Input File - c_input
Difference Output File - difference

x Dimension - 512
y Dimension - 512

clip

Figure 10 - Demonstration. This function takes an input image that is signed (short) and a threshold value. It returns an image that is identical to the input for all values above the threshold. All values in the input image that are at or below the threshold are set to zero in the output image. This technique filters noise in the unchanged pixels and creates an image that can be compressed using run length techniques. It is at this point that the 9-bit signed difference is shifted right to an 8-bit signed difference image.

Defaults:

Unclipped Input Image - i_image
Clipped Output Image - o_image

Clipping Threshold - not given

x Dimension - 512
y Dimension - 512

rl

Figure 11 - Demonstration - This function takes the 8-bit signed difference image and scans it for strings of zero (no change) values. These strings are encoded by storing the length of the string in one file and a single zero in another. Strings of non-zero values are stored as the length of the string and a single entry for each non-zero value. Compression on the image is on the order of the length-of-the-string-to-2 for each string of zeros. This function works on a single image.

Defaults:

Input Image - not given
Run Length File - rl
Output Data File - dat

x Dimension - 512
y Dimension - 512

recover

Figure 12 - Demonstration function. recover takes the single image encoded in the output of rl and rebuilds the change image. It then adds the change image to the reference image to recover the original image. Note that the 128 x 128 recovered image is indistinguishable from the original image of that size, except for the 1-bit loss in half the pixels due to shifting to the 8-bit representation. However, the original 512 x 512 image cannot be recovered from this image.

Defaults:

Input Data File - not given
Input Run Length File - not given
Recovered Image - change
Reference Image - not given

dsply

Figure 13 - A developmental routine, dsply takes as input compressed run length and data files as well as a reference image. It performs the full decompression algorithm and writes each input out as a serialized VIFF image. These VIFF images may then be read in by a program such as "animate" for animated display. This is the actual decompression and works on input files that are not limited to a single image.

Defaults:

Reference Image Input File - ref

Run Length Input File - rl
Image Data Input File - data
VIFF Array Output File - series.vff.x

record_vf

Figure 14 - Another developmental routine. record_vf performs a full compression on the input images just as in the final deliverable version of this program. The difference is that this routine served as a transitional state between the DataCUBE hardware that the project was originally developed on, and the Videopix hardware the client uses. For that reason, record_vf takes as its input a multi-band VIFF image. This is a format that is common to all Khoros users and is therefore more universal than the hardware-dependent function used in the final product.

Defaults:

Multi-Band VIFF Imgae Filename	not given
Output Refernce Image	ref
Output Run Length File	rl
Output Data File	data
Clipping Threshold	25
Output Path	./

record

Figure 15 - This is the final deliverable version of the data compression portion of the project. It takes as its input a VIFF image from a Videopix image acquisition board. For output it produces a run length and a data file. The first image it acquires is the reference and this is written directly out without differencing or run length compression. This function appends to its data files as it acquires new images. For this reason it is highly recommended that the operation be performed in a new directory or the output be written to a new set of filenames for each new set of data to be acquired. Results have been unpredictable when an old data file is opened for appending during a new iteration of the program. The number of images to be taken by the record routine is open ended. Each new image is followed by reading a flag from a temporary file that serves as a communication vehicle with the parent process. So long as the flag is set FALSE (don't stop) the process will continue with a new image.

Defaults:

Input Stop-Flag Filename	stopfile
Output Reference Image	ref
Output Fun Length File	rl
Output Data File	data
Clipping Threshold	25
Output Path	./

Note: These defaults and figure 15 are out of date. For more accurate information FIX THIS ENTRY.

rec_strt

Figure 16 - This is a temporary function used to test the final version of the project. This function will be replaced by similar action in the parent process as the project is integrated. This function opens the stopfile and writes a "don't stop" flag so that the process may be started and continue to read images.

Defaults:

Record control file - stopfile
Output Path - ./

rec_stp

Figure 17 - Identical to rec_strt except that it writes a stop flag to the output file to order the record process to exit.

Defaults:

Record control file - stopfile
Output Path - ./

decompress

Figures 18 - 20. This function is the standard Khoros animate utility with the front end data building portion of the original stripped and replaced by a run length decompressor that builds the requisite multiband VIFF structure from compressed input files created by record. The function further adds the capability to zoom an integer factor of the 128 x 128 image size. Zoom factors range from 1 to 8. Zooming is achieved by simply copying a single pixel into an x-by-x block of pixels in the new image. The results become blocky and do not improve resolution. Zoom factors greater than three or four slow the building of the multiband VIFF and do not provide a better image. Figure 18 represents the main xv pane. Several items that appear are related to the original use of animate and are no longer applicable. Defaults of interest include:

Reference Image Filename - ref
Run Length Filename - rl
Compressed Data Filename - data
Zoom Factor - 1

Once invoked, decompress builds the complete VIFF structure before the images are passed to the animate portion of the program for display. The more stored images and the larger the zoom factor, the longer the delay in displaying the images. The main display is shown in Figure 19. Images are shown in sequence from first to last. Sequences may run in either direction and may be single step (single arrow) or continuous (double arrow). The "Input" button is of no real use in this version of the animation. However, the "Options" button will open the window of figure 20. Here, the user can choose between 3 types of continuous animation. Loop will scroll through the images, returning to the first (and continuing) upon reaching the last. Single will scroll through and stop on the last image. Autoreverse changes direction of scroll when it reaches the first or last image. Scrolling then continues. The user may also set the frame speed of the scroll in seconds. The "Show Frame Number" button will make the index of the current frame appear in the image window, when set to True. Default values are as they appear in figure 20.

V. Analyses and Observations

The project being delivered in conjunction with this documentation is operational and ready for integration into the parent system. Precise analysis of its performance is unavailable as shall be seen. However, in this section we present both a general analysis and a discussion of suggested improvements that should accompany or immediately follow system integration.

Throughout this project frame rates have varied between about 5 and 8 seconds. These frame rates are slower than the target rate. The primary factor seems to be the architecture the project is running on and the loading of the particular machine. Suggestions given below may speed compression slightly through algorithmic improvements, but the real gains will be made as the project is moved to newer, faster, more capable dedicated platforms. Such improvement is planned.

The most significant compression ratio is that due to image subsampling. This is a fixed 16x. As discussed above, this is accompanied by a loss in resolution and an original image that cannot be recovered. Discussion of run length compression can only be made in general terms. The efficiency is highly data dependent. Two factors affect the yield directly: threshold value and the number of pixels in the unchanged portion of the image. The higher the threshold and the more unchanged pixels, the better the compression. Data presented in the ISE paper saw compression factor of between 1.4 and 2.0 due to run length encoding. Much better ratios might be achieved in an environment where the change takes up a smaller percentage of the image. Critical analysis of this issue depends on examining "real" data from an actual security installation.

Several "next steps" suggest themselves to the developers of this project. As the team is changing the guard, it is important that the shortcomings of the project be noted so that they may be acted on. Here, then, are the logical directions things might go.

- System integration - Since this subsystem is a separate process, this step has been designed out, after a fashion, and should be a relatively simple matter. The functionality of rec_strt and rec_stp need to be absorbed by the parent system. After that, a method needs to be devised to signal the subsystem to start. Perhaps, the subsystem can loop in the background waiting for the stopfile to be written with a start flag which could also serve as the don't stop flag. In this way, the system would loop between an idle loop and an acquisition loop.

- Image size generalization - This is probably the next most important step. Right now, the system is designed to work on input images of a hard-coded 512 x 512 pixels. The current Videopix system is running images of 640 x 480. The solution has been to clip the right side of the image, losing information and slowing the process as the input image is copied into the fixed size array. Generalizing input image (and consequently subsampled image) size should be a relatively simple matter. The image returned by the camera read is in VIFF format. This gives the programmer direct access to the image size. Either this size or the reduced size must be passed to all the subroutines in the compression process. Furthermore, these values must be passed to decompress through one of the output files. So, while the changes are simple, they are pervasive.

- Improve decompress file reading - The decompress routine now reads a single data value at a time, slowing the process with disc reads. A better solution would be to read both the run length and the data files into arrays and then index the arrays. Arrays can be malloced with sizes based on the size of the file (returned by stat ()). This change was attempted and worked well on most data. However, some data produced intractible segmentation faults.

REMOVE THIS??? Just want to double check...

- Updating the reference - Run length efficiency could be improved by making the current image the reference for the next image rather than relying on a single early reference. Images that are closer together in time are likely to have fewer changed pixels than those that are separated. A brief attempt was made to perform this update. A closer look is needed at this technique.

- Interprocess communications - Currently the parent and subprocess communicate stop commands through a common file. As the system grows, such a file might be formatted for use to communicate a good deal more information. One such piece of information that is needed right now is the clipping threshold. Currently that value must be entered into the pane of the record process.

- Automated thresholding - The current plan for setting the clipping threshold is trial-and-error and to get the threshold as high as possible (without losing too much resolution) in order to improve run length compression ratios. A variety of techniques exist to automate the threshold determination process. These techniques could also be extended to the parent process which performs its own thresholding. EXAMPLES AND REFERENCES NEEDED HERE.

This concludes this programmer's direct involvement with this project. The work has been fruitful in many ways. Techniques in image processing we learned and new modifications of those techniques developed. The Khoros tool was incorporated into the personal toolbox. Experience was gained in intricate coding. Personal limits were tested. Finally, horizons were broadened as contacts were pushed beyond the limits of this university. If I might be forgiven the personal comment, these reasons and others made this project a satisfying and personally productive endeavor.

References

1. "Khoros Manual Volume 2 - Programmer's Manual", University of New Mexico, 1991

2. "Khoros Manual Volume 2 - User's Manual", University of New Mexico, 1991

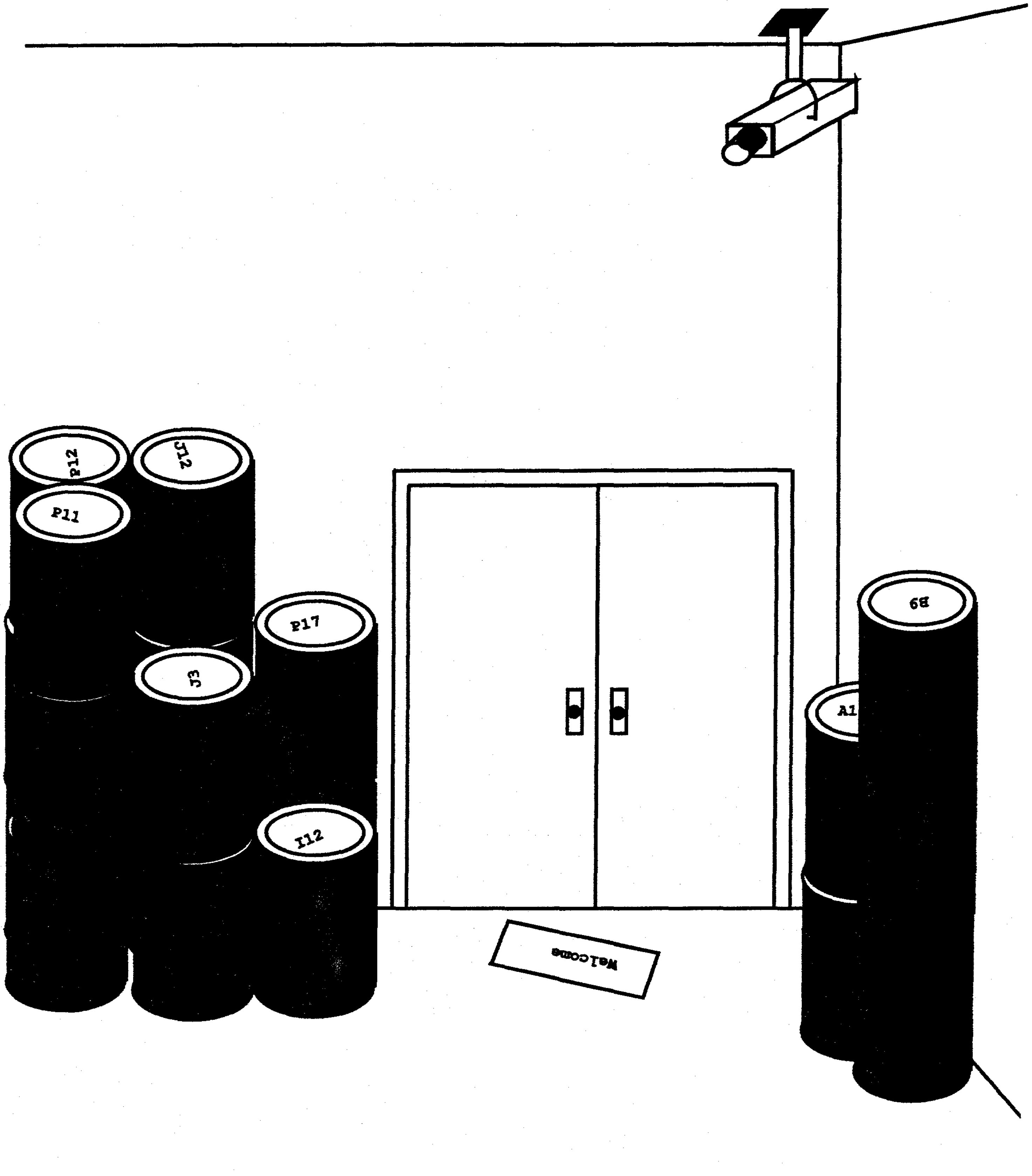


Figure 1 – Static Secure Environment

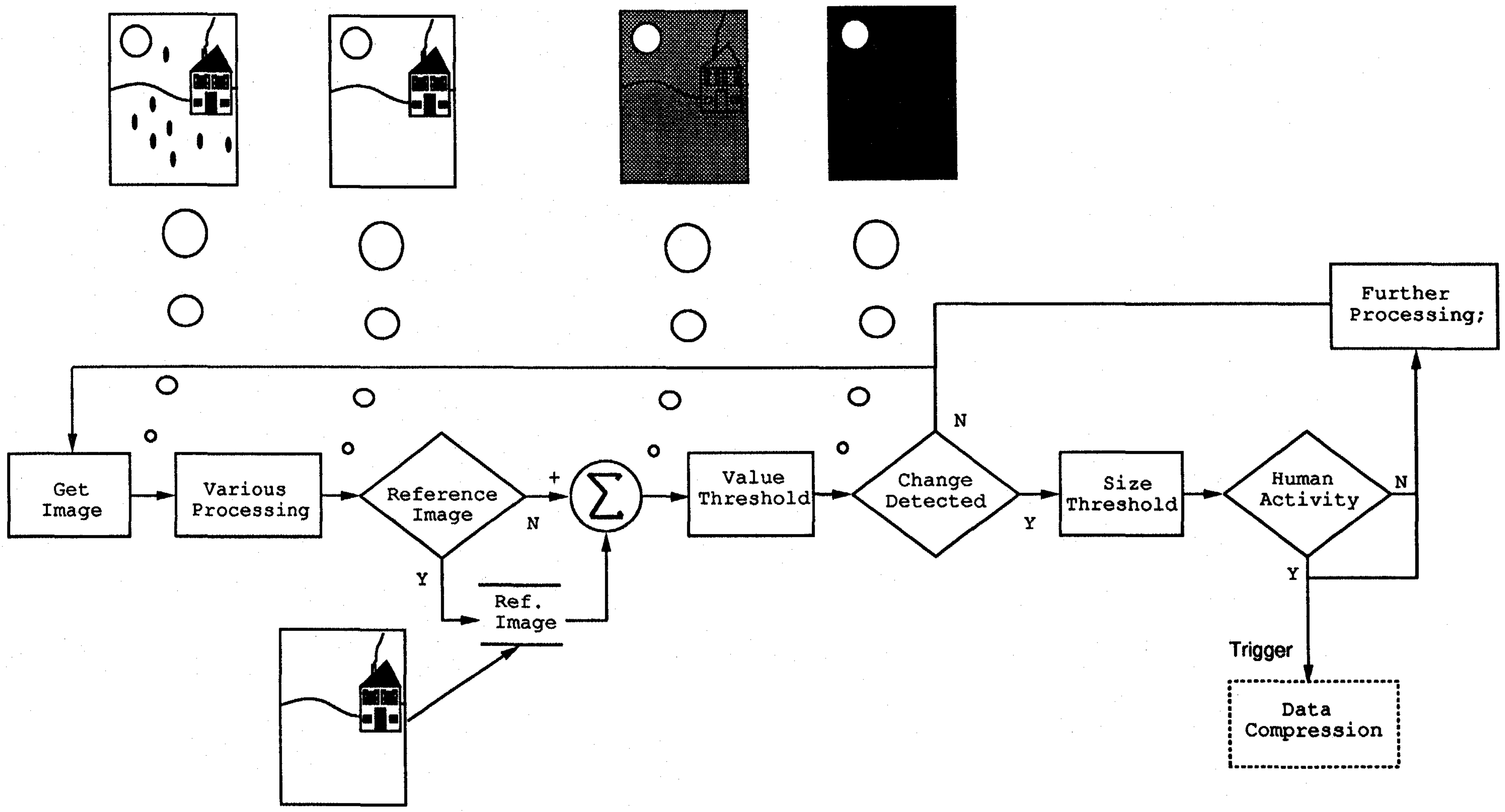


Figure 2 – Parent Project – Current LANL Architecture

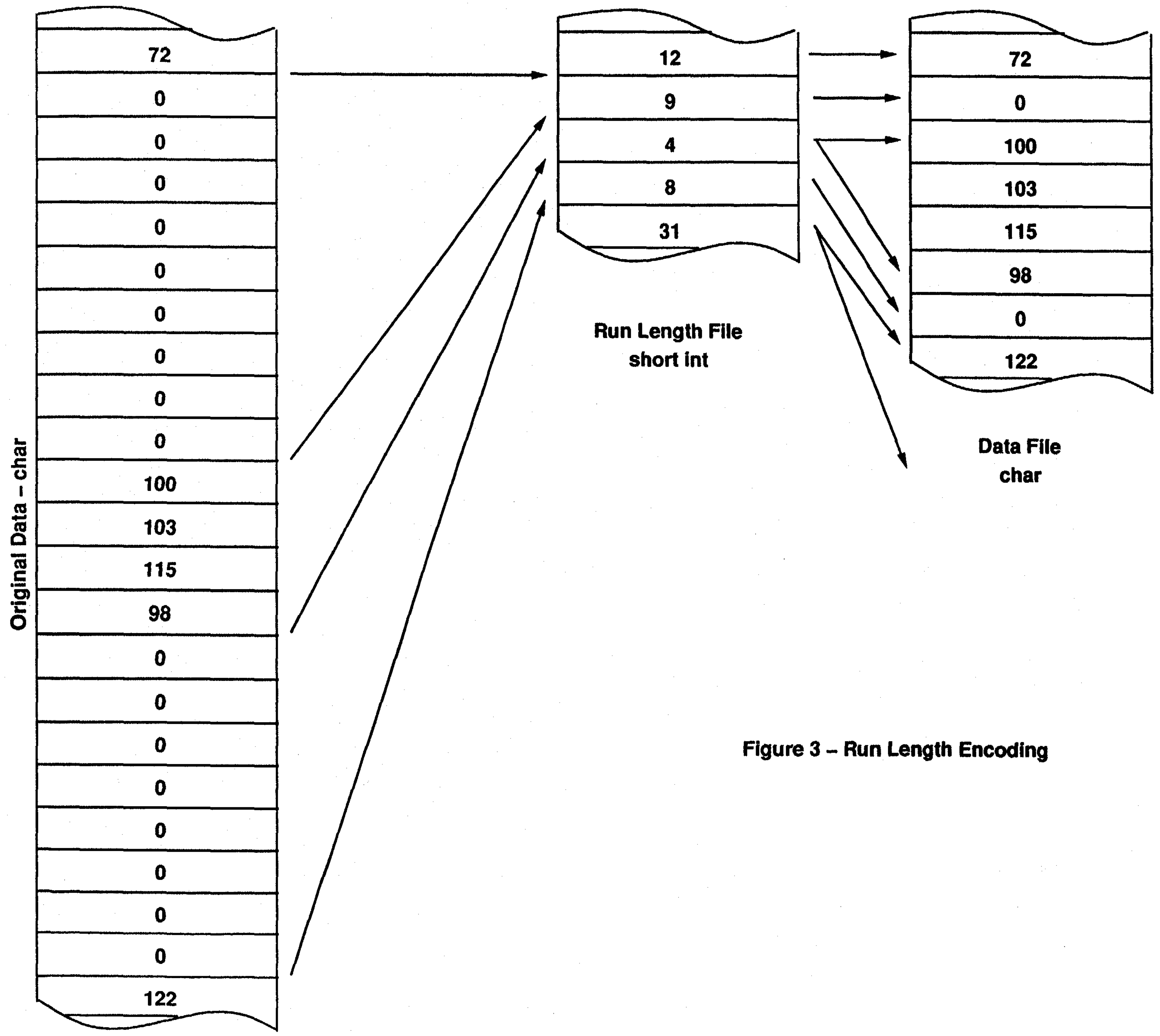


Figure 3 – Run Length Encoding

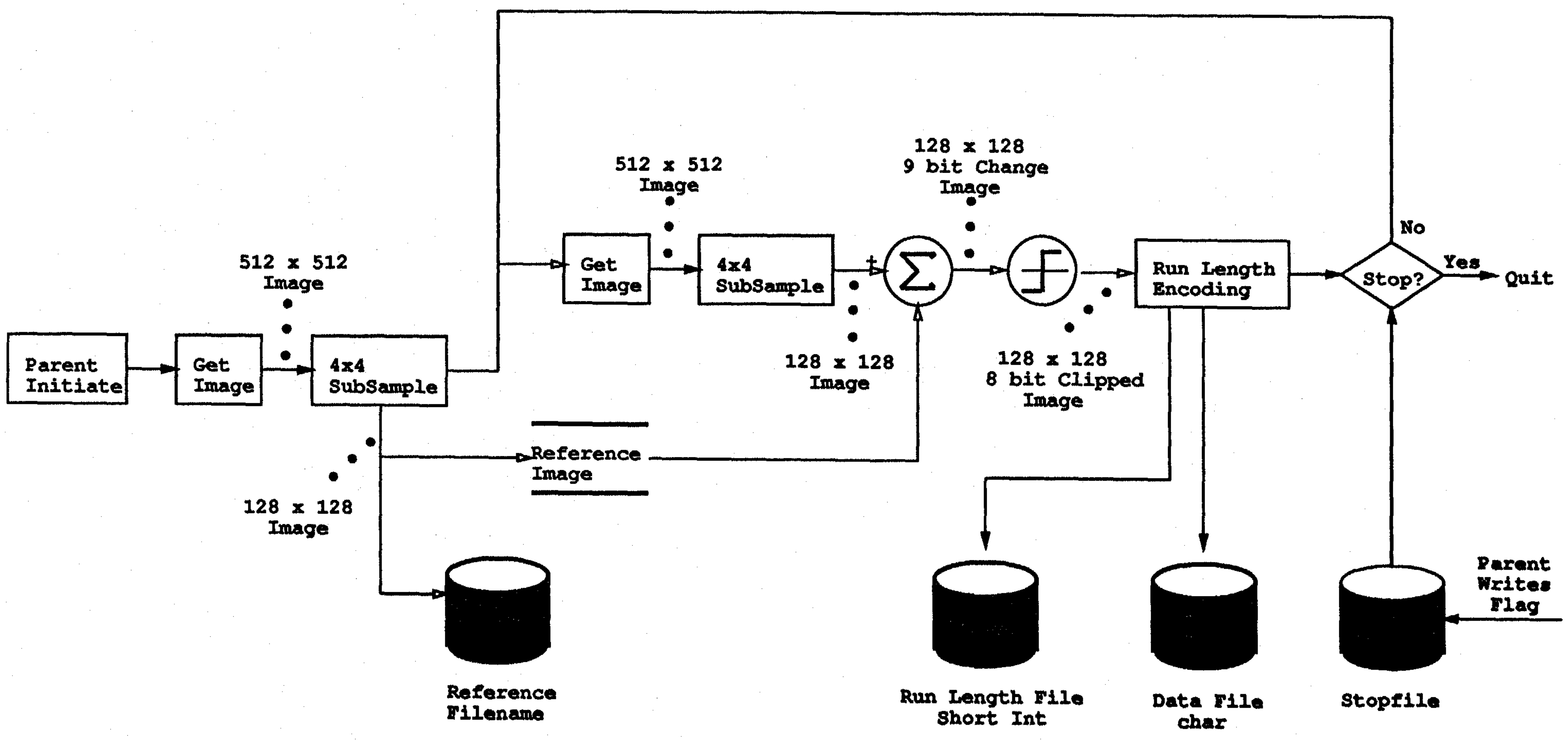


Figure 4 – Acquisition/Storage/Retrieval Subsystem

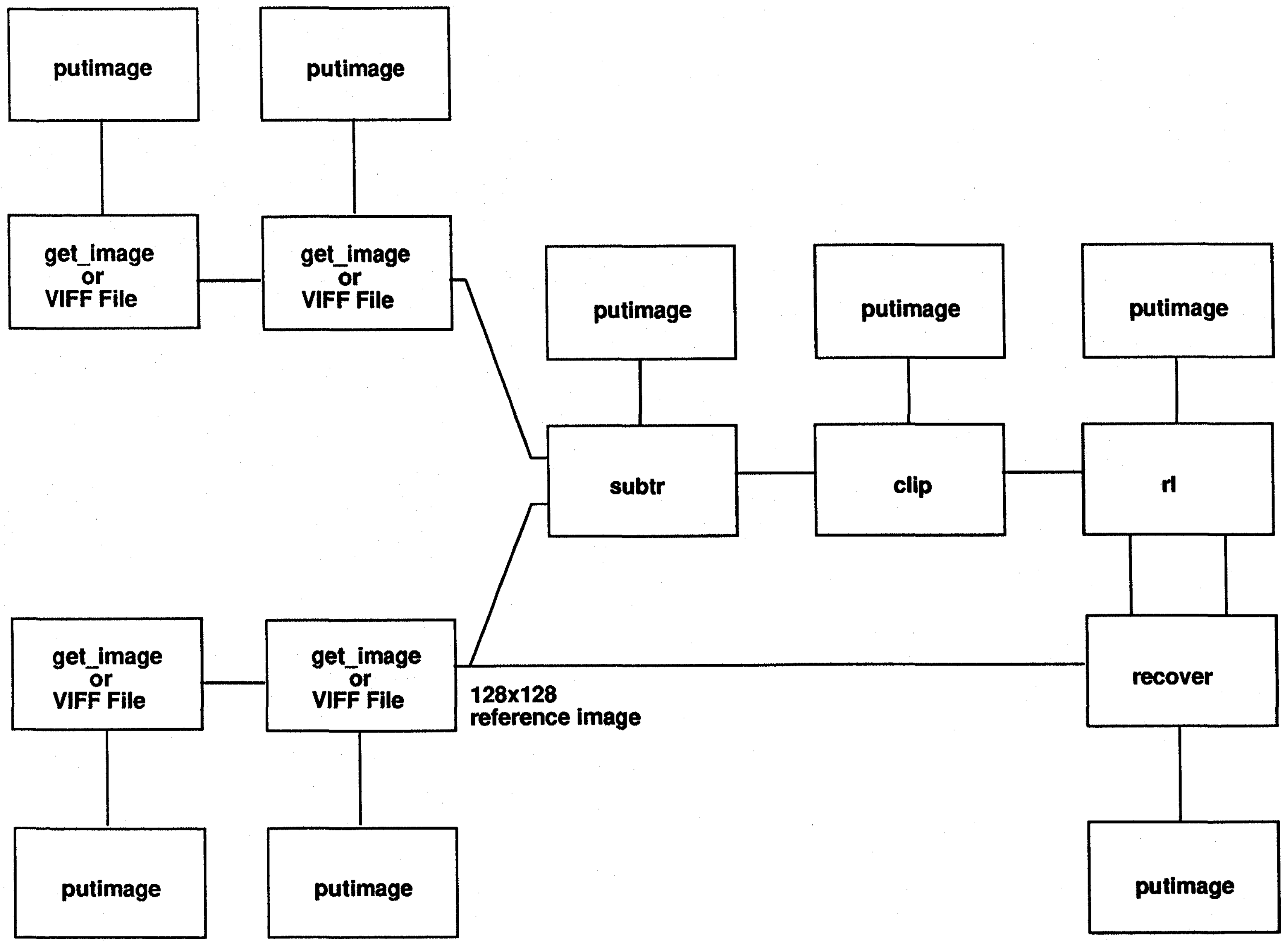


Figure 5 – Cantata Demonstration Setup

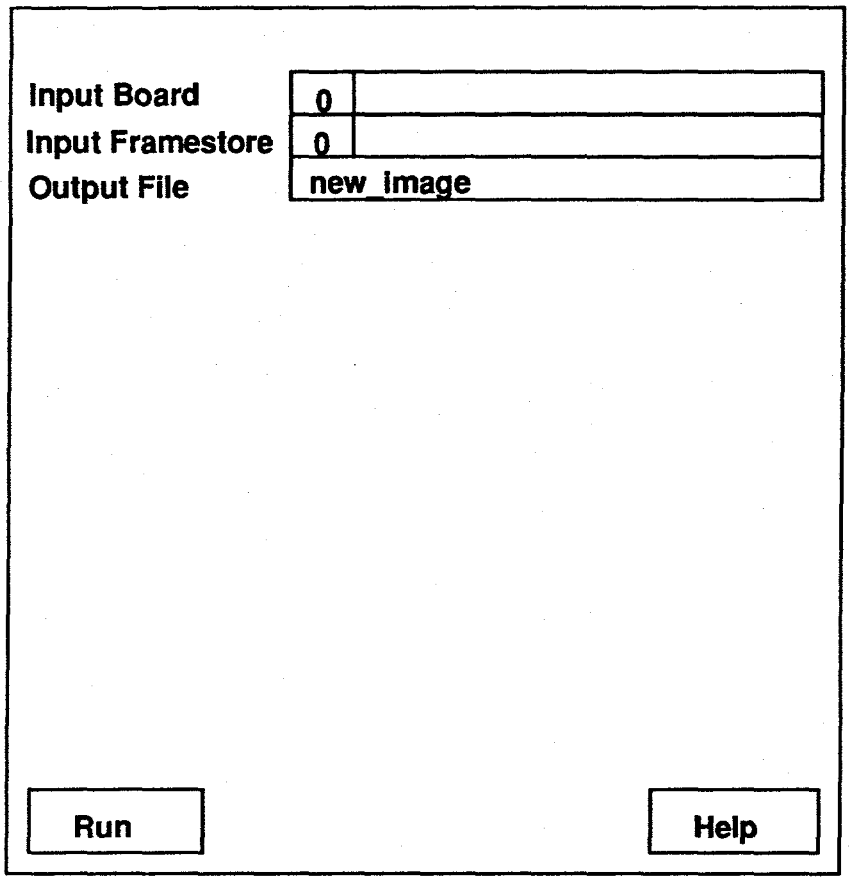

Input Board 0
Input Framestore 0
Output File new_image

Figure 7 – get_image Pane

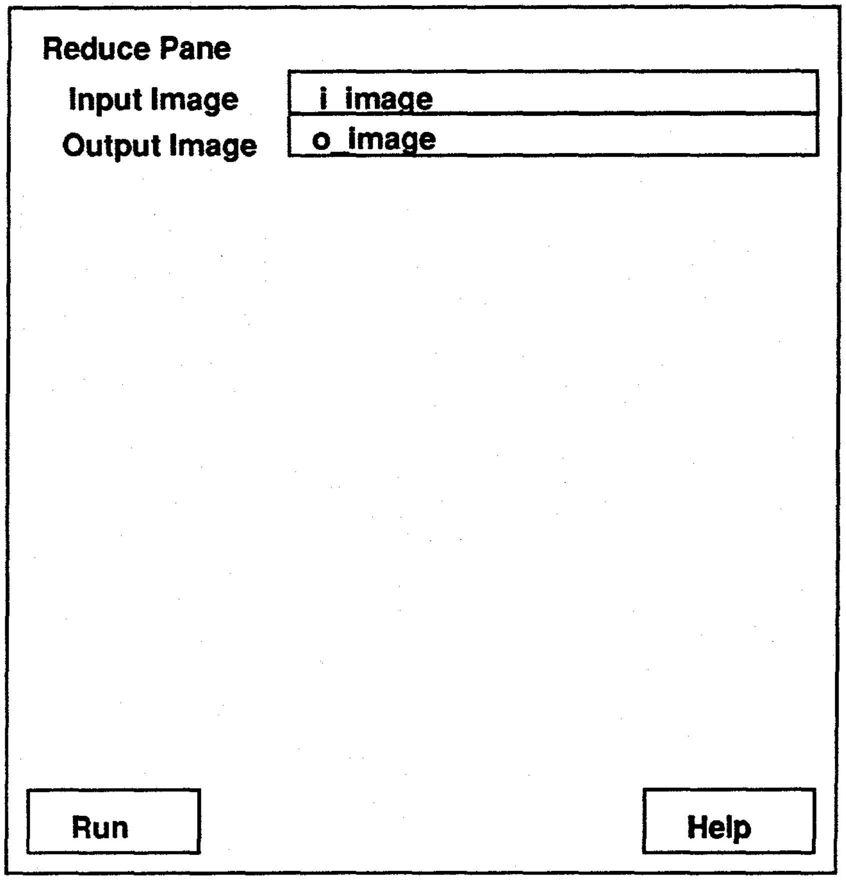
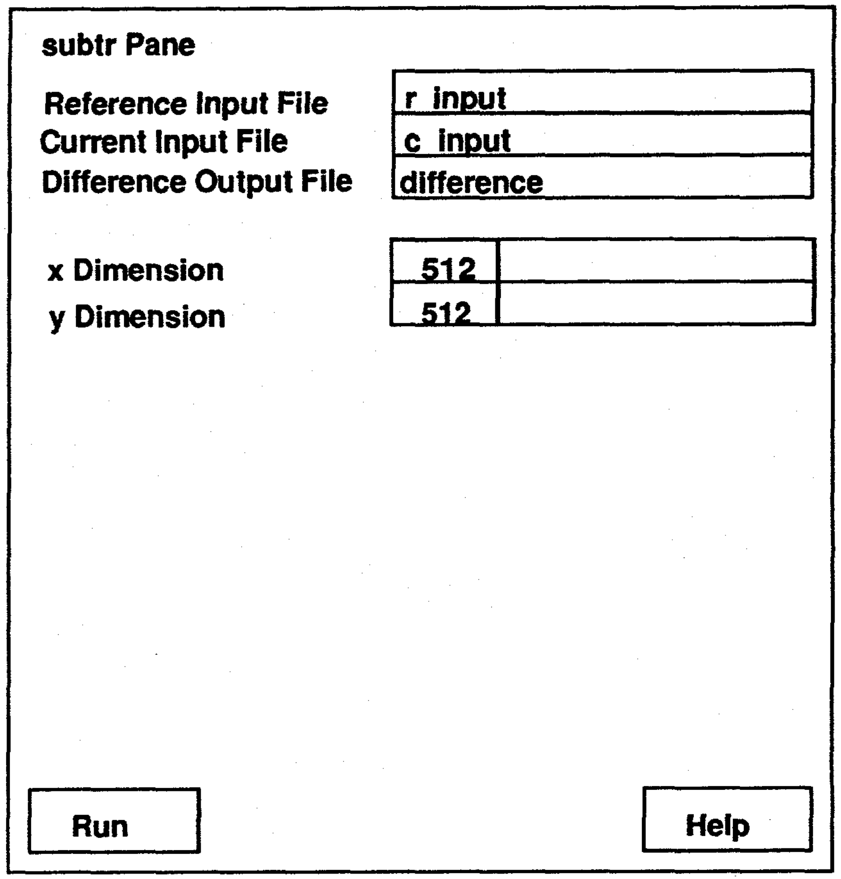


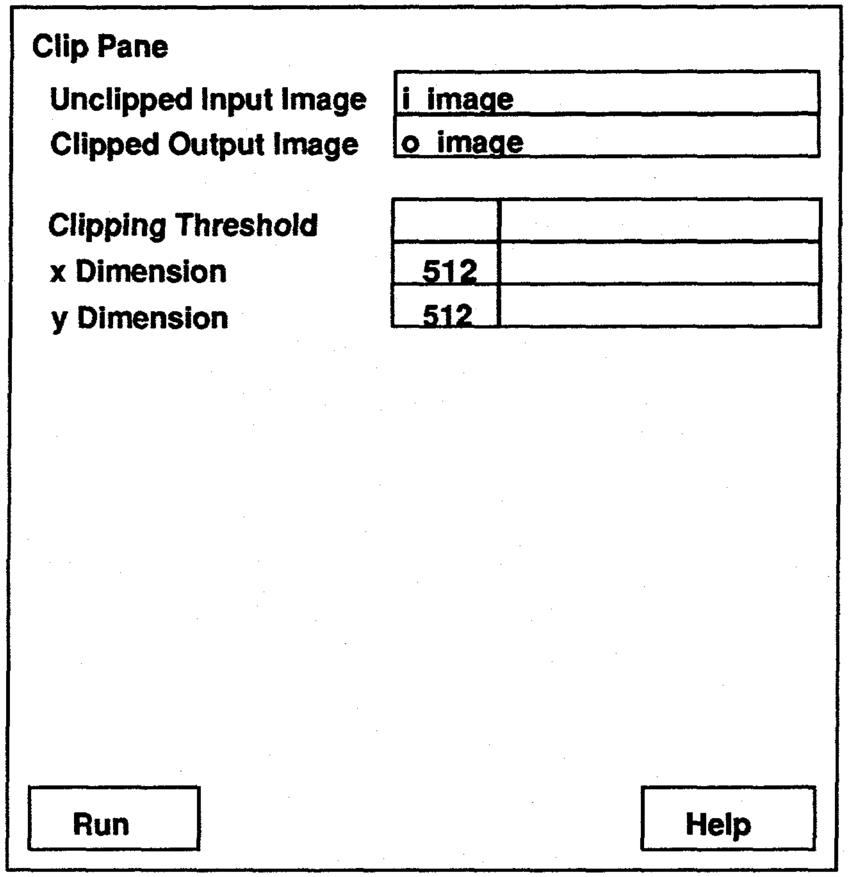
Figure 8 – reduce Pane


subtr Pane

Reference Input File	r_input
Current Input File	c_input
Difference Output File	difference

x Dimension	512	
y Dimension	512	

Figure 9 – subtr Pane

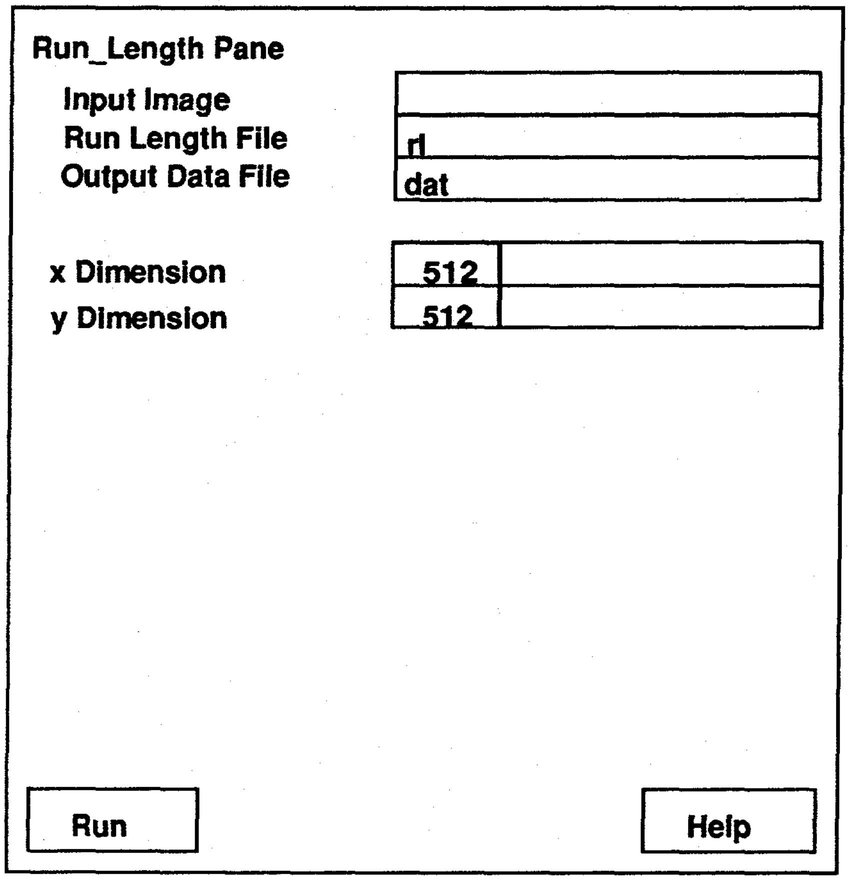


Clip Pane

Unclipped Input Image	i_image	
Clipped Output Image	o_image	

Clipping Threshold		
x Dimension	512	
y Dimension	512	

Figure 10 – clip Pane


Run_Length Pane

Input Image	
Run Length File	rl
Output Data File	dat
x Dimension	512
y Dimension	512

Figure 11 – rl Pane

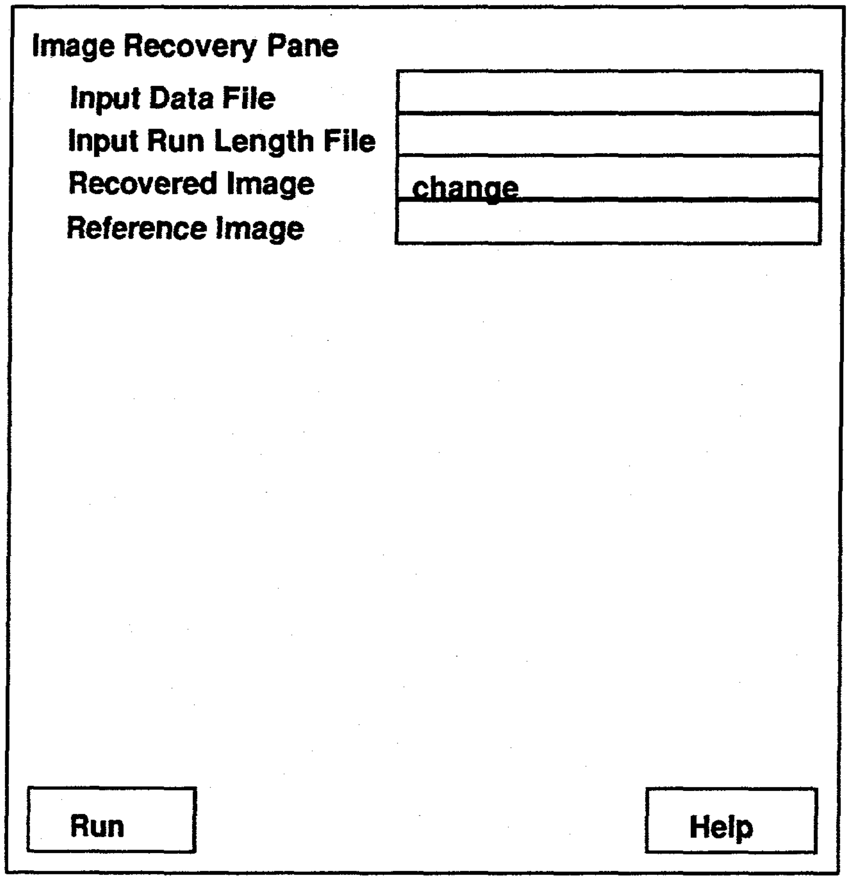
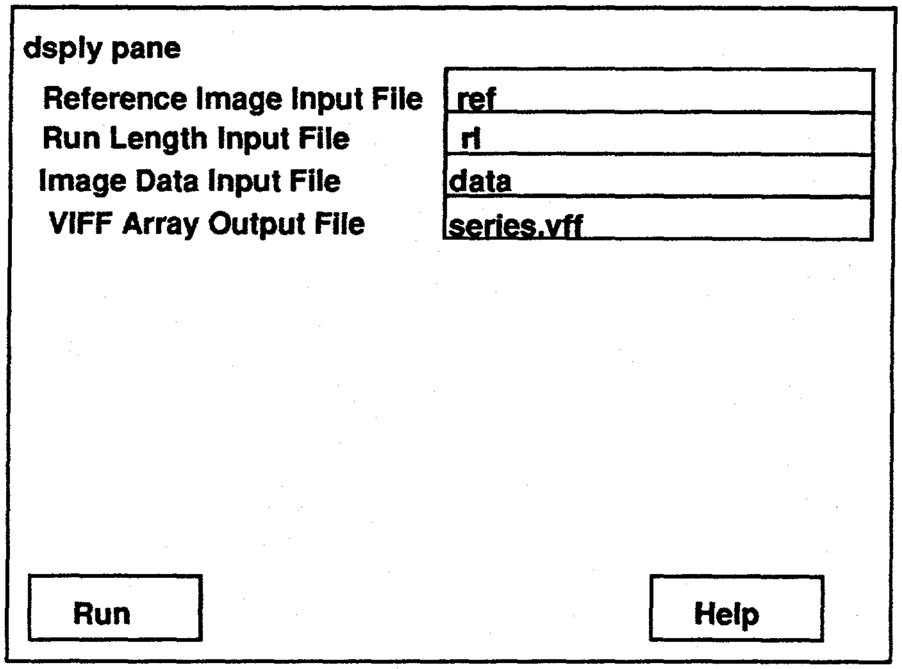
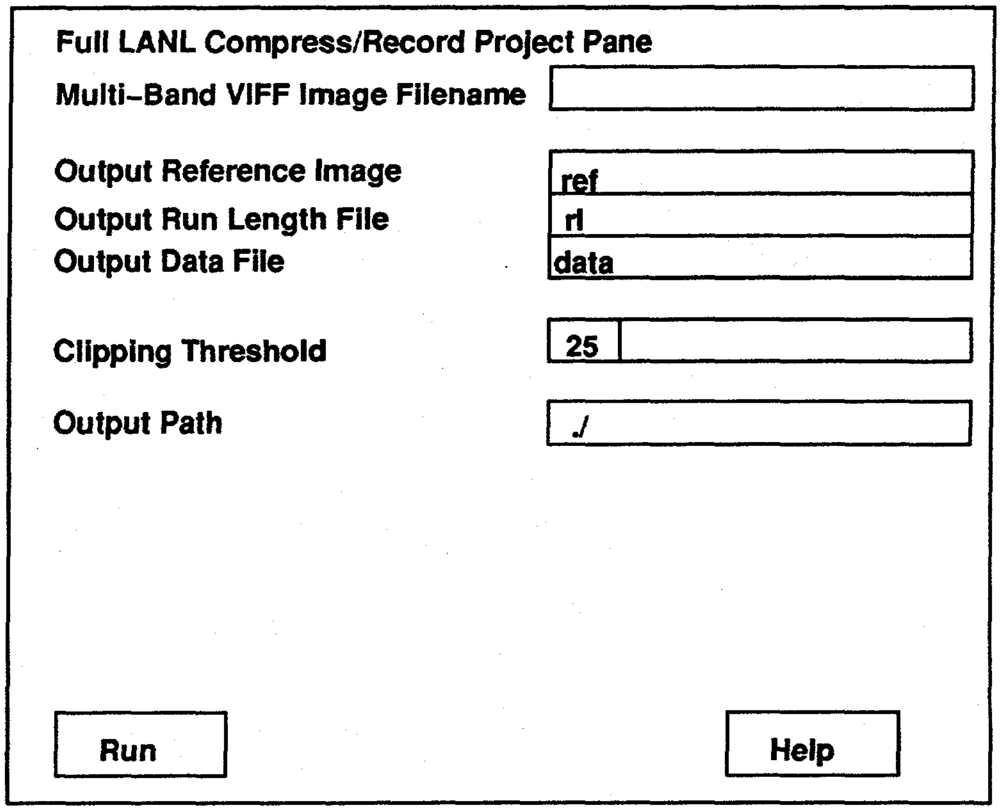


Figure 12 – recover Pane


dsply pane

Reference Image Input File	ref
Run Length Input File	rl
Image Data Input File	data
VIFF Array Output File	series.vff

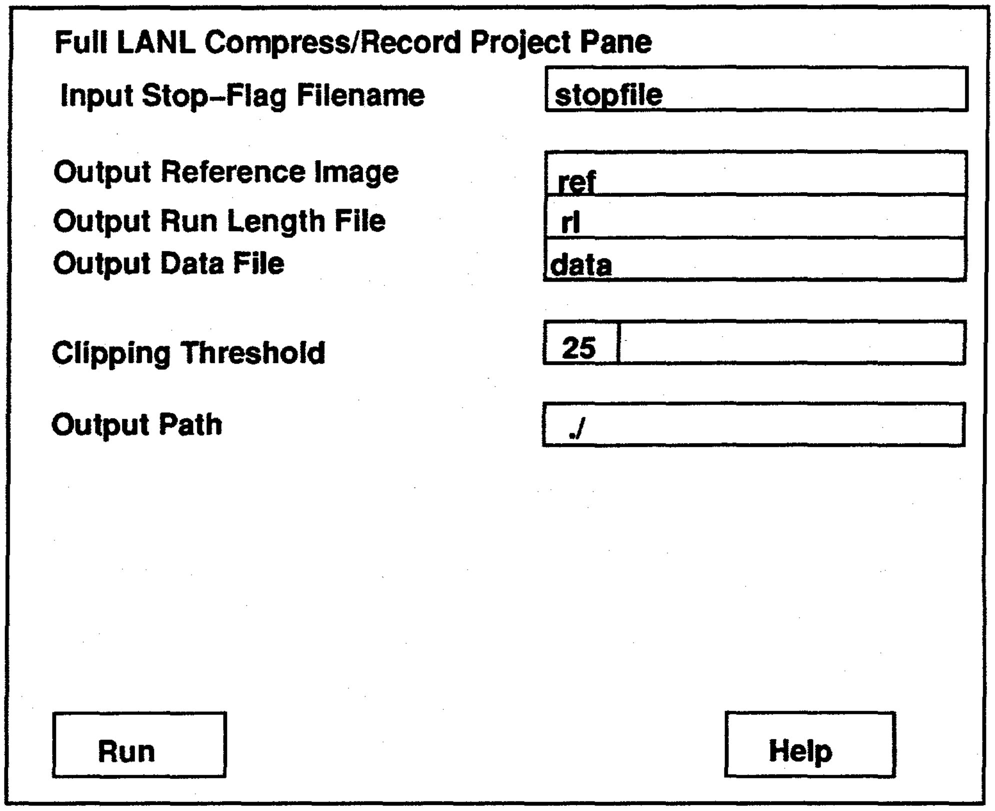
Figure 13 – dsply Pane



Full LANL Compress/Record Project Pane

Multi-Band VIFF Image Filename	
Output Reference Image	ref
Output Run Length File	rl
Output Data File	data
Clipping Threshold	25
Output Path	./

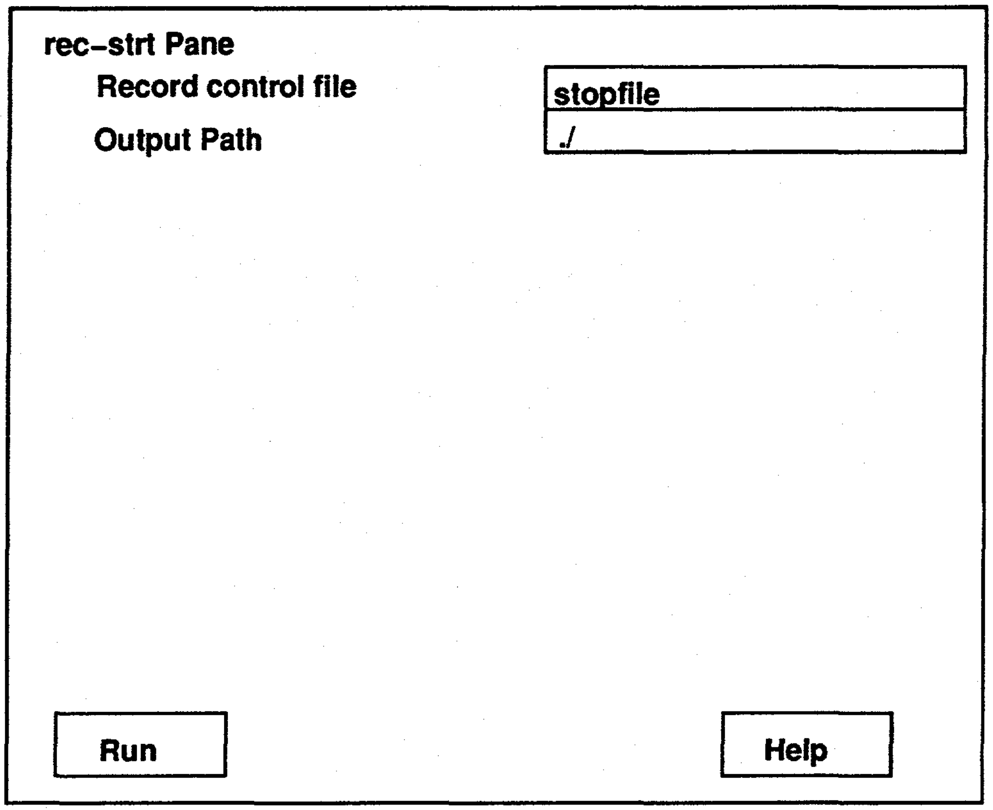
Figure 14 – record_vf Pane



Full LANL Compress/Record Project Pane

Input Stop-Flag Filename	stopfile
Output Reference Image	ref
Output Run Length File	rl
Output Data File	data
Clipping Threshold	25
Output Path	./

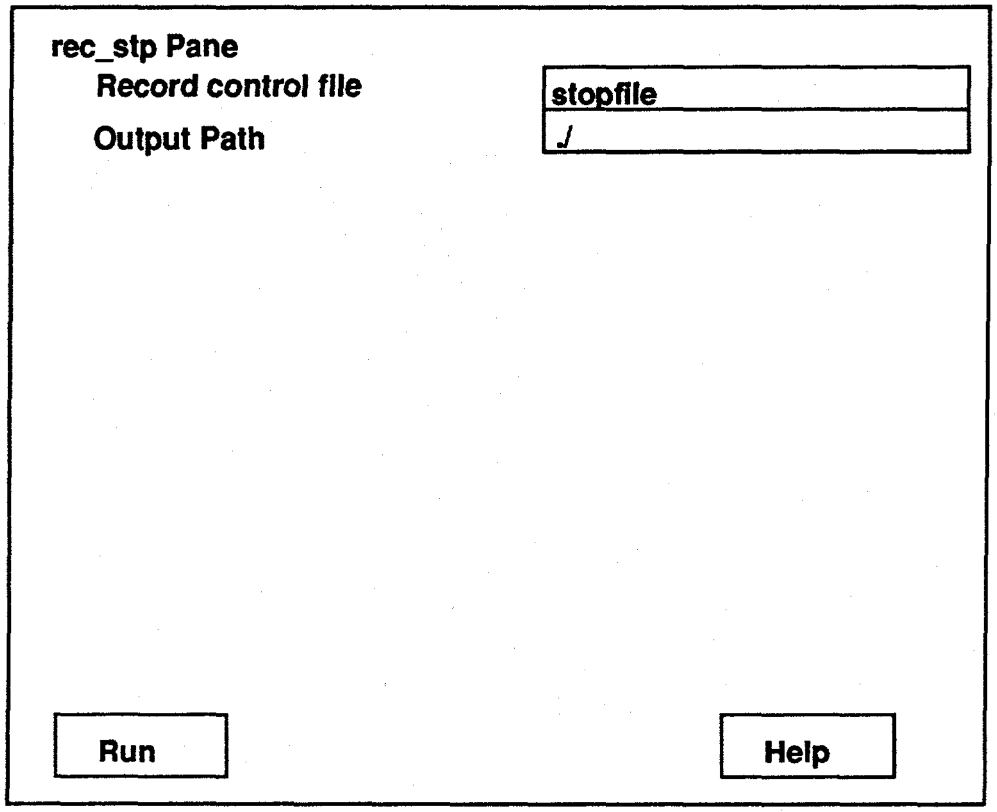
Figure 15 – record Pane


rec-strt Pane

Record control file — stopfile

Output Path — ./

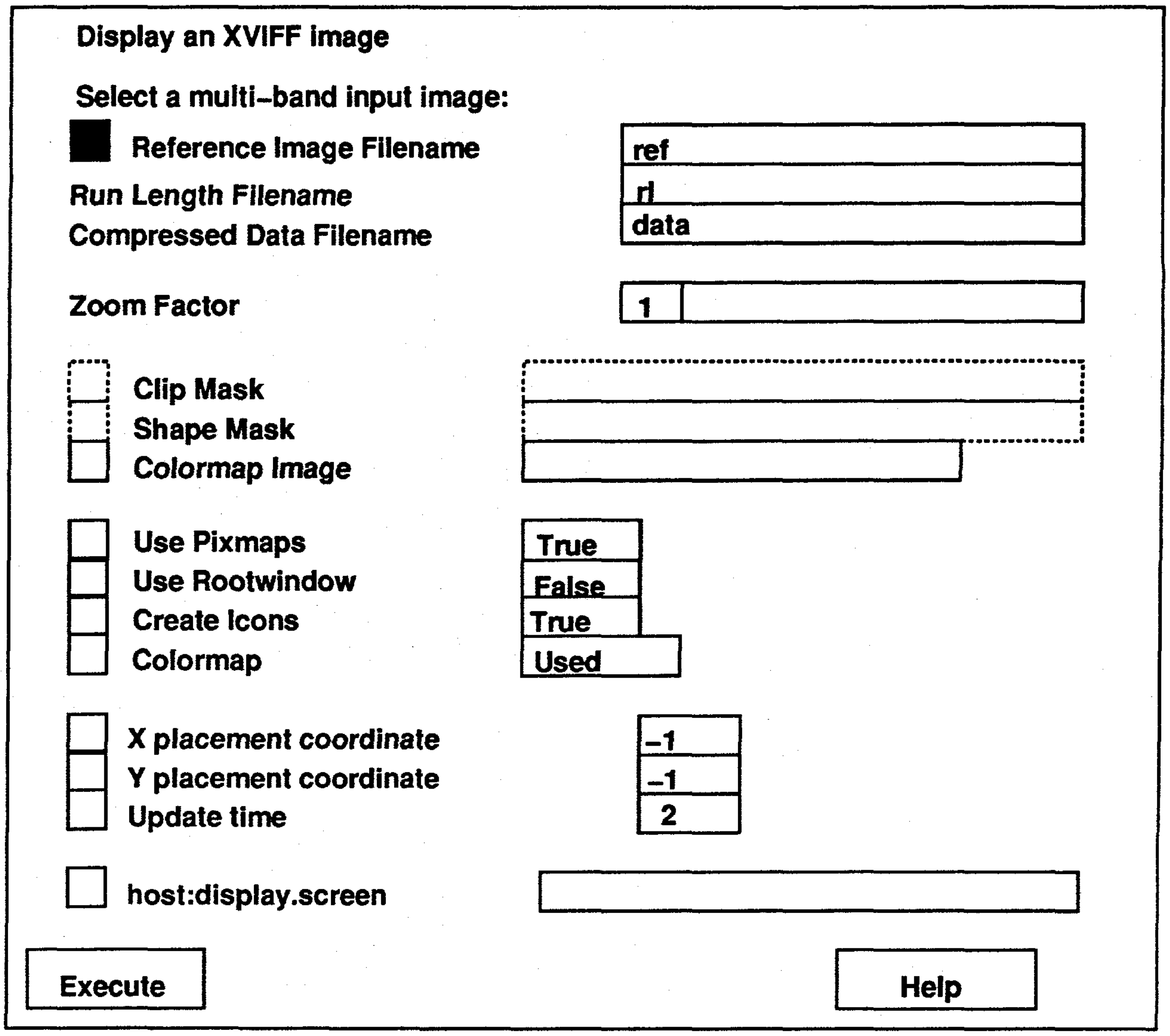
Figure 16 – rec_strt Pane



rec_stp Pane

Record control file	stopfile
Output Path	./

Figure 17 – rec_stp Pane



Display an XVIFF image

Select a multi-band input image:

■	Reference Image Filename	ref
	Run Length Filename	rl
	Compressed Data Filename	data
	Zoom Factor	1
☐	Clip Mask	
☐	Shape Mask	
☐	Colormap Image	
☐	Use Pixmaps	True
☐	Use Rootwindow	False
☐	Create Icons	True
☐	Colormap	Used
☐	X placement coordinate	–1
☐	Y placement coordinate	–1
☐	Update time	2
☐	host:display.screen	

Figure 18 – decompress Pane

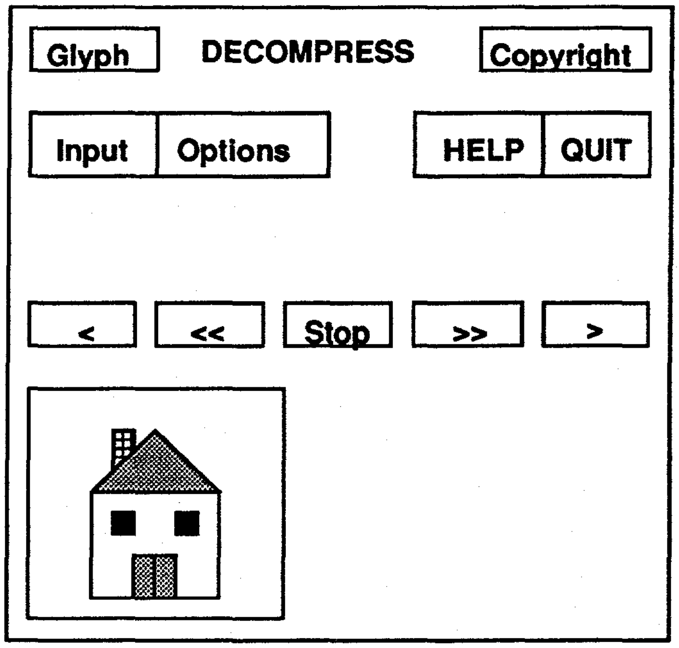
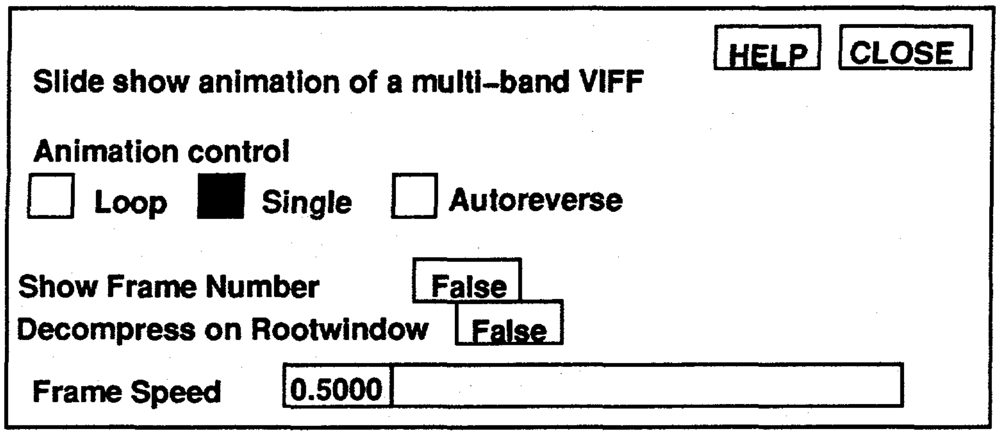


Figure 19 – decompress Animation Screen



HELP CLOSE

Slide show animation of a multi–band VIFF

Animation control

Loop [✓ Single] Autoreverse

Show Frame Number False

Decompress on Rootwindow False

Frame Speed 0.5000

Figure 20 – decompress Animation Options